A PRACTICAL GUIDE TO THE PICCOLO AUTOPILOT

By

ANTON MORNHINWEG

Bachelor of Science Mechanical and Aerospace Engineering

Oklahoma State Unversity

Stillwater, Oklahoma

2009

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2014

A PRACTICAL GUIDE TO THE PICCOLO AUTOPILOT

Thesis Approved:

Dr. Andrew Arena

Thesis Adviser

Dr. Joseph Conner

Dr. Rick Gaeta

Name: Anton Mornhinweg

Date of Degree: July, 2014

Title of Study: A PRACTICAL GUIDE TO THE PICCOLO AUTOPILOT

Major Field: Mechanical and Aerospace Engineering

In support of a UAV contract the Piccolo SL and Piccolo II autopilots were installed and operated on various aircraft. Numerous problems with the autopilot setup and analysis processes were found along with numerous problems with documentation and autopilot system information. Major areas of concern are identified along with objectives to eliminate the major areas of concern. Piccolo simulator vehicle gain calculations and Piccolo generation 2 version 2.1.4 control laws are reverse engineered. A complete modeling guide is created. Methods are developed to perform and analyze doublet maneuvers. A series of flight procedures are outlined that include methods for tuning gains. A series of MATLAB graphical user interfaces were created to analyze flight data and pertinent control loop data for gain tuning.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

Introduction

The goal of this thesis is to create a practical guide for the Piccolo autopilot running the Fixed Wing Generation 2 Version 2.1.4 firmware. The experience of installing and operating the Piccolo autopilot on a variety of different platforms, including Nexstar, Noctua, 25% Cub, and the Diamondback, exposed many short comings of the documentation and data analysis programs Cloud Cap provided in support of its Piccolo autopilot system. There was a lack of information and step by step instructions across the entire setup, and flight testing processes. Additionally the information given to describe the proprietary control systems that govern the decision making processes of the autopilot were inadequate.

1.1. Overview of Piccolo Issues

The following bullet points highlight some of the areas where issues arose:

- **Setup Process**

The setup process began with modeling aircraft geometry and control surface configurations in AVLEditor. Cloud Cap provided a setup guide; however, the setup guide only dedicated a small portion to AVLEditor modeling. It was found that there were many specific small things that had to be done correctly in the modeling that were not made clear. Numerous glitches in AVLEditor were found as well. The AVLEditor model was important because it was used to generate an “Alpha File” which is used by the piccolo simulator to simulate the flying qualities of the modeled aircraft and to generate initial Vehicle Gains for the autopilot. Vehicle Gains are used across an array of different control loops and limits in the Piccolo. Even something as simple as naming the control surfaces can have a profound effect on the results of the simulator’s estimated Vehicle Gains and the model’s performance in simulations. The setup guide did provide definitions for each parameter that is saved in the alpha file; however, it did not give any indication as to what equations the simulator used to calculate the Vehicle Gain estimates. The Vehicle Gain calculations are critical to know to be able to debug any potential issues that may arise with an aircraft model. For example, the simulator estimated two Vehicle Gains, elevator effectiveness and vertical tail arm, for Noctua to be 0. The vertical tail arm was easily replaced as it is simply the length between the center of gravity of the aircraft and the center of gravity of the vertical tail; however, without knowledge of the equations that the simulator uses to calculate Vehicle Gains it was initially impossible to manually calculate a replacement value for elevator effectiveness.

- **Doublet Maneuvers**

Another issue that arose was doublet maneuvers. Doublet maneuvers are used to experimentally determine values for specific Vehicle Gains during flight. Cloud Cap provided a guide to conducting and analyzing doublet maneuvers; however, the guide was very general and didn't actually provide direct guidance as to how to appropriately analyze the results. Cloud Cap provided two different MATLAB scripts that could be used to analyze doublet results via two different methods. One of the scripts relied on the piccolo telemetry log file for surface deflection commands; however, this script required code in the script file to be edited for each different control surface configuration without clear instructions that editing the code would be required or how to do it. Both scripts were found to have undocumented glitches and even errors in Vehicle Gain calculations. Additionally it was found that the procedure established by Cloud Cap to effectively analyze the results was flawed; thus, a comprehensive guide with new analysis procedures was needed for both the Nexstar and Noctua platforms. Due to the poor design and lack of user friendliness of the piccolo telemetry log file method a complete revamp of the analysis tools became a necessity for use on current and future platforms.

- **Control Laws**

Cloud Cap provided some insight into the control schemes used in the Piccolo; however, the information was fragmented and scattered across various documents. Additionally the most descriptive that the documentation gave was simple definitions of control loop gains, and definitions of command loops. In addition to the gains there were various limits that effect control laws and control loops. Most of the limits are defined in Cloud Cap documentation; however, it was found that there are limits such as a z – acceleration limiter that is not mentioned in any documentation. The limiter plays a huge role in elevator control and was actually discovered because it came into play during an auto land of Noctua. There was also found to be Longitudinal Modes that greatly affect elevator and throttle control; however, Longitudinal Modes were only referred to briefly in one sentence of the software release notes. The reference only referred to what one longitudinal mode was named. Additionally Cloud Cap did not provide any information on how to tune control loops. The Nexstar had severe tracking control issues and the only way it could be tuned at the time was to mindlessly change gains that may or may not affect the track control performance. It became clear early on that a system for tuning control loop gains would be very useful for operating the Piccolo. With so much uncertainty and gaps in explanations it became necessary to develop as much of an understanding of the control laws as possible. A comprehensive understanding of the control laws would not only be useful for tuning control loop gains but even assessing control loop performance after flights. It was found through experience with Noctua that even if, in flight, all systems appeared to be operating appropriately a more thorough analysis after flight could catch underlying issues that were not readily apparent.

- **Flight Procedures**

The Cloud Cap documentation also lacked formal flight procedures. There was no real guidance for flight planning, flight testing, flight procedures, and post flight analysis. It became essential to develop a clear method for planning for flights and analyzing them afterwards. When the autopilot was installed and operated on the Cub, Diamondback, and Nexstar there were essentially no flight guidelines for operating the piccolo autopilot. Without formal procedures for flight planning, testing, and post flight analysis, coupled with the lack of knowledge of the control system, the entire process of ultimately achieving fully autonomous flight of the Nexstar was painfully long, clumsy, and unorganized.

- **Data Analysis**

The piccolo software comes with the ability to collect and log inordinate amounts of data. The data is stored in what are usually enormous text files. It quickly became clear that in order to properly analyze flights there needed to be a way to efficiently analyze the mass of data that can be recorded from piccolo and controller telemetry. Cloud Cap offered a few MATLAB scripts that would organize data from piccolo telemetry files, import them into MATLAB workspace variables, and then automatically create 20 figures; however, the figures were mostly random information and not all pertinent to post flight analysis. Additionally there were no MATLAB scripts to organize and view data recorded from the controller telemetry which is data that is extremely useful to tune gains. It was found at one point that the controller telemetry does not display or record any energy or energy rate commands and feedback. Energy and energy rate is core to throttle control and there was absolutely no way of assessing the energy control loop performance.

Amidst all of the issues and lack of knowledge that was encountered it was decided to create an in house guide for Oklahoma State University to correct much of the confusion and uncertainty that came with the experience of installing and operating the piccolo autopilot on the Cub, Diamondback, Nexstar, and Noctua. The following objectives were created for the guide:

1) Search for existing work.
2) Determine as much as possible about how the simulator calculates vehicle gains that are applicable to fixed wing gen 2 version 2.1.4
3) Create a step by step detailed guide to create aircraft models for the piccolo simulator.
4) Create a guide for conducting and analyzing doublet maneuvers based on the experiences of the Nexstar and Noctua.
5) Perform simulation experiments to estimate proprietary control systems.
6) Develop a system for tuning autopilot control loops.
7) Create guidelines for operating the piccolo autopilot
    a. Flight Planning
    b. Pre – Flight
    c. First Flight Operations

d. Flight Procedures

e. In Flight Analysis

f. Post – Flight Analysis

1.2. Literature Review

1.2.1. Piccolo Simulator Vehicle Gains

Cloud Cap offered two documents that mention the Vehicle Gains. According to documentation the simulator has the ability to generate vehicle parameters from the aerodynamic model. (Piccolo Simulator 48). Each vehicle gain is defined in PccUsersGuide pgs. 115 – 117. Unfortunately neither document directly expresses how the vehicle gains are calculated. According to the documentation the final step in the modelling process is the alpha sweep which creates the alpha file (Piccolo Simulator 19). The contents of the alpha file are the same as the stability derivatives that the documentation describes the simulator as using from AVL (Piccolo Simulator 53). Additionally the purpose of AVL is described as being used for the simulator's aerodynamic model in which the AVL code was modified to output an XML file of the stability derivatives representing a virtual wind tunnel angle of attack sweep (Piccolo Simulator 7). These descriptions of the simulator pointed towards the stability derivatives in the alpha file as the main source for calculating vehicle gains.

1.2.2. Aircraft Modeling for the Piccolo Simulator

Cloud Cap offers one document that describes the modeling process for the Piccolo simulator. The document is called "Piccolo Simulator." The document mainly introduces the software that is used for the modeling process and describes the logistical basic steps to use them. There is one section that lists steps to complete an entire model for the simulator. With respect to AVLEditor the document offers a small step by step example of creating a wing. While this provides a logistical guide that describes what buttons in AVLEditor to click and how to use them it does not explicitly state how to create an entire AVLEditor aircraft model. The document also describes some of the basics of reinforcing AVLEditor models with Xfoil and performing alpha sweeps. Again the steps mainly describe the logistics of how to use the software instead of an actual guide for setting up and properly performing the analysis. A lot of the complexities that were ran into during the experience of setting up for the Diamondback, Cub, Nexstar, and Noctua were not covered in the documentation. The document also lists and details all of the parameters that can be specified in the simulator text file. It is very helpful with regards to simulator file options.

A thesis written for evaluating and testing flight dynamics of a Yak – 54 with a piccolo autopilot system investigated the use of AVL to estimate aircraft dynamics for the piccolo autopilot (Jager 39). The author compares dynamic model solutions of different modeling programs. The focus of the modeling portion of the thesis was to test the accuracy of different modeling techniques. Additionally the thesis was written for an old version of the piccolo software, version 2.0.5. The modeling process was much different than it is in 2.1.4, AVLEditor was not even used at the time.

1.2.3. Doublet Maneuvers for testing Vehicle Gains

Cloud Cap provides one document to explain how to conduct and analyze doublet maneuvers to test the surface effectiveness vehicle gains, “Piccolo Doublet Analysis Tool”. Doublet maneuvers are discussed in “PccUsersGuide.” The user’s guide describes doublets maneuvers to be used to deflect an axis of the aircraft and measure its response (PccUsersGuide 96). The document also gives a brief description of the doublet command functions in PCC.

The doublet analysis tool document is not actually included with the cloud cap install package. The document is found in the CCTMATLAB extra which contains Cloud Cap MATLAB scripts. CCTMATLAB is part of the “Aircraft Modeling Tools” which can be downloaded from Cloud Cap's webpage. The document describes logistically how to use the two different MATLAB scripts provided for analyzing doublet maneuvers. The document gives brief guidelines to conduct aileron and elevator doublet maneuvers. The instructions for aileron doublet maneuvers call for a relatively small aileron deflection to create a modest roll rate of 20 deg/sec (Piccolo Doublet Analysis Tool 3). The instructions for elevator doublet maneuvers state that the elevator should be deflected to two different positions that will not take the aircraft to stall, and keep the throttle fixed (Piccolo Doublet Analysis Tool 3). The document offers a few guidelines on how to use the analysis to tool to properly analyze the results of doublet maneuvers. The document instructs the user to pick response points on the doublet plots that correspond with momentarily steady pitch or roll rates (Piccolo Doublet Analysis Tool 6). The document also provides examples of data with too much noise that could be due to vibrations. Additionally if deflections are too large other parameters could come into play and influence the solutions while deflections that are too small can cause the solutions to be more susceptible to noise (Piccolo Doublet Analysis Tool 8). The author of the document advises the user to conduct multiple iterations of maneuvers, disregard the highest and lowest results, and average the rest to come up with a final value for a given surface effectiveness test (Piccolo Doublet Analysis Tool 8).

1.2.4. Piccolo Proprietary Control Laws

Cloud Cap provides a couple of documents that shed light on the control processes of the piccolo autopilot. The "PccUsersGuide" provides definitions for command loops (pg. 100), control loop gains (110 – 113), limits, (114 - 115), and vehicle gains (115 – 117). The document "Tuning piccolo control laws 2.0.x" provides additional gain definitions and control law explanations; however, the document describes an older version of the controller, 2.0.x, rather than 2.1.4.x. Other than gain and parameter definitions there is not any further explanation of the control laws; however, the gain definitions were good enough to begin experimental testing procedures to back out the control laws.

A thesis written for evaluating and testing flight dynamics of a Yak – 54 with a piccolo autopilot system referenced Cloud Cap gain definition documents to shed light on the details of the piccolo control laws. The author ultimately stated that the control algorithims are not publicly available (Jager 24). Additionally the piccolo that was written about was operating old software, version 2.0.5.

1.2.5. System for tuning control loops

There are not any Cloud Cap documents that explicitly state any type of methods or system to tune control loops. One document refers the user to follow the initial flight test cards 2.x (Steps to Autonomous Flight 11); however, no such flight test cards exist. The very same document also directed the user to the vehicle integration guide for the process for tuning gains (Steps to Autonomous Flight 6). The vehicle integration guide states that the default gains should be sufficient for any aircraft that is similar to a Cub. The guide also says that for any new aircraft the user must start from the beginning to develop new gains, and that there is no formula for determining gains the process must be done by trial and error. The document ends its gain tuning instructions by referring the user to the "Initial Flight Test Cards" document (Piccolo Vehicle Integration Guide 25). Both documents refer the user to the non-existent flight cards.

A thesis written for evaluating and testing flight dynamics of a Yak – 54 with a piccolo autopilot system presented flight test cards for validating bank angle control (Jager 179), heading control (Jager 181), and airspeed control (Jager 183). A major problem with the flight test cards was that they were designed for version 2.0.5 of the piccolo. The control system the flight test card were designed to test do not directly apply to version 2.1.4.x.

1.2.6. Flight Guidelines

Cloud Cap provides a few flight guidelines throughout a couple of documents. One document, "Piccolo Vehicle Integration Guide", details a list of pre flight checks to perform during pre flight. The list covers procedures for testing control surface deflections, communications signal strengths, and inertial measuring sensors (Piccolo Vehicle Integration Guide 27). One document is a flight summary log sheet to be used during flight (Flight Summary Log Sheet). The flight summary log sheet includes space for writing in aircraft modifications, flight objectives, flight notes, mission comments, and aircraft configuration details. Items from both the pre flight checklist and the flight summary log sheet were used to develop the guidelines outlined in Chapter 8.

In a thesis that was written for evaluating and testing flight dynamics of a Yak – 54 with a piccolo autopilot system there was a list of pre-flight checks to perform (Jager 172). The list was mostly specific to the Yak configuration that was being used and did not include any additional components from Cloud Cap's pre – flight checklist that were included in the pre-flight checklist outlined in Chapter 8.

1.3. Overview of Platforms Piccolo Operated On

The following sections detail the success and failures of each platform that the autopilot was installed on.

1.3.1.   Diamondback

- Ran a Piccolo II, unit 1478, with version 2.1.1.e
- Electric propulsion system
- Successfully navigated waypoints after manual take off and coupled with manual land
    - Flew laps around box patterns
- No attempted auto take offs or auto lands.

- Crashed immediately following takeoff after piccolo communications were lost (manual pilot was flying through the ground station and the piccolo comms). Refer to Appendix A for further information on what caused the crash. Piccolo unit was destroyed.

1.3.2. 25% Cub

- Ran the Piccolo SL, unit 20679, with version 2.1.1.e
- Gas propulsion system.
    - Had major vibration issues
- Successfully navigated waypoints after manual takeoffs and coupled with manual lands.
- Flew 5 flights.
- Crashed during Flight 5 when a servo locked up and pulled over the piccolo maximum servo amperage of 2 amps. Refer to Appendix A for further information on what caused the crash. Piccolo unit miraculously survived the crash.

1.3.3. Nexstar

- Ran the Piccolo SL, unit 20679. The first 15 flights flew with version 2.1.1.e. Flights 16 – 26 flew with version 2.1.4.f. Flights 27 – 29 flew with version 2.1.4.g.
- Electric Propulsion System.
- Manual pilot flew through the JR level shifter, outside of the piccolo communications.
- Successfully achieved fully autonomous flight (2 fully auto flights with wheeled landing and wheeled takeoff).
- Flew 29 Flights.

1.3.4. Noctua

- Ran the Piccolo SL 565, unit 4563.
- Gas propulsion system.
- Manual pilot flew through the JR level shifter, outside of the piccolo communications.
- At the time of writing this thesis:
    - Successfully navigate waypoints.
    - Successfully auto landed, 1 time, with laser altimeter.
    - Minor crash after attempting 1 wheeled takeoff.
    - Flown 13 Flights.
    - Plan to launch with a rail launcher in the future.
    - Plan to test repeatability of auto lands with laser altimeter.

1.4. Experimental Strategy to Extract Simulator Vehicle Gain Calculations

As mentioned in Section 1.2.1 the Piccolo Simulator document pointed towards the alpha file, generated by AVLEditor, as the primary source for simulator vehicle gain calculations. The main strategy was to use the definitions of vehicle gains to develop theories as to which stability derivatives could be most likely used to calculate each vehicle gain. Once a theory was formulated the alpha file of the Nexstar model would be modified, via Notepad. The theorized stability coefficients would be altered to enormously large values, one at a time, to observe if any changes were made to the original model’s vehicle gain calculations. Each stability coefficient in the alpha file contained multiple values with different angles of attack. Only one angle of attack value of each coefficient in question would be changed for each iteration. Different stability coefficients were altered one value at a time and their vehicle gain effects were noted.

The following example outlines the process of reverse engineering how the simulator calculates rudder power. Rudder power is defined as the change in yawing moment coefficient over change in rudder deflection (PccUsersGuide 116). One of the stability coefficients in the alpha file, Cnd#, is defined as the change in yawing moment coefficient with control surface deflection (Piccolo Simulator 55). The Nexstar alpha file ranged from -2 to 10 by 2 degrees. The rudder was control surface 4 in the AVLEditor model. Values of Cnd4 were iteratively altered to enormous values, starting with the value at -2 alpha, to observe any changes in the rudder power calculation. All of the Cnd4 alpha values were altered and it was discovered that Cnd4 (alpha = 0) can affect the simulator's calculation of rudder power. Using the definition of rudder power and the knowledge that Cnd4 was in units of /deg the following equation, Equation 1, was theorized and used to attempt to estimate the calculation of rudder power:

$$Rudder\ Power = C_{n_{\delta_r}}(\alpha = 0) * \frac{180}{\pi}$$

Equation 1 Rudder Power Proposed Calculation

The equation was found to be correct; however, vehicle gain calculations were also subject to scalar terms in the simulator file (Piccolo Simulator 21). It was theorized that the scalar terms are multiplied by their corresponding Cnd values; hence, with respect to rudder power the scalar term would simply be multiplied to the rudder power calculation. Equation 2 below was found to be correct.

$$Rudder\ Power = C_{n_{\delta_r}}(\alpha = 0) * \frac{180}{\pi} * C_{n_\delta}Scaler$$

Equation 2 Rudder Power

1.5. Experimental Strategy to Extract the Piccolo's Proprietary Control Systems

1) Start with the design purpose of the control law as explained in Tuning piccolo control laws 2.0.x. For example elevator control is supposed to control altitude in altitude control and control airspeed in airspeed control.

2) Construct control loop schematics from gain definitions

    a) Start with definitions from PccUsersGuide

    b) Reinforce with extra definitions in Tuning piccolo control laws 2.0.x (where applicable)

3) Reinforce schematics with command loops from their definitions.

4) Reinforce schematics with limits from their definitions.

5) Conduct software in the loop simulations to extract as much information as possible. All of the control laws final control loop consisted of a feedforward path and a feedback path. The feedforward paths typically use vehicle gains to calculate actuator deflections from the final loop commands

    a) As a result the first step in the simulations was to isolate the feedforward loop by setting all of the feedback gains to 0.

    b) The vehicle gains that were known to be associated with the corresponding feedforward loop would be set to 0 one at a time. If the resulting actuator deflections oscillated between maximum and minimum deflections that meant the deflections were essentially oscillating between positive and negative infinity; thus, the vehicle gain was divided in the feedforward equation. Similarly if the resulting actuator deflections were 0 then the vehicle gain was multiplied in the feedforward equation.

c) The vehicle gain definitions were combined with the findings of zeroing the vehicle gains and equations were referenced that could apply to the calculations and equations were hypothesized.

d) To test the hypothesized equations the outer loop gains were isolated to a gain value of 1, any outer loop integral or derivative gains were zeroed, and the outer loop commands were assumed to be equal to the outer loop error to estimate inner loop commands (if the inner loop commands were not known such as in throttle control).

e) The hypothesized equations from c) combined with the commands of d) were experimented with until the actuator deflection commands were correctly predicted.

f) The process would be repeated for the feedback loop paths which always also involved vehicle gains.

CHAPTER II

CCT Software

2. Introduction

Cloud Cap provides many software tools in support of operating the piccolo autopilot. All of the operating programs are included in the two installer files that come in the main software kit. In addition to operational programs Cloud Cap also provides MATLAB scripts to assist the user in different types of analysis. The assortment of script files perform functions like importing data from telemetry files in MATLAB structures, analyzing doublet maneuver data, and even generating propeller geometry files.

The Cloud Cap install software comes in two installer files. “PccInstaller.msi” and “PiccoloInstaller”.msi. These files can be downloaded from the Cloud Cap Technology webpage https://www.cloudcaptech.com/validate/default.asp. A login username and password is required to download the zipped folder. The zipped folder is password protected and requires “7-Zip” to unzip it. The current install files should be located on the network in the “Piccolo” drive, “Piccolo_Dev_Kit_2.1.4.i”.

2.1. PccInstaller

The PccInstaller installs the following components:

1) Piccolo Command Center (PCC)

2) PCC Documentation

3) Cloud Cap Plugin SDK (software development kit)

2.2. PiccoloInstaller

The piccolo installer installs the following components:

1) Autopilot firmware

2) AVLEditor, which includes:

    a. avlcct (cloud cap's edited version of avl)

    b. xfoil

3) DevInterface (Piccolo Development Interface)

4) NavFilterInterface

5) Piccolo Docs

6) Ground Station, and Piccolo firmware files

7) Programmer Software. The programmer is used to update firmware on the piccolo autopilot and groundstation units

8) Piccolo Simulator

9) Software in the loop simulation programs

10) Communications and Simulator Software Development Kits

2.3. Software Development Kits

Cloud Cap offers some software development kits that allow the user to add to the software capabilities of PCC, and the Simulator. None of the software development kits are documented, and there are no real guidelines for how to utilize them. This section offers a brief description of each software development kit so that the user is aware of the possibilities for programming their own features.

The communications software development kit, or “CommSDK”, is located in the Cloud Cap folder under “Cloud Cap\Piccolo 2.x.x.x\Piccolo Tools\”. The kit comes with examples of comms apps and it also includes presumably all of the piccolo communications code. There is a document that discusses the communications protocols of the autopilot, “piccolo communications.pdf”, located in the Cloud Cap folder under “Cloud Cap\Piccolo 2.x.x.x\Piccolo Docs\Software\”.

The simulator software development kit, or “SimulatorSDK”, is located in the Cloud Cap folder under “Cloud Cap\Piccolo 2.x.x.x\Piccolo Tools\”. The kit includes C++ header files and C++ project files that presumably the simulator uses.

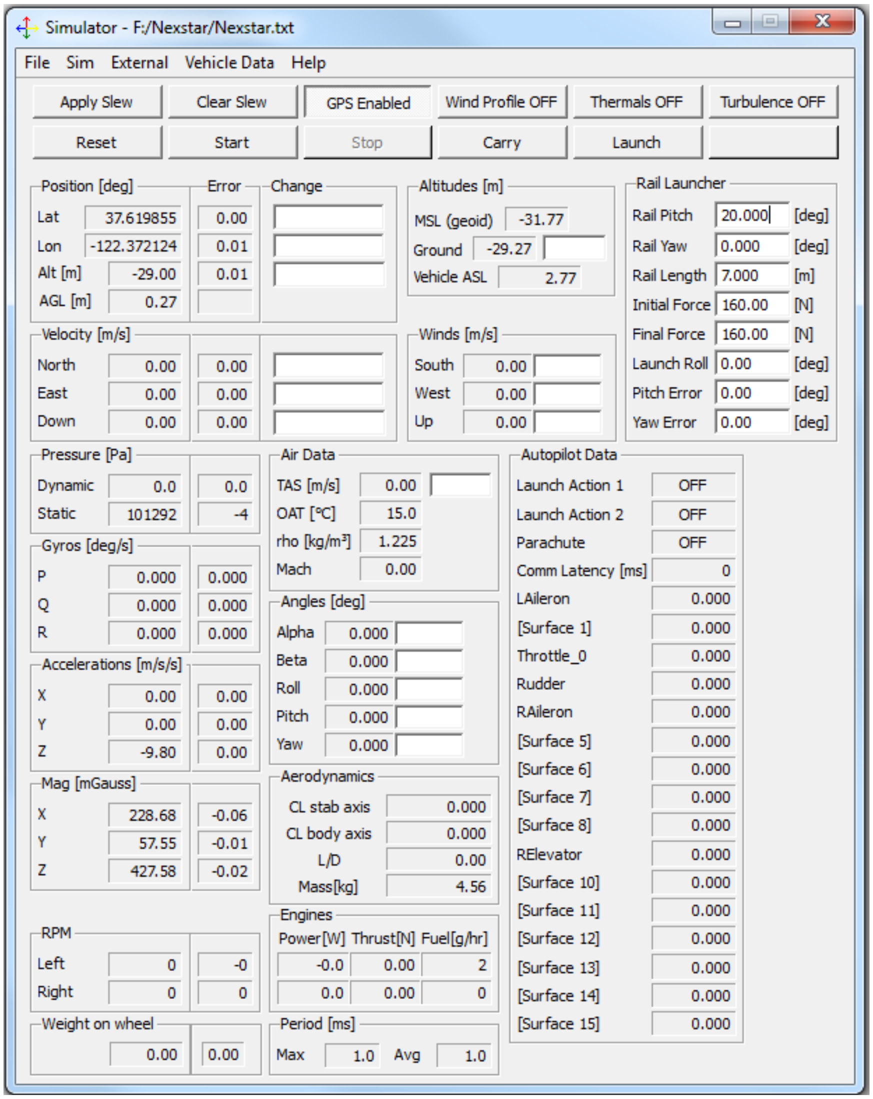
The Cloud Cap plugin software development kit, or “CloudCapPluginSDK”, is located in the Cloud Cap folder under “Cloud Cap\Piccolo 2.x.x.x\Piccolo Command Center\Tools\”. The plugin sdk comes with example plugins for the Gimbal Camera and a generic PCC plugin example. The PCC install provides 4 plugins, an Antenna, FootPrint, Gimbal, and StripChart plugin. All four plugins have to be unlocked by purchasing the corresponding software package from Cloud Cap. The files for each of the four plugins can be found in the Cloud Cap folder under “Cloud Cap\Piccolo 2.x.x.x\Piccolo Command Center\plugins\”. There is a user’s guide for both the Antenna and Gimbal plugins. The “AntennaPluginUsersGuide.pdf” is located under

“Cloud Cap\Piccolo 2.x.x.x\Piccolo Command Center\plugins\PCCPlugin_Antenna\”. The Gimbal plugin documentation is called “ViewPointUsersGuide.pdf” and is located under “Cloud Cap\Piccolo2.x.x.x\Piccolo Command Center\plugins\PCCPlugin_Gimbal\docs\”.

2.4. Piccolo Simulator

The simulator from cloud cap is designed to simulate flying specific aircraft with the piccolo autopilot. The simulator can be used to conduct HiL (hardware in the loop) and SiL (software in the loop) simulations. This section covers the operational capabilities that the simulator creates for the user and how to use it. Chapter 3 discusses simulator mechanics such as how the simulator uses the aircraft simulator file to predict the aircraft’s flying qualities.

Cloud Cap hardly provides any documentation on the operational aspects of the Piccolo Simulator. “Piccolo Setup Guide” pg. 31 contains a very small amount of information on how to output the simulations to Flight Gear. Most of the simulator documentation is concerned with setting up simulations and creating models and simulator files.



Simulator - F:/Nexstar/Nexstar.txt

Position [deg]		Error	Change
Lat	37.619855	0.00	
Lon	-122.372124	0.01	
Alt [m]	-29.00	0.01	
AGL [m]	0.27		

Altitudes [m]		
MSL (geoid)	-31.77	
Ground	-29.27	
Vehicle ASL	2.77	

Rail Launcher		
Rail Pitch	20.000	[deg]
Rail Yaw	0.000	[deg]
Rail Length	7.000	[m]
Initial Force	160.00	[N]
Final Force	160.00	[N]
Launch Roll	0.00	[deg]
Pitch Error	0.00	[deg]
Yaw Error	0.00	[deg]

Velocity [m/s]			
North	0.00	0.00	
East	0.00	0.00	
Down	0.00	0.00	

Winds [m/s]		
South	0.00	
West	0.00	
Up	0.00	

Pressure [Pa]		
Dynamic	0.0	0.0
Static	101292	-4

Gyros [deg/s]		
P	0.000	0.000
Q	0.000	0.000
R	0.000	0.000

Accelerations [m/s/s]		
X	0.00	0.00
Y	0.00	0.00
Z	-9.80	0.00

Mag [mGauss]		
X	228.68	-0.06
Y	57.55	-0.01
Z	427.58	-0.02

RPM		
Left	0	-0
Right	0	0

Weight on wheel	
0.00	0.00

Air Data		
TAS [m/s]	0.00	
OAT [°C]	15.0	
rho [kg/m³]	1.225	
Mach	0.00	

Angles [deg]		
Alpha	0.000	
Beta	0.000	
Roll	0.000	
Pitch	0.000	
Yaw	0.000	

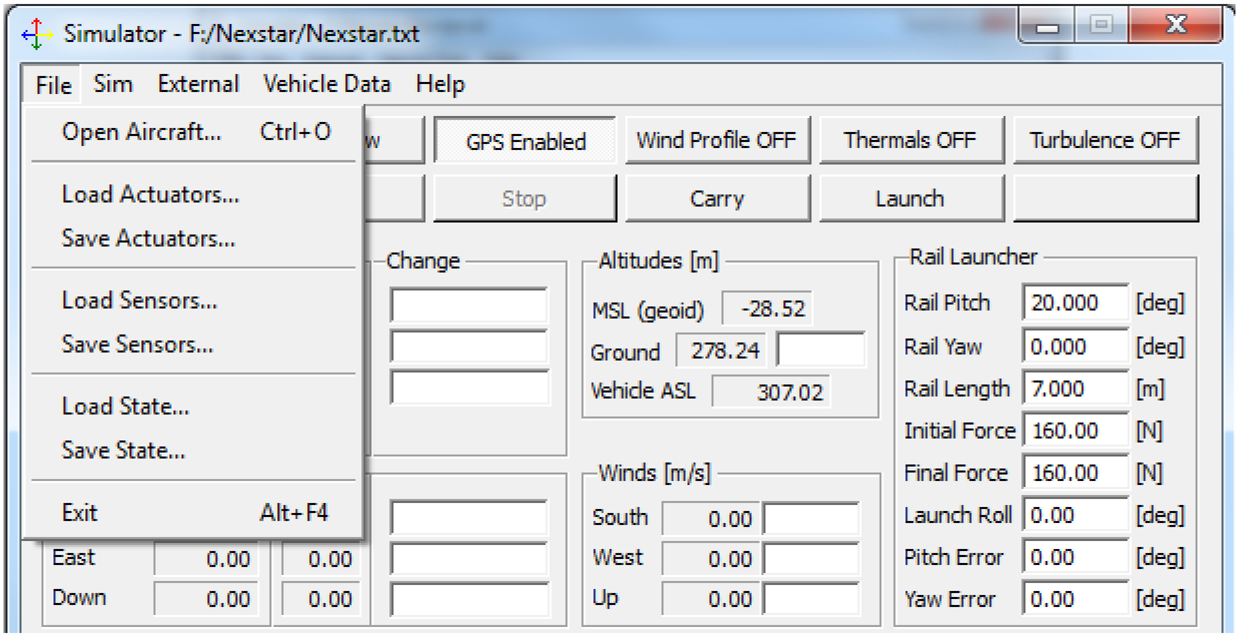
Aerodynamics	
CL stab axis	0.000
CL body axis	0.000
L/D	0.00
Mass[kg]	4.56

Engines		
Power[W]	Thrust[N]	Fuel[g/hr]
-0.0	0.00	2
0.0	0.00	0

Period [ms]			
Max	1.0	Avg	1.0

Autopilot Data	
Launch Action 1	OFF
Launch Action 2	OFF
Parachute	OFF
Comm Latency [ms]	0
LAileron	0.000
[Surface 1]	0.000
Throttle_0	0.000
Rudder	0.000
RAileron	0.000
[Surface 5]	0.000
[Surface 6]	0.000
[Surface 7]	0.000
[Surface 8]	0.000
RElevator	0.000
[Surface 10]	0.000
[Surface 11]	0.000
[Surface 12]	0.000
[Surface 13]	0.000
[Surface 14]	0.000
[Surface 15]	0.000

The simulator can load aircraft by loading a simulator text file through “File”, “Open Aircraft”.



Altitudes [m]

MSL (geoid) -28.52

Ground 278.24

Vehicle ASL 307.02

Winds [m/s]

South 0.00

West 0.00

Up 0.00

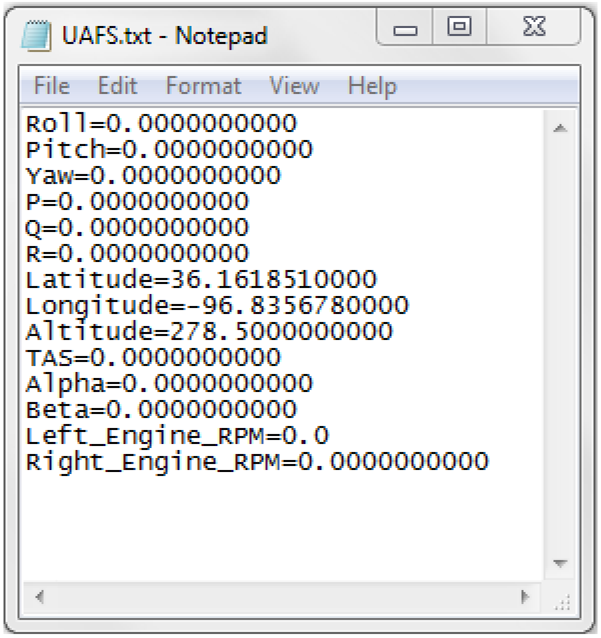
East 0.00 0.00

Down 0.00 0.00

Rail Launcher

Rail Pitch	20.000	[deg]
Rail Yaw	0.000	[deg]
Rail Length	7.000	[m]
Initial Force	160.00	[N]
Final Force	160.00	[N]
Launch Roll	0.00	[deg]
Pitch Error	0.00	[deg]
Yaw Error	0.00	[deg]

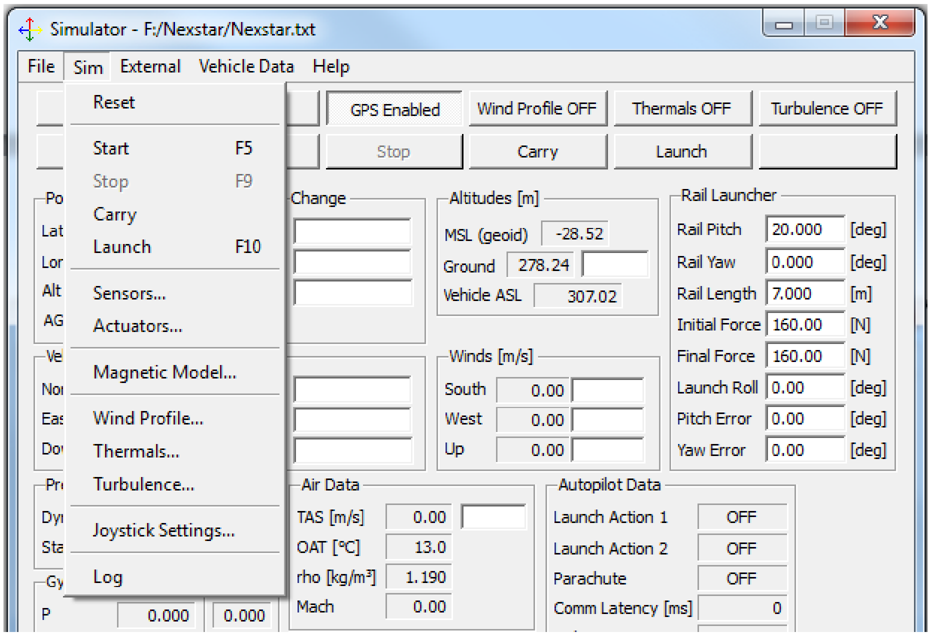
There are additional options to load Actuator, Sensor, and State text files. Piccolo software comes with built in actuator and sensor text files. They are located in the install directory “Simulator” folder. The software includes actuator files for “fast, sloppy, slow, standard, and very slow” actuators. The sensor files come with the options of “perfect, and poor” sensors. Actuator and sensor files can be called inside simulator text files or they can be loaded directly in the File drop down menu. The file menu also allows the user the ability to load state files. State files dictate the latitude and longitude of the aircraft, and they also can determine the orientation (roll, pitch, yaw, p, q, r, alpha, and beta) of the aircraft in its initial state.



```
Roll=0.0000000000
Pitch=0.0000000000
Yaw=0.0000000000
P=0.0000000000
Q=0.0000000000
R=0.0000000000
Latitude=36.1618510000
Longitude=-96.8356780000
Altitude=278.5000000000
TAS=0.0000000000
Alpha=0.0000000000
Beta=0.0000000000
Left_Engine_RPM=0.0
Right_Engine_RPM=0.0000000000
```

Figure 1 Simulator State File

Figure 1 is an example of a state file. The State File is simply a text file that pre defines the initial state of an aircraft. All of the parameters in the file can be changed in the Simulator interface. The state file is handy for setting the initial location of the aircraft at the UAS airfield.



Altitudes [m]
MSL (geoid) -28.52
Ground 278.24
Vehicle ASL 307.02

Winds [m/s]
South 0.00
West 0.00
Up 0.00

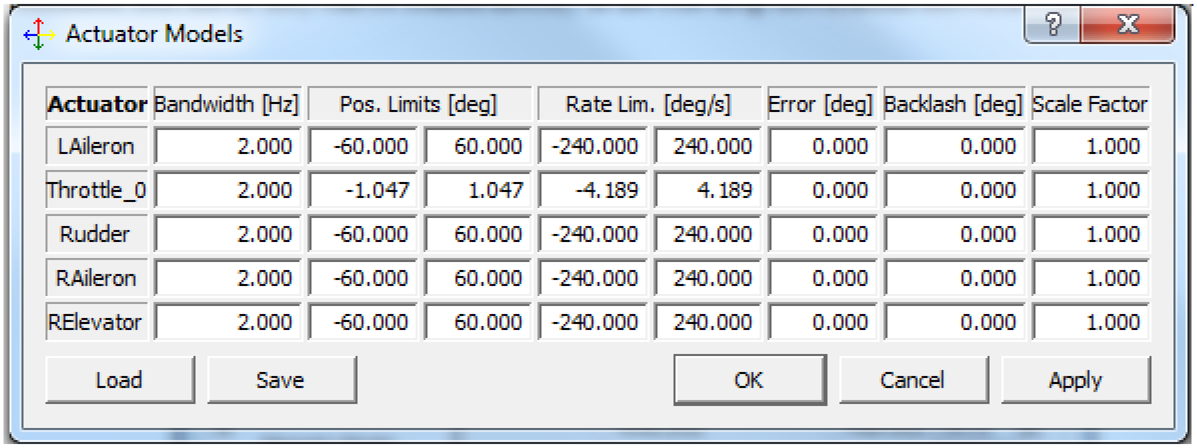
Rail Launcher
Rail Pitch 20.000 [deg]
Rail Yaw 0.000 [deg]
Rail Length 7.000 [m]
Initial Force 160.00 [N]
Final Force 160.00 [N]
Launch Roll 0.00 [deg]
Pitch Error 0.00 [deg]
Yaw Error 0.00 [deg]

Air Data
TAS [m/s] 0.00
OAT [°C] 13.0
rho [kg/m³] 1.190
Mach 0.00

Autopilot Data
Launch Action 1 OFF
Launch Action 2 OFF
Parachute OFF
Comm Latency [ms] 0

P 0.000 0.000

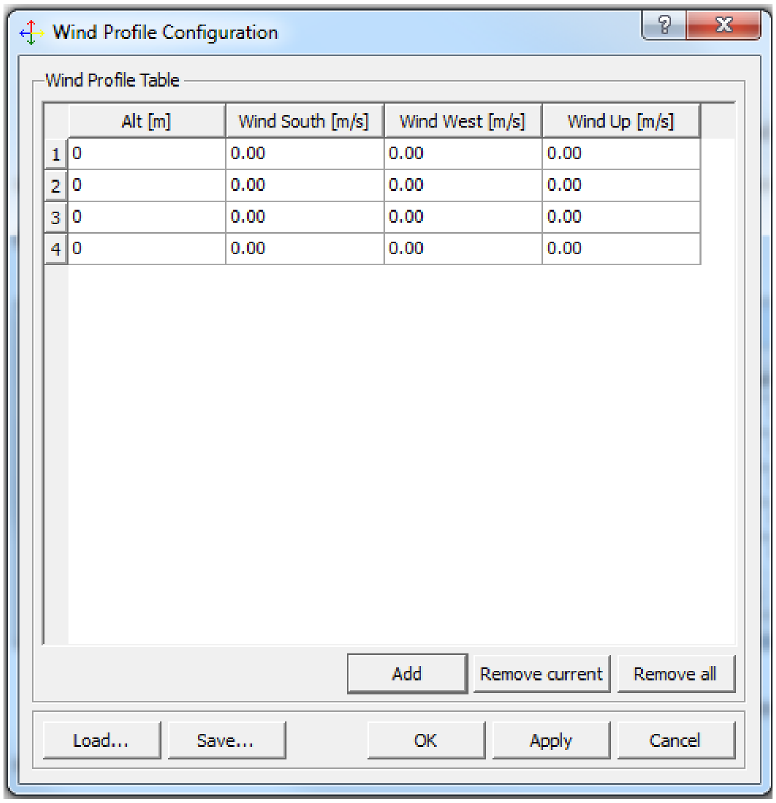
The actuator and sensor files dictate the performance of the actuators and sensors during the simulation. In the "Sim" drop down menu "Sensors" and "Actuators" show sensor and actuator models that are currently loaded.



Actuator	Bandwidth [Hz]	Pos. Limits [deg]		Rate Lim. [deg/s]		Error [deg]	Backlash [deg]	Scale Factor
LAileron	2.000	-60.000	60.000	-240.000	240.000	0.000	0.000	1.000
Throttle_0	2.000	-1.047	1.047	-4.189	4.189	0.000	0.000	1.000
Rudder	2.000	-60.000	60.000	-240.000	240.000	0.000	0.000	1.000
RAileron	2.000	-60.000	60.000	-240.000	240.000	0.000	0.000	1.000
RElevator	2.000	-60.000	60.000	-240.000	240.000	0.000	0.000	1.000

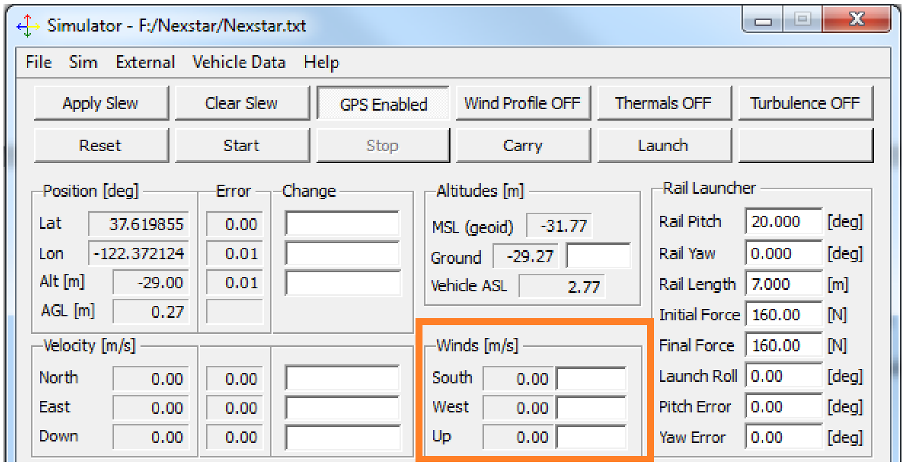
In the actuator and sensor model screens the user can edit any parameters desired. Actuator and sensor files can be saved as well.

Also in the "Sim" drop down menu is the "Wind Profile" options. There are two separate ways for the user to define wind profiles.



	Alt [m]	Wind South [m/s]	Wind West [m/s]	Wind Up [m/s]
1	0	0.00	0.00	0.00
2	0	0.00	0.00	0.00
3	0	0.00	0.00	0.00
4	0	0.00	0.00	0.00

"Wind Profile" opens the "Wind Profile Configuration" window. Here wind profiles can be used to define wind speeds at different altitudes. The user can "Add" and "Remove" as many points as desired. The user can also save and load different wind profiles. The piccolo software comes with a built in simulated atmosphere called "sample_atmosphere.xml". Sample atmosphere is located in the "Simulator" folder. Another way to create wind in the simulations is using the wind input boxes in the simulator display.



Simulator - F:/Nexstar/Nexstar.txt

Position [deg] | Error | Change
Lat 37.619855 | 0.00
Lon -122.372124 | 0.01
Alt [m] -29.00 | 0.01
AGL [m] 0.27

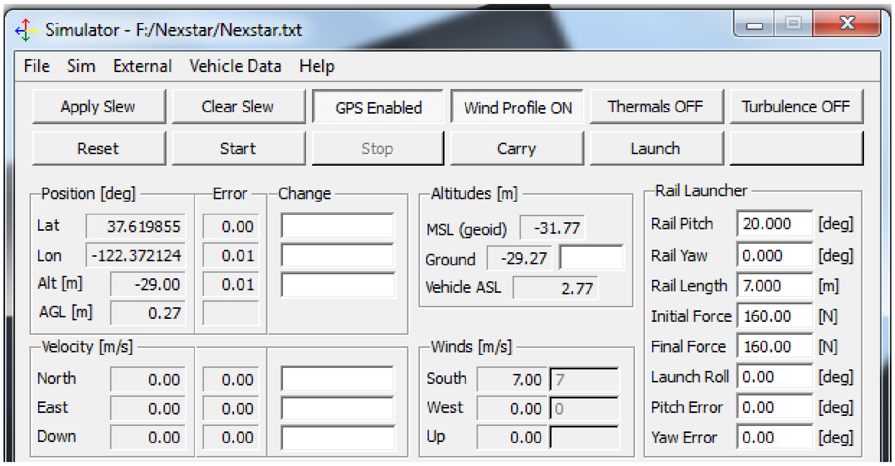
Velocity [m/s]
North 0.00 | 0.00
East 0.00 | 0.00
Down 0.00 | 0.00

Altitudes [m]
MSL (geoid) -31.77
Ground -29.27
Vehicle ASL 2.77

Winds [m/s]
South 0.00
West 0.00
Up 0.00

Rail Launcher
Rail Pitch 20.000 [deg]
Rail Yaw 0.000 [deg]
Rail Length 7.000 [m]
Initial Force 160.00 [N]
Final Force 160.00 [N]
Launch Roll 0.00 [deg]
Pitch Error 0.00 [deg]
Yaw Error 0.00 [deg]

Entering wind speeds in the wind section creates uniform wind speed that is applied to all altitudes. The direction is defined as the direction that the wind is blowing from, so a 10 m/s wind in the “South” input would create wind blowing North. To utilize the wind inputs just type in a number in the input box next to the appropriate direction (negative denotes the opposite direction). Then hit enter and click “Wind Profile OFF” which will turn into “Wind Profile ON” once clicked.



Simulator - F:/Nexstar/Nexstar.txt

Position [deg] | Error | Change
Lat 37.619855 | 0.00
Lon -122.372124 | 0.01
Alt [m] -29.00 | 0.01
AGL [m] 0.27

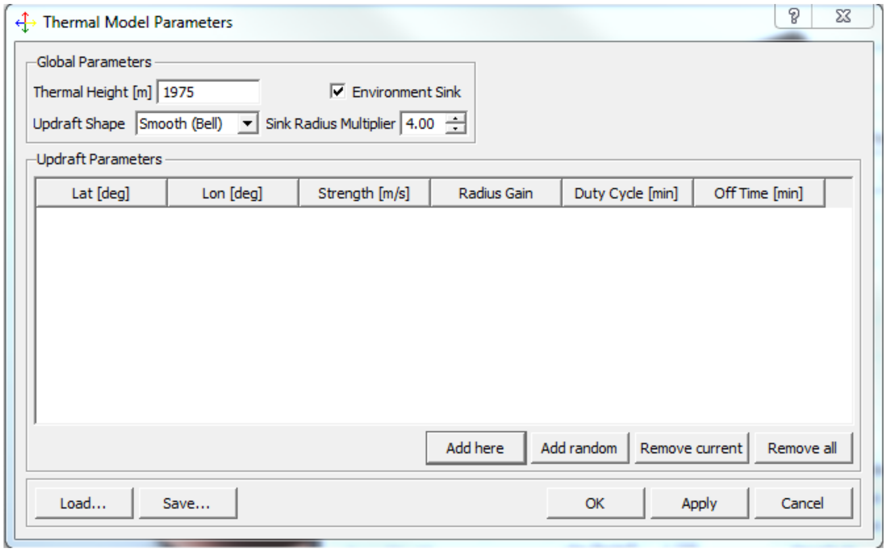
Velocity [m/s]
North 0.00 | 0.00
East 0.00 | 0.00
Down 0.00 | 0.00

Altitudes [m]
MSL (geoid) -31.77
Ground -29.27
Vehicle ASL 2.77

Winds [m/s]
South 7.00 | 7
West 0.00 | 0
Up 0.00

Rail Launcher
Rail Pitch 20.000 [deg]
Rail Yaw 0.000 [deg]
Rail Length 7.000 [m]
Initial Force 160.00 [N]
Final Force 160.00 [N]
Launch Roll 0.00 [deg]
Pitch Error 0.00 [deg]
Yaw Error 0.00 [deg]

The “Sim” dropdown menu also includes options for “Thermals” and “Turbulence”. Just like the previous items both thermals and turbulence windows can be used to create scenarios of thermals and turbulence.



Thermal Model Parameters

Global Parameters

Thermal Height [m] 1975

☑ Environment Sink

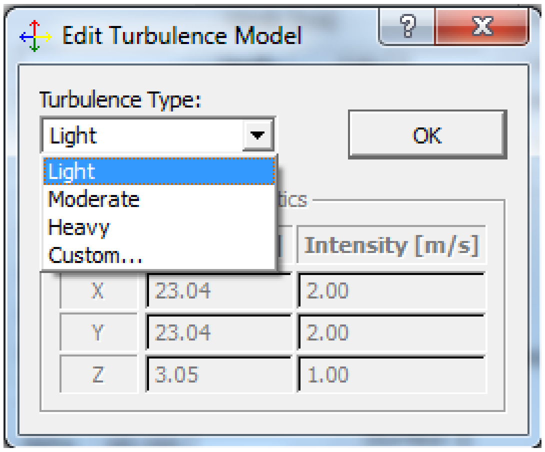
Updraft Shape Smooth (Bell)

Sink Radius Multiplier 4.00

Updraft Parameters

Lat [deg]	Lon [deg]	Strength [m/s]	Radius Gain	Duty Cycle [min]	Off Time [min]

The thermal model parameters window can automatically generate thermals, in addition to allowing the user to create his/her own thermals and load them from a saved thermal file.



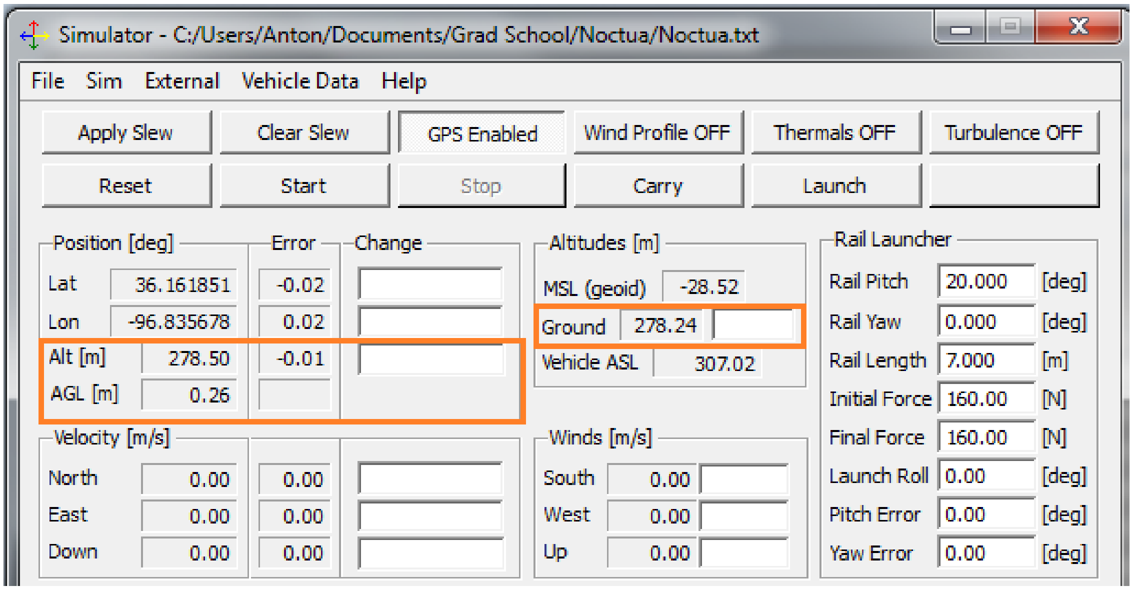
Edit Turbulence Model

Turbulence Type: Light

		Intensity [m/s]
X	23.04	2.00
Y	23.04	2.00
Z	3.05	1.00

The Turbulence model window allows the user to select from built in turbulence types in addition to creating his/her own custom model. In order to enable thermals and turbulence the user must click the “Thermals OFF” and “Turbulence OFF” buttons.

The “File” menu also allows the user the ability to load “State Files”. State files set the initial (pre-flight) latitude, longitude, and elevation of the aircraft, and they also can set the orientation (roll, pitch, yaw, p, q, r, alpha, and beta) and airspeed of the aircraft. All of the variables, except for p q r, can be set in the Simulator interface prior to launch.


Simulator - C:/Users/Anton/Documents/Grad School/Noctua/Noctua.txt

Position [deg]		Error	Change
Lat	36.161851	-0.02	
Lon	-96.835678	0.02	
Alt [m]	278.50	-0.01	
AGL [m]	0.26		

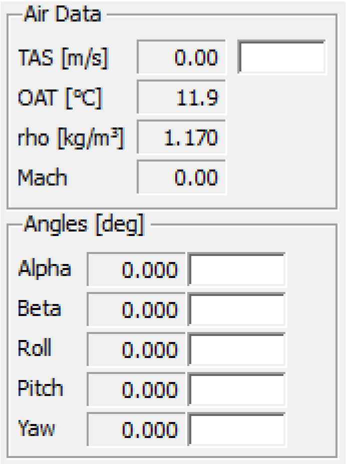
Velocity [m/s]			
North	0.00	0.00	
East	0.00	0.00	
Down	0.00	0.00	

Altitudes [m]		
MSL (geoid)	-28.52	
Ground	278.24	
Vehicle ASL	307.02	

Winds [m/s]		
South	0.00	
West	0.00	
Up	0.00	

Rail Launcher		
Rail Pitch	20.000	[deg]
Rail Yaw	0.000	[deg]
Rail Length	7.000	[m]
Initial Force	160.00	[N]
Final Force	160.00	[N]
Launch Roll	0.00	[deg]
Pitch Error	0.00	[deg]
Yaw Error	0.00	[deg]

In the “Position” section the current altitude is displayed along with the AGL (above ground level) of the aircraft. The simulator has internal elevations set for presumably every location in the earth. The estimated ground elevation is displayed in the “Altitudes” section. The user can change the altitude of the aircraft in the input box next to “Alt”. Hit enter after the value has been input to see the AGL adjust properly. The input box next to “Ground” in the “Altitudes section allows the user to change the elevation of the ground at the current location. Note that the ground altitudes will change automatically when the user changes the location via altering “Lat” and “Lon”.


Air Data

TAS [m/s] 0.00

OAT [ºC] 11.9

rho [kg/m³] 1.170

Mach 0.00

Angles [deg]

Alpha 0.000

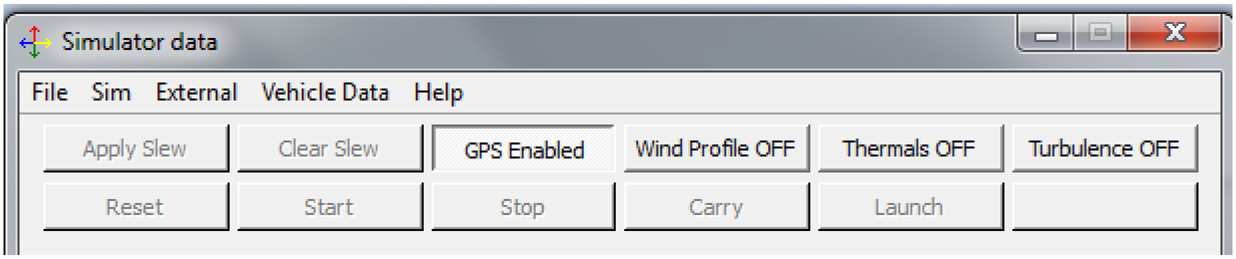
Beta 0.000

Roll 0.000

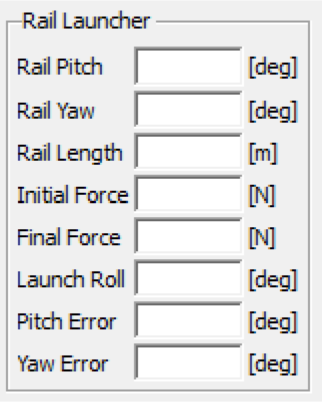
Pitch 0.000

Yaw 0.000

The initial airspeed and orientations can also be set in the Simulator user interface in the "Air Data" and "Angles" sections. Just as before input the desired values into the input boxes and hit enter. The displayed air data, OAT (outside air temperature), and rho (air density), are set automatically by the Simulator based on the altitude of the aircraft.



The user also has the option to disable "GPS". When GPS is disabled the autopilot's navigation solution will fly in "AHRS" mode where it uses inertial measurements and magnetometer readings.



The Simulator interface also allows the user the option of simulating a rail launch. If there are no parameters entered in the rail launch the simulator will simulate a wheeled launch based on the launch settings set on the autopilot or the simulated autopilot (SiL). Launch settings are located in the “Launch” tab of the “Controller Configuration” window in PCC.

Before beginning the user will have to click the “Apply Slew” button to apply all of the initial settings. To start the simulation the user can either click the “Launch” button or the “Start” button. The “Start” button just begins the simulation with the aircraft in whatever location and orientation is set in the Simulator settings instead of actually launching the aircraft. Section 4.7 and 4.8 cover how to setup SiL and HiL simulations in greater detail.

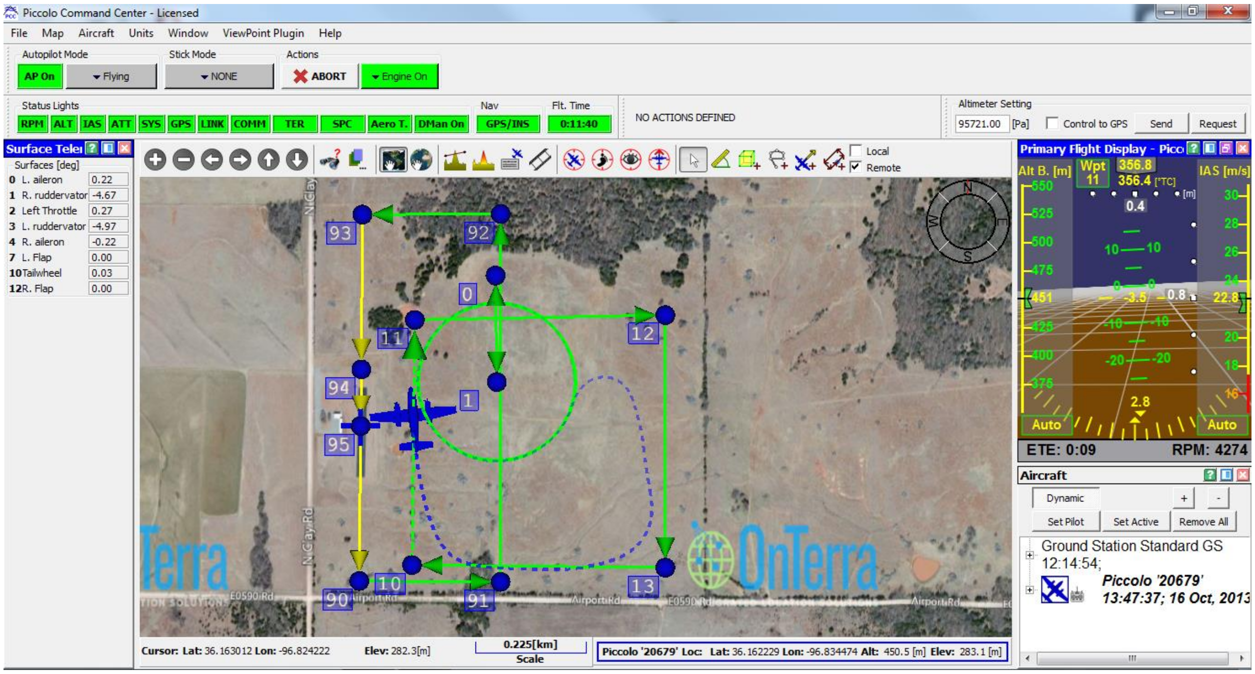
2.5. PCC (Piccolo Command Center)

The Piccolo autopilot system uses Cloud Cap’s Piccolo Command Center (PCC) as the interface between the remote pilot and the autopilot. All commands and settings are set on the autopilot through PCC. PCC is a very powerful tool with many different applications.

PCC can communicate with multiple piccolo units simultaneously. In addition to displaying live telemetry data PCC records a telemetry file (“.tel”) for each unit that is connected. The tel files can be used to replay any flight. The replay will show all the commands and telemetry data that took place throughout a given flight. Also whenever PCC is running, in a replay or live flight, almost all of the telemetry data is recorded in a massive text file (".log"). The text file is referred to as the flight log file.

Cloud Cap provides a user's guide for the functionality of PCC, "PccUsersGuide.pdf" (pgs. 1 -99). The following sections highlight and elaborate on some of the more important operational aspects of PCC.

2.5.1. Main Screen



Altimeter Setting: 95721.00 [Pa]

Surfaces [deg]

0	L. aileron	0.22
1	R. ruddervator	-4.67
2	Left Throttle	0.27
3	L. ruddervator	-4.97
4	R. aileron	-0.22
7	L. Flap	0.00
10	Tailwheel	0.03
12	R. Flap	0.00

Ground Station Standard GS 12:14:54;

Piccolo '20679' 13:47:37; 16 Oct, 2013

Cursor: Lat: 36.163012 Lon: -96.824222 Elev: 282.3[m] 0.225[km]

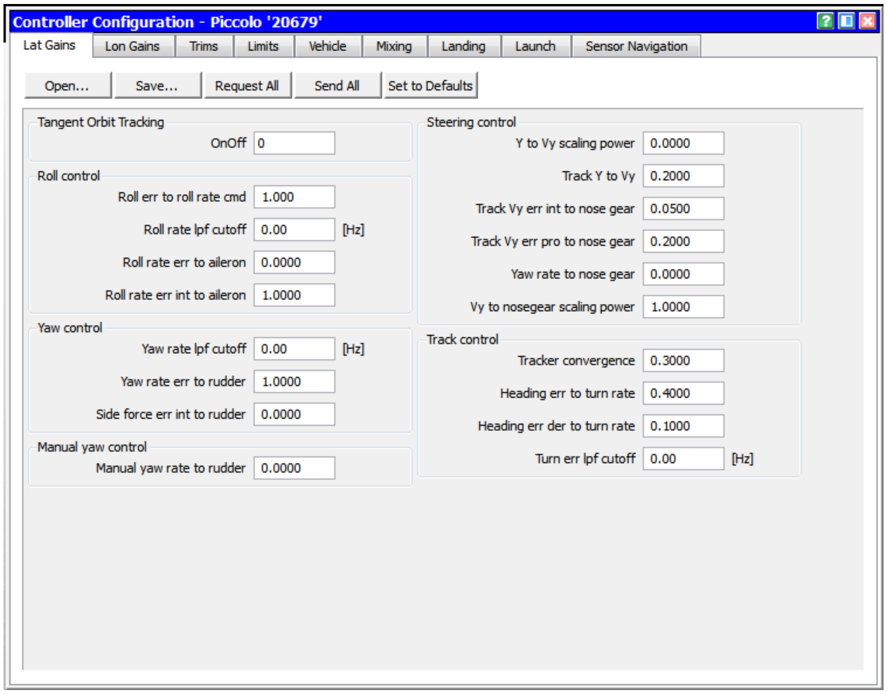
Piccolo '20679' Loc: Lat: 36.162229 Lon: -96.834474 Alt: 450.5 [m] Elev: 283.1 [m]

Figure 2 PCC Main Screen

The main screen that is viewed when flying in PCC will display map data (if map layers are uploaded), flight plans, and the currently piloted aircraft. The configuration shown above has Surface Telemetry, Aircraft, and the Primary Flight Display (PFD) docked in the main screen. Docked windows are movable, and other windows may be docked as well. The latitude, longitude, and elevation data at the bottom represent where the mouse cursor is currently pointing. The blue highlighted area the bottom shows the latitude, longitude, altitude, and elevation of the piccolo unit that is currently "active".

Notice in Figure 2 that there are three different flight plans. Waypoints 10-13 are a box pattern. Waypoints 0-1 is a loitering flight plan that is defined so that the aircraft indefinitely loiters around waypoint 1. Waypoints 90-95 define the current landing plan. The yellow track lines signify that the flight path travels below the minimum altitude set in the Mission Limits window. The blue cross waypoint, waypoint 95, denotes that the waypoint is the touchdown waypoint in the land plan. If a waypoint lies below ground level the track line will be red.

2.5.2. Controller Configuration Window



Tangent Orbit Tracking
OnOff 0

Roll control
Roll err to roll rate cmd 1.000
Roll rate lpf cutoff 0.00 [Hz]
Roll rate err to aileron 0.0000
Roll rate err int to aileron 1.0000

Yaw control
Yaw rate lpf cutoff 0.00 [Hz]
Yaw rate err to rudder 1.0000
Side force err int to rudder 0.0000

Manual yaw control
Manual yaw rate to rudder 0.0000

Steering control
Y to Vy scaling power 0.0000
Track Y to Vy 0.2000
Track Vy err int to nose gear 0.0500
Track Vy err pro to nose gear 0.2000
Yaw rate to nose gear 0.0000
Vy to nosegear scaling power 1.0000

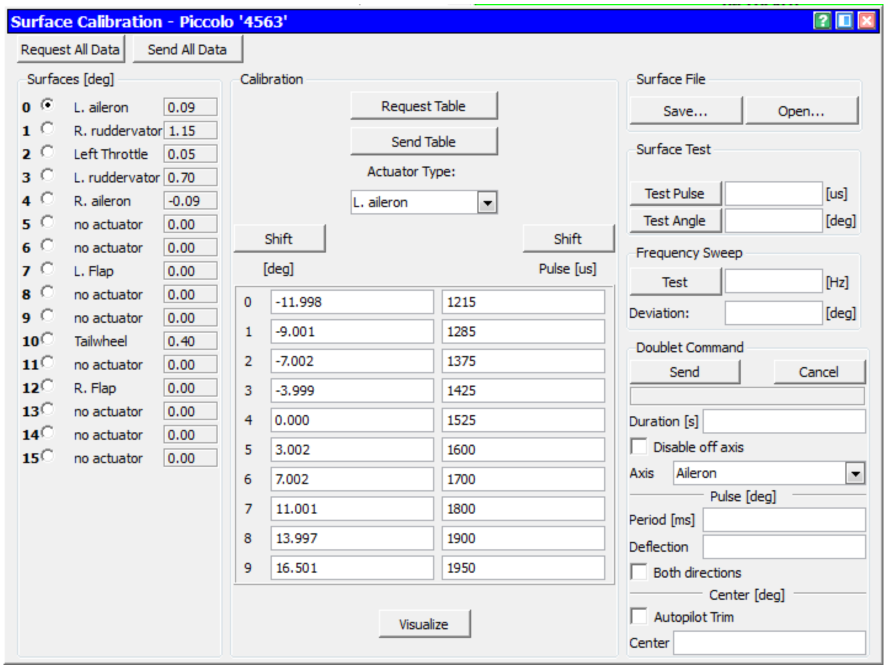
Track control
Tracker convergence 0.3000
Heading err to turn rate 0.4000
Heading err der to turn rate 0.1000
Turn err lpf cutoff 0.00 [Hz]

The Controller Configuration Window can be opened from the PCC toolbar under Window > Advanced Windows> Controller Configuration. The Controller Configuration Window displays and allows the User to change gains, limits, and landing and launch parameters. Each time that a tab is clicked PCC will "Request" the current settings for that tab from the autopilot. Click “Request All” at any time in any tab to download the current settings from the autopilot.

To change a gain or setting simply type in the box of the desired gain to be changed. The box where the change occurred will be highlighted red until the new settings have been sent to the autopilot. Click “Send All” to send the gains or settings to the autopilot.

“Open” and “Save” will allow the user to load and save gains and settings for each tab to and from a file. Similar to manually changing gains, when a file is loaded all of the boxes that change will be highlighted red with their new values in store. Also similar to manually changing gains the new values must be sent to the autopilot via “Send All”.

2.5.3. Surface Calibration Window



Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	●	L. aileron	0.09
1	○	R. ruddervator	1.15
2	○	Left Throttle	0.05
3	○	L. ruddervator	0.70
4	○	R. aileron	-0.09
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.40
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: L. aileron

	Shift [deg]	Shift Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File

Surface Test

Test Pulse [us]

Test Angle [deg]

Frequency Sweep

Test [Hz]

Deviation: [deg]

Doublet Command

Duration [s]

☐ Disable off axis

Axis Aileron

Pulse [deg]

Period [ms]

Deflection

☐ Both directions

Center [deg]

☐ Autopilot Trim

Center

The Surface Calibration Window can be opened from the PCC toolbar under Window > Advanced Windows> Surface Calibration. The Surface Calibration window is the interface tool for the user to assign control surfaces to specific servo lines. In addition to declaring control surfaces this window is also where the control surfaces are calibrated. Surfaces are calibrated by commanding a pulse width signal and measuring the angle of deflection that responds. The table in the center of the window serves as a look up table for the autopilot when commanding surface deflections. The servo lines are defined in the Cloud Cap document “Piccolo External Interface”. The Surface Calibration window is also where doublet commands can be sent for doublet maneuvers.

2.5.4. Creating PCC Flight Plans

Flight plans are represented by track lines connecting waypoints. The direction of the flight plan is denoted by the arrow on the track lines or flight path. The flight path lines are the lines that the autopilot will track as the aircraft travels through the waypoints.

The main screen can display flight plans on two different modules. “Local” flight plans are flight plans that exist on the PCC computer only and have not yet been sent to the autopilot. “Remote” flight plans are the flight plans that are present on the autopilot unit. Check the box next to “Local” or “Remote” to view the flight plans. In Figure 3 below the grey colored waypoints symbolize that the flight plan exists only in the “Local” directory. The autopilot cannot target waypoints that are not present on Remote.

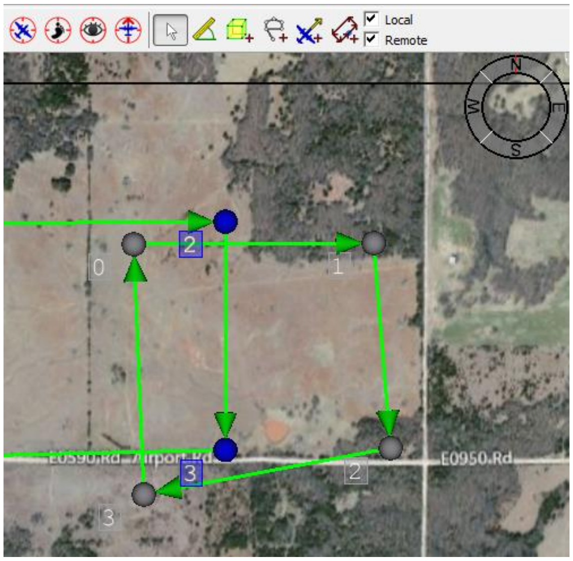
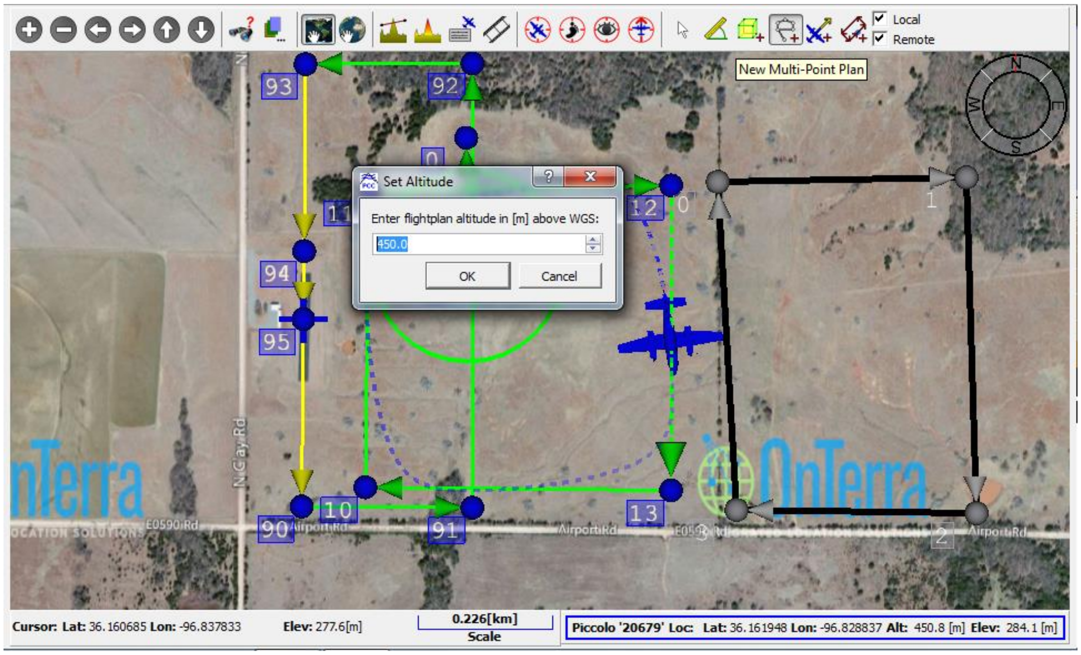


Figure 3 New Local Waypoints

There are two methods for creating flight plans. One is using the 'New Multi-Point Plan" button on the Map Action Bar. The other is "Quick Flight Plan" also on the Map Action Bar. Quick Flight Plan only creates a two waypoint flight plan and automatically sends the waypoints to the autopilot. New Multi-Point Plan is used for creating real flight plans. Note that when creating a New Multi-Point Plan the waypoints will be Local.



Set Altitude

Enter flightplan altitude in [m] above WGS:

450.0

OK Cancel

Cursor: Lat: 36.160685 Lon: -96.837833 Elev: 277.6[m] 0.226[km] Scale

Piccolo '20679' Loc: Lat: 36.161948 Lon: -96.828837 Alt: 450.8 [m] Elev: 284.1 [m]

Figure 4 Flight Plan Altitude

After "New Multi-Point Plan" has been clicked each mouse click on the map will create a new waypoint where the mouse cursor was clicked. Double click the last waypoint to declare that the flight plan is completed and a pop window will ask for the flight plan altitude. Initially waypoints will have the same altitude; however, waypoint altitudes can be changed at any time by right clicking a way point and selecting "Edit Waypoint".

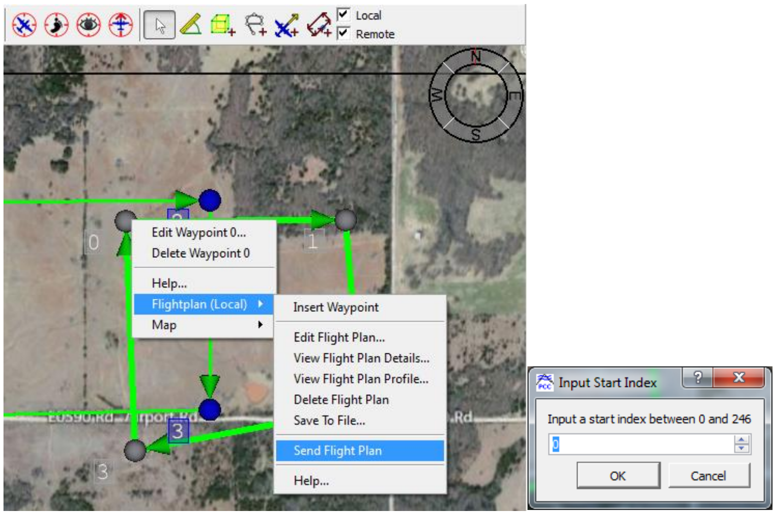
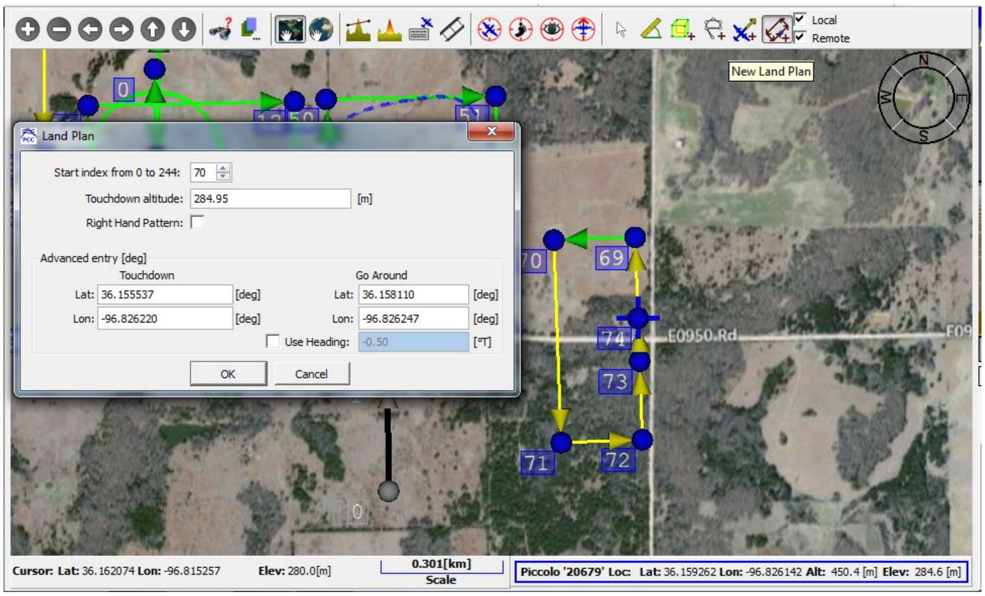


Figure 5 Send Local Flight Plan

To send a Local flight plan to the autopilot, or "Remote", right click any waypoint on the Local flight plan desired, highlight "Flight Plan" and click "Send Flight Plan". A box will pop up asking for the index number to number the waypoints with. The number that is input into the box will be the starting number for the first waypoint in the flight plan. Make sure that the numbering doesn't interfere with any existing Remote waypoints. If it does PCC will issue a warning and a second confirmation request to send over.

2.5.5. Creating Land Plans

Land plans do more than just place a waypoint at ground level. Land plans contain different control logic than other flight plans, thus land plans cannot be created using normal waypoints. This section describes the logistics of creating land plans in PCC.



Land Plan

Start index from 0 to 244: 70

Touchdown altitude: 284.95 [m]

Right Hand Pattern: □

Advanced entry [deg]

	Touchdown			Go Around	
Lat:	36.155537	[deg]	Lat:	36.158110	[deg]
Lon:	-96.826220	[deg]	Lon:	-96.826247	[deg]
			□ Use Heading:	-0.50	[°T]

Cursor: Lat: 36.162074 Lon: -96.815257 Elev: 280.0[m]

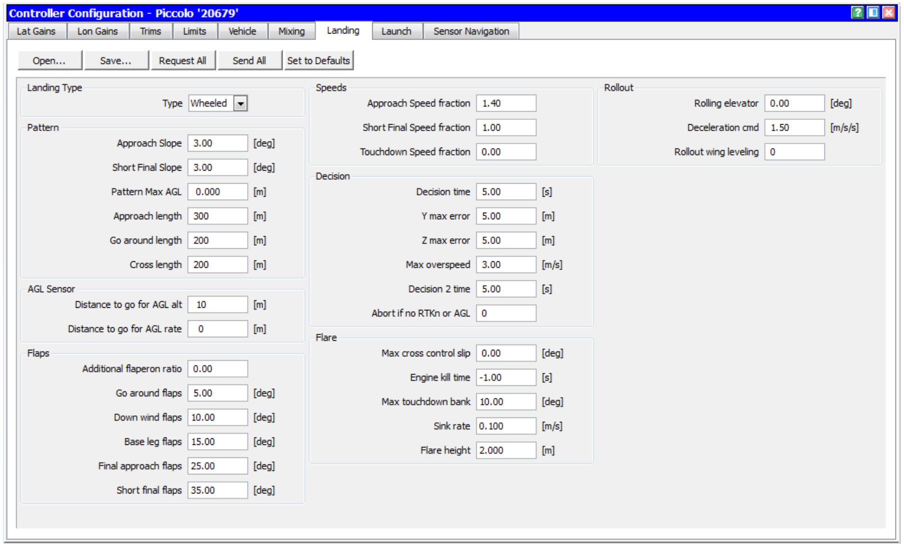
0.301[km] Scale

Piccolo '20679' Loc: Lat: 36.159262 Lon: -96.826142 Alt: 450.4 [m] Elev: 284.6 [m]

Figure 6 Create A New Landplan

When creating a new land plan the user is required to click in two locations. The first location is where the touchdown waypoint is to be located, and the second click designates the direction of the final approach. In Figure 6 the waypoint with a cross symbolizes the touchdown waypoint. After the two clicks the Land Plan window will pop up offering a number of editing options and asking for the number that the first waypoint in the land plan should be numbered. If there are specific coordinates for touchdown and go around the clicked locations can be adjusted here. The heading (direction) of the final approach can also be adjusted.

After clicking “OK” PCC will draw out the landing plan according to pattern settings defined in the “Landing” tab of the Controller Configuration Window.



Landing Type

Type Wheeled

Pattern

Approach Slope 3.00 [deg]

Short Final Slope 3.00 [deg]

Pattern Max AGL 0.000 [m]

Approach length 300 [m]

Go around length 200 [m]

Cross length 200 [m]

AGL Sensor

Distance to go for AGL alt 10 [m]

Distance to go for AGL rate 0 [m]

Flaps

Additional flaperon ratio 0.00

Go around flaps 5.00 [deg]

Down wind flaps 10.00 [deg]

Base leg flaps 15.00 [deg]

Final approach flaps 25.00 [deg]

Short final flaps 35.00 [deg]

Speeds

Approach Speed fraction 1.40

Short Final Speed fraction 1.00

Touchdown Speed fraction 0.00

Decision

Decision time 5.00 [s]

Y max error 5.00 [m]

Z max error 5.00 [m]

Max overspeed 3.00 [m/s]

Decision 2 time 5.00 [s]

Abort if no RTKn or AGL 0

Flare

Max cross control slip 0.00 [deg]

Engine kill time -1.00 [s]

Max touchdown bank 10.00 [deg]

Sink rate 0.100 [m/s]

Flare height 2.000 [m]

Rollout

Rolling elevator 0.00 [deg]

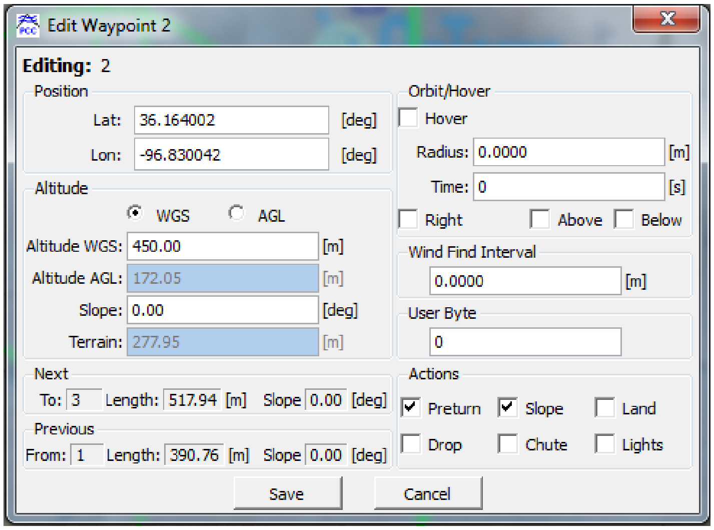
Deceleration cmd 1.50 [m/s/s]

Rollout wing leveling 0

All of the inputs in the section “Pattern” are used by PCC to draw the landing plan around the touchdown waypoint. Any of the waypoints in a landing plan can be edited and moved just like regular waypoints.

2.5.6. Editing Waypoints/Creating Loiter Waypoints

The Edit Waypoint window, shown below in Figure 7, displays current settings and information on the distance and slope of the track paths to and from the current waypoint.



Editing: 2

Position

Lat: 36.164002 [deg]

Lon: -96.830042 [deg]

Altitude

● WGS ○ AGL

Altitude WGS: 450.00 [m]

Altitude AGL: 172.05 [m]

Slope: 0.00 [deg]

Terrain: 277.95 [m]

Next

To: 3 Length: 517.94 [m] Slope 0.00 [deg]

Previous

From: 1 Length: 390.76 [m] Slope 0.00 [deg]

Orbit/Hover

□ Hover

Radius: 0.0000 [m]

Time: 0 [s]

□ Right □ Above □ Below

Wind Find Interval

0.0000 [m]

User Byte

0

Actions

☑ Preturn ☑ Slope □ Land

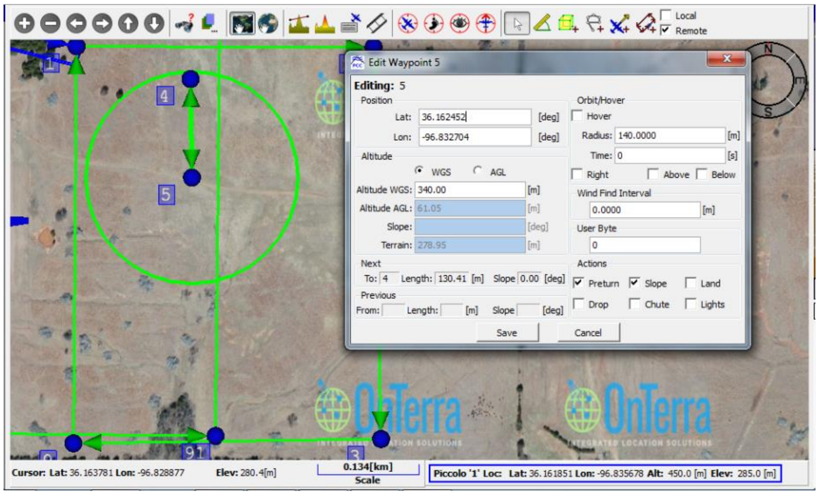
□ Drop □ Chute □ Lights

Figure 7 Edit Waypoint Window

Altitude. Specifying an altitude must be done with caution. The altitude can be defined as “AGL” (above ground level) or “WGS”. If the altitude is defined as AGL the autopilot will use the AGL sensor; however, if there is no AGL sensor the autopilot will use the estimated ground elevation from PCC’s loaded elevation maps and the measured altitude (from Barometer or GPS) to determine AGL. If it is desirable to define a waypoint at a certain altitude above ground level it is possible to set the altitude as “AGL” and then re define the waypoint as “WGS” to avoid using AGL to target the waypoint. To do so click “AGL”, fill in the input box, then click “WGS”.

Slope. Designating a slope in altitude will define the altitude of the selected waypoint such that the targeted flight path from the previous waypoint to the current waypoint will match the slope.

Loiter (Orbit). To define a waypoint as a loitering waypoint simply input a value for the radius. Time sets the amount of time that the aircraft orbits before moving on to the next waypoint in the flight plan. If time is 0 the autopilot will loiter indefinitely until the user targets a different waypoint. If it is desired to make a flight plan for loitering only one waypoint, such as the lost communications waypoint, the user will have to create two waypoints. PCC doesn’t allow the creation of one waypoint flight plans. In such a scenario time equal to 0 would keep the aircraft orbiting the loiter waypoint only and ignoring the second waypoint in the flight plan. Figure 8 shows a lost communications flight plan where it was desired to loiter indefinitely.



Edit Waypoint 5

Editing: 5

Position

Lat: 36.162452 [deg]

Lon: -96.832704 [deg]

Altitude

● WGS ○ AGL

Altitude WGS: 340.00 [m]

Altitude AGL: 61.05 [m]

Slope: [deg]

Terrain: 278.95 [m]

Next

To: 4 Length: 130.41 [m] Slope 0.00 [deg]

Previous

From: Length: [m] Slope [deg]

Orbit/Hover

☐ Hover

Radius: 140.0000 [m]

Time: 0 [s]

☐ Right ☐ Above ☐ Below

Wind Find Interval

0.0000 [m]

User Byte

0

Actions

☑ Preturn ☑ Slope ☐ Land

☐ Drop ☐ Chute ☐ Lights

Save Cancel

Cursor: **Lat:** 36.163781 **Lon:** -96.828877 **Elev:** 280.4[m]

0.134[km] Scale

Piccolo '1' Loc: **Lat:** 36.161851 **Lon:** -96.835678 **Alt:** 450.0 [m] **Elev:** 285.0 [m]

Figure 8 Indefinite Loiter

The default direction of orbiting loiter waypoints is counter clockwise (left). Selecting "Right" in the Orbit/Hover section will change the direction of orbit to clockwise.

<u>Preturn.</u> Preturn defines the behavior of the aircraft as it approaches and passes the targeted waypoint. If preturn is selected the aircraft won't actually fly through the waypoint. Figure 9 below shows a preturn. The preturn is initiated depending on the location and direction of the flight path for the next way point.

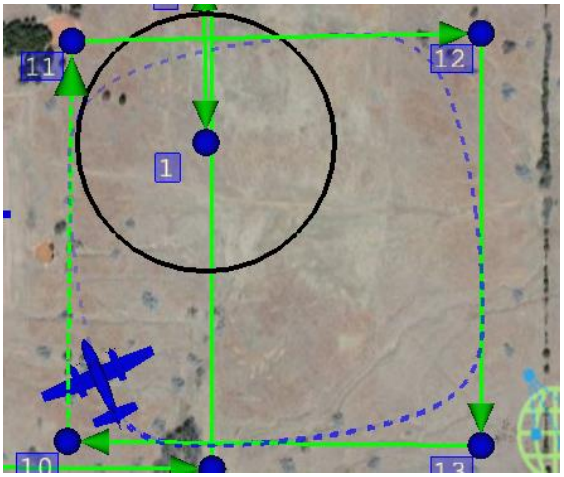


Figure 9 Preturn

If preturn is not selected the aircraft will fly through each waypoint, and then target the next flight path as shown below.

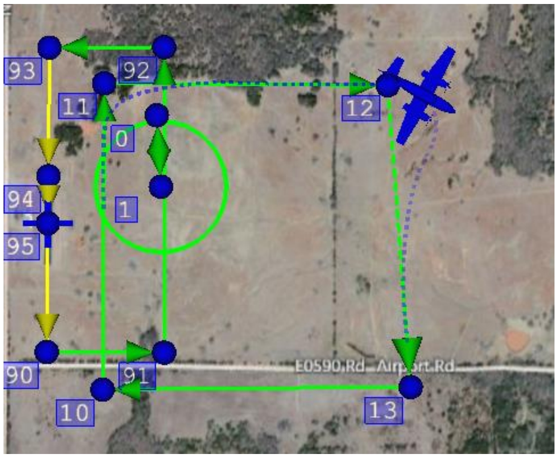


Figure 10 No Preturn

Slope Checkbox. The Slope checkbox is checked by default. This checkbox determines the manner in which the autopilot will attempt to climb or descend to the selected waypoint. By checking Slope the autopilot will attempt to climb or descend at a constant rate between the selected and previous waypoints. This also has to do with the command loop "VRate" (vertical rate).

2.5.7. Navigating Waypoints

There are two different methods using the main screen display to command the autopilot to target a new waypoint. Right clicking a waypoint will present the user with two options: "Track to Waypoint" and "Go To Waypoint".

Track To Waypoint. "Track to" will command the autopilot to target the designated flight path that approaches the waypoint in its flight plan. In Figure 11 the autopilot was commanded to "Track To" Waypoint 0.

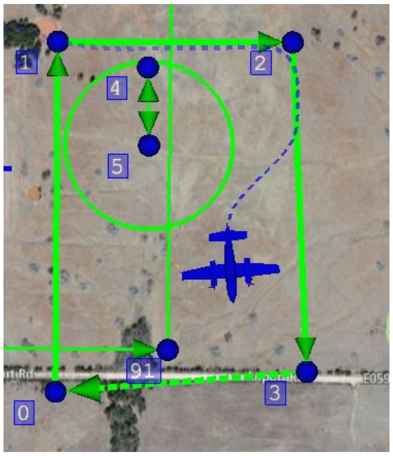


Figure 11 Track To Waypoint

Go To Waypoint. "Go to" will command the autopilot to track a straight line from its present location directly to the waypoint. Figure 12 shows the autopilot being commanded to "Go To" waypoint 13.

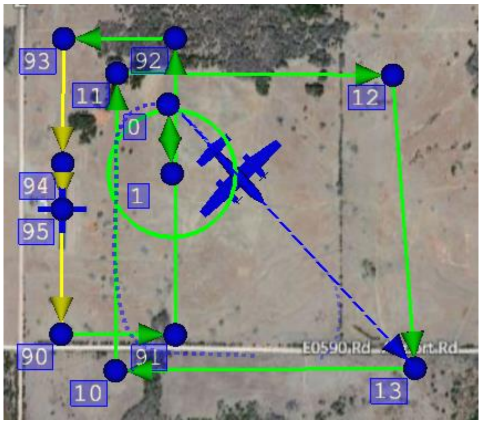


Figure 12 Go To Waypoint

Note that in both cases if the targeted waypoint or flight path is at a different altitude than the aircraft, the autopilot will command the new altitude as a step input; thus, the aircraft will climb or descend at the maximum or minimum vertical rate.

2.5.8. Aircraft Label

PCC allows the optional use of an Aircraft Label. The Aircraft Label displays a blue box next to the aircraft on the main screen with telemetry data. The label can be turned on or off via a button in the action bar highlighted in Figure 13.

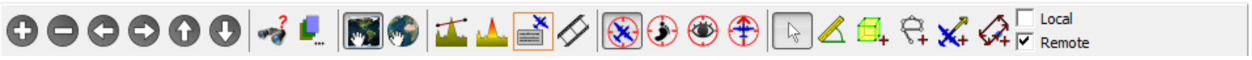
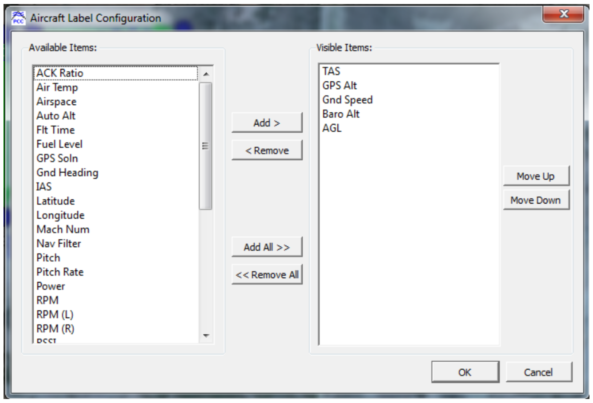


Figure 13 Aircraft Label

The telemetry that the label displays can be customized in the Aircraft Label Configuration window located in the menu bar of PCC, Aircraft > Select Custom Telemetry.



Available Items:

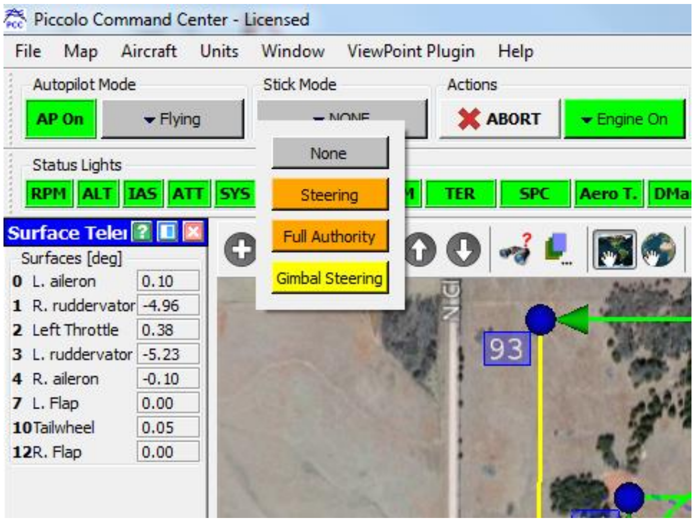
ACK Ratio
Air Temp
Airspace
Auto Alt
Flt Time
Fuel Level
GPS Soln
Gnd Heading
IAS
Latitude
Longitude
Mach Num
Nav Filter
Pitch
Pitch Rate
Power
RPM
RPM (L)
RPM (R)

Visible Items:

TAS
GPS Alt
Gnd Speed
Baro Alt
AGL

2.5.9. Manual Steering Mode

The manual r/c pilot has the authority to take over at any time via the remote control transmitter; however, manual pilot authority can be restricted to steering only.



Surfaces [deg]

0	L. aileron	0.10
1	R. ruddervator	-4.96
2	Left Throttle	0.38
3	L. ruddervator	-5.23
4	R. aileron	-0.10
7	L. Flap	0.00
10	Tailwheel	0.05
12	R. Flap	0.00

Figure 14 Manual Steering Mode

By selecting “Steering”, as shown in Figure 14, the manual pilot will only be able to control the ailerons and rudder. This feature could be useful for many different applications. In the event of an aircraft not able to maintain steady level flight this could be used to keep the plane straight while performing longitudinal gain tuning methods.

2.5.10. Primary Flight Display (PFD)

The Primary Flight Display is described in PccUsersGuide.pdf pgs (55-56). It is important to note that there are several ways to view the command altitude and airspeeds on the PFD. One method is to simply mouse over the area with the green slider bar. Figure 15 is a screen capture of a mouse over of the commanded altitude.

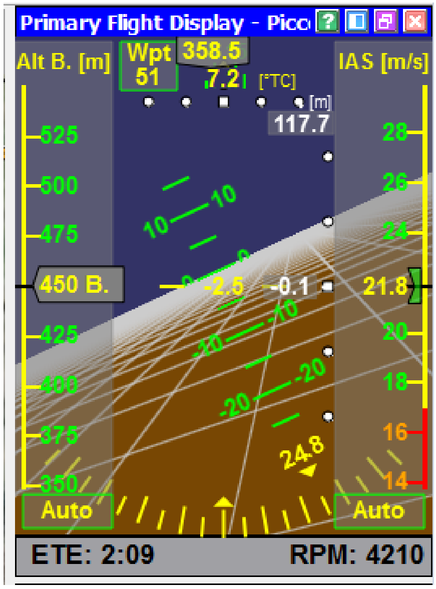
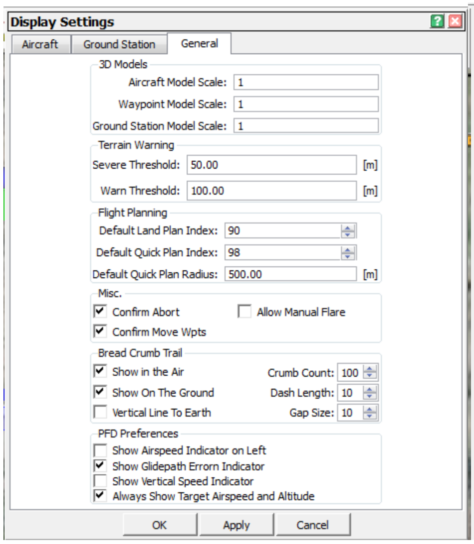


Figure 15 PFD

Another method is to change one of the PFD settings found in the General tab of the Display Settings (File > Display Settings). Select "Always Show Target Airspeed and Altitude".



Display Settings

Aircraft | Ground Station | General

3D Models
Aircraft Model Scale: 1
Waypoint Model Scale: 1
Ground Station Model Scale: 1

Terrain Warning
Severe Threshold: 50.00 [m]
Warn Threshold: 100.00 [m]

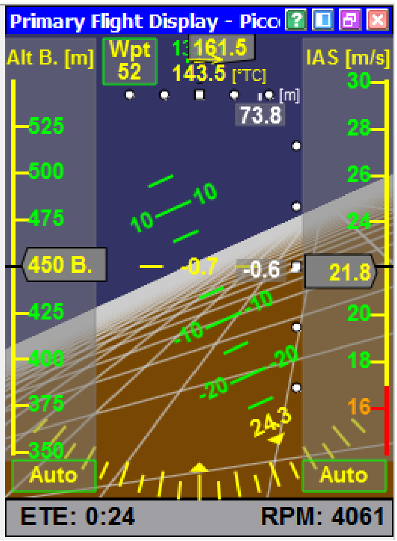
Flight Planning
Default Land Plan Index: 90
Default Quick Plan Index: 98
Default Quick Plan Radius: 500.00 [m]

Misc.
☑ Confirm Abort ☐ Allow Manual Flare
☑ Confirm Move Wpts

Bread Crumb Trail
☑ Show in the Air Crumb Count: 100
☑ Show On The Ground Dash Length: 10
☐ Vertical Line To Earth Gap Size: 10

PFD Preferences
☐ Show Airspeed Indicator on Left
☑ Show Glidepath Errorn Indicator
☐ Show Vertical Speed Indicator
☑ Always Show Target Airspeed and Altitude

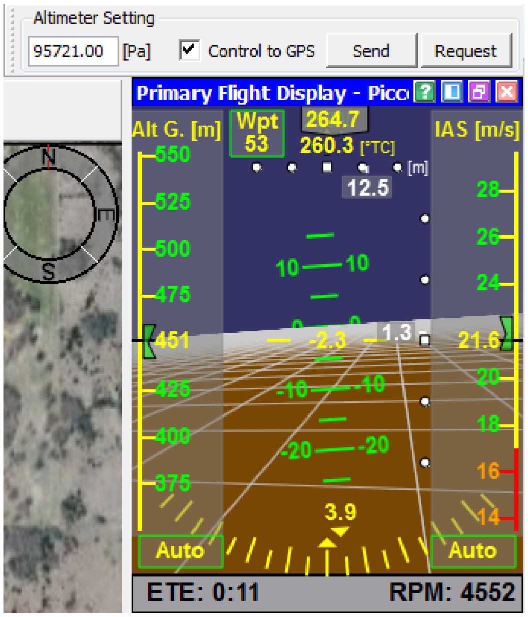
OK | Apply | Cancel



Another important thing is notice "Alt B. (m)" in the upper left hand corner of the PFD in Figure 15. This graphic will display whether the Barometer or GPS is being used to determine altitude.

2.5.11. Declaring GPS or Barometer for Altitude Sensor

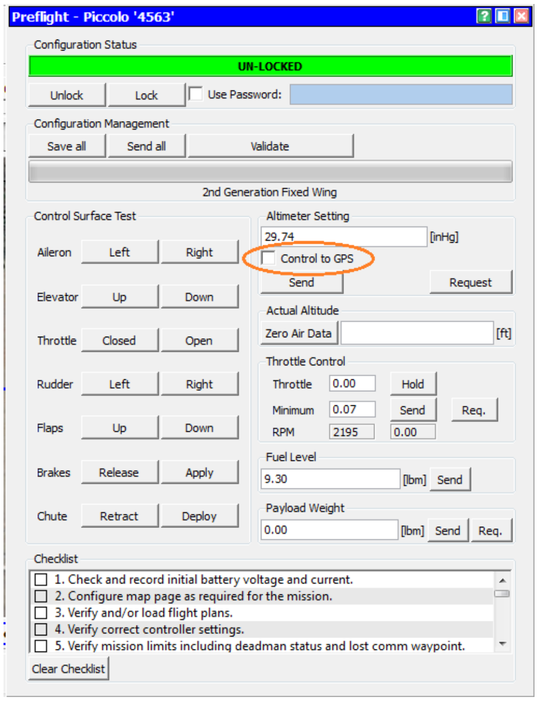
The Altimeter bar is displayed at the top of the main screen in PCC. Checking the “Control to GPS” box and clicking send will force the autopilot to use GPS for determining altitude. If the box is unchecked the autopilot will use the Barometer for determining altitude. Figure 16 shows an example of an autopilot in GPS Altitude Control. Notice that the upper left hand corner of the PFD displays “Alt G. (m)” rather than “Alt B.”.



Altimeter Setting

95721.00 [Pa] ☑ Control to GPS

Figure 16 PFD GPS Altitude Control



☐ Use Password:

Altimeter Setting: 29.74 [inHg]

☐ Control to GPS

Actual Altitude: [ft]

Throttle Control: Throttle 0.00; Minimum 0.07; RPM 2195 0.00

Fuel Level: 9.30 [lbm]

Payload Weight: 0.00 [lbm]

Checklist

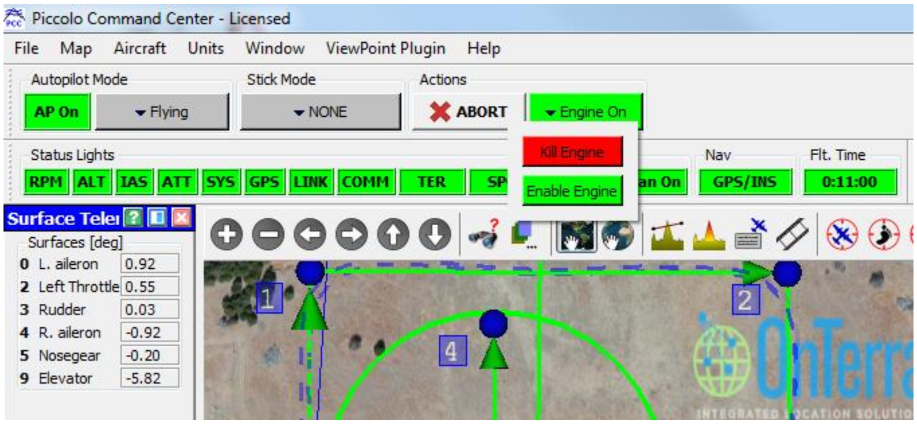
☐ 1. Check and record initial battery voltage and current.
☐ 2. Configure map page as required for the mission.
☐ 3. Verify and/or load flight plans.
☐ 4. Verify correct controller settings.
☐ 5. Verify mission limits including deadman status and lost comm waypoint.

GPS can also be assigned altitude authority in the Preflight window.

2.5.12. Geo Boundaries

Geo Boundaries can be used to designate airspace. If a geo fence is created it will keep the user from drawing any waypoints outside of the fence. Unfortunately the geo fence does not stop aircraft from violating the geo fence airspace boundaries. It is strictly a user interface feature to keep the user from making flight plans in restricted airspace. If an aircraft appears to be on course to violate a boundary PCC will warn the user however no evasive action or change to the control decisions of the autopilot will take place and the aircraft can and will blaze right through a boundary as if it didn’t even exist. The boundary drawing tool can be found in the toolbar above the main screen in PCC, and is designated as a bright yellow box.

2.5.13. Engine ON/OFF

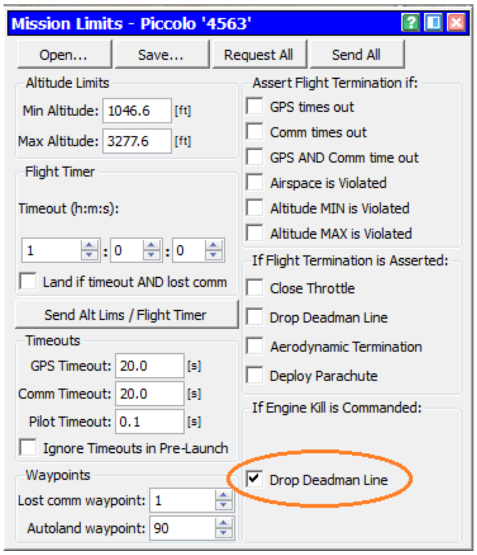


Surfaces [deg]

0	L. aileron	0.92
2	Left Throttle	0.55
3	Rudder	0.03
4	R. aileron	-0.92
5	Nosegear	-0.20
9	Elevator	-5.82

Figure 17 Kill Engine

The Engine ON/OFF button not only displays the state of the Engine, it can also be used to command the Engine to shut off, or “Kill Engine”. Switching to Kill Engine has two different effects on the autopilot. The autopilot will immediately command 0 throttle. No matter what situation the aircraft is in the autopilot will no longer command throttle. This feature can be useful for aircraft without engine ignition switches, such as aircraft with electric propulsion systems. The other effect it can have, with one pre-condition, is switching off the ignition of propulsion systems utilizing the deadman tach board. The deadman tach is a Cloud Cap add on that provides power to the engine ignition switch. In the event that a Kill Engine command is executed the deadman tach will shutoff power to the engine ignition switch as long as the autopilot has been given the authority to be able to do so. The user and the autopilot will only be able to kill the ignition if the “Drop Deadman Line” checkbox, in the “Mission Limits” window of PCC, is checked as shown below in Figure 18.



Mission Limits - Piccolo '4563'

Altitude Limits

Min Altitude: 1046.6 [ft]

Max Altitude: 3277.6 [ft]

Flight Timer

Timeout (h:m:s): 1 : 0 : 0

□ Land if timeout AND lost comm

Timeouts

GPS Timeout: 20.0 [s]

Comm Timeout: 20.0 [s]

Pilot Timeout: 0.1 [s]

□ Ignore Timeouts in Pre-Launch

Waypoints

Lost comm waypoint: 1

Autoland waypoint: 90

Assert Flight Termination if:

□ GPS times out

□ Comm times out

□ GPS AND Comm time out

□ Airspace is Violated

□ Altitude MIN is Violated

□ Altitude MAX is Violated

If Flight Termination is Asserted:

□ Close Throttle

□ Drop Deadman Line

□ Aerodynamic Termination

□ Deploy Parachute

If Engine Kill is Commanded:

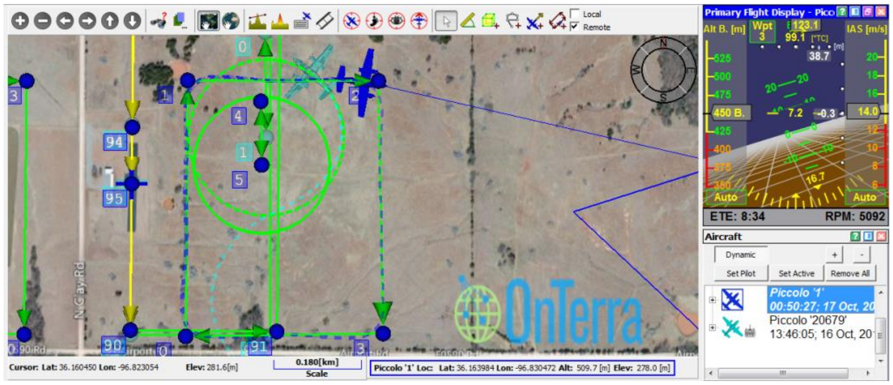
☑ Drop Deadman Line

Figure 18 Engine Kill Authority

It is important to note that during land plans if the autopilot believes the aircraft is on the ground in the short final or at the touchdown point and the Landing setting “Engine kill time” is >=0 the autopilot will automatically switch the motor to “OFF”. This exact situation resulted in a non-fatal crash with the Nexstar because the barometer incorrectly indicated that the aircraft was on the ground when it was in fact about 6 feet above the ground resulting in the engine being disabled and the manual pilot unable to manually abort the landing.

2.5.14. Operating Multiple Piccolo Units Simultaneously

Currently PCC has the capability to communicate with up to 25 piccolo units at the same time. When PCC is connected to multiple units it will write separate log files, one for each unit.



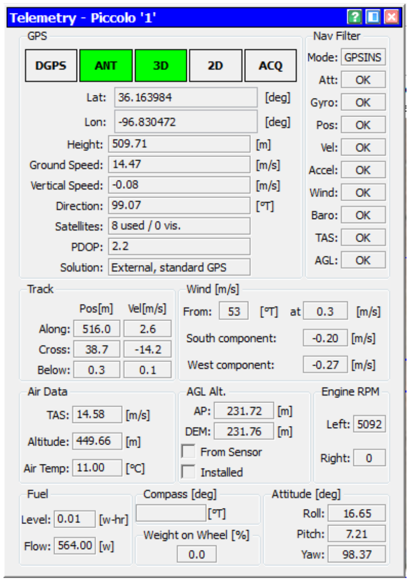
Cursor: Lat: 36.160450 Lon: -96.823054 Elev: 281.6[m]

Piccolo '1' Loc: Lat: 36.163984 Lon: -96.830472 Alt: 509.7 [m] Elev: 278.0 [m]

Figure 19 PCC Operating Multiple Autopilots

Figure 19 shows two piccolo units in simultaneous operation. Each unit is colored differently, and the images of units that are not active are ghosted. The waypoints of the active unit will be displayed in the main screen while the waypoints of the other units will be ghosted. The colors of each unit can be set in the Display Settings.

In the aircraft window, shown below the PFD in Figure 19, the user can choose which piccolo unit to "Set Pilot" and "Set Active". Set Pilot designates the unit that the manual pilot has control over. A remote control icon is displayed next to the unit where the manual pilot has been set. Set Active designates the unit that the user can issue commands to via PCC. Set Active also designates the unit that PCC is displaying telemetry data from. The title bar of each window will match the color of the active piccolo unit and display its name.



Telemetry - Piccolo '1'

GPS

Lat: 36.163984 [deg]
Lon: -96.830472 [deg]
Height: 509.71 [m]
Ground Speed: 14.47 [m/s]
Vertical Speed: -0.08 [m/s]
Direction: 99.07 [°T]
Satellites: 8 used / 0 vis.
PDOP: 2.2
Solution: External, standard GPS

Nav Filter

Mode: GPSINS
Att: OK
Gyro: OK
Pos: OK
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: OK

Track

	Pos[m]	Vel[m/s]
Along:	516.0	2.6
Cross:	38.7	-14.2
Below:	0.3	0.1

Wind [m/s]

From: 53 [°T] at 0.3 [m/s]
South component: -0.20 [m/s]
West component: -0.27 [m/s]

Air Data

TAS: 14.58 [m/s]
Altitude: 449.66 [m]
Air Temp: 11.00 [°C]

AGL Alt.

AP: 231.72 [m]
DEM: 231.76 [m]
□ From Sensor
□ Installed

Engine RPM

Left: 5092
Right: 0

Fuel

Level: 0.01 [w-hr]
Flow: 564.00 [w]

Compass [deg]

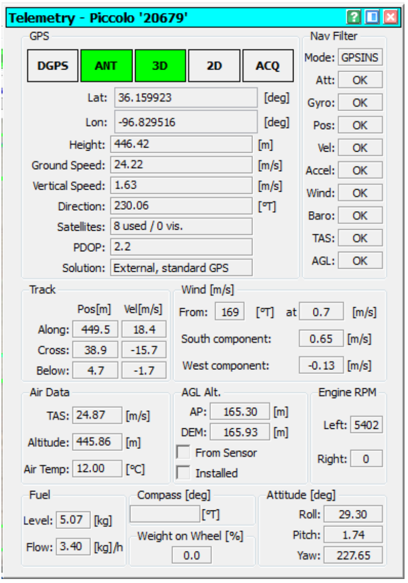
[°T]

Weight on Wheel [%]

0.0

Attitude [deg]

Roll: 16.65
Pitch: 7.21
Yaw: 98.37



Telemetry - Piccolo '20679'

GPS

Lat: 36.159923 [deg]
Lon: -96.829516 [deg]
Height: 446.42 [m]
Ground Speed: 24.22 [m/s]
Vertical Speed: 1.63 [m/s]
Direction: 230.06 [°T]
Satellites: 8 used / 0 vis.
PDOP: 2.2
Solution: External, standard GPS

Nav Filter

Mode: GPSINS
Att: OK
Gyro: OK
Pos: OK
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: OK

Track

	Pos[m]	Vel[m/s]
Along:	449.5	18.4
Cross:	38.9	-15.7
Below:	4.7	-1.7

Wind [m/s]

From: 169 [°T] at 0.7 [m/s]
South component: 0.65 [m/s]
West component: -0.13 [m/s]

Air Data

TAS: 24.87 [m/s]
Altitude: 445.86 [m]
Air Temp: 12.00 [°C]

AGL Alt.

AP: 165.30 [m]
DEM: 165.93 [m]
□ From Sensor
□ Installed

Engine RPM

Left: 5402
Right: 0

Fuel

Level: 5.07 [kg]
Flow: 3.40 [kg]/h

Compass [deg]

[°T]

Weight on Wheel [%]

0.0

Attitude [deg]

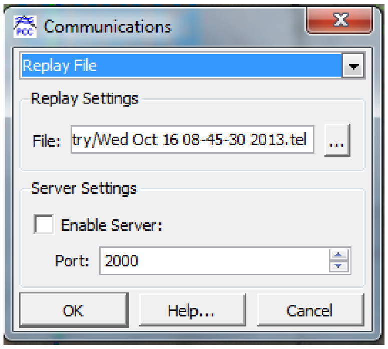
Roll: 29.30
Pitch: 1.74
Yaw: 227.65

2.5.15. Viewing Replay Files

To replay a flight:

1) Open PCC

2) In the communications window select "Replay File" from the drop down menu at the top.



Communications

Replay File

Replay Settings

File: try/Wed Oct 16 08-45-30 2013.tel

Server Settings

□ Enable Server:

Port: 2000

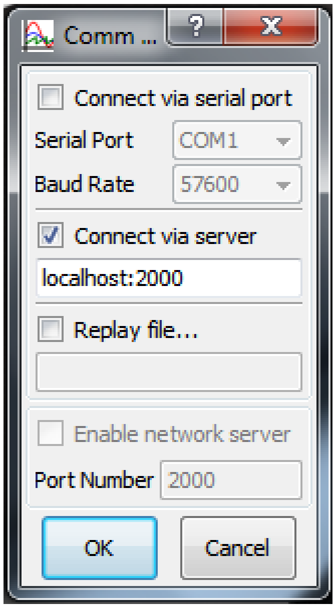
3) Load the tel file of the flight desired to watch by clicking the "…" button and selecting the desired replay file.

During replays PCC will record the telemetry in a log file just as it does during live flight. There is a toolbar at the bottom of PCC's main screen that allows the user to pause, change the speed of the replay. There is also a scroll bar where the user can jump to different periods of time in the flight. Be careful using the scroll bar to navigate. If a parameter is changed in the time span that is skipped the change will not be visible to the user. For example if a gain change occurred during a skipped time period and the user was to view the value of this gain the value displayed would be the old unchanged value and not the correct current value.

2.6. DevInterface

The DevInterface is a powerful tool provided by Cloud Cap to aid in flight and post flight analysis. DevInterface is located in the Cloud Cap folder under "Cloud Cap\Piccolo 2.x.x.x\Dev Interfaces\". There should be a shortcut to launch the DevInterface on the desktop of the UAS Laptop. The only documentation provided by Cloud Cap that discusses the DevInterface is a document titled "PiccoloDevInterface.pdf". The document is located in the Cloud Cap folder of both the UAS Desktop and the UAS Laptop under "Cloud Cap\Piccolo 2.x.x.x\Piccolo Docs\Software\". The document is outdated. This section will briefly go over the capabilities that the DevInterface offers.

Part of the startup process is to select the communications port or replay file to stream data from.



Comm ...

□ Connect via serial port

Serial Port: COM1

Baud Rate: 57600

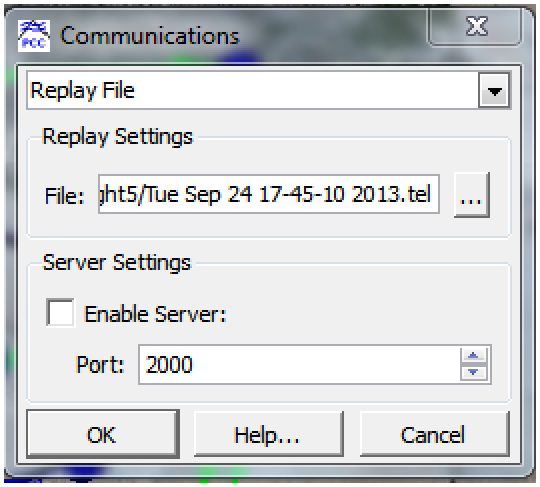
☑ Connect via server

localhost:2000

□ Replay file...

□ Enable network server

Port Number: 2000



Communications

Replay File

Replay Settings

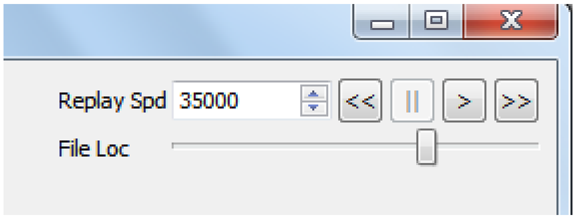
File: ght5/Tue Sep 24 17-45-10 2013.tel

Server Settings

□ Enable Server:

Port: 2000

The DevInterface can stream data output from PCC, via the server port, or replay data from a PCC telemetry file. It is also possible to watch replay files via the server when they are being played by PCC. To “Connect via server” the Enable Server box must be checked in the PCC Communications window and the server numbers must match.



Replay Spd: 35000

File Loc

If the DevInterface is viewing a replay from a piccolo telemetry file the controls in the upper right hand corner of the DevInterface will be active. The use will have to click the play button to begin the simulation. The scroll bar can be used to skip through the replay file to different times.

When viewing any of the plots in the DevInterface the user can zoom in and out by mousing over the appropriate plot and scrolling up or down on the scroll wheel. Plots can also be panned by clicking, grabbing, and dragging the plots; however, if the data is live the plot will automatically return to the current time. The time scale on all of the plots represent the piccolo clock which begins at time 0 when the autopilot is powered on.

2.6.1.Telemetry Tab

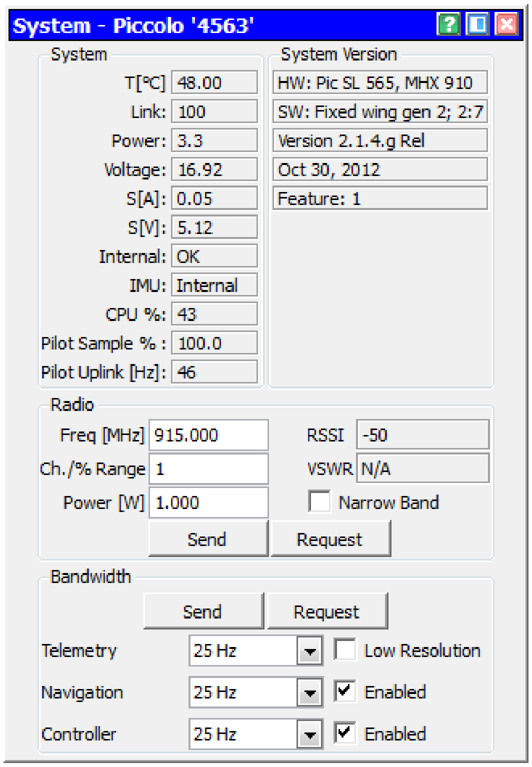
The Telemetry tab contains 3 sub tabs, “Inertial Sensors”, “Euler angles, rates”, and “V,h,RPM.” The Inertial Sensors tab displays the accelerometer data, and the rate gyros. “p,q,r” is the roll, pitch, and yaw rates. The Euler angles, rates tab displays the euler angles and rates.

2.6.2. Vibration Tab

The Vibration tab can be used when the user is performing vibration tests. Here the vibration analysis can be performed just by viewing the data or it can also be saved to a text file.

2.6.3. Fixed Wing Gen2 Tab

The Fixed Wing Gen2 tab will record and display data that is specific to control loops and control loop performance. This tab only exists when controller telemetry is enabled.



System - Piccolo '4563'

System

T[°C] 48.00
Link: 100
Power: 3.3
Voltage: 16.92
S[A]: 0.05
S[V]: 5.12
Internal: OK
IMU: Internal
CPU %: 43
Pilot Sample % : 100.0
Pilot Uplink [Hz]: 46

System Version

HW: Pic SL 565, MHX 910
SW: Fixed wing gen 2; 2:7
Version 2.1.4.g Rel
Oct 30, 2012
Feature: 1

Radio

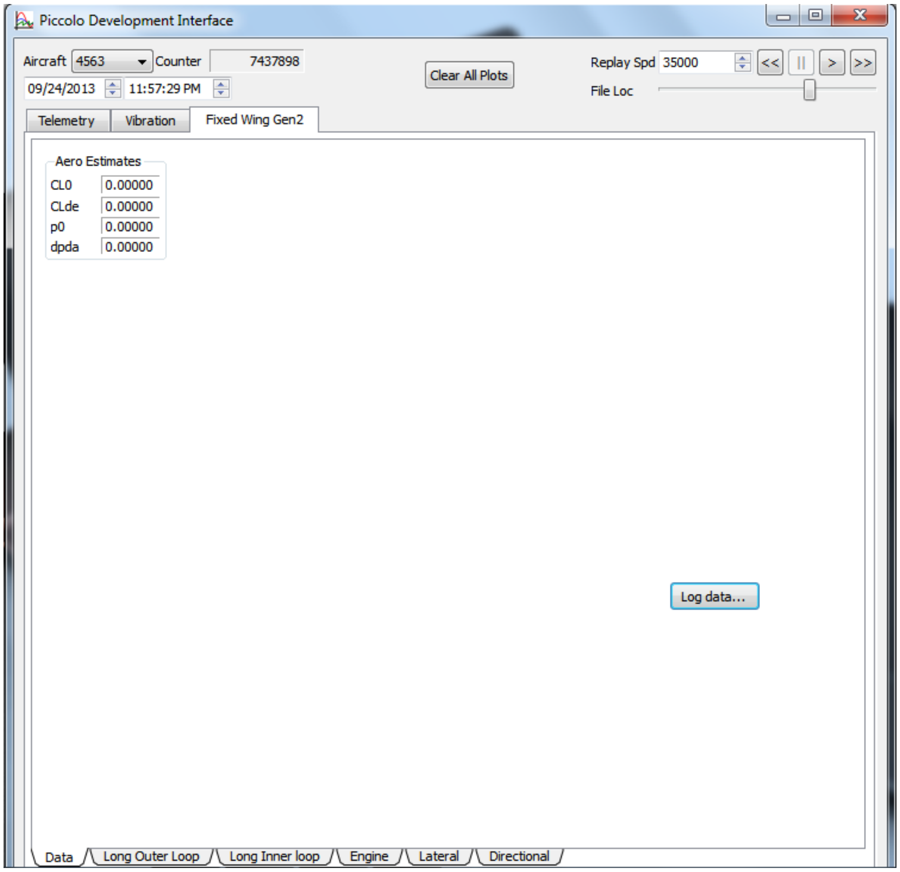
Freq [MHz] 915.000
Ch./% Range 1
Power [W] 1.000
RSSI -50
VSWR N/A
☐ Narrow Band

Bandwidth

Telemetry 25 Hz ☐ Low Resolution
Navigation 25 Hz ☑ Enabled
Controller 25 Hz ☑ Enabled

Figure 20 PCC System Window

Controller telemetry can be enabled in the “System” window of PCC. If controller telemetry is not enabled this tab will not be available, even if the DevInterface is viewing a piccolo telemetry file.



Aircraft 4563 Counter 7437898
09/24/2013 11:57:29 PM
Replay Spd 35000

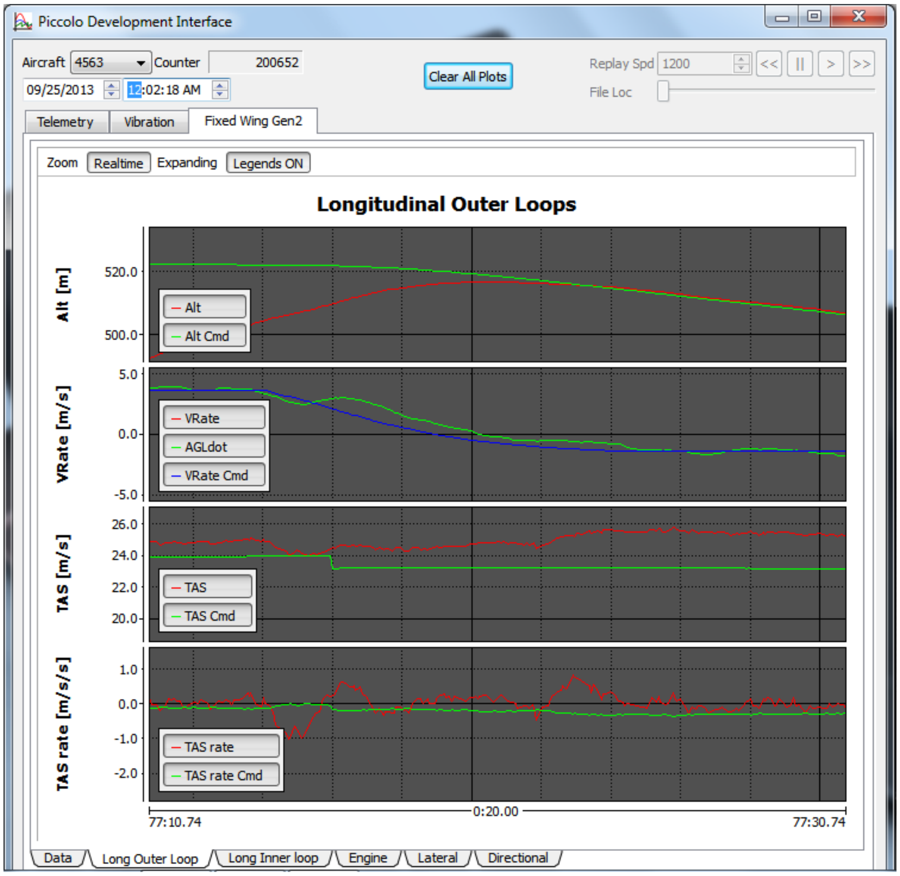
Aero Estimates

CL0	0.00000
CLde	0.00000
p0	0.00000
dpda	0.00000

Figure 21 DevInterface Controller Telemetry

There are 6 sub tabs in Fixed Wing Gen2. The "Data" tab allows the user the ability to save the controller telemetry to a log file. Controller telemetry can be written to a log file at any time that the DevInterface is running. It does not matter if the DevInterface is replaying a replay file, streaming data from PCC during an actual flight, or streaming data from PCC while it is running a replay file. The log file, or Dev log file, is utilized heavily for control loop analysis.

At the top of the Fixed Wing Gen2 tab the user has a few viewing options. Selecting "Zoom" will change the mouse cursor to the zoom cursor. If zoom is not selected the zoom function is as described earlier using the mouse scroll wheel. If "Realtime" is not selected the plots will be frozen in time and the user can pan as desired. "Expanding" will automatically zoom in and out vertically on all of the plots to keep the measured value in view. "Legends ON" will turn the legends on and off appropriately.



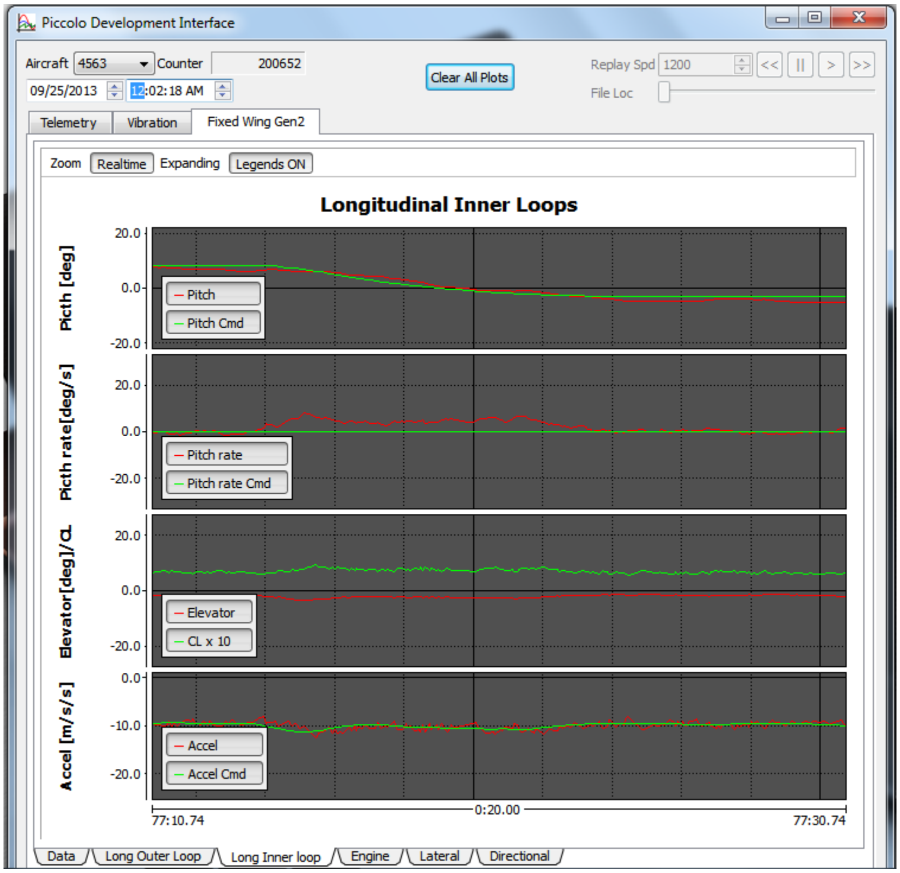
Piccolo Development Interface

Aircraft 4563 Counter 200652

09/25/2013 12:02:18 AM

Replay Spd 1200

The “Long Outer Loop” tab displays data that is pertinent to the outer loops of longitudinal control.



Piccolo Development Interface

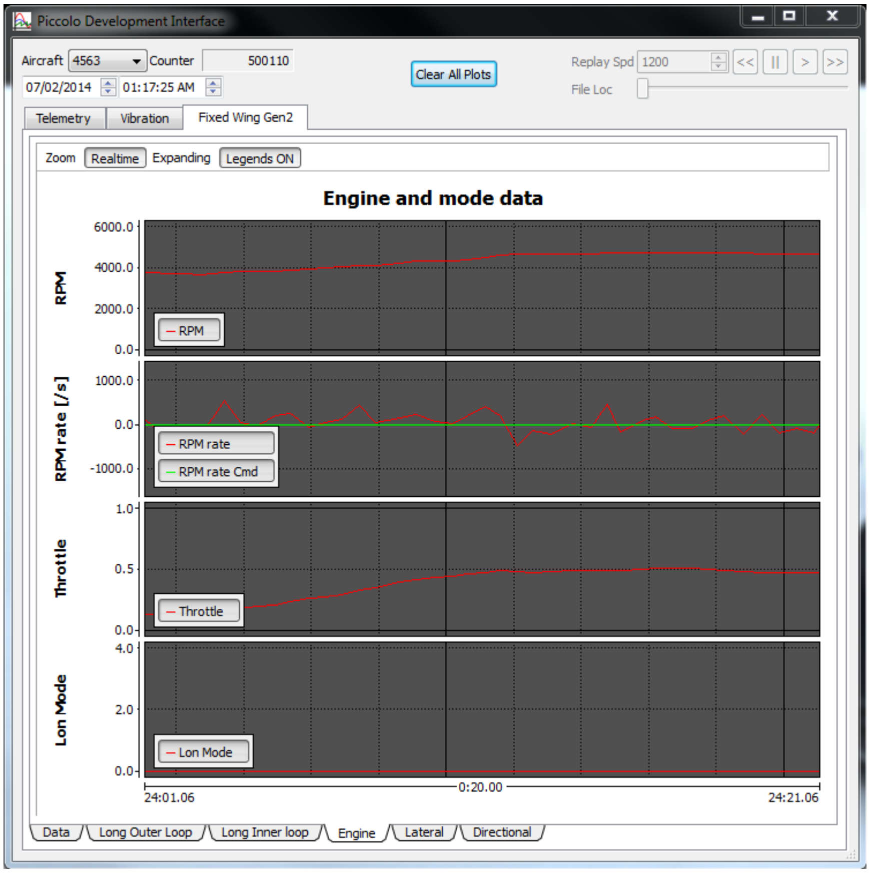
Aircraft 4563 Counter 200652

09/25/2013 12:02:18 AM

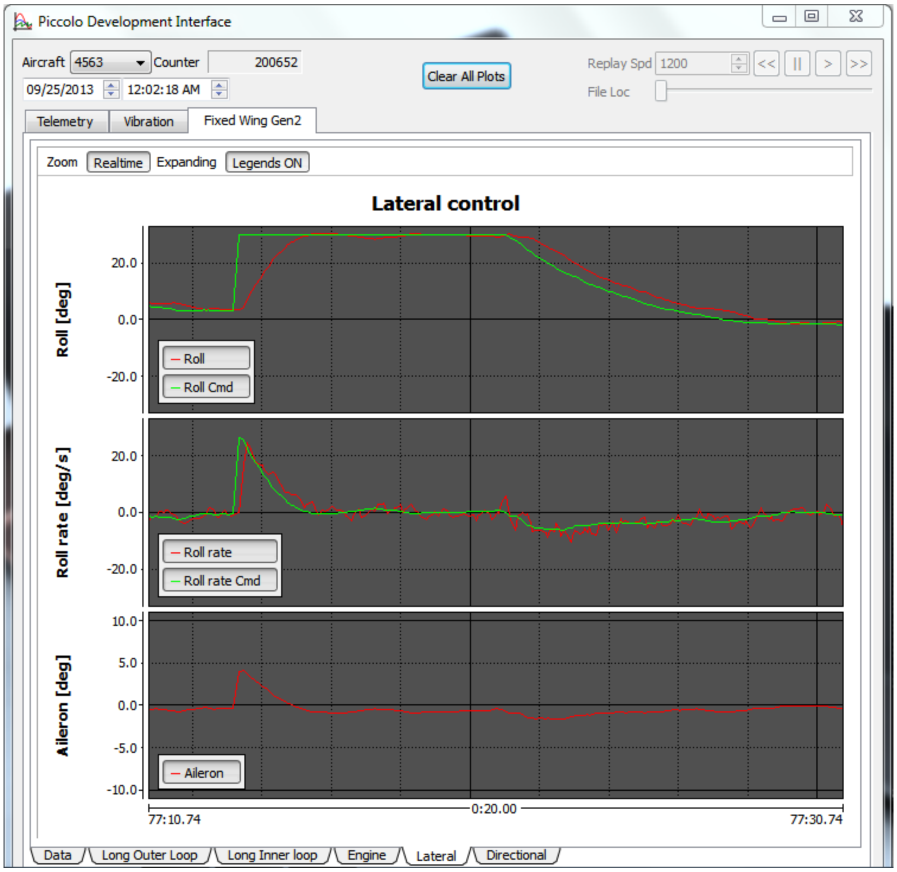
Replay Spd 1200

File Loc

The “Long Inner Loop” tab displays data that is pertinent to the inner loops of longitudinal control.

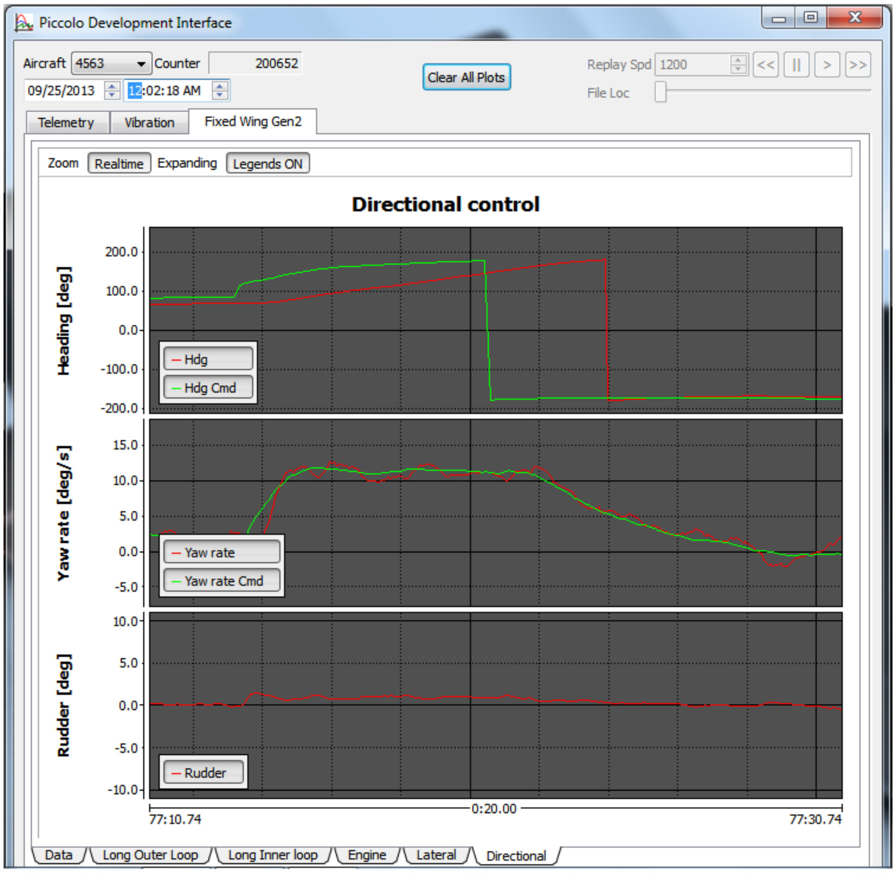

Aircraft 4563 Counter 500110
07/02/2014 01:17:25 AM
Replay Spd 1200

The “Engine” tab displays Throttle and RPMs along with the current Lon Mode (longitudinal mode) that the autopilot is operating in.



Aircraft 4563
Counter 200652
09/25/2013 12:02:18 AM
Replay Spd 1200

The “Lateral” tab displays data that is pertinent to the inner loops of lateral control.



Aircraft 4563 Counter 200652

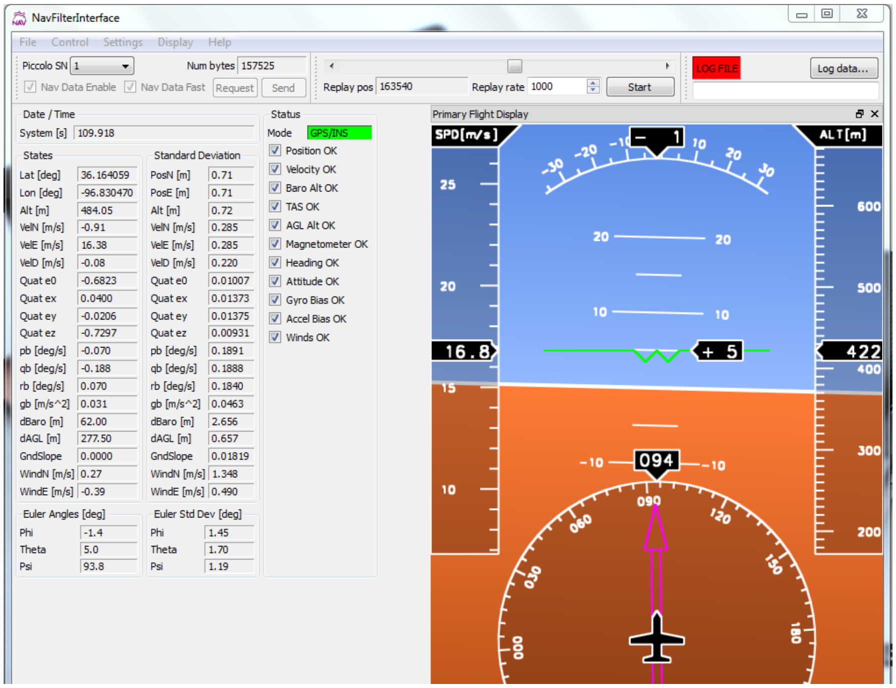
09/25/2013 12:02:18 AM

Replay Spd 1200

The "Directional" tab displays data that is pertinent to outer loop lateral control, and yaw control.

2.7. NavFilterInterface

The navigation filter interface, or NavFilterInterface, is located in the Cloud Cap folder under "Cloud Cap\Piccolo 2.x.x.x\Dev Interfaces\". There should be a shortcut to launch the NavFilterInterface on the desktop of the UAS Laptop. The NavFilterInterface offers a whole range of options to display detailed navigation information on the state of the autopilot during flight.



NavFilterInterface

Piccolo SN 1 Num bytes 157525

☑ Nav Data Enable ☑ Nav Data Fast

Replay pos 163540 Replay rate 1000

Date / Time

System [s] 109.918

States		Standard Deviation	
Lat [deg]	36.164059	PosN [m]	0.71
Lon [deg]	-96.830470	PosE [m]	0.71
Alt [m]	484.05	Alt [m]	0.72
VelN [m/s]	-0.91	VelN [m/s]	0.285
VelE [m/s]	16.38	VelE [m/s]	0.285
VelD [m/s]	-0.08	VelD [m/s]	0.220
Quat e0	-0.6823	Quat e0	0.01007
Quat ex	0.0400	Quat ex	0.01373
Quat ey	-0.0206	Quat ey	0.01375
Quat ez	-0.7297	Quat ez	0.00931
pb [deg/s]	-0.070	pb [deg/s]	0.1891
qb [deg/s]	-0.188	qb [deg/s]	0.1888
rb [deg/s]	0.070	rb [deg/s]	0.1840
gb [m/s^2]	0.031	gb [m/s^2]	0.0463
dBaro [m]	62.00	dBaro [m]	2.656
dAGL [m]	277.50	dAGL [m]	0.657
GndSlope	0.0000	GndSlope	0.01819
WindN [m/s]	0.27	WindN [m/s]	1.348
WindE [m/s]	-0.39	WindE [m/s]	0.490

Euler Angles [deg]		Euler Std Dev [deg]	
Phi	-1.4	Phi	1.45
Theta	5.0	Theta	1.70
Psi	93.8	Psi	1.19

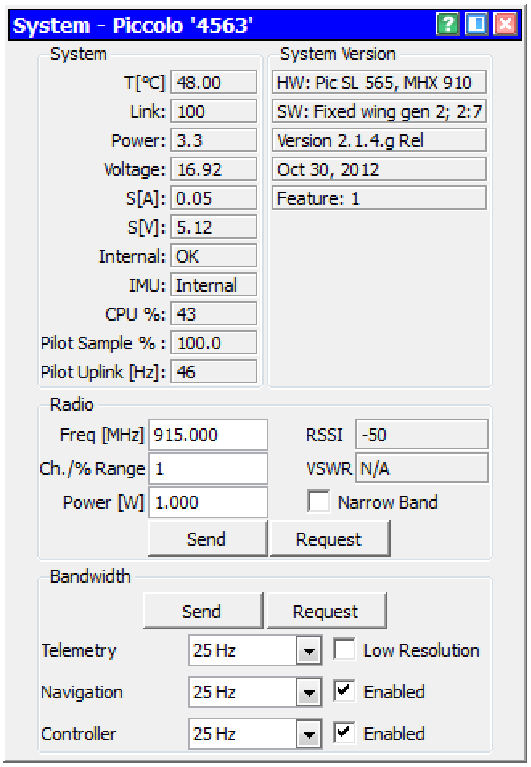
Status

Mode GPS/INS

☑ Position OK
☑ Velocity OK
☑ Baro Alt OK
☑ TAS OK
☑ AGL Alt OK
☑ Magnetometer OK
☑ Heading OK
☑ Attitude OK
☑ Gyro Bias OK
☑ Accel Bias OK
☑ Winds OK

Primary Flight Display

Similar to the DevInterface the NavInterface can stream data from live flights or replay files and can save navigation data to a log file. In order to function properly navigation telemetry must be enabled during the flight.



System - Piccolo '4563'

System

T[°C] 48.00

Link: 100

Power: 3.3

Voltage: 16.92

S[A]: 0.05

S[V]: 5.12

Internal: OK

IMU: Internal

CPU %: 43

Pilot Sample % : 100.0

Pilot Uplink [Hz]: 46

System Version

HW: Pic SL 565, MHX 910

SW: Fixed wing gen 2; 2:7

Version 2.1.4.g Rel

Oct 30, 2012

Feature: 1

Radio

Freq [MHz] 915.000

Ch./% Range 1

Power [W] 1.000

RSSI -50

VSWR N/A

□ Narrow Band

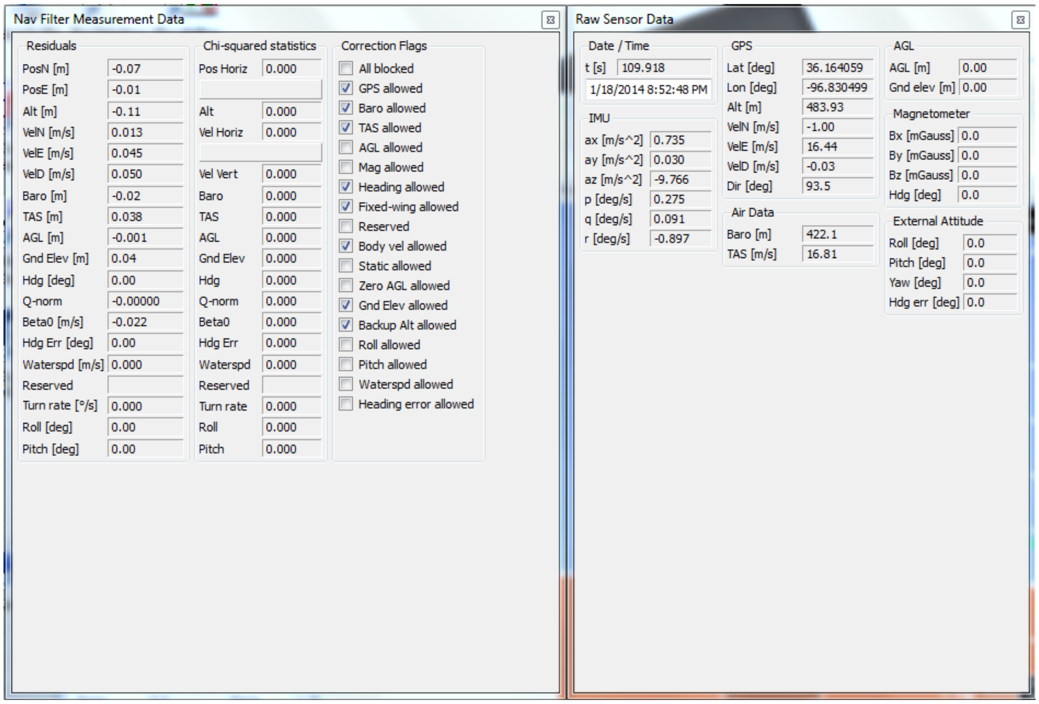
Bandwidth

Telemetry 25 Hz □ Low Resolution

Navigation 25 Hz ☑ Enabled

Controller 25 Hz ☑ Enabled

Navigation telemetry can be enabled in the “System” window of PCC. The main display of the navfilter interface can be altered by clicking and moving sections around. Additionally the NavFilter can display “Nav Filter Measurement Data” and “Raw Sensor Data”.



Nav Filter Measurement Data

Residuals		Chi-squared statistics	
PosN [m]	-0.07	Pos Horiz	0.000
PosE [m]	-0.01		
Alt [m]	-0.11	Alt	0.000
VelN [m/s]	0.013	Vel Horiz	0.000
VelE [m/s]	0.045		
VelD [m/s]	0.050	Vel Vert	0.000
Baro [m]	-0.02	Baro	0.000
TAS [m]	0.038	TAS	0.000
AGL [m]	-0.001	AGL	0.000
Gnd Elev [m]	0.04	Gnd Elev	0.000
Hdg [deg]	0.00	Hdg	0.000
Q-norm	-0.00000	Q-norm	0.000
Beta0 [m/s]	-0.022	Beta0	0.000
Hdg Err [deg]	0.00	Hdg Err	0.000
Waterspd [m/s]	0.000	Waterspd	0.000
Reserved		Reserved	
Turn rate [°/s]	0.000	Turn rate	0.000
Roll [deg]	0.00	Roll	0.000
Pitch [deg]	0.00	Pitch	0.000

Correction Flags

- □ All blocked
- ☑ GPS allowed
- ☑ Baro allowed
- ☑ TAS allowed
- □ AGL allowed
- □ Mag allowed
- ☑ Heading allowed
- ☑ Fixed-wing allowed
- □ Reserved
- ☑ Body vel allowed
- □ Static allowed
- □ Zero AGL allowed
- ☑ Gnd Elev allowed
- ☑ Backup Alt allowed
- □ Roll allowed
- □ Pitch allowed
- □ Waterspd allowed
- □ Heading error allowed

Raw Sensor Data

Date / Time

t [s] 109.918
1/18/2014 8:52:48 PM

IMU

ax [m/s^2]	0.735
ay [m/s^2]	0.030
az [m/s^2]	-9.766
p [deg/s]	0.275
q [deg/s]	0.091
r [deg/s]	-0.897

GPS

Lat [deg]	36.164059
Lon [deg]	-96.830499
Alt [m]	483.93
VelN [m/s]	-1.00
VelE [m/s]	16.44
VelD [m/s]	-0.03
Dir [deg]	93.5

Air Data

Baro [m]	422.1
TAS [m/s]	16.81

AGL

AGL [m]	0.00
Gnd elev [m]	0.00

Magnetometer

Bx [mGauss]	0.0
By [mGauss]	0.0
Bz [mGauss]	0.0
Hdg [deg]	0.0

External Attitude

Roll [deg]	0.0
Pitch [deg]	0.0
Yaw [deg]	0.0
Hdg err [deg]	0.0

2.8. CCT MATLAB

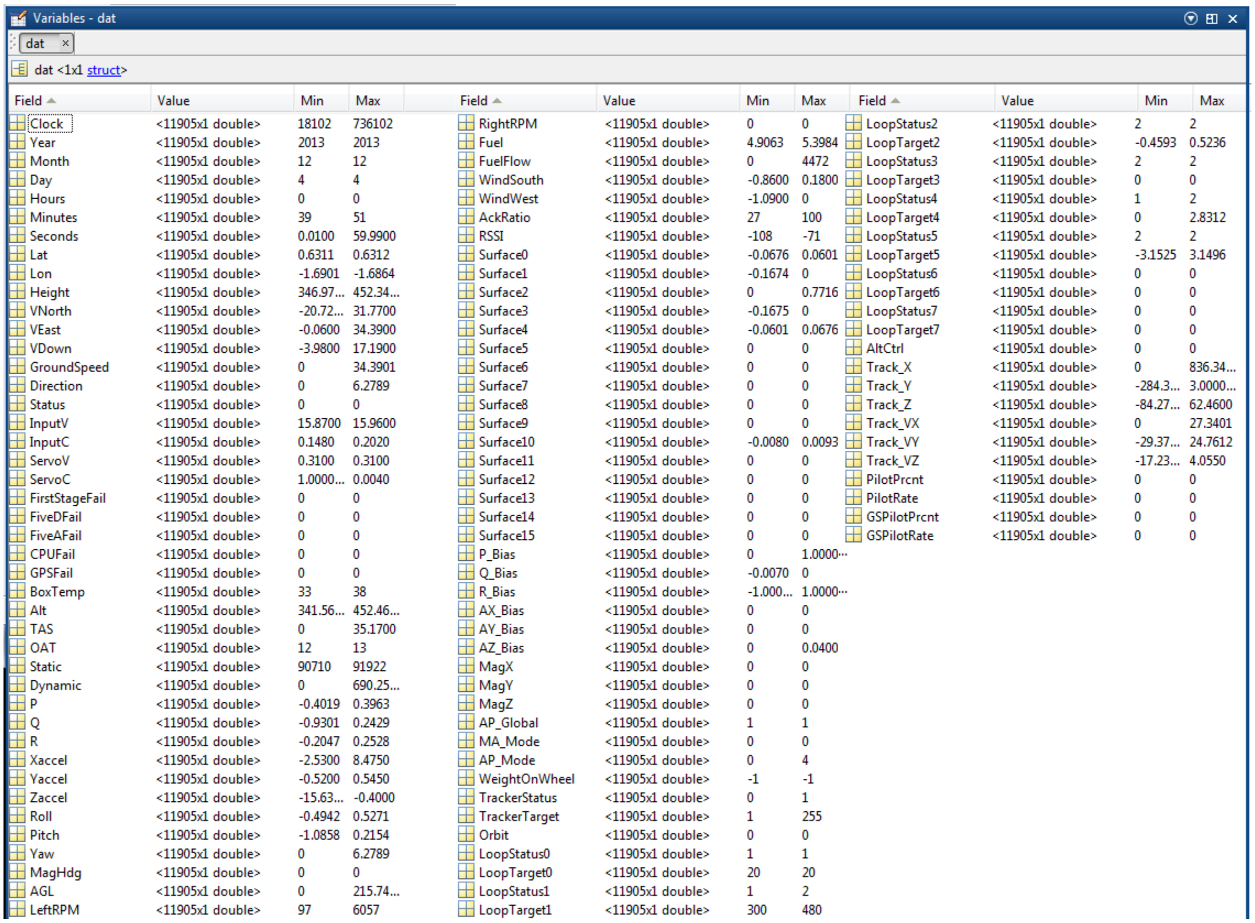
CCT Matlab is a folder full of various MATLAB scripts that have been written by Cloud Cap. The scripts range from graphing piccolo log files to performing doublet maneuver analysis and even generating geometry profiles of APC propellers.

This guide utilizes “plotpiccolo.m”, “doublet.m”, and “plotdoublet.m”. The following sections describe these scripts and changes that have been made to them.

2.8.1. plotpiccolo

“plotpiccolo.m” is designed for generating various plots of piccolo telemetry data. Plotpiccolo calls the script “loadlogfilepiccolo.m” to load data from the piccolo telemetry log files into MATLAB workspace variables.

Loadlogfilepiccolo imports the data from the piccolo telemetry text files and creates a variable for each column of data in the text file. Plotpiccolo moves all the variables into a structure called “dat”.



Variables - dat

dat

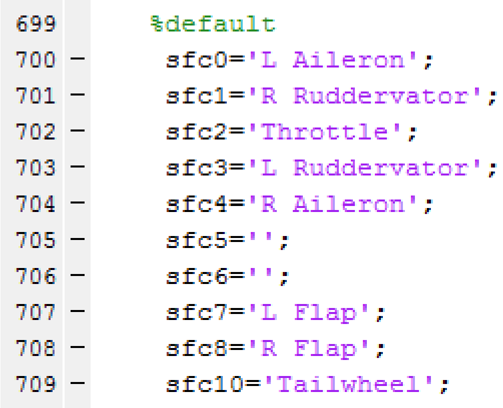
dat <1x1 struct>

Field	Value	Min	Max	Field	Value	Min	Max	Field	Value	Min	Max
Clock	<11905x1 double>	18102	736102	RightRPM	<11905x1 double>	0	0	LoopStatus2	<11905x1 double>	2	2
Year	<11905x1 double>	2013	2013	Fuel	<11905x1 double>	4.9063	5.3984	LoopTarget2	<11905x1 double>	-0.4593	0.5236
Month	<11905x1 double>	12	12	FuelFlow	<11905x1 double>	0	4472	LoopStatus3	<11905x1 double>	2	2
Day	<11905x1 double>	4	4	WindSouth	<11905x1 double>	-0.8600	0.1800	LoopTarget3	<11905x1 double>	0	0
Hours	<11905x1 double>	0	0	WindWest	<11905x1 double>	-1.0900	0	LoopStatus4	<11905x1 double>	1	2
Minutes	<11905x1 double>	39	51	AckRatio	<11905x1 double>	27	100	LoopTarget4	<11905x1 double>	0	2.8312
Seconds	<11905x1 double>	0.0100	59.9900	RSSI	<11905x1 double>	-108	-71	LoopStatus5	<11905x1 double>	2	2
Lat	<11905x1 double>	0.6311	0.6312	Surface0	<11905x1 double>	-0.0676	0.0601	LoopTarget5	<11905x1 double>	-3.1525	3.1496
Lon	<11905x1 double>	-1.6901	-1.6864	Surface1	<11905x1 double>	-0.1674	0	LoopStatus6	<11905x1 double>	0	0
Height	<11905x1 double>	346.97...	452.34...	Surface2	<11905x1 double>	0	0.7716	LoopTarget6	<11905x1 double>	0	0
VNorth	<11905x1 double>	-20.72...	31.7700	Surface3	<11905x1 double>	-0.1675	0	LoopStatus7	<11905x1 double>	0	0
VEast	<11905x1 double>	-0.0600	34.3900	Surface4	<11905x1 double>	-0.0601	0.0676	LoopTarget7	<11905x1 double>	0	0
VDown	<11905x1 double>	-3.9800	17.1900	Surface5	<11905x1 double>	0	0	AltCtrl	<11905x1 double>	0	0
GroundSpeed	<11905x1 double>	0	34.3901	Surface6	<11905x1 double>	0	0	Track_X	<11905x1 double>	0	836.34...
Direction	<11905x1 double>	0	6.2789	Surface7	<11905x1 double>	0	0	Track_Y	<11905x1 double>	-284.3...	3.0000...
Status	<11905x1 double>	0	0	Surface8	<11905x1 double>	0	0	Track_Z	<11905x1 double>	-84.27...	62.4600
InputV	<11905x1 double>	15.8700	15.9600	Surface9	<11905x1 double>	0	0	Track_VX	<11905x1 double>	0	27.3401
InputC	<11905x1 double>	0.1480	0.2020	Surface10	<11905x1 double>	-0.0080	0.0093	Track_VY	<11905x1 double>	-29.37...	24.7612
ServoV	<11905x1 double>	0.3100	0.3100	Surface11	<11905x1 double>	0	0	Track_VZ	<11905x1 double>	-17.23...	4.0550
ServoC	<11905x1 double>	1.0000...	0.0040	Surface12	<11905x1 double>	0	0	PilotPrcnt	<11905x1 double>	0	0
FirstStageFail	<11905x1 double>	0	0	Surface13	<11905x1 double>	0	0	PilotRate	<11905x1 double>	0	0
FiveDFail	<11905x1 double>	0	0	Surface14	<11905x1 double>	0	0	GSPilotPrcnt	<11905x1 double>	0	0
FiveAFail	<11905x1 double>	0	0	Surface15	<11905x1 double>	0	0	GSPilotRate	<11905x1 double>	0	0
CPUFail	<11905x1 double>	0	0	P_Bias	<11905x1 double>	0	1.0000···				
GPSFail	<11905x1 double>	0	0	Q_Bias	<11905x1 double>	-0.0070	0				
BoxTemp	<11905x1 double>	33	38	R_Bias	<11905x1 double>	-1.000...	1.0000···				
Alt	<11905x1 double>	341.56...	452.46...	AX_Bias	<11905x1 double>	0	0				
TAS	<11905x1 double>	0	35.1700	AY_Bias	<11905x1 double>	0	0				
OAT	<11905x1 double>	12	13	AZ_Bias	<11905x1 double>	0	0.0400				
Static	<11905x1 double>	90710	91922	MagX	<11905x1 double>	0	0				
Dynamic	<11905x1 double>	0	690.25...	MagY	<11905x1 double>	0	0				
P	<11905x1 double>	-0.4019	0.3963	MagZ	<11905x1 double>	0	0				
Q	<11905x1 double>	-0.9301	0.2429	AP_Global	<11905x1 double>	1	1				
R	<11905x1 double>	-0.2047	0.2528	MA_Mode	<11905x1 double>	0	0				
Xaccel	<11905x1 double>	-2.5300	8.4750	AP_Mode	<11905x1 double>	0	4				
Yaccel	<11905x1 double>	-0.5200	0.5450	WeightOnWheel	<11905x1 double>	-1	-1				
Zaccel	<11905x1 double>	-15.63...	-0.4000	TrackerStatus	<11905x1 double>	0	1				
Roll	<11905x1 double>	-0.4942	0.5271	TrackerTarget	<11905x1 double>	1	255				
Pitch	<11905x1 double>	-1.0858	0.2154	Orbit	<11905x1 double>	0	0				
Yaw	<11905x1 double>	0	6.2789	LoopStatus0	<11905x1 double>	1	1				
MagHdg	<11905x1 double>	0	0	LoopTarget0	<11905x1 double>	20	20				
AGL	<11905x1 double>	0	215.74...	LoopStatus1	<11905x1 double>	1	2				
LeftRPM	<11905x1 double>	97	6057	LoopTarget1	<11905x1 double>	300	480				

Figure 22 Plot Piccolo Data Structure

Figure 22 is a snapshot of the “dat” structure in MATLAB. The figure shows most, but not all variables that dat contains. “Alt” is the barometer measured altitude. “Height” is the GPS estimated altitude. “Clock” is the piccolo clock that starts at 0 when the piccolo is powered on. The “year, month, day, hourse, minutes, and seconds” come from the GPS clock. “Direction” is the true heading estimate. “AckRatio” represents “Link” in the “Piccolo System” window of PCC. Link is a measure of packet loss between the piccolo autopilot and the ground station where 100% means no packet loss and 0% means no packets being received by the ground station. “Surface0” – “Surface15” are surface deflections where Surface0 represents the control surface wired into “Servo 0”. “AP_Global” represents whether or not the autopilot is control where a value of “1” means autopilot control and a value of “0” means manual r/c pilot control. All of the “LoopTarget” variables represent command loop targets; however, note that the command loop values recorded by PCC lag behind when they were actually changed by the autopilot. The DevInterface records a more accurate representation of the command loop commands as it records controller telemetry directly. “Track_X” is the x distance away from the target waypoint. “Track_Y” is the cross track error, or distance away from the target flight path in the y-direction. “Track_Z” is the vertical distance, or altitude, error between the aircraft and the target flight path. “PilotPrcnt” represents the signal strength of the manual r/c pilot with the optional JR receivers. “GSPilotPrcnt” represents the signal strength of the manual r/c pilot when the ground station remote control is being used.

Plotpiccolo also creates some extra variables. One of the more notable ones is “tClock.” tClock represents change in time where the first recorded data point is treated as time zero. Another set of variables to note are the surface names which are saved as string variables and named as “sfc0 sfc1…” These variables are actually created in the plotpiccolo code and they set the control surface names based on the names designated in the code.



```
%default
 sfc0='L Aileron';
 sfc1='R Ruddervator';
 sfc2='Throttle';
 sfc3='L Ruddervator';
 sfc4='R Aileron';
 sfc5='';
 sfc6='';
 sfc7='L Flap';
 sfc8='R Flap';
 sfc10='Tailwheel';
```

Figure 23 Plotpiccolo Noctua B1 Code

.

Figure 23 shows the control surface names for Noctua B1. If the user wants to use the plotpiccolo script to accurately plot control surface deflections then these lines of code will have to be changed appropriately to represent the control surface configuration otherwise the surface deflection plots will have the incorrect names for each surface.

2.8.1.1. Script Errors

The plotpiccolo mat file is the saved workspace of all the variables that plotpiccolo creates. Plotpiccolo includes some very useful plots such as a GPS plot of signal strength and a plot of GPS position which is shown below.

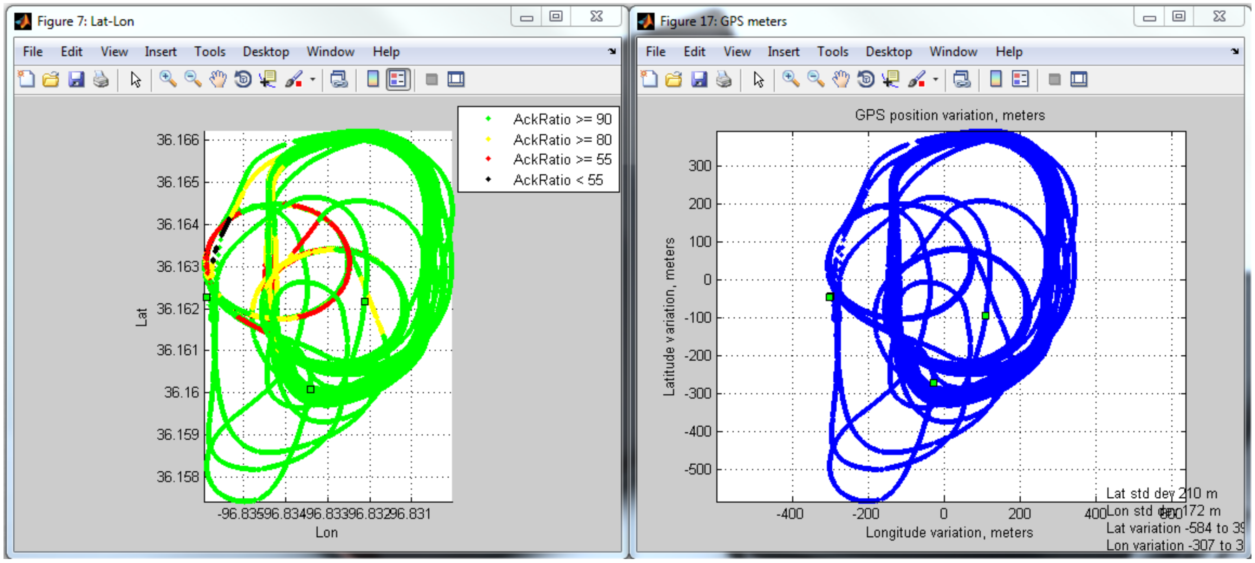
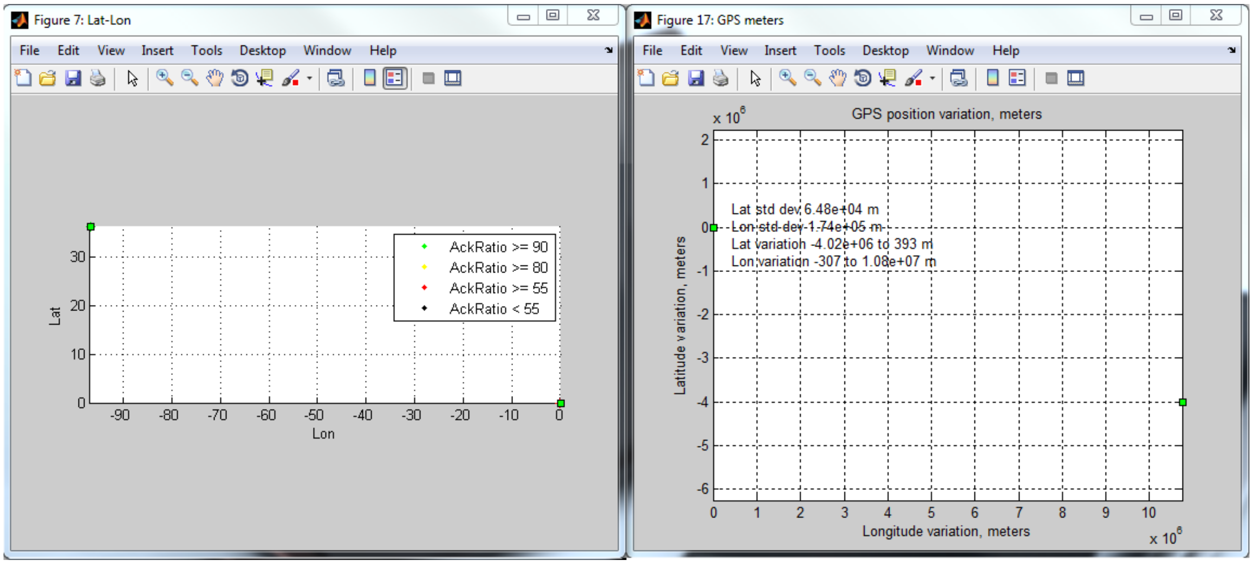


Figure 24 GPS Position Plots

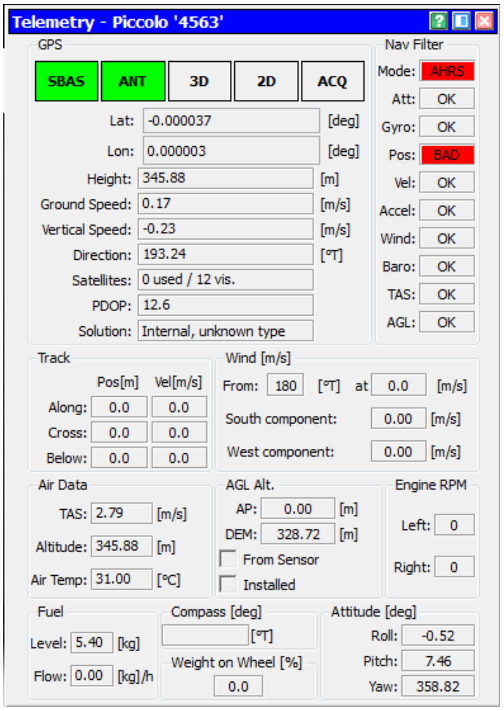
Initially there was an error in two of the original plotpiccolo plots that plotted latitude and longitude. The plots use the maximum and minimum recorded values of the estimated latitude and longitude to scale the axes. The problem occurred when the telemetry file began recording latitude and longitude while the autopilot was still acquiring GPS satellites; thus, the autopilot navigation solution was not in GPS/INS and the latitude and longitude recordings were nearly 0 degrees in both directions.



Lat std dev 6.48e+04 m
Lon std dev 1.74e+05 m
Lat variation -4.02e+06 to 393 m
Lon variation -307 to 1.08e+07 m

Figure 25 GPS Plot Error

Figure 25 shows the two position plots generated from the same piccolo telemetry log file as Figure 24, but with the original plotpiccolo code.



Telemetry - Piccolo '4563'

GPS

SBAS ANT 3D 2D ACQ

Lat: -0.000037 [deg]
Lon: 0.000003 [deg]
Height: 345.88 [m]
Ground Speed: 0.17 [m/s]
Vertical Speed: -0.23 [m/s]
Direction: 193.24 [°T]
Satellites: 0 used / 12 vis.
PDOP: 12.6
Solution: Internal, unknown type

Nav Filter

Mode: AHRS
Att: OK
Gyro: OK
Pos: BAD
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: OK

Track

	Pos[m]	Vel[m/s]
Along:	0.0	0.0
Cross:	0.0	0.0
Below:	0.0	0.0

Wind [m/s]

From: 180 [°T] at 0.0 [m/s]
South component: 0.00 [m/s]
West component: 0.00 [m/s]

Air Data

TAS: 2.79 [m/s]
Altitude: 345.88 [m]
Air Temp: 31.00 [°C]

AGL Alt.

AP: 0.00 [m]
DEM: 328.72 [m]
□ From Sensor
□ Installed

Engine RPM

Left: 0
Right: 0

Fuel

Level: 5.40 [kg]
Flow: 0.00 [kg]/h

Compass [deg]

[°T]

Weight on Wheel [%]

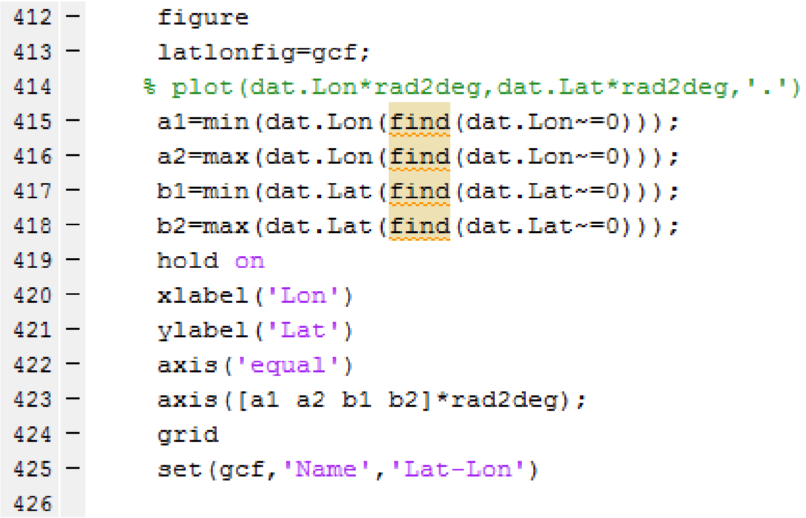
0.0

Attitude [deg]

Roll: -0.52
Pitch: 7.46
Yaw: 358.82

Figure 26 Initial GPS Location Data

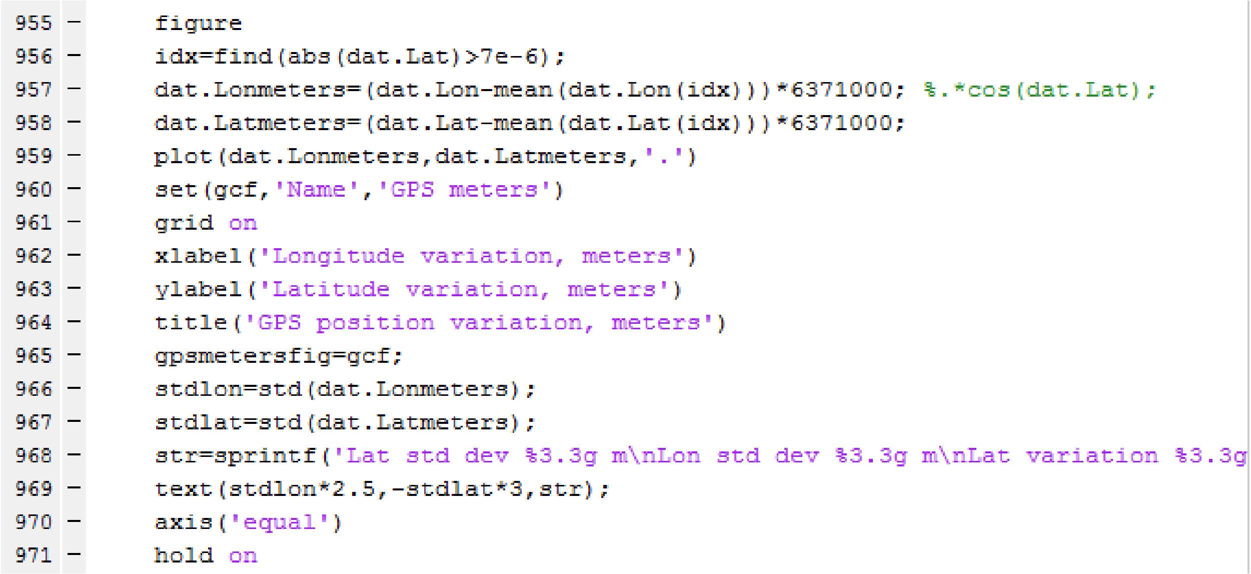
Figure 26 depicts the scenario where the initial latitude and longitude points are garbage. The number of GPS satellites used is 0, the navigation mode is “AHRS” and the latitude and longitudes are -0.000037 and 0.000003. Initially plotpiccolo tried to eliminate this error by searching for latitude values greater than 7*10^{-6} and searching for latitude and longitude values not equal to 0.



```
    figure
    latlonfig=gcf;
   % plot(dat.Lon*rad2deg,dat.Lat*rad2deg,'.')
    a1=min(dat.Lon(find(dat.Lon~=0)));
    a2=max(dat.Lon(find(dat.Lon~=0)));
    b1=min(dat.Lat(find(dat.Lat~=0)));
    b2=max(dat.Lat(find(dat.Lat~=0)));
    hold on
    xlabel('Lon')
    ylabel('Lat')
    axis('equal')
    axis([a1 a2 b1 b2]*rad2deg);
    grid
    set(gcf,'Name','Lat-Lon')

```

Figure 27 Original GPS AckRatio Code

Figure 27 shows the original code for the AckRatio Position plot shown in Figure 24. The bad latitude and longitude points were not zero therefore they were erroneously included in the plot and which resulted in the empty plots shown in Figure 25.

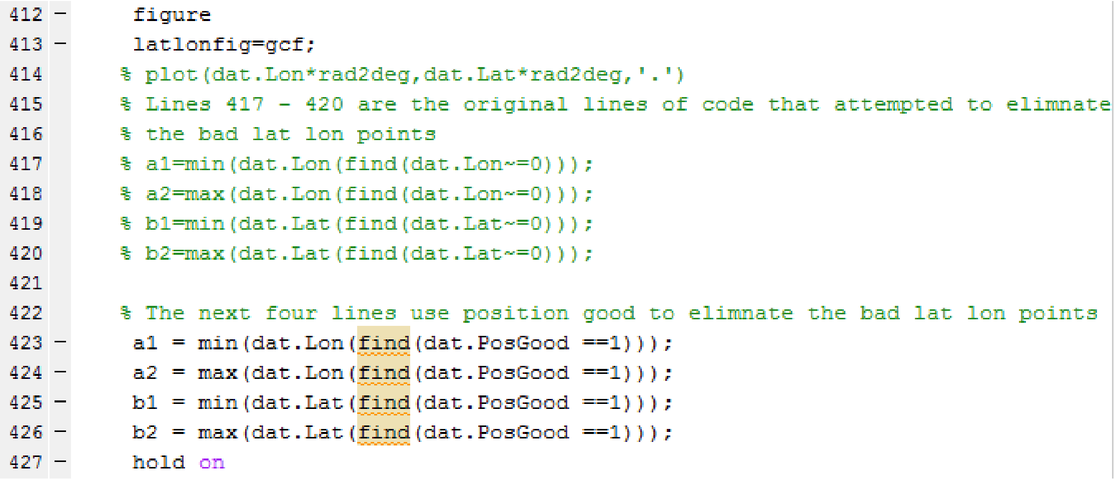


```
955 -      figure
956 -      idx=find(abs(dat.Lat)>7e-6);
957 -      dat.Lonmeters=(dat.Lon-mean(dat.Lon(idx)))*6371000; %.*cos(dat.Lat);
958 -      dat.Latmeters=(dat.Lat-mean(dat.Lat(idx)))*6371000;
959 -      plot(dat.Lonmeters,dat.Latmeters,'.')
960 -      set(gcf,'Name','GPS meters')
961 -      grid on
962 -      xlabel('Longitude variation, meters')
963 -      ylabel('Latitude variation, meters')
964 -      title('GPS position variation, meters')
965 -      gpsmetersfig=gcf;
966 -      stdlon=std(dat.Lonmeters);
967 -      stdlat=std(dat.Latmeters);
968 -      str=sprintf('Lat std dev %3.3g m\nLon std dev %3.3g m\nLat variation %3.3g
969 -      text(stdlon*2.5,-stdlat*3,str);
970 -      axis('equal')
971 -      hold on
```

Figure 28 Original GPS Position Code

Figure 28 shows the original code for the GPS Position plot. Line 956 shows that the code was looking for the magnitude of values of latitude larger than 0.000007. Unfortunately bad latitude values can be larger than 0.000007, and in this example the bad latitude values are larger.

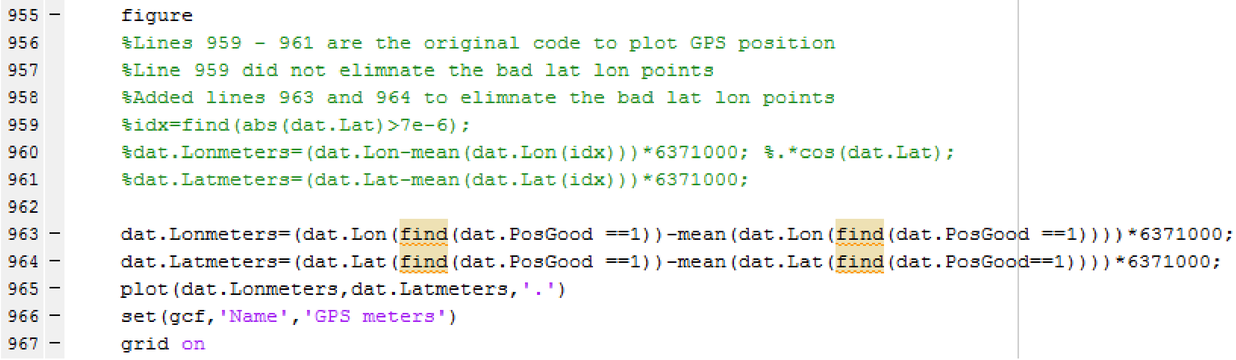
The code was altered to utilize the position good variable. The piccolo telemetry file records the state of the navigation solution under the column “PosGood” where 1 means the navigation solution is GPS/INS and 0 means the navigation solution is in AHRS. Likewise plotpiccolo creates a variable in the dat structure for “PosGood”. The code was edited to look for longitude and latitude data that corresponds with PosGood values of 1, where the navigation solution is GPS/INS. The following two figures show snapshots of the new code.



```
412 -     figure
413 -     latlonfig=gcf;
414      % plot(dat.Lon*rad2deg,dat.Lat*rad2deg,'.')
415      % Lines 417 - 420 are the original lines of code that attempted to elimnate
416      % the bad lat lon points
417      % a1=min(dat.Lon(find(dat.Lon~=0)));
418      % a2=max(dat.Lon(find(dat.Lon~=0)));
419      % b1=min(dat.Lat(find(dat.Lat~=0)));
420      % b2=max(dat.Lat(find(dat.Lat~=0)));
421
422      % The next four lines use position good to elimnate the bad lat lon points
423 -     a1 = min(dat.Lon(find(dat.PosGood ==1)));
424 -     a2 = max(dat.Lon(find(dat.PosGood ==1)));
425 -     b1 = min(dat.Lat(find(dat.PosGood ==1)));
426 -     b2 = max(dat.Lat(find(dat.PosGood ==1)));
427 -     hold on
```

Figure 29 New GPS AckRatio Code

.



```
955 -      figure
956        %Lines 959 - 961 are the original code to plot GPS position
957        %Line 959 did not elimnate the bad lat lon points
958        %Added lines 963 and 964 to elimnate the bad lat lon points
959        %idx=find(abs(dat.Lat)>7e-6);
960        %dat.Lonmeters=(dat.Lon-mean(dat.Lon(idx)))*6371000; %.*cos(dat.Lat);
961        %dat.Latmeters=(dat.Lat-mean(dat.Lat(idx)))*6371000;
962
963 -      dat.Lonmeters=(dat.Lon(find(dat.PosGood ==1))-mean(dat.Lon(find(dat.PosGood ==1))))*6371000;
964 -      dat.Latmeters=(dat.Lat(find(dat.PosGood ==1))-mean(dat.Lat(find(dat.PosGood==1))))*6371000;
965 -      plot(dat.Lonmeters,dat.Latmeters,'.')
966 -      set(gcf,'Name','GPS meters')
967 -      grid on
```

Figure 30 New GPS Position Code

Lines 423 – 426 in Figure 29, and lines 963 – 964 in Figure 30 depict the modified lines of code.

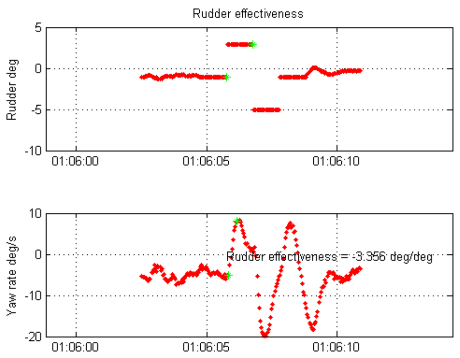
2.8.2. doublet

The “doublet.m” MATLAB script is designed to analyze doublet maneuvers, and calculate their corresponding effectiveness parameters, from the control surface deflection plots and variables created by plotpiccolo. The script is not very user friendly, and contains some errors. As a result a new MATLAB script with a GUI was created to take the place of ‘doublet’. The GUI is called “DoubletPiccoloLog” and is detailed in Section 6.3.1. The doublet maneuvers chapter instructs the user to use DoubletPiccoloLog instead of doublet.m. The rest of this section describes the errors that occur with doublet.m just to inform the user of the issues.

Cloud Cap provides documentation for instructions on how to use “doublet.m” in “Piccolo Doublet Analysis Tool”. The document is located in the CCT MATLAB folder. The document leaves out some important details and errors.

In order for “doublet.m” to function properly the user will have to change the code in the script that pertains to declaring which surface deflection recorded in the piccolo telemetry log file corresponds to which actuator type. As an example the rudder on the Nexstar was wired into actuator 3; thus, rudder deflections were recorded as “Surface3” deflections and stored in the dat structure by plotpiccolo. The code had to be modified as “Rudder = dat.Surface3”.

The doublet scrip contains an error when it calculates rudder effectiveness. The script plots and uses the yaw rate to calculate rudder effectiveness. Rudder effectiveness is defined as change in sideslip angle over change in rudder deflection. The script does not multiply the yaw rate by any unit of time; therefore, the result it provides is change in yaw rate over change in rudder deflection which is not rudder effectiveness.



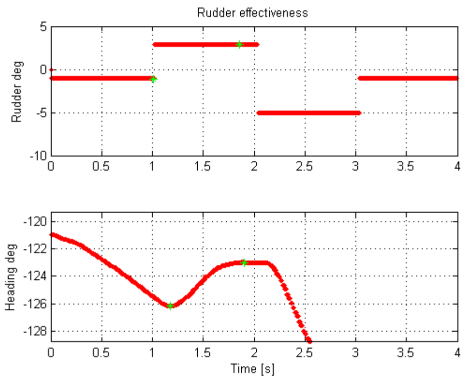


Figure 31 Doublet.m Rudder Doublet

Figure 32 Plotdoublet Rudder Doublet

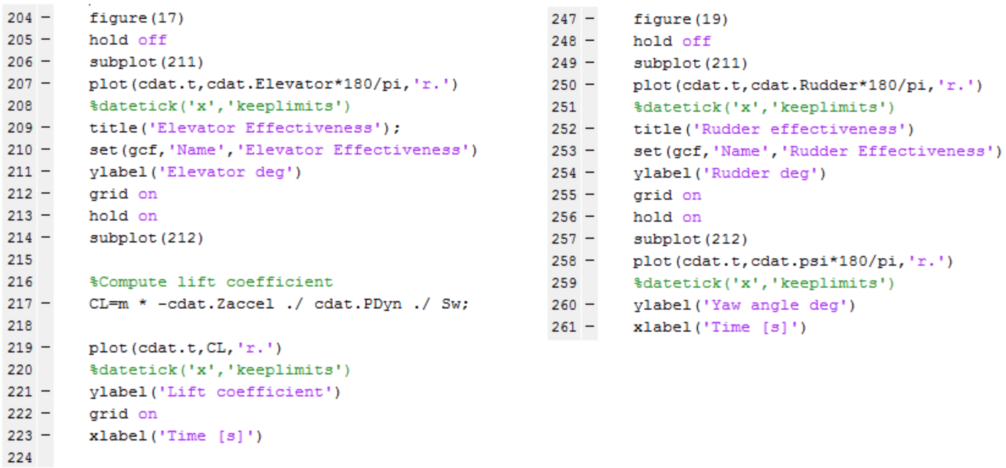
Figure 31 is an example of a rudder doublet maneuver as analyzed by doublet.m. The solution that the calculations provided was -3.356 deg/deg. Figure 32 is an example of the same rudder doublet maneuver as analyzed by the original plotdoublet script, which analyzes doublet maneuvers via their corresponding doublet files. “Plotdoublet.m” calculates the rudder effectiveness from the change in heading over the change in rudder deflection. The change in heading is essentially the change in yaw, which in the case of quick rudder doublet maneuvers, can be used to accurately represent the rudder effectiveness. The value produced by “plotdoublet.m” was -0.792 deg/deg which was considerably lower than the incorrect value given by “doublet.m”.

2.8.3. plotdoublet

“plotdoublet.m” is a MATLAB script that is designed to analyze doublet maneuvers, and calculate their corresponding effectiveness parameters, from the doublet files that can be generated by PCC during doublet maneuvers. The script generically plots all of the plots that pertain to aileron, rudder, and elevator doublets. The data from the doublet file is loaded using the ‘loadlogfilepiccolo.m’ script. Loadlogfilepiccolo simply imports data from raw text files and assigns each column to its own variable.

Plotdoublet relies on user input to determine which analysis to proceed with after a doublet file is loaded. Once an analysis has been selected the script calls the script ‘deltadoublet.m’. Deltadoublet provides the user with the capability to select 4 points on the appropriate effectiveness plot. The four user selected points are used to calculate the corresponding effectiveness parameter. Section 6.2.1 describes the use of plotdoublet for doublet maneuvers in greater detail.

Initially plotdoublet was designed to plot elevator deflection versus CL for elevator doublets and rudder deflection versus heading for rudder doublets. These plots were changed to include the pitch and yaw rates to help the user determine where to place the 3rd and 4th points on the effectiveness plots.

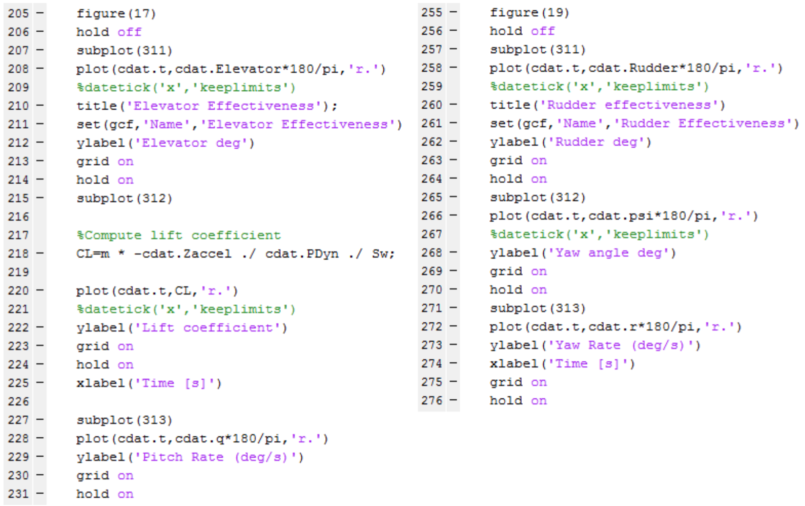


```
figure(17)
hold off
subplot(211)
plot(cdat.t,cdat.Elevator*180/pi,'r.')
%datetick('x','keeplimits')
title('Elevator Effectiveness');
set(gcf,'Name','Elevator Effectiveness')
ylabel('Elevator deg')
grid on
hold on
subplot(212)

%Compute lift coefficient
CL=m * -cdat.Zaccel ./ cdat.PDyn ./ Sw;

plot(cdat.t,CL,'r.')
%datetick('x','keeplimits')
ylabel('Lift coefficient')
grid on
xlabel('Time [s]')

```

```
figure(19)
hold off
subplot(211)
plot(cdat.t,cdat.Rudder*180/pi,'r.')
%datetick('x','keeplimits')
title('Rudder effectiveness')
set(gcf,'Name','Rudder Effectiveness')
ylabel('Rudder deg')
grid on
hold on
subplot(212)
plot(cdat.t,cdat.psi*180/pi,'r.')
%datetick('x','keeplimits')
ylabel('Yaw angle deg')
xlabel('Time [s]')
```

Figure 33 Original Plotdoublet Code

Figure 33 depicts the original plotdoublet code for the elevator and rudder plots. Initially they consisted of two subplots without the pitch and yaw rates.



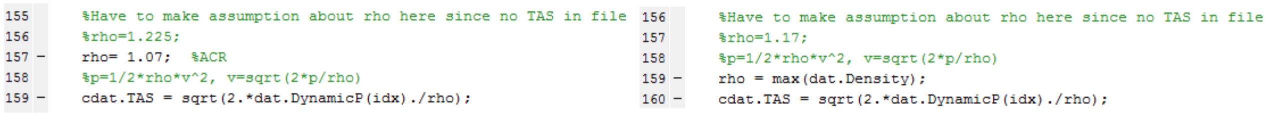
```
205     figure(17)
206     hold off
207     subplot(311)
208     plot(cdat.t,cdat.Elevator*180/pi,'r.')
209     %datetick('x','keeplimits')
210     title('Elevator Effectiveness');
211     set(gcf,'Name','Elevator Effectiveness')
212     ylabel('Elevator deg')
213     grid on
214     hold on
215     subplot(312)
216
217     %Compute lift coefficient
218     CL=m * -cdat.Zaccel ./ cdat.PDyn ./ Sw;
219
220     plot(cdat.t,CL,'r.')
221     %datetick('x','keeplimits')
222     ylabel('Lift coefficient')
223     grid on
224     hold on
225     xlabel('Time [s]')
226
227     subplot(313)
228     plot(cdat.t,cdat.q*180/pi,'r.')
229     ylabel('Pitch Rate (deg/s)')
230     grid on
231     hold on
```

```
255     figure(19)
256     hold off
257     subplot(311)
258     plot(cdat.t,cdat.Rudder*180/pi,'r.')
259     %datetick('x','keeplimits')
260     title('Rudder effectiveness')
261     set(gcf,'Name','Rudder Effectiveness')
262     ylabel('Rudder deg')
263     grid on
264     hold on
265     subplot(312)
266     plot(cdat.t,cdat.psi*180/pi,'r.')
267     %datetick('x','keeplimits')
268     ylabel('Yaw angle deg')
269     grid on
270     hold on
271     subplot(313)
272     plot(cdat.t,cdat.r*180/pi,'r.')
273     ylabel('Yaw Rate (deg/s)')
274     xlabel('Time [s]')
275     grid on
276     hold on
```

Figure 34 New Plotdoublet Code

Figure 34 depicts the modified plotdoublet code. Both elevator and rudder plots were altered to consist of 3 subplots where the 3rd subplot graphed the pitch and yaw rate versus time. The 3rd subplots are declared on lines 227 – 231, and lines 271 – 276.

When the plotdoublet code was written the doublet files did not log the air density. The plotdoublet code uses the air density to calculate the true airspeed. The true airspeed is used in calculating the dimensionless roll rate for aileron effectiveness calculations. In one of the software updates the doublet file was changed to where it does log the air density as estimated by the piccolo.



```
%Have to make assumption about rho here since no TAS in file
%rho=1.225;
rho= 1.07;  %ACR
%p=1/2*rho*v^2, v=sqrt(2*p/rho)
cdat.TAS = sqrt(2.*dat.DynamicP(idx)./rho);
```

```
%Have to make assumption about rho here since no TAS in file
%rho=1.17;
%p=1/2*rho*v^2, v=sqrt(2*p/rho)
rho = max(dat.Density);
cdat.TAS = sqrt(2.*dat.DynamicP(idx)./rho);
```

Figure 35 Original and Modified Air Density Code

Figure 35 depicts the original code versus the modified plotdoublet code. In the original code the air density had to be declared manually by the user in the code. In the modified version the air density value is extracted directly from the doublet file.

CHAPTER III

SIMULATOR MECHANICS

3. Introduction

Cloud Cap provides some documentation on the simulator and how to set it up in the document “Piccolo Simulator”. “Piccolo Simulator” is located in the Cloud Cap folder under “Piccolo Docs\Software”. The next chapter covers the entire setup process in great detail. Generally speaking the modeling process consists of building a model in AVLEditor and conducting an alpha sweep to create an alpha file that serves as a look up table for the simulator while simulations are being ran. The simulator is also used to generate a vehicle file which consists of calculated values for vehicle gains. In the setup process the vehicle file is loaded into the piccolo autopilot’s vehicle gain settings where they act as initial vehicle gains; therefore, it is important that these values are reasonably accurate.

The purpose of this chapter is to explain what exactly an alpha file is and how the simulator uses the alpha file to calculate vehicle gains. The objective for this chapter is to serve as a starting point to debug any problems that may arise with any vehicle gain estimates on future aircraft models. This chapter walks the user step by step through the calculations that the simulator performs to calculate vehicle gains that are pertinent to the Fixed Wing Generation 2 firmware. This chapter also highlights the important difference between the axes in AVLEditor and the axes in the simulator along with how the impact that the axes have in creating simulator files.

3.1. Alpha Sweep File

The entire purpose of creating an aircraft model in AVLEditor is to generate an alpha sweep file or alpha file. The alpha file is used by the simulator to simulate the aerodynamics and stability characteristics of an aircraft. Additionally the simulator uses the alpha file, along with some parameters defined in the simulator file, to calculate estimates of the aircraft's vehicle parameters. As a part of the process the simulator estimated vehicle parameters are the only way that Cloud Cap provides the user to develop initial estimates before flying.

Alpha files are generated by running "AVL Analysis" in the AVLEditor. AVL Analysis uses Cloud Cap's modified version of AVL 3.32 to perform run cases for AVLEditor aircraft models. Upon initiating AVL Analysis the user is prompted for the alpha sweep range and increment. Cloud Cap's modified AVL, "avlcct", simply performs the run cases defined by the user in the alpha sweep prompt. The aircraft model is loaded into AVL with all of the geometry and aircraft data defined in the AVLEditor model and ran at each alpha as specified by the user input. After each run avlcct automatically records the stability derivatives in a specific format in the alpha file. Additionally avlcct also records specific aircraft data into the alpha file. Running an alpha sweep is the same thing as manually iteratively executing run cases at different alphas in AVL. The AVLEditor models save text files with all of the necessary data to load into AVL and run manually.

The only variable that changes with each run is the alpha angle. Even the airspeed remains constant for each run. Alpha angles are not always the same as angle of attack. An alpha angle is the angle between the rotated AVL axes and the original AVL axes. When the alpha angle is changed AVL rotates the aircraft by rotating the AVL axes. If the wings are not level with the XZ plane of the AVL axes, i.e. incidence, then the alpha angle will not be the same as the angle of attack.

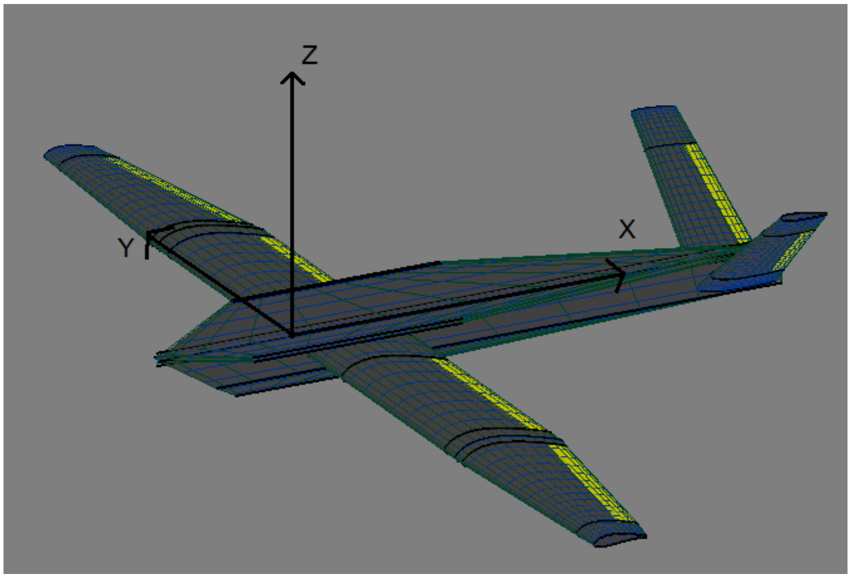


Figure 36 AVLEditor Axes

Figure 36 depicts the AVL axes in an AVLEditor model. Positive Z is up, positive X is downstream, and positive Y is starboard. (0,0,0) is determined by the user when entering geometry measurements. In the figure the model's (0,0,0,) as the center of the wing leading edge.

Parts of the stability derivatives that an alpha file contains are specific to the deflection of control surfaces. Each control surface will have a calculation for coefficients of change in lift, drag, roll moment, pitching moment, yaw moment, and side force. The sign conventions of deflections go as follows:

1) Any control surface that does not travel solely in the z axis (such as a conventional rudder) positive deflection is trailing edge down and negative deflection is trailing edge up.

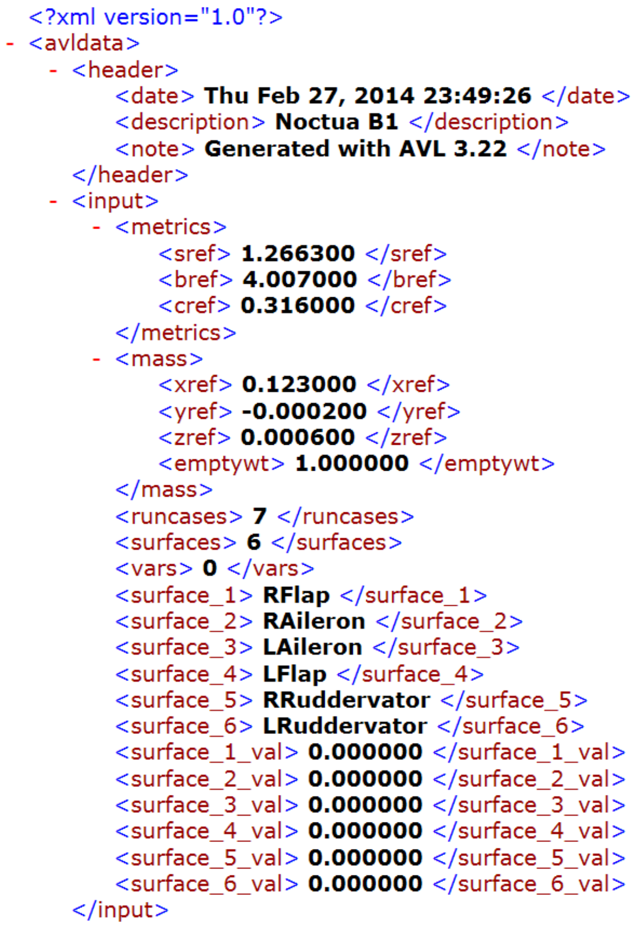
2) Any control surface that travels solely in the z axis, or purely vertical (such as a conventional rudder), positive deflection is trailing edge starboard; negative deflection is trailing edge port.

Additionally positive moment directions are:

1) Positive pitch moment is pitch up
2) Positive roll moment is roll starboard
3) Positive yaw moment is yaw clockwise

These sign conventions dictate the signs of the recorded Clδ, Cmδ, and Cnδ for each control surface. The sign conventions are important because they are a contributing factor to the results of the simulator vehicle parameter calculations. Additionally the sign convention of control surface deflections can be changed in AVLEditor by changing the value of the “Gain” of a control surface from +1 to -1. It is not recommended to do so because the simulator will automatically adjust surface deflections as necessary for particular control surface mixtures and control surface types as long as they are configured correctly.

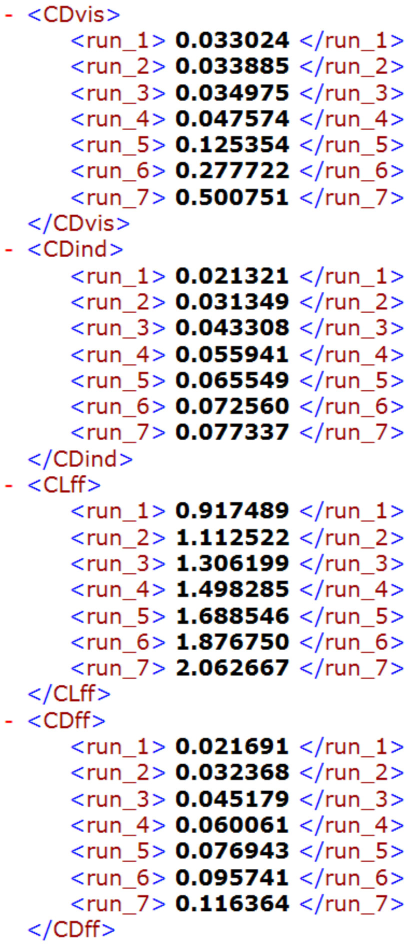
AVLEditor allows control surfaces to be named. Control surface names impact how the simulator calculates vehicle parameter estimates, and is critical to the modeling process. Additionally if Y-Symmetry is used the alpha sweep will automatically designate mirrored control surfaces with a prefix for left, “L”, and right “R” control surface designations.



```
   <?xml version="1.0"?>
 - <avldata>
   - <header>
       <date> Thu Feb 27, 2014 23:49:26 </date>
       <description> Noctua B1 </description>
       <note> Generated with AVL 3.22 </note>
     </header>
   - <input>
     - <metrics>
         <sref> 1.266300 </sref>
         <bref> 4.007000 </bref>
         <cref> 0.316000 </cref>
       </metrics>
     - <mass>
         <xref> 0.123000 </xref>
         <yref> -0.000200 </yref>
         <zref> 0.000600 </zref>
         <emptywt> 1.000000 </emptywt>
       </mass>
       <runcases> 7 </runcases>
       <surfaces> 6 </surfaces>
       <vars> 0 </vars>
       <surface_1> RFlap </surface_1>
       <surface_2> RAileron </surface_2>
       <surface_3> LAileron </surface_3>
       <surface_4> LFlap </surface_4>
       <surface_5> RRuddervator </surface_5>
       <surface_6> LRuddervator </surface_6>
       <surface_1_val> 0.000000 </surface_1_val>
       <surface_2_val> 0.000000 </surface_2_val>
       <surface_3_val> 0.000000 </surface_3_val>
       <surface_4_val> 0.000000 </surface_4_val>
       <surface_5_val> 0.000000 </surface_5_val>
       <surface_6_val> 0.000000 </surface_6_val>
     </input>
```

Figure 37 Alpha File Header

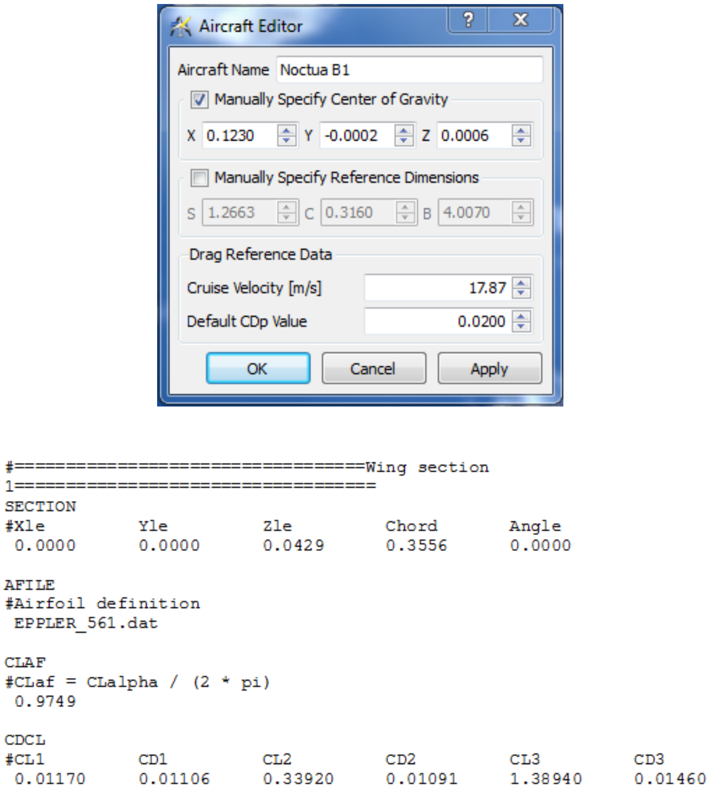
Figure 37 is a snapshot of the beginning of an alpha file. The alpha file contains the cg location (xref, yref, zref) and reference dimensions (sref bref cref). Sref should be the wing area, bref should be the wingspan, and cref should be the wing chord. All the units should be SI. The values for the reference dimensions and cg location are extracted directly from the AVL file of the aircraft model. The values are also located in the aircraft data window in AVLEditor. The surface numbers are shown as well to indicate which surface number represents which control surface. The control surface stability derivatives are recorded according to their surface numbers as C_d#. For example the change in pitching moment per change in deflection of “Lruddervator” in Figure 37 would be recorded as “Cmd6”. The control surface derivatives are recorded at the end of the alpha file.



```
- <CDvis>
      <run_1> 0.033024 </run_1>
      <run_2> 0.033885 </run_2>
      <run_3> 0.034975 </run_3>
      <run_4> 0.047574 </run_4>
      <run_5> 0.125354 </run_5>
      <run_6> 0.277722 </run_6>
      <run_7> 0.500751 </run_7>
  </CDvis>
- <CDind>
      <run_1> 0.021321 </run_1>
      <run_2> 0.031349 </run_2>
      <run_3> 0.043308 </run_3>
      <run_4> 0.055941 </run_4>
      <run_5> 0.065549 </run_5>
      <run_6> 0.072560 </run_6>
      <run_7> 0.077337 </run_7>
  </CDind>
- <CLff>
      <run_1> 0.917489 </run_1>
      <run_2> 1.112522 </run_2>
      <run_3> 1.306199 </run_3>
      <run_4> 1.498285 </run_4>
      <run_5> 1.688546 </run_5>
      <run_6> 1.876750 </run_6>
      <run_7> 2.062667 </run_7>
  </CLff>
- <CDff>
      <run_1> 0.021691 </run_1>
      <run_2> 0.032368 </run_2>
      <run_3> 0.045179 </run_3>
      <run_4> 0.060061 </run_4>
      <run_5> 0.076943 </run_5>
      <run_6> 0.095741 </run_6>
      <run_7> 0.116364 </run_7>
  </CDff>
```

Figure 38 Alpha File CL CD

Figure 38 depicts some of the CL and CD values that the alpha file records. The simulator uses CLff, trefftz plane lift, for calculations that require CL. The simulator uses CDff, trefftz plane drag, for induced drag, and it uses Cdvis for viscous drag. AVL analysis calculates viscous drag with two components.



Aircraft Name Noctua B1

☑ Manually Specify Center of Gravity

X 0.1230 Y -0.0002 Z 0.0006

□ Manually Specify Reference Dimensions

S 1.2663 C 0.3160 B 4.0070

Drag Reference Data

Cruise Velocity [m/s] 17.87

Default CDp Value 0.0200

```
#================================Wing section
1=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0000      0.0000      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1         CD1         CL2         CD2         CL3         CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460
```

Figure 39 AVL File Drag Polar

The first component is the user input CDp, or profile drag as cloud cap defines it (Piccolo Simulator pg. 8). The second component comes from the Xfoil generated drag polar. Figure 39 shows an example from an AVLEditor model where the drag polar has been generated by Xfoil. The drag polar is recorded as 3 cdcl points. The first point represents C_{lmin}, the second point represents C_{dmin}, and the third point represents C_{lmax}. AVL extrapolates specific parabolic functions between and beyond each set of points to estimate drag values at a given C_l. At each alpha run AVL analysis will calculate Cdvis by combining CDp and Cd where Cd is extracted from the drag polar. If there is no drag polar Cdvis will simply be equal to CDp for each alpha run. If there is no drag polar and no CDp then Cdvis will be equal to zero. If there is no viscous drag in the model it will be noticeable during simulations. When the Nexstar model was first created there was no viscous drag and the aircraft seemed to descend rapidly even at shallow descents. Once viscous drag was added to the model the aircraft's descent rate decreased noticeably.

There is one cloud cap document that defines each recorded stability derivative; however, the document does not describe which coefficients are actually used and how. The stability derivative list can be found in the document "Piccolo Simulator" pgs. 50-51.

3.1.1. Rudder Glitch

There are a couple of rudder glitches that can affect the simulators calculation of rudder effectiveness. The first glitch occurs if a vertical rudder surface is mirrored using Y-Symmetry.

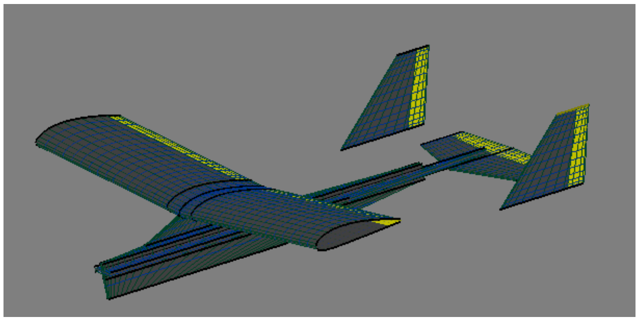


Figure 40 AVLEditor Dual Rudder Glitch

Figure 40 depicts an aircraft model that was modified to have two rudders as an example. If the rudders are mirrored the alpha sweep will calculate a positive Cndr for one surface and a negative Cndr for the other. In this scenario both the rudder effectiveness and vertical tail arm estimates will be 0. If a model has multiple vertical rudders they should be modeled separately without Y-Symmetry; however, there is another glitch that can cause similar headaches.

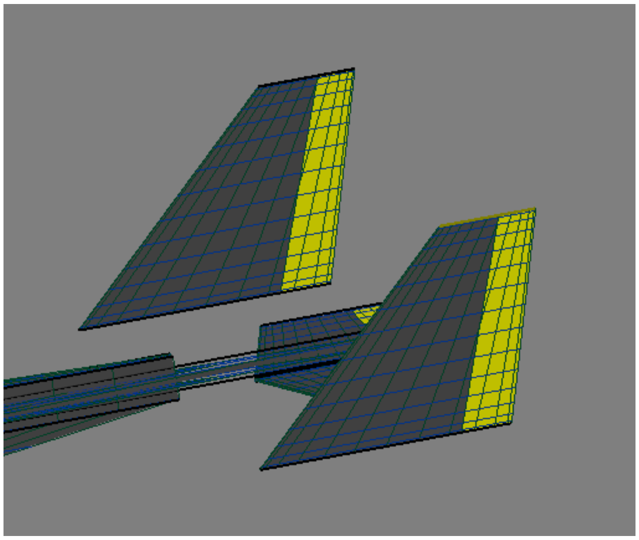


Figure 41 Rudder Section Numbering

The order of the section numbering on a vertical surface determines the sign convention of the control surface's stability derivatives. Figure 41 depicts a twin rudder model. The surfaces did not use Y-Symmetry. Section 1 is at the top of the vertical surfaces and is highlighted yellow on the left vertical surface. In this configuration with the section numbering beginning at the top Cndr will be positive and Cydr will be negative. In that scenario the simulator would calculate rudder effectiveness as a negative number and rudder power as a positive number. Every time that the AVL model is saved the direction of the section numbers changes. If the model in Figure 41 was to be saved section 1 would be at the bottom instead of the top, and Cndr would be negative and Cydr would be positive. In that scenario the simulator would calculate rudder effectiveness as a positive number and rudder power as a negative number. Technically the piccolo wants rudder effectiveness to be negative and rudder power to be positive; however, the autopilot will automatically correct the sign convention of the rudder effectiveness and power if they are input with the wrong sign.

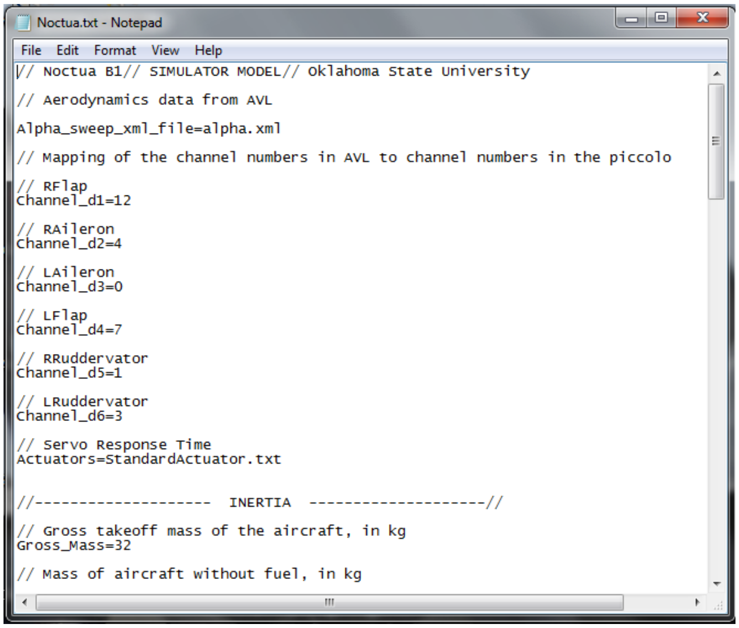
The section number order can be a problem when multiple vertical rudder surfaces are being created. If the surfaces are not created within the same save they can have opposite section numbering directions which will result in rudder effectiveness calculations of 0.

3.2. Simulator File

The simulator file is a text file that the simulator uses to load aircraft models into simulations. The simulator file directs the simulator to the appropriate files to load for specific parameters and additionally there are many different parameters and configurations that can be specified in the simulator file itself.

Cloud Cap provides a lot of documentation that details the numerous parameters that can be defined in the simulator file. The document is "Piccolo Simulator." Piccolo Simulator does a decent job at explaining how the different components of the simulator file work; therefore, this section provides a quick overview of the simulator file functions and provides some clarification on the axes of the simulator versus the axes of AVLEditor.

The simulator file allows the user to reference files to load to simulate the performance of the aircraft dynamics (alpha file), the propeller performance (prop file), the engine performance (motor file), servo performance (actuator file), and sensor performance (sensor file). The alpha, prop, and motor (if gas engine) files are required for the simulator to function. The piccolo software comes with text files for the simulator to load to simulate servo and sensor performance; however, these files are not required to run simulations. While the alpha file is used to load aircraft dynamics the empty, and takeoff mass along with the mass moments of inertia (Ix, Iy, Iz) have to be input manually by the user into the simulator file.



```
// Noctua B1// SIMULATOR MODEL// Oklahoma State University

// Aerodynamics data from AVL

Alpha_sweep_xml_file=alpha.xml

// Mapping of the channel numbers in AVL to channel numbers in the piccolo

// RFlap
Channel_d1=12

// RAileron
Channel_d2=4

// LAileron
Channel_d3=0

// LFlap
Channel_d4=7

// RRuddervator
Channel_d5=1

// LRuddervator
Channel_d6=3

// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
```

Figure 42 Simulator File Example

Figure 42 depicts the intro to the Noctua B1 simulator file. Any line that begins in "//" is only a comment line. The simulator will not read those lines. Parameters that are defined cannot have spaces in them. There cannot be any spaces before and after the equals sign. If there is a space in a line that line will not be read by the simulator.

The simulator file references the alpha file that the simulator will load to simulate the flying qualities of the aircraft. Essentially the simulator treats the aircraft as a point mass, at the location of the center of gravity as read from the alpha file. The point mass will have the characteristics of the aircraft's AVL model as defined by the stability derivatives and coefficients in the alpha file. The simulator file accepts input parameters that can define hard points such as landing gear and ground contact points. The landing gear parameters are used to simulate the locations of the wheels. The landing gear is also used to simulate forces that will be applied to the center of gravity point mass during landings and launch. The ground contact points are meant to define certain points on the aircraft that extend from the center of gravity point mass so that if they were to come into contact with the ground the simulator would respond appropriately. The ground contact points do not have any direct effect on the aircraft dynamics; they are only there to simulate external forces that would be created from contacting the ground. The simulator file also allows the location of the propeller to be input, so that the simulator can simulate the forces and moments that will act on the aircraft, or point mass, when the propeller is spinning. Additionally the simulator file allows the user to specify the location of the autopilot, and the GPS receiver. As a result the simulator can simulate the forces and moments that would act on the center of gravity point mass and the forces and moments that the autopilot would measure in flight simulations.

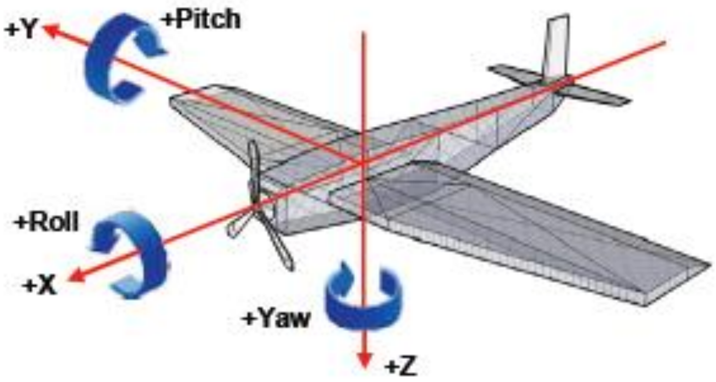


Figure 43 Simulator Axes

All of the locations of the simulator file parameters that are defined by locations are measured from the center of gravity of the aircraft. The center of gravity is to be the reference location, and it is defined in the alpha file as "(xref,yref,zref)"; however, the sign convention of axes used by the Simulator are different than that of AVLEditor. Figure 43 is in the document "Piccolo Simulator", pg. 6 and it depicts the sign convention used by the simulator. The sign convention of the rates is the same as AVLEditor, but the sign convention of the z and x axes is different. Positive x axis is upstream, versus downstream in AVLEditor. The positive z axis is down, versus up in AVLEditor. With respect to the aircraft's dynamics model from the alpha file the change in axes does not create a problem interfacing between the simulator file and the avl model. Even though the center of gravity location will be different in the simulator than it is in ALVEditor all of the stability derivatives in the alpha file will still be applied to the point mass at the center of gravity regardless of where exactly the simulator believes it is located. Only the parameters that are defined by locations will be affected by the axis sign convention change. The landing gear, ground contact points, autopilot location, GPS location, and propeller locations must all be measured with respect to the center of gravity with the simulator axes sign convention.

3.3. Vehicle Parameter Estimates

The simulator uses the alpha file, and some parameters from the simulator file, to calculate the vehicle parameters. There is no documentation that details how each calculation is performed. Without knowing how the simulator calculates these parameters it was nearly impossible to debug modeling issues let alone know if the values estimated were accurate or garbage. As a result how the simulator calculates each vehicle parameter calculation, that's pertinent to the autopilot's required vehicle parameters for fixed wing generation two firmware, was determined through experimentation.

A few of the vehicle parameters are dependent on the stability derivatives of control surfaces according to their controls surface, or actuator, type. Rudder Effectiveness, Rudder Power, and Vertical Tail Arm are dependent on stability coefficients over change in rudder deflection. Elevator Effectiveness and Elevator Power are dependent on stability coefficients over change in elevator deflection. Aileron Effectiveness and Aileron Power are dependent on stability coefficients over change in aileron deflection. Flap effectiveness is dependent on stability coefficients over change in flap deflection. The calculation of these vehicle parameters by the simulator depends on what actuator type the simulator interprets each control surface to be, as recorded in the alpha file.

The simulator interprets the actuator type of each control surface based on its name in the alpha file. For example consider an alpha file with a control surface named “Rudder” on surface number 6. The Rudder Effectiveness and Power calculations, along with Vertical Tail Arm, depend both on the stability derivative C_n/δ_r, or change in yawing moment over change in rudder deflection, amongst other variables. Every control surface in the alpha file will have its own C_n/δ values for each alpha run; however, in this scenario the simulator would interpret Cnd6 as C_n/δ_r; thus, only Cnd6 would be used to calculate Rudder Effectiveness and Power. This means that the names assigned to control surfaces in AVLEditor are extremely important. Similar to the rudder example control surfaces named “Elevator” are included in Elevator Effectiveness and Power calculations. Control surfaces named “Aileron” are included in aileron effectiveness and power calculations. Control surfaces named “Flap” are included in flap effectiveness calculations.

The piccolo has built in mixed actuator types, such as Ruddervators (elevator + rudder), and Elevons (elevator + aileron). The simulator is programmed to recognize the built in mixed actuator types as mixed control surfaces in alpha files. The simulator, theoretically, is supposed to un mix mixed control surfaces when it calculates the vehicle parameters dependent on specific control surface functions. Some of the mixed actuator types require sign convention changes that are dependent on whether or not the control surface is on the left or right side of the aircraft. As a result mixed control surfaces names, in AVLEditor, must include designations for left, “L”, or right, “R”. Recall that when a control surface is mirrored, via Y-Symmetry, in AVLEditor, the alpha sweep function will record the left and right control surface names with “L” and “R” prefix designations automatically. For example consider an alpha file with “Lruddervator”, and “Rruddervator”. Ruddervators are not completely vertical; therefore, the sign convention, of the stability coefficients over surface deflection, complies with elevator sign convention; positive deflection trailing edge down, negative deflection trailing edge up. With respect to rudder deflection a positive rudder deflection will deflect rudders trailing edge starboard, which will induce a positive yawing moment; however, a positive rudder deflection will deflect the Lruddervator negative because the ruddervators follow elevator sign convention. As such the Cnd recorded for the Lruddervator will be negative while the Cnd recorded for the Rruddervator will be positive. Theoretically, with respect to rudder dependent parameter calculations, the simulator is supposed to properly unmix the ruddervators and correct the sign convention of the Lruddervator stability derivatives that apply, so that the Cnd for both ruddervator control surfaces will be treated as positive and not cancel each other out. The simulator does do this as it should when calculating rudder effectiveness and power; however, it does not do this for calculating vertical tail arm and will calculate 0 for the vertical tail arm estimates.

In order for the simulator to have a change to attempt calculating these vehicle parameters they must be named as control surfaces that the piccolo, and thus the simulator, recognize. Every mixed actuator type will be unmixed according to their mixture definitions. Table 1 below comes from the list of actuator types in PccUsersGuide pgs. 102 – 103. The table is meant to provide an example for how to name control surfaces in AVLEditor according to their mixtures. Note that the ailerons are the only non-mixed actuator types that require “L” and “R” designations in its AVLEditor control surface names. Elevators, flaps, and rudders can be designated as “L” and “R”; however, it does not change the manner in which the simulator uses their stability derivatives for vehicle parameter calculations.

Table 1 Actuator Control Surface Names

Actuator Type	Alpha File Control Surface Name
L. Aileron	Laileron
R. Aileron	Raileron
Elevator	Elevator or Lelevator Relevator
Canard	Canard or Lcanard Rcanard
Rudder	Rudder or Lrudder Rrudder
Flap	Flap or Lflap Rflap
L. Elevon	Lelevon
R. Elevon	Relevon
L. Ruddervator	Lruddervator

R. Ruddervator	Rruddervator
L. Inv. Ruddervator	LinvRuddervator
R. Inv. Ruddervator	RinvRuddervator
L. Canarderon	Lcanarderon
R. Canarderon	Rcanarderon

Note that the calculation of the vehicle parameters is not critical for the simulator to accurately simulate how the aircraft flies in simulations. The calculations do not affect how the aircraft model will fly in the simulator with respect to how the simulator models the aircraft's dynamics. The simulator calculates these estimates for the sole purpose of entering them into the piccolo autopilot. Even if vehicle parameters are calculated incorrectly it will not have an effect on the simulator side, but it will affect how the autopilot responds if the incorrect values are entered into the autopilot's vehicle parameters. For example consider a scenario where control surface 6 is assigned to actuator number 3 and is named elevator in the alpha file, but it is actually a ruddervator and is designated as ruddervator by the actuator type in the piccolo. The rudder effectiveness, power, and vertical tail arm calculations will not be correct because they will not include control surface 6. The piccolo will command control surface 6 deflections when it commands rudder deflections to actuator 3, and the simulator will correctly model the changes that deflection will create with respect to the yawing moment of the aircraft. The difference comes on the piccolo side where the autopilot will not correctly estimate how much rudder deflection is required to create a specific change in yawing moment; and thus, can adversely affect the autopilot's performance in simulations.

In order to determine how the simulator was calculating each applicable vehicle parameter values of specific stability derivative and coefficients in an alpha file were altered to observe how they affected the simulator's calculations. Initially the alpha file coefficients that were altered were determined by the vehicle parameter's definition, i.e. CL at zero elevator began with changing alpha run values of the CL and Cm coefficients. If a parameter could not be calculated each coefficient in the alpha file was painstakingly altered to observe the effects of each coefficient on the simulator's calculations.

It should be noted that, even though they are presented as such, there is no guarantee that any of the following sections detail the exact logic of the simulator as it performs vehicle parameter calculations. At the very least the user will have a good idea as to how the simulator is calculating a particular parameter; thus, the user will be able to ascertain any errors and determine how to calculate a parameter manually if necessary.

3.4. Scaling Terms

In the event that the elevator, rudder, or aileron effectiveness is determined from doublet maneuver tests and the results do not match the simulator estimated values the simulator file offers the user the capability to alter the simulator vehicle parameter calculations via scaling terms. The scaling terms are as follows: Cld, Cmd, Cnd.

The scaling terms do not actually change the values of their corresponding stability derivatives in the alpha file. The scaling terms exist only as multipliers from the simulator's point of view. The format for the scaling terms in the simulator file is as follows:

1) Cld_scaler_d#=

2) Cmd_scaler_d#=

3) Cnd_scaler_d#=

4) CLd_scaler_d#=

5) CDffd_scaler_d#=

The “#” designates the number of the control surface as it is assigned by the alpha file. There cannot be any spaces anywhere in the lines of code. Not even after the equals sign. If there is a space the simulator will ignore the line.

3.5. CL at Zero Elevator

The simulator calculates C_L at zero elevator by using specific coefficients in the alpha file to find the alpha that the aircraft model trims at with zero elevator deflection and thus calculate the corresponding lift at that alpha. By altering various alpha file coefficients and analyzing their effects on the C_L at zero elevator calculation it was determined that the following alpha file coefficients are used by the simulator to calculate C_L at zero elevator deflection:

1) Cmtot
2) CLff
3) CDff
4) Cdvis

The first calculation uses the Cmtot versus alpha curve to solve for the alpha that the aircraft model will trim at, or where Cmtot is equal to zero. The four coefficients are calculated at each alpha run with no control surface deflections, so the simulator only has to use their corresponding alpha curves to solve for their values at specific alphas.

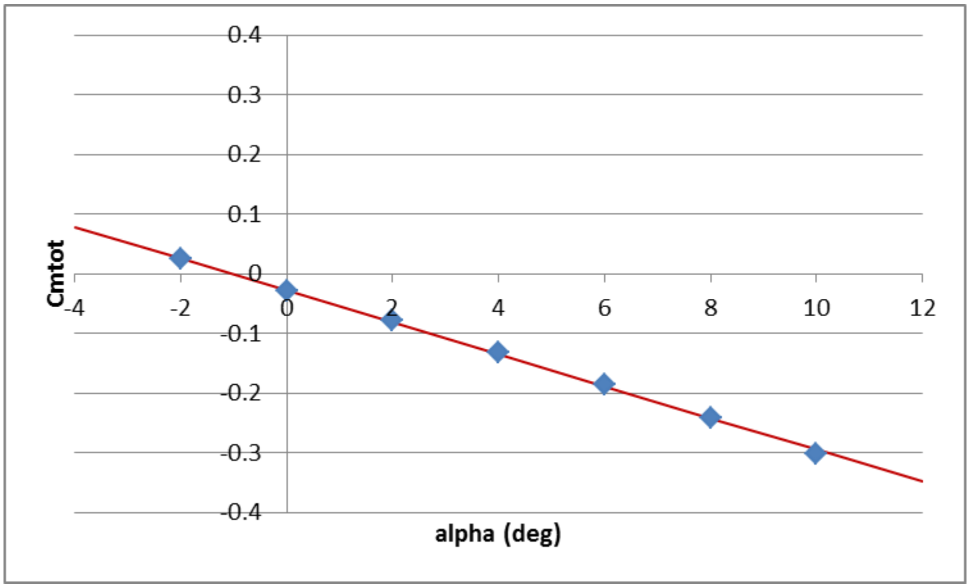


Figure 44 Alpha File Cmalpha

Figure 44 depicts a Cmtot versus alpha curve, from the Nexstar model. The simulator calculates the equation of the line between the two data points that cross the alpha axis to calculate what the value of alpha is at Cmtot equal to zero. In the figure above the red line represents the equation of the line between the two points at alpha = 0,-2. In the event that an alpha file does not contain enough data points for the Cmtot values to cross over Cmtot = 0 the simulator will use the two data points closest to the alpha axis to calculate the equation of the Cmtot alpha line and back out a value for alpha at Cmtot = 0. The process is repeated for all three of the other coefficients using the same two alpha run points that were used to calculate alpha at Cmtot = 0.

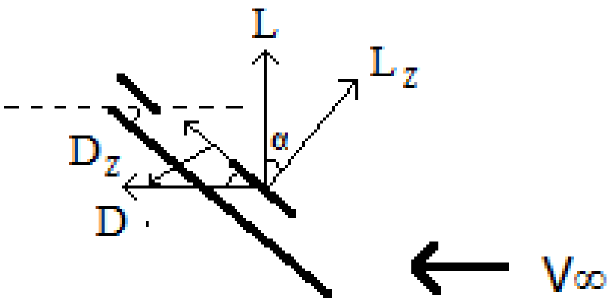


Figure 45 Simulator Lift Drag Vectors

Figure 45 depicts the free body diagram of the Nexstar aircraft model when the elevator is deflection is 0 degrees. The trim alpha angle of the Nexstar with zero elevator deflection was -1.04°; therefore, the diagram depicts the aircraft pitching down. When the simulator calculates C_L at different angles of attack it actually calculates C_L based on the z force that the autopilot would measure in such a scenario. L_Z represents the force that would act in the z direction of the autopilot axes. In this scenario the drag of the aircraft has a component that would subtract from the z force that the autopilot would measure. The drag force z component is represented in the figure as D_Z. The measured C_L would be $C_L \cos(\alpha) + C_D \sin(\alpha)$. Initially it was found that the drag component was decreasing the value of C_L at zero elevator; however, it was later found that the drag component is added it only subtracted from the value of the Nexstar model because the alpha angle was negative. This was verified by testing the calculation of a model where the trim alpha angle was positive. In that scenario the drag component was added to the C_L calculation, not subtracted.

The following steps detail the steps that the simulator appears to follow to calculate C_L at zero elevator. Remember that the simulator uses the two alpha runs that encompass alpha at Cmtot = 0 or the last two alpha runs closest to Cmtot = 0 to calculate the line equations of all of the coefficients versus alpha.

1) Calculate α trim at zero elevator deflection

$$0 = C_{mtot} = \frac{\Delta C_{mtot}}{\Delta\alpha} * \alpha + Intercept \Rightarrow \alpha_{trim} = \frac{-Intercept}{\frac{\Delta C_{mtot}}{\Delta\alpha}}$$

2) Calculate CLff at α trim

$$C_{Lff} = \frac{\Delta C_{Lff}}{\Delta\alpha} * \alpha_{trim} + Intercept$$

3) Calculate CDff at α trim

$$C_{Dff} = \frac{\Delta C_{Dff}}{\Delta\alpha} * \alpha_{trim} + Intercept$$

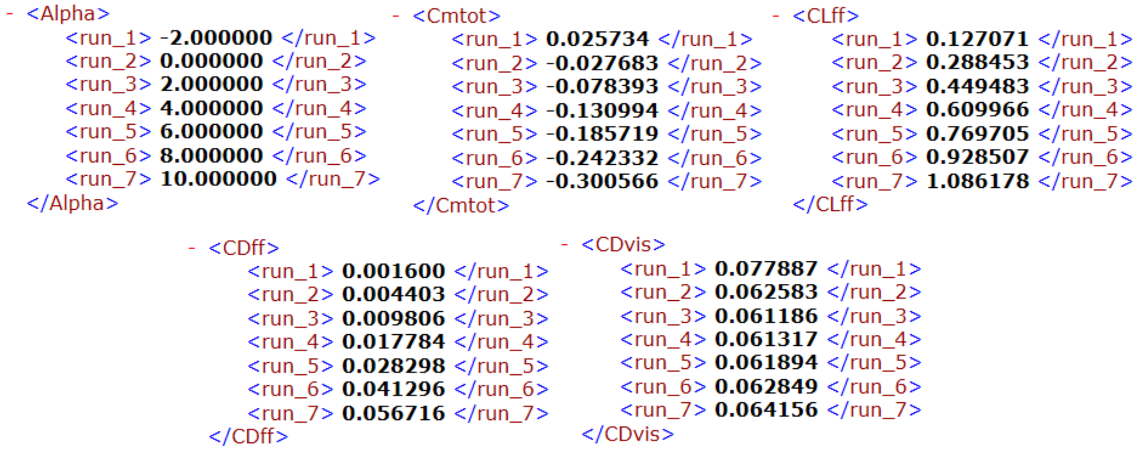
4) Calculate Cdvis at α trim

$$C_{Dvis} = \frac{\Delta C_{Dvis}}{\Delta\alpha} * \alpha_{trim} + Intercept$$

5) Calculate CLZ which will be Cle0

$$C_L\delta_{e0} = C_{L_Z} = C_{Lff} * \cos(\alpha_{trim}) + \left[C_{Dff} + C_{Dvis}\right] * \sin(\alpha_{trim})$$

3.5.1.Nexstar Example Calculation



```
- <Alpha>
    <run_1> -2.000000 </run_1>
    <run_2> 0.000000 </run_2>
    <run_3> 2.000000 </run_3>
    <run_4> 4.000000 </run_4>
    <run_5> 6.000000 </run_5>
    <run_6> 8.000000 </run_6>
    <run_7> 10.000000 </run_7>
  </Alpha>
- <Cmtot>
    <run_1> 0.025734 </run_1>
    <run_2> -0.027683 </run_2>
    <run_3> -0.078393 </run_3>
    <run_4> -0.130994 </run_4>
    <run_5> -0.185719 </run_5>
    <run_6> -0.242332 </run_6>
    <run_7> -0.300566 </run_7>
  </Cmtot>
- <CLff>
    <run_1> 0.127071 </run_1>
    <run_2> 0.288453 </run_2>
    <run_3> 0.449483 </run_3>
    <run_4> 0.609966 </run_4>
    <run_5> 0.769705 </run_5>
    <run_6> 0.928507 </run_6>
    <run_7> 1.086178 </run_7>
  </CLff>
- <CDff>
    <run_1> 0.001600 </run_1>
    <run_2> 0.004403 </run_2>
    <run_3> 0.009806 </run_3>
    <run_4> 0.017784 </run_4>
    <run_5> 0.028298 </run_5>
    <run_6> 0.041296 </run_6>
    <run_7> 0.056716 </run_7>
  </CDff>
- <CDvis>
    <run_1> 0.077887 </run_1>
    <run_2> 0.062583 </run_2>
    <run_3> 0.061186 </run_3>
    <run_4> 0.061317 </run_4>
    <run_5> 0.061894 </run_5>
    <run_6> 0.062849 </run_6>
    <run_7> 0.064156 </run_7>
  </CDvis>
```

Figure 46 Nexstar Cle0 Calculation

Figure 46 depicts coefficients from Nexstar's alpha file that are relevant to calculating C_L at zero elevator.

1) The alpha sweep produced two alpha runs that encompass Cmtot = 0, runs 1 and 2. Additionally run 2 was calculated on alpha equal to zero so the intercept of the line equation for all of the coefficients were simply C_ (run 2).

$$\frac{\Delta C_{mtot}}{\Delta\alpha} = \frac{-0.027683 - 0.025734}{0 - (-2)} = -0.026709 \qquad Intercept = \ C_{mtot}(run\ 2)$$

$$= -0.027683$$

$$\alpha_{trim} = \frac{-Intercept}{\frac{\Delta C_{mtot}}{\Delta\alpha}} = \frac{0.027726}{-0.026683} = -1.036487°$$

2)

$$\frac{\Delta C_{Lff}}{\Delta\alpha} = \frac{0.288453 - 0.127071}{0 - (-2)} = 0.080691 \qquad Intercept = \ C_{Lff}(run\ 2) = 0.288453$$

$$C_{Lff} = \frac{\Delta C_{Lff}}{\Delta\alpha} * \alpha_{trim} + Intercept = 0.080691 * -1.036487 + 0.288453$$

$$= 0.204818$$

3)

$$\frac{\Delta C_{Dff}}{\Delta\alpha} = \frac{0.004403 - 0.001600}{0 - (-2)} = 0.001402 \qquad Intercept = \ C_{Dff}(run\ 2) = 0.004403$$

$$C_{Dff} = \frac{\Delta C_{Dff}}{\Delta\alpha} * \alpha_{trim} + Intercept = 0.001402 * -1.036487 + 0.004403$$

$$= 0.0029504$$

4)

$$\frac{\Delta C_{Dvis}}{\Delta \alpha} = \frac{0.062583 - 0.077887}{0 - (-2)} = -0.00765 \qquad Intercept = \ C_{Dvis}(run\ 2) = 0.062583$$

$$C_{Dvis} = \frac{\Delta C_{Dvis}}{\Delta \alpha} * \alpha_{trim} + Intercept = -0.00765 * -1.036487 + 0.062583$$

$$= 0.0705142$$

5)

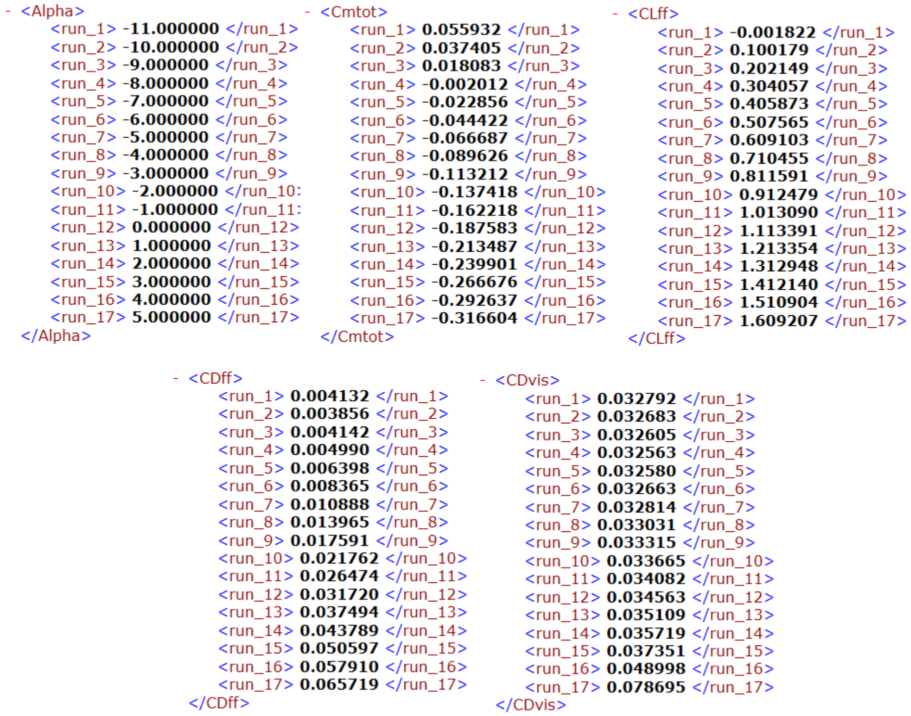
$$C_L \delta_{e0} = C_{L_Z} = C_{Lff} * \cos(\alpha_{trim}) - \left[C_{Dff} + C_{Dvis}\right] * \sin(\alpha_{trim})$$

$$C_L \delta_{e0} = 0.204818 * \cos(-1.036487) + [0.0029504 + 0.0705142] * \sin(-1.036487)$$

$$= 0.203455$$

The manual calculation produced a C_L at zero elevator value of 0.203455 which was an exact match of the simulator calculated value of 0.203455.

3.5.2. Noctua B1 Example Calculation



```
- <Alpha>
    <run_1> -11.000000 </run_1>
    <run_2> -10.000000 </run_2>
    <run_3> -9.000000 </run_3>
    <run_4> -8.000000 </run_4>
    <run_5> -7.000000 </run_5>
    <run_6> -6.000000 </run_6>
    <run_7> -5.000000 </run_7>
    <run_8> -4.000000 </run_8>
    <run_9> -3.000000 </run_9>
    <run_10> -2.000000 </run_10
    <run_11> -1.000000 </run_11
    <run_12> 0.000000 </run_12>
    <run_13> 1.000000 </run_13>
    <run_14> 2.000000 </run_14>
    <run_15> 3.000000 </run_15>
    <run_16> 4.000000 </run_16>
    <run_17> 5.000000 </run_17>
  </Alpha>

- <Cmtot>
    <run_1> 0.055932 </run_1>
    <run_2> 0.037405 </run_2>
    <run_3> 0.018083 </run_3>
    <run_4> -0.002012 </run_4>
    <run_5> -0.022856 </run_5>
    <run_6> -0.044422 </run_6>
    <run_7> -0.066687 </run_7>
    <run_8> -0.089626 </run_8>
    <run_9> -0.113212 </run_9>
    <run_10> -0.137418 </run_10>
    <run_11> -0.162218 </run_11>
    <run_12> -0.187583 </run_12>
    <run_13> -0.213487 </run_13>
    <run_14> -0.239901 </run_14>
    <run_15> -0.266676 </run_15>
    <run_16> -0.292637 </run_16>
    <run_17> -0.316604 </run_17>
  </Cmtot>

- <CLff>
    <run_1> -0.001822 </run_1>
    <run_2> 0.100179 </run_2>
    <run_3> 0.202149 </run_3>
    <run_4> 0.304057 </run_4>
    <run_5> 0.405873 </run_5>
    <run_6> 0.507565 </run_6>
    <run_7> 0.609103 </run_7>
    <run_8> 0.710455 </run_8>
    <run_9> 0.811591 </run_9>
    <run_10> 0.912479 </run_10>
    <run_11> 1.013090 </run_11>
    <run_12> 1.113391 </run_12>
    <run_13> 1.213354 </run_13>
    <run_14> 1.312948 </run_14>
    <run_15> 1.412140 </run_15>
    <run_16> 1.510904 </run_16>
    <run_17> 1.609207 </run_17>
  </CLff>

- <CDff>
    <run_1> 0.004132 </run_1>
    <run_2> 0.003856 </run_2>
    <run_3> 0.004142 </run_3>
    <run_4> 0.004990 </run_4>
    <run_5> 0.006398 </run_5>
    <run_6> 0.008365 </run_6>
    <run_7> 0.010888 </run_7>
    <run_8> 0.013965 </run_8>
    <run_9> 0.017591 </run_9>
    <run_10> 0.021762 </run_10>
    <run_11> 0.026474 </run_11>
    <run_12> 0.031720 </run_12>
    <run_13> 0.037494 </run_13>
    <run_14> 0.043789 </run_14>
    <run_15> 0.050597 </run_15>
    <run_16> 0.057910 </run_16>
    <run_17> 0.065719 </run_17>
  </CDff>

- <CDvis>
    <run_1> 0.032792 </run_1>
    <run_2> 0.032683 </run_2>
    <run_3> 0.032605 </run_3>
    <run_4> 0.032563 </run_4>
    <run_5> 0.032580 </run_5>
    <run_6> 0.032663 </run_6>
    <run_7> 0.032814 </run_7>
    <run_8> 0.033031 </run_8>
    <run_9> 0.033315 </run_9>
    <run_10> 0.033665 </run_10>
    <run_11> 0.034082 </run_11>
    <run_12> 0.034563 </run_12>
    <run_13> 0.035109 </run_13>
    <run_14> 0.035719 </run_14>
    <run_15> 0.037351 </run_15>
    <run_16> 0.048998 </run_16>
    <run_17> 0.078695 </run_17>
  </CDvis>
```

Figure 47 Noctua B1 Cle0 Calculation

Figure 47 depicts coefficients from Noctua B1's alpha file that are relevant to calculating C_L at zero elevator.

1) The alpha sweep produced two alpha runs that encompass Cmtot = 0, runs 3 and 4.

$$\frac{\Delta C_{mtot}}{\Delta \alpha} = \frac{-0.00201 - 0.018083}{-9-(-8)} = -0.020095 \qquad Intercept = \ -0.162772$$

$$\alpha_{trim} = \frac{-Intercept}{\frac{\Delta C_{mtot}}{\Delta \alpha}} = \frac{-(-0.162772)}{-0.020095} = -8.100124°$$

2)

$$\frac{\Delta C_{Lff}}{\Delta \alpha} = \frac{0.304057 - 0.202149}{-9-(-8)} = 0.101908 \qquad Intercept = \ C_{Lff}(run\ 2) = 1.119321$$

$$C_{Lff} = \frac{\Delta C_{Lff}}{\Delta\alpha} * \alpha_{trim} + Intercept = 0.101908 * -8.100124 + 1.119321$$

$$= 0.29385352$$

3)

$$\frac{\Delta C_{Dff}}{\Delta\alpha} = \frac{0.004990 - 0.004142}{-9-(-8)} = 0.000848 \qquad Intercept = \ C_{Dff}(run\ 2) = 0.011774$$

$$C_{Dff} = \frac{\Delta C_{Dff}}{\Delta\alpha} * \alpha_{trim} + Intercept = 0.000848 * -8.100124 + 0.011774$$

$$= 0.004905$$

4)

$$\frac{\Delta C_{Dvis}}{\Delta\alpha} = \frac{0.032563 - 0.032605}{-9-(-8)} = -0.000042 \qquad Intercept = \ C_{Dvis}(run\ 2) = 0.032227$$

$$C_{Dvis} = \frac{\Delta C_{Dvis}}{\Delta\alpha} * \alpha_{trim} + Intercept = -0.000042 * -8.100124 + 0.032227$$

$$= 0.032567$$

5)

$$C_L\delta_{e0} = C_{L_Z} = C_{Lff} * \cos(\alpha_{trim}) - \left[C_{Dff} + C_{Dvis}\right] * \sin(\alpha_{trim})$$

$$C_L\delta_{e0} = 0.29385352 * \cos(-8.100124) + [0.004905 + 0.032567] * \sin(-8.100124)$$

$$= 0.285642$$

The manual calculation produced a C_L at zero elevator value of 0.285642 which was an exact match of the simulator calculated value of 0.28542.

3.6. Vertical Tail Arm

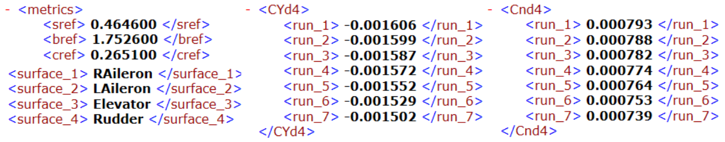
The simulator calculates the vertical tail arm using the rudder control surface derivatives, and wing span from the alpha file.

$$-\frac{C_{n_{\delta_r}}(run\ 1)}{C_{Y_{\delta_r}}(run\ 1)} * b = l_v$$

Equation 3 Vertical Tail Arm

It was found that the simulator uses Equation 3 to calculate the vertical tail arm. $C_{n\delta r}$ (run 1) is the summation of the change in yawing moment per change in control surface deflection of each control surface that is interpreted as a rudder by the simulator in the first alpha run. Similarly $C_{Y\delta r}$ (run 1) is the summation of the change in side force over change in control surface deflection of each control surface that is interpreted as a rudder by the simulator in the first alpha run.

3.6.1. Nexstar Example Calculation



```
- <metrics>
      <sref> 0.464600 </sref>
      <bref> 1.752600 </bref>
      <cref> 0.265100 </cref>
  <surface_1> RAileron </surface_1>
  <surface_2> LAileron </surface_2>
  <surface_3> Elevator </surface_3>
  <surface_4> Rudder </surface_4>
```

```
- <CYd4>
      <run_1> -0.001606 </run_1>
      <run_2> -0.001599 </run_2>
      <run_3> -0.001587 </run_3>
      <run_4> -0.001572 </run_4>
      <run_5> -0.001552 </run_5>
      <run_6> -0.001529 </run_6>
      <run_7> -0.001502 </run_7>
  </CYd4>
```

```
- <Cnd4>
      <run_1> 0.000793 </run_1>
      <run_2> 0.000788 </run_2>
      <run_3> 0.000782 </run_3>
      <run_4> 0.000774 </run_4>
      <run_5> 0.000764 </run_5>
      <run_6> 0.000753 </run_6>
      <run_7> 0.000739 </run_7>
  </Cnd4>
```

Figure 48 Nexstar Vertical Tail Arm Calculation

Figure 48 depicts the alpha file coefficients of Nexstar that were relevant to the vertical tail arm calculation. The rudder surface was surface 5, and the wingspan was 1.7526 m.

$$Vertical\ Tail\ Arm = -\frac{C_{n_{\delta_r}}(run\ 1)}{C_{Y_{\delta_r}}(run\ 1)} * b$$

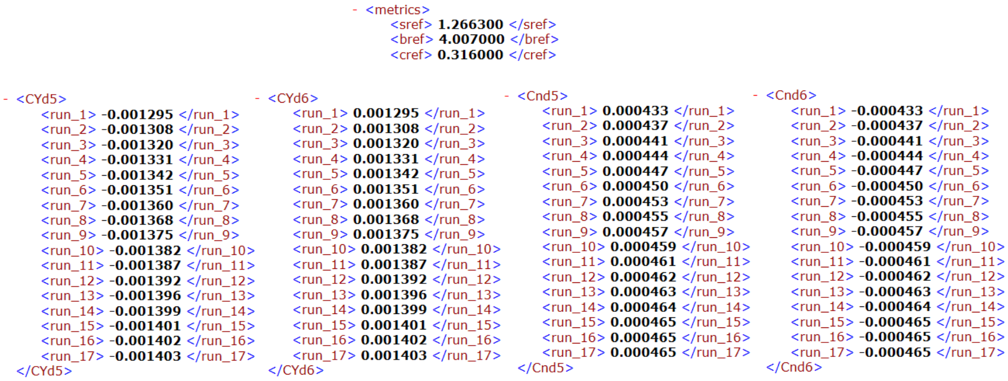
$$Vertical\ Tail\ Arm = -\frac{Cnd4(run\ 1)}{CYd4(run\ 1)} * bref$$

$$= -\frac{0.000793}{-0.001606} * 1.7526$$

$$= 0.865387\ m$$

The manual calculation produced a vertical tail arm of 0.865387 m. The manual calculation was an exact match to the simulator calculation of 0.865387 m.

3.6.2.Noctua B1 Example Calculation



```
- <metrics>
    <sref> 1.266300 </sref>
    <bref> 4.007000 </bref>
    <cref> 0.316000 </cref>

- <CYd5>
    <run_1> -0.001295 </run_1>
    <run_2> -0.001308 </run_2>
    <run_3> -0.001320 </run_3>
    <run_4> -0.001331 </run_4>
    <run_5> -0.001342 </run_5>
    <run_6> -0.001351 </run_6>
    <run_7> -0.001360 </run_7>
    <run_8> -0.001368 </run_8>
    <run_9> -0.001375 </run_9>
    <run_10> -0.001382 </run_10>
    <run_11> -0.001387 </run_11>
    <run_12> -0.001392 </run_12>
    <run_13> -0.001396 </run_13>
    <run_14> -0.001399 </run_14>
    <run_15> -0.001401 </run_15>
    <run_16> -0.001402 </run_16>
    <run_17> -0.001403 </run_17>
  </CYd5>

- <CYd6>
    <run_1> 0.001295 </run_1>
    <run_2> 0.001308 </run_2>
    <run_3> 0.001320 </run_3>
    <run_4> 0.001331 </run_4>
    <run_5> 0.001342 </run_5>
    <run_6> 0.001351 </run_6>
    <run_7> 0.001360 </run_7>
    <run_8> 0.001368 </run_8>
    <run_9> 0.001375 </run_9>
    <run_10> 0.001382 </run_10>
    <run_11> 0.001387 </run_11>
    <run_12> 0.001392 </run_12>
    <run_13> 0.001396 </run_13>
    <run_14> 0.001399 </run_14>
    <run_15> 0.001401 </run_15>
    <run_16> 0.001402 </run_16>
    <run_17> 0.001403 </run_17>
  </CYd6>

- <Cnd5>
    <run_1> 0.000433 </run_1>
    <run_2> 0.000437 </run_2>
    <run_3> 0.000441 </run_3>
    <run_4> 0.000444 </run_4>
    <run_5> 0.000447 </run_5>
    <run_6> 0.000450 </run_6>
    <run_7> 0.000453 </run_7>
    <run_8> 0.000455 </run_8>
    <run_9> 0.000457 </run_9>
    <run_10> 0.000459 </run_10>
    <run_11> 0.000461 </run_11>
    <run_12> 0.000462 </run_12>
    <run_13> 0.000463 </run_13>
    <run_14> 0.000464 </run_14>
    <run_15> 0.000465 </run_15>
    <run_16> 0.000465 </run_16>
    <run_17> 0.000465 </run_17>
  </Cnd5>

- <Cnd6>
    <run_1> -0.000433 </run_1>
    <run_2> -0.000437 </run_2>
    <run_3> -0.000441 </run_3>
    <run_4> -0.000444 </run_4>
    <run_5> -0.000447 </run_5>
    <run_6> -0.000450 </run_6>
    <run_7> -0.000453 </run_7>
    <run_8> -0.000455 </run_8>
    <run_9> -0.000457 </run_9>
    <run_10> -0.000459 </run_10>
    <run_11> -0.000461 </run_11>
    <run_12> -0.000462 </run_12>
    <run_13> -0.000463 </run_13>
    <run_14> -0.000464 </run_14>
    <run_15> -0.000465 </run_15>
    <run_16> -0.000465 </run_16>
    <run_17> -0.000465 </run_17>
  </Cnd6>
```

Figure 49 Noctua B1 Vertical Tail Arm Calculation

Figure 49 depicts the alpha file coefficients of Noctua B1 that were relevant to the vertical tail arm calculation. The rudder surfaces were surfaces 5 and 6.

$$LRuddervator = Elevator + \ -Rudder$$

$$RRuddervator = Elevator + Rudder$$

The two equations above define ruddervator deflection commands. The vertical tail arm calculation is only concerned with rudder deflections; therefore, the rudder deflections are exactly the same magnitude as the ruddervator deflections, i.e. a rudder deflection command of 5 degrees would deflect both ruddervator control surfaces 5 degrees. The ruddervator deflection sign convention follows that of elevators; therefore, a positive rudder deflection will result in a negative left ruddervator deflection.

$$Vertical\,Tail\,Arm = -\frac{C_{n_{\delta_r}}(run\ 1)}{C_{Y_{\delta_r}}(run\ 1)} * b$$

$$LRudder = \ -LRuddervator = -d6 \qquad RRudder = RRuddervator = d5$$

$$C_{n_{\delta_r}}(run\ 1) = -Cnd6(run\ 1) + Cnd5(run\ 1) \qquad C_{Y_{\delta_r}}(run\ 1)$$

$$= -CYd6(run\ 1) + CYd5(run\ 1)$$

$$Vertical\,Tail\,Arm = -\ \frac{-Cnd6(run\ 1) + Cnd5(run\ 1)}{-CYd6(run\ 1) + CYd5(run\ 1)}\ * bref$$

$$= \ -\frac{[-(-0.000433) + 0.000433]}{[-0.00392 + (-0.00392)]} * 4.007$$

$$= 1.3398\,m$$

The manual calculation produced a vertical tail arm of 1.3398 m. The simulator estimate was 0 because the simulator does not adjust the rudder deflection sign convention for ruddervators when it calculates the vertical tail arm. A manual measurement was made on Noctua B1 with a result of 1.28 m.

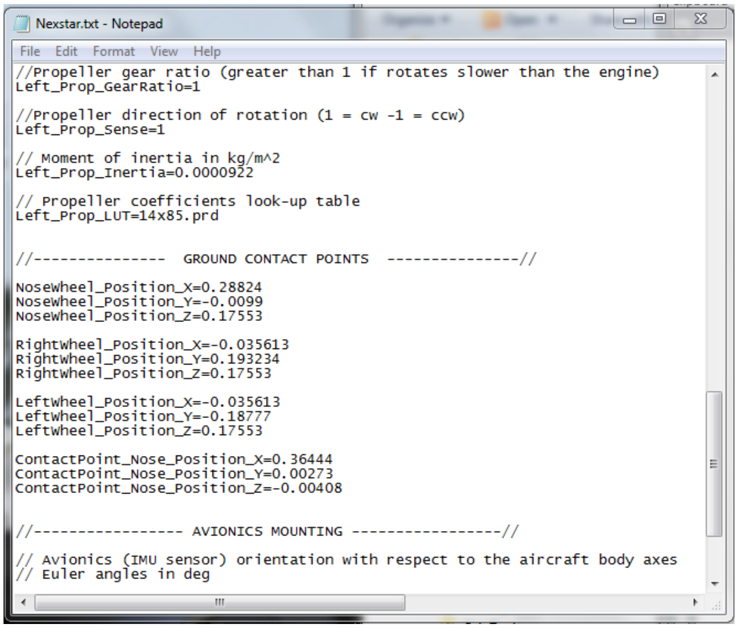
3.7. Steering Arm

The simulator calculates the steering arm directly from the simulator file. The simulator takes the x – location of the nose wheel position and subtracts the average of the x-locations of the left and right wheel positions. Equation 4 below uses the simulator file terminology for the wheel locations.

$$Steering\ Arm = NoseWheel\_Position\_X - average[Left, Right\ Wheel\_Position\_X]$$

Equation 4 Steering Arm

3.7.1.Nexstar Example Calculation



```
//Propeller gear ratio (greater than 1 if rotates slower than the engine)
Left_Prop_GearRatio=1

//Propeller direction of rotation (1 = cw -1 = ccw)
Left_Prop_Sense=1

// Moment of inertia in kg/m^2
Left_Prop_Inertia=0.0000922

// Propeller coefficients look-up table
Left_Prop_LUT=14x85.prd


//---------------  GROUND CONTACT POINTS  ---------------//

NoseWheel_Position_X=0.28824
NoseWheel_Position_Y=-0.0099
NoseWheel_Position_Z=0.17553

RightWheel_Position_X=-0.035613
RightWheel_Position_Y=0.193234
RightWheel_Position_Z=0.17553

LeftWheel_Position_X=-0.035613
LeftWheel_Position_Y=-0.18777
LeftWheel_Position_Z=0.17553

ContactPoint_Nose_Position_X=0.36444
ContactPoint_Nose_Position_Y=0.00273
ContactPoint_Nose_Position_Z=-0.00408


//----------------- AVIONICS MOUNTING -----------------//

// Avionics (IMU sensor) orientation with respect to the aircraft body axes
// Euler angles in deg
```

Figure 50 Nexstar Steering Arm Calculation

Figure 50 depicts the ground contact points of Nexstar from Nexstar’s simulator file.

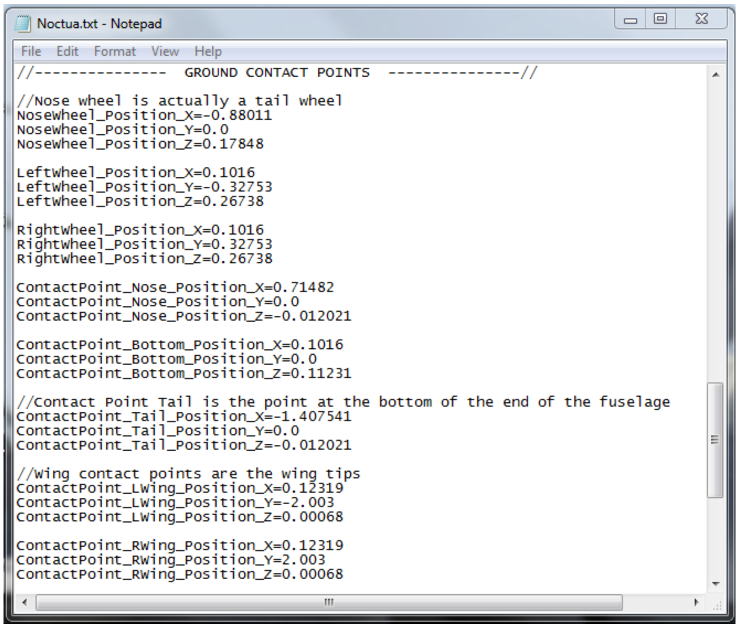
$$Steering\ Arm = NoseWheel\_Position\_X - average[Left, Right\ Wheel\_Position\_X]$$

$$Steering\ Arm = \ 0.28824 - 0.035613 + -0.0356132$$

$$= \ 0.323853\ m$$

The manual calculation produced a steering arm value of 0.323853 meters which was an exact match with the simulator calculation of 0.323853 meters.

3.7.2. Noctua B1 Example Calculation



```
//---------------  GROUND CONTACT POINTS  ---------------//

//Nose wheel is actually a tail wheel
NoseWheel_Position_X=-0.88011
NoseWheel_Position_Y=0.0
NoseWheel_Position_Z=0.17848

LeftWheel_Position_X=0.1016
LeftWheel_Position_Y=-0.32753
LeftWheel_Position_Z=0.26738

RightWheel_Position_X=0.1016
RightWheel_Position_Y=0.32753
RightWheel_Position_Z=0.26738

ContactPoint_Nose_Position_X=0.71482
ContactPoint_Nose_Position_Y=0.0
ContactPoint_Nose_Position_Z=-0.012021

ContactPoint_Bottom_Position_X=0.1016
ContactPoint_Bottom_Position_Y=0.0
ContactPoint_Bottom_Position_Z=0.11231

//Contact Point Tail is the point at the bottom of the end of the fuselage
ContactPoint_Tail_Position_X=-1.407541
ContactPoint_Tail_Position_Y=0.0
ContactPoint_Tail_Position_Z=-0.012021

//Wing contact points are the wing tips
ContactPoint_LWing_Position_X=0.12319
ContactPoint_LWing_Position_Y=-2.003
ContactPoint_LWing_Position_Z=0.00068

ContactPoint_RWing_Position_X=0.12319
ContactPoint_RWing_Position_Y=2.003
ContactPoint_RWing_Position_Z=0.00068
```

Figure 51 Noctua B1 Steering Arm Calculation

Figure 51 depicts the ground contact points of Noctua B1 from Noctua's simulator file. The nosewheel position is negative because Noctua had a tailwheel, not a nosewheel.

$$Steering\ Arm = NoseWheel\_Position\_X - average[Left, Right\ Wheel\_Position\_X]$$

$$Steering\ Arm = -0.88011 - \frac{0.1016 + 0.1016}{2}$$

$$= -0.981710\ m$$

The manual calculation produced a steering arm value of -0.981710 meters which was an exact match with the simulator calculation of -0.981710 meters.

3.8. Aileron Effectiveness

The simulator calculates aileron effectiveness using the aileron control surface derivatives, and the coefficient of change in roll moment over change in roll rate, from the alpha file. Additionally the simulator will scale the aileron effectiveness according to the Cld scaling term in the alpha file.

$$Aileron\ Effectiveness = \frac{C_{l_{\delta_a}}(\alpha = 0) * \frac{180}{\pi}}{C_{l_p}} * C_{l_\delta}Scaler$$

Equation 5 Aileron Effectiveness

It was found that the simulator uses Equation 5 to calculate the aileron effectiveness. $C_{l\delta a}$ (α = 0) is the summation of the change in rolling moment per change in control surface deflection of each control surface that is interpreted as an aileron by the simulator at alpha equal to zero. C_{lp} (α = 0) is the change in roll moment per change in roll rate at alpha equal to zero. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the value of $C_{l\delta a}$ at alpha equal to zero. $C_{l\delta}$Scaler is a scaling term that can be applied in the simulator file to adjust the model to reflect doublet maneuver test results.

$$LAileron = \ Aileron$$

$$RAileron = \ -Aileron$$

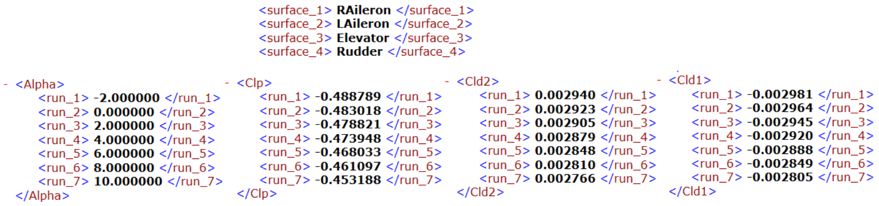
The sign conventions of left and right aileron control surface deflections are inverted. A positive left aileron deflection produces a positive roll moment. A negative right aileron deflection produces a positive roll moment. $C_{l\delta}$ of right aileron control surfaces is negative and $C_{l\delta}$ of left aileron control surfaces is positive. Aileron effectiveness is defined by Cloud Cap to be positive; therefore, the $C_{l\delta}$ values of left aileron control surfaces must be multiplied by -1 so that the aileron effectiveness solution is positive.

Additionally there is another roll control vehicle parameter that the simulator calculates; however, it is not applicable to generation two. Aileron power is a gain for generation three controllers. The calculation is included just for informational purposes only. The simulator calculates aileron power using Equation 6 below.

Equation 6 Aileron Power

$$Aileron\ Power = C_{l_{\delta_a}}(\alpha = 0) * \frac{180}{\pi} * C_{l_\delta}Scaler$$

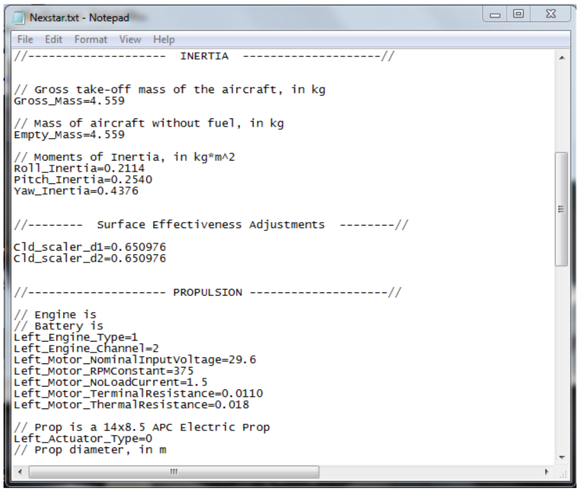
3.8.1. Nexstar Example Calculation



```
                                    <surface_1> RAileron </surface_1>
                                    <surface_2> LAileron </surface_2>
                                    <surface_3> Elevator </surface_3>
                                    <surface_4> Rudder </surface_4>

- <Alpha>                        - <Clp>                           - <Cld2>                          - <Cld1>
     <run_1> -2.000000 </run_1>       <run_1> -0.488789 </run_1>        <run_1> 0.002940 </run_1>         <run_1> -0.002981 </run_1>
     <run_2> 0.000000 </run_2>        <run_2> -0.483018 </run_2>        <run_2> 0.002923 </run_2>         <run_2> -0.002964 </run_2>
     <run_3> 2.000000 </run_3>        <run_3> -0.478821 </run_3>        <run_3> 0.002905 </run_3>         <run_3> -0.002945 </run_3>
     <run_4> 4.000000 </run_4>        <run_4> -0.473948 </run_4>        <run_4> 0.002879 </run_4>         <run_4> -0.002920 </run_4>
     <run_5> 6.000000 </run_5>        <run_5> -0.468033 </run_5>        <run_5> 0.002848 </run_5>         <run_5> -0.002888 </run_5>
     <run_6> 8.000000 </run_6>        <run_6> -0.461097 </run_6>        <run_6> 0.002810 </run_6>         <run_6> -0.002849 </run_6>
     <run_7> 10.000000 </run_7>       <run_7> -0.453188 </run_7>        <run_7> 0.002766 </run_7>         <run_7> -0.002805 </run_7>
  </Alpha>                         </Clp>                            </Cld2>                           </Cld1>
```

Figure 52 Nexstar Aileron Effectiveness Calculation

Figure 52 depicts the alpha file coefficients of Nexstar that were relevant to the aileron effectiveness calculation. The aileron surfaces were surfaces 1 and 2. Alpha equal to zero occurred at run 2.



```
//--------------------  INERTIA  --------------------//


// Gross take-off mass of the aircraft, in kg
Gross_Mass=4.559

// Mass of aircraft without fuel, in kg
Empty_Mass=4.559

// Moments of Inertia, in kg*m^2
Roll_Inertia=0.2114
Pitch_Inertia=0.2540
Yaw_Inertia=0.4376


//--------  Surface Effectiveness Adjustments  --------//

Cld_scaler_d1=0.650976
Cld_scaler_d2=0.650976


//------------------- PROPULSION -------------------//

// Engine is
// Battery is
Left_Engine_Type=1
Left_Engine_Channel=2
Left_Motor_NominalInputVoltage=29.6
Left_Motor_RPMConstant=375
Left_Motor_NoLoadCurrent=1.5
Left_Motor_TerminalResistance=0.0110
Left_Motor_ThermalResistance=0.018

// Prop is a 14x8.5 APC Electric Prop
Left_Actuator_Type=0
// Prop diameter, in m
```

Figure 53 Nexstar Cld Scaler

Figure 53 shows a snapshot of the Nexstar simulator file. The simulator file had a Cld scaling term of 0.650976 that had been determined from the results of the aileron doublet maneuvers.

$$Aileron\ Effectiveness = \frac{C_{l_{\delta_a}}(\alpha = 0) * \frac{180}{\pi}}{C_{l_p}(\alpha = 0)} * C_{l_\delta}Scaler$$

$$LAileron = \ -d2 \quad RAileron = d1$$

$$C_{l_{\delta_a}}(\alpha = 0) = -Cld2(\alpha = 0) + Cld1(\alpha = 0)$$

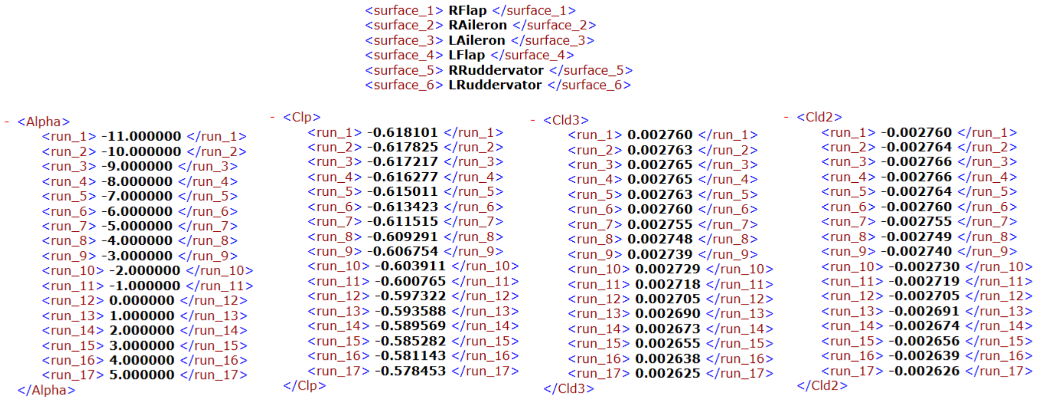
$$Aileron\ Effectiveness = \frac{-Cld2(run\ 2) + Cld3(run\ 2)}{Clp(run\ 2)} * \frac{180}{\pi} * Cld\_scaler$$

$$= \frac{-(0.002923) + -0.002964}{-0.483018} * \frac{180}{\pi} * 0.650976$$

$$= 0.454586$$

The manual calculation produced an aileron effectiveness of 0.454586 /rad which was an exact match with the simulator calculated value of 0.454586 /rad.

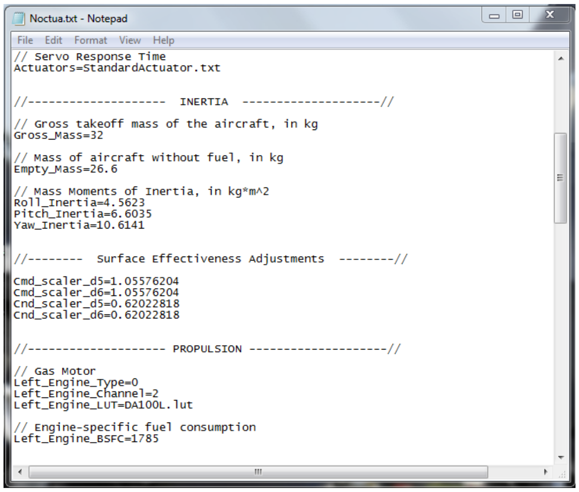
3.8.2. Noctua B1 Example Calculation



```
<surface_1> RFlap </surface_1>
<surface_2> RAileron </surface_2>
<surface_3> LAileron </surface_3>
<surface_4> LFlap </surface_4>
<surface_5> RRuddervator </surface_5>
<surface_6> LRuddervator </surface_6>

- <Alpha>
    <run_1> -11.000000 </run_1>
    <run_2> -10.000000 </run_2>
    <run_3> -9.000000 </run_3>
    <run_4> -8.000000 </run_4>
    <run_5> -7.000000 </run_5>
    <run_6> -6.000000 </run_6>
    <run_7> -5.000000 </run_7>
    <run_8> -4.000000 </run_8>
    <run_9> -3.000000 </run_9>
    <run_10> -2.000000 </run_10>
    <run_11> -1.000000 </run_11>
    <run_12> 0.000000 </run_12>
    <run_13> 1.000000 </run_13>
    <run_14> 2.000000 </run_14>
    <run_15> 3.000000 </run_15>
    <run_16> 4.000000 </run_16>
    <run_17> 5.000000 </run_17>
  </Alpha>

- <Clp>
    <run_1> -0.618101 </run_1>
    <run_2> -0.617825 </run_2>
    <run_3> -0.617217 </run_3>
    <run_4> -0.616277 </run_4>
    <run_5> -0.615011 </run_5>
    <run_6> -0.613423 </run_6>
    <run_7> -0.611515 </run_7>
    <run_8> -0.609291 </run_8>
    <run_9> -0.606754 </run_9>
    <run_10> -0.603911 </run_10>
    <run_11> -0.600765 </run_11>
    <run_12> -0.597322 </run_12>
    <run_13> -0.593588 </run_13>
    <run_14> -0.589569 </run_14>
    <run_15> -0.585282 </run_15>
    <run_16> -0.581143 </run_16>
    <run_17> -0.578453 </run_17>
  </Clp>

- <Cld3>
    <run_1> 0.002760 </run_1>
    <run_2> 0.002763 </run_2>
    <run_3> 0.002765 </run_3>
    <run_4> 0.002765 </run_4>
    <run_5> 0.002763 </run_5>
    <run_6> 0.002760 </run_6>
    <run_7> 0.002755 </run_7>
    <run_8> 0.002748 </run_8>
    <run_9> 0.002739 </run_9>
    <run_10> 0.002729 </run_10>
    <run_11> 0.002718 </run_11>
    <run_12> 0.002705 </run_12>
    <run_13> 0.002690 </run_13>
    <run_14> 0.002673 </run_14>
    <run_15> 0.002655 </run_15>
    <run_16> 0.002638 </run_16>
    <run_17> 0.002625 </run_17>
  </Cld3>

- <Cld2>
    <run_1> -0.002760 </run_1>
    <run_2> -0.002764 </run_2>
    <run_3> -0.002766 </run_3>
    <run_4> -0.002766 </run_4>
    <run_5> -0.002764 </run_5>
    <run_6> -0.002760 </run_6>
    <run_7> -0.002755 </run_7>
    <run_8> -0.002749 </run_8>
    <run_9> -0.002740 </run_9>
    <run_10> -0.002730 </run_10>
    <run_11> -0.002719 </run_11>
    <run_12> -0.002705 </run_12>
    <run_13> -0.002691 </run_13>
    <run_14> -0.002674 </run_14>
    <run_15> -0.002656 </run_15>
    <run_16> -0.002639 </run_16>
    <run_17> -0.002626 </run_17>
  </Cld2>
```

Figure 54 Noctua B1 Aileron Effectiveness Calculation

Figure 54 depicts the alpha file coefficients of Noctua B1 that were relevant to the aileron effectiveness calculation. The aileron surfaces were surfaces 2 and 3. Alpha equal to zero occurred at run 12.



```
// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
Empty_Mass=26.6

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=4.5623
Pitch_Inertia=6.6035
Yaw_Inertia=10.6141


//--------  Surface Effectiveness Adjustments  --------//

Cmd_scaler_d5=1.05576204
Cmd_scaler_d6=1.05576204
Cnd_scaler_d5=0.62022818
Cnd_scaler_d6=0.62022818


//-------------------- PROPULSION --------------------//

// Gas Motor
Left_Engine_Type=0
Left_Engine_Channel=2
Left_Engine_LUT=DA100L.lut

// Engine-specific fuel consumption
Left_Engine_BSFC=1785
```

Figure 55 Noctua B1 Cld Scaler

Figure 55 shows a snapshot of the Noctua B1 simulator file. The simulator file did not have a Cld scaling term because the aileron doublet maneuver results were nearly identical to the original model estimate, so aileron effectiveness did not need to be adjusted in the aircraft model.

$$Aileron\ Effectiveness = \frac{C_{l_{\delta_a}}(\alpha = 0) * \frac{180}{\pi}}{C_{l_p}(\alpha = 0)} * C_{l_\delta}Scaler$$

$$LAileron = \ -d3 \quad RAileron = d2$$

$$C_{l_{\delta_a}}(\alpha = 0) = -Cld3(\alpha = 0) + Cld2(\alpha = 0)$$

$$Aileron\ Effectiveness = \frac{-Cld3(run\ 12) + Cld2(run\ 12)}{Clp(run\ 12)} * \frac{180}{\pi} * Cld\_scaler$$

$$= \frac{-0.002705 + (-0.002705)}{-0.59732} * \frac{180}{\pi} * 1$$

$$= 0.518933$$

The manual calculation produced an aileron effectiveness of 0.518933 /rad which was an exact match with the simulator calculated value of 0.518933 /rad.

3.9. Rudder Power

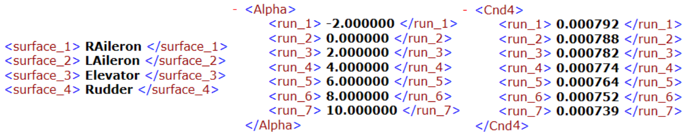
The simulator calculates rudder power using the rudder control surface derivatives from the alpha file. Additionally the simulator will scale the rudder power according to the Cnd scalar term in the alpha file.

$$Rudder\ Power = C_{n_{\delta_r}}(\alpha = 0) * \frac{180}{\pi} * C_{n_\delta}Scaler$$

Equation 7 Rudder Power

It was found that the simulator uses Equation 7 to calculate the rudder power. $C_{n\delta r}$ (α = 0) is the summation of the change in yawing moment per change in control surface deflection of each control surface that is interpreted as a rudder by the simulator at alpha equal to zero. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the value of $C_{n\delta r}$ at alpha equal to zero. $C_{n\delta}$Scaler is a scaling term that can be applied in the simulator file to adjust the model to reflect doublet maneuver test results.

3.9.1. Nexstar Example Calculation



```
                              - <Alpha>                                    - <Cnd4>
                                    <run_1> -2.000000 </run_1>                   <run_1> 0.000792 </run_1>
                                    <run_2> 0.000000 </run_2>                    <run_2> 0.000788 </run_2>
<surface_1> RAileron </surface_1>   <run_3> 2.000000 </run_3>                    <run_3> 0.000782 </run_3>
<surface_2> LAileron </surface_2>   <run_4> 4.000000 </run_4>                    <run_4> 0.000774 </run_4>
<surface_3> Elevator </surface_3>   <run_5> 6.000000 </run_5>                    <run_5> 0.000764 </run_5>
<surface_4> Rudder </surface_4>     <run_6> 8.000000 </run_6>                    <run_6> 0.000752 </run_6>
                                    <run_7> 10.000000 </run_7>                   <run_7> 0.000739 </run_7>
                                </Alpha>                                     </Cnd4>
```

Figure 56 Nexstar Rudder Power Calculation

Figure 56 depicts the alpha file coefficients of Nexstar that were relevant to the rudder power calculation. The rudder surface was on surface 4. Alpha equal to zero occurred at run 2. Additionally there was not a Cnd scaling term in the Nexstar simulator file.

$$Rudder\ Power = C_{n_{\delta_r}}(\alpha = 0) * \frac{180}{\pi} * C_{n_\delta}Scaler$$

$$C_{n_{\delta_r}}(\alpha = 0) = Cnd5(\alpha = 0)$$

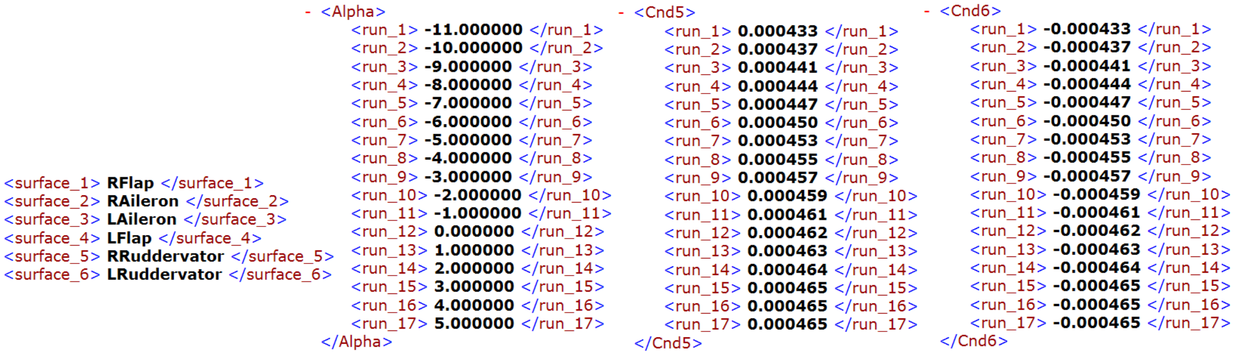
$$Rudder\ Power = Cnd4(run\ 2) * \frac{180}{\pi} * Cnd\_Scaler$$

$$= \ 0.000788 * \frac{180}{\pi} * 1$$

$$= 0.045149\ /rad$$

The manual calculation produced a rudder power of 0.045149 /rad which was an exact match with the simulator calculated value of 0.045149 /rad.

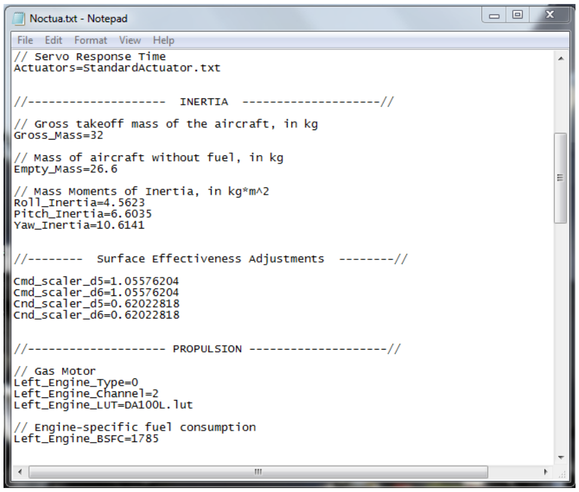
3.9.2. Noctua B1 Example Calculation



```
<surface_1> RFlap </surface_1>
<surface_2> RAileron </surface_2>
<surface_3> LAileron </surface_3>
<surface_4> LFlap </surface_4>
<surface_5> RRuddervator </surface_5>
<surface_6> LRuddervator </surface_6>

- <Alpha>
    <run_1> -11.000000 </run_1>
    <run_2> -10.000000 </run_2>
    <run_3> -9.000000 </run_3>
    <run_4> -8.000000 </run_4>
    <run_5> -7.000000 </run_5>
    <run_6> -6.000000 </run_6>
    <run_7> -5.000000 </run_7>
    <run_8> -4.000000 </run_8>
    <run_9> -3.000000 </run_9>
    <run_10> -2.000000 </run_10>
    <run_11> -1.000000 </run_11>
    <run_12> 0.000000 </run_12>
    <run_13> 1.000000 </run_13>
    <run_14> 2.000000 </run_14>
    <run_15> 3.000000 </run_15>
    <run_16> 4.000000 </run_16>
    <run_17> 5.000000 </run_17>
  </Alpha>

- <Cnd5>
    <run_1> 0.000433 </run_1>
    <run_2> 0.000437 </run_2>
    <run_3> 0.000441 </run_3>
    <run_4> 0.000444 </run_4>
    <run_5> 0.000447 </run_5>
    <run_6> 0.000450 </run_6>
    <run_7> 0.000453 </run_7>
    <run_8> 0.000455 </run_8>
    <run_9> 0.000457 </run_9>
    <run_10> 0.000459 </run_10>
    <run_11> 0.000461 </run_11>
    <run_12> 0.000462 </run_12>
    <run_13> 0.000463 </run_13>
    <run_14> 0.000464 </run_14>
    <run_15> 0.000465 </run_15>
    <run_16> 0.000465 </run_16>
    <run_17> 0.000465 </run_17>
  </Cnd5>

- <Cnd6>
    <run_1> -0.000433 </run_1>
    <run_2> -0.000437 </run_2>
    <run_3> -0.000441 </run_3>
    <run_4> -0.000444 </run_4>
    <run_5> -0.000447 </run_5>
    <run_6> -0.000450 </run_6>
    <run_7> -0.000453 </run_7>
    <run_8> -0.000455 </run_8>
    <run_9> -0.000457 </run_9>
    <run_10> -0.000459 </run_10>
    <run_11> -0.000461 </run_11>
    <run_12> -0.000462 </run_12>
    <run_13> -0.000463 </run_13>
    <run_14> -0.000464 </run_14>
    <run_15> -0.000465 </run_15>
    <run_16> -0.000465 </run_16>
    <run_17> -0.000465 </run_17>
  </Cnd6>
```

Figure 57 Noctua B1 Rudder Power Calculation

Figure 57 depicts the alpha file coefficients of Noctua B1 that were relevant to the rudder power calculation. The rudder surfaces were on surfaces 5 and 6. Alpha equal to zero occurred at run 12.



```
// Servo Response Time
Actuators=StandardActuator.txt


//-------------------  INERTIA  -------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
Empty_Mass=26.6

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=4.5623
Pitch_Inertia=6.6035
Yaw_Inertia=10.6141


//--------  Surface Effectiveness Adjustments  --------//

Cmd_scaler_d5=1.05576204
Cmd_scaler_d6=1.05576204
Cnd_scaler_d5=0.62022818
Cnd_scaler_d6=0.62022818


//------------------- PROPULSION -------------------//

// Gas Motor
Left_Engine_Type=0
Left_Engine_Channel=2
Left_Engine_LUT=DA100L.lut

// Engine-specific fuel consumption
Left_Engine_BSFC=1785
```

Figure 58 Noctua B1 Cnd Scaler

Figure 58 shows a snapshot of the Noctua B1 simulator file. The simulator file had a $C_{n\delta}$Scaler value of 0.62022818 that had been determined from the results of the rudder doublet maneuver tests.

$$LRuddervator = Elevator + -Rudder$$

$$RRuddervator = Elevator + Rudder$$

The two equations above define ruddervator deflection commands. The rudder power calculation is only concerned with commanding rudder deflections; therefore, the rudder deflections are exactly the same magnitude as the ruddervator deflections, i.e. a rudder deflection command of 5 degrees would deflect both ruddervator control surfaces 5 degrees. The ruddervator deflection sign convention follows that of elevators; therefore, a positive rudder deflection will result in a negative left ruddervator deflection.

$$Rudder\ Power = C_{n_{\delta_r}}(\alpha = 0) * \frac{180}{\pi} * C_{n_\delta}Scaler$$

$$LRudder = -LRuddervator = -d6 \quad RRudder = RRuddervator = d5$$

$$C_{n_{\delta_r}}(\alpha = 0) = -Cnd6(\alpha = 0) + Cnd5(\alpha = 0)$$

$$Rudder\ Power = [-Cnd6(run\ 12) + Cnd5(run\ 12)] * \frac{180}{\pi} * Cnd\_Scaler$$

$$= [-(-0.000462) + 0.000462] * \frac{180}{\pi} * 0.62022818$$

= 0.032836 /rad

The manual calculation produced a rudder power of 0.032836 /rad which was an exact match with the simulator calculated value of 0.032836 /rad.

3.10. Rudder Effectiveness

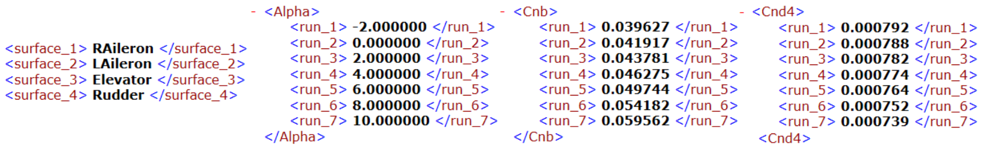
The simulator calculates rudder effectiveness using the rudder control surface derivatives and the stability derivative Cnb from the alpha file. Additionally the simulator will scale the rudder effectiveness according to the Cnd scalar term in the alpha file.

$$Rudder\ Effectiveness = -\frac{C_{n_{\delta_r}}(\alpha = 0)}{C_{n_\beta}(\alpha = 0) * \frac{\pi}{180}} * C_{n_\delta}Scaler$$

Equation 8 Rudder Effectiveness

It was found that the simulator uses Equation 8 to calculate the rudder effectiveness. $C_{n\delta r}$ ($\alpha = 0$) is the summation of the change in yawing moment per change in control surface deflection of each control surface that is interpreted as a rudder by the simulator at alpha equal to zero. $C_{n\beta}$ ($\alpha = 0$) is the change in yawing moment per change in sideslip angle, and is denoted as "Cnb" in the alpha file. Additionally Cnb is recorded in units of /rad and therefore must be converted to /deg for the calculation. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the values of $C_{n\delta r}$ and $C_{n\beta}$ at alpha equal to zero. $C_{n\delta}$Scaler is the scaling term that can be applied in the simulator file to adjust the model to reflect doublet maneuver test results.

3.10.1. Nexstar Example Calculation



```
<surface_1> RAileron </surface_1>
<surface_2> LAileron </surface_2>
<surface_3> Elevator </surface_3>
<surface_4> Rudder </surface_4>

- <Alpha>
    <run_1> -2.000000 </run_1>
    <run_2> 0.000000 </run_2>
    <run_3> 2.000000 </run_3>
    <run_4> 4.000000 </run_4>
    <run_5> 6.000000 </run_5>
    <run_6> 8.000000 </run_6>
    <run_7> 10.000000 </run_7>
  </Alpha>

- <Cnb>
    <run_1> 0.039627 </run_1>
    <run_2> 0.041917 </run_2>
    <run_3> 0.043781 </run_3>
    <run_4> 0.046275 </run_4>
    <run_5> 0.049744 </run_5>
    <run_6> 0.054182 </run_6>
    <run_7> 0.059562 </run_7>
  </Cnb>

- <Cnd4>
    <run_1> 0.000792 </run_1>
    <run_2> 0.000788 </run_2>
    <run_3> 0.000782 </run_3>
    <run_4> 0.000774 </run_4>
    <run_5> 0.000764 </run_5>
    <run_6> 0.000752 </run_6>
    <run_7> 0.000739 </run_7>
  <Cnd4>
```

Figure 59 Nexstar Rudder Effectiveness Calculation

Figure 59 depicts the alpha file coefficients of Nexstar that were relevant to the rudder effectiveness calculation. The rudder surface was on surface 4. Alpha equal to zero occurred at run 2. Additionally there was not a Cnd scaling term in the Nexstar simulator file.

$$Rudder\ Effectiveness = -\frac{C_{n_{\delta_r}}(\alpha = 0)}{C_{n_\beta}(\alpha = 0) * \frac{\pi}{180}} * C_{n_\delta}Scaler$$

$$C_{n_{\delta_r}}(\alpha = 0) = Cnd4(\alpha = 0)$$

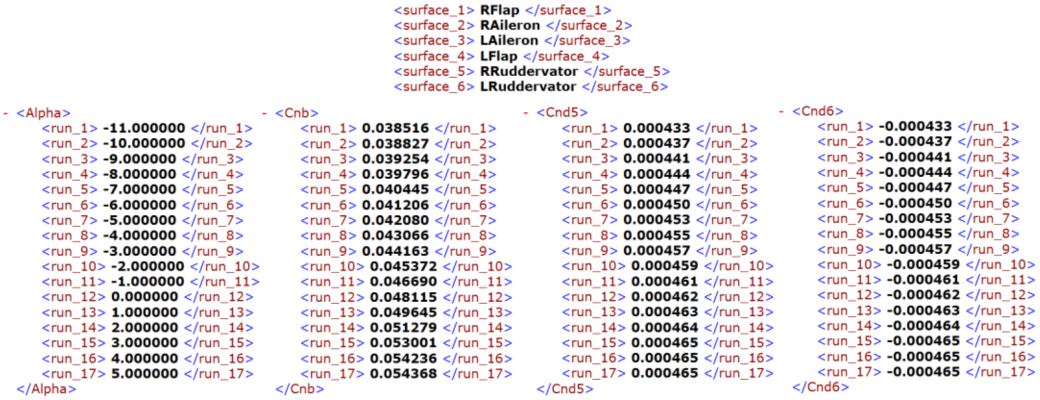
$$Rudder\ Effectivness = -\frac{Cnd5(run\ 2)}{Cnb(run\ 2) * \frac{\pi}{180}} * Cnd\_scaler$$

$$= -\frac{0.000788}{0.041917 * \frac{\pi}{180}} * 1$$

$$= -1.077107$$

The manual calculation produced a rudder effectiveness of -1.077107. The simulator calculated the rudder effectiveness to be -1.077161. The percent difference is 0.00506%.

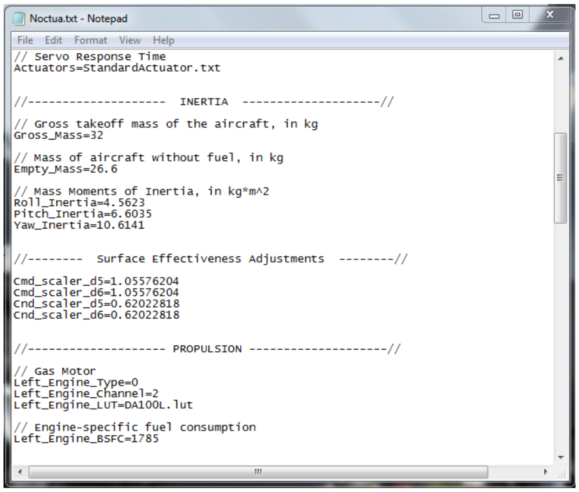
3.10.2. Noctua B1 Example Calculation



```
<surface_1> RFlap </surface_1>
<surface_2> RAileron </surface_2>
<surface_3> LAileron </surface_3>
<surface_4> LFlap </surface_4>
<surface_5> RRuddervator </surface_5>
<surface_6> LRuddervator </surface_6>

- <Alpha>
    <run_1> -11.000000 </run_1>
    <run_2> -10.000000 </run_2>
    <run_3> -9.000000 </run_3>
    <run_4> -8.000000 </run_4>
    <run_5> -7.000000 </run_5>
    <run_6> -6.000000 </run_6>
    <run_7> -5.000000 </run_7>
    <run_8> -4.000000 </run_8>
    <run_9> -3.000000 </run_9>
    <run_10> -2.000000 </run_10>
    <run_11> -1.000000 </run_11>
    <run_12> 0.000000 </run_12>
    <run_13> 1.000000 </run_13>
    <run_14> 2.000000 </run_14>
    <run_15> 3.000000 </run_15>
    <run_16> 4.000000 </run_16>
    <run_17> 5.000000 </run_17>
  </Alpha>
- <Cnb>
    <run_1> 0.038516 </run_1>
    <run_2> 0.038827 </run_2>
    <run_3> 0.039254 </run_3>
    <run_4> 0.039796 </run_4>
    <run_5> 0.040445 </run_5>
    <run_6> 0.041206 </run_6>
    <run_7> 0.042080 </run_7>
    <run_8> 0.043066 </run_8>
    <run_9> 0.044163 </run_9>
    <run_10> 0.045372 </run_10>
    <run_11> 0.046690 </run_11>
    <run_12> 0.048115 </run_12>
    <run_13> 0.049645 </run_13>
    <run_14> 0.051279 </run_14>
    <run_15> 0.053001 </run_15>
    <run_16> 0.054236 </run_16>
    <run_17> 0.054368 </run_17>
  </Cnb>
- <Cnd5>
    <run_1> 0.000433 </run_1>
    <run_2> 0.000437 </run_2>
    <run_3> 0.000441 </run_3>
    <run_4> 0.000444 </run_4>
    <run_5> 0.000447 </run_5>
    <run_6> 0.000450 </run_6>
    <run_7> 0.000453 </run_7>
    <run_8> 0.000455 </run_8>
    <run_9> 0.000457 </run_9>
    <run_10> 0.000459 </run_10>
    <run_11> 0.000461 </run_11>
    <run_12> 0.000462 </run_12>
    <run_13> 0.000463 </run_13>
    <run_14> 0.000464 </run_14>
    <run_15> 0.000465 </run_15>
    <run_16> 0.000465 </run_16>
    <run_17> 0.000465 </run_17>
  </Cnd5>
- <Cnd6>
    <run_1> -0.000433 </run_1>
    <run_2> -0.000437 </run_2>
    <run_3> -0.000441 </run_3>
    <run_4> -0.000444 </run_4>
    <run_5> -0.000447 </run_5>
    <run_6> -0.000450 </run_6>
    <run_7> -0.000453 </run_7>
    <run_8> -0.000455 </run_8>
    <run_9> -0.000457 </run_9>
    <run_10> -0.000459 </run_10>
    <run_11> -0.000461 </run_11>
    <run_12> -0.000462 </run_12>
    <run_13> -0.000463 </run_13>
    <run_14> -0.000464 </run_14>
    <run_15> -0.000465 </run_15>
    <run_16> -0.000465 </run_16>
    <run_17> -0.000465 </run_17>
  </Cnd6>
```

Figure 60 Noctua B1 Rudder Effectiveness Calculation

Figure 60 depicts the alpha file coefficients of Noctua B1 that were relevant to the rudder effectiveness calculation. The rudder surfaces were on surfaces 5 and 6. Alpha equal to zero occurred at run 12.



```
// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
Empty_Mass=26.6

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=4.5623
Pitch_Inertia=6.6035
Yaw_Inertia=10.6141


//--------  Surface Effectiveness Adjustments  --------//

Cmd_scaler_d5=1.05576204
Cmd_scaler_d6=1.05576204
Cnd_scaler_d5=0.62022818
Cnd_scaler_d6=0.62022818


//-------------------- PROPULSION --------------------//

// Gas Motor
Left_Engine_Type=0
Left_Engine_Channel=2
Left_Engine_LUT=DA100L.lut

// Engine-specific fuel consumption
Left_Engine_BSFC=1785
```

Figure 61 Noctua B1 Cnd Scaler

Figure 61 shows a snapshot of the Noctua B1 simulator file. The simulator file had a $C_{n\delta}$Scaler value of 0.62022818 that had been determined from the results of the rudder doublet maneuver tests.

$$LRuddervator = Elevator + \, -Rudder$$

$$RRuddervator = Elevator + Rudder$$

The two equations above define ruddervator deflection commands. The rudder effectiveness calculation is only concerned with commanding rudder deflections; therefore, the rudder deflections are exactly the same magnitude as the ruddervator deflections, i.e. a rudder deflection command of 5 degrees would deflect both ruddervator control surfaces 5 degrees. The ruddervator deflection sign convention follows that of elevators; therefore, a positive rudder deflection will result in a negative left ruddervator deflection.

$$Rudder\ Effectiveness = -\frac{C_{n_{\delta_r}}(\alpha = 0)}{C_{n_\beta}(\alpha = 0) * \frac{\pi}{180}} * C_{n_\delta}Scaler$$

$$LRudder = \ -LRuddervator = -d6 \quad RRudder = RRuddervator = d5$$

$$C_{n_{\delta_r}}(\alpha = 0) = -Cnd6(\alpha = 0) + Cnd5(\alpha = 0)$$

$$Rudder\ Effectivness = -\frac{-Cnd6(run\ 12) + Cnd5(run\ 12)}{Cnb(run\ 12) * \frac{\pi}{180}} * Cnd\_scaler$$

$$= -\frac{[-(-0.000462) + 0.000462]}{0.048115 * \frac{\pi}{180}} * 0.62022818$$

$$= -0.682442$$

The manual calculation produced a rudder effectiveness of -0.682442. The simulator calculated the rudder effectiveness to be -0.681820. The percent difference is 0.09%.

3.11. Sideslip Effectiveness

The simulator calculates sideslip effectiveness using the change in side force per change in sideslip angle stability derivative along with CDff and Cdvis from the alpha file.

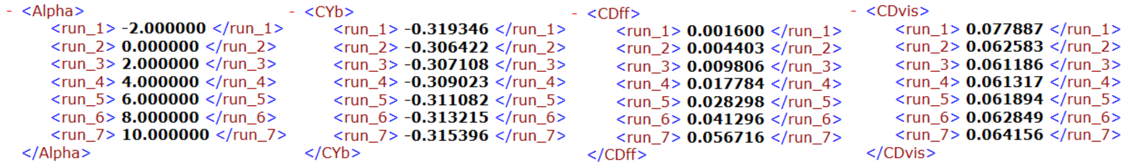
$$Sideslip\ Effectiveness = C_{Y_\beta}(\alpha = 0) - CDff(\alpha = 0) - CDvis(\alpha = 0)$$

Equation 9 Sideslip Effectiveness

It was found that the simulator uses an equation nearly the same as Equation 9 to calculate the sideslip effectiveness. $C_{Y\beta}$ (α = 0) is the change in side force per change in sideslip angle at alpha equal to zero. CDff (α = 0) is the induced, trefftz plane, drag at alpha equal to zero. Cdvis (α = 0) is the viscous drag at alpha equal to zero. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the values of $C_{Y\beta}$, CDff, and Cdvis at alpha equal to zero.

The effect of drag was found to not be exactly equal to CDff + Cdvis. The difference was so small that it was assumed to be a rounding error or some small vectoring angle contribution. The coefficients that contribute to the sideslip effectiveness calculation were for certain narrowed down to CDff and Cdvis at zero alpha; however, when Equation 9 was used to calculate the sideslip effectiveness of two different aircraft models the manually calculated drag was too high by 0.0209%. The difference can be seen in the following example manual calculations.

3.11.1. Nexstar Example



```
- <Alpha>
    <run_1> -2.000000 </run_1>
    <run_2> 0.000000 </run_2>
    <run_3> 2.000000 </run_3>
    <run_4> 4.000000 </run_4>
    <run_5> 6.000000 </run_5>
    <run_6> 8.000000 </run_6>
    <run_7> 10.000000 </run_7>
  </Alpha>
- <CYb>
    <run_1> -0.319346 </run_1>
    <run_2> -0.306422 </run_2>
    <run_3> -0.307108 </run_3>
    <run_4> -0.309023 </run_4>
    <run_5> -0.311082 </run_5>
    <run_6> -0.313215 </run_6>
    <run_7> -0.315396 </run_7>
  </CYb>
- <CDff>
    <run_1> 0.001600 </run_1>
    <run_2> 0.004403 </run_2>
    <run_3> 0.009806 </run_3>
    <run_4> 0.017784 </run_4>
    <run_5> 0.028298 </run_5>
    <run_6> 0.041296 </run_6>
    <run_7> 0.056716 </run_7>
  </CDff>
- <CDvis>
    <run_1> 0.077887 </run_1>
    <run_2> 0.062583 </run_2>
    <run_3> 0.061186 </run_3>
    <run_4> 0.061317 </run_4>
    <run_5> 0.061894 </run_5>
    <run_6> 0.062849 </run_6>
    <run_7> 0.064156 </run_7>
  </CDvis>
```

Figure 62 Nexstar Sideslip Effectiveness Calculation

Figure 62 depicts the alpha file coefficients of Nexstar that were relevant to the side slip effectiveness calculation. Alpha equal to zero occurred at run 2.

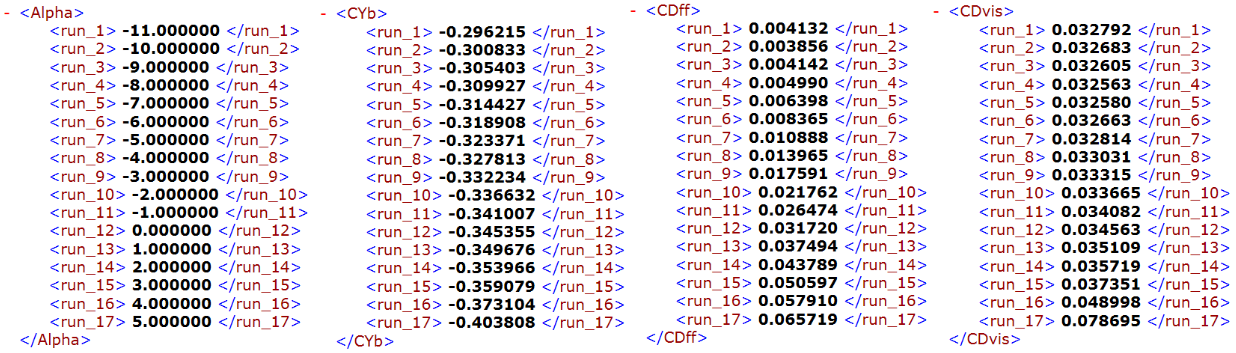
$$Sideslip\ Effectiveness = CYb(run\ 2) - CDff(run\ 2) - CDvis(run\ 2)$$

$$= -0.306422 - 0.004403 - 0.062583$$

$$= \ -0.373408\ /rad$$

The manual calculation produced a sideslip effectiveness of -0.373408 /rad. The simulator calculated the sideslip effectiveness to be -0.373208 /rad. The percent difference is 0.05% which is attributed to the drag contribution.

3.11.2. Noctua B1 Example



```
- <Alpha>
    <run_1> -11.000000 </run_1>
    <run_2> -10.000000 </run_2>
    <run_3> -9.000000 </run_3>
    <run_4> -8.000000 </run_4>
    <run_5> -7.000000 </run_5>
    <run_6> -6.000000 </run_6>
    <run_7> -5.000000 </run_7>
    <run_8> -4.000000 </run_8>
    <run_9> -3.000000 </run_9>
    <run_10> -2.000000 </run_10>
    <run_11> -1.000000 </run_11>
    <run_12> 0.000000 </run_12>
    <run_13> 1.000000 </run_13>
    <run_14> 2.000000 </run_14>
    <run_15> 3.000000 </run_15>
    <run_16> 4.000000 </run_16>
    <run_17> 5.000000 </run_17>
  </Alpha>
- <CYb>
    <run_1> -0.296215 </run_1>
    <run_2> -0.300833 </run_2>
    <run_3> -0.305403 </run_3>
    <run_4> -0.309927 </run_4>
    <run_5> -0.314427 </run_5>
    <run_6> -0.318908 </run_6>
    <run_7> -0.323371 </run_7>
    <run_8> -0.327813 </run_8>
    <run_9> -0.332234 </run_9>
    <run_10> -0.336632 </run_10>
    <run_11> -0.341007 </run_11>
    <run_12> -0.345355 </run_12>
    <run_13> -0.349676 </run_13>
    <run_14> -0.353966 </run_14>
    <run_15> -0.359079 </run_15>
    <run_16> -0.373104 </run_16>
    <run_17> -0.403808 </run_17>
  </CYb>
- <CDff>
    <run_1> 0.004132 </run_1>
    <run_2> 0.003856 </run_2>
    <run_3> 0.004142 </run_3>
    <run_4> 0.004990 </run_4>
    <run_5> 0.006398 </run_5>
    <run_6> 0.008365 </run_6>
    <run_7> 0.010888 </run_7>
    <run_8> 0.013965 </run_8>
    <run_9> 0.017591 </run_9>
    <run_10> 0.021762 </run_10>
    <run_11> 0.026474 </run_11>
    <run_12> 0.031720 </run_12>
    <run_13> 0.037494 </run_13>
    <run_14> 0.043789 </run_14>
    <run_15> 0.050597 </run_15>
    <run_16> 0.057910 </run_16>
    <run_17> 0.065719 </run_17>
  </CDff>
- <CDvis>
    <run_1> 0.032792 </run_1>
    <run_2> 0.032683 </run_2>
    <run_3> 0.032605 </run_3>
    <run_4> 0.032563 </run_4>
    <run_5> 0.032580 </run_5>
    <run_6> 0.032663 </run_6>
    <run_7> 0.032814 </run_7>
    <run_8> 0.033031 </run_8>
    <run_9> 0.033315 </run_9>
    <run_10> 0.033665 </run_10>
    <run_11> 0.034082 </run_11>
    <run_12> 0.034563 </run_12>
    <run_13> 0.035109 </run_13>
    <run_14> 0.035719 </run_14>
    <run_15> 0.037351 </run_15>
    <run_16> 0.048998 </run_16>
    <run_17> 0.078695 </run_17>
  </CDvis>
```

Figure 63 Noctua B1 Sideslip Effectiveness Calculation

Figure 63 depicts the alpha file coefficients of Noctua B1 that were relevant to the side slip effectiveness calculation. Alpha equal to zero occurred at run 12.

$$Sideslip\ Effectiveness = C_{Y_\beta}(run\ 12) - CDff(run\ 12) - CDvis(run\ 12)$$

$$= -0.345355 - 0.031720 - 0.034563$$

$$= \ -0.411638\ /rad$$

The manual calculation produced a sideslip 119llinotiveness of -0.411638 /rad. The simulator calculated the sideslip effectiveness to be -0.411414 /rad. The percent difference is 0.05% which is attributed to the drag contribution.

3.12. Elevator Power

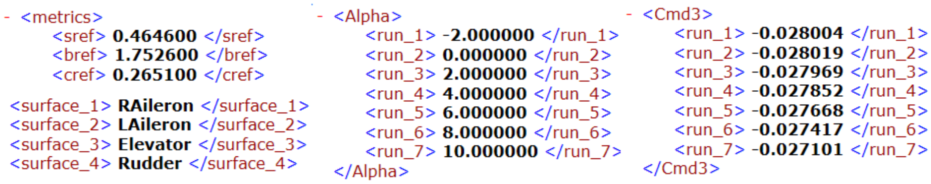
The simulator calculates elevator power using the elevator control surface derivatives along with wingspan, wing chord, and wing area from the alpha file. Additionally the simulator will scale elevator power according to the Cmd scalar term in the alpha file.

$$Elevator\ Power = C_{m_{\delta_e}}(\alpha = 0) * \frac{180}{\pi} * \frac{c * b}{S_w} * C_{m_\delta}Scaler$$

Equation 10 Elevator Power

It was found that the simulator uses Equation 10 to calculate the elevator power. $C_{m\delta e}$ (α = 0) is the summation of the change in pitching moment per change in control surface deflection of each control surface that is interpreted to be an elevator by the simulator at alpha equal to zero. The units of . $C_{m\delta e}$ are /deg so they are converted to /rad as the units of elevator power are defined as /rad. The wing chord, or c, is labeled as “cref” in the alpha file. The wing span, b, is labeled as “bref” in the alpha file. The wing area, Sw, is labeled as “sref” in the alpha file. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the values of $C_{n\delta r}$ and $C_{n\beta}$ at alpha equal to zero. $C_{m\delta}$Scaler is the scaling term that can be applied in the simulator file to adjust the model to reflect doublet maneuver test results.

3.12.1. Nexstar Example Calculation



```
- <metrics>
    <sref> 0.464600 </sref>
    <bref> 1.752600 </bref>
    <cref> 0.265100 </cref>
  <surface_1> RAileron </surface_1>
  <surface_2> LAileron </surface_2>
  <surface_3> Elevator </surface_3>
  <surface_4> Rudder </surface_4>

- <Alpha>
    <run_1> -2.000000 </run_1>
    <run_2> 0.000000 </run_2>
    <run_3> 2.000000 </run_3>
    <run_4> 4.000000 </run_4>
    <run_5> 6.000000 </run_5>
    <run_6> 8.000000 </run_6>
    <run_7> 10.000000 </run_7>
  </Alpha>

- <Cmd3>
    <run_1> -0.028004 </run_1>
    <run_2> -0.028019 </run_2>
    <run_3> -0.027969 </run_3>
    <run_4> -0.027852 </run_4>
    <run_5> -0.027668 </run_5>
    <run_6> -0.027417 </run_6>
    <run_7> -0.027101 </run_7>
  </Cmd3>
```

Figure 64 Nexstar Elevator Power Calculation

Figure 64 depicts the alpha file coefficients of Nexstar that were relevant to the elevator power calculation. The elevator surface was surfaces 3. Alpha equal to zero occurred at run 2.

$$Elevator\ Power = C_{m_{\delta_e}}(\alpha = 0) * \frac{180}{\pi} * \frac{c * b}{S_w} * C_{m_\delta}Scaler$$

$$C_{m_{\delta_e}}(\alpha = 0) = Cmd3(\alpha = 0)$$

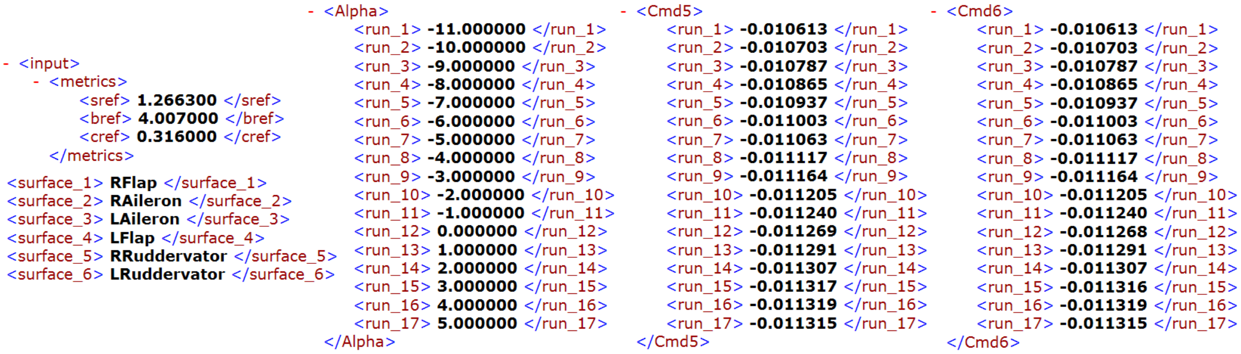
$$= Cmd3(run\ 2) * \frac{180}{\pi} * \frac{cref * bref}{sref} * Cmd\_scaler$$

$$= \ -0.028019 * \frac{180}{\pi} * \frac{0.2651 * 1.7526}{0.4646} * 1$$

$$= \ -1.605420 \text{ /rad}$$

The manual calculation produced an elevator power of -1.605420 /rad which was an exact match for the simulator calculated number of -1.605420 /rad.

3.12.2. Noctua B1 Example Calculation



```
- <input>
   - <metrics>
        <sref> 1.266300 </sref>
        <bref> 4.007000 </bref>
        <cref> 0.316000 </cref>
     </metrics>
  <surface_1> RFlap </surface_1>
  <surface_2> RAileron </surface_2>
  <surface_3> LAileron </surface_3>
  <surface_4> LFlap </surface_4>
  <surface_5> RRuddervator </surface_5>
  <surface_6> LRuddervator </surface_6>
```

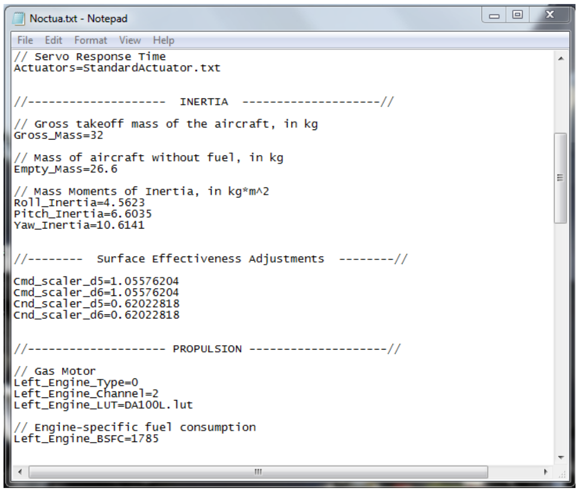
```
- <Alpha>
     <run_1> -11.000000 </run_1>
     <run_2> -10.000000 </run_2>
     <run_3> -9.000000 </run_3>
     <run_4> -8.000000 </run_4>
     <run_5> -7.000000 </run_5>
     <run_6> -6.000000 </run_6>
     <run_7> -5.000000 </run_7>
     <run_8> -4.000000 </run_8>
     <run_9> -3.000000 </run_9>
     <run_10> -2.000000 </run_10>
     <run_11> -1.000000 </run_11>
     <run_12> 0.000000 </run_12>
     <run_13> 1.000000 </run_13>
     <run_14> 2.000000 </run_14>
     <run_15> 3.000000 </run_15>
     <run_16> 4.000000 </run_16>
     <run_17> 5.000000 </run_17>
  </Alpha>
```

```
- <Cmd5>
     <run_1> -0.010613 </run_1>
     <run_2> -0.010703 </run_2>
     <run_3> -0.010787 </run_3>
     <run_4> -0.010865 </run_4>
     <run_5> -0.010937 </run_5>
     <run_6> -0.011003 </run_6>
     <run_7> -0.011063 </run_7>
     <run_8> -0.011117 </run_8>
     <run_9> -0.011164 </run_9>
     <run_10> -0.011205 </run_10>
     <run_11> -0.011240 </run_11>
     <run_12> -0.011269 </run_12>
     <run_13> -0.011291 </run_13>
     <run_14> -0.011307 </run_14>
     <run_15> -0.011317 </run_15>
     <run_16> -0.011319 </run_16>
     <run_17> -0.011315 </run_17>
  </Cmd5>
```

```
- <Cmd6>
     <run_1> -0.010613 </run_1>
     <run_2> -0.010703 </run_2>
     <run_3> -0.010787 </run_3>
     <run_4> -0.010865 </run_4>
     <run_5> -0.010937 </run_5>
     <run_6> -0.011003 </run_6>
     <run_7> -0.011063 </run_7>
     <run_8> -0.011117 </run_8>
     <run_9> -0.011164 </run_9>
     <run_10> -0.011205 </run_10>
     <run_11> -0.011240 </run_11>
     <run_12> -0.011268 </run_12>
     <run_13> -0.011291 </run_13>
     <run_14> -0.011307 </run_14>
     <run_15> -0.011316 </run_15>
     <run_16> -0.011319 </run_16>
     <run_17> -0.011315 </run_17>
  </Cmd6>
```

Figure 65 Noctua B1 Elevator Power Calculation

Figure 65 depicts the alpha file coefficients of Noctua B1 that were relevant to the elevator power calculation. The elevator surfaces were on surfaces 5 and 6. Alpha equal to zero occurred at run 12.



```
// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
Empty_Mass=26.6

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=4.5623
Pitch_Inertia=6.6035
Yaw_Inertia=10.6141


//--------  Surface Effectiveness Adjustments  --------//

Cmd_scaler_d5=1.05576204
Cmd_scaler_d6=1.05576204
Cnd_scaler_d5=0.62022818
Cnd_scaler_d6=0.62022818


//-------------------- PROPULSION --------------------//

// Gas Motor
Left_Engine_Type=0
Left_Engine_Channel=2
Left_Engine_LUT=DA100L.lut

// Engine-specific fuel consumption
Left_Engine_BSFC=1785
```

Figure 66 Noctua B1 Cmd Scaler

Figure 66 shows a snapshot of the Noctua B1 simulator file. The simulator file had a $C_{m\delta}$Scaler value of 1.05576204 that had been determined from the results of the elevator doublet maneuver tests.

$$LRuddervator = Elevator + \ -Rudder$$

$$RRuddervator = Elevator + Rudder$$

The two equations above define ruddervator deflection commands. The elevator power calculation is only concerned with commanding elevator deflections; therefore, the elevator deflections are exactly the same magnitude as the ruddervator deflections, i.e. an elevator deflection command of 10 degrees would deflect both ruddervator control surfaces 10 degrees.

$$Elevator\ Power = C_{m_{\delta_e}}(\alpha = 0) * \frac{180}{\pi} * \frac{c * b}{S_w} * C_{m_\delta}Scaler$$

$$LElevator = \ LRuddervator = d6 \quad RElevator = RRuddervator = d5$$

$$C_{m_{\delta_e}}(\alpha = 0) = Cmd6(\alpha = 0) + Cmd5(\alpha = 0)$$

$$= [Cmd6(run\ 12) + Cmd5(run\ 12)] * \frac{180}{\pi} * \frac{cref * bref}{sref} * Cmd\_scaler$$

$$= [-0.011268 + -0.011269] * \frac{180}{\pi} * \frac{0.3160 * 4.0070}{1.2663} * 1.05576204$$

$$= -1.363184 \text{ /rad}$$

The manual calculation produced an elevator power of -1.363184 /rad which was an exact match for the simulator calculated number of -1.363184 /rad.

3.13. Elevator Effectiveness

The piccolo is defined as to interpret elevator effectiveness as the change in CL of an aircraft per change in elevator deflection. The piccolo calculates CL using measurements of the z – accelerometer data amongst a few other parameters. With respect to the z – accelerometer measurements the autopilot interprets a change in CL as the change in measured z – force. Essentially elevator effectiveness is how much the aircraft will pitch with a given elevator deflection.

The coefficients from the alpha file that the simulator uses to estimate elevator effectiveness were determined by iteratively altering coefficients in the alpha file and observing whether or not they had an effect on the simulator's elevator effectiveness calculation. It was found that the following coefficients are used:

1) Cmtot

2) CLff

3) Clde

- Where Clde is the summation of CLd# of all of the control surfaces that are interpreted as elevator control surfaces

4) Cmde

    - Where Cmde is the summation of Cmd# of all of the control surfaces that are interpreted as elevator control surfaces

5) CDff

6) Cdvis

7) Cdffde

    - Where Cdffde is the summation of CDffd# of all of the control surfaces that are interpreted as elevator control surfaces

It was also determined that the simulator only uses specific alpha values of each coefficient. After many iterations, of manipulating alpha file coefficients and even eliminating drag, the exact calculation used by the simulator could not be found. It was found that Clde was decreasing the elevator effectiveness by almost exactly its magnitude. It was also found that both drag coefficients Cdffde and CDff add to the elevator effectiveness estimate. At the time these experiments were conducted the Noctua B1 model had a problem with the elevator effectiveness calculation. The simulator was estimating an elevator effectiveness of zero. The model had been emailed to Cloud Cap and the response shed light as to how the simulator estimates elevator effectiveness.

Cloud Cap support stated that the simulator first looks for values of CLff between 0.1 and 0.9 in the alpha range of 0 to 5 degrees. This range is considered by the simulator to be the "linear range". The simulator estimates elevator effectiveness using only alpha values in the linear range. Additionally the following instructions were given to manually estimate elevator effectiveness using the avl program itself:

1) Set all the physical parameters i.e. cruise velocity, inertial parameters, aircraft mass, gravity.

2) Deflect the elevators to the largest negative value and iterate runs changing alpha to trim the aircraft.

3) Record the resulting CL.

4) Increment the elevator deflection a couple degrees.

5) Re-trim by driving alpha so pitching moment is zero.

6) Record the resulting CL.

7) Repeat until you reach the highest positive elevator deflection.

8) Extract the largest (most negative) difference in C_L/δ_e and use that as the initial estimate for elevator effectiveness.

The steps shed a lot of light on how the simulator estimates elevator effectiveness. At each alpha value in the linear range the simulator trims the aircraft with the required elevator deflection, records the CL and elevator deflection, and finds the largest negative difference of C_L/δ_e between all of the alpha values that exist.

It appeared that the simulator calculates an elevator trim deflection for each alpha in the alpha file, record the CL, record the elevator deflection, and continue through all of the alpha runs in the linear range. In order to account for the change in lift of the aircraft profile when the elevator is deflected (Clde) it appears that the simulator adds the change in CL from the elevator deflection to the aircraft's overall CLff value at each alpha (Clde * de). Similar to the estimation of Clde0 the drag appeared to be included by vectoring the forces where drag would actually impact the z-force measurement.

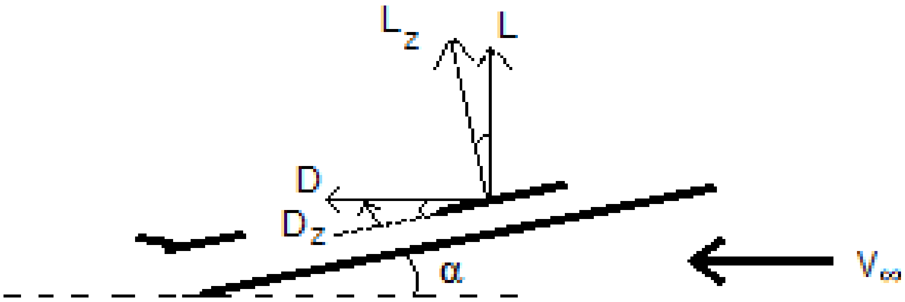


Figure 67 Elevator Effectiveness Lift Drag Vectors

Figure 67 depicts an aircraft profile as it is oriented for each alpha in alpha sweeps. Lz and Dz are the lift and drag force vectors that act along the autopilot's axes. The z force lift that the autopilot would measure in Figure 67 is Lcos(α) + Dsin(α). In the figure the aircraft's actual lift, L, is the lift represented by CLff + Clde*de. Similarly the aircrafts drag, D, is the drag represented by CDff + Cdvis+Cdffde*de. In order to trim the aircraft at an alpha run it was theorized that the simulator will calculate Cmtrim, and detrim from the value of Cmtot and Cmde at the alpha run in question. Additionally it was theorized that the simulator will calculate CL, and CD and transform them into z force values at the alpha run in question. The equations below depict the process. Additionally if a Cmd scaling term exists in the simulator file it will scale Cmde according to its value and affect the calculations.

$$C_{m\,trim}(\alpha) = -C_{mtot}(\alpha) \Rightarrow \delta_e(\alpha) = \frac{C_{m\,trim}(\alpha)}{C_m/_{\delta_e}(\alpha) * Cmd\_scaler}$$

$$C_L(\alpha) = \left(C_L ff(\alpha) + \frac{C_L}{\delta_e}(\alpha) * \delta_e(\alpha)\right) \quad C_D(\alpha) = \left(C_D ff(\alpha) + C_D vis(\alpha) + \frac{C_D ff}{\delta_e}(\alpha) * \delta_e(\alpha)\right)$$

$$\Rightarrow C_{L_Z}(\alpha) = C_L(\alpha) * \cos\alpha + C_D(\alpha) * \sin\alpha$$

At the end of the calculations the result would have a z force lift and an elevator deflection for the alpha iteration in question. From there one would only have to repeat the same for each alpha in the linear range, and find the largest negative difference in z lift force and elevator deflection.

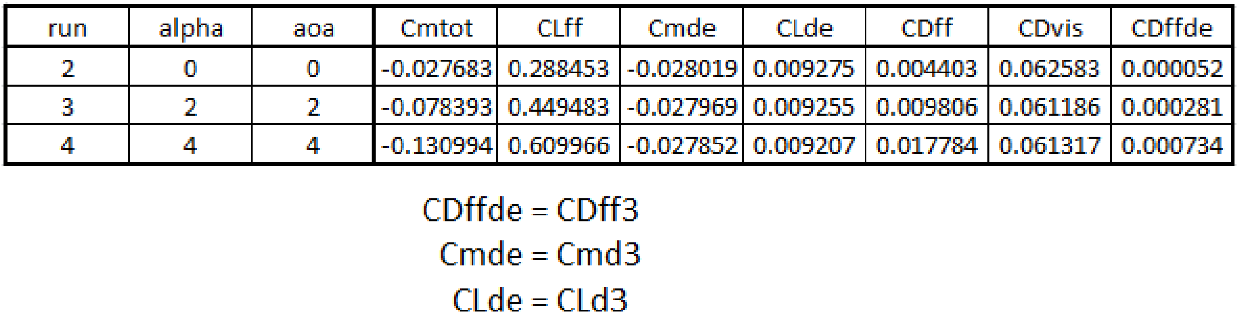
$$Elevator\ Effectiveness = \frac{C_{L_Z}(\alpha_i) - C_{L_Z}(\alpha_j)}{\delta_e(\alpha_i) - \delta_e(\alpha_j)}\ (/deg)$$

The process outlined is not exactly how the simulator estimates elevator effectiveness. It is slight off from what the simulator estimates; however, the difference is very small. Even though the process is not exactly in line with how the simulator estimates elevator effectiveness it can at least be a good place to start to debug any issues with any future models. The effect of drag on the estimation is relatively small. Typically drag will increase a model's simulator estimated elevator effectiveness by 2% at the most. Conversely if a model had erroneously large errors in the drag estimation it could indeed cause the simulator to over estimate elevator effectiveness by a significant amount. Cdvis is an area where errors could occur since it is calculated from the user input xfoil data and CDp. It is important to remember that the whole process is just an estimate and there is a margin for some error; however, it is always better to overestimate than to underestimate. In some cases elevator effectiveness can be off by 1 /rad. Noctua B1 flights ranged from an elevator effectiveness of -5.177 /rad to an elevator effectiveness of -6.1935 /rad with no noticeable difference in performance.

The following two sections detail examples of estimating elevator effectiveness with the process outlined in this section.

3.13.1. Nexstar Example

Since the simulator is supposed to calculate elevator effectiveness from angles of attack between 0 and 5 the linear range was determined by choosing the alpha runs that corresponded with 0 to 5 degrees angle of attack. The alpha angles were the same as the angles of attack; therefore, the range was determined to be alpha 0 to alpha 4 (alpha 5 did not exist because the alpha sweep was ran at increments of 2). The following data was extracted from Nexstar's alpha file.



run	alpha	aoa	Cmtot	CLff	Cmde	CLde	CDff	CDvis	CDffde
2	0	0	-0.027683	0.288453	-0.028019	0.009275	0.004403	0.062583	0.000052
3	2	2	-0.078393	0.449483	-0.027969	0.009255	0.009806	0.061186	0.000281
4	4	4	-0.130994	0.609966	-0.027852	0.009207	0.017784	0.061317	0.000734

CDffde = CDff3
Cmde = Cmd3
CLde = CLd3

A δ_e and C_{LZ} value was calculated for each alpha run. A sample calculation is shown below of alpha run 2.

$$C_{m\,trim}(run\ 2) = -C_{mtot}(run\ 2) = -(-0.027683)$$

$$\delta_e(run\ 2) = \frac{C_{m\,trim}(run\ 2)}{C_m/_{\delta_e}(run\ 2)} = \frac{0.027683}{-0.028019} = -0.98801\ deg$$

$$C_L(run\ 2) = \left(C_L ff(run\ 2) + \frac{C_L}{\delta_e}(run\ 2) * \delta_e(run\ 2)\right)$$

$$= (0.288453 + 0.009275 * -0.98801)$$

$$= 0.279289$$

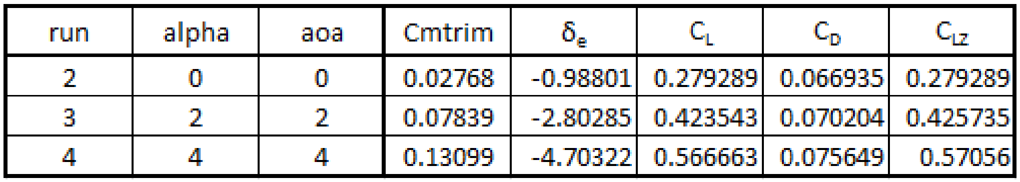
$$C_D(run\ 2) = \left(C_D ff(run\ 2) + C_D vis(run\ 2) + \frac{C_D ff}{\delta_e}(run\ 2) * \delta_e(run\ 2)\right)$$

$$= 0.004403 + 0.062583 + 0.000052 * -0.98801$$

$$= 0.066935$$

$$C_{L_Z}(run\ 2) = C_L(run\ 2) * cos(0) + C_D(run\ 2) * sin(0)$$

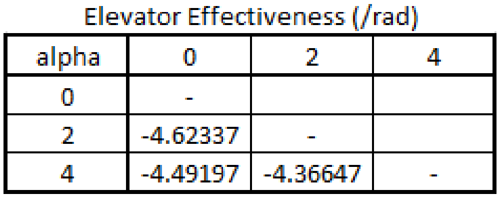
$$= 0.279289 * \cos(0) + 0.066935 * \sin(0) = 0.279289$$



run	alpha	aoa	Cmtrim	δ_e	C_L	C_D	C_{LZ}
2	0	0	0.02768	-0.98801	0.279289	0.066935	0.279289
3	2	2	0.07839	-2.80285	0.423543	0.070204	0.425735
4	4	4	0.13099	-4.70322	0.566663	0.075649	0.57056

Table 2 depicts the results of all of the calculations. The final step was to find the largest $\Delta C_{LZ}/\Delta\delta_e$ between all of the alpha runs.

Table 2 Nexstar Elevator Effectiveness Calculation Results



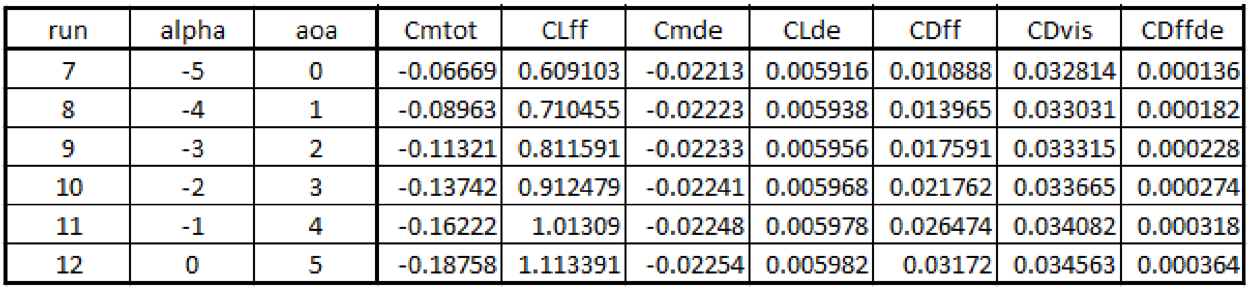
Elevator Effectiveness (/rad)

alpha	0	2	4
0	-		
2	-4.62337	-	
4	-4.49197	-4.36647	-

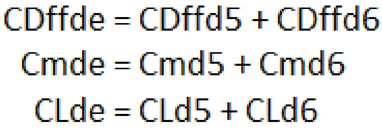
The largest occurred between alpha 0 and 2 with an elevator effectiveness of -4.62337 /rad. The simulator calculated an elevator effectiveness of -4.633491 /rad. The percent difference between the two calculations was 0.219%.

3.13.2. Noctua B1 Example

The simulator did not provide a result for estimating the elevator effectiveness of Noctua B1; however, the process can be compared to the results of the elevator doublet maneuver tests. Noctua B1 had a wing incidence of 5 degrees. Since the simulator is supposed to calculate elevator effectiveness from angles of attack between 0 and 5 the linear range was determined by choosing the alpha runs that corresponded with 0 to 5 degrees angle of attack. The range was determined to be from alpha -5 to alpha 0. The following data was extracted from Noctua B1's alpha file.



run	alpha	aoa	Cmtot	CLff	Cmde	CLde	CDff	CDvis	CDffde
7	-5	0	-0.06669	0.609103	-0.02213	0.005916	0.010888	0.032814	0.000136
8	-4	1	-0.08963	0.710455	-0.02223	0.005938	0.013965	0.033031	0.000182
9	-3	2	-0.11321	0.811591	-0.02233	0.005956	0.017591	0.033315	0.000228
10	-2	3	-0.13742	0.912479	-0.02241	0.005968	0.021762	0.033665	0.000274
11	-1	4	-0.16222	1.01309	-0.02248	0.005978	0.026474	0.034082	0.000318
12	0	5	-0.18758	1.113391	-0.02254	0.005982	0.03172	0.034563	0.000364



CDffde = CDffd5 + CDffd6
Cmde = Cmd5 + Cmd6
CLde = CLd5 + CLd6

A δ_e and C_{LZ} value was calculated for each alpha run. A sample calculation is shown below of alpha run 7.

$$C_{m\,trim}(run\ 7) = -C_{mtot}(run\ 7) = -(-0.06669)$$

$$\delta_e(run\ 7) = \frac{C_{m\,trim}(run\ 7)}{{C_m}/{\delta_e}\ (run\ 7) * Cmd\_scaler} = \frac{0.06669}{-0.02213 * 1} = -3.01397\ deg$$

$$C_L(run\ 7) = \left(C_L ff(run\ 7) + \frac{C_L}{\delta_e}(run\ 7) * \delta_e(run\ 7)\right)$$

$$= (0.609103 + 0.005916 * -3.01397)$$

$$= 0.591272$$

$$C_D(run\ 7) = \left(C_D ff(run\ 7) + C_D vis(run\ 7) + \frac{C_D ff}{\delta_e}(run\ 7) * \delta_e(run\ 7)\right)$$

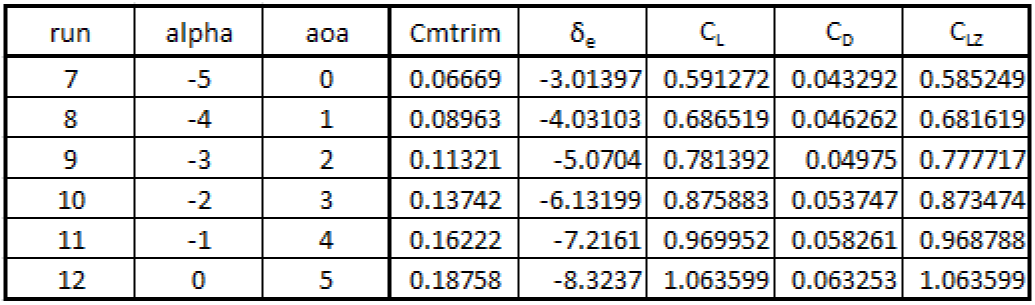
$$= 0.010888 + 0.032814 + 0.000136 * -3.01397$$

$$= 0.043292$$

$$C_{L_Z}(run\ 7) = C_L(run\ 7) * cos(-5) + C_D(run\ 7) * sin(-5)$$

$$= 0.591272 * \cos(-5) + 0.043292 * \sin(-5)$$

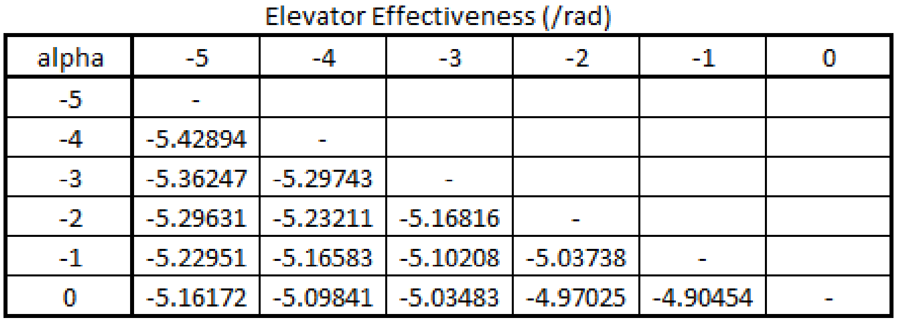
$$= 0.585249$$



run	alpha	aoa	Cmtrim	δ_e	C_L	C_D	C_{LZ}
7	-5	0	0.06669	-3.01397	0.591272	0.043292	0.585249
8	-4	1	0.08963	-4.03103	0.686519	0.046262	0.681619
9	-3	2	0.11321	-5.0704	0.781392	0.04975	0.777717
10	-2	3	0.13742	-6.13199	0.875883	0.053747	0.873474
11	-1	4	0.16222	-7.2161	0.969952	0.058261	0.968788
12	0	5	0.18758	-8.3237	1.063599	0.063253	1.063599

Table 3 below depicts the results of all of the calculations. The final step was to find the largest $\Delta C_{LZ}/\Delta\delta_e$ between all of the alpha runs.

Table 3 Noctua B1 Elevator Effectiveness Calculation Results



Elevator Effectiveness (/rad)

alpha	-5	-4	-3	-2	-1	0
-5	-					
-4	-5.42894	-				
-3	-5.36247	-5.29743	-			
-2	-5.29631	-5.23211	-5.16816	-		
-1	-5.22951	-5.16583	-5.10208	-5.03738	-	
0	-5.16172	-5.09841	-5.03483	-4.97025	-4.90454	-

The largest occurred between alpha -5 and -4 with an elevator effectiveness of -5.42894 /rad. The elevator doublet tests results concluded that Noctua B1 had an elevator effectiveness of -5.46594 /rad. The estimate was indeed close to the flight test results. The error was 0.68%.

3.14. Flap Effectiveness

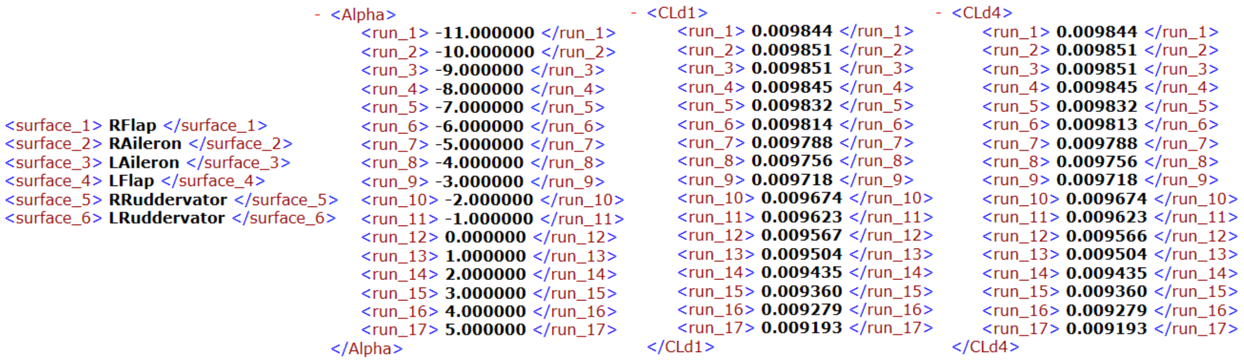
The simulator calculates flap effectiveness using the flap control surface derivative $C_{L\delta f}$.

$$Flap\ Effectiveness = C_{L_{\delta_f}}(\alpha = 0) * \frac{180}{\pi} * C_{L_{\delta}}Scaler$$

Equation 11 Flap Effectiveness

It was found that the simulator uses Equation 11 to calculate the flap effectiveness. $C_{L\delta f}$ ($\alpha = 0$) is the summation of the change in lift per change in control surface deflection of each control surface that is interpreted to act as a flap by the simulator at alpha equal to zero. If there is not an alpha run at alpha equal to zero the simulator will calculate the equation of the line between either the two points that bound alpha equal to zero or the two points closest to it to back out the value of $C_{L\delta f}$ at alpha equal to zero. $C_{L\delta}$Scaler is the scaling term that can be applied in the simulator file to adjust the simulator estimated flap effectiveness to reflect doublet maneuver test results.

3.14.1. Noctua B1 Example



```
<surface_1> RFlap </surface_1>
<surface_2> RAileron </surface_2>
<surface_3> LAileron </surface_3>
<surface_4> LFlap </surface_4>
<surface_5> RRuddervator </surface_5>
<surface_6> LRuddervator </surface_6>
```

```
- <Alpha>
     <run_1> -11.000000 </run_1>
     <run_2> -10.000000 </run_2>
     <run_3> -9.000000 </run_3>
     <run_4> -8.000000 </run_4>
     <run_5> -7.000000 </run_5>
     <run_6> -6.000000 </run_6>
     <run_7> -5.000000 </run_7>
     <run_8> -4.000000 </run_8>
     <run_9> -3.000000 </run_9>
     <run_10> -2.000000 </run_10>
     <run_11> -1.000000 </run_11>
     <run_12> 0.000000 </run_12>
     <run_13> 1.000000 </run_13>
     <run_14> 2.000000 </run_14>
     <run_15> 3.000000 </run_15>
     <run_16> 4.000000 </run_16>
     <run_17> 5.000000 </run_17>
  </Alpha>
```

```
- <CLd1>
     <run_1> 0.009844 </run_1>
     <run_2> 0.009851 </run_2>
     <run_3> 0.009851 </run_3>
     <run_4> 0.009845 </run_4>
     <run_5> 0.009832 </run_5>
     <run_6> 0.009814 </run_6>
     <run_7> 0.009788 </run_7>
     <run_8> 0.009756 </run_8>
     <run_9> 0.009718 </run_9>
     <run_10> 0.009674 </run_10>
     <run_11> 0.009623 </run_11>
     <run_12> 0.009567 </run_12>
     <run_13> 0.009504 </run_13>
     <run_14> 0.009435 </run_14>
     <run_15> 0.009360 </run_15>
     <run_16> 0.009279 </run_16>
     <run_17> 0.009193 </run_17>
  </CLd1>
```

```
- <CLd4>
     <run_1> 0.009844 </run_1>
     <run_2> 0.009851 </run_2>
     <run_3> 0.009851 </run_3>
     <run_4> 0.009845 </run_4>
     <run_5> 0.009832 </run_5>
     <run_6> 0.009813 </run_6>
     <run_7> 0.009788 </run_7>
     <run_8> 0.009756 </run_8>
     <run_9> 0.009718 </run_9>
     <run_10> 0.009674 </run_10>
     <run_11> 0.009623 </run_11>
     <run_12> 0.009566 </run_12>
     <run_13> 0.009504 </run_13>
     <run_14> 0.009435 </run_14>
     <run_15> 0.009360 </run_15>
     <run_16> 0.009279 </run_16>
     <run_17> 0.009193 </run_17>
  </CLd4>
```

Figure 68 Noctua B1 Flap Effectiveness Calculation

Figure 68 depicts the alpha file coefficients of Noctua B1 that were relevant to the flap effectiveness calculation. The flap surfaces were surfaces 1 and 4. Alpha zero was run 12. There was no CLd scaling term for the flaps in the simulator file.

$$Flap\ Effectiveness = C_{L_{\delta_f}}(\alpha = 0) * \frac{180}{\pi} * C_{L_\delta}Scaler$$

$$LFlap = d4 \quad RFlap = d1$$

$$C_{L_{\delta_f}}(\alpha = 0) = CLd4(\alpha = 0) + CLd1(\alpha = 0)$$

$$Flap\ Effectiveness = [CLd4(run\ 12) + CLd1(run\ 12)] * \frac{180}{\pi} * CLd\_scaler$$

$$= (0.009566 + 0.009567) * \frac{180}{\pi} * 1$$

$$= 1.096240$$

The manual calculation produced a flap effectiveness of 1.096240 /rad which was an exact match with the simulator calculated value of 1.096240 /rad.

3.15. CL Max, Max Nom, Climb, and Cruise

It was not determined exactly how the simulator calculates C_L max; however, it was determined that the simulator uses the following coefficients to calculate C_L max:

1) Cmtot

2) CLff

3) CDff

4) Cdvis

5) Clde

6) Cmde

7) Cdffde

It appears that the simulator is iterating calculations of C_L for each alpha and selecting the largest calculated C_L, similar to elevator effectiveness where it calculates the C_L of the aircraft at each alpha with the elevator deflected enough to trim the aircraft at each alpha. How the simulator determines which alpha values to calculate the C_L at is a mystery. It appears to have some limit set where it will only calculate the alpha runs that don’t require a certain amount of elevator deflection to trim. Either way the simulator seems to over estimate the calculations of C_L max. By Cloud Cap’s definition C_L max is supposed to denote “the maximum lift coefficient that the vehicle can sustain with the flaps up” (PccUsersGuide pg. 117).

It would be safe to manually set C_L max in the vehicle parameters as a known near stall value. Another option would be to use the value of CLff at the last alpha of what the user determined to be the linear range in the AVL model.

The simulator calculates the following C_L values directly from its calculation of C_L max:

1) CL max nom = CL max – 0.15

2) CL climb = CL max – 0.30

3) CL cruise = CL max – 0.50

CHAPTER IV

PICCOLO SOFTWARE SETUP GUIDE

4. Introduction

Cloud Cap provides three documents to guide the user through the setup process. “Piccolo Setup Guide”, located in the Cloud Cap folder under “Piccolo Docs”, is a guide designed to instruct the user on how physically connect all of the different autopilot and ground station systems. The document contains one chapter that describes how to setup the software for hardware in the loop simulations in pages 25-39. The information is somewhat fragmented as it only describes a few steps and a few random things for the user to pay attention to. The instructions are focused mostly on using PCC with a few random tips on how to setup flight plans. With respect to the piccolo simulator the instructions only describe how to output data externally to Flight Gear.

“Piccolo Simulator”, located in the Cloud Cap folder under “Piccolo Docs\Software”, details a guide for setting up the piccolo software. The guide covers some of the setup process in good detail, while other important parts are left or easily lost in the clutter of the overall document.

Cloud Cap provides documentation for hardware installation of piccolo and other external hardware add ons. Use Cloud Cap's "Vehicle Integration Guide" to properly install the autopilot, communications antennas, and GPS antennas. Note that the autopilot does not have to be installed on the center of gravity; however, the autopilot does have to be oriented parallel with the vehicle axes. Additionally the orientation must match one of the orientation options in the Sensor Configuration window of PCC.

Cloud Cap offers pre made wiring harnesses to go with autopilots. Custom wiring harnesses can also be made and used for the piccolo autopilots. Use the Cloud Cap document "Piccolo External Interface" to make custom wiring harnesses. The document lists the pin outs and their corresponding servo, IO, and comms designations for all of the piccolo autopilot units. Additionally extra sensors will require manual designation of their function in the Payload IO settings. Refer to Cloud Cap's "PccUsersGuide" pgs. 85-95 to see all of the available options for adding external sensors.

Cloud Cap provides documentation on some specific external add ons. The documents can be found in the Piccolo Docs\Hardware folder. There is a document for the JR level shifter board, magnetometer, Novatel DGPS, OAT sensor, servo gimbal cameras (not the TASE gimbal cameras, but cameras where the autopilot actually commands the camera servos itself), the deadman tach board (RPM sensor), and iridium (satellite) communications.

4.1. AVLEditor Aircraft Model

The Piccolo Simulator requires aircraft to be modeled using the AVLEditor. The AVLEditor software is installed from the "PiccoloInstaller.msi" installation file. AVLEditor should be located in the Cloud Cap folder under "Cloud Cap\Piccolo 2.x.x.x\AVLEditor\". Save AVLEditor models in the aircraft's folder.

Creating a model in AVLEditor consists of measuring aircraft geometry, determining cg location, importing airfoils, creating surfaces (wings, tails, fuselage shapes, etc.), creating control surfaces (ailerons, flaps, elevators, rudders, ruddervators, etc.), Xfoil analysis, and Vortex Lattice Settings.

Control Surfaces must be named according to their function; however, the names must also be identical to an actuator type for the Simulator to recognize them when it calculates the estimates for the Vehicle Gains file. For example if a control surface is supposed to be an aileron and it is not named "Aileron" then that control surface's contribution to roll when its deflected will not be included in the estimation of the Aircraft's Aileron Effectiveness and the number estimated in the Vehicle Gains file will be incorrect.

Once an aircraft has been created in AVLEditor the final step will be to run the AVL Analysis. AVL Analysis launches Cloud Caps modified version of AVL3.22, "avlcct". According to Cloud Cap the only difference is the format of exported data, so that the exported data can be loaded correctly by the Simulator. AVL Analysis will ask for a range of alphas and increment to perform an "Alpha Sweep". During an Alpha Sweep AVLEditor will run the aircraft model in avlcct at each angle of attack specified. At the end of each run AVLEditor will store all of the stability derivatives that AVL calculates in a text file called an "alpha".

The alpha file is the end result and purpose of using AVL. The alpha file will be used by the simulator to calculate initial estimates for Vehicle Gains, and coupled with the Simulator text file will dictate the aircraft dynamics in simulations. The accuracy of the simulations will depend on the accuracy of the model. A list and description of the stability derivatives can be found in the following cloud cap documents:

1) "Piccolo Simulator" pgs. 54-55 located in the Cloud Cap folder under "Cloud Cap\Piccolo 2.x.x.x\Piccolo Docs\Software".

2) “CreatingPiccoloAircraftModelsWithAVL” pgs. 20-21 located in the Piccolo folder under “Piccolo\Miscellaneous Documents\”.

4.1.1. Measuring Aircraft Geometry

Aircraft geometry is measured in accordance with AVLEditor axes. The positive x-direction points downstream, positive y-direction points starboard, and positive z-direction points upward. AVL doesn’t care where the reference point is; however, choose the center of the leading edge of the wing to be the reference point, (0,0,0), that all geometry is measured from as shown below in Figure 69.

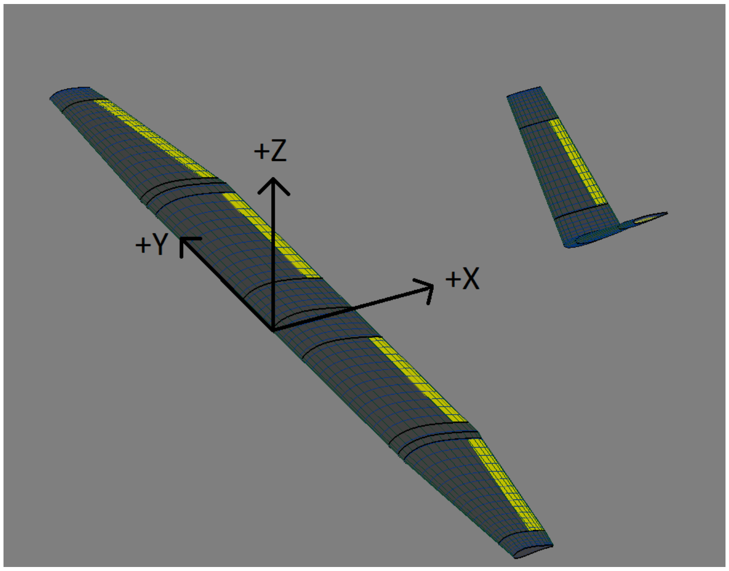


Figure 69 AVL Model Reference Point

AVLEditor uses sections to create surfaces. Measure and record an (x,y,z) location at the leading edge of each surface when the leading edge or trailing edge changes slope in the y-direction and whenever a control surface begins and ends. If the surface is symmetric along the y-axis, like most wings are, only points on the starboard side need to be measured.

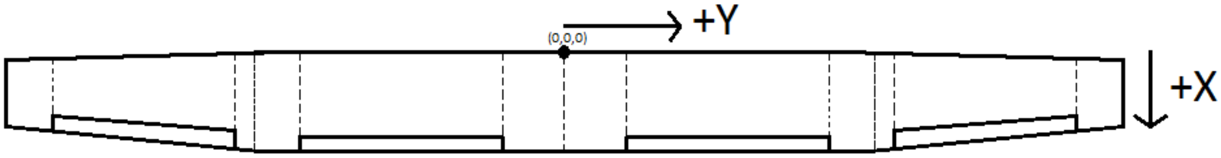
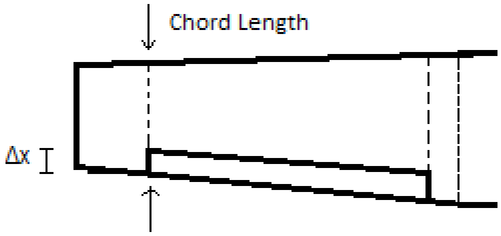


Figure 70 Measuring Sections

The length of each section, Δx, also needs to be measured. Δx of each section is defined as the Chord Length in AVLEditor.

At each section where there is a control surface measure the x length (Δx) of the control surface and record its "Chord Fraction". The chord fraction is the percent of the chord that is NOT a part of the control surface. Below further illustrate measuring the chord fraction.

Figure 71 Chord Fraction



$$Chord\ Fraction = 1 - \frac{\Delta x}{Chord\ Length}$$

Equation 12 Chord Fraction

Technically AVL uses dimensionless units of length that are defined in a "plane.mass" text file. Unfortunately when AVLEditor loads the plane mass file it does not actually use it. AVLEditor automatically assumes the units of length to be in meters regardless of the existence and content of the plane.mass file. Measure the aircraft's geometry in whatever units are most convenient; however, make sure they are converted to meters when being entered into AVLEditor.

4.1.2. Measuring Fuselage Surfaces

AVLEditor has a fuselage tool separate from creating surfaces called “Body Editor”. The Body Editor rarely works properly and can easily ruin models; therefore, the fuselage is to be modeled as two flat plates. One flat plate will represent the side view (xz plane) while the other will represent the bird’s eye view (xy plane). Use the following steps to measure fuselage surfaces:

1) Measure the 2-D shape of the side view of the fuselage. Measure an (x,z) point at each location that the leading or trailing edges change slope in the z-direction. Also measure the lengths of each section (x-direction). All the points of the side view should have y = 0.

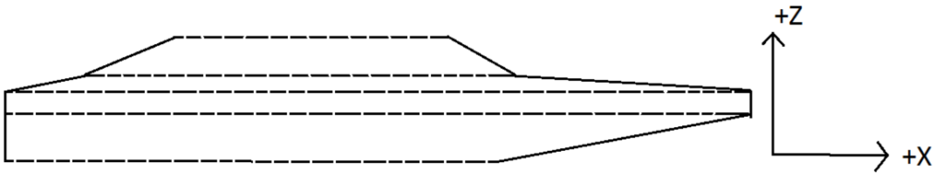


Figure 72 Fuselage Side View Surface

2) Measure the bird’s eye view. Similar to measuring surfaces measure an (x,y) point each time there is a change in slope in the y-direction. Also measure the lengths of each section (x-direction). All of the points of the bird’s eye view should have the same z – location (at the centerline of the fuselage).

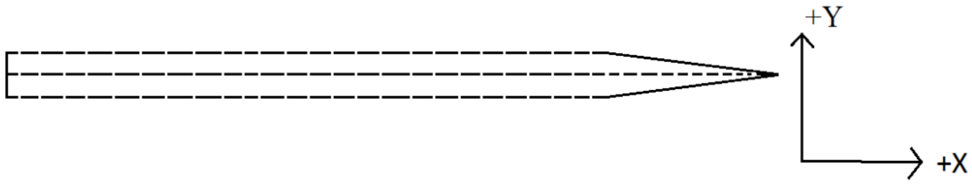


Figure 73 Fuselage Top View Surface

4.1.3. CG/Inertia Model

AVL needs to know the location of the Center of Gravity ($\overline{x}$, $\overline{y}$, $\overline{z}$) to properly estimate stability derivatives. The Mass Moments of Inertia (Ix, Iy, Iz) are needed for the aircraft's Simulator File and the Aircraft's Vehicle Gains.

This guide presents only one detailed way of estimating cg location and mass moment of inertia. There are experimental methods that can be used if desired, but none of them are included in this guide.

If it is desired to use Solid Works, or some experimental technique, skip 4.1.3.1 Excel Component Modeling; otherwise continue to Section 5.3.1. If you wish to use Solid Works the process is self-explanatory. Create a model of the aircraft, including all the servos, batteries, and other parts on the plane. Assign weights to each part and Solid Works will calculate CG and Inertia of the aircraft. Make sure the Inertia being used is mass moment of inertia (kg-m^2). The results must be in SI.

4.1.3.1. Excel Component Modeling

Component modeling consists of measuring the CG location of each individual item with respect to the aircraft axes, weighing each individual item, and measuring characteristic lengths of each item.

Component modeling sums up all of the local CG locations multiplied by their mass and divides by the total weight to determine the CG location of the aircraft in each axis.

$$\overline{x} = \frac{1}{M}\sum_{i=1}^{n}[\,\overline{x}_i * m_i] \quad \overline{y} = \frac{1}{M}\sum_{i=1}^{n}[\,\overline{y}_i * m_i] \quad \overline{z} = \frac{1}{M}\sum_{i=1}^{n}[\,\overline{z}_i * m_i]$$

Mass moment of inertia is calculated by estimating the mass moment of inertia of each individual piece with respect to its own axes, using parallel axis theorem to define the object’s inertia about the CG of the entire aircraft, and summing up the moment of inertia of all the individual objects.

$$R_x = \overline{x}_i - \overline{x} \quad R_y = \overline{y}_i - \overline{y} \quad R_z = \overline{z}_i - \overline{z}$$

$$I_x = I_{xx} + m\left({R_y}^2 + {R_z}^2\right) \quad I_y = I_{yy} + m\left({R_x}^2 + {R_z}^2\right) \quad I_z = I_{zz} + m\left({R_x}^2 + {R_y}^2\right)$$

$$I_x = \sum_{i=1}^{n} I_{x_i} \quad I_y = \sum_{i=1}^{n} I_{y_i} \quad I_z = \sum_{i=1}^{n} I_{z_i}$$

Mass moment of inertia of each object is estimated by assuming either a rectangular shape with constant density distribution, a cylinder with constant density distribution, or a point mass.

Objects that can be reasonably estimated as rectangles with constant density use the following equations:

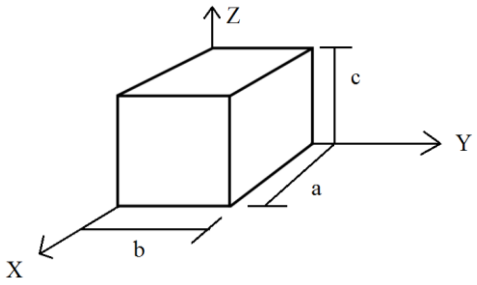


Figure 74 Mass Moment of Inertia of Rectangles

$$I_{xx} = \frac{1}{12} m * (b^2 + c^2) \quad I_{yy} = \frac{1}{12} m * (a^2 + c^2) \quad I_{zz} = \frac{1}{12} m * (a^2 + b^2)$$

Equation 13 Mass Moments of Inertia of Rectangles

Objects that can be reasonably estimated as cylinders with constant density use the following equations:

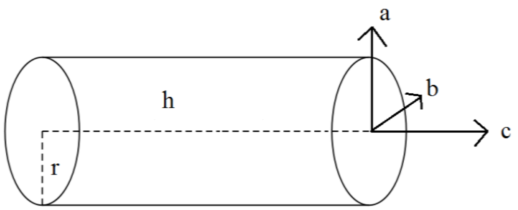


Figure 75 Mass Moment of Inertia of Cylinders

$$I_a = I_b = \frac{1}{12} m * (3r^2 + h^2) \quad I_c = \frac{1}{2} m * r^2$$

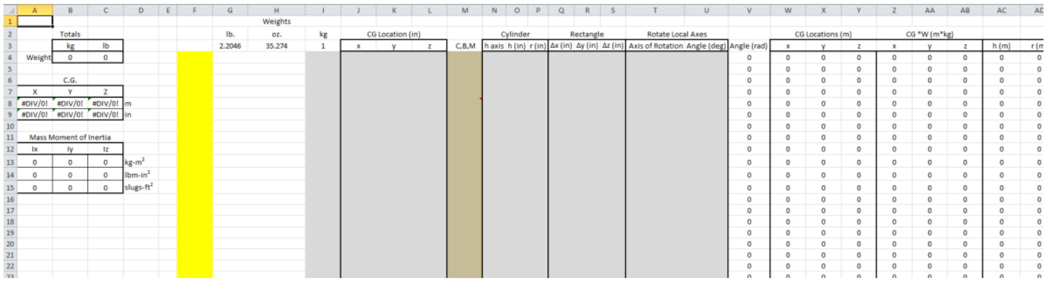
Equation 14 Mass Moments of Inertia of Cylinders

Point masses don’t have local inertias (Ixx, Iyy, Izz) and are calculated using:

$$I_x = \frac{1}{12} m\left({R_y}^2 + {R_z}^2\right) \quad I_y = \frac{1}{12} m\left({R_x}^2 + {R_z}^2\right) \quad I_z = \frac{1}{12} m\left({R_x}^2 + {R_y}^2\right)$$

Equation 15 Mass Moments of Inertia of Point Masses

An Excel Spreadsheet has been created and formatted for Component Modeling called “CGInertiaModel”. CGInertiaModel is located in the “New Aircraft” folder on the piccolo network drive under “P: \New Aircraft”, and should exist in the aircraft folder of the aircraft being modeled. The spreadsheet will automatically calculate the center of gravity and mass moment of inertia of the entire configuration along with the individual mass moment of inertias of each individual component.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1								Weights								
2		Totals					lb.	oz.	kg	CG Location (in)				Cylinder		
3		kg	lb				2.2046	35.274	1	x	y	z	C,B,M	h axis	h (in)	r (in)
4	Weight	0	0													
5																
6		C.G.														
7	X	Y	Z													
8	#DIV/0!	#DIV/0!	#DIV/0!	m												
9	#DIV/0!	#DIV/0!	#DIV/0!	in												
10																
11	Mass Moment of Inertia															
12	Ix	Iy	Iz													
13	0	0	0	kg-m^2												
14	0	0	0	lbm-in^2												
15	0	0	0	slugs-ft^2												
16																
17																
18																
19																
20																
21																
22																

	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1														
2	Rectangle			Rotate Local Axes			CG Locations (m)			CG *W (m*kg)				
3	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)	Angle (rad)	x	y	z	x	y	z	h (m)	r (m
4						0	0	0	0	0	0	0	0	0
5						0	0	0	0	0	0	0	0	0
6						0	0	0	0	0	0	0	0	0
7						0	0	0	0	0	0	0	0	0
8						0	0	0	0	0	0	0	0	0
9						0	0	0	0	0	0	0	0	0
10						0	0	0	0	0	0	0	0	0
11						0	0	0	0	0	0	0	0	0
12						0	0	0	0	0	0	0	0	0
13						0	0	0	0	0	0	0	0	0
14						0	0	0	0	0	0	0	0	0
15						0	0	0	0	0	0	0	0	0
16						0	0	0	0	0	0	0	0	0
17						0	0	0	0	0	0	0	0	0
18						0	0	0	0	0	0	0	0	0
19						0	0	0	0	0	0	0	0	0
20						0	0	0	0	0	0	0	0	0
21						0	0	0	0	0	0	0	0	0
22						0	0	0	0	0	0	0	0	0

Figure 76 CGInertiaModel Format

Figure 76 depicts a portion of the formatted spreadsheet. Gray areas are designated for user inputs. The yellow column is used by the spreadsheet to determine whether or not a component should be used in the center of gravity and mass moment of inertia calculations. If a cell contains a “y” then the spreadsheet will include the corresponding component in the calculations. The column exists to provide the user the ability to easily add and remove different components from the center of gravity and inertia calculations without having to edit or remove the entire component and its data. The brown column determines how the local mass moment of inertia of each row is calculated based on the value of each individual cell. This column provides the user with the ability to easily classify each component as a specific shape or point mass. The results are boxed in at the far left of the spreadsheet. All white cells that have been boxed in are calculations performed automatically by the spreadsheet.

If an object is at an angle around one axis, which would cause two axes of the object to not be parallel to the aircraft axes, the spreadsheet will use the following equations to adjust the local inertias before using the parallel axis theorem:

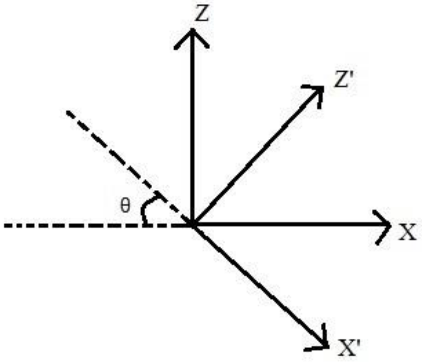


Figure 77 Y-Axis Rotation

$$I_{xx} = I'_{xx} * \cos\theta + I'_{zz} * \sin\theta$$

$$I_{zz} = I'_{xx} * \sin\theta + I'_{zz} * \cos\theta$$

Equation 16 Mass Moments of Inertia Y-Axis Rotation

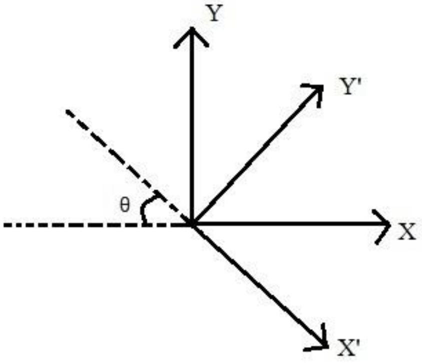


Figure 78 Z – Axis Rotation

$$I_{xx} = I'_{xx} * \cos\theta + I'_{yy} * \sin\theta$$

$$I_{yy} = I'_{xx} * \sin\theta + I'_{yy} * \cos\theta$$

Equation 17 Mass Moments of Inertia Z-Axis Rotation

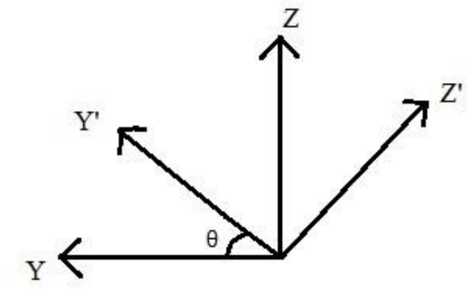
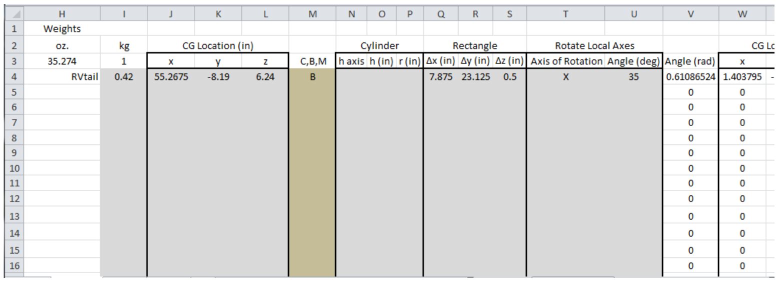


Figure 79 X – Axis Rotation

$$I_{yy} = I'_{yy} * \cos\theta + I'_{zz} * \sin\theta$$

$$I_{zz} = I'_{yy} * \sin\theta + I'_{zz} * \cos\theta$$

Equation 18 Mass Moment of Inertia X – Axis Rotation



	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
1	Weights																
2	oz.	kg	CG Location (in)				Cylinder			Rectangle			Rotate Local Axes			CG Lc	
3	35.274	1	x	y	z	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)	Angle (rad)	x	
4	RVtail	0.42	55.2675	-8.19	6.24	B				7.875	23.125	0.5	X	35	0.61086524	1.403795	-
5															0	0	
6															0	0	
7															0	0	
8															0	0	
9															0	0	
10															0	0	
11															0	0	
12															0	0	
13															0	0	
14															0	0	
15															0	0	
16															0	0	

Figure 80 CGInertiaModel Axis Rotation

As shown in the example above, Figure 80, the spreadsheet will adjust the local inertias based on the user input in the “Rotate Local Axes” section. The user will need to input the axis of rotation and the angle in degrees.

Determine the components that need to be modeled. Objects that are relatively small in mass can be excluded. Use the Excel Spreadsheet to record measurements of each object.

For each object:

1) Weigh the object and enter its mass and name in the spreadsheet

2) Measure the location of the center of gravity with respect to the aircraft axes in inches (the center of the wings at the leading edge should be (0,0,0)) as shown below in Figure 81. The spreadsheet will automatically convert cg measurements to meters.

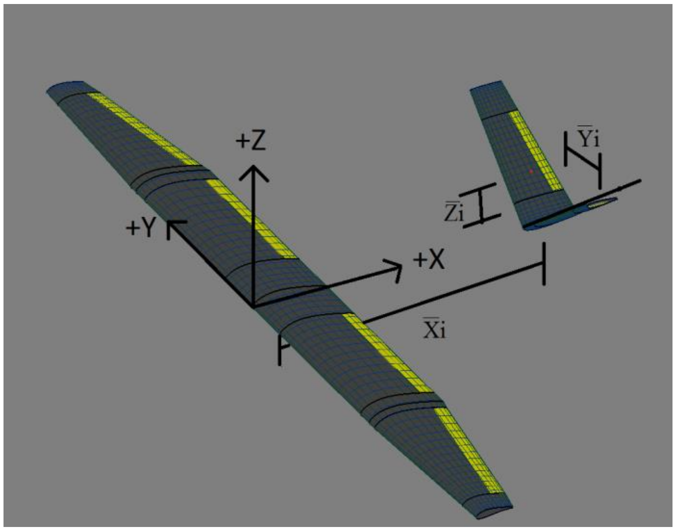
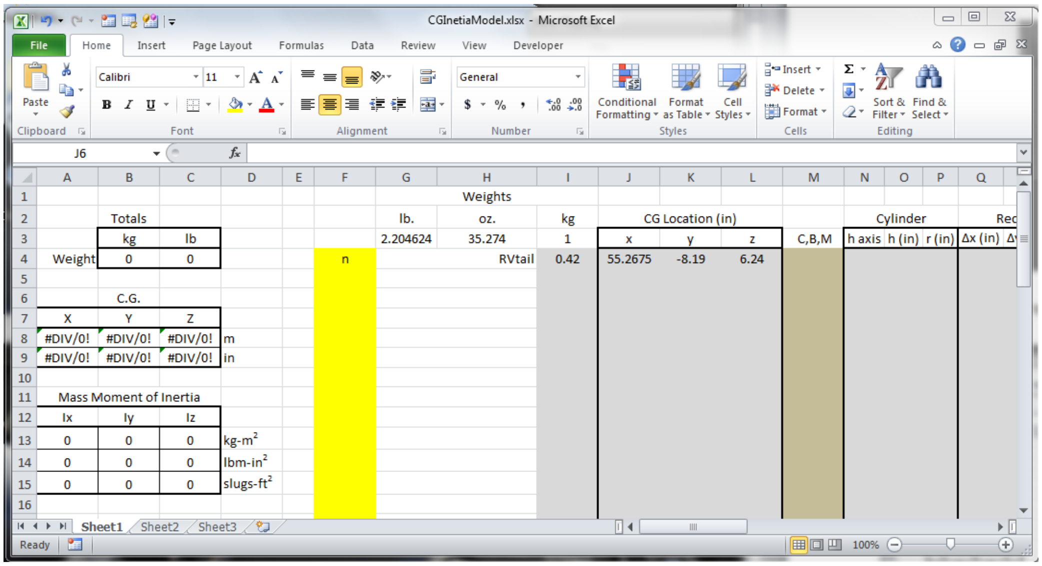


Figure 81 Component CG Location Measurements

Estimating the CG location of each object is up to the judgment of the user. The CG can be estimated as the center of an object if the object's mass is nearly evenly distributed. Make sure to do servos and surfaces separately.

3) Input the values in the spreadsheet in their appropriate locations.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
1								Weights										
2		Totals					lb.	oz.	kg	CG Location (in)				Cylinder			Rec	
3		kg	lb				2.204624	35.274	1	x	y	z	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δ
4	Weight	0	0			n		RVtail	0.42	55.2675	-8.19	6.24						
5																		
6		C.G.																
7	X	Y	Z															
8	#DIV/0!	#DIV/0!	#DIV/0!	m														
9	#DIV/0!	#DIV/0!	#DIV/0!	in														
10																		
11	Mass Moment of Inertia																	
12	Ix	Iy	Iz															
13	0	0	0	kg-m^2														
14	0	0	0	lbm-in^2														
15	0	0	0	slugs-ft^2														
16																		

Figure 82 Component Weight and CG Input

4) If the object can be modeled as a rectangle for moment of inertia go to a). If the object can be modeled as a cylinder go to b). If the object must be modeled as a point mass go to c).

a) Measure the lengths of the object in the x,y, and z directions (inches). Make sure the local axes are parallel to their corresponding aircraft axes. In Figure 83 below the Horizontal Tail was estimated as a rectangle. Δx was calculated as the mean chord of the tail.

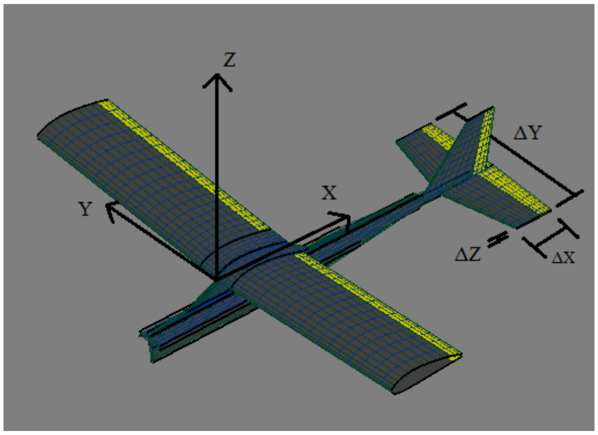


Figure 83 Component Modeling as a Rectangle

If the local axes are not parallel to the aircraft's AVLEditor axes then the local inertias will have to be adjusted before using parallel axis theorem. Measure Δx, Δy,Δz in their local axes, but make sure that the axes positive directions are oriented in accordance with the aircraft AVLEditor axes.

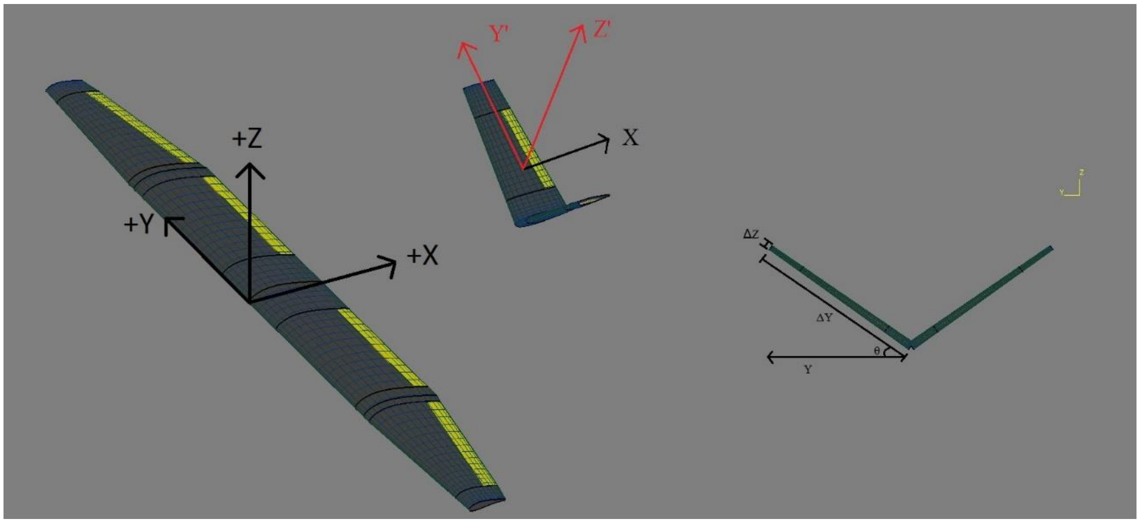
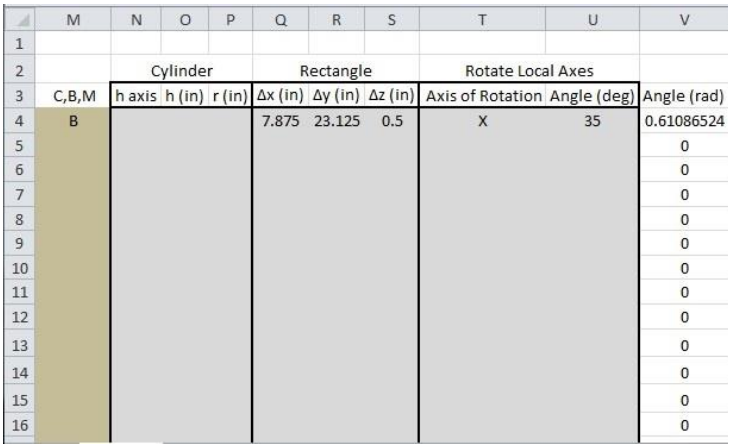


Figure 84 Component Model X-Axis Rotation Example

In the Excel Spreadsheet Type a "B" in the brown column (column "C,B,M"), and input the measured lengths in "Rectangle". Continue to 5) when finished.

If the axes need to be rotated input the axis of rotation and angle (deg) in the "Rotate Local Axes" section. In Figure 85 below a right Vtail is estimated as a rectangle and rotated 35 degrees about the x-axis.



	M	N	O	P	Q	R	S	T	U	V
1										
2		Cylinder			Rectangle			Rotate Local Axes		
3	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)	Angle (rad)
4	B				7.875	23.125	0.5	X	35	0.61086524
5										0
6										0
7										0
8										0
9										0
10										0
11										0
12										0
13										0
14										0
15										0
16										0

Figure 85 Component Model Example Rectangle Entry

b) Measure the radius I and length (h) in inches. Input the values in the spreadsheet in the “Cylinder” area.

Type “C” in the brown column “C,B,M” for cylinder. Under “h axis” input the aircraft AVLEditor axis that the length of the cylinder travels along (x,y, or z). The spreadsheet will automatically adjust the equations of local inertia for the orientation of cylinder.

As an example the following describes the addition of a wing spar to a cg/inertia model.

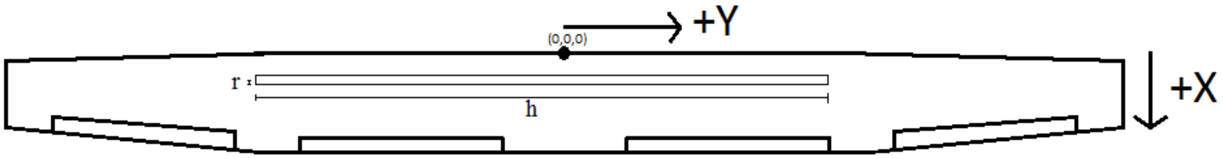
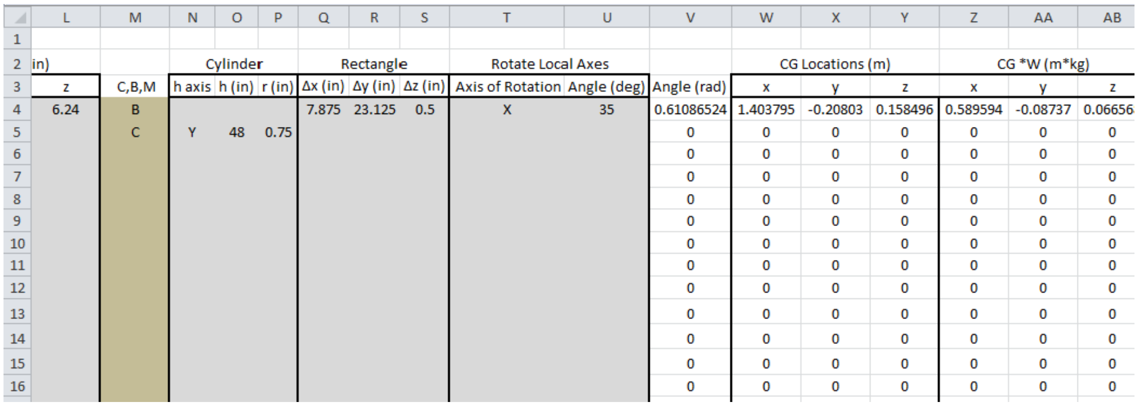


Figure 86 Wing Spar Inertia Example

With “h” traveling along the y axis the spreadsheet will calculate inertias using Equation 14 and setting $I_{xx} = I_a \ \ I_{yy} = I_c \ \ I_{zz} = I_b$.

$$I_{xx} = I_{zz} = \frac{1}{12} m * (3r^2 + h^2) \quad I_{yy} = \frac{1}{2} m * r^2$$

If the axes of the cylinder are not parallel to the aircraft AVLEditor axes use the steps detailed above in part a) to rotate the axes.



	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1																	
2	in)		Cylinder			Rectangle			Rotate Local Axes			CG Locations (m)			CG *W (m*kg)		
3	z	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)	Angle (rad)	x	y	z	x	y	z
4	6.24	B				7.875	23.125	0.5	X	35	0.61086524	1.403795	-0.20803	0.158496	0.589594	-0.08737	0.06656
5		C	Y	48	0.75						0	0	0	0	0	0	0
6											0	0	0	0	0	0	0
7											0	0	0	0	0	0	0
8											0	0	0	0	0	0	0
9											0	0	0	0	0	0	0
10											0	0	0	0	0	0	0
11											0	0	0	0	0	0	0
12											0	0	0	0	0	0	0
13											0	0	0	0	0	0	0
14											0	0	0	0	0	0	0
15											0	0	0	0	0	0	0
16											0	0	0	0	0	0	0

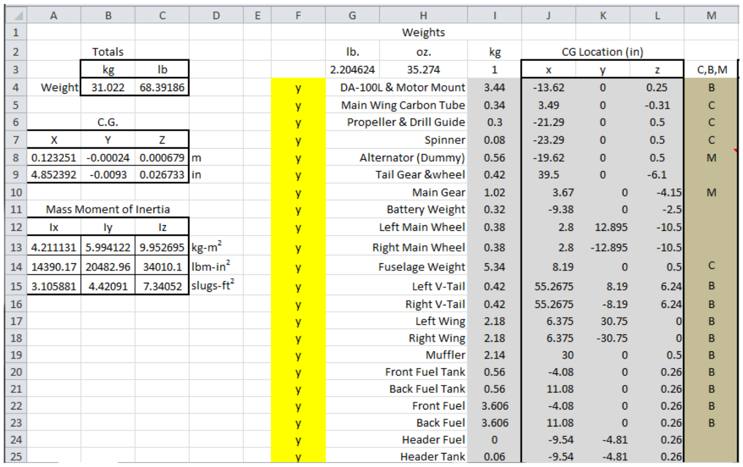
Figure 87 Wing Spar Inertia Example Input

c) Type an “M” in the brown column “C,B,M” to designate a component’s inertia to be modeled as a point mass. The local inertias will be calculated by the spreadsheet using Equation 15.

5) Repeat Steps 1-4 until all the objects have been entered into the spreadsheet.

6) Place “y” into the yellow column next to each object you want included in the calculations.

7) The results will be displayed at the far left of the spreadsheet, as depicted in Figure 88.

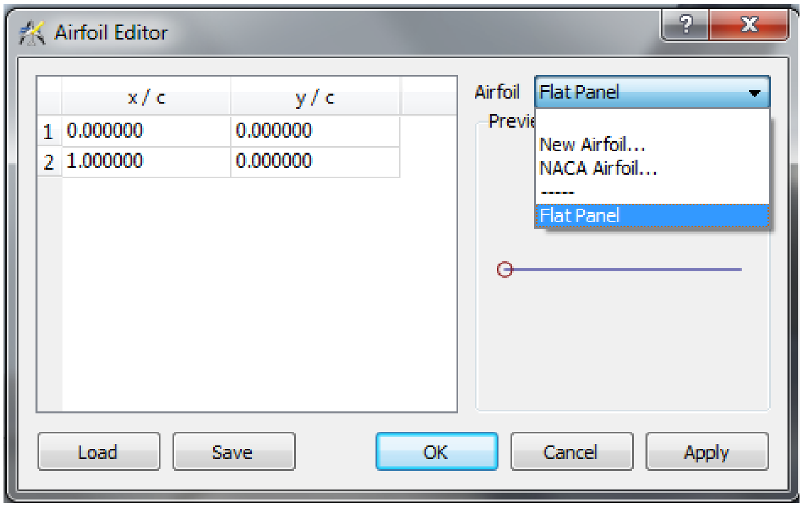


	A	B	C	D	E	F	G	H	I	J	K	L	M
1								Weights					
2		Totals					lb.	oz.	kg		CG Location (in)		
3		kg	lb				2.204624	35.274	1	x	y	z	C,B,M
4	Weight	31.022	68.39186			y		DA-100L & Motor Mount	3.44	-13.62	0	0.25	B
5						y		Main Wing Carbon Tube	0.34	3.49	0	-0.31	C
6		C.G.				y		Propeller & Drill Guide	0.3	-21.29	0	0.5	C
7	X	Y	Z			y		Spinner	0.08	-23.29	0	0.5	C
8	0.123251	-0.00024	0.000679	m		y		Alternator (Dummy)	0.56	-19.62	0	0.5	M
9	4.852392	-0.0093	0.026733	in		y		Tail Gear &wheel	0.42	39.5	0	-6.1	
10						y		Main Gear	1.02	3.67	0	-4.15	M
11		Mass Moment of Inertia				y		Battery Weight	0.32	-9.38	0	-2.5	
12	Ix	Iy	Iz			y		Left Main Wheel	0.38	2.8	12.895	-10.5	
13	4.211131	5.994122	9.952695	kg-m^2		y		Right Main Wheel	0.38	2.8	-12.895	-10.5	
14	14390.17	20482.96	34010.1	lbm-in^2		y		Fuselage Weight	5.34	8.19	0	0.5	C
15	3.105881	4.42091	7.34052	slugs-ft^2		y		Left V-Tail	0.42	55.2675	8.19	6.24	B
16						y		Right V-Tail	0.42	55.2675	-8.19	6.24	B
17						y		Left Wing	2.18	6.375	30.75	0	B
18						y		Right Wing	2.18	6.375	-30.75	0	B
19						y		Muffler	2.14	30	0	0.5	B
20						y		Front Fuel Tank	0.56	-4.08	0	0.26	B
21						y		Back Fuel Tank	0.56	11.08	0	0.26	B
22						y		Front Fuel	3.606	-4.08	0	0.26	B
23						y		Back Fuel	3.606	11.08	0	0.26	B
24						y		Header Fuel	0	-9.54	-4.81	0.26	
25						y		Header Tank	0.06	-9.54	-4.81	0.26	

Figure 88 Component Model Results

4.1.4. Creating Airfoils

AVLEditor manages airfoils using “Airfoil Editor”. Every surface created in AVLEditor is initially defined as a “Flat Panel” by default.



Airfoil Editor

	x / c	y / c
1	0.000000	0.000000
2	1.000000	0.000000

Airfoil: Flat Panel

Figure 89 Airfoil Editor

Airfoil Editor presents three different methods for loading airfoils into an AVLEditor model.

1) The Airfoil Editor has the built in capability to automatically generate 4-Digit Series NACA Airfoils.

2) The Airfoil Editor can load airfoils from airfoil files.

Accepted airfoil file extensions are ".txt", ".dat", and notepad files with no extension.

    1) The Airfoil Editor allows manually creating airfoil files by entering each point that lies on an airfoil profile (x/c, y/c).

Method #1 is the easiest method even though it does have a few glitches. Generating NACA airfoils avoids a number of formatting issues that can occur with the other two methods. If the airfoil is not an NACA 4-Digit Series airfoil, then methods #2, or #3 must be used.

Open the editor by clicking the "Edit Airfoils" airfoil symbol in the toolbar as illustrated in Figure 90. Go to Section 4.1.4.1 for method #1, Section 4.1.4.2 for method #2, or Section 4.1.4.3 for method #3.

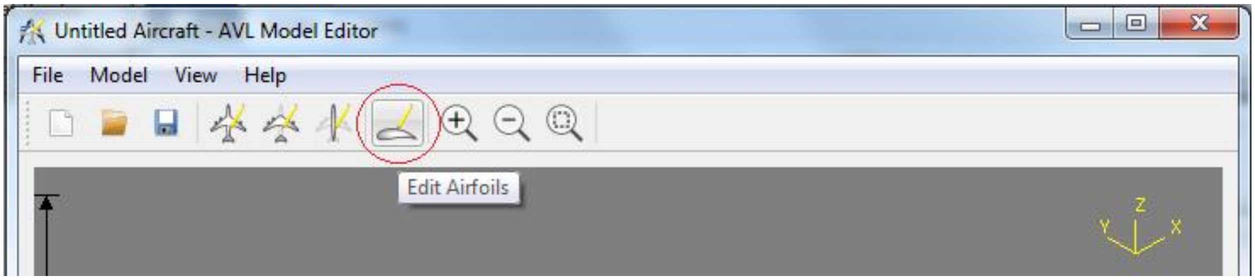


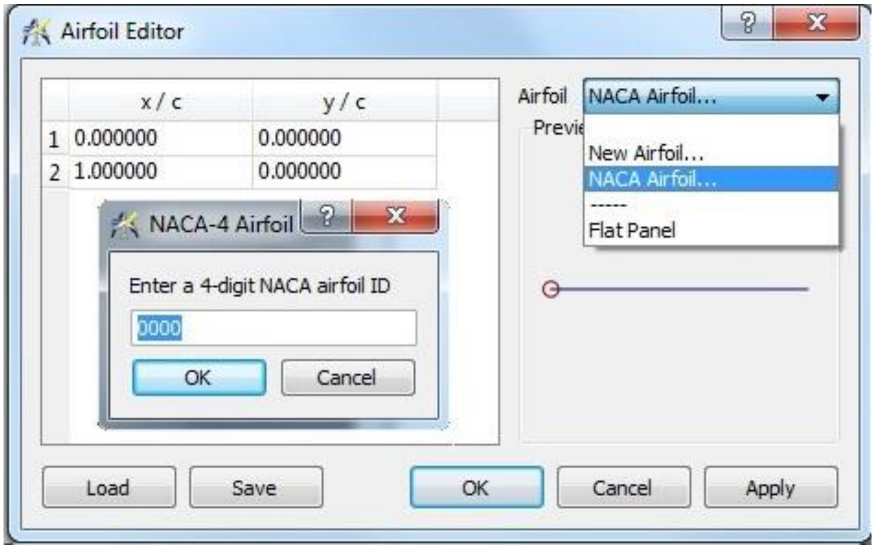
Figure 90 Open Airfoil Editor

4.1.4.1. Generating NACA airfoils with Airfoil Editor

1) Open Airfoil Editor and click the airfoil drop down menu next to "Airfoil".

2) Select "NACA Airfoil".

An input box will appear asking for the 4-digit code of the NACA airfoil.



	x / c	y / c
1	0.000000	0.000000
2	1.000000	0.000000

Enter a 4-digit NACA airfoil ID

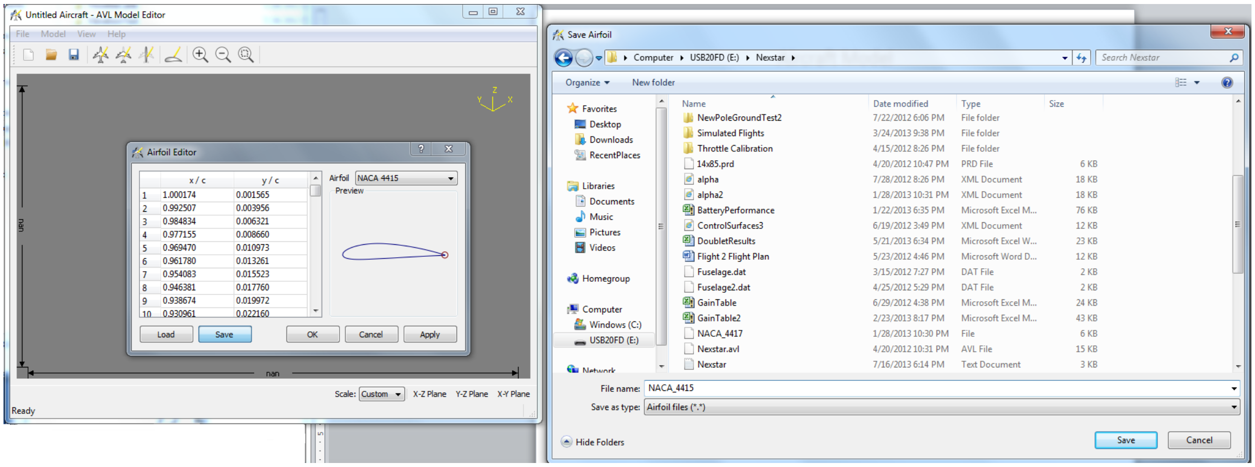
0000

Figure 91 Generate NACA Airfoil

Make sure to follow the following steps exactly as described.

3) Enter the 4-digit code and click OK.

4) Click "Save" in the airfoil editor to save the points it just generated into an airfoil file. Name the airfoil appropriately. The name can include spaces and underscores i.e. (NACA_4415 or NACA 4415). Make sure to save the airfoil file in the same directory that the AVL model will be saved. They have to be saved in the same directory.



	x / c	y / c
1	1.000174	0.001565
2	0.992507	0.003956
3	0.984834	0.006321
4	0.977155	0.008660
5	0.969470	0.010973
6	0.961780	0.013261
7	0.954083	0.015523
8	0.946381	0.017760
9	0.938674	0.019972
10	0.930961	0.022160

Airfoil: NACA 4415

Preview

Name	Date modified	Type	Size
NewPoleGroundTest2	7/22/2012 6:06 PM	File folder	
Simulated Flights	3/24/2013 9:38 PM	File folder	
Throttle Calibration	4/15/2012 8:26 PM	File folder	
14x85.prd	4/20/2012 10:47 PM	PRD File	6 KB
alpha	7/28/2012 8:26 PM	XML Document	18 KB
alpha2	1/28/2013 10:31 PM	XML Document	18 KB
BatteryPerformance	1/22/2013 6:35 PM	Microsoft Excel M...	76 KB
ControlSurfaces3	6/19/2012 3:49 PM	XML Document	12 KB
DoubletResults	5/21/2013 6:34 PM	Microsoft Excel W...	23 KB
Flight 2 Flight Plan	5/23/2012 4:46 PM	Microsoft Word D...	12 KB
Fuselage.dat	3/15/2012 7:27 PM	DAT File	2 KB
Fuselage2.dat	4/25/2012 5:29 PM	DAT File	2 KB
GainTable	6/29/2012 4:38 PM	Microsoft Excel M...	24 KB
GainTable2	2/23/2013 8:17 PM	Microsoft Excel M...	43 KB
NACA_4417	1/28/2013 10:30 PM	File	6 KB
Nexstar.avl	4/20/2012 10:31 PM	AVL File	15 KB
Nexstar	7/16/2013 6:14 PM	Text Document	3 KB

File name: NACA_4415

Save as type: Airfoil files (*.*)

Figure 92 Save Generated NACA Airfoil

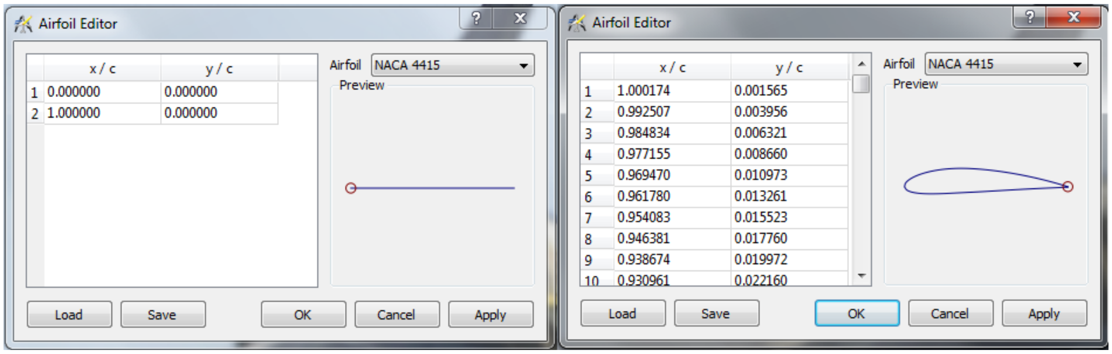
5) After the airfoil has been saved click “Cancel” to close the Airfoil Editor and completely close AVLEditor without saving anything.

6) Open AVLEditor.

7) Open Airfoil Editor.

8) Click “Load” and load the airfoil file that was just created.

9) Click the dropdown menu and select the airfoil that was just loaded to view the airfoil preview.



	x / c	y / c
1	0.000000	0.000000
2	1.000000	0.000000

	x / c	y / c
1	1.000174	0.001565
2	0.992507	0.003956
3	0.984834	0.006321
4	0.977155	0.008660
5	0.969470	0.010973
6	0.961780	0.013261
7	0.954083	0.015523
8	0.946381	0.017760
9	0.938674	0.019972
10	0.930961	0.022160

Figure 93 Load Generated NACA Airfoil

10) Click “OK”

The reason for not saving the airfoil the first time it was generated is because the airfoil editor won’t reference the airfoil file unless the airfoil was loaded from a file through the editor. Even though the Airfoil Editor created that airfoil file it will be referencing the airfoil points from internal memory and not the file itself. In and of itself that would not be a problem; however the problem arrives later when running Xfoil analysis. Xfoil analysis needs an airfoil file to load, and if Airfoil Editor did not load an airfoil from an airfoil file Xfoil analysis will attempt to load the airfoil from Airfoil Editor’s internal memory and freeze AVLEditor. The only way to be able to successfully run Xfoil analysis is if all the airfoils were loaded through the Airfoil Editor from an airfoil file.

11) Repeat Steps 1-11 for each airfoil then continue to Section 4.1.5.

4.1.4.2. Loading an airfoil file

The airfoil editor will load airfoil files from notepad files with extensions “.dat”, “.txt”, and “.”. If the airfoils on the aircraft are not NACA 4-digit series there are plenty of options available for the user to generate airfoil files without needing to manually measure points. Two such possibilities are listed below:

1) Webpages such as http://www.ae.illinois.edu/m-selig/ads/coord_database.html#C have large databases of airfoil “.dat” files

2) Profili also has a large database of airfoils that can be exported to “.dat” or “.txt” files. Profili also has the capability of creating points from photos, or scanned tracings of airfoil shapes.

In order to properly load an airfoil file the airfoil file must have a particular format. The requirements are listed in the following:

1) The first line needs to be the name of the airfoil.

2) The following lines should have two columns.

3) The first column should be “x/c”.

4) The second column should be “y/c”.

5) The first point should begin at the top layer of the trailing edge of the airfoil, and continue all the way around the airfoil, counterclockwise, to the bottom layer of the trailing edge.

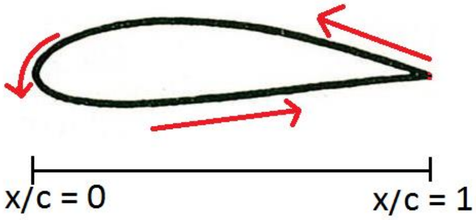
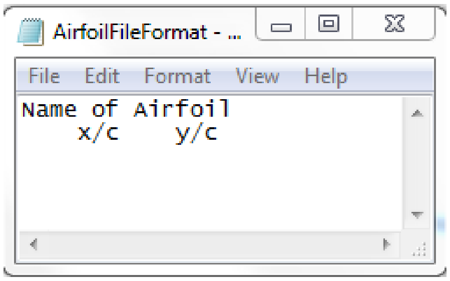


Figure 94 Airfoil File Point Order

The order of x/c should be from 1.0 to 0.0 and back to 1.0.



```
Name of Airfoil
    x/c     y/c
```

Figure 95 Airfoil File Format

Some airfoil file formats draw out the top and bottom layers separately with two sets of rows, both starting at x/c=0.

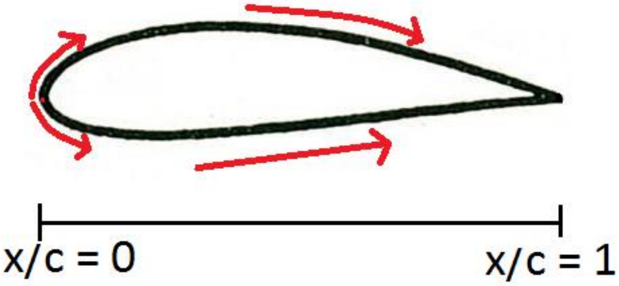


Figure 96 Incorrect Airfoil File Point Order

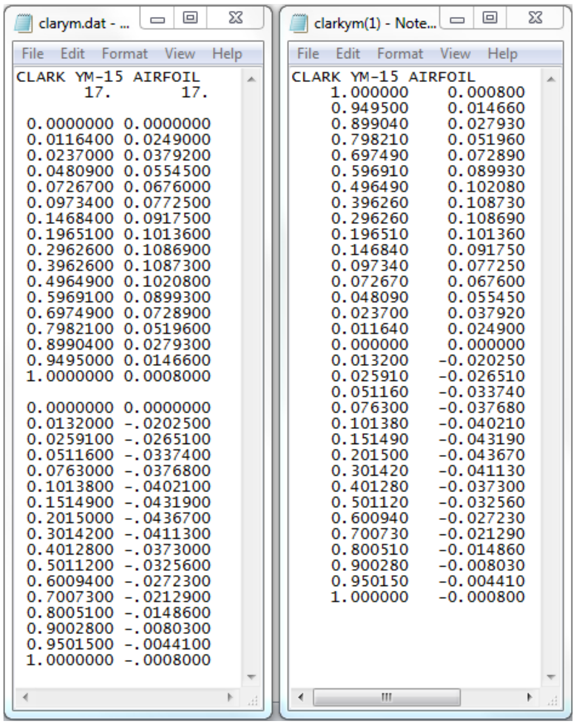
If the airfoil file format does not match the format depicted in Figure 94 continue through the following steps; otherwise, proceed to Step 4).

1) If the format is not correct open the airfoil file with notepad

2) Re-format the file to match the proper format.

   Make sure to remove any extra rows that aren't x/c, y/c points, or the name of the airfoil, and make sure there is only one row of (0,0). Save the modified file as a text file, ".txt".

3) Example:



```
CLARK YM-15 AIRFOIL
        17.        17.

 0.0000000 0.0000000
 0.0116400 0.0249000
 0.0237000 0.0379200
 0.0480900 0.0554500
 0.0726700 0.0676000
 0.0973400 0.0772500
 0.1468400 0.0917500
 0.1965100 0.1013600
 0.2962600 0.1086900
 0.3962600 0.1087300
 0.4964900 0.1020800
 0.5969100 0.0899300
 0.6974900 0.0728900
 0.7982100 0.0519600
 0.8990400 0.0279300
 0.9495000 0.0146600
 1.0000000 0.0008000

 0.0000000 0.0000000
 0.0132000 -.0202500
 0.0259100 -.0265100
 0.0511600 -.0337400
 0.0763000 -.0376800
 0.1013800 -.0402100
 0.1514900 -.0431900
 0.2015000 -.0436700
 0.3014200 -.0411300
 0.4012800 -.0373000
 0.5011200 -.0325600
 0.6009400 -.0272300
 0.7007300 -.0212900
 0.8005100 -.0148600
 0.9002800 -.0080300
 0.9501500 -.0044100
 1.0000000 -.0008000
```

```
CLARK YM-15 AIRFOIL
     1.000000     0.000800
     0.949500     0.014660
     0.899040     0.027930
     0.798210     0.051960
     0.697490     0.072890
     0.596910     0.089930
     0.496490     0.102080
     0.396260     0.108730
     0.296260     0.108690
     0.196510     0.101360
     0.146840     0.091750
     0.097340     0.077250
     0.072670     0.067600
     0.048090     0.055450
     0.023700     0.037920
     0.011640     0.024900
     0.000000     0.000000
     0.013200    -0.020250
     0.025910    -0.026510
     0.051160    -0.033740
     0.076300    -0.037680
     0.101380    -0.040210
     0.151490    -0.043190
     0.201500    -0.043670
     0.301420    -0.041130
     0.401280    -0.037300
     0.501120    -0.032560
     0.600940    -0.027230
     0.700730    -0.021290
     0.800510    -0.014860
     0.900280    -0.008030
     0.950150    -0.004410
     1.000000    -0.000800
```

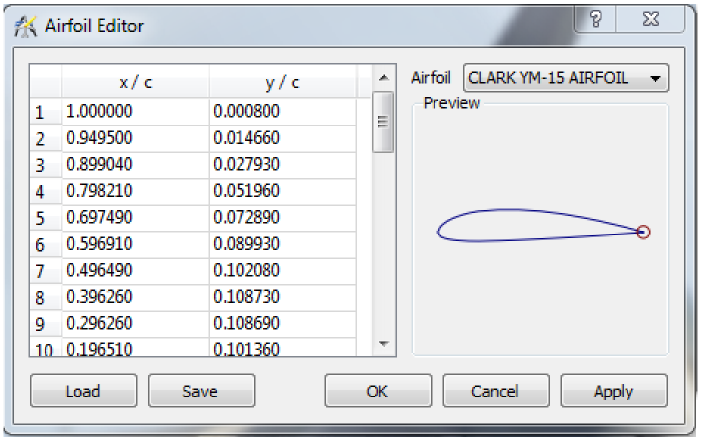
Figure 97 Example Airfoil File Reformat

Figure 97 shows the same airfoil formatted in two different ways. The airfoil file on the left is a ".dat" airfoil file with the same formatting scheme as the airfoil that was depicted in Figure 96. The airfoil file on the right has been modified to the acceptable format shown in Figure 94. Notice the points weren't only re-arranged, but also one row of (0,0) was deleted, and the second row was deleted. Spacing and tabs aren’t a problem as long as they are consistent throughout the file.

4) Open Airfoil Editor.

5) Click “Load” and load the airfoil file that was just modified.

6) Click the dropdown menu and select the airfoil that was just loaded to view the airfoil preview.



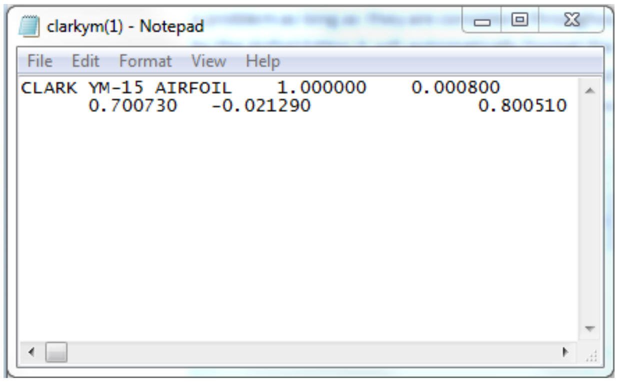
Airfoil CLARK YM-15 AIRFOIL

	x / c	y / c
1	1.000000	0.000800
2	0.949500	0.014660
3	0.899040	0.027930
4	0.798210	0.051960
5	0.697490	0.072890
6	0.596910	0.089930
7	0.496490	0.102080
8	0.396260	0.108730
9	0.296260	0.108690
10	0.196510	0.101360

Figure 98 Preview Modified Airfoil File

7) Click “Save”. Save the airfoil in the directory that the AVLEditor model will be created in.

The airfoil editor will automatically change the spacing of the rows and columns in the airfoil file. “Enters” for new lines are removed and replaced with spaces. For each new line Airfoil Editor will remove “Enter” and instead use 16 “spaces”. Figure 99 shows the airfoil file “clarkym(1).txt” from Figure 97 after it had been loaded and saved through the Airfoil Editor.



```
CLARK YM-15 AIRFOIL     1.000000     0.000800
          0.700730    -0.021290             0.800510
```

Figure 99 Airfoil File Format Change

8) Repeat the previous steps as necessary for any remaining airfoils that are on the aircraft.

9) Continue to Section 4.1.5.

4.1.4.3. Manually Create an Airfoil

As an option of last resort measure x/c and y/c points on the airfoil manually. One way would be to trace the airfoil onto a sheet of paper. Draw the chord line (straight line from leading edge tip to trailing edge tip). Determine the x intervals for measuring points based on the number of points desired. The number of points to measure is up to the discretion of the user. AVLEditor generated airfoil files have 257 upper and 257 lower points; however, the example airfoil file in Section 5.4.2 contained only 33 points total. Don't go any lower than 33. Further on in the process the number of panel nodes on the airfoils will be increased.

Measure the distance from the chord line to the upper and lower boundaries along with their x distance at each interval. Divide each measurement by chord length to get x/c and y/c.

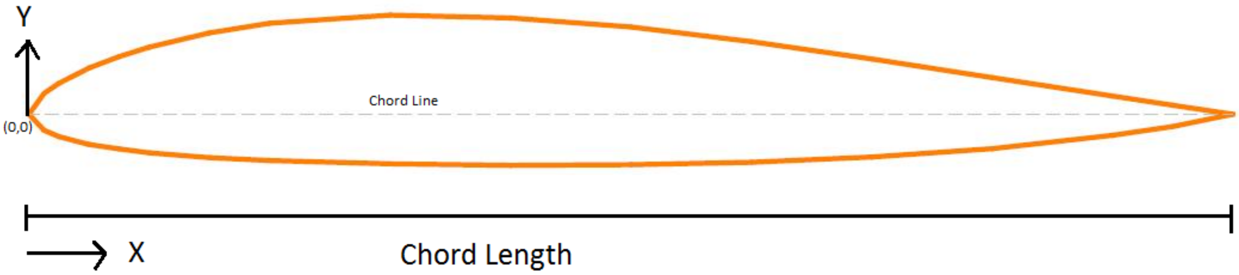


Figure 100 Manual Airfoil Point Measurement

The points can be input directly into the Airfoil Editor, or into a notepad text file with the proper formatting. Refer to Section 4.1.4.2 for proper formatting.

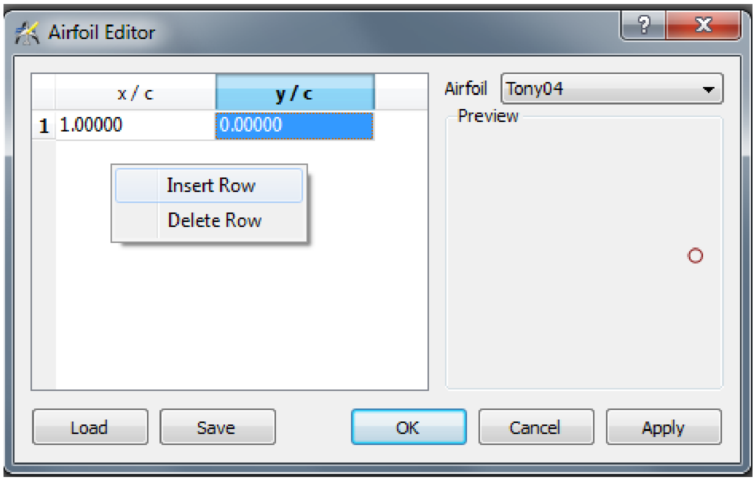
1) Open Airfoil Editor.

2) Click on the airfoil drop down menu, and select “New Airfoil”.

3) Right click in the empty table to insert or delete rows, and input the appropriate points.

4) Make sure to use the proper formatting from Section 4.1.4.2.

Start from x/c = 1, go all the way around the top to the bottom and back to x/c = 1 again.



	x / c	y / c
1	1.00000	0.00000

Airfoil Tony04

Figure 101 Manual Airfoil Point Entry

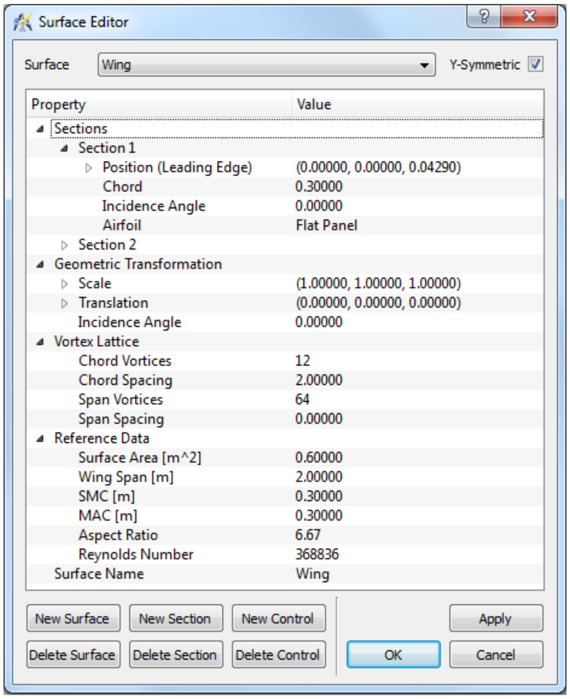
5) Click “Save” when finished. Make sure to save the airfoil file in the same directory that the aircraft AVLEditor model will be saved in.

6) Repeat the previous steps as necessary for any remaining airfoils that are on the aircraft.

7) Continue to Section 4.1.5.

4.1.5. Creating Surfaces in AVLEditor

Surface Editor is used to create surfaces (Wings, Tails, and Fuselages), surface sections, and control surfaces (Ailerons, Elevators, Rudders).



Surface Editor

Surface: Wing

Y-Symmetric ☑

Property	Value
◢ Sections	
◢ Section 1	
▷ Position (Leading Edge)	(0.00000, 0.00000, 0.04290)
Chord	0.30000
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Section 2	
◢ Geometric Transformation	
▷ Scale	(1.00000, 1.00000, 1.00000)
▷ Translation	(0.00000, 0.00000, 0.00000)
Incidence Angle	0.00000
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	2.00000
Span Vortices	64
Span Spacing	0.00000
◢ Reference Data	
Surface Area [m^2]	0.60000
Wing Span [m]	2.00000
SMC [m]	0.30000
MAC [m]	0.30000
Aspect Ratio	6.67
Reynolds Number	368836
Surface Name	Wing

Surface Editor is also used for assigning airfoils to sections, setting incidence angles, and editing vortex lattice settings on each surface. Surface Editor also calculates Reference Data of each surface such as Surface Area, Wing Span, Aspect Ratio, Reynolds Number, and Mean Aerodynamic Chord.

4.1.5.1. Create New Surfaces

Open AVLEditor and click “Edit Surfaces” in the toolbar to open the Surface Editor and begin.

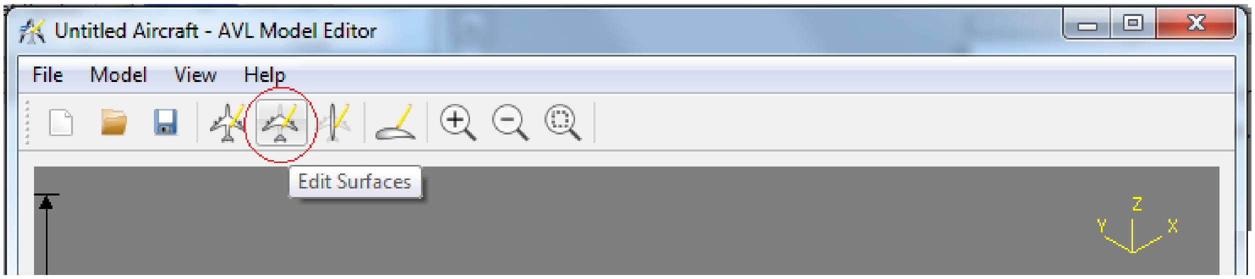
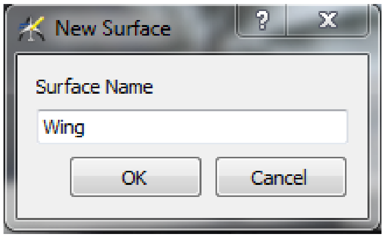


Figure 102 Edit Surfaces

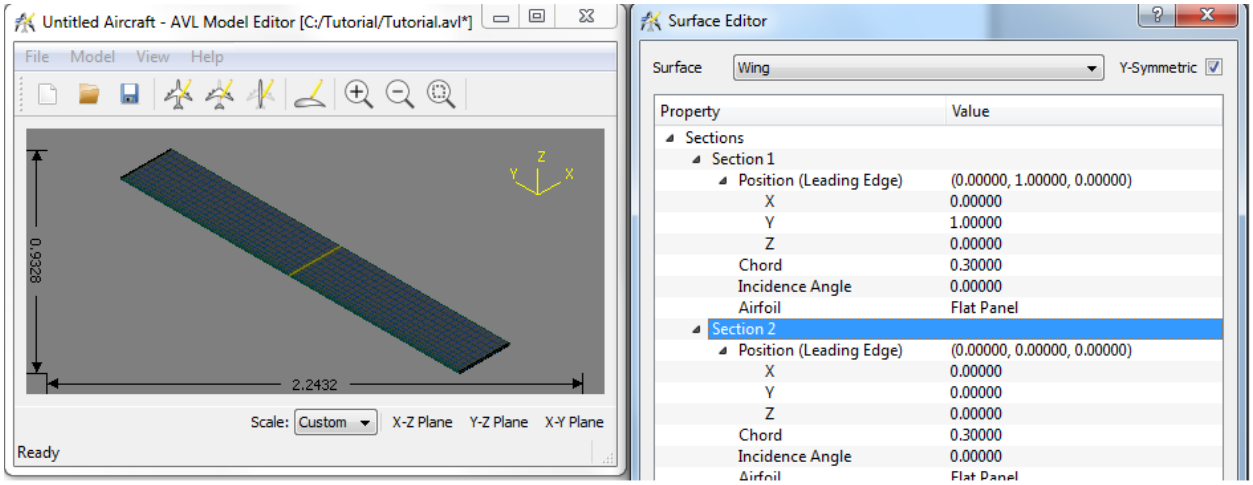
1) Click “New Surface”. A window will pop up asking for the name of the surface. Name it appropriately i.e. “Wing”, “Vertical”, “Horizontal”, “Vtail”, etc. The name of the Surface is not critical to the modeling process.



Surface Name: Wing

Figure 103 New Surface Window

AVLEditor initially creates a surface consisting of two mirrored sections. Section 1 is the outside section at (0,1,0) with a chord of 0.30 and a flat panel airfoil. Section 2 is the center section at (0,0,0) with a chord of 0.30 and a flat panel airfoil. Treat “Section 2” as the center of the surface and “Section 1” as the wing tip of the surface. When a new section is created AVLEditor will automatically place it between two sections. If Section 1 is not treated as the tip and a new section is added that belongs outside Section 1 then when the new section's location is refined it will be outside of Section 1 and cause AVLEditor to crash. The nature of creating sections is from the outside in.



Surface: Wing   Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(0.00000, 1.00000, 0.00000)
X	0.00000
Y	1.00000
Z	0.00000
Chord	0.30000
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(0.00000, 0.00000, 0.00000)
X	0.00000
Y	0.00000
Z	0.00000
Chord	0.30000
Incidence Angle	0.00000
Airfoil	Flat Panel

Figure 104 New Surface Section Numbers

The Y-Symmetric (mirroring) box will be checked by default. Y-Symmetry creates a mirror of the surface across the x-z plane onto the – y axis. If a control surface is mirrored AVL Analysis will deflect both control surfaces simultaneously to determine the control's effects on the aircraft.

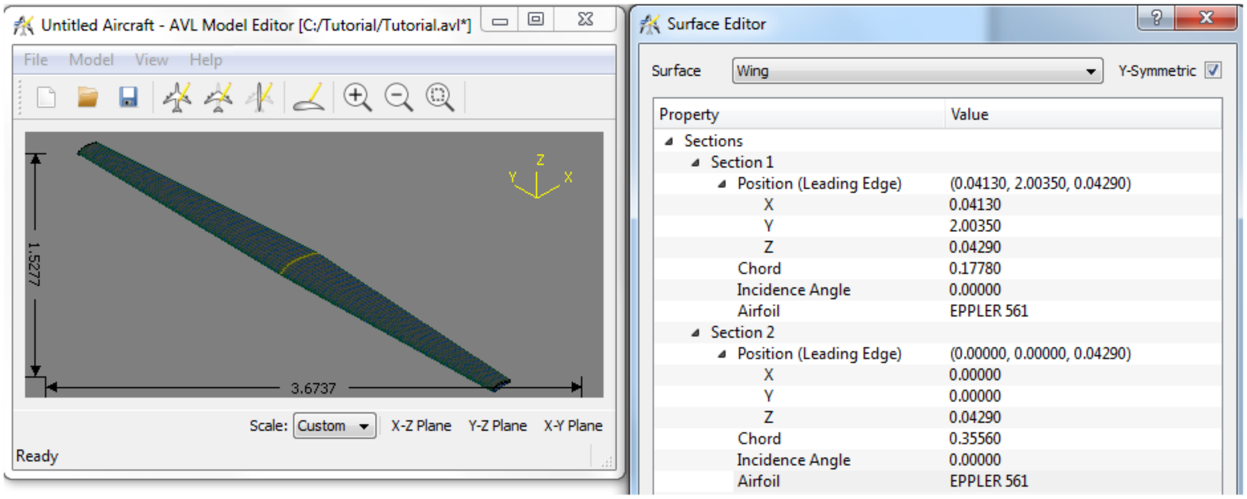
2) Use mirroring for all surfaces that are equal and opposite across the x-z plane EXCEPT for surfaces containing rudders. DO NOT mirror rudders. The only exception for mirroring surfaces containing rudders is if the rudder is involved in a built in mixture that is recognized by the piccolo software such as V-Tails, and Y-Tails where the control surfaces are ruddervators. Table 1 on page 90 lists the control surface mixtures that are recognized by the piccolo simulator.

3) Change the location of Sections 1 and 2 by clicking the arrow next to each Section and entering in their correct X,Y,Z locations. If using mirroring input the coordinates for the positive y direction.

4) Set the chord lengths of the two sections.

5) Open the Airfoil Editor and load the airfoil file, created in Section 5.4, for the current surface.

6) Back in Surface Editor assign the airfoils to the proper sections using the up/down arrows on the row "Airfoil" to select the name of the airfoil file. Click apply when done. The result should be similar to the example in Figure 105.



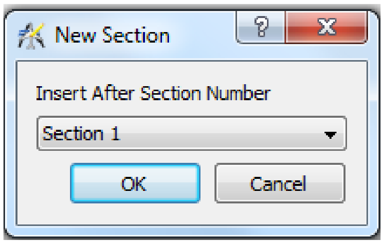
Surface: Wing

Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(0.04130, 2.00350, 0.04290)
X	0.04130
Y	2.00350
Z	0.04290
Chord	0.17780
Incidence Angle	0.00000
Airfoil	EPPLER 561
Section 2	
Position (Leading Edge)	(0.00000, 0.00000, 0.04290)
X	0.00000
Y	0.00000
Z	0.04290
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561

Figure 105 Assigning Airfoils to Surfaces

7) Click "New Section" to add the remaining sections that need to be placed. Continuously add sections to Section 1 until the number of sections matches the number required. If mirroring is being used only one side of sections is required.



Insert After Section Number

Section 1

Figure 106 New Section Window

8) Click Apply after all the sections have been added. They should have defaulted to locations in between Sections 2 and 1.

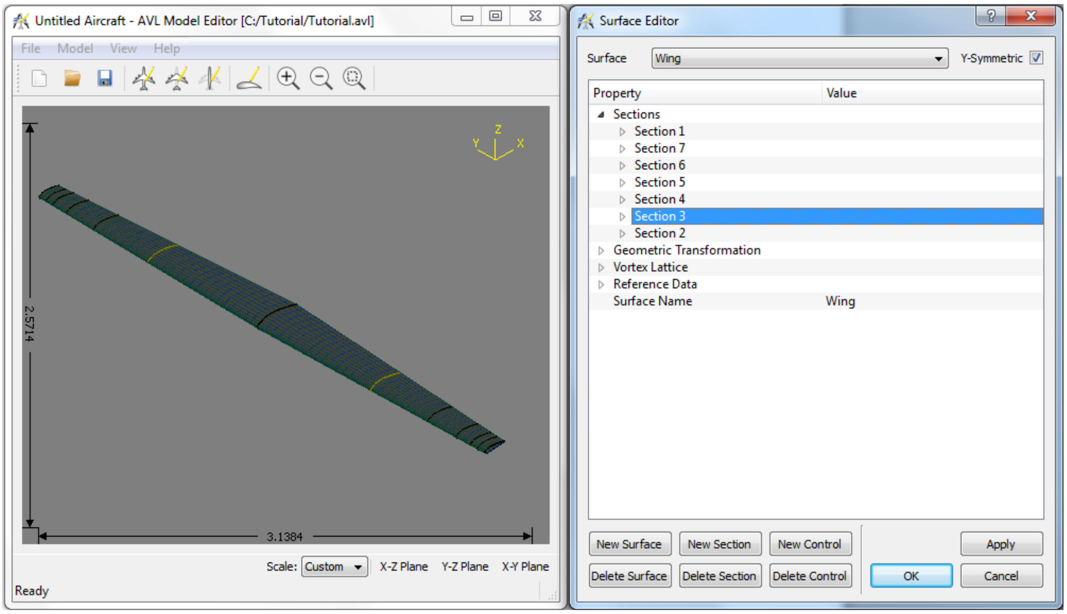


Figure 107 Assigning New Sections to a Surface

The default numbering of the sections is actually a glitch in AVLEditor. After the model is saved and re-opened the numbering of the sections will change, starting with Section 1 at the center and the highest numbered section at the surface tip.

Surfaces that are vertical are ordered top to bottom; however, every time that a model is saved the order of sections of vertical surfaces changes. This glitch can cause confusion, and in the case of Vertical Tails the section numbering order will determine the sign (+/-) of stability derivatives that depend on control surface deflections (Rudders).

9) If the current surface is not a vertical tail or there is only one vertical tail go to a). If the current surface is a vertical tail and there is more than one vertical tail go to b).

a) Save the model.

b) DO NOT save the model until all of the vertical tail surfaces have been created. Every time the model is saved the numbering of sections of vertical surfaces changes direction (ascending to descending, descending to ascending). If one vertical tail is saved before the other the ordering of the section numbers will be out of sync and cause the Simulator to calculate 0 Rudder Effectiveness. Figure 108 shows the discrepancy that can occur.

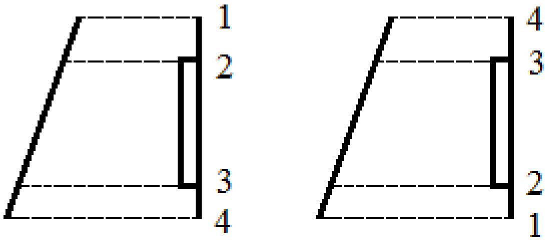
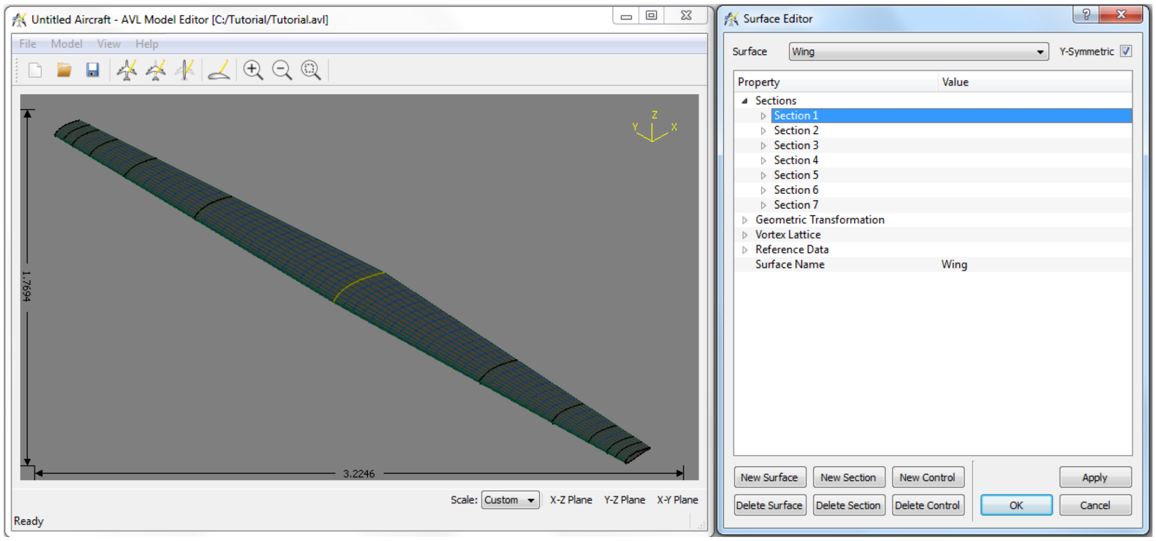


Figure 108 Rudder Section Numbering Glitch

Save the model after all of the vertical tail surfaces have been created.

10) Close and re-open AVLEditor. Load the aircraft.

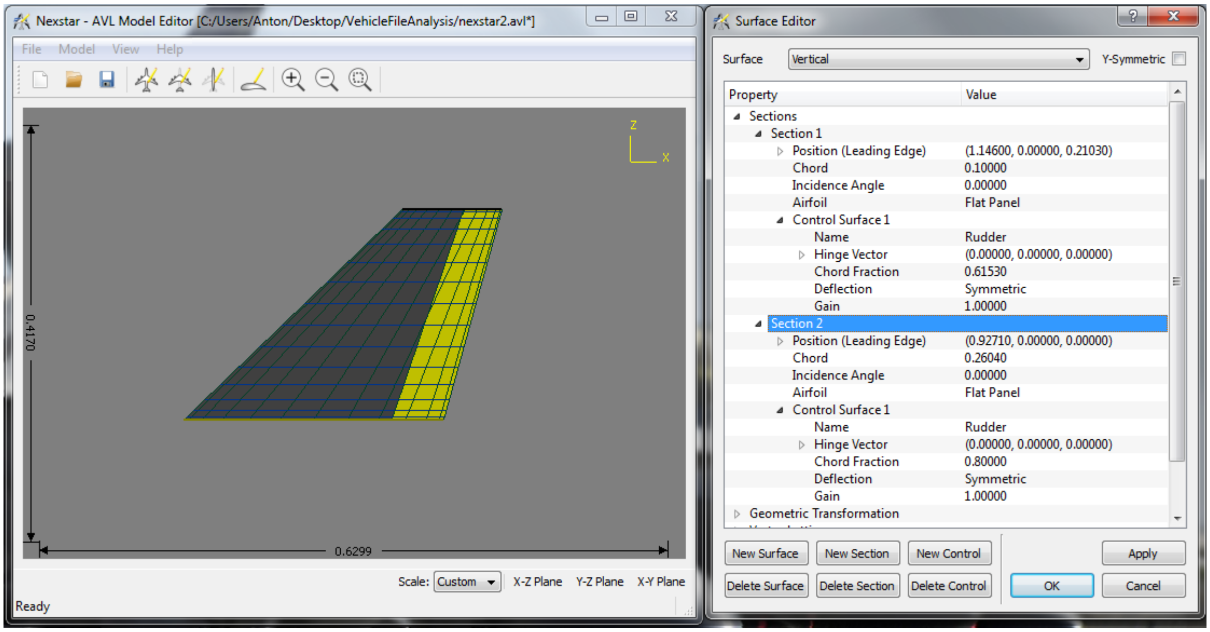
The sections should be numbered in ascending order from Section 1 in the center to the final Section at the surface tip as shown in Figure 109.



Property	Value
Sections	
Section 1	
Section 2	
Section 3	
Section 4	
Section 5	
Section 6	
Section 7	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Wing

Figure 109 Surface Section Re-Numbered

If the surface is a vertical surface the numbering should be either ascending or descending. If there are multiple vertical surfaces that contain the same control surfaces make sure that the sections are numbered in the same direction. Figure 110 shows a Vertical Tail.



Property	Value
Sections	
Section 1	
Position (Leading Edge)	(1.14600, 0.00000, 0.21030)
Chord	0.10000
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Rudder
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.61530
Deflection	Symmetric
Gain	1.00000
Section 2	
Position (Leading Edge)	(0.92710, 0.00000, 0.00000)
Chord	0.26040
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Rudder
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80000
Deflection	Symmetric
Gain	1.00000
Geometric Transformation	

Figure 110 Vertical Tail Section Number Example

11) Starting at Section 2, one section out from Section 1, change the following as required:

a) X,Y,Z Position

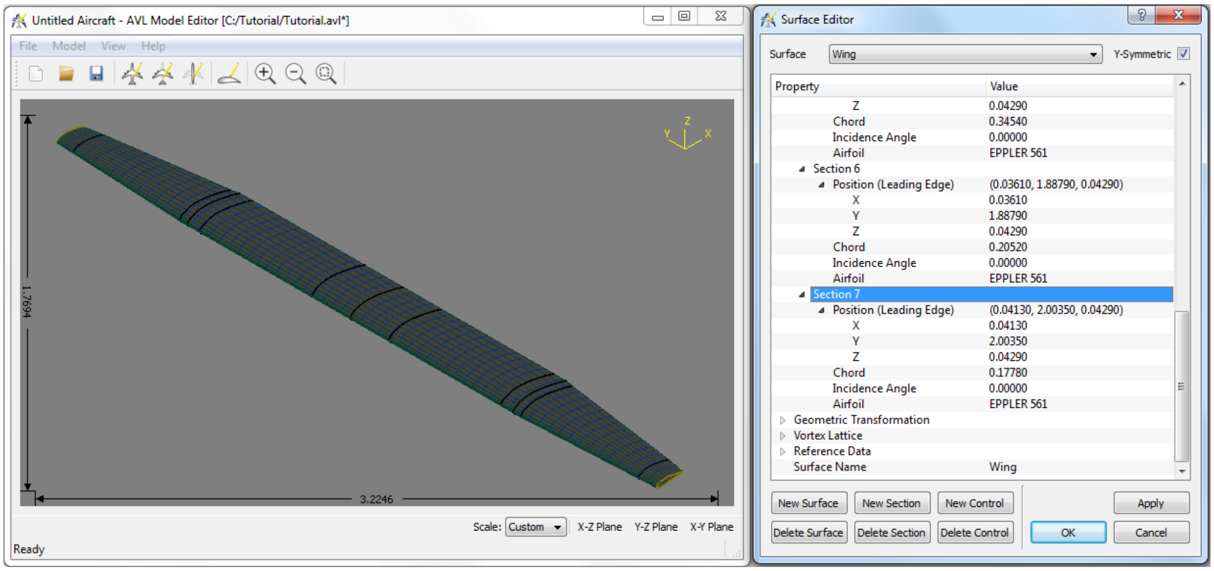
b) Chord Length

c) Incidence

d) Airfoils if needed

e) Click Apply after each section to view the changes

f) Save after all sections are correct



Property	Value
Z	0.04290
Chord	0.34540
Incidence Angle	0.00000
Airfoil	EPPLER 561
Section 6	
Position (Leading Edge)	(0.03610, 1.88790, 0.04290)
X	0.03610
Y	1.88790
Z	0.04290
Chord	0.20520
Incidence Angle	0.00000
Airfoil	EPPLER 561
Section 7	
Position (Leading Edge)	(0.04130, 2.00350, 0.04290)
X	0.04130
Y	2.00350
Z	0.04290
Chord	0.17780
Incidence Angle	0.00000
Airfoil	EPPLER 561
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Wing

Figure 111 Surface Section Data

12) If all surfaces have been created continue to Section 4.1.6 to begin creating control surfaces, otherwise go back to Step 1) and create the remaining surfaces.

4.1.6. Creating Control Surfaces

Control Surfaces at the most basic consist of Ailerons, Flaps, Elevators, and Rudders. Each control has its own function. When deflected ailerons create rolling moments, flaps create drag, elevators create pitching moments, and rudders create yawing moments. When the simulator calculates vehicle gains from the model it will interpret each control surface based on its name. Naming is therefore critical to creating control surfaces. The Piccolo has built in mixtures of controls for specific configurations that mix control surfaces. If the mixed controls are named by their designated mixed names the Simulator will recognize that they have multiple functions when calculating Vehicle Gains. For example if an aircraft has a V-Tail and the controls are named “Ruddervator” the Simulator will include the deflection of the V-Tail’s control surfaces in calculating both Elevator Effectiveness and Power, and Rudder Effectiveness and Power. If the surface was named “Rudder” then the V-tails contribution to the pitching moment when deflected would be excluded from the calculation of Elevator Effectiveness and Power. Below is a list of recognized control surface names that apply to AVL Modeling. Some of the names can be found in “PccUsersGuide.pdf” pg. 102-103.

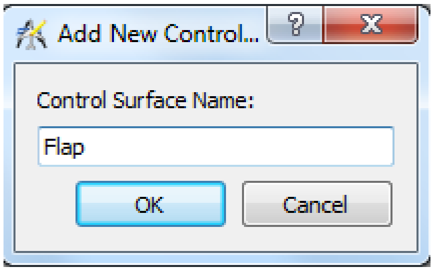
Aileron	Roll Control
Elevator	Pitch Control
Canard	Pitch Control (forward of CG)
Rudder	Yaw Control
Flap	Additional Drag
Elevon	Pitch & Roll Control (Elevator + Aileron)
Ruddervator	Pitch & Yaw Control (Elevator + Rudder)
Inverted Ruddervator	Pitch & Yaw Control (Elevator + Inverted Rudder)
Canarderon	Pitch & Roll Control (Canard + Aileron)
Outboard Elevon	Elevator + Aileron – Flap
Inboard Elevon	Elevator + Aileron + Flap

Table 4 Recognized Control Surface Mixtures

Use the following steps to create control surfaces:

1) Open Surface Editor.

2) Click “New Control”.



Control Surface Name:
Flap

Figure 112 Add New Control Surface

The first window that pops up will ask for the control surface name. If the surface that the control is on is mirrored then name it appropriately according to its function using Table 4. Make sure the name is an exact match for one of the names in the table.

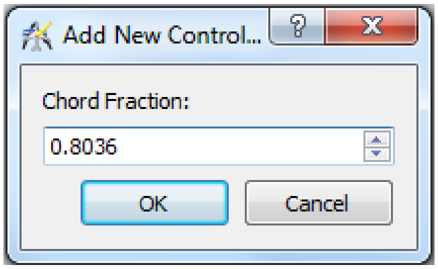
If the control is a mixture that is not in the table above:

a) Name the controls by their most basic functions (Aileron, Elevator, Rudder, Flap)

b) If the controls are a combination of a recognized mixture and basic controls use both. i.e. a Y-Tail consists of a “Ruddervator” and a “Rudder”

If the control is not on a mirrored surface add an “L” to the beginning of the name of the port side Control Surface, and add an “R” to the beginning of the name of the starboard side Control Surface.

3) Click OK.

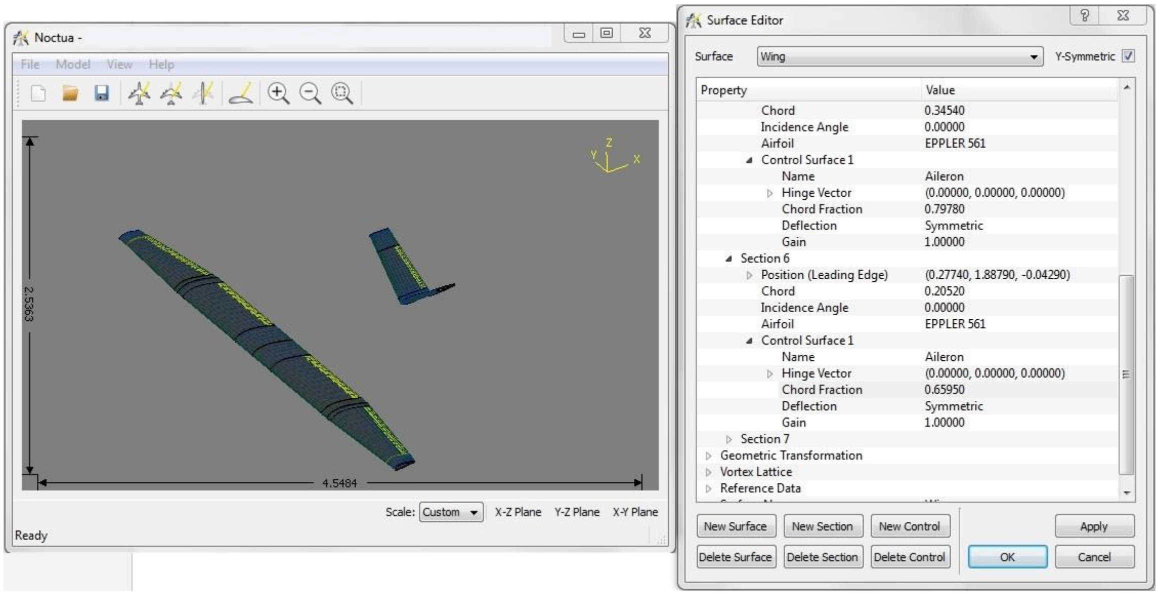


Chord Fraction:
0.8036

Figure 113 New Control Surface Chord Fraction Window

The next window asks for the Chord Fraction. The Chord Fraction should already have been measured and recorded during Geometry Measuring. Input the Chord Fraction of the first Section (inner most section) where the Control Surface begins.

AVLEditor automatically assumes that the chord fraction is constant across all sections. If the Chord Fraction changes at other sections that it is attached to (i.e. the surface tapers but the control surface does not) then the chord fraction at the other sections will have to be changed manually in Surface Editor after the input process has been completed.



Surface: Wing

Y-Symmetric ☑

Property	Value
Chord	0.34540
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Aileron
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.79780
Deflection	Symmetric
Gain	1.00000
Section 6	
Position (Leading Edge)	(0.27740, 1.88790, -0.04290)
Chord	0.20520
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Aileron
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.65950
Deflection	Symmetric
Gain	1.00000
Section 7	
Geometric Transformation	
Vortex Lattice	
Reference Data	

Figure 114 Chord Fraction Input Example

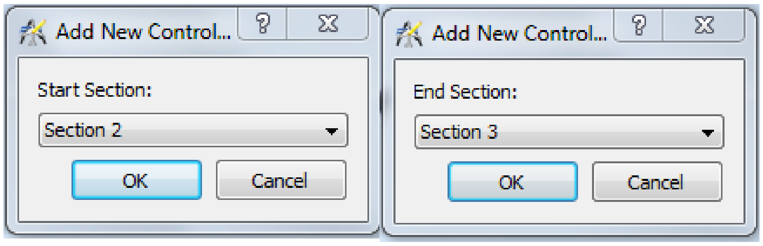
Figure 114 depicts an example where the chord fraction of the ailerons on a tapered wing had to be adjusted after the control surface was initially created. The chord fraction was input as 0.79780 which was correct for Section 5, but not for Section 6. The chord fraction for Section 6 was manually changed to 0.6950 in the Surface Editor.

4) Click OK.

5) Select the start section of the control from the drop down menu. This should be the inner most section.

6) Click OK.

7) Select the section where the control ends from the drop down menu. The control should span across the sections in ascending order towards the surface tip.



Add New Control...

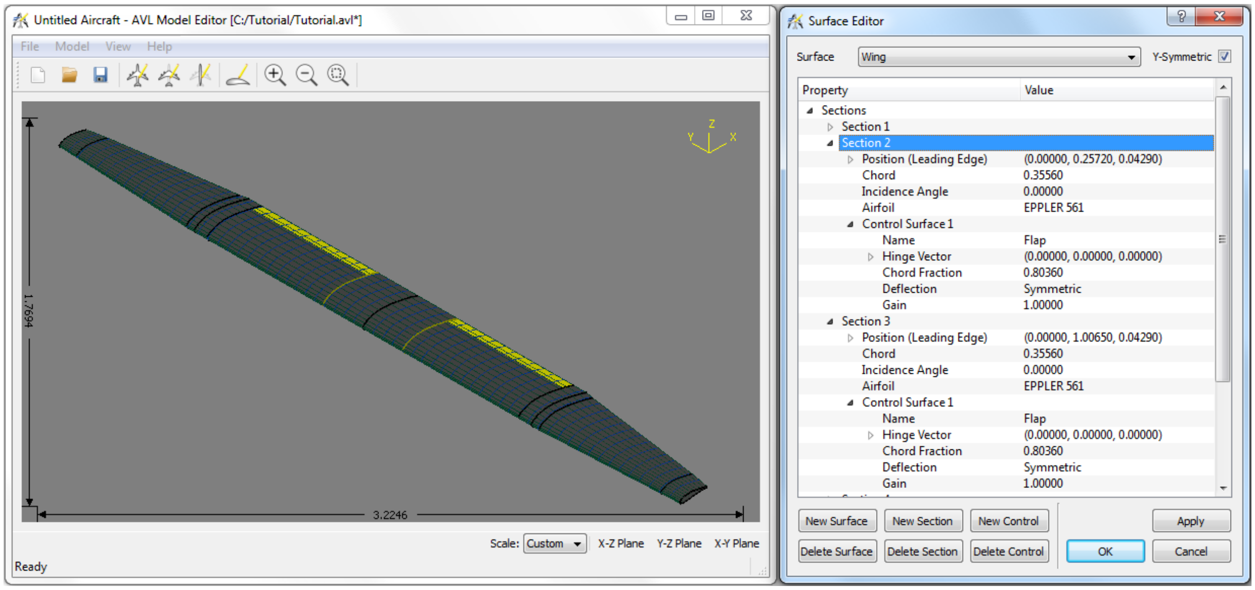
Start Section: Section 2

Add New Control...

End Section: Section 3

Figure 115 New Control Surface Start and End Section

8) Click Apply in the Surface Editor to view the changes. There should be a new tab under each section in the Surface Editor labeled "Control Surface #".



Untitled Aircraft - AVL Model Editor [C:/Tutorial/Tutorial.avl*]

Scale: Custom

Ready

Surface Editor

Surface: Wing

Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Section 2	
Position (Leading Edge)	(0.00000, 0.25720, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Flap
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80360
Deflection	Symmetric
Gain	1.00000
Section 3	
Position (Leading Edge)	(0.00000, 1.00650, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Flap
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80360
Deflection	Symmetric
Gain	1.00000

Figure 116 New Control Surface Definition in Surface Editor

9) If the Chord Fraction needs to be changed:

a) Click the drop down arrow on the appropriate Section.

b) Click the drop down arrow next to “Control Surface #”.

c) Change Chord Fraction.

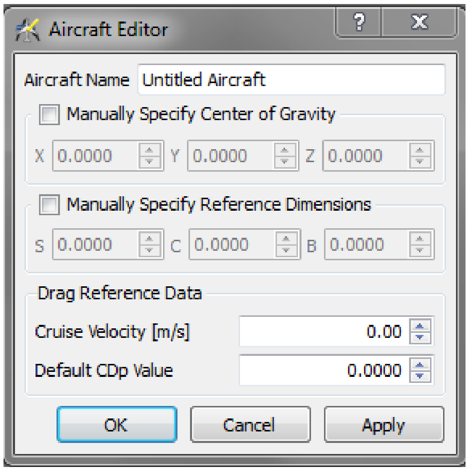
d) Repeat for each Section that requires a different Chord Fraction.

e) Click Apply when finished.

10) If all the control surfaces have been created save the model and continue to Section 4.1.7. Otherwise return to Step 1) and create the remaining control surfaces.

4.1.7. Aircraft Data

The “Aircraft Editor” is used to set Cruise Velocity, CDp, Center of Gravity location, Reference Dimensions, and the aircraft’s name. The Center of Gravity, Reference Dimensions, Cruise Velocity, and CDp are all used for AVL Analysis. The Center of Gravity and Reference Dimensions are recorded in the Alpha Sweep file to be loaded into the Simulator. The Cruise Velocity and CDp are also used in Xfoil analysis with Cruise Velocity affecting the Reynolds Number of each surface.



Aircraft Editor

Aircraft Name: Untitled Aircraft

☐ Manually Specify Center of Gravity

X 0.0000 Y 0.0000 Z 0.0000

☐ Manually Specify Reference Dimensions

S 0.0000 C 0.0000 B 0.0000

Drag Reference Data

Cruise Velocity [m/s]: 0.00

Default CDp Value: 0.0000

Figure 117 Aircraft Editor

AVLEditor will estimate the center of gravity location based on the surfaces in the model if the center of gravity is not manually specified. AVLEditor sets the Reference Dimensions to be equal to the Wing's dimensions.

1) Click "Edit Aircraft Data" in the toolbar.

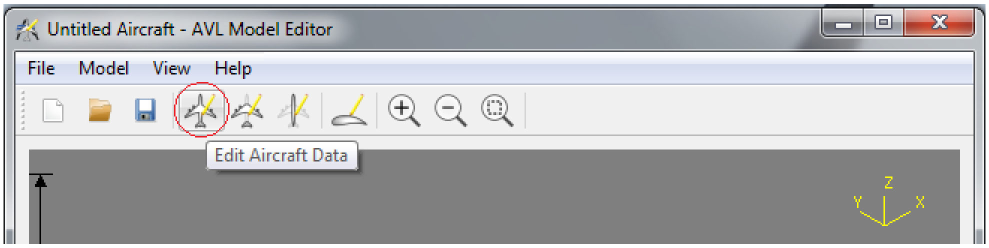


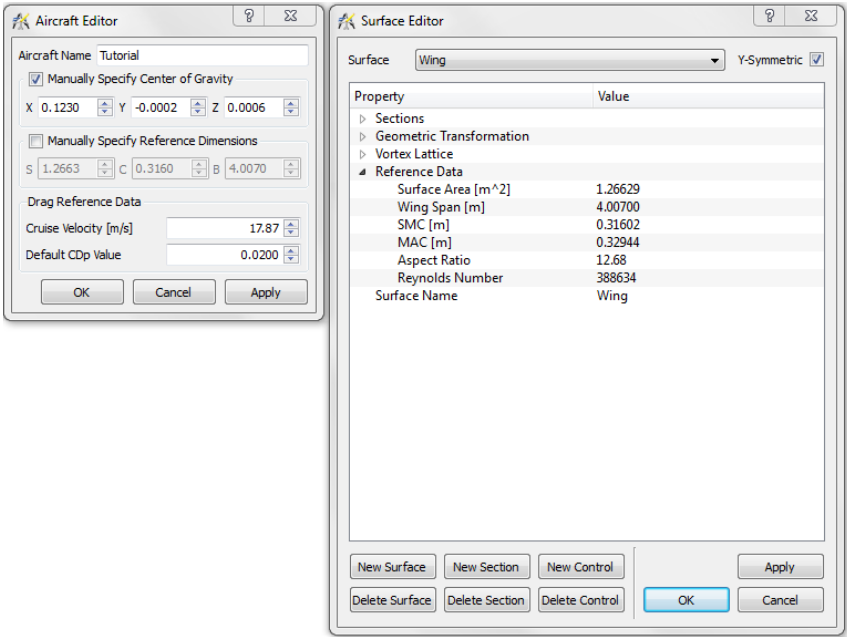
Figure 118 Launch Aircraft Editor

2) Type in a name for the aircraft.

3) Click the checkbox to "Manually Specify Center of Gravity".

4) The input boxes should now be accessible. Input the location of the Center of Gravity of the aircraft. Make sure that the units are Meters, and that the location is with respect to the wing leading edge at (0,0,0,).

5) Check that the Reference Dimensions that AVLEditor estimates are identical to the Wing's dimensions.



Aircraft Name: Tutorial

☑ Manually Specify Center of Gravity

X 0.1230 Y -0.0002 Z 0.0006

☐ Manually Specify Reference Dimensions

S 1.2663 C 0.3160 B 4.0070

Drag Reference Data

Cruise Velocity [m/s]: 17.87

Default CDp Value: 0.0200

Surface: Wing   Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
▷ Vortex Lattice	
◢ Reference Data	
Surface Area [m^2]	1.26629
Wing Span [m]	4.00700
SMC [m]	0.31602
MAC [m]	0.32944
Aspect Ratio	12.68
Reynolds Number	388634
Surface Name	Wing

Figure 119 Reference Dimensions

S = Surface Area (m^2) C = Chord (m) B = Wingspan (m)

6) If the reference dimensions are not correct, manually specify them.

7) Set the Cruise Velocity as the Velocity that the aircraft is expected to operate at in m/s.

8) Input a value for CDp. AVLEditor defines CDp as the profile drag, or in other words CD0. Estimation of CD0 should include drag estimates for landing gear, and all of the extra extremities of the aircraft. CDp should also include the CD contribution of flat plate surfaces.

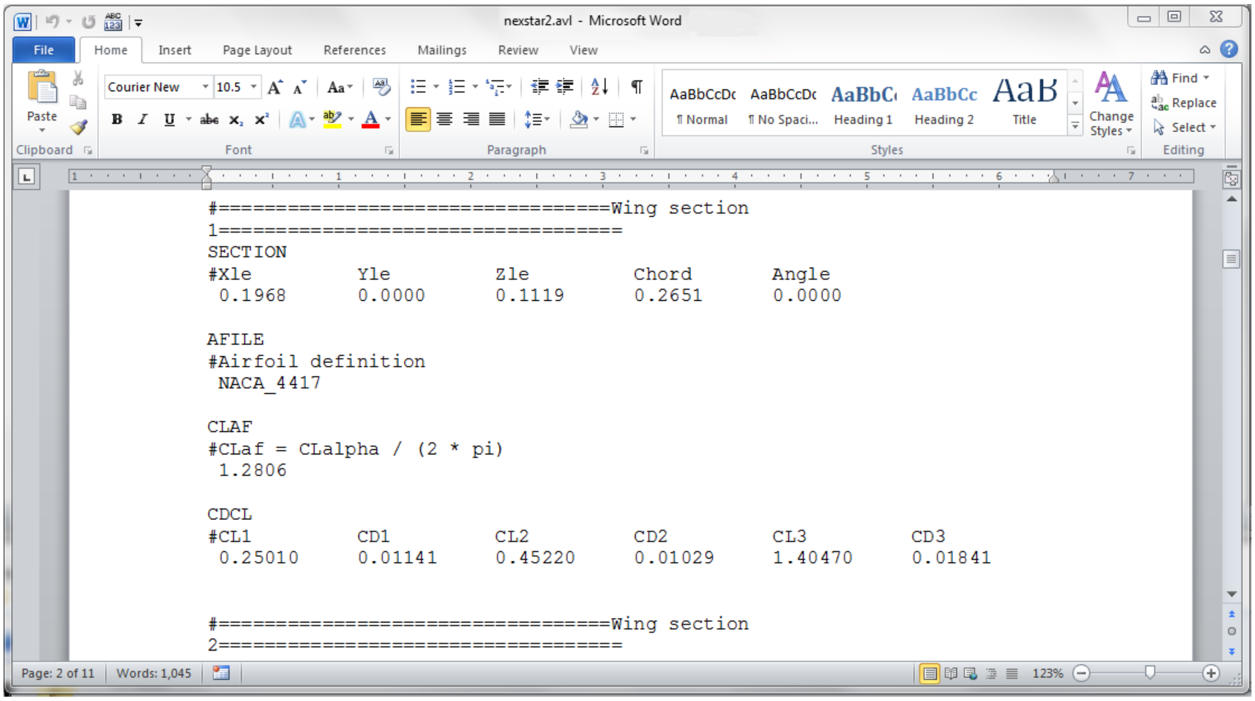
   CDp should NOT include CD0 of Wings and Tails that are not flat plates. CD0 of the Wings and Tails with airfoil shapes will be calculated in Xfoil Analysis.

   AVL Analysis treats CDp as a constant and adds it to Cdvis which is calculated from the drag polar produced by Xfoil Analysis.

9) Click OK when finished. Save the model. Continue to Section 4.1.8 AVLEditor Xfoil Analysis.

4.1.8. AVLEditor Xfoil Analysis

Xfoil Analysis uses Xfoil to add viscous effects to the AVL Model for AVL Analysis by calculating and storing the drag polar (CdCl curves) and 2D Lift Slope (ΔCl/Δα) in the model's ".avl" text file. An example of a stored drag polar in an avl file is shown in Figure 120.



```
#==================================Wing section
1===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      0.0000      0.1119      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 1.2806

CDCL
#CL1         CD1         CL2         CD2         CL3         CD3
 0.25010     0.01141     0.45220     0.01029     1.40470     0.01841


#==================================Wing section
2===================================
```

Figure 120 AVL File Drag Polar

Each Cl Cd value recorded represents a point on the drag polar. CdCl1 represents the location of the smallest Cl value. CdCl2 represents the location of the smallest Cd value, and CdCl3 represents the location of the largest Cl value.

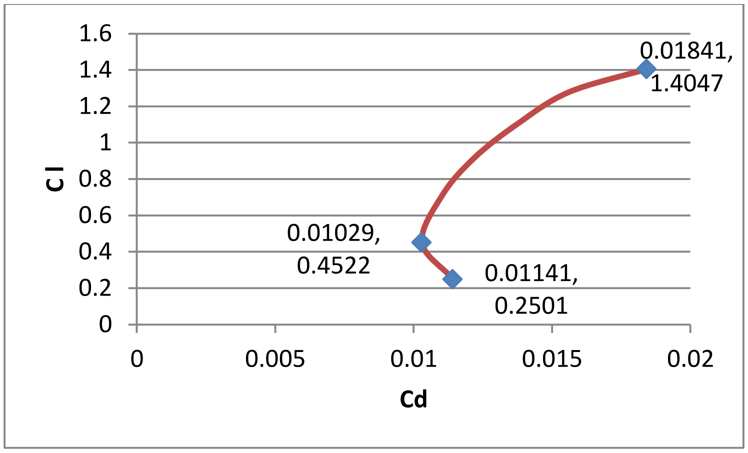


Figure 121 Drag Polar Extrapolation

AVL uses the three saved points to extrapolate the CdCl curves by assuming the paths between the top two and bottom two points are 2nd degree polynomials as depicted in Figure 121.

AVL treats the 2-D lift slope as a function of thickness to chord ratio using the parameter “CLAF”. CLAF is a correction factor that is supposed to predict the effects of thick airfoils on the 2-D lift slope (dCl/dα).

$$\text{CLAF} = 1 + 0.77*t/c$$

Equation 19 CLAF

$$\Delta Cl/\Delta\alpha = 2\pi * \text{CLAF}$$

Equation 20 AVL File 2D Lift Slope Declaration

Equation 19 and Equation 20 come from the avl text file “avl_doc.txt” that installs with AVL 3.32. In Equation 19 t and c are the max thickness and chord of the airfoil. If the airfoil is thin then ΔCl/Δα should be defined as 2π, in accordance with thin airfoil theory.

4.1.8.1. CLAF

AVLEditor uses Xfoil Analysis to determine CLAF and the drag polar. Unfortunately the values it calculates for CLAF have nothing to do with the thickness equation that it is supposed to. AVLEditor's CLAF can vary wildly depending on the range of alpha angles the user selects for Xfoil Analysis, and doesn't have anything to do with Equation 19. CLAF has a large impact on Vehicle Gain estimates such as Elevator Effectiveness, CL at Zero Elevator, LiftSlope, and Clmax.

As a result CLAF has to be determined manually through Xfoil itself, outside of AVLEditor. The drag polar will still be calculated using AVLEditor's Xfoil Analysis. Manually determining CLAF requires using Xfoil to obtain the 2-D lift curve of the airfoil and using that curve to determine the 2-D lift slope of the airfoil. The process will have to be done for each airfoil on each surface. Additionally the range of alphas used to calculate the lift slope of the wings will be used to determine the linear range of alphas for Xfoil and AVL analysis.

Xfoil is included in the Piccolo install "PiccoloInstaller.msi" and can be found in the AVLEditor folder under "Cloud Cap\Piccolo 2.x.x.x\".

1) Launch Xfoil 6.96 by opening "xfoil.exe"

2) If instructions on how to use Xfoil are needed proceed to Steps 3) through 8); otherwise do the following:

    a) Load the airfoil.

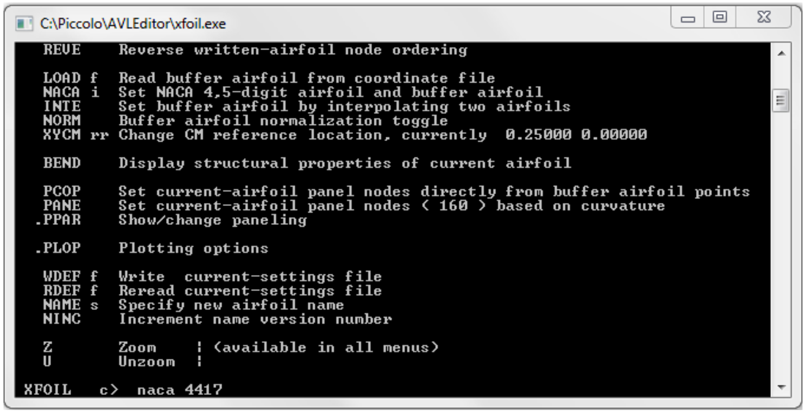
    b) Set the number of airfoil panel nodes to 160.

c) Run Xfoil from -15 to +15 degrees angle of attack in viscous mode with the Reynolds number equal to the Reynolds number of the surface that the airfoil is on. Use the Mach number that the aircraft is expected to fly at.

d) Go to Step 9).

3) Load the airfoil. Instructions below explain how to do so. If the airfoil is an NACA 4 or 5 digit series go to a). Otherwise go to b).

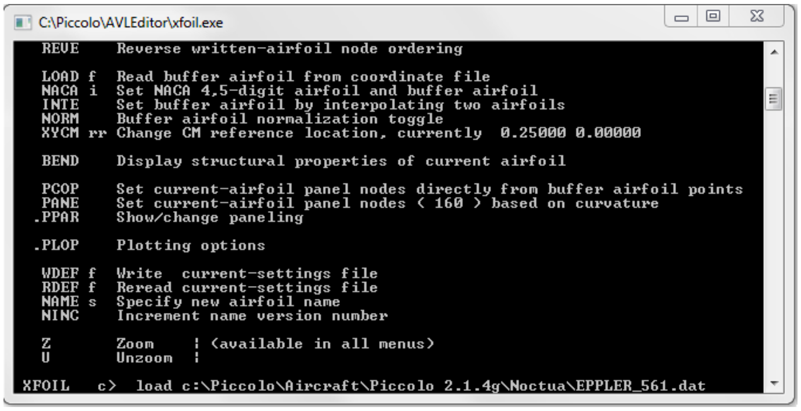
a) Type in “naca“ and the 4 or 5 digits of the airfoil and hit enter. Continue to 3).



```
  REVE    Reverse written-airfoil node ordering

  LOAD f  Read buffer airfoil from coordinate file
  NACA i  Set NACA 4,5-digit airfoil and buffer airfoil
  INTE    Set buffer airfoil by interpolating two airfoils
  NORM    Buffer airfoil normalization toggle
  XYCM rr Change CM reference location, currently  0.25000 0.00000

  BEND    Display structural properties of current airfoil

  PCOP    Set current-airfoil panel nodes directly from buffer airfoil points
  PANE    Set current-airfoil panel nodes ( 160 ) based on curvature
 .PPAR    Show/change paneling

 .PLOP    Plotting options

  WDEF f  Write  current-settings file
  RDEF f  Reread current-settings file
  NAME s  Specify new airfoil name
  NINC    Increment name version number

  Z       Zoom    | (available in all menus)
  U       Unzoom  |

XFOIL   c>  naca 4417
```

Figure 122 Xfoil NACA Example

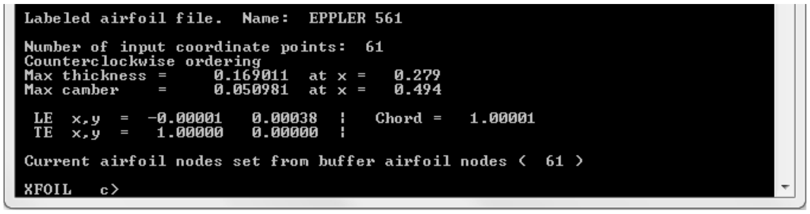
b) Type “load “and the location of the airfoil file in the command line



```
  REVE    Reverse written-airfoil node ordering

  LOAD f  Read buffer airfoil from coordinate file
  NACA i  Set NACA 4,5-digit airfoil and buffer airfoil
  INTE    Set buffer airfoil by interpolating two airfoils
  NORM    Buffer airfoil normalization toggle
  XYCM rr Change CM reference location, currently  0.25000 0.00000

  BEND    Display structural properties of current airfoil

  PCOP    Set current-airfoil panel nodes directly from buffer airfoil points
  PANE    Set current-airfoil panel nodes ( 160 ) based on curvature
 .PPAR    Show/change paneling

 .PLOP    Plotting options

  WDEF f  Write  current-settings file
  RDEF f  Reread current-settings file
  NAME s  Specify new airfoil name
  NINC    Increment name version number

  Z       Zoom    | (available in all menus)
  U       Unzoom  |

XFOIL   c>  load c:\Piccolo\Aircraft\Piccolo 2.1.4g\Noctua\EPPLER_561.dat
```

Figure 123 Load Airfoil File in Xfoil

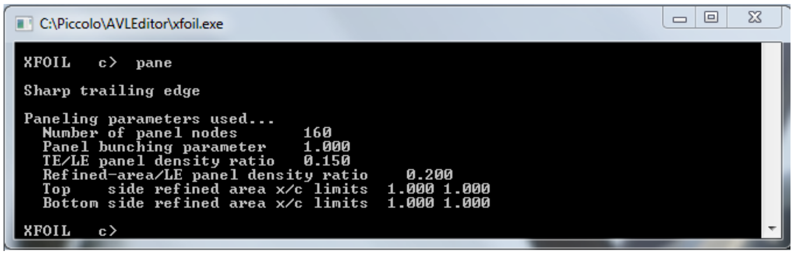
The number of panel nodes needs to be set by Xfoil. If the airfoil file was saved through AVLEditor it usually only saves 61 nodes. The number of nodes is displayed after loading.



```
Labeled airfoil file.  Name:  EPPLER 561

Number of input coordinate points:  61
Counterclockwise ordering
Max thickness =     0.169011  at x =   0.279
Max camber    =     0.050981  at x =   0.494

 LE  x,y  =  -0.00001   0.00038  |   Chord =   1.00001
 TE  x,y  =   1.00000   0.00000  |

Current airfoil nodes set from buffer airfoil nodes (  61 )

XFOIL   c>
```

Figure 124 Airfoil Nodes

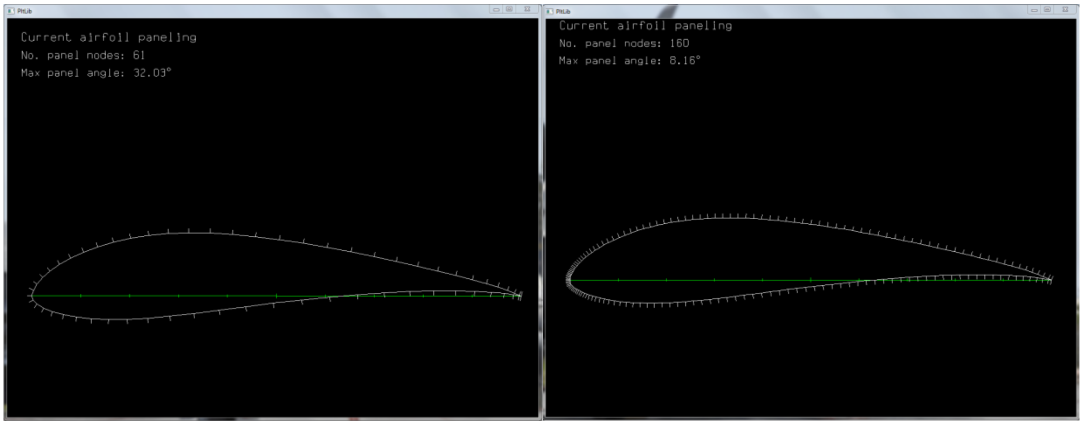
Type “pane” in the command line to have Xfoil set the number of nodes. When pane is entered it should automatically set the number of nodes to 160.



```
XFOIL   c>  pane

Sharp trailing edge

Paneling parameters used...
  Number of panel nodes       160
  Panel bunching parameter    1.000
  TE/LE panel density ratio   0.150
  Refined-area/LE panel density ratio    0.200
  Top    side refined area x/c limits  1.000 1.000
  Bottom side refined area x/c limits  1.000 1.000

XFOIL   c>
```

Figure 125 Generate Extra Nodes

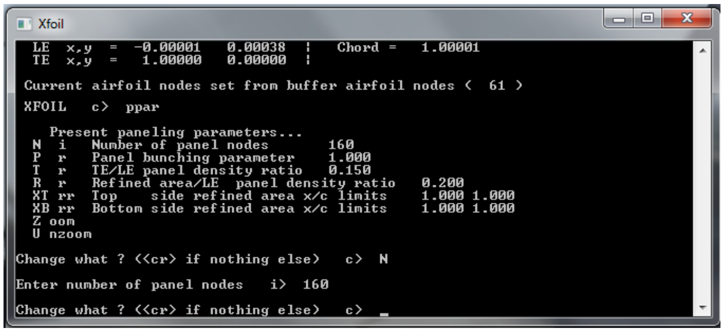
To view the nodes type “ppar” in the command line. If the number of panel nodes is still incorrect they can be changed manually by entering an “N” and then the number of nodes. Continue to 4). In Figure 126 the airfoil on the left depicts the original profile of a loaded airfoil file and the airfoil on the right depicts the airfoil profile after pane was commanded.



```
Current airfoil paneling
No. panel nodes: 61
Max panel angle: 32.03°
```

```
Current airfoil paneling
No. panel nodes: 160
Max panel angle: 8.16°
```

Figure 126 Airfoil Panel Nodes Comparison



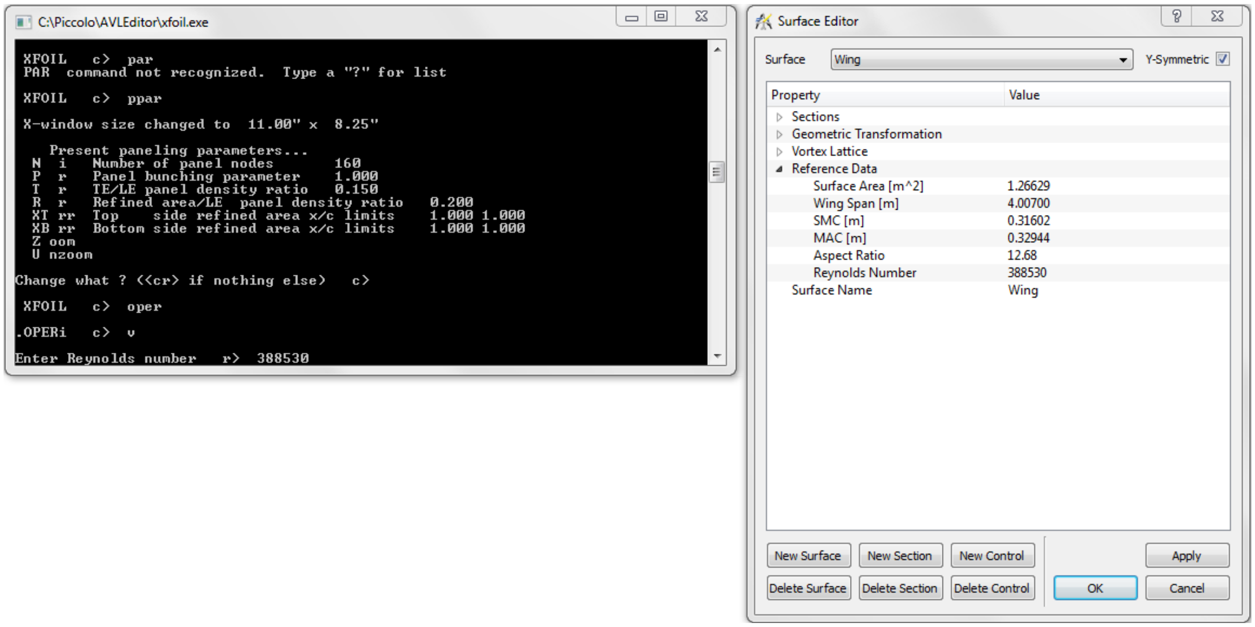
```
  LE  x,y  =  -0.00001   0.00038  ¦   Chord =   1.00001
  TE  x,y  =   1.00000   0.00000  ¦

 Current airfoil nodes set from buffer airfoil nodes (  61 )

 XFOIL   c>  ppar

    Present paneling parameters...
  N  i   Number of panel nodes       160
  P  r   Panel bunching parameter    1.000
  T  r   TE/LE panel density ratio   0.150
  R  r   Refined area/LE  panel density ratio   0.200
  XT rr  Top    side refined area x/c limits    1.000 1.000
  XB rr  Bottom side refined area x/c limits    1.000 1.000
  Z oom
  U nzoom

Change what ? (<cr> if nothing else)   c>  N

Enter number of panel nodes   i>  160

Change what ? (<cr> if nothing else)   c> _
```

Figure 127 Manually Specify Number of Panel Nodes

4) Type “Oper” in the command line.

5) Type “v” in the command line for a viscid solution.

6) Xfoil will ask for a Reynolds number. Input the Reynolds number of the surface the airfoil is on. Check the surface reference data in the Surface Editor of AVLEditor to make sure the Reynolds numbers match.



C:\Piccolo\AVLEditor\xfoil.exe

```
 XFOIL   c>  par
 PAR  command not recognized.  Type a "?" for list

 XFOIL   c>  ppar

 X-window size changed to  11.00" x  8.25"

    Present paneling parameters...
  N  i   Number of panel nodes       160
  P  r   Panel bunching parameter    1.000
  T  r   TE/LE panel density ratio   0.150
  R  r   Refined area/LE  panel density ratio    0.200
  XT rr  Top    side refined area x/c limits     1.000 1.000
  XB rr  Bottom side refined area x/c limits     1.000 1.000
  Z oom
  U nzoom

Change what ? (<cr> if nothing else)   c>

 XFOIL   c>  oper

.OPERi   c>  v

Enter Reynolds number   r>  388530
```

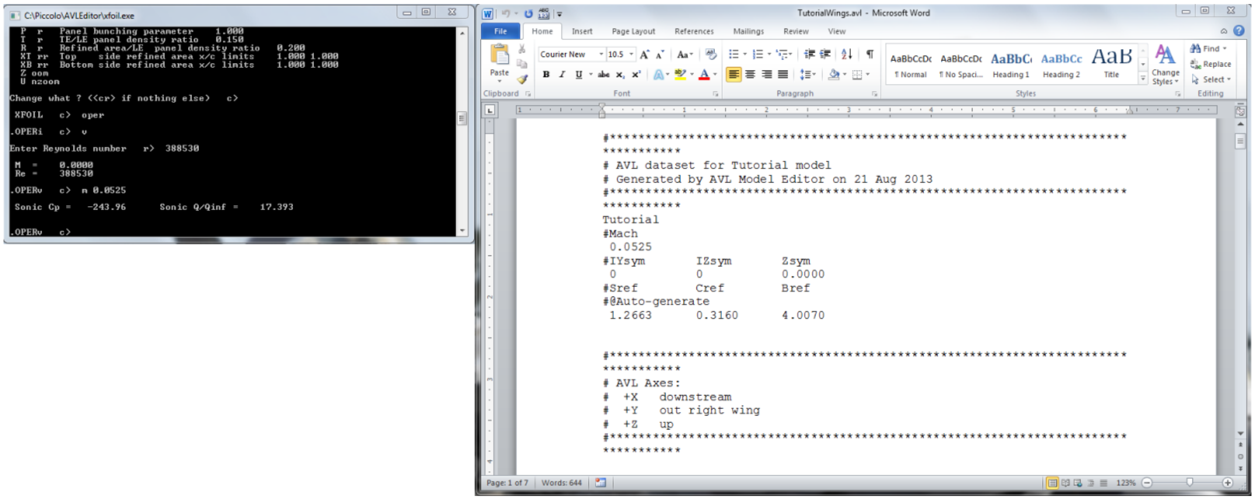
Surface Editor

Surface: Wing   Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
▷ Vortex Lattice	
◢ Reference Data	
Surface Area [m^2]	1.26629
Wing Span [m]	4.00700
SMC [m]	0.31602
MAC [m]	0.32944
Aspect Ratio	12.68
Reynolds Number	388530
Surface Name	Wing

Figure 128 Define Xfoil Reynolds Number

7) Type “m” in the command line to set the Mach umber. Mach number can be found at the beginning of the avl text file of the aircraft if it is not already known.



C:\Piccolo\AVLEditor\xfoil.exe

```
  P  r   Panel bunching parameter    1.000
  T  r   TE/LE panel density ratio   0.150
  R  r   Refined area/LE  panel density ratio    0.200
  XT rr  Top    side refined area x/c limits     1.000 1.000
  XB rr  Bottom side refined area x/c limits     1.000 1.000
  Z oom
  U nzoom

Change what ? (<cr> if nothing else)   c>

 XFOIL   c>  oper

.OPERi   c>  v

Enter Reynolds number   r>  388530

 M  =    0.0000
 Re =    388530

.OPERv   c>  m 0.0525

 Sonic Cp =   -243.96      Sonic Q/Qinf =    17.393


.OPERv   c>
```

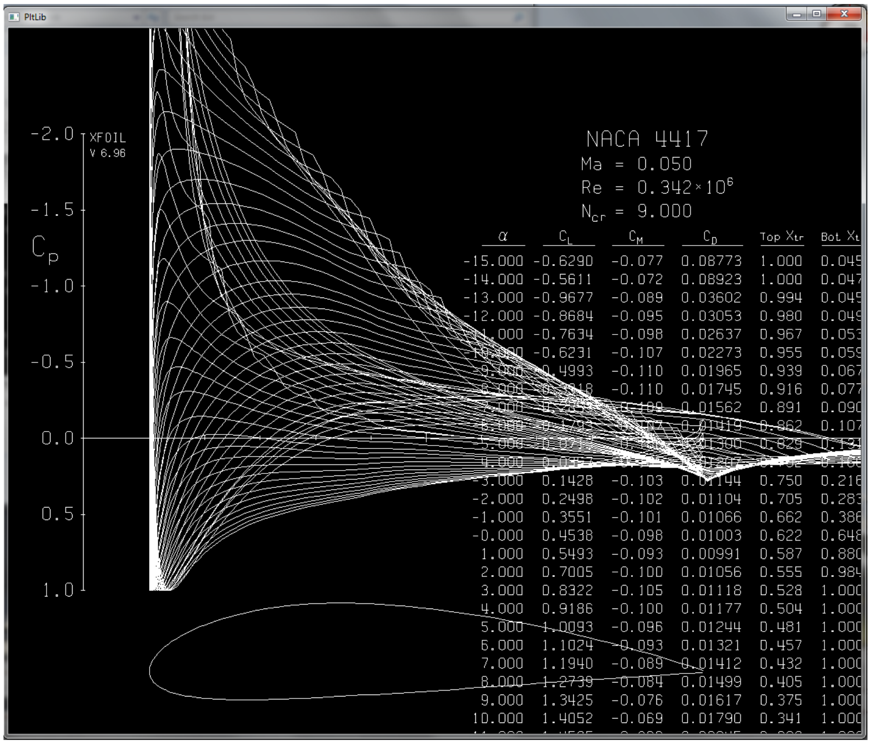
TutorialWings.avl - Microsoft Word

```
#*******************************************************************
***********
# AVL dataset for Tutorial model
# Generated by AVL Model Editor on 21 Aug 2013
#*******************************************************************
***********
Tutorial
#Mach
 0.0525
#IYsym       IZsym       Zsym
 0           0           0.0000
#Sref        Cref        Bref
#@Auto-generate
 1.2663      0.3160      4.0070


#*******************************************************************
***********
# AVL Axes:
#  +X   downstream
#  +Y   out right wing
#  +Z   up
#*******************************************************************
***********
```

Figure 129 Define Xfoil Mach Number

8) Type “aseq” and run Xfoil from -15 to +15 degrees angle of attack.

9) Record the Cl and Cd at each angle.



```
        NACA 4417
       Ma  =  0.050
       Re  =  0.342×10^6
       Ncr =  9.000

    α        CL        CM        CD     Top Xtr  Bot Xt
-15.000  -0.6290   -0.077   0.08773   1.000   0.045
-14.000  -0.5611   -0.072   0.08923   1.000   0.047
-13.000  -0.9677   -0.089   0.03602   0.994   0.045
-12.000  -0.8684   -0.095   0.03053   0.980   0.049
-11.000  -0.7634   -0.098   0.02637   0.967   0.053
-10.000  -0.6231   -0.107   0.02273   0.955   0.059
 -9.000  -0.4993   -0.110   0.01965   0.939   0.067
 -8.000  [illegible] -0.110   0.01745   0.916   0.077
 -7.000  [illegible] -0.109   0.01562   0.891   0.090
 -6.000  -0.1793  [illegible] 0.01419   0.862   0.107
 -5.000  [illegible] [illegible] 0.01300   0.829   0.131
 -4.000  [illegible] [illegible] [illegible] [illegible] [illegible]
 -3.000   0.1428   -0.103   0.01144   0.750   0.216
 -2.000   0.2498   -0.102   0.01104   0.705   0.283
 -1.000   0.3551   -0.101   0.01066   0.662   0.386
 -0.000   0.4538   -0.098   0.01003   0.622   0.648
  1.000   0.5493   -0.093   0.00991   0.587   0.880
  2.000   0.7005   -0.100   0.01056   0.555   0.984
  3.000   0.8322   -0.105   0.01118   0.528   1.000
  4.000   0.9186   -0.100   0.01177   0.504   1.000
  5.000   1.0093   -0.096   0.01244   0.481   1.000
  6.000   1.1024   -0.093   0.01321   0.457   1.000
  7.000   1.1940   -0.089   0.01412   0.432   1.000
  8.000   1.2739   -0.084   0.01499   0.405   1.000
  9.000   1.3425   -0.076   0.01617   0.375   1.000
 10.000   1.4052   -0.069   0.01790   0.341   1.000
```

Figure 130 Xfoil AOA Sweep

10) Plot Cl vs. alpha.

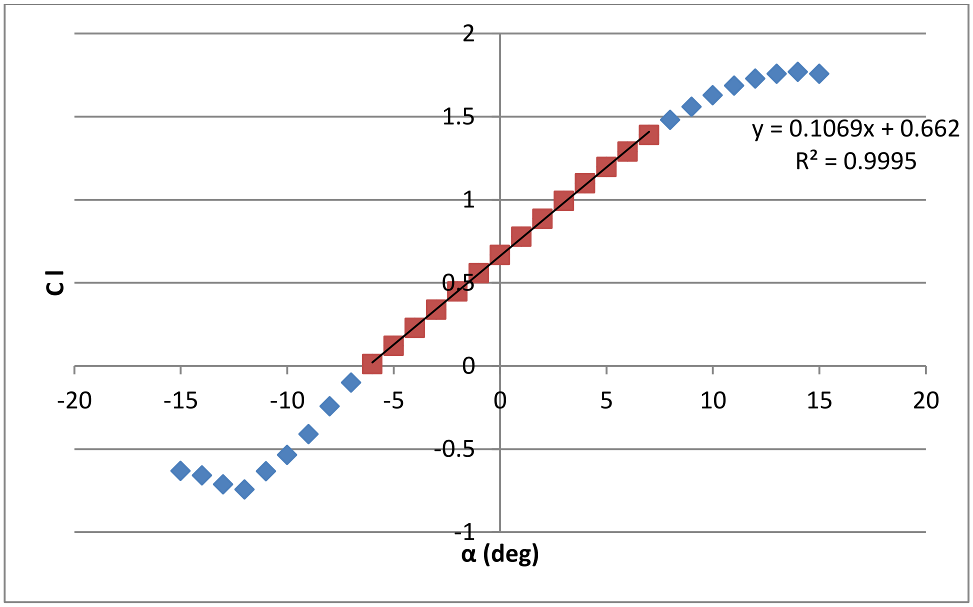


Figure 131 Cla Slope

11) Use the linear section of the curve to calculate ΔCl/Δα. Make sure to convert the slope from /deg to /rad.

12) Calculate CLAF = (ΔCl/Δα * 180/π)/(2π).

13) If the surface being analyzed is the wings the linear section will be used to define the linear range for use later in AVLEditor alpha sweeps.

14) Repeat Steps 1) – 12) until CLAF has been calculated for all of the airfoils on all of the surfaces of the aircraft. Continue to Section 4.1.8.2 Running Xfoil Analysis.

4.1.8.2. Running Xfoil Analysis

Even though CLAF is calculated manually, and the drag polar could be determined manually as well, it is still useful to use Xfoil Analysis in AVLEditor. Xfoil Analysis will calculate the drag polar properly and has the added benefit of properly recording the Xfoil data in each section it applies to in the ".avl" text file; thus, AVLEditor will essentially format the avl text file for Xfoil data.

1) Open AVLEditor and load the aircraft model.

2) Above the toolbar click "Model" and click "Xfoil Analysis…" from the drop down menu to launch the "Xfoil Output" window.

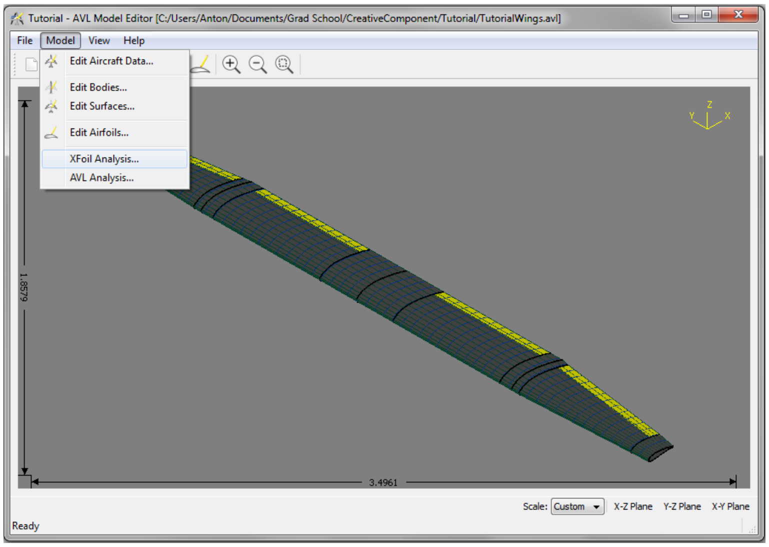


Figure 132 AVLEditor Xfoil Analysis

3) In the “Xfoil Output” window click “File” and click “Start Xfoil …” from the drop down menu.

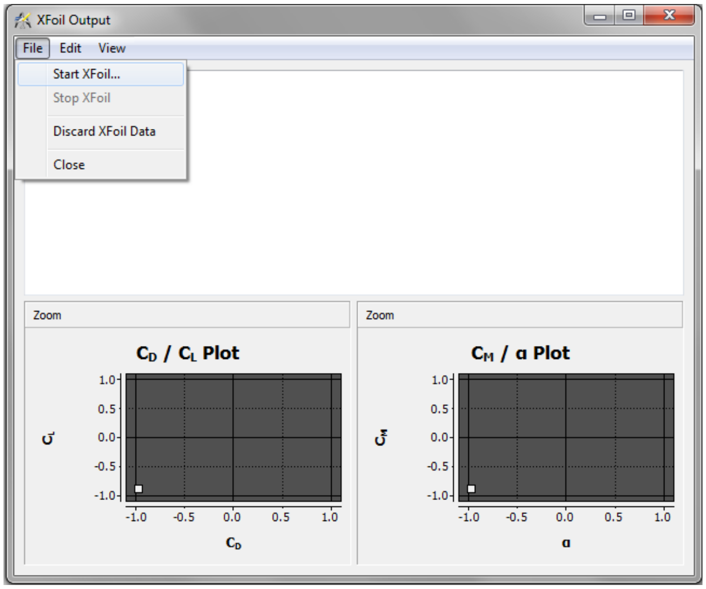
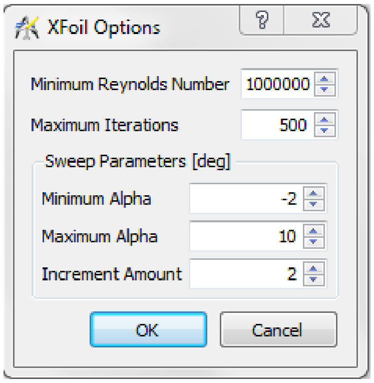


Figure 133 AVLEditor Xfoil Output Window

4) The Xfoil Options window will pop up.



XFoil Options

Minimum Reynolds Number 1000000

Maximum Iterations 500

Sweep Parameters [deg]

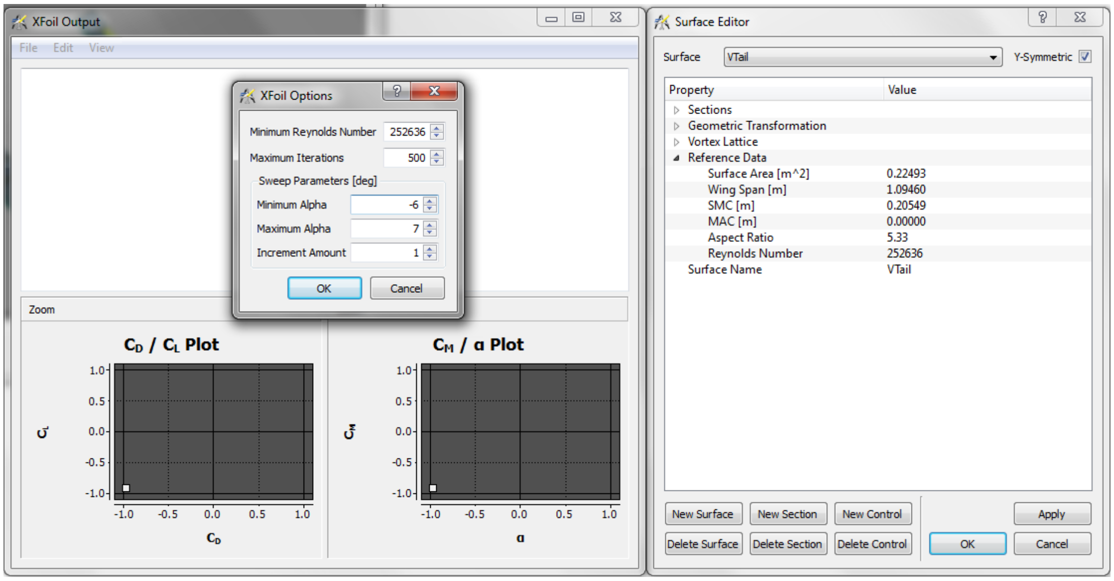
Minimum Alpha -2

Maximum Alpha 10

Increment Amount 2

Figure 134 AVLEditor Xfoil Options

Set the minimum Reynolds Number to be <= the smallest Reynolds number out of all of the surfaces that exist on the aircraft. In Figure 135 below the Vtail has the smallest Reynolds number so its Reynolds number was used as the minimum.



XFoil Output

XFoil Options

Minimum Reynolds Number 252636

Maximum Iterations 500

Sweep Parameters [deg]

Minimum Alpha -6

Maximum Alpha 7

Increment Amount 1

Surface Editor

Surface VTail Y-Symmetric ☑

Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Area [m^2]	0.22493
Wing Span [m]	1.09460
SMC [m]	0.20549
MAC [m]	0.00000
Aspect Ratio	5.33
Reynolds Number	252636
Surface Name	VTail

Figure 135 Input Xfoil Analysis Parameters

5) Set the range of alpha to be equal to the range that was determined from the 2D lift slope of the wings (the linear range). Click OK.

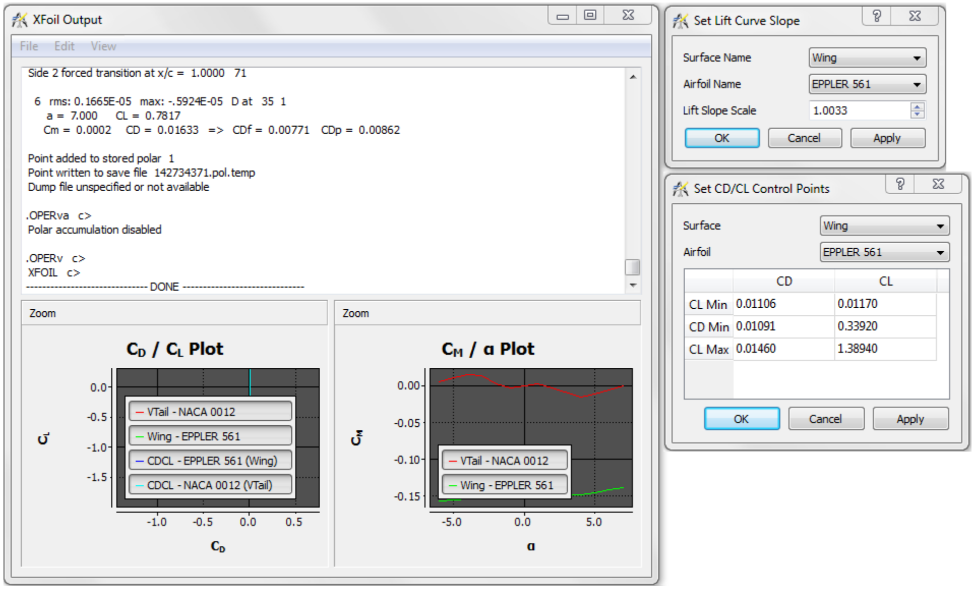
6) When Xfoil Analysis is finished save the model. If the model is not saved after running Xfoil Analysis the results will not be saved to the model’s avl text file.

7) Continue to Section 4.1.8.3 Xfoil Results.

4.1.8.3. Xfoil Results

The Xfoil Output window shows all commands that AVLEditor commanded to Xfoil. The plots show the drag polar for each surface and Cmα of the entire aircraft. Click “View” in the toolbar to edit what the Xfoil Output is showing.

“Edit” in Xfoil Output is supposed to give the user the capability to manually edit CLAF and the drag polar points; however, the Xfoil Output tools don’t actually change anything.



```
Side 2 forced transition at x/c =  1.0000   71

  6   rms: 0.1665E-05   max: -.5924E-05   D at   35  1
     a =  7.000     CL =  0.7817
    Cm =  0.0002    CD =  0.01633   =>   CDf =  0.00771   CDp =  0.00862

Point added to stored polar  1
Point written to save file  142734371.pol.temp
Dump file unspecified or not available

.OPERva   c>
Polar accumulation disabled

.OPERv   c>
XFOIL   c>
------------------------------ DONE ------------------------------
```

Set Lift Curve Slope

Surface Name: Wing

Airfoil Name: EPPLER 561

Lift Slope Scale: 1.0033

Set CD/CL Control Points

Surface: Wing

Airfoil: EPPLER 561

	CD	CL
CL Min	0.01106	0.01170
CD Min	0.01091	0.33920
CL Max	0.01460	1.38940

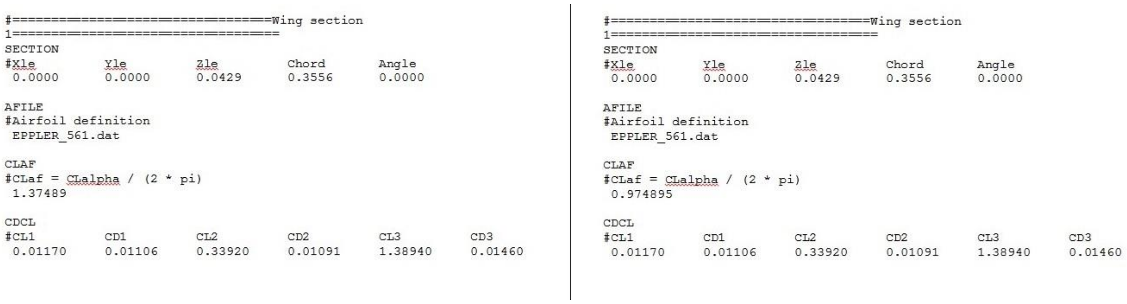
Figure 136 AVLEditor Xfoil Edit Options

Since these windows don’t work manually changing CLAF requires opening the model’s avl text file and copy and pasting at each section. Note that CLAF is labeled as “Lift Slope Scale” in the Lift Curve Slope window shown in Figure 136.

Perform the following steps:

2) Close AVLEditor and open the aircraft's avl text file in Microsoft Word. Notepad can also be used; however, the formatting is not desirable to use for editing values.

3) Adjust CLAF appropriately for ALL surface sections that have airfoils. Note that flat plate sections are excluded from Xfoil Analysis. Make sure that the appropriate CLAF value is used for each section. Figure 137 below is an example of a model where the AVLEditor's generated CLAF value was too high and had to be changed manually.



```
#================================Wing section
1==================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0000      0.0000      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.37489

CDCL
#CL1         CD1         CL2         CD2         CL3         CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460
```

```
#================================Wing section
1==================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0000      0.0000      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.974895

CDCL
#CL1         CD1         CL2         CD2         CL3         CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460
```

Figure 137 Example CLAF Edit

4) When all the CLAF values have been adjusted save the avl file. If a warning from Microsoft Word about formatting pops up click "Yes" to continue and save.

5) Continue to Section 4.1.9 Creating Fuselages.

4.1.9. Creating Fuselages

The objective for creating a fuselage is to treat the fuselage as two surfaces. One surface represents the cross sectional area of the top view of the fuselage (x-y plane), and the other surface represents the cross sectional area of the side view of the fuselage (x-z plane). The two surfaces should intersect each other at the center of the fuselage. Do not use the Body Editor in AVLEditor for fuselages.

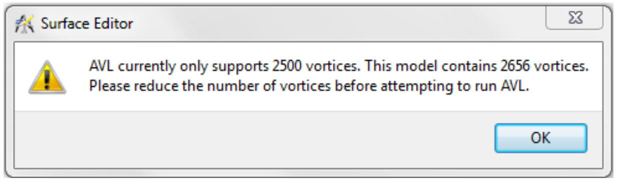
4.1.9.1. Fuselage Top View Surface

1) Open AVLEditor and load the aircraft.

2) Open the Surface Editor and Click New Surface.

3) Name the Surface “FuselageTop”.

4) If there is a warning about too many vortices being used, just click OK and continue. Vortex Lattice settings will be adjusted later in Section 4.1.10.



AVL currently only supports 2500 vortices. This model contains 2656 vortices. Please reduce the number of vortices before attempting to run AVL.

Figure 138 AVLEditor Vortex Warning

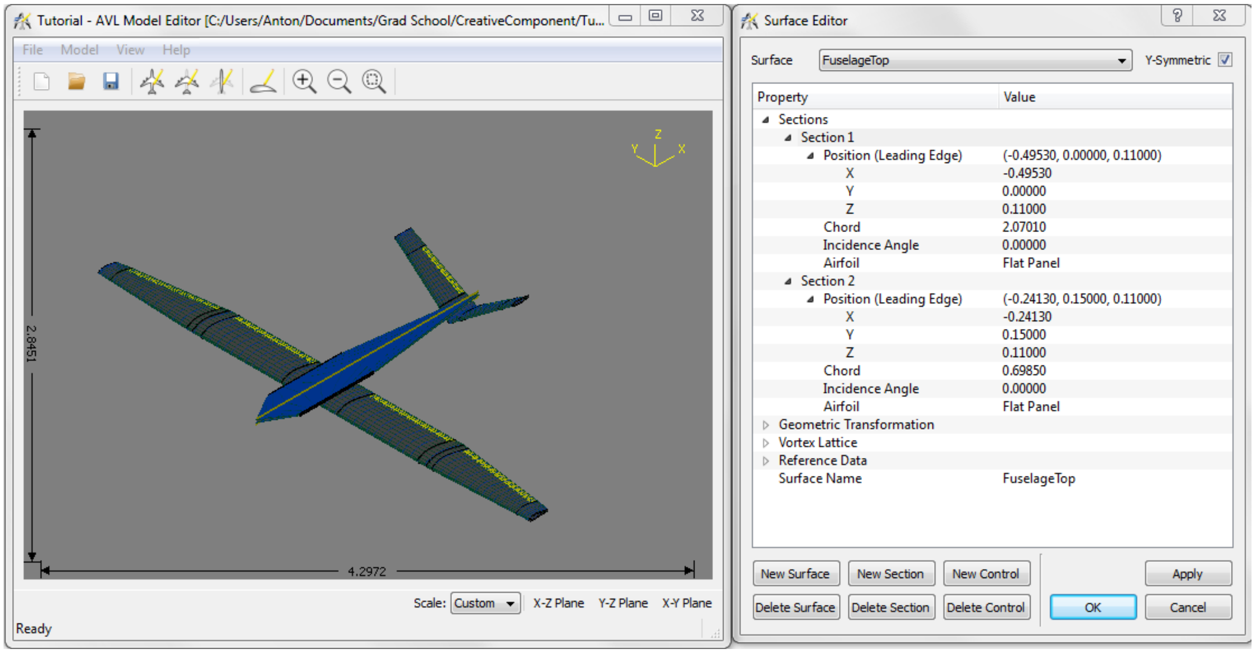
5) Use Y-Symmetry. Make sure the box is checked.

6) Similar to creating surfaces begin by setting Section 1 as the final section in the Y – direction.

7) Set the x,y,z positions and chord lengths of Sections 1 and 2. The surface’s z position should position the surface so that it travels through the center of the fuselage.

8) Airfoil should be set to “Flat Panel”. Click Apply.

9) Create as many sections as are required, but don’t bother changing their position yet.



Property	Value
Sections	
Section 1	
Position (Leading Edge)	(-0.49530, 0.00000, 0.11000)
X	-0.49530
Y	0.00000
Z	0.11000
Chord	2.07010
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(-0.24130, 0.15000, 0.11000)
X	-0.24130
Y	0.15000
Z	0.11000
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	FuselageTop

Figure 139 AVLEditor Top View Fuselage Surface

10) Save the model. Close and re-open so that the sections are numbered in ascending order, starting at section 1 on the inside and progressing towards the edge.

11) Edit the x,y,z locations and chord lengths of the inside sections so that FuselageTop matches the cross sectional area that was measured during geometry measuring.

12) Save, and continue to Section 4.1.9.2 Fuselage Side View Surface.

4.1.9.2. Fuselage Side View Surface

1) Open the Surface Editor and Click New Surface

2) Name the Surface “FuselageSide”

3) Do not use Y-Symmetry. Uncheck the Y-Symmetric box.

4) Set Section 1 as the top section in the Z – direction.

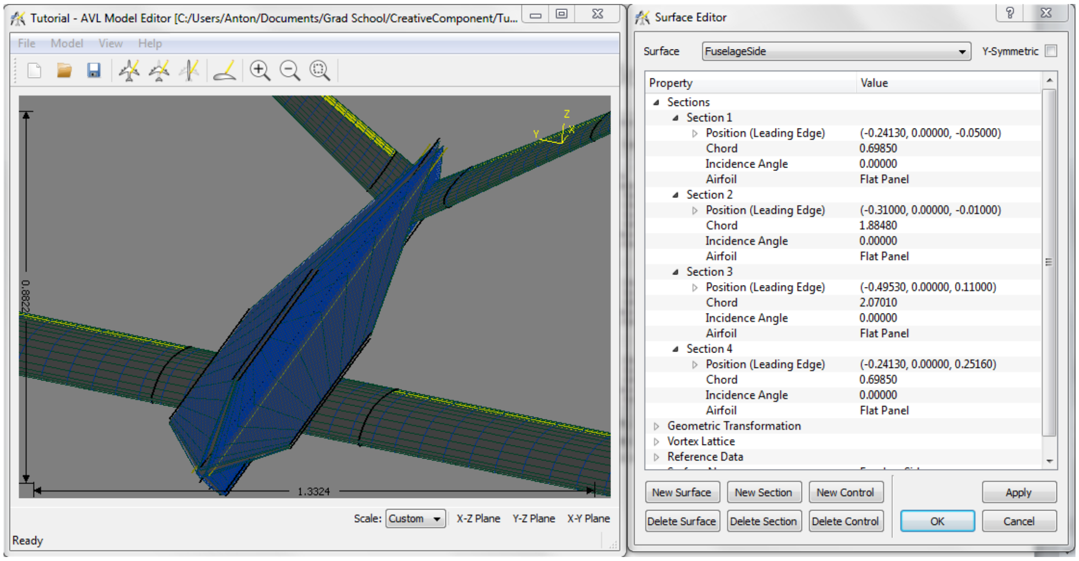
5) Set Section 2 as the bottom section.

6) Airfoil should be set to “Flat Panel”. Click Apply.

7) Create the remaining number of sections required.

8) Edit the x,y,z locations and chord lengths of the inside sections so that FuselageSide matches the cross sectional area that was measured during geometry measuring.

9) Click Apply. FuselageSide should intersect the center of FuselageTop.



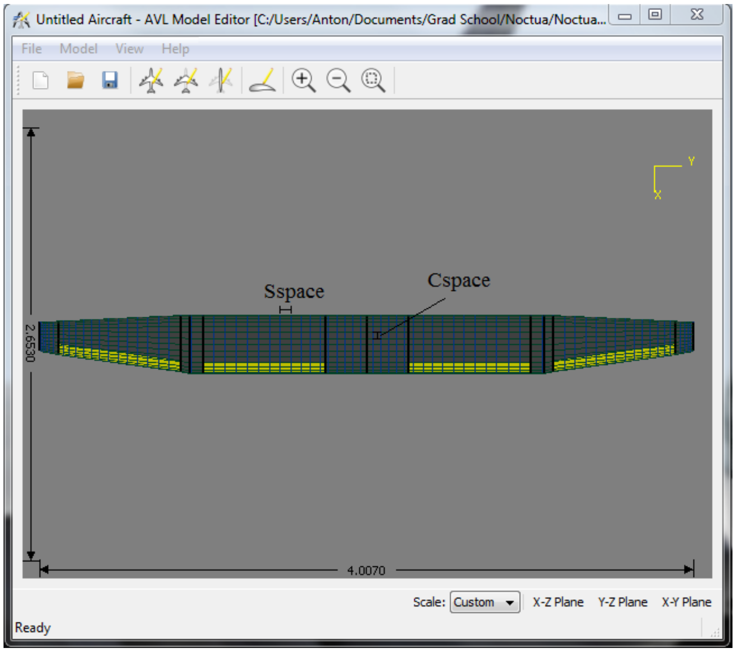
Property	Value
Sections	
Section 1	
Position (Leading Edge)	(-0.24130, 0.00000, -0.05000)
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(-0.31000, 0.00000, -0.01000)
Chord	1.88480
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 3	
Position (Leading Edge)	(-0.49530, 0.00000, 0.11000)
Chord	2.07010
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 4	
Position (Leading Edge)	(-0.24130, 0.00000, 0.25160)
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Geometric Transformation	
Vortex Lattice	
Reference Data	

Figure 140 AVLEditor Side View Fuselage Surface

10) Continue to Section 4.1.10 Vortex Lattice Settings.

4.1.10. Vortex Lattice Settings

The Vortex Lattice settings determine the number of horseshoe vortices and their spacing in both the spanwise and chordwise directions. Each surface is broken down into a grid of nodes. The Vortex Lattice settings define the number of nodes and their spacing in both the spanwise direction and chordwise direction. The number of nodes is defined by “Nchord” and “Nspan”. Spanwise spacing is defined as “Sspace” and chordwise spacing is defined as “Cspace”.

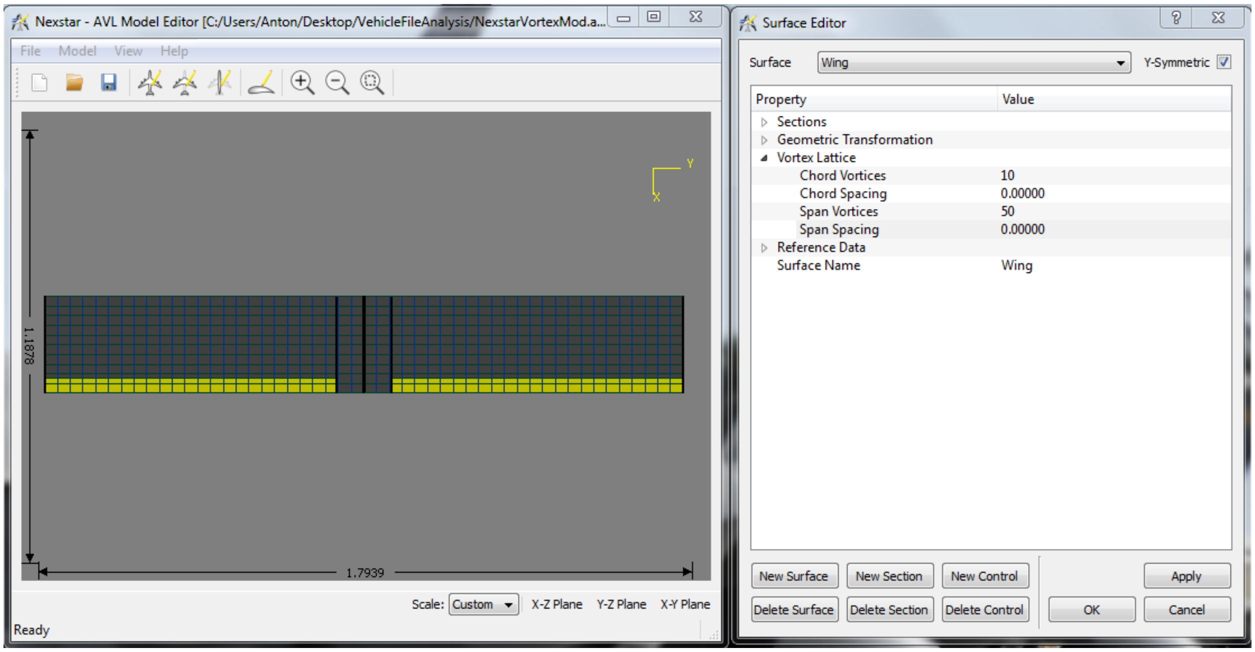

Untitled Aircraft - AVL Model Editor [C:/Users/Anton/Documents/Grad School/Noctua/Noctua...

Scale: Custom

Ready

Figure 141 Vortex Spacing

Sspace and Cspace can be any number between -3 and 3. Each whole number represents a different type of distribution. Sspace distributes the number of nodes from the first section to the last section. Cspace distributes the number of nodes from the leading edge to the trailing edge. Figure 142 below depicts vortex sheet of 10 chord nodes, and 50 span nodes (Nchord = 10, Nspan = 50). The nodes are evenly spaced between both the leading and trailing edges, and the centerline to wing tip (Sspace = 0, Cspace = 0).



Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Chord Vortices	10
Chord Spacing	0.00000
Span Vortices	50
Span Spacing	0.00000
Reference Data	
Surface Name	Wing

Figure 142 Vortex Settings

Examine the following table and read through the explanation of vortex spacing from the avl help text file “avl_doc.txt”.

Sspace/Cspace	spacing	First Section/LE -------------Last Section/TE
3.0	equal	| | | | | | | | |
2.0	sine	|| | | | | | | |
1.0	cosine	|| | | | | | ||
0.0	equal	| | | | | | | | |
-1.0	cosine	|| | | | | | ||
-2.0	-sine	| | | | | | | ||
-3.0	equal	| | | | | | | | |

“An intermediate parameter value will result in a blended distribution.

The most efficient distribution (best accuracy for a given number of vortices) is usually the cosine (1.0) chordwise and spanwise. If the wing does not have a significant chord slope discontinuity at the centerline, such as a straight, elliptical, or slightly tapered wing, then the – sine (-2.0) distribution from root to tip will be more efficient. This is equivalent to a cosine distribution across the whole span. The basic rule is that a tight chordwise distribution is needed at the leading and trailing edges, and a tight spanwise distribution is needed wherever the circulation is changing rapidly, such as taper breaks, and especially at flap breaks and wingtips.

A number of vortex-spacing rules must be followed to get good results from AVL, or any other vortex-lattice method:

1) In a standard VL method, a trailing vortex leg must not pass close to a downstream control point, else the solution will be garbage. In practice, this means that surfaces which are lined up along the x direction (i.e. have the same or nearly the same y,z coordinates), MUST have the same spanwise vortex spacing. AVL relaxes this requirementby employing a finite core size for each vortex on a surface which is influencing a control point in another aurface (unless the two surfaces share the same INDEX declaration). This feature can be disabled by setting the core size to zero in the OPER sub-menu, Option sub-sub-menu, command C. This reverts AVL to the standard AVL method.

2) Spanwise vortex spacings should be "smooth", with no sudden changes in spanwise strip width. Adjust Nspan and Sspace parameters to get a smooth distribution. Spacing should be bunched at dihedral and chord breaks, control surface ends, and especially at wing tips. If a single spanwise spacing distribution is specified for a surface with multiple sections, the spanwise distribution will be fudged as needed to ensure that a point falls exactly on the section location. Increase the number of spanwise points if the spanwise spacing looks ragged because of this fudging.

3) If a surface has a control surface on it, an adequate number of chordwise vortices Nchord should be used to resolve the discontinuity in the camberline angle at the hingeline. It is possible to define the control surface as a separate SURFACE entity. Cosine chordwise spacings then produce bunched points exactly at the hinge line, giving the best accuracy. The two surfaces must be given the same INDEX and the same spanwise point spacing for this to work properly. Such extreme measures are rarely necessary in practice, however. Using a single surface with extra chordwise spacing is usually sufficient.

4) When attempting to increase accuracy by using more vortices, it is in general necessary to refine the vortex spacings in both the spanwise AND in the chordwise direction. Refining only along one direction may not converge to the correct result, especially locally wherever the bound vortex line makes a sudden bend, such as a dihedral break, or at the center of a swept wing. In some special configurations, such as an unswept planar wing, the chordwise spacing may not need to be refined at all to get good accuracy, but for most cases the chordwise spacing will be significant."

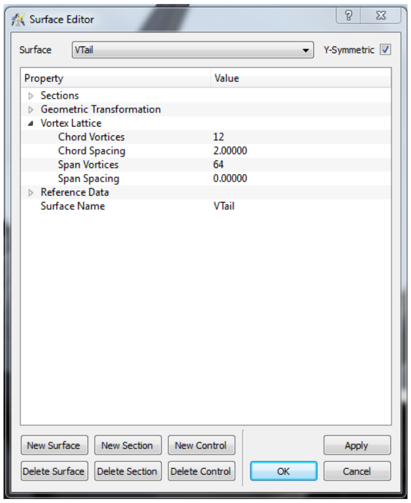
Finding the correct settings for each surface is kind of an art. The process begins by setting the spanwise spacing, followed by number of spanwise nodes, chordwise spacing, and number of chordwise nodes.

4.1.10.1. Adjusting Vortex Lattice Settings

1) Spanwise spacing is somewhat straightforward. There needs to be nodes clustered at wing tips, and dihedral breaks. The description from the avl text files also says that there needs to be nodes clustered at each chord break and when a control surface begins and ends; however, since there will be a new section each time both occur, and we use surfaces of multiple sections, AVL will automatically fudge the distribution so that there are nodes at each new section. Thus there will be nodes at each chord break and control

surface as a result of the way we build surfaces and there is no need to focus on these areas when adjusting spanwise spacing.

2) Each surface's vortex settings are set by default to the settings shown below in Figure 143.



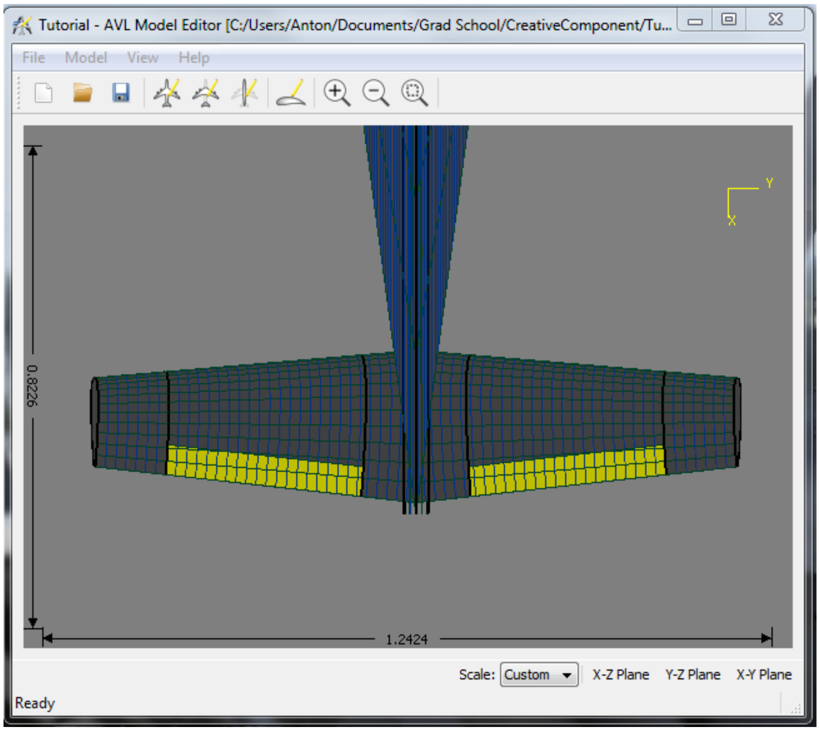
Surface: VTail
Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	2.00000
Span Vortices	64
Span Spacing	0.00000
▷ Reference Data	
Surface Name	VTail

Figure 143 AVLEditor Default Vortex Settings

Choose a surface to begin. Do not choose the fuselage surfaces. Fuselage surfaces are sized differently.

Set the point of view of the AVL model to bird's eye view (usually x-y plane) and examine the default distribution. It might be necessary to temporarily translate the fuselage surfaces down and out of the way of viewing the surface's nodes. This can be done in "Geometric Transformation" in the Surface Editor.

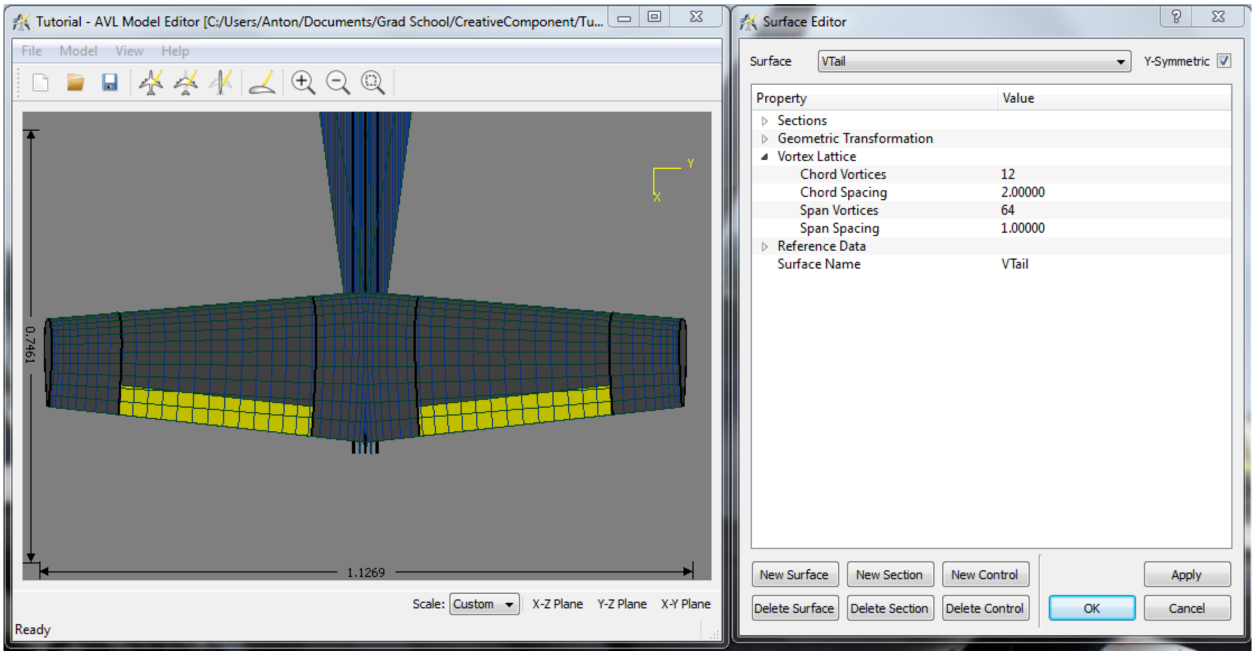


Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...

Figure 144 Default Vortex Spanwise Spacing

Determine the span spacing required for the surface. There needs to be nodes clustered at wing tips and dihedral breaks. If the surface has dihedral that begins a couple sections out from the center of the surface then it might be necessary to split the surface into two different surfaces. Set the span spacing in the Surface Editor under the "Vortex Lattice" drop down menu.

Figure 144 depicts the initial default vortex distribution on a V-Tail. In Figure 144 above the V-Tail has dihedral beginning at the center; therefore, nodes should be bunched at the center in addition to the tip. The default span wise distribution is 0, so there were no nodes bunched at the center or the tips.


Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...
File Model View Help
0.7461
1.1269
Y
X
Scale: Custom
X-Z Plane Y-Z Plane X-Y Plane
Ready
Surface Editor
New Surface New Section New Control Apply
Delete Surface Delete Section Delete Control OK Cancel

Surface VTail Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	2.00000
Span Vortices	64
Span Spacing	1.00000
▷ Reference Data	
Surface Name	VTail

Figure 145 Refined Vortex Spanwise Spacing

As shown in Figure 145 changing the span spacing to equal 1 caused vortex nodes to bunch at the center and tips of the V-Tail.

3) The correct number of vortices is hard to determine. Having too many vortices can be just as detrimental as not having enough vortices. The spanwise distribution needs to be smooth all the way across the surface until it nears the tip or dihedral section.

Zoom in and determine if the Spanwise Vortices are bunching up too much or not enough at the appropriate sections. Figure 146 below shows an example of too many vortices (left) and not enough vortices (right) at the tip of a V-Tail.

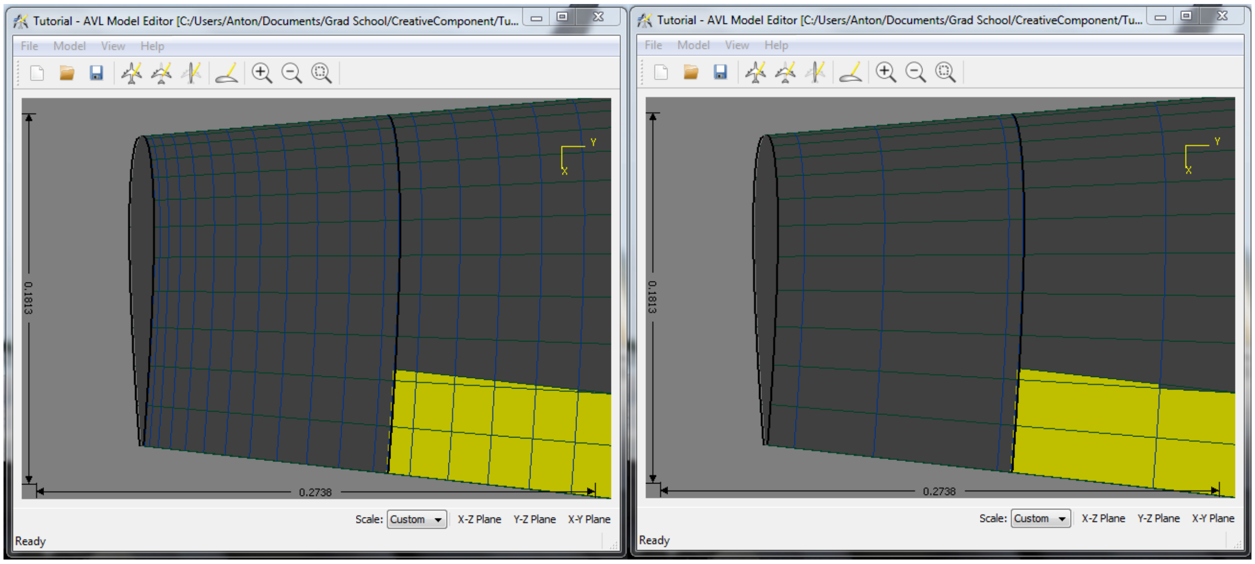
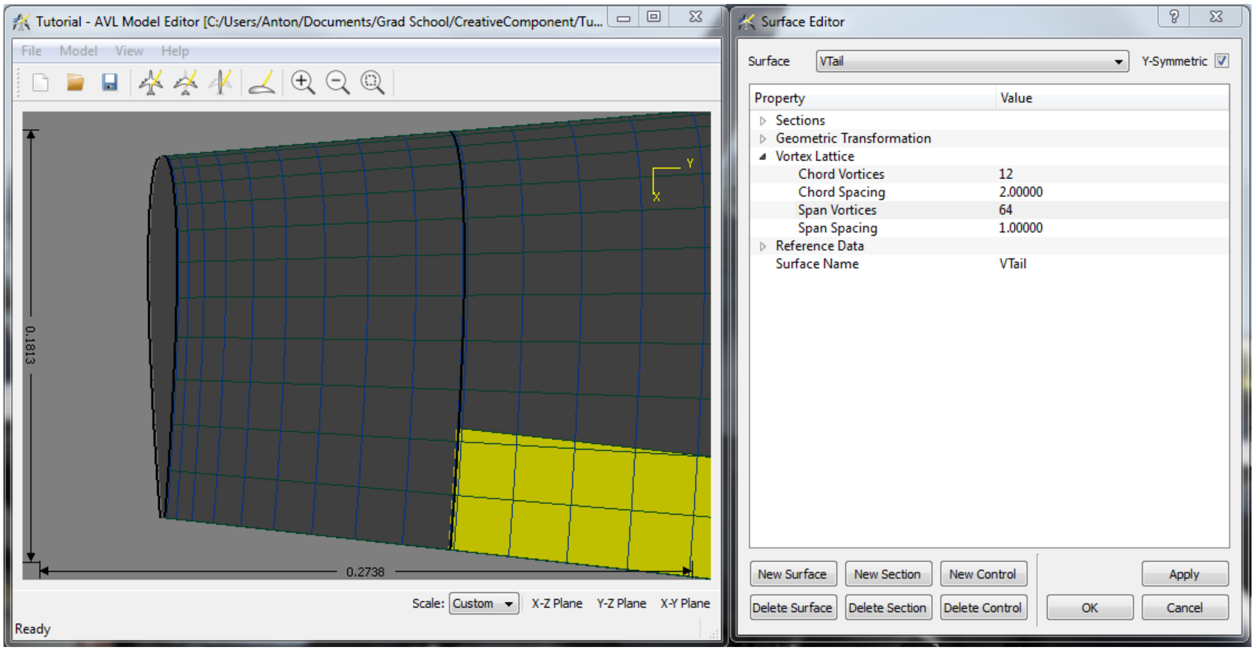

Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...
File Model View Help
0.1813
0.2738
Y
X
Scale: Custom
X-Z Plane Y-Z Plane X-Y Plane
Ready

Figure 146 Vortex Spanwise Node Numbering

In the case of Figure 145 the default number of Spanwise Vortices, 64, was sufficient.

Figure 147 depicts the V-Tail example's final vortex distribution at the tip.



Surface: VTail
Y-Symmetric ☑

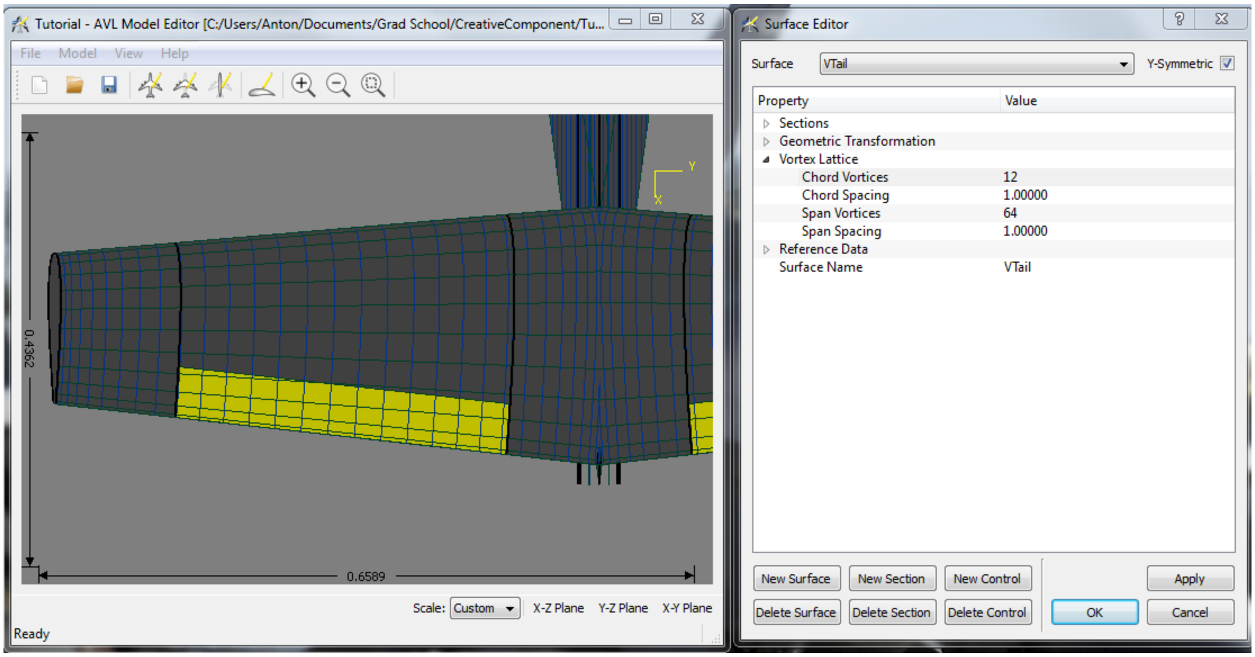
Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Chord Vortices	12
Chord Spacing	2.00000
Span Vortices	64
Span Spacing	1.00000
Reference Data	
Surface Name	VTail

Figure 147 Acceptable Vortex Node Bunch

4) Set the chord spacing to 1, so that there will be vortices clustered at the leading and trailing edges.

5) Setting the number of chord vortices is similar to the number of span vortices except that extra attention is needed for the areas with control surfaces. Avoid bunching up too many vortices at the leading and trailing edges. Try to adjust the number so that there are control points that fall on the hinge of any control surfaces.

In the V-Tail example that has been shown in the previous figures the default number of chords was sufficient as depicted below in .



Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...

Surface Editor

Surface: VTail ▾ Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	1.00000
Span Vortices	64
Span Spacing	1.00000
▷ Reference Data	
Surface Name	VTail

Figure 148 Vortex Chordwise Node Numbering

6) Go back to 2) and repeat the process for all the surfaces that remain, excluding fuselage surfaces. Read Section 4.1.10.2 to see another example of adjusting vortex settings, or continue on to Section 4.1.10.3 Fuselage Surface Vortex Lattice Settings.

4.1.10.2. Wing Example

In this example the wing has no dihedral, does taper, and has ailerons and flaps. Beginning with the default vortex settings there are no clustered vortices at the tip or the center.

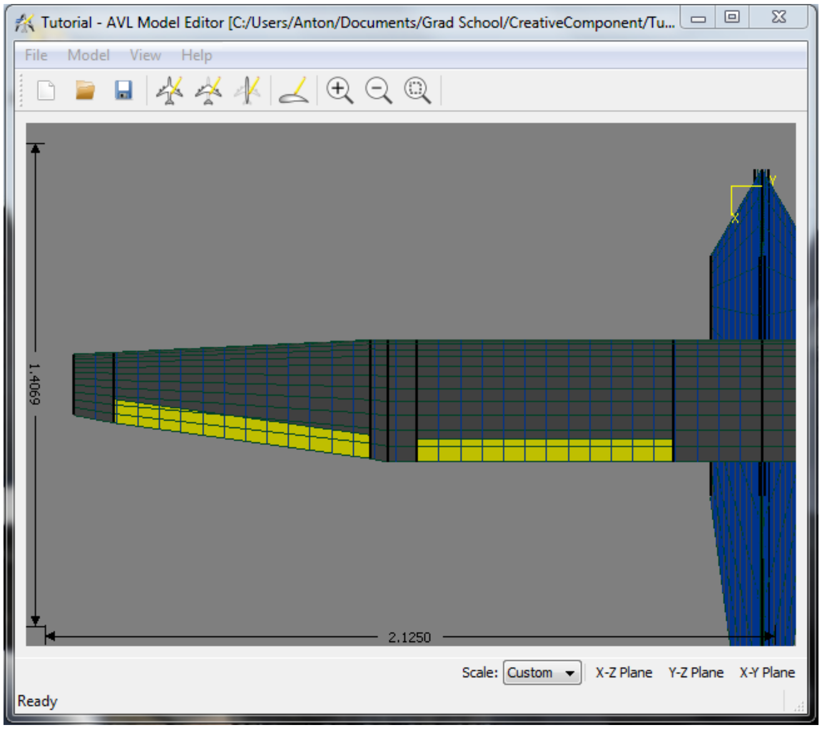

Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...
File
Model
View
Help
1.4069
2.1250
Scale:
Custom
X-Z Plane
Y-Z Plane
X-Y Plane
Ready

Figure 149 Example Vortex Settings Default Distribution

The wing does not have dihedral and the centerline does not taper so there is no need for clustered vortices at the center. The spanwise spacing is set to -2 so that the vortex cluster occurs at the wing tip only.

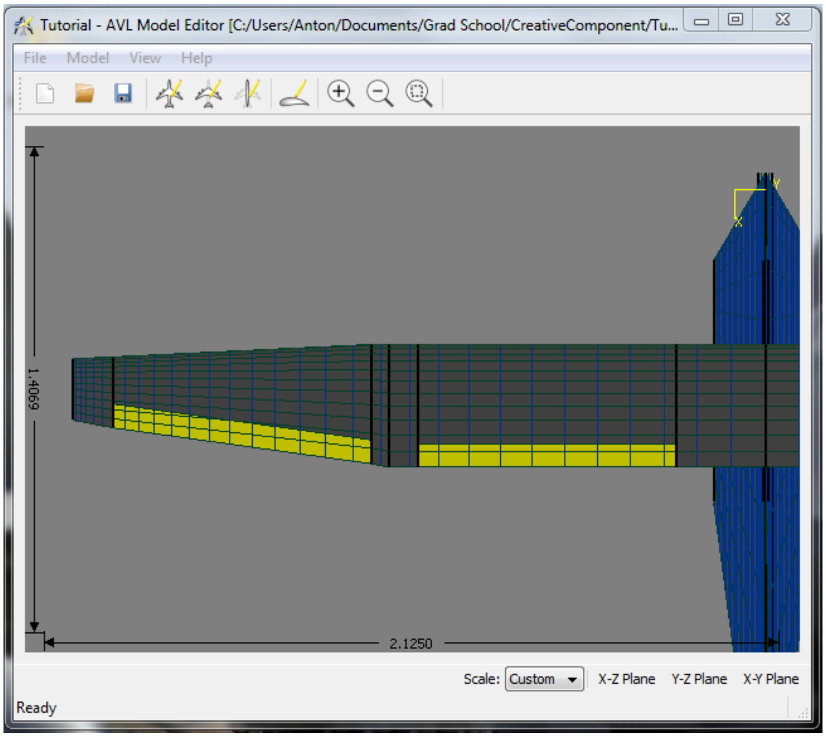

Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...
File
Model
View
Help
1.4069
2.1250
Scale:
Custom
X-Z Plane
Y-Z Plane
X-Y Plane
Ready

Figure 150 Example Vortex Settings Spanwise Spacing

The default number of spanwise vortices is sufficient. The chordwise spacing is set to 1 so that the vortices are clustered at the leading and trailing edges.

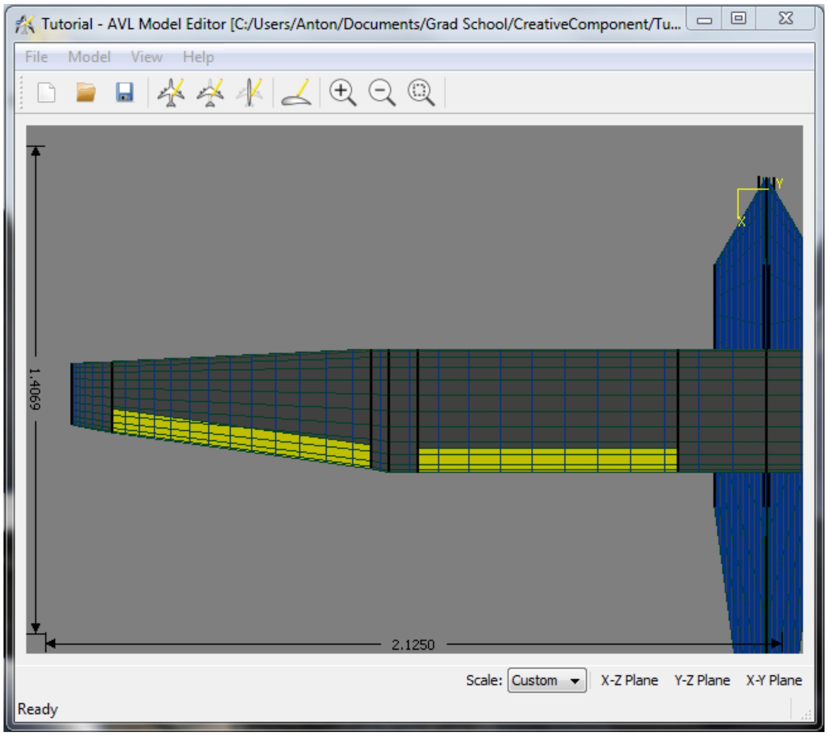
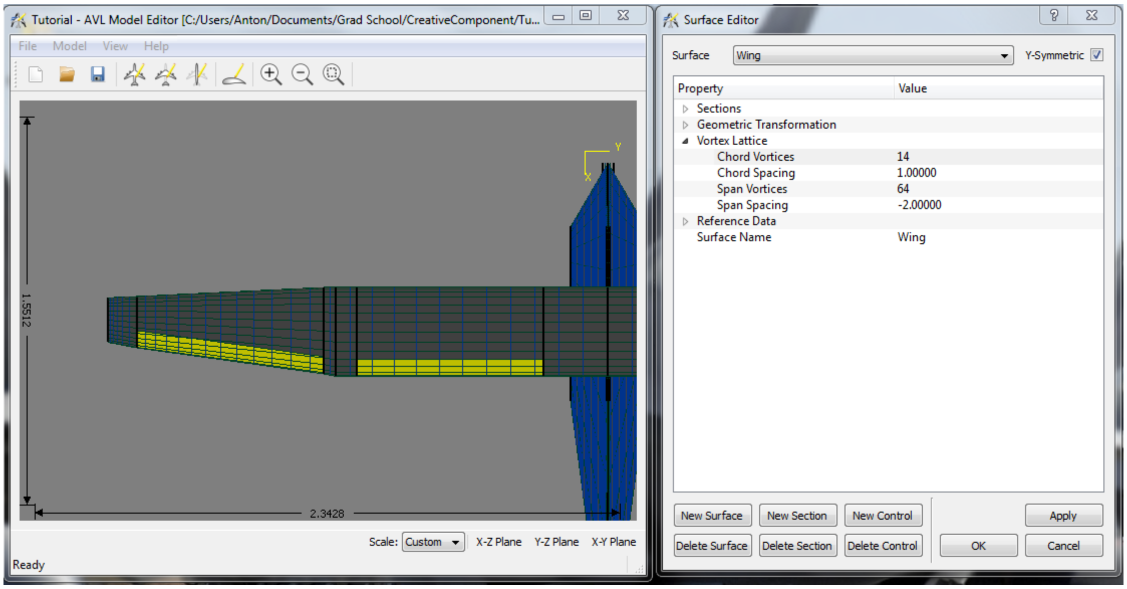

Tutorial - AVL Model Editor [C:/Users/Anton/Documents/Grad School/CreativeComponent/Tu...
File
Model
View
Help
1.4069
2.1250
Y
X
Scale:
Custom
X-Z Plane
Y-Z Plane
X-Y Plane
Ready

Figure 151 Example Vortex Settings Chordwise Spacing

The number of chordwise vortices needs to be increased so that there are vortices at the hinge of the Flaps (control surface on the non-tapered area of the wing). The number of chordwise vortices is increased to 14.



Surface: Wing Y-Symmetric ☑

Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Chord Vortices	14
Chord Spacing	1.00000
Span Vortices	64
Span Spacing	-2.00000
Reference Data	
Surface Name	Wing

Figure 152 Example Vortex Settings Final Distribution

4.1.10.3. Fuselage Surface Vortex Lattice Settings

Use the following steps to set the Vortex Lattice Settings for the fuselage surfaces:

1) The fuselage surfaces don’t need very many vortices. Cut the number of span vortices on both surfaces down to 20 or less.

2) Set the chord spacing to 1.

3) Set the span spacing for FuselageTop to -2.0.

4) Set the span spacing for FuselageSide to -1.0, so that both sections 1 and the final section have clustered vortices.

5) Click Apply. By now the warning of exceeding the limit for the number of vortices should no longer appear. If it does go back through the surfaces and decrease the number of vortices, and adjust the chordwise and spanwise spacing so that the grid is still acceptable.

4.2. Alpha Sweep

1) In AVLEditor click Model > AVL Analysis…

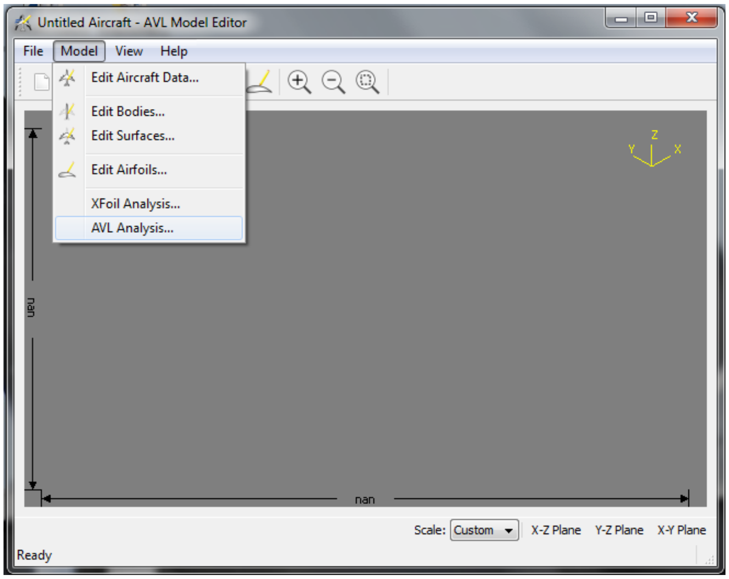


Figure 153 AVLEditor AVL Analysis

2) In the AVL Output window click File > Start AVL…

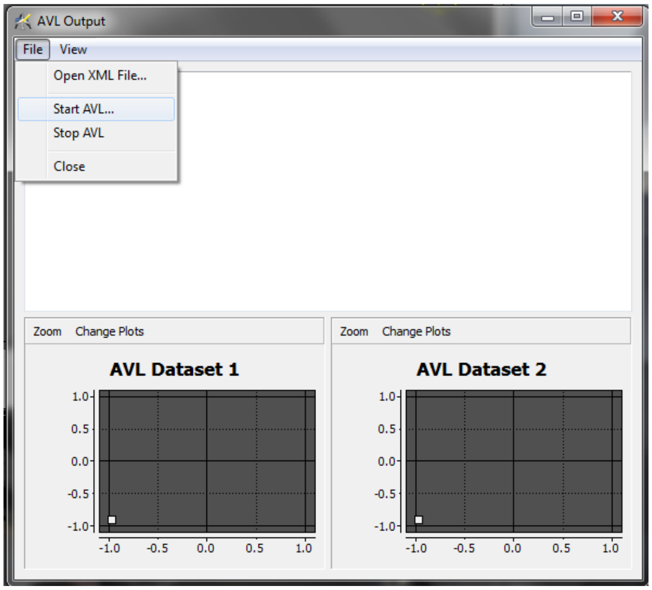
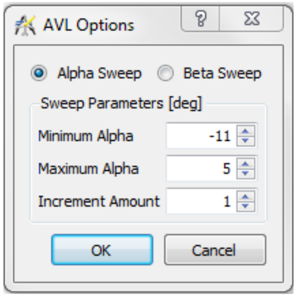


Figure 154 AVL Output Window

3) In the AVL Options window make sure that “Alpha Sweep” is selected.

4) Set the minimum and maximum alpha so that the range of alphas is the same as the linear range that was determined from the 2D lift slope of the wings. Use an alpha of at least 10 degrees for the maximum alpha.



● Alpha Sweep ○ Beta Sweep

Sweep Parameters [deg]

Minimum Alpha	-11
Maximum Alpha	5
Increment Amount	1

Figure 155 Alpha Sweep Range

If the wings have incidence subtract the incidence from the minimum and maximum alphas and use those values as the minimum and maximum. This is done so that the alpha sweep actually analyzes the range of angles of attack that correspond with the linear range.

5) Click OK to run the AVL analysis.

6) Save the alpha file in aircraft folder.

4.3. Simulator File

The simulator file is a plain text file that declares all the variables necessary for the simulator to function. Cloud cap offers one document, “Piccolo Simulator”, which details all the different components that the simulator file can contain.

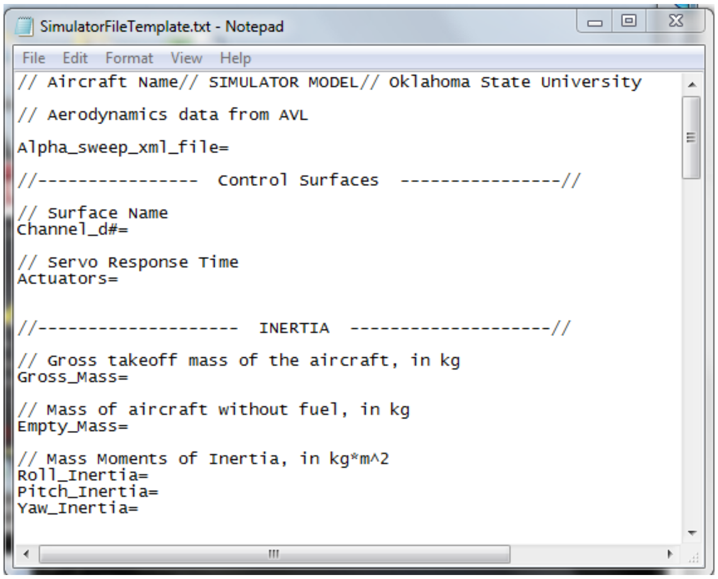
The simulator mechanics section details specifics about simulator files. Make sure to understand the following simulator file format requirements:

1) Lines with “//” designate comment lines that the simulator ignores.

2) When a file is called to be loaded the file name must include the file extension.

3) There cannot be any spaces in any of the lines that define parameters for the simulator. If there is a space the simulator will not read the line.

Open the “SimulatorFileTemplate” text file in the new aircraft folder to begin the process.


SimulatorFileTemplate.txt - Notepad
File Edit Format View Help

```
// Aircraft Name// SIMULATOR MODEL// Oklahoma State University

// Aerodynamics data from AVL

Alpha_sweep_xml_file=

//----------------  Control Surfaces  ----------------//

// Surface Name
Channel_d#=

// Servo Response Time
Actuators=


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=

// Mass of aircraft without fuel, in kg
Empty_Mass=

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=
Pitch_Inertia=
Yaw_Inertia=
```

Figure 156 Simulator File Control Surfaces and Inertia

1) Name the aircraft in the first line.

2) Enter the name of the alpha file from the AVLEditor model.

3) Assign each control surface that is contained in the alpha file to its corresponding piccolo servo number.

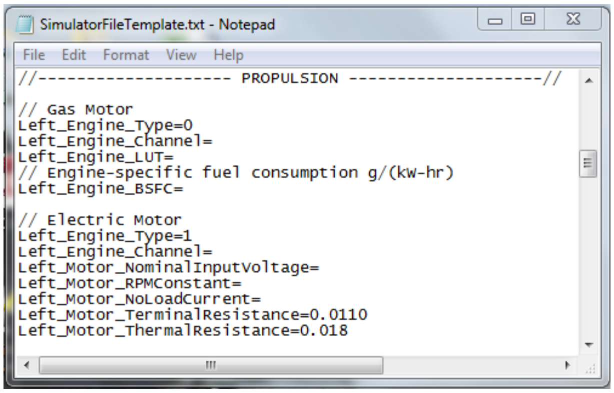
The control surface number designated in the alpha file is to take the place of the "#" directly after "d". The corresponding servo number is to be declared after the equals sign. No spaces.

4) Declare the name of the servo response file that is desired. There should be 5 actuator files in the new aircraft folder that can be chosen to represent the servo response time of the servos on the aircraft. They are FastActuator, SloppyActuator, SlowActuator, StandardActuator, and VerySlowActuator.

5) Enter the full and empty weights of the aircraft with respect to fuel weight, in kg. The simulator uses these two numbers to calculate the fuel mass.

6) Enter the mass moments of inertia in kg/m^2. There are other optional inertia parameters that can be entered if desired. Refer to Piccolo Simulator pg. 21 for the entire list of inertial commands that are available.

7) The simulator file template propulsion section includes the required parameters for gas and electric motors.


SimulatorFileTemplate.txt - Notepad
File Edit Format View Help

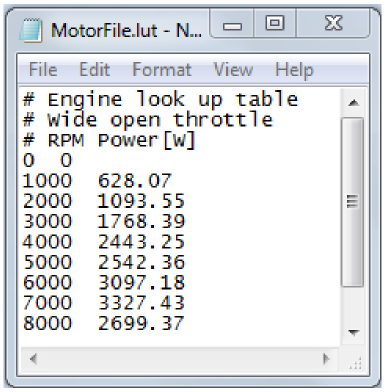
```
//-------------------- PROPULSION --------------------//

// Gas Motor
Left_Engine_Type=0
Left_Engine_Channel=
Left_Engine_LUT=
// Engine-specific fuel consumption g/(kW-hr)
Left_Engine_BSFC=

// Electric Motor
Left_Engine_Type=1
Left_Engine_Channel=
Left_Motor_NominalInputVoltage=
Left_Motor_RPMConstant=
Left_Motor_NoLoadCurrent=
Left_Motor_TerminalResistance=0.0110
Left_Motor_ThermalResistance=0.018
```

Figure 157 Simulator File Propulsion Section

Delete the section that does not represent the propulsion system of the aircraft.

8) Enter the values for the required engine parameters.

9) Gas motors require a motor file.



```
# Engine look up table
# Wide open throttle
# RPM Power[W]
0   0
1000  628.07
2000  1093.55
3000  1768.39
4000  2443.25
5000  2542.36
6000  3097.18
7000  3327.43
8000  2699.37
```

Figure 158 Motor File

The motor file is the power curve or RPM versus Power (W). If there is no data for the engine a power curve will have to be made artificially. Use the motor file template, “MotorFile.lut, as the initial motor file for the model. After the entire model is complete run the model in a SiL or HiL and adjust the values of power until the simulator accurately estimates the achievable RPM at full throttle. This will not be an exact propulsion model; however, it will at least constrain the aircraft to the maximum power limitations it will have in real life. Note that there must be a “knee” in the power curve, where the power decreases with an increase in RPM.

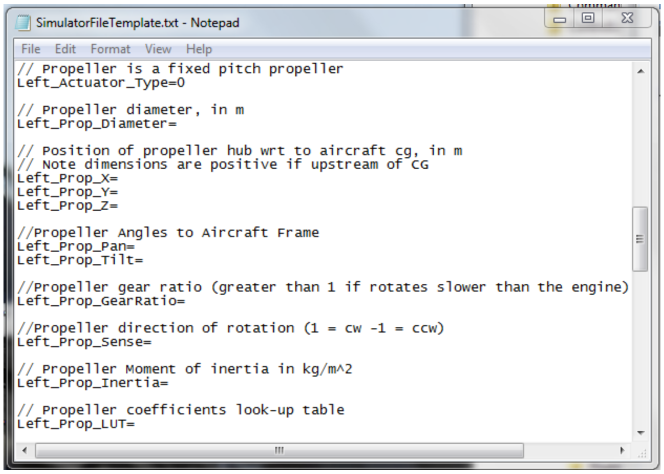
If there is power curve data use the motor file template, “MotorFile.lut”, in the new aircraft folder and replace the values with the correct power curve values. Note that there must be a “knee” in the power curve, where the power decreases with an increase in RPM.

10) For gas motors enter the specific fuel consumption. The simulator uses this number along with motor data to estimate the fuel consumption during simulated flights. The

piccolo has parameters that can be entered to estimate the consumption of amp hours for electric motors; however, those parameters are set on the piccolo side not the simulator side.

11) For both electric and gas motors there are additionally parameters that can be specified such as governors. Refer to “Piccolo Simulator” pgs. 23-25 for the additional commands.

12) Define all of the propeller parameters that are in the simulator file template. The default propeller type is fixed pitch propeller, type 0. For additional available propeller parameters refer to the Cloud Cap document “Piccolo Simulator” pgs. 26-27.



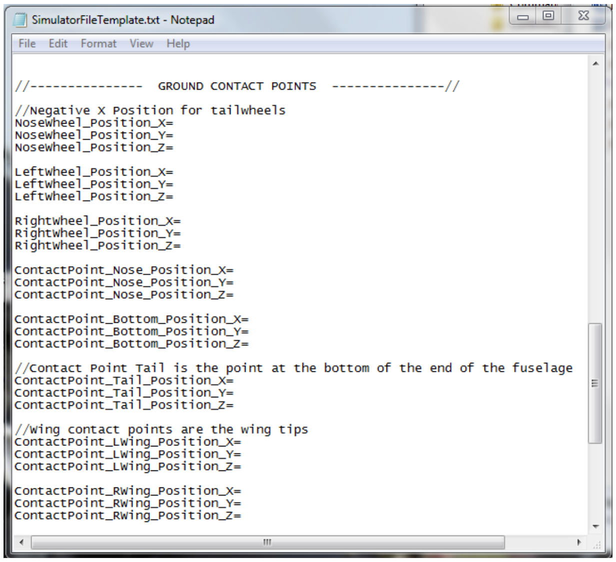
```
// Propeller is a fixed pitch propeller
Left_Actuator_Type=0

// Propeller diameter, in m
Left_Prop_Diameter=

// Position of propeller hub wrt to aircraft cg, in m
// Note dimensions are positive if upstream of CG
Left_Prop_X=
Left_Prop_Y=
Left_Prop_Z=

//Propeller Angles to Aircraft Frame
Left_Prop_Pan=
Left_Prop_Tilt=

//Propeller gear ratio (greater than 1 if rotates slower than the engine)
Left_Prop_GearRatio=

//Propeller direction of rotation (1 = cw -1 = ccw)
Left_Prop_Sense=

// Propeller Moment of inertia in kg/m^2
Left_Prop_Inertia=

// Propeller coefficients look-up table
Left_Prop_LUT=
```

Figure 159 Simulator File Propeller Definitions

13) The propeller model requires a propeller file. The propeller file is to contain coefficient of power, coefficient of thrust, and efficiency data for advance ratios from at least -1 to 1. CCT Matlab has a propeller modeling program that will load propeller geometry files and generate propeller files.

Use Cloud Cap’s guide “Creating Propeller Models” to create a propeller file unless experimental data or other programs are available to do so.

14)



```
//---------------  GROUND CONTACT POINTS  ---------------//

//Negative X Position for tailwheels
NoseWheel_Position_X=
NoseWheel_Position_Y=
NoseWheel_Position_Z=

LeftWheel_Position_X=
LeftWheel_Position_Y=
LeftWheel_Position_Z=

RightWheel_Position_X=
RightWheel_Position_Y=
RightWheel_Position_Z=

ContactPoint_Nose_Position_X=
ContactPoint_Nose_Position_Y=
ContactPoint_Nose_Position_Z=

ContactPoint_Bottom_Position_X=
ContactPoint_Bottom_Position_Y=
ContactPoint_Bottom_Position_Z=

//Contact Point Tail is the point at the bottom of the end of the fuselage
ContactPoint_Tail_Position_X=
ContactPoint_Tail_Position_Y=
ContactPoint_Tail_Position_Z=

//Wing contact points are the wing tips
ContactPoint_LWing_Position_X=
ContactPoint_LWing_Position_Y=
ContactPoint_LWing_Position_Z=

ContactPoint_RWing_Position_X=
ContactPoint_RWing_Position_Y=
ContactPoint_RWing_Position_Z=
```

Figure 160 Simulator File Ground Contact Points

Define the ground contact points. Remember that the locations are referenced to the center of gravity location specified in the alpha file, and recall that the positive direction of the simulator axes are different than that of the AVLEditor axes.

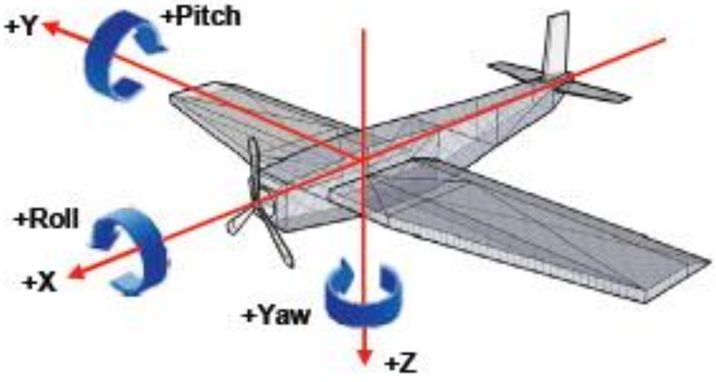
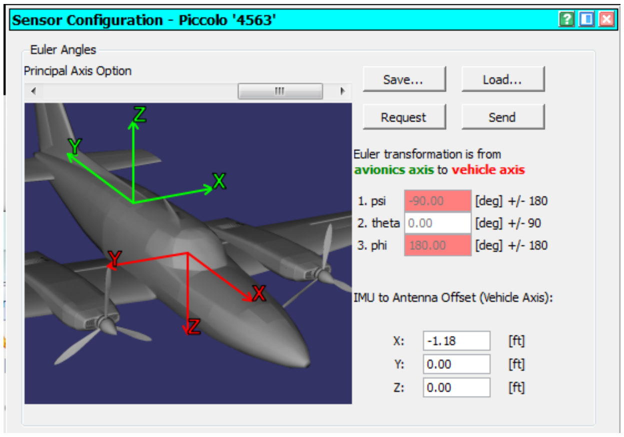


Figure 161 Simulator Axes

Figure 161 is from "Piccolo Simulator" pg. 6. The x and z axes positive directions are different than that of AVLEditor. Do not copy locations from the CG/Inertia model unless they were taken with the same axis definitions.

There are additional ground contact points that can be defined. Refer to “Piccolo Simulator” pgs. 36-38.

15) Define the piccolo orientation with respect to the aircraft body axes. The direction of the axes of the piccolo is printed on the physical unit itself. It can be helpful to use the Sensor Configuration window in PCC to help define the euler angles correctly. The values of the angles in Sensor Configuration need to be inverted when entered into the simulator file.



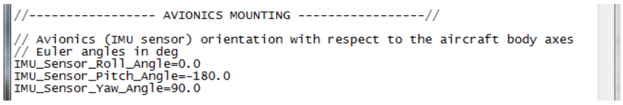
Euler transformation is from avionics axis to vehicle axis

1. psi -90.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 180.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]

Figure 162 Piccolo Orientation Sensor Configuration Window

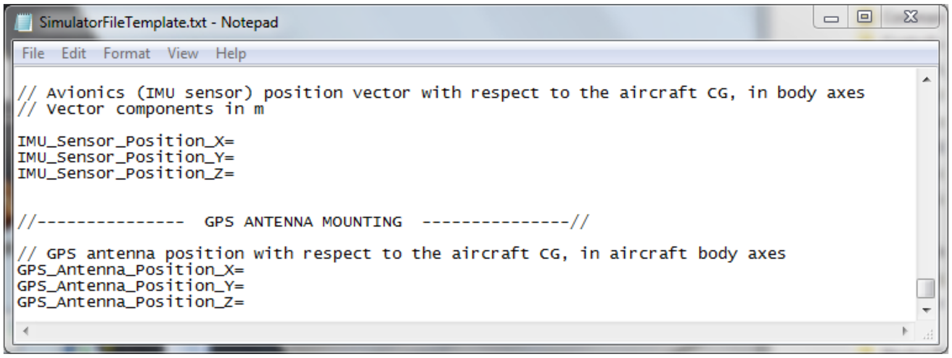


```
//---------------- AVIONICS MOUNTING -----------------//

// Avionics (IMU sensor) orientation with respect to the aircraft body axes
// Euler angles in deg
IMU_Sensor_Roll_Angle=0.0
IMU_Sensor_Pitch_Angle=-180.0
IMU_Sensor_Yaw_Angle=90.0
```

Figure 163 Simulator File Piccolo Orientation Definitions

Figure 162 and Figure 163 are examples of a configuration. The sensor configuration window was used to define the angles. They were defined as -90 psi (yaw angle) and 180 phi (pitch angle). When they were entered into the simulator file they were inverted to 90, and -180.

16)


SimulatorFileTemplate.txt - Notepad
File Edit Format View Help

```
// Avionics (IMU sensor) position vector with respect to the aircraft CG, in body axes
// Vector components in m

IMU_Sensor_Position_X=
IMU_Sensor_Position_Y=
IMU_Sensor_Position_Z=

//---------------  GPS ANTENNA MOUNTING  ---------------//

// GPS antenna position with respect to the aircraft CG, in aircraft body axes
GPS_Antenna_Position_X=
GPS_Antenna_Position_Y=
GPS_Antenna_Position_Z=
```

Figure 164 Simulator File Piccolo and GPS Locations

Define the location of the piccolo with respect to the aircraft's center of gravity. Again make sure to use the simulator axes directions when defining locations.

17) Define the location of the GPS antenna with respect to the aircraft's center of gravity.

4.4. Initial Vehicle Parameters

1) Open the Simulator.

2) Click File > Open Aircraft.

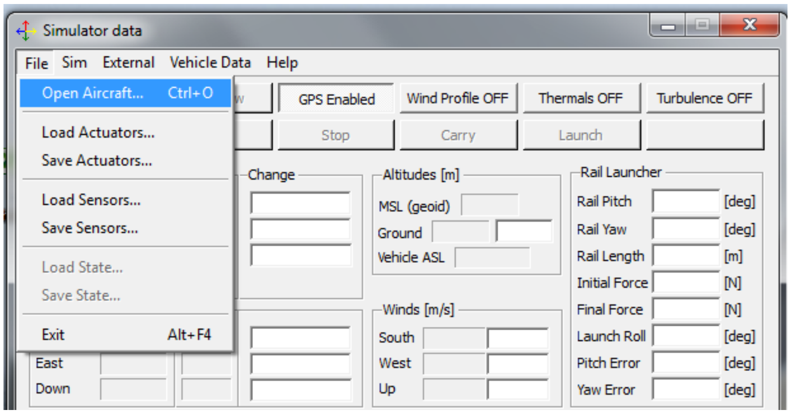


Figure 165 Open Aircraft

3) Load the aircraft's simulator file.

4) Click Vehicle Data > Fixed Wing Gen 2/3…

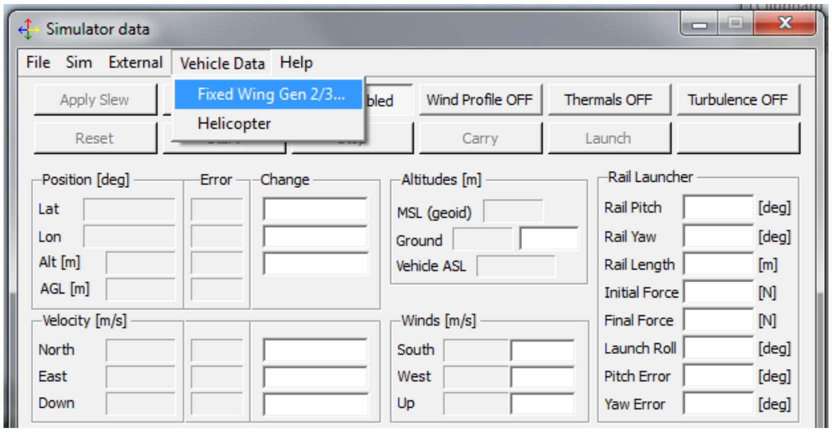


Figure 166 Vehicle Data

5) Save the vehicle gains file.

6) Open the vehicle gains file and look over the estimated vehicle parameters.

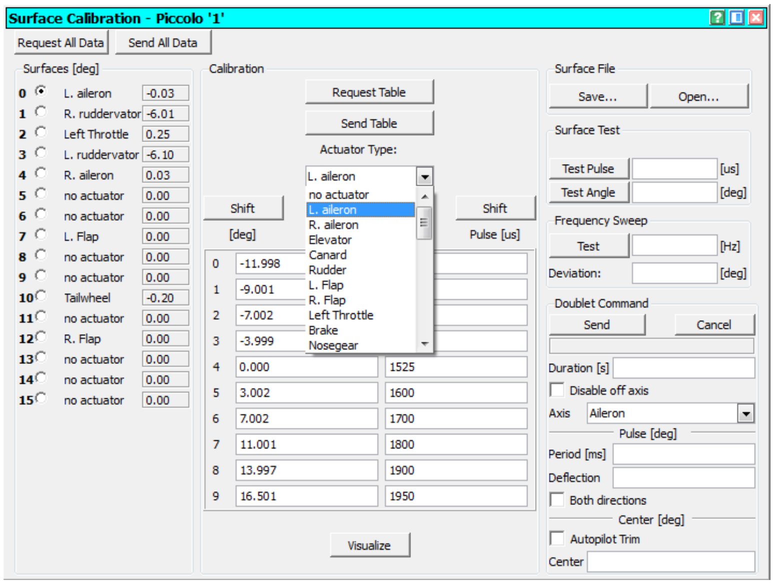
7) If there are any issues such as a parameter calculated as 0 or infinity refer to Chapter 3 Simulator Mechanics to help determine the cause of the miscalculation. It may be necessary to manually calculate any miscalculated parameters.

4.5. Autopilot Initial Setup

4.5.1. Control Surfaces Calibration

1) With the autopilot power on and communicating with PCC, open the Surface Calibration Window in PCC.

2) For each control surface:



Surface Calibration - Piccolo '1'

Surfaces [deg]

0	●	L. aileron	-0.03
1	○	R. ruddervator	-6.01
2	○	Left Throttle	0.25
3	○	L. ruddervator	-6.10
4	○	R. aileron	0.03
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	-0.20
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	
1	-9.001	
2	-7.002	
3	-3.999	
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File

Surface Test

Test Pulse [us]
Test Angle [deg]

Frequency Sweep

Test [Hz]
Deviation: [deg]

Doublet Command

Duration [s]
□ Disable off axis
Axis: Aileron

Pulse [deg]

Period [ms]
Deflection
□ Both directions

Center [deg]

□ Autopilot Trim
Center

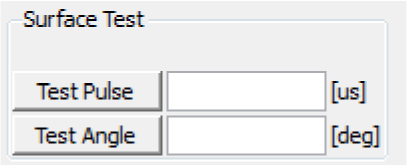
Figure 167 Surface Calibration

a) Select the servo number that the control surface is wired into.

b) From the drop down menu select the actuator type that describes the function of the control surface.

3) After all of the servos have been assigned their corresponding control surfaces click “Send All Data” to send the parameters to the piccolo.

4) For each control surface other than throttle:



Surface Test

Test Pulse [us]
Test Angle [deg]

Figure 168 Surface Test

a) Use the “Test Pulse” command to command pulse widths and measure the corresponding angle of deflection of the control surface.

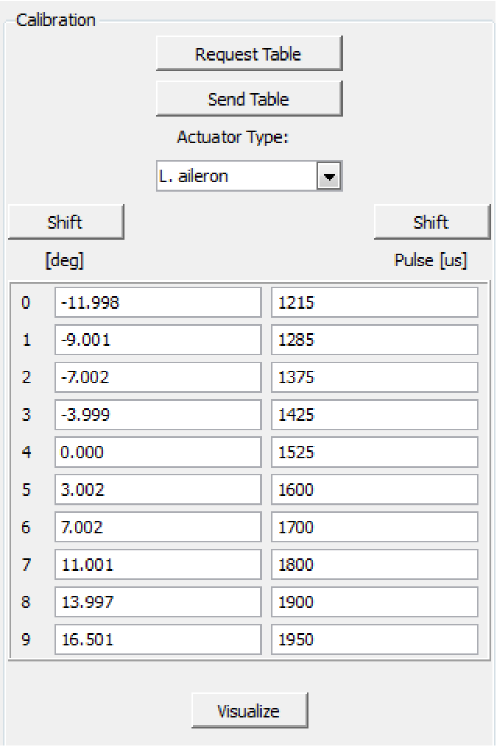
b) Find the pulse width that yields a physical maximum deflection in one of the directions of deflection and iterate increasing or decreasing pulse width signals until the physical maximum deflection is reached in the other direction while measuring the angle of deflection for each pulse width signal. Cloud cap stated that the range of operable pulse widths is between 1000 – 2000 us in a phone conversation. It has been found that the autopilot will command over and under those limits; however, proceed with caution. If a servo requires a pulse width out of that range it might be necessary to remount the servo so that it operates within 1000 – 2000 us.

c) A couple options for measuring the angle of deflection of the control surfaces are to use the angle pro digital inclinometer, or the magnet angle measurement tool. The deflections can also be measured using a ruler and trigonometry.

d) The surface table requires 10 points. Make sure that maximum, minimum, and 0 degree deflections are 3 of the 10 points.

e) Choose the 7 other points to be the best that represent the movement of the servo. Fewer points are required for range of pulse widths that are linear with servo deflection. Usually the servo deflections are non linear near the maximum and minimum deflections. Try to have more points in the non linear areas.

f) Enter the 10 pulse widths with their corresponding angles of deflection into the table.



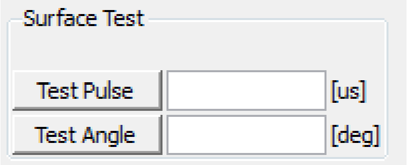
Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Figure 169 Surface Calibration Table

g) Click "Send Table" to send the points to the piccolo.

5) For throttle:



a) Use the "Test Pulse" command to command pulse widths and measure their corresponding RPMs.

b) While testing the pulse width response of the motor make sure that the motor is running the propeller that aircraft is intended to fly with. If a different propeller is used the throttle will need to be re calibrated for each propeller because the response will vary.

c) Find the pulse width that corresponds with the lowest throttle setting desired. For electric this can be the pulse width that kills throttle. For gas motors usually the lowest throttle setting is set at just before the motor stalls out and dies. Minimum throttle can be specified later to keep the piccolo from ever commanding that low or the lowest pulse width can be set just above that. Cloud cap states that the operable range of pulse widths is between 1000 – 2000 us REFERENCE. It has been found that the autopilot will command over and under those limits; however, proceed with caution. If a servo requires a pulse width out of that range it might be necessary to remount the servo so that it operates within 1000 – 2000 us.

d) Measure the RPMs that correspond with each pulse width.

e) Iterate increasing or decreasing pulse width signals until the motor reaches its physical maximum RPMs.

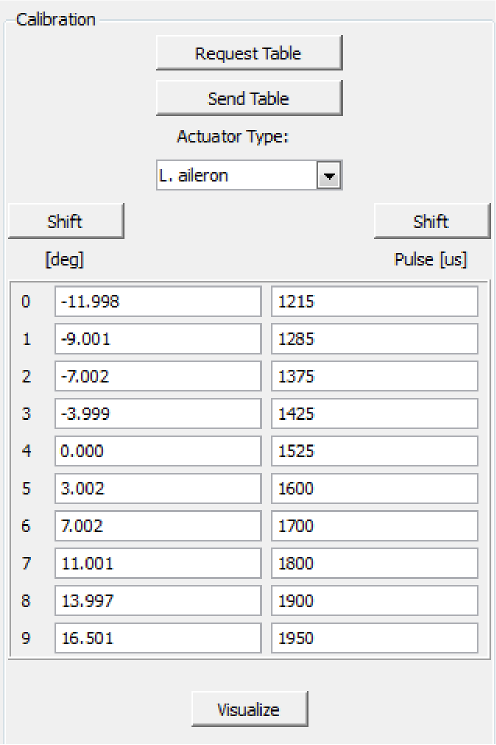
f) Calculate the throttle setting, 0 – 1, for each pulse width as the RPMs cubed of the current pulse width divided by the maximum RPMs cubed. RPMs cubed is used because RPM cubed is proportional to power and the percent throttle is supposed to represent the amount of power that throttle setting will add to the system. The autopilot multiplied the throttle setting by the maximum engine power to estimate the energy rate that a throttle setting will add to the system.

$$Throttle = \frac{RPM^3}{RPM_{max}^3}$$

g) 2 of the 10 points need to be the minimum and maximum RPMs. The minimum throttle setting does not have to be 0.

h) Choose the other 8 points that will produce the closest fit to the entire range of pulse widths versus throttle. The linear areas of the test results don't need as many points as the non linear areas.

i) Enter the 10 pulse widths with their corresponding throttle settings into the table. The throttle settings should be between 0 and 1.



Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

j) Click "Send Table" to send the points to the piccolo.

k) If the propeller model is accurate the propeller file can be used to calculate the amount of power that the engine was able to produce at the maximum achievable RPMs. If desired to do so calculate the power required to spin the propeller at the maximum RPMs with Equation 21 below.

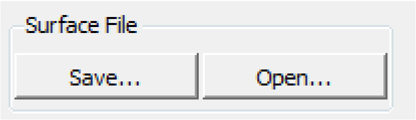
$$P_{req} = C_p \rho n^3 D^5$$

Equation 21 Power Required

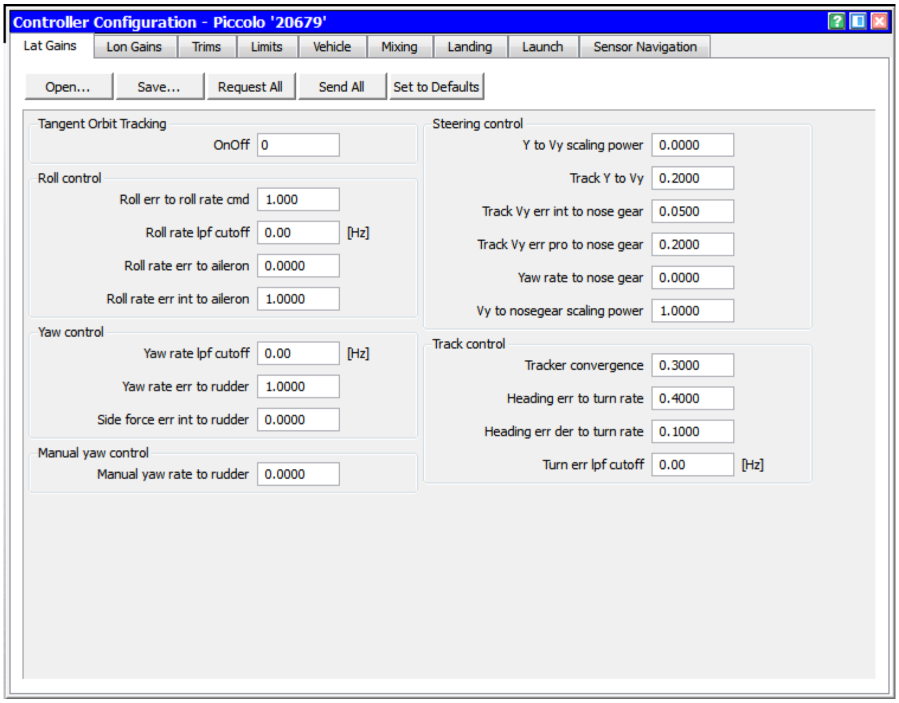
Note that the value of Cp should be taken from the advance ratio, J, of 0 assuming that the throttle calibration tests took place statically. Also note that the RPMs, n, need to be converted to rotations per second before being cubed. Use the resulting power calculated as the max engine power in the vehicle parameters.

6) After all the surfaces have been calibrated double check each control surface's deflections. Command a "Test Angle" rather than a "Test Pulse", and measure the resulting deflection of the control surface.

7) Save the surface calibration to an xml file.



4.5.2. Initial Lateral Control Gains



Tangent Orbit Tracking

OnOff 0

Roll control

Roll err to roll rate cmd 1.000

Roll rate lpf cutoff 0.00 [Hz]

Roll rate err to aileron 0.0000

Roll rate err int to aileron 1.0000

Yaw control

Yaw rate lpf cutoff 0.00 [Hz]

Yaw rate err to rudder 1.0000

Side force err int to rudder 0.0000

Manual yaw control

Manual yaw rate to rudder 0.0000

Steering control

Y to Vy scaling power 0.0000

Track Y to Vy 0.2000

Track Vy err int to nose gear 0.0500

Track Vy err pro to nose gear 0.2000

Yaw rate to nose gear 0.0000

Vy to nosegear scaling power 1.0000

Track control

Tracker convergence 0.3000

Heading err to turn rate 0.4000

Heading err der to turn rate 0.1000

Turn err lpf cutoff 0.00 [Hz]

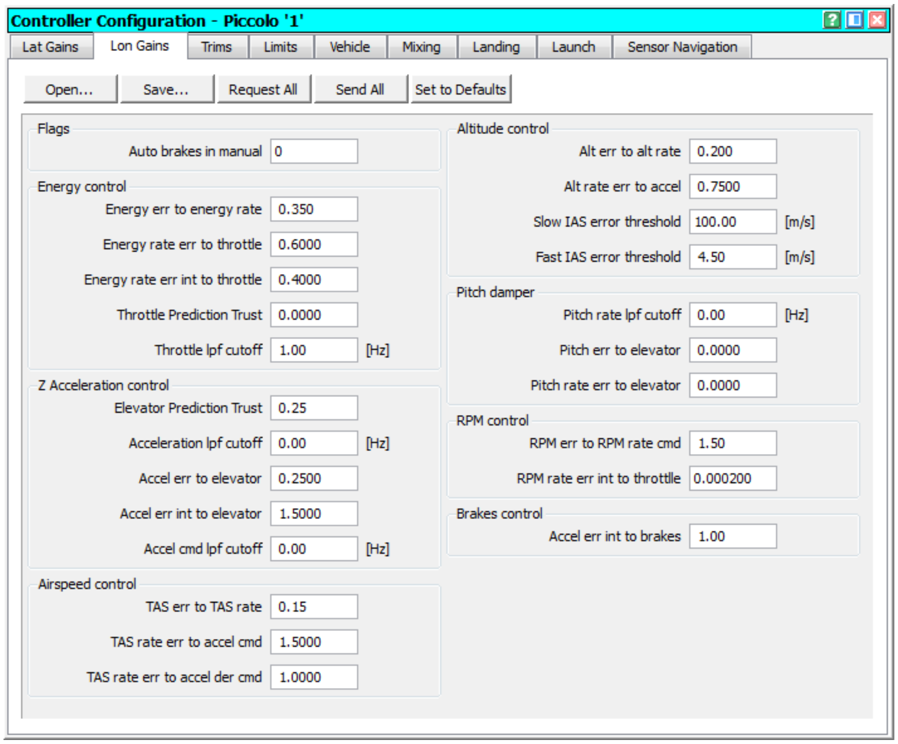
Figure 170 Lateral Control Gains Window

1) With the autopilot powered on and communicating with PCC, open the Controller Configuration window in PCC and click on the “Lat” tab.

2) Click “Set to defaults” to set the lateral control gains to their default values.

3) The default lateral gains use the yaw rate control loop to command rudder deflections; thus, the “Side force err int to rudder” is zero. If it is desired to use the Side force control loop rather than the Yaw Rate control loop set the “Side force err int to rudder” to 1.0. Refer to Section 5.8.2 on Rudder Control for more information.

4.5.3. Initial Longitudinal Control Gains



Flags

Auto brakes in manual: 0

Energy control

Energy err to energy rate: 0.350

Energy rate err to throttle: 0.6000

Energy rate err int to throttle: 0.4000

Throttle Prediction Trust: 0.0000

Throttle lpf cutoff: 1.00 [Hz]

Z Acceleration control

Elevator Prediction Trust: 0.25

Acceleration lpf cutoff: 0.00 [Hz]

Accel err to elevator: 0.2500

Accel err int to elevator: 1.5000

Accel cmd lpf cutoff: 0.00 [Hz]

Airspeed control

TAS err to TAS rate: 0.15

TAS rate err to accel cmd: 1.5000

TAS rate err to accel der cmd: 1.0000

Altitude control

Alt err to alt rate: 0.200

Alt rate err to accel: 0.7500

Slow IAS error threshold: 100.00 [m/s]

Fast IAS error threshold: 4.50 [m/s]

Pitch damper

Pitch rate lpf cutoff: 0.00 [Hz]

Pitch err to elevator: 0.0000

Pitch rate err to elevator: 0.0000

RPM control

RPM err to RPM rate cmd: 1.50

RPM rate err int to throttle: 0.000200

Brakes control

Accel err int to brakes: 1.00

Figure 171 Longitudinal Gains Window

1) With the autopilot powered on and communicating with PCC, open the Controller Configuration window in PCC and click on the “Lon” tab.

2) Click “Set to defaults” to set the longitudinal control gains to their default values.

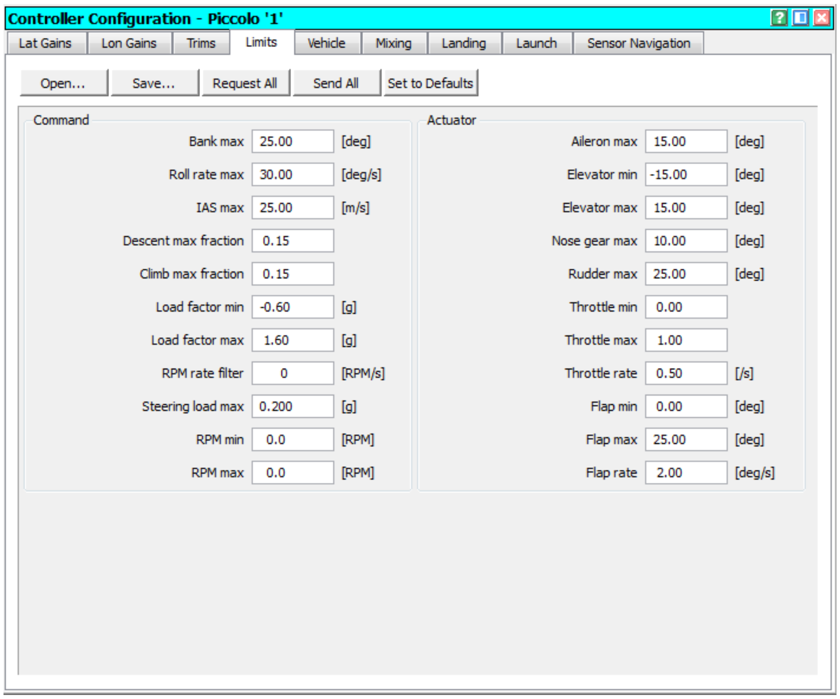
3) By default there is no pitch damper; thus, all of the pitch damper gains are zero. If it is desired to use a pitch damper the initial gains will have to be determined in hardware in the loop simulations. Additionally it is likely that the gains will need tuning in real flight as well.

4) The “Slow IAS error threshold” is very large by default because typically it is not desired for the autopilot to go into slow airspeed mode just because the airspeed is below the target airspeed. Such scenarios could create periods of large losses in

altitude. The slow airspeed mode is mainly designed for scenarios where the aircraft is climbing and the airspeed falls below the minimum airspeed.

5) Alter the “Fast IAS error Threshold” as desired. From Section 5.8.5, the “Fast IAS error Threshold” determines when the autopilot will switch into fast airspeed mode (Lon Mode 3). Usually if the aircraft’s maximum descent fraction is set appropriately there should not be a need for the aircraft to enter fast airspeed mode in shallow descents.

4.5.4. Limits



Command

Bank max	25.00	[deg]
Roll rate max	30.00	[deg/s]
IAS max	25.00	[m/s]
Descent max fraction	0.15	
Climb max fraction	0.15	
Load factor min	-0.60	[g]
Load factor max	1.60	[g]
RPM rate filter	0	[RPM/s]
Steering load max	0.200	[g]
RPM min	0.0	[RPM]
RPM max	0.0	[RPM]

Actuator

Aileron max	15.00	[deg]
Elevator min	-15.00	[deg]
Elevator max	15.00	[deg]
Nose gear max	10.00	[deg]
Rudder max	25.00	[deg]
Throttle min	0.00	
Throttle max	1.00	
Throttle rate	0.50	[/s]
Flap min	0.00	[deg]
Flap max	25.00	[deg]
Flap rate	2.00	[deg/s]

Figure 172 Limits Window

Setting the limits is entirely up to the user. There are default limits; however, the limits heavily depend on the capabilities of the aircraft that the autopilot is installed in, and how the user intends to fly it. Figure 172 depicts the default limit values.

It is not recommended to increase the limits of the load factor max and min. Even though the definitions of the “Load fact min” and “Load factor max” make it seem like these limits are concerned with the amount of g force the structure can handle these limits also dictate the z – acceleration limiter. If these values are too large the autopilot can hit the elevators to hard and flip the aircraft. If it is desired to increase these limits do so with caution.

If it is desired to set an “RPM min” and/or an “RPM max” know that doing so will enable RPM Control. Make sure to fully understand the implications of RPM control and how it affects longitudinal control of the aircraft. There are catastrophic scenarios that can result from improperly setting RPM limits.

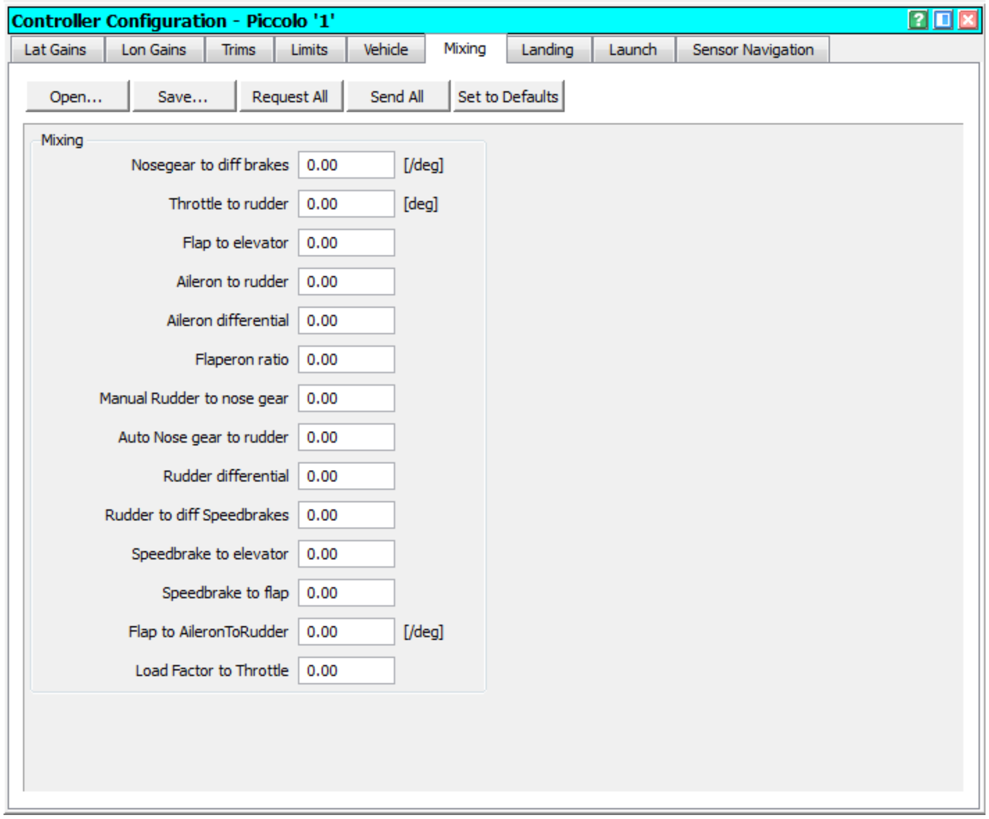
Typically the control surface limits are set according to the aerodynamic design of the aircraft. The highest and lowest deflection values in the control surface’s actuator tables, which should represent the physical deflection limitations of the control surface, also factor in to the maximum and minimum deflection commands. Regardless of what the control surface limits are set as in the “Limits” tab the autopilot will not command beyond the range of deflections that are defined in the control surface calibration tables. Additionally the autopilot will not command control surface deflections that violate a specific control surface’s limit set in the “Limits” tab. There is one caveat to these conditions however. The “Throttle max” and “Throttle min” limits will be ignored in pre-launch. Pre-launch throttle is set in the “Launch” tab, and will command any throttle command that exists in the throttle control surface calibration table.

If there are multiple elevator control surfaces make sure that the maximum elevator deflection is at least low enough that there will not be differential elevator deflection due to one elevator having a lower physical maximum deflection than the other. Also make sure that the aileron, and rudder limits do not allow differential deflection to occur due to an imbalance in physical limitations as well. Check to make sure that this type of error won’t exist in the aircraft’s configuration as there are many different mixed control surface actuator types where this scenario could cause an adverse effect. For example in a V-Tail, ruddervator configuration, if a left ruddervator’s negative deflection could physically deflect farther than the right ruddervator’s positive deflection and the rudder max was set so that the left ruddervator could deflect to its physical maximum deflection then the extra deflection by the left ruddervator would induce a pitch and roll moment. Imagine a scenario where a left ruddervator could deflect negative 20 degrees maximum based on physical limitations, and a right ruddervator could only deflect positive 15 degrees based on physical limitations, but the “Rudder max” was set to 20 degrees. In such a scenario the autopilot could command 20 degrees rudder deflection and if it did the left ruddervator would have 5 degrees of deflection that would create a positive pitching moment, and a negative roll moment.

Note that the aileron, nose gear, and rudder max limits are magnitude deflection limits, so they are applied as positive and negative deflection limits.

4.5.5. Mixing

The actuator types have built in control surface mixture where the control surfaces act as dual surfaces; however, there are also additional mixing options in the “Mixing” tab of the Controller Configuration window in PCC.



Mixing

Nosegear to diff brakes 0.00 [/deg]

Throttle to rudder 0.00 [deg]

Flap to elevator 0.00

Aileron to rudder 0.00

Aileron differential 0.00

Flaperon ratio 0.00

Manual Rudder to nose gear 0.00

Auto Nose gear to rudder 0.00

Rudder differential 0.00

Rudder to diff Speedbrakes 0.00

Speedbrake to elevator 0.00

Speedbrake to flap 0.00

Flap to AileronToRudder 0.00 [/deg]

Load Factor to Throttle 0.00

Figure 173 Mixing Window

Note that the flaperon mixture mixes flap deflection to ailerons, so the ailerons can be used as flaps. It does not mix ailerons into flaps so the flaps can be used as ailerons. If it is desired to use a positive flaperon ratio, where the ailerons deflect with flaps, then the flap effectiveness vehicle parameter will have to be re calculated to account for the extra aileron deflection. This can be done manually, using the equations in Section 3.14 that describe how the simulator estimates the flap effectiveness, or it can be done through AVLEditor. If it is to be re calculated by the AVLEditor do the following:

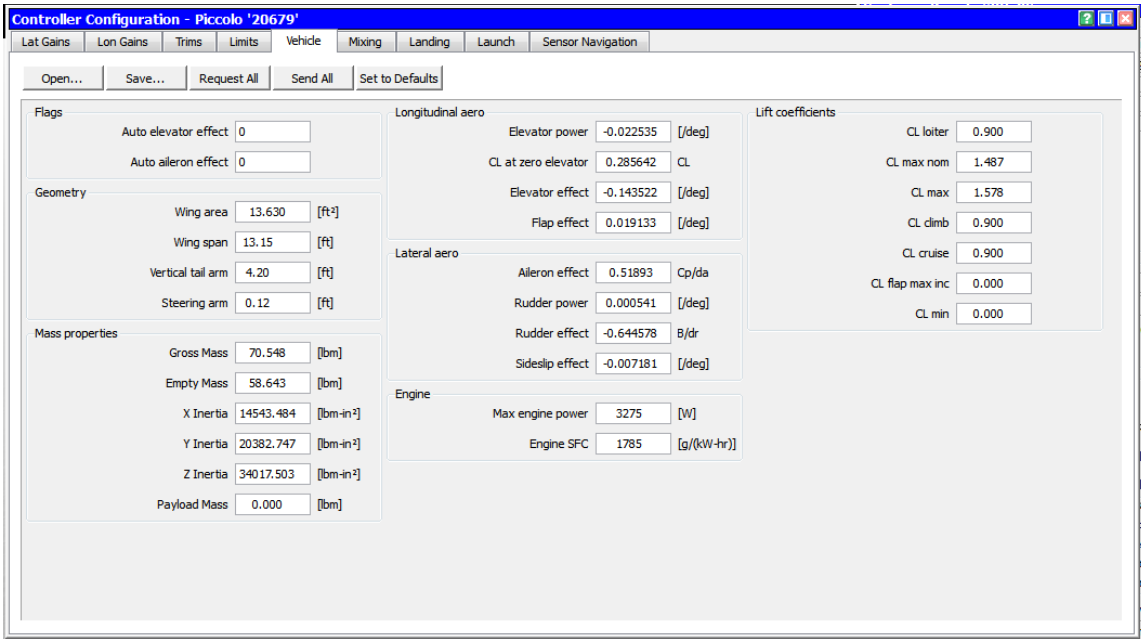
1) Add a control surface to the sections that contain the ailerons, and name the control surface “Flap”. If the surface is not mirrored then the left and right control surfaces will have to be named “Lflap” and “Rflap” respectively. Make sure that the chord fraction of the flaps matches the chord fraction of the ailerons at each section.

2) Re-run AVL analysis.

3) Generate a Vehicle Parameters file from the new alpha file.

4) The flap effectiveness calculation should now be correct, and it should be the only value that changed in the Vehicle Parameters file.

4.5.6. Vehicle Gains



Flags

Auto elevator effect 0

Auto aileron effect 0

Geometry

Wing area 13.630 [ft²]

Wing span 13.15 [ft]

Vertical tail arm 4.20 [ft]

Steering arm 0.12 [ft]

Mass properties

Gross Mass 70.548 [lbm]

Empty Mass 58.643 [lbm]

X Inertia 14543.484 [lbm-in²]

Y Inertia 20382.747 [lbm-in²]

Z Inertia 34017.503 [lbm-in²]

Payload Mass 0.000 [lbm]

Longitudinal aero

Elevator power -0.022535 [/deg]

CL at zero elevator 0.285642 CL

Elevator effect -0.143522 [/deg]

Flap effect 0.019133 [/deg]

Lateral aero

Aileron effect 0.51893 Cp/da

Rudder power 0.000541 [/deg]

Rudder effect -0.644578 B/dr

Sideslip effect -0.007181 [/deg]

Engine

Max engine power 3275 [W]

Engine SFC 1785 [g/(kW-hr)]

Lift coefficients

CL loiter 0.900

CL max nom 1.487

CL max 1.578

CL climb 0.900

CL cruise 0.900

CL flap max inc 0.000

CL min 0.000

Figure 174 Vehicle Gains Window

1) With the autopilot powered on and communicating with PCC, open the Controller Configuration window in PCC and click on the “Vehicle” tab.

2) Click “Open…” to load the vehicle parameters from the vehicle parameter file generated by the simulator.

3) Click “Send All” to send the parameters to the autopilot.

4) If there are any parameters that were calculated incorrectly by the simulator input them manually; however, beware that the units in the “Vehicle” tab are different than the units used by the simulator and the vehicle parameter examples. Any parameter that was estimated as /rad is to be entered in /deg.

5) If the “Max engine power” is not accurate, which would be the case if the motor file was artificially constructed, make sure to change it to the best known value. If the maximum engine power was calculated earlier during the throttle calibration tests make sure to enter that value manually.

6) Alter any of the C_L values as needed so that the aircraft does not attempt to fly at airspeeds that it is not intended to. The simulator tends to overestimate CL max. Recall that CL max is used to represent the lift force measured by the autopilot’s z – accelerometer at stall speed. It is up to the user as to whether or not to use a conservative value as a precaution.

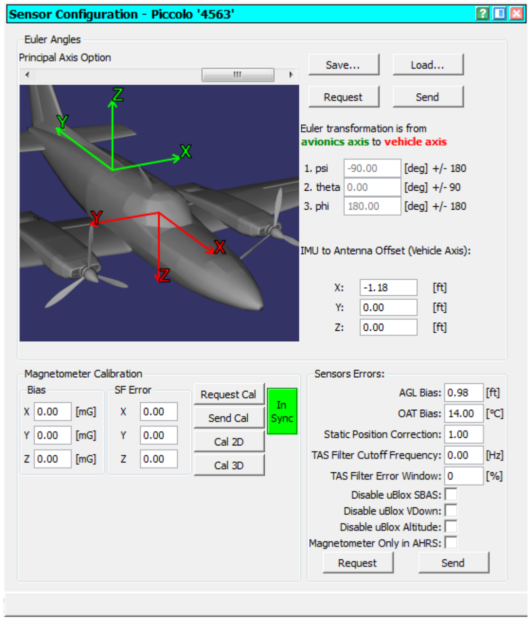
4.5.7. Payload IO Settings

The payload IO settings designate the function of the input/output lines to the piccolo. Input/output lines that are used to command control surface deflections are to be defined as “Default.” There are many additional options for configuring sensor input and outputs on different IO lines. Refer to the Cloud Cap document “PccUsersGuide” pgs. 84-94 for details on all of the available options.

Note that the IO numbers do not always match the corresponding servo numbers in version 2.1.4.i. For example on the Piccolo SL 555 (original) the IO number of Servo 5 is 6, and the IO number of Servo 6 is 5. Refer to the Cloud Cap document “Piccolo External Interface” for additional details. The document lists the pin numbers, IO numbers, and servo numbers of all of the lines for every piccolo unit.

4.5.8. Sensor Configuration

1) With the autopilot powered on and communicating with PCC, open the Sensor Configuration window in PCC.



Sensor Configuration - Piccolo '4563'

Euler Angles

Principal Axis Option

Euler transformation is from avionics axis to vehicle axis

1. psi -90.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 180.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]

Magnetometer Calibration

Bias			SF Error	
X	0.00	[mG]	X	0.00
Y	0.00	[mG]	Y	0.00
Z	0.00	[mG]	Z	0.00

Sensors Errors:

AGL Bias: 0.98 [ft]
OAT Bias: 14.00 [°C]
Static Position Correction: 1.00
TAS Filter Cutoff Frequency: 0.00 [Hz]
TAS Filter Error Window: 0 [%]
Disable uBlox SBAS: □
Disable uBlox VDown: □
Disable uBlox Altitude: □
Magnetometer Only in AHRS: □

Figure 175 Sensor Configuration

2) Drag the scroll bar at the top and find the orientation that the piccolo is mounted at with respect to the vehicle axis. If the piccolo is mounted at an orientation that does not match any of the options then the piccolo needs to be remounted.

3) Input the x, y, and z distance of the GPS antenna from the back of the autopilot as documented in Cloud Cap's PccUsersGuide, pg. 99. Use the direction of the vehicle axes (red axes) to define the x, y, and z directions.

4) Define the sensor errors and settings as desired. Refer to PccUsersGuide pgs. 99-100 for further details on each setting. Note that the "OAT Bias" is used to estimate the outside air temperature in the absence of an OAT sensor. This is important as the autopilot uses

the outside air temperature to calculate the air density and thus calculate the true airspeed (Section 5.1.2).

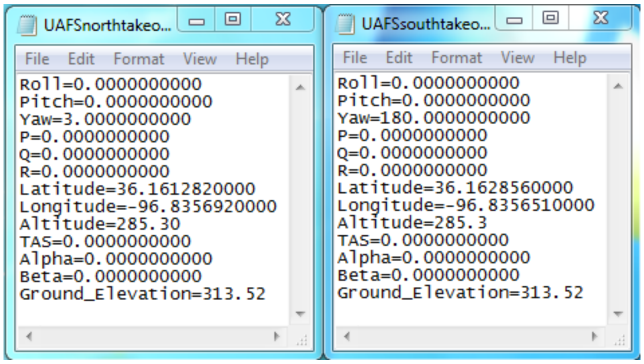
5) If there is a magnetometer it will need to be calibrated. Use Cloud Cap's "Magnetometer Integration Guide" to calibrate the magnetometer.

4.5.9. Launch and Land Settings

Setting up the launch and land settings is entirely up to the user's discretion as they are very dependable on aircraft capabilities and the expected performance. Usually the land settings are initially set from flying hardware in the loop simulations of the aircraft. The launch settings can be set from hardware or software in the loop simulations; however, it is usually best to analyze the manual takeoff of the aircraft in real flights to help set the launch settings properly.

Make sure to fully understand all of the land and launch parameters, they are defined in Cloud Cap's "PccUsersGuide" pgs. 105-109. It is important to note that for wheeled landings, of aircraft without weight on wheel or agl sensors, if the autopilot believes it has passed through the touchdown waypoint within the y and z maximum errors it will assume it has landed even if the altitude measurements are wrong and the aircraft is not actually on the ground. The autopilot does use the z – accelerometer to help determine touchdown; however, it will assume it has landed in the aforementioned scenario even if there is no spike in the z – accelerometer measurements. As such it might be wise to set the engine kill time to a negative value so that the autopilot will not issue a "kill engine" command. This way the manual pilot can immediately takeover and take corrective action with throttle still available.

4.6. Simulator State File



```
Roll=0.0000000000
Pitch=0.0000000000
Yaw=3.0000000000
P=0.0000000000
Q=0.0000000000
R=0.0000000000
Latitude=36.1612820000
Longitude=-96.8356920000
Altitude=285.30
TAS=0.0000000000
Alpha=0.0000000000
Beta=0.0000000000
Ground_Elevation=313.52
```

```
Roll=0.0000000000
Pitch=0.0000000000
Yaw=180.0000000000
P=0.0000000000
Q=0.0000000000
R=0.0000000000
Latitude=36.1628560000
Longitude=-96.8356510000
Altitude=285.3
TAS=0.0000000000
Alpha=0.0000000000
Beta=0.0000000000
Ground_Elevation=313.52
```

Figure 176 Simulator State Files

The new aircraft folder contains two state files, “UAFSnorthtakeoff” and “UAFSsouthtakeoff”. The latitude, longitude, and altitudes are set so as to place the aircraft at the north and south ends of the UAFS airstrip. The UAFS airstrip is at 285 meters altitude; however, the ground elevation in the state file is defined as 313.52 meters. The simulator adds the “MSL (geoid)” value to the “Ground” elevation that the user specifies in the simulator. At the UAFS airstrip the simulator value for MSL is -28.52; therefore, the ground elevation has to be set at 313.52 for the simulator to set the correct ground elevation. The altitude setting is unique to each aircraft. The altitude should be high enough that the main landing gear are just above the ground or just on the ground. In the example shown the aircraft’s left and right wheels were 0.26 meters below the center of gravity of the aircraft model; therefore, the state file was set to place the aircraft to where the wheels were just above the ground. If the aircraft is in the ground when the simulation is started the simulator will glitch and the aircraft will not start in its proper orientation. The state files can be altered to reflect the current aircraft model. Also the starting altitude can be increased if it is desired to simply start the simulation in flight rather than simulating takeoffs.

4.7. Software in the Loop Simulation Setup

The following files are needed for a software in the loop simulation:

1) Simulator File

2) Propeller File

3) Motor File (if gas motor)

4) Alpha File

5) Actuator File (Optional)

6) State File (Optional).

To launch a software in the loop simulation:

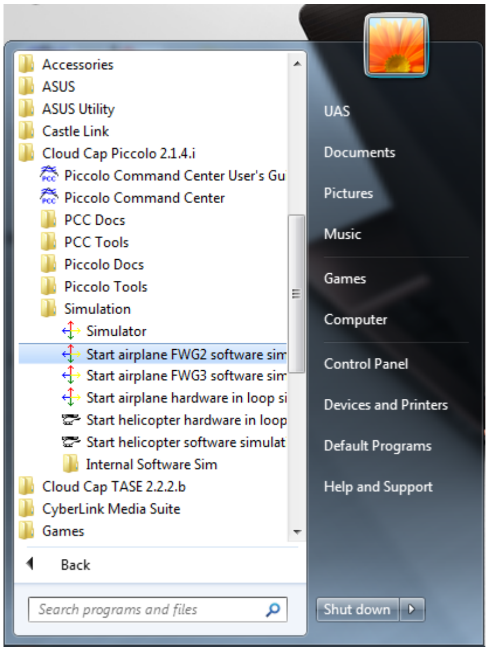


Figure 177 Software in the Loop Simulation

1) Go to the start menu > Cloud Cap Piccolo 2.x.x.x > Simulation > Start airplane FWG2 software simulation. PCC and the simulator will automatically open.

2) Click File > Open Aircraft in the Simulator and load the aircraft's simulator file.

3) Click File > Load State and load the state file.

4) Alter any of the parameters in the Simulator as desired.

5) Create wind profiles, thermals, turbulence, or even just constant wind from one direction as desired. Recall that these disturbances can be made in the simulator and the piccolo software comes with some example atmosphere files.

6) Click "Launch" in the simulator if it is desired to simulate a catapult launch. Recall that using the simulator's launch function will cause the simulator to apply all of the loads that are set in the "Rail Launcher" section of the simulator user interface.

7) Click "Start" in the simulator to simply start the simulation. This method of beginning simulations is typically used for simulating wheeled takeoffs or starting the simulation with the aircraft already at altitude. After start has been clicked the autopilot can be commanded to "Launch" through PCC which will initiate an auto takeoff. The altitude can be changed in the simulator so that the aircraft begins the simulation already in flight.

4.8. Hardware in the Loop Simulation Setup

The following files are needed for a software in the loop simulation:

1) Simulator File
2) Propeller File
3) Motor File (if gas motor)
4) Alpha File

5) Actuator File (Optional)

6) State File (Optional)

To setup a hardware in the loop simulation do the following:

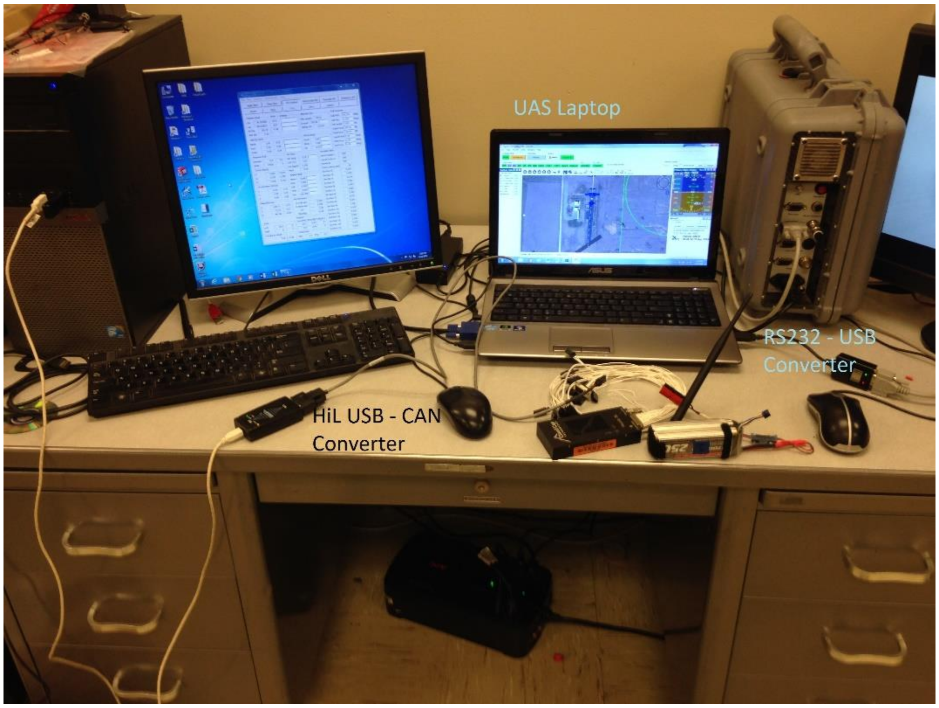
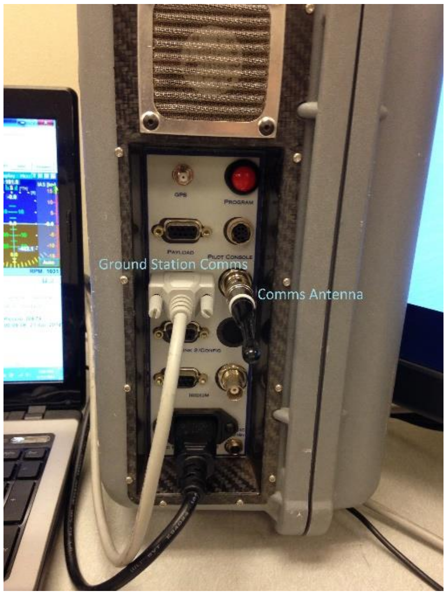


Figure 178 Hardware in the Loop Simulation Setup

1) Connect the HiL USB-CAN Converter to either the UAS Laptop or a desktop computer that has the piccolo software installed before powering on the computer.

2) Connect the RS232-USB Converter to the UAS Laptop.

3) Connect the Ground Station Comms Cord to the RS232-USB Converter.

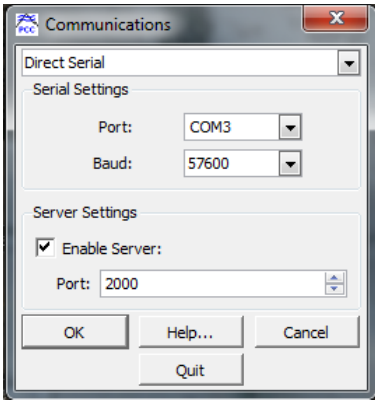


4) Connect the Ground Station Comms Cord to “Link 1” on the ground station.

5) Connect one of the spare Comms Antennas to “Link 1” on the ground station.

6) Open PCC on the UAS Desktop.

7) In the Communications Window:


Communications

Direct Serial

Serial Settings

Port: COM3

Baud: 57600

Server Settings

☑ Enable Server:

Port: 2000

Figure 179 PCC Communications Window

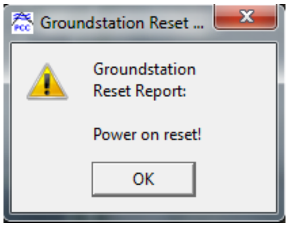
a) Select “Direct Serial” from the dropdown menu at the top.

b) Select the com port that is assigned to the RS232-USB Converter.

c) Set the baud rate to 57600.

d) Check “Enable Server” to allow the use of the Development Interface (Optional).

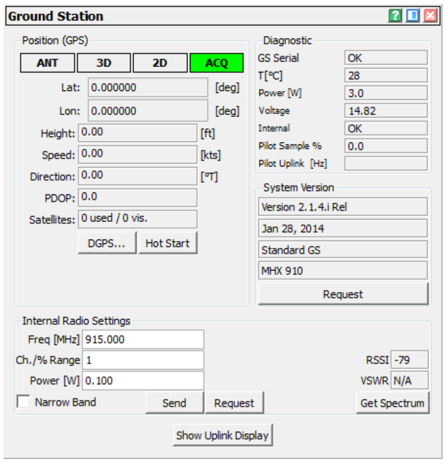
8) Power on the ground station. A message box, notifying the user that the ground station is powered on, should appear.


Groundstation Reset ...

Groundstation Reset Report:

Power on reset!

9) Open the Ground Station Window and set the radio settings appropriately. The power needs to be set to 0.1 watts so that the close proximity of the autopilot and ground station antennas will not damage the communications hardware.


Ground Station

Position (GPS)

Lat: 0.000000 [deg]
Lon: 0.000000 [deg]
Height: 0.00 [ft]
Speed: 0.00 [kts]
Direction: 0.00 [°T]
PDOP: 0.0
Satellites: 0 used / 0 vis.

Diagnostic

GS Serial: OK
T[°C]: 28
Power [W]: 3.0
Voltage: 14.82
Internal: OK
Pilot Sample %: 0.0
Pilot Uplink [Hz]:

System Version

Version 2.1.4.i Rel
Jan 28, 2014
Standard GS
MHX 910

Internal Radio Settings

Freq [MHz] 915.000
Ch./% Range 1
Power [W] 0.100
□ Narrow Band
RSSI -79
VSWR N/A

Figure 180 Ground Station Window

10) On the computer that the USB-CAN adapter is attached to open the simulator. The simulator should be located in the start menu under Cloud Cap Piccolo 2.x.x.x > Simulation.

11) Click File > Open Aircraft in the Simulator and load the aircraft's simulator file.

12) Click File > Load State and load the state file.

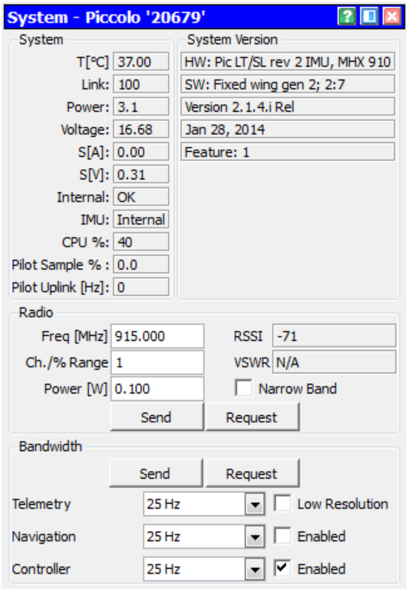
13) Alter any of the parameters in the Simulator as desired.

14) Create wind profiles, thermals, turbulence, or even just constant wind from one direction as desired. Recall that these disturbances can be made in the simulator and the piccolo software comes with some example atmosphere files.

15) Plug the CAN side of the USB-CAN converter into the CAN connector on the autopilot's wiring harness.

16) Power on the autopilot.

17) In the System Window in PCC:



System - Piccolo '20679'

System

T[°C] 37.00
Link: 100
Power: 3.1
Voltage: 16.68
S[A]: 0.00
S[V]: 0.31
Internal: OK
IMU: Internal
CPU %: 40
Pilot Sample % : 0.0
Pilot Uplink [Hz]: 0

System Version

HW: Pic LT/SL rev 2 IMU, MHX 910
SW: Fixed wing gen 2; 2:7
Version 2.1.4.i Rel
Jan 28, 2014
Feature: 1

Radio

Freq [MHz] 915.000
Ch./% Range 1
Power [W] 0.100
RSSI -71
VSWR N/A
☐ Narrow Band

Bandwidth

Telemetry 25 Hz ☐ Low Resolution
Navigation 25 Hz ☐ Enabled
Controller 25 Hz ☑ Enabled

Figure 181 System Window

a) Set the Radio settings to match the settings of the ground station. If the autopilot is unable to communicate with the ground station the power settings can be increased in the piccolo and/or the ground station radio settings.

b) Set the telemetry rates as desired.

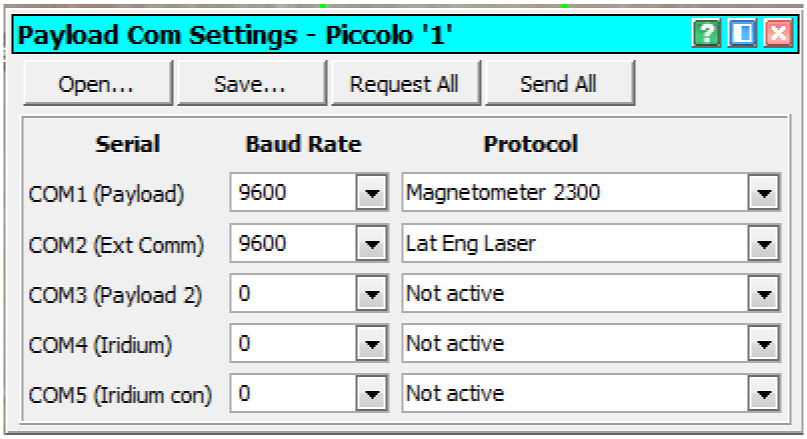
c) Check “Enabled” next to “Controller” to enable controller telemetry (Optional). Enabling controller telemetry allows the Piccolo Development Interface to be used.

18) Check to make sure that all of the aircraft specific parameters and gains are correct.

19) Click “Launch” in the simulator if it is desired to simulate a catapult launch. Recall that using the simulator’s launch function will cause the simulator to apply all of the loads that are set in the “Rail Launcher” section of the simulator user interface.

20) Click “Start” in the simulator to simply start the simulation. This method of beginning simulations is typically used for simulating wheeled takeoffs or starting the simulation with the aircraft already at altitude. After start has been clicked the autopilot can be commanded to “Launch” through PCC which will initiate an auto takeoff. The altitude can be changed in the simulator so that the aircraft begins the simulation already in flight.

4.9. Simulator Extras



Serial	Baud Rate	Protocol
COM1 (Payload)	9600	Magnetometer 2300
COM2 (Ext Comm)	9600	Lat Eng Laser
COM3 (Payload 2)	0	Not active
COM4 (Iridium)	0	Not active
COM5 (Iridium con)	0	Not active

Figure 182 Payload Com Settings

The simulator can be configured to simulate some of the extra hardware add-ons. It can be useful to include a magnetometer in simulations to accurately simulate auto takeoffs. In software in the loop simulations the laser altimeter and magnetometer can be added to the simulated aircraft through the Payload Com Settings window in PCC. If it is desired to simulate using the laser altimeter in a hardware in the loop simulation the laser altimeter will have to be unlocked on the current autopilot unit in order for it to work. The payload com settings cannot be set to "Lat Eng Laser" on the actual autopilot unit unless that feature has been unlocked.

CHAPTER V

PICCOLO MECHANICS

5. Introduction

The Piccolo control structure is setup with command loops, control loops, feedback gains, feedforward gains, vehicle gains, and limits. The entire process is somewhat complex and the Cloud Cap documentation does not present the entire control scheme. This Chapter explains the control structure to a detailed degree. The Chapter exposes limits and control scheme connections that Cloud Cap does not disclose.

This Chapter mainly describes Piccolo control logic in as much detail as possible. The first two sections describe how the piccolo calculates indicated and true airspeeds. Section 5.7 Control Loops combines the control loop definitions from the PccUsersGuide with extra definitions found in the old Cloud Cap document “Tuning piccolo control laws 2.0.x”. Some of the definitions in Tuning piccolo control laws 2.0.x are old and no longer current; therefore, those definitions were filtered to only include the ones that are still applicable to version 2.1.4.x.

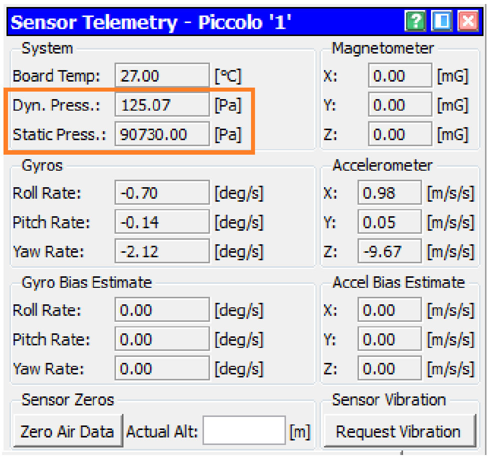
Section 5.8 Control Laws presents equations that were proven to produce calculated actuator deflections. The equations were derived from experimental simulations which are detailed in Appendix D. Each control law directs a command loop command through corresponding control loops, subject to corresponding limits, to produce actuator deflection commands. One of the most impactful discoveries was that of Longitudinal Modes, or Lon Modes. The Lon Modes are described in full with exlpanations for what they are, how the autopilot decides to transition between them, and what effect they have on the control laws.

The last section, Section 5.10, contains complete control schematics of the control laws described in Section 5.8. The schematics include applicable limits, and switches for different Lon Modes.

5.1. Airspeed

The Piccolo is configured to measure static and dynamic pressure from a pitot tube to determine airspeeds. Directly underneath the wiring harness there are two ports to attach tubes to labeled "Static" and "Dynamic" shown in Figure 183 below.

The Piccolo uses these two ports to measure static pressure (Ps) and dynamic pressure (q). The measured values can be viewed during flight in PCC in the "Sensor Telemetry" window.



System
Board Temp: 27.00 [°C]
Dyn. Press.: 125.07 [Pa]
Static Press.: 90730.00 [Pa]

Magnetometer
X: 0.00 [mG]
Y: 0.00 [mG]
Z: 0.00 [mG]

Gyros
Roll Rate: -0.70 [deg/s]
Pitch Rate: -0.14 [deg/s]
Yaw Rate: -2.12 [deg/s]

Accelerometer
X: 0.98 [m/s/s]
Y: 0.05 [m/s/s]
Z: -9.67 [m/s/s]

Gyro Bias Estimate
Roll Rate: 0.00 [deg/s]
Pitch Rate: 0.00 [deg/s]
Yaw Rate: 0.00 [deg/s]

Accel Bias Estimate
X: 0.00 [m/s/s]
Y: 0.00 [m/s/s]
Z: 0.00 [m/s/s]

Sensor Zeros
Actual Alt: [m]

Sensor Vibration

Figure 183 Static & Dynamic Pressures

The autopilot commands indicated airspeed (IAS) and the Control Logic converts the IAS to true airspeed (TAS) before sending the command to the appropriate Control Loops.

5.1.1. Indicated Airspeed

There is no documentation that explicitly states how the autopilot calculates indicated airspeed. A few tests were devised wherein different airspeed equations were used in different scenarios to try to duplicate the indicated airspeed reported by the autopilot. It was determined that the autopilot calculates indicated airspeed using the measured dynamic pressure and the following equation:

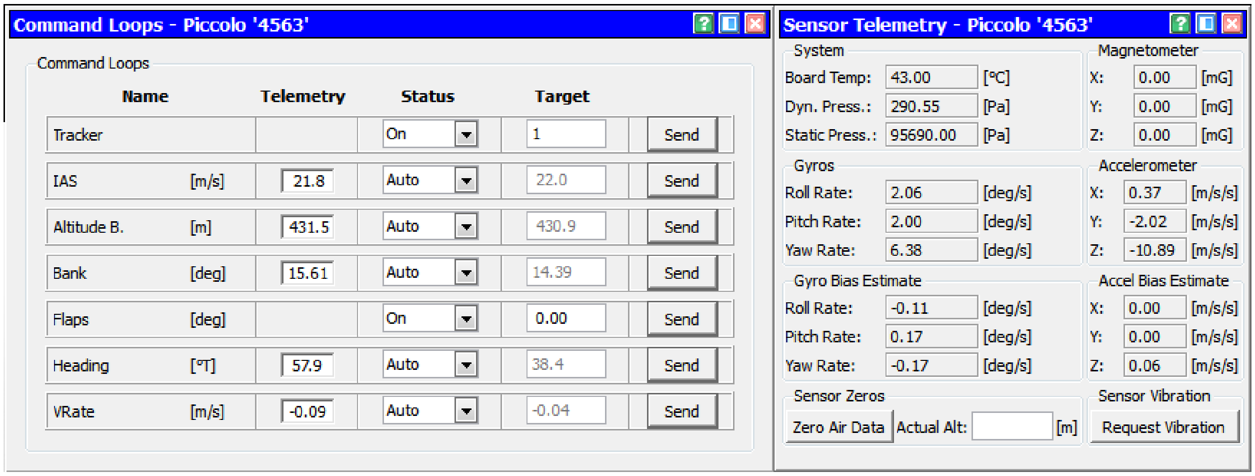
Equation 22 Indicated Airspeed

$$q = \frac{1}{2}\rho V^2 \Rightarrow IAS = \sqrt{\frac{2 * q}{\rho_o}}$$

$$where\ p_o = air\ density\ at\ sea\ level = 1.225\ {}^{kg}\!/_{m^3}$$

$$q = dynamic\ pressure\ (Pa)$$

The following two examples are proofs that Equation 22 will calculate the same indicated airspeed as the piccolo.



Command Loops - Piccolo '4563'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	1	Send
IAS	[m/s]	21.8	Auto	22.0	Send
Altitude B.	[m]	431.5	Auto	430.9	Send
Bank	[deg]	15.61	Auto	14.39	Send
Flaps	[deg]		On	0.00	Send
Heading	[°T]	57.9	Auto	38.4	Send
VRate	[m/s]	-0.09	Auto	-0.04	Send

Sensor Telemetry - Piccolo '4563'

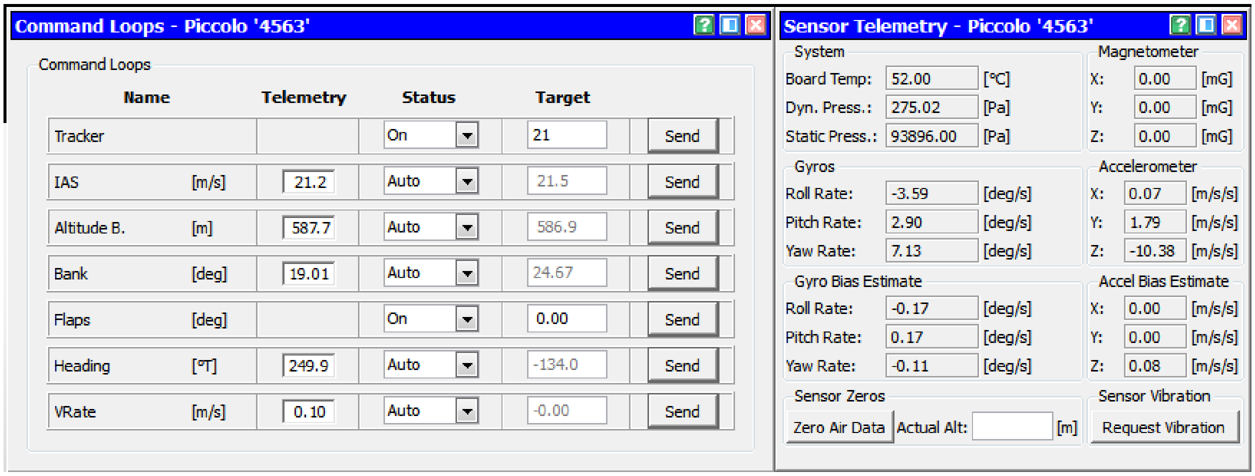
System			Magnetometer		
Board Temp:	43.00	[°C]	X:	0.00	[mG]
Dyn. Press.:	290.55	[Pa]	Y:	0.00	[mG]
Static Press.:	95690.00	[Pa]	Z:	0.00	[mG]
Gyros			**Accelerometer**		
Roll Rate:	2.06	[deg/s]	X:	0.37	[m/s/s]
Pitch Rate:	2.00	[deg/s]	Y:	-2.02	[m/s/s]
Yaw Rate:	6.38	[deg/s]	Z:	-10.89	[m/s/s]
Gyro Bias Estimate			**Accel Bias Estimate**		
Roll Rate:	-0.11	[deg/s]	X:	0.00	[m/s/s]
Pitch Rate:	0.17	[deg/s]	Y:	0.00	[m/s/s]
Yaw Rate:	-0.17	[deg/s]	Z:	0.06	[m/s/s]

Figure 184 Example IAS Calculation

Figure 184 contains screen shots of an instance in a hardware in the loop simulation. From the Sensor Telemetry window the dynamic pressure is 290.55 Pa, and from the Command Loops window the IAS is reported as 21.8 m/s.

$$IAS = \sqrt{\frac{2 * 290.55}{1.225}} = 21.8\ m/s$$

Plugging in the measured value of dynamic pressure in Equation 22 we can see that the autopilot is indeed calculating IAS using sea level air density. Figure 185 is another example at a higher altitude, 587 meters.



Command Loops - Piccolo '4563'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	21	Send
IAS	[m/s]	21.2	Auto	21.5	Send
Altitude B.	[m]	587.7	Auto	586.9	Send
Bank	[deg]	19.01	Auto	24.67	Send
Flaps	[deg]		On	0.00	Send
Heading	[°T]	249.9	Auto	-134.0	Send
VRate	[m/s]	0.10	Auto	-0.00	Send

Sensor Telemetry - Piccolo '4563'

System			Magnetometer		
Board Temp:	52.00	[°C]	X:	0.00	[mG]
Dyn. Press.:	275.02	[Pa]	Y:	0.00	[mG]
Static Press.:	93896.00	[Pa]	Z:	0.00	[mG]
Gyros			**Accelerometer**		
Roll Rate:	-3.59	[deg/s]	X:	0.07	[m/s/s]
Pitch Rate:	2.90	[deg/s]	Y:	1.79	[m/s/s]
Yaw Rate:	7.13	[deg/s]	Z:	-10.38	[m/s/s]
Gyro Bias Estimate			**Accel Bias Estimate**		
Roll Rate:	-0.17	[deg/s]	X:	0.00	[m/s/s]
Pitch Rate:	0.17	[deg/s]	Y:	0.00	[m/s/s]
Yaw Rate:	-0.11	[deg/s]	Z:	0.08	[m/s/s]

Sensor Zeros: Zero Air Data Actual Alt: [m]

Sensor Vibration: Request Vibration

Figure 185 Example IAS Calculation 2

Similar to the first example indicated airspeed is reported as 21.2 m/s with a dynamic pressure of 275.02Pa. Using Equation 22 again IAS is calculated to equal 21.2, the same IAS the autopilot is reporting.

$$IAS = \sqrt{\frac{2 * 275.02}{1.225}} = 21.2\ m/s$$

5.1.2. True Airspeed

There is no documentation that explicitly states how the autopilot calculates true airspeed. Similarly to indicated airspeed a few tests were devised wherein different airspeed equations were used in different scenarios to try to duplicate the true airspeed reported by the autopilot. It was found that the autopilot uses Equation 23 below to calculate true airspeed.

Equation 23 True Airspeed

$$TAS = IAS * \sqrt{\frac{\rho_o}{\rho}}$$

$$Where\ \rho = \ estimated\ air\ density\ \left(\frac{kg}{m^3}\right)\quad \rho_o = air\ density\ at\ sea\ level = 1.225\ (\frac{kg}{m^3})$$

Additionally it was found that the autopilot estimates air density using the ideal gas law, Equation 24. It is extremely important to note that if there is no external OAT (outside air temperature) sensor being used the autopilot will estimate the outside air temperature by taking the board temperature of the piccolo unit and subtracting the value set for “OAT Bias”. OAT Bias is found in the Sensor Configuration window of PCC.

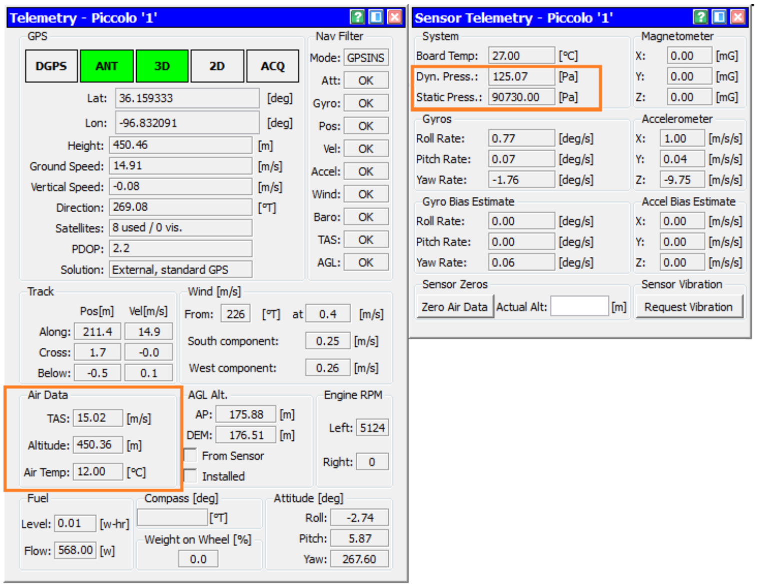
Equation 24 Ideal Gas Law

$$P = \rho RT$$

$$where\ P = Measured\ Static\ Pressure\ (Pa)\ \ R = 287.058\frac{J}{kg * K}$$

$$T = measured\ outside\ air\ temeperature\ (K)$$

The following examples are proofs that Equation 23 and Equation 24 are used to calculate true airspeed.



Lat: 36.159333 [deg]
Lon: -96.832091 [deg]
Height: 450.46 [m]
Ground Speed: 14.91 [m/s]
Vertical Speed: -0.08 [m/s]
Direction: 269.08 [°T]
Satellites: 8 used / 0 vis.
PDOP: 2.2
Solution: External, standard GPS

Mode: GPSINS
Att: OK
Gyro: OK
Pos: OK
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: OK

	Pos[m]	Vel[m/s]
Along:	211.4	14.9
Cross:	1.7	-0.0
Below:	-0.5	0.1

From: 226 [°T] at 0.4 [m/s]
South component: 0.25 [m/s]
West component: 0.26 [m/s]

TAS: 15.02 [m/s]
Altitude: 450.36 [m]
Air Temp: 12.00 [°C]

AP: 175.88 [m]
DEM: 176.51 [m]
□ From Sensor
□ Installed

Left: 5124
Right: 0

Level: 0.01 [w-hr]
Flow: 568.00 [w]

[°T]
Weight on Wheel [%]
0.0

Roll: -2.74
Pitch: 5.87
Yaw: 267.60

Board Temp: 27.00 [°C]
Dyn. Press.: 125.07 [Pa]
Static Press.: 90730.00 [Pa]

X: 0.00 [mG]
Y: 0.00 [mG]
Z: 0.00 [mG]

Roll Rate: 0.77 [deg/s]
Pitch Rate: 0.07 [deg/s]
Yaw Rate: -1.76 [deg/s]

X: 1.00 [m/s/s]
Y: 0.04 [m/s/s]
Z: -9.75 [m/s/s]

Roll Rate: 0.00 [deg/s]
Pitch Rate: 0.00 [deg/s]
Yaw Rate: 0.06 [deg/s]

X: 0.00 [m/s/s]
Y: 0.00 [m/s/s]
Z: 0.00 [m/s/s]

Actual Alt: [m]

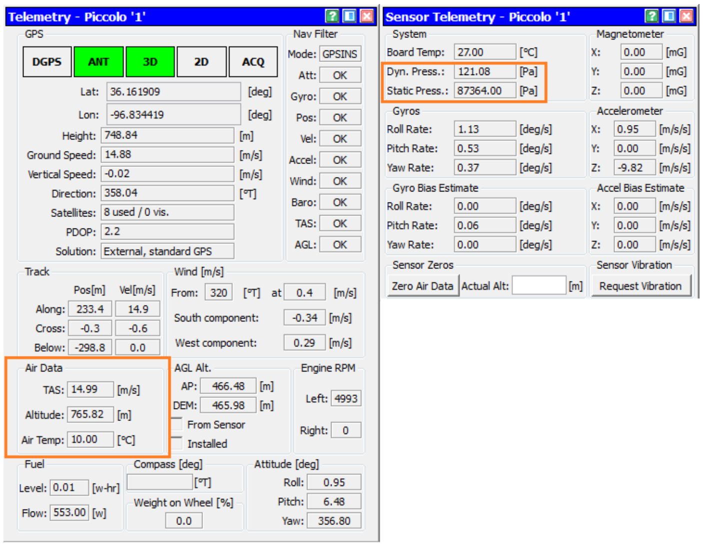
Figure 186 Example TAS Calculation

The first example presents a scenario where an aircraft was at 450 m altitude in a hardware in the loop simulation. As shown in Figure 186 the outside air temperature was 12°C, the static pressure was 90730Pa, and the dynamic pressure was 125.07Pa.

$$\rho = \frac{90730}{287.058 * (12 + 273.15)} = 1.108429 \frac{kg}{m^3}$$

$$\Rightarrow TAS = \sqrt{\frac{2 * 125.07}{1.108429}} = 15.02\, m/s$$

The calculations above calculated a true airspeed of 15.02 m/s which matched the true airspeed in Figure 186.



GPS

Lat: 36.161909 [deg]

Lon: -96.834419 [deg]

Height: 748.84 [m]

Ground Speed: 14.88 [m/s]

Vertical Speed: -0.02 [m/s]

Direction: 358.04 [°T]

Satellites: 8 used / 0 vis.

PDOP: 2.2

Solution: External, standard GPS

Nav Filter

Mode: GPSINS

Att: OK

Gyro: OK

Pos: OK

Vel: OK

Accel: OK

Wind: OK

Baro: OK

TAS: OK

AGL: OK

Track

	Pos[m]	Vel[m/s]
Along:	233.4	14.9
Cross:	-0.3	-0.6
Below:	-298.8	0.0

Wind [m/s]

From: 320 [°T] at 0.4 [m/s]

South component: -0.34 [m/s]

West component: 0.29 [m/s]

Air Data

TAS: 14.99 [m/s]

Altitude: 765.82 [m]

Air Temp: 10.00 [°C]

AGL Alt.

AP: 466.48 [m]

DEM: 465.98 [m]

□ From Sensor

□ Installed

Engine RPM

Left: 4993

Right: 0

Fuel

Level: 0.01 [w-hr]

Flow: 553.00 [w]

Compass [deg]

[°T]

Weight on Wheel [%]

0.0

Attitude [deg]

Roll: 0.95

Pitch: 6.48

Yaw: 356.80

System

Board Temp: 27.00 [°C]

Dyn. Press.: 121.08 [Pa]

Static Press.: 87364.00 [Pa]

Magnetometer

X: 0.00 [mG]

Y: 0.00 [mG]

Z: 0.00 [mG]

Gyros

Roll Rate: 1.13 [deg/s]

Pitch Rate: 0.53 [deg/s]

Yaw Rate: 0.37 [deg/s]

Accelerometer

X: 0.95 [m/s/s]

Y: 0.00 [m/s/s]

Z: -9.82 [m/s/s]

Gyro Bias Estimate

Roll Rate: 0.00 [deg/s]

Pitch Rate: 0.06 [deg/s]

Yaw Rate: 0.00 [deg/s]

Accel Bias Estimate

X: 0.00 [m/s/s]

Y: 0.00 [m/s/s]

Z: 0.00 [m/s/s]

Sensor Zeros

Actual Alt: [m]

Sensor Vibration

Figure 187 Example TAS Calculation 2

In the second example, depicted in Figure 187, an aircraft was at 750 m altitude in a hardware in the loop simulation. The outside air temperature was 10°C, the static pressure was 87364Pa, and the dynamic pressure was 121.08Pa.

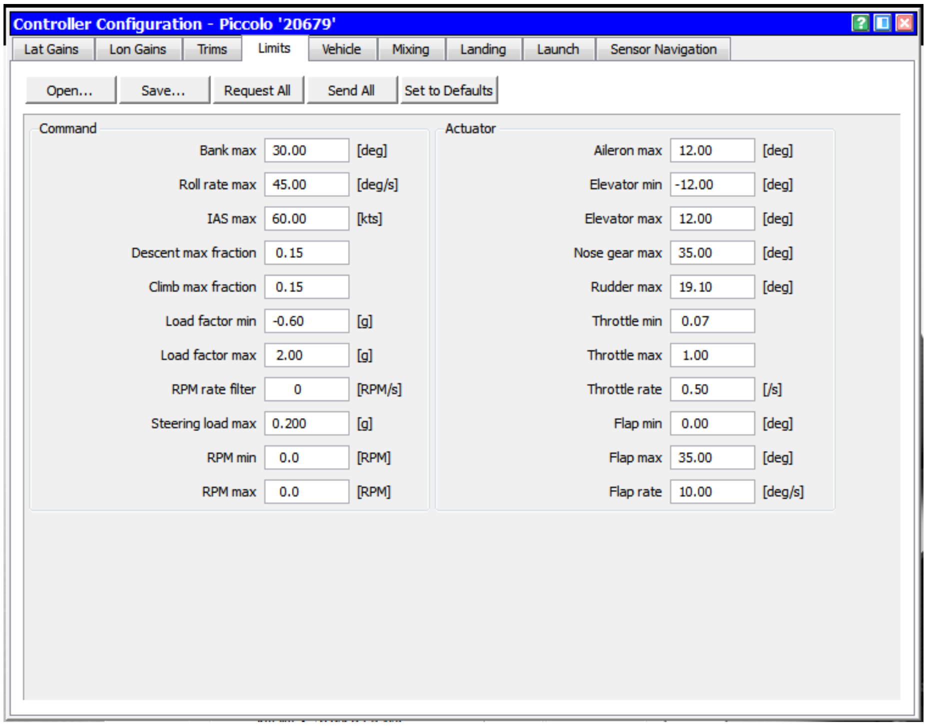
$$\rho = \frac{87364}{287.058 * (10 + 273.15)} = 1.074846 \frac{kg}{m^3}$$

$$\Rightarrow TAS = \sqrt{\frac{2 * 121.08}{1.074846}} = 15.00\, m/s$$

The calculations above calculated a true airspeed of 15.00 m/s which essentially matched the true airspeed in Figure 187 of 14.99 m/s.

5.2. Limits

The Piccolo has 22 Limits that can be set in the “Limits” tab of the Controller Configuration window in PCC depicted below.



Command

Bank max	30.00	[deg]
Roll rate max	45.00	[deg/s]
IAS max	60.00	[kts]
Descent max fraction	0.15	
Climb max fraction	0.15	
Load factor min	-0.60	[g]
Load factor max	2.00	[g]
RPM rate filter	0	[RPM/s]
Steering load max	0.200	[g]
RPM min	0.0	[RPM]
RPM max	0.0	[RPM]

Actuator

Aileron max	12.00	[deg]
Elevator min	-12.00	[deg]
Elevator max	12.00	[deg]
Nose gear max	35.00	[deg]
Rudder max	19.10	[deg]
Throttle min	0.07	
Throttle max	1.00	
Throttle rate	0.50	[/s]
Flap min	0.00	[deg]
Flap max	35.00	[deg]
Flap rate	10.00	[deg/s]

Figure 188 Piccolo Limits

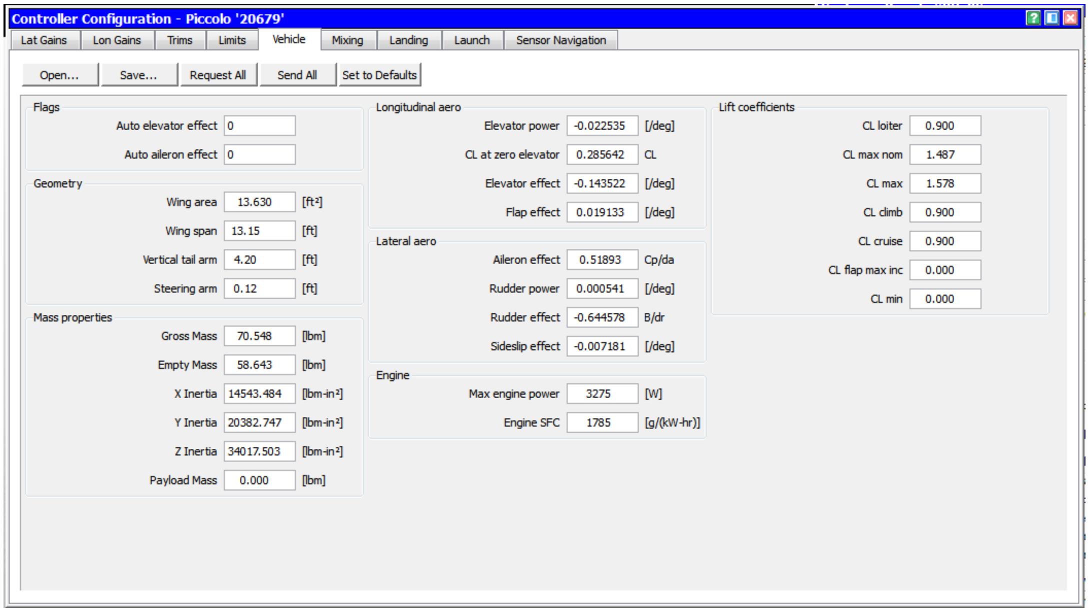
The limits can and do influence control loops. Each limit is defined in Cloud Cap's document "PccUsersGuide" on pgs. 114-115. It is important to note a few important things about some of the limits that are left out of the limit definitions. The load factor max and min limits are designed to be used as structural load limits that represent how much vertical acceleration the aircraft can handle. The limits are multiplied by one g, -9.81 m/s^2, and applied to Z – Acceleration Control as acceleration limiters.

The RPM min and max limits do more than just limit the rpms. If a value greater than 0 is input into either RPM Control will be enabled. RPM Control does have an impact on Energy Control and throttle. RPM Control needs to be fully understood if it is desired to be used.

The Climb and Descent max fractions are used to calculate the minimum and maximum vertical rate commands. The value of the limits is combined with the airspeed of the aircraft to calculate the vertical rate command limits. The vertical rate command limits are applied to the VRate Control Loop and limit the amount of vertical rate that the autopilot can command.

5.3. Vehicle Gains

Vehicle gains are aircraft parameters that reflect facts and aerodynamic capabilities of the vehicle that the autopilot is installed in. All of the vehicle gains are in some way linked to a control loop or multiple control loops; thus, it is very important that they are reasonably accurate. They don't have to be exact, but they need to be at least in the ball park.



Flags
Auto elevator effect 0
Auto aileron effect 0

Geometry
Wing area 13.630 [ft^2]
Wing span 13.15 [ft]
Vertical tail arm 4.20 [ft]
Steering arm 0.12 [ft]

Mass properties
Gross Mass 70.548 [lbm]
Empty Mass 58.643 [lbm]
X Inertia 14543.484 [lbm-in^2]
Y Inertia 20382.747 [lbm-in^2]
Z Inertia 34017.503 [lbm-in^2]
Payload Mass 0.000 [lbm]

Longitudinal aero
Elevator power -0.022535 [/deg]
CL at zero elevator 0.285642 CL
Elevator effect -0.143522 [/deg]
Flap effect 0.019133 [/deg]

Lateral aero
Aileron effect 0.51893 Cp/da
Rudder power 0.000541 [/deg]
Rudder effect -0.644578 B/dr
Sideslip effect -0.007181 [/deg]

Engine
Max engine power 3275 [W]
Engine SFC 1785 [g/(kW-hr)]

Lift coefficients
CL loiter 0.900
CL max nom 1.487
CL max 1.578
CL climb 0.900
CL cruise 0.900
CL flap max inc 0.000
CL min 0.000

Figure 189 Vehicle Gains Window

The vehicle gains are input and displayed in the "Vehicle" tab of the Controller Configuration window in PCC. The sections Longitudinal aero, Lateral aero, Lift coefficients, and Max Engine Power are all initially estimated through the Simulator which estimates these values based on the results of the AVLEditor alpha sweeps. The Simulator will generate a file with all of the Vehicle Parameters included; however, Geometry and Mass properties are not estimated by the Simulator. Those values come straight from the actual Simulator file where they are input by the user.

Cloud Cap provides documentation that defines the vehicle gains in PccUsersGuide; however, there is an additional legacy document that contains definitions that include more detail and are more informative. The legacy document, “Tuning piccolo control laws 2.0.x”, does contain information that is not correct in version 2.1.4.x. As a result the definitions that apply to version 2.1.4.i were combined with the definitions from the PccUsersGuide to provide detailed straightforward definitions of the vehicle gains. Additionally further explanations have been added from the experience gained throughout the user of the piccolo. Note that the vehicle gains that play roles directly in control loops will be discussed further in the corresponding control loop sections.

Wing area: Vehicle reference wing area [m2]. Used to scale calculations involving the aerodynamic coefficients (PccUsersGuide pg. 115). The Wing area is used with the Elevator Effectiveness parameter to back out the change in CL over the change in Elevator deflection. Used in the Z – Acceleration Control Loop.

Wing span: Vehicle reference wing span [m]. Used to scale calculations involving the rolling moment, yawing moment, aileron, or rudder (PccUsersGuide pg. 115). The Wing span is used with the Aileron Effectiveness term to back out the change in Roll Rate per change in Aileron Deflection. Used in the Roll Control Loop.

Vertical Tail Arm: Distance from the center of gravity to the vertical tail aerodynamic center [m]. This value is used to estimate the rudder required to turn coordination. Set to zero to disable turn coordination (PccUsersGuide pg. 115). Used in the Side Force Control Loop with Rudder Effectiveness to calculate Rudder deflections when the Side Force Integral gain is zero.

Nose gear steering arm: Distance from the fixed gear to the steerable gear [m]. This value is used (along with speed) to predict how far to turn the nose gear in order to affect a desired vehicle turn rate (PccUsersGuide pg. 115).

Gross Mass: Mass of the aircraft full of fuel and payload [Kg]. Must be greater than or equal to the empty mass. The aircraft mass is estimated based upon the empty mass, the payload mass, and the fuel mass (which varies). The gross mass is used to limit the amount of fuel or payload that the user can indicate is stored in the aircraft. Electric vehicles, or vehicles for which the fuel burn is unpredictable should set the gross mass equal to the empty mass plush payload mass (PccUsersGuide pg. 115).

Empty Mass: Mass of the aircraft with no fuel or payload [Kg] (PccUsersGuide pg. 115).

X Inertia: Inertia of the vehicle about the X axis [Kg-m2] (PccUsersGuide pg. 115). X Inertia is not actually used in Fixed Wing Generation 2.

Y Inertia: (Inertia of the vehicle about the Y axis [Kg-m2]. This value is used in the scaling of the pitch rate feedback (PccUsersGuide pg. 115). The Y – Inertia is only used if the Pitch Damper is utilized.

Z Inertia: Inertia of the vehicle about the Z axis [Kg-m2] . This value is used in the scaling of the yaw rate feedback (PccUsersGuide pg. 116). The Z-Inertia is used in Yaw Control.

Payload Mass: Mass of the payload [Kg] (PccUsersGuide pg. 116).

Elevator Power: Change in pitch moment coefficient per change in elevator [/rad]. Increasing elevator angles should produce decreasing pitch moments, hence this number is negative (PccUsersGuide pg. 116).

CL at zero Elevator: The lift coefficient of the vehicle when the elevator is at zero. This value is used along with the elevator trim position to estimate where to place the elevator when the control loops turn on (PccUsersGuide pg. 116).

Elevator Effectiveness: Steady state change in lift coefficient per change in elevator position [/rad]. This is the primary elevator control power term. Under steady state assumptions, if the aircraft is statically stable, the angle of attack and hence the lift coefficient are assumed to depend linearly on the elevator according to this term and the "CL at zero elevator". The controller uses this number to predict the correct elevator position based upon the acceleration command, and to scale the elevator feedback gains. Reducing this value causes the controller to move the elevator further. This value should always be negative. If the elevator effectiveness varies over the operating envelope of the aircraft than the largest magnitude value should be given. This is typically the value that occurs at high speeds where trim forces are not significant (PccUsersGuide pg. 116).

The most important step in tuning is to make sure that the elevator effectiveness term is correct. If the elevator effectiveness is too low the system will drive the elevator too far causing fast oscillations and/or speed control divergence. If it is too high it will force the integral feedback term to work harder, reducing acceleration control bandwidth. For some vehicles the elevator effectiveness can vary over the operational speed range. In that case always choose the largest value. The simulator can estimate the elevator effectiveness. To check (or measure) the elevator effectiveness in flight fly the vehicle at two elevator settings, with the throttle fixed (i.e. you will need to climb or descend). Choose elevator positions that do not take the aircraft near stall so that aerodynamic non-linearity can be avoided. For each case compute the lift coefficient as the mass of the aircraft (kg) times the negative of body z acceleration (m/s/s) divided by the dynamics pressure (Pa) and divided by the wing area (m2). The elevator effectiveness is computed by dividing the change in lift coefficient by the change in elevator deflection (Tuning piccolo control laws 2.0.x pg. 8).

Elevator effectiveness is used in the Z – Acceleration Control Loop where it is in both the feedforward and feedback portions of the control loop.

Flap Effectiveness: Increment in lift coefficient per radian of flap deployment (PccUsersGuide pg. 116).

Aileron Effectiveness: Dimensionless roll rate (pb/2V) per change in aileron position [/rad]. This is the primary aileron control power term. Under steady state assumptions, if the roll damping is large and the roll axis inertia is small, the dimensionless roll rate depends only on the aileron angle according to this term. The controller uses this number to predict the correct aileron position, and to scale the aileron feedback gains. Reducing this value causes the controller to move the aileron further. This value should always be greater than zero (PccUsersGuide pg. 116).

The most important step in tuning is to make sure that the aileron effectiveness term is correct. If the aileron effectiveness is too low the system will drive the ailerons too far causing fast oscillations and/or exceeding the bank rate limit. If it is too high it will force the integral feedback term to work harder, reducing bank angle control bandwidth and possibly causing slow oscillations. The simulator can estimate the aileron effectiveness. To check (or measure) the aileron effectiveness in flight a relatively small aileron deflection should be used, such that the vehicle rolls at a moderate rate of, say, 20°/sec. The dimensionless roll rate is computed by taking the actual roll rate and multiplying it by (b/2V) where V is the true air speed. The aileron effectiveness is computed by dividing the change in steady state dimensionless roll rate by the change in aileron deflection (Tuning piccolo control laws 2.0.x pg. 6).

Rudder Power: Yawing moment coefficient per change in rudder position [/rad]. This is the primary rudder control power term. In combination with the Z-axis inertia this term is used to scale the gains of the yaw damper. Reducing this value cause the controller to move the rudder further (PccUsersGuide pg. 116). Rudder power scales the feedback of the yaw rate control loop where it is multiplied to the yaw rate error as if it were a proportional gain.

Rudder Effectiveness: Change in sideslip per change in rudder position [rad/rad]. In combination with the tail moment arm this number is used to estimate the amount of rudder deflection required to coordinate a turn. It is only used if the side force integral feedback gain is zero. Reducing this value causes the controller to move the rudder more (PccUsersGuide pg. 116). Rudder effectiveness is used to scale the feedback of the side force control loop where the side force error is divided by this term amongst other terms to calculate rudder deflections. Rudder effectiveness is also used to calculate rudder deflections in the feed forward portion of the yaw rate control loop.

Sideslip Effectiveness: Change in side force coefficient per change in side slip [/rad]. This term is used to scale the side force integral feedback for feedback turn coordination. Reducing this value causes the controller to move the rudder more (PccUsersGuide pg. 116). Sideslip effectiveness is used to scale the feedback of the side force control loop where the side force error is divided by this term amongst other terms to calculate rudder deflections.

Max Engine Power: Maximum engine power [W]. This number is used to predict the how far to move the throttle in response to the power required computed by the controller. It is assumed that the net power at throttle of 1.0 will be equal to this value. It is also used to scale the throttle feedback gains. Reducing this number causes the controller to move the throttle further (PccUsersGuide pg. 116).

The max engine power is used to scale all of the engine gains and to predict the amount of throttle motion required to achieve a desired energy rate. It is difficult to exactly determine this number since it varies based upon flight and atmospheric condition. You may be required to rely on the engine manufacturer's data. If you have a high fidelity aerodynamic model and can estimate vehicle drag than you may be able to determine max power in flight by going to a max throttle climb condition and measuring the power lost to drag and adding in the power going into the climb. Making this number larger will reduce the throttle motion and vice versa (Tuning piccolo control laws 2.0.x pg. 10).

Engine SFC: Engine specific fuel consumption in grams of fuel per hour per kilowatt of power [g/(kW-hr)]. Set this to zero or less than zero to indicate that the aircraft is electric. If this is positive than the controller will combine it with the throttle position, and the max engine power, to estimate the fuel burn rate. The fuel burn rate will be used to debit the mass of the aircraft to account for fuel burned off. The mass will not be allowed to fall before the "empty mass" value (PccUsersGuide pg. 117).

CL Loiter: The lift coefficient for best endurance. This number is used to determine what speed the vehicle should fly when the airspeed control loop is in AUTO and the vehicle is orbiting (PccUsersGuide pg. 117).

CL max nominal: The maximum lift coefficient that the vehicle can normally operate at with the flaps up. This is used to determine how much aerodynamic acceleration is allowed. Finally this number is used to compute the minimum indicated airspeed, such that the dynamic pressure at the minimum speed is 1.1 times the dynamic pressure at CL max nominal (PccUsersGuide pg. 117).

CL max: The maximum lift coefficient that the vehicle can sustain with the flaps up. This number is used during the landing to determine how much acceleration can be developed before stalling the vehicle. It is also used to distinguish between loads that are due to aerodynamics/turbulence and loads that are due to ground contact. Finally this number is used to compute the minimum indicated airspeed in short final and later landing modes (PccUsersGuide pg. 117).

CL climb: The lift coefficient at which the vehicle climbs best. This number is used to determine what speed the vehicle should fly when the airspeed control loop is in AUTO and the vehicle is climbing (PccUsersGuide pg. 117).

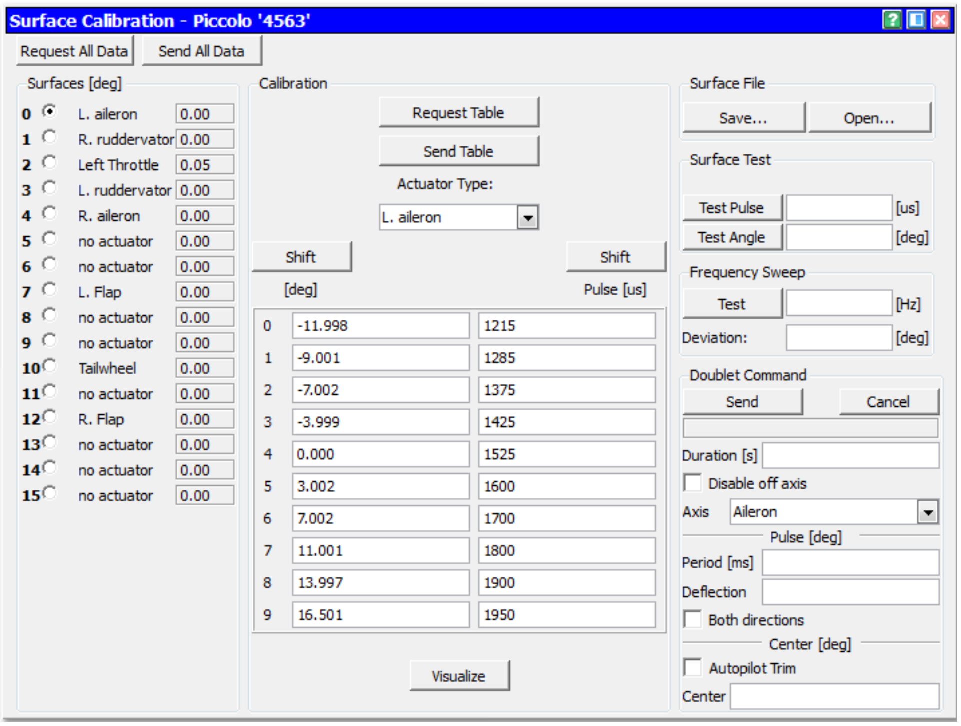
CL cruise: The lift coefficient at which the vehicle cruises best. This number is used to determine what speed the vehicle should fly when the airspeed control loop is in AUTO and the vehicle is cruising (PccUsersGuide pg. 117).

CL flap max inc: Maximum increment in lift coefficient due to flap deflection. When flaps are deployed the maximum lift coefficient will be increased according to “Flap Effect”, up to “CL flap max” (PccUsersGuide pg. 117).

CL min: The minimum lift coefficient the vehicle is allowed to operate at (PccUsersGuide pg. 117).

5.4. Control Surfaces

Control surfaces are determined by their settings in PCC in the “Surface Calibration” window. Each surface is defined as the servo, or actuator number that the surface is wired into. The cloud cap documentation, in the pdf “Piccolo External Interface.pdf", refers to the I/O lines as “servo” lines.



Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	●	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.05
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File

Surface Test

Test Pulse [us]
Test Angle [deg]

Frequency Sweep

Test [Hz]
Deviation: [deg]

Doublet Command

Duration [s]
□ Disable off axis
Axis: Aileron
Pulse [deg]
Period [ms]
Deflection
□ Both directions
Center [deg]
□ Autopilot Trim
Center

Figure 190 Surfaces Calibration Window

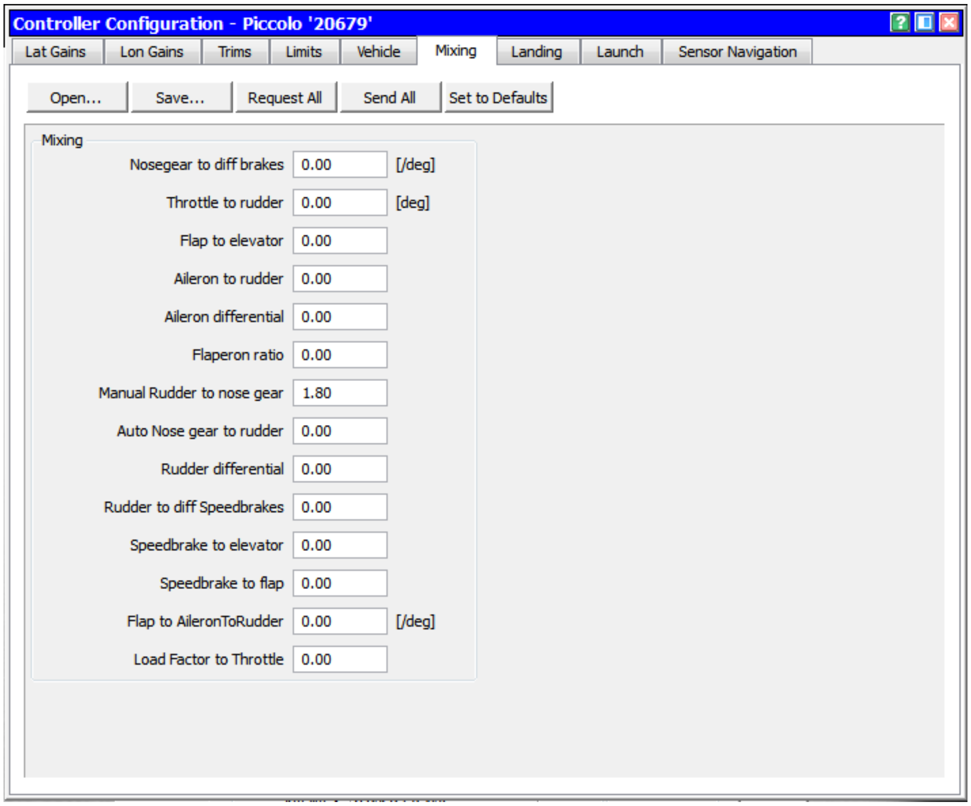
In Figure 190 the left aileron is wired into servo line 0, and the right aileron is wired into servo line 4. The servo lines correspond to specific pin numbers that are dependent on the autopilot that is being used (piccolo SL, piccolo II, etc.). The "Actuator Type" scroll down menu determines how the autopilot interprets the function of the control surface plugged into the selected servo line.

At the most basic the control loops consider control surfaces to be ailerons, elevators, rudders, or flaps. Ailerons are defined as surfaces that induce roll. Elevators are defined as surfaces that induce pitch. Rudders are surfaces that induce yaw. Flaps are defined as surfaces that alter the CL of the aircraft (Gen Two does not use Flaps to reduce energy).

The autopilot has built in control surface mixtures, or actuator types, that will treat a control surface as having more than one function. For example there is an actuator type for a V-Tail configuration, called "Ruddervators". The ruddervators are treated as elevators and rudders.

If the autopilot commands 5 degrees elevator deflection both the left and right ruddervator will deflect down 5 degrees. If the autopilot commands 5 degrees rudder deflection the left ruddervator will deflect up 5 degrees and the right ruddervator will deflect down 5 degrees (positive rudder deflection is defined as trailing edge of the surface starboard). The available actuator types are documented in the “PccUsersGuide.pdf” pgs. 103 – 104.

PCC also offers additional mixing options that are available in the “Mixing” tab of the Controller Configuration window.



Mixing

Nosegear to diff brakes 0.00 [/deg]

Throttle to rudder 0.00 [deg]

Flap to elevator 0.00

Aileron to rudder 0.00

Aileron differential 0.00

Flaperon ratio 0.00

Manual Rudder to nose gear 1.80

Auto Nose gear to rudder 0.00

Rudder differential 0.00

Rudder to diff Speedbrakes 0.00

Speedbrake to elevator 0.00

Speedbrake to flap 0.00

Flap to AileronToRudder 0.00 [/deg]

Load Factor to Throttle 0.00

Figure 191 Mixing Window

The major difference using these mixtures shows up during modeling and estimating vehicle parameters. The simulator will appropriately include the mixtures that are actuator types when it estimates the vehicle parameters. If control surfaces are mixed using the “Mixing” settings then the estimated vehicle parameters will have to be adjusted accordingly because the simulator will not estimate the surfaces as having multiple control functions. For example if an aircraft had a "Flaperon ratio" where the flaps would deflect a given percentage of the amount of the commanded aileron deflection then the aileron effectiveness term estimated by the simulator would be too small. The actuator type of the flap surfaces would be "flaps" and since the piccolo doesn't consider flaps to be surfaces that induce roll neither will the simulator.

The control surfaces are deflected via signals of different pulse widths sent to the servos by the autopilot. The autopilot determines the pulse width signal for a given deflection based on the lookup tables in the Surface Calibration window which are specific for each surface. Figure 190 above shows the pulse width deflection settings for a left aileron plugged into the servo line 0.

5.5. Command Loops

Fixed Wing Generation 2 v.7 has seven command loops. All of the commands that the autopilot receives come from the command loops. Even remote pilot commands, executed in PCC, go through the command loops. Each command is routed through corresponding Control Loops. The Control Laws or Control Logic dictates the flow of commands through Control Loops. The Control Loops contain all of the piccolo gains and translate the commands into actuator outputs.

Piccolo documentation on the command loops is located in one pdf document. "PccUsersGuide.pdf" pgs. 99-100.

The command loops are interfaced with a Universal Controller to feed commands into the piccolo control laws.

“The autopilot aircraft control laws are defined according to an interface called the universal controller. Universal controllers interact with the user and the remainder of the autopilot in the following ways:

By defining command loops and targets for each of these loops. All controllers support a tracker loop for navigation.

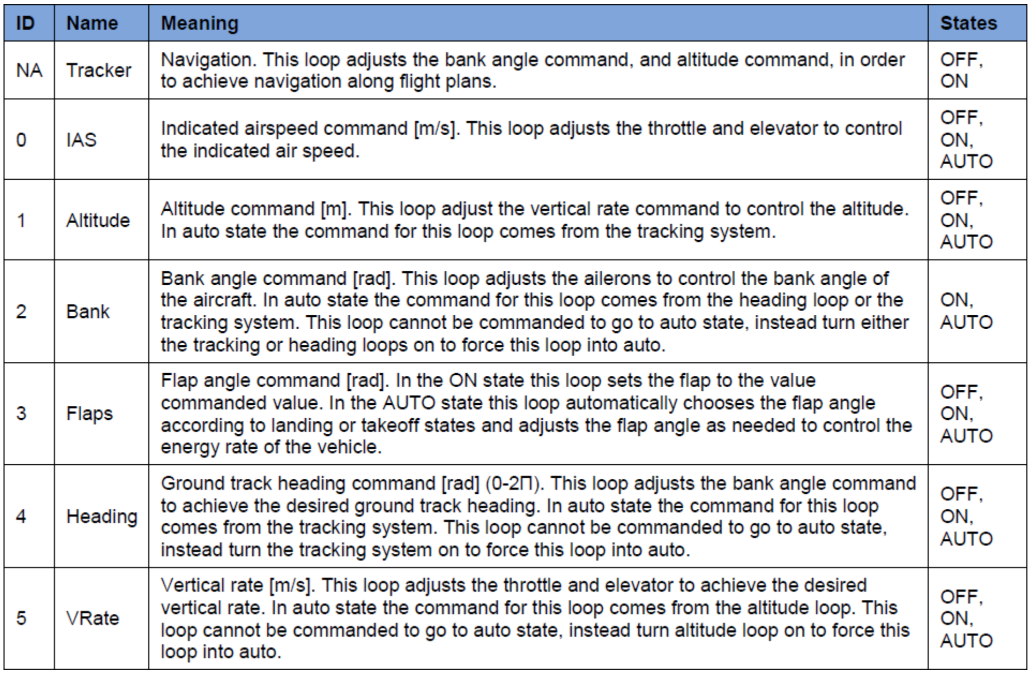
By defining actuator types which relate controller outputs to actuator outputs.

By defining controller states. All controllers support state zero, which is pre-launch (the state the autopilot starts in).

By defining categories of data that govern how the controller functions. These categories of data can be queried or changed and are stored in nonvolatile memory on the autopilot.

A universal controller has a simple enumerated type to identify it. The user interface queries the autopilot for the type and version of the controller and dynamically recons itself as needed to support that controller. The reconfiguration is accomplished by using the type and version to load a configuration file (in XML format) which gives the user interface all the details regarding the controller.” (PccUsersGuide pg. 99)

Figure 192 from PccUsersGuide pg. 100 shows the seven different command loops.



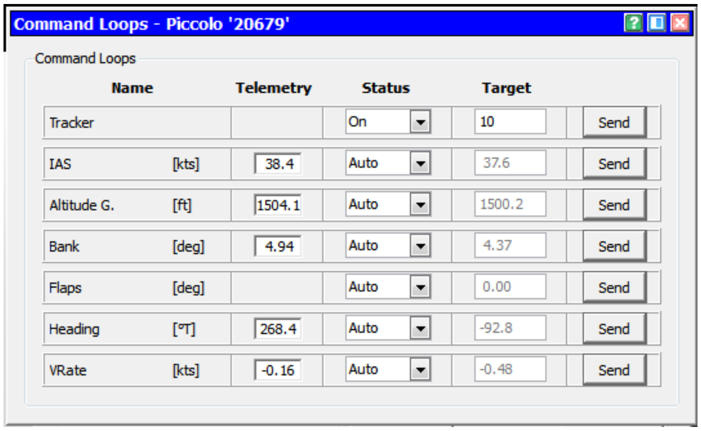
ID	Name	Meaning	States
NA	Tracker	Navigation. This loop adjusts the bank angle command, and altitude command, in order to achieve navigation along flight plans.	OFF, ON
0	IAS	Indicated airspeed command [m/s]. This loop adjusts the throttle and elevator to control the indicated air speed.	OFF, ON, AUTO
1	Altitude	Altitude command [m]. This loop adjust the vertical rate command to control the altitude. In auto state the command for this loop comes from the tracking system.	OFF, ON, AUTO
2	Bank	Bank angle command [rad]. This loop adjusts the ailerons to control the bank angle of the aircraft. In auto state the command for this loop comes from the heading loop or the tracking system. This loop cannot be commanded to go to auto state, instead turn either the tracking or heading loops on to force this loop into auto.	ON, AUTO
3	Flaps	Flap angle command [rad]. In the ON state this loop sets the flap to the value commanded value. In the AUTO state this loop automatically chooses the flap angle according to landing or takeoff states and adjusts the flap angle as needed to control the energy rate of the vehicle.	OFF, ON, AUTO
4	Heading	Ground track heading command [rad] (0-2Π). This loop adjusts the bank angle command to achieve the desired ground track heading. In auto state the command for this loop comes from the tracking system. This loop cannot be commanded to go to auto state, instead turn the tracking system on to force this loop into auto.	OFF, ON, AUTO
5	VRate	Vertical rate [m/s]. This loop adjusts the throttle and elevator to achieve the desired vertical rate. In auto state the command for this loop comes from the altitude loop. This loop cannot be commanded to go to auto state, instead turn altitude loop on to force this loop into auto.	OFF, ON, AUTO

Figure 192 Piccolo Command Loops

Notice that all of the command loops can be changed to “ON”, where the user can command one constant value and nearly all of them have an “OFF” option. If control gains need to be tuned disabling AUTO and even turning OFF some of the command loops can help determine whether the outer loop control gains or the inner loop control gains are in need of tuning.

Altitude, Bank, IAS, and VRate command loops are subject to limits (Alt Max Min, Bank Max, Climb and Descent Max Fractions) that are specified throughout aircraft settings in PCC.

The “Flaps” command loop is only used in the Generation Three controller. Flaps will only be deployed, in Generation Two, when the user commands it, or on Land Plans that have the land settings configured for Flap deflections. Flaps will not be used AUTO to shed energy as suggested by Figure 193.



Command Loops - Piccolo '20679'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	10	Send
IAS	[kts]	38.4	Auto	37.6	Send
Altitude G.	[ft]	1504.1	Auto	1500.2	Send
Bank	[deg]	4.94	Auto	4.37	Send
Flaps	[deg]		Auto	0.00	Send
Heading	[°T]	268.4	Auto	-92.8	Send
VRate	[kts]	-0.16	Auto	-0.48	Send

Figure 193 Command Loops Window

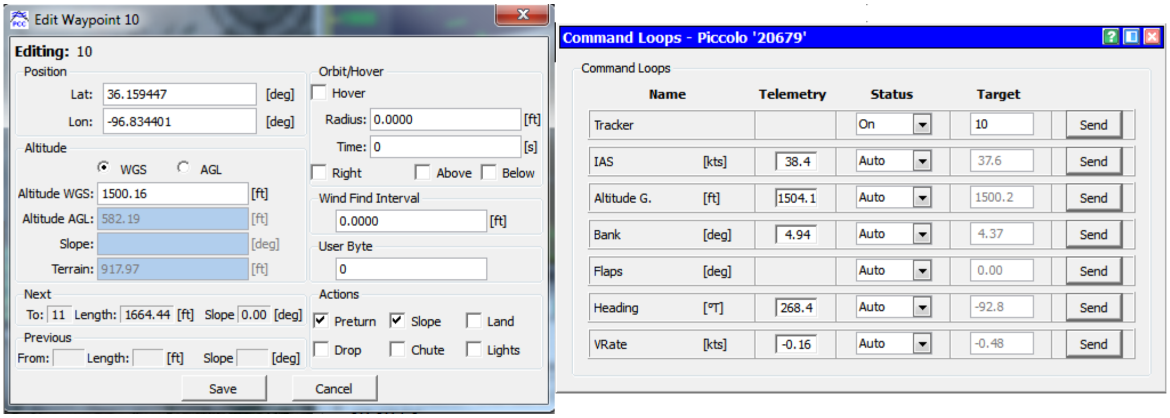
Figure 193 shows the Command Loops window in PCC. The Command Loops window allows the user to make changes and commands to specific command loops. The units presented in the command loops window depends on what the units are set to in PCC; however, the command loops themselves use SI and all the recorded data will be in SI regardless of the PCC units setting.

5.5.1. Tracker

The Tracker navigates the aircraft based on the target waypoint set by the user and the waypoints in the flight plan that the user designates the aircraft to fly on. The Tracker's state will always be "On" unless the user turns the Heading or Bank loops to "On" and commands a specific heading, or Bank angle. Turning the Altitude or VRate loops to "On" will not turn off Tracker. In such a situation the aircraft will track the waypoints in the flight plan, but it will ignore the altitude of the waypoints and target altitude according to the manual inputs.

Depending on the number of GPS satellites used and the Sensor Configuration that has been set the Tracker will use either a GPS based solution or a Barometer/IMU based solution to determine the aircraft's location, groundspeed, vertical rate, and velocity vectors.

When the user commands a waypoint for the aircraft to target ("On" state) Tracker uses the aircraft's current orientation and location in addition to the target waypoint's location to determine the path to track. Tracker will set the Altitude and Heading commands according to the path. Figure 194 shows the info of a targeted waypoint and the command loops window at that time.



Edit Waypoint 10

Editing: 10

Position

Lat: 36.159447 [deg]

Lon: -96.834401 [deg]

Altitude

● WGS ○ AGL

Altitude WGS: 1500.16 [ft]

Altitude AGL: 582.19 [ft]

Slope: [deg]

Terrain: 917.97 [ft]

Next

To: 11 Length: 1664.44 [ft] Slope 0.00 [deg]

Previous

From: Length: [ft] Slope [deg]

Orbit/Hover

□ Hover

Radius: 0.0000 [ft]

Time: 0 [s]

□ Right □ Above □ Below

Wind Find Interval

0.0000 [ft]

User Byte

0

Actions

☑ Preturn ☑ Slope □ Land

□ Drop □ Chute □ Lights

Command Loops - Piccolo '20679'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	10	Send
IAS	[kts]	38.4	Auto	37.6	Send
Altitude G.	[ft]	1504.1	Auto	1500.2	Send
Bank	[deg]	4.94	Auto	4.37	Send
Flaps	[deg]		Auto	0.00	Send
Heading	[°T]	268.4	Auto	-92.8	Send
VRate	[kts]	-0.16	Auto	-0.48	Send

Figure 194 Tracking Example

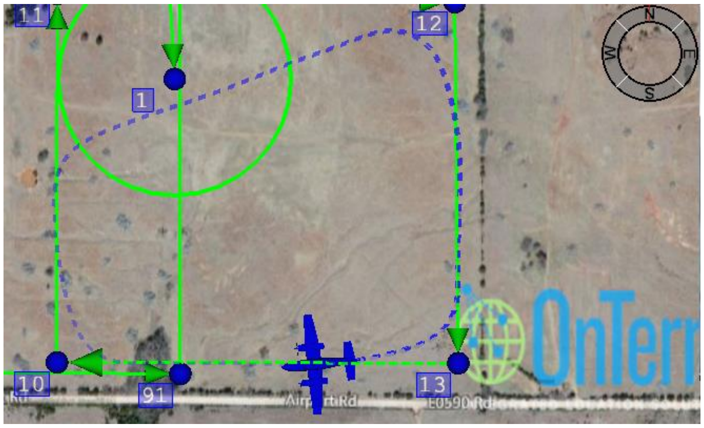


Figure 195 Tracking Example Flight Paths

Looking at the figure it can be seen the Heading target is nearly straight west of the aircraft just like the target waypoint. Note that the heading telemetry is reported on a different scale than the heading target.

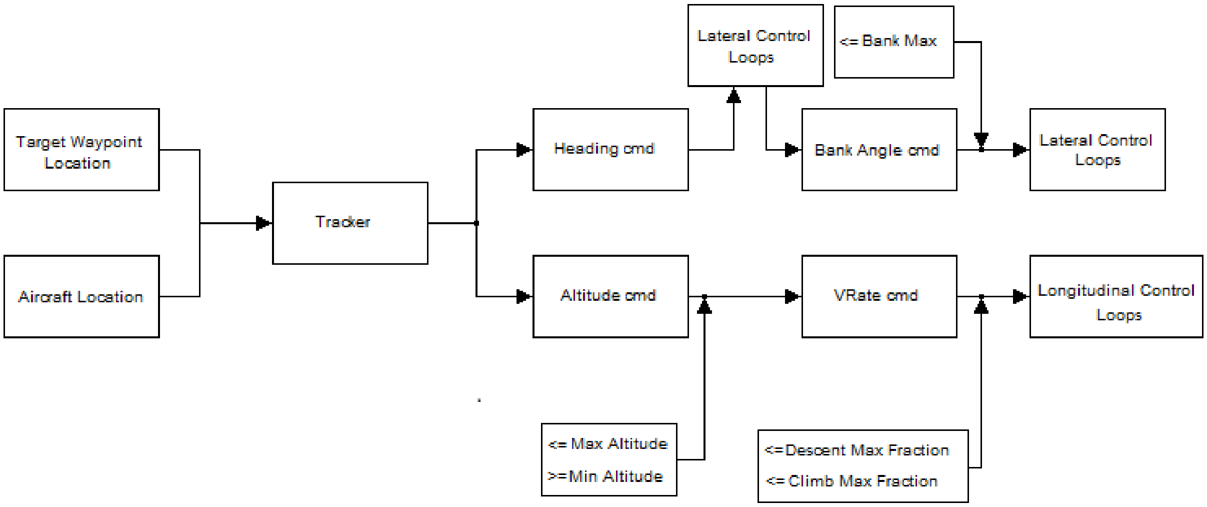


Figure 196 Tracker Flow Chart

Figure 196 is a flow chart of the commands that come from Tracker. The target waypoint is either designated by the user during flight, or designated by the flight plan that the aircraft is navigating.

5.5.2. Altitude & VRate Commands

In AUTO the Altitude command comes from Tracker. The behavior of the Altitude command is dependent on how the target waypoint is targeted. If the new target waypoint is simply the next waypoint in the current flight plan that the aircraft is navigating then the autopilot will command a constant slope, climb or descent, (ramp input) throughout the duration of the climb or descent.

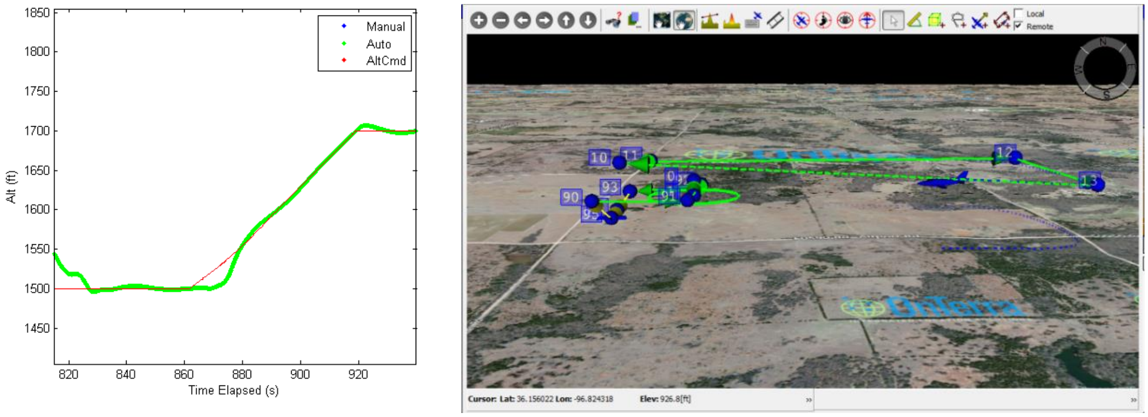


Figure 197 Altitude Command Ramp Input

Figure 197 illustrates a ramp input for the Altitude command. The autopilot was targeting Waypoint 10 and since it was on the flight plan the autopilot tracked the slope of the flight path.

The Altitude command determines the vertical rate command or VRate command. The VRate command actually has its own control loop that is an inner loop to the Altitude Control Loop. The VRate command is calculated from the altitude error feedback in the Altitude Control Loop.

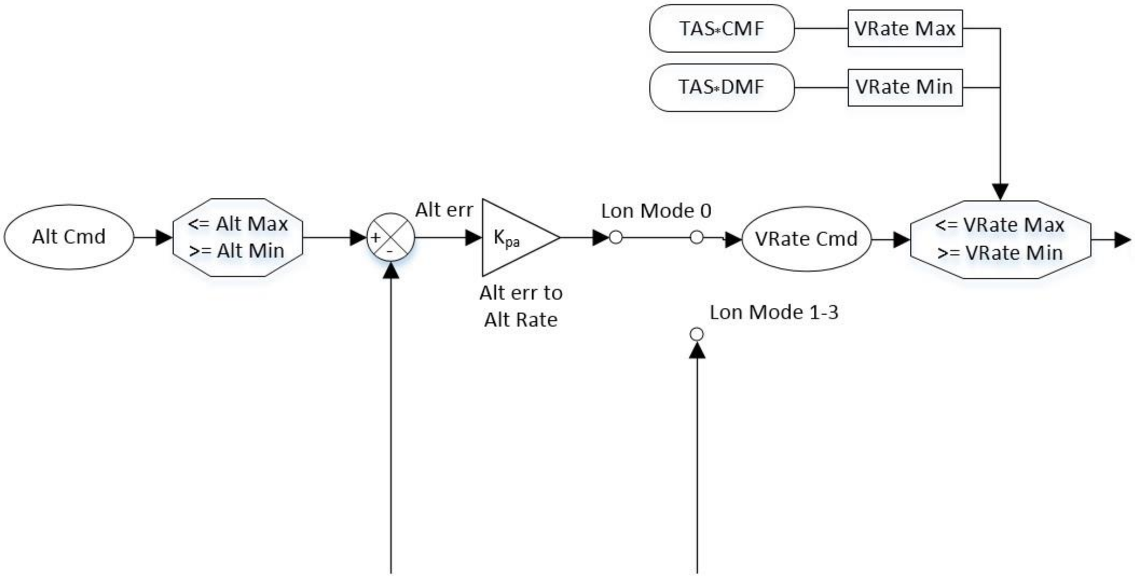


Figure 198 Altitude & Vertical Rate Command

Figure 198 is a snapshot of the Altitude Control Loop. From the figure it can be seen that the altitude command goes through the altitude control outer loop and commands the appropriate VRate based on the altitude error. The outer loop will alter the VRate command throughout a climb in order to eliminate Altitude Error, and is dependent on the magnitude of the proportional gain "Alt err to Alt Rate." Figure 199 shows the VRate command of the climb in Figure 198.

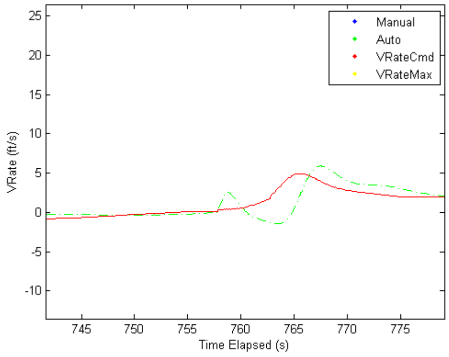


Figure 199 VRate Command

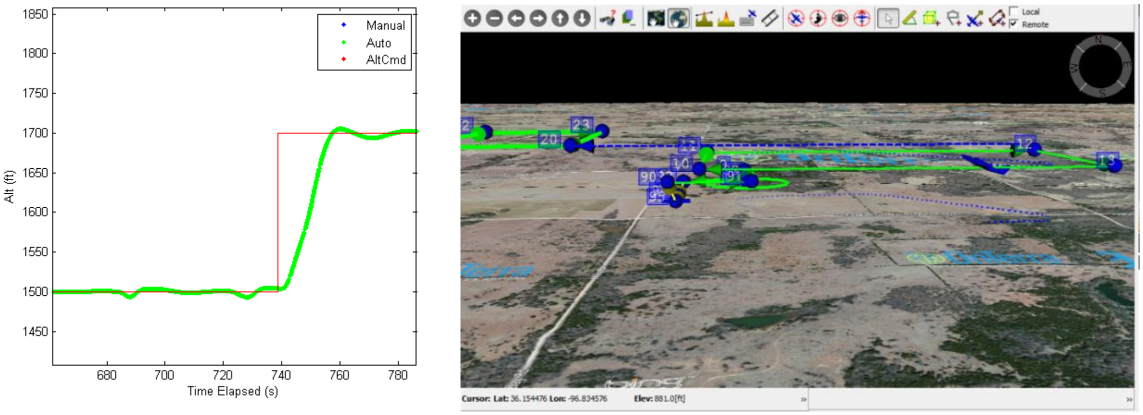


Figure 200 Altitude Command Step Input

If the target waypoint lies in a different flight plan or is out of order of the current flight plan the Altitude command will command the new altitude as a step input. Figure 200 depicts a scenario where the autopilot targeted a Waypoint (waypoint 20) located on a different flight plan. In that scenario the autopilot immediately set the target altitude as the altitude of the new waypoint, effectively acting as a step input. Usually in such a scenario the autopilot will end up commanding the maximum or minimum vertical rate because of the large Altitude Error due to the step input. Recall that the VRate max and min are set by the Climb and Descent Max Fractions in the Limits tab of the Controller Configuration window.

5.5.3. Heading & Bank Angle Commands

The Heading command comes directly from the location of the targeted waypoint, specified by Tracker, and the location of the aircraft. The Heading command and Heading telemetry are based off two different directional numbering schemes. Tracker commands Heading defining North as 0°, East as 90°, South as -/+ 180°, and West as -90°. This numbering scheme is depicted in Figure 201 and is used for rudder doublets in the Doublet File Method. Heading telemetry, recorded in the PCC log file, and displayed in PCC during flight, utilizes the same numbering scheme as Yaw angle. Yaw angle is defined as 0° North and progressing clockwise full circle to 360°. The Yaw angle numbering scheme is depicted in Figure 202.

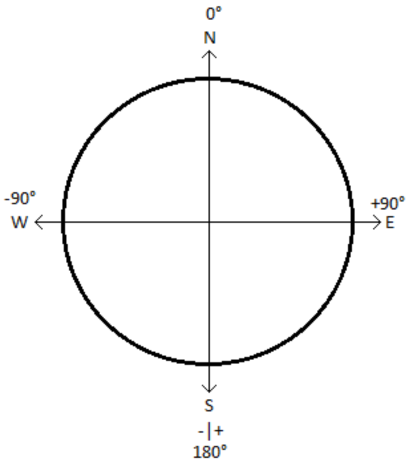


Figure 201 Heading Command Format

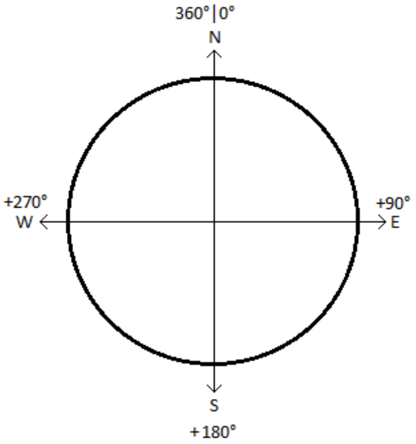
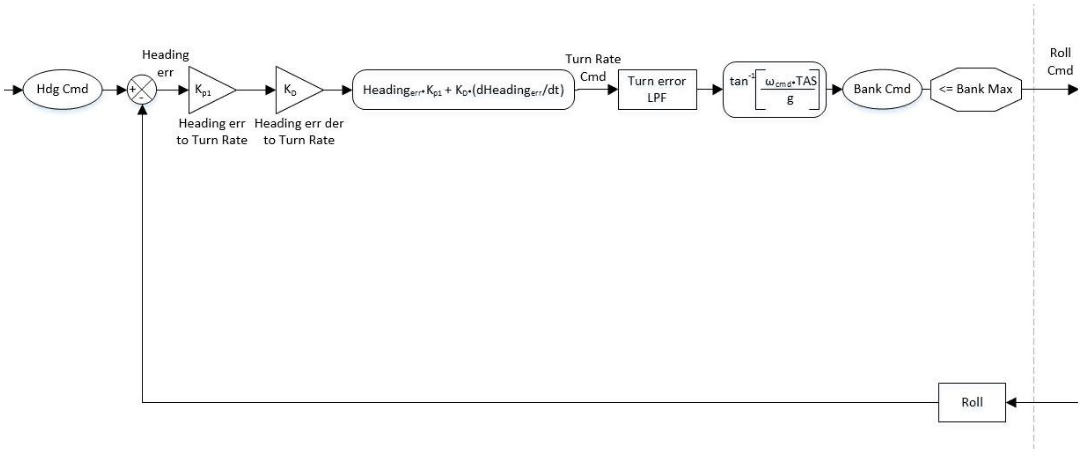


Figure 202 Heading Telemetry Format

In the command loop window the target Heading is -92.8° and the current Heading telemetry is reporting 268.4° which is nearly on target (180+92.8 = 272.8). The Heading command is routed through the Lateral Control loop called "Track Control." Track Control produces the Bank angle command.



Hdg Cmd
Heading err
+
-
K_{p1}
Heading err to Turn Rate
K_D
Heading err der to Turn Rate
$Heading_{err} \cdot K_{p1} + K_D \cdot (dHeading_{err}/dt)$
Turn Rate Cmd
Turn error LPF
$\tan^{-1}\left[\frac{\omega_{cmd} \cdot TAS}{g}\right]$
Bank Cmd
<= Bank Max
Roll Cmd
Roll

Figure 203 Heading Loop

Figure 203 is a snapshot of the Track Control Loop. The Bank angle command comes from the Heading command through the Track Control Loop. Bank angle is commanded to change the orientation of the aircraft so that the aircraft's GPS velocity vectors are pointing in the direction of the heading command. The Bank angle command is limited by "Bank Max". When the aircraft is flying into a turn it will turn the maximum amount that the bank angle maximum will allow if the Heading command requires a Bank angle greater than or equal to the bank angle maximum. The bank angle command is routed into the Later Control loop "Roll Control". Roll Control ultimately concludes with an aileron deflection command. Section 5.8.1.2 addresses Roll Control in greater detail.

5.5.4. IAS Command Loop

In AUTO the IAS command comes from specific vehicle gains, set in the Control Configuration window of PCC.

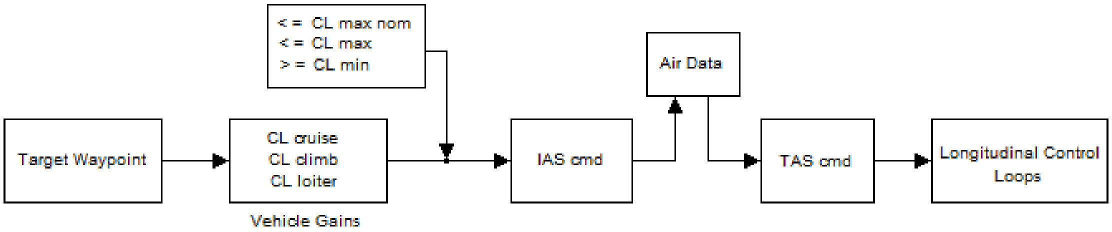


Figure 204 IAS Command Loop Flow Chart

Depending on the state of the path of the aircraft the autopilot will use CL values to determine the commanded indicated airspeed (IAS). If the aircraft is climbing the autopilot will use “CL climb” to command IAS. If the aircraft is orbiting or flying a loiter waypoint the autopilot will use “CL loiter” to command IAS. If the aircraft is traveling along a level flight path the autopilot will command IAS based on “CL cruise”. IAScmd is limited by CL max nom (used in normal flight), CL max (used for landing logic), and CL min.

To calculate indicated airspeed from the appropriate CL setting the controller uses the standard lift equation with density equal to air density at sea level, and lift equal to the current weight estimate as illustrated in Equation 25.

Equation 25 IAS Command

$$L = W = \frac{1}{2}\rho S V^2 C_L$$

$$\Rightarrow \; IAS_{cmd} = \sqrt{\frac{2*(mg)}{C_{L\,cmd} * S_w * \rho_o}}$$

$$where\ S_w = Wing\ Area\ (m^2) \quad \rho_o = 1.225\ {}^{kg}/_{m^3}$$

$$m = mass\ (kg) = \ m_{empty} + m_{fuel} \quad g = 9.81\ {}^{m}/_{s^2}$$

The empty mass is taken from the vehicle gain "Empty Mass" and the mass of the fuel is taken from the current fuel mass estimate. The fuel mass is initially determined by subtracting the vehicle gains "Gross Mass" and "Empty Mass", and the fuel is adjusted based on the estimated fuel burn throughout the flight. The wing area also comes from the vehicle gains.

Since there is no documentation that explicitly states that Equation 25 is used by the autopilot to calculate the IAS command the following example is provided as a proof.

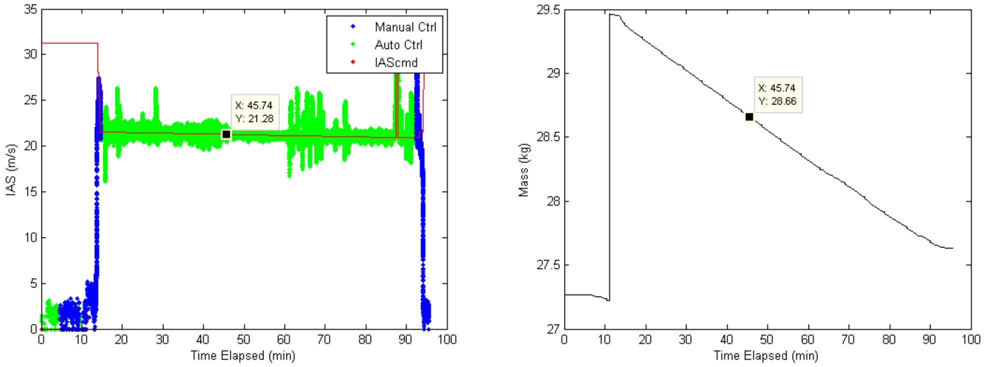


Figure 205 IAS Command Example

Figure 205 depicts data from an actual flight (Noctua B1 Flight 3). The aircraft in the example has a wing area of 1.2663m^2 and CL$_{cruise}$ is set to 0.8. Figure 205 shows the estimated mass at 45.74 minutes to be 28.66kg. The following calculation was done with Equation 25.

$$IAS_{cmd} = \sqrt{\frac{2 * (28.66 * 9.81)}{0.8 * 1.2663 * 1.225}} = 21.28\ m/s$$

The calculated IAS cmd of 21.28 m/s matched the reported IAS cmd in Figure 205. IAS cmd is routed through the air data and converted into true airspeed (TAS) as described in Section 5.1.2. After the command is converted to true airspeed the command is routed into the longitudinal control loops. More specifically the TAS command will be sent to Energy Control.

If the autopilot is in Airspeed Control the TAS command will also be sent to the Airspeed Control Loop to ultimately command elevator deflections.

5.5.5. Enabling/Disabling Command Loops

In general command loops can be turned “Off”, “On” (manual input), and returned to “Auto” mode. The Universal Controller will only allow outer command loops (Altitude, Heading, Tracker) to be turned off, but they cannot be turned off directly. Even though the Command Loop window offers an “Off” option, none of the command loops can actually be turned off in that manner. The “Off” option is used for presenting a command loop’s state as being turned off rather than having any real operational functionality. In order to turn off outer command loops, specific inner command loops must be turned “On” with a manual input value. In some cases switching command loops to the “On” state can affect the states of other command loops.

The following presents the states that loops are capable of being changed too and how the changes affect other command loops:

1) Heading ON.

   - Tracker “Off”. Note if Altitude is on “Auto” it will be turned to “On” and it will be commanding the same altitude that was being commanded at the time of turning off Tracker.

   - Bank “Auto”

2) Bank ON

   - Tracker “Off”

   - Heading “Off”

3) Tracker ON

- Heading “Auto”

- Bank “Auto”

4) VRate ON

- Altitude “Off”

5) Altitude ON

- VRate “Auto”

6) Altitude AUTO

- VRate “Auto”

7) IAS ON

- No effect on the states of other command loops

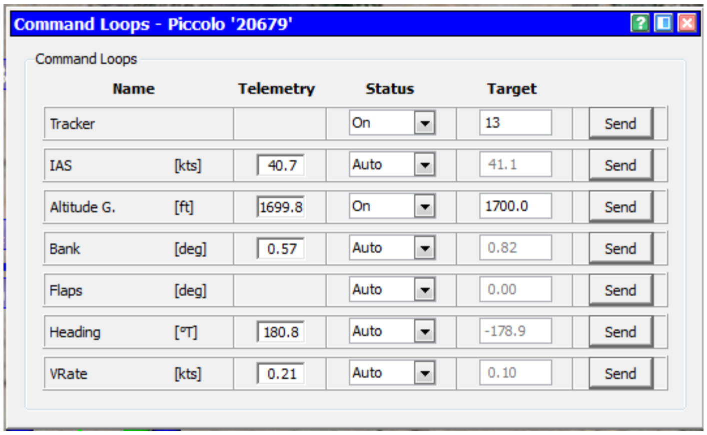
8) IAS AUTO

- No effect on the states of other command loops

It is important to note that only IAS and Altitude can be switched to “Auto”. The command loops associated with lateral control can only be turned to “Auto” by turning their corresponding outer command loops to “On”.

There are two different methods available to manually command a value for command loops or switch them to the “On” state.

1) Command Loops Window



Command Loops - Piccolo '20679'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	13	Send
IAS	[kts]	40.7	Auto	41.1	Send
Altitude G.	[ft]	1699.8	On	1700.0	Send
Bank	[deg]	0.57	Auto	0.82	Send
Flaps	[deg]		Auto	0.00	Send
Heading	[°T]	180.8	Auto	-178.9	Send
VRate	[kts]	0.21	Auto	0.10	Send

Figure 206 Command Loop Status On

All of the command loops can be turned on in the command loop window by selecting the drop down menu next to each command loop and selecting "On", entering a value into the empty box in the column labeled "Target" and clicking "Send". In Figure 206 Altitude has been turned to "On" and commanded 1700 ft.

2) Primary Flight Display (PFD)

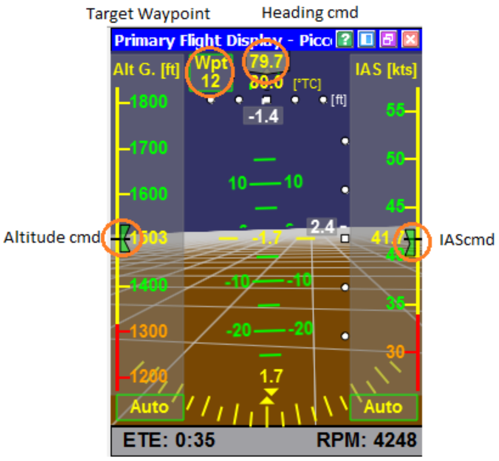
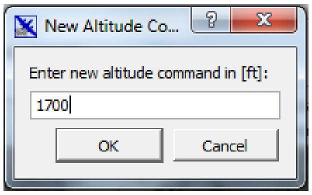


Figure 207 Primary Flight Display Commands

PCC's Primary Flight Display can be used to manually command IAS, altitude, heading, and the target waypoint. To change commands using the PFD refer to the following steps for each.

a) Altitude Command. Double click the green slider on the left side of the PFD that shows the commanded altitude and an input window will pop up asking for the "new altitude command".

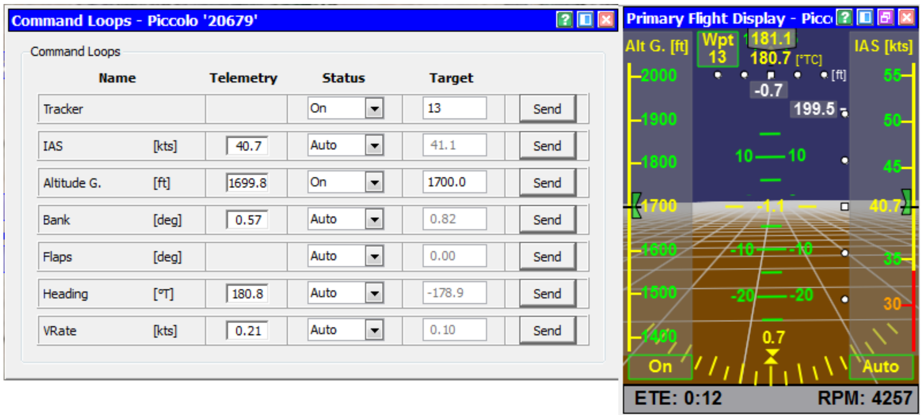

New Altitude Co...

Enter new altitude command in [ft]:

1700

Figure 208 New Altitude Command

Enter the desired altitude in the units that PCC is currently in. In Figure 208 an altitude of 1700 ft was commanded. Figure 209 below shows the corresponding changes in the command loop window. The status of the Altitude command loop state has been changed from "Auto" to "On" and the target is now 1700ft.


Command Loops - Piccolo '20679'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	13	Send
IAS	[kts]	40.7	Auto	41.1	Send
Altitude G.	[ft]	1699.8	On	1700.0	Send
Bank	[deg]	0.57	Auto	0.82	Send
Flaps	[deg]		Auto	0.00	Send
Heading	[°T]	180.8	Auto	-178.9	Send
VRate	[kts]	0.21	Auto	0.10	Send

Primary Flight Display - Picc

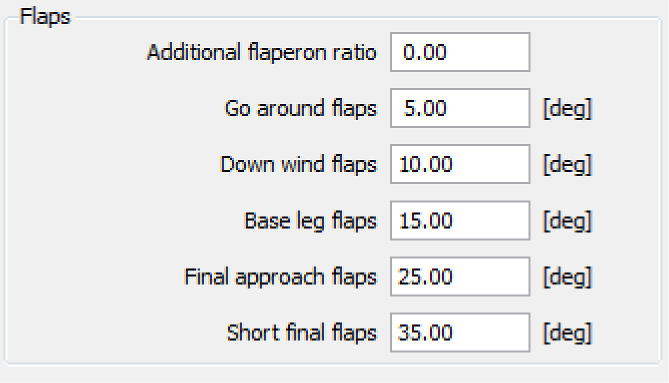
Figure 209 Altitude Command On

5.5.6. Flaps

As stated previously Flaps will not be deployed in the “Auto” state. The autopilot will command 0 flap deflection at all times in the “Auto” state.

Flap deflection can be commanded by the remote pilot through the command loops window, similar to the other command loops. Select “On” from the drop down menu in the “Status” column of flaps. The flaps will deflect, upon clicking “Send”, the amount of deflection set in the “Target” input box.

Flap deflections can be commanded by the autopilot in land plans without action from the remote user. In order to do so the flap command loop status must be set to “Auto”. In the “Landing” tab of the “Controller Configuration” window in PCC the flap deflection for each leg of the landing plan must be defined.



Additional flaperon ratio	0.00	
Go around flaps	5.00	[deg]
Down wind flaps	10.00	[deg]
Base leg flaps	15.00	[deg]
Final approach flaps	25.00	[deg]
Short final flaps	35.00	[deg]

Figure 210 Landing Flap Settings

The autopilot will command the preset flap deflections according to each leg of the landing plan. Note that if the command loop status is “Off” or “On” the preset landing deflections will not be used.

5.6. Autopilot Modes

Autopilot modes define the state of the autopilot. The state the autopilot is in determines the Control Laws that are actually utilized and whether or not actuator commands are actually issued. Mode 4 “Flying” is the mode that the autopilot will be in during flight. All of the other modes deal with takeoff and landing.

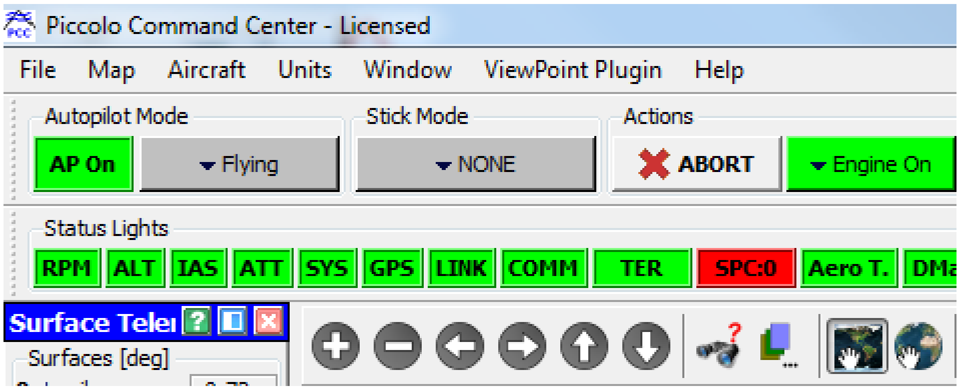
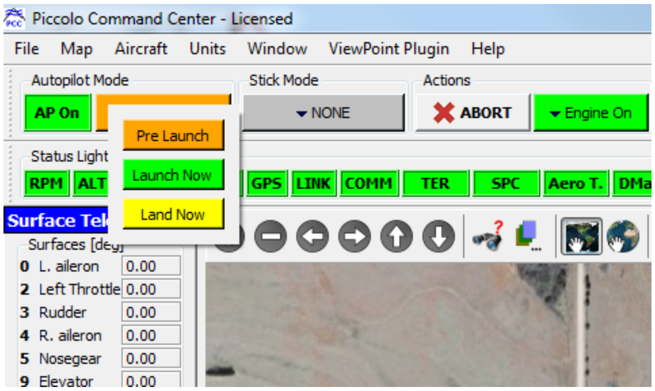


Figure 211 Autopilot Mode Display

The mode of the autopilot is displayed in the upper left corner of PCC shown in Figure 211. The autopilot can be forced into “Pre Launch”, “Launch Now”, or “Land Now” by clicking on the drop down box.



0	L. aileron	0.00
2	Left Throttle	0.00
3	Rudder	0.00
4	R. aileron	0.00
5	Nosegear	0.00
9	Elevator	0.00

Figure 212 Autopilot Mode Choices

Clicking “Launch Now” will start the auto takeoff process. Clicking “Land Now” will force the autopilot to target the first waypoint of the current land plan that exists on board the piccolo. All of the different flight modes are detailed in Cloud Cap’s PccUsersGuide pg. 101.

5.7. Control Loops

Fixed Wing Generation Two version Seven has 13 control loops that contain 41 control gains. The control loops are utilized via control laws, or control logic, with input commands from the command loops. The control loops contain varying control structures to condition command loop commands into actuator commands.

The purpose of this section is simply to provide the definitions of the gains of each control loop along with a control schematic of each control loop. This section does not address the control logic, or how loops interact with each other to ultimately command actuator deflections. The following section, Section 5.8 Control Laws, addresses these details.

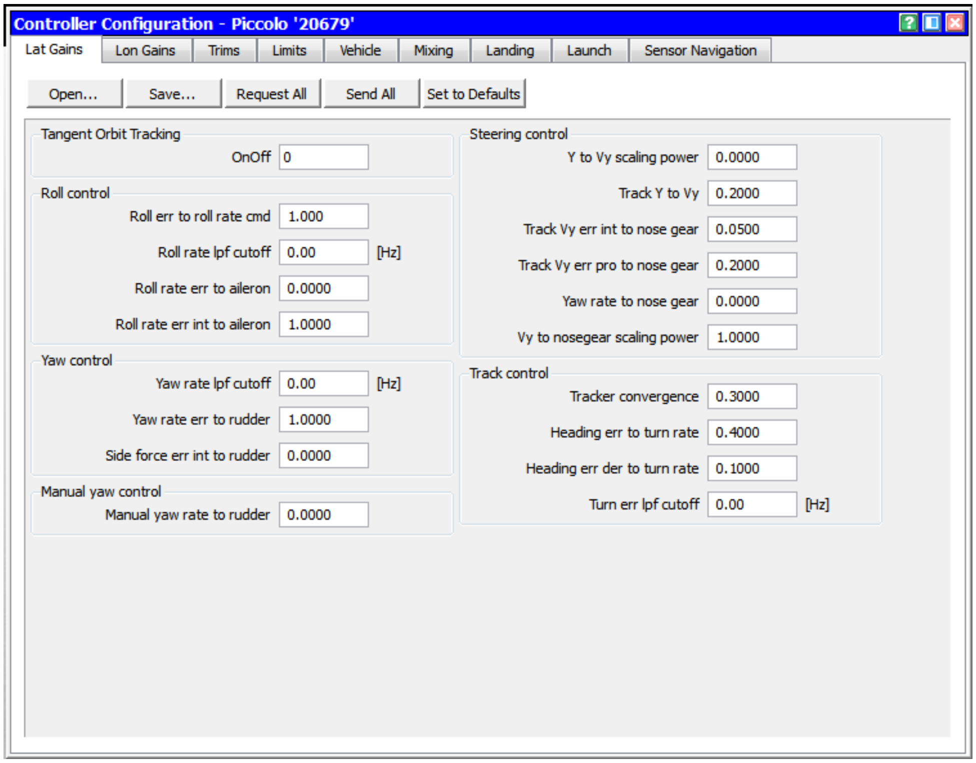
Unfortunately Cloud Cap only offers fragmented pieces of information in their documentation that shed light on how the control loops are structured. A lot of the information on the control loops can be found from the tables of gain definitions in “PccUsersGuide.pdf” (pgs. 110-113). Combined with the user’s guide there is another document, “Tuning piccolo control laws 2.0.x.pdf” that provides further insight into control loops and gain definitions; however, this document is out of date and portions of it are not correct for 2.1.4.x. As a result only the gains definitions that were later found to be current for version 2.1.4.x are included with the PccUsersGuide gain definitions.

5.7.1. Lateral Control Loops

The Lateral Control Loops are concerned with directional control of the aircraft. Lateral Control consists of 4 control loops: Roll Control, Yaw Control, Steering Control, and Track

Control. All the control loops, combined with the control logic, ultimately issue commands for Aileron, and Rudder deflections.

The Lateral Control Loop gains can be viewed in the PCC window "Controller Configuration", "Lat Gains" tab.



Tangent Orbit Tracking
OnOff 0

Roll control
Roll err to roll rate cmd 1.000
Roll rate lpf cutoff 0.00 [Hz]
Roll rate err to aileron 0.0000
Roll rate err int to aileron 1.0000

Yaw control
Yaw rate lpf cutoff 0.00 [Hz]
Yaw rate err to rudder 1.0000
Side force err int to rudder 0.0000

Manual yaw control
Manual yaw rate to rudder 0.0000

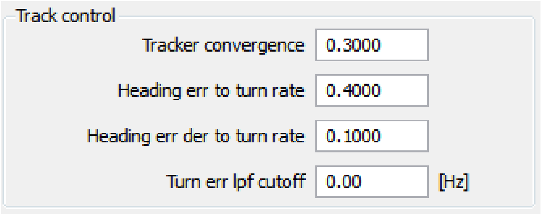
Steering control
Y to Vy scaling power 0.0000
Track Y to Vy 0.2000
Track Vy err int to nose gear 0.0500
Track Vy err pro to nose gear 0.2000
Yaw rate to nose gear 0.0000
Vy to nosegear scaling power 1.0000

Track control
Tracker convergence 0.3000
Heading err to turn rate 0.4000
Heading err der to turn rate 0.1000
Turn err lpf cutoff 0.00 [Hz]

Figure 213 Lateral Gains Window

5.7.1.1. Track Control Loop

The Track Control Loop is the outer loop to the Roll Control Loop. Track Control receives commands from the Heading command and outputs a turn rate.



Track control

Tracker convergence	0.3000	
Heading err to turn rate	0.4000	
Heading err der to turn rate	0.1000	
Turn err lpf cutoff	0.00	[Hz]

Figure 214 Track Control Gains

The Track Control Loop consists of 2 gains, 1 filter, and 1 parameter. "Tracker Convergence" is a parameter as it is more of a setting than a gain. Tracker Convergence is actually outside of the Track Control Loop. Tracker Convergence influences the Heading command by plotting the elliptical trajectory that the autopilot attempts to navigate when targeting track lines.

Tracker Convergence: Tracker convergence parameter in dimensionless units. Decreasing this number causes the vehicle to try to fly more closely to the track. Making this value too small will cause track oscillations (PccUsersGuide 111).

"Tracker convergence" is a lateral gain that gives a dimensionless number setting the size of the elliptical trajectory used to guide the vehicle onto track. The track controller plots an elliptical trajectory to guide the vehicle onto the desired track line. The ellipse is defined by a semi-major axis (a) which is four times the semi-minor axis. The semi-minor axis is computed with the square of the speed and the tracker converge parameter, which defaults to 0.35. Making this value smaller will cause the aircraft to more rapidly converge onto a track segment. Making it too small could exceed the bandwidth of the track controller, causing track oscillations (Tuning piccolo control laws 2.0.x 15)

Heading err to turn rate: Gain relating heading error (rad) to turn rate (rad/s). Used to provide the primary steering input from either the heading controller or the track controller. The available gain depends on the bandwidth of the inner loop lateral controller (PccUsersGuide 111).

"Heading err to turn rate" is a lateral gain that sets the desired turn rate (rad/s) as a function of the error in heading (rad). This gain sets how hard the vehicle attempts to turn in response to an error in heading. Since turn rate is really the derivative of heading than this gain basically sets the time constant required to null a heading error. The default gain is 0.4. Increasing this gain will improve tracker bandwidth, but going too far will cause heading (bank angle) oscillations (Tuning piccolo control laws 2.0.x 15).

Heading error derivative to turn rate: Gain relating the derivative of heading error (rad) to turn rate (rad/s). For vehicles with poor inner loop lateral control bandwidth this gain can help reduce track oscillations, otherwise this gain can be zero (PccUsersGuide 111).

"Heading err der to turn rate" is a lateral gain that sets the desired turn rate (rad/s) as a function of the rate of change of error in heading (rad/s). The derivative term helps to slow the track controller down as it gets close to the desired heading. This can be useful for systems that have lag in their bank angle control. Finding the optimal value for this gain is challenging. The default value is 0.1. Too much gain will cause fast bank angle oscillations, but too little gain may cause slow oscillations due to inner loop lag. For vehicles that have fast bank angle control it may be best to set this gain to zero (Tuning piccolo control laws 2.0.x 15).

Turn error lpf cutoff: A low pass filter used to estimate the vehicle asymmetry, which is the turn rate that results at zero bank angle. Set to zero to disable. Enabling this allows the autopilot to correct for asymmetrical vehicles when tracking on a flight plan (PccUsersGuide 111).

Figure 215 is a schematic of the Track Control Loop as it is defined by the gain definitions.

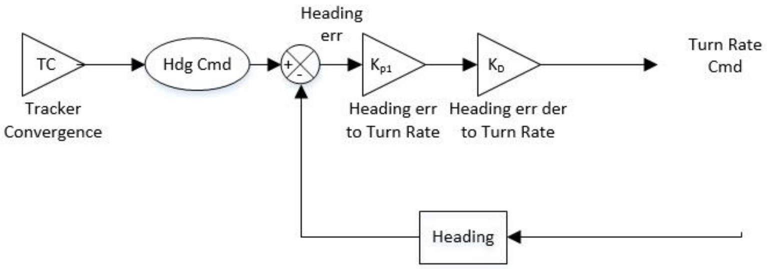
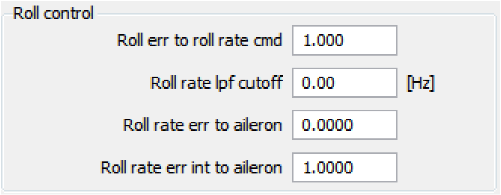


Figure 215 Track Control Loop

5.7.1.2. Roll Control Loop

The Roll Control Loop contains an inner and outer loop. The outer loop receives the Bank Angle Command and outputs a Roll Rate command. The inner loop receives the Roll Rate command and outputs an Aileron Deflection.


Roll control

Roll err to roll rate cmd	1.000	
Roll rate lpf cutoff	0.00	[Hz]
Roll rate err to aileron	0.0000	
Roll rate err int to aileron	1.0000	

Figure 216 Roll Control Gains

Roll Control consists of 3 feedback gains, 1 filter, and 2 Vehicle Parameters. The 2 Vehicle Parameters used are “Aileron Effectiveness” and “Wingspan”. The inner loop utilizes Aileron Effectiveness, combined with the Wingspan, as a feed forward gain. Aileron Effectiveness also scales the inner loop feedback gains.

Roll err to roll rate cmd: Gain relating roll angle error (rad) to roll rate command (rad/s). Increasing this gain increases the available bandwidth of the inner loop lateral control. Do not zero this gain since that will disable lateral control (PccUsersGuide 110).

Roll err to roll rate cmd" is a lateral gain that computes the desired roll rate from the bank angle error. The desired roll rate is limited to be less than p MAX . The inverse of this gain is the time constant required to reduce the error by 60%. This is an outer loop gain which sets how fast the system attempts to reduce the bank angle error. This gain has a default setting of 1.0. In general vehicles with greater aileron bandwidth (fast big ailerons combined with low roll inertia) can tolerate a larger gain and vice versa. Increasing this gain will provide more bank angle control bandwidth and an attendant improvement to tracking (Tuning piccolo control laws 2.0.x 6).

Roll rate lpf cutoff: Roll rate low pass filter cutoff frequency (Hz). Zero to disable. This filter can be used to remove high frequency noise on the roll rate signal. Enabling this filter will reduce the available bandwidth on the lateral control loop (PccUsersGuide 110).

Roll rate err to aileron: Gain relating roll rate error (rad/s) to aileron output (rad). Used to increase the roll damping of the vehicle. Most conventional fixed wing aircraft do not need extra roll damping and this gain can be zero (PccUsersGuide 110).

Roll rate err int to aileron: Gain relating the integral of roll rate error ((rad/s)*s) to aileron output (rad). This gain is used to trim errors in the ailerons. Increasing this gain increases the rate at which the autopilot can respond to events that change the aileron trim. Do not zero this gain since that will disable the ability of the autopilot to trim out aileron errors (PccUsersGuide 110).

"Roll rate err int to aileron" is a lateral gain that computes the aileron deflection as the time integral of the error between the desired roll rate and the actual roll rate. Increasing this gain will improve the ability of the controller to achieve the desired roll rate even if the aileron effectiveness parameters is not correct. Increasing this gain too far will cause oscillations. The next most important parameters is the roll rate error integrator. It is the job of this term to find the aileron trim, and to account for any errors in the aileron effectiveness. The default gain for this term is 1.0. If the aileron effectiveness is correctly estimated there is little benefit to making this term larger or smaller since the roll rate error will be small in that case. If the aileron effectiveness is wrong, or if the aileron trim changes rapidly due to other effects, than increasing this term will improve the ability of the controller to adapt to the errors. Increasing the gain too much will cause fast oscillations of the bank angle. If the system exhibits fast oscillations in roll, and the aileron effectiveness has been set correctly, than try reducing this term (Tuning piccolo control laws 2.0.x 6)

Figure 217 below is a schematic of the Roll Control Loops as described by the definitions of the Roll Control Loop gains. Note that the outer loop is actually an inner loop to the Track Control Loop.

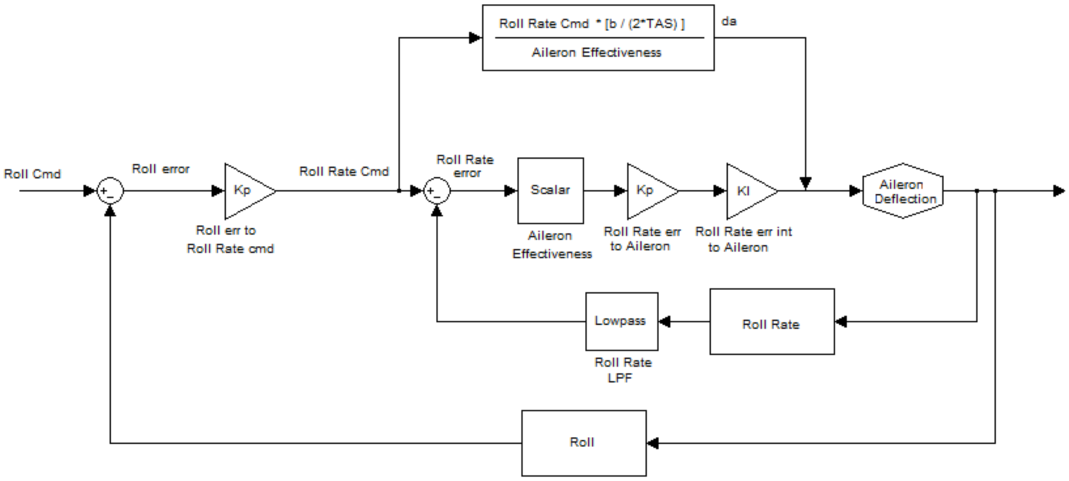
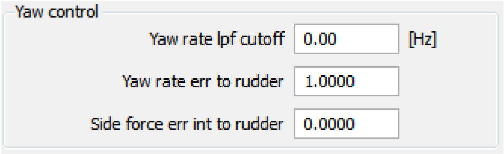


Figure 217 Roll Control Loop

5.7.1.3. Yaw Control Loop

The yaw control loop receives the yaw rate command and commands rudder deflections. The yaw control loop is a part of Yaw Control where it is combined with the Side Force Control

Loop in order to command rudder deflections. The yaw control loop actually operates as a yaw damper.



Yaw control

Yaw rate lpf cutoff 0.00 [Hz]

Yaw rate err to rudder 1.0000

Side force err int to rudder 0.0000

Figure 218 Yaw Control Loop Gains

The Yaw Control Loop consists of 1 gain and 1 filter. "Side force err int to rudder" is a part of yaw control; however, it exists in the side force control loop.

Yaw Rate lpf cutoff: Yaw rate low pass filter cutoff frequency [Hz]. Zero to disable. This filter can be used to remove high frequency noise on the yaw rate signal. Enabling this filter will reduce the available bandwidth of the yaw damper (PccUsersGuide 110).

Yaw rate error to rudder: Gain relating yaw rate error [rad/s] to rudder output [rad]. Used to provide yaw damping. Conventional vehicles with large vertical tails and long tail moment arms typically do not need yaw damping, however short tailed vehicles, or vehicles with excessive dihedral effect usually do need it. Yaw damping is the best way to stop dutch roll (PccUsersGuide 111).

Figure 219 is a schematic of the Yaw Control Loop process as described by the definitions of the Yaw Control Loop gains.

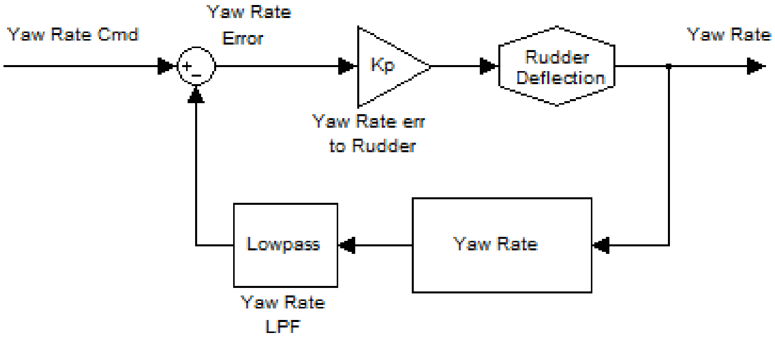
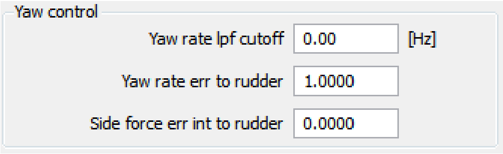


Figure 219 Yaw Control Loop

5.7.1.4. Side Force Control Loop

The Side Force Control Loop receives the Side Force command (which is always 0), and outputs rudder deflection.



Yaw control

Yaw rate lpf cutoff 0.00 [Hz]

Yaw rate err to rudder 1.0000

Side force err int to rudder 0.0000

The side force control loop only contains one gain, “Side force err int to rudder”.

<u>Side force err int to rudder:</u> Gain relating the integral of side force error [[m/s/s]*s] to rudder output [rad] while flying. Used to provide automatic turn coordination by driving the rudder to the position that zeros the side force. If this gain is zero the autopilot will attempt to coordinate the turn using vehicle parameter information. Note that turn coordination is usually not important unless the vehicle has a very long tail moment arm and flies slowly (PccUsersGuide 111).

Figure 220 is a schematic of the Side Force Control Loop as described by the definition of the Side Force Control Loop gain. Note that there are a few components of Yaw Control that are missing.

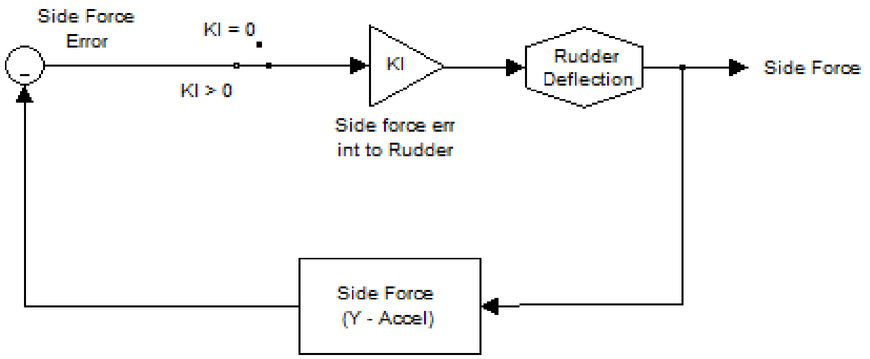
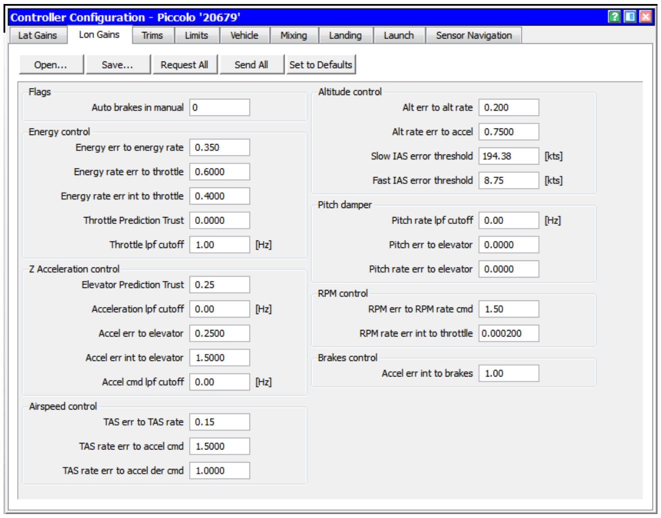


Figure 220 Side Force Control Loop

5.7.2. Longitudinal Control Loops

The Longitudinal Control loops are concerned with tracking Altitude and Airspeed. There are 4 Longitudinal Control Loops: Energy Control, Z Acceleration Control, Airspeed Control, Altitude Control, and 2 optional Control Loops: RPM Control, and Pitch Damping. All the control loops, combined with the control laws, ultimately issue commands for Throttle, and Elevator deflections.

The Longitudinal Control Loop gains can be viewed in the PCC window “Controller Configuration”, “Lon Gains” tab.



Flags
Auto brakes in manual: 0

Energy control
Energy err to energy rate: 0.350
Energy rate err to throttle: 0.6000
Energy rate err int to throttle: 0.4000
Throttle Prediction Trust: 0.0000
Throttle lpf cutoff: 1.00 [Hz]

Z Acceleration control
Elevator Prediction Trust: 0.25
Acceleration lpf cutoff: 0.00 [Hz]
Accel err to elevator: 0.2500
Accel err int to elevator: 1.5000
Accel cmd lpf cutoff: 0.00 [Hz]

Airspeed control
TAS err to TAS rate: 0.15
TAS rate err to accel cmd: 1.5000
TAS rate err to accel der cmd: 1.0000

Altitude control
Alt err to alt rate: 0.200
Alt rate err to accel: 0.7500
Slow IAS error threshold: 194.38 [kts]
Fast IAS error threshold: 8.75 [kts]

Pitch damper
Pitch rate lpf cutoff: 0.00 [Hz]
Pitch err to elevator: 0.0000
Pitch rate err to elevator: 0.0000

RPM control
RPM err to RPM rate cmd: 1.50
RPM rate err int to throttle: 0.000200

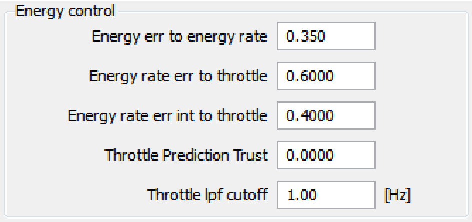
Brakes control
Accel err int to brakes: 1.00

Figure 221 Longitudinal Gains Window

Longitudinal Gain definitions come from “PccUsersGuide” pgs. 112 – 113. Specific gain definitions were supplemented with more detailed definitions from “Tuning piccolo control laws 2.0.x.” As stated in the intro to Control Loops some of the definitions are no longer current with version 2.1.4.i; therefore, some gain definition statements have been omitted, and some have been altered appropriately.

5.7.2.1. Energy Control Loop

The Energy Control Loop consists of 2 control loops. The outer loop receives commands from the TAS and Altitude commands and outputs an Energy Rate command. The inner loop receives the Energy Rate command from the outer loop and commands Throttle. Note that in Altitude Control the Energy Rate command will also come from the VRate command.



Energy control
Energy err to energy rate: 0.350
Energy rate err to throttle: 0.6000
Energy rate err int to throttle: 0.4000
Throttle Prediction Trust: 0.0000
Throttle lpf cutoff: 1.00 [Hz]

Figure 222 Energy Control Loop Gains

The Energy Control Loop contains 3 feedback gains, 1 feed forward gain, and 1 low pass filter. Also included in the Energy Control Loops is the Vehicle Parameter “Max Engine Power”. Max Engine Power is utilized in the feed forward gain, “Throttle Prediction Trust”, and is also used to scale the inner loop feedback gains.

Energy err to energy rate: Gain relating energy error to energy rate command. Increasing this gain increases how aggressively the autopilot moves the throttle to maintain energy (PccUsersGuide 112).

Energy rate err to throttle: Gain relating the energy rate error to throttle (PccUsersGuide 112).

Energy rate err int to throttle: Gain relating the integral of energy rate error to throttle. This gain must not be zero since it sets the throttle trim (PccUsersGuide 112).

Throttle Prediction Trust: Ratio (0.0 - 1.0) describing how much to trust the predicted throttle form vehicle parameters. Lower numbers are safer, higher numbers usually perform better. When prediction trust is 1.0 the throttle will instantly respond to changes in required power, according to power predicted from vehicle parameters. When prediction trust is 0.0 the throttle will only move in response to feedback errors (PccUsersGuide 112).

“Throttle prediction trust” is a longitudinal gain that determines how much of the predicted throttle should actually be used. The autopilot computes the amount of throttle required to achieve a given energy rate. The resulting throttle, times the throttle prediction trust, is effectively a proportional feed forward gain. The difficultly associated with predicting the actual engine power is such that the default value for this gain is zero. However you may be able to get better performance or responsiveness by increasing this value, up to 1.0. Larger throttle prediction trust values will cause the throttle to move more, and may cause excessive throttle motion. This is why the default value is set to zero. If maximum engine performance is desired than it is good to “linearize” the throttle actuation table. The autopilot assumes that the throttle is basically a power lever. However the typical small engine has a highly nonlinear relationship between throttle position and engine power. You can improve the performance by adjusting the throttle actuator table such that the power is in fact linear with the throttle command (Tuning piccolo control laws 2.0.x 10).

Throttle lpf cutoff: Throttle low pass filter cutoff frequency (Hz). Zero to disable. This filter can be used to quiet engine transients caused by sensor noise (PccUsersGuide 112).

Figure 223 is a schematic of the Energy Control Loops as described by the definitions of the Energy Control Loop gains. Note that there are several additional complexities that exist in Energy Control.

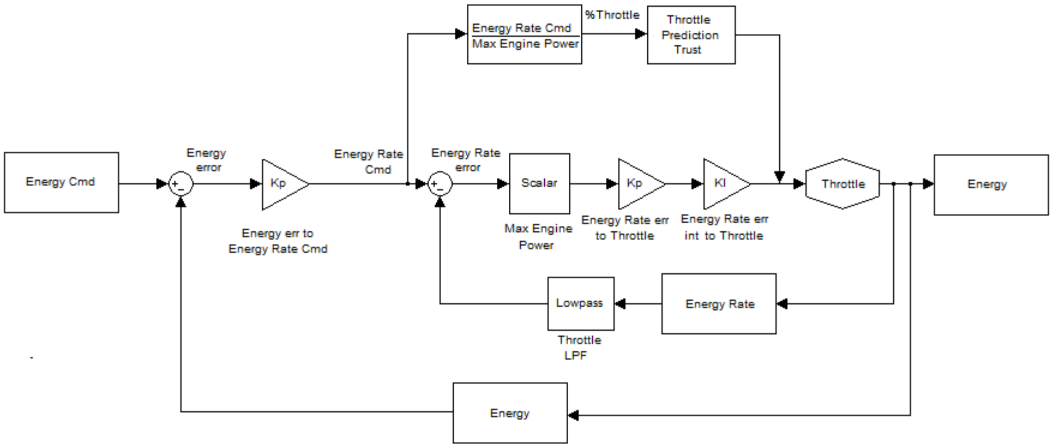
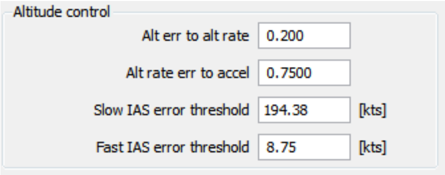


Figure 223 Energy Control Loop

5.7.2.2. Altitude & VRate Control Loops

The Altitude Control Loop contains an inner and outer loop. The outer loop receives the Altitude command and outputs the Vertical Rate command, or VRate command. The inner loop receives the VRate command and outputs a Z – Acceleration command. In the grand scheme of Altitude Control, if the autopilot is in Altitude Control, the Z – Acceleration command from these two loops will ultimately command Elevator deflections.



Altitude control

Alt err to alt rate 0.200

Alt rate err to accel 0.7500

Slow IAS error threshold 194.38 [kts]

Fast IAS error threshold 8.75 [kts]

Figure 224 Altitude & VRate Control Loop Gains

The Altitude Control Loop contains 2 feedback gains and 2 control limits. The “Slow IAS error Threshold” and “Fast IAS error Threshold” are used in the Longitudinal Control Laws to determine when the autopilot should switch from Altitude Control to Airspeed Control.

Alt err to alt rate: Gain relating altitude error (m) to commanded altitude rate (m/s). Must not be zero (PccUsersGuide 113).

This gain is simply the outer loop to the gain above, setting the desired vertical rate as a function of altitude error. The default value for this gain is 0.2. Too much gain will cause pitch oscillations and too little will impair the altitude control (Tuning piccolo control laws 2.0.x 13).

Alt rater err to accel: Gain relating altitude rate error (m/s) to Z acceleration command (m/s/s). This gain sets the bandwidth with which the vehicle tries to achieve the desired vertical rate. It must not be zero. In most cases this gain must be at least as large as “Alt err to alt rate” (PccUsersGuide 113).

The theory behind this gain is that an error in vertical rate is corrected by accelerating up or down. The gain has a default value of 0.75, hence most of the vertical rate command should be achieved in something like 2 seconds. Increasing this gain will increase the bandwidth of the altitude controller. Since the acceleration command appears directly on the elevator (via the elevator prediction trust) making this gain too high can cause pitch oscillations. Making this gain too small will impair the ability of the autopilot to regulate altitude. (Tuning piccolo control laws 2.0.x 13).

<u>Slow IAS error Threshold:</u> The amount that the longitudinal controller is allowed to let the airspeed fall below command (when throttle is full) before airspeed control takes priority over altitude control (PccUsersGuide 13).

<u>Fast IAS error Threshold:</u> The amount that the longitudinal controller is allowed to let the airspeed exceed command (when throttle is idle) before airspeed control takes priority over altitude control. Use a negative value to make airspeed control always have priority (PccUsersGuide 13).

Figure 225 is a schematic of the Altitude Control Loop based on the gain definitions.

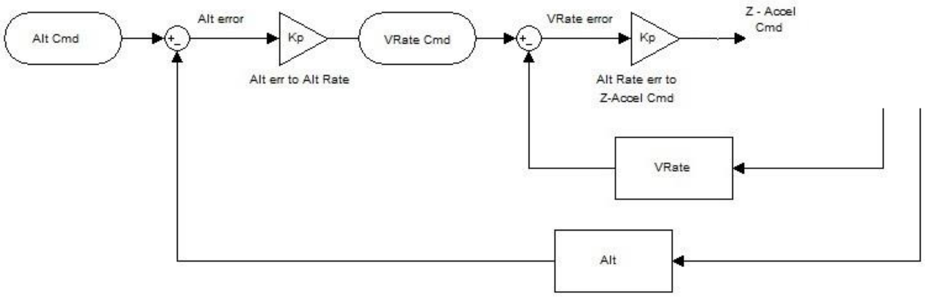
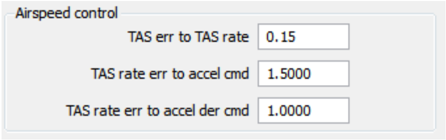


Figure 225 Altitude & VRate Control Loops

5.7.2.3. Airspeed Control Loop

The Airspeed Control Loop contains an inner and outer loop. The outer loop receives the True Airspeed command (TAS command), and outputs the True Airspeed Rate command (TASRate command). The inner loop receives the TASRate command and outputs a Z – Acceleration command. In the grand scheme of Airspeed Control, if the autopilot is in Airspeed Control, the Z – Acceleration command from these two loops will ultimately command Elevator deflections; however, part of the Altitude Control Loop still affect the Z – Acceleration command through the VRate command. Also note that the VRate command is determined differently when the autopilot is in Airspeed Control.



Airspeed control

TAS err to TAS rate 0.15

TAS rate err to accel cmd 1.5000

TAS rate err to accel der cmd 1.0000

Figure 226 Airspeed Control Loop Gains

The Airspeed Control Loop contains 3 feedback gains.

TAS err to TAS rate: Gain relating true air speed error [m/s] to true air speed rate command [m/s/s]. Increasing this gain will increase the bandwidth with which the autopilot tries to control airspeed. This gain must not be zero (PccUsersGuide 112).

“TAS err to TAS rate” is a longitudinal gain that sets the desired true airspeed rate based upon the true airspeed error. This gain is the airspeed outer loop which simply sets the desired rate of change of true airspeed based upon the true airspeed error. The default value for this gain is 0.15. Increasing this gain will produce more aggressive airspeed tracking and vice versa. If the gain is too high a pitch or oscillation will result (Tuning piccolo control laws 2.0.x 12).

TAS rate err to accel cmd: Gain relating true air speed rate error [m/s/s] to Z acceleration command [m/s/s]. If the vehicle is not changing airspeed at the desired rate this gain causes the flight path trajectory to curve up or down as needed to correct this problem. This gain must not be zero. This gain is only used when the autopilot is in airspeed mode. Most of the time the autopilot is in altitude mode, unless it detects a power problem (PccUsersGuide 113).

“TAS rate err to accel cmd” is a longitudinal gain that computes the vertical acceleration command (or Z – Acceleration command) based upon the airspeed rate error. The theory behind this gain is that if the airspeed is not changing at the rate we want then the vehicles trajectory should be curved – i.e. a vertical acceleration should be commanded. Hence if we are speeding up too quickly command a vertical acceleration which will cause the vehicle to pull up so that gravity can slow us down and vice versa. The default value for this gain is 1.5. Since the acceleration command appears directly on the elevator (via the elevator prediction trust) making this gain too high can cause pitch oscillations. Making this gain too small will impair the ability of the autopilot to regulate airspeed (Tuning piccolo control laws 2.0.x 12).

<u>TAS rate err to accel der cmd:</u> Gain relating true air speed rate error derivative [m/s/s/s] to Z acceleration command [m/s/s]. This gain can be zero but slightly better airspeed control can be achieved if it is not zero. This gain is only used when the autopilot is in airspeed mode. Most of the time the autopilot is in altitude mode, unless it detects a power problem (PccUsersGuide 113).

Note that the Tuning piccolo control laws 2.0.x definitions for “TAS err to TAS Rate” and “TAS Rate err to accel cmd” have been edited due to changes in the Longitudinal Control Logic details that are no longer true have been edited and omitted.

Figure 227 is a schematic of the Airspeed Control Loop based on the gain definitions. Note that this control loop is not always in use. It is only utilized when the autopilot is in Airspeed Control.

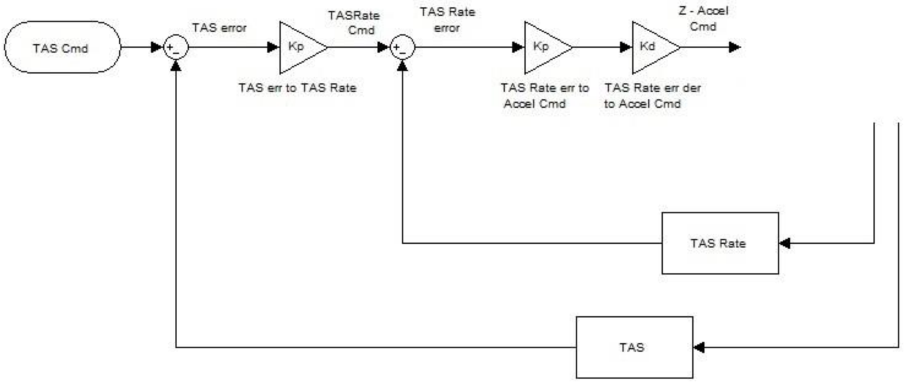
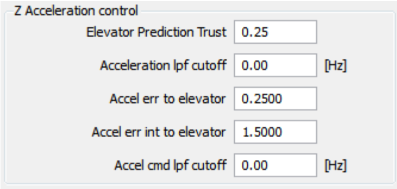


Figure 227 Airspeed Control Loop

5.7.2.4. Z Acceleration Control Loop

The Z – Acceleration Control Loop is the inner loop to either the Altitude or Airspeed Control Loops, depending on what Longitudinal Mode the autopilot is operating in. The Z – Acceleration Control Loop receives the Z – Acceleration command and outputs Elevator deflections.



Z Acceleration control

Elevator Prediction Trust 0.25

Acceleration lpf cutoff 0.00 [Hz]

Accel err to elevator 0.2500

Accel err int to elevator 1.5000

Accel cmd lpf cutoff 0.00 [Hz]

Figure 228 Z Acceleration Control Loop Gains

The Z – Acceleration Control Loop consists of 2 feedback gains, 1 feed forward gain, 2 low pass filters, and 2 Vehicle Parameters. The 2 Vehicle Parameters are the Elevator Effectiveness, and Wing Area. Elevator Effectiveness, combined with the Wing Area, is utilized with the feed forward gain "Elevator Prediction Trust" to act as a feed forward loop. Elevator Effectiveness is also used to scale the feedback gains.

Elevator Prediction Trust: Ratio (0.0 – 1.0) describing how much to trust the elevator prediction from vehicle parameters, from 0.0 (no trust) to 1.0 (full trust). Lower numbers are safer, higher numbers perform better. When using high elevator prediction trust values the Z acceleration error integral to elevator must be strong enough to overcome errors in prediction, otherwise the vehicle could diverge due to miss predicting elevator motion (PccUsersGuide 112).

"Elevator prediction trust" is a longitudinal gain that determines how much of the predicted elevator should actually be used. The autopilot computes the amount of elevator required to achieve a given acceleration based upon the elevator effectiveness, the measured dynamic pressure, the current mass estimate, and the wing area of the vehicle. The elevator effectiveness is like a proportional gain. However sometimes the vehicle plant dynamics require that the proportional gain be smaller than the elevator effectiveness implies. This is due to the time delay that occurs between elevator command and change in acceleration. For those cases the elevator predictive trust can be reduced. The elevator predictive trust can go from 0.0 (no prediction) to 1.0 (full prediction). For most vehicles the best performance occurs when the predictive trust is at 1.0. For vehicles with large time delays in the elevator actuation (high altitude, slow actuators, strange aerodynamics) the predictive trust should be reduced. If fast oscillations in pitch are observed, and the elevator effectiveness and mass estimate are correct, than reduce the predictive trust (Tuning piccolo control laws 2.0.x 8).

Acceleration lpf cutoff: Z acceleration low pass filter cutoff frequency [Hz]. Zero to disable. This filter can be used to remove noise on the z-acceleration measurement. Enabling this filter will reduce the vertical rate control bandwidth (PccUsersGuide 112).

Accel err to elevator: Gain relating the Z acceleration error [[m/s/s]*s] to elevator. When elevator prediction trust is used this gain can be zero. Increase this gain to improve elevator responsiveness and reduce overshoot on acceleration control due to integral wind up (PccUsersGuide 112).

Accel err int to elevator: Gain relating the integral of Z acceleration error [[m/s/s]*s] to elevator. This is the primary gain that moves the elevator and must not be zero. In particular this gain must be strong enough to overcome elevator prediction errors. This gain should be as high as practical in order to maximize the bandwidth of the vertical acceleration and vertical rate control (PccUsersGuide 112).

“Accel err int to elevator” is a longitudinal gain that computes the elevator deflection as the time integral of the acceleration error. The acceleration error integrator is used to find the errors in the elevator prediction. The default value for this gain is 1.5. Increasing this gain will compensate for errors in the elevator predictor and will improve the altitude and airspeed performance. Too much gain will cause a fast pitch oscillation. Vehicles with low elevator bandwidth may need to reduce this gain. If this gain is too low, relative to the predictive trust, the vehicle may enter a speed divergence (Tuning piccolo control laws 2.0.x 8).

Figure 229 is a schematic of the Z – Acceleration Control Loop based on the gain definitions.

Accel cmd lpf: Z acceleration command low pass fileter cutoff frequency (Hz). Zero to disable. Used to reduce the bandwidth the elevator output motion. This is useful for vehicles that have slow elevator actuators (PccUsersGuide 112).

Figure 228 is a schematic of the Z-Acceleration Control Loop based on the gain definitions.

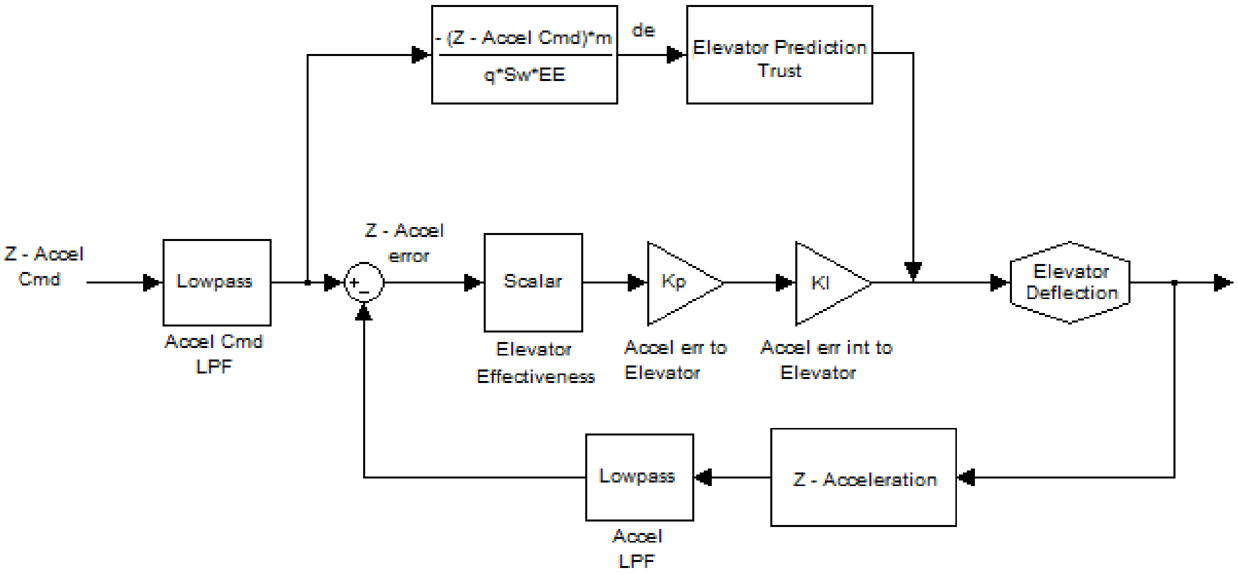
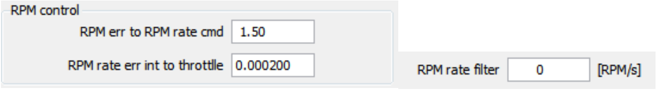


Figure 229 Z Acceleration Control Loop

5.7.2.5. RPM Control Loop

The RPM Control Loop contains an inner and outer loop. The outer loop receives the RPM command and outputs a RPM Rate command. The inner loop receives the RPM Rate command and outputs the Throttle command. Note that the RPM Control Loop only has command authority when there are values for RPM Max and/or RPM Min. RPM Control will completely eliminate Energy Control from Throttle command authority.



RPM control

RPM err to RPM rate cmd 1.50

RPM rate err int to throttle 0.000200

RPM rate filter 0 [RPM/s]

Figure 230 RPM Control Loop Gains

The RPM Control Loop contains 2 feedback gains and 1 filter. For some reason the RPM rate filter is listed under “Limits”; however, it is certainly a filter in the RPM Control Loop.

RPM err to RPM rate cmd: Gain relating RPM error to RPM rate command. If the rpm limiter is enabled this gain controls the bandwidth with which the limiter tries to achieve the RPM command. This gain cannot be zero if the limiter is enabled (PccUsersGuide 113).

RPM rate err int to throttle: Gain relating the integral of RPM rate error to throttle. If the rpm limiter is enabled this gain finds the throttle required to achieve the desired RPM rate. This gain cannot be zero if the limiter is enabled (PccUsersGuide 113).

RPM rate filter: Maximum rate of change of the RPM signal in RPMs per second. When the RPM rate exceeds this value the autopilot assumes the RPM reading is bad and ignores it, using the previous reading. Set to zero to disable this function (PccUsersGuide 114).

Figure 231 is a schematic of RPM Control based on the gain definitions.

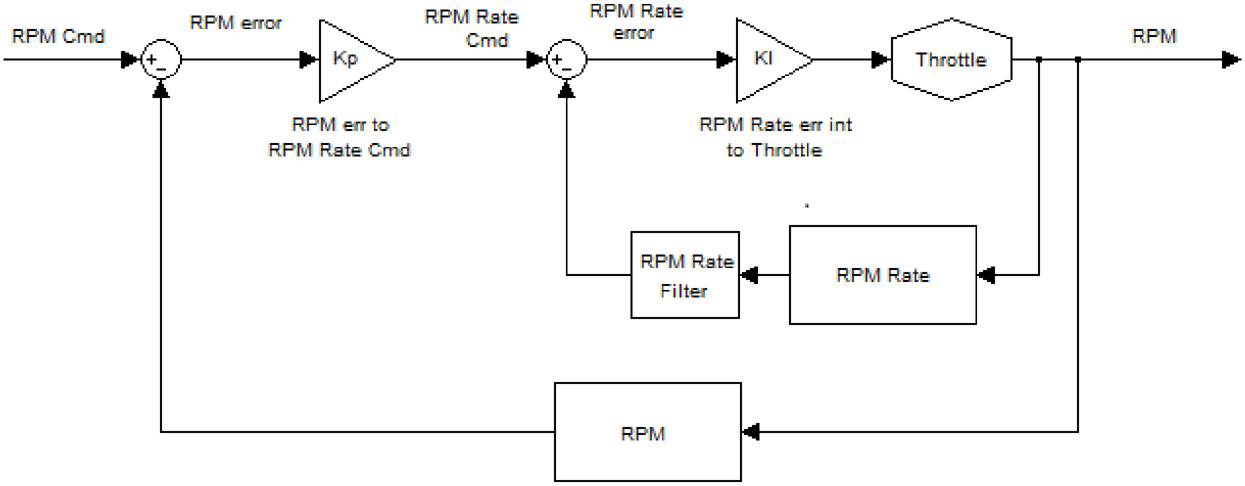


Figure 231 RPM Control Loop

5.8. Control Laws

The control laws are the logic that the autopilot uses to integrate the command loops with their corresponding control loops and limits to command actuator deflections. The following sections detail the control laws for Fixed Wing Generation 2 Version 2.1.4.x.

5.8.1. Aileron Control

In flight Aileron Control integrates the Heading and Bank angle commands with the Track, and Roll Control Loops to produce Aileron and Rudder deflections.

During the Launch and Land modes Lateral Control will command Nosegear and Tailwheel deflections with the Steering Control Loop to control the direction of the aircraft on the ground; however, this guide does not detail the specifics of steering control. There would not be any benefit to numerically solving the steering control loop as the definitions of the steering control loop gains provide as much insight as can be useful to help tune steering control gains.

5.8.1.1. Track Control

Track Control is used during flight to guide an aircraft to the commanded track line via the Track and Roll Control Loops. Track Control commands Heading and Bank Angle through the Track Control Loop, and routes the Bank Angle command into the Roll Control Loop where ultimately Aileron deflections are commanded. The specific goal of Track Control is to zero the Cross Track (y-direction) so that the aircraft will travel along the commanded track line.

The Heading command is calculated from the cross track error, track error in the y-direction, and the Track Control Loop gain “Tracker Convergence”. As was stated in the definitions of Track Control Loop gains the Tracker Convergence plots an elliptical trajectory from the aircraft’s current position to the target path.

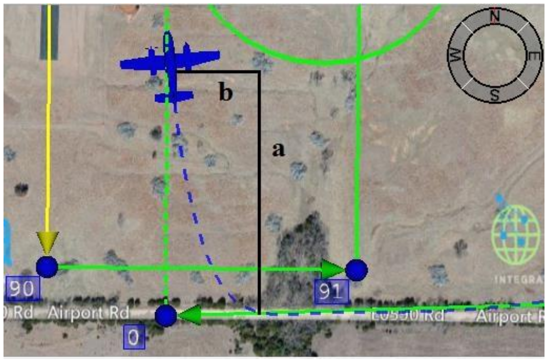


Figure 232 Track Control Elliptical Trajectory

Figure 232 illustrates an aircraft flying an elliptical trajectory to converge with a new track line. Track Control calculates the elliptical trajectory from the following equations.

$$b = (TAS)^2 * TC$$

$$where\ TC = Tracker\ Convergence$$

Equation 26 Semi Minor Axis

$$a = 4 * b$$

Equation 27 Semi Major Axis

$$\frac{CrossTrackError^2}{b^2} + \frac{y^2}{a^2} = 1$$

Equation 28 Elliptical Trajectory

$$Heading\ Cmd = \overrightarrow{(CrossTrackError, y)}$$

Equation 29 Heading Command

The tracker convergence parameter essentially determines when the autopilot decides to begin the pre turn to the new track path. As soon as the distance (y – direction) between the aircraft and the next flight path is equal to the length of the semi major axis the autopilot will target the next flight path. Presumably the autopilot constantly re calculates the elliptical trajectories as errors in actual heading and changes in true airspeed will alter the course of the aircraft and change the calculations of the semi major and semi minor axes.

Track Control routes the heading command into the Track Control Loop. In the Track Control Loop the heading error is converted into a turn rate command. Heading feedback is not exactly equal to the heading recorded by the Piccolo Development Interface. The Piccolo Development Interface records the true heading in the direction that the aircraft is traveling. The heading feedback term is the direction that the aircraft is pointing. During situations where there is a crosswind the autopilot will crab the aircraft and the direction that the aircraft is pointing will be different than the true heading that the aircraft is traveling as recorded by the Piccolo Development Interface. The piccolo telemetry data records direction and it is called "Direction". If there is a magnetometer the autopilot will use the magnetometer to determine direction. If there is not a magnetometer the autopilot will calculate the direction of the aircraft using the velocity vectors of the aircraft. The piccolo telemetry records the velocity vectors as “VNorth”, “VEast”, and “VDown” where the vectors are defined as acting in the corresponding cardinal directions (north, south, east, west). The autopilot will calculate the resultant vector, which is actually the same number that the autopilot interprets as “Groundspeed”.

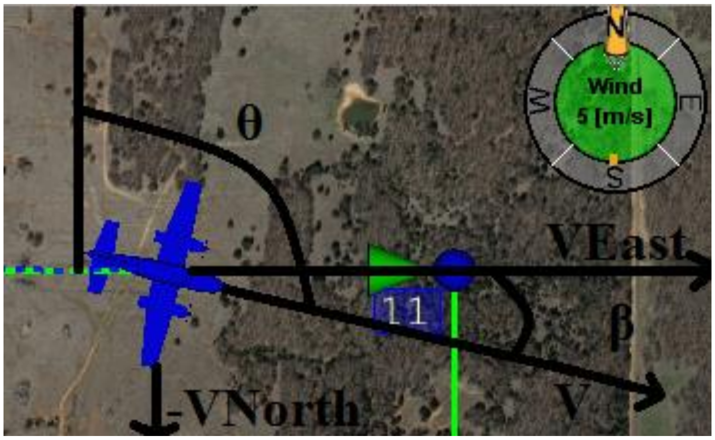


Figure 233 Direction

The direction is calculated from the angle between the groundspeed velocity vector and the velocity vector that corresponds to the direction that the target flight path is on. In Figure 233 above an aircraft is traveling on an easterly flight path. The angle θ represents the direction. The angle β is the angle between the groundspeed velocity vector and the east velocity vector or VEast and is essentially the crab angle.

$$\vec{V}_{GroundSpeed} = \sqrt{VEast^2 + VNorth^2 + VDown^2} \;\Rightarrow\; \beta = \cos^{-1}\left[\frac{VEast}{\vec{V}_{GroundSpeed}}\right]$$

Equation 30 Crab Angle

$$Direction = \theta = \frac{\pi}{2} + \beta$$

Equation 31 Direction

Equation 30 and Equation 31 above show the calculations that would occur in the scenario shown in Figure 233. The final direction calculation is used as heading for the feedback of the heading control loop; hence, the heading error is essentially equal to the heading command minus the direction.

The turn rate command that results depends on the values of the gains in the Track Control loop and the heading error. The gains are multiplied to the heading error and the result is the turn rate command.

$$\omega_{cmd} = K_{p1} * Heading_{err} + K_D * \frac{dHeading_{err}}{dt}$$

Equation 32 Turn Rate Command

$$where\ \omega = turn\ rate\ \left(\frac{rad}{s}\right)\ \ K_{p1} = Heading\ err\ to\ turn\ rate$$

$$K_D = Heading\ err\ der\ to\ turn\ rate$$

Track Control uses the turn rate command to calculate the Bank Angle Command. The Bank Angle command is backed out of the turn rate command by re arranging the turn rate equation, Equation 33.

$$\omega = \frac{g\tan(\varphi)}{V_\infty}$$

Equation 33 Turn Rate

$$\Rightarrow\ \ \varphi_{cmd} = \tan^{-1}\left[\frac{\omega_{cmd} * TAS}{g}\right]$$

Equation 34 Bank Angle Command

$$where\ g = 9.81\ m/s^2$$

$$TAS = true\ airspeed\ (m/s)\ \ \varphi_{cmd} = Bank\ Command\ (rad)$$

The turn rate command is not recorded or displayed by either the DevInterface or PCC. The bank angle command is interpreted by the autopilot as the roll command. They are both exactly the same numbers. The bank angle command is displayed by PCC and the roll command is displayed by the DevInterface; however, beware relying on the bank angle command recorded by PCC, the piccolo telemetry command loop data is downloaded at a slower rate than other telemetry data and always lag behind when the controller actually changes the commands.

Note that if the Tracker Convergence parameter is too low the elliptical trajectory plots can be too small and can cause the aircraft to have short quick oscillations that don't dampen out. On that same note if the Tracker Convergence is too high the elliptical trajectory plots can be too large and cause the aircraft to not ever converge with the targeted track line. In some cases the Tracker Convergence could be larger than the actual length of the track line and cause the aircraft to skip an entire leg of the flight plan altogether. The following figures depict the extremes that can result when Tracker Convergence is too high or too low.

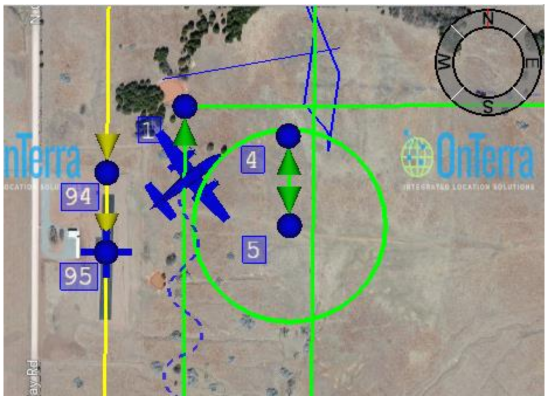


Figure 234 Tracker Convergence Too Low

Figure 234 depicts a scenario where Tracker Convergence is too low. The elliptical trajectory was too small and caused the aircraft to oscillate in and out of the track line continuously.

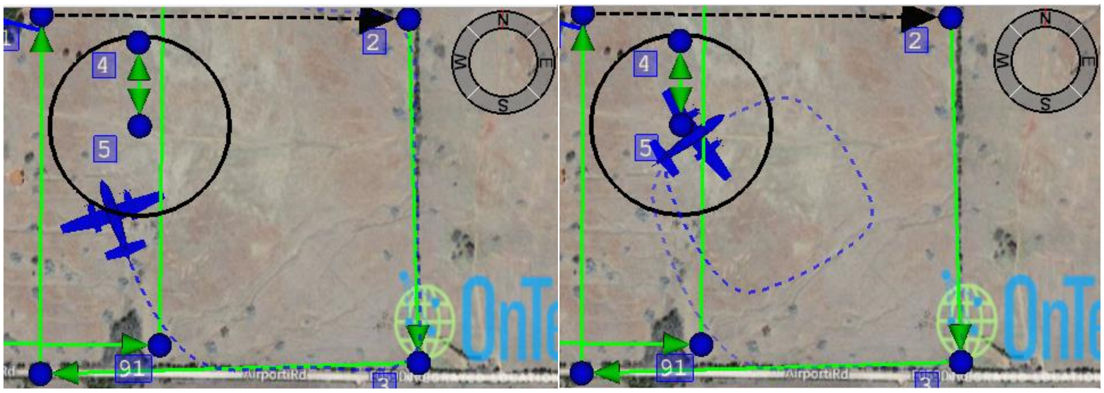


Figure 235 Tracker Convergence Too High

Figure 235 above depicts a scenario where Tracker Convergence is too high. The image on the left shows the autopilot targeting waypoint 2 far before reaching waypoint 1. The image on the right shows the same scenario played out for one whole lap of the box flight plan. The aircraft was never able to get close to converging with any of the track lines on the flight pattern.

5.8.1.2. Roll Control

Roll Control uses the roll control loop combined with the roll commands produced by track control to command aileron deflections. The roll command is routed through the roll control outer loop. In the roll control outer loop the roll error is routed through the outer loop proportional feedback gain to produce the roll rate command.

$$Roll\ Rate\ Cmd = K_{p2} * Roll_{err}\ \ (rad/s)$$

Equation 35 Roll Rate Command

$$where\ K_{p2} = Roll\ rate\ err\ to\ roll\ rate\ cmd$$

Equation 35 is used to calculate the roll rate command from the roll error. The roll rate command is routed to the inner loop of roll control. The inner loop contains two pathways to command aileron deflections. One pathway acts as a feed forward process while the other is a feedback process. The feed forward process utilizes the aileron effectiveness and wingspan vehicle parameters. With these parameters the roll rate command is used to calculate aileron deflections.

$$\delta_{a1} = \frac{p_{cmd} * b}{2 * TAS * AE}$$

Equation 36 Aileron Deflection Command Feed Forward Loop

$$where\ p_{cmd} = Roll\ Rate\ Cmd\ (rad/s)\ b = wingspan\ (m)\ TAS = true\ airspeed\ \left(\frac{m}{s}\right)$$

$$AE = aileron\ effectiveness\ (/rad)$$

Equation 36 is used to calculate the feed forward aileron deflection command, δ_{a1}. The feedback loop combines the roll rate error with the roll rate proportional gain, integral gain, and a scaling term to calculate aileron deflections. The scaling term consists of the aileron effectiveness and wingspan vehicle parameters.

$$\delta_{a2} = \frac{\left[p_{error} * K_{p3} + K_I \int p_{error} dt\right] * b}{2 * TAS * AE}$$

Equation 37 Aileron Deflection Command Feedback Loop

$$where\ p_{error} = Roll\ Rate\ error\ (rad/s)\ \ K_{p3} = roll\ rate\ err\ to\ aileron$$

$$K_I = roll\ rate\ err\ int\ to\ aileron$$

Equation 37 is used to calculate the aileron deflection, δ_{a2}. Roll control adds the two aileron deflection commands to issue the final aileron deflection command. Combining the two equations leads to the following equation, Equation 38, for calculating aileron deflections.

$$\delta_a = \frac{\left[p_{cmd} + \left(p_{error} * K_{p3} + K_I \int p_{error} dt\right)\right] * b}{2 * TAS * AE}$$

Equation 38 Aileron Deflection Command

5.8.2. Rudder Control

Rudder Control uses the yaw control loop and side force control loop to command rudder deflections. The yaw control loop is utilized as a yaw damper. There is a third component that routes the yaw rate command from the yaw control loop and converts the yaw rate command into rudder deflections. The third component is meant to provide turn coordination via the estimated sideslip angle.

$$r_{cmd} = \frac{-\Delta\beta * TAS}{l_v}$$

Equation 39 Yaw Rate Command for Sideslip Angle Turn Coordination

$$where\ r_{cmd} = yaw\ rate\ cmd \left(\frac{rad}{s}\right)\ \beta = sideslip\ angle\ (rad)\ \ l_v = vertical\ tail\ arm\ (m)$$

The yaw control loop receives input commands from the yaw rate command. If turn coordination is enabled the yaw rate command is calculated by Equation 39 where it will attempt to zero the sideslip angle. Turn coordination is enabled when the vertical tail parameter value is greater than zero. The yaw rate command is routed through two different routes. The first route combines the yaw rate command with the vertical tail arm and rudder effectiveness vehicle parameters to calculate rudder deflections. This route essentially attempts to coordinate the turn with the vehicle parameters and can be considered a feed forward gain.

$$\Delta\delta_{r1} = \frac{r_{cmd}}{RE} * \frac{l_v}{TAS}$$

Equation 40 Rudder Deflection Command for Sideslip Angle Turn Coordination

$$where\ RE = Rudder\ Effectiveness = {}^{\Delta\beta}/_{\Delta\delta_r}$$

Equation 40 is the equation that yaw control uses to calculate rudder deflections for turn coordination if the side force control loop is disabled. The second path routes the yaw rate command through the yaw control loop which essentially acts as a yaw damper. The yaw rate command remains the same so it is attempting to coordinate the turn (zero sideslip); however, the control loop acts as a damper because it commands rudder deflections in response to the yaw rate errors.

$$\Delta\delta_{r2} = \frac{(r_{error} * K_p) * I_z}{RP * qSb}$$

Equation 41 Rudder Deflection Command from Yaw Damper

$$where\ K_p = yaw\ rate\ err\ to\ rudder\ \ I_z = z - inertia\ \ q = dynamic\ pressure\ \left(\frac{N}{m^2}\right)$$

$$S = wing\ area\ (m^2)\ b = wingspan\ (m)\ RP = rudder\ power = {}^{\Delta C_n}/_{\Delta\delta_r}$$

The rudder deflections from the yaw rate control loop are calculated with Equation 41 above. The calculation scales the yaw rate error with a scaling term that includes the z – inertia, rudder power, wing area, and wing span vehicle parameters. The measured dynamic pressure is also used in the scaling term. If turn coordination is enabled the two rudder deflection commands

(δ_{r1}, δ_{r2}) are simply added together to command rudder deflection. If the side force control loop is enabled then δ_{r1} surrenders rudder command authority to δ_{r3}.

$$\Delta\delta_{r3} = \frac{K_I \int Y_{error} dt}{(SE)(RE)} * \frac{m}{qS}$$

Equation 42 Rudder Deflection Command from Side Force Control Loop

$$where\ K_I = side\ force\ error\ int\ to\ rudder\ \ m = mass\ estimate\ (kg)$$

$$Y_{error} = y - acceleration\ (m/s^2)\ \ SE = side\ slip\ effectiveness = {}^{\Delta C_y}\!/_{\Delta\beta}\ \ (/rad)$$

δ_{r3} is the rudder deflection command that is commanded by the side force control loop. The side force control loop calculates rudder deflection commands from Equation 42 above. The side force control loop attempts to coordinate the turn by commanding rudder deflections to zero the measured side force or y acceleration. The side force control loop always commands 0 side force; hence, the side force error is exactly equal to the side force that is measured. The side force control loop combines the side force error feedback with a scaling term that includes the side slip effectiveness, rudder effectiveness, and wing area vehicle parameters. The scaling term also uses the mass estimate and measured dynamic pressure. The side force control loop is enabled by assigning a value greater than 0 to the side force control integral gain. The yaw damper is still utilized when the side force control loop is used to coordinate turns. In such a scenario the δ_{r3} and δ_{r2} commands are added together to command rudder deflections.

Turn coordination can be disabled by setting the vertical tail arm vehicle parameter equal to 0. If turn coordination is disabled yaw control will no longer attempt to zero side slip and instead it will simply attempt to provide yaw damping while a turn is being performed. In such a scenario only δ_{r2} will have rudder deflection command authority. Additionally the yaw rate command calculation will be changed to command the measured turn rate of the vehicle. Recall from track control that track control issues turn rate commands based on the heading errors in the track control loop which in turn used to calculate roll commands. In this case the yaw rate command is calculated from the actual turn rate of the aircraft; hence, it is calculated from the roll angle of the aircraft.

$$r_{cmd} = \omega = \frac{g\tan(Roll)}{V}$$

Equation 43 Yaw Rate Command for No Turn Coordination

$$where\ \omega = turn\ rate\ \left(\frac{rad}{s}\right)\ V = velocity\ in\ direction\ of\ target\ fligth\ path$$

Equation 43 is how the yaw rate command is calculated when turn coordination is disabled. The velocity variable changes dependent on the direction of the target path. If the target path travels east the velocity variable that is used will be “VEast” which is the east west velocity vector. Similarly if the target path travels north the velocity variable used will be “VNorth”. **Error! Reference source not found.** below depicts yaw control in its entirety.

5.8.3. Elevator Control

Elevator Control routes the Altitude Command to the Altitude, Airspeed, and Z – Acceleration Control Loops to ultimately command elevator deflections. The altitude and airspeed control loops both issue z – acceleration commands to the z – acceleration control loop, which then calculates and commands elevator deflections; however, the altitude and airspeed control loops do not simultaneously possess z – acceleration command authority. The longitudinal mode of the autopilot dictates which control loop has authority to issue z – acceleration commands to the z – acceleration control loop. Longitudinal Modes are covered later in Section 5.8.5 Longitudinal Modes (Lon Modes).

5.8.3.1. Altitude Control

In altitude control the autopilot uses the elevator to control altitude. More specifically altitude control commands elevator deflections through the altitude, vrate, and z – acceleration control loops. Altitude control possess elevator command authority when the autopilot is in Lon Mode 0 (Altitude Mode). When altitude control has elevator command authority the vertical rate command calculated from the altitude control loop will be the vertical rate command that the autopilot routes to the vertical rate control loop. Additionally altitude control can issue commands to energy control and effect the throttle commands depending on the longitudinal mode of the autopilot. In Lon Mode 0 the vertical rate command from altitude control is added to the energy rate command. If the autopilot transitions into any of the airspeed modes (Lon Modes 1-3) the altitude command will have sole authority to issue energy commands to the energy control outer loop. Section 5.8.4.1 Energy Control covers energy control in greater detail.

Altitude control routes altitude commands into the altitude control loop. The altitude control loop proportional gain “Alt err to Alt Rate” (Kpa) is multiplied by the altitude feedback error and calculated into a vertical rate command as shown below in Equation 44 .

$$VRate_{cmd} = K_{pa} * Alt_{err}$$

Equation 44 Vertical Rate Command

It is important to recall that the altitude command is dictated by the flight plan and target waypoint. The altitude command could be a step input or ramp input command. The vertical rate command is limited by the climb and descent max fractions with the equations below.

$$VRate_{max} = Climb\ Max\ Fraction * TAS$$

Equation 45 Vertical Rate Max

$$VRate_{min} = -Descent\ Max\ Fraction * TAS$$

Equation 46 Vertical Rate Min

The vertical rate command is routed through the altitude control inner loop where the vertical rate error is multiplied by the inner loop proportional gain "Alt Rate err to Z-Accel Cmd". The vertical rate error feedback is multiplied by the proportional gain and is calculated into a z – acceleration command. There are other components that contribute to the calculation of the z – acceleration command. During testing the actual z – acceleration command was never able to be predicted. It is likely that the elevator prediction trust is used to alter the z – acceleration commands based on predictions; however, this was unable to be numerically duplicated. It is likely that the vertical rate control loop portion of the z – acceleration command is calculated with Equation 47 below.

$$Z\ Accel\ Cmd = K_{pv} * VRate_{err}$$

Equation 47 Z - Acceleration Command from VRate Control Loop

$$where\ K_{pv} = Alt\ Rate\ err\ to\ Z\ Accel\ Cmd$$

The z – acceleration command is limited by two limits and three different CL parameters. The load factor max and load factor min set maximum and minimum z – acceleration command limits and are designed to represent the structural load limits of the aircraft. Equation 48 and Equation 49 below represent the structural load limits.

$$Z\ Accel\ Cmd_{max1} = Load\ Factor\ Max * -9.81$$

Equation 48 Z - Acceleration Command Max 1

$$Z\ Accel\ Cmd_{min1} = Load\ Factor\ Min * -9.81$$

Equation 49 Z - Acceleration Command Min 1

Both the CL max and CL max nom vehicle gains are also used to calculate a maximum z – acceleration command. CL max nom is used during normal flight, while CL max is used when the autopilot is navigating land plans. The vehicle gain CL min is used to calculate a minimum z – acceleration command. Additionally the mass estimate, wing area vehicle gain, and measured dynamic pressure are used to calculate both minimum and maximum CL based z – acceleration command limits. Equation 50 and Equation 51 below represent the CL based z – acceleration command limits.

$$Z\ Accel\ Cmd_{max2} = \frac{-CL_{max} * q * S_w}{m}$$

Equation 50 Z - Acceleration Command Max 2

$$Z\ Accel\ Cmd_{min2} = \frac{-CL_{min} * q * S_w}{m}$$

Equation 51 Z - Acceleration Command Min 2

$$where\ q = measured\ dynamic\ pressure\ (Pa)\ \ S_w = wing\ area\ (m^2)$$
$$m = [Gross\ Mass - Estimated\ Fuel\ Mass]\ (kg)$$

The z – acceleration command is routed to the z – acceleration control loop. The z – acceleration command can be filtered by the “Accel cmd lowpass filter”. As stated by the filter’s definition it is designed to be used to filter changes in z – acceleration command for systems with actuator lag (slow actuators). The z – acceleration control loop calculates elevator deflection commands through two different paths. One path acts as a feed forward loop while the other acts as a feedback loop. The feed forward loop calculates elevator deflections from the z acceleration command, measured dynamic pressure, mass estimate, elevator prediction trust, and the elevator effectiveness, wing area, and CL at zero elevator vehicle gains. The feed forward loop first calculates a CL command, and then an elevator deflection command as shown below in Equation 52 and Equation 53.

$$C_{L_{feedforward}} = \frac{-Z\ Accel_{cmd} * m}{q * S_w}$$

Equation 52 CL Command from Feed Forward Loop

$$\delta_{e1} = EPT * \left[\frac{C_{L_{feedforward}} - C_{L_{\delta_{e0}}}}{EE}\right]$$

Equation 53 Elevator Deflection Command from Feed Forward Loop

$$where\ EE = Elevator\ Effectiveness = {}^{\Delta C_L}/_{\Delta\delta_e}\ (/rad)$$

$$C_{L_{\delta_{e0}}} = CL\ at\ zero\ elevator \quad EPT = Elevator\ Prediction\ Trust$$

The z acceleration command is used to calculate the C_L command from the feed forward loop. The C_L at zero elevator vehicle parameter is subtracted from $C_{Lfeedforward}$ to determine the desired change in C_L. The desired change in C_L is divided by the elevator effectiveness and multiplied by the elevator prediction trust. The feedback loop routes the z acceleration error through the z acceleration control loop proportional gain and integral gain. The z acceleration error is then calculated into an elevator deflection with almost the same process as the feed forward loop. The feedback loop first calculates a CL command, and then an elevator deflection command as shown below in Equation 54 and Equation 55.

$$C_{L_{feedback}} = \frac{\left[K_{pz} * Zerror + K_I * \int Zerror\ dt\right] * m}{q * S_w}$$

Equation 54 CL Command from Feedback Loop

$$\delta_{e2} = \left[\frac{C_{L_{feedback}} - C_{L_{\delta_{e0}}}}{EE}\right]$$

Equation 55 Elevator Deflection Command from Feedback Loop

$$\delta_e = \delta_{e1} + \delta_{e2}$$

Equation 56 Elevator Deflection Command

$$where \quad K_{pz} = Accel\ err\ to\ elevator \quad K_I = Accel\ err\ int\ to\ elevator$$

$$Zerror = Z_{cmd} - Z_{accel}$$

The two calculated elevator deflections are added together to issue the final elevator deflection command as shown in Equation 56 above.

5.8.3.2. Airspeed Control

In airspeed control the autopilot uses the elevator to control airspeed. If the airspeed is above the commanded airspeed the autopilot will command elevator up to pitch the aircraft up in order to decrease airspeed. If the airspeed is below the commanded airspeed the autopilot will command elevator down to pitch the aircraft down in order to increase airspeed. It is important to note that when the autopilot is in airspeed control altitude control reliability will be reduced as altitude will be controlled with the throttle via energy control.

When the autopilot is in altitude control (Lon Mode 0) the autopilot is designed to maintain airspeed with the throttle via energy control.

Airspeed control corresponds to Longitudinal Modes 1-3.

Airspeed control possess elevator command authority when the autopilot is in Longitudinal Modes 1-3 (Airspeed Mode, Slow Airspeed Mode, Fast Airspeed Mode). When airspeed control has elevator command authority the vertical rate command is calculated from the airspeed control inner loop, TAS rate, and will be the vertical rate command that the autopilot routes to the vertical rate control loop. The differences between Lon Modes 1-3 has more to do with throttle control and less to do with airspeed control specifically. In all 3 Lon Modes the elevator control portion of airspeed control does not change. Section 5.8.5 Longitudinal Modes (Lon Modes) covers Lon Modes in detail. Note that in airspeed control airspeed does not issue any commands to energy control; thus, airspeed will not directly affect throttle. If the autopilot transitions into altitude control mode (Lon Mode 0) the airspeed command will have sole authority to issue energy commands to the energy control outer loop in which case airspeed will directly affect throttle. Section 5.8.4.1 Energy Control covers energy control in greater detail.

Airspeed control converts the commanded indicated airspeed into a true airspeed command with the Equation 23 and Equation 24 discussed in Section 5.1.2 True Airspeed. The true airspeed command is routed to the airspeed control outer loop. The airspeed control outer loop commands a true airspeed rate, “TASRate”, to the airspeed control inner loop based on the true airspeed error and the outer loop proportional gain “TAS err to TAS Rate”.

$$TASRate_{cmd} = K_{p1} * TASerror$$

Equation 57 TASRate Command

$$where\ K_{p1} = TAS\ err\ to\ TAS\ Rate$$

The inner loop calculates the VRate command based off of the true airspeed rate error, the inner loop proportional gain “TASRate err to Accel Cmd”, and the inner loop derivative gain “TASRate err Der to Accel Cmd”.

$$VRate_{cmd} = K_{p2} * TASRate_{error} + K_D \frac{dTASRate_{error}}{dt}$$

Equation 58 VRate Command Airspeed Control

$$where\ K_{p2} = TASRate\ err\ to\ Accel\ Cmd \quad K_D = TAS\ Rate\ err\ Der\ to\ Accel\ Cmd$$

Similar to altitude control the actual z – acceleration command was never able to be predicted and it is likely that the elevator prediction trust is used to alter the z – acceleration commands based on predictions; thus, it is likely that the z – acceleration command from the vertical rate control loop is calculated with Equation 47 from Section 5.8.3.1.

The Z – Acceleration Control Loop transforms the z – acceleration commands into elevator deflection commands directly through the feed forward gain Elevator Prediction Trust, and via the z – acceleration error through the feedback loop as described earlier in Section 5.8.3.1.

5.8.4. Throttle Control

Throttle control determines an energy command and routes the energy command through the Energy Control Loop to command throttle. The Longitudinal Mode of the autopilot dictates how throttle control calculates the energy, energy rate, and throttle commands. Additionally the Longitudinal Mode dictates the measured feedback term that's used in the outer energy control loop. If the autopilot is in Lon Mode 0 (Altitude Mode) throttle control will use the indicated airspeed command to calculate an energy command and the measured indicated airspeed will be used to calculate the energy feedback term. Additionally the VRate command will be used to add to the energy rate command. If the autopilot is in Lon Mode 1 (Airspeed Mode) throttle control will use the altitude command to calculate an energy command and the measured altitude will be used to calculate the energy feedback term. In Lon Mode 2 (Slow Airspeed Mode) the throttle is held fixed at Throttle Max, and in Lon Mode 3 (Fast Airspeed Mode) the throttle is held fixed at Throttle Min.

The basic idea is that in altitude control throttle is designed to control airspeed, and in airspeed control the throttle is designed to control altitude. In airspeed control throttle almost only indirectly controls altitude. If there is a negative altitude error (aircraft below commanded altitude) the throttle will increase and cause the aircraft's airspeed to increase above the commanded airspeed. As a result the autopilot will pitch the aircraft up to decrease airspeed; thus, the aircraft will climb. Similarly, if there is a positive altitude error the throttle will decrease and cause the aircraft's airspeed to decrease below the commanded airspeed. As a result the autopilot will pitch the aircraft down to increase airspeed; thus, the aircraft will descend.

5.8.4.1. Energy Control

Energy Control commands throttle to maintain either altitude or airspeed via the Energy Control Loops. The outer loop calculates an energy rate command. The energy rate command is routed through the inner energy control loop which calculates throttle commands.

In both altitude and airspeed control the outer energy control loop will command an energy rate based on the energy error. The outer loop determines energy error based on the energy command and energy feedback where energy is either the kinetic or potential energy terms of the energy equation shown below in Equation 59.

$$Energy\ (J) = Kinetic + Potential = \frac{1}{2}m(TAS)^2 + mg(Alt)$$

$$where\ m = mass\ estimate\ (kg)\quad g = 9.81\ m/s^2$$

$$TAS = true\ airspeed\ (m/s)\quad Alt\ = altitude\ (m)$$

Equation 59 Energy

Since the energy equation includes mass energy control is also dependent on the vehicle mass estimate. If the autopilot is in altitude control the energy command and feedback will be based only on airspeed via the kinetic energy portion of the energy equation. This applies to the outer energy control loop and is calculated as shown in Equation 60.

$$Energy\ Cmd\ _{LM0}(J) = \frac{1}{2} * m(TAS_{cmd})^2$$

Equation 60 Energy Command in Altitude Control

If the autopilot is in airspeed control the energy command and feedback will be based only on altitude via the potential energy portion of the energy equation. This applies to the outer energy control loop and is calculated as shown below in Equation 61.

$$Energy\ Cmd\ _{LM1}(J) = mg(Alt_{cmd})$$

Equation 61 Energy Command in Airspeed Control

The outer loop calculates an energy rate command from the energy error and outer loop proportional gain "Energy err to Energy Rate Cmd".

$$ERate\ Cmd_1(W) = K_{pe} * Energy_{err}$$

Equation 62 Energy Rate Command from Energy Outer Loop

$$where\ K_{pe} = Energy\ err\ to\ Energy\ Rate\ Cmd$$

Additionally in altitude control, throttle control will port an extra energy rate command term from the VRate command issued by Altitude Control. Throttle control calculates an energy rate command by inserting the VRate command into the potential energy portion of the energy rate equation.

$$Energy\ Rate\ Cmd_2\ (W) = mg(VRate_{cmd})$$

Equation 63 Energy Rate Command from VRate Command

$$with\ VRate\ in\ (m/s)$$

$$Energy\ Rate\ Cmd\ (W) = Energy\ Rate\ Cmd_1 + Energy\ Rate\ Cmd_2$$

Equation 64 Energy Rate Command

The final energy rate command is the addition of Equation 62 and Equation 63. In airspeed control Energy Rate Cmd$_2$ will be 0. The inner loop of energy control transforms energy rate error and commands into throttle commands. The inner loop is not directly dependent on whether or not the autopilot is in altitude or airspeed control. The measured energy rate feedback term is calculated with the measured vertical rate via the potential energy equation, shown in Equation 65 below, regardless of altitude or airspeed control.

$$Energy\ Rate\ (W) = mg(VRate)$$

Equation 65 Energy Rate

The inner loop of energy control contains both a feedback and a feed forward loop. The feed forward loop calculates throttle commands from the energy rate command, the max engine power vehicle gain, and the energy control feed forward gain throttle prediction trust. Equation 66 below represents the feed forward throttle command calculation.

$$Throttle_1 = \frac{Energy\ Rate\ Cmd * TPT}{MEP}$$

Equation 66 Throttle Command from Feed Forward Loop

$$where\ TPT = Throttle\ Prediction\ Trust \quad MEP = Max\ Engine\ Power\ (W)$$

The feedback loop issues throttle commands from the energy rate error, inner loop proportional gain “Energy Rate err to Throttle”, inner loop integral feedback gain “Energy Rate err Int to Throttle”, and the vehicle gain “Max Engine Power”. Equation 67 below represents the feedback throttle command calculation.

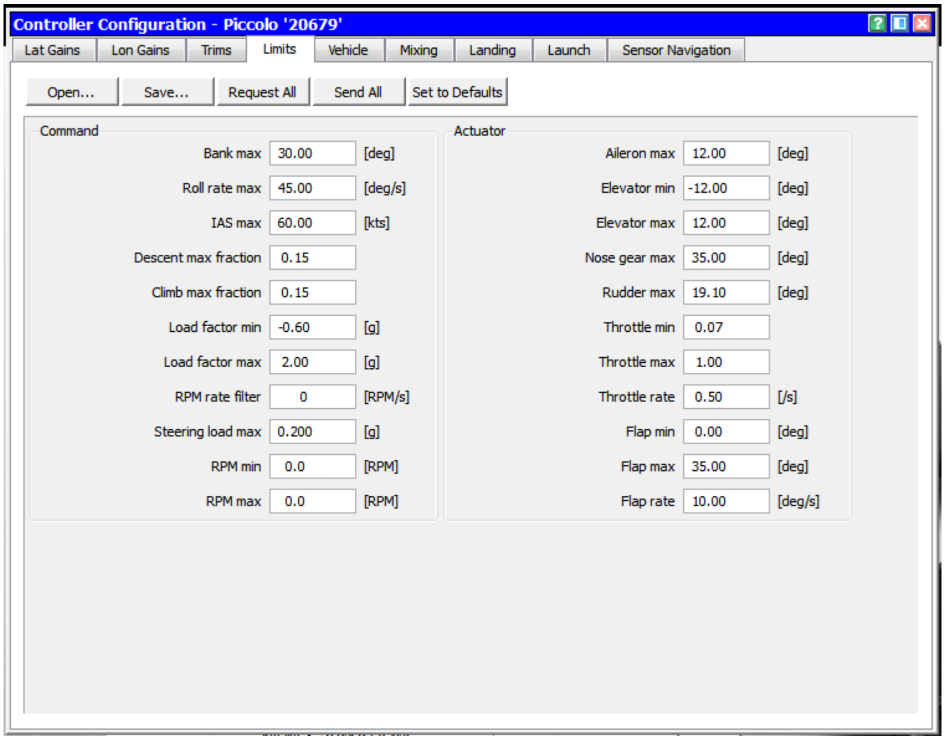
$$Throttle_2 = \frac{K_{pt} * EnergyRate_{err} + KI * \int EnergyRate_{err}\, dt}{MEP}$$

Equation 67 Throttle Command from Feedback Loop

$$where\ Kpt = Energy\ Rate\ err\ to\ Throttle \quad KI = Energy\ Rate\ err\ Int\ to\ Throttle$$

5.8.4.2. RPM Control

RPM Control allows the user the option to set rpm limits and in some cases force the autopilot to operate in a specified range of rpms. In RPM Control the RPM Control Loops command throttle based off of the IAS error; thus, RPM Control will attempt to maintain indicated airspeed within the limitations set by “RPM Max” and/or “RPM Min”.



Command

Bank max	30.00	[deg]
Roll rate max	45.00	[deg/s]
IAS max	60.00	[kts]
Descent max fraction	0.15	
Climb max fraction	0.15	
Load factor min	-0.60	[g]
Load factor max	2.00	[g]
RPM rate filter	0	[RPM/s]
Steering load max	0.200	[g]
RPM min	0.0	[RPM]
RPM max	0.0	[RPM]

Actuator

Aileron max	12.00	[deg]
Elevator min	-12.00	[deg]
Elevator max	12.00	[deg]
Nose gear max	35.00	[deg]
Rudder max	19.10	[deg]
Throttle min	0.07	
Throttle max	1.00	
Throttle rate	0.50	[/s]
Flap min	0.00	[deg]
Flap max	35.00	[deg]
Flap rate	10.00	[deg/s]

Figure 236 RPM Control Limits

RPM Control can be enabled by entering in any positive number into either the "RPM min" or "RPM max" input boxes located in the "Limits" tab of the Controller Configuration window in PCC as shown above in Figure 236.

RPM Control is dependent on the aircraft's indicated airspeed, indicated airspeed command, indicated airspeed limits, rpm, and Energy Control throttle commands. RPM Control logic is also dependent on whether or not one or both rpm limits exist. In some scenarios RPM Control will supersede Energy Control and possess throttle command authority. In some cases Energy Control will maintain throttle command authority even with the existence of an rpm limit. The different scenarios are detailed below.

RPM Max Only:

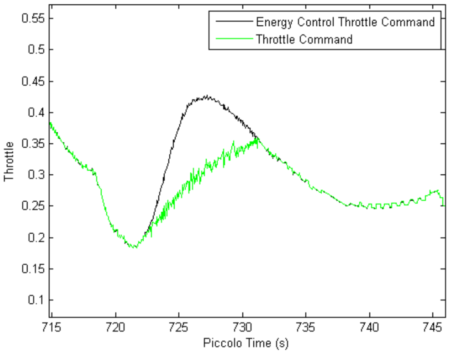


Figure 237 RPM Control Throttle Command RPM Max

Energy Control will lose throttle command authority if the Energy Control throttle command is greater than the RPM Control throttle command as shown in Figure 237. If the commanded airspeed cannot be reached without violating RPM Max the autopilot will simply command throttle to hold the rpm at RPM Max.

RPM Min Only:

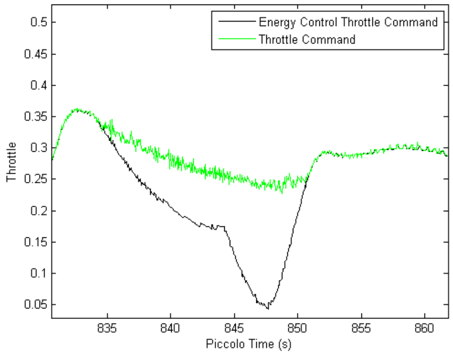


Figure 238 RPM Control Throttle Command RPM Min

Energy Control will lose throttle command authority when the Energy Control throttle command is less than the RPM Control throttle command. If the commanded airspeed cannot be reached without violating RPM Min the autopilot will command throttle to hold rpm at RPM Min.

RPM Min and RPM Max:

Energy Control will lose throttle command authority when either the Energy Control throttle command is greater than or lower than the RPM Control throttle command. Energy Control can control throttle in this scenario; however, it's mostly a result of the Energy Control throttle commands being near or the same as the RPM Control throttle commands.

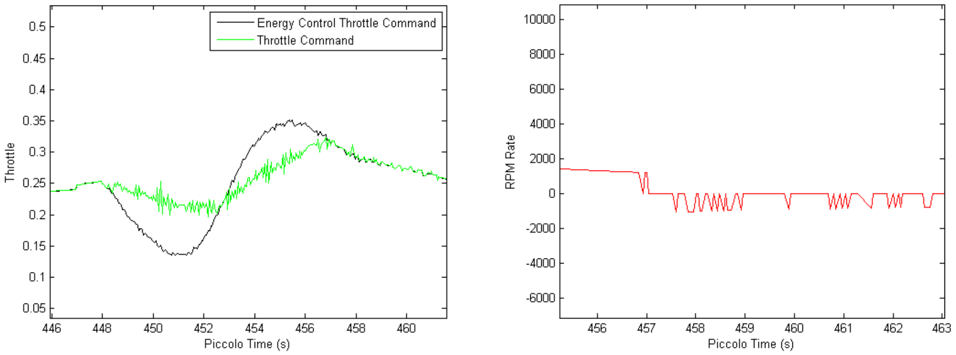


Figure 239 RPM Control Throttle Command RPM Min & Max

In Figure 239 above the time period from about 457 to 463 seconds depicts throttle command authority alternating between Energy and RPM Control throttle commands in a simulation. RPM Control mainly commands the throttle commands and every now and then Energy Control receives command authority. The RPM Rate graph in Figure 239 depicts the rpm rate command as recorded by the controller telemetry. When Energy Control possess throttle command authority the rpm rate command is zero.

The user should be aware that operating in RPM Control can be dangerous. The longitudinal control system is designed to operate with the throttle and elevator both being able to control, and influence, altitude and airspeed. It is possible for the autopilot to enter Airspeed Control when the throttle is being controlled by RPM Control; thus, degrading the autopilot's ability to control altitude to dangerously low levels. When operating in RPM Control the user needs to make sure that any climbs or descents are not steep enough to cause the autopilot to violate any fast or slow airspeed limits that could cause the autopilot to switch to some form of Airspeed Control while operating within the restricted RPM range. Scenarios of Airspeed Control in RPM Control are described below.

Slow Airspeed Mode (Lon Mode 2): If the maximum RPM were to be set low enough that it would force the airspeed of the aircraft to fall below the minimum airspeed the autopilot would switch to Slow Airspeed Mode. In this scenario Energy Control would command full throttle, but RPM Control would have throttle control authority and would restrict throttle to RPM Max. The controller bases the Airspeed Control decision on the throttle commanded by Energy Control whether Energy Control actually has throttle command authority or not. The elevator would be used to control airspeed, and the Throttle would never be able to maintain airspeed because it would always be restricted by RPM Max; therefore, the autopilot would continuously pitch the aircraft down to gain airspeed and eventually crash into the ground. For this scenario to play out to destruction it would have to be extreme negligence and failure by part of the remote pilot; however, it is possible. The figures below depict this scenario as it occurred in a hardware in the loop simulation.

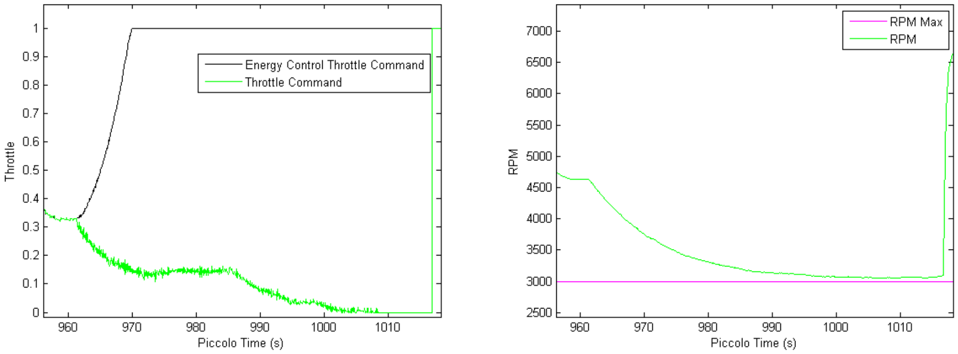


Figure 240 RPM Control Lon Mode 2 Throttle Command

In the simulation the RPM Max was set low enough (at about 965 seconds) that the aircraft could not maintain airspeed above the minimum indicated airspeed; thus, forcing the autopilot into Slow Airspeed Mode. Figure 240 shows that the Energy Control Throttle command climbed to full throttle, but the actual throttle command throttled down in order to comply with RPM Max.

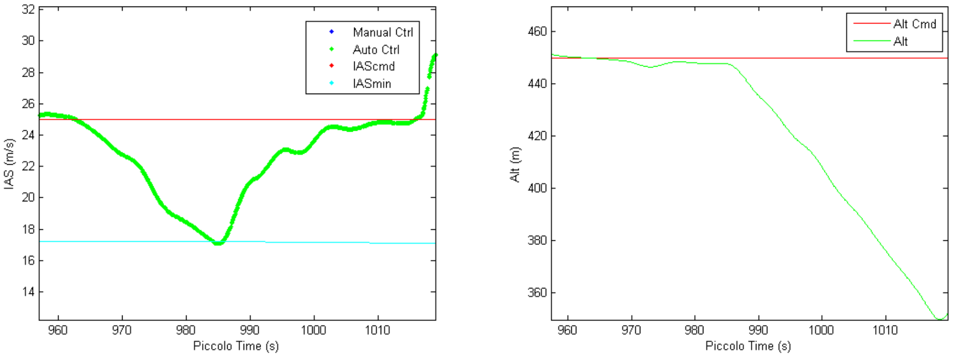


Figure 241 RPM Control Lon Mode 2 Airspeed & Altitude

Figure 241 shows that the autopilot successfully increased the airspeed up to the commanded airspeed, which was done via elevator deflections. The figure also shows that the increase in airspeed came at a cost of altitude as the aircraft pitched down to increase airspeed. In order to avoid crashing the aircraft the RPM Max limit was removed at 1015 seconds. The aircraft would have continuously descended until it reached the scene of the crash if the RPM Max limit had not been removed.

Fast Airspeed Mode (Lon Mode 3): If RPM Min was set high enough that it would force the airspeed of the aircraft to violate either the Fast IAS error Threshold or IAS_{max} the autopilot would switch to Fast Airspeed Mode. In this scenario Energy Control would command minimum throttle, but RPM Control would have throttle control authority and would restrict throttle to RPM Min. Since the controller bases the Airspeed Control decision on the throttle commanded by Energy Control, regardless of whether Energy Control actually has throttle command authority or not, the throttle would never be able to maintain airspeed because it would always be restricted by RPM Min. The elevator would be used to control airspeed; thus, the autopilot would continuously pitch the aircraft up to lose airspeed and climb indefinitely. For this scenario to play out to destruction it would have to be extreme negligence and failure by part of the remote pilot; however, it is possible. The figures below depict this scenario as it occurred in a hardware in the loop simulation.

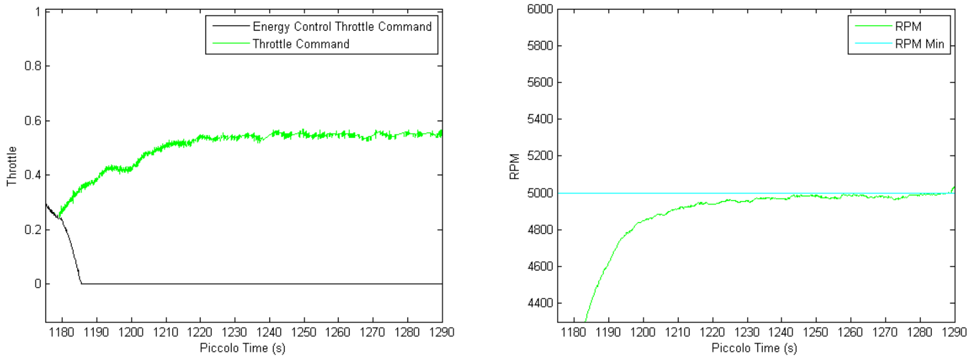


Figure 242 RPM Control Lon Mode 3 Throttle Command

In the simulation the RPM Min was set high enough (at about 1170 seconds) that the aircraft would violate the Fast IAS error Threshold (4 m/s); thus, forcing the autopilot into Fast

Airspeed Mode. Figure 242 shows that the Energy Control throttle command decreased to Throttle Min, but the RPM Control throttle command increased in order to comply with RPM Min.

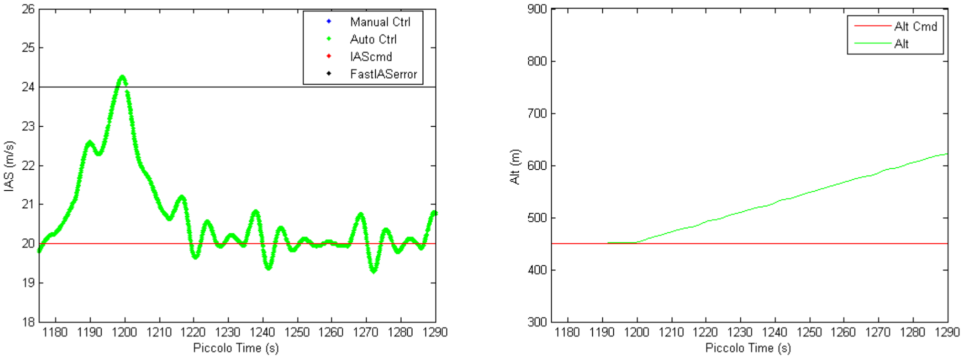


Figure 243 RPM Control Lon Mode 3 Airspeed & Altitude

Figure 243 shows that the autopilot successfully decreased the airspeed to the commanded airspeed, which was done via Elevator deflections. The figure also shows that the airspeed decrease came at a cost of altitude as the autopilot pitched the aircraft up in order to decrease airspeed. In order to stop the aircraft from climbing indefinitely the RPM Min limit was removed. The aircraft would have continuously climbed if RPM Min had not been removed or altered.

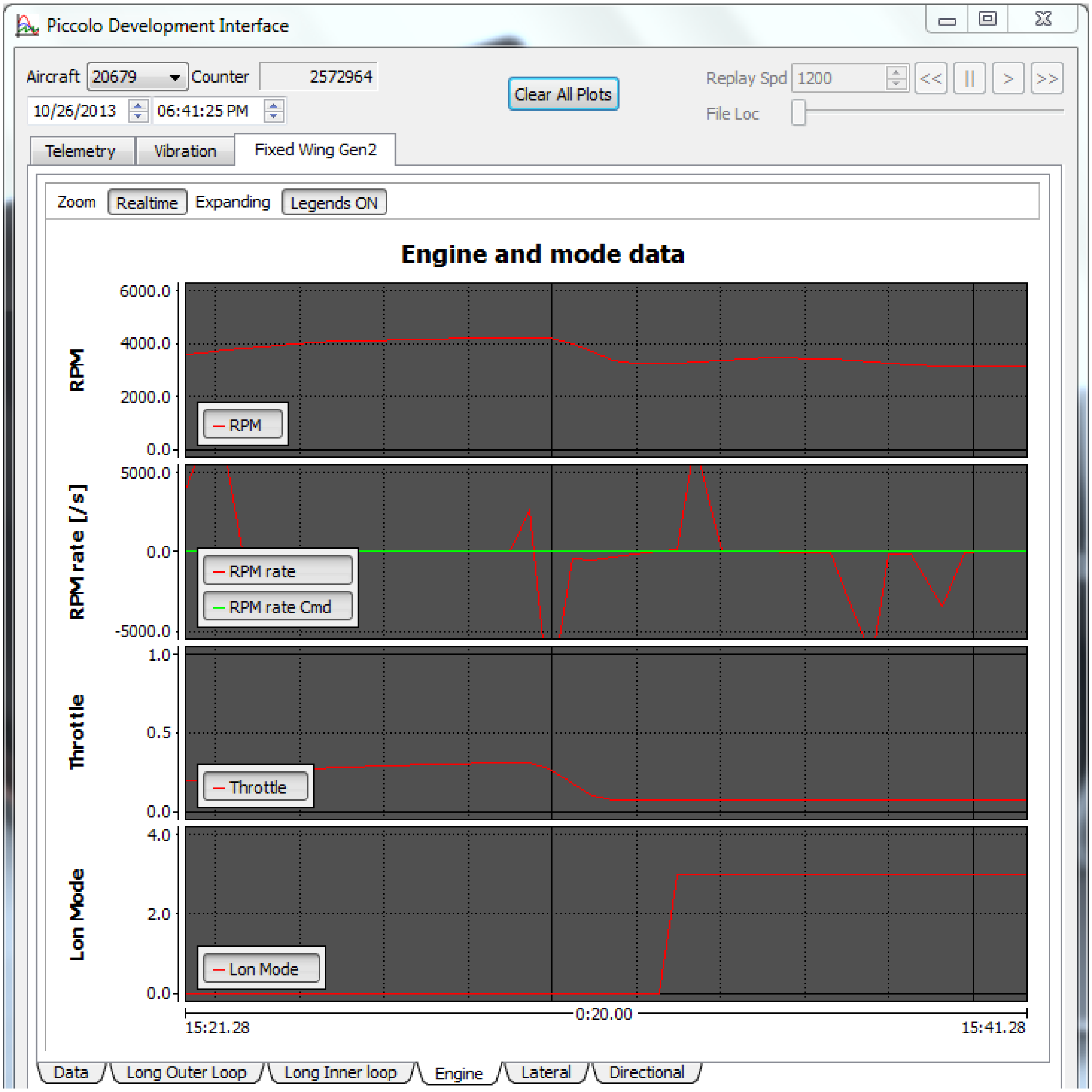
5.8.5. Longitudinal Modes (Lon Modes)

The upgrade from version 2.1.2.h to 2.1.4.a introduced new controller logic with new longitudinal flight modes called “Lon Modes”. Lon Modes are used to change the longitudinal control state of the autopilot from Altitude Control to various forms of Airspeed Control.

There are four Lon Modes total, numbered from 0-1. Lon Mode 0 is Altitude Control, and Lon Modes 1-3 are various forms of Airspeed Control with differing versions of Energy Control. The Lon Modes are titled as follows:

Lon Mode 0:	Altitude Mode
Lon Mode 1:	Airspeed Mode
Lon Mode 2:	Slow Airspeed Mode
Lon Mode 3:	Fast Airspeed Mode

Airspeed Control operates the same throughout Lon Modes 1-3 where the elevator is sued to control airspeed. Lon Modes 1-3 require certain conditions to be met in order for the autopilot to transition out of Lon Mode 0. The only difference is that Lon Modes 2-3 are entered automatically by the autopilot, when it is determined that throttle cannot appropriately control airspeed based on user limits and vehicle gains. The autopilot will stay in Lon Modes 2-3 until certain conditions are met to signify that the throttle can be used to control airspeed again. The Longitudinal Mode of the autopilot is displayed in the Dev Interface pictured in Figure 244 below.



Aircraft 20679 Counter 2572964

10/26/2013 06:41:25 PM

Replay Spd 1200

File Loc

Figure 244 Lon Mode

Lon Mode 0 (Altitude Mode):

Lon Mode 0 represents Altitude Control and is the default longitudinal mode that the piccolo operates in.

Lon Mode 1 (Airspeed Mode):

Entrance Conditions:

1) Fast IAS error Threshold < 0

Exit Conditions:

1) Fast IAS error Threshold >= 0

Lon Mode 1 is Airspeed Control. This mode is for operating completely in Airspeed Control, and can only be entered and exited intentionally by the remote pilot via manipulation of the Fast IAS error Threshold.

Lon Mode 2 (Slow Airspeed Mode):

Entrance Conditions:

1) IAS < IAS_{min} or IAS < Slow IAS error Threshold

2) Throttle >= 90% of Throttle Max

Exit Conditions:

1) IAS > IAS_{min} and IAS > Slow IAS error Threshold

2) A significant amount of Altitude Error has been removed. Enough to cause Energy Control to begin decreasing the Energy Rate command.

Lon Mode 2 utilizes Airspeed Control to increase airspeed in the event that the throttle cannot add enough Energy to the system. This mode is designed to prevent the autopilot from trying to climb beyond the capabilities of the aircraft; however, it relies heavily upon the limitations set in the autopilot's settings. The autopilot will not go into Lon Mode 2 if the airspeed is not below IAS_{min} or the Slow IAS error Threshold. Also the Climb Max Fraction, which dictates the maximum vertical rate that the autopilot can command, affects how steep of a climb the autopilot attempts to perform in the first place.

The manner that the autopilot controls the elevator is the same as normal Airspeed Control; however, throttle is different. Throughout the duration of Lon Mode 2 the throttle trim is automatically set to Throttle Max. By holding throttle at Throttle Max the autopilot is able to continue to climb. Initially when the autopilot enters Lon Mode 2 the airspeed is too low so the autopilot will command the elevator to pitch the aircraft down and increase airspeed. Since the throttle is held at Throttle Max the aircraft won't have to pitch down much at all to increase the airspeed; thus the aircraft will continue to climb with a shallower slope. After a couple of pitch down and up oscillations the autopilot will eventually find the elevator deflection that can maintain airspeed above the minimum airspeed while climbing with full throttle.

Since the autopilot is in Airspeed Control, Energy Control will command an Energy Rate based on the altitude error. At some point the altitude error will decrease enough that the Energy Rate command will begin to decrease. When the Energy Rate commands decrease enough the throttle integrator will begin to command a decrease in throttle trim. At this point, if the indicated airspeed is greater than the minimum airspeed limits the autopilot will return to Lon Mode 0.

Lon Mode 3 (Fast Airspeed Mode):

Entrance Conditions:

1) $IAS > IAS_{max}$ or $IAS >$ Fast IAS error Threshold

2) Throttle = Throttle Min

Exit Conditions:

1) $IAS < IAS_{max}$ and $IAS <$ Fast IAS error Threshold

2) A significant amount of Altitude Error has been removed. Enough to cause Energy Control to begin decreasing the Energy Rate command.

Lon Mode 3 utilizes Airspeed Control to decrease airspeed in the event that the Throttle cannot remove enough Energy from the system. This mode is designed to prevent the autopilot from trying to descend beyond the capabilities of the aircraft; however, it relies heavily upon the limitations set in the autopilot's settings. The autopilot will not go into Lon Mode 3 if the airspeed is not above IAS_{max} or the Fast IAS error Threshold. Also the Descent Max Fraction, which dictates the minimum vertical rate (or maximum negative vertical rate) that the autopilot can command, affects how steep of a descent the autopilot attempts to perform in the first place.

The manner that the autopilot controls the elevator is the same as normal Airspeed Control; however, throttle is different. Throughout the duration of Lon Mode 3 the throttle trim is automatically set to Throttle Min. Initially when the autopilot enters Lon Mode 3 the airspeed is too high so the autopilot will command the elevator to pitch the aircraft up and decrease airspeed. Since the throttle is held at Throttle Min the aircraft won't have to pitch up much at all to decrease the airspeed; thus the aircraft will continue to descend with a shallower slope. After a couple of pitch up and down oscillations the autopilot will eventually find the elevator deflection that can maintain airspeed below the maximum airspeed limits while descending with minimum throttle.

Since the autopilot is in Airspeed Control, Energy Control will command an Energy Rate based on the altitude error. At some point the altitude error will decrease enough that the Energy Rate command will begin to decrease. When the Energy Rate commands decrease enough that the actual Energy Rate is more negative than the Energy Rate command, Energy Control will begin to want to add Energy to the system. At this point, if the indicated airspeed is below the maximum airspeed limits the autopilot will return to Lon Mode 0.

5.9. Altitude vs. Airspeed Control Summary

Altitude and Airspeed Control are categorized by Longitudinal Modes, or Lon Modes. There are 4 Lon Modes numbered from 0-3. Lon Modes 1-3 correspond to Airspeed Control with different entrance and exit conditions. Lon Mode 0 is Altitude Control and is the standard flight Lon Mode.

In Altitude Control (Lon Mode 0) the Altitude Control Loop commands Elevator deflections in order to maintain Altitude. The autopilot will depend on the Throttle, via Energy Control, to maintain airspeed and VRate.

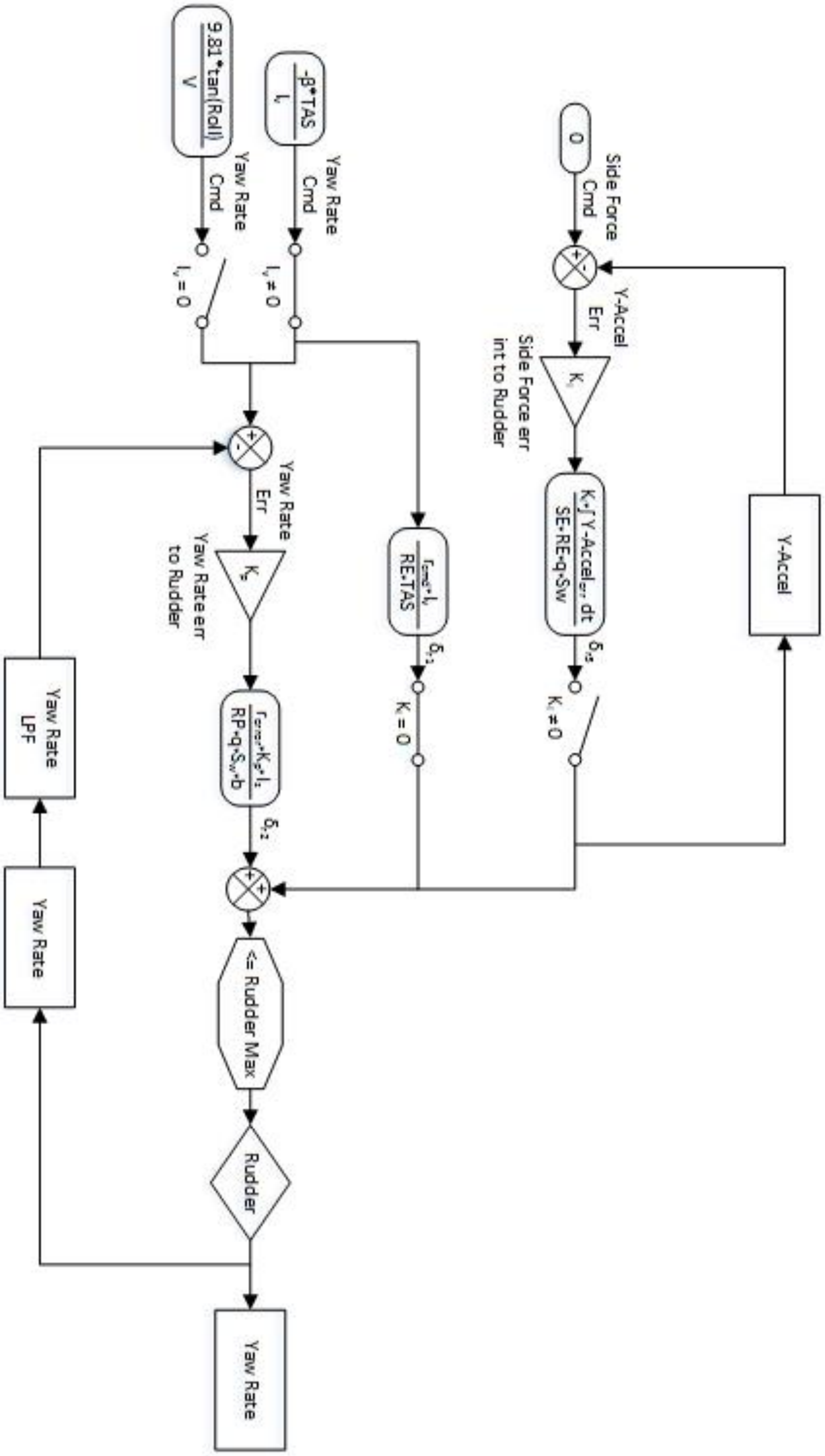
In Airspeed control (Lon Modes 1 -3) the Airspeed Control Loop commands Elevator deflections in order to maintain airspeed. The autopilot will pitch the plane up to decrease airspeed and pitch the plane down to increase airspeed. The autopilot will depend on the Throttle, via Energy Control, to maintain Altitude. Energy Control will use the measured and commanded TAS with the Kinetic Energy equation to command Energy Rates and Throttle.

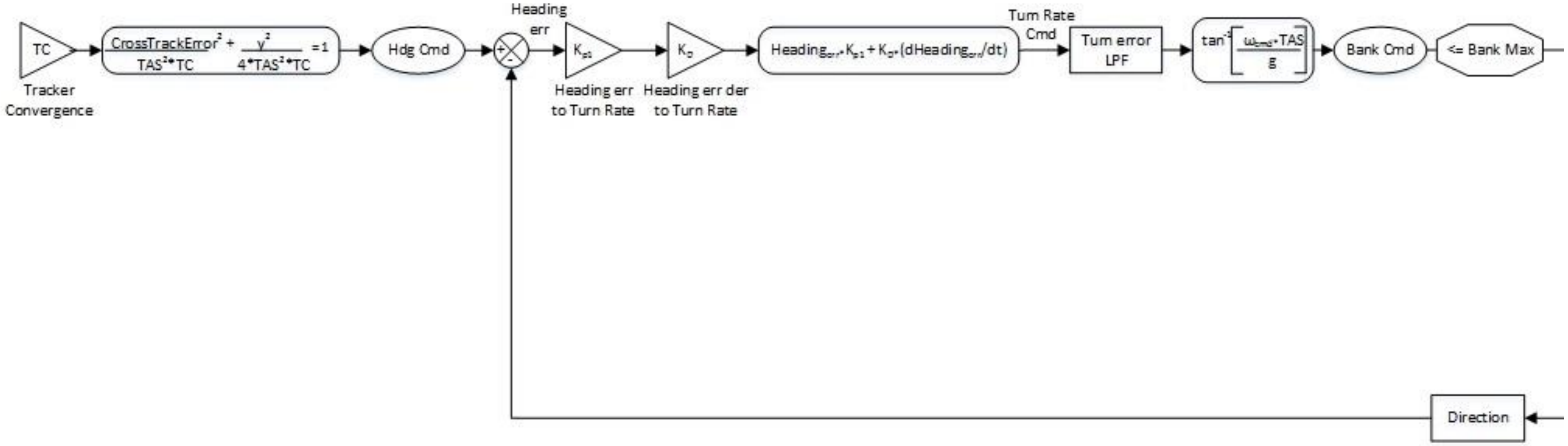
Operating in Airspeed Mode (Lon Mode 1) greatly reduces the autopilot's ability to control Altitude. In Airspeed Control the autopilot controls Altitude almost by accident. If the aircraft needs to increase altitude the autopilot will increase Throttle, and as a result the airspeed will increase above the commanded airspeed; thus, the autopilot will pitch the plane up to decrease airspeed and the aircraft just happens to climb.

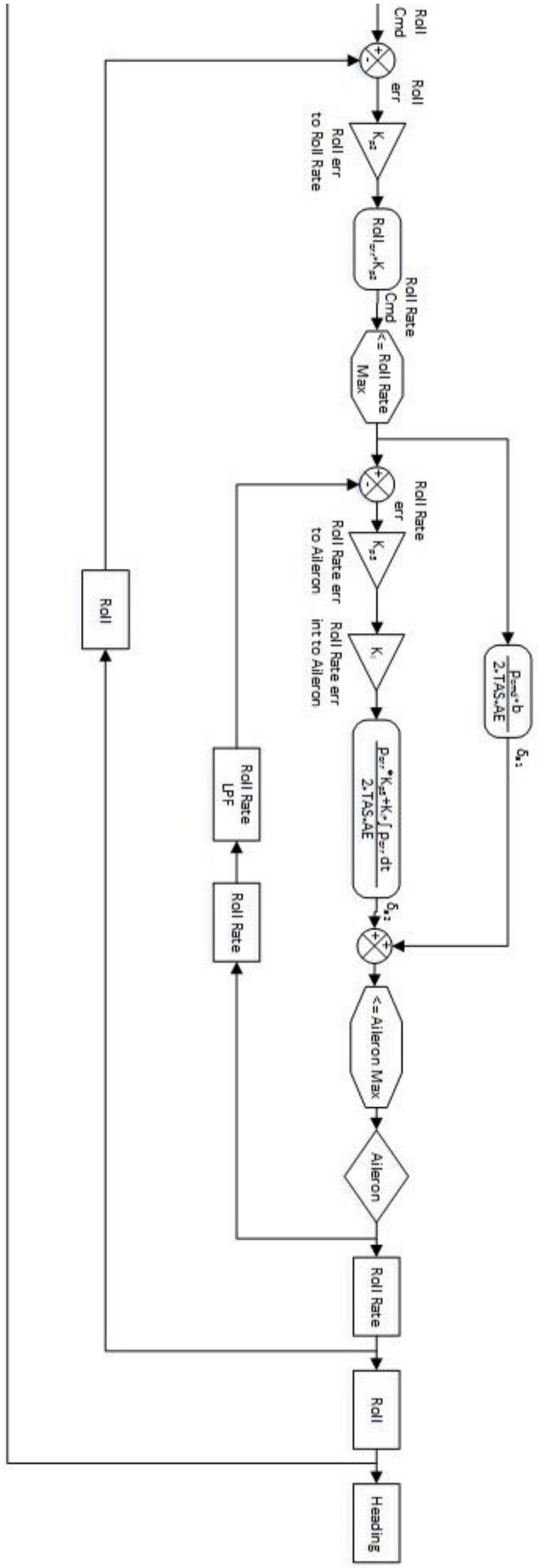
Slow Airspeed Mode (Lon Mode 2) is used to increase airspeed when the airspeed cannot be maintained by Throttle alone. Lon Mode 2 essentially gives airspeed priority over altitude by holding the Throttle at Throttle Max and deflecting the Elevator to increase airspeed.

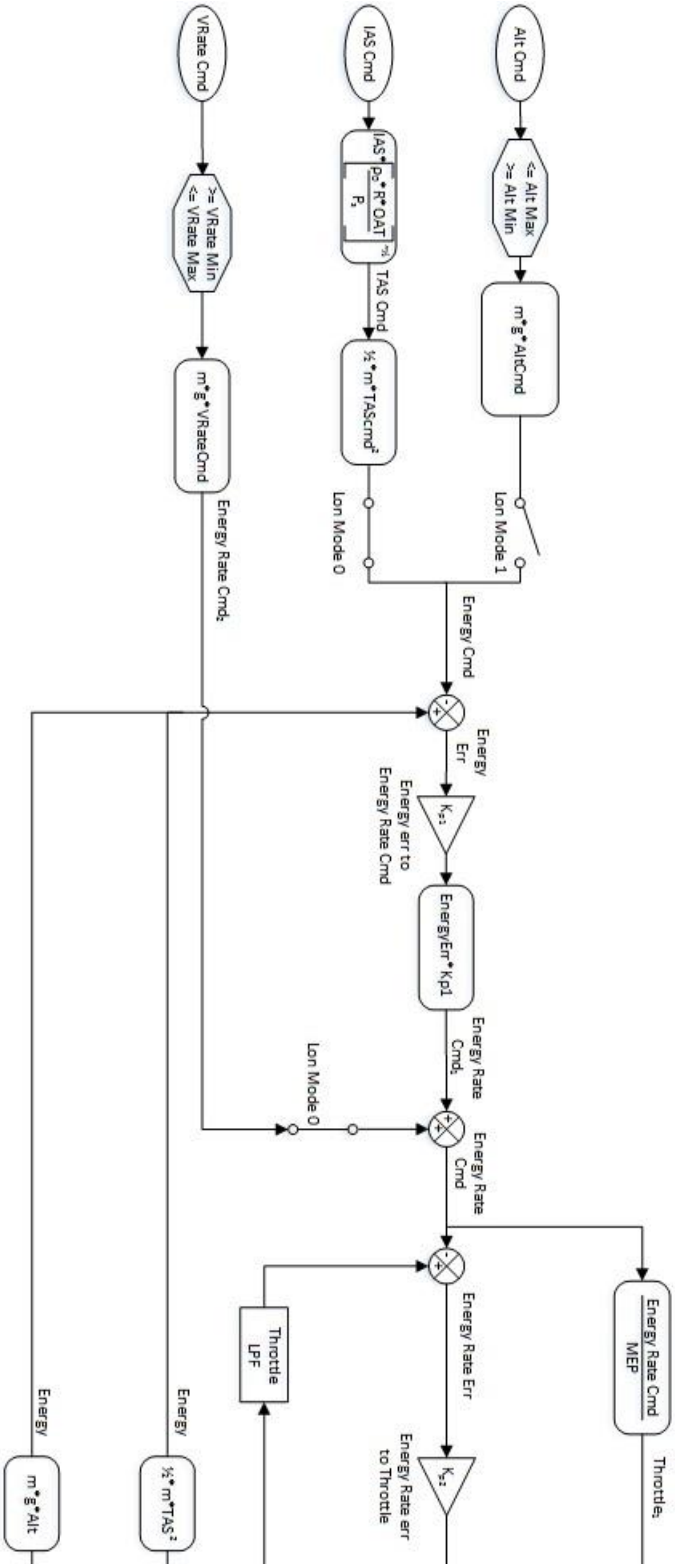
Fast Airspeed Mode (Lon Mode 3) is used to decrease airspeed when the airspeed cannot be decreased by Throttle alone. Lon Mode 3 essentially gives airspeed priority over altitude by holding the Throttle at Throttle Min and deflecting the Elevator to decrease airspeed.

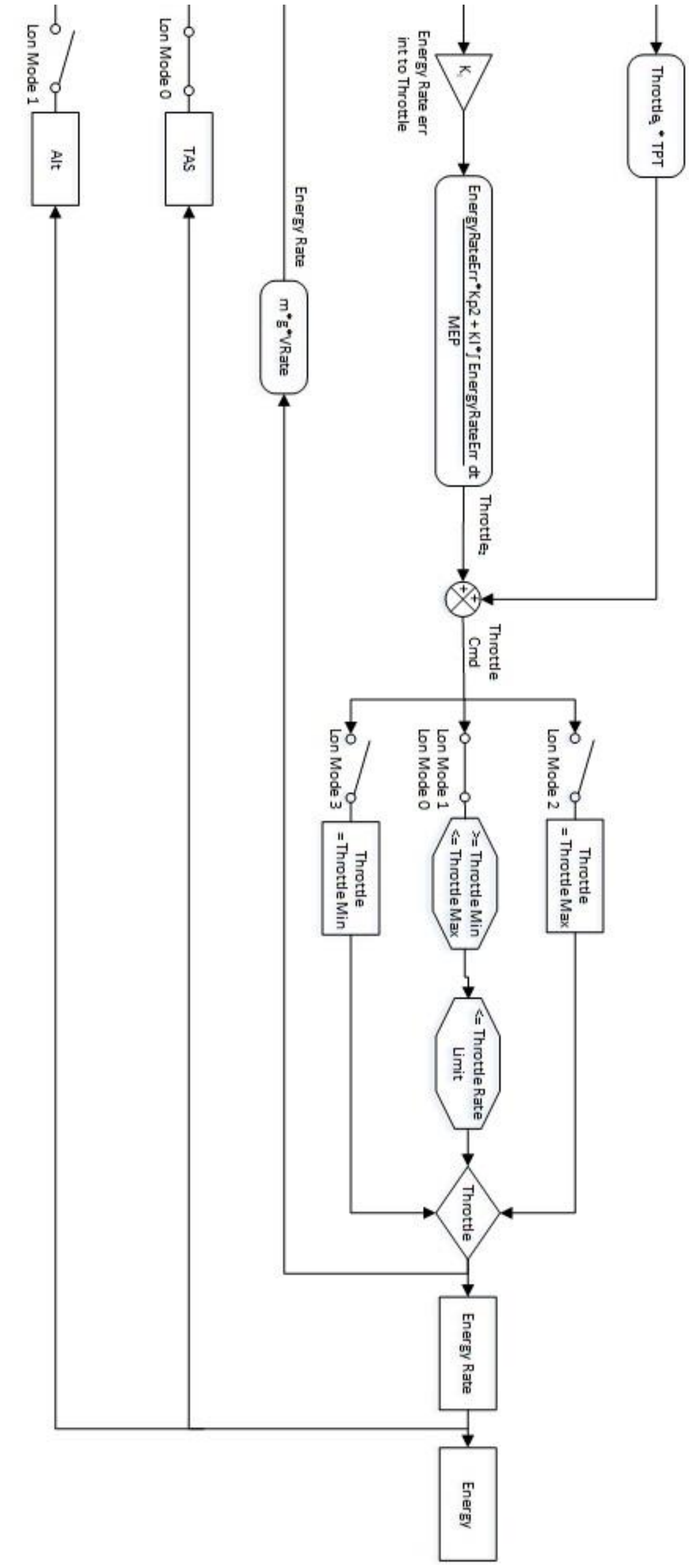
5.10. Complete Control Schematics

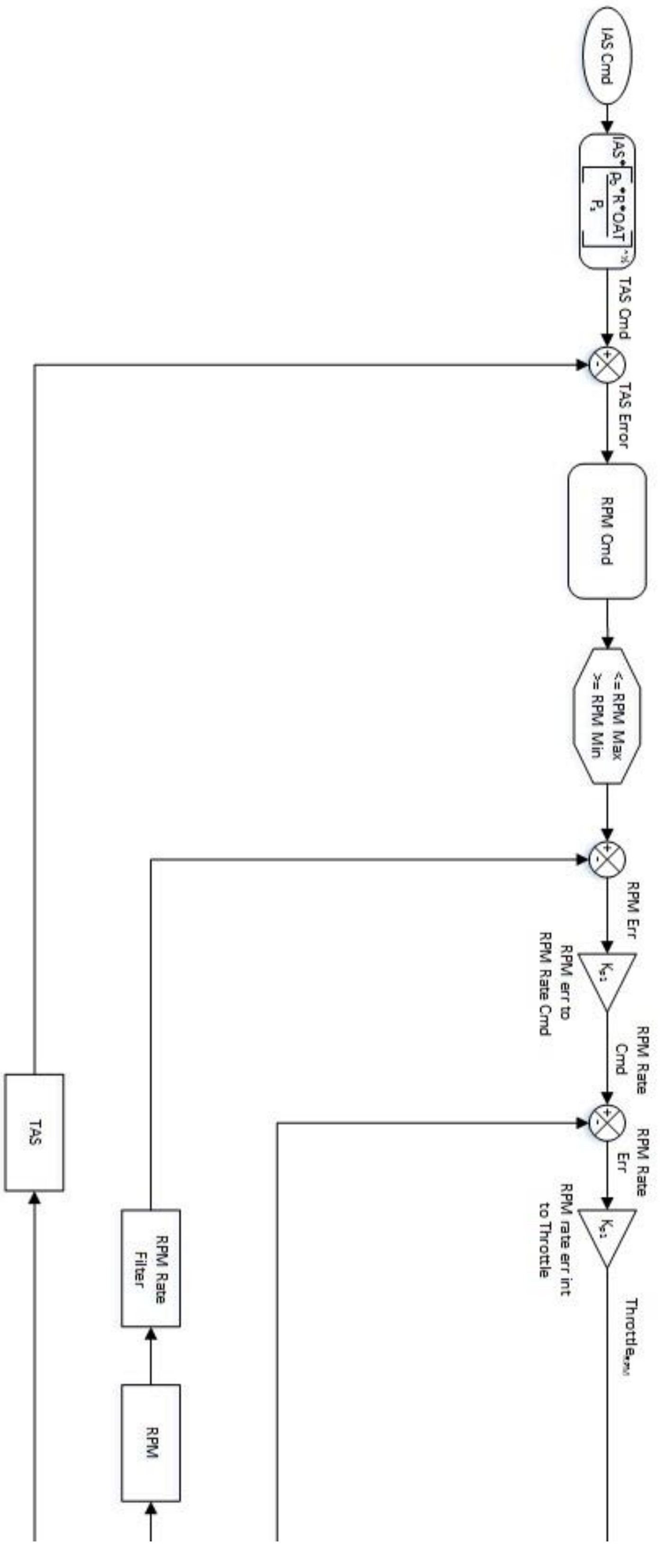


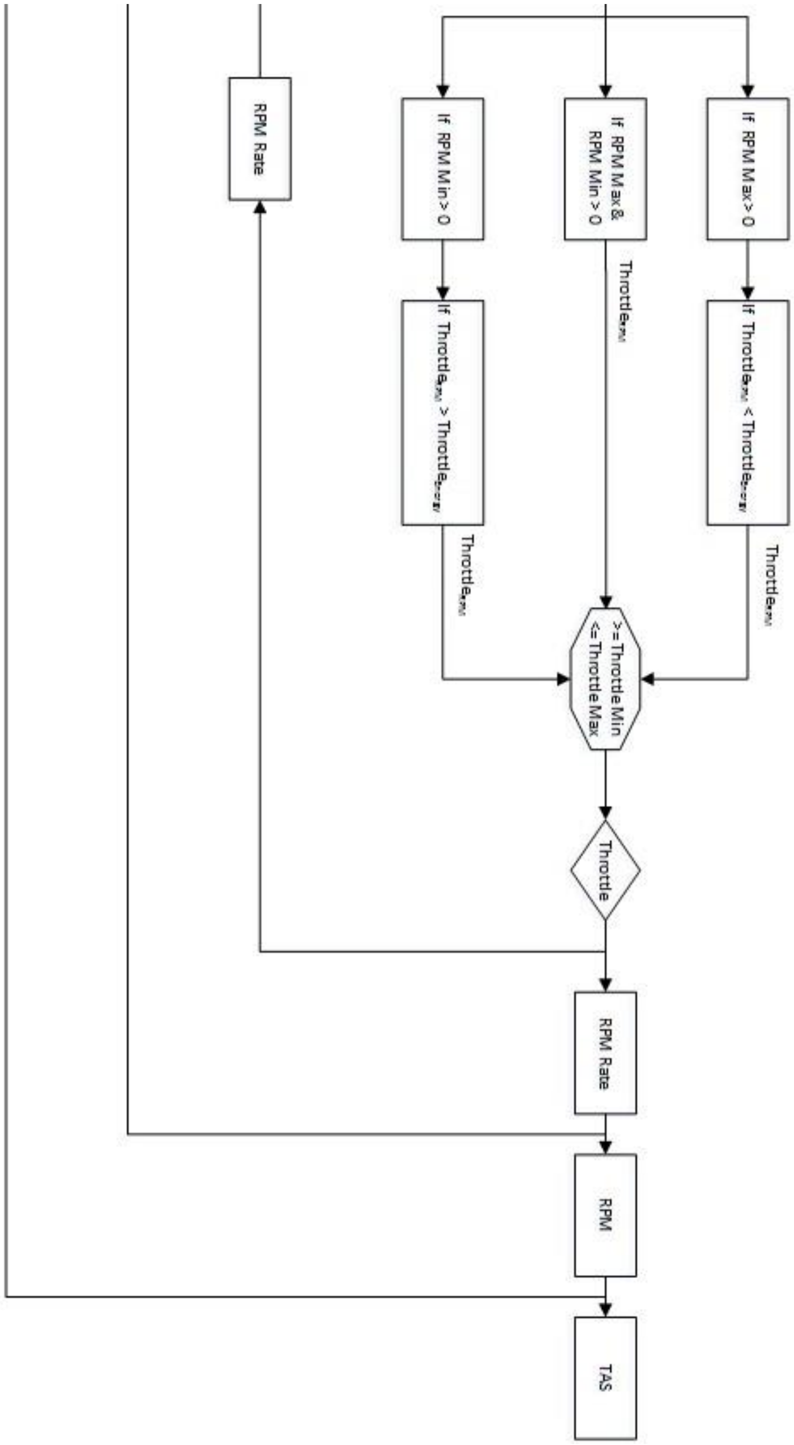

TC
Tracker
Convergence
$\frac{CrossTrackError^2}{TAS^2*TC} + \frac{y^2}{4*TAS^2*TC} = 1$
Hdg Cmd
+
-
Heading
err
K_{p1}
Heading err
to Turn Rate
K_D
Heading err der
to Turn Rate
$Heading_{err}*K_{p1} + K_D*(dHeading_{err}/dt)$
Turn Rate
Cmd
Turn error
LPF
$\tan^{-1}\left[\frac{\omega_{cmd}*TAS}{g}\right]$
Bank Cmd
<= Bank Max
Direction


Roll Cmd
+
-
Roll err
$K_{\phi 2}$
Roll err to Roll Rate
$Roll_{err} \cdot K_{\phi 2}$
Roll Rate Cmd
<= Roll Rate Max
+
-
Roll Rate err
K_{p2}
Roll Rate err to Aileron
K_i
Roll Rate err int to Aileron
$\frac{p_{cmd} \cdot b}{2 \cdot TAS \cdot AE}$
δ_{a1}
$\frac{p_{err} \cdot K_{p2} + K_i \cdot \int p_{err} \, dt}{2 \cdot TAS \cdot AE}$
δ_{a2}
+
+
<= Aileron Max
Aileron
Roll Rate
Roll
Heading
Roll Rate LPF
Roll Rate
Roll

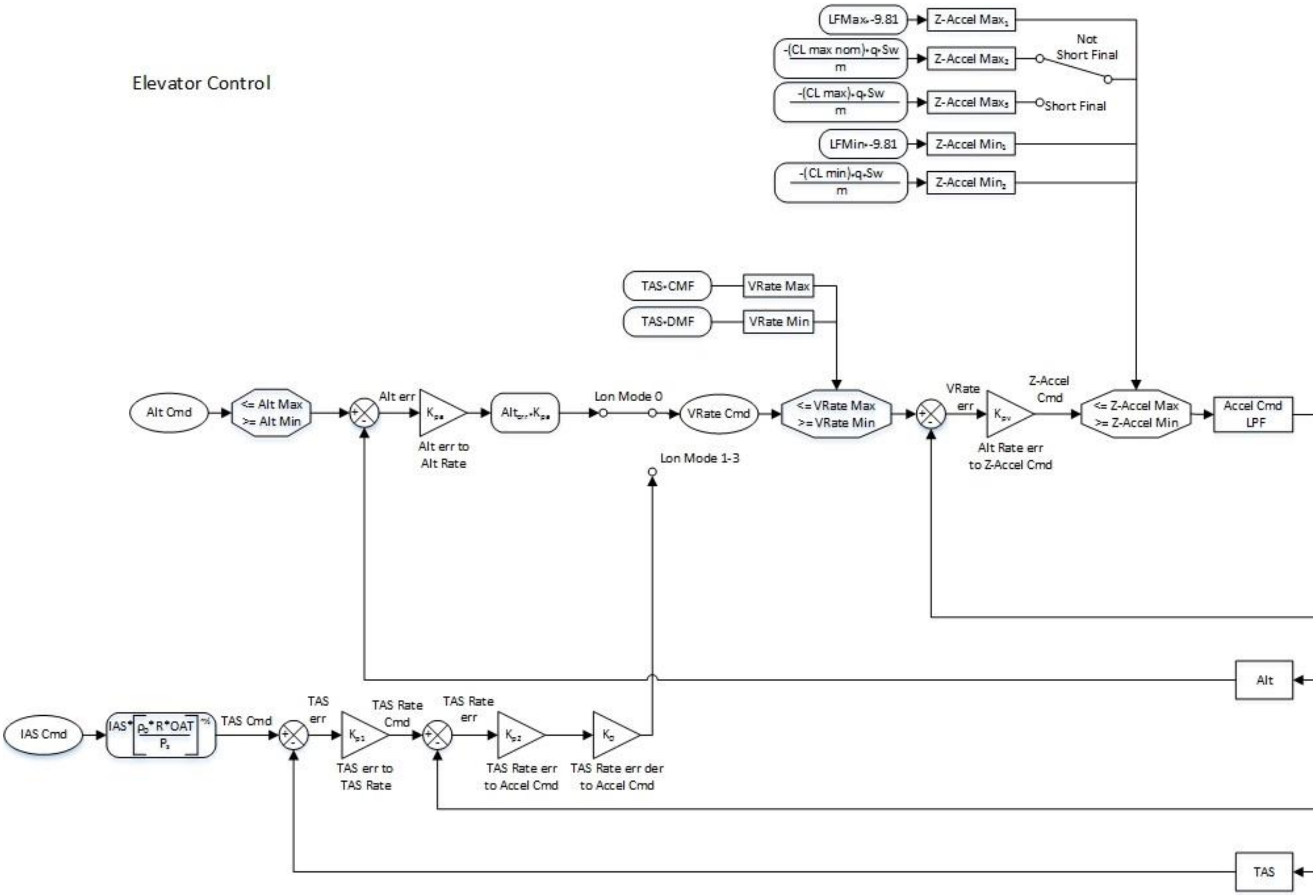

Alt Cmd
<= Alt Max
>= Alt Min
m*g*AltCmd
Lon Mode 1
IAS Cmd
IAS*(P0*R*OAT / Ps)^½
TAS Cmd
½*m*TAScmd²
Lon Mode 0
VRate Cmd
>= VRate Min
<= VRate Max
m*g*VRateCmd
Energy Rate Cmd₂
Energy Cmd
-
+
Energy Err
Kp1
Energy err to Energy Rate Cmd
EnergyErr*Kp1
Energy Rate Cmd₁
+
+
Lon Mode 0
Energy Rate Cmd
-
+
Energy Rate Cmd / MEP
Throttle₁
Energy Rate Err
Kp2
Energy Rate err to Throttle
Throttle LPF
Energy
½*m*TAS²
Energy
m*g*Alt

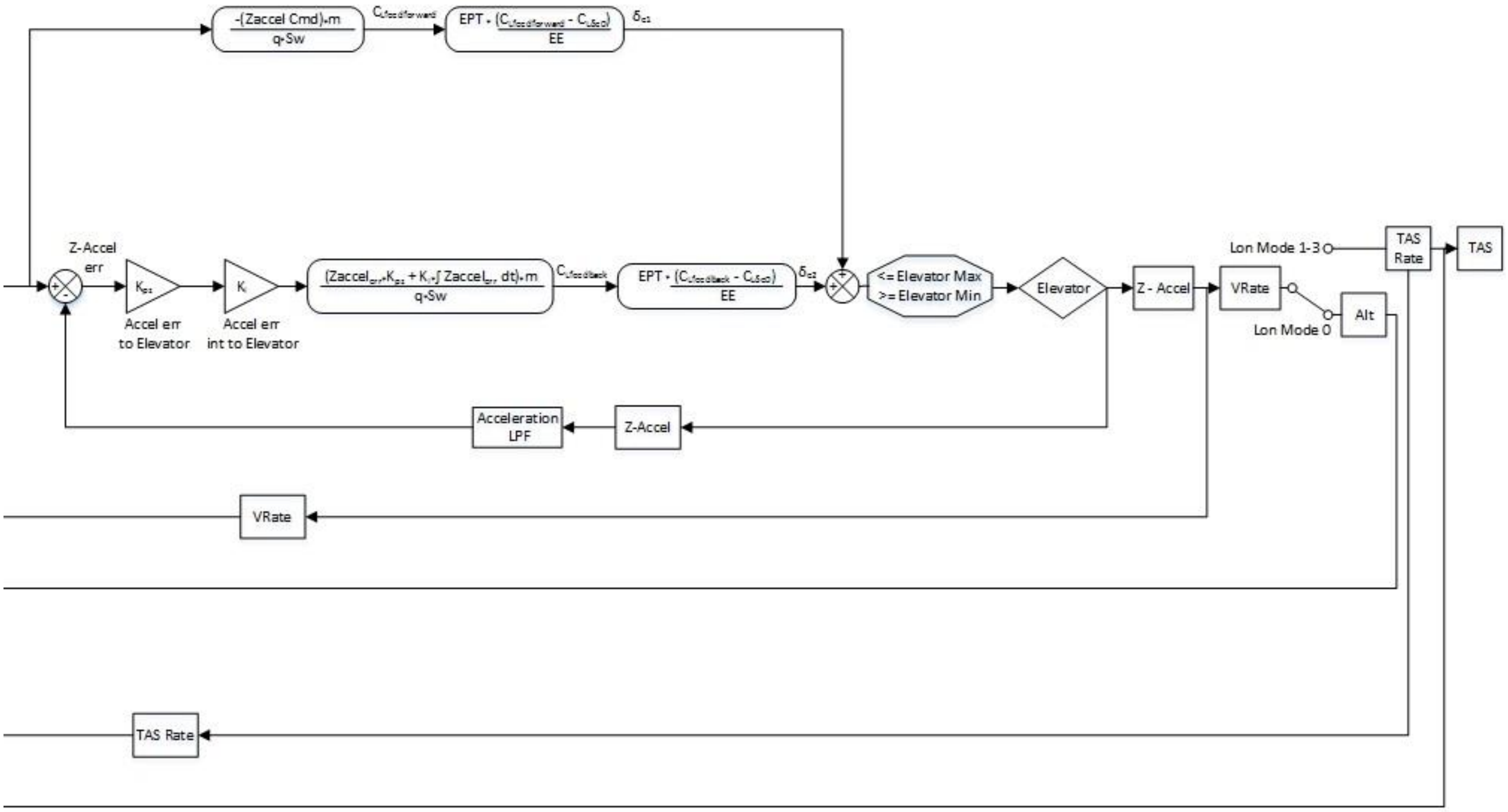

Throttle$_0$ * TPT
K_i
Energy Rate err
int to Throttle
EnergyRateErr*Kp2 + KI*∫EnergyRateErr dt
MEP
Throttle$_1$
+
+
Throttle
Cmd
Energy Rate
m*g*VRate
Lon Mode 0
TAS
Lon Mode 1
Alt
Lon Mode 2
Throttle
= Throttle Max
Lon Mode 1
Lon Mode 0
>= Throttle Min
<= Throttle Max
Lon Mode 3
Throttle
= Throttle Min
<= Throttle Rate
Limit
Throttle
Energy Rate
Energy


IAS Cmd
IAS*[P0*R*OAT / Ps]^½
TAS Cmd
+
-
TAS Error
RPM Cmd
<= RPM Max
>= RPM Min
+
-
RPM Err
Ks1
RPM err to
RPM Rate Cmd
RPM Rate
Cmd
+
-
RPM Rate
Err
Ks1
RPM rate err int
to Throttle
Throttle_cmd
TAS
RPM Rate
Filter
RPM


If RPM Max > 0
If Throttle$_{RPM}$ < Throttle$_{Energy}$
Throttle$_{RPM}$
If RPM Max & RPM Min > 0
Throttle$_{RPM}$
If RPM Min > 0
If Throttle$_{RPM}$ > Throttle$_{Energy}$
Throttle$_{RPM}$
>= Throttle Min
<= Throttle Max
Throttle
RPM Rate
RPM
TAS
RPM Rate

Elevator Control




$\frac{-(\text{Zaccel Cmd})*m}{q*Sw}$
$C_{Lfeedforward}$
$EPT * \frac{(C_{Lfeedforward} - C_{L\delta e0})}{EE}$
δ_{e1}
Z-Accel err
+
-
K_{pz}
Accel err to Elevator
K_i
Accel err int to Elevator
$\frac{(\text{Zaccel}_{err}*K_{pz} + K_i*\int \text{Zaccel}_{err}\, dt)*m}{q*Sw}$
$C_{Lfeedback}$
$EPT * \frac{(C_{Lfeedback} - C_{L\delta e0})}{EE}$
δ_{e2}
+
+
<= Elevator Max
>= Elevator Min
Elevator
Z - Accel
VRate
Lon Mode 1-3
Lon Mode 0
Alt
TAS Rate
TAS
Acceleration LPF
Z-Accel
VRate
TAS Rate

CHAPTER VI

DOUBLET MANEUVERS

6. Introduction

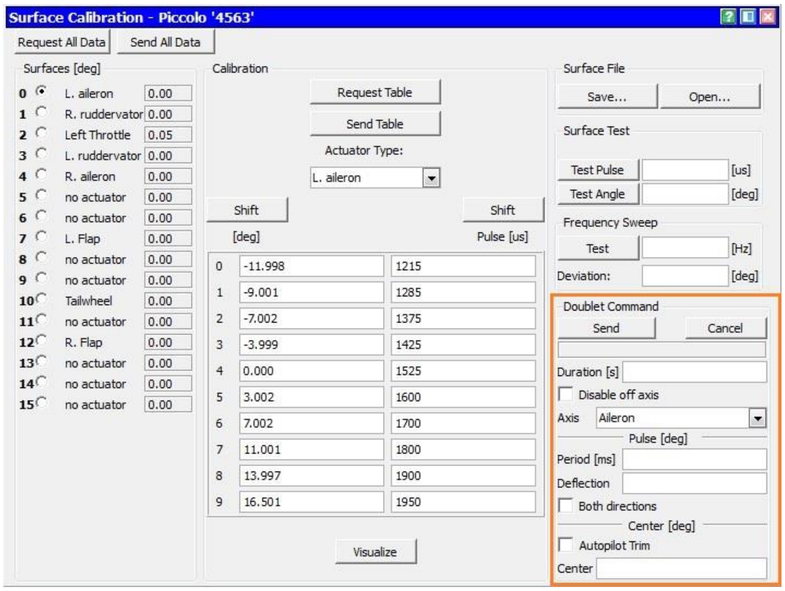
The vehicle parameters can have a significant impact on aircraft performance. More specifically the effectiveness and power terms, of the surfaces that act as ailerons, elevators, and rudders, influence aircraft response to disturbances and can cause oscillatory actuator response if they are incorrect. As a result Cloud Cap recommends performing "Doublet Maneuvers" to test specific effectiveness terms if an aircraft experiences control issues. These doublet maneuvers are centered on deflecting a specific control surface and measuring a response that applies to the vehicle parameter that is meant to represent the performance of the surface that is deflected.

The autopilot can be commanded to perform doublet maneuvers by the user through the doublet command function. Doublet maneuvers can also be performed manually by the r/c pilot; however, using the autopilot to do so is usually preferable as the autopilot is very good at holding deflections constant throughout the test. Cloud Cap offers one document, "Piccolo Doublet Analysis Tool.pdf", to explain how to conduct doublet maneuvers, in addition to one small section in "PccUsersGuide.pdf" pg. 96. The document largely deals with how to analyze the results of doublet tests. It also introduces the user to two different methods of analyzing doublet data, one method utilizes the doublet file that can be generated by PCC during autopilot performed doublet maneuvers, and the other method utilizes the piccolo log file. The two different methods will be referred to as the “Doublet File Method” and the “Piccolo Log File Method.”

Cloud cap offers two separate MATLAB scripts, discussed in Section 2.8 CCT MATLAB, to analyze doublet data, “plotdoublet.m”, and “doublet.m”. doublet.m is not very user friendly with respect to the logistics of how to select the time span for analysis. Additionally the doublet.m code sets specific control surfaces to specific actuator numbers (ie elevator = surface0), which means that the code has to be altered for aircraft that don’t have the exact same surfaces assigned to the same actuator. As a result a GUI with a corresponding script was created to replace doublet.m. The new script is called “DoubletPiccoloLog.m”. The program utilizes portions of doublet.m; however, it is much more user friendly and does not require the code to be changed in order to designate which surface number corresponds with which actuator. The Doublet File Method utilizes the “plotdoublet” script while the Piccolo Log File Method utilizes the “DoubletPiccoloLog” script.

6.1. Commanding Doublet Maneuvers

PCC comes equipped with a doublet command function built in. The doublet command function is a tool that allows the user to command the autopilot to perform specific doublet maneuvers by disabling control loops and comanding specific surface deflections. The doublet command function is located in the “Surface Calibration” window of PCC as shown below in the orange highlighted area of Figure 245.



Surface Calibration - Piccolo '4563'

Request All Data | Send All Data

Surfaces [deg]

0	●	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.05
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Request Table | Send Table

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File: Save... | Open...

Surface Test: Test Pulse [us] | Test Angle [deg]

Frequency Sweep: Test [Hz] | Deviation: [deg]

Doublet Command

Duration [s]

□ Disable off axis

Axis: Aileron

Pulse [deg]

Period [ms]

Deflection

□ Both directions

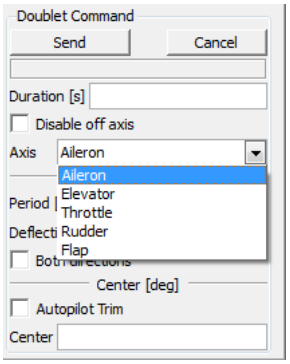
Center [deg]

□ Autopilot Trim

Center

Figure 245 Surface Calibration Window

The available surfaces to deflect using the doublet command function is the aileron, elevator, rudder, throttle, and flaps. The vehicle parameters that the doublet command function can test is the aileron, elevator, rudder, and flap effectivenesses. In the version that this guide is designed for (Fixed Wing Gen 2 2.1.4.i) the flaps are not used automaitcally by the autopilot; therefore, flap doublets will not be discussed as the flap effectiveness parameter is not used in any control loop calculations.



Doublet Command

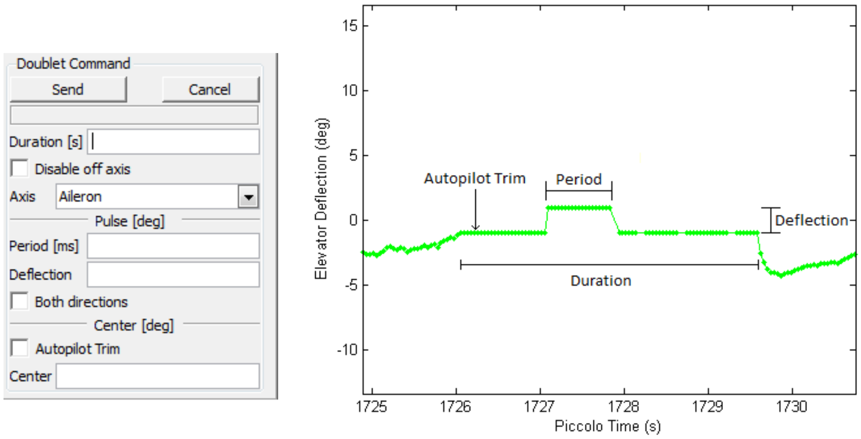
Duration [s]

□ Disable off axis

Axis: Aileron

Figure 246 Doublet Command Axis

The drop down menu next to “Axis” displays the five different control surfaces that the user can choose from to perform doublet maneuvers for. Figure 246 depicts the choices which are “Aileron”, “Elevator”, “Throttle”, “Rudder”, and “Flap”. Note that it doesn’t matter what kind of mixed surface actuators exist (Ruddervators or Elevons etc.) as long as they have been appropriately assigned via the actuator type drop down menu or via the “mixing” tab of the controller configuration window. Any surface that contains the selected axis will be deflected in the doublet maneuever test. For example a surface listed as a “Ruddervator” will be deflected for both elevator and rudder doublets. Control surface mixing will not create issues if the Doublet File Method is used, because the surface deflections are recorded as their basic independent actuator types; however, the Piccolo Log File method will require control surface unmixing by the user.


Doublet Command

Duration [s]
□ Disable off axis
Axis: Aileron
Pulse [deg]
Period [ms]
Deflection
□ Both directions
Center [deg]
□ Autopilot Trim
Center

Figure 247 Doublet Maneuver Settings

The following explains each doublet command parameter:

1) **<u>Duration</u>** sets the amount of time, in seconds, that the controller samples and records the doublet data. The initial surface deflection, determined by the parameters in the “Center” section, will be held for 1 second before the doublet surface deflection(s) is

commanded. The initial surface deflection will be held for the remainder of the duration after the doublet surface deflection(s) are completed.

2) **Disable off axis** will hold all other surfaces constant at the deflection they had when the doublet test began. Note that throttle is automatically held constant for all doublet maneuvers regardless of this setting.

3) **Period** defines the amount of time, in milliseconds, that the commanded deflection is held. If both directions is used then each deflection (the + and -) will be held for the length of the period setting.

4) **Deflection** sets the amount of deflection, in degrees, that the control surface will deflect from the initial state of the control surface. The deflection can be positive or negative.

5) **Both directions** will command both positive and negative deflection of the magnitude of the deflection value. "When doing both directions, the sign of the deflection identifies which way the surface moves first" (PccUsersGuide pg. 96).

6) **Autopilot trim** will use the trim value as estimated by the autopilot control for the selected surface to set as the initial deflection for the doublet test. This value will be held for 1 second, after the doublet command is issued, before the change in deflection occurs. This value will also be the deflection that the autopilot returns to after the desired deflection, and will be held until the duration is complete.

7) **Center** allows the user to determine the initial deflection for the doublet maneuver to begin with. This value will be held for 1 second, after the doublet command is issued, before the desired deflection occurs. This value will also be the deflection

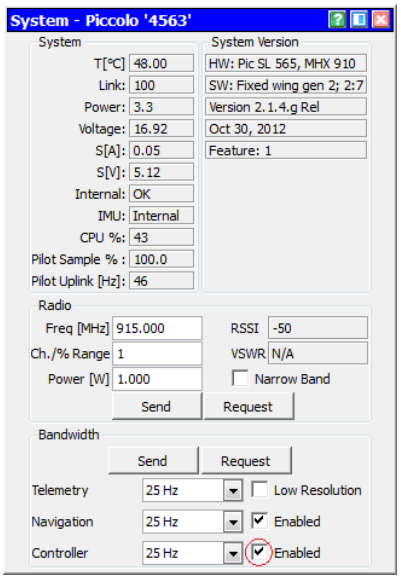
that the autopilot returns to after the desired deflection, and will be held until the duration is complete.

6.2. Doublet File Method

The Doublet File Method utilizes the doublet file, which can be generated by PCC when conducting doublet maneuvers. The Cloud Cap provided program “plotdoublet.m” is used to analyze doublet maneuvers via the Doublet File Method.

The doublet file is recorded by PCC as a notepad text file. Each time a doublet file is generated it is automatically saved in the PCC folder with the name “Doublet#” where # is the current number of the doublet done in the current flight. The doublet file records all data associated with all of the possible doublets, it is not unique to each control surface. The data is recorded directly from the controller at a frequency of 100 Hz which is much higher than the max frequency of the piccolo telemetry log file (25 Hz).

The doublet file method records surface deflections as actuator types rather than actuator numbers. This means that the doublet analysis is not concerned with mixed surface actuators. As a result the doublet file method is preferable for aircraft with mixed surface actuators.



System - Piccolo '4563'

System
T[°C] 48.00
Link: 100
Power: 3.3
Voltage: 16.92
S[A]: 0.05
S[V]: 5.12
Internal: OK
IMU: Internal
CPU %: 43
Pilot Sample % : 100.0
Pilot Uplink [Hz]: 46

System Version
HW: Pic SL 565, MHX 910
SW: Fixed wing gen 2; 2:7
Version 2.1.4.g Rel
Oct 30, 2012
Feature: 1

Radio
Freq [MHz] 915.000
RSSI -50
Ch./% Range 1
VSWR N/A
Power [W] 1.000
□ Narrow Band

Bandwidth

Telemetry 25 Hz □ Low Resolution
Navigation 25 Hz ☑ Enabled
Controller 25 Hz ☑ Enabled

Figure 248 Controller Telemetry

In order to utilize the doublet file method controller telemetry must be disabled. If controller telemetry is not disabled PCC will not record the doublet file. After a doublet manevuer is performed the doublet notepad file will pop up.

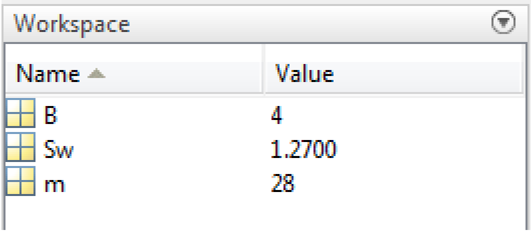
6.2.1. plotdoublet

The plotdoublet.m MATLAB script is provided by Cloud Cap. The purpose of the script is to analyze aileron, elevator, and rudder doublets by importing the data from the generated doublet file and create MATLAB variables and plots with a series of user input actions. The script also references two other scripts, “loadlogfilepiccolo.m”, and “deltadoublet.m” in order to complete all the necessary functions.

6.2.1.1. plotdoublet mechanics

The script begins by asking the user to select the doublet text file. When the user selects the file the script calls the loadlogfilepiccolo script to import the data from the text file into variables in the MATLAB workspace. Each column of the doublet text file is stored as a variable in a structure called “dat”. It is from this structure that the appropriate variables are called from to use in calculations.

Once a file has been selected the script will load the aircraft data that is stored in the .mat file “aircraftdata.mat” in the current MATLAB directory. Aircraftdata.mat stores the winspan, wing area, and mass in SI units as the variables B, Sw, and m. Figure 248 below depicts the variables.



Name	Value
B	4
Sw	1.2700
m	28

Figure 249 PlotDoublet Aircraft Data Mat File

The script will display the current aircraft data settings in the command window and provide the user the opportunity to change them or accept them. After the aircraft data has been set by the user the script automatically generates 8 figures. The first 5 figures are plots of various system response and deflections. The last 3 figures are automatically numbered as Figures 17 – 19 and are the plots that are used for doublet analysis.

Figure 17 is the elevator effectiveness plot. The original Cloud Cap code would plot C_L and elevator deflection as two separate subplots. The code was modified to include the pitch rate as a third subplot. The inclusion of the pitch rate was to help the user discern where to select the 3rd and 4th points during analysis. The script calculates C_L with the mass and wing area from the aircraft data and the z – accelerometer and dynamic pressure measurements as shown in Equation 68.

$$C_L = \frac{m(-Zaccel)}{qS_w}$$

Equation 68 PlotDoublet CL Calculation

Figure 18 is the aileron effectiveness plot which plots the dimensionless roll rate vs. aileron deflection. The script calculates the dimensionless roll rate with the wingspan from the aircraft data and the measured roll rate, dynamic pressure, and estimated air density.

$$TAS = \sqrt{\frac{2q}{\rho}}$$

Equation 69 PlotDoublet TAS Calculation

$$Dimensionless\ Roll\ Rate = \frac{Roll\ Rate\ * b}{2(TAS)}$$

Equation 70 PlotDoublet Dimensionless Roll Rate Calculation

Note that doublet files don't record TAS, so it has to be calculated. The original version of the plotdoublet script calculated TAS with a pre-determined density set in the code. It was found that the doublet files do record the air density as estimated by the autopilot. As a result the plotdoublet code was modified to use the recorded air density rather than the pre-determined air density that would have to be manually entered in the code.

Figure 19 is the rudder effectiveness plot which plots the heading vs. rudder deflection. There is a problem with the rudder effectiveness plots. The controller records heading telemetry as +/- 180°, so if a rudder doublet were to cause the aircraft to yaw across the + to – threshold the rudder effectiveness calculation would calculate the change in heading incorrectly. Additionally if the heading were to cross over 0 degrees there would be a variable gap in the heading data.

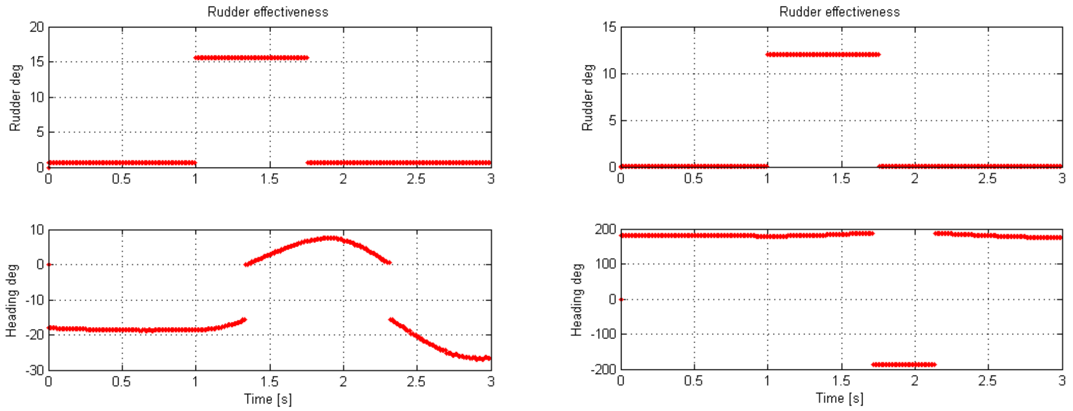


Figure 250 PlotDoublet Heading Error

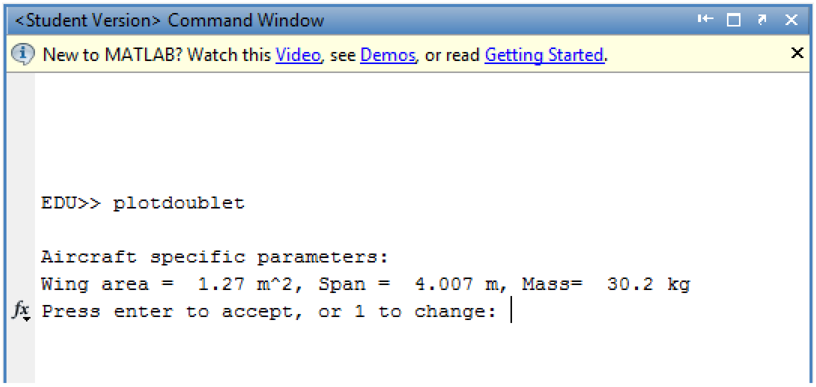
Figure 250 shows an example of two rudder doublets that encountered the heading telemetry error. The rudder doublet in the plot on the left crossed over 0 degrees heading. The gap that is created has to do with true heading correction. It is always present when the heading crosses over from one side to the other; however, the gap is not constant it is variable. The rudder doublet plot on the right shows a rudder doublet that caused an aircraft to yaw across the positive/negative 180° threshold. The rudder doublet caused the aircraft to yaw 20° from a heading of 170° to 190°; however, the recorded heading is 170° to -170°. In that scenario plotdoublet incorrectly calculated the change in heading as -340° (-170 -170).

After the plots are generated the script will launch the “deltadoublet.m” script and there will be a prompt for the user in the command window asking which surface the analysis is for. When a surface is selected the script will select the appropriate figure and the mouse cursor will become a cross hair pointer that allows the user to select points on the plot simply by clicking on the plots and the user can begin the analysis as described earlier.

6.2.1.2. plotdoublet instructions

The following steps detail how to use plotdoublet with doublet maneuvers. To begin open MATLAB and set the directory to the CCTMatlab folder. Launch "plotdoublet" by typing "plotdoublet" into the command line and follow the following steps:

1) A prompt will appear in the command window asking the user if specific aircraft parameters are correct.



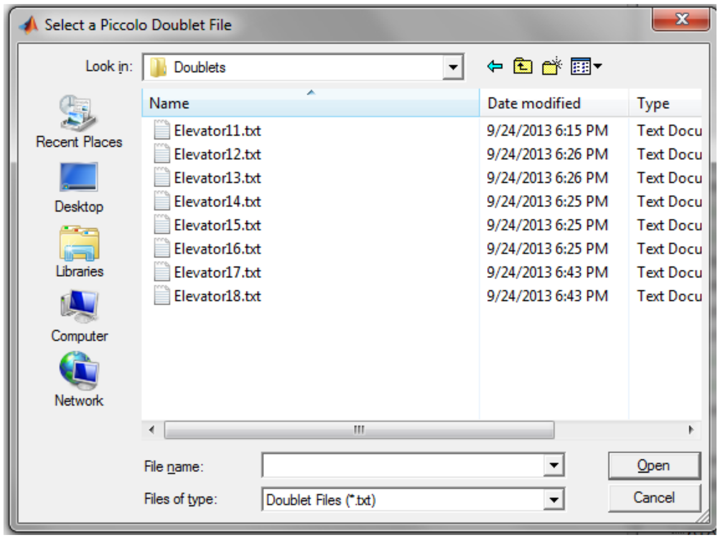
<Student Version> Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
EDU>> plotdoublet

Aircraft specific parameters:
Wing area =  1.27 m^2, Span =  4.007 m, Mass=  30.2 kg
Press enter to accept, or 1 to change:
```

Figure 251 PlotDoublet Aircraft Data Prompt

As prompted hit enter if they are correct or press 1 to change. Note that the mass is just an estimate and is used only in the elevator doublets. Since the elevator doublets are concerned with the change in C_L rather than its actual magnitude the exact mass of the vehicle at the time of the test does not need to be known since the doublet maneuver should not last longer than a matter of seconds at the most.

2) Select the appropriate doublet file as prompted by the popup window



Select a Piccolo Doublet File

Look in: Doublets

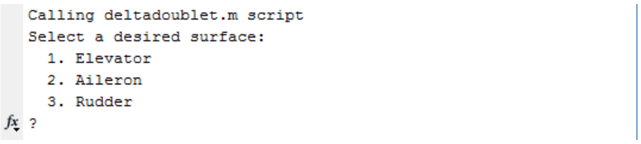
Name	Date modified	Type
Elevator11.txt	9/24/2013 6:15 PM	Text Docu
Elevator12.txt	9/24/2013 6:26 PM	Text Docu
Elevator13.txt	9/24/2013 6:26 PM	Text Docu
Elevator14.txt	9/24/2013 6:25 PM	Text Docu
Elevator15.txt	9/24/2013 6:25 PM	Text Docu
Elevator16.txt	9/24/2013 6:25 PM	Text Docu
Elevator17.txt	9/24/2013 6:43 PM	Text Docu
Elevator18.txt	9/24/2013 6:43 PM	Text Docu

File name:

Files of type: Doublet Files (*.txt)

Figure 252 PlotDoublet Doublet File Prompt

3)



```
Calling deltadoublet.m script
Select a desired surface:
  1. Elevator
  2. Aileron
  3. Rudder
fx ?
```

Figure 253 PlotDoublet Surface Selection

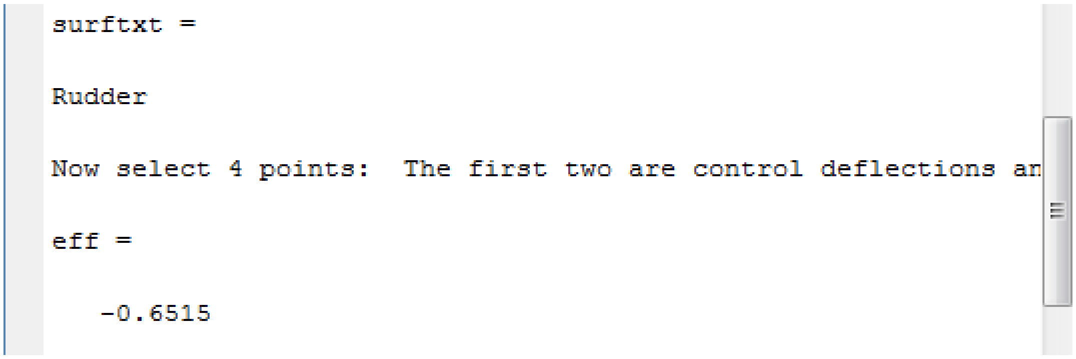
Before selecting the surface from the MATLAB command window prompt find the figure that contains the desired surface's deflection and response (Figure 17, 18, or 19). Check to see if the scale is appropriate for the system response. Zoom and pan as needed; however, make sure to constrain the zoom and pan to "vertical" zoom and pan. It is important that the x-axis for both subplots be in the exact same horizontal location and on the same scale. To zoom vertically click the zoom tool, then right click on the subplot, scroll down to "Zoom Options", and select "Vertical Zoom", and likewise for pan.

4) Check to make sure that the x – axes are aligned between the two subplots.

5) Type in the number of the surface that is to be analyzed. The appropriate figure will be selected automatically.

6) The mouse pointer will automatically turn into a cross hair cursor. Use the instructions in Section 6.4 Analyzing Doublet Maneuvers to select the appropriate points for analysis.

7) The calculated effectiveness will be displayed on the figure and also in the command window.



```
surftxt =

Rudder

Now select 4 points:  The first two are control deflections an

eff =

   -0.6515
```

Figure 254 PlotDoublet Solution Display

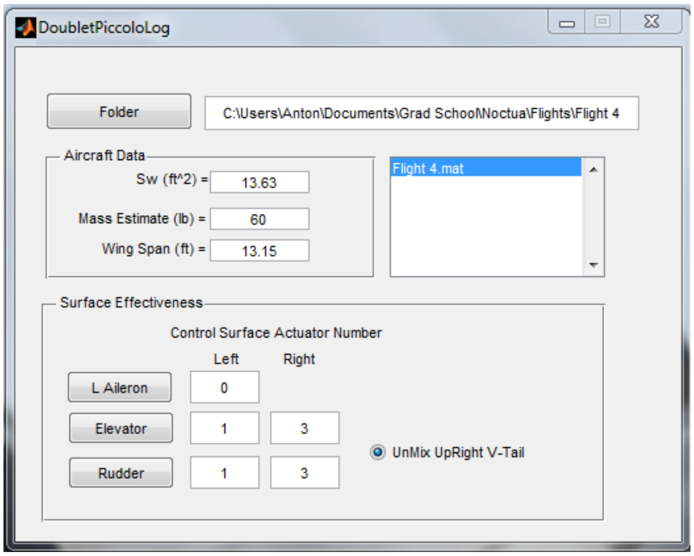
6.3. Piccolo Log File Method

The Piccolo Log File Method utilizes the piccolo telemetry log file via the DoubletPiccoloLog script to analyze doublet maneuvers. The telemetry rate of data for this method is a maximum of 25 Hz since it is dependent on the piccolo log file. Additionally any control surface mixtures that exist will have to be unmixed manually by editing the code for all mixtures except ruddervators (V-Tail configurations). The GUI associated with DoubletPiccoloLog has a built in function to automatically unmix ruddervators.

6.3.1. DoubletPiccoloLog

The DoubletPiccoloLog.m MATLAB script launches the corresponding GUI DoubletPiccoloLog.fig where together they make up a user friendly program that analyzes doublet maneuvers from the piccolo telemetry log file. DoubletPiccoloLog is based off of Cloud Cap's MATLAB scripts "doublet.m" and "deltadoublet.m". Lines of code from the two scripts were modified and integrated into DoubletPiccoloLog.

6.3.1.1. DoubletPiccoloLog Mechanics



C:\Users\Anton\Documents\Grad School\Noctua\Flights\Flight 4

Sw (ft^2) = 13.63

Mass Estimate (lb) = 60

Wing Span (ft) = 13.15

Flight 4.mat

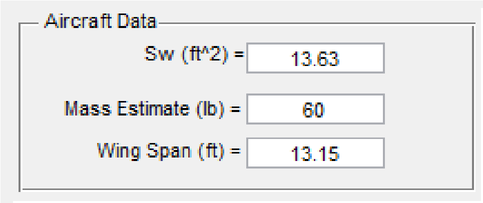
Control Surface Actuator Number

	Left	Right
L Aileron	0	
Elevator	1	3
Rudder	1	3

● UnMix UpRight V-Tail

Figure 255 DoubletPiccoloLog

Figure 255 shows the GUI for DoubletPiccoloLog. DoubletPiccoloLog loads piccolo mat files that are created by "CreatePiccoloMatFile.m". The piccolo mat files are used to access flight data from the flight telemetry log files. The list box displays the piccolo mat files that exist in the current directory. If a file name is selected in the list box the actual mat file itself does not load, it simply designates the name of the .mat file for the program to load later. In order to determine what files are actually piccolo mat files the GUI searches the directory for files with the extension ".mat". Then it looks into each ".mat" file for the variable "tClock". The current directory is displayed in the bar at the top of the GUI next to "Folder". The "Folder" button allows the user to change the directory. The list box will update automatically if the directory is changed.


Aircraft Data

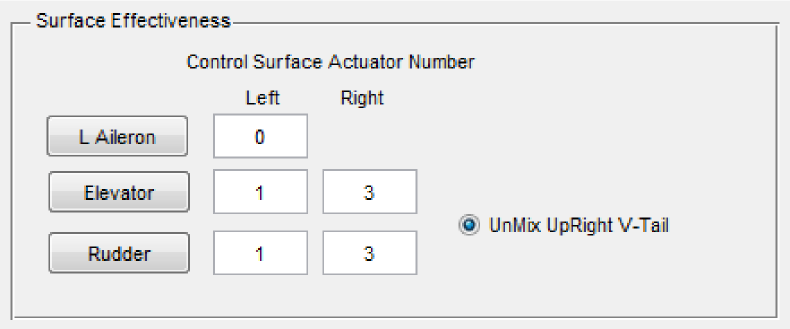
Sw (ft^2) = 13.63

Mass Estimate (lb) = 60

Wing Span (ft) = 13.15

Figure 256 DoubletPiccoloLog Aircraft Data Input

The "Aircraft Data" section takes inputs for the wing area, mass estimate, and wing span of the aircraft. Each time that a value is typed into the aircraft data input boxes the values will be saved in the GUI so that the user doesn't have to reenter them each time the GUI is launched. The wing area and mass estimate are required for elevator effectiveness calculations. The wing span is required for aileron effectiveness calculations. None of the aircraft data is required for rudder effectiveness. If any of the effectiveness analyses are ran the GUI will stop the script and warn the user if any of the required aircraft data parameters are missing.


Surface Effectiveness

Control Surface Actuator Number

	Left	Right
L Aileron	0	
Elevator	1	3
Rudder	1	3

● UnMix UpRight V-Tail

Figure 257 DoubletPiccoloLog Surface Effectiveness

In the "Surface Effectiveness" section the button "L Aileron" launches the aileron effectiveness program. The button "Elevator" launches the elevator effectiveness program. The button "Rudder" launches the rudder effectiveness program. Similarly to the aircraft data, each time that a value is typed into the input boxes the values will be saved in the GUI so that the user doesn't have to reenter them each time the GUI is launched.

The plotpiccolo mat file stores surface deflections as actuator deflections and they are labeled as "Surface#" where # is the actuator number (Surface0 Surface1 Surface2 etc.). In order for DoubletPiccoloLog to know which surface deflection to use the user has to input the actuator number that corresponds with the desired surface into the input boxes located next to the surface buttons. The doublet analysis only cares about control surface deflection commands because it needs to know what the commanded aileron, elevator, or rudder deflection was throughout the test. Even if there are mixed control surfaces, or multiple control surfaces, as long as there is one surface on the aircraft that is solely an aileron, elevator, or rudder then only that surface needs to be used.

The program has a built in function to un-mix up right ruddervators for elevator and rudder doublets. In these configurations each surface deflection is a mixture of the rudder and elevator deflection commands. The ruddervator function requires both the left and right surface actuator numbers so that it can properly un-mix them. The program un-mixes the ruddervators according to their definition. Ruddervators are classified as "elevator + rudder". Ruddervators sign convention is defined by pitch where pitch up is negative ruddervator deflection and pitch down is positive ruddervator deflection (just like elevator deflection sign convention) (PccUsersGuide pg. 103). Rudder deflection is defined as positive when the trailing edge is deflected starboard (PccUsersGuide pg. 102); therefore, a positive rudder deflection command will deflect the left ruddervator surface negative.

$$LRuddervator = Elevator - Rudder$$

Equation 71 Left Ruddervator Control Definition

$$RRuddervator = Elevator + Rudder$$

Equation 72 Right Ruddervator Control Definition

Setting both Equation 71 and Equation 72 equal to Elevator and combining them leads to the following:

$$Elevator = LRuddervator + Rudder \qquad Elevator = RRuddervator - Rudder$$

$$LRuddervator + Rudder = RRuddervator - Rudder$$

$$2Rudder = \ -LRuddervator + RRuddervator$$

$$Rudder = \ \frac{-LRuddervator + RRuddervator}{2}$$

Equation 73 Ruddervator Rudder Deflection

Similarly setting both Equation 71 and Equation 72 equal to Rudder and combining them leads to the following:

$$Rudder = Elevator - LRuddervator \qquad Rudder = RRuddervator - Elevator$$

$$Elevator - \ LRuddervator = RRuddervator - Elevator$$

$$2Elevator = \ LRuddervator + RRuddervator$$

$$Elevator = \ \frac{LRuddervator + RRuddervator}{2}$$

Equation 74 Ruddervator Elevator Deflection

The program uses Equation 73 and Equation 74 to calculate the elevator and rudder deflection commands from the recorded ruddervator deflections.

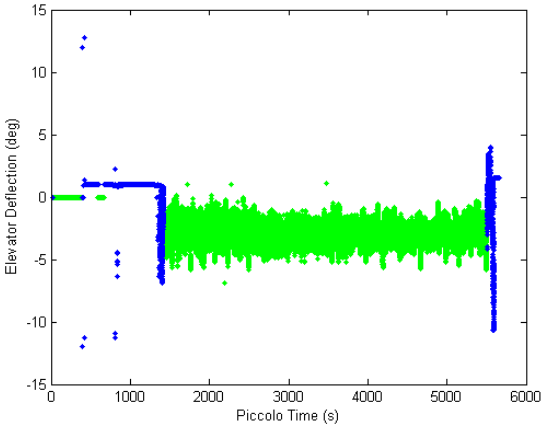


Figure 258 DoubletPiccoloLog Deflection Plot

When one of the control surface buttons is pressed the program will plot the surface deflection versus time. When this happens the code is still running but it is paused and waiting for the user to hit enter to continue. The down time is designed for the user to be able to zoom and pan the figure to find the desired doublet deflection. The green data denotes that the autopilot is in control and the blue data denotes that the manual pilot is in control. Instructions for this step are displayed in the command window.

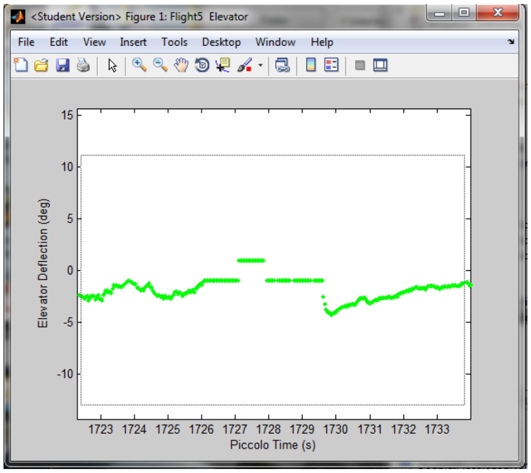


Figure 259 DoubletPiccoloLog Time Period Selection

After the user hits enter the user will be able to select the time period for analysis using the plot. The user will be able to click, hold the click, and drag across the plot to select the appropriate area. The click and drag method code came directly from Cloud Cap's "doublet.m" script.

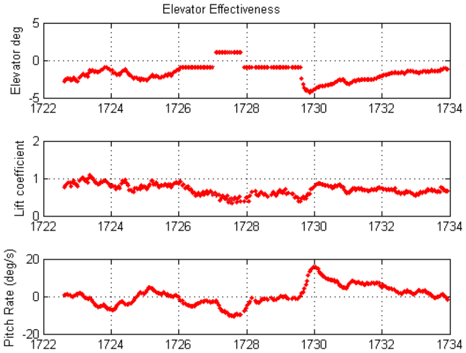


Figure 260 DoubletPiccoloLog Effectiveness Plot

After the area is selected the program will plot the surface deflection along with its appropriate system response. This portion of the process is exactly the same as the doublet file method and was copied from Cloud Cap's script "deltadoublet.m" with some modifications. The original detladoubet code calculated effectiveness the same for all three different effectiveness calculations. In DoubletPiccoloLog the rudder effectiveness deltadoublet section was modified so that the rudder effectiveness calculations would calculate change in yaw angle from the change in yaw rate multiplied by the change in time. Cloud Cap's "doublet.m" script will incorrectly calculate the rudder effectiveness from the change in yaw rate alone.

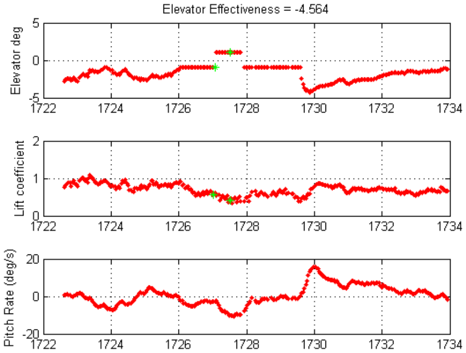


Figure 261 DoubletPiccoloLog Effectiveness Points

From here the user will be asked to select 4 points on the figure. The first two points represent the surface deflections, and the second two points represent the values of the system response. The next section issues instructions on how to select the four points. After all four points have been selected the program will calculate the surface effectiveness as the change in deflection, from the first two points, divided by the change in system response from the second two points. The solution will be displayed in the command window and in the title of the figure, as it is in Figure 261 above.

All of the data that the program analyzes comes from the structure labeled “dat” in the piccolo mat file. The time scale used is based on the piccolo time in seconds via “dat.Clock/1000”. The program calculates C_L using the same equation as plotdoublet, Equation 68, with the following GUI inputs and dat variables:

$$Sw = GUI\ Input \quad Zaccel = dat.Zaccel \quad q = dat.Dynamic$$

$$m = GUI\ Input$$

The program calculates the dimensionless roll rate using the same equation as plotdoublet, Equation 70, with the following GUI input and dat variables:

$$b = GUI\ Input \quad TAS = dat.TAS \quad Roll\ Rate = dat.p$$

Cloud cap's "doublet.m" program, which was designed to use the piccolo telemetry log file for doublet maneuver analysis, calculates the rudder effectiveness system response as the change in yaw rate. The problem is the rudder effectiveness is the change in sideslip, not the change in yaw rate. plotpiccolo calculates this number as the change in heading. When the rudder effectiveness calculation was integrated into DoubletPiccoloLog the rudder effectiveness system response calculation was changed. Instead of using the change in yaw rate the calculation will multiply the change in yaw rate by the change in time between the two selected points to back out the change in yaw.

$$\Delta r = \frac{\Delta Yaw}{\Delta t}$$

$$\Rightarrow \Delta r * \Delta t = \Delta Yaw$$

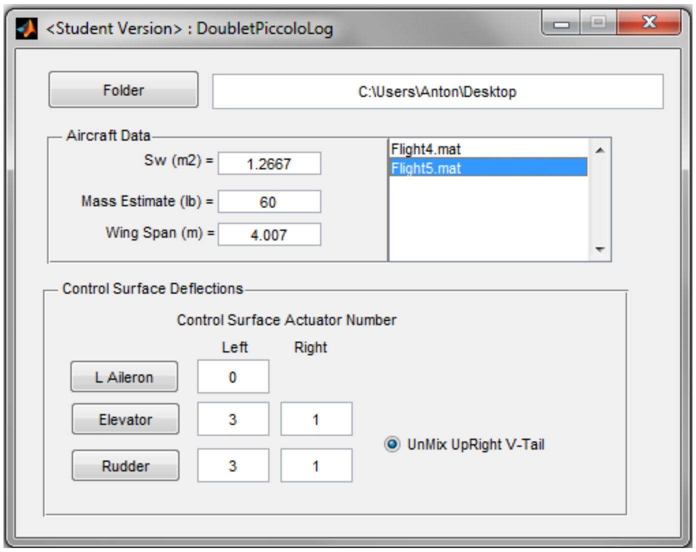
Equation 75 DoubletPiccoloLog Yaw Calculation

6.3.1.2. DoubletPiccoloLog Instructions

The following steps detail how to use DoubletPiccoloLog with doublet maneuvers. To begin open MATLAB and set the directory to the "MatLabPrograms" folder in the aircraft's folder. Launch "DoubletPiccoloLog" by typing "DoubletPiccoloLog" into the command line and follow the following steps:

1) If the plotpiccolo mat file is not located in the current directory click "Folder" to change the directory.

2) If there are multiple plotpiccolo mat files in the directory make sure the proper file is selected in the list box.



<Student Version> : DoubletPiccoloLog

C:\Users\Anton\Desktop

Aircraft Data

Sw (m2) = 1.2667

Mass Estimate (lb) = 60

Wing Span (m) = 4.007

Flight4.mat
Flight5.mat

Control Surface Deflections

Control Surface Actuator Number

	Left	Right
L Aileron	0	
Elevator	3	1
Rudder	3	1

● UnMix UpRight V-Tail

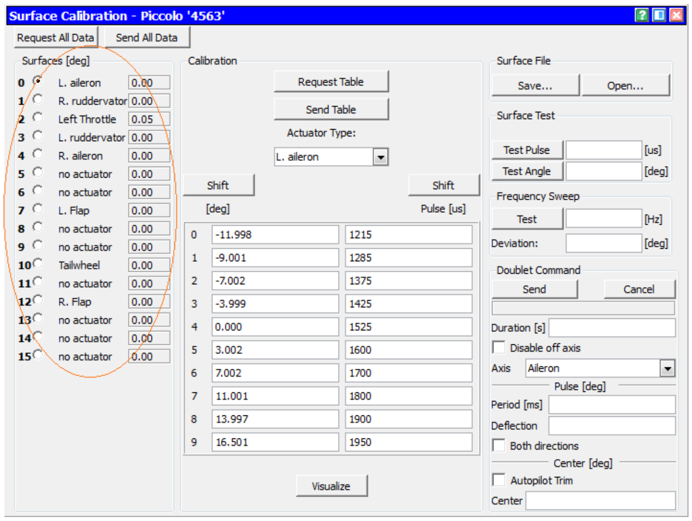
Figure 262 DoubletPiccoloLog Piccolo Mat File Selection

3) Set the following aircraft data as applies:

   a. Aileron Doublets input the wing span in meters

   b. Elevator Doublets input the wing area in meters squared and the mass estimate in pounds. The elevator doublets are only concerned with the change in C_L rather than its actual magnitude; therefore, the exact mass of the vehicle at the time of the test does not need to be known since the doublet maneuver should not last longer than a matter of seconds at the most.

4) Input the proper actuator number into the input box of the surface that was tested. The actuator numbers are the servo numbers that the surfaces were assigned in the “Surface Calibration” window of PCC.



Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	●	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.05
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File

Surface Test

Test Pulse [us]

Test Angle [deg]

Frequency Sweep

Test [Hz]

Deviation: [deg]

Doublet Command

Duration [s]

☐ Disable off axis

Axis Aileron

Pulse [deg]

Period [ms]

Deflection

☐ Both directions

Center [deg]

☐ Autopilot Trim

Center

Figure 263 DoubletPiccoloLog Control Surface Numbers

If the tail configuration is not a V-Tail then only the left control surface actuator number is needed. If there are multiple actuators assigned as elevators or rudders only one of them is needed for each.

5) If the doublet is an elevator or rudder doublet and the tail configuration is an upright V-Tail then click the radio button next to “UnMix UpRight V-Tail”, otherwise make sure that the radio button is not selected.

6) Click “L Aileron” to evaluate aileron doublets. Click “Elevator” to evaluate elevator doublets. Click “Rudder” to evaluate rudder doublets.

7) Use the zoom tool (horizontal and vertical zoom) to find the appropriate period of deflection.

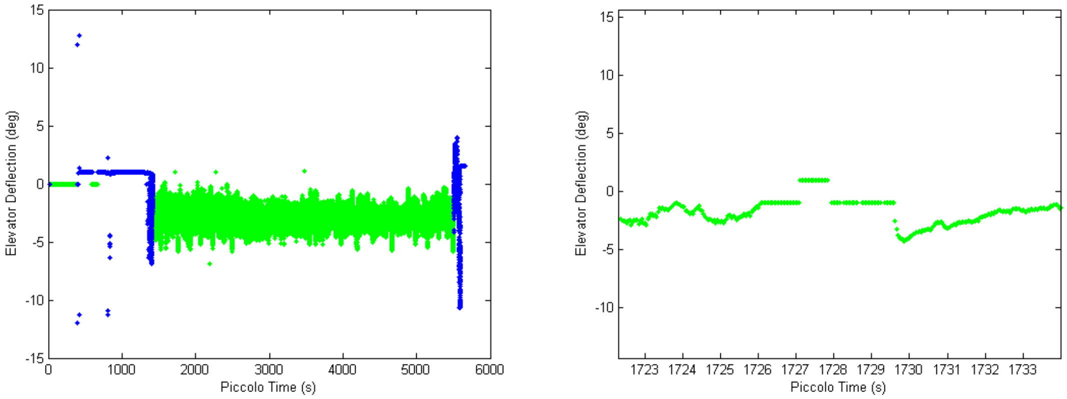


Figure 264 DoubletPiccoloLog Deflection Plot Example

8) With the figure selected press enter.

9) Hold down the left click in the far left portion of the plot and drag the mouse across to the end of the plot.

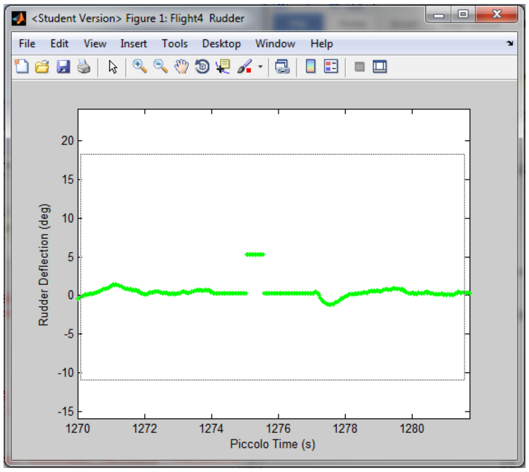


Figure 265 DoubletPiccoloLog Time Period Selection Example

10) The corresponding surface effectiveness plot should appear. Use the instructions in Section 6.4 368to select the appropriate points for analysis.

11) The solution should be printed on the figure (doesn't always work), and in the command window.

6.4. Analyzing Doublet Maneuvers

Regardless of which method and MATLAB scripts are used to analyze the results of doublet maneuvers they both end up with the same user input format to calculate the results. At the end of both MATLAB scripts the user will be required to click four points on a system response plot. System response plots plot two subplots where the top subplot is the surface deflection of the surface that was tested and the bottom subplot is the system response that corresponds to the surface that was deflected. The first two points that the user clicks will be the surface deflections points. The analyses calculate the change in surface deflection as the y value of the second point minus the y values of the first point. The last two points that the user clicks will be the system response points in the bottom subplot. The analyses calculate the change in system response as the y value of the second point minus the y value of the first point (in all cases except for rudder doublets using the Log File Doublet Method which is explained later).

Logistically speaking before the user can begin selecting points the appropriate plots will already exist and the user will be prompted to begin by pressing enter on the keyboard. This period of time should be used to zoom and pan on the subplots so that the system response is in full view. If the user chooses to zoom DO NOT use regular zoom. It is important to keep the x axes of the subplots on the same scale and locations in the figure. In MATLAB zoom, and pan, can both be constrained to “Vertical Zoom” or “Vertical Pan”. Use vertical zoom and vertical pan to adjust the system response subplots.

Properly selecting the first two points is generally simple. The analysis is supposed to capture the change in surface deflection. Properly selecting the second two points can be a little difficult. Cloud Cap’s guidelines simply suggest that the user select system response points that directly correlate with the time (x axis) of the deflection points; however, those directions would only work if the period of the test was just right. If the period of the doublet maneuver is not long enough for the system to fully respond the solution will not be good as the effect of the deflection did not have time to fully respond. If the period of the doublet maneuver is too long and the user had selected the last data point of the surface deflection as the 2nd point then the corresponding 4th point would be too late on the system response plot.

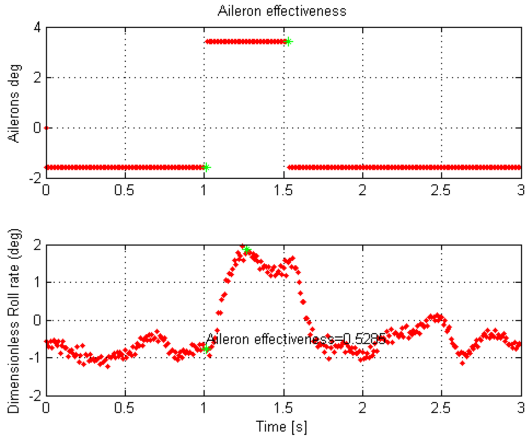


Figure 266 Aileron Effectiveness Plot

Figure 266 is an example of a system response plot of an aileron doublet maneuver. The first two points were selected so that the script would calculate the correct aileron deflection of 4 degrees. The third point corresponded with the first point as it should be expected that once the deflection was commanded the roll rate would begin to change. Instead of selecting the fourth point to correlate with the second point at just after 1.5 seconds the fourth point was selected at the peak of the dimensionless roll rate. If the fourth point had been selected to line up with the third point on the x axis the calculated change in dimensionless roll rate would have been lower, and thus the calculated aileron effectiveness would have been lower.

Generally speaking the end result that the doublet maneuvers should convey is the immediate effect that a control surface deflection will have on the angular rates of the aircraft. Recall that aileron, elevator, and rudder effectiveness’s are used to scale the feedback gains; and thus, scale the commanded aileron, elevator, and rudder deflections in response to a given error in the system. Essentially all three of the doublet maneuvers system responses are meant to indicate some type of moment that is induced on the aircraft by a surface deflection.

The idea is that the appropriate angular rates should indicate the time span to analyze the system responses. As an angular rate begins to increase after a doublet deflection should indicate where to place the 3rd point. As an angular rate peaks and then begins to decrease or flat line should indicate where to place the 4th point. Originally only one doublet analysis, aileron doublets, would correctly use an angular rate, roll rate, to determine the range of system response for calculating effectiveness; however this created some problems. Without observing the angular rates the results would be impacted by how long the periods of the deflections were. Essentially the user could force the effectiveness calculation to yield whatever he/she would please.

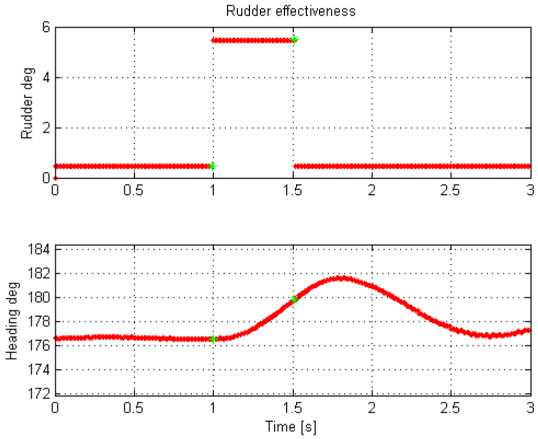


Figure 267 Rudder Effectiveness Plot Original Code

Figure 267 is an example of the original plotdoublet analysis of rudder doublets. The change in heading continued well beyond the period of the deflection. The heading peaked after the rudder deflection had returned to the trim deflection. The rudder effectiveness calculation essentially depended heavily on the length of the period of the rudder deflections. The calculated rudder effectiveness value would literally increase and decrease just by varying the length of the period.

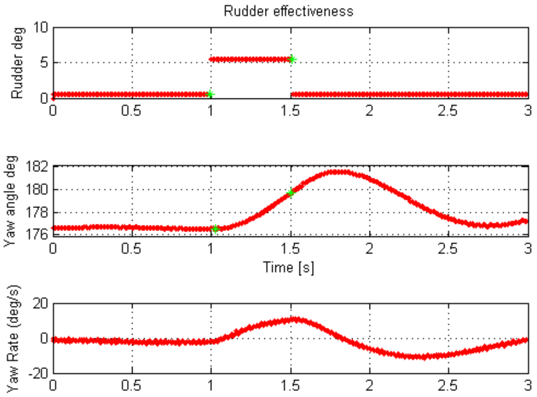


Figure 268 Rudder Effectiveness Plot New Code

Figure 268 is an example of the same rudder doublet via plotdoublet. The yaw rate showed the change in yaw rate began to decrease just before the period of the doublet was complete. Since the change in yaw rate had begun to decrease it could be said with certainty that the period of the rudder deflection was not too short and the 4th point accurately represented the end of the range of change in yaw angle.

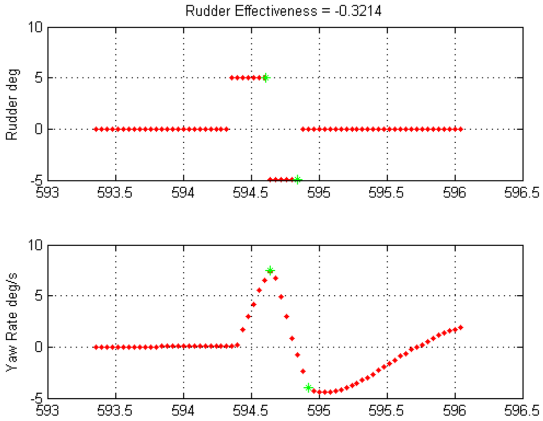


Figure 269 Rudder Doublet Period Too Short

Figure 269 depicts an example of a rudder doublet using the Doublet File Method. In the doublet test the period was too short. This can be seen because the yaw rate did not actually peak and begin to flatten. After the first deflection the yaw rate stopped its rate of increase and decreased rapidly due to the change in rudder deflection; therefore, it did not have enough time to peak on its own. The calculated rudder effectiveness in the example would have been too small.

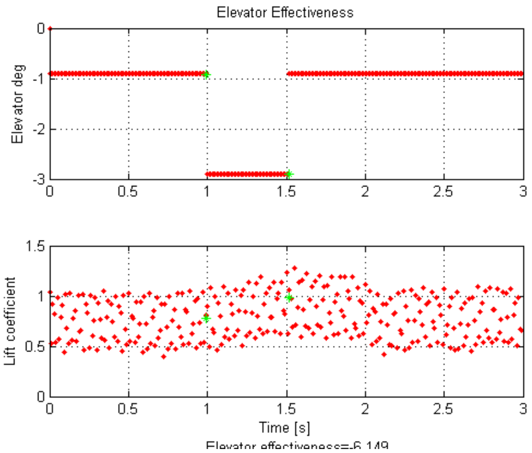


Figure 270 Noisy Doublet Data

Figure 270 is an example of an elevator doublet that had too much noise in the CL measurements. The user will have to discern when a test has too much noise in the measured system response to be accurately analyzed. Typically elevator doublets are the most prone to vibrations as the C_L calculations are based on the z accelerometer measurements. If vibrations are a problem try the following suggestions:

1) Conduct the maneuvers at different throttle settings (by changing the airspeed command)
2) Remount the autopilot to reduce vibrations
3) Try the doublet tests on a different day. Turbulent atmospheric conditions could be a cause to the vibrations. Sometimes vibrations can occur during one flight test and not occur during the next without any changes being made.

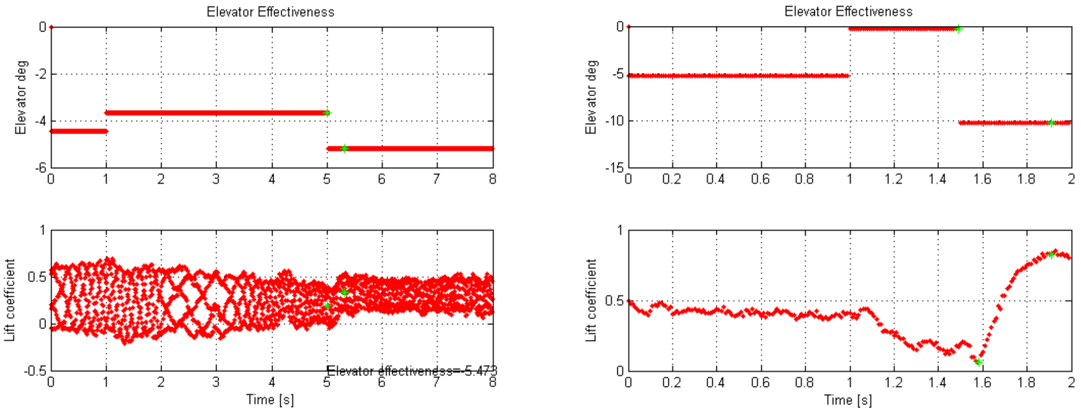


Figure 271 Nexstar Doublet Noise Example

Figure 271 depicts Nexstar elevator doublets 1 and 8. They were conducted during different flights. Elevator doublet 1 had significant amounts of noise in the CL measurements, while elevator doublet 8 did not; however, no changes were made to the aircraft or the autopilot’s physical configuration.

6.4.1.1. Analyzing Aileron Doublet Maneuvers

One Deflection:

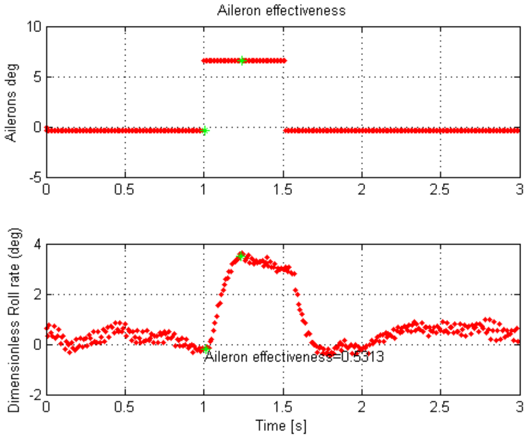


Figure 272 Aileron Single Deflection Doublet Maneuver Analysis

1) Select the 1st point to represent the initial aileron deflection.

2) Select the 2nd point to represent the aileron doublet deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 3rd point to represent the initial dimensionless roll rate when the deflection change was commanded, just before the roll rate begins to increase in magnitude. Ideally the 3rd point should line up with the 1st point in the x axis; however, sometimes responses are delayed.

4) Select the 4th point to be just at or slightly after the dimensionless roll rate has peaked. If the peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak then the period was too short and the solution should not be used as well.

Two Deflections:

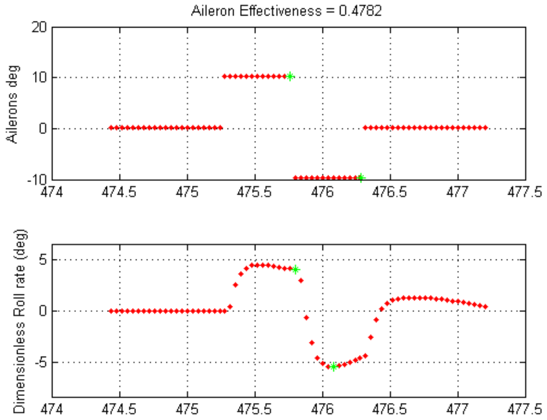


Figure 273 Aileron Dual Deflection Doublet Maneuver Analysis

1) If it is desired to analyze each deflection separately, as if there were only one deflection, use the steps for one deflection.

2) Select the 1st point to represent the first aileron deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 2nd point to represent the second aileron deflection. The point can lie anywhere on the surface deflection in the x axis.

4) Select the 3rd point to represent the dimensionless roll rate just after the second deflection is commanded, and when the dimensionless roll rate begins to rapidly increase in magnitude. Do not select the 3rd point to be the first peak of the dimensionless roll rate unless it happens to be the same time that the period of the first aileron deflection ends and the second aileron deflection period begin. If the dimensionless roll rate did not peak during the period of the first deflection then the period was too short and the solution should not be used.

5) Select the 4th point to represent the dimensionless roll rate just at or slightly after the dimensionless roll rate peak. If the peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak then the period was too short and the solution should not be used as well.

6.4.1.2. Analyzing Elevator Doublet Maneuvers

One Deflection:

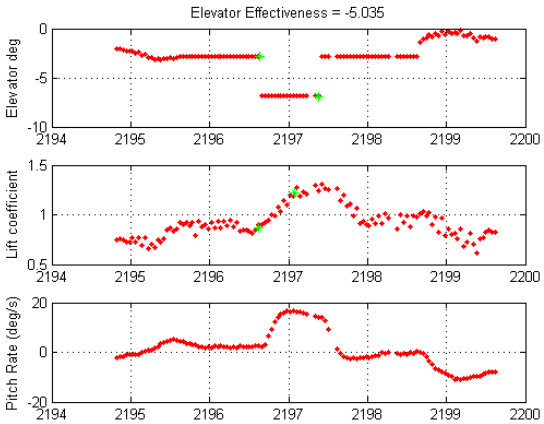


Figure 274 Elevator Single Deflection Doublet Maneuver Analysis

1) Select the 1st point to represent the initial elevator deflection.

2) Select the 2nd point to represent the elevator doublet deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 3rd point to represent the initial C_L when the deflection change was commanded, just as the pitch rate begins to increase in magnitude. Typically there will be a small delay between the time that the elevator deflection was commanded and the time that the pitch rate actually begins to change.

4) Select the 4th point to represent the C_L slightly after the pitch rate has peaked. If the pitch rate peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak in the pitch rate then the period was too short and the solution should not be used as well.

Two Deflections:

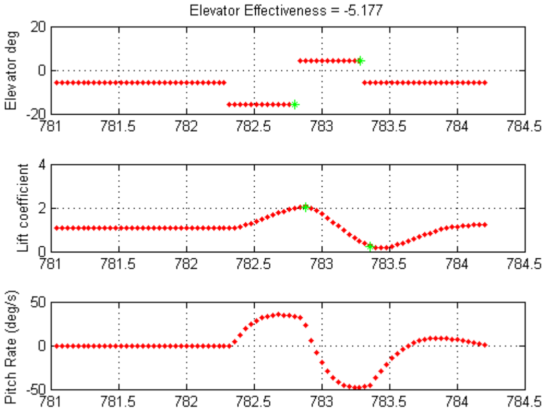


Figure 275 Elevator Dual Deflection Doublet Maneuver Analysis

1) If it is desired to analyze each deflection separately, as if there were only one deflection, use the steps for one deflection.

2) Select the 1st point to represent the first elevator deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 2nd point to represent the second elevator deflection. The point can lie anywhere on the surface deflection in the x axis.

4) Select the 3rd point to represent CL just after the second deflection is commanded, and when the pitch rate begins to rapidly increase in magnitude. If the pitch rate did not peak during the period of the first deflection then the period was too short and the solution should not be used.

5) Select the 4th point to represent CL after the pitch rate has peaked and has started to decrease in magnitude. If the peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak then the period was too short and the solution should not be used as well.

6.4.1.3. Analyzing Rudder Doublet Maneuvers

One Deflection:

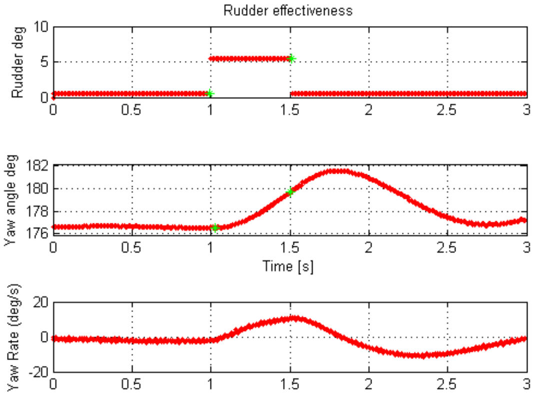


Figure 276 Rudder Single Deflection Piccolo Log File Method Doublet Analysis

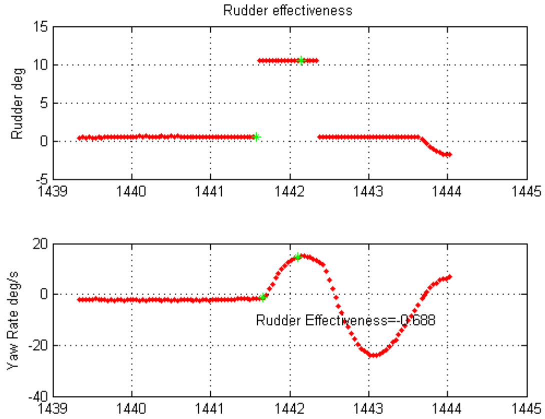


Figure 277 Rudder Single Deflection Doublet File Method Doublet Analysis

1) Select the 1st point to represent the initial rudder deflection.

2) Select the 2nd point to represent the rudder doublet deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 3rd point to represent the initial yaw angle (Doublet File Method), or yaw rate (Piccolo Log File Method), when the deflection change was commanded, just as the yaw rate begins to increase in magnitude.

4) Select the 4th point to represent the yaw angle (Doublet File Method), or yaw rate (Piccolo Log File Method), at the time just after the yaw rate peaked. If the yaw rate peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak in the yaw rate then the period was too short and the solution should not be used as well.

Two Deflections:

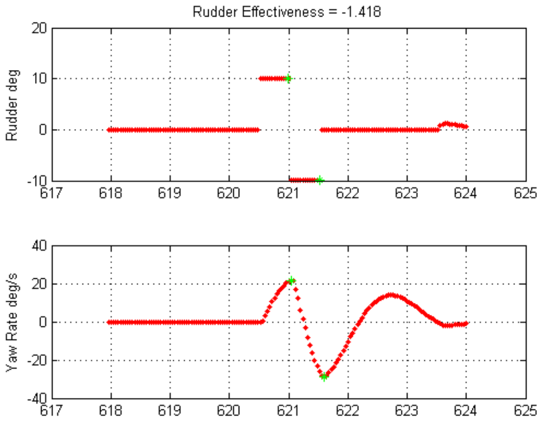


Figure 278 Rudder Dual Deflection Doublet Analysis

1) If it is desired to analyze each deflection separately, as if there were only one deflection, use the steps for one deflection.

2) Select the 1st point to represent the first rudder deflection. The point can lie anywhere on the surface deflection in the x axis.

3) Select the 2nd point to represent the second rudder deflection. The point can lie anywhere on the surface deflection in the x axis.

4) Select the 3rd point to represent the yaw angle (Doublet File Method), or yaw rate (Piccolo Log File Method), just after the second deflection is commanded, and when the yaw rate begins to rapidly increase in magnitude. If the yaw rate did not peak during the period of the first deflection then the period was too short and the solution should not be used.

5) Select the 4th point to represent the yaw angle (plotdoublet), or yaw rate (DoubletPiccoloLog), at the time that the yaw rate peaks. If the peak occurred at or after the end of the doublet deflection then the period was too short and the solution should not be used. Additionally if there was no peak then the period was too short and the solution should not be used as well.

CHAPTER VII

FLIGHT ANALYSIS TOOLS

7. Introduction

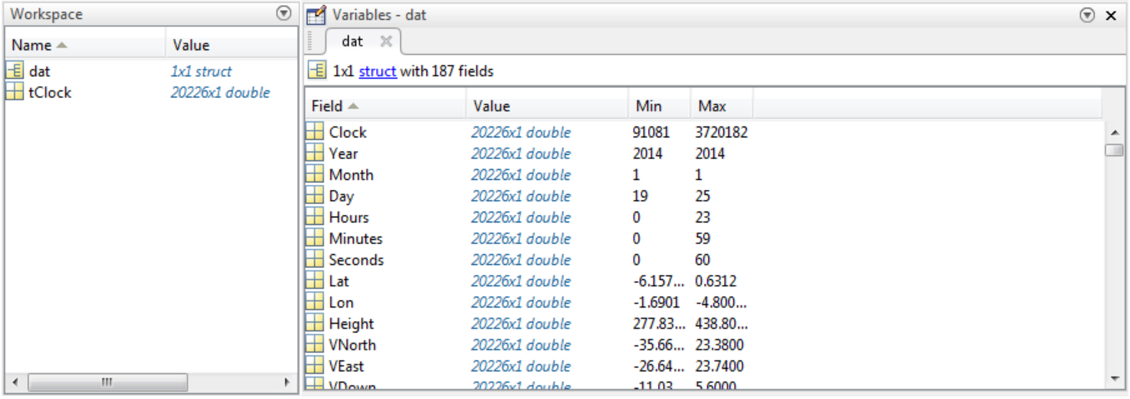
In support of analyzing flights and gain tuning methods MATLAB GUI’s (graphical user interfaces) were created. ‘AnalyzePiccolo’ and ‘AnalyzeDevInterface’ are designed for analyzing telemetry data. ‘AltitudeControl’, ‘LateralTracking’, ‘AirspeedControl’, and ‘EnergyControl’ are designed to help assess control loop performance. Each GUI will require some type of MATLAB workspace file, or “mat” file, to load. Most GUI’s rely on the Dev mat file which is created from AnalyzeDevInterface. Some rely on the piccolo mat file which is created by ‘CreatePiccoloMatFile’. Each section will specify what the GUI requires.

‘FuelCorrection’, ‘CreatePiccoloMatFile’, and ‘PiccoloRPMRateFilter’ are MATLAB scripts designed to perform specific functions. ‘FuelCorrection’ will calculate the engine specific fuel consumption for a flight and re – calculate the fuel mass estimate based on the calculated ESFC and final weight. ‘PiccoloRPMRateFilter’ will filter the RPM data based on user input and with the same logic as the piccolo’s rpm filter. This can be used to determine if and at what setting the piccolo rpm filter can be effectively used. ‘CreatePiccoloMatFile’ incorporates Cloud Cap’s ‘loadpiccolologfil’ MATLAB script with some custom variables to create piccolo mat files. Piccolo mat files contain a workspace structure with all of the piccolo telemetry data and is used in all of the GUI’s and scripts except for ‘AnalyzeDevInterface’. Each GUI has a corresponding script that is named the same and is attached to the GUI. The code for all of the scripts is located in Appendix E.

7.1. CreatePiccoloMatFile

‘CreatePiccoloMatFile.m’ is a MATLAB script that is designed to create variables from piccolo telemetry log file data and save those variables as a ‘.mat’ file. The ‘.mat’ file that this script generates is used by the AnalyzePiccolo, ‘AnalyzeDevInterface’, ‘DoubletPiccoloLog’, ‘LateralTracking’, ‘EnergyControl’, and ‘AltitudeControl’ GUIs. The code for ‘CreatePiccoloMatFile’ is located in Appendix E.

CreatePiccoloMatFile begins with a dialog box for the user to select the piccolo telemetry log file to load. The log file is loaded with Cloud Cap’s script ‘loadpiccolologfile.m’. Loadpiccolologfile assigns each column of data in the log file to a variable, and names the variables according to their column headers. The output of loadpiccolologfile is stored in a structure named ‘dat’. After dat has been created CreatePiccoloMatFile will create a time variable ‘tClock’. tClock represents the change in time of each row of data where the first row is time zero. One of the variables in the dat structure is ‘Clock’. Clock is a record of the piccolo time, measured in milliseconds, where the Clock begins at 0 as soon as the piccolo is turned on. ‘tClock’ treats the first recorded piccolo time as time 0 and the rest of the time data as a delta t since time 0.



Name	Value
dat	1x1 struct
tClock	20226x1 double

1x1 struct with 187 fields

Field	Value	Min	Max
Clock	20226x1 double	91081	3720182
Year	20226x1 double	2014	2014
Month	20226x1 double	1	1
Day	20226x1 double	19	25
Hours	20226x1 double	0	23
Minutes	20226x1 double	0	59
Seconds	20226x1 double	0	60
Lat	20226x1 double	-6.157...	0.6312
Lon	20226x1 double	-1.6901	-4.800...
Height	20226x1 double	277.83...	438.80...
VNorth	20226x1 double	-35.66...	23.3800
VEast	20226x1 double	-26.64...	23.7400
VDown	20226x1 double	-11.03	5.6000

Figure 279 Piccolo Mat File Variables

After tClock has been created CreatePiccoloMatFile will save the workspace variables ‘dat’ and ‘tClock’ to a ‘.mat’ file in the same folder that the selected telemetry log file is located. Figure 279 shows the contents of the resulting mat file, and a preview of the contents of the dat structure.

The following steps outline how to use CreatePiccoloMatFile.

1) Open MATLAB

2) Set the directory to the folder where the script is located.

3) Launch the script by right clicking the script in the Current Folder window and clicking run or type ‘CreatePiccoloMatFile’ in the command line.

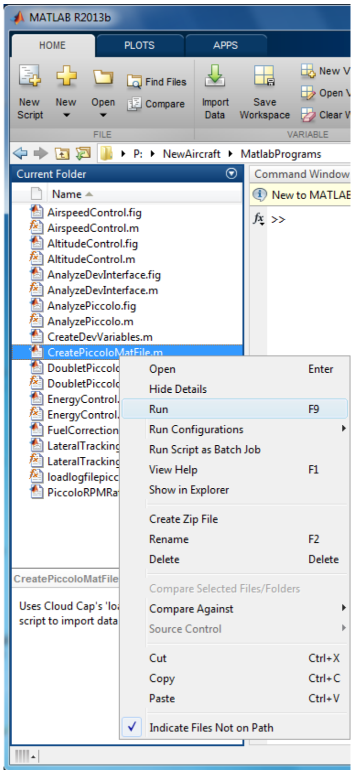


Figure 280 Launch CreatePiccoloMatFile

4) A dialog box will pop up asking for the piccolo telemetry log file. Locate the log file and select it.

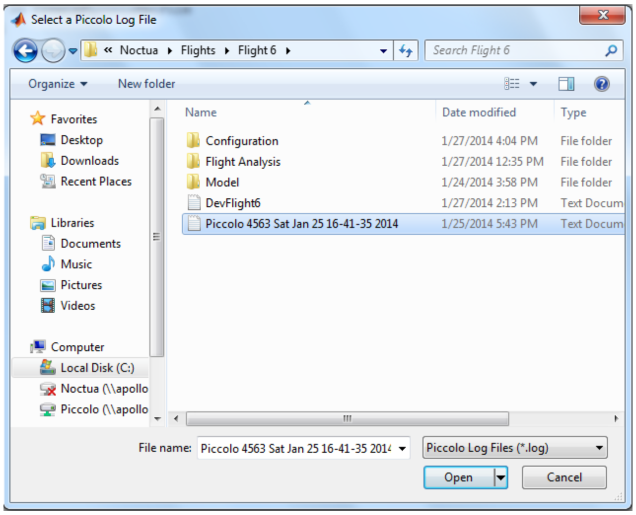
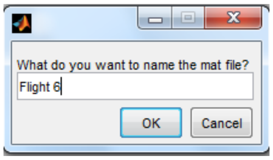

Select a Piccolo Log File
« Noctua ▸ Flights ▸ Flight 6 ▸
Search Flight 6
Organize
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Noctua (\\apollo
Piccolo (\\apollo
Name
Date modified
Type
Configuration 1/27/2014 4:04 PM File folder
Flight Analysis 1/27/2014 12:35 PM File folder
Model 1/24/2014 3:58 PM File folder
DevFlight6 1/27/2014 2:13 PM Text Docum
Piccolo 4563 Sat Jan 25 16-41-35 2014 1/25/2014 5:43 PM Text Docum
File name: Piccolo 4563 Sat Jan 25 16-41-35 2014
Piccolo Log Files (*.log)
Open
Cancel

Figure 281 Load Piccolo Log File

5) Enter a name for the mat file in the input dialog box that pops up after the script has completed importing all of the data. It may take some time for the script to load all of the data depending on how fast the computer is and how large the log file is.



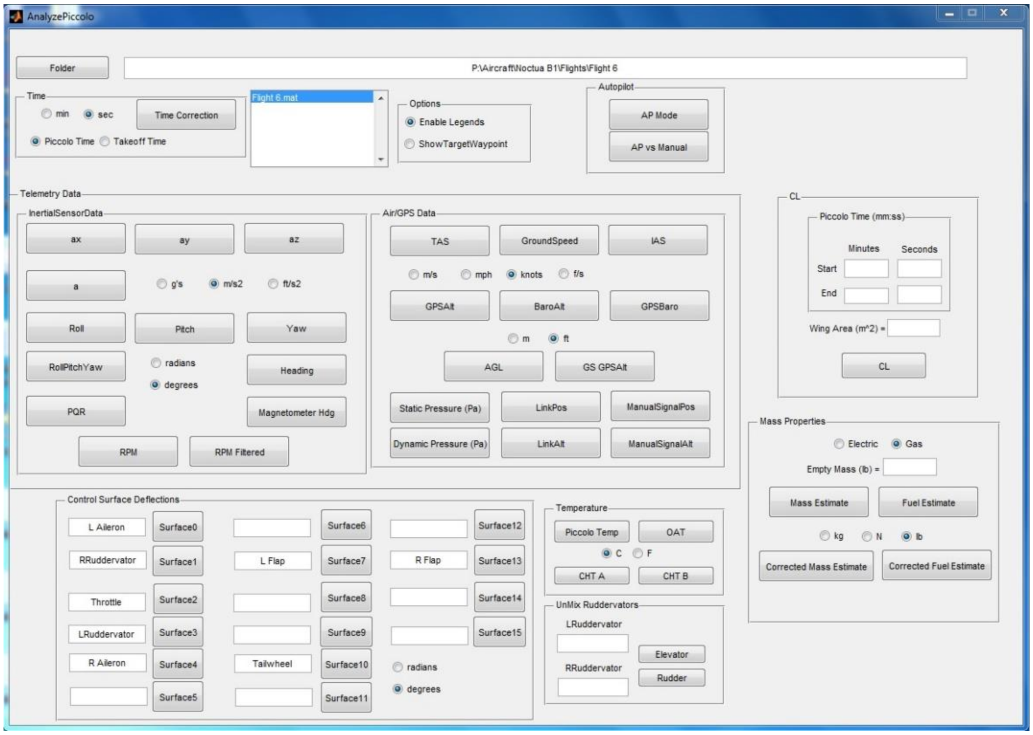
What do you want to name the mat file?

Flight 6

Figure 282 Name Piccolo Mat File

7.2. AnalyzePiccolo

‘AnalyzePiccolo.m’ is a MATLAB script that launches a figure that acts as a GUI, ‘AnalyzePiccolo.fig’. AnalyzePiccolo is designed to provide the user a user friendly way of plotting various telemetry data that is recorded in the piccolo telemetry log files.

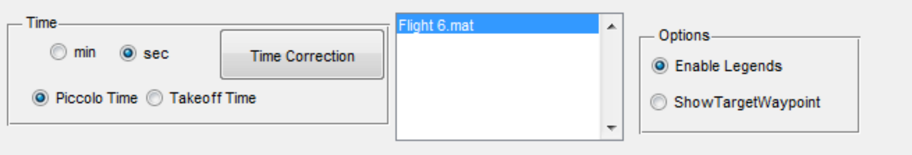


P:\Aircraft\Noctua B1\Flights\Flight 6

Flight 6.mat

Figure 283 AnalyzePiccolo GUI

AnalyzePiccolo loads telemetry data from the piccolo mat files generated by ‘CreatePiccoloMatFile’. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The “Folder” button will pull up a dialog box for the user to select another folder as the directory.



Time

○ min ● sec

● Piccolo Time ○ Takeoff Time

Flight 6.mat

Options

● Enable Legends

○ ShowTargetWaypoint

Figure 284 Analyze Piccolo Time & Mat File Options

Just below the directory there is a section for time options, a list box, and a section for options. Minutes and seconds are the available units of time that the user can select from. “Piccolo Time” and “Takeoff Time” determine what time scale to use for the plots. Piccolo Time goes by the clock on the piccolo which begins at 0 when the piccolo is turned on. The variable for piccolo time is recorded in the piccolo mat file as “dat.Clock” in milliseconds. Takeoff time allows the user to set time 0 as the time when takeoff occurred. The GUI determines the takeoff time by asking the user to input the takeoff time. The GUI creates two plots to help aid the user figure out when the takeoff time was.

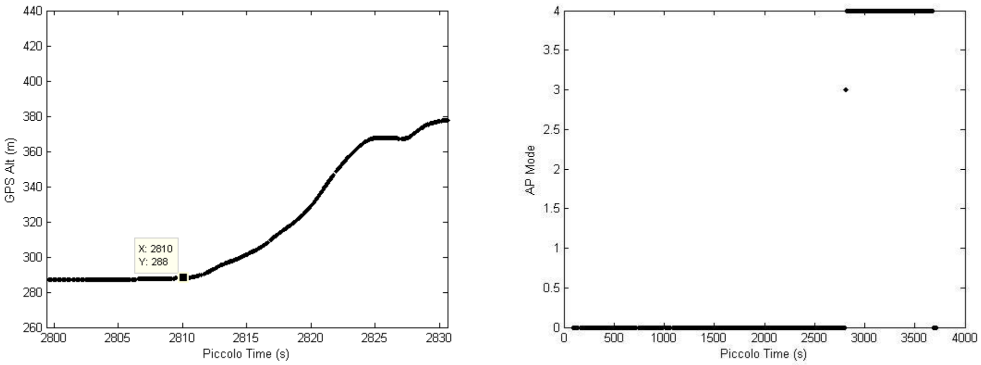


Figure 285 Takeoff Time Plots

Figure 285 is an example of the plots that will show up. One plot is the GPS Altitude plot and the other plot is the Autopilot Mode plot. The Autopilot Mode will switch to AP Mode 3 or 4 immediately during manual takeoffs when the autopilot senses it has lifted off. The takeoff time is to be input into the command window prompt. Figure 286 below shows an example of a barometer altitude plot on two different time scales.

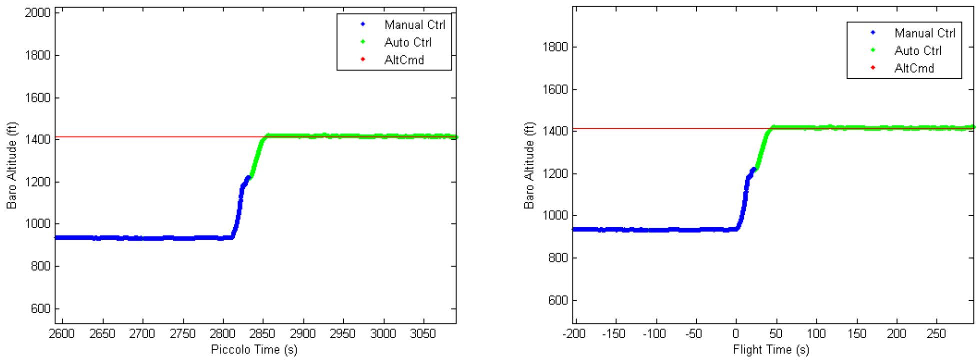


Figure 286 Baro Altitude Takeoff Time Example

The “Time Correction” button launches a function that will correct the time recordings for piccolo resets. If the piccolo is powered off and on in the same PCC recording session the time stamp will reset on the piccolo clock and there will be overlapping data when telemetry is plotted versus time.

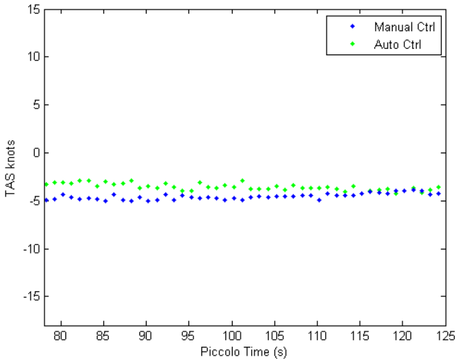
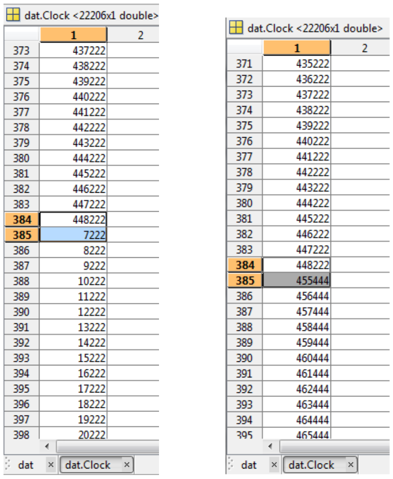


Figure 287 Time Correction

Figure 287 shows what can happen when the time stamps overlap. Time correction fixes this error and adjusts the time recordings accordingly.



dat.Clock <22206x1 double>

	1	2
373	437222	
374	438222	
375	439222	
376	440222	
377	441222	
378	442222	
379	443222	
380	444222	
381	445222	
382	446222	
383	447222	
384	448222	
385	7222	
386	8222	
387	9222	
388	10222	
389	11222	
390	12222	
391	13222	
392	14222	
393	15222	
394	16222	
395	17222	
396	18222	
397	19222	
398	20222	

dat.Clock <22206x1 double>

	1	2
371	435222	
372	436222	
373	437222	
374	438222	
375	439222	
376	440222	
377	441222	
378	442222	
379	443222	
380	444222	
381	445222	
382	446222	
383	447222	
384	448222	
385	455444	
386	456444	
387	457444	
388	458444	
389	459444	
390	460444	
391	461444	
392	462444	
393	463444	
394	464444	
395	465444	

Figure 288 Time Correction Reset Detection

Time correction detects resets by looking for time recordings that are less than their previous recordings. Figure 288 shows an example of a piccolo mat file where a reset had occurred. In the data on the left Cell 385 had a time recording of 7222 ms which was less than the previous cell 448222 ms. The data on the right shows the same piccolo mat file after time correction had been run. The function took the time value at cell 384 and added it to every value after that. There is no limit to the number of resets that time correction will fix in any one given piccolo mat file. If there had been other resets the correction function would continue to adjust all of the time stamps.

The list box displays all of the piccolo mat files that are present in the current directory. In order to determine what files are actually piccolo mat files the GUI searches the directory for files with the extension “.mat”. Then it looks into each “.mat” file for the variable “tClock”. All the user has to do is click on the file desired in the list box for the GUI to select it. When a file name is selected in the list box the GUI doesn’t actually load the file, it saves the name of the file to a handle where the file will be loaded whenever a plot button is clicked. The name of the selected piccolo mat file is used as the title of the plot. Figure 289 below shows a plot of barometer altitude from a piccolo mat file named “Flight 4”. The plot was titled “Flight 4 Barometer Altitude.”

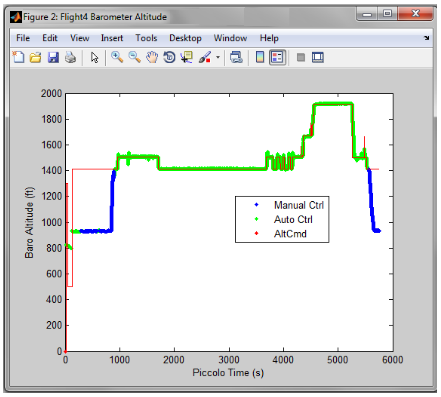


Figure 289 Figure Titles

The options section allows for the user to turn legends on or off. It also allows the option of plotting when waypoint targets change. The target waypoint option only actually plots the target waypoints for the following plots: Roll, Pitch, Yaw, az, TAS, GPSAlt, BaroAlt, and RPM.

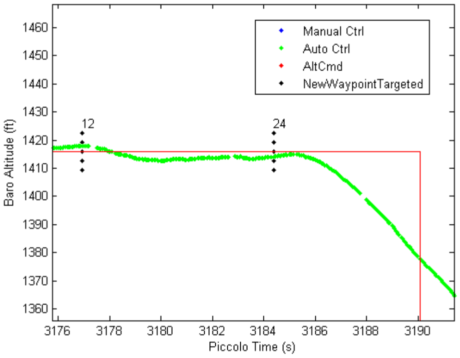


Figure 290 Show Target Waypoint

Figure 290 is an example of a barometer altitude plot with the show target waypoint option. The option plots 5 points, 2 above and 2 below the telemetry data with a text label for the waypoint number. The points are compatible with any units of telemetry and time.

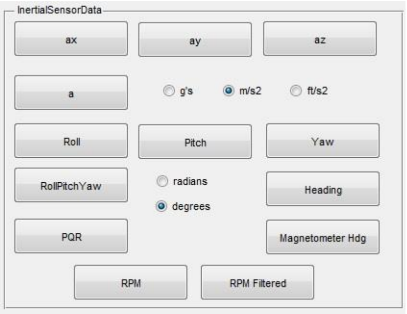


Figure 291 Inertial Sensor Data

The "InertialSensorData" section has functions to plot sensor telemetry. The radio buttons designate units. "ax" is the measured x acceleration, "ay" is the measured y acceleration, and "az" is the measured z acceleration. "a" will plot all three in one figure as subplots. "PQR" plots the roll, pitch, and yaw rates in one figure as subplots. The units for the angular rates are set by the attitude units; degrees will set the Roll Pitch Yaw units to degrees, and PQR to degrees/s. "Piccolo Temp" is the piccolo board temperature. "RPM Filtered" requires the variable "RPM" that is created from the script "PiccoloRPMRateFilter.m". The script filters the RPM data by the exact same process that the RPM filter on the piccolo uses so that the user can observe how the RPM data would be filtered if the RPM filter was used.

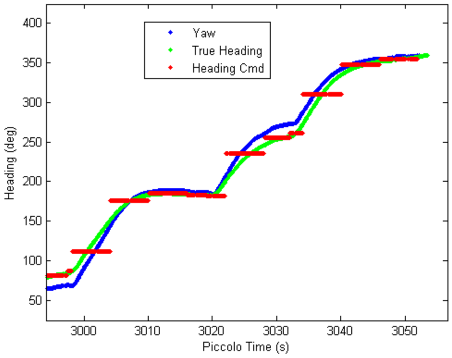


Figure 292 Heading Plot

“Heading” plots the true heading, heading command, and yaw angle estimate. When analyzing heading for lateral control purposes it is better to use plots that come from the DevInterface data.

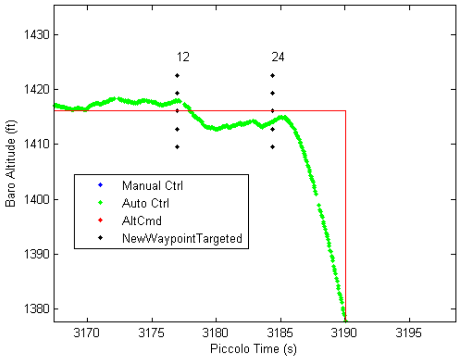


Figure 293 Piccolo Telemetry Altitude Command

It is important to note that the command loop commands that the piccolo telemetry log file records are delayed from when the commands actually changed, and when compared to their counterparts from DevInterface logs the difference is visible. Figure 293, shows that waypoint 24 was targeted about 6 seconds before the piccolo telemetry recorded that the commanded altitude had changed. The altitude command actually changed at the same time that waypoint 24 was targeted. The figure also shows that the altitude of the aircraft had decreased before the recorded altitude command had decreased which would not make sense.

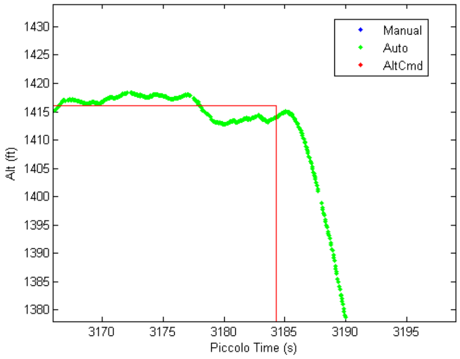
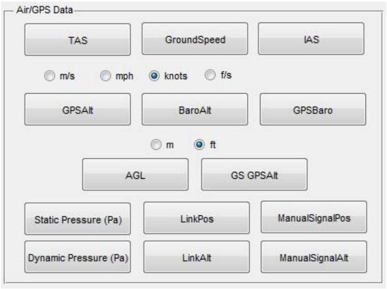


Figure 294 Controller Telemetry Altitude Command

Figure 294 shows the exact same time and flight recorded from the DevInterface log file. In that figure it can be seen that the altitude command actually changed at the same time that the target waypoint changed, 3184 seconds, and not at 3190 seconds as recorded by the piccolo telemetry log.



○ m/s ○ mph ● knots ○ f/s

○ m ● ft

Figure 295 Air & GPS Data

The section "Air/GPS Data" has functions to plot GPS and Air data telemetry. TAS is a recorded variable in the piccolo mat file; however, IAS is not. The GUI calculates IAS from the recorded dynamic pressure and air density at sea level using Equation 76 below.

$$IAS = \sqrt{\frac{2 * q}{\rho_o}}$$

Equation 76 AnalyzePiccolo IAS Calculation

$$where\ \rho_o = 1.225\frac{kg}{m^3} \quad q = Dynamic\ Pressure\ (Pa)$$

The IAS function also comes with the options of plotting the maximum and minimum indicated airspeeds and the fast IAS error threshold limit. The function can calculate both minimum airspeeds which are based on CL max and CL max nom. In order to calculate IAS$_{\mathrm{min}}$ the function will ask the user for CL max nom and CL max, along with the wing area, and the empty weight. The minimum airspeeds are calculated the same way that the autopilot does in Equation 77 and Equation 78 below.

$$IAS_{min} = \sqrt{\frac{2 * mg}{C_{L\,Max} * S_w * \rho_o}}$$

Equation 77 Minimum Indicated Airspeed

$$IAS_{\mathrm{min}\,nom} = \sqrt{\frac{2 * mg}{C_{L\,\mathrm{max}\,nom} * S_w * \rho_o} * 1.1}$$

Equation 78 Nominal Minimum Indicated Airspeed

$$where\ S_w = wing\ area\ (m^2)\ \ m = mass\ estimate\ (kg)$$

$$\rho_o = 1.225\frac{kg}{m^3} \quad g = 9.81\,m/s^2$$

The function will also ask for the maximum indicated airspeed and the Fast IAS Error Threshold. The user can leave any of the inputs empty or click cancel to not include them in the plot. Figure 296Figure 296 IAS Plot below is an example of the IAS plot with all of the parameters input.

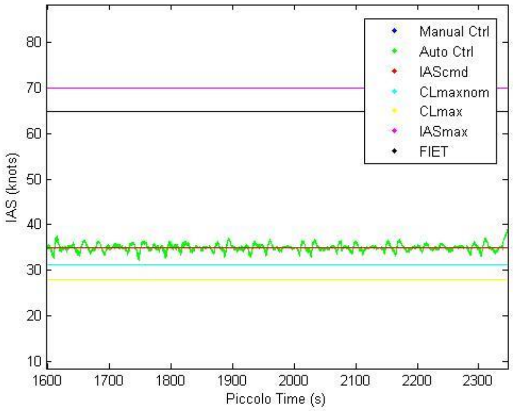


Figure 296 IAS Plot

“GPSBaro” will plot both the GPS altitude and the barometer altitude. “GS GPSAlt” plots the ground station GPS altitude. “LinkPos” plots the piccolo packet loss with respect to the aircraft’s position. The function uses GPS data to obtain the latitude and longitude paths of the aircraft and plots the points with different colors to represent different levels of packet loss. Similarly “ManualSignalPos” plots the signal strength of the manual pilot where the system is using the JR level shifter to fly the aircraft with its own antennas outside of the piccolo communications.

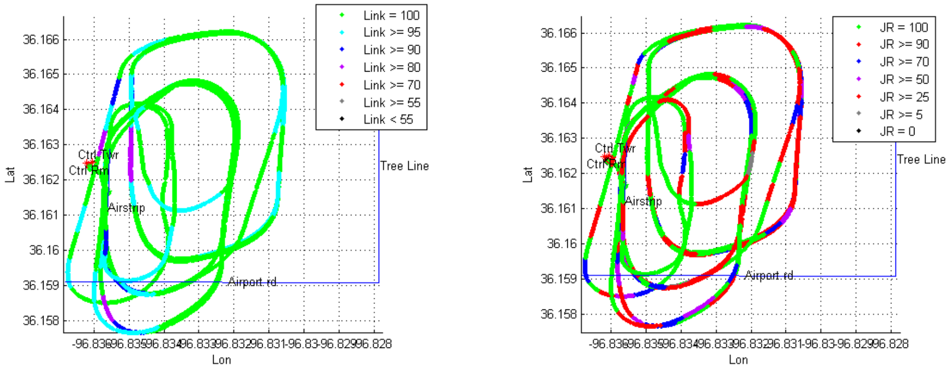


Figure 297 Communication Signal Strength vs. GPS Latitude and Longitude Location

Figure 297 depicts example plots of the Link (left) & Manual Signal (right) position plots. The plots also include geo locations of objects that exist at the OSU UAFS to help provide a frame of reference for analysis. The plots display the locations of the control room, control tower, east side of the airstrip, airport road, and the tree line to the east.

Additionally the Link and Manual Signal position plots also come with the option to specify minimum and maximum altitude. The altitude function allows the user to analyze the effects of altitude on the signal strengths. If an altitude minimum and/or altitude maximum is specified AnalyzePiccolo will disregard signal strength data that corresponds with altitudes outside of the minimum and/or maximum.

“LinkAlt” plots the piccolo packet loss versus altitude with different colors to indicate the latitude of the aircraft at the data points. The plots plot the data points green when the aircraft is south of the control room and blue when the aircraft is north of the control room. Similarly “ManualSignalAlt” plots the JR signal strength versus altitude with the same variations in color to indicate latitude. Figure 298 below illustrates an example of the signal strength altitude plots.

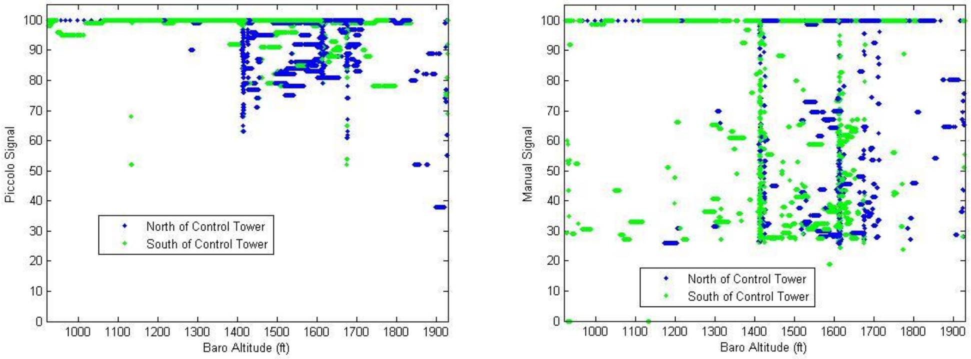


Figure 298 Communications Signal Strength vs. Altitude

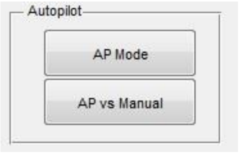


Figure 299 AnalyzePiccolo Autopilot Section

“AP vs Manual” plots the GPS latitude and longitude data for the aircraft with different colors to represent when the autopilot was in control and when the manual pilot was in control. It also plots the two control modes versus time. This function is mostly useful for making sure that the manual pilot didn’t lose control when he/she was supposed to have it. “AP Mode” plots the mode that the autopilot is in versus time and is shown below in Figure 301.

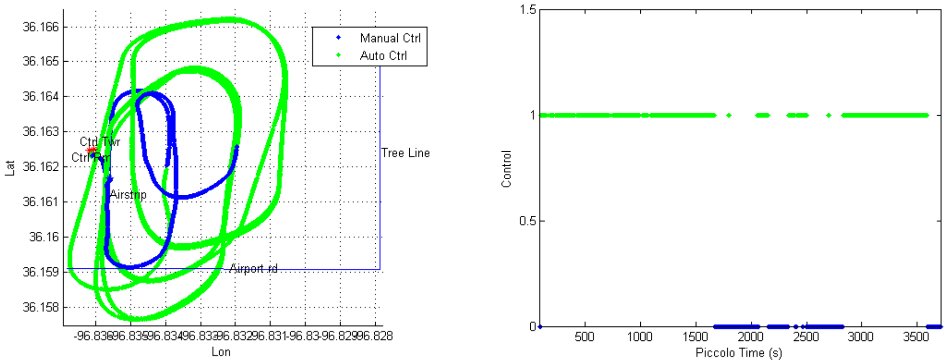


Figure 300 AP vs Manual Plots

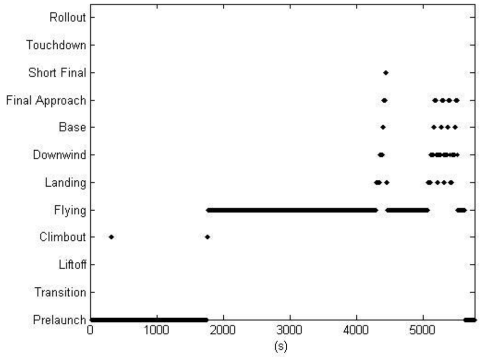
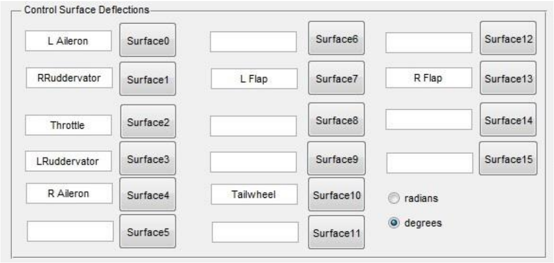


Figure 301 AP Mode Plot



L Aileron

RRuddervator

Throttle

LRuddervator

R Aileron

L Flap

Tailwheel

R Flap

○ radians

● degrees

Figure 302 Control Surface Deflections

The “Control Surface Deflections” section allows the user the ability to plot any control surface deflection and name the control surfaces as applies to the aircraft. Each surface button will plot the corresponding surface telemetry data where the plots will be titled and labeled according to the control surface name specified in the corresponding input box. Each time an input box is edited the entire GUI will save itself so that the user doesn’t have to input the control surface names each time the GUI is launched.

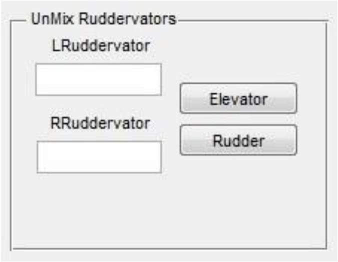


Figure 303 UnMix Ruddervators

The “UnMix Ruddervators” section allows the user to plot elevator and rudder deflections instead of just left and right ruddervator deflections. The user must specify the number of the control surface for the left and right ruddervators before the function will work. The function uses Equation 73 and Equation 74 from Section 6.3.1.1 to un mix upright ruddervators.

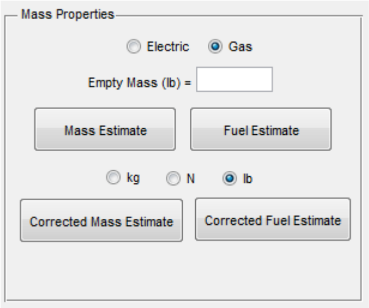


Figure 304 Mass Properties

The “Mass Properties” section allows the user to perform various plots of mass. The fuel estimate plots the in flight fuel weight that the piccolo estimated. The mass estimate plots the fuel estimate plus the empty weight. Corrected fuel estimate plots the variable “CorrectedFuel” which comes from the script “FuelCorrection.m”. The corrected fuel uses the measured fuel burn to correct the estimate of fuel burn throughout the flight. Corrected mass estimate plots the corrected fuel plus the empty weight.

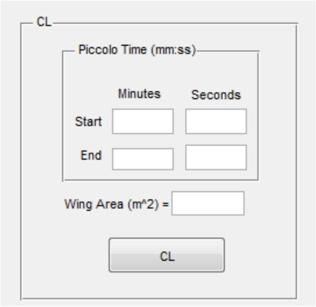


Figure 305 CL

The “CL” section calculates and plots CL. The user has the option to define a time range for CL analysis. The time scale is piccolo time. If no time is input the function will plot CL for the entire flight. The function requires the wing area and empty mass. The function will use the corrected mass estimate if the corrected fuel variable exists, otherwise it will use the mass estimate to calculate CL. The CL calculation also relies on z – acceleration and dynamic pressure measurements and uses the following equation.

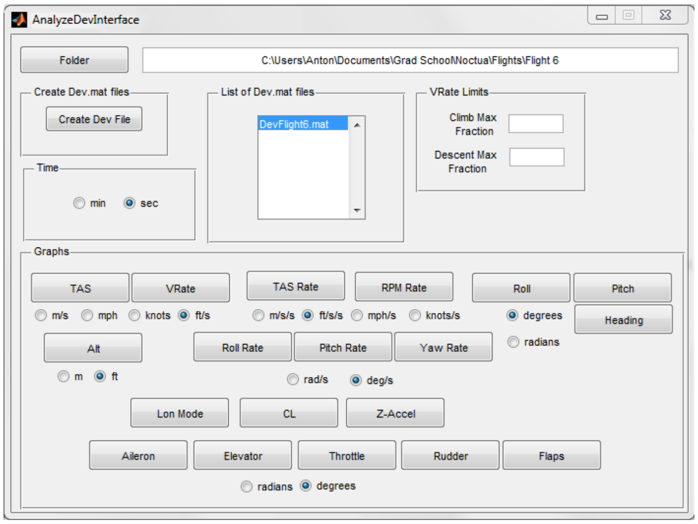
$$C_L = \frac{-m * Z_{accel}}{qS_w}$$

$$where\ Z_{accel} = \left(\frac{m}{s^2}\right)\ \ q = Dynamic\ Pressure\ (Pa)\ \ S_w = Wing\ Area\ (m^2)\ \ m = mass\ (kg)$$

Figure 306 AnalyzePiccolo CL Calculation

7.3. AnalyzeDevInterface

"AnalyzeDevInterface.m" is a MATLAB script that launches a figure that acts as a GUI, "AnalyzeDevInterface.fig". AnalyzeDevInterface is designed to provide the user a user friendly way of plotting the controller telemetry data that can be recorded by the DevInterface into log files. AnalyzeDevInterface imports data from the DevInterface log files and imports variables from the piccolo mat file to create a "Dev.mat" file that contains the Dev variables.



AnalyzeDevInterface

Folder: C:\Users\Anton\Documents\Grad School\Noctua\Flights\Flight 6

List of Dev.mat files: DevFlight6.mat

Time: ○ min ● sec

○ m/s ○ mph ○ knots ● ft/s

○ m/s/s ● ft/s/s ○ mph/s ○ knots/s

● degrees ○ radians

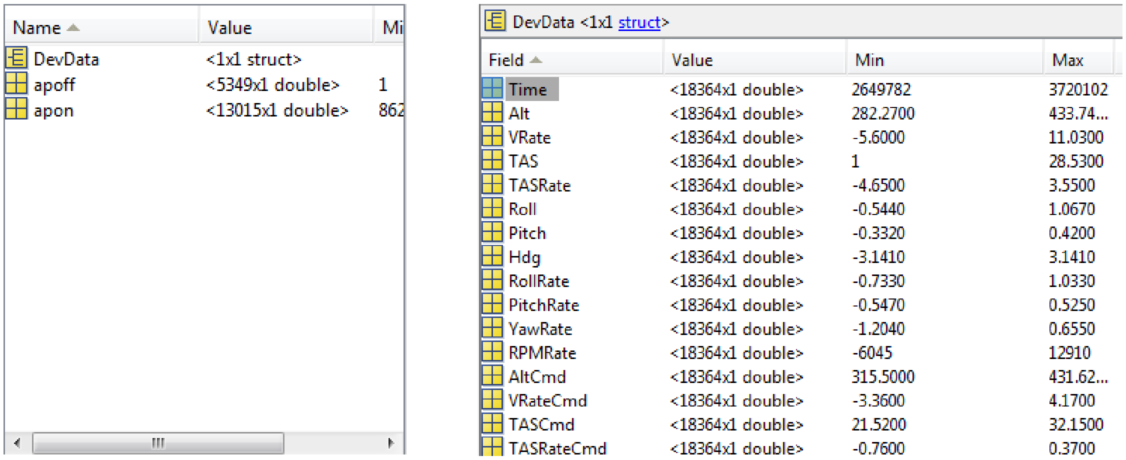
○ m ● ft

○ rad/s ● deg/s

○ radians ● degrees

Figure 307 AnalyzeDevInterface GUI

AnalyzeDevInterface loads Dev mat files that are selected in the list box. The list box lists Dev mat files that exist in the current directory. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The “Folder” button will pull up a dialog box for the user to select another folder as the directory.



Name	Value	Mi
DevData	<1x1 struct>	
apoff	<5349x1 double>	1
apon	<13015x1 double>	862

DevData <1x1 struct>

Field	Value	Min	Max
Time	<18364x1 double>	2649782	3720102
Alt	<18364x1 double>	282.2700	433.74...
VRate	<18364x1 double>	-5.6000	11.0300
TAS	<18364x1 double>	1	28.5300
TASRate	<18364x1 double>	-4.6500	3.5500
Roll	<18364x1 double>	-0.5440	1.0670
Pitch	<18364x1 double>	-0.3320	0.4200
Hdg	<18364x1 double>	-3.1410	3.1410
RollRate	<18364x1 double>	-0.7330	1.0330
PitchRate	<18364x1 double>	-0.5470	0.5250
YawRate	<18364x1 double>	-1.2040	0.6550
RPMRate	<18364x1 double>	-6045	12910
AltCmd	<18364x1 double>	315.5000	431.62...
VRateCmd	<18364x1 double>	-3.3600	4.1700
TASCmd	<18364x1 double>	21.5200	32.1500
TASRateCmd	<18364x1 double>	-0.7600	0.3700

Figure 308 Dev Mat File Data

Dev mat files consist of a structure ‘DevData’, and 3 variables ‘apon’, and ‘apoff’. Figure 308 shows a preview of some of the variables in the DevData structure. DevData contains each column of the Dev log file as a variable. All of the variables that the Dev log file records are available to plot in the AnalyzeDevInterface GUI. ‘apon’ and ‘apoff’ are integrated from the piccolo telemetry log files. ‘apon’ represents data in which the autopilot was in control and ‘apoff’ represents data in which the manual pilot was in control.

The “Create Dev File” button launches the function that creates Dev mat files. The user will be required to select the Dev log file and the corresponding piccolo mat file. The function will import the Dev log file data into the DevData structure. The function uses the piccolo mat file to import the apon and apoff variables. Since controller telemetry is not always recorded at the same rate as the piccolo telemetry the function corrects the apon and apoff data for gaps in time that may occur if the data logging rates are not equivalent. After the Dev mat file is created the GUI will automatically save the mat file in the directory that the DevInterface is currently in and reload the list box so that it contains the new Dev mat file as an option. The mat file will be saved with the same name as the Dev log file that it loaded.

The “Time” section allows the user to specify minutes or seconds. The time scale recorded by the DevInterface is the piccolo clock, which is the same clock as ‘dat.Clock’ in the piccolo mat files.

The “VRate Limits” section provides the user with the option to include the VRate maximum and VRate minimum in the VRate plots. VRate max is calculated from the Climb Max Fraction and VRate min is calculated from the Descent Max Fraction with Equation 45 and Equation 46 from Section 5.8.3.1. If they are left blank they will not be included in the VRate plots.

All of the Controller Telemetry plots are self explanatory. They don’t require any special input from the user. Similar to “AnalyzePiccolo” each figure will have the title of the mat file as the title of the figure. Figure 309 below is an example of a VRate plot from AnalyzeDevInterface. It is important to note that the VRate that the DevInterface actually logs is always the GPS/INS solution (GPS based or barometer based). If a laser altimeter is being used the DevInterface will display both the GPS/INS and laser altimeter based vertical rates; however, the laser altimeter based vertical rate will not be recorded in the DevInterface log file.

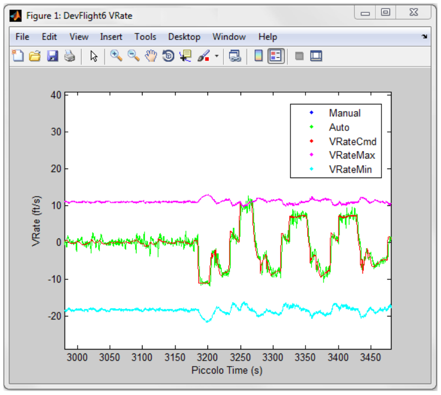
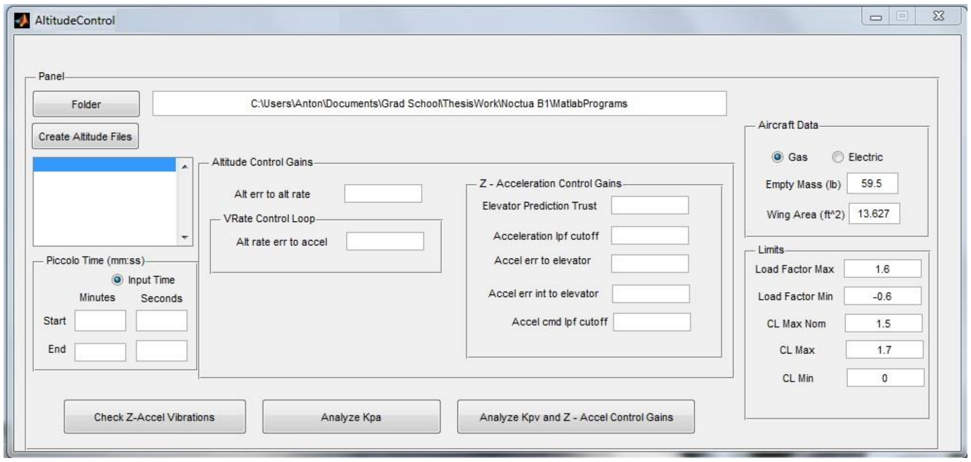


Figure 309 VRate Example Plot

7.4. AltitudeControl

“AltitudeControl.m” is a MATLAB script that launches a figure that acts as a GUI, “AltitudeControl.fig”. AltitudeControl is designed to provide the user a user friendly way to analyze altitude control performance. AltitudeControl imports data from the DevInterface log files and plots the telemetry data that is applicable to the altitude control control loops. It is important to note that this GUI is designed for analyzing altitude control when the aircraft is actually in altitude control (Lon Mode 0).



C:\Users\Anton\Documents\Grad School\ThesisWork\Noctua B1\MatlabPrograms

● Gas ○ Electric

Empty Mass (lb) 59.5

Wing Area (ft^2) 13.627

Load Factor Max 1.6

Load Factor Min -0.6

CL Max Nom 1.5

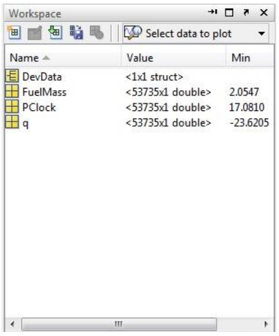
CL Max 1.7

CL Min 0

● Input Time

Figure 310 AltitudeControl GUI

AltitudeControl loads “AltitudeCtrl.mat” mat files that are selected in the list box. The list box lists AltitudeCtrl mat files that exist in the current directory. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The “Folder” button will pull up a dialog box for the user to select another folder as the directory.



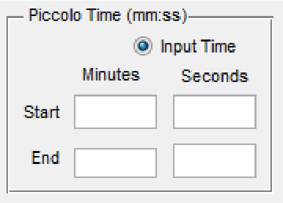
Name	Value	Min
DevData	<1x1 struct>	
FuelMass	<53735x1 double>	2.0547
PClock	<53735x1 double>	17.0810
q	<53735x1 double>	-23.6205

Figure 311 AltitudeControl Mat File

AltitudeCtrl mat files consist of a structure ‘DevData’, and the variables “FuelMass”, “PClock”, and “q”. DevData contains each column of the Dev log file as a variable. FuelMass, PClock, and q are extracted from the user selected piccolo telemetry log file. PClock is simply the time as recorded in the piccolo telemetry log. The GUI searches each mat file in the current directry for the DevData structure, and the variables time and apon. Time and apon exist in Dev and Energy mat files; thus, mat files containing those variables are excluded from the list box.

The “Create Altitude Files” button launches the function that creates AltitudeCtrl mat files. The user will be required to select the appropriate Dev log file. The function will import the Dev log file data into the DevData structure. Then the user will be prompted to select the corresponding piccolo telemetry log file. After the AltitudeCtrl mat file is created the GUI will automatically save the mat file in the directory that the list box is currently in and reload the list box so that it contains the new AltitudeCtrl mat file as an option. The mat file will be saved as

"AltitudeCtrl" plus the flight name from the Dev log file name (removes the "Dev" portion of the file name).



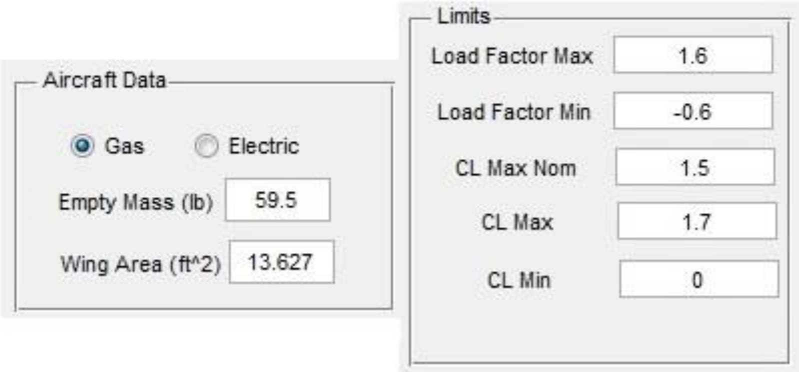
Piccolo Time (mm:ss)

● Input Time

	Minutes	Seconds
Start		
End		

Figure 312 Piccolo Time Section

The "Piccolo Time" section allows the user to enter a time period for the initial plots. The initial plots allow the user to select the time period for analysis by clicking and dragging on the initial plots. The time scale used is the piccolo time which is displayed in the DevInterface during flight. The "Input Time" radio button declares whether or not to use user input time. If user input time is to be used the initial plots will plot only the time period that has been declared by the user; otherwise, the initial plots will include the entire time span of the AltitudeCtrl mat file.



Aircraft Data

● Gas ○ Electric

Empty Mass (lb) 59.5

Wing Area (ft^2) 13.627

Limits

Load Factor Max 1.6

Load Factor Min -0.6

CL Max Nom 1.5

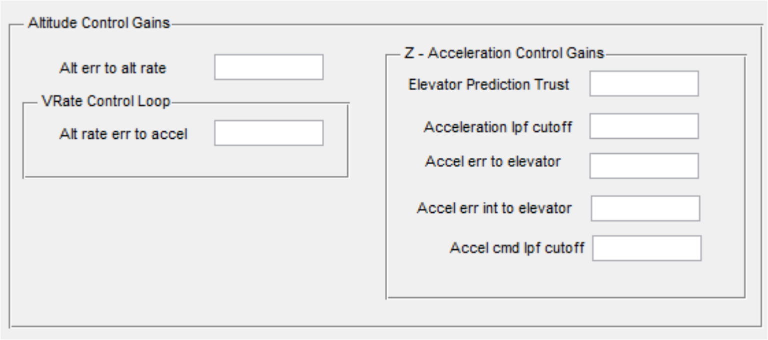
CL Max 1.7

CL Min 0

Figure 313 Aircraft Data & Limits Sections

The "Aircraft Data" and "Limits" sections are used to calculate the z – acceleration limits that the controller uses in the z – acceleration limiter. If any values are changed the GUI will save so that the user does not have to change the values every time that the program is used.

The option for Electric in Aircraft Data exists so that the program does not try to use the FuelMass variable for aircraft running electric propulsion systems. The z acceleration limit calculations uses the equations found in Section 5.8.3.1.



The “Altitude Control Gains” section takes inputs for the control loop gains. If the user inputs a gain value that gain value will appear at the top of each plot in the analysis to help the user keep track of which system response occurred with which gains.

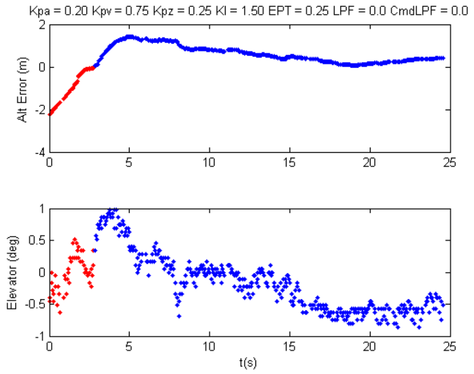


Figure 314 AltitudeControl Plot

Figure 314 is an example of a plot with all of the altitude control gains input. Any gains that are left blank will not appear at the top of the plots. The following table decodes the abbreviations for all of the control loop gains.

Table 5 Altitude Control Loop Gain Abbreviations

Kpa =	Alt err to alt rate	KI =	Accel err int to elevator
Kpv =	Alt rate err to accel	EPT =	Elevator prediction trust
Kpz =	Accel err to elevator	LPF =	Accel lpf cutoff
Cmd LPF =	Accel cmd lpf cutoff		

There are three different buttons for different analyses. All three include initial plots to select periods of time for analysis. The initial plots give the user the opportunity to zoom and pan to the desired time period for analysis. To begin the selection process the user must press “enter” on the keyboard.

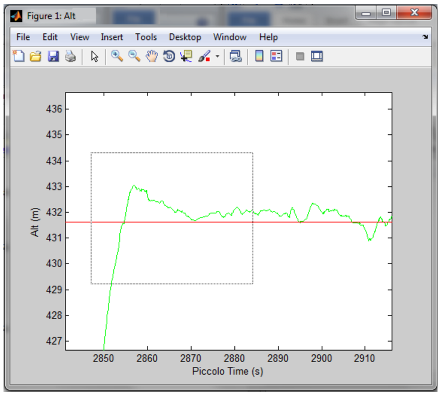


Figure 315 Time Period Selection

After pressing enter the user can click, hold, and drag to select a time period for analysis. Figure 315 is an example of the user selecting a time period from an initial plot. The initial plot for “Check Z-Accel Vibrations” and “Analyze Kpa” is altitude versus time. The initial plots for “Analyze Kpv and Z – Accel Control Gains” are altitude and vrate versus time.

The “Check Z-Accel Vibrations” button launches the vibrations analysis. The vibrations analysis requires the corresponding piccolo log file in addition to the AltitudeCtrl mat file. RPMs are not recorded by the DevInterface so they must be taken from the piccolo log file. Before selecting a time period for analysis the user will have to select the appropriate piccolo log file.

After selecting the piccolo log file the user will then be prompted by the initial altitude versus time plot. After the user selects a time period for analysis a series of relevant plots will occur as shown below.

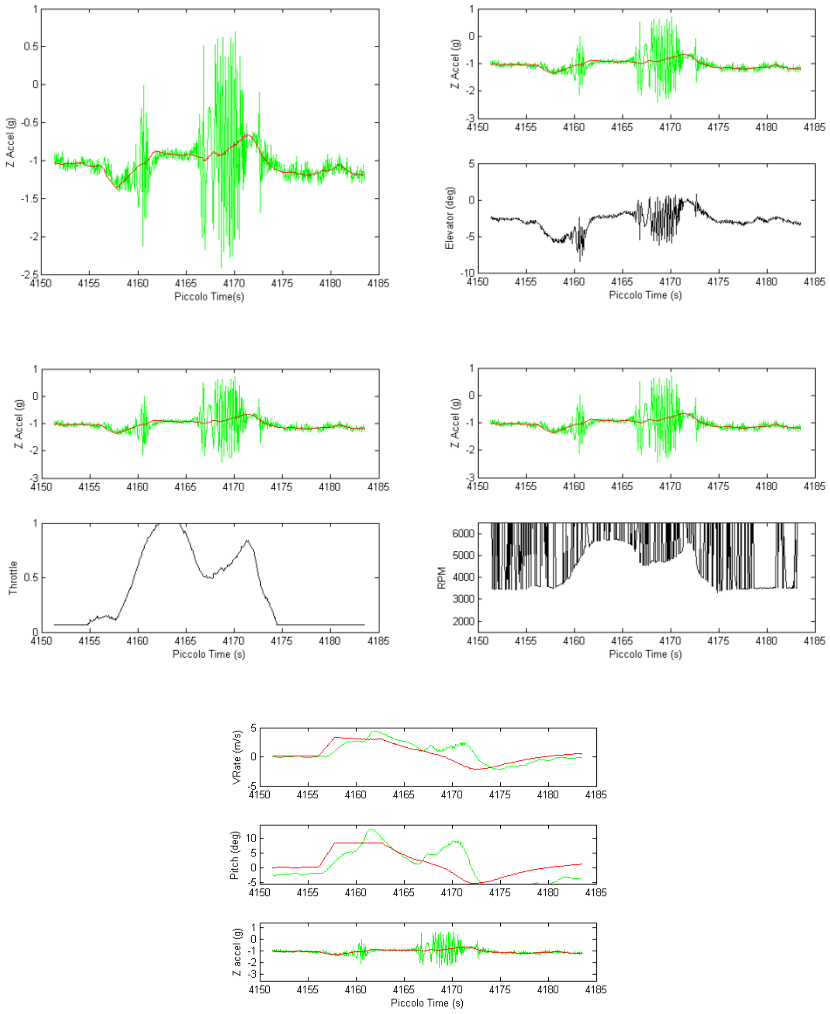


Figure 316 Example Z - Accel Vibration Plots

Figure 316 is an example of a flight analysis (Noctua Flight 4) where there were Z – Acceleration issues. There were resonant vibrations in the z accelerometer measurements at about 3800 – 4800 RPMs. Notice the sudden pitch up motion at both vibration instances that caused the aircraft to overshoot the target altitude after climbs.

The “Analyze Kpa” button launches the altitude control outer loop analysis. All the user is required to do is select a time period from the initial altitude versus time plot. The outer loop only contains one gain, Kpa (Alt err to alt rate), and directly issues the VRate command. As a result there are only three plots that are plotted for analysis.

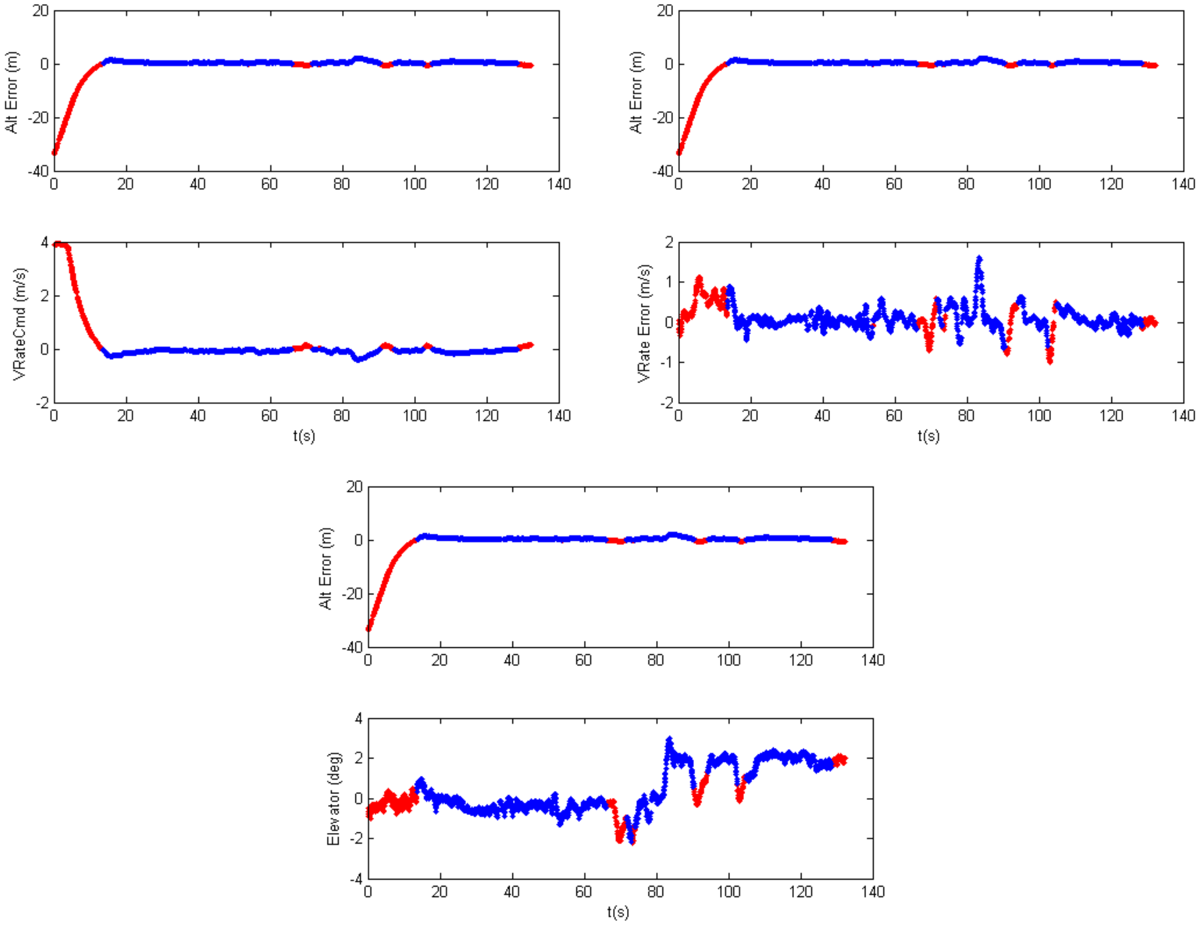
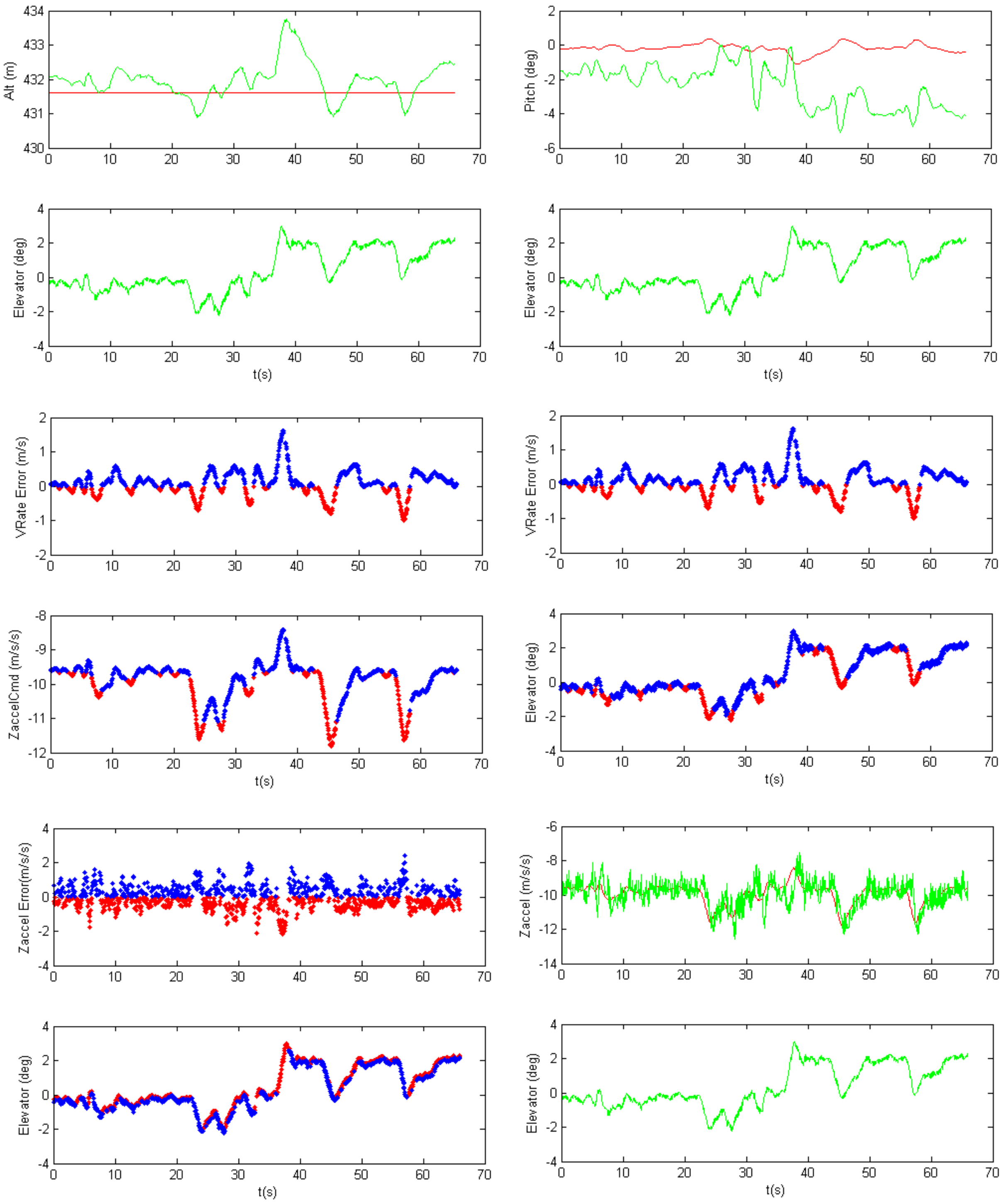
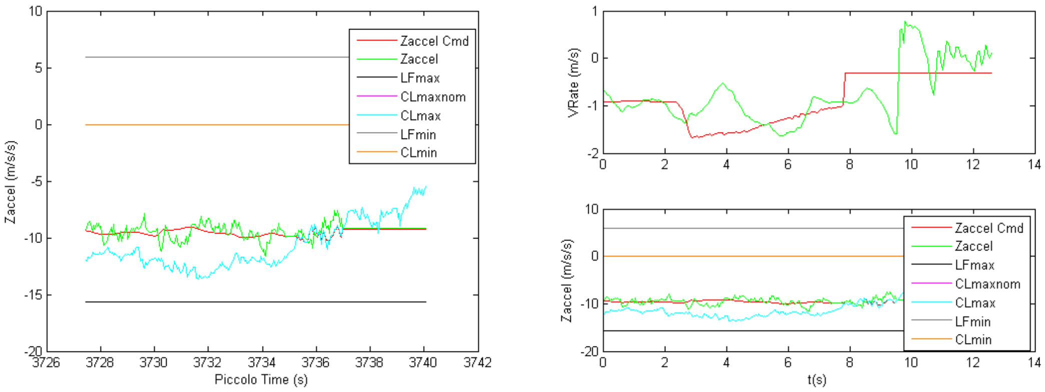


Figure 317 Analyze Kpa Plots

The red data denotes that the error plotted in the top subplot of each graph is negative; blue denotes that the error is positive.

The “Analyze Kpv and Z – Accel Control Gains” button launches the altitude control inner loop analysis. All the user is required to do is select a time period from either the initial altitude versus time plot or the initial vrate versus time plot. The reason that the vrate gain is included in the same analysis as the z – acceleration control loop gains is because only the vrate command can be set to a constant via manual command. The z – acceleration command cannot be issued manually by the user; therefore, the gain tuning process for the z – acceleration control loop gains will have to include the vrate control loop gain as well. The function plots the plots shown below.


Alt (m)
434
433
432
431
430
Pitch (deg)
2
0
-2
-4
-6
Elevator (deg)
4
2
0
-2
-4
t(s)
VRate Error (m/s)
1
-1
ZaccelCmd (m/s/s)
-8
-9
-10
-11
-12
Zaccel Error(m/s/s)
Zaccel (m/s/s)
-6
-8
-10
-12
-14
0
10
20
30
40
50
60
70



The red data denotes that the error plotted in the top subplot is negative; blue denotes that the error is positive. In plots with a red and green line the red line denotes the commanded value and the green line denotes the measured value.

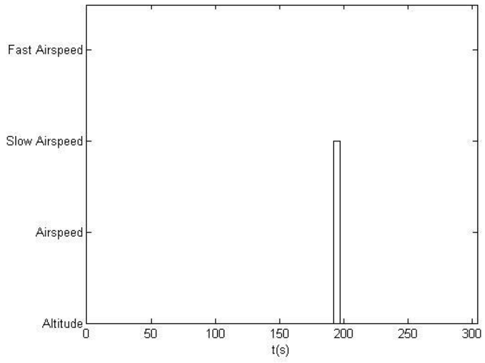
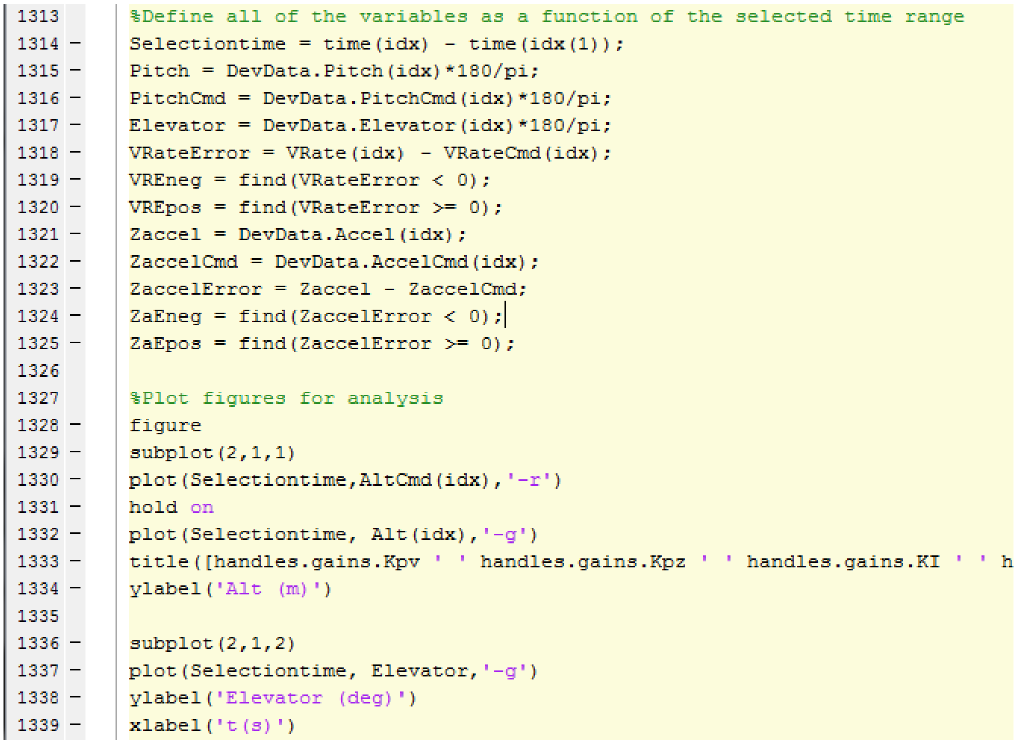


Figure 318 Lon Mode Warning

When a time period is selected for any of the three different analysis options the GUI will check to make sure that the autopilot was in Altitude Control throughout the entire time period. It does so by checking the recorded Lon Modes. If any part of the selected time period contains any Lon Mode other than 1 a warning dialog box will appear along with a Lon Mode plot, shown in Figure 318.

It is possible that additional plots could need to be made for tuning altitude control gains. If that is the case the code can be easily modified by copying and pasting the plot sections that already exist and changing the variables.



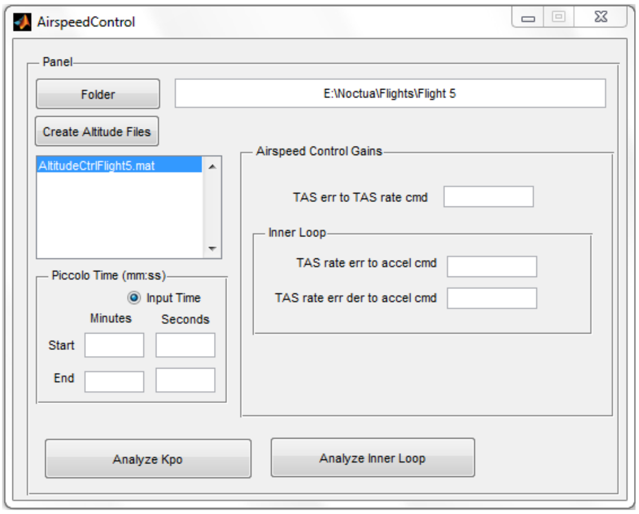
```
%Define all of the variables as a function of the selected time range
Selectiontime = time(idx) - time(idx(1));
Pitch = DevData.Pitch(idx)*180/pi;
PitchCmd = DevData.PitchCmd(idx)*180/pi;
Elevator = DevData.Elevator(idx)*180/pi;
VRateError = VRate(idx) - VRateCmd(idx);
VREneg = find(VRateError < 0);
VREpos = find(VRateError >= 0);
Zaccel = DevData.Accel(idx);
ZaccelCmd = DevData.AccelCmd(idx);
ZaccelError = Zaccel - ZaccelCmd;
ZaEneg = find(ZaccelError < 0);
ZaEpos = find(ZaccelError >= 0);

%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime,AltCmd(idx),'-r')
hold on
plot(Selectiontime, Alt(idx),'-g')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' ' h
ylabel('Alt (m)')

subplot(2,1,2)
plot(Selectiontime, Elevator,'-g')
ylabel('Elevator (deg)')
xlabel('t(s)')
```

Figure 319 AltitudeControl Plot Code

Figure 319 is a snapshot of some of the code for AltitudeControl.m. The variables available can be viewed in any AltitudeControl mat file; however, they are all the variables that the DevInterface records. They can be assigned to variables by referencing the structure, DevData, and make sure to include the designation for the time selected period “idx”. DevData.Variable(idx).

7.5. AirspeedControl

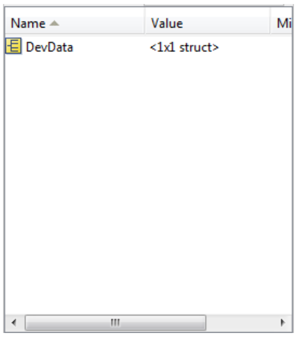
“AirspeedControl.m” is a MATLAB script that launches a figure that acts as a GUI, “AirspeedControl.fig”. AirspeedControl is designed to provide the user a user friendly way to analyze airspeed control performance and tune airspeed control gains during Lon Modes 2, and 3. The GUI is not designed to analyze actual airspeed control (Lon Mode 1). AirspeedControl imports data from the DevInterface log files and plots the telemetry data that is applicable to the airspeed control control loops.



E:\Noctua\Flights\Flight 5

AltitudeCtrlFlight5.mat

AirspeedControl uses the same mat files as AltitudeControl, “AltitudeCtrl.mat” mat files. The mat files are visible in the list box. The list box lists AltitudeCtrl mat files that exist in the current directory. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The “Folder” button will pull up a dialog box for the user to select another folder as the directory.

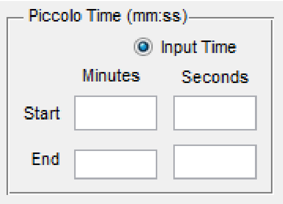


Name ▲	Value	Mi
DevData	<1x1 struct>	

AltitudeCtrl mat files consist of a structure ‘DevData’. DevData contains each column of the Dev log file as a variable. The GUI searches each mat file in the current directry for the

DevData structure, and the variables time and apon. Time and apon exist in Dev and Energy mat files, and mat files containing those variables are excluded from the list box.

The “Create Altitude Files” button launches the function that creates AltitudeCtrl mat files. The user will be required to select the appropriate Dev log file. The function will import the Dev log file data into the DevData structure. After the Dev mat file is created the GUI will automatically save the mat file in the directory that the DevInterface is currently in and reload the list box so that it contains the new AltitudeCtrl mat file as an option. The mat file will be saved as “AltitudeCtrl” plus the flight name from the Dev log file name (removes the “Dev” portion of the file name).



The “Piccolo Time” section allows the user to enter a time period for the initial plots. The initial plots allow the user to select the time period for analysis by clicking and dragging on the initial plots. The time scale used is the piccolo time which is displayed in the DevInterface during flight. The “Input Time” radio button declares whether or not to use user input time. If user input time is to be used the initial plots will plot only the time period that has been declared by the user; otherwise, the initial plots will include the entire time span of the AltitudeCtrl mat file.

The “Airspeed Control Gains” section takes inputs for the control loop gains. If the user inputs a gain value that gain value will appear at the top of each plot in the analysis to help the user keep track of which system response occurred with which gains.

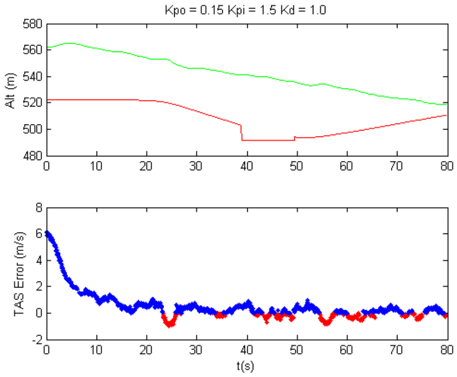


Figure 320 Airspeed Control Plot

Figure 320 is an example of a plot with all of the airspeed control gains input. Any gains that are left blank will not appear at the top of the plots. The following table decodes the abbreviations for all of the control loop gains.

Kpo =	TAS err to TAS rate cmd
Kpi =	TAS rate err to accel cmd
Kd =	TAS rate err der to accel cmd

There are two different buttons for different analyses. When any one of them is clicked the initial plots will pop up. The initial plots give the user the opportunity to zoom and pan to the desired time period for analysis. To begin the selection process the user must press “enter” on the keyboard.

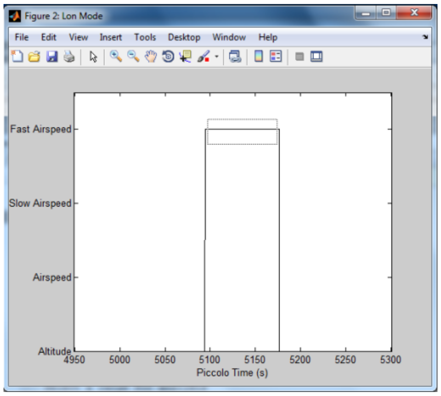
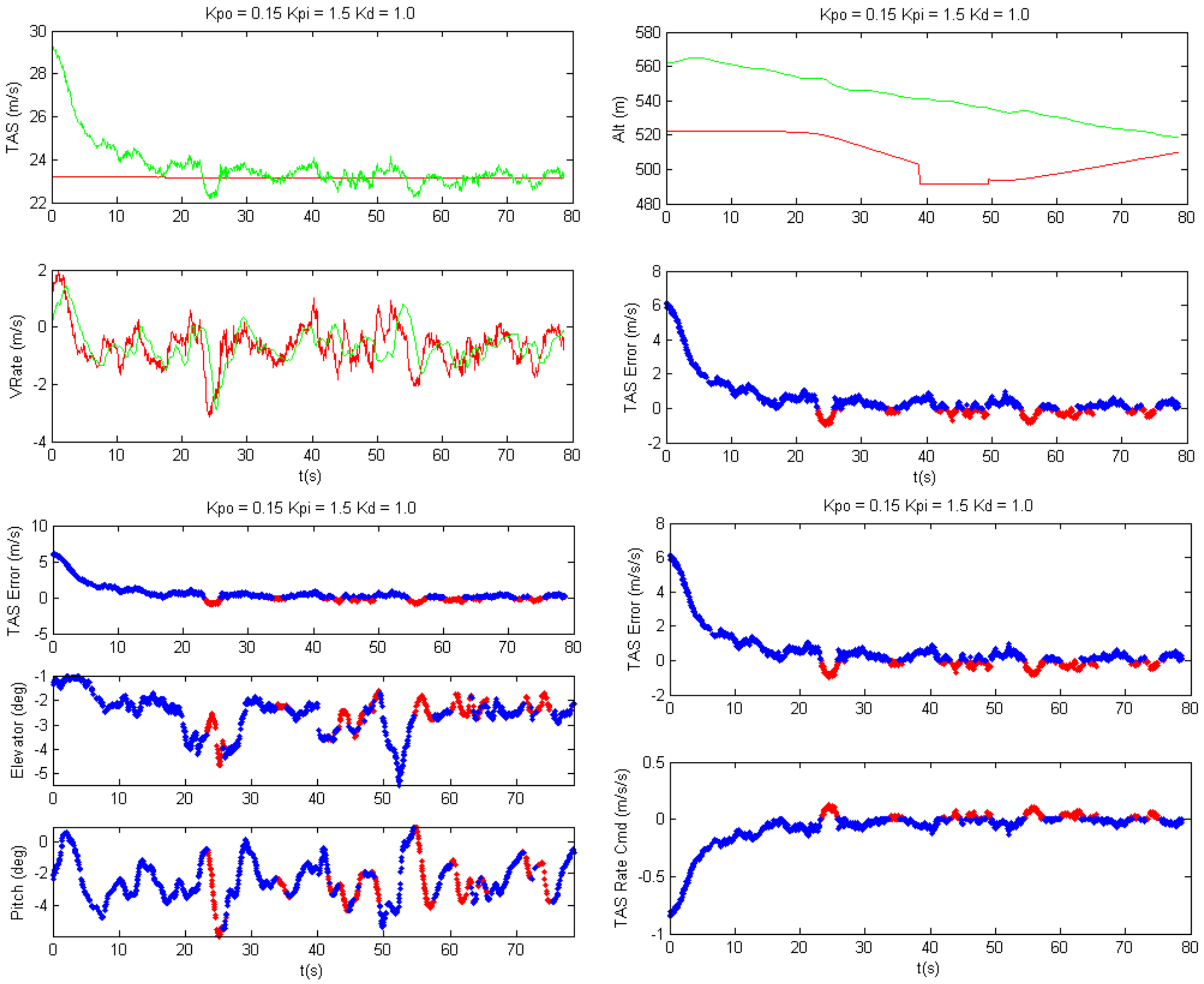


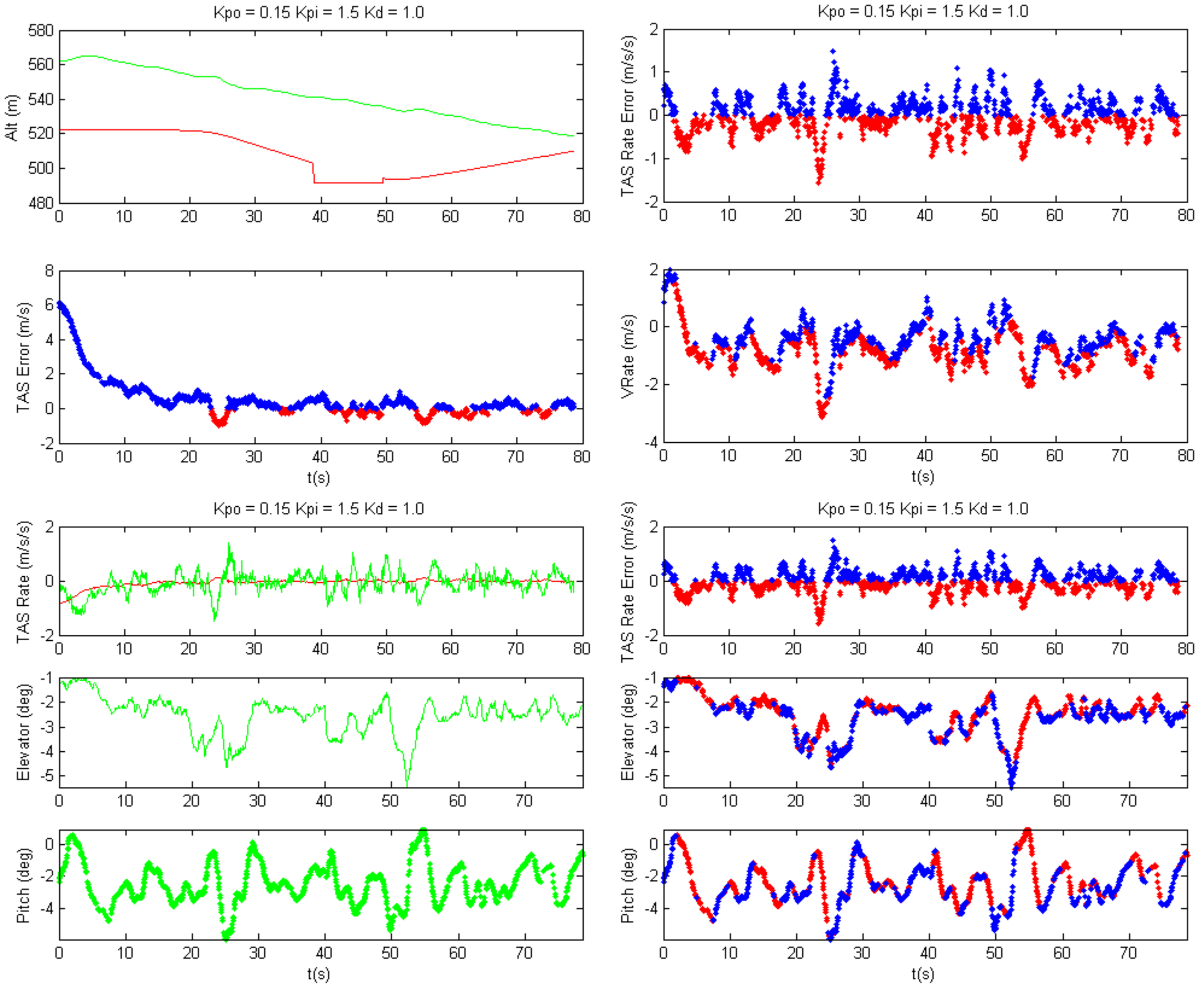
Figure 321 Airspeed Conrol Select Time Period

After pressing enter the user can click, hold, and drag to select a time period for analysis. Figure 321 is an example of the user selecting a time period from an initial plot. The initial plots for “Analyze Kpo” and “Analyze Inner Loop” are TAS and Lon Mode versus time.

The “Analyze Kpo” button launches the airspeed control outer loop analysis. All the user is required to do is select a time period from one of the initial plots. The function plots the plots shown in the figure below.



The “Analyze Inner Loop” button launches the airspeed control inner loop analysis. All the user is required to do is select a time period from one of the initial plots. Recall that the TASRate command actually commands vrate and passes through the vrate control loop before entering into the z – acceleration control loop and finally reaching the elevator. Both the vrate and z – acceleration control loops are excluded from the airspeed control analysis because this analysis is for systems that are meant to operate in altitude control so the vrate and z – acceleration control loops should be tuned for controlling altitude and for controlling airspeed in fast or slow airspeed modes. The function plots the plots shown in the figure below.

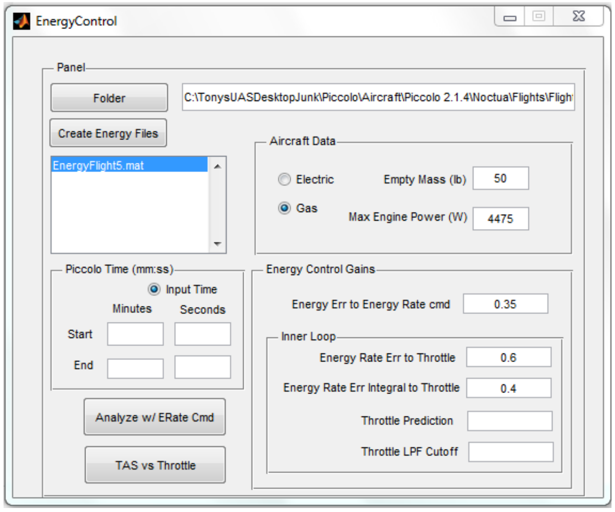


The red data denotes that the error plotted in the top subplot is negative; blue denotes that the error is positive. In plots with a red and green line the red line denotes the commanded value and the green line denotes the measured value.

In the code all of the units are metric because the piccolo control laws calculate with metric units; however, for the plots the units were converted to english units. The conversion is for only plots of TAS, altitude, and vrate.

7.6. EnergyControl

"EnergyControl.m" is a MATLAB script that launches a figure that acts as a GUI, "EnergyControl.fig". EnergyControl is designed to provide the user a user friendly way to analyze throttle performance and tune energy control gains during Altitude Control, Lon Mode 0. The GUI is designed to analyze the throttle's ability to maintain airspeed; therefore, it is not designed to analyze throttle performance in Airspeed Control. In Airspeed Control the throttle is used to control altitude. EnergyControl imports data from the DevInterface log files and plots the telemetry data that is applicable to the energy control control loops. The Energy Rate feedback and commands actually have to be calculated because the DevInterface does not log or display these values.



C:\TonysUASDesktopJunk\Piccolo\Aircraft\Piccolo 2.1.4\Noctua\Flights\Fligh

EnergyFlight5.mat

○ Electric
● Gas

Empty Mass (lb) 50

Max Engine Power (W) 4475

● Input Time

Energy Err to Energy Rate cmd 0.35

Energy Rate Err to Throttle 0.6

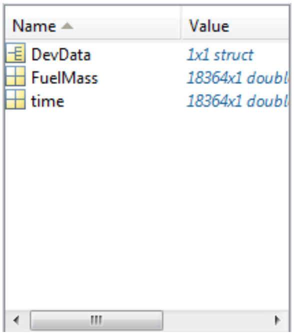
Energy Rate Err Integral to Throttle 0.4

Throttle Prediction

Throttle LPF Cutoff

Figure 322 EnergyControl GUI

EnergyControl loads "Energy.mat" mat files that are selected in the list box. The list box lists Energy mat files that exist in the current directory. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The "Folder" button will pull up a dialog box for the user to select another folder as the directory.



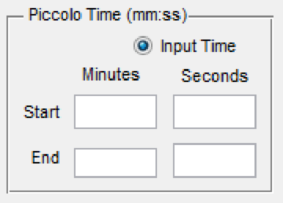
Name	Value
DevData	1x1 struct
FuelMass	18364x1 doubl
time	18364x1 doubl

Figure 323 Energy Mat File

Energy mat files consist of a structure 'DevData', and two variables "Fuel Mass", and "time". DevData contains each column of the Dev log file as a variable. FuelMass contains the fuel data from the corresponding piccolo telemetry log file that has been adjusted to match the time stamps and telemetry rate of the Dev log data. The variable time is DevData.Time/1000 and exists so that the other GUI's can distinguish Energy mat files from other types of mat files. The GUI searches each mat file in the current directry for the DevData structure, and the variable FuelMass. FuelMass only exists in Energy files. Any files that don't contain DevData and FuelMass will be excluded from the list box.

The "Create Energy Files" button launches the function that creates Energy mat files. The user will be required to select the appropriate Dev log file and the corresponding piccolo telemetry log file if necessary. The function will import the Dev log file data into the DevData structure and create the variable FuelMass from the fuel mass recorded by the piccolo log file; thus, if the motor type is electric the GUI will not require a piccolo telemetry log file.

Additionally the script will automatically adjust the fuel mass data to match the time scale of the DevInterface data since they are not necessarily recorded at the same rate and the same piccolo time stamps. When the Energy mat file is created the GUI will automatically save the mat file in the directory that the list box is currently in and reload the list box so that it contains the new Energy mat file as an option. The mat file will be saved as "Energy" plus the flight name from the Dev log file name (removes the "Dev" portion of the file name).

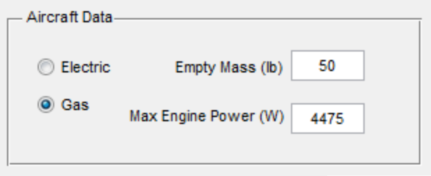

Piccolo Time (mm:ss)

● Input Time

	Minutes	Seconds
Start		
End		

Figure 324 EnergyControl Piccolo Time Section

The "Piccolo Time" section allows the user to enter a time period for the initial plots. The initial plots allow the user to select the time period for analysis by clicking and dragging on the initial plots. The time scale used is the piccolo time which is displayed in the DevInterface during flight. The "Input Time" radio button declares whether or not to use user input time. If user input time is to be used the initial plots will plot only the time period that has been declared by the user; otherwise, the initial plots will include the entire time span of the Energy mat file.


Aircraft Data

○ Electric

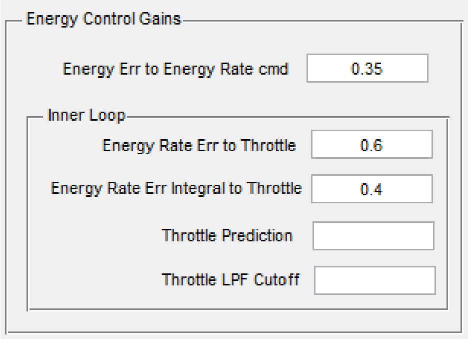
● Gas

Empty Mass (lb): 50

Max Engine Power (W): 4475

Figure 325 EnergyControl Aircraft Data

The "Aircraft Data" section asks for inputs of "Emtpy Mass" and "Max Engine Power". These parameters are only required for "Analyze w/ ERate Cmd". Both of these parameters must match exactly what the piccolo thought they were during the periods for analysis. Both parameters can be found in the Vehicle tab of the Configuration window in PCC. If one of the values was input incorrectly into the autopilot during flight that value still must be used in the analysis because those two values are the values that the autopilot uses to determine how to respond with throttle regardless of whether or not they were the correct values at the time. Additionally there are two radio buttons “Electric” and “Gas”. The radio buttons designate the type of propulsion system. PCC stores the fuel mass estimate for both types of motors. In the case of electric motors the fuel mass estimate is in units of watts and is only used for estimating the w-hr that have been consumed; therefore, it is necessary to distinguish between the two types so that the energy control calculations don’t interpret the electric fuel mass (W) as actual fuel mass (kg). If Electric is selected the GUI will consider fuel mass to be 0 throughout the entire flight and the energy control calculations will use the empty mass value for the mass of the aircraft.



Energy Control Gains

Energy Err to Energy Rate cmd 0.35

Inner Loop

Energy Rate Err to Throttle 0.6

Energy Rate Err Integral to Throttle 0.4

Throttle Prediction

Throttle LPF Cutoff

Figure 326 EnergyControl Gain Inputs

The “Energy Control Gains” section takes inputs for the control loop gains. If the user inputs a gain value that gain value will appear at the top of each plot in the analysis to help the user keep track of which system response occurred with which gains. Additionally the gain “Kpo” is used to calculate the Energy Rate Command and the Energy Rate error.

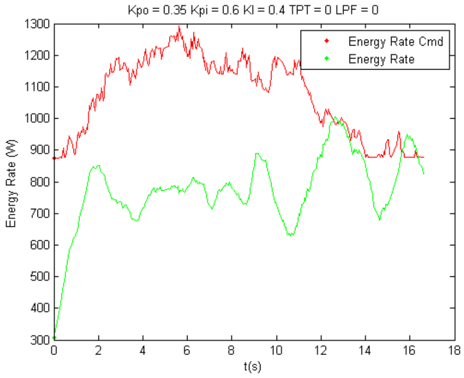


Figure 327 EnergyControl Example Plot with Input Gains

Figure 327 is an example of a plot with all of the energy control gains input. Any gains that are left blank will not appear at the top of the plots. The following table decodes the abbreviations for all of the control loop gains.

Kpo =	Energy err to Energy Rate cmd	TPT =	Throttle Prediction Trust
Kpi =	Energy Rate err to Throttle	LPF =	Throttle LPF Cutoff
KI =	Energy Rate err Integral to Throttle		

Table 6 EnergyControl Gain Abbreviations

There are two different buttons for different analyses. When any one of them is clicked the initial plots will pop up. The initial plots give the user the opportunity to zoom and pan to the desired time period for analysis. If the user inputs a time period the initial plots will all be on one figure; otherwise, each plot will have its own figure. To begin the selection process the user must press “enter” on the keyboard.

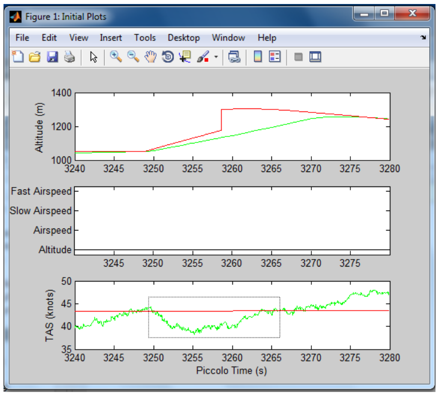


Figure 328 EnergyControl Select Time Period

After pressing enter the user can click, hold, and drag to select a time period for analysis. Figure 328 is an example of the user selecting a time period from an initial plot. Any of the three subplots in the figure could be used to select time from. The initial plots for both buttons are altitude, TAS, and Lon Mode versus time. Lon Mode is included for the user to make sure that the period selected is in Lon Mode 0 only. If any part of the selected period is in any of the airspeed modes there will be a warning message.

The "Analyze w/ ERate Cmd" button launches the in depth energy control analysis where the GUI calculates the Energy Rate command and Energy Rate errors. In order to run the user needs to have input the empty mass, max engine power, and a value for Energy Err to Energy Rate cmd. After clicking the button the user will have to select the time period for analysis then the plots will begin. The function plots the plots shown in the figure below. The data that is in red represents negative errors while the data in blue denotes positive errors.

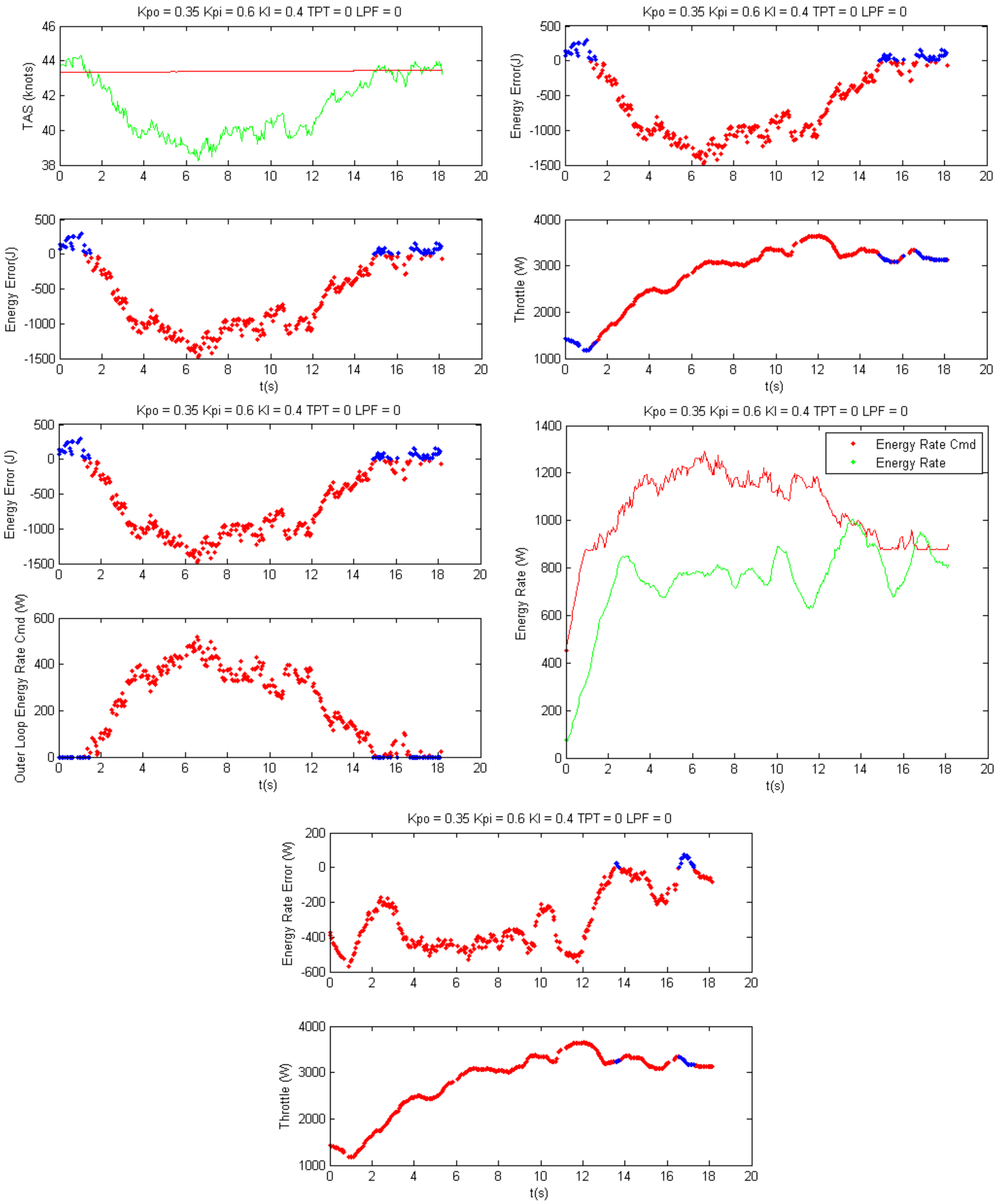


Figure 329 Analyze w/ ERate Cmd Plots

The "TAS vs Throttle" button launches a simple energy control analysis where the user only cares to analyze the throttle response against the TAS data. After clicking the button the user will have to select the time period for analysis then the plots will begin. The function plots the plots shown in the figure below.

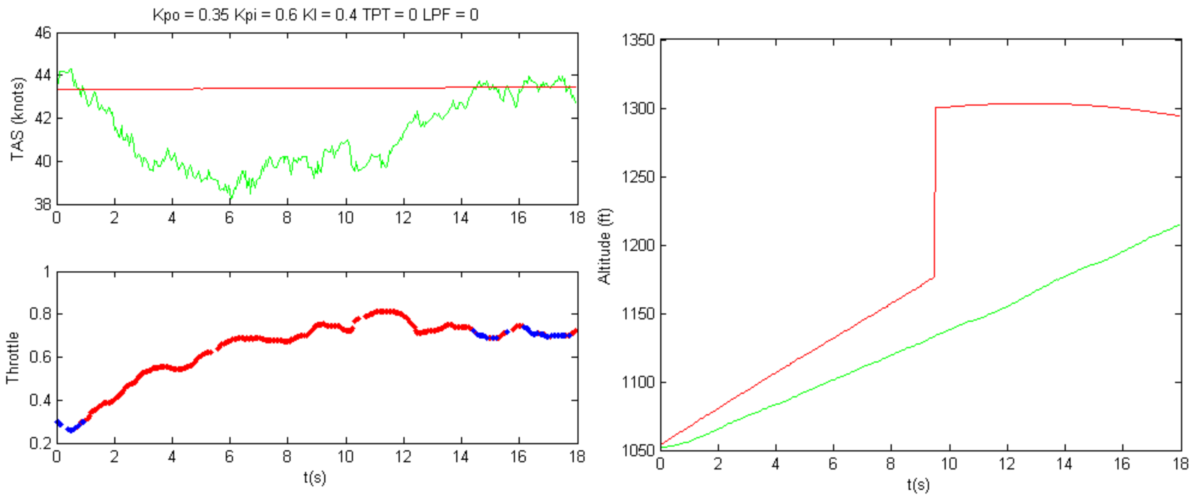
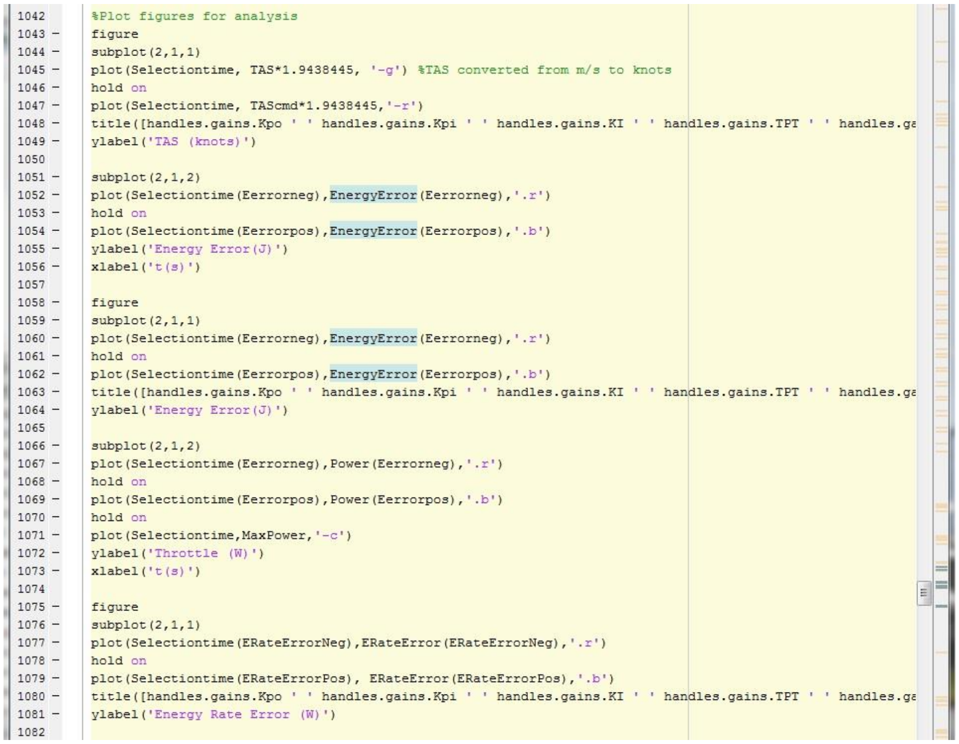


Figure 330 TAS vs. Throttle Plots

It is possible that additional plots could be needed or desired for tuning energy control gains. If that is the case the code can be easily modified by copying and pasting the plot sections that already exist and changing the variables defined in the plot code lines.



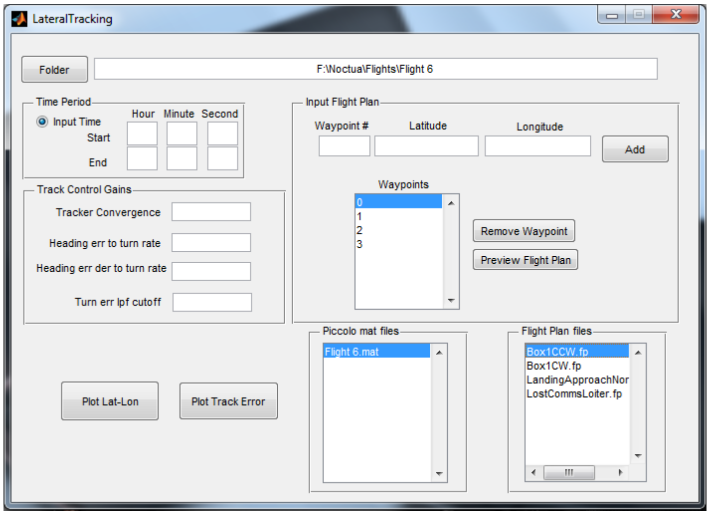
```
%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime, TAS*1.9438445, '-g') %TAS converted from m/s to knots
hold on
plot(Selectiontime, TAScmd*1.9438445,'-r')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' ' handles.gains.TPT ' ' handles.ga
ylabel('TAS (knots)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg),EnergyError(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),EnergyError(Eerrorpos),'.b')
ylabel('Energy Error(J)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(Eerrorneg),EnergyError(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),EnergyError(Eerrorpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' ' handles.gains.TPT ' ' handles.ga
ylabel('Energy Error(J)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg),Power(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),Power(Eerrorpos),'.b')
hold on
plot(Selectiontime,MaxPower,'-c')
ylabel('Throttle (W)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(ERateErrorNeg),ERateError(ERateErrorNeg),'.r')
hold on
plot(Selectiontime(ERateErrorPos), ERateError(ERateErrorPos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' ' handles.gains.TPT ' ' handles.ga
ylabel('Energy Rate Error (W)')
```

Figure 331 EnergyControl Plot Code

Figure 331 is a snapshot of some of the code for EnergyControl.m. Any of the figure codes can be copied and the variables in the plot altered as desired.

7.7. LateralTracking

“LaterlTracking.m” is a MATLAB script that launches a figure that acts as a GUI, “LateralTracking.fig”. LateralTracking is designed to provide the user a user friendly way to analyze track control, or more specifically the track control loop. LateralTracking uses data from the piccolo telemetry log files and plots GPS tracking data such as position (latitude and longitude) and cross track error.



F:\Noctua\Flights\Flight 6

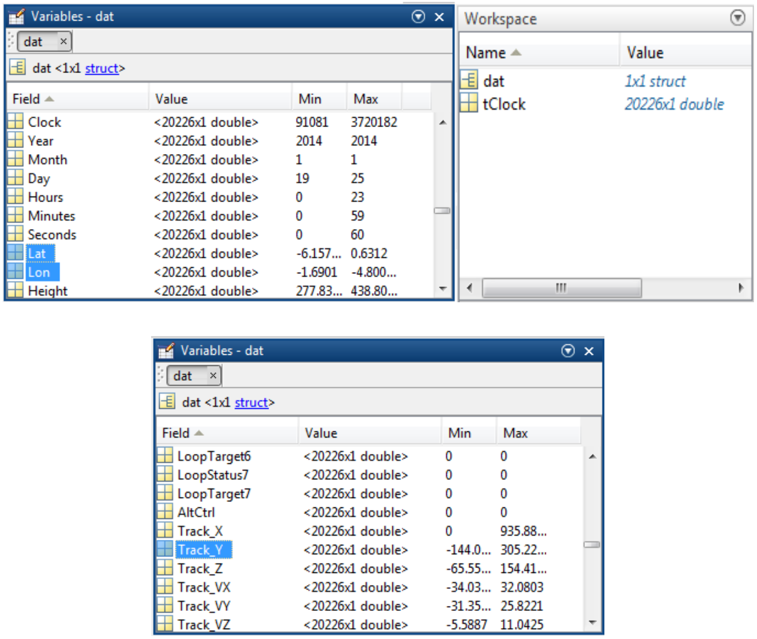
Waypoints:
0
1
2
3

Piccolo mat files:
Flight 6.mat

Flight Plan files:
Box1CCW.fp
Box1CW.fp
LandingApproachNor
LostCommsLoiter.fp

Figure 332 Lateral Tracking GUI

LateralTracking loads piccolo mat files that are selected in the “Piccolo mat files” list box. The list box lists piccolo mat files that exist in the current directory. When the GUI is launched the directory is set to the current MATLAB directory by default. The current directory is displayed at the top of the GUI. The “Folder” button will pull up a dialog box for the user to select another folder as the directory.



Field	Value	Min	Max
Clock	<20226x1 double>	91081	3720182
Year	<20226x1 double>	2014	2014
Month	<20226x1 double>	1	1
Day	<20226x1 double>	19	25
Hours	<20226x1 double>	0	23
Minutes	<20226x1 double>	0	59
Seconds	<20226x1 double>	0	60
Lat	<20226x1 double>	-6.157...	0.6312
Lon	<20226x1 double>	-1.6901	-4.800...
Height	<20226x1 double>	277.83...	438.80...

Name	Value
dat	1x1 struct
tClock	20226x1 double

Field	Value	Min	Max
LoopTarget6	<20226x1 double>	0	0
LoopStatus7	<20226x1 double>	0	0
LoopTarget7	<20226x1 double>	0	0
AltCtrl	<20226x1 double>	0	0
Track_X	<20226x1 double>	0	935.88...
Track_Y	<20226x1 double>	-144.0...	305.22...
Track_Z	<20226x1 double>	-65.55...	154.41...
Track_VX	<20226x1 double>	-34.03...	32.0803
Track_VY	<20226x1 double>	-31.35...	25.8221
Track_VZ	<20226x1 double>	-5.5887	11.0425

Figure 333 LateralTracking dat Variables

Piccolo mat files consist of a structure ‘dat’. Dat contains each column of the piccolo log file as a variable. The 3 main vairables that LaterTracking uses is ‘Lat’,’Lon’, and ‘Track_Y’. Lat and Lon are the GPS lattitude and longitude of the aircraft each time telemetry data is sampled. Track_Y is the cross track error. The GUI searches each mat file in the current directry for the dat structure, and the variable ‘tClock’. Only mat files that have dat and tClock are loaded into the Piccolo mat files list box.

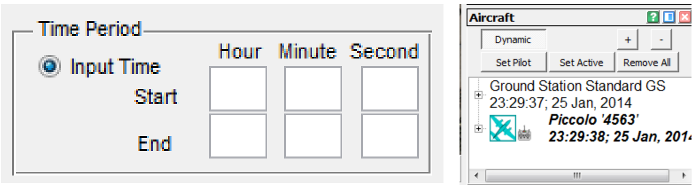
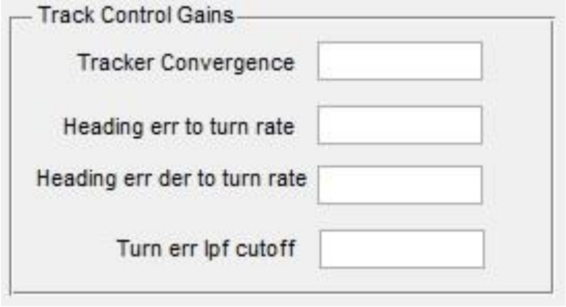


Figure 334 LateralTracking Input Time

The “Time Period” section allows the user to enter a time period for the Lat Lon plot. The “Input Time” radio button declares whether or not to use user input time. The time scale is the GPS clock which appears in the bottom right of the PCC main display showed in Figure 334. If the user chooses not to use input time the Lat Lon plot will span the entire flight.



Track Control Gains

Tracker Convergence

Heading err to turn rate

Heading err der to turn rate

Turn err lpf cutoff

Figure 335 Track Control Gains Input

The “Track Control Gains” section takes inputs for the control loop gains. If the user inputs a gain value that gain value will appear at the top of each plot in the analysis to help the user keep track of which system response occurred with which gains.

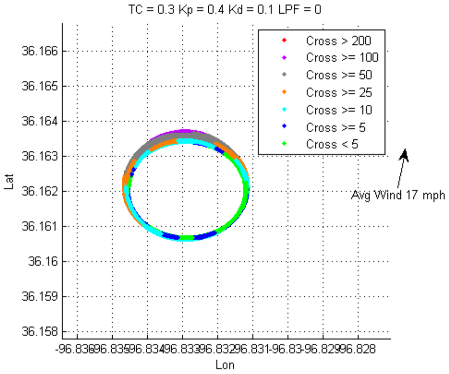


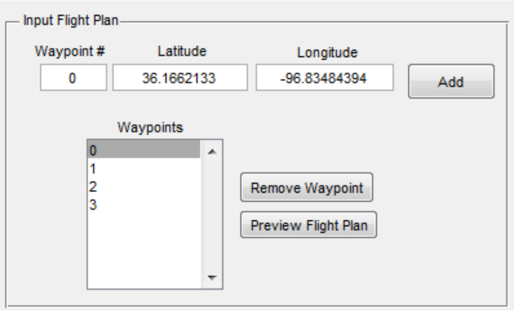
Figure 336 LateralTracking Lat-Lon Plot

Figure 336 is an example of a Lat Lon plot with all of the track control gains input. Any gains that are left blank will not appear at the top of the plots. The following table decodes the abbreviations for all of the control loop gains.

TC =	Tracker Convergence
Kp =	Heading err to turn rate
Kd =	Heading err der to turn rate
LPF =	Turn err lpf cutoff

Table 7 LateralTracking Gain Abbreviations

The “Input Flight Plan” section has a couple of functions. The “Waypoint#”, “Latitude”, and “Longitude” input boxes allow the user to input waypoints by number and location. The units for latitude and longitude are for degrees. The “Add” button adds the waypoint information in the waypoint input boxes to internal waypoint handles and adds the waypoint number to the waypoint list box.



Input Flight Plan

Waypoint # 0

Latitude 36.1662133

Longitude -96.83484394

Waypoints

0
1
2
3

Figure 337 Input Flight Plan

The “Waypoints” list box lists all of the waypoints, by number, that are stored in the internal handles. If the user selects one of the waypoints in the list box the stored information on that waypoint will appear in the waypoint input boxes. The waypoints are used to trace out flight plans on the Lat Lon plots. The GUI will trace from the first waypoint on the list all the way down and back to the first waypoint. The “Preview Flight Plan” button plots the flight plan that currently exists. The “Remove Waypoint” button will remove the selected waypoint from the list box and from the internal handles. Figure 338 below shows the flight plan preview of the waypoints from Figure 337.

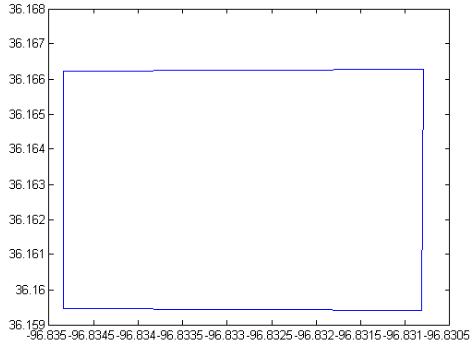
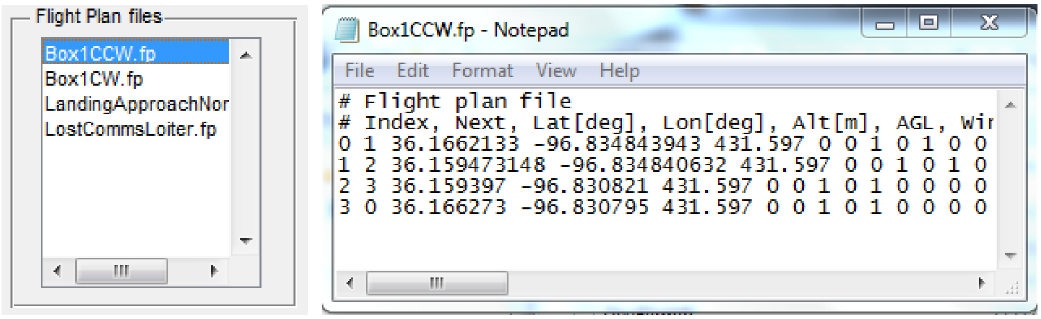


Figure 338 Flight Plan Preview



```
# Flight plan file
# Index, Next, Lat[deg], Lon[deg], Alt[m], AGL, Wir
0 1 36.1662133 -96.834843943 431.597 0 0 1 0 1 0 0
1 2 36.159473148 -96.834840632 431.597 0 0 1 0 1 0
2 3 36.159397 -96.830821 431.597 0 0 1 0 1 0 0 0 0
3 0 36.166273 -96.830795 431.597 0 0 1 0 1 0 0 0 0
```

Figure 339 PCC Flight Plan File

The “Flight Plan Files” list box lists all of the “.fp” files that exist in the current directory. Flight plan files are created when the user saves flight plans in PCC. They are text files that include the latitude and longitude of the waypoint in addition to its number. Figure 339 is an example of the contents of a waypoint file. When a waypoint file is selected the GUI will automatically clear all of the current waypoints that are stored and replace them with the waypoints of the file that was selected.

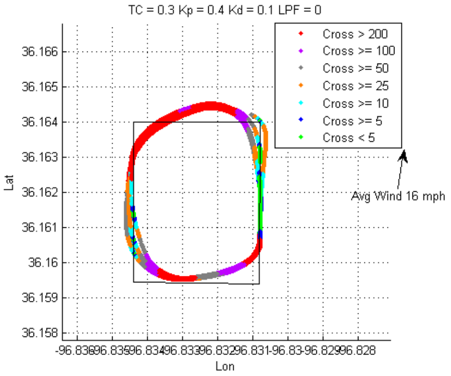


Figure 340 Plot Lat-Lon

There are two different buttons for analysis, "Plot Lat-Lon", and "Plot Track Error". Plot Track Error requires a Lat Lon plot to exist; therefore, Plot Lat-Lon must be run first. Plot Lat – Lon, shown in Figure 340, plots the aircrafts latitude and longitude at each instant that telemetry data was sampled. The colors of the data points that are plotted are dependent on the cross track error. The legend in the upper right corner of the figure describes the color scheme. The units of cross track error are feet. It is important to note that sometimes the aircraft doesn't have enough time to converge on the flight path before it begins to target the next waypoint. Figure 340 is an example of that scenario. The north leg was not long enough and before the aircraft could converge on the flight path the cross track error shot back up, or in some cases stayed, above 200 feet. To the right of the plot on the figure there is a wind arrow that symbolizes the average wind direction during the time period. The text displays the average wind speed in miles per hour.

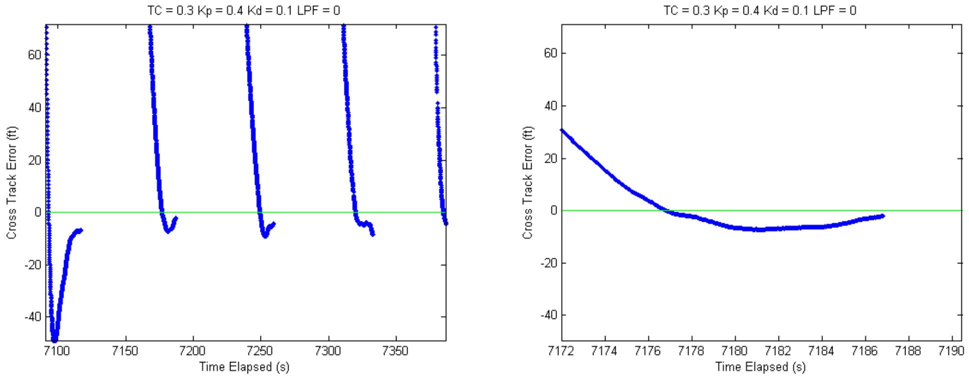


Figure 341 Plot Track Error

Plot Track Error is designed to plot the cross track error versus time for each pass on a leg of a flight plan. Figure 341 shows the result of plotting the east leg of the flight plan shown in Figure 340. The user can zoom and pan to closely analyze each pass, as shown on the right graph of Figure 341. It is important to fly the aircraft on long enough flight paths that the analysis will be useful. If the flight path is too short the aircraft won’t have enough time to attempt to converge on a straight track. When the user clicks the Plot Track Error button there will be command window prompt with instructions on how to properly select a leg of the flight plan for further analysis.

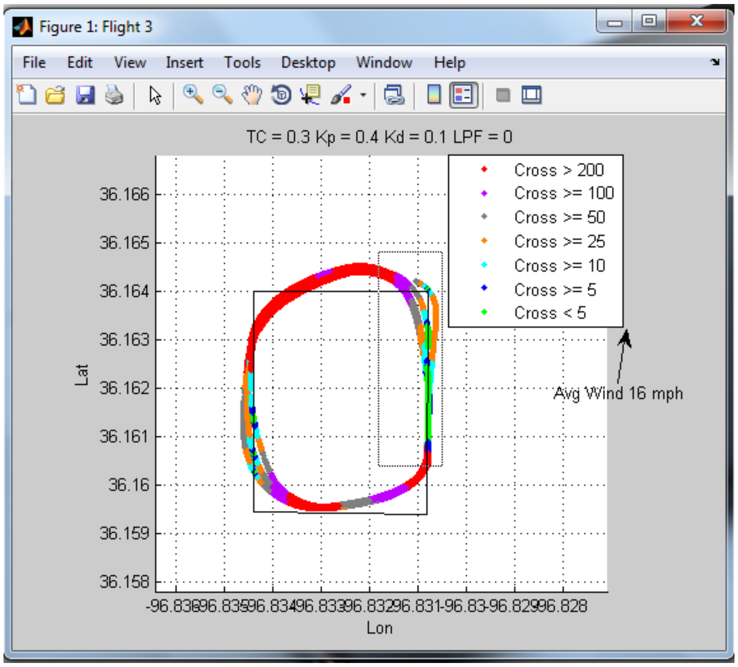


Figure 342 Track Error Selection

Figure 342 shows an example of the selection of a leg of a flight plan. The user selected an area by clicking and dragging a box around the leg of the flight path. The first click must be after the waypoint of the leg has been targeted. The selected area must end when the next waypoint in the flight plan is being targeted. The entire box must include all of the track paths on the selected leg.

7.8. FuelCorrection

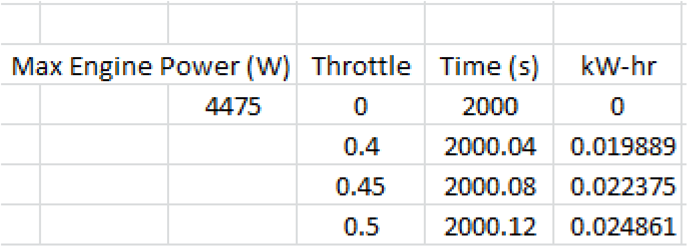
"FuelCorrection.m" is a MATLAB script that is designed to import data from a piccolo mat file and correct the estimated fuel mass throughout the flight based on user input values. The script also calculates the Engine SFC for the flight according to how the piccolo interprets specific fuel consumption.

Piccolo mat files contain the estimated fuel mass throughout the flight in the variable 'dat.Fuel'. The piccolo estimates fuel weight based on Throttle, Max Engine Power, Engine SFC, Gross Mass, and Empty Mass. The piccolo uses the Gross Mass and Empty Mass to calculate the initial fuel mass. The piccolo multiplies throttle by the Max Engine Power to determine the power consumed and multiplies that number by the Engine SFC to estimate the amount of fuel burned.

$$ESFC = \frac{Fuel\ Burned}{\%Throttle * Max\ Engine\ Power * \Delta t}$$

Equation 79 ESFC Calculation

FuelCorrection takes the actual measured fuel burn, from initial and post flight aircraft weights, and combines that with the estimated kW-hr of the flight to calculate a specific fuel consumption based on actual measurements of weight. The script iteratively calculates kW-hr by assuming that each recorded data point of throttle greater than 0 has occurred throughout the time that spanned between the current data point and the previous data point and summing all of the kW-hrs.



Max Engine Power (W)	Throttle	Time (s)	kW-hr
4475	0	2000	0
	0.4	2000.04	0.019889
	0.45	2000.08	0.022375
	0.5	2000.12	0.024861

Figure 343 kW-hr Example Calculation

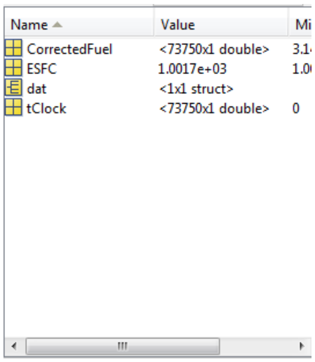
Figure 343 shows an example of a kW-hr calculation where the throttle increased from 0 at just after 2000 seconds. The kW-hrs were calculated for each row using the following equation:

$$kW - hr(n) = Max\ Engine\ Power * Throttle(n) * \big(t(n) - t(n-1)\big)/3600$$

Equation 80 Kilowatt Hours

The process is more accurate for higher telemetry rates, such as 25 Hz. The entire process is mostly an estimate anyways because the piccolo assumes that the throttle settings accurately represent the amount of max engine power that the throttle setting consumes.

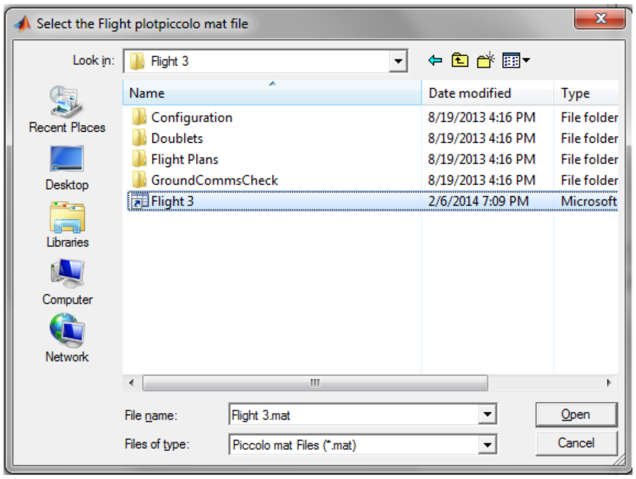
The script will create a new variable for the Engine SFC and it will create a new variable for the fuel mass estimate, ‘CorrectedFuel’. CorrectedFuel is calculated with Equation 79 above by rearranging the equation to solve for the amount of fuel burned at each recorded time stamp which is also the same way that the piccolo estimates fuel mass in flight.



Name	Value	Mi
CorrectedFuel	<73750x1 double>	3.1
ESFC	1.0017e+03	1.0
dat	<1x1 struct>	
tClock	<73750x1 double>	0

Figure 344 FuelCorrection Variables

To use FuelCorrection run the script by right clicking the file in MATLAB and clicking “Run”, or typing “FuelCorrection” in the MATLAB command window. The first dialog box will ask for the user to select the piccolo mat file.



Select the Flight plotpiccolo mat file

Look in: Flight 3

Name	Date modified	Type
Configuration	8/19/2013 4:16 PM	File folder
Doublets	8/19/2013 4:16 PM	File folder
Flight Plans	8/19/2013 4:16 PM	File folder
GroundCommsCheck	8/19/2013 4:16 PM	File folder
Flight 3	2/6/2014 7:09 PM	Microsoft

File name: Flight 3.mat

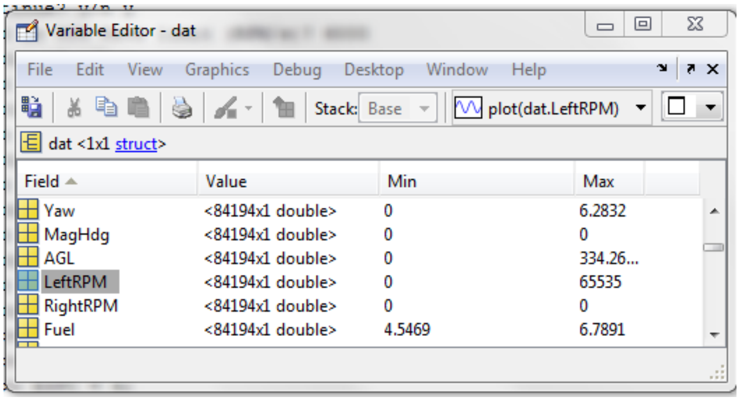
Files of type: Piccolo mat Files (*.mat)

Figure 345 Select Piccolo Mat File

After the user selects the piccolo mat file the script will ask for the max engine power and there will be a series of input questions regarding weight. The script will ask for the gross takeoff weight, empty weight, and final weight. All of the weights will be asked for in lb. The script performs all of the calculations in kg. After the user enters all of the required weights the script will create the new variables and automatically save the piccolo mat file with the new variables, same mat file name, and in the same location as the original mat file. The corrected mass estimate can be plotted with AnalyzePiccolo.

7.9. PiccoloRPMRateFilter

"PiccoloRPMRateFilter.m" is a MATLAB script that is designed to filter RPM data from piccolo telemetry log files by the same method as the piccolo "RPM rate filter". The script should be used to help the user determine if the RPM rate filter would be useful and what the setting should be.



Field	Value	Min	Max
Yaw	<84194x1 double>	0	6.2832
MagHdg	<84194x1 double>	0	0
AGL	<84194x1 double>	0	334.26...
LeftRPM	<84194x1 double>	0	65535
RightRPM	<84194x1 double>	0	0
Fuel	<84194x1 double>	4.5469	6.7891

Figure 346 LeftRPM Variable

PiccoloRPMRateFilter works by loading the RPM data from ‘dat.LeftRPM’ in the user selected piccolo mat file. The script copies each RPM data point, one by one, to a new variable “RPMFiltered”. Based on the user input frequency limit (RPM/s) the script will assign RPM data that violates the limit the value of the previous RPM data point. The process operates in accordance with how the piccolo filters the RPMs based on the RPM rate limit. The script allows the user to iteratively change the frequency and view the impact it will have on the RPMs without having to run the script over and over again for different values. In the end the script will ask the user if they wish to save. If so the script will save over the mat file it loaded with the new variable RPMFiltered. The new variable can be plotted with AnalyzePiccolo.

The script can be ran by right clicking PiccoloRPMRateFilter.m in MATLAB and clicking “Run” or by typing “PiccoloRPMRateFilter” in the command window. The first prompt for the user will be a dialog box asking for the piccolo mat file.

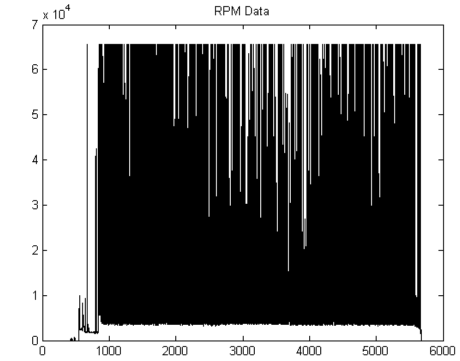


Figure 347 Unfiltered Noisy RPM Data

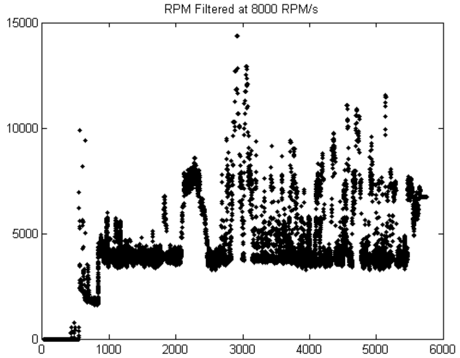
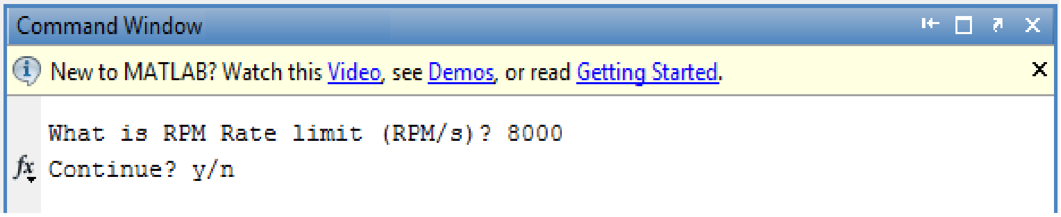


Figure 348 Filtered Noisy RPM Data

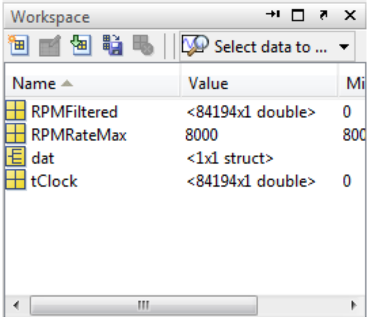
After the mat file has been selected a figure will appear that plots the RPM data that was recorded by the piccolo in dat.LeftRPM. The plot is titled “RPM Data”. Figure 347 is an example of very noisy RPM data. This plot is used for the user to compare the filtered plots against the recorded plots. In the command window there will be a prompt asking the user for the RPM rate limit. After the user types in a value a new figure will popup displaying the filtered RPM data with the RPM rate displayed in the title of the plot and figure, shown above in Figure 348.


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
What is RPM Rate limit (RPM/s)? 8000
Continue? y/n
```

Figure 349 Continue Command Window Prompt

In the command window there will be a prompt that asks if the user wants to continue. The user must type “y” to continue. Anything other than “y” will be interpreted as “no”. The user can keep trying different RPM rates as desired. Each time there will be a new plot of the resulting RPM data. If the script is told not to continue a dialog box will pop up asking the user if they want to save the filtered rpm data.



Name	Value	Mi
RPMFiltered	<84194x1 double>	0
RPMRateMax	8000	800
dat	<1x1 struct>	
tClock	<84194x1 double>	0

Figure 350 RPM Filter New Variables

The user must enter “y” to save the mat file. If the user elects to save the new variables ‘FilteredRPM’ and ‘RPMRateMax’ will be saved in the piccolo mat file that was selected.

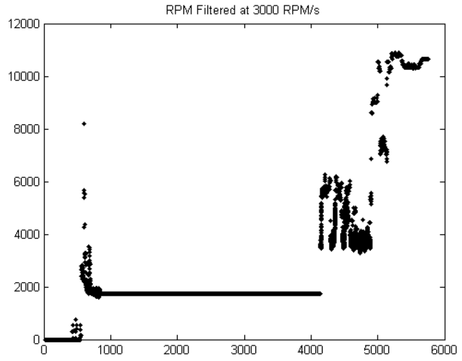


Figure 351 RPM Filter Too Low

This script can help the user determine how much noise can be filtered out using the piccolo's RPM Rate filter. Figure 351 is an example of what can happen if the RPM Rate limit is too low.

CHAPTER VIII

FLIGHT PROCEDURES

8. Introduction

The flight procedures outlined in this chapter are designed primarily for Fixed Wing Generation 2 version 2.1.4. The equipment list and control room setup are designed specifically for OSU's Noctua team operating at the OSU UAFS airfield. The gain tuning methods are designed for tuning autopilots operating in altitude control.

8.1. Equipment

The equipment listed is specific to Oklahoma State University and the OSU UAFS airfield. The DML Equipment is everything that needs to be loaded from the DML and taken to the airfield.

8.1.1. DML Equipment

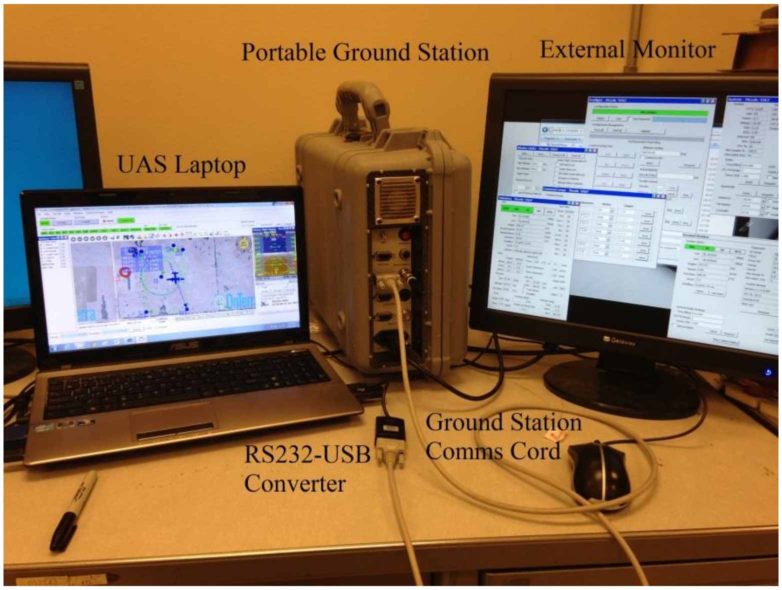


Figure 352 DML Equipment

1) UAS ASUS Laptop

2) External Monitor

3) Portable Ground Station w/ Power Cord

4) RS232-USB Converter

5) Ground Station Comms Cord

6) HiL USB-CAN Converter

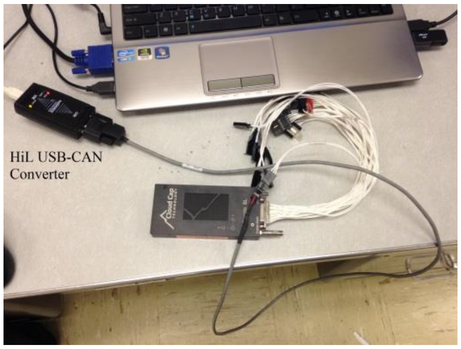


Figure 353 HiL USB-CAN Converter

7) Ground Station Futaba Transmitter

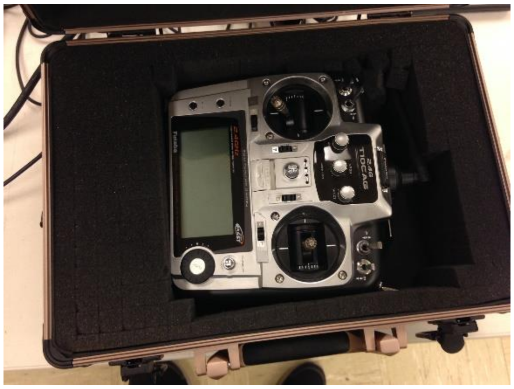


Figure 354 Ground Station Futaba Transmitter

8) JR Spektrum Transmitter (if applicable)

9) Piccolo Toolbox

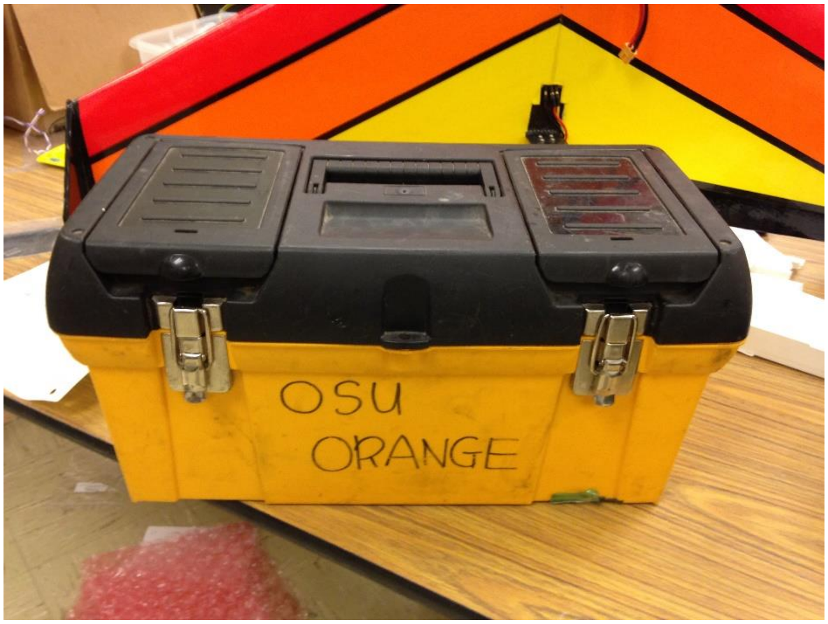


Figure 355 Piccolo Toolbox

Which should include:

- Futaba Transmitter to Ground Station chord

- Piccolo Programmer

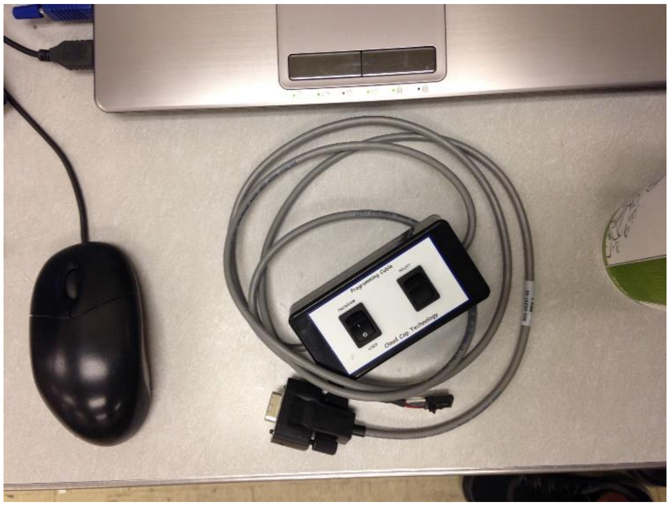


Figure 356 Piccolo Programmer

- Spare Comms Antennas

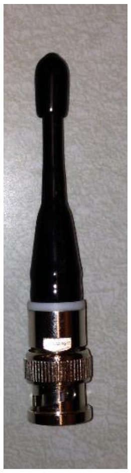


Figure 357 Spare Communications Antenna

- GPS Module w/ GPS Ground Plane
- Any adapters that could be needed to connect to the autopilot on the aircraft
- Any necessary accessories

10) Power Supply

- Use the power supply from the UAS workstation at the DML

11) Flash Drive (personal flash drive, there is no UAS flash drive)

8.1.2. Control Room Equipment

1) UAS Control Room Desktop
2) Control Room Monitor
3) Control Tower antenna cable
4) Network Router

8.2. Control Room Setup

The control room setup is designed for systems utilizing the JR Level Shifter board so that the manual pilot does not have to be attached to the ground station. If a JR Level Shifter board is not used then alterations to the setup will have to be made.

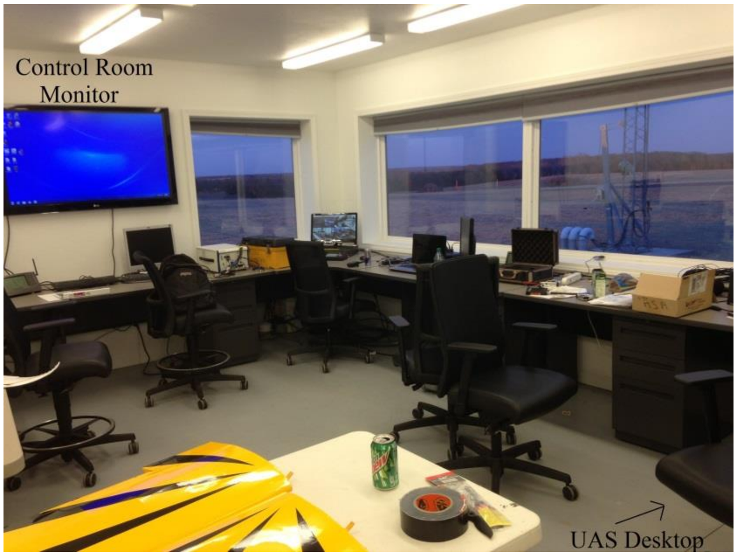


Figure 358 Control Room Overview

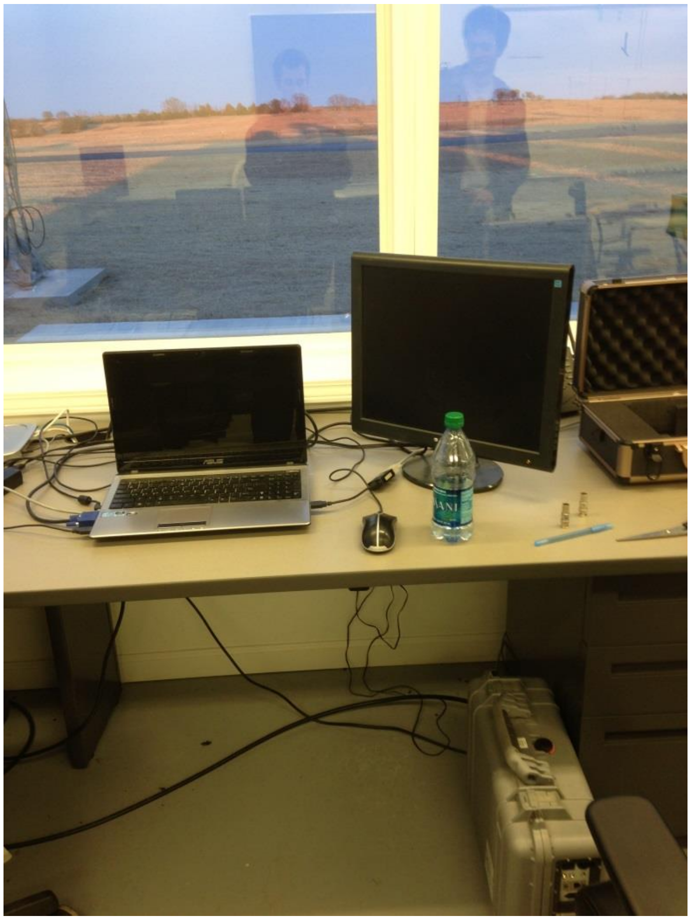


Figure 359 Flight Operator Work Station

Figure 360 UAS DML Power Supply

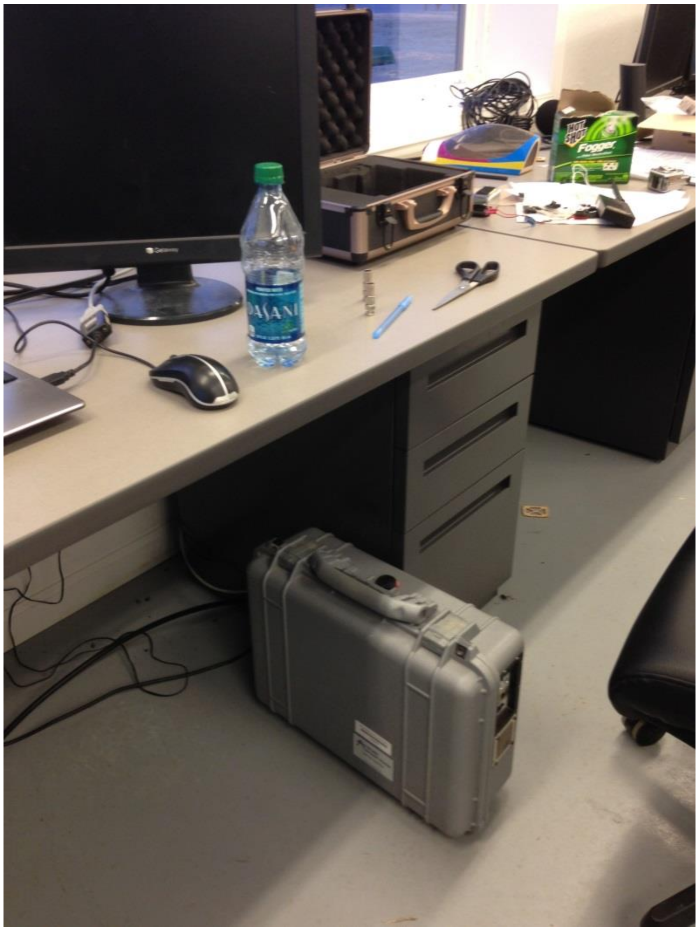


Figure 361 Ground Station Configuration

As shown in the pictures the UAS Laptop, External Monitor, and Portable Ground Station are setup in front of the east side windows. The power supply sits behind the UAS Laptop on the table. The Portable Ground Station should be placed underneath the desk.

1) Plug the Portable Ground Station, UAS Laptop, and External Monitor into the power supply on the battery backup connections. There should be double battery backup redundancies for the ground station and laptop with the battery backup on the power supply and the internal batteries that both components have.

2) Connect the Portable Ground Station to the UAS Laptop with the R232 – USB Converter and the Ground Station Communications cord. The cord should run through the back of the desk rather than hang over the front of the desk.
3) The control tower antenna cord should be resting underneath the desk upon arrival, and should have plenty of slack to reach the Portable Ground Station. Connect the control tower antenna cord to the Portable Ground Station. Make sure that there is enough leg room underneath the desk for the pilot.
4) Make sure that the ground station is all the way underneath the desk to avoid accidentally kicking it over.
5) Disable the wireless network connection on the UAS Laptop and connect it to the network router with the spare ethernet port and cable which should already be plugged into the router.
6) Connect the External Monitor to the UAS Laptop. It is highly recommended to use at least two screens while piloting the piccolo.
7) Power on the UAS Laptop prior to powering on the Portable Ground Station. If the ground station is power on while the laptop is booting up the laptop will interpret the RS232-USB connection as a mouse and mouse cursor will randomly move and click.
8) Log on to the “Piccolo” user account on the UAS Control Room Desktop. Set the screen settings so that the desktop is dual screened with the Control Room Monitor.

8.3. Flight Planning

A flight plan should be made for every flight. The plan should include all of the flight patterns that the aircraft is expected to fly along with the objectives for the flight. Note that changes to flight plans during flight are inevitable and okay, such as altering waypoints and flight patterns, as long as they are done so reasonably and with caution.

Use the following guidelines for flight planning:

1) Create detailed flight plans with planned flight patterns and clear objectives.

2) Create a lost comms waypoint if one does not already exist. The lost comms waypoint should always be a loiter point near the control tower; however, make sure it is not too close to the control tower and control room.

3) Save the planned PCC flight plans on the UAS Laptop where they can be easily loaded during pre-flight.

4) Fly every flight pattern in simulations. Make sure to practice the entire flight, sending the aircraft to and from all the different planned flight patterns.

5) Always have auto land flight patterns developed for landing from both the north and south directions regardless of whether or not the autopilot is ready for attempting auto lands. Make sure to test the auto land flight patterns in simulations.

6) Practice any planned flight test maneuvers in simulations.

7) For flights containing doublet maneuvers:

    a. Setup specific flight patterns that adhere to the doublet maneuver guidelines for testing each axis that is desired.

    b. Experiment in hardware in the loop simulations with different durations, periods, and deflections to develop a good plan.

    c. Plan for at least 6 doublet tests for each axis. The tests should consist of at least two small deflections, two moderate deflections, and two large deflections (but not too large). It can be difficult to determine what deflections can be classified as moderate and large just by using the hardware in the loop simulation. Some in flight adjustments to the planned deflections could be necessary.

d. Practice conducting the planned maneuvers in hardware in the loop simulations.

e. Practice using both the Doublet File Method, and the Piccolo Log File Method to analyze the results.

f. Theoretically the results should match the simulator estimated values; however, they will not always be exactly the same. If the results don’t match try changing the planned doublet settings, such as the period and duration to see if they will make the results match more closely to the estimated values. Don’t worry if the results never match, and don’t change the vehicle parameters based on the results from the hardware in the loop simulations.

8) If planning to tune control gains setup flight plans according to the guidelines for the appropriate control loops if they exist. Practice altering control loop gains and analyzing their response in HiL simulations. Develop an understanding for how each gain, in the control loop in question, can affect the system response.

8.4. First Flight

The first flight requires some extra planning and tasks to be performed to prove certain systems are airworthy and ready for flight.

8.4.1. Flight Planning

Use the flight planning procedures outlined in Section 8.3 along with the following extra procedures:

1) Read and know issues that have caused crashes in the past so they are not repeated, Appendix A.

2) Make sure to understand the control laws and how they operate as described in Chapter 5.

3) Make sure to understand how to operate all of the MATLAB scripts that are associated with gain tuning, Chapter 7, and doublet maneuvers from Chapter 6.

4) Practice, in normal HiL simulations, performing the gain tuning operations for all of the control loops. Alter gains and analyze the system response to develop a familiarity with how each gain can affect the performance of the autopilot. Make sure to be familiar with how the gain tuning MATLAB programs operate.

5) Perform a ground test with the aircraft setup in a hardware in the loop simulation. The aircraft should be fully operational, including throttle if possible, so all of the surface deflections can be observed while the autopilot flies in the simulator.

    a. With the entire system setup fully operational in a HiL begin the simulation with a manual r/c pilot take off and transfer control of the aircraft to the autopilot as planned.

6) Plan to perform aileron, elevator and rudder doublet maneuvers.

7) With the entire system setup fully operational in a HiL practice commanding doublet maneuvers. Make sure that the appropriate surfaces are indeed deflecting as they should be.

8.4.2. Pre-Flight

Before performing the normal Pre-Flight operations (Section 8.5) perform the following tests:

1) With the autopilot mounted inside the aircraft and powered on transport the aircraft up and down the runway and around the area of the UAS Airfield to test both the piccolo comms link and the manual r/c pilot comms link.

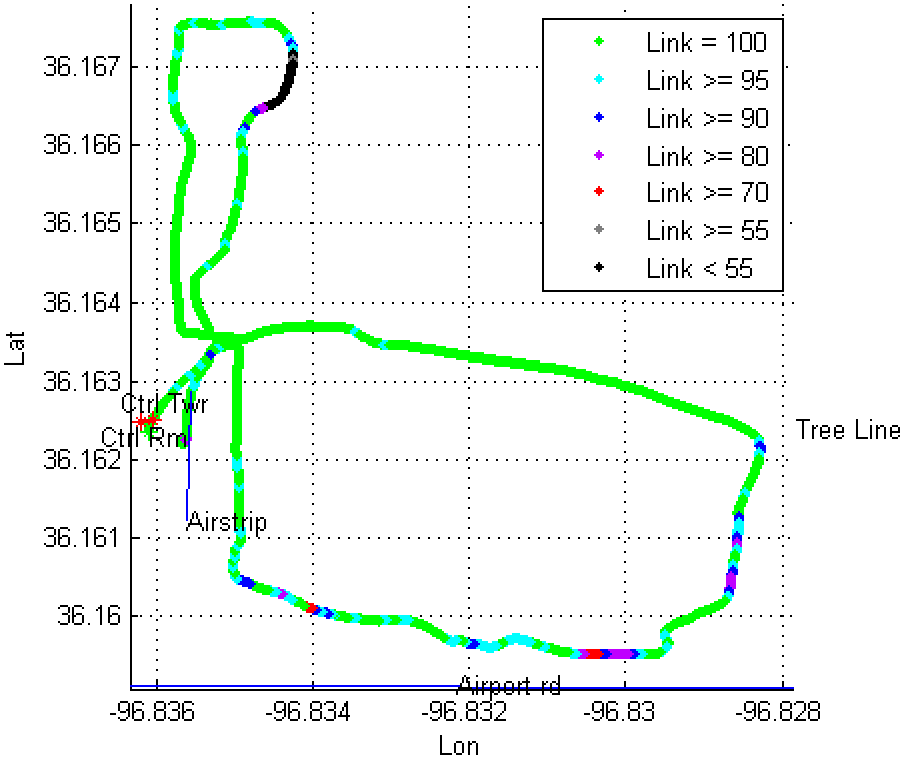


Figure 362 Ground Comms Check

Figure 362 depicts an example from Noctua B1 Flight 3. When analyzing the results keep in mind that some comms losses could be due to the fact that the aircraft is not airborne with the fuselage or possibly other objects blocking the line of sight. Generally speaking there should not be any packet loss with the piccolo communications link if there is no loss of line of sight.

2) Run the motor and allow it to warm up. Then test that the RPMs match the percent throttle that they were calibrated for. If they are way off re-calibrating throttle could be necessary. It doesn't have to be prefect as temperature and other effects can cause the engine performance to vary.

3) Continue performing all the normal Pre-Flight operations outlined in Section 8.5.

8.4.3. First Flight

In addition to the flight guidelines in Section 8.6 make sure to do the following:

1) Have the manual r/c pilot fly the aircraft around the airfield at varying distances and watch the "Link" and "Pilot Sample" (if using JR) to observe the signal strength of the communications.

2) Keep a close eye on the piccolo comms signal (Link) and the manual JR signal (Pilot Sample), if using JR. It may be necessary to use the "LinkPos" and "ManualPilotConnection" functions in "AnalyzePiccolo.m" to determine if there are any signal strength problems during flight, or this can be done post flight.

3) Enable Controller Telemetry at the full 25 Hz before sending control over to the autopilot.

4) Instruct the manual r/c pilot to switch over to autopilot control as planned.

5) If there are obvious control issues gain tuning will need to be done, go to Section 8.10.2. If there are no obvious control issues continue to step 6.

6) Perform Aileron Doublets.

7) Perform Elevator Doublets.

8) Perform Rudder Doublets.

9) Analyze Doublet Results and make changes to the Vehicle Gains as necessary.

10) Test C_L Max according to Section 8.9.1.

11) Test C_L at zero elevator according to Section 8.9.2.

12) Test slow airspeed mode to make sure the elevator can properly increase the airspeed with out a large loss of altitude. Put the aircraft in a scenario where it will enter Lon Mode 2, such as a hard climb. Make sure there is plenty of altitude and the manual pilot is aware of what is happening.

13) Test fast airspeed mode to make sure the elevator can properly decrease the airspeed without attempting to climb. Put the aircraft in a scenario where it will enter Lon Mode 3, such as a steep descent or a low Fast IAS Error Threshold. Make sure there is plenty of altitude and the manual pilot is aware of what is happening.

8.4.4. Post-Flight Analysis

Use the script “AnalyzePiccolo.m” to analyze the flight. Analyze any parameters that are applicable to the system including the following:

1) Piccolo and manual r/c pilot signal strengths.

2) Check the accelerometer measurements for resonant vibrations.

3) Use the script “AnalyzeDevInterface.m” to analyze the performance of the command loops such as Altitude performance after climbs and descents. Look for over shoots or undershoots in performance that might not have been noticed in flight.

4) Check the air density recorded by the doublet files if they are available. Make sure that the piccolo air density estimate is close to that which is expected.

5) Check control surface deflections. Make sure they are functioning as expected, and that the aircraft wasn’t put in any situation where the maximum deflections were hampering autopilot performance.

8.5. Pre-Flight

The Pre-Flight checklist (Appendix F) contains the standard general piccolo system pre-flight setup. It is up to the user to add any aircraft specific requirements that may exist as each platform is different.

8.6. Flight Guidelines

During Flight:

1) Frequently check the:

   a) IAS

   b) Altitude Tracking

   c) Lateral Tracking

2) Occasionally check:

   a) Piccolo Voltage / Current

   b) Servo Voltage / Current

c) Piccolo Temperature (80° C Max.)

d) Link

e) Pilot Sample

3) Do not leave the UAS Laptop unattended while the autopilot is in control. This means that if control gain tuning needs to be done there must be a co-pilot or someone to watch the autopilot vitals while the user is performing the necessary analyses on the UAS Control Room desktop.

4) Notify the r/c manual pilot when making any changes such as targeting a new waypoint or performing a doublet maneuver.

5) Do not let anyone coerce you into deviating from the flight plan. If someone is distracting in anyway do not hesitate to kick them out of the control room.

8.7. Post Flight Analysis

After each flight a post flight analysis should be performed. Generally speaking it is good to check the communication signal strengths after every flight to make sure that no communications issues arise. The analysis should focus on the objectives of the flight. The analysis should also include checking command loop performance, such as airspeed and altitude, to make sure that no unknown issues occurred during the flight. Take each command loop down to the inner control loops such as analyzing vertical rate. Also check to see if the autopilot ever switched Longitudinal Modes and determine whether or not that is desirable and what to do to avoid that if necessary. Use AnalyzePiccolo and AnalyzeDevInterface to help assess autopilot performance.

8.8. Doublet Maneuvers

Doublet maneuvers can be performed manually by the manual r/c pilot or by the remote pilot via PCC. The following procedures detail instructions for performing doublets using PCC. It can be beneficial to have the manual r/c pilot perform doublet maneuvers if the autopilot is unable to maintain flight without divergently oscillating in some manner. If the maneuvers are performed manually by the pilot there are a few guidelines in each section to follow to do so. Both MATLAB scripts, "DoubletPiccoloLog.m" and "plotdoublet.m", are applicable for analyzing doublet maneuvers conducted through PCC; however, only DoubletPiccoloLog can be used to analyze doublet maneuvers conducted manually by the r/c pilot.

Before beginning note the following warning for specific doublets:

1) Aileron Doublets

    a. Try to conduct aileron doublets when the aircraft is flying into the wind, and avoid conducting aileron doublets where the aircraft experiences a cross wind. Since the aileron effectiveness calculation uses airspeed measurements cross winds can cause noise in the airspeed data; thus, creating noise in the measured dimensionless roll rate.

2) Elevator Doublets

    a. Make sure the negative deflections (pitch up) commanded are not high enough to cause the aircraft to stall. There are no protections against stall during doublet maneuvers and stall will compromise the results.

3) Rudder Doublets

    a. If the Doublet File Method is to be used do not conduct the rudder doublets where the aircraft heading will cross 180° or 0° (North and South). Since the doublet files record true heading from the controller telemetry for use in rudder doublet analysis the reported heading will have “jumps” in the heading data that completely ruin the doublet analysis. In addition to the jumps in heading the recorded heading will also create gaps from changing notation since anything above 180° becomes negative values.

To begin setup a simple box pattern with the following guidelines:

1) Close proximity to the control tower so that there is no danger of lost comms during testing.

2) High enough AGL so that there is no danger of crashing into elevated objects since the aircraft will deviate from the flight paths during doublet testing.

3) Constant altitude waypoints. There should not be any climbs or descents.

Use the following steps to setup and conduct the doublet maneuvers.

1) If using the Doublet File Method make sure to disable Controller Telemetry.

2) In the Surface Calibration window select the appropriate surface to be deflected

3) Check “Autopilot Trim” so that the autopilot will deflect the surface from the trim position. If autopilot trim is not selected the autopilot will hold a surface deflection of 0 degrees for 1 second before it deflects them the commanded value.

4) Set the “Period” so that it is long enough that the measured response will settle at a peak value before the surface deflection returns to the trim setting. It can be useful to try one doublet and analyze the results to make sure that the period is not too short or too long. Remember that the period is defined in milliseconds!

5) Check “Disable off axis” to disable the other control surfaces from being deflected during the test.

6) Select the appropriate type of axis to be tested from the “Axis” drop down menu.

7) Set the “Duration” so that it is long enough to capture the entire doublet test. Remember that the autopilot will hold the trim setting for 1 full second before deflecting the control surface. Generally it is a good idea to add at least 1 ½ seconds to the period to allow for the initial 1 second and to ensure PCC will record the results of the return to the trim setting. Also recall that the Duration simply sets the amount of time that PCC will record the doublet data.

   If the doublet files are not intended to be utilized set the duration to 0 so that PCC will not record a text doublet data file. The autopilot will command the surface deflection for the length of the period setting regardless of the duration setting.

8) Set the desired deflection.

9) When the aircraft is flying along one of the straight legs of the flight plan, and not in the middle of a turn, click “Send” to command the doublet maneuver.

   Notify the manual pilot just before issuing the doublet command so that pilot is aware of what the autopilot is attempting to do. Note that the manual pilot does have the ability to takeover full control during doublet maneuvers.

10) Record each doublet using the “Doublet Maneuvers” sheets.

11) When all of the doublets have been conducted for the desired surface, continue to the analysis instructions.

8.8.1. Doublet Analysis

Use the instructions from Section 6.4 to analyze the doublet maneuvers along with the following logistical instructions for analyzing doublets in the control room. It is up to the user to determine if the aircraft should land or loiter while the analysis is being done.

1) Use one of the two programs that correspond with the method used, DoubeltPiccoloLog or plotdoublet.

2) Use the UAS Control Room Desktop to run MATLAB. If using the log file doublet method copy the piccolo log file, from the UAS Laptop and run the script "CreateMatFile" to generate the piccolo mat file.

3) Use the excel spreadsheet "DoubletResults.xlsx" to record the results from the analysis. Make sure to fill in all the details of each test (Doublet Name, Duration, Period, Deflection).

4) If a test contains doublet data that has too much noise or doesn't have a clear enough peak to evaluate then put an "N/A" in the result area of the test. If a lot of the doublet results for a specific surface are excluded then do another set of doublet tests for the surface in question. Also check the following to narrow down what could cause the doublet data to be insufficient:

    a. If there are simply not enough data points, and data points seem to be missing it is likely that the doublet maneuvers are being conducted in a location where there is packet loss between the ground station and the autopilot. Try conducting the doublets in a different location.

b. If there is noise in the CL measurements (Elevator Doublets) it is likely that there is Z – Accelerometer vibrations. Try a couple more tests to see if the vibration issues continue. Sometimes some tests just don't work.

c. If there is noise in the dimensionless roll rate measurements (Aileron Doublets) then it is likely that there are some pitot tube issues (airspeed measurements). Try conducting the aileron doublets where there isn't any crosswind.

8.8.2. Aileron Doublet Results

Use the "DoubletResults.xlsx" spreadsheet to drop the highest and lowest values and average the rest of the results. The aileron doublet results are in the correct units to be input directly into PCC in the controller configuration window. Compare the results with the simulator estimated value. If the test results seem too far off of the simulator estimated value then conduct another set of aileron doublets and compare those results against the simulator estimated result.

Enter the value of the new Aileron Effectiveness parameter into the "Vehicle" tab of the "Controller Configuration" window in PCC and send it to the autopilot.

8.8.3. Elevator Doublet Analysis

Use the "DoubletResults.xlsx" spreadsheet to drop the highest and lowest values and average the rest of the results. The elevator doublet results are calculated as /radians. The input box in PCC calls for the elevator effectiveness in /degrees. The DoubletResults spreadsheet automatically converts the result into /degrees and is highlighted yellow. Compare the results with the simulator estimated value. If the test results seem too far off of the simulator estimated value then conduct another set of elevator doublets and compare those results against the simulator estimated result.

There is another parameter that was estimated by the simulator model, the “Elevator Power”. If the elevator effectiveness parameter needs to be changed then the elevator power also needs to be changed. Take the percent difference of the measured and estimated elevator effectiveness and apply that percentage to the estimated elevator power. Note that the simulator estimated elevator power, from the “Vehicle” file, is in /radians. Make sure to convert the elevator power to /degrees (elevator power * π / 180). The DoubletResults spreadsheet is setup to calculate elevator power, and convert it to /degrees, automatically.

Enter the values of the new Elevator Effectiveness and Elevator Power parameters into the “Vehicle” tab of the “Controller Configuration” window in PCC and send it to the autopilot. These values should both be in /degrees.

8.8.4. Rudder Doublet Analysis

Use the DoubletResults.xlsx spreadsheet to drop the highest and lowest values and average the rest of the results. The rudder doublet results are dimensionless (rad/rad or deg/deg); therefore, the calculated rudder effectiveness does not need to be converted before it is input into PCC. The input box in PCC calls for the elevator effectiveness in /degrees. Compare the results with the simulator estimated value. If the test results seem too far off of the simulator estimated value then conduct another set of rudder doublets and compare those results against the simulator estimated result.

There is another parameter that was estimated by the simulator model, the “Rudder Power”. If the rudder effectiveness parameter needs to be changed from the simulator estimated value then the rudder power also needs to be changed. Take the percent difference of the measured and estimated rudder effectiveness and apply that percentage to the estimated rudder power. Note that the simulator estimated rudder power, from the “Vehicle” file, is in /radians.

Make sure to convert the rudder power to /degrees (rudder power * π / 180). The DoubletResults spreadsheet is setup to calculate rudder power, and convert it to /degrees, automatically.

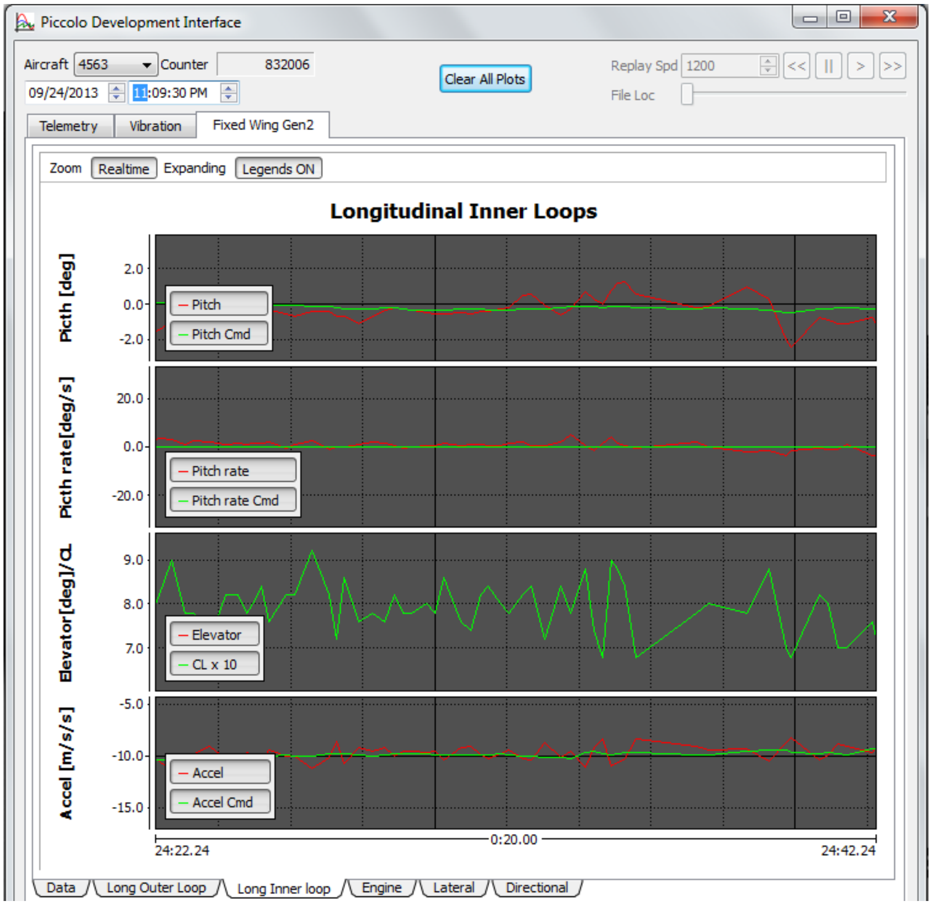
Enter the values of the new Rudder Effectiveness and Rudder Power parameters into the "Vehicle" tab of the "Controller Configuration" window in PCC and send it to the autopilot. Rudder Effectiveness should be dimensionless and come straight from the results without any conversion. Rudder Power should be converted from /radians to /degrees.

8.9. Flight Testing

8.9.1. CL max

Determining $C_{L\ max}$ can be dangerous as the pilot will have to take the aircraft to or near stall. The $C_{L\ max}$ parameter doesn't have to be exactly $C_{L\ max}$; it can be slightly lower just to be safe. Already the user should have an estimate as to what $C_{L\ max}$ is and it should be set safely enough for the aircraft to fly.

There are a couple of options for the user to determine the maximum C_L that was obtained during the flight test. The user can simply observe the C_L values reported by the DevInterface and note the C_L value when the aircraft stalled or neared stall. The C_L values are displayed in the DevInterface, "Long Inner Loop" tab as C_L * 10 shown below in the figure below.



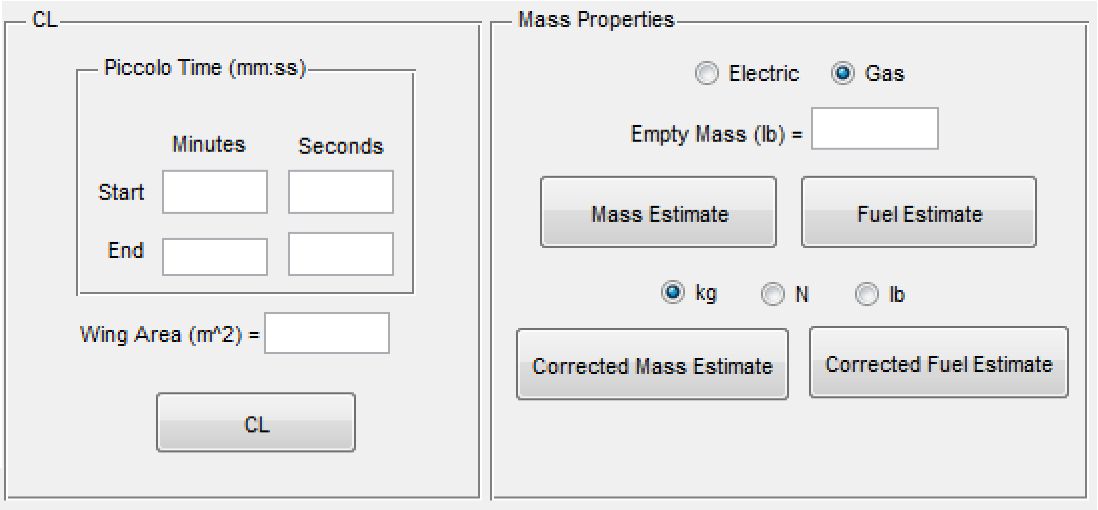
Aircraft 4563 Counter 832006

09/24/2013 11:09:30 PM

Replay Spd 1200

Unfortunately the DevInterface log file does not record C_L values; therefore, the DevInterface log file cannot be used to go back and view the C_L values with the script “AnalyzeDevInterface.m”.

The other option is to analyze the PCC telemetry data via the MATLAB script “AnalyzePiccolo.m”. AnalyzePiccolo can estimate C_L based off of the plotpiccolo mat file.



○ Electric ● Gas

● kg ○ N ○ lb

The figure above shows a snapshot of the C_L and Mass Properties portion of AnalyzePiccolo. To calculate C_L the program will need the start time, end time, wing area, and empty takeoff mass. The time scale uses the piccolo time which is the time reported by the DevInterface.

Already the user should have an estimate as to what $C_{L\ max}$ is and it should already be set safely enough for the aircraft to fly. Both the C_L displayed by the DevInterface and the C_L calculated by AnalyzePiccolo are dependent on the mass estimate of the aircraft; thus, the accuracy of $C_{L\ max}$ will depend on the accuracy of the mass estimate. As a result it is better to analyze the tests post flight, after the mass estimate has been corrected. Recall from Section 7.8 that the mass correction uses the measured GTOW and Final Weight to determine the ESFC of the flight and adjusts the Fuel mass estimate throughout the flight accordingly.

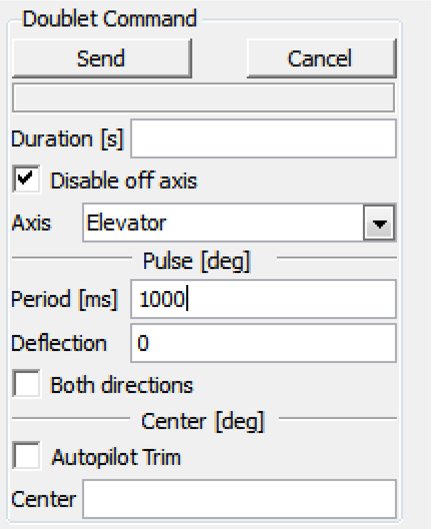
Have the manual pilot fly the aircraft to as near stall as desired. Use the $C_{L\ max}$ sheet to record the initial and end times of the test. Determine $C_{L\ max}$ visually from the DevInterface during the test or use AnalyzePiccolo. Remember that AnalyzePiccolo uses the plotpiccolo mat file which is generated from the script "plotpiccolo."

The AnalyzePiccolo C_L tool will display a $C_{L\ max}$ value in the command window after it has been ran; however, beware using this value as it could be an anomaly and not representative of the actual C_L max due to vibrations or noise. Make sure to at least inspect the plot and pick the appropriate value.

Recall that the vehicle parameter "$C_{L\ max}$" determines the minimum airspeed during auto lands, and the vehicle parameter "$C_{L\ max\ nom}$" determines the minimum airspeed during flight. Set "$C_{L\ max}$" to the value determined. Set "$C_{L\ max\ nom}$" to be as close to $C_{L\ max}$ as desired. They can both be the same; however, PCC will not allow $C_{L\ max\ nom}$ to be greater than $C_{L\ max}$. Recall also that the autopilot takes 1.1 times the dynamic pressure at $C_{L\ max\ nom}$ to determine the minimum airspeed during flight, so there is already a built in cushion to make sure that the minimum airspeed is at least just above stall speed.

8.9.2. C_L at zero elevator

C_L at zero elevator (Vehicle Parameter) can be determined by manipulating the elevator doublet function to command only 0 elevator deflection. Note that this can be dangerous as, assuming the aircraft is statically stable in pitch, the aircraft will pitch nose down, and sometimes rather quickly.



Duration [s]

☑ Disable off axis

Axis Elevator

Pulse [deg]

Period [ms] 1000

Deflection 0

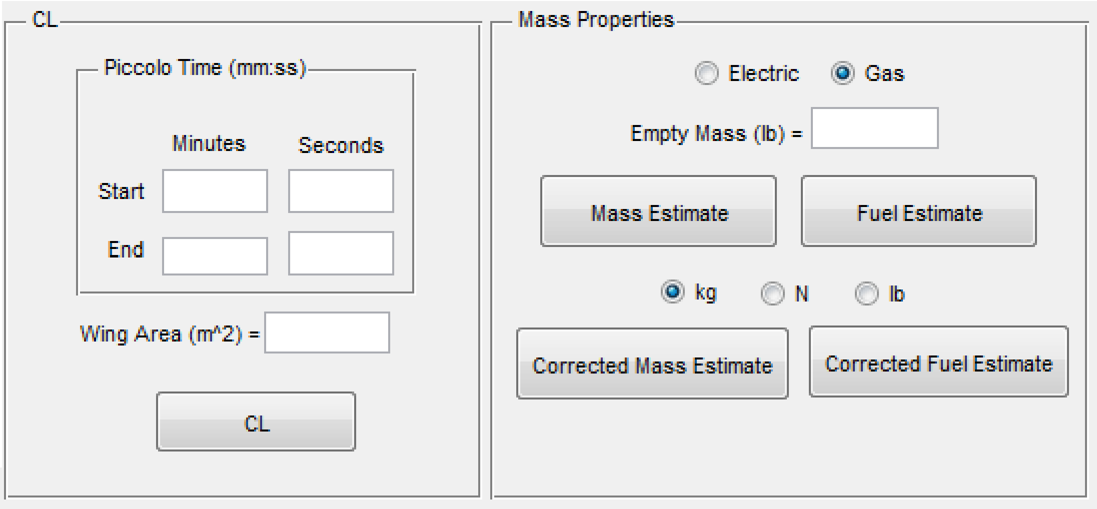
□ Both directions

Center [deg]

□ Autopilot Trim

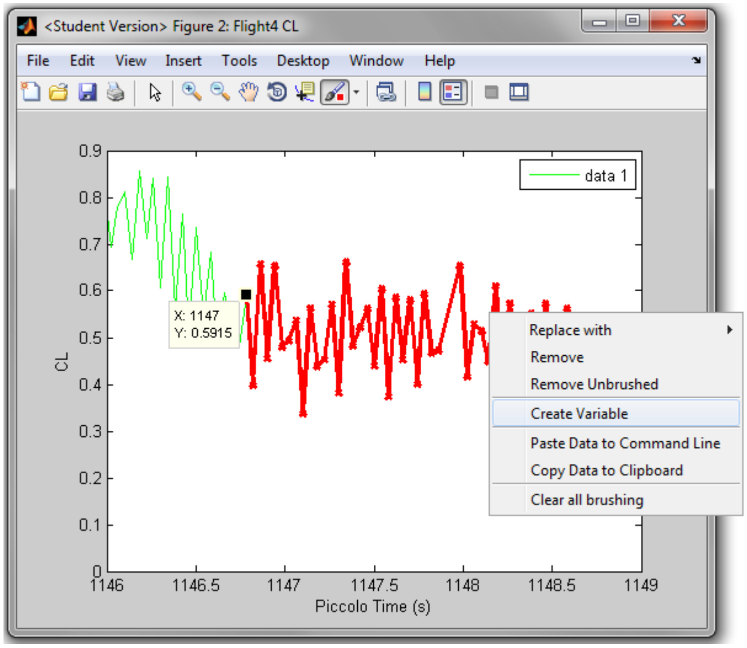
Center

As shown in the figure above enter 0 for elevator deflection and do not click either “Both directions” or “Autopilot Trim”. This way the elevator command is simply 0 degrees deflection. There is no need to input a value for “Duration” since the doublet file will not be used. It is up to the user to determine the period. The period needs to be long enough for the aircraft to settle, but short enough that the aircraft won’t nose dive into the ground. In NoctuaB1 Flight 4 I used a period of 3 seconds which was long enough to get a good estimate of C_L at zero elevator. The aircraft lost 50 feet of altitude during the time span of the test. Record the piccolo time just before starting and after the test is finished.

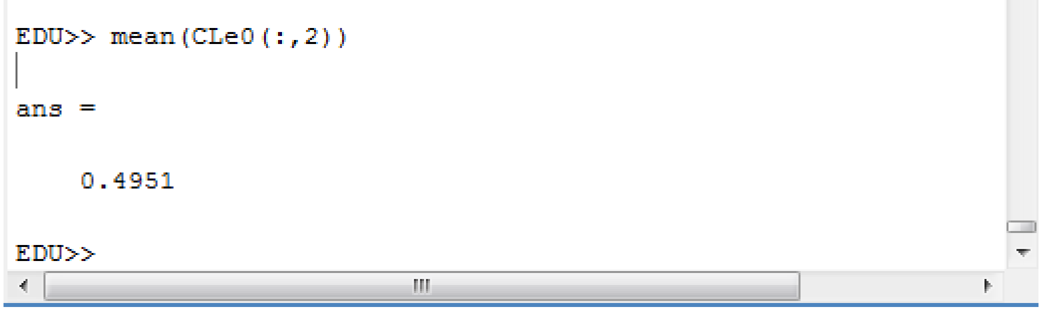


Use the C_L section in “AnalyzePiccolo” to evaluate the C_L values that occurred during the test. To calculate C_L the program will need the start time, end time, wing area, and empty takeoff mass. The C_L values calculated by AnalyzePiccolo are dependent on the mass estimate of the aircraft; thus, the accuracy of the C_L calculations will depend on the accuracy of the mass estimate. As a result it is better to analyze the tests post flight, after the mass estimate has been corrected. Recall from Section 7.8 that the mass correction uses the measured GTOW and Final Weight to determine the ESFC of the flight and adjusts the Fuel mass estimate throughout the flight accordingly.

If there is some noise in the C_L measurements use the mean which can be estimated visually or the actual mean can be calculated by MATLAB. To use MATLAB to calculate the mean utilize the paint brush tool, which selects data from plots. Highlight the appropriate range of C_L values. Right click the selected data and click "Create Variable" as depicted below.



Name it as desired. Use the MATLAB command window to determine the mean by typing "mean(VariableName(:,2))". The "(:,2)" portion designates the entire second column for the calculation and is used because the variable created will have 2 columns, one will be the time (x values) and the second column will be the actual C_L values (y values). In the example shown below the variable name is "CLe0".



```
EDU>> mean(CLe0(:,2))

ans =

    0.4951

EDU>>
```

8.10. Gain Tuning

The following sections detail methods of testing specific control loops. The methods consist of flying specific flight patterns, and disabling outer loops where applicable. Ultimately it will be up to the user to determine which gains to change and whether or not to implement the feedforward gains and filters that are available.

Gain tuning for longitudinal control is designed for aircraft that are intended to operate in altitude control. Both of the altitude control and airspeed control tuning sections and GUIs are designed for tuning systems that are meant to operate in altitude control. The user should feel free to alter any methods and even make changes to the MATLAB GUIs as appropriate for different scenarios. If the user decides to make changes to the GUIs make sure to save the programs in the aircraft's folder and not to overwrite the programs that were created initially in the default new aircraft folder.

8.10.1. Analysis Methods

While tuning gains use the DevInterface to view the pertinent system information (TAS, VRate, etc.). Use the appropriate Gain Tuning sheet to keep track of the gains that have been changed and keep track of the visual observations of the systems response along with the start and end times. In some cases using the DevInterface in flight alone can be enough to effectively tune the gains that are needed. If tuning cannot be resolved using the DevInterface alone, use the gain tuning MATLAB GUIs to analyze the response in greater detail.

Note that using the MATLAB GUIs takes some time. In some cases it could be better to try a lot of different gains with the DevInterface and use the MATLAB GUIs for post flight analysis of the gain changes. The following directions are for using the GUIs during flight. The MATLAB GUIs are to be run on the UAS Control Room Desktop. Since the GUIs depend on the log files to import the appropriate data copy the piccolo telemetry log file and the DevInterface log file from the UAS Laptop to the UAS Desktop. Before launching the GUIs the last row of the log files may need to be erased. Since the log files are copied while they are being written to they can end up with an incomplete row and incomplete rows will ruin MATLAB's capability to import the text files. The DevInterface log file is formatted neatly and when opened in notepad the last row is simply the last row via scrolling down. Once this has been done the appropriate GUI can be launched and utilized.

8.10.2. Gain Tuning Steps

Important things to note for tuning gains:

1) Make sure that the manual pilot is aware of the possibilities of instability and oscillations that could result during the gain tuning.

2) Make sure that the manual pilot understands that he or she should take control of the aircraft at any time they believe it is necessary.

3) Make sure that the manual pilot understands that he or she should take control of the aircraft if told do so regardless of what they think.

4) Slowly adjust gains, don't make large changes as hard oscillations could result. It is important to note that if communications are lost during gain tuning the autopilot will be stuck with the gains it had when comms were lost.

5) When disabling command loops:

    a) Make sure to fully understand the mechanics of how to re-enable the command loops so it can be done quickly if needed.

    b) If communications are lost with the piccolo all of the command loops will be automatically switched to "Auto" and the autopilot will target the lost comms waypoint.

c) Command loop status can be altered while the manual pilot is in control.

6) Make sure that the aircraft doesn't venture outside of the manual pilot's communication range. If it does and piccolo comms are lost the autopilot will be operating with the gain values it had at the time of lost comms which could potentially be dangerous.

7) Make sure that "Controller Telemetry" is enabled at the highest sampling rate as the data logged by the DevInterface is the core source of data for gain tuning.

Use the following steps to determine the course of action to take given control stability issues in flight:

Pitch

1) Check Z – Accelerometer data for vibration issues
   a) Fly the aircraft in either auto or manual control at different throttle settings
   b) Use "AltitudeControl.m" to analyze the data. Look for resonating vibrations at certain RPM ranges.
   c) If there are resonant vibrations either the piccolo or the motor will need to be remounted
   d) If the z – accelerometer measurements contain too much noise try utilizing the "Accel lpf cutoff".

   If vibrations are determined to not be the cause continue to 2.
2) Perform Elevator Doublets
3) Test CL at zero elevator
4) Go to Section 8.10.3 to tune the Altitude Control Gains

Lateral Control

1) Perform Aileron Doublets
2) Perform Rudder Doublets
3) Go to Section 8.10.5 to tune Lateral Control Gains.

8.10.3. Altitude Control

Ultimately tuning Altitude Control comes down to eliminating Pitch oscillations, or increasing Elevator response to disturbances. Recall that there are three loops that contribute to Elevator commands in Altitude Control; the altitude control outer loop, the VRate control loop, and the Z – Acceleration control loop. The Command Loops window provides the user with the ability to disable the altitude control outer loop and issue constant commands for VRate; therefore the process will utilize disabling the altitude control outer loop to attempt to isolate the gains that are causing the elevator issues.

Use any of the following tools to help analyze the system response:

1) DevInterface

2) Altitude Control Gain Tuning sheet

3) Altitude Control Gain Definitions

4) Altitude Control GUI

Make sure to enable “Controller Telemetry” at the max sample rate while gain tuning. Throughout gain tuning it is recommended to fly a simple flight pattern with the waypoints all at the same altitude; however, it is completely up to the discretion user.

1) Disable the altitude command loop by commanding a VRate of 0 in the command loops window.

2) If the autopilot is able to maintain the VRate command jump to step 4 otherwise begin the trial and error tuning of the Z – Acceleration control gains.

3) If the autopilot cannot maintain the VRate command by tuning the Z – Acceleration control gains then alter Kpv and go back to Step 2, otherwise proceed to step 4.

4) Command a moderate climb or descent by issuing a VRate command (1 is usually enough). Allow the autopilot time to respond, then change the VRate command back to 0. Analyze the response. If the autopilot was able to maintain the VRate commands proceed to step 5 otherwise perform steps 2 & 3 while climbing moderately. Make sure to not let the aircraft climb too high.

5) Re – enable the Altitude command loop. If the autopilot can maintain Altitude and is relatively stable in pitch then you are finished tuning. Otherwise proceed to step 6.

6) Alter Kpa until the autopilot is able to maintain Altitude and is relatively stable in pitch.

8.10.4. Energy Control

Tuning Energy Control is subjective as to the throttle response that is desired by the user. The focus of this guide is to ensure that the throttle response is quick enough that the aircraft will not descend below the minimum airspeed. Since the minimum airspeed is set essentially as the stall speed and the controller will not transition into slow airspeed mode until the throttle has reached 90% it is very important that the throttle will respond quickly enough to minimize the time that the autopilot could spend operating below the minimum airspeed. There are other applications for tuning Energy Control such as reducing throttle oscillations. For any application other than reducing drops in airspeed use the descriptions of Energy Control, and the gain tuning tools to tune the gains as desired.

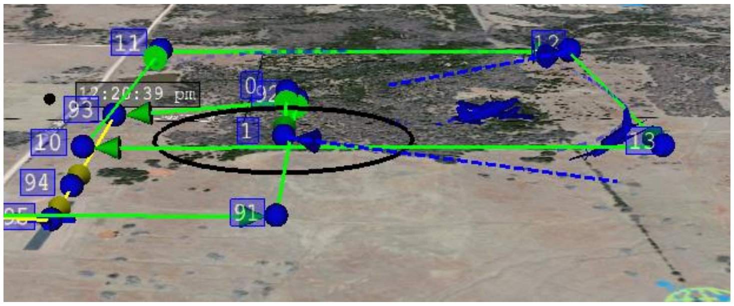
Tuning Energy Control is somewhat different than the other loops in a couple of different ways. First there are no user functions that allow the user to disable the outer loop and command a constant Energy Rate in order to tune the inner loop first. Secondly neither PCC, nor the DevInterface, allow the user to view what the Energy Rate, and Energy Rate commands actually are. Thirdly the inner loop is affected by other control loops, Altitude Control, via VRate feedback (Energy Rate feedback) and VRate commands (affect Energy Rate Commands). As a result all that can really be done is fly specific patterns to observe the airspeed undershoot, and then use trial and error to tune the gains.

As described in Section 5.8.4.1 Energy Control contains three feedback gains. The outer loop gain “Energy err to energy rate” is like a proportional gain and is referred to as “Kpo”. The inner loop gain “Energy rate err to throttle” is also like a proportional gain and is referred to as “Kpi”. The inner loop gain “Energy rate err int to throttle” is an integral gain and is referred to as “KI”. The outer loop commands Energy Rate by multiplying Kpo by the Energy error. Remember that in altitude control the energy equations are based off of TAS via kinetic energy. The inner loop uses Kpi by multiplying it by the Energy Rate error. KI integrates the Energy Rate error to the degree of its value.

The default gains are usually sufficient to be able to maintain airspeed in steady level flight. Usually airspeed will decrease at the beginning of a climb, after a descent, or after a turn. The biggest drops occur when the autopilot targets a new altitude as a step input, which occurs when the autopilot targets waypoints of different flight plans or targets a waypoint that is not the next waypoint in the current flight plan. As a result the best way to test the throttle response to large airspeed errors is to put the autopilot through likely worst case scenarios that the aircraft will see in flight such as where altitude is commanded as a step input. These tests will be subject to the altitude control limits that will dictate how hard the aircraft initially tries to climb.

Make sure that the aircraft does not switch out of Lon Mode 0 during the tests. If it does then the results of that test are void as throttle would no longer be controlling airspeed.

Setup a box pattern at the altitude that the aircraft is expected to normally fly at, or at an altitude higher than the lost communications waypoint. The idea is to descend the aircraft from the box pattern to the lost comms loiter waypoint and climb from the lost comms waypoint back to the box pattern to analyze the airspeed undershoot. The lost comms waypoint is used in order to make sure that the autopilot can successfully navigate to the lost comms waypoint in case communications are ever lost.



Fly as many iterations as necessary to reduce the airspeed undershoot so that it does not fall below IAS$_{min}$. Send the aircraft into the descents and climbs at the same locations each time so that the response with different gains can be compared. Do at least two iterations with the same gains for repeatability. Set the altitude of the box pattern so that the climb and descent slopes are similar to those that the aircraft are expected to navigate. Make sure that the climbs and descents aren't too steep as to cause the autopilot to switch into the Slow or Fast Airspeed modes.

Typically the default value for Kpo commands high enough Energy Rates to the inner loop and does not need to be changed. It is best to start by increasing KI. If increasing KI doesn't reduce the airspeed undershoot enough then set KI back to the default value, raise Kpo, and start the process over again. Raising Kpi usually draws out the recovery time of the airspeed undershoot. If the Energy Rate error just isn't large enough to trigger a quick Throttle response raise Kpo so that the Energy Rate commands will be higher given Energy errors and thus the Energy Rate error will be larger. Ultimately it is up to the user to decide whether or not to use the Throttle Prediction Trust and the low pass filter cutoff.

Use the DevInterface to view the live TAS response. Use the Primary Flight Display to watch the airspeeds with the Energy Control Gain Tuning sheets to keep track of the airspeed errors. If desired, use “Energy Control.m” to analyze the tests in greater detail. “Energy Control.m” interfaces with the Energy Control Gain Tuning sheets so that the user can analyze the different periods with different gain values.

Use the Energy Control Gain Definitions sheet for easy access to gain definitions and their effects on throttle. Note that raising Kpo will cause the outer loop to command higher Energy Rates in response to Energy (TAS) errors. Raising Kpi will cause the autopilot to command higher Throttle in response to Energy Rate errors. Raising KI will cause the autopilot to command higher Throttle in response to the integral of the Energy Rate error. Note that KI sums the integral of the Energy Rate errors throughout the entire time it is operating.

The following figures present examples of the impact increasing Kpi and KI can have on airspeed performance. The examples illustrate a scenario from a hardware in the loop simulation where an aircraft iteratively descended and climbed from a box pattern to the lost comms waypoint. The first drop in airspeed represents a descent and the second drop represents a climb.

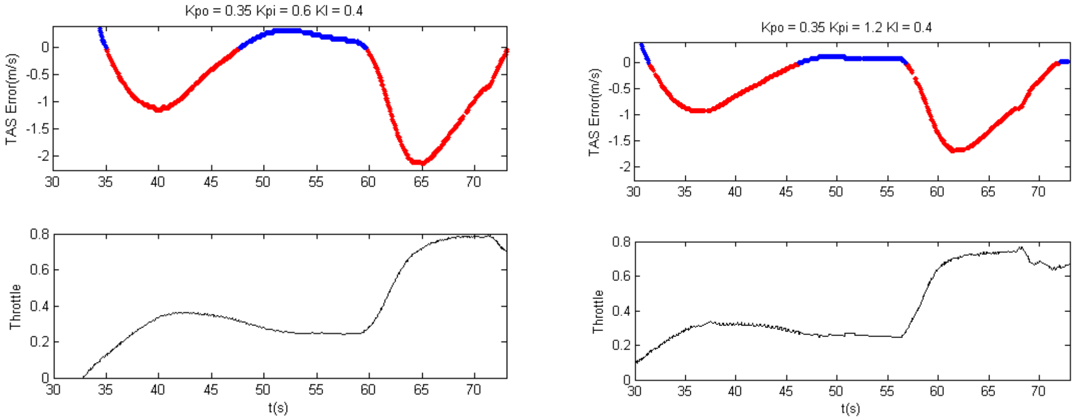


Figure 363 Energy Control Gain Example

Figure 363 Energy Control Gain Example above illustrates how raising Kpi can increase the period of the airspeed undershoot and slightly decrease the peak of the undershoot. Table 8 below provides the numerical results.

Table 8 Energy Control Gain Example 1

Kpi		Undershoot (m/s)	Period (s)
0.6	Climb	-2.12	13.28
1.2	Climb	-1.68	15.28
0.6	Descent	-1.14	12.86
1.2	Descent	-0.92	14.84

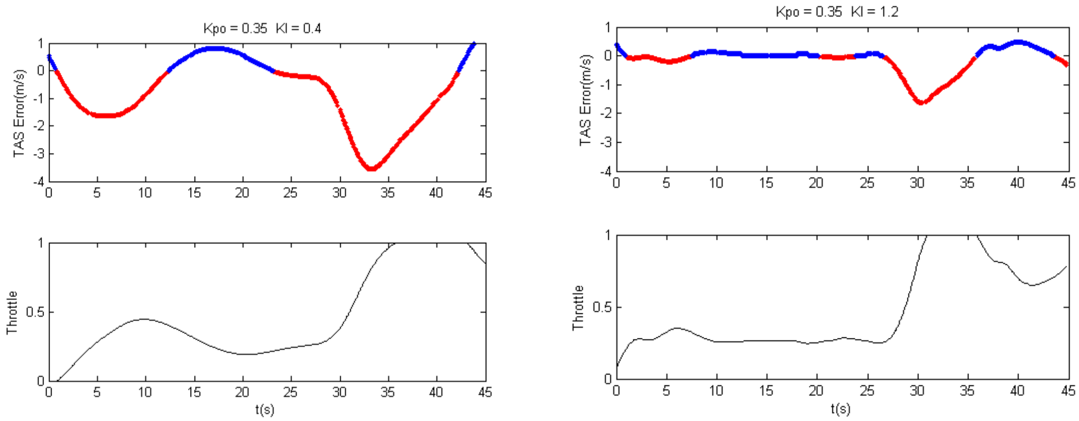


Figure 364 Energy Control Gain Example

Figure 364 illustrates how raising KI can decrease the peak of the undershoot and the time it takes for airspeed to climb out of negative error. Table 9 below provides the numerical results.

Table 9 Energy Control Gain Example 2

KI		Undershoot (m/s)	Period (s)
0.4	Climb	-3.53	14.32
1.2	Climb	-1.63	9.40
0.4	Descent	-1.63	11.64
1.2	Descent	-0.92	6.60

Note that in real flights there can be a lot of noise in the airspeed readings and viewing the TAS plots alone can be difficult to determine the impact that the gain tuning is having on the airspeed performance.

8.10.5. Lateral Tracking

1) If the aircraft is simply pre turning too soon and/or continuously targeting the next waypoint before the aircraft can track onto a flight path it is likely that the tracker convergence is too large. Incrementally decrease the tracker convergence until the autopilot is able to guide the aircraft onto a target flight path.
2) Create a box pattern where at least one leg of the flight plan is long enough that the autopilot will have time to be able to attempt to converge on it.
3) While flying the box pattern disable the heading command loop by issuing a bank angle command. Make sure to command a modest bank angle, don't command maximum bank. Recall that commanding the autopilot to target a waypoint will re enable the heading command loop and set it to auto.
4) There should be some overshoot and possibly a couple of small oscillations in roll before the autopilot settles on a roll angle. If the autopilot is unable to maintain the bank angle command then the roll control loop gains need to be tuned. Alter the roll control gains until

the autopilot is able to maintain bank angle commands. Use the roll control gain definitions sheet to assist tuning.

5) If the autopilot is able to maintain the roll command and track control oscillations still occur then the track control gains need to be tuned. Track control can be difficult to tune because the tracker convergence parameter is more of a setting than a gain and it plays a huge role in influencing track control. Decreasing the tracker convergence can make up for any inner loop lag that may exist because it will command tighter elliptical trajectories. Anytime that the tracker convergence is adjusted it is likely that the track control proportional and derivative gains will need to be adjusted as well.

   Allow the aircraft to complete one whole lap each time gains are changed. Use the Lateral Track Control gain tuning sheet to keep track of the gains used. Use the track data strip chart in PCC to tune track control on the fly. Use the LateralTracking MATLAB graphical user interface, coupled with the Laterl Track Control gain tuning sheet to take the analysis a step further.

   While tuning track control gains keep an eye on yaw control. It is possible that yaw control could be creating issues. If there is a lot of yaw rate error experiment using the side force control loop, or even disabling turn coordination to observe the effects.

REFERENCES

Anderson, John D, Jr. *aircraft performance and design*. WCB/McGraw-Hill, 1999.

Cloud Cap Technology. *Flight Summary Log Sheet*. Hood River, 11 October 2012. PDF.

—. *PccUsersGuide*. Hood River, 3 January 2013. PDF.

—. *Piccolo Doublet Analysis Tool*. Hood River, 2 February 2009. PDF.

—. *Piccolo Simulator*. Hood River, 17 October 2012. PDF.

—. *Piccolo Vehicle Integration Guide*. Hood River, 17 October 2012. PDF.

—. *Steps to Autonomous Flight*. Hood River, 17 October 2012. PDF.

—. *Tuning piccolo control laws 2.0.x*. Hood River, 20 June 2007. PDF.

Jager, Rylan. "https://kuscholarworks.ku.edu/dspace/bitstream/1808/3997/1/umi-ku-2513_1.pdf." 25 April 2008. *University of Kansas Scholar Works.* PDF. 2014.

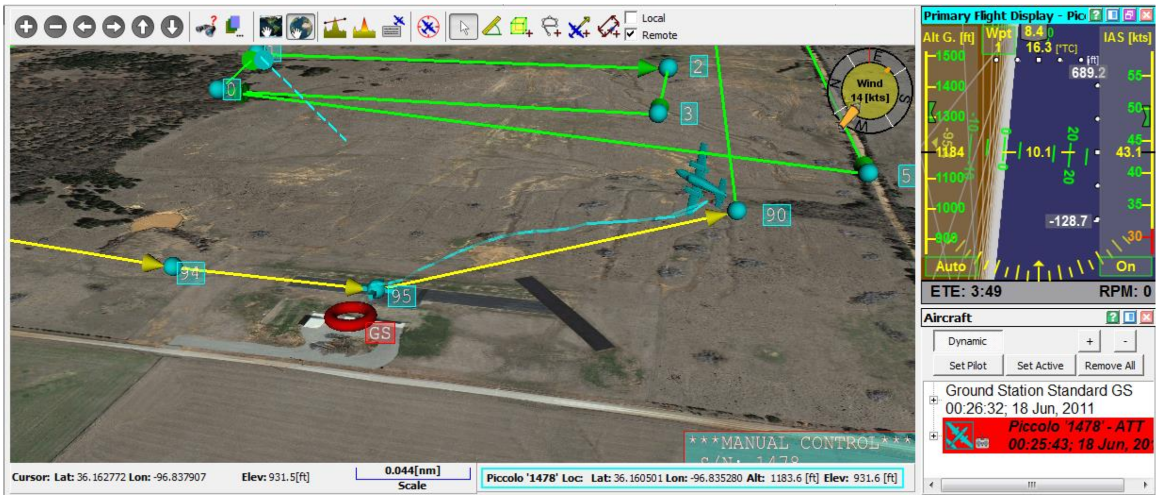
Nelson, Robert C. *Flight Stability and Automatic Control*. McGraw-Hill Higher Education, 1998.

APPENDIX A

HISTORY OF PICCOLO CRASHES

1. Diamondback

The Diamondback crashed in its second flight immediately after takeoff. At the time the piccolo was being flown through the ground station with the futaba connector. This was before the control room and control tower existed, so the antenna used was placed on top of the rotunda covering the concrete walkway next to the hanger. Note that the map shown below in Figure 365 is newer and does show the control room; however the control room did not exist at the time of the flight.



Cursor: Lat: 36.162772 Lon: -96.837907 Elev: 931.5[ft] 0.044[nm] Scale

Piccolo '1478' Loc: Lat: 36.160501 Lon: -96.835280 Alt: 1183.6 [ft] Elev: 931.6 [ft]

ETE: 3:49 RPM: 0

Ground Station Standard GS
00:26:32; 18 Jun, 2011

Piccolo '1478' - ATT
00:25:43; 18 Jun, 20

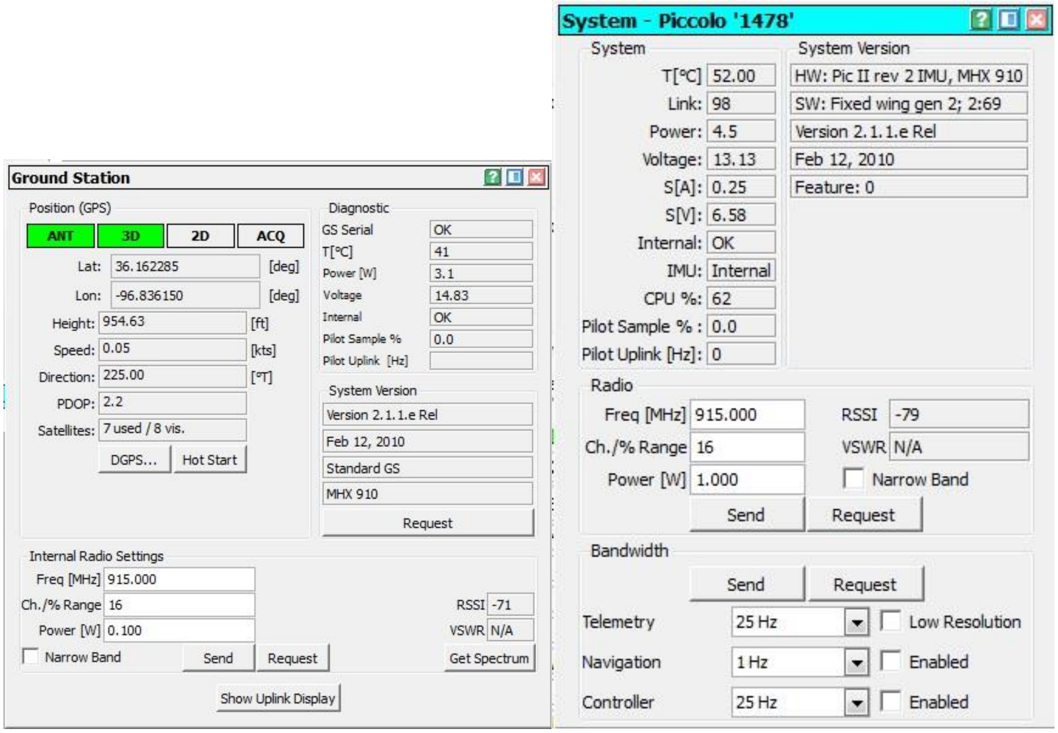
Figure 365 Diamondback Last Contact.

Figure 365 above shows the last contact point with the Diamondback. Just after takeoff a hard right bank was taken and at this point is where piccolo communications were lost. The antenna on the Diamondback was mounted on the top of the fuselage and was sticking straight out from the fuselage; thus, the antenna was pointed 90 degrees at the antenna attached to the ground station which was a null zone. The Diamondback held the 90 degree bank and flew straight into the ground destroying the plane and the Piccolo II 1478.

There were two things done that could have saved the flight.

1) Check the radio settings.

There was no real flight procedures setup at the time that the Diamondback flew, so there were not any real pre flight checks performed.



Ground Station

Position (GPS)

Lat: 36.162285 [deg]
Lon: -96.836150 [deg]
Height: 954.63 [ft]
Speed: 0.05 [kts]
Direction: 225.00 [°T]
PDOP: 2.2
Satellites: 7 used / 8 vis.

Diagnostic

GS Serial	OK
T[°C]	41
Power [W]	3.1
Voltage	14.83
Internal	OK
Pilot Sample %	0.0
Pilot Uplink [Hz]	

System Version

Version 2.1.1.e Rel
Feb 12, 2010
Standard GS
MHX 910

Internal Radio Settings

Freq [MHz] 915.000
Ch./% Range 16
Power [W] 0.100
□ Narrow Band
RSSI -71
VSWR N/A

System - Piccolo '1478'

System

T[°C]	52.00
Link:	98
Power:	4.5
Voltage:	13.13
S[A]:	0.25
S[V]:	6.58
Internal:	OK
IMU:	Internal
CPU %:	62
Pilot Sample % :	0.0
Pilot Uplink [Hz]:	0

System Version

HW: Pic II rev 2 IMU, MHX 910
SW: Fixed wing gen 2; 2:69
Version 2.1.1.e Rel
Feb 12, 2010
Feature: 0

Radio

Freq [MHz] 915.000
Ch./% Range 16
Power [W] 1.000
RSSI -79
VSWR N/A
□ Narrow Band

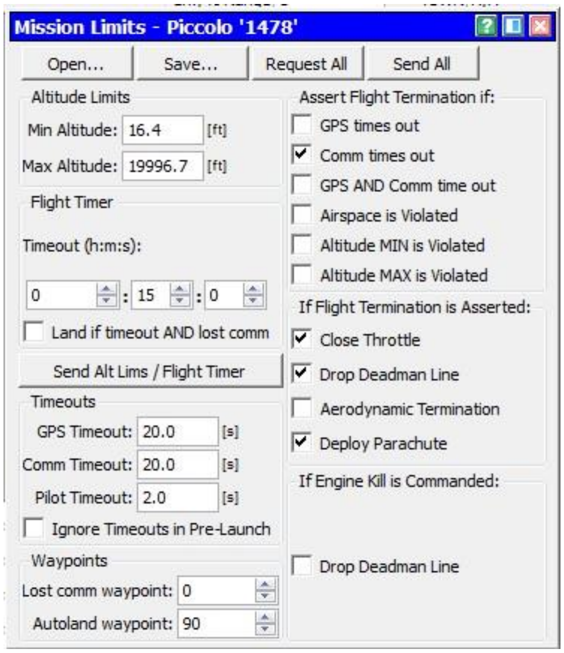
Bandwidth

Telemetry 25 Hz □ Low Resolution
Navigation 1 Hz □ Enabled
Controller 25 Hz □ Enabled

Figure 366 Diamondback Radio Settings

The radio settings of the ground station had only 0.1W instead of 1.0W power the radio. Even though the comms hit a null zone it is possible this could have helped and possibly allowed the ground station to reconnect to the autopilot before it nosed into the ground.

2) Set the pilot timeout to a lower value.



Altitude Limits
Min Altitude: 16.4 [ft]
Max Altitude: 19996.7 [ft]

Flight Timer
Timeout (h:m:s): 0 : 15 : 0
☐ Land if timeout AND lost comm

Timeouts
GPS Timeout: 20.0 [s]
Comm Timeout: 20.0 [s]
Pilot Timeout: 2.0 [s]
☐ Ignore Timeouts in Pre-Launch

Waypoints
Lost comm waypoint: 0
Autoland waypoint: 90

Assert Flight Termination if:
☐ GPS times out
☑ Comm times out
☐ GPS AND Comm time out
☐ Airspace is Violated
☐ Altitude MIN is Violated
☐ Altitude MAX is Violated

If Flight Termination is Asserted:
☑ Close Throttle
☑ Drop Deadman Line
☐ Aerodynamic Termination
☑ Deploy Parachute

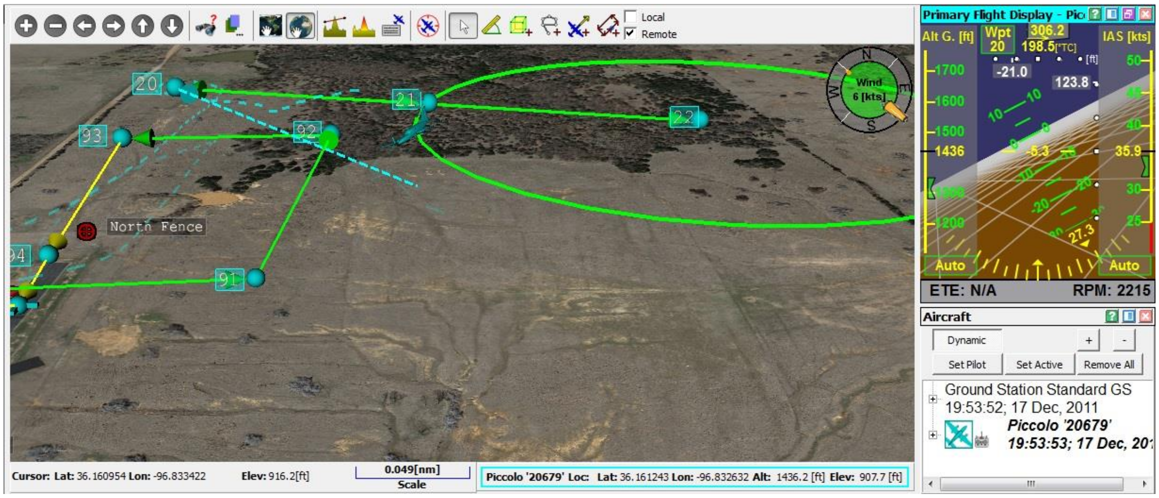
If Engine Kill is Commanded:
☐ Drop Deadman Line

Figure 367 Diamondback Mission Limits

The pilot timeout was set to 2 seconds. Within less than 2 seconds the Diamondback had already crashed. Had the pilot timeout been shorter the autopilot could have taken over and possibly saved the aircraft and the piccolo.

2. Cub

The Cub crashed after a servo had locked up and drained all of the available servo current to where none of the other servos could be controlled. The servos were being powered off of the piccolo servo line which is limited by the piccolo to 2 amps.



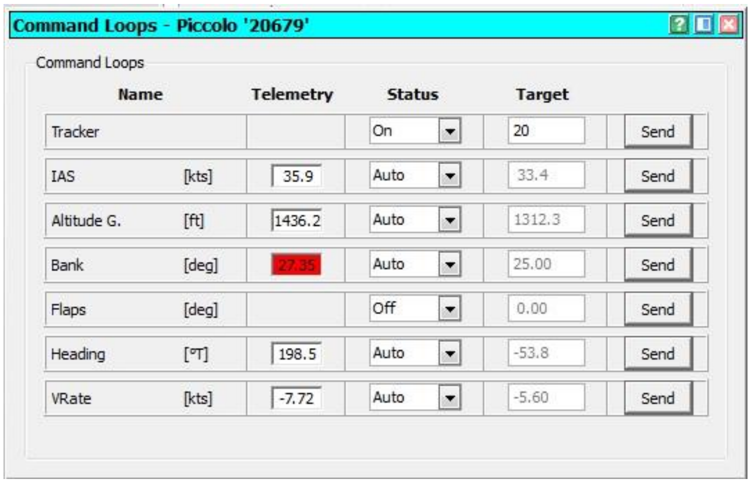
Cursor: Lat: 36.160954 Lon: -96.833422 Elev: 916.2[ft] 0.049[nm] Scale

Piccolo '20679' Loc: Lat: 36.161243 Lon: -96.832632 Alt: 1436.2 [ft] Elev: 907.7 [ft]

Ground Station Standard GS
19:53:52; 17 Dec, 2011
Piccolo '20679'
19:53:53; 17 Dec, 20

Figure 368 Cub Initial Turn

Figure 368 depicts the Cub just before things went bad. The autopilot commanded a 25 degree bank starboard, as shown in Figure 369, and was successfully en route to completing the commanded bank.

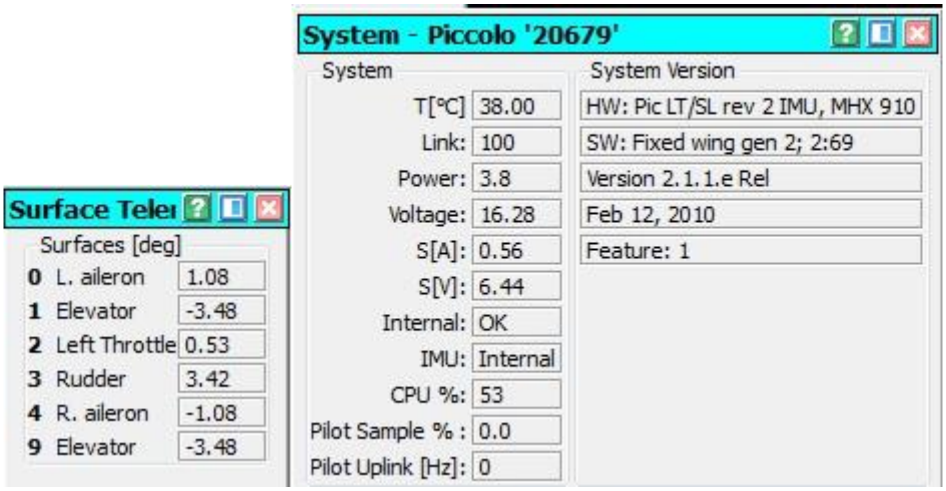


Command Loops - Piccolo '20679'

Command Loops

Name		Telemetry	Status	Target	
Tracker			On	20	Send
IAS	[kts]	35.9	Auto	33.4	Send
Altitude G.	[ft]	1436.2	Auto	1312.3	Send
Bank	[deg]	27.35	Auto	25.00	Send
Flaps	[deg]		Off	0.00	Send
Heading	[°T]	198.5	Auto	-53.8	Send
VRate	[kts]	-7.72	Auto	-5.60	Send

Figure 369 Cub Initial Turn Bank Command



Surface Tele
Surfaces [deg]

0	L. aileron	1.08
1	Elevator	-3.48
2	Left Throttle	0.53
3	Rudder	3.42
4	R. aileron	-1.08
9	Elevator	-3.48

System - Piccolo '20679'

System

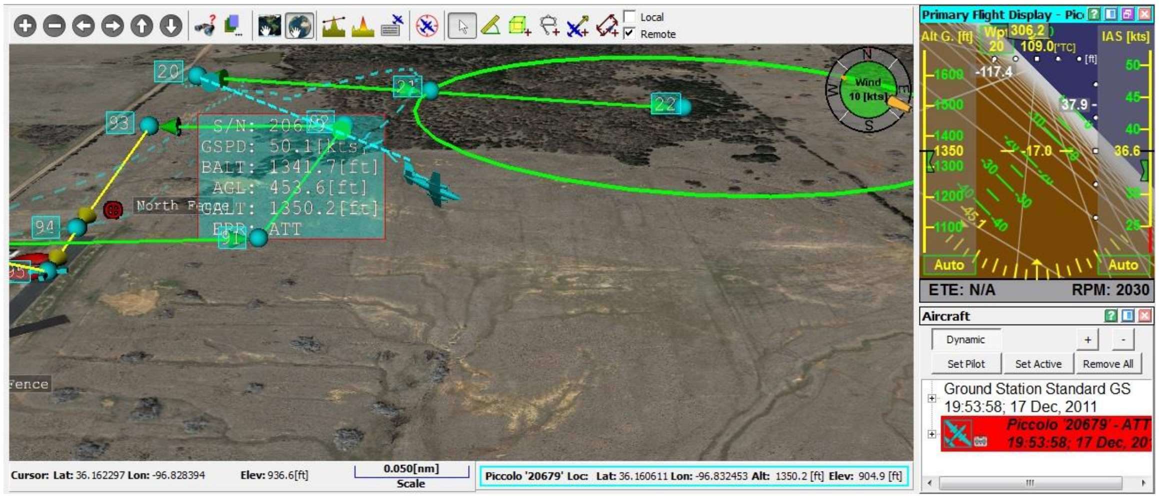
T[°C]:	38.00
Link:	100
Power:	3.8
Voltage:	16.28
S[A]:	0.56
S[V]:	6.44
Internal:	OK
IMU:	Internal
CPU %:	53
Pilot Sample % :	0.0
Pilot Uplink [Hz]:	0

System Version

HW: Pic LT/SL rev 2 IMU, MHX 910
SW: Fixed wing gen 2; 2:69
Version 2.1.1.e Rel
Feb 12, 2010
Feature: 1

Figure 370 Cub Initial Turn Control Surfaces

During the initial turn the ailerons were deflected a modest 1 degree in order to maintain the 27 – 25 degree bank angle. The servos were drawing about 0.56 amps which was the normal servo draw for the Cub. Immidiately following the initial turn the Cub suddenly banked port side and began to lose altitude.



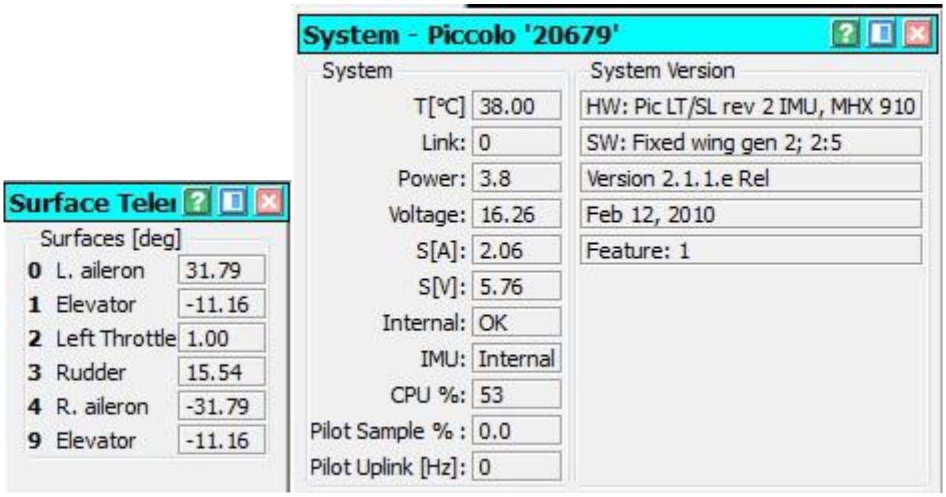
S/N: 20679
GSPD: 50.1[kts]
BALT: 1341.7[ft]
AGL: 453.6[ft]
BALT: 1350.2[ft]
ERR: ATT

Ground Station Standard GS
19:53:58; 17 Dec, 2011
Piccolo '20679' - ATT
19:53:58; 17 Dec, 20

Cursor: Lat: 36.162297 Lon: -96.828394 Elev: 936.6[ft] 0.050[nm] Scale Piccolo '20679' Loc: Lat: 36.160611 Lon: -96.832453 Alt: 1350.2 [ft] Elev: 904.9 [ft]

Figure 371 Cub Bad Turn

At the time this happened the servo draw spiked up to 2.06 amps, and the control surface deflections showed the ailerons being commanded full deflection to roll starboard.


Surface Tele

Surfaces [deg]

0	L. aileron	31.79
1	Elevator	-11.16
2	Left Throttle	1.00
3	Rudder	15.54
4	R. aileron	-31.79
9	Elevator	-11.16

System - Piccolo '20679'

System

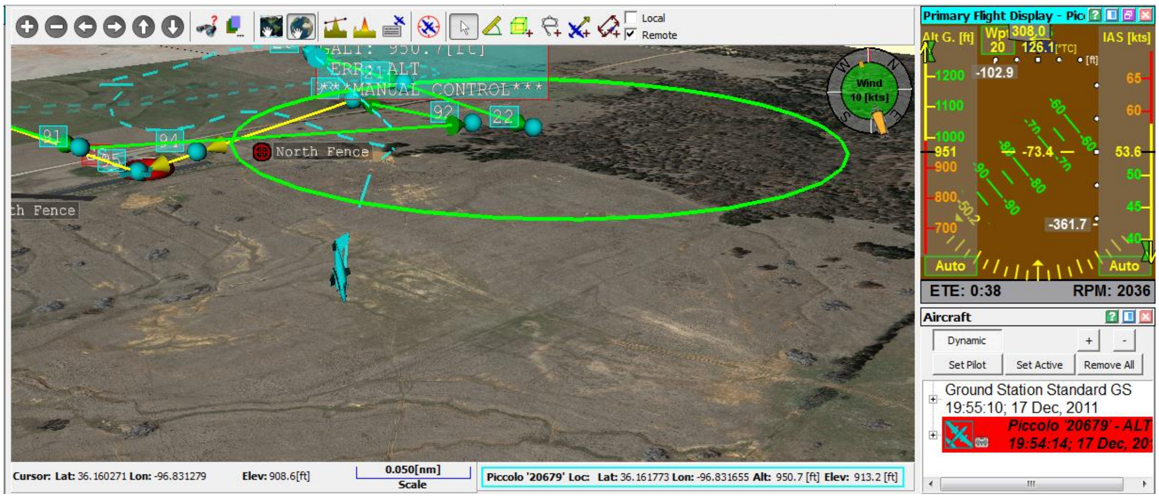
T[°C]	38.00
Link:	0
Power:	3.8
Voltage:	16.26
S[A]:	2.06
S[V]:	5.76
Internal:	OK
IMU:	Internal
CPU %:	53
Pilot Sample % :	0.0
Pilot Uplink [Hz]:	0

System Version

HW: Pic LT/SL rev 2 IMU, MHX 910
SW: Fixed wing gen 2; 2:5
Version 2.1.1.e Rel
Feb 12, 2010
Feature: 1

Figure 372 Cub Bad Turn Control Surfaces

At this point the autopilot had no control over the aircraft. Shortly after the bad turn occurred the manual pilot switched over control and was also unable to deflect any control surfaces. The Cub continued to spiral and nose dove straight into the ground at 65 MPH.



Cursor: Lat: 36.160271 Lon: -96.831279 Elev: 908.6[ft] 0.050[nm]

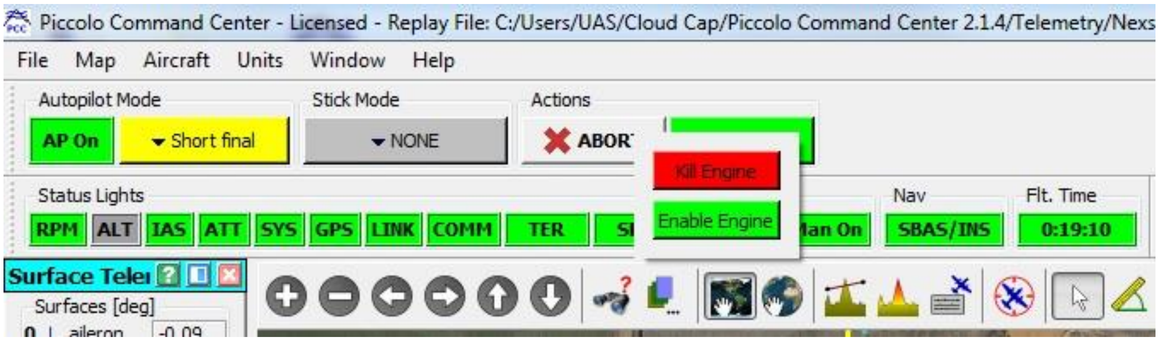
Piccolo '20679' Loc: Lat: 36.161773 Lon: -96.831655 Alt: 950.7 [ft] Elev: 913.2 [ft]

Ground Station Standard GS
19:55:10; 17 Dec, 2011
Piccolo '20679' - ALT
19:54:14; 17 Dec, 20

Figure 373 Cub Last Contact

The Piccolo SL 20679 miraculously survived the crash and was used in all 29 Nexstar flights.

3. Nexstar



The Nexstar had a minor crash during one of the attempted landings. The engine kill time in the land settings was not set to a negative number, it was set at 0. Nexstar was using a barometer for altitude measurements and when the aircraft crossed through the touchdown waypoint the autopilot thought it was at ground level; however, Nexstar was actually 6 feet above the ground. Since the autopilot thought it made it to the touchdown waypoint and the engine kill time was 0 the autopilot shut off the engine. When the manual pilot tried to take over he had no throttle control because “Engine Off” had occurred (even though it was an electric plane this button still kept throttle from being commanded). The plane crashed before the operater had time to turn the engine back to “Engine On”. The engine kill time was changed to -1 to avoid a repeat.

4. Noctua

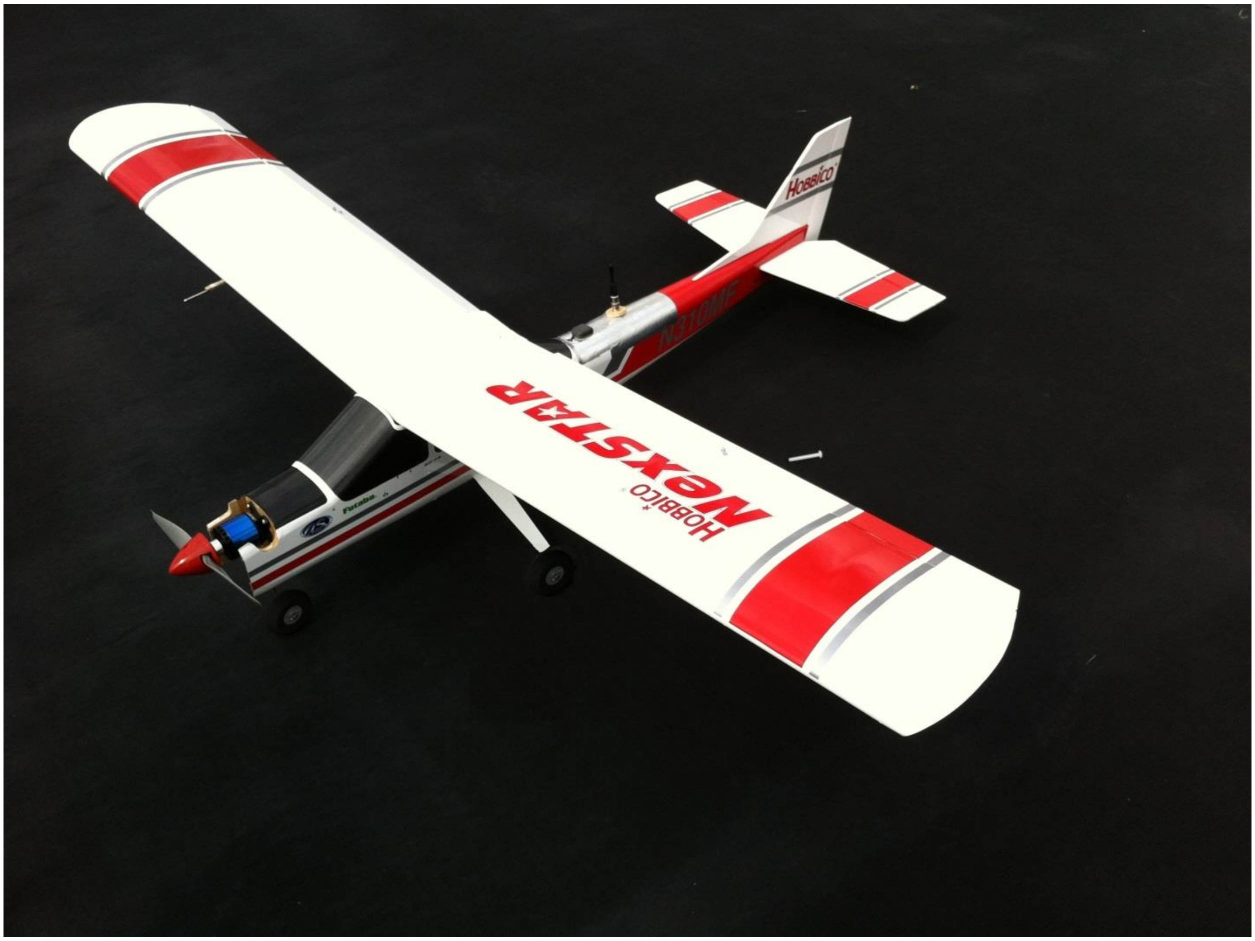
1) Noctua had a minor crash on the first attempted takeoff. The aircraft actually lifted off the ground before it reached the rotation speed. Even though the autopilot was reading a positive vertical rate it did not transition into climbout mode because the rotation airspeed had not yet been reached. Additionally the rudder was not mixed with the nose gear (tailwheel) in the mixing settings, so while Noctua was airborne the autopilot tried to steer the aircraft to keep track with only the tailwheel. As a result the maximum track error was reached before the rotation speed was reached so the autopilot auto aborted the takeoff while Noctua was in the air.

2) Noctua had a close call that could have caused a crash. Unbeknownst at the time when the laser altimeter is used in official land plans the laser will target the ground as soon as it is enabled. The laser is enabled in land plans based on the x distance away from the touchdown waypoint (set by user in land settings). It doesn’t matter if the touchdown waypoint is above the ground, say to fly mock approaches, the laser will target 0 AGL to the touchdown waypoint. Additionally if the laser takes over before it flys through the decision time waypoint it will target 0 AGL to the decision time waypoint. So wherever the aircraft is in the land plan when the laser takes over the altitude command will immediately target 0 AGL to the waypoint that is being tracked.

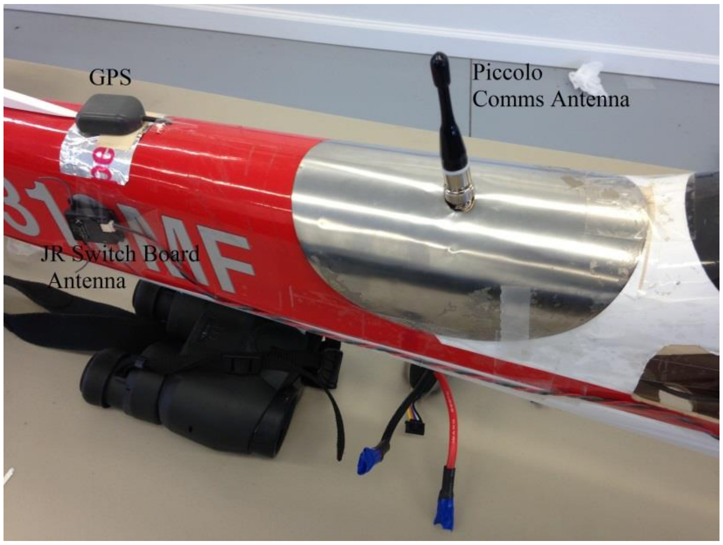
APPENDIX B

NEXSTAR

1. Nexstar



1.1. Hardware Configuration



1.1.1. Autopilot

Nexstar used the Piccolo SL unit 20679.

1.1.2. JR Level Shifter Board

Nexstar used the JR Level Shifter Board from Cloud Cap so the manual pilot could fly the aircraft through two antennas, independent of the piccolo communications. Of the two JR antennas one was mounted inside the center of the aircraft on the fuselage wall and the other was mounted outside the aircraft where it was secured to the side of the fuselage with Velcro and tape.

1.1.3. GPS Module

Nextar used the GPS module offered by Cloud Cap for the SL. Initially the GPS module was mounted on top of the fuselage and between the piccolo communications antenna and the back end of the cockpit; however, it was later moved downstream near the vertical tail. Ground plan tape was used for the GPS ground plane, and the GPS module was mounted directly on top of the ground plane tape.

1.1.4. Communications Antenna

Nexstar used the 900 Mhz ¼ wave antenna offered by Cloud Cap along with the corresponding ground plane. The antenna was mounted on top of the fuselage just behind the wings and upstream of the GPS module.

1.1.5. Pitot Tube

Nextar used a static and dynamic pitot tube from Hobby King. The pitot tube was mounted halfway across the starboard wing.

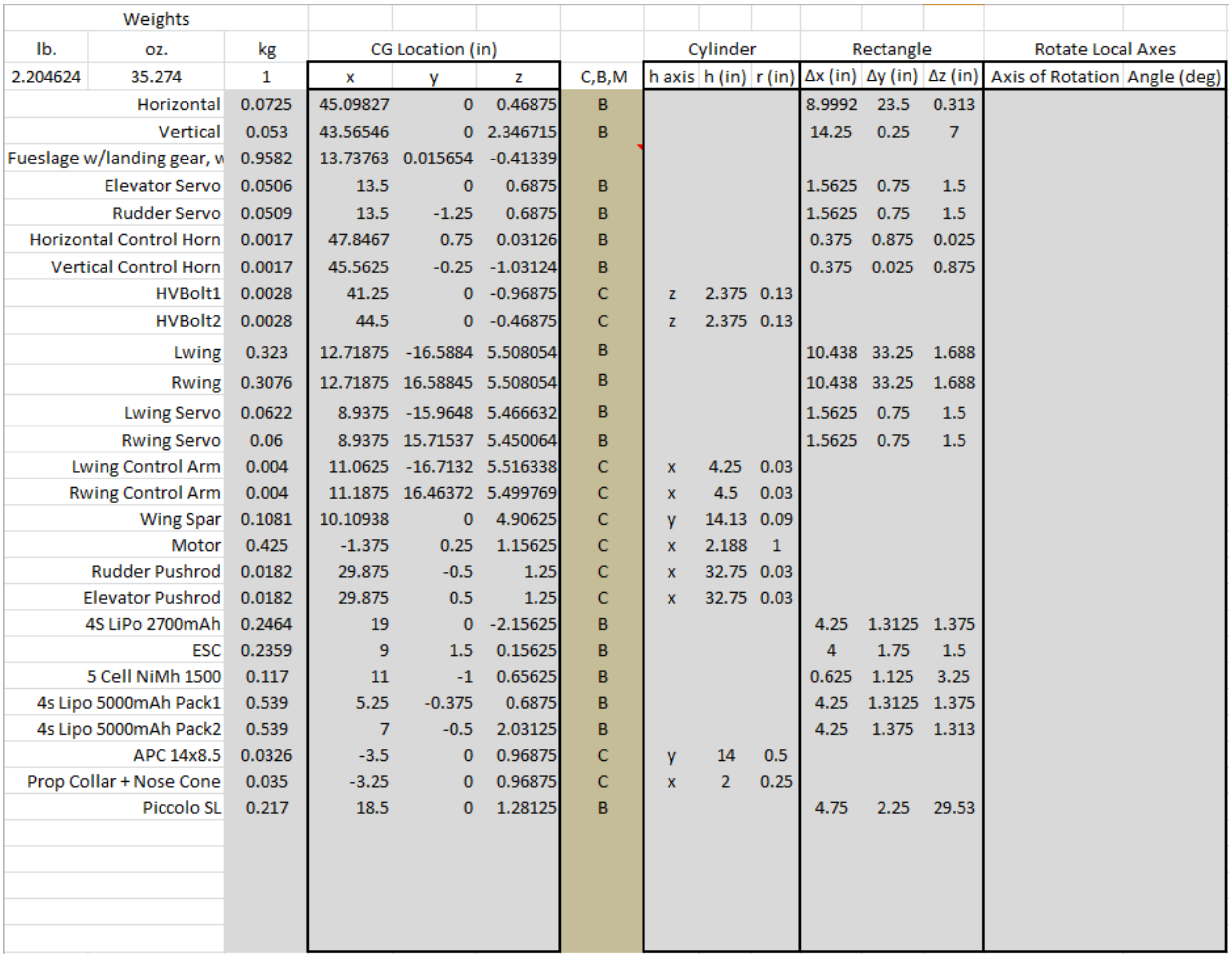
1.1.6. Motor

1.1.7. Propeller

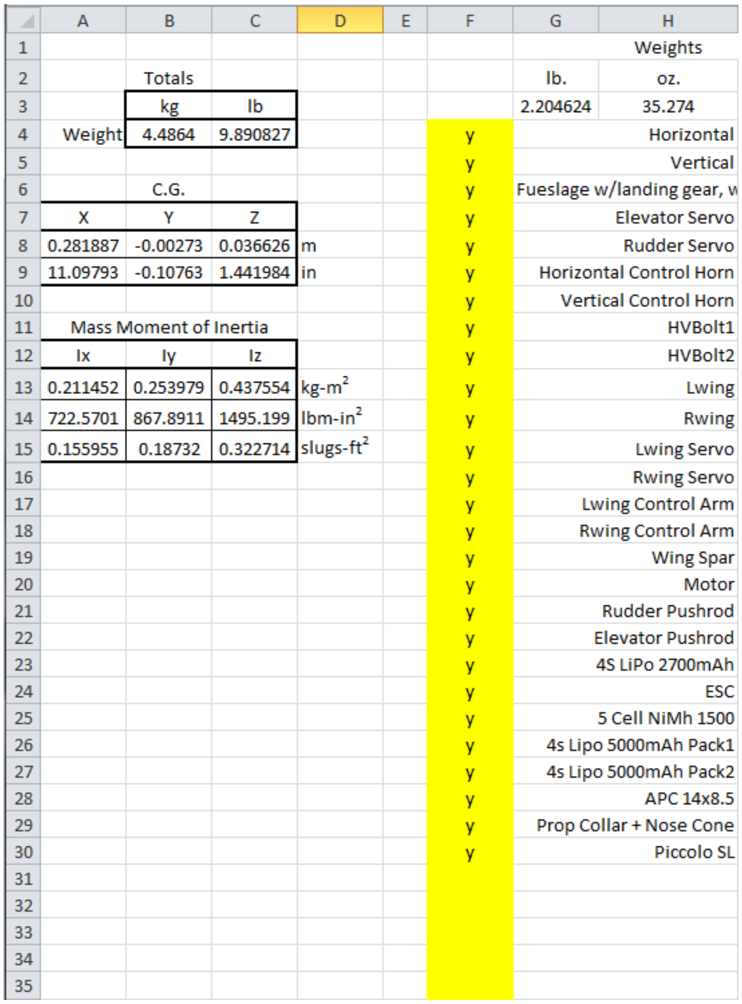
Nexstar used a 14x8.5 inch electric propeller. The propeller occasionally skimmed the runway and had to be replaced several times.

1.2. CG/Inertia Model

A component model was made using the “CGInertiaModel” spreadsheet from the New Aircraft folder. All of the individual components were weighed. Additionally the individual components’ dimensions were measured and the location of the estimated center of gravity of each component was measured with respect to the firewall just like the aircraft geometry was. Each component was assumed to have a uniform density distribution and classified as either a rectangle or cylinder to estimate the mass moment of inertia. The fuselage center of gravity was estimated by splitting the fuselage up into sections. Each section’s surface area was measured and calculated. The percentage of each section’s surface area over the total surface are of the fuselage was estimated to be the same percentage of weight of each section.



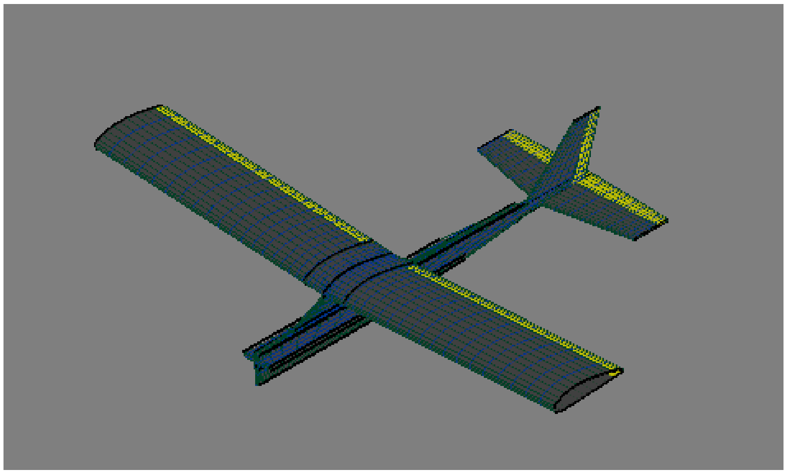
Weights														
lb.	oz.	kg	CG Location (in)				Cylinder			Rectangle			Rotate Local Axes	
2.204624	35.274	1	x	y	z	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)
Horizontal		0.0725	45.09827	0	0.46875	B				8.9992	23.5	0.313		
Vertical		0.053	43.56546	0	2.346715	B				14.25	0.25	7		
Fueslage w/landing gear, w		0.9582	13.73763	0.015654	-0.41339									
Elevator Servo		0.0506	13.5	0	0.6875	B				1.5625	0.75	1.5		
Rudder Servo		0.0509	13.5	-1.25	0.6875	B				1.5625	0.75	1.5		
Horizontal Control Horn		0.0017	47.8467	0.75	0.03126	B				0.375	0.875	0.025		
Vertical Control Horn		0.0017	45.5625	-0.25	-1.03124	B				0.375	0.025	0.875		
HVBolt1		0.0028	41.25	0	-0.96875	C	z	2.375	0.13					
HVBolt2		0.0028	44.5	0	-0.46875	C	z	2.375	0.13					
Lwing		0.323	12.71875	-16.5884	5.508054	B				10.438	33.25	1.688		
Rwing		0.3076	12.71875	16.58845	5.508054	B				10.438	33.25	1.688		
Lwing Servo		0.0622	8.9375	-15.9648	5.466632	B				1.5625	0.75	1.5		
Rwing Servo		0.06	8.9375	15.71537	5.450064	B				1.5625	0.75	1.5		
Lwing Control Arm		0.004	11.0625	-16.7132	5.516338	C	x	4.25	0.03					
Rwing Control Arm		0.004	11.1875	16.46372	5.499769	C	x	4.5	0.03					
Wing Spar		0.1081	10.10938	0	4.90625	C	y	14.13	0.09					
Motor		0.425	-1.375	0.25	1.15625	C	x	2.188	1					
Rudder Pushrod		0.0182	29.875	-0.5	1.25	C	x	32.75	0.03					
Elevator Pushrod		0.0182	29.875	0.5	1.25	C	x	32.75	0.03					
4S LiPo 2700mAh		0.2464	19	0	-2.15625	B				4.25	1.3125	1.375		
ESC		0.2359	9	1.5	0.15625	B				4	1.75	1.5		
5 Cell NiMh 1500		0.117	11	-1	0.65625	B				0.625	1.125	3.25		
4s Lipo 5000mAh Pack1		0.539	5.25	-0.375	0.6875	B				4.25	1.3125	1.375		
4s Lipo 5000mAh Pack2		0.539	7	-0.5	2.03125	B				4.25	1.375	1.313		
APC 14x8.5		0.0326	-3.5	0	0.96875	C	y	14	0.5					
Prop Collar + Nose Cone		0.035	-3.25	0	0.96875	C	x	2	0.25					
Piccolo SL		0.217	18.5	0	1.28125	B				4.75	2.25	29.53		



	A	B	C	D	E	F	G	H
1								Weights
2		Totals					lb.	oz.
3		kg	lb				2.204624	35.274
4	Weight	4.4864	9.890827			y		Horizontal
5						y		Vertical
6		C.G.				y		Fueslage w/landing gear, w
7	X	Y	Z			y		Elevator Servo
8	0.281887	-0.00273	0.036626	m		y		Rudder Servo
9	11.09793	-0.10763	1.441984	in		y		Horizontal Control Horn
10						y		Vertical Control Horn
11	Mass Moment of Inertia					y		HVBolt1
12	Ix	Iy	Iz			y		HVBolt2
13	0.211452	0.253979	0.437554	kg-m^2		y		Lwing
14	722.5701	867.8911	1495.199	lbm-in^2		y		Rwing
15	0.155955	0.18732	0.322714	slugs-ft^2		y		Lwing Servo
16						y		Rwing Servo
17						y		Lwing Control Arm
18						y		Rwing Control Arm
19						y		Wing Spar
20						y		Motor
21						y		Rudder Pushrod
22						y		Elevator Pushrod
23						y		4S LiPo 2700mAh
24						y		ESC
25						y		5 Cell NiMh 1500
26						y		4s Lipo 5000mAh Pack1
27						y		4s Lipo 5000mAh Pack2
28						y		APC 14x8.5
29						y		Prop Collar + Nose Cone
30						y		Piccolo SL
31								
32								
33								
34								
35								

The figures above depict the Nexstar components and the results of the component model.

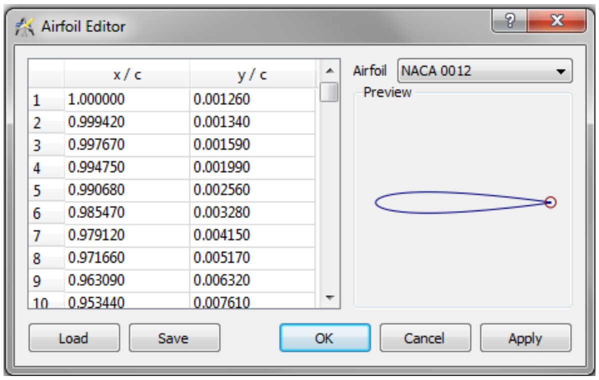
1.3. AVLEditor Model



1.3.1. Airfoils

The wings of the Nexstar had a Clark Y airfoil. At the time that the Nexstar was modeled it was not yet known how to import airfoil dat files properly, so the wing was modeled with an NACA 4417 airfoil instead as it essentially has the same shape as a Clark Y. The vertical and horizontal tails of the Nexstar were flat plates.

1.3.1.1. NACA 4417

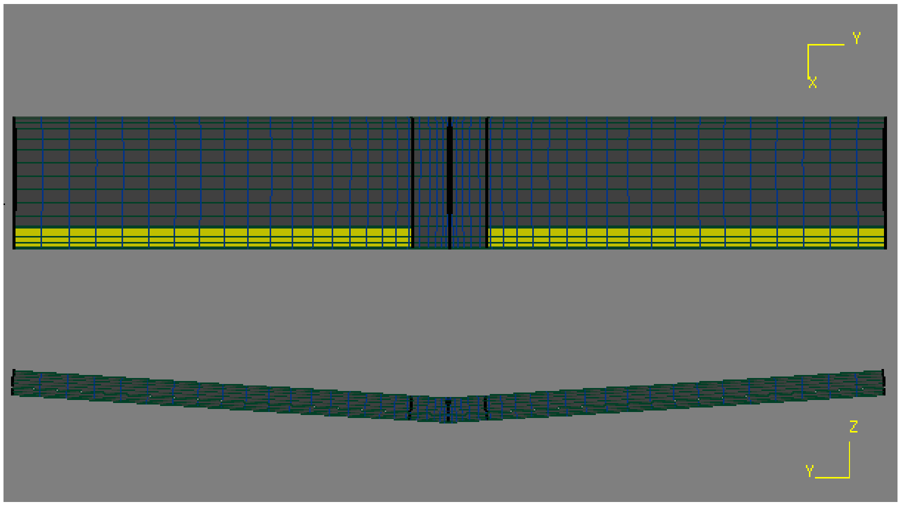


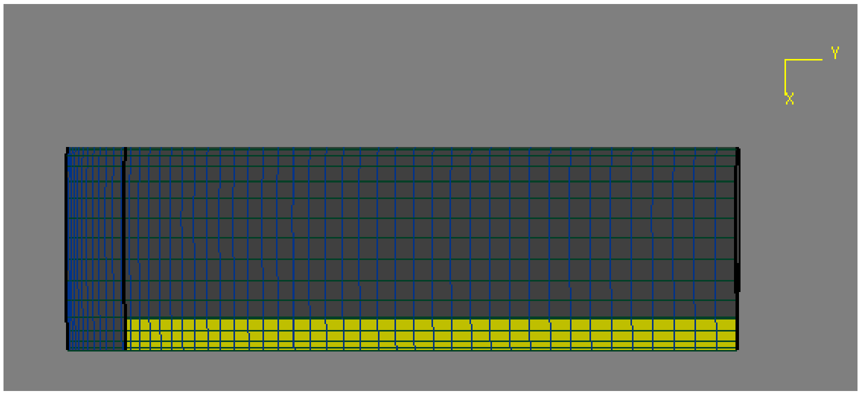
	x / c	y / c
1	1.000000	0.001260
2	0.999420	0.001340
3	0.997670	0.001590
4	0.994750	0.001990
5	0.990680	0.002560
6	0.985470	0.003280
7	0.979120	0.004150
8	0.971660	0.005170
9	0.963090	0.006320
10	0.953440	0.007610

Airfoil: NACA 0012

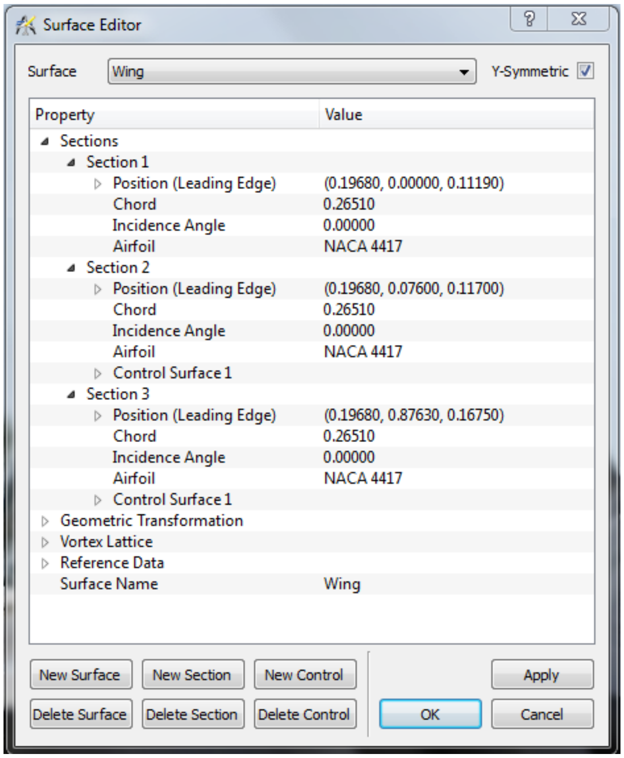
The NACA 4417 airfoil file was generated using AVLEditor's Airfoil Editor.

1.3.2.Wing





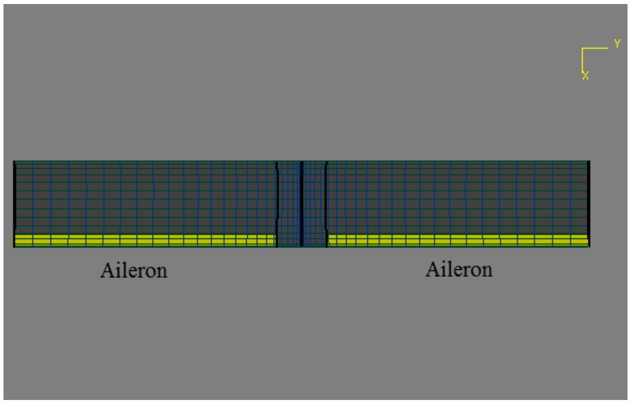
The wing was created with 3 sections, counting the centerline and wing tip. Y-Symmetry was used to mirror the wing on the port side. The wing had a dihedral of about 6 degrees.



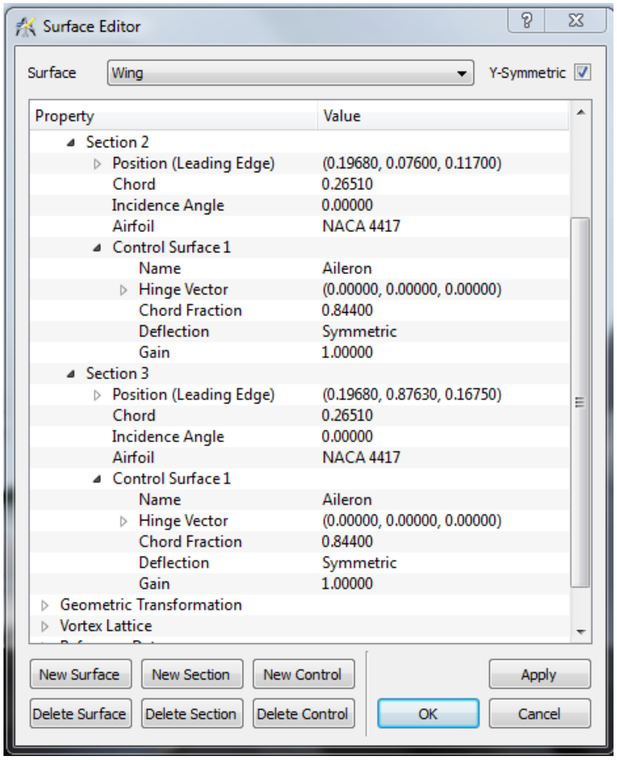
Property	Value
Sections	
Section 1	
Position (Leading Edge)	(0.19680, 0.00000, 0.11190)
Chord	0.26510
Incidence Angle	0.00000
Airfoil	NACA 4417
Section 2	
Position (Leading Edge)	(0.19680, 0.07600, 0.11700)
Chord	0.26510
Incidence Angle	0.00000
Airfoil	NACA 4417
Control Surface 1	
Section 3	
Position (Leading Edge)	(0.19680, 0.87630, 0.16750)
Chord	0.26510
Incidence Angle	0.00000
Airfoil	NACA 4417
Control Surface 1	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Wing

The figure above shows the sections and their positions. The surface was named "Wing". The airfoil was assigned as the NACA 4417 for all of the wing sections.

1.3.2.1. Control Surfaces



Nexstar had only one control surface on each wing, ailerons.

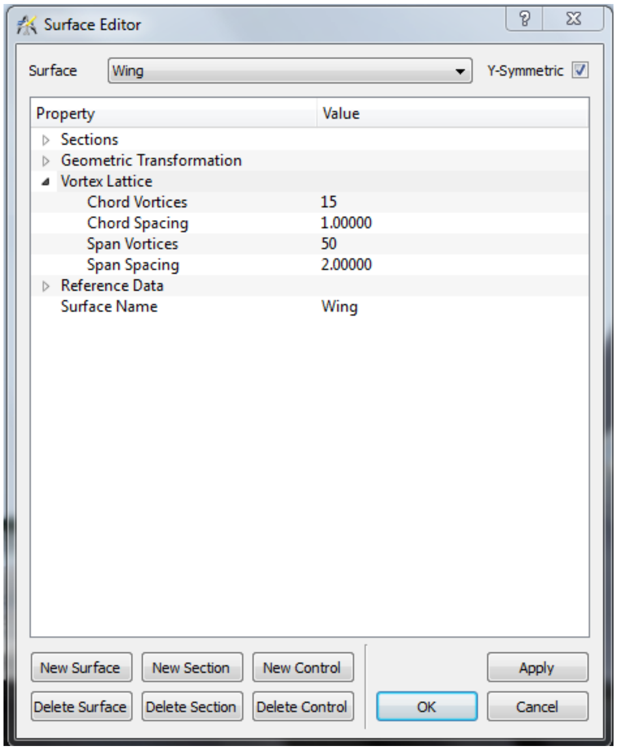


Surface: Wing Y-Symmetric ☑

Property	Value
Section 2	
Position (Leading Edge)	(0.19680, 0.07600, 0.11700)
Chord	0.26510
Incidence Angle	0.00000
Airfoil	NACA 4417
Control Surface 1	
Name	Aileron
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.84400
Deflection	Symmetric
Gain	1.00000
Section 3	
Position (Leading Edge)	(0.19680, 0.87630, 0.16750)
Chord	0.26510
Incidence Angle	0.00000
Airfoil	NACA 4417
Control Surface 1	
Name	Aileron
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.84400
Deflection	Symmetric
Gain	1.00000
Geometric Transformation	
Vortex Lattice	

The ailerons were added to sections 2 and 3 as show in the figure above. The control surface was named “Aileron”. Y-Symmetry automatically designated the two ailerons as separate control surfaces, “LAileron” and “RAileron”. “Deflection” was set at the default setting “Symmetric”. “Gain” was set at the default value of 1.

1.3.2.2. Vortex Settings



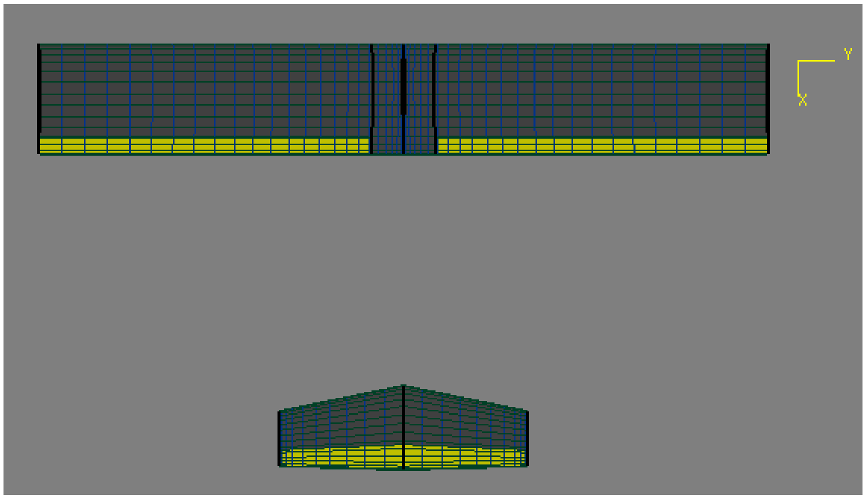
Surface Editor

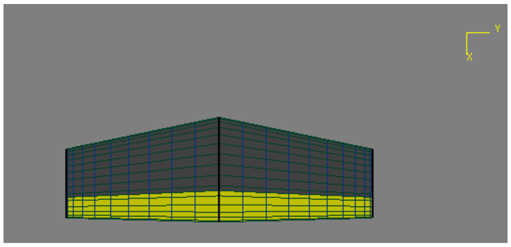
Surface: Wing   Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	15
Chord Spacing	1.00000
Span Vortices	50
Span Spacing	2.00000
▷ Reference Data	
Surface Name	Wing

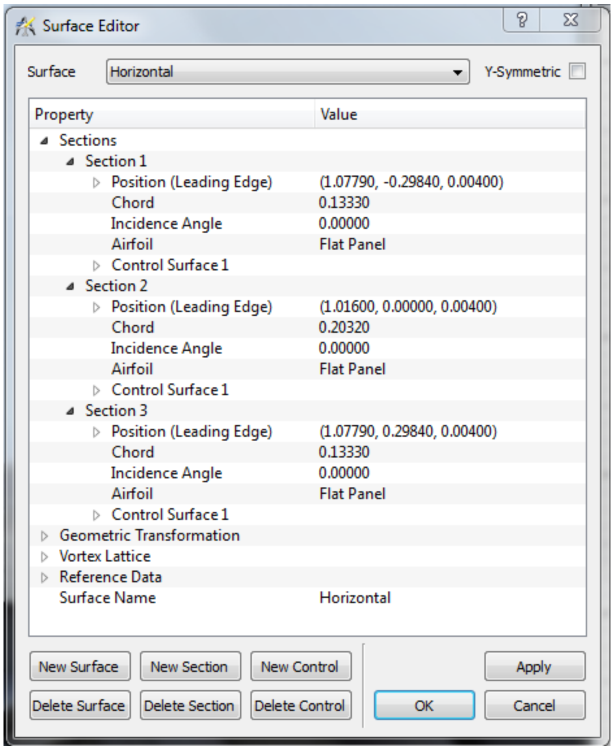
The vortex settings are shown in the figure above. The settings were not set according to the guidelines in Section 4.1.10 because those guidelines had not been established in a system for modeling. If Nexstar were modeled again the span spacing would be “1.0” instead of “2.0” so that there would be vortices bunched at the wing tips and dihedral break.

1.3.3. Horizontal Tail





The horizontal tail was made with 3 sections. Y-Symmetry was not used because the elevator control surface on the horizontal tail had one actuator; therefore, it was one single control surface rather than two separate control surfaces.

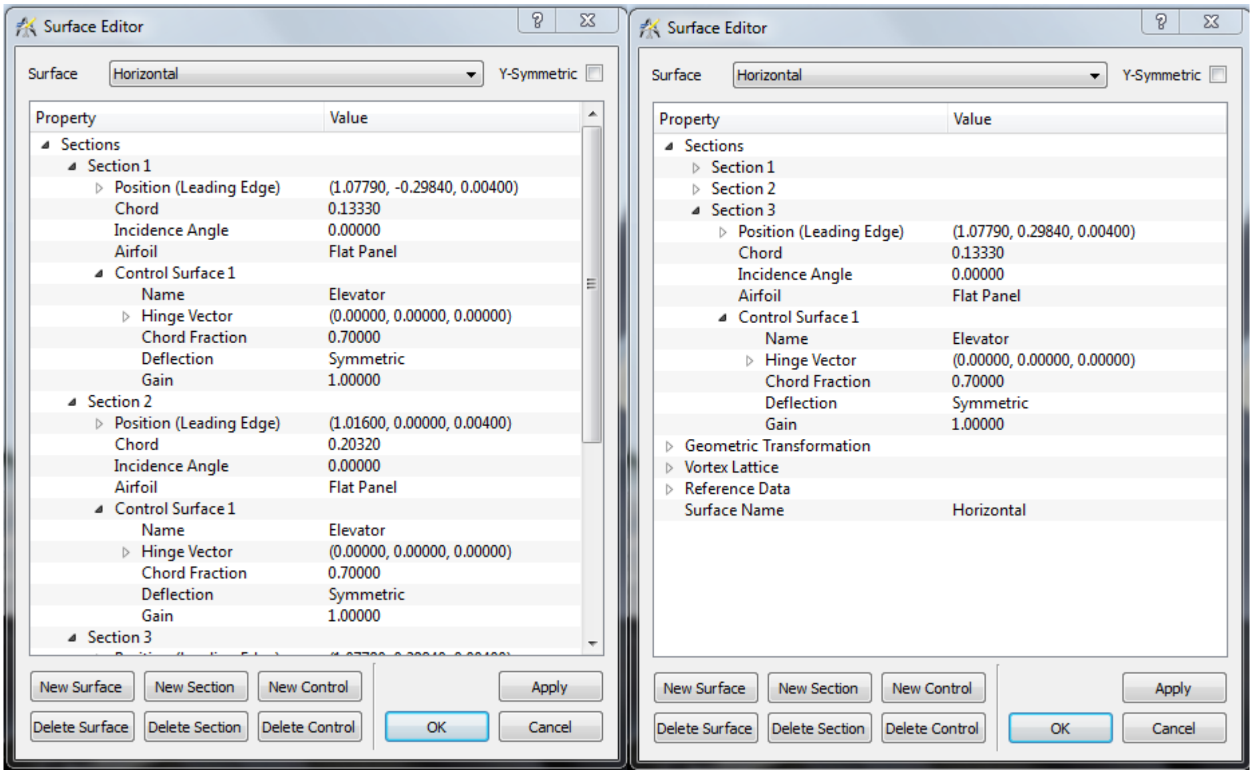


Surface: Horizontal ▾ Y-Symmetric □

Property	Value
◢ Sections	
◢ Section 1	
▷ Position (Leading Edge)	(1.07790, -0.29840, 0.00400)
Chord	0.13330
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Control Surface 1	
◢ Section 2	
▷ Position (Leading Edge)	(1.01600, 0.00000, 0.00400)
Chord	0.20320
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Control Surface 1	
◢ Section 3	
▷ Position (Leading Edge)	(1.07790, 0.29840, 0.00400)
Chord	0.13330
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Control Surface 1	
▷ Geometric Transformation	
▷ Vortex Lattice	
▷ Reference Data	
Surface Name	Horizontal

The figure above shows the sections and their positions. The surface was named "Horizontal". The horizontal was a flat plate, so no airfoil file was assigned to the surface sections.

1.3.3.1. Control Surfaces



Surface: Horizontal   Y-Symmetric ☐

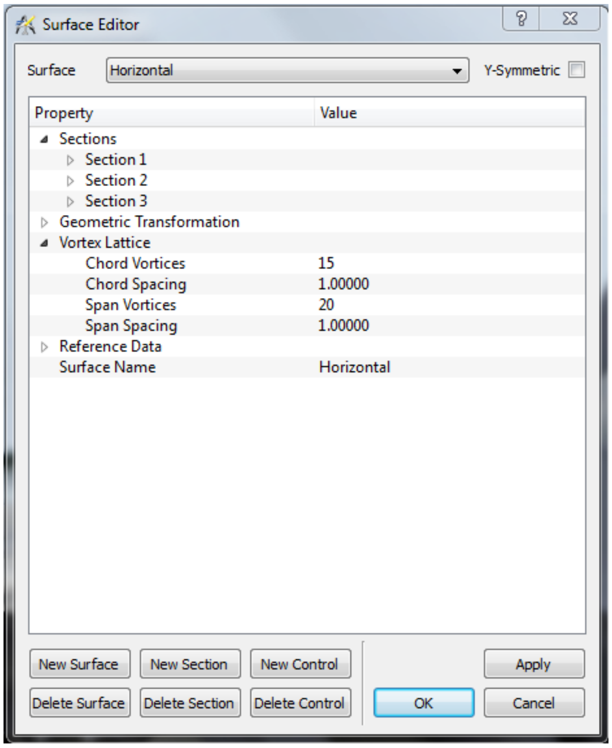
Property	Value
Sections	
Section 1	
Position (Leading Edge)	(1.07790, -0.29840, 0.00400)
Chord	0.13330
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Elevator
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.70000
Deflection	Symmetric
Gain	1.00000
Section 2	
Position (Leading Edge)	(1.01600, 0.00000, 0.00400)
Chord	0.20320
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Elevator
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.70000
Deflection	Symmetric
Gain	1.00000
Section 3	

Surface: Horizontal   Y-Symmetric ☐

Property	Value
Sections	
Section 1	
Section 2	
Section 3	
Position (Leading Edge)	(1.07790, 0.29840, 0.00400)
Chord	0.13330
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Elevator
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.70000
Deflection	Symmetric
Gain	1.00000
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Horizontal

The horizontal tail had only one control surface, the elevator. The elevator was controlled by one servo; thus it acted as one control surface and was assigned to all three sections. The elevator was named “Elevator”. The elevator tapered with the horizontal tail, so the chord fraction was constant for all three sections. The “Deflection” was set at the default setting “Symmetric”. “Gain” was set at the default value of 1.

1.3.3.2. Vortex Settings



Surface Editor

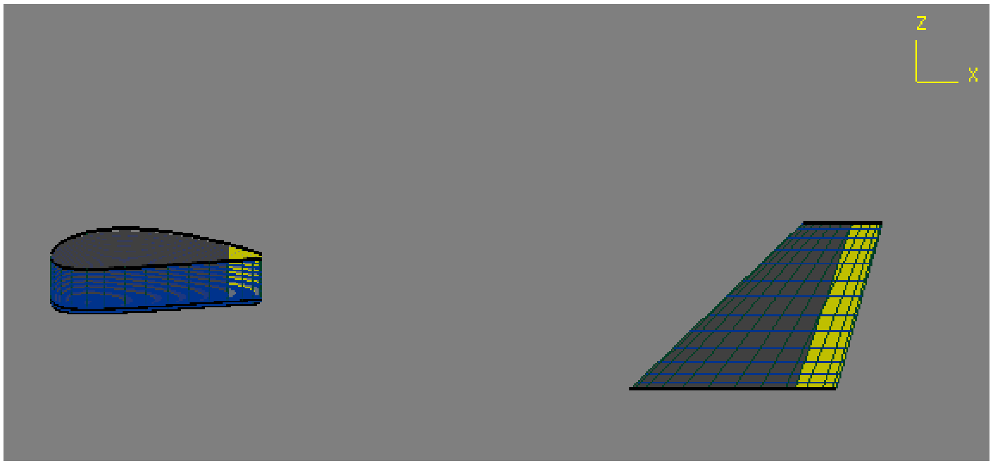
Surface: Horizontal

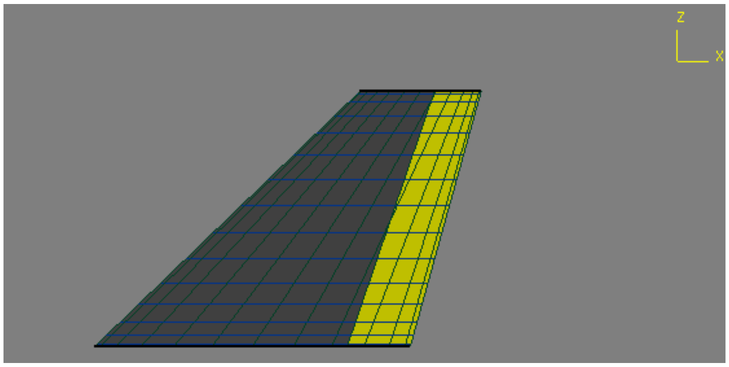
Y-Symmetric □

```
Property                      Value
◢ Sections
    ▷ Section 1
    ▷ Section 2
    ▷ Section 3
▷ Geometric Transformation
◢ Vortex Lattice
    Chord Vortices            15
    Chord Spacing             1.00000
    Span Vortices             20
    Span Spacing              1.00000
▷ Reference Data
  Surface Name                Horizontal
```

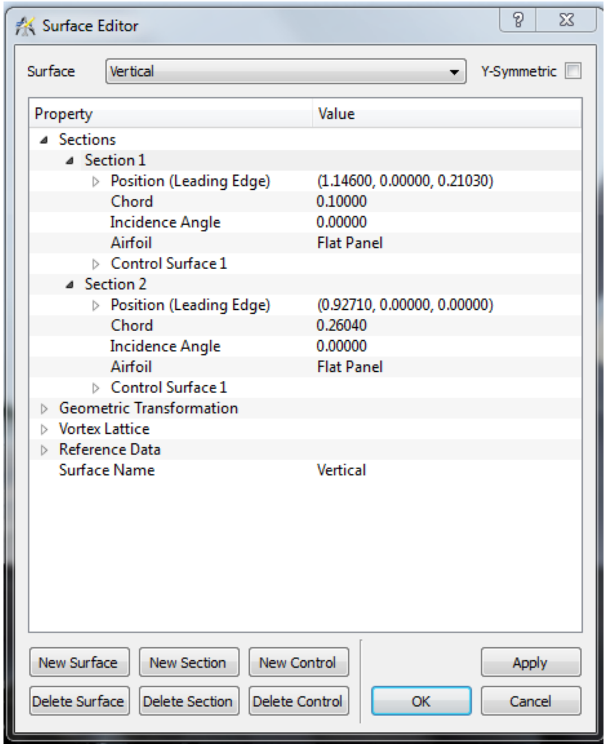
The figure above depicts the vortex settings of the horizontal tail. The settings were not set according to the guidelines in Section 4.1.10 because those guidelines had not been established in a system for modeling; however, the settings were appropriate.

1.3.4. Vertical Tail





The vertical was created with 2 sections.

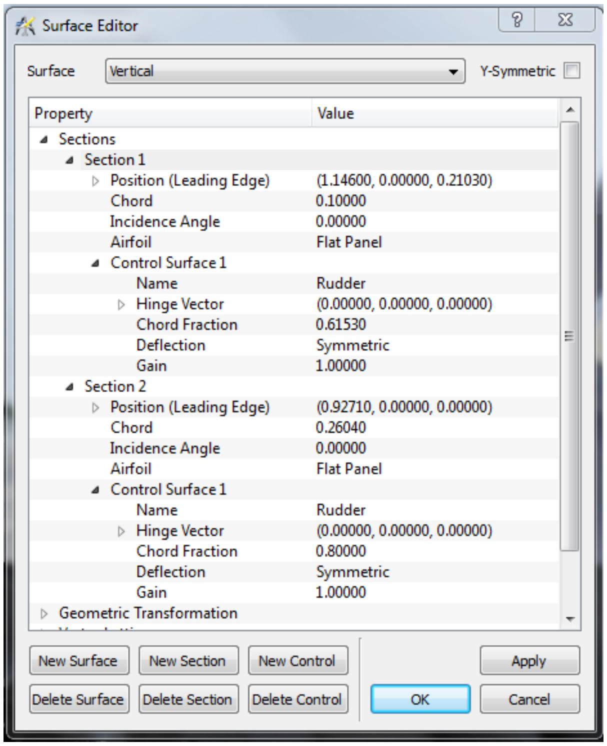


Surface Vertical Y-Symmetric □

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(1.14600, 0.00000, 0.21030)
Chord	0.10000
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Section 2	
Position (Leading Edge)	(0.92710, 0.00000, 0.00000)
Chord	0.26040
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Vertical

The figure above shows the sections and their positions. The surface was named "Vertical". Section 1 was designated as the top section to avoid the rudder deflection glitch described in Section 4.1.10. The vertical tail was a flat plate, so there was no airfoil file assigned to any of the sections.

1.3.4.1. Control Surfaces

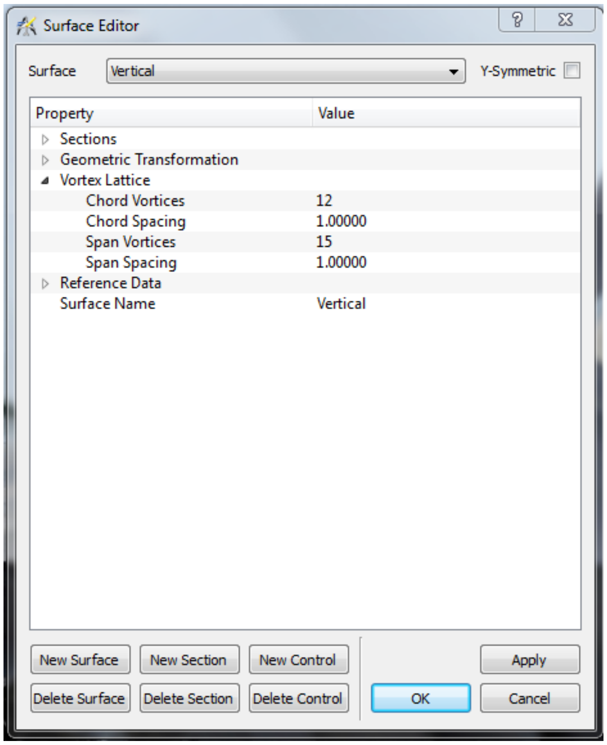


Surface: Vertical
Y-Symmetric □

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(1.14600, 0.00000, 0.21030)
Chord	0.10000
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Rudder
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.61530
Deflection	Symmetric
Gain	1.00000
Section 2	
Position (Leading Edge)	(0.92710, 0.00000, 0.00000)
Chord	0.26040
Incidence Angle	0.00000
Airfoil	Flat Panel
Control Surface 1	
Name	Rudder
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80000
Deflection	Symmetric
Gain	1.00000
Geometric Transformation	

There was one control surface on the vertical tail, the rudder. The rudder was assigned to sections 1 and 2 and was named "Rudder". The chord fraction of section 2 had to be adjusted because the rudder was constant width, the vertical tail was tapered, and the AVLEditor automatically assumed that the chord fraction stayed constant; therefore, the rudder was initially tapered. The "Deflection" was set at the default setting "Symmetric". "Gain" was set at the default value of 1.

1.3.4.2. Vortex Settings

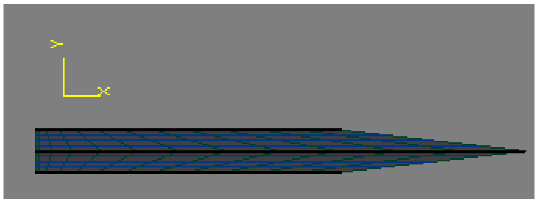


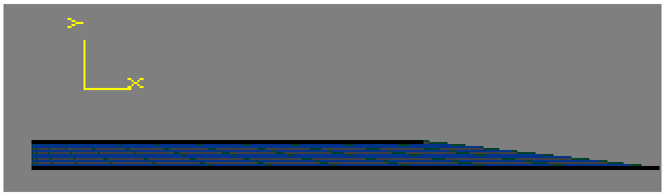
Surface: Vertical
Y-Symmetric □

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	1.00000
Span Vortices	15
Span Spacing	1.00000
▷ Reference Data	
Surface Name	Vertical

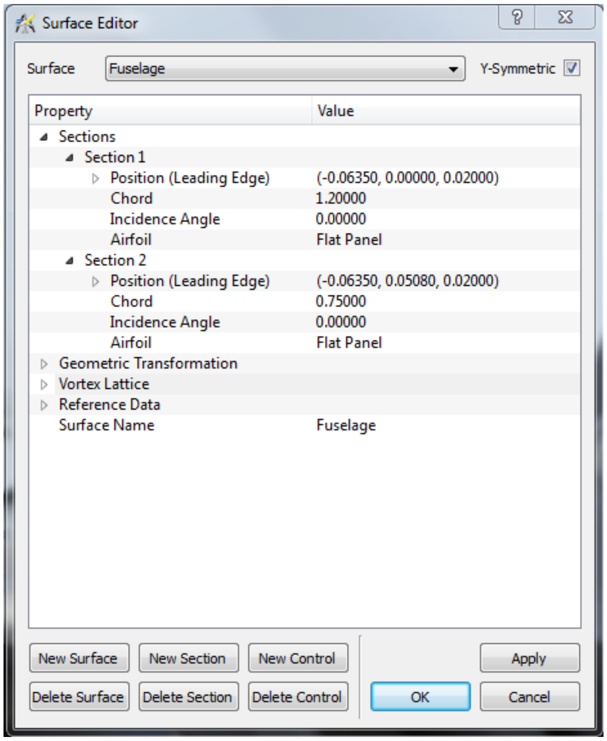
The figure above depicts the vortex settings for the vertical tail. The settings were not set according to the guidelines in Section 4.1.10 because those guidelines had not been established in a system for modeling; however, the settings were appropriate.

1.3.5. Fuselage Top





The top fuselage surface was made up of 2 sections. One section was at the center of the fuselage and one was at the outside diameter. The top section was located in the x-y plane and was mirrored across the x axis with the Y-Symmetry option. The length of the surface traveled downstream, along the x – axis.

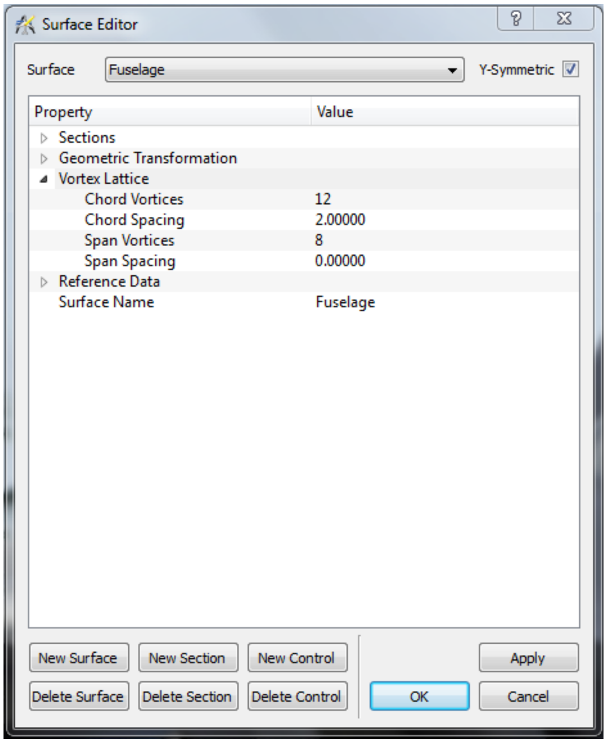


Surface: Fuselage   Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(-0.06350, 0.00000, 0.02000)
Chord	1.20000
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(-0.06350, 0.05080, 0.02000)
Chord	0.75000
Incidence Angle	0.00000
Airfoil	Flat Panel
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	Fuselage

The figure above shows the sections and their positions. The surface was named "Fuselage". The fuselage surface was treated as a flat panel.

1.3.5.1. Vortex Settings

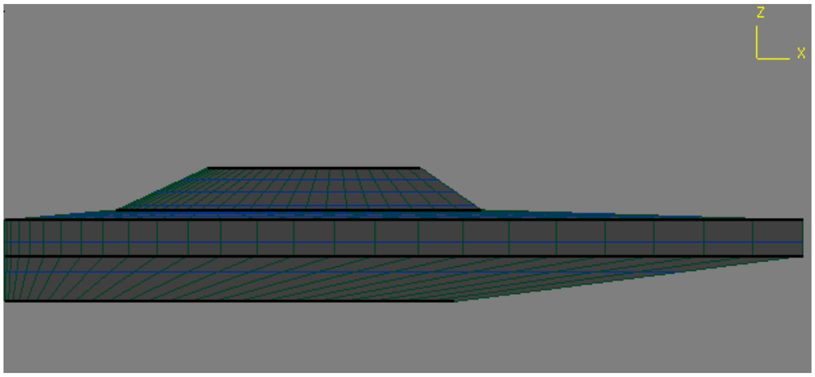


Surface: Fuselage ▾ Y-Symmetric ☑

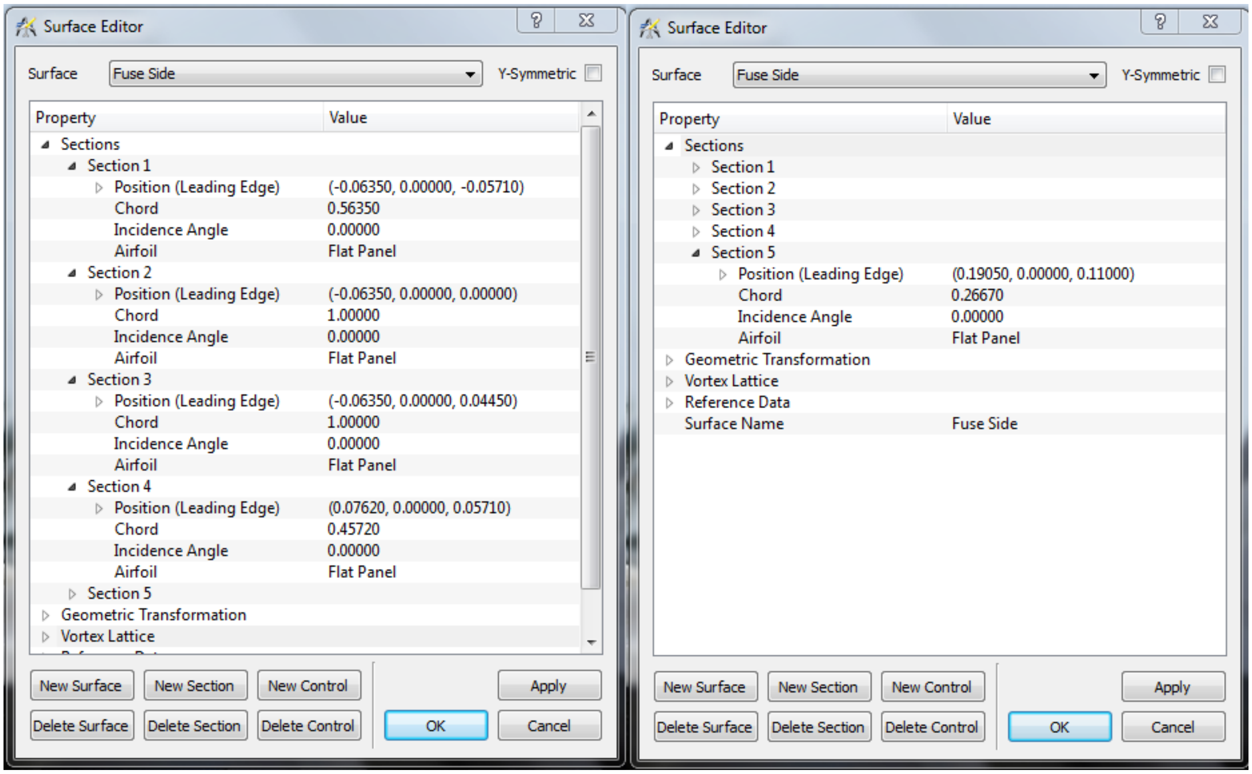
Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	2.00000
Span Vortices	8
Span Spacing	0.00000
▷ Reference Data	
Surface Name	Fuselage

The Vortex settings are shown in the figure above. The settings were not set according to the guidelines in Section 4.1.10 because those guidelines had not been established in a system for modeling; however, the settings were appropriate. If it were done again the span spacing should be set to -2.0 so that a couple vortices would bunch at the outside section.

1.3.6. Fuselage Side



The side fuselage surface was made up of 5 sections. The side fuselage surface was located in the x-z plane so it had no y – dimensions; thus, Y-Symmetry was not used. The length of the surface traveled downstream, along the x – axis.



Surface: Fuse Side ▾ Y-Symmetric □

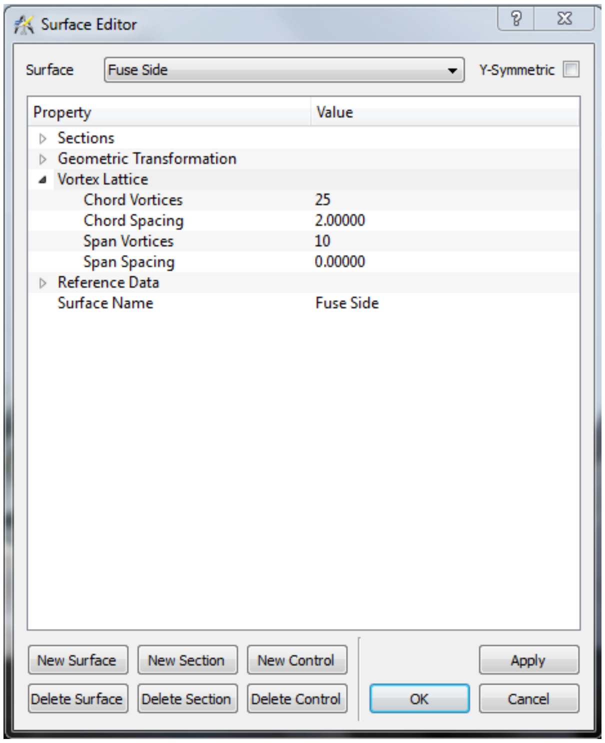
Property	Value
◢ Sections	
◢ Section 1	
▷ Position (Leading Edge)	(-0.06350, 0.00000, -0.05710)
Chord	0.56350
Incidence Angle	0.00000
Airfoil	Flat Panel
◢ Section 2	
▷ Position (Leading Edge)	(-0.06350, 0.00000, 0.00000)
Chord	1.00000
Incidence Angle	0.00000
Airfoil	Flat Panel
◢ Section 3	
▷ Position (Leading Edge)	(-0.06350, 0.00000, 0.04450)
Chord	1.00000
Incidence Angle	0.00000
Airfoil	Flat Panel
◢ Section 4	
▷ Position (Leading Edge)	(0.07620, 0.00000, 0.05710)
Chord	0.45720
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Section 5	
▷ Geometric Transformation	
▷ Vortex Lattice	

Surface: Fuse Side ▾ Y-Symmetric □

Property	Value
◢ Sections	
▷ Section 1	
▷ Section 2	
▷ Section 3	
▷ Section 4	
◢ Section 5	
▷ Position (Leading Edge)	(0.19050, 0.00000, 0.11000)
Chord	0.26670
Incidence Angle	0.00000
Airfoil	Flat Panel
▷ Geometric Transformation	
▷ Vortex Lattice	
▷ Reference Data	
Surface Name	Fuse Side

The figure above shows the sections and their positions. The surface was named “Fuse Side”. The fuselage surface was treated as a flat panel.

1.3.6.1. Vortex Settings



Surface Editor

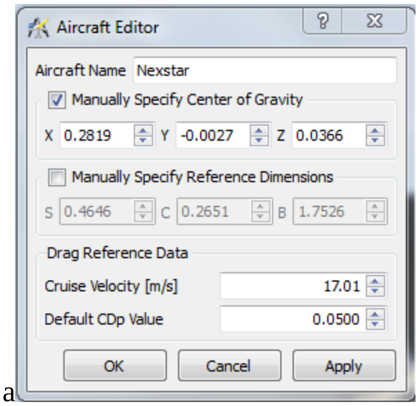
Surface: Fuse Side

Y-Symmetric □

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	25
Chord Spacing	2.00000
Span Vortices	10
Span Spacing	0.00000
▷ Reference Data	
Surface Name	Fuse Side

The vortex settings were not set according to the guidelines in Section 4.1.10 because those guidelines had not been established in a system for modeling. If the model was made again the chord spacing would be set to 1.0 so there would be vortices bunched up at the front and back, and the span spacing would be set to 1.0 so there would be vortices bunched up at the top and bottom.

1.3.7. Aircraft Data



Aircraft Name: Nexstar

☑ Manually Specify Center of Gravity

X 0.2819 Y -0.0027 Z 0.0366

☐ Manually Specify Reference Dimensions

S 0.4646 C 0.2651 B 1.7526

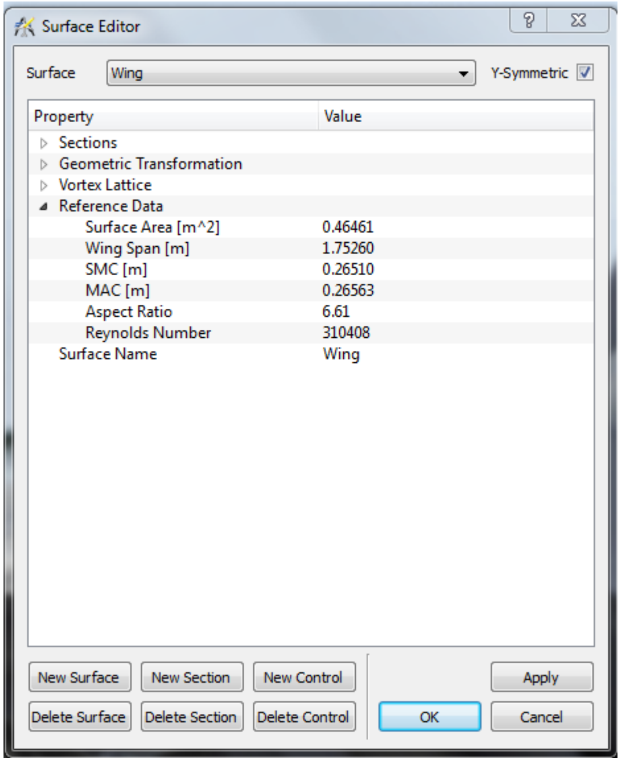
Drag Reference Data

Cruise Velocity [m/s]: 17.01

Default CDp Value: 0.0500

a

The figure above depicts the aircraft data that was set for Nexstar. The aircraft was named “Nexstar”. The center of gravity was manually specified using the results of the Excel Component Model. The center of gravity location was measured from (0,0,0) at the center of the firewall just as the aircraft geometry was. The units in the aircraft editor are in meters.



Surface: Wing Y-Symmetric ☑

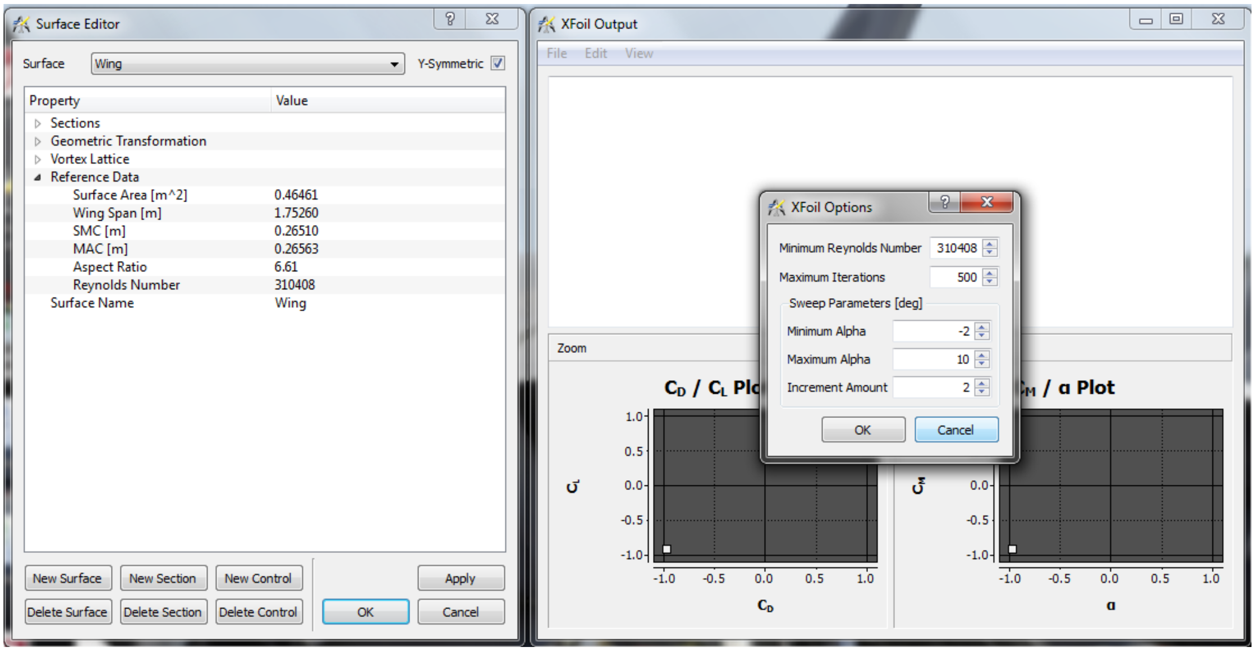
Property	Value
▷ Sections	
▷ Geometric Transformation	
▷ Vortex Lattice	
◢ Reference Data	
Surface Area [m^2]	0.46461
Wing Span [m]	1.75260
SMC [m]	0.26510
MAC [m]	0.26563
Aspect Ratio	6.61
Reynolds Number	310408
Surface Name	Wing

The automatically generated reference dimensions were acceptable. As shown in the figure above above the reference dimensions matched those of the Wing which is what the reference dimensions are supposed to be. The cruise velocity for Nexstar was set at 17.01 m/s. There was not a good model of Nexstar's profile drag. Initially a value of 0.02 was used generically; however, after a couple flights it was clear that the drag estimate was too low. In hardware in the loop simulations Nexstar descended way too quickly compared to the descents in actual flight. The drag was increased until the simulated descents more closely resembled descents in actual flights. The final value was 0.05.

1.3.8. Xfoil Analysis

1.3.8.1. Wing

At the time that the Nexstar was modeled the guidelines for running xfoil analysis had not yet been made. As a result xfoil analysis was done by simply running xfoil analysis through the AVLEditor. Do not run Xfoil analysis this way, instead use the guidelines in Section 4.1.8.



Surface Editor

Surface: Wing

Y-Symmetric ☑

Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Area [m^2]	0.46461
Wing Span [m]	1.75260
SMC [m]	0.26510
MAC [m]	0.26563
Aspect Ratio	6.61
Reynolds Number	310408
Surface Name	Wing

XFoil Options

Minimum Reynolds Number: 310408

Maximum Iterations: 500

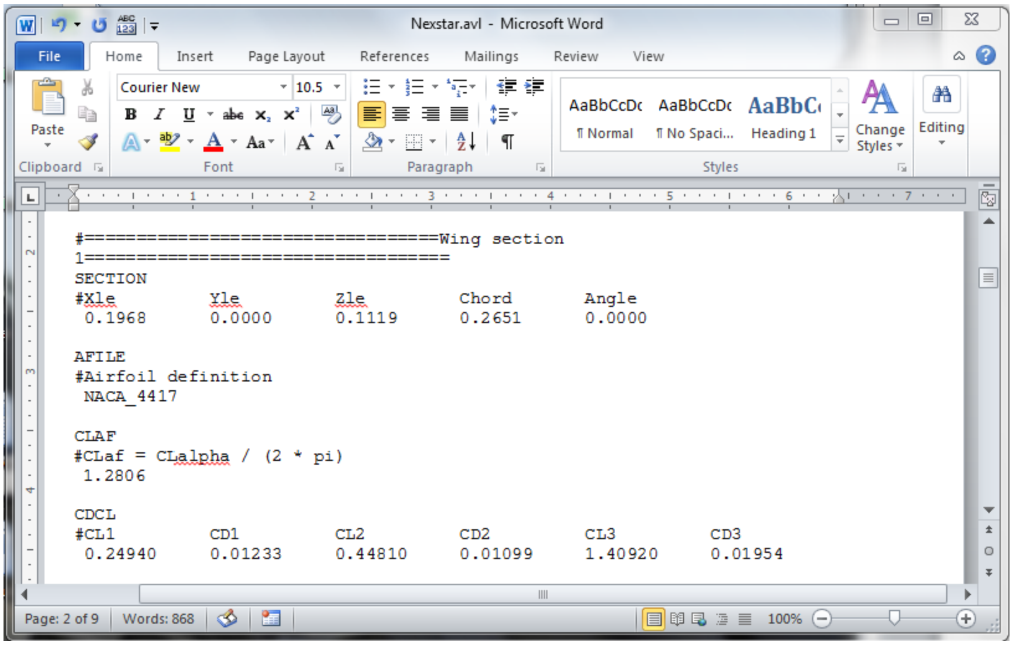
Sweep Parameters [deg]

Minimum Alpha: -2

Maximum Alpha: 10

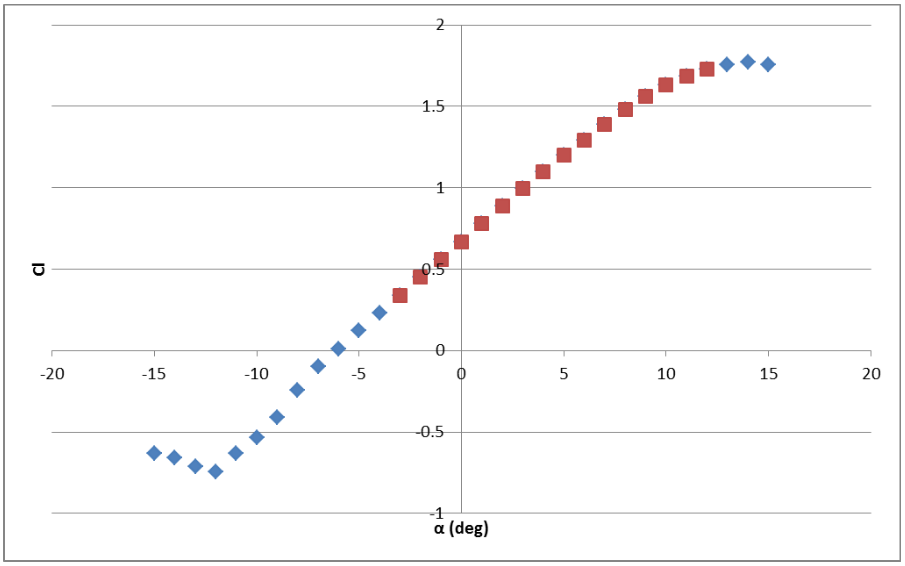
Increment Amount: 2

The minimum reynolds number was set to match the reynolds number of the wing, 310408. The alpha range was set at -2 to 10 by steps of 2. The results of the xfoil analysis were automatically generated into the aircraft's avl file by the AVLEditor.



```
#==================================Wing section
1==================================
SECTION
#Xle          Yle          Zle          Chord        Angle
 0.1968       0.0000       0.1119       0.2651       0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 1.2806

CDCL
#CL1          CD1          CL2          CD2          CL3          CD3
 0.24940      0.01233      0.44810      0.01099      1.40920      0.01954
```

Upon inspecting the values it was noticed that the CLAF value was 1.2806, well above 1.0. At the time the guidelines for determining CLAF had not been created; however, it was altered in a similar fashion. Xfoil was used to generate the 2D lift curve of the NACA 4417.



The 2D lift slope was estimated from the range of -3 <= α <= 12. The 2D lift slope was determined to be 0.096415 /deg and CLAF was calculated to be 0.8792. The calculations are shown below.

$$C_{l_{\propto}} = 0.096415 \,/\, \deg * \frac{180}{\pi} = 5.524177 \,/rad$$

$$CLAF = \frac{C_{l_{\propto}}}{2\pi} = \frac{5.524177}{2\pi} = 0.8792$$

The drag polar generated by AVLEditor's xfoil analysis was used as well. No xfoil analysis was ran on the tail surfaces because they were flat plates.

1.3.9. AVL Aircraft File

```
#*************************************************************
**********************
# AVL dataset for Nexstar model
# Generated by AVL Model Editor on 18 Mar 2014
#*************************************************************
**********************
Nexstar
#Mach
 0.0500
#IYsym       IZsym       Zsym
 0           0           0.0000
#Sref        Cref        Bref
#@Auto-generate
 0.4646      0.2651      1.7526


#*************************************************************
**********************
# AVL Axes:
#  +X   downstream
#  +Y   out right wing
#  +Z   up
#*************************************************************
**********************


#Xref        Yref        Zref
0.2819      -0.0027     0.0366
#CDp
0.0500
```

```
#*************************************************************
**********************
# Surfaces
#*************************************************************
**********************


#=======================================Wing==================
======================
#@Yduplicate 3 0.00000 Wing
SURFACE
RWing
#Nchord      Cspace      Nspan       Sspace
 15          1.0000      25          2.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 1


#==================================Wing section
1===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      0.0000      0.1119      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954
```

```
#==================================Wing section
2===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      0.0760      0.1170      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
#@Basename Aileron
 RAileron    1.0000      0.8440      0.0000      0.0000
0.0000      1


#==================================Wing section
3===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      0.8763      0.1675      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
#@Basename Aileron
 RAileron    1.0000      0.8440      0.0000      0.0000
0.0000      1
```

```
#==================================Wing
(mirror)==================================
#@Ignore
SURFACE
LWing
#Nchord      Cspace      Nspan       Sspace
 15          1.0000      25          -2.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 1


#=================================Wing section
4==================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      -0.8763     0.1675      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
#@Basename Aileron
 LAileron    1.0000      0.8440      0.0000      0.0000
0.0000      1
```

```
#==================================Wing section
5===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      -0.0760     0.1170      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
#@Basename Aileron
 LAileron    1.0000      0.8440      0.0000      0.0000
0.0000      1


#==================================Wing section
6===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1968      0.0000      0.1119      0.2651      0.0000

AFILE
#Airfoil definition
 NACA_4417

CLAF
#CLaf = CLalpha / (2 * pi)
 0.8792

CDCL
#CL1         CD1         CL2         CD2         CL3
CD3
 0.24940     0.01233     0.44810     0.01099     1.40920
0.01954


#====================================Horizontal===============
======================
SURFACE
Horizontal
```

```
#Nchord      Cspace      Nspan       Sspace
 15          1.0000      20          1.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000


#===============================Horizontal section
1================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 1.0779      -0.2984     0.0040      0.1333      0.0000

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
 Elevator    1.0000      0.7000      0.0000      0.0000
0.0000      1


#===============================Horizontal section
2================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 1.0160      0.0000      0.0040      0.2032      0.0000

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
 Elevator    1.0000      0.7000      0.0000      0.0000
0.0000      1


#===============================Horizontal section
3================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 1.0779      0.2984      0.0040      0.1333      0.0000

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
 Elevator    1.0000      0.7000      0.0000      0.0000
0.0000      1
```

```
#=====================================Vertical================
======================
SURFACE
Vertical
#Nchord      Cspace      Nspan       Sspace
 12          1.0000      15          1.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000


#================================Vertical section
1=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 1.1460      0.0000      0.2103      0.1000      0.0000

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
 Rudder      1.0000      0.6153      0.0000      0.0000
0.0000      1


#================================Vertical section
2=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.9271      0.0000      0.0000      0.2604      0.0000

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec
Zhvec       SgnDup
 Rudder      1.0000      0.8000      0.0000      0.0000
0.0000      1


#=====================================Fuse
Side=====================================
SURFACE
Fuse Side
#Nchord      Cspace      Nspan       Sspace
 25          2.0000      10          0.0000
```

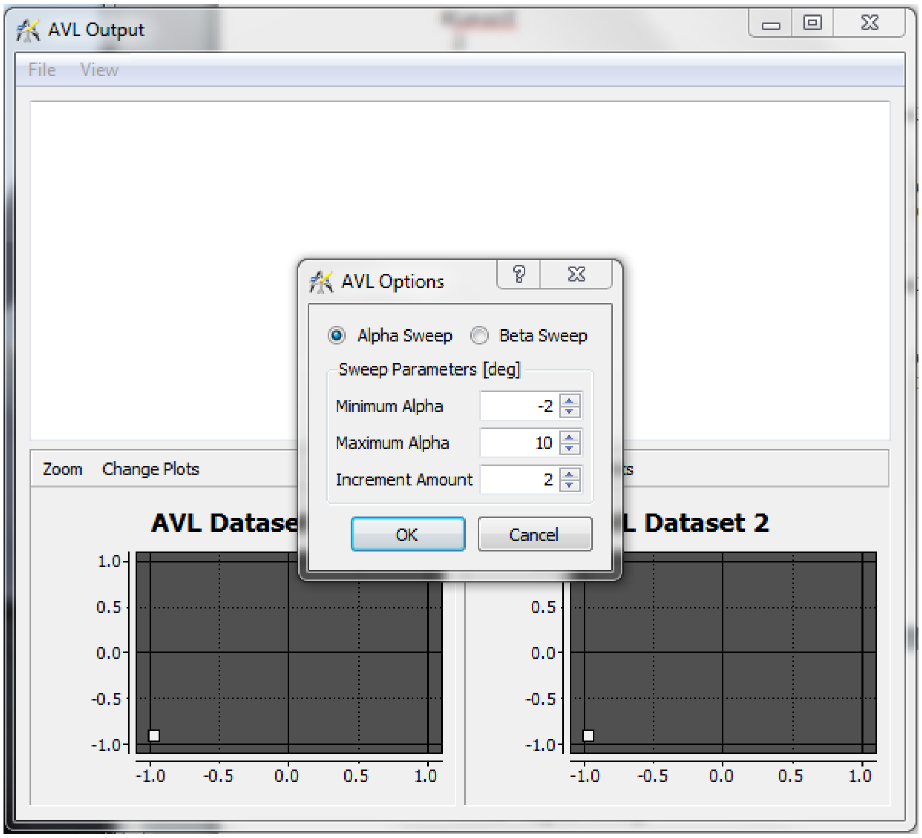
```
SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000


#=================================Fuse Side section
1=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0000      -0.0571     0.5635      0.0000


#=================================Fuse Side section
2=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0000      0.0000      1.0000      0.0000


#=================================Fuse Side section
3=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0000      0.0445      1.0000      0.0000


#=================================Fuse Side section
4=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0762      0.0000      0.0571      0.4572      0.0000


#=================================Fuse Side section
5=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.1905      0.0000      0.1100      0.2667      0.0000


#====================================Fuselage================
======================
#@Yduplicate 2 0.00000 Fuselage
SURFACE
RFuselage
#Nchord      Cspace      Nspan       Sspace
```

```
 12          2.0000      4           0.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 2


#=================================Fuselage section
1=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0000      0.0200      1.2000      0.0000


#=================================Fuselage section
2=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0508      0.0200      0.7500      0.0000


#=================================Fuselage
(mirror)=================================
#@Ignore
SURFACE
LFuselage
#Nchord      Cspace      Nspan       Sspace
 12          2.0000      4           0.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000

INDEX
```

```
#Lsurf
 2


#================================Fuselage section
3=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     -0.0508     0.0200      0.7500      0.0000


#================================Fuselage section
4=================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 -0.0635     0.0000      0.0200      1.2000      0.0000
```

1.3.10. Alpha Sweep

At the time Nexstar was modeled the guidelines for performing an alpha sweep had not yet been determined; therefore, there was no predetermined linear range to define the range of the alpha sweep. The alpha sweep was ran at the default settings with the minimum alpha at -2, maximum alpha at 10, and increment amount at 2. The alpha xml file was saved as “alpha” in Nexstar’s aircraft folder. Do not perform alpha sweeps this way, instead use the method in Section 4.2.



AVL Options

● Alpha Sweep ○ Beta Sweep

Sweep Parameters [deg]

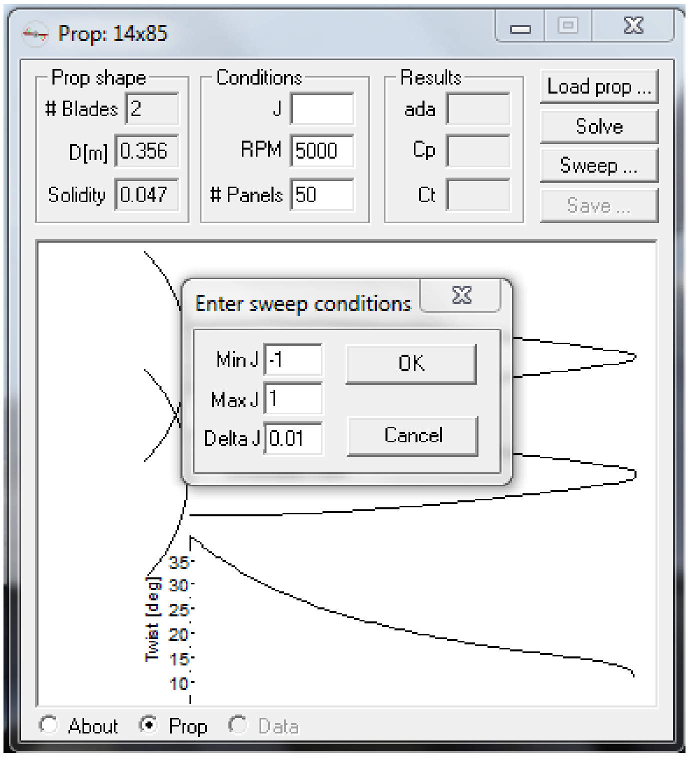
Minimum Alpha -2

Maximum Alpha 10

Increment Amount 2

1.4. Propeller File

Nexstar used an APC 14x8.5 inch propeller. The propeller file was generated using cloud cap's "createprop.m" script and "Prop.exe". The createprop MATLAB script was used to generate the propeller file that defines the shape of the propeller for the Prop program to load. The Prop program generates the propeller file for the simulator. The process was done according to the guidelines in the CCT Matlab document "CreatingPropModels".


Prop: 14x85

Prop shape
Blades 2
D[m] 0.356
Solidity 0.047

Conditions
J
RPM 5000
Panels 50

Results
ada
Cp
Ct

Enter sweep conditions
Min J -1
Max J 1
Delta J 0.01

The figure above depicts the settings that the prop file was generated at. The sweep advance ratio started at -1, and iterated to positive 1 by 0.01.

1.5. Control Surfaces Calibration

The control surface deflections were calibrated by commanding test pulse widths to the piccolo and measuring the corresponding surface deflections on the aircraft. Specific pulse widths were commanded to the control surfaces with the “Surface Test” function that is available in the Surface Calibration window in PCC.

The control surface deflections of the aileron, elevator, and rudder were measured using a digital angle level. The nosegear deflections were measured by setting the nosegear on a secured piece of paper and tracing a straight line on the right outside edge of the tailwheel and calculating the angle between each line and the line at 0 degrees (centered) nosegear deflection.

The following steps outline how each control surface was measured, except for the throttle.

1) Determined the pulse width for maximum down deflection.

2) Determined the pulse width for maximum up deflection.

3) Determine the pulse width for zero deflection.

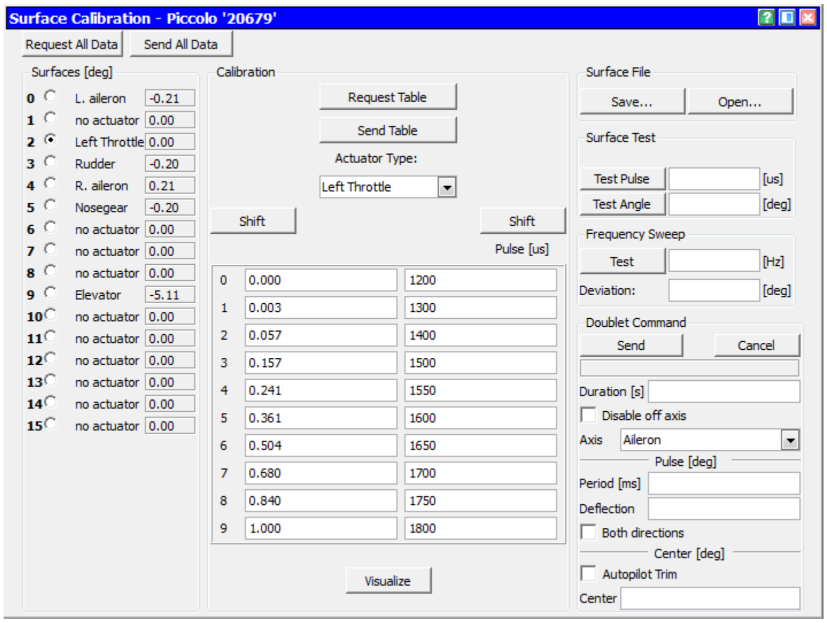
4) Attempted to create an even distribution of change in deflection angles to fill in the empty points between 0 and the maximum and minimum deflections.

The throttle calibration was slightly different since throttle is calibrated using rpm as an indicator of power. The motor was an electric motor. The rpm was measured using a hand held digital rpm readout. The process began at 1000 ms pulse width and increased by 50 until the motor actually commanded an rpm. Once the motor began spinning the propeller the minimum pulse width was set as the last pulse width signal that had resulted in 0 throttle. The pulse width was increased by 100 and 50 ms increments until maximum rpm was reached. The maximum rpm was limited by the current draw of the motor. The maximum current draw of the motor was 60A. The current was measured using a hand held amp meter.

Pulse Width	RPM	RPM^3	Throttle
1200	0	0	0
1300	1442	2998442888	0.00289
1400	3901	59364641701	0.057227
1500	5460	1.62771E+11	0.15691
1550	6300	2.50047E+11	0.241043
1600	7211	3.74961E+11	0.361459
1650	8058	5.23217E+11	0.504376
1700	8900	7.04969E+11	0.679583
1750	9550	8.70984E+11	0.839619
1800	10123	1.03736E+12	1

The table above contains the results of the throttle calibration test. The throttle settings were calculated with rpm cubed since rpm cubed is proportional to power; thus, percent throttle would be equivalent to percent power.

$$Throttle = \frac{RPM^3}{RPM_{max}^3}$$



Surface Calibration - Piccolo '20679'

Surfaces [deg]

0	○	L. aileron	-0.21
1	○	no actuator	0.00
2	●	Left Throttle	0.00
3	○	Rudder	-0.20
4	○	R. aileron	0.21
5	○	Nosegear	-0.20
6	○	no actuator	0.00
7	○	no actuator	0.00
8	○	no actuator	0.00
9	○	Elevator	-5.11
10	○	no actuator	0.00
11	○	no actuator	0.00
12	○	no actuator	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

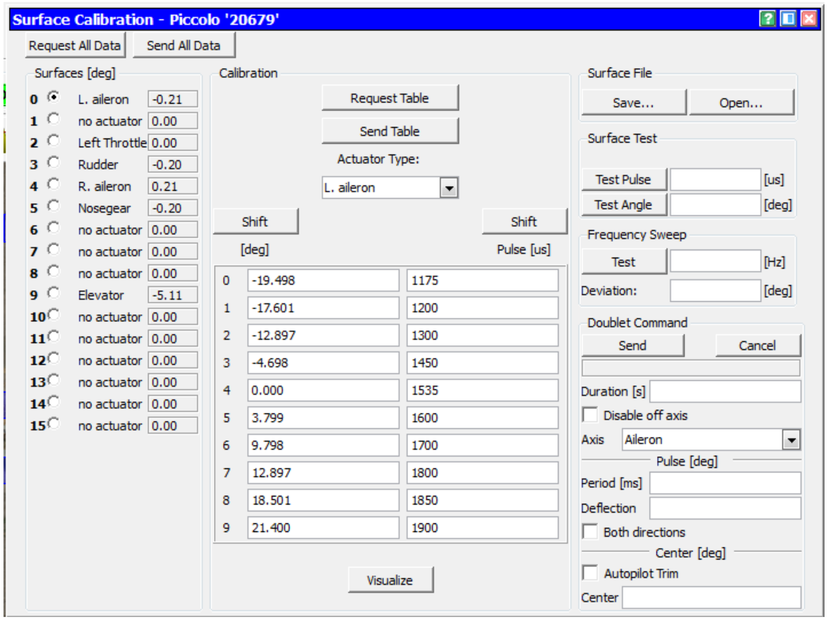
Calibration

Actuator Type: Left Throttle

		Pulse [us]
0	0.000	1200
1	0.003	1300
2	0.057	1400
3	0.157	1500
4	0.241	1550
5	0.361	1600
6	0.504	1650
7	0.680	1700
8	0.840	1750
9	1.000	1800

Axis: Aileron

1.5.1. LAileron



Surfaces [deg]

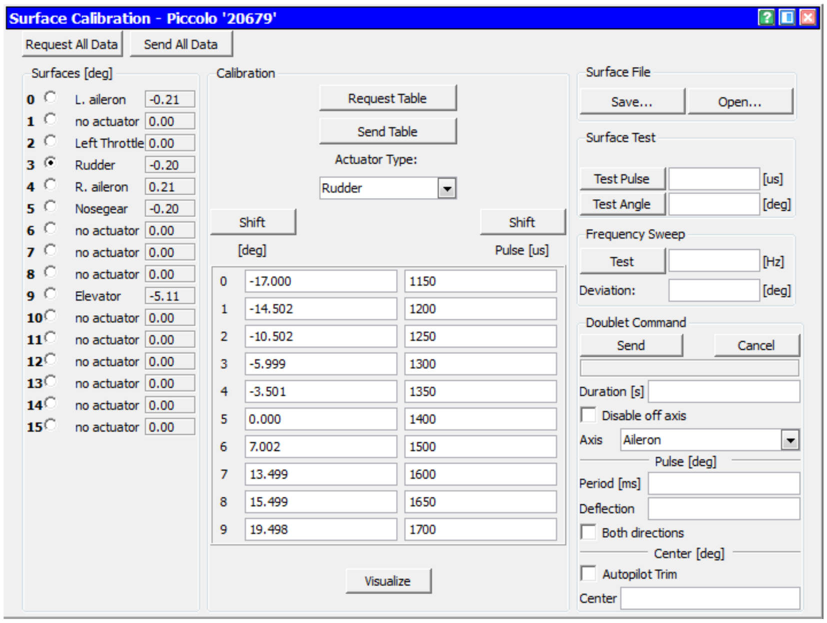
0	●	L. aileron	-0.21
1	○	no actuator	0.00
2	○	Left Throttle	0.00
3	○	Rudder	-0.20
4	○	R. aileron	0.21
5	○	Nosegear	-0.20
6	○	no actuator	0.00
7	○	no actuator	0.00
8	○	no actuator	0.00
9	○	Elevator	-5.11
10	○	no actuator	0.00
11	○	no actuator	0.00
12	○	no actuator	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-19.498	1175
1	-17.601	1200
2	-12.897	1300
3	-4.698	1450
4	0.000	1535
5	3.799	1600
6	9.798	1700
7	12.897	1800
8	18.501	1850
9	21.400	1900

Test Pulse [us]
Test Angle [deg]
Test [Hz]
Deviation: [deg]
Duration [s]
□ Disable off axis
Axis Aileron
Period [ms]
Deflection
□ Both directions
□ Autopilot Trim
Center

1.5.2. Rudder



Surfaces [deg]

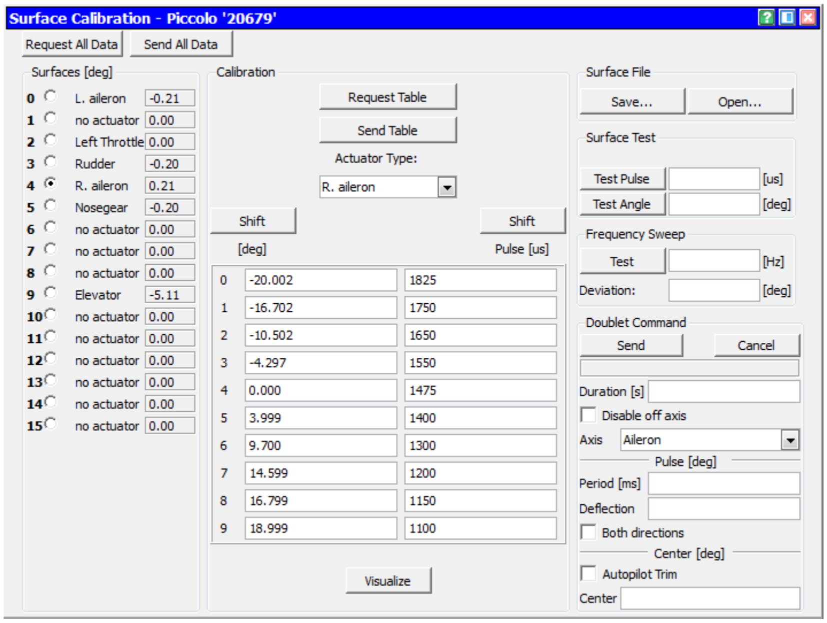
0	○	L. aileron	-0.21
1	○	no actuator	0.00
2	○	Left Throttle	0.00
3	●	Rudder	-0.20
4	○	R. aileron	0.21
5	○	Nosegear	-0.20
6	○	no actuator	0.00
7	○	no actuator	0.00
8	○	no actuator	0.00
9	○	Elevator	-5.11
10	○	no actuator	0.00
11	○	no actuator	0.00
12	○	no actuator	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: Rudder

	[deg]	Pulse [us]
0	-17.000	1150
1	-14.502	1200
2	-10.502	1250
3	-5.999	1300
4	-3.501	1350
5	0.000	1400
6	7.002	1500
7	13.499	1600
8	15.499	1650
9	19.498	1700

Test Pulse [us]
Test Angle [deg]
Test [Hz]
Deviation: [deg]
Duration [s]
□ Disable off axis
Axis Aileron
Period [ms]
Deflection
□ Both directions
□ Autopilot Trim
Center

1.5.3. RAileron


Surface Calibration - Piccolo '20679'

Surfaces [deg]

0	○	L. aileron	-0.21
1	○	no actuator	0.00
2	○	Left Throttle	0.00
3	○	Rudder	-0.20
4	●	R. aileron	0.21
5	○	Nosegear	-0.20
6	○	no actuator	0.00
7	○	no actuator	0.00
8	○	no actuator	0.00
9	○	Elevator	-5.11
10	○	no actuator	0.00
11	○	no actuator	0.00
12	○	no actuator	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: R. aileron

	[deg]	Pulse [us]
0	-20.002	1825
1	-16.702	1750
2	-10.502	1650
3	-4.297	1550
4	0.000	1475
5	3.999	1400
6	9.700	1300
7	14.599	1200
8	16.799	1150
9	18.999	1100

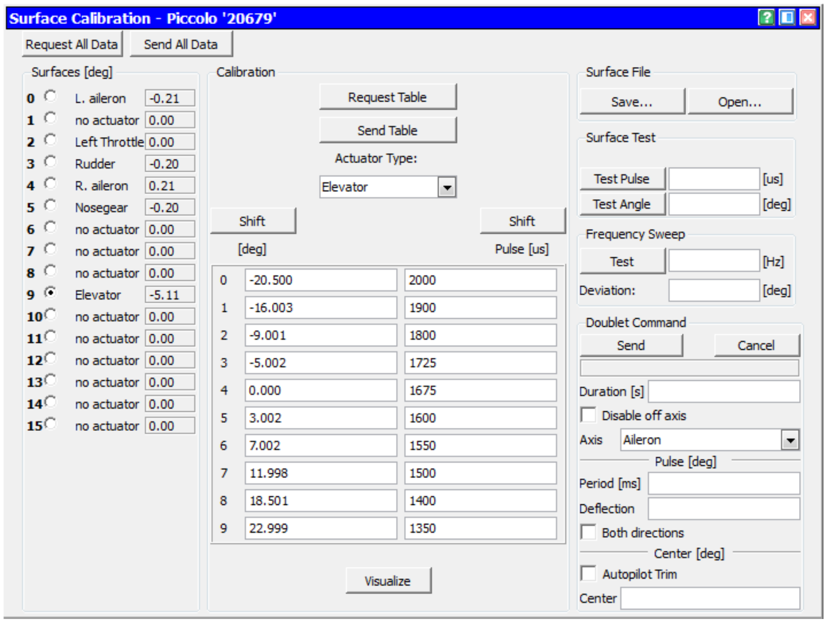
□ Disable off axis

Axis: Aileron

□ Both directions

□ Autopilot Trim

1.5.4. Elevator


Surface Calibration - Piccolo '20679'

Surfaces [deg]

0	○	L. aileron	-0.21
1	○	no actuator	0.00
2	○	Left Throttle	0.00
3	○	Rudder	-0.20
4	○	R. aileron	0.21
5	○	Nosegear	-0.20
6	○	no actuator	0.00
7	○	no actuator	0.00
8	○	no actuator	0.00
9	●	Elevator	-5.11
10	○	no actuator	0.00
11	○	no actuator	0.00
12	○	no actuator	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: Elevator

	[deg]	Pulse [us]
0	-20.500	2000
1	-16.003	1900
2	-9.001	1800
3	-5.002	1725
4	0.000	1675
5	3.002	1600
6	7.002	1550
7	11.998	1500
8	18.501	1400
9	22.999	1350

□ Disable off axis

Axis: Aileron

□ Both directions

□ Autopilot Trim

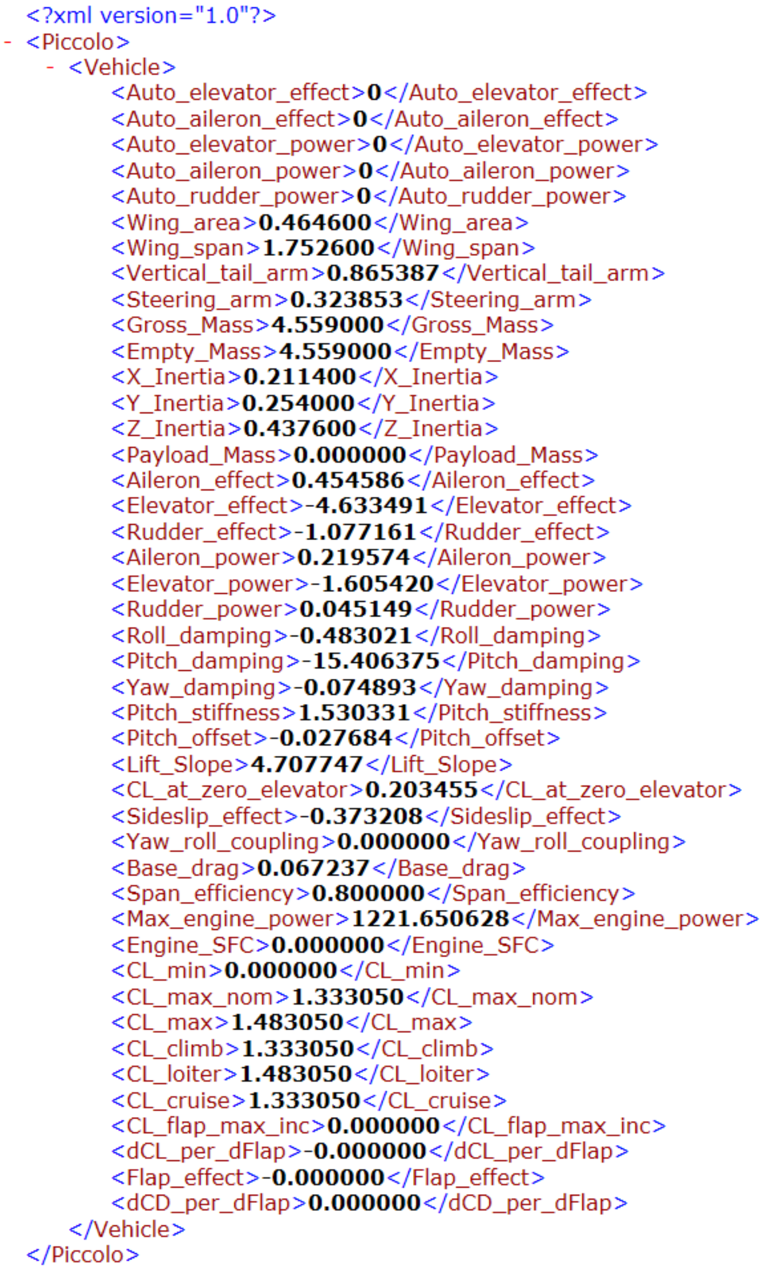
1.6. Simulator File

```
// Nexstar
// SIMULATOR MODEL
// Oklahoma State University

// Aerodynamics data from AVL
Alpha_sweep_xml_file=alpha.xml


// Mapping of the channel numbers in AVL to channel numbers in the

// Right aileron
Channel_d1=4

// Left aileron
Channel_d2=0

// Elevator
Channel_d3=9

// Rudder
Channel_d4=3

// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//


// Gross take-off mass of the aircraft, in kg
Gross_Mass=4.559

// Mass of aircraft without fuel, in kg
Empty_Mass=4.559

// Moments of Inertia, in kg*m^2
Roll_Inertia=0.2114
Pitch_Inertia=0.2540
Yaw_Inertia=0.4376


//--------  Surface Effectiveness Adjustments  --------//

Cld_scaler_d1=0.650976
Cld_scaler_d2=0.650976


//-------------------- PROPULSION --------------------//

// Engine is electric
Left_Engine_Type=1
Left_Engine_Channel=2
Left_Motor_NominalInputVoltage=29.6
```

```
Left_Motor_RPMConstant=375
Left_Motor_NoLoadCurrent=1.5
Left_Motor_TerminalResistance=0.0110
Left_Motor_ThermalResistance=0.018

// Prop is a 14x8.5 APC Electric Prop
Left_Actuator_Type=0


// Prop diameter, in m
Left_Prop_Diameter=0.3556

// Position of propeller hub wrt to aircraft cg, in m

// Note dimensions are positive if upstream of CG
Left_Prop_X=0.08255
Left_Prop_Y=0.0
Left_Prop_Z=0.02461

//Propellor Angles to Aircraft Frame
Left_Prop_Pan=0
Left_Prop_Tilt=0

//Propeller gear ratio (greater than 1 if rotates slower than the engine)
Left_Prop_GearRatio=1

//Propeller direction of rotation (1 = cw -1 = ccw)
Left_Prop_Sense=1

// Moment of inertia in kg/m^2
Left_Prop_Inertia=0.0000922

// Propeller coefficients look-up table
Left_Prop_LUT=14x85.prd


//---------------  GROUND CONTACT POINTS  ---------------//

NoseWheel_Position_X=0.28824
NoseWheel_Position_Y=-0.0099
NoseWheel_Position_Z=0.17553

RightWheel_Position_X=-0.035613
RightWheel_Position_Y=0.193234
RightWheel_Position_Z=0.17553

LeftWheel_Position_X=-0.035613
LeftWheel_Position_Y=-0.18777
LeftWheel_Position_Z=0.17553

ContactPoint_Nose_Position_X=0.36444
```

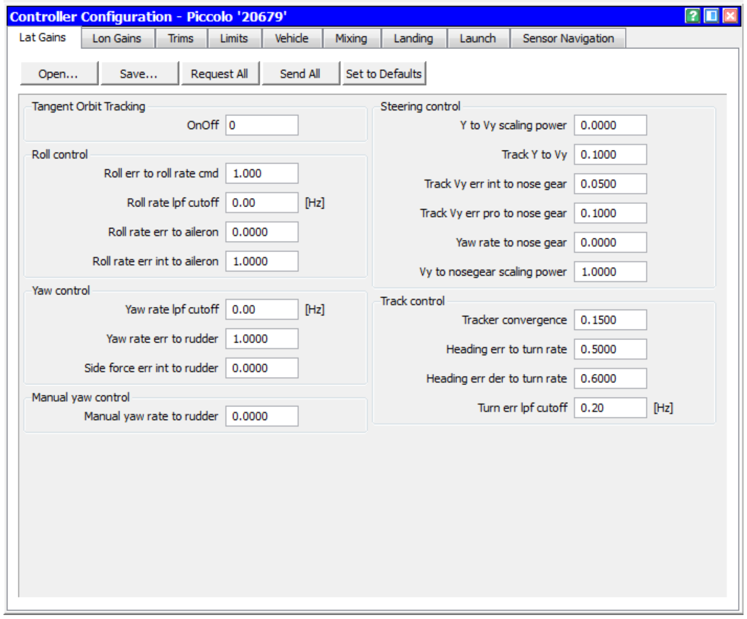
```
ContactPoint_Nose_Position_Y=0.00273
ContactPoint_Nose_Position_Z=-0.00408


//----------------- AVIONICS MOUNTING -----------------//

// Avionics (IMU sensor) orientation with respect to the aircraft body axes
// Euler angles in deg
IMU_Sensor_Roll_Angle=0.0
IMU_Sensor_Pitch_Angle=0.0
IMU_Sensor_Yaw_Angle=-180.0

// Avionics (IMU sensor) position vector with respect to the aircraft CG, in body axes
// Vector components in m
IMU_Sensor_Position_X=-0.18801
IMU_Sensor_Position_Y=0.00273
IMU_Sensor_Position_Z=0.00408
```

1.7. Simulator Vehicle File



```
  <?xml version="1.0"?>
- <Piccolo>
  - <Vehicle>
      <Auto_elevator_effect>0</Auto_elevator_effect>
      <Auto_aileron_effect>0</Auto_aileron_effect>
      <Auto_elevator_power>0</Auto_elevator_power>
      <Auto_aileron_power>0</Auto_aileron_power>
      <Auto_rudder_power>0</Auto_rudder_power>
      <Wing_area>0.464600</Wing_area>
      <Wing_span>1.752600</Wing_span>
      <Vertical_tail_arm>0.865387</Vertical_tail_arm>
      <Steering_arm>0.323853</Steering_arm>
      <Gross_Mass>4.559000</Gross_Mass>
      <Empty_Mass>4.559000</Empty_Mass>
      <X_Inertia>0.211400</X_Inertia>
      <Y_Inertia>0.254000</Y_Inertia>
      <Z_Inertia>0.437600</Z_Inertia>
      <Payload_Mass>0.000000</Payload_Mass>
      <Aileron_effect>0.454586</Aileron_effect>
      <Elevator_effect>-4.633491</Elevator_effect>
      <Rudder_effect>-1.077161</Rudder_effect>
      <Aileron_power>0.219574</Aileron_power>
      <Elevator_power>-1.605420</Elevator_power>
      <Rudder_power>0.045149</Rudder_power>
      <Roll_damping>-0.483021</Roll_damping>
      <Pitch_damping>-15.406375</Pitch_damping>
      <Yaw_damping>-0.074893</Yaw_damping>
      <Pitch_stiffness>1.530331</Pitch_stiffness>
      <Pitch_offset>-0.027684</Pitch_offset>
      <Lift_Slope>4.707747</Lift_Slope>
      <CL_at_zero_elevator>0.203455</CL_at_zero_elevator>
      <Sideslip_effect>-0.373208</Sideslip_effect>
      <Yaw_roll_coupling>0.000000</Yaw_roll_coupling>
      <Base_drag>0.067237</Base_drag>
      <Span_efficiency>0.800000</Span_efficiency>
      <Max_engine_power>1221.650628</Max_engine_power>
      <Engine_SFC>0.000000</Engine_SFC>
      <CL_min>0.000000</CL_min>
      <CL_max_nom>1.333050</CL_max_nom>
      <CL_max>1.483050</CL_max>
      <CL_climb>1.333050</CL_climb>
      <CL_loiter>1.483050</CL_loiter>
      <CL_cruise>1.333050</CL_cruise>
      <CL_flap_max_inc>0.000000</CL_flap_max_inc>
      <dCL_per_dFlap>-0.000000</dCL_per_dFlap>
      <Flap_effect>-0.000000</Flap_effect>
      <dCD_per_dFlap>0.000000</dCD_per_dFlap>
    </Vehicle>
  </Piccolo>
```

1.8. Lat Gains



Tangent Orbit Tracking

OnOff 0

Roll control

Roll err to roll rate cmd 1.000

Roll rate lpf cutoff 0.00 [Hz]

Roll rate err to aileron 0.0000

Roll rate err int to aileron 1.0000

Yaw control

Yaw rate lpf cutoff 0.00 [Hz]

Yaw rate err to rudder 1.0000

Side force err int to rudder 0.0000

Manual yaw control

Manual yaw rate to rudder 0.0000

Steering control

Y to Vy scaling power 0.0000

Track Y to Vy 0.1000

Track Vy err int to nose gear 0.0500

Track Vy err pro to nose gear 0.1000

Yaw rate to nose gear 0.0000

Vy to nosegear scaling power 1.0000

Track control

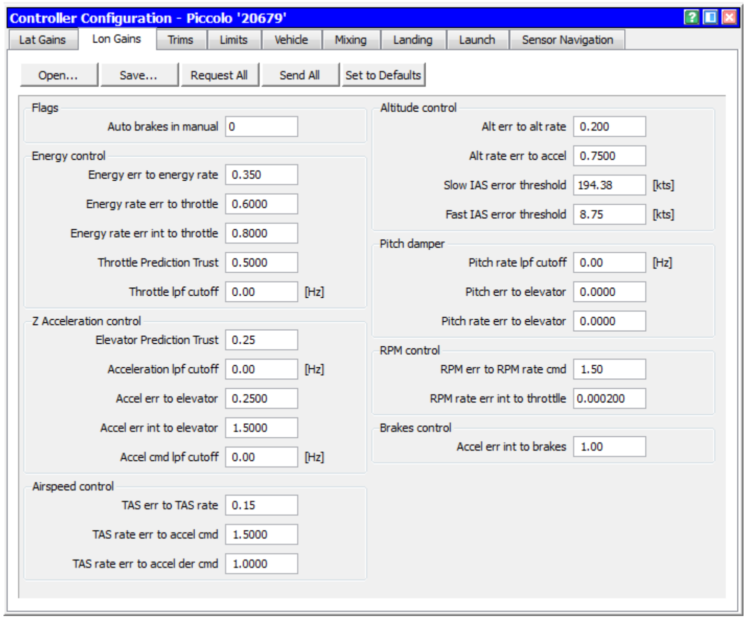
Tracker convergence 0.1500

Heading err to turn rate 0.5000

Heading err der to turn rate 0.6000

Turn err lpf cutoff 0.20 [Hz]

1.9. Lon Gains



Flags

Auto brakes in manual 0

Energy control

Energy err to energy rate 0.350

Energy rate err to throttle 0.6000

Energy rate err int to throttle 0.8000

Throttle Prediction Trust 0.5000

Throttle lpf cutoff 0.00 [Hz]

Z Acceleration control

Elevator Prediction Trust 0.25

Acceleration lpf cutoff 0.00 [Hz]

Accel err to elevator 0.2500

Accel err int to elevator 1.5000

Accel cmd lpf cutoff 0.00 [Hz]

Airspeed control

TAS err to TAS rate 0.15

TAS rate err to accel cmd 1.5000

TAS rate err to accel der cmd 1.0000

Altitude control

Alt err to alt rate 0.200

Alt rate err to accel 0.7500

Slow IAS error threshold 194.38 [kts]

Fast IAS error threshold 8.75 [kts]

Pitch damper

Pitch rate lpf cutoff 0.00 [Hz]

Pitch err to elevator 0.0000

Pitch rate err to elevator 0.0000

RPM control

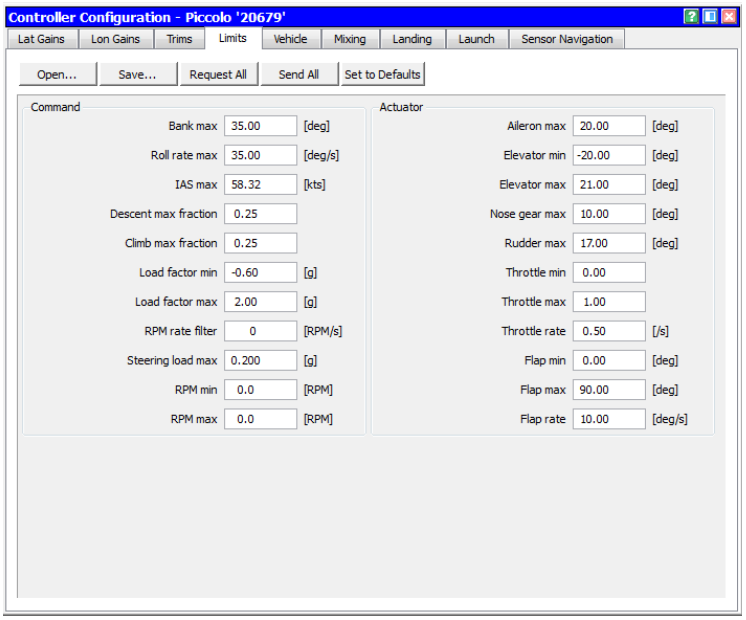
RPM err to RPM rate cmd 1.50

RPM rate err int to throttle 0.000200

Brakes control

Accel err int to brakes 1.00

1.10. Limits



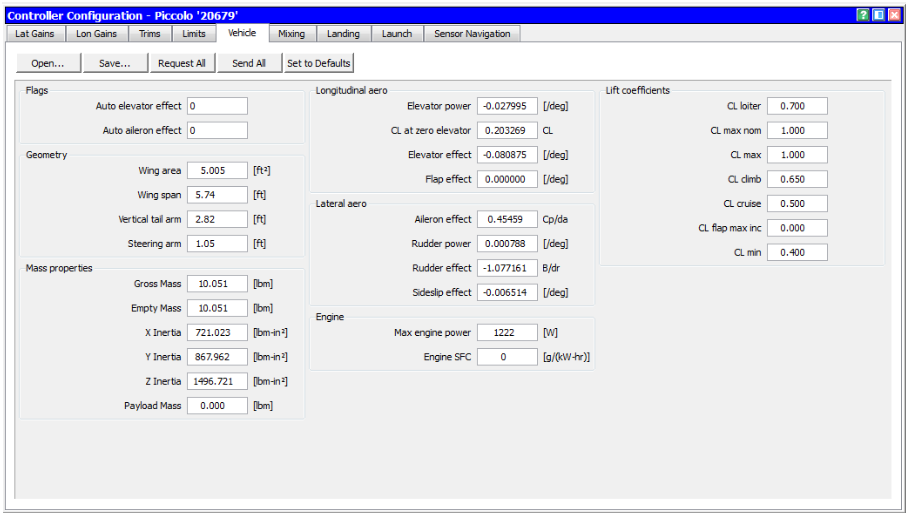
Command

Bank max 35.00 [deg]
Roll rate max 35.00 [deg/s]
IAS max 58.32 [kts]
Descent max fraction 0.25
Climb max fraction 0.25
Load factor min -0.60 [g]
Load factor max 2.00 [g]
RPM rate filter 0 [RPM/s]
Steering load max 0.200 [g]
RPM min 0.0 [RPM]
RPM max 0.0 [RPM]

Actuator

Aileron max 20.00 [deg]
Elevator min -20.00 [deg]
Elevator max 21.00 [deg]
Nose gear max 10.00 [deg]
Rudder max 17.00 [deg]
Throttle min 0.00
Throttle max 1.00
Throttle rate 0.50 [/s]
Flap min 0.00 [deg]
Flap max 90.00 [deg]
Flap rate 10.00 [deg/s]

1.11. Vehicle Parameters



Flags

Auto elevator effect 0
Auto aileron effect 0

Geometry

Wing area 5.005 [ft²]
Wing span 5.74 [ft]
Vertical tail arm 2.82 [ft]
Steering arm 1.05 [ft]

Mass properties

Gross Mass 10.051 [lbm]
Empty Mass 10.051 [lbm]
X Inertia 721.023 [lbm-in²]
Y Inertia 867.962 [lbm-in²]
Z Inertia 1496.721 [lbm-in²]
Payload Mass 0.000 [lbm]

Longitudinal aero

Elevator power -0.027995 [/deg]
CL at zero elevator 0.203269 CL
Elevator effect -0.080875 [/deg]
Flap effect 0.000000 [/deg]

Lateral aero

Aileron effect 0.45459 Cp/da
Rudder power 0.000788 [/deg]
Rudder effect -1.077161 B/dr
Sideslip effect -0.006514 [/deg]

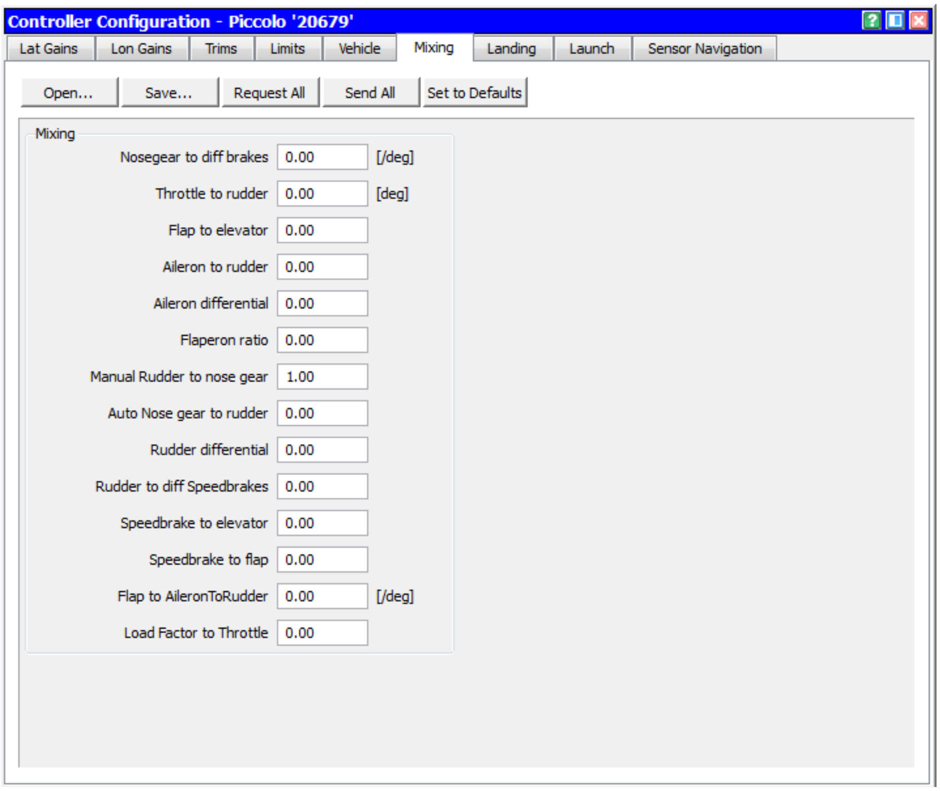
Engine

Max engine power 1222 [W]
Engine SFC 0 [g/(kW-hr)]

Lift coefficients

CL loiter 0.700
CL max nom 1.000
CL max 1.000
CL climb 0.650
CL cruise 0.500
CL flap max inc 0.000
CL min 0.400

1.12. Mixing



Mixing

Nosegear to diff brakes 0.00 [/deg]

Throttle to rudder 0.00 [deg]

Flap to elevator 0.00

Aileron to rudder 0.00

Aileron differential 0.00

Flaperon ratio 0.00

Manual Rudder to nose gear 1.00

Auto Nose gear to rudder 0.00

Rudder differential 0.00

Rudder to diff Speedbrakes 0.00

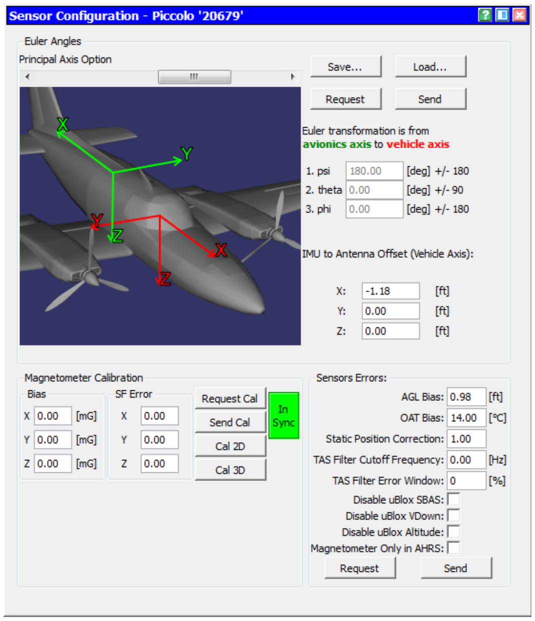
Speedbrake to elevator 0.00

Speedbrake to flap 0.00

Flap to AileronToRudder 0.00 [/deg]

Load Factor to Throttle 0.00

1.13. Sensor Configuration



Euler transformation is from
avionics axis to vehicle axis

1. psi 180.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 0.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]

Bias

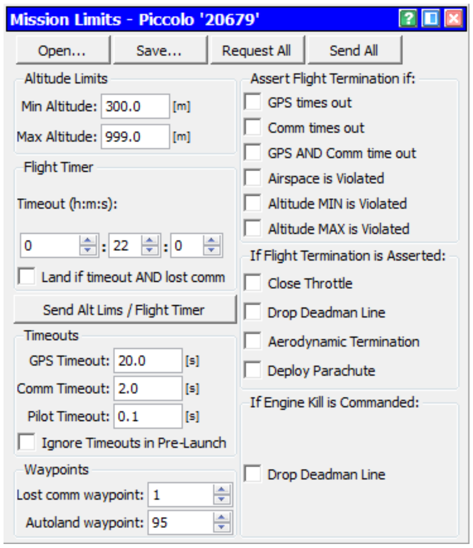
X 0.00 [mG]
Y 0.00 [mG]
Z 0.00 [mG]

SF Error

X 0.00
Y 0.00
Z 0.00

AGL Bias: 0.98 [ft]
OAT Bias: 14.00 [°C]
Static Position Correction: 1.00
TAS Filter Cutoff Frequency: 0.00 [Hz]
TAS Filter Error Window: 0 [%]
Disable uBlox SBAS: □
Disable uBlox VDown: □
Disable uBlox Altitude: □
Magnetometer Only in AHRS: □

1.14. Mission Limits



Mission Limits - Piccolo '20679'

Open... | Save... | Request All | Send All

Altitude Limits
Min Altitude: 300.0 [m]
Max Altitude: 999.0 [m]

Flight Timer
Timeout (h:m:s): 0 : 22 : 0
□ Land if timeout AND lost comm
Send Alt Lims / Flight Timer

Timeouts
GPS Timeout: 20.0 [s]
Comm Timeout: 2.0 [s]
Pilot Timeout: 0.1 [s]
□ Ignore Timeouts in Pre-Launch

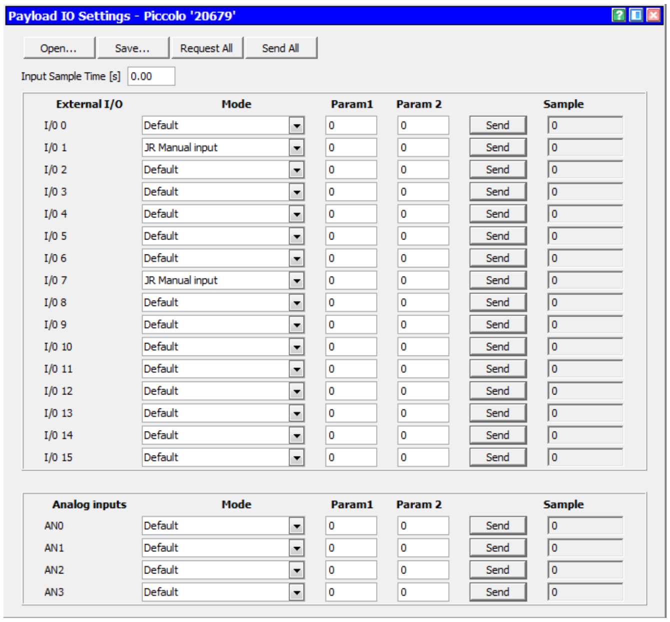
Waypoints
Lost comm waypoint: 1
Autoland waypoint: 95

Assert Flight Termination if:
□ GPS times out
□ Comm times out
□ GPS AND Comm time out
□ Airspace is Violated
□ Altitude MIN is Violated
□ Altitude MAX is Violated

If Flight Termination is Asserted:
□ Close Throttle
□ Drop Deadman Line
□ Aerodynamic Termination
□ Deploy Parachute

If Engine Kill is Commanded:
□ Drop Deadman Line

1.15. Payload IO Settings



Payload IO Settings - Piccolo '20679'

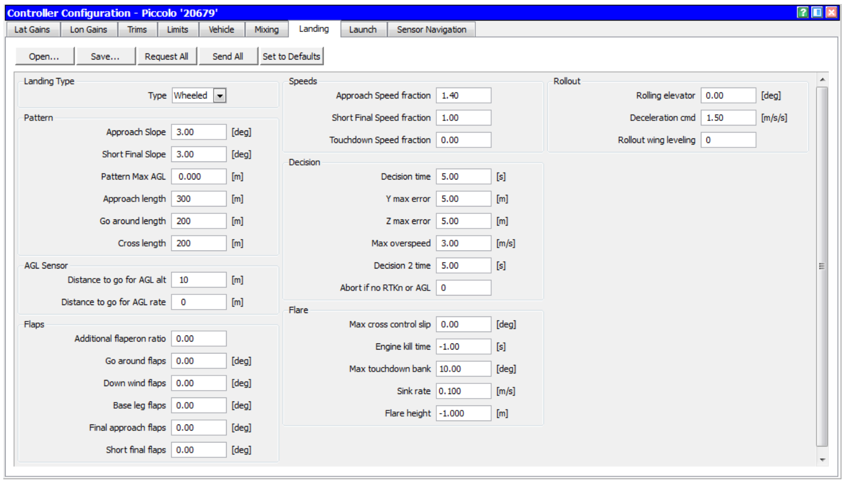
Open... | Save... | Request All | Send All

Input Sample Time [s] 0.00

External I/O	Mode	Param1	Param 2		Sample
I/0 0	Default	0	0	Send	0
I/0 1	JR Manual input	0	0	Send	0
I/0 2	Default	0	0	Send	0
I/0 3	Default	0	0	Send	0
I/0 4	Default	0	0	Send	0
I/0 5	Default	0	0	Send	0
I/0 6	Default	0	0	Send	0
I/0 7	JR Manual input	0	0	Send	0
I/0 8	Default	0	0	Send	0
I/0 9	Default	0	0	Send	0
I/0 10	Default	0	0	Send	0
I/0 11	Default	0	0	Send	0
I/0 12	Default	0	0	Send	0
I/0 13	Default	0	0	Send	0
I/0 14	Default	0	0	Send	0
I/0 15	Default	0	0	Send	0

Analog inputs	Mode	Param1	Param 2		Sample
AN0	Default	0	0	Send	0
AN1	Default	0	0	Send	0
AN2	Default	0	0	Send	0
AN3	Default	0	0	Send	0

1.16. Landing



Landing Type

Type Wheeled

Pattern

Approach Slope 3.00 [deg]

Short Final Slope 3.00 [deg]

Pattern Max AGL 0.000 [m]

Approach length 300 [m]

Go around length 200 [m]

Cross length 200 [m]

AGL Sensor

Distance to go for AGL alt 10 [m]

Distance to go for AGL rate 0 [m]

Flaps

Additional flaperon ratio 0.00

Go around flaps 0.00 [deg]

Down wind flaps 0.00 [deg]

Base leg flaps 0.00 [deg]

Final approach flaps 0.00 [deg]

Short final flaps 0.00 [deg]

Speeds

Approach Speed fraction 1.40

Short Final Speed fraction 1.00

Touchdown Speed fraction 0.00

Decision

Decision time 5.00 [s]

Y max error 5.00 [m]

Z max error 5.00 [m]

Max overspeed 3.00 [m/s]

Decision 2 time 5.00 [s]

Abort if no RTKn or AGL 0

Flare

Max cross control slip 0.00 [deg]

Engine kill time -1.00 [s]

Max touchdown bank 10.00 [deg]

Sink rate 0.100 [m/s]

Flare height -1.000 [m]

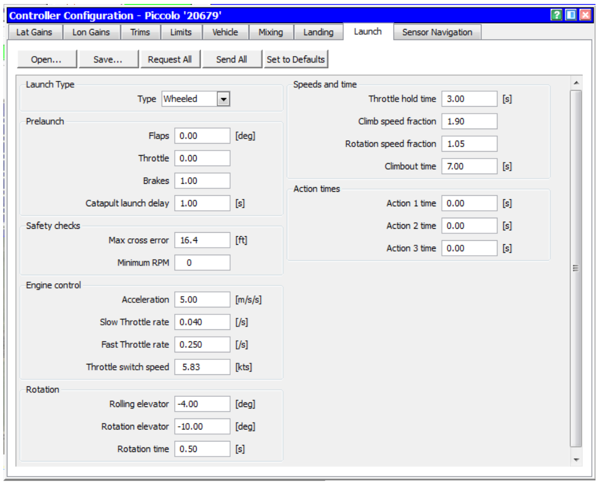
Rollout

Rolling elevator 0.00 [deg]

Deceleration cmd 1.50 [m/s/s]

Rollout wing leveling 0

1.17. Launch



Launch Type

Type Wheeled

Prelaunch

Flaps 0.00 [deg]

Throttle 0.00

Brakes 1.00

Catapult launch delay 1.00 [s]

Safety checks

Max cross error 16.4 [ft]

Minimum RPM 0

Engine control

Acceleration 5.00 [m/s/s]

Slow Throttle rate 0.040 [/s]

Fast Throttle rate 0.250 [/s]

Throttle switch speed 5.83 [kts]

Rotation

Rolling elevator -4.00 [deg]

Rotation elevator -10.00 [deg]

Rotation time 0.50 [s]

Speeds and time

Throttle hold time 3.00 [s]

Climb speed fraction 1.90

Rotation speed fraction 1.05

Climbout time 7.00 [s]

Action times

Action 1 time 0.00 [s]

Action 2 time 0.00 [s]

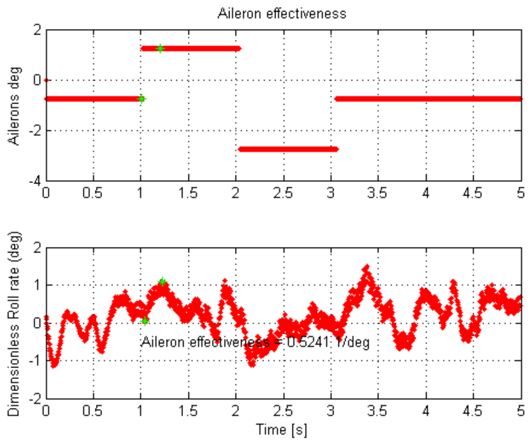
Action 3 time 0.00 [s]

1.18. Doublet Results

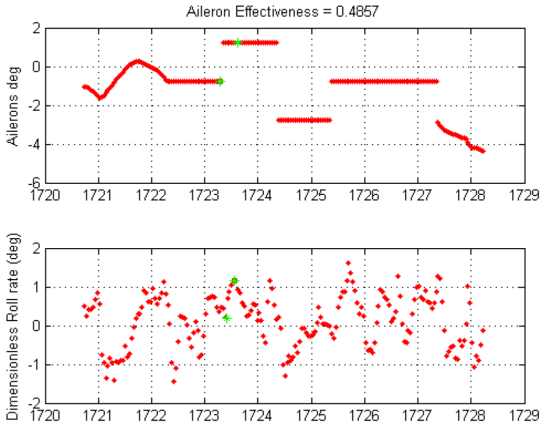
1.18.1. Flight 2

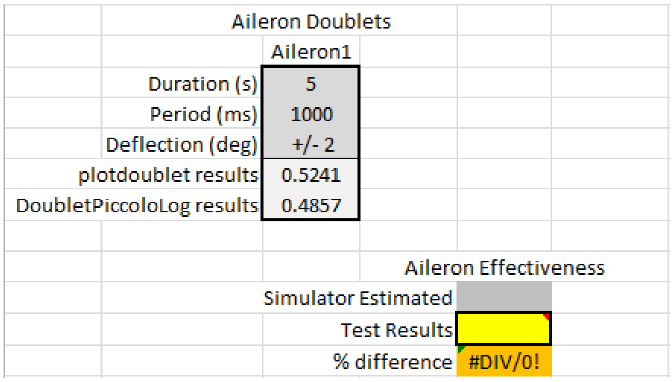
1.18.1.1. Aileron Doublets

1) Doublet File Method



2) Piccolo Log File Method



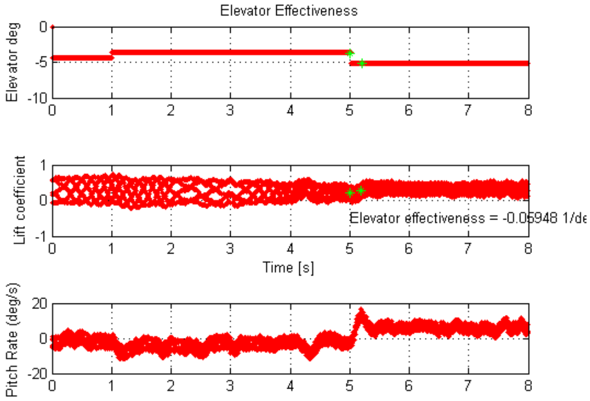


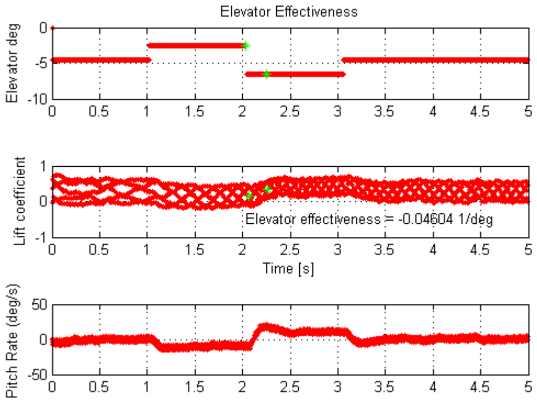
	Aileron Doublets
	Aileron1
Duration (s)	5
Period (ms)	1000
Deflection (deg)	+/- 2
plotdoublet results	0.5241
DoubletPiccoloLog results	0.4857

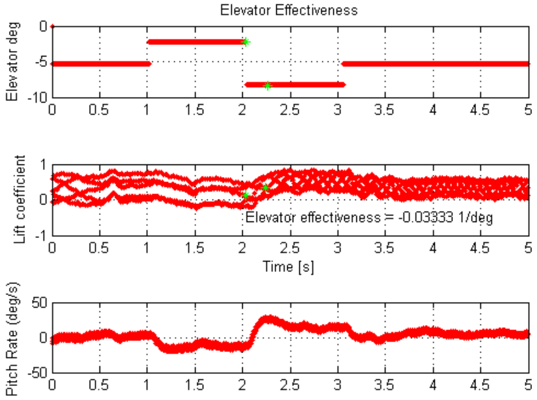
	Aileron Effectiveness
Simulator Estimated	
Test Results	
% difference	#DIV/0!

1.18.1.2. Elevator Doublets

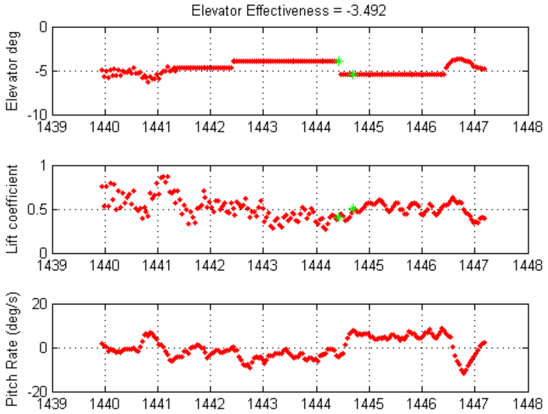
1) Doublet File Method

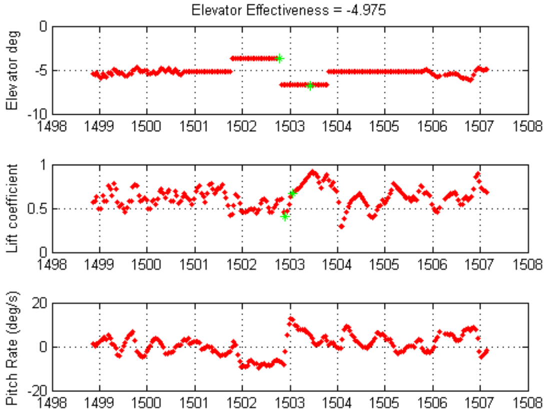


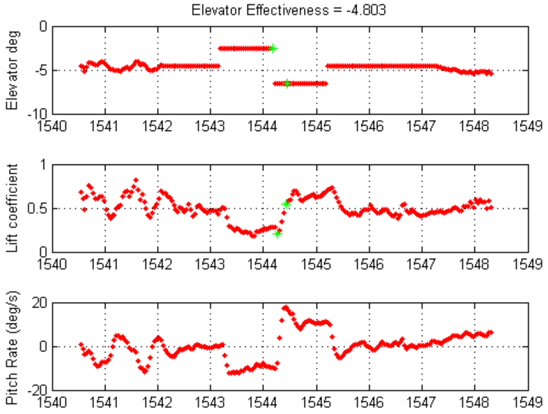


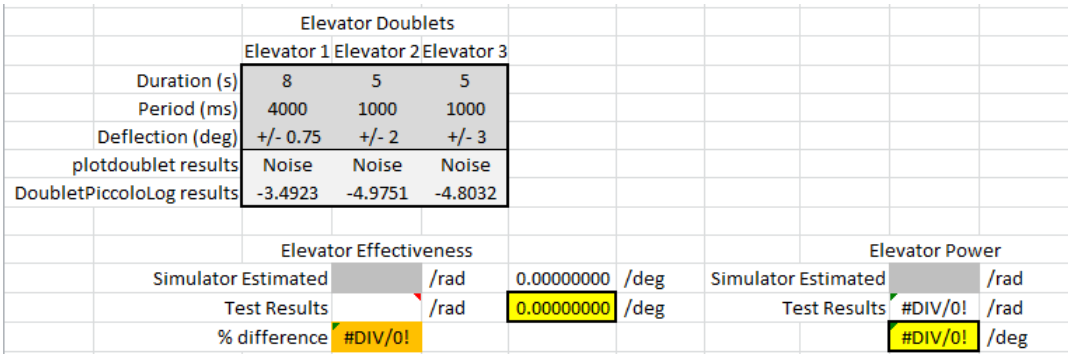


2) Piccolo Log File Method


Elevator Effectiveness = -3.492
Elevator deg
0
-5
-10
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
20
0
-20


Elevator Effectiveness = -4.975
Elevator deg
0
-5
-10
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
20
0
-20


Elevator Effectiveness = -4.803
Elevator deg
0
-5
-10
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
20
0
-20



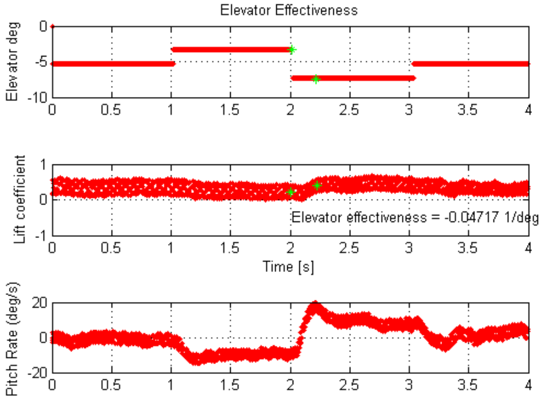
	Elevator Doublets		
	Elevator 1	Elevator 2	Elevator 3
Duration (s)	8	5	5
Period (ms)	4000	1000	1000
Deflection (deg)	+/- 0.75	+/- 2	+/- 3
plotdoublet results	Noise	Noise	Noise
DoubletPiccoloLog results	-3.4923	-4.9751	-4.8032

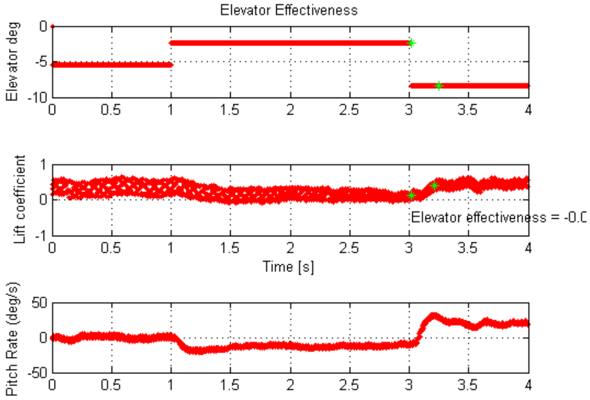
	Elevator Effectiveness					Elevator Power	
Simulator Estimated		/rad	0.00000000	/deg	Simulator Estimated		/rad
Test Results		/rad	0.00000000	/deg	Test Results	#DIV/0!	/rad
% difference	#DIV/0!					#DIV/0!	/deg

1.18.2. Flight 3

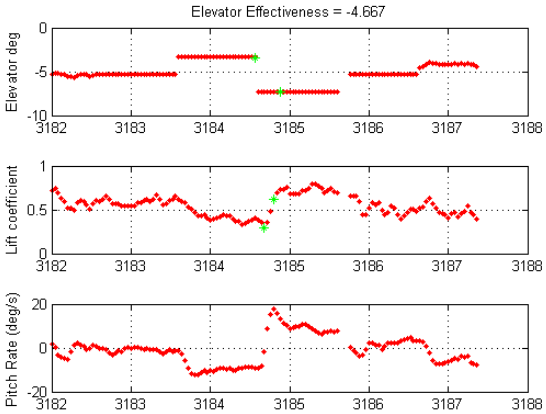
1.18.2.1. Elevator Doublets

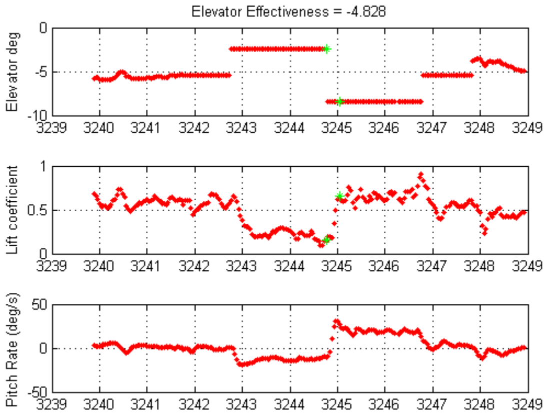
1) Doublet File Method

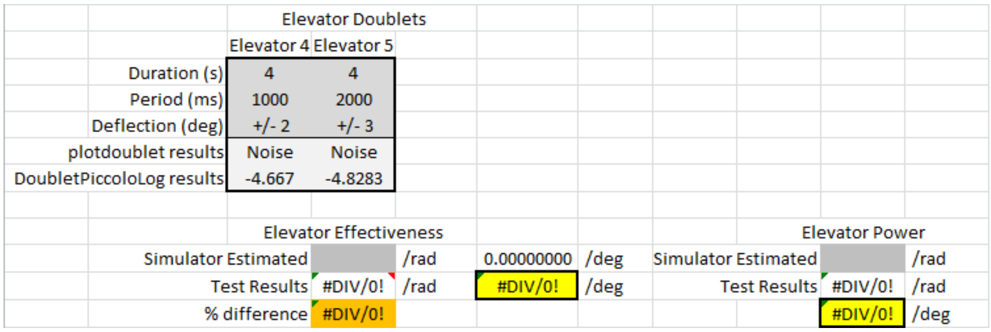




2) Piccolo Log File Method





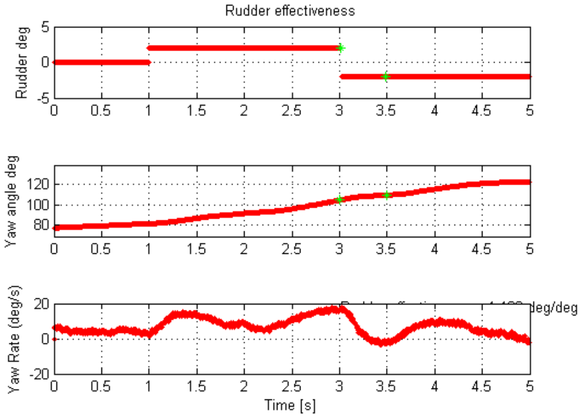


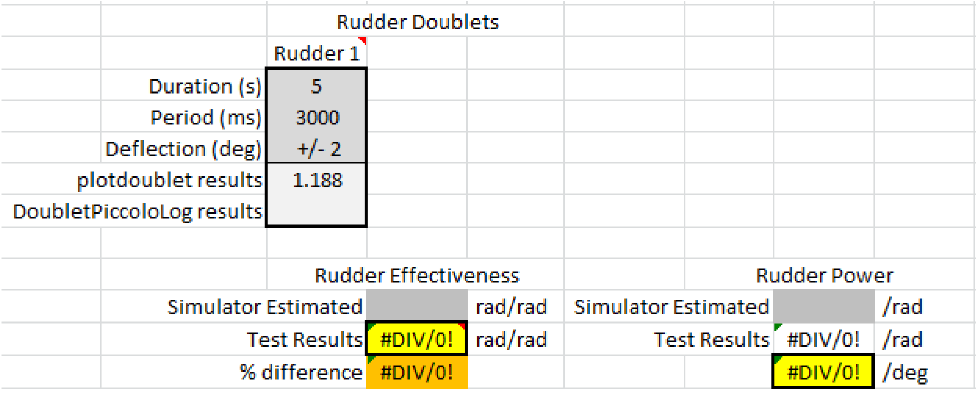
	Elevator Doublets	
	Elevator 4	Elevator 5
Duration (s)	4	4
Period (ms)	1000	2000
Deflection (deg)	+/- 2	+/- 3
plotdoublet results	Noise	Noise
DoubletPiccoloLog results	-4.667	-4.8283

	Elevator Effectiveness					Elevator Power	
Simulator Estimated		/rad	0.00000000	/deg	Simulator Estimated		/rad
Test Results	#DIV/0!	/rad	#DIV/0!	/deg	Test Results	#DIV/0!	/rad
% difference	#DIV/0!					#DIV/0!	/deg

1.18.2.2. Rudder Doublets

1) Piccolo Log File Method



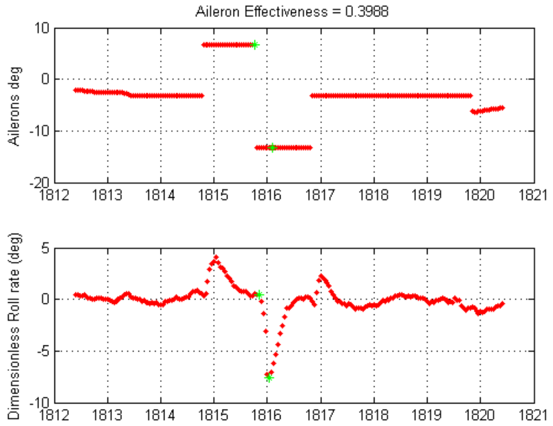


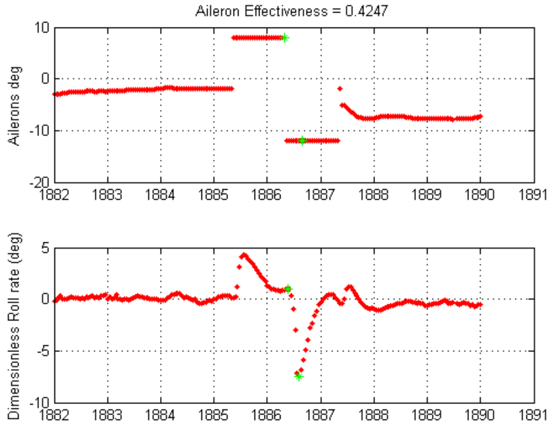
	Rudder Doublets				
	Rudder 1				
Duration (s)	5				
Period (ms)	3000				
Deflection (deg)	+/- 2				
plotdoublet results	1.188				
DoubletPiccoloLog results					
	Rudder Effectiveness			Rudder Power	
Simulator Estimated		rad/rad	Simulator Estimated		/rad
Test Results	#DIV/0!	rad/rad	Test Results	#DIV/0!	/rad
% difference	#DIV/0!			#DIV/0!	/deg

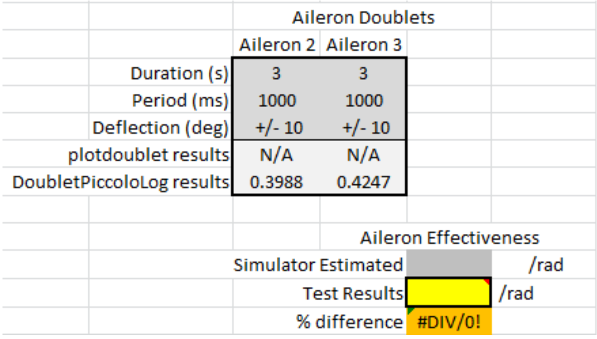
1.18.3. Flight 4

1.18.3.1. Aileron Doublets

1) Piccolo Log File Method



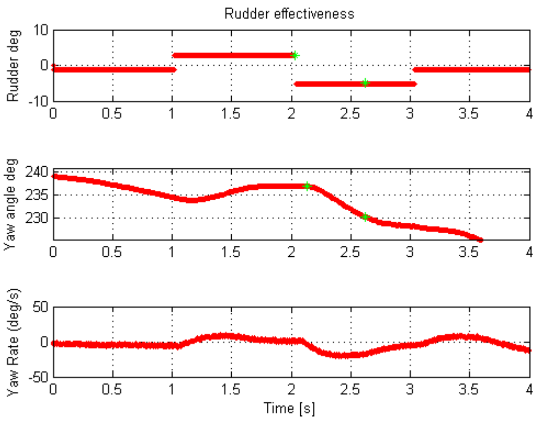




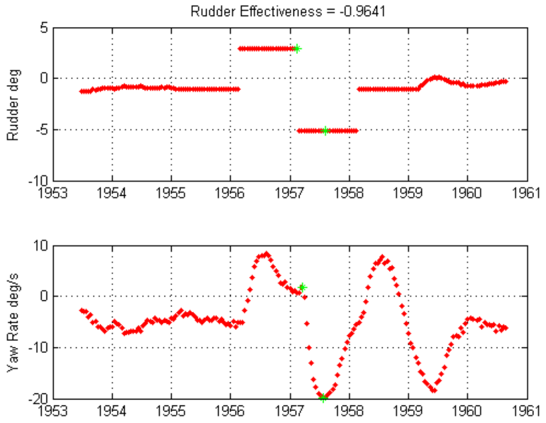
	Aileron Doublets			
	Aileron 2	Aileron 3		
Duration (s)	3	3		
Period (ms)	1000	1000		
Deflection (deg)	+/- 10	+/- 10		
plotdoublet results	N/A	N/A		
DoubletPiccoloLog results	0.3988	0.4247		
			Aileron Effectiveness	
		Simulator Estimated		/rad
		Test Results		/rad
		% difference	#DIV/0!	

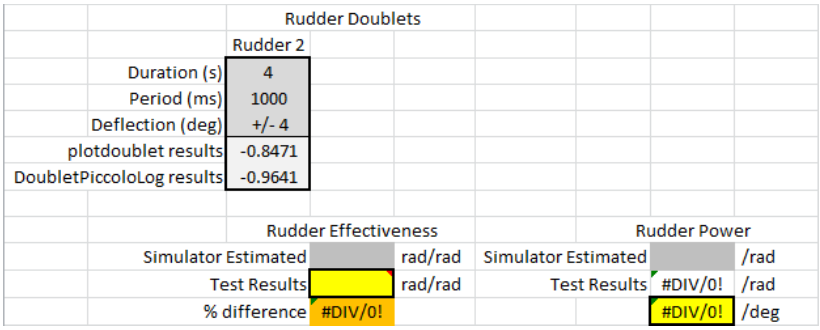
1.18.3.2. Rudder Doublets

1) Doublet File Method



2) Piccolo Log File Method



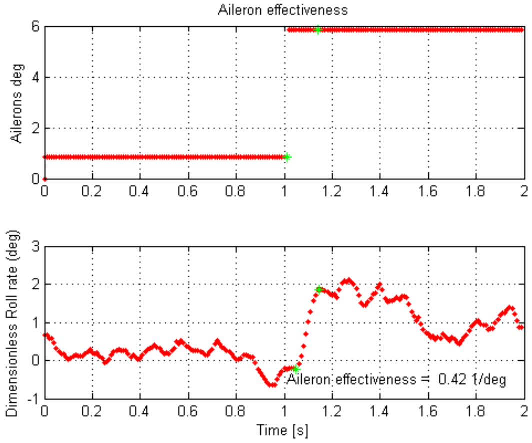


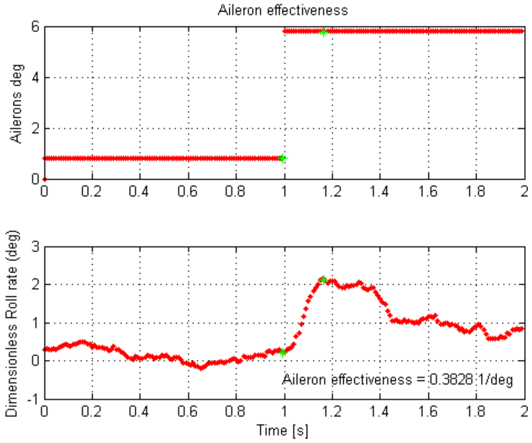
		Rudder Doublets						
		Rudder 2						
	Duration (s)	4						
	Period (ms)	1000						
	Deflection (deg)	+/- 4						
	plotdoublet results	-0.8471						
	DoubletPiccoloLog results	-0.9641						
		Rudder Effectiveness					Rudder Power	
		Simulator Estimated		rad/rad		Simulator Estimated		/rad
		Test Results		rad/rad		Test Results	#DIV/0!	/rad
		% difference	#DIV/0!				#DIV/0!	/deg

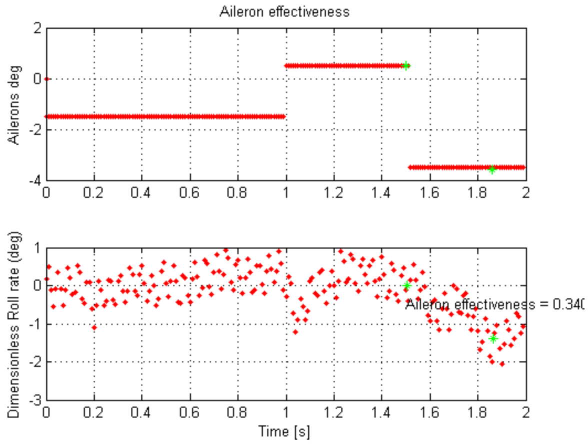
1.18.4. Flight 29

1.18.4.1. Aileron Doublets

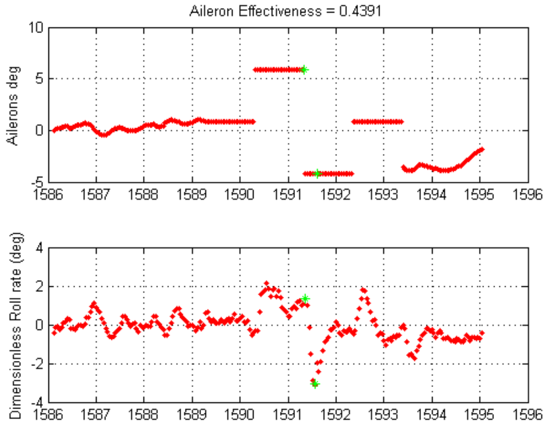
1) Doublet File Method

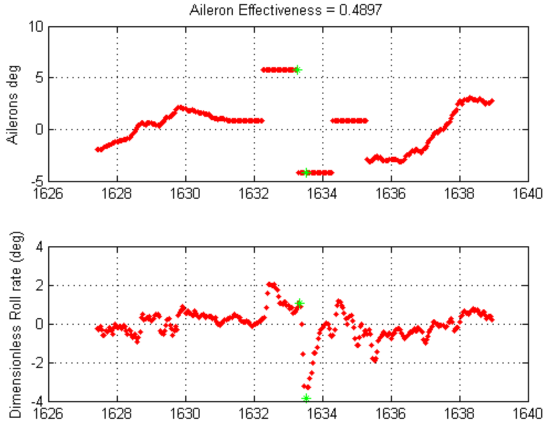


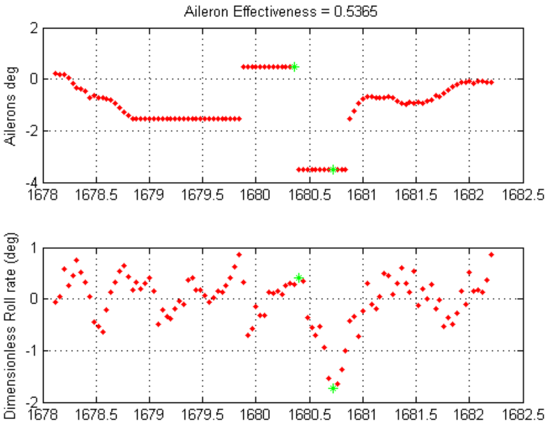


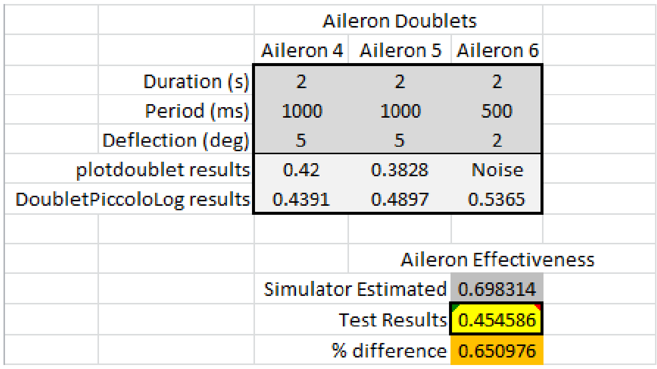


2) Piccolo Log File Method







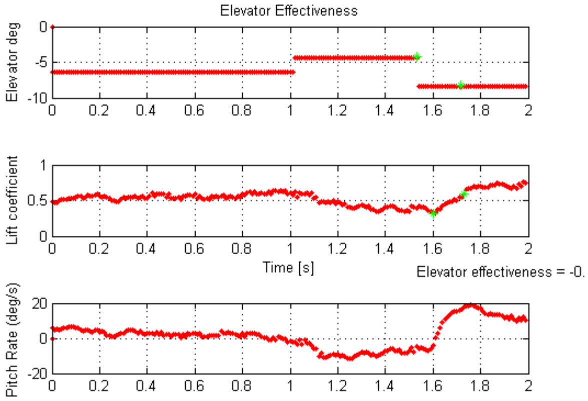


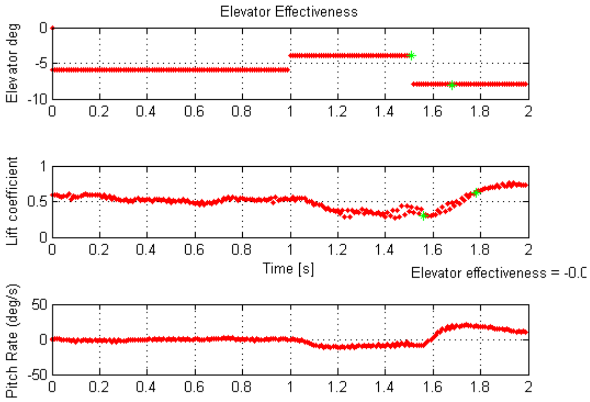
	Aileron Doublets		
	Aileron 4	Aileron 5	Aileron 6
Duration (s)	2	2	2
Period (ms)	1000	1000	500
Deflection (deg)	5	5	2
plotdoublet results	0.42	0.3828	Noise
DoubletPiccoloLog results	0.4391	0.4897	0.5365

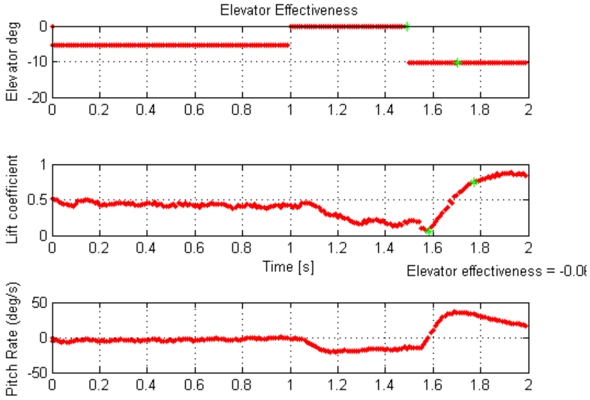
	Aileron Effectiveness
Simulator Estimated	0.698314
Test Results	0.454586
% difference	0.650976

1.18.4.2. Elevator Doublets

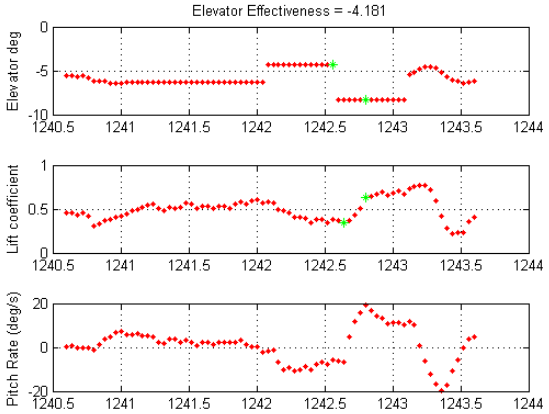
1) Doublet File Method

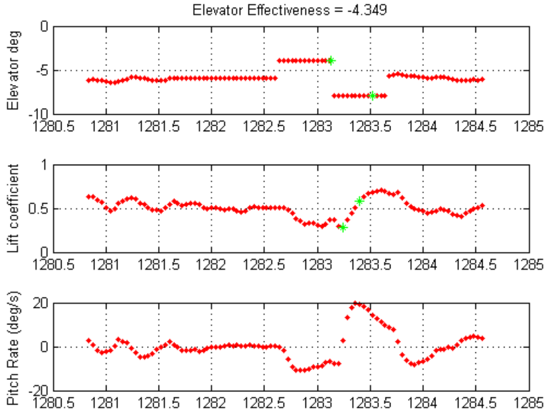


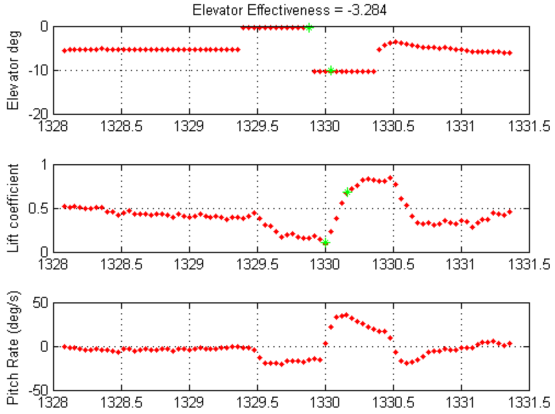


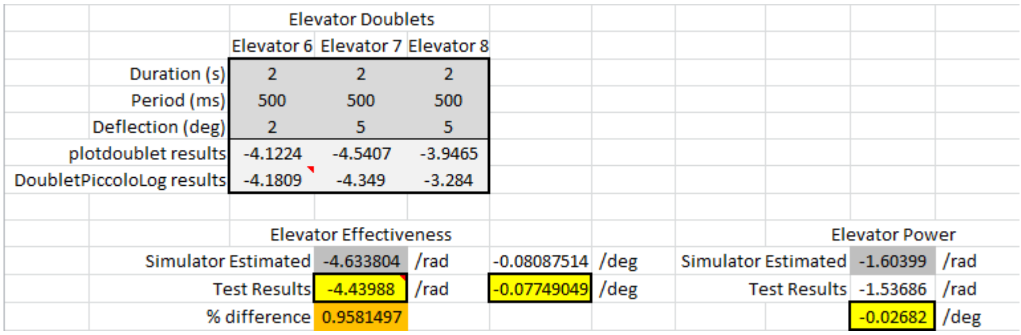


2) Piccolo Log File Method


Elevator Effectiveness = -4.181
Elevator deg
0
-5
-10
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
20
0
-20
1240.5
1241
1241.5
1242
1242.5
1243
1243.5
1244


Elevator Effectiveness = -4.349
Elevator deg
0
-5
-10
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
20
0
-20
1280.5
1281
1281.5
1282
1282.5
1283
1283.5
1284
1284.5
1285

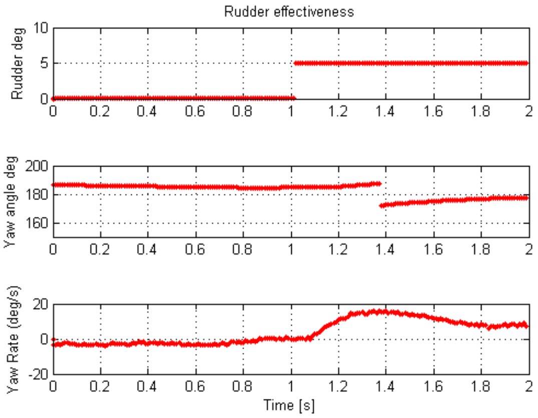

Elevator Effectiveness = -3.284
Elevator deg
0
-10
-20
Lift coefficient
1
0.5
0
Pitch Rate (deg/s)
50
0
-50
1328
1328.5
1329
1329.5
1330
1330.5
1331
1331.5

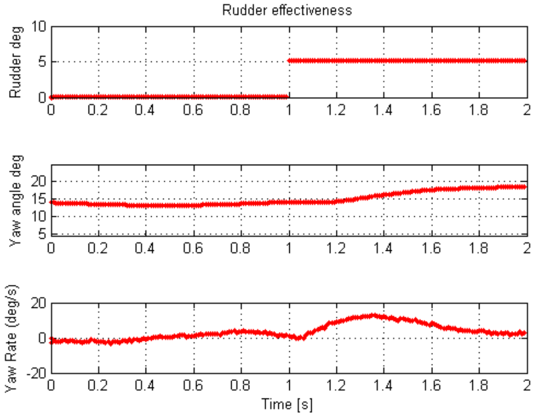


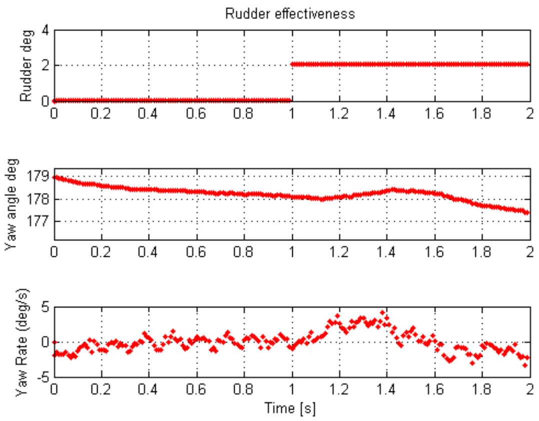
		Elevator Doublets						
	Elevator 6	Elevator 7	Elevator 8					
Duration (s)	2	2	2					
Period (ms)	500	500	500					
Deflection (deg)	2	5	5					
plotdoublet results	-4.1224	-4.5407	-3.9465					
DoubletPiccoloLog results	-4.1809	-4.349	-3.284					
		Elevator Effectiveness					Elevator Power	
	Simulator Estimated	-4.633804	/rad	-0.08087514	/deg	Simulator Estimated	-1.60399	/rad
	Test Results	-4.43988	/rad	-0.07749049	/deg	Test Results	-1.53686	/rad
	% difference	0.9581497					-0.02682	/deg

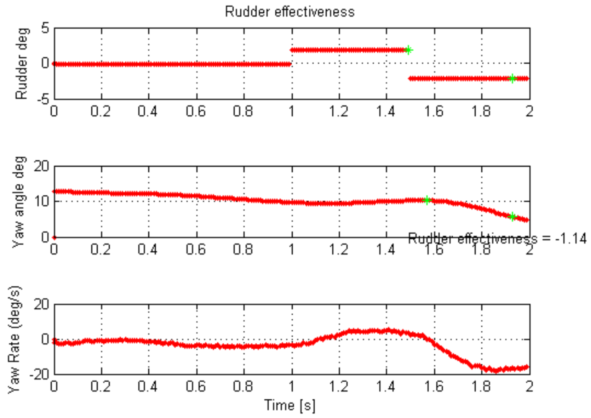
1.18.4.3. Rudder Doublets

1) Doublet File Method

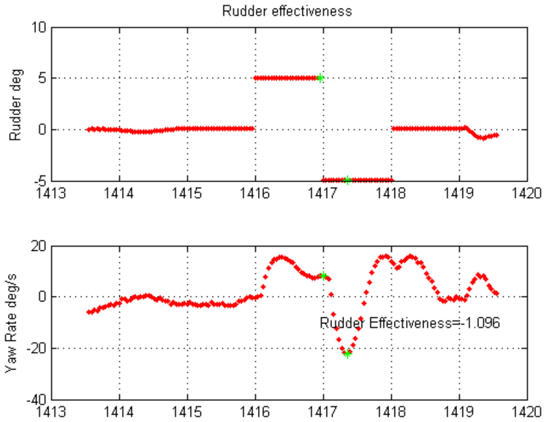


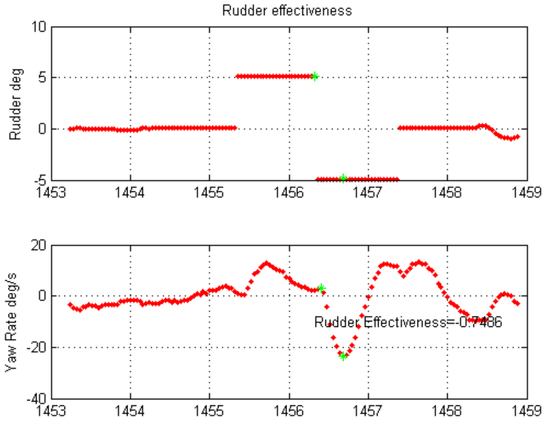


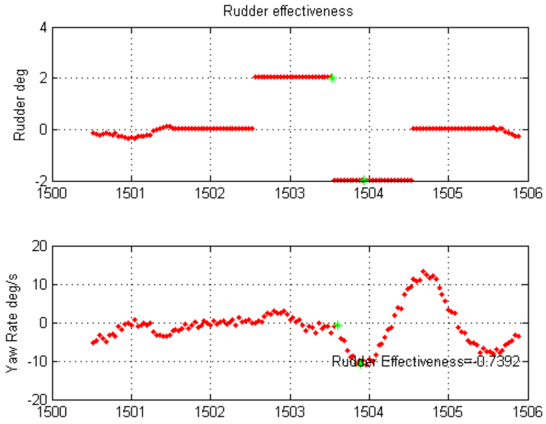


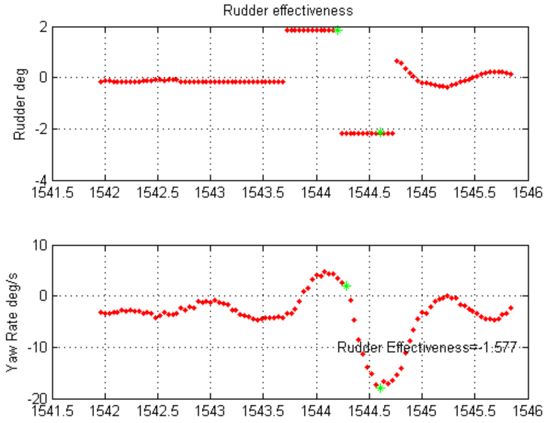


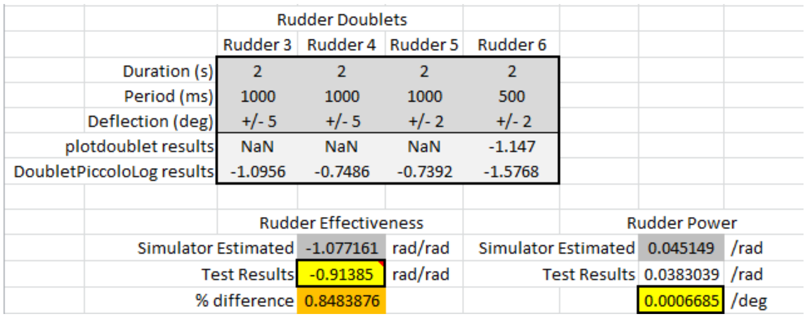
2) Piccolo Log File Method


Rudder effectiveness
Rudder deg
10
5
0
-5
1413
1414
1415
1416
1417
1418
1419
1420
Yaw Rate deg/s
20
0
-20
-40
Rudder Effectiveness=-1.096


Rudder effectiveness
Rudder deg
10
5
0
-5
1453
1454
1455
1456
1457
1458
1459
Yaw Rate deg/s
20
0
-20
-40
Rudder Effectiveness=-0.7486







	Rudder Doublets						
	Rudder 3	Rudder 4	Rudder 5	Rudder 6			
Duration (s)	2	2	2	2			
Period (ms)	1000	1000	1000	500			
Deflection (deg)	+/- 5	+/- 5	+/- 2	+/- 2			
plotdoublet results	NaN	NaN	NaN	-1.147			
DoubletPiccoloLog results	-1.0956	-0.7486	-0.7392	-1.5768			
	Rudder Effectiveness				Rudder Power		
	Simulator Estimated	-1.077161	rad/rad		Simulator Estimated	0.045149	/rad
	Test Results	-0.91385	rad/rad		Test Results	0.0383039	/rad
	% difference	0.8483876				0.0006685	/deg

1.19. Flight Summary

1.19.1. Doublet Maneuver Results

At the time that the Nexstar was flown and doublet maneuvers were performed guidelines for how to conduct doublet maneuvers and analyze their results had not yet been developed. A lot of the doublet maneuvers were done just to test the process of conducting doublet maneuvers. As a result the rudder and elevator effectiveness terms were not ever changed according to the results of the doublet maneuvers. The majority of the meaningful elevator and rudder doublet maneuvers were not completed until the final flight of the Nexstar. Additionally the first couple of rudder and elevator doublet results were near the simulator estimated values; therefore, they were reasonably assumed to be accurate at the time. The only vehicle parameter that was actually changed as a result of the doublet maneuvers was the aileron effectiveness.

The aileron effectiveness was estimated by the simulator to be 0.698314. The first aileron doublet, performed in Flight 2, was analyzed after Flight 3. The result that the first analysis of Aileron1 produced was 0.4315. The aileron effectiveness parameter was changed to 0.4315 for Flights 4 all the way through Flight 29. Additional aileron doublets were performed in Flights 4 and 29. The final analysis of the aileron doublet maneuvers produced a value of 0.454586. The final value came from averaging the results of all of the recorded aileron effectiveness results and dropping the highest and lowest values.

The final analysis of all of the elevator doublet maneuvers resulted in an elevator effectiveness of -4.43988 /rad. The simulator estimated value was -4.633491. Since the resulting elevator effectiveness was similar to the simulator estimated value and the simulator estimated value had been used in 29 flights it was decided to keep the final elevator effectiveness value set at -4.633491.

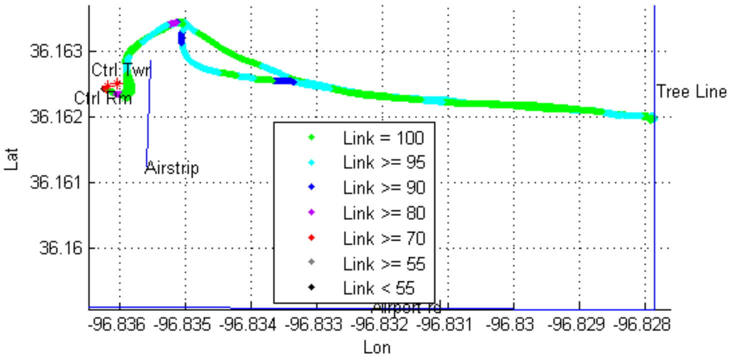
The final analysis of all of the rudder doublet maneuvers resulted in a rudder effectiveness of -0.91385. Rudder1 test results were excluded from the final rudder effectiveness calculation because the yaw angle steadily increased throughout the entire test, even though the change in rudder deflection was positive and then negative. The Doublet File Method results of Rudder3 – Rudder5 could not be used because the period was not long enough for the doublet file log to capture the entire doublet test. The simulator estimated value of rudder effectiveness was -1.077161. Since the rudder effectiveness results from the doublet maneuver tests were similar to the simulator estimated value and the simulator estimated value had been used in 29 flights it was decided to keep the final rudder effectiveness value set at -1.077161.

1.19.2. Software versions

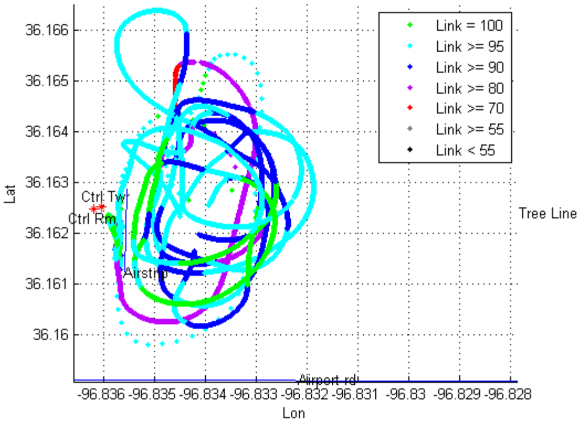
Flights 1 – 15 flew with version 2.1.1.e of the fixed wing 2 piccolo firmware. Flights 16 – 26 flew version 2.1.4.f. Flights 27 – 29 flew version 2.1.4g. All of the final gains and settings were set for version 2.1.4.g.

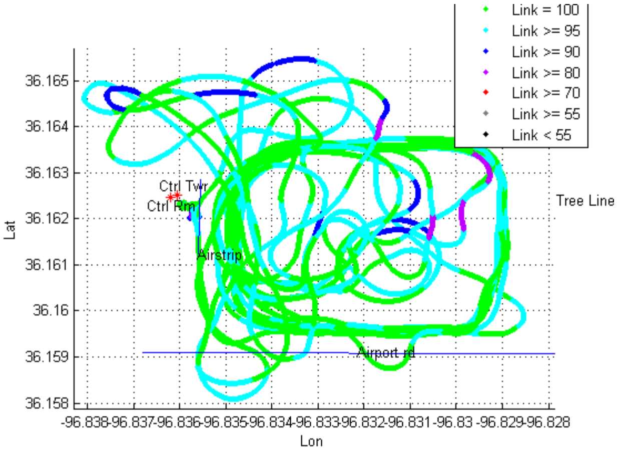
1.19.3. Piccolo Communications Issues

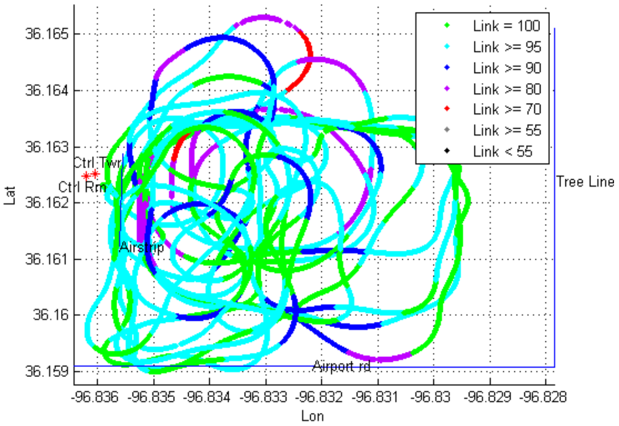
The Nexstar was flown with 3 different antennas on the control tower throughout all of the 29 flights. The first antenna was used for flights 1-7.

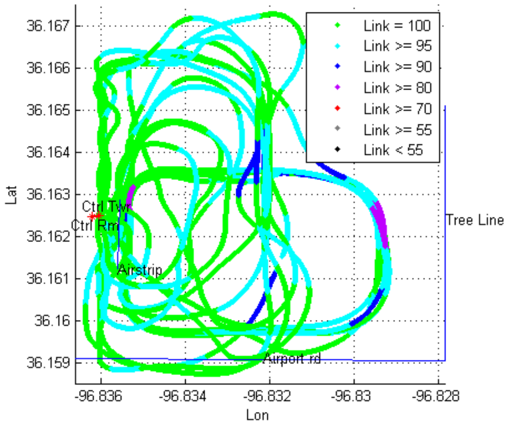


The figure above depicts the ground communications check that was conducted before the first flight. The ground communications check indicated that the flights could expect minimal packet loss.

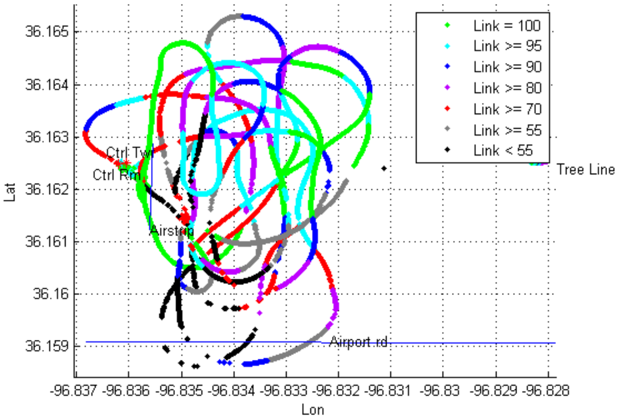

Link = 100
Link >= 95
Link >= 90
Link >= 80
Link >= 70
Link >= 55
Link < 55
36.166
36.165
36.164
36.163
36.162
36.161
36.16
Lat
Ctrl Twr
Ctrl Rm
Airstrip
Tree Line
Airport rd
-96.836 -96.835 -96.834 -96.833 -96.832 -96.831 -96.83 -96.829 -96.828
Lon

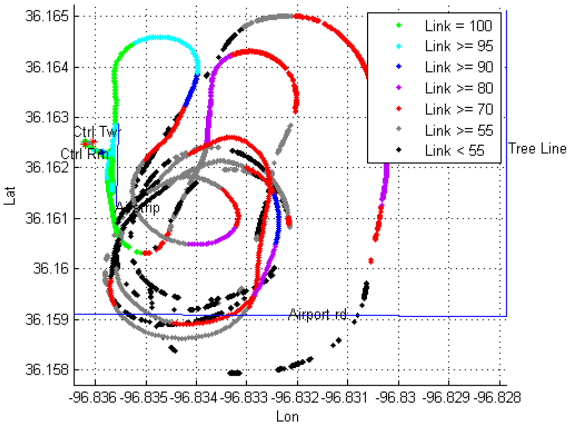

Link = 100
Link >= 95
Link >= 90
Link >= 80
Link >= 70
Link >= 55
Link < 55
36.165
36.164
36.163
36.162
36.161
36.16
36.159
36.158
Lat
Ctrl Twr
Ctrl Rm
Airstrip
Tree Line
Airport rd
-96.838-96.837-96.836-96.835-96.834-96.833-96.832-96.831-96.83 -96.829-96.828
Lon



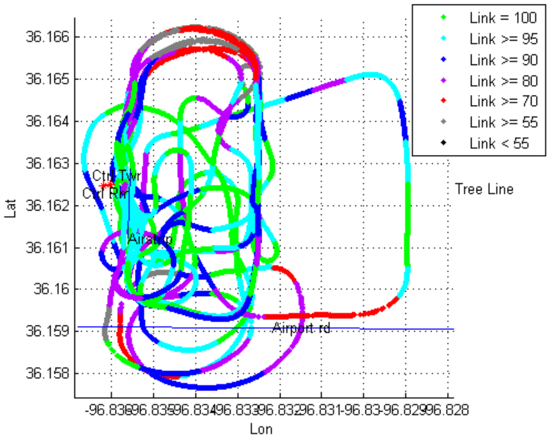


The figures above depict the Link plots of the flights 1 – 4. There was significant packet loss in all four flights. Flight 3 had instances of Link in the 70s. Some flights had more instances of packet loss than others; however, there were a lot of long periods of Link in the 80s throughout all of the flights. It was decided that in light of the fact that the piccolo communications should be good for miles there was something wrong with the antenna. It was concluded that there should be minimal packet loss at best, due to temporary loss of line of sight amongst other possible small factors. As a result a new antenna was ordered to replace the first antenna on the control tower.





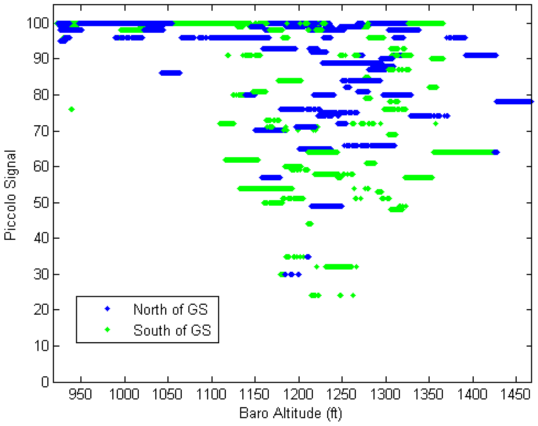
The second control tower antenna was first used for Flights 8, 9, and 10 which occurred on the same flight day. The figures above depict the Link plots of the flights. There was a large amount of packet loss. The piccolo signal was worse than it was prior to the new antenna. In Flight 8 Link fell below 55 and there were periods of complete piccolo comms loss. Flight 9 Link was consistently in the 60s and lower. Flight 9 also had numerous occasions of complete piccolo comms loss.

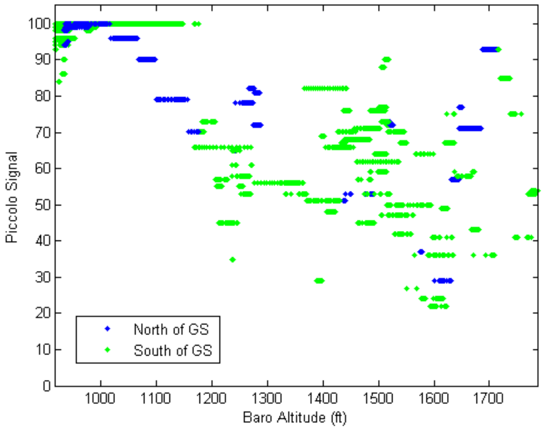


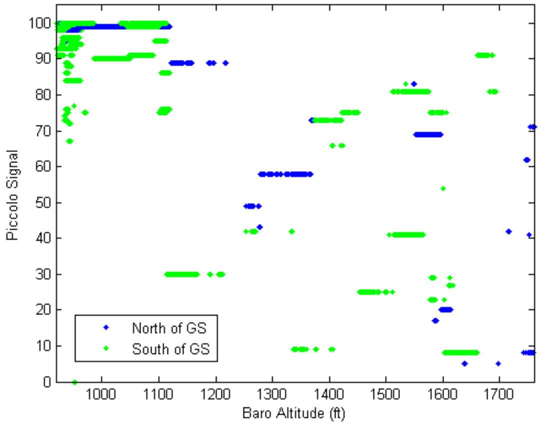
The comms loss was bad enough that Flight 10 was flown with the ground station antenna, outside of the control room and control tower. The ground station antenna did not do much better as it was on the ground and there were probably numerous occasions of loss of line of sight.

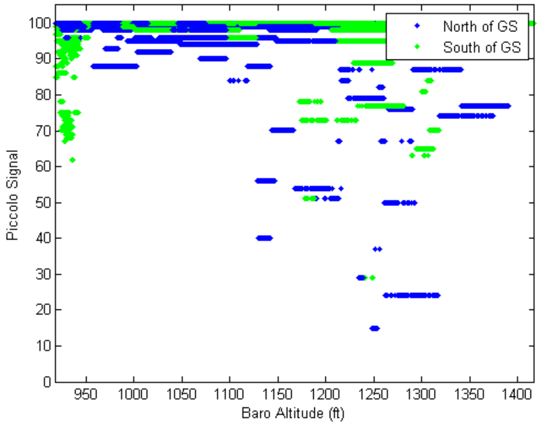
Flights 11 and 12 were flown using the ground station antenna while the control tower antenna was being modified to attempt to improve piccolo signal strength. The signal strength with the ground station antenna was not optimal either; however, since the Nexstar was configured to use the JR switch board the manual pilot was able to fly Nexstar outside of the piccolo communications without having to worry about the piccolo signal strength.

Flights 14 and 15 were flown with the control tower antenna after it had been modified. Flights 13 and 14 were on the same flight day; however, Flight 13 was flown with the ground station antenna. In Flight 15 the piccolo antenna was moved from the fuselage to the starboard wing to see if the piccolo signal strength would be improved by moving the antenna on the Nexstar. It did not help; the packet loss remained the same.

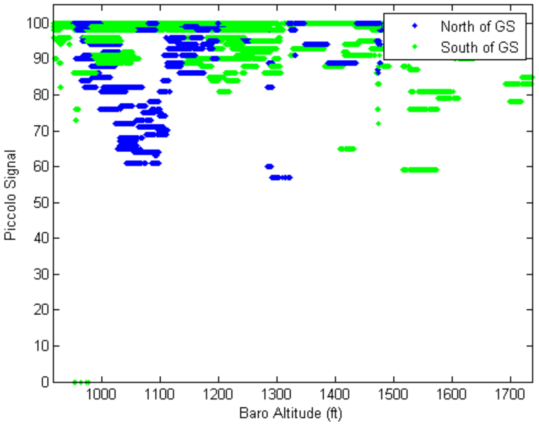

Piccolo Signal
Baro Altitude (ft)
100
90
80
70
60
50
40
30
20
10
0
950
1000
1050
1100
1150
1200
1250
1300
1350
1400
1450
North of GS
South of GS


Piccolo Signal
Baro Altitude (ft)
100
90
80
70
60
50
40
30
20
10
0
1000
1100
1200
1300
1400
1500
1600
1700
North of GS
South of GS

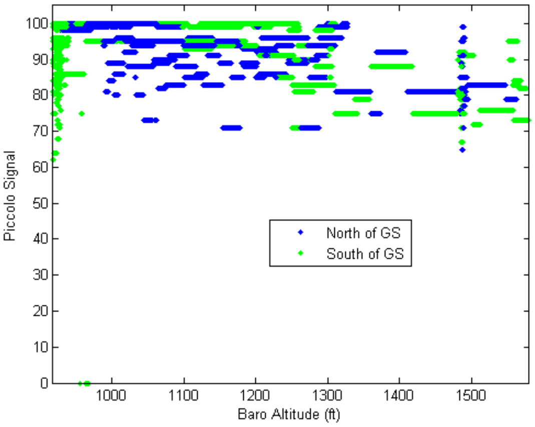




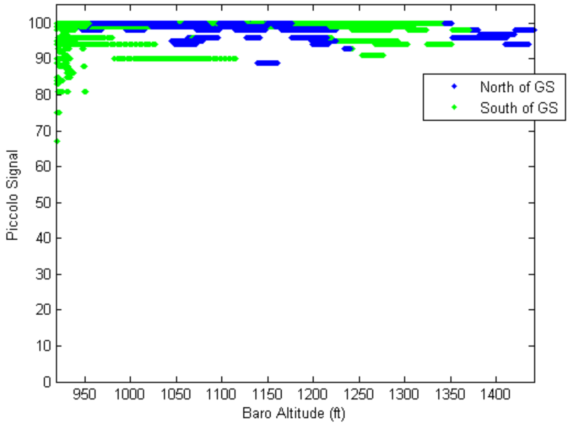
After Flight 15 if was decided to analyze the packet loss versus altitude. The figures above depict the piccolo signal packet loss of Flights 8,9, 14, and 15. There was a correlation between altitude and packet loss. At 1100 feet altitude and above the packet losses increased significantly.

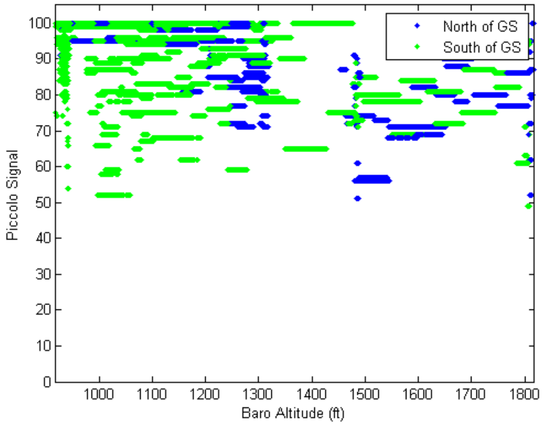


North of GS
South of GS
Piccolo Signal
Baro Altitude (ft)
0
10
20
30
40
50
60
70
80
90
100
1000
1100
1200
1300
1400
1500
1600
1700

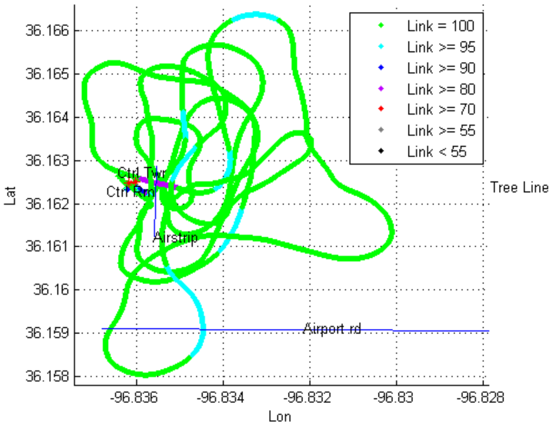


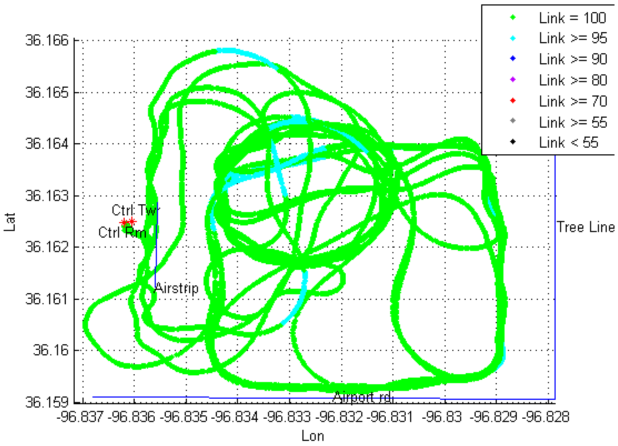
North of GS
South of GS
Piccolo Signal
Baro Altitude (ft)
0
10
20
30
40
50
60
70
80
90
100
1000
1100
1200
1300
1400
1500

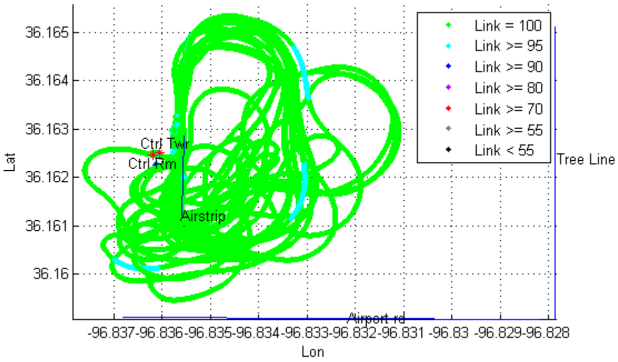




For comparison the packet loss versus altitude was plotted for the flights that used the ground station antenna, Flights 10, 11, 12, and 13. The packet losses were the same across the entire range of altitudes for each of the four flights.







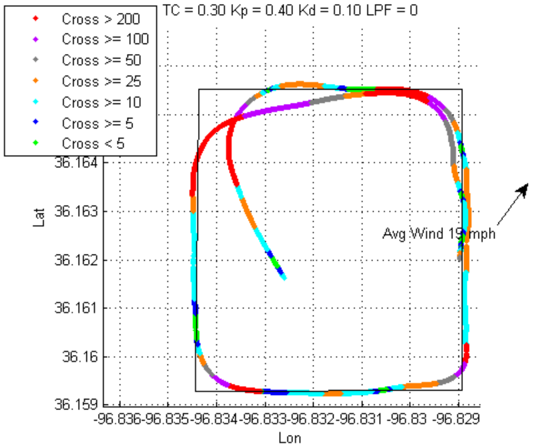
It was decided to order a new antenna for the control tower. The new antenna was first used in Flight 16. The figures above depict the piccolo packet losses in Flights 16, 17, and 18. The packet losses in the piccolo communications link were greatly reduced with the new antenna.

There were only short instances of packet loss and the packet losses were less than 5% (Link > 95%). There was one instance of Link in the 80s in Flight 16; however, that instance can be attributed to loss of line of sight. As the Nexstar flew over the control tower the aircraft itself was blocking the line of sight between the antenna on the aircraft and the antenna on the control tower.

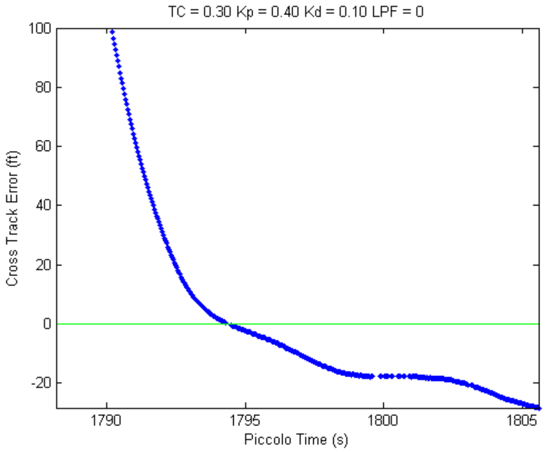
1.19.4. Lateral Control Gain Tuning

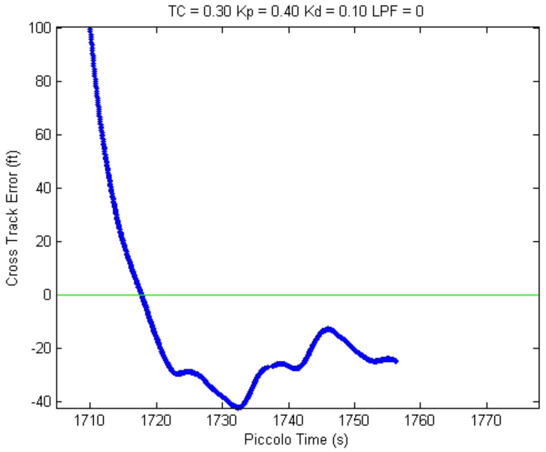
The default lateral control gains did not work well with the Nexstar. The only lateral control gains that were attempted to tune were the Track Control gains. At the time lateral control was not fully understood. If it were to be done again Roll Control would be checked and tuned first before Track Control. This would be done by disabling the Heading command loop and commanding Bank angles or commanding periods of constant Heading and analyzing the autopilot's response.

The track control gains were randomly altered for the first 10 flights. Auto lands were attempted even before the track control gains were tuned.

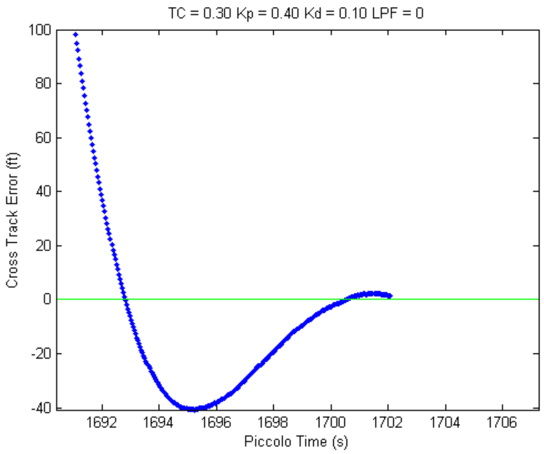


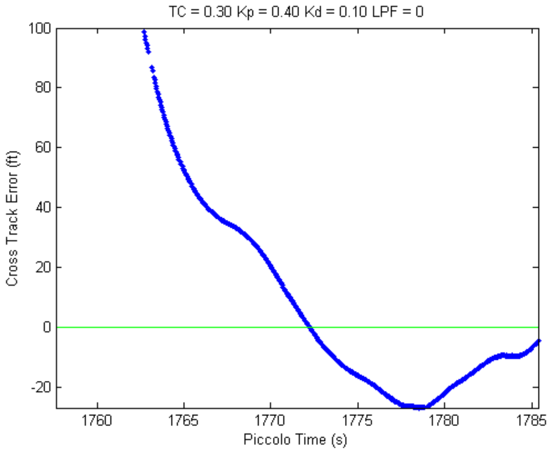
The figure above depicts the box flight plan that the Nexstar flew in Flight 11. The box pattern was flown with default track control gains. The autopilot was unable to converge onto any of the flight paths in the box pattern.



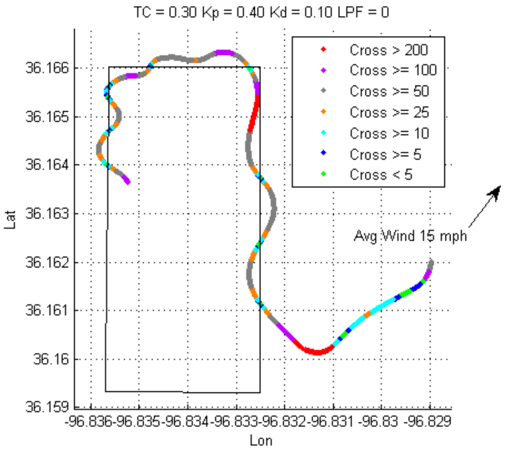


The two figures above depict the cross track error on the west and east legs of the flight plan. In both cases the autopilot was unable to converge onto the track path. On the east leg the autopilot attempted to converge a couple times but “bounced” out. On the west leg the autopilot never even directed the aircraft towards the flight path after it passed through it.

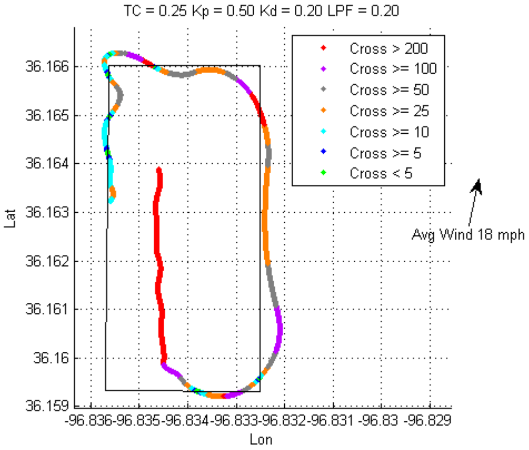




On the north and south legs the autopilot was nearly able to converge to the track path, as depicted by the figures above. The north and south legs just weren't long enough for the autopilot to complete the convergence.

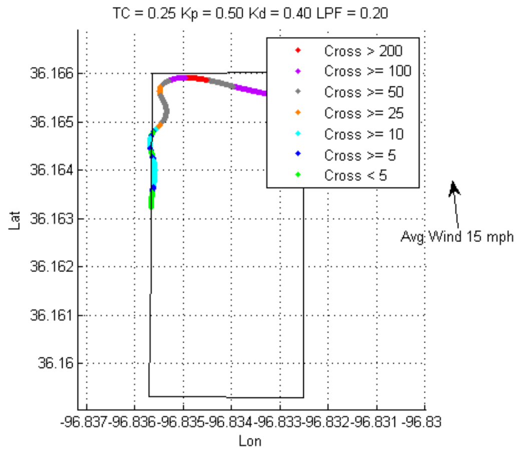


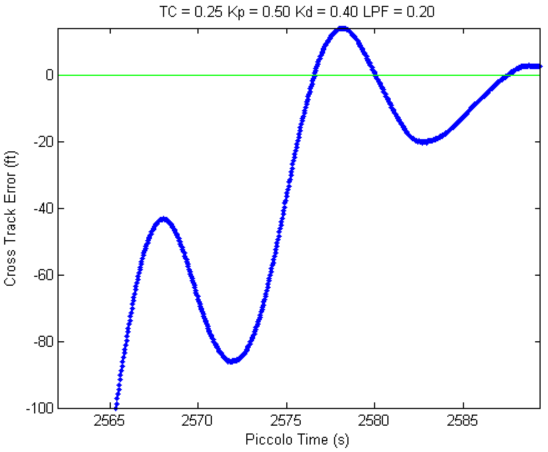
The inability to control lateral tracking was much more apparent on the attempted auto lands. The aircraft oscillated inside and outside the flight paths. After the first land attempt the tracker convergence was lowered from 0.30 to 0.25, the heading error to turn rate was increased from 0.40 to 0.50, the heading error derivative to turn rate was increased from 0.10 to 0.20, and the turn error lpf cutoff was set from 0 to 0.20 Hz.



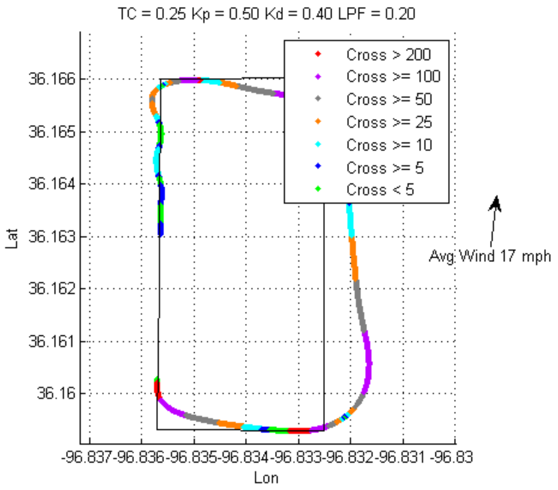
The figure above depicts the second land attempt with the new track control gains. The track oscillations decreased; however, the aircraft was still unable to converge onto the track segment.

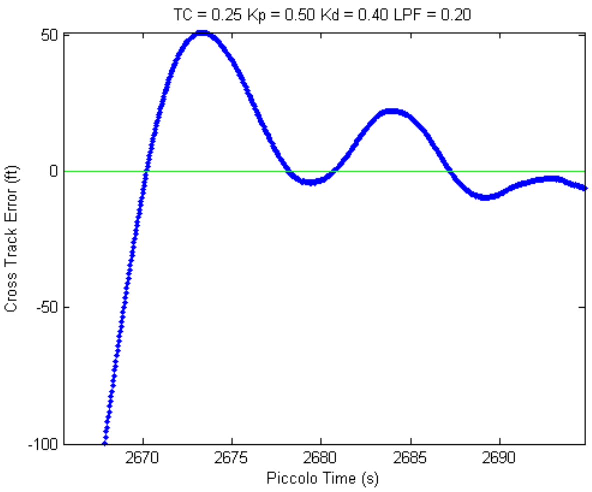
In Flight 12 the heading error derivative to turn rate was changed, it was increased from 0.20 to 0.30.





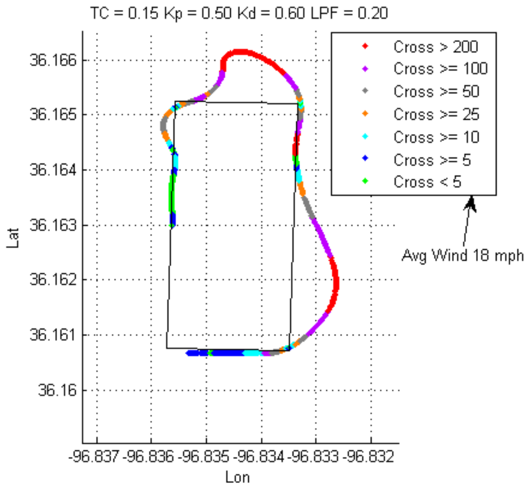
After the first land attempt the heading error derivative to turn rate was increased from 0.30 to 0.40. The figure above depicts the lateral tracking of the second land attempt. The aircraft nearly converged as it crossed the touchdown waypoint. It settled to a cross track error of less than 5 feet.

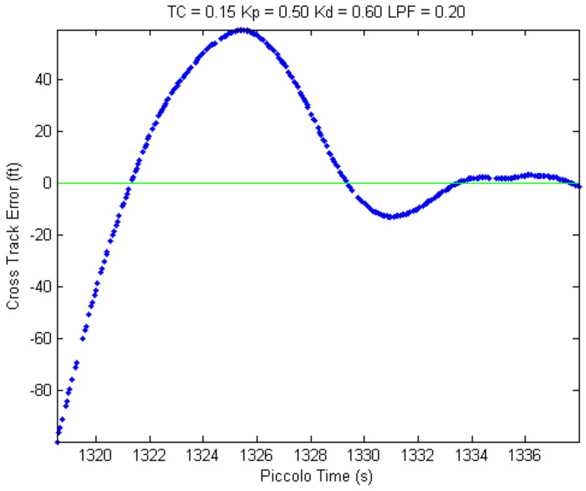




The third land attempt used the same gains as the second. There was less of a pre turn going into the approach. The aircraft oscillated to less than a 5 foot track error and then began to slowly diverge again.

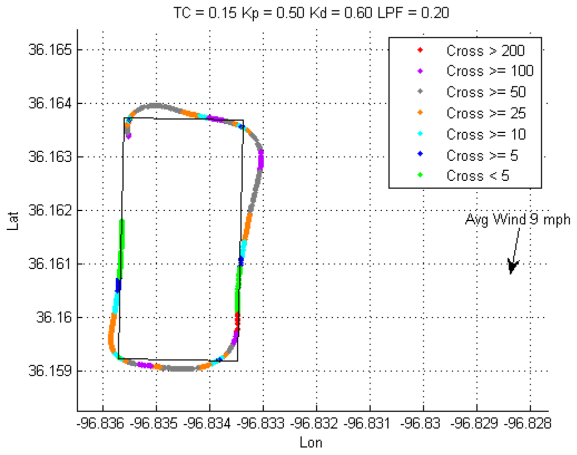
The track control gains weren't changed again until Flight 20. Prior to the flight track control gains had been altered in hardware in the loop simulations to attempt to create the control issues that were observed in real flight. It was noticed that the simulated flights closely resembled the real flights when the tracker convergence parameter was too large. Additionally it was found that raising the heading error derivative to turn rate gain could help offset the tracker convergence error. As a result the tracker convergence was lowered from 0.25 to 0.15 and the heading error derivative to turn rate was increased from 0.40 to 0.60.

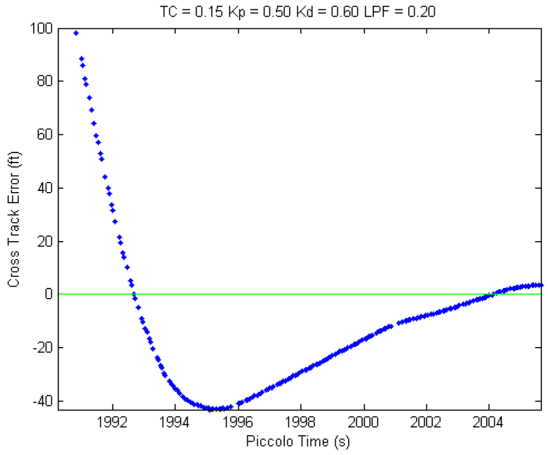




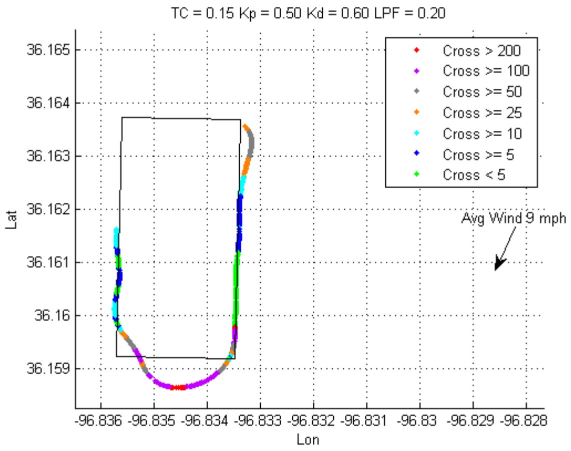
The figure above shows that the track performance improved significantly. After the initial overshoot the cross track error actually dampened out and appeared to converge. The land was manually aborted because the aircraft was nearly to the touchdown waypoint and had not yet converged.

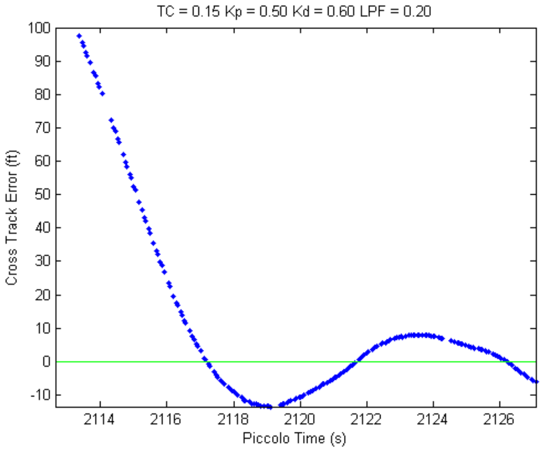
In Flight 21 there were 5 lands attempted with the same track control gains that were set in Flight 20. The aircraft was able to converge to within a 5 foot cross track error through each attempt. The following figures detail the tracking performance of the five land attempts. All five land attempts approached the airstrip from the south due to the wind. The touchdown waypoint was located on the west (left)leg of the land plan in the following figures.



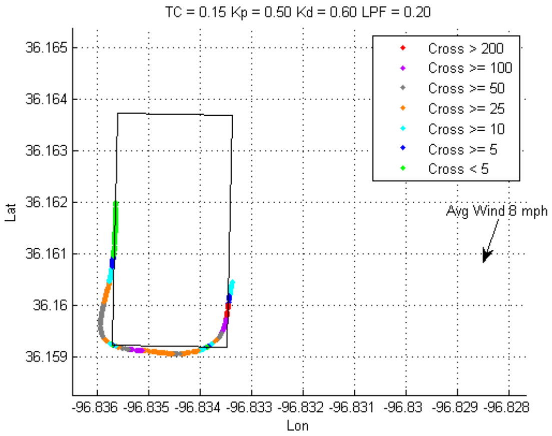


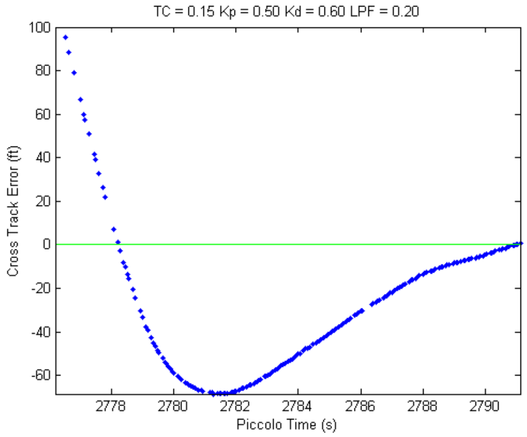
The first land attempt was abort by the manual pilot due to the apparent altitude error. The aircraft had oscillated to only 3 feet and was beginning to dampen out.



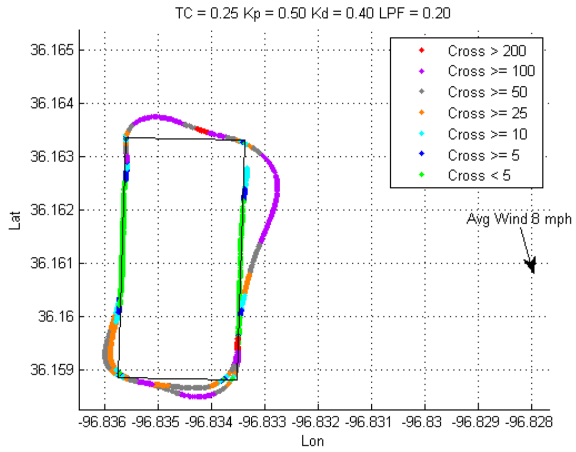


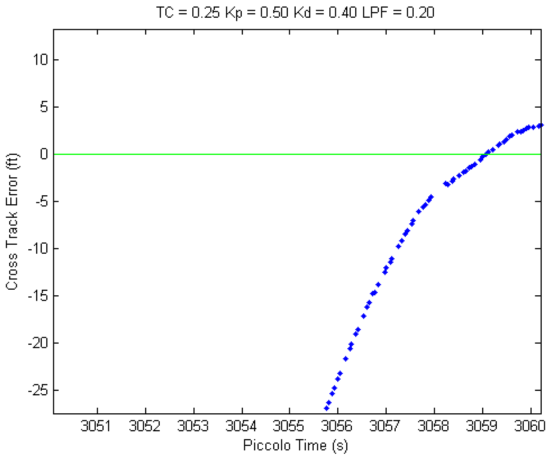
The second land attempt was aborted by the manual pilot because the aircraft’s direction was not straight down the runway due to the oscillations that had not yet dampened out. The aircraft had oscillated to 8 feet inside the flight path and was headed towards another oscillation peak on the other side of the flight path.

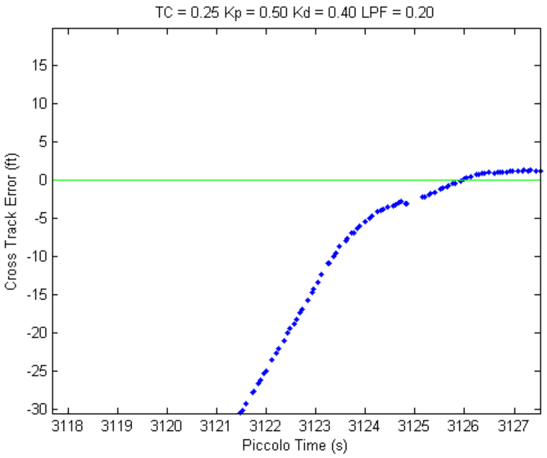




The third land attempt was a successful auto land. The aircraft converged to the track path and was very slowly oscillating towards in the inside of the flight plan.





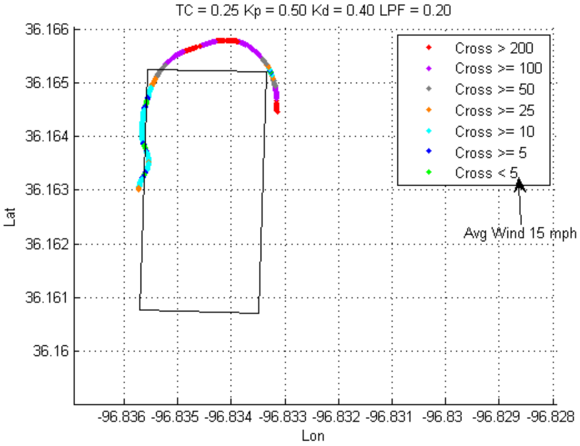


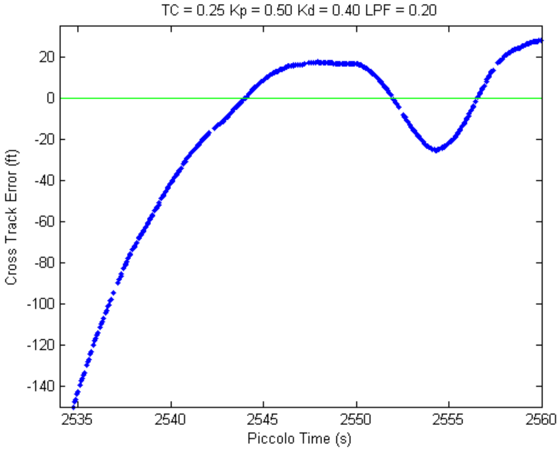
The figures above depict the third and fourth auto land attempts of Flight 21. The third and fourth land attempts were auto aborted due to altitude errors. In both instances the aircraft converged within 5 feet of the flight path on the short final leg of the land plan. After both auto aborts the aircraft was able to stay within the 5 foot cross track error while it flew towards the go around waypoint.

Flight	Tracker Convergence	Hdg err to turn rate	Hdg err der to turn rate	Turn error LPF cutoff
11	0.30	0.40	0.10	0.0
11	0.25	0.50	0.20	0.20
12	0.25	0.50	0.30	0.20
12	0.25	0.50	0.40	0.20
20	0.15	0.50	0.60	0.20

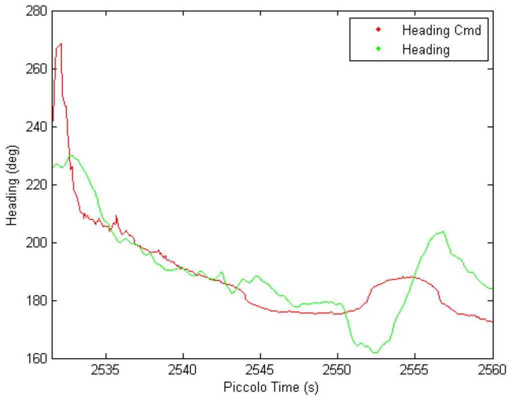
The table above depicts the track control gains as they were altered. The gains set in Flight 20 became the final track control gains. There were numerous flights and successful auto lands after Flight 21, which signaled that the track control gains did not need any more tuning.

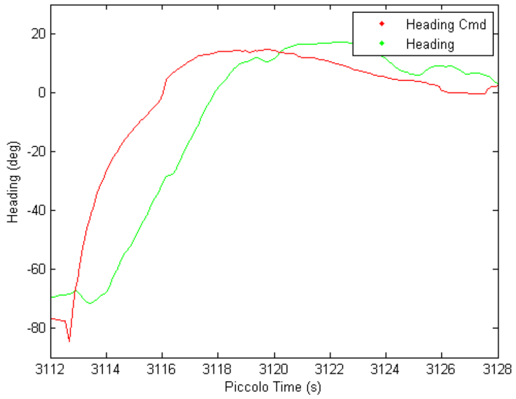
Flights 1 – 16 did not have "Controller" telemetry enabled. As a result the DevInterface data was not able to be used to analyze the lateral control performance. Flights 19 and 21 did have controller telemetry enabled; therefore, further analysis and comparisons were able to be made between the track control gains used in Flight 19 and the track control gains used in Flight 21. The track control gains in Flight 19 were the same as the track control gains set in Flight 12 shown in the table above.



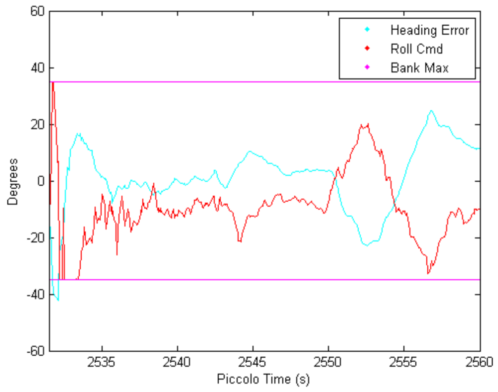


Flight 19 flew a land plan from the north. The aircraft oscillated in and out of the final approach track path three times before the attempt was aborted by the manual pilot as the aircraft neared the touchdown waypoint.

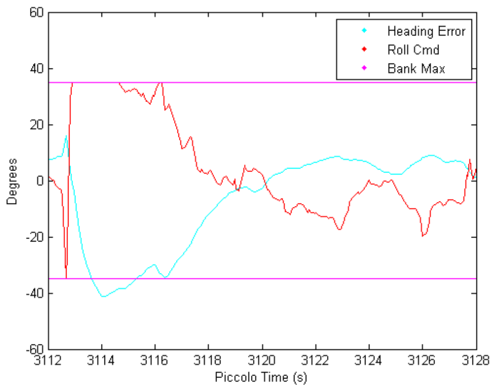




The figures above depict the heading command and actual heading of the Nexstar during the final approach leg of the land plans. The heading command in Flight 19 was oscillatory and not smooth when the autopilot initially targeted the final approach. Additionally the heading commands appeared to have targeted different trajectories many times during the final approach. The heading command in Flight 21 actually resembled elliptical trajectories. The trajectory was re-targeted two times during the final approach of Flight 21 before the autopilot auto aborted and flew to the go around waypoint.



Flight 17



Flight 21

The figures above depict the heading error versus roll command of both approaches. The heading error was plotted against the roll command because the roll command is calculated directly from the heading error after it passes through the track control loop gains. Flight 21 had a larger initial heading error. As a result the roll command was held at max bank angle for a longer period of time than it was during the approach of Flight 19. Near the end of Flight 19's approach, at 2550 seconds, the roll command increased dramatically from -11 to +4 degrees. As the roll command increased, and for a while afterwards the roll of the aircraft was still negative; thus, the heading error became negative as the aircraft oscillated inside the targeted flight path.

Due to the fact that there was not any DevInterface data for all of the flights where gain tuning occurred, and due to the fact that the roll control loop was not tuned, a conclusion from the detailed analysis cannot be made. It is possible that the elliptical trajectories that the autopilot was commanding contributed the most to the track control issues. Lowering the tracker convergence parameter essentially caused the autopilot to command tighter trajectories to converge on the targeted flight path. The results from the Flight 21 analysis show that the plotted trajectory created a large initial heading error which did cause the autopilot to command a long period of large bank commands. It is also possible that there was inner loop lag. Increasing the derivative gain seemed to help and by definition the derivative gain is used to offset inner loop lag.

1.19.5. Energy Control Gain Tuning

At the time that the energy control gains were tuned in the Nexstar guidelines for tuning energy control had not yet been created. Additionally a complete understanding of energy control had not been developed. As a result the gain tuning of energy control was a learning experience. Energy control wasn't first attempted to be tuned until Flight 17. In all of the flights prior to Flight 16 the Nexstar was flown with version 2.1.1.e. Prior to Flight 16 the software was upgraded to version 2.1.4.f. The new software came with changes to longitudinal control, most notably energy control. In the new version new gains and control loops were added to energy control to improve performance; however, the new control laws also created the need for gain tuning.

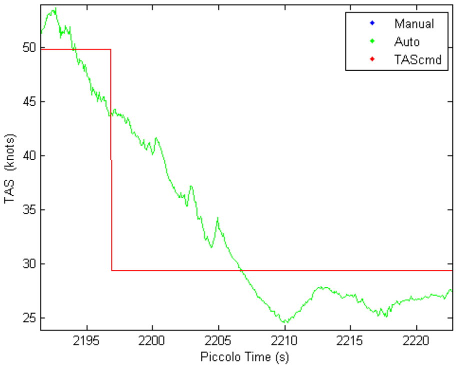
At the time the MATLAB GUI 'EnergyControl' had not yet been created; thus, the capability to analyze the control loops and gain effects of energy control did not exist. The only tool available to analyze energy control performance was the DevInterface which only provided throttle and airspeed commands. The DevInterface was used during flight to observe the airspeed performance of the aircraft in descents and climbs. After descents there would always be airspeed undershoot, where the autopilot would slowly throttle up from minimum throttle allowing the airspeed to drop well below the commanded airspeed. Additionally there would always be airspeed undershoot when a climb was initiated. As a result the aircraft would fly descents from a box to the lost comms loiter waypoint and climb from the loiter waypoint back to the box to gauge the airspeed performance of different variations of energy control gains. The vertical rate limits on the Nexstar were large due to the fact that the Nexstar was overpowered and there was a reasonable amount of confidence in the Nexstar's capability to pull off nearly vertical climbs; therefore, the commanded climb rates were on average around 12 ft/s. The large vertical climb rate commands meant that the aircraft would attempt to fly steep climbs; therefore, the throttle response needed to be relatively quick.

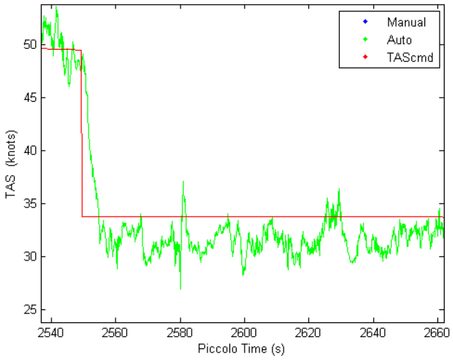
Initially it was thought that the energy and energy rates commanded through energy control always consisted of the entire energy equation where it would take into account not only airspeed (kinetic energy) errors but also altitude (potential energy) errors; thus, it was believed that the integral gain was integrating airspeed and altitude errors. The problem with integrating altitude errors was that during descents commanded between two separate flight plans the autopilot would command a step input for altitude thus creating a large altitude error that existed throughout the duration of the descent. It was believed that the large altitude error was integrated; thus, the autopilot would have to unwind the integral windup that resulted from the large area of positive altitude error that would occur during descents. This lead the initial energy control gain tuning to center around minimizing the integral gain with the idea that the integral gain was creating the airspeed undershoots after descents. Of course it is known that these assumptions are not true (Section 5.8.4.1); however, that was not known at the time.

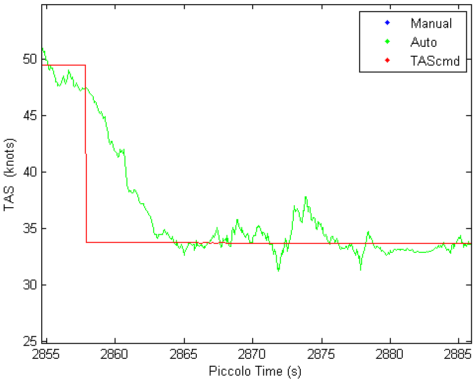
During energy control flight testing there was a couple of other issues that were not known at the time. In the software version 2.1.4.f the autopilot attempted to use “airspeed float” where it altered the airspeed commands during descents and climbs to help improve airspeed performance. The airspeed float was removed in version 2.1.4.g because it did not work properly; however, it still existed during the energy control maneuvers performed for the Nexstar. As a result some of the maneuvers include seemingly random airspeed command changes during the descents and climbs. The other issue that was not known but did impact the energy control results was longitudinal modes. There was absolutely no documentation or mention of longitudinal modes. During some of the energy control descents the autopilot did go into Lon

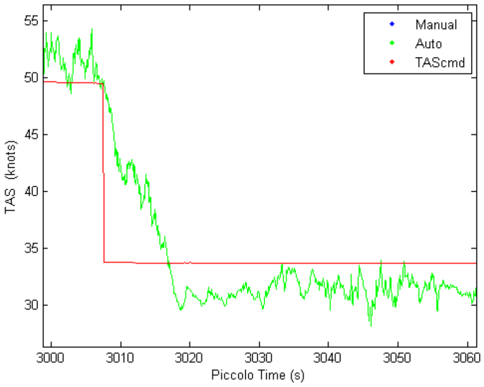
Mode 3 (fast airspeed mode). Fast airspeed mode skewed the results of the energy control maneuvers that it occurred in since the elevator was used to control airspeed.

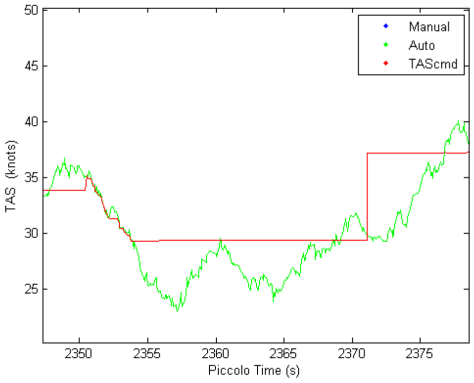
The first attempts at tuning energy control occurred in Flight 17. There were 4 descents and 3 climbs that were performed with different energy control gains. The autopilot went into fast airspeed mode during descent #2. The following figures depict the airspeed performance of each climb and descent.

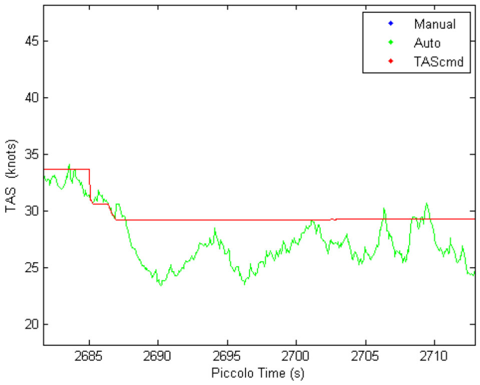


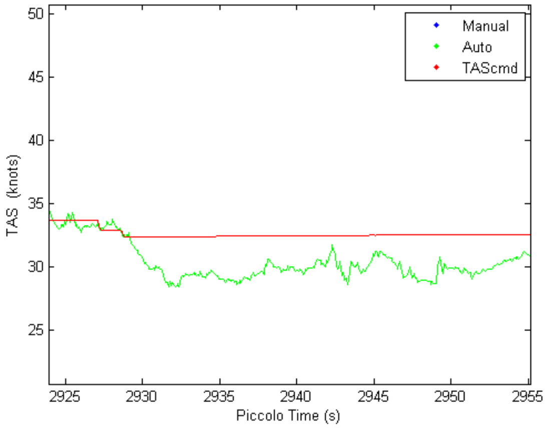












	Kpo	Kpi	KI	TPT	LPF	Climb TAS (knots)	Descent TAS (knots)
1	0.35	0.6	0.4	0	0	-6.37	-4.82
2	1.5	1	0.01	0.5	0.25	-5.72	N/A
3	2	1	0.01	0.5	0.25	-3.94	0
4							-4.04

After the default gains were attempted the outer proportional gain, “Energy err to energy rate”, was increased, the inner loop proportional gain, “Energy rate err to throttle”, was increased, and the inner loop integral gain, “Energy rate err int to throttle”, was decreased. Additionally the throttle prediction trust and low pass cutoff filter were enabled. The results of the second descent seemed to signal a near zero airspeed undershoot. At the time it seemed like improvement; however, the results were skewed by the fact that the autopilot went into fast airspeed mode. The second climb indicated improvement in throttle response as the airspeed undershoot decreased.

The third descent indicated no airspeed undershoot and at the time it was interpreted as an affirmation of the second descent even though the outer loop proportional gain had been increased. The third climb showed another decrease in the airspeed undershoot. The fourth descent was done for repeatability of the third descent and it produced an airspeed undershoot less than the undershoot of descents 1, and 2.

The energy control gains seemed to be heading in the right direction during the gain tuning in Flight 17; however, after the flight it was discussed that the throttle could be heard oscillating wildly during the flight. Further analysis and experimentation was done in software in the loop simulations and it was found that the outer loop proportional gain, “Energy err to energy rate”, was likely too high and could be driving the throttle oscillations. It was also theorized that the throttle prediction trust could be contributing to the throttle oscillations as well. It was not known at the time that the high outer loop gain was essentially amplifying the effects of noise in the airspeed measurements as it was commanding high energy rates due to small variations in airspeed error at each instance.

In Flight 19 the energy control tuning maneuver, box to loiter, was performed 3 times with 3 descents and 2 climbs. The plan was to decrease the throttle prediction trust, decrease energy err to energy rate, and try actually using the integral gain. It was not yet known that the integral gain was not integrating the altitude error during descents but it was decided to try the gain because there should be a good reason for it to exist in the first place. Unfortunately the autopilot went into fast airspeed mode (Lon Mode 3) during each descent, unknowingly at the time. At the time the results of the airspeed undershoot from the descents did seem to not change consistently with the gain changes. The results of the descents were useless for analyzing the effects of the different energy control gains; however, the airspeed undershoot during the climbs occurred in altitude control and could be analyzed. The following table details the peak airspeed undershoot after each climb was initiated with the different energy control gains.

Kpo	Kpi	KI	TPT	LPF	Climb TAS (knots)	Descent TAS (knots)
2	1	0.01	0.25	0.25	-5.91	N/A
1	1	0.4	0.25	0.25	-7.54	N/A

It was discovered, after Flight 26, that the integral gain was integrating the energy rate error, where the energy rate feedback was based purely off of the actual vertical rate of the aircraft. Additionally it was discovered that the energy rate command from the outer loop was based off of airspeed errors (in altitude control) and not altitude errors. As a result simulations were ran, performing energy control gain tuning in which the integral gain was increased first. It was found that increasing the integral gain, before increasing the proportional gains, could eliminate airspeed undershoot without increasing throttle oscillations.

Flight 27 put the simulation results into practice. In Flight 27 there were 7 descents and 6 climbs performed to test energy control gains. Unfortunately the autopilot went into fast airspeed mode (Lon Mode 3) during all but one of the descents, unknowingly at the time. As a result only the sixth descent could be used to analyze energy control performance; however, all 6 climbs could be used to analyze energy control performance. The tests began with default energy control gains and increased the integral gain until the airspeed undershoot on descent stopped decreasing. The last test added throttle prediction trust. Another issue that could have skewed the results of the tests in Flight 27 was that “uBlox VDown” was disabled in Flight 24 in order to help with auto takeoffs. That means that the vertical rate of the aircraft was calculated primarily from the barometer instead of primarily from GPS velocity vectors. As a result it is likely that the vertical rate estimations were not entirely accurate. The table below details the results of the tests.

	Kpo	Kpi	KI	TPT	LPF	Climb TAS (knots)	Descent TAS (knots)
1	0.35	0.60	0.40	0.0	0.0	-7.02	N/A
2	0.35	0.60	0.60	0.0	0.0	-11.79	N/A
3	0.35	0.60	0.80	0.0	0.0	-8.6	N/A
4	0.35	0.60	1.0	0.0	0.0	-7.47	N/A
5	0.35	0.60	0.80	0.0	0.0	-3.85	N/A

6	0.35	0.60	0.80	0.50	0.0	-4.37	-1.63

The results of Flight 27 were the final energy control tests ran on the Nexstar. The energy control gains used in the last maneuver, test 6, were set as the final values of energy control. The following table combines all of the results of the tests together.

Kpo	Kpi	KI	TPT	LPF	Climb TAS (knots)	Descent TAS (knots)
0.35	0.60	0.40	0.0	0.0	-7.02	N/A
0.35	0.60	0.60	0.0	0.0	-11.79	N/A
0.35	0.60	0.80	0.0	0.0	-8.6	N/A
0.35	0.60	1.0	0.0	0.0	-7.47	N/A
0.35	0.60	0.80	0.0	0.0	-3.85	N/A
0.35	0.60	0.80	0.50	0.0	-4.37	-1.63
0.35	0.6	0.4	0	0	-6.37	-4.82
1.5	1	0.01	0.5	0.25	-5.72	N/A
2	1	0.01	0.5	0.25	-3.94	-4.04
2	1	0.01	0.25	0.25	-5.91	N/A
1	1	0.4	0.25	0.25	-7.54	N/A

The results indicated that raising the integral gain helped reduce the airspeed undershoot during descents and climbs, but the impact was larger on descents. There was not enough data on the high outer loop proportional gain to make a conclusion to its effects with respect to airspeed undershoot. The high gain certainly contributed to heavy throttle oscillations. The throttle prediction trust seemed to have had an impact; however, it was changed too often with other gains at the same time making it hard to conclude anything from the results. The gains could be further tuned by performing the same maneuvers, except, adjusting the “Fast IAS error Threshold” to avoid the autopilot switching to fast airspeed mode during descents, and by using the gain tuning guidelines outlined in Section 8.10.

1.19.6. Auto Land & Auto Takeoff

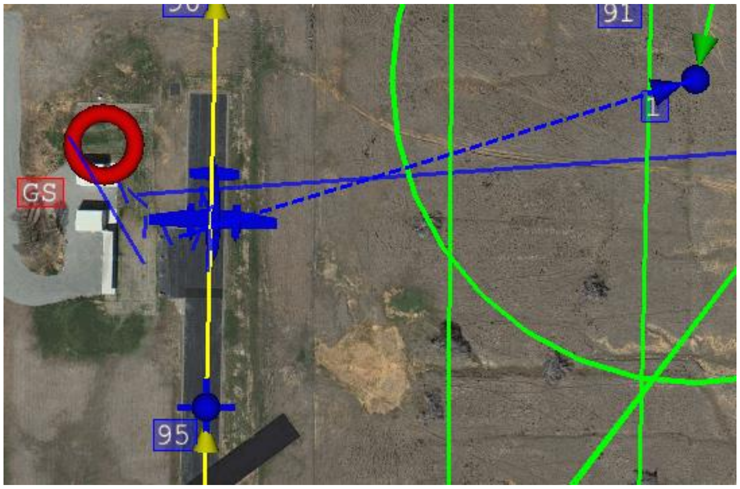
Autolands were attempted throughout many of the Nexstar flights, even before the gains were properly tuned. As a result there were a lot of aborted land attempts, manually aborted and auto aborted. The auto aborts were typically due to altitude errors where the altitude error was positive. Since the barometer was used there were several times that the aircraft was landed and the barometer was re – zeroed. When the barometer was zeroed the altitude was set typically 5 feet higher than it really was so that the aircraft could make it to the ground. The Nexstar used "Wheeled" landing as the land type.

Manual aborts were most commonly done because of lateral track oscillations and offsets. Since the track control gains were not completely tuned until Flight 20 there were many manual aborts in the flights preceding Flight 20. Some manual aborts were due to altitude. The auto land issues were mostly unrelated to the actual land settings. The only land setting change that seemed to help was increasing the approach speed fraction. The Nexstar's flight dynamics seemed to change when it flew slow, close to the estimated stall speed. The following table lists successful auto lands with their corresponding flight numbers. In total there were 7 successful autolands.

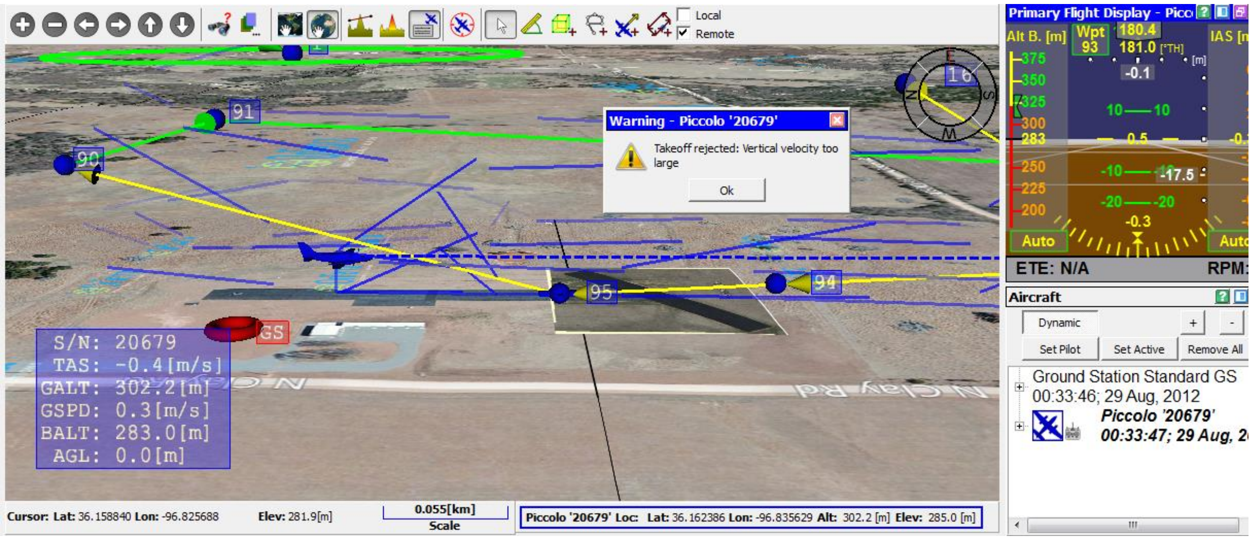
The Nexstar used the "Wheeled" launch type for the takeoff settings. Auto takeoffs were first attempted in Flight 23. The rotation and rolling elevator deflections were initially determined from analyzing manual takeoffs. Additionally the acceleration setting was determined from analyzing manual takeoffs. The settings were practiced in software in the loop simulations.

Initially, in Flight 23, the acceleration was set at 3 m/s^2, and the rolling elevator was set to 0 degrees with a rotation time of 1 second. The first auto abort error that was encountered was

“Takeoff Rejected: cross track error too large”. The waypoint that was being targeted was the loiter waypoint and it was off to the side of the runway.



It appeared that the autopilot interpreted the current state of the aircraft from the targeted path to be a cross track error and it prevented the aircraft from taking off. The second auto abort error that was encountered was “Takeoff Rejected: vertical velocity too large.”


Warning - Piccolo '20679'

Takeoff rejected: Vertical velocity too large

Ok

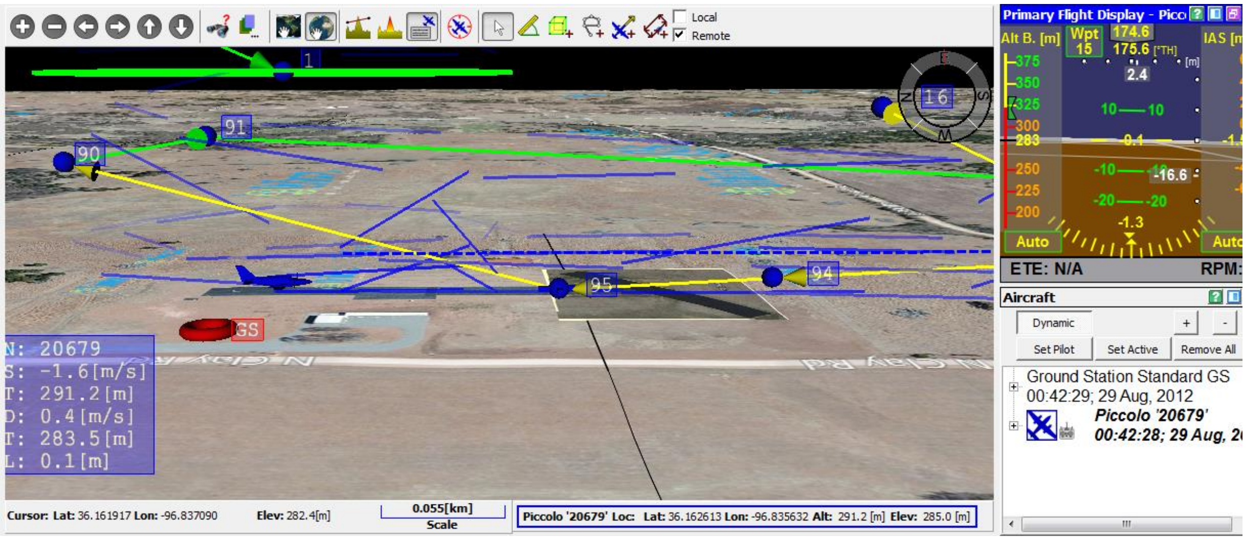
S/N: 20679
TAS: -0.4[m/s]
GALT: 302.2[m]
GSPD: 0.3[m/s]
BALT: 283.0[m]
AGL: 0.0[m]

Cursor: Lat: 36.158840 Lon: -96.825688 Elev: 281.9[m] 0.055[km] Scale

Piccolo '20679' Loc: Lat: 36.162386 Lon: -96.835629 Alt: 302.2 [m] Elev: 285.0 [m]

Ground Station Standard GS
00:33:46; 29 Aug, 2012
Piccolo '20679'
00:33:47; 29 Aug, 2

It turned out that because the autopilot thought that the aircraft was above the target flight path it perceived a positive altitude error and prevented the autopilot from taking off because it thought that the targeted waypoint was not commanding a positive vertical rate, or climb. It was found that there are two ways to eliminate this issue. One option is to disable the use of GPS to determine vertical rate by unchecking the “Disable uBlox VDown” option in the sensor configuration window of PCC. Another option is to taxi the aircraft on the runway until the GPS/INS solution recognizes that the aircraft is below the altitude of the targeted waypoint.



N: 20679
S: -1.6[m/s]
T: 291.2[m]
D: 0.4[m/s]
T: 283.5[m]
L: 0.1[m]

Cursor: Lat: 36.161917 Lon: -96.837090 Elev: 282.4[m] 0.055[km] Scale

Piccolo '20679' Loc: Lat: 36.162613 Lon: -96.835632 Alt: 291.2 [m] Elev: 285.0 [m]

ETE: N/A RPM:

Ground Station Standard GS
00:42:29; 29 Aug, 2012
Piccolo '20679'
00:42:28; 29 Aug, 2

The figure above depicts eliminating the perceived vertical rate error by taxiing the aircraft around the runway. This was done in Flight 23 in order to eliminate the takeoff rejections. On the fourth attempt the auto takeoff was successful, although the aircraft veered to the left of the runway as it took off.

The auto takeoffs were tested further in Flights 24 – 26 for repeatability. In Flight 24 there were issues with steering the aircraft straight down the runway. The autopilot kept steering the Nexstar off course of the center of the runway. In an attempt to correct the steering two of the steering gains were decreased. “Track Y to Vy” was decreased from 0.20 to 0.10 and “Track Vy err pro to nose gear” was decreased from 0.20 to 0.10. These gains were lowered because it appeared that the autopilot was over reacting to small cross track errors during ground roll. The ground roll was still not straight after altering the steering control gains. The rolling elevator was increased from 0 to -4 degrees in an attempt to take weight off of the nose gear. The acceleration was increased from 3 m/s2 to 4 m/s2 and the rotation time was decreased from 1 second to 0.5 seconds. After all of the changes the autopilot was still unable to successfully takeoff. On the last attempt the aircraft initially veered off course, but the autopilot did not have a snap response to bring it back, the aircraft was actually slowly steered back towards the center of the runway.

At the beginning of Flight 25 an auto takeoff was successfully attempted on the first try using the land and steering control settings that Flight 24 ended with. The autopilot steered the Nexstar nearly down the entire length of the runway and was able to keep it steady. Flight 25 was also the first fully autonomous flight. The table below contains the totals and their corresponding flights.

Flight	Auto Lands	Auto Takeoffs	Fully Autonomous Flight
18	2	0	0
21	1	0	0
22	3	0	0
23	1	0	0
25	1	2	1
26	1	1	1

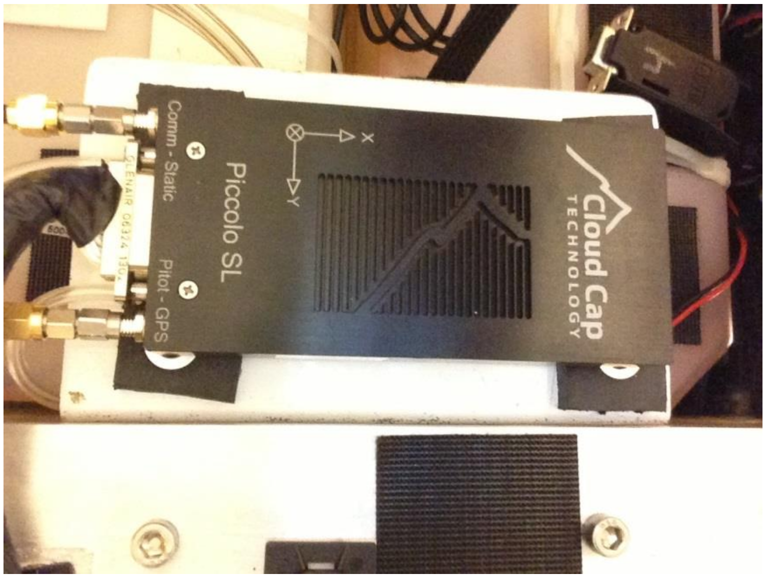
APPENDIX C

NOCTUA B1

1. Noctua B1

1.1. Hardware Configuration

1.1.1. Autopilot



Noctua B1 used the Piccolo SL 565 unit 4563. The autopilot was mounted upside down onto the same support that the landing gear was attached too.

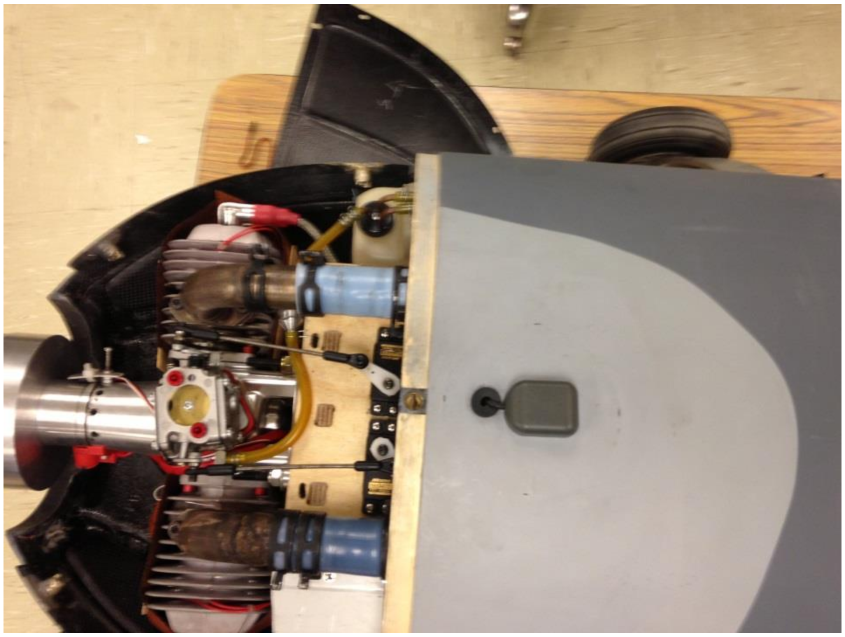
1.1.2. JR Level Shifter Board

Noctua B1 used the JR Level Shifter Board from Cloud Cap so the manual pilot could fly the aircraft through two antennas, independent of the piccolo communications. The JR antennas were mounted in the vertical tail, one in each tail.

1.1.3. Tach/Deadman Board

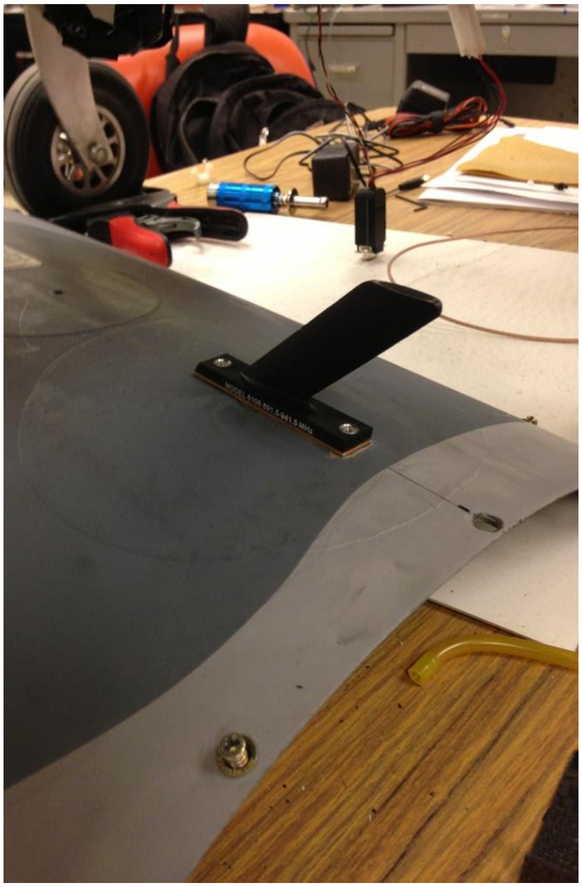
Noctua B1 used the 5V CDI Tach/Deadman Board from Cloud Cap. The board was used to read the rpm of the motor and cutoff power to the engine ignition.

1.1.4. GPS Module



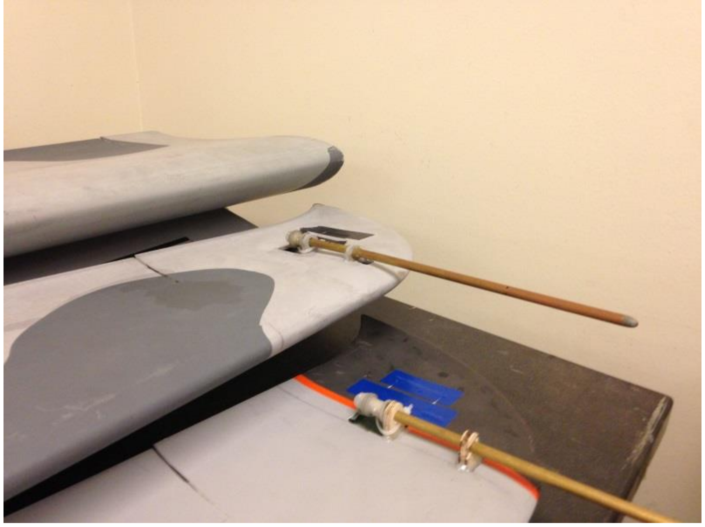
Noctua B1 used the GPS module offered by Cloud Cap for the SL along with the corresponding GPS ground plane. The GPS module was mounted on top of the fuselage near the cowling. The ground plane was underneath the paint. There was not paint directly underneath the module so that the module could make contact with the ground plane.

1.1.5. Communications Antenna



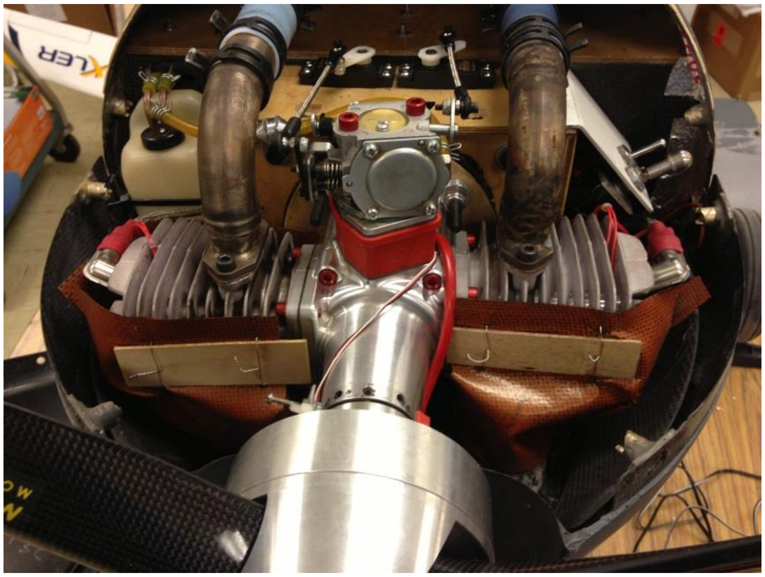
Noctua B1 used a Haigh-Farr 891.5 – 941.5 MHz antenna for piccolo communications along with Cloud Cap’s 900MHz ground plane. The antenna was mounted under the belly of the aircraft. The ground plane was underneath the paint. There was no paint directly underneath the antenna so that the antenna could make direct contact with the ground plane.

1.1.6. Pitot Tube



Noctua B1 used a pitot tube that was manufactured in house at Oklahoma State University. The pitot tube was mounted on the leading edge of the starboard wing tip.

1.1.7. Motor



Noctua B1 used a DA-100L gas motor.

1.1.8. Propeller

Noctua B1 used a Mejzlik 22x12 3-blade carbon fiber propeller.

1.1.9. Wiring Harness

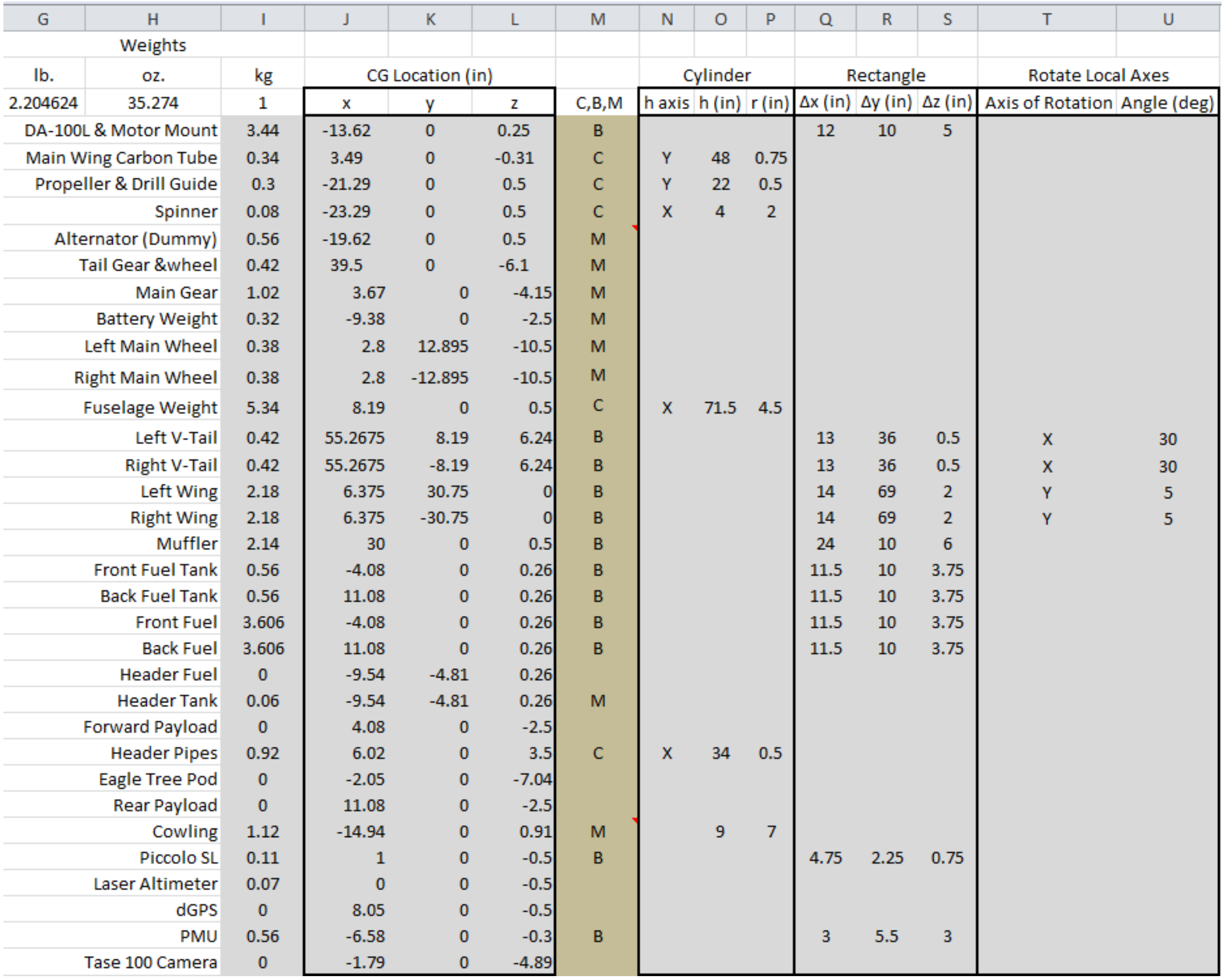
Noctua B1 used a custom wiring harness that was made in house at Oklahoma State University specifically for Noctua and a Piccolo SL unit. The following table details all of the pins that the configuration uses including the servo lines that each control surface operates from. All of the SL descriptions and designations come from Cloud Cap's "Piccolo External Interface" document.

Function	SL Pin #	I/O Index	SL Name	SL Function
Throttle	23	2	SERVO_2_PWM	TPU_A[2]
Left Flap	27	7	SERVO_7_PWM	TPU_B[14]
Right Flap	45	12	SERVO_12_PWM	AN70/TPU_B[0]
Left Aileron	21	0	SERVO_0_PWM	TPU_A[0]
Right Aileron	25	4	SERVO_4_PWM	TPU_A[4]
Hook	13	11	SERVO_11_PWM	AN69/TPU_A[13]
Tail Wheel	16	10	SERVO_10_PWM	AN68/TPU_A[12]
Right Ruddervator	24	3	SERVO_3_PWM	TPU_A[3]
Left Ruddervator	22	1	SERVO_1_PWM	TPU_A[1]
Tach/Deadman	30	5	SERVO_6_PWM	TPU_B[2]

JR Level Shifter	26	9	SERVO_9_PWM	TPU_A[5]
JR Level Shifter	47	6	SERVO_5_PWM	TPU_B[3]

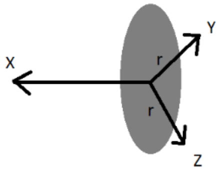
1.2. CG/Inertia Model

Upon installing the piccolo in Noctua B1 a component model of the aircraft had already been put together by a member of the Noctua team James Combs. His component model included each component, its weight, and the location of its center of gravity. All of the location measurements had been made with the same reference point as the geometry measurements, (0,0,0) at the center on the wing leading edge. The components were copied into the "CGInertiaModel" excel spreadsheet.



G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	Weights													
lb.	oz.	kg	CG Location (in)				Cylinder			Rectangle			Rotate Local Axes	
2.204624	35.274	1	x	y	z	C,B,M	h axis	h (in)	r (in)	Δx (in)	Δy (in)	Δz (in)	Axis of Rotation	Angle (deg)
	DA-100L & Motor Mount	3.44	-13.62	0	0.25	B				12	10	5		
	Main Wing Carbon Tube	0.34	3.49	0	-0.31	C	Y	48	0.75					
	Propeller & Drill Guide	0.3	-21.29	0	0.5	C	Y	22	0.5					
	Spinner	0.08	-23.29	0	0.5	C	X	4	2					
	Alternator (Dummy)	0.56	-19.62	0	0.5	M								
	Tail Gear &wheel	0.42	39.5	0	-6.1	M								
	Main Gear	1.02	3.67	0	-4.15	M								
	Battery Weight	0.32	-9.38	0	-2.5	M								
	Left Main Wheel	0.38	2.8	12.895	-10.5	M								
	Right Main Wheel	0.38	2.8	-12.895	-10.5	M								
	Fuselage Weight	5.34	8.19	0	0.5	C	X	71.5	4.5					
	Left V-Tail	0.42	55.2675	8.19	6.24	B				13	36	0.5	X	30
	Right V-Tail	0.42	55.2675	-8.19	6.24	B				13	36	0.5	X	30
	Left Wing	2.18	6.375	30.75	0	B				14	69	2	Y	5
	Right Wing	2.18	6.375	-30.75	0	B				14	69	2	Y	5
	Muffler	2.14	30	0	0.5	B				24	10	6		
	Front Fuel Tank	0.56	-4.08	0	0.26	B				11.5	10	3.75		
	Back Fuel Tank	0.56	11.08	0	0.26	B				11.5	10	3.75		
	Front Fuel	3.606	-4.08	0	0.26	B				11.5	10	3.75		
	Back Fuel	3.606	11.08	0	0.26	B				11.5	10	3.75		
	Header Fuel	0	-9.54	-4.81	0.26									
	Header Tank	0.06	-9.54	-4.81	0.26	M								
	Forward Payload	0	4.08	0	-2.5									
	Header Pipes	0.92	6.02	0	3.5	C	X	34	0.5					
	Eagle Tree Pod	0	-2.05	0	-7.04									
	Rear Payload	0	11.08	0	-2.5									
	Cowling	1.12	-14.94	0	0.91	M		9	7					
	Piccolo SL	0.11	1	0	-0.5	B				4.75	2.25	0.75		
	Laser Altimeter	0.07	0	0	-0.5									
	dGPS	0	8.05	0	-0.5									
	PMU	0.56	-6.58	0	-0.3	B				3	5.5	3		
	Tase 100 Camera	0	-1.79	0	-4.89									

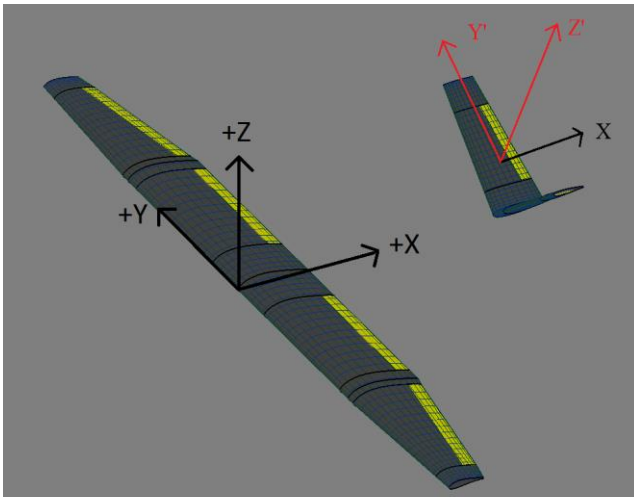
Most of the components were treated as constant density rectangles (“B”), cylinders (“C”), or point masses (“M”); however, two components were treated differently. The “Alternator” and “Cowling” were marked as “M”, but their corresponding inertia calculations were changed. The alternator was treated as a constant density disk. The cowling was treated as a constant density hollow sphere. The sphere equation was for an entire sphere so the equations were divided by 2 since the cowling was only one half of a sphere. The equations that were adjusted were located in the corresponding “Local Mass Moment of Inertia” cells. The inertial calculations for both are shown below.



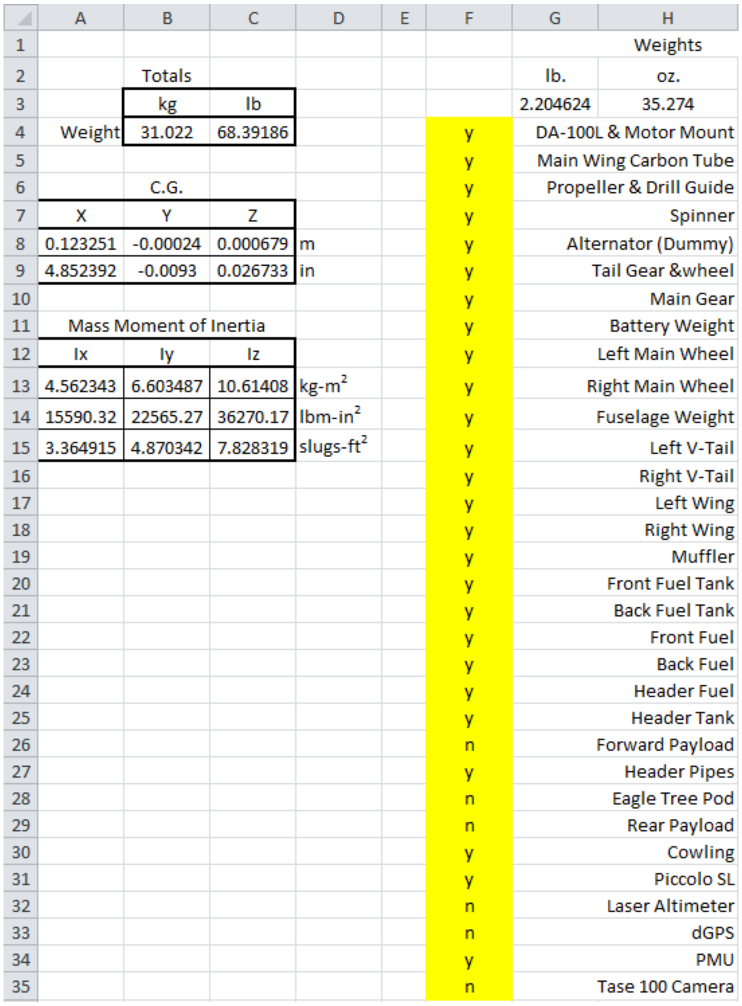
$$I_{xx} = \frac{1}{2}mr^2 \quad I_{yy} = I_{zz} = \frac{1}{4}mr^2$$

$$Hollow\ Sphere\colon\ I_{xx} = I_{yy} = I_{zz} = \frac{2}{3}mr^2$$

The V-Tail surface local inertias were adjusted to correct for the 30 degree dihedral of the surfaces around the x - axis. Since the correction only corrects for one rotation the tail incidence of 4.5 degrees was ignored because the effect of the 30 degree dihedral on the tails would have a larger effect than the incidence with respect to local inertia.



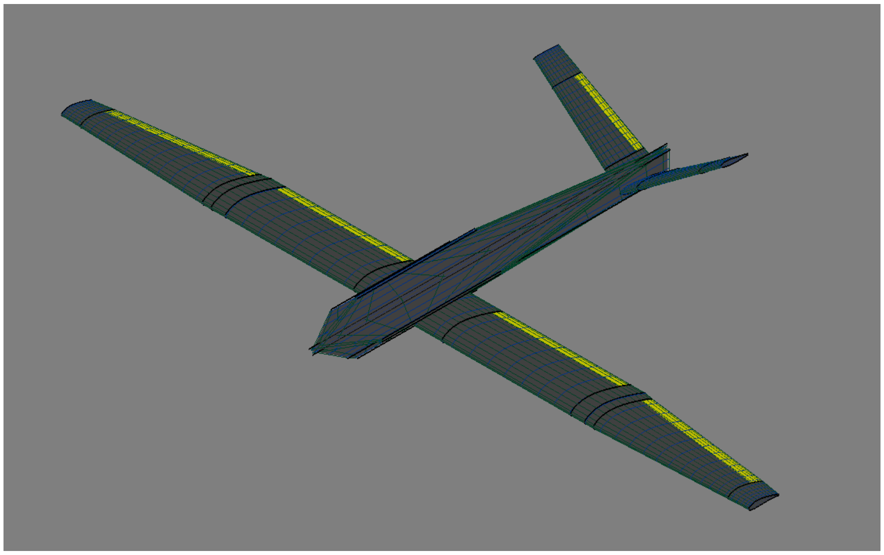
The figure above depicts the local axes that were rotated so the spreadsheet could use the parallel axis theorem to translate the inertias to the center of gravity. Similarly the wings were adjusted for their 5 degrees of incidence which rotated the local axes of the wing about the y – axis.



	A	B	C	D	E	F	G	H
1								Weights
2		Totals					lb.	oz.
3		kg	lb				2.204624	35.274
4	Weight	31.022	68.39186			y		DA-100L & Motor Mount
5						y		Main Wing Carbon Tube
6		C.G.				y		Propeller & Drill Guide
7	X	Y	Z			y		Spinner
8	0.123251	-0.00024	0.000679	m		y		Alternator (Dummy)
9	4.852392	-0.0093	0.026733	in		y		Tail Gear &wheel
10						y		Main Gear
11		Mass Moment of Inertia				y		Battery Weight
12	Ix	Iy	Iz			y		Left Main Wheel
13	4.562343	6.603487	10.61408	kg-m^2		y		Right Main Wheel
14	15590.32	22565.27	36270.17	lbm-in^2		y		Fuselage Weight
15	3.364915	4.870342	7.828319	slugs-ft^2		y		Left V-Tail
16						y		Right V-Tail
17						y		Left Wing
18						y		Right Wing
19						y		Muffler
20						y		Front Fuel Tank
21						y		Back Fuel Tank
22						y		Front Fuel
23						y		Back Fuel
24						y		Header Fuel
25						y		Header Tank
26						n		Forward Payload
27						y		Header Pipes
28						n		Eagle Tree Pod
29						n		Rear Payload
30						y		Cowling
31						y		Piccolo SL
32						n		Laser Altimeter
33						n		dGPS
34						y		PMU
35						n		Tase 100 Camera

The figure above depicts the results of the component model. The center of gravity location is measured with respect to the center of the wing leading edge. The Forward Payload, Eagle Tree Pod, Laser Altimeter, dGPS, and Tase 100 Camera were excluded from the calculations because they did not exist in the aircraft at the time that the model was created.

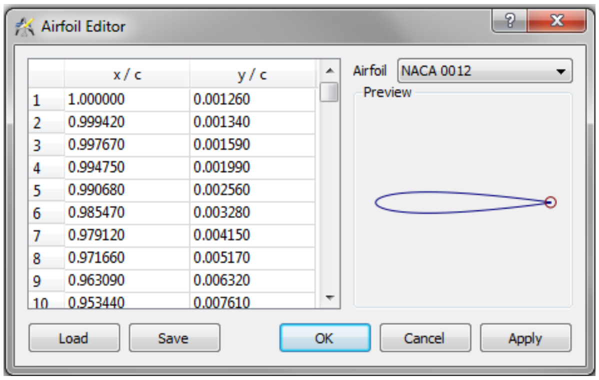
1.3. AVLEditor Model



1.3.1. Airfoils

Noctua had an Eppler 561 airfoil on the wings, and an NACA 0012 on the tail.

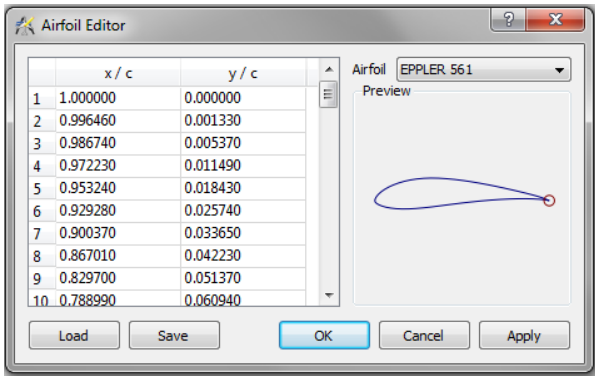
1.3.1.1. NACA 0012



	x / c	y / c
1	1.000000	0.001260
2	0.999420	0.001340
3	0.997670	0.001590
4	0.994750	0.001990
5	0.990680	0.002560
6	0.985470	0.003280
7	0.979120	0.004150
8	0.971660	0.005170
9	0.963090	0.006320
10	0.953440	0.007610

NACA 0012 airfoil file was generated with AVLEditor’s Airfoil Editor.

1.3.1.2. Eppler 561

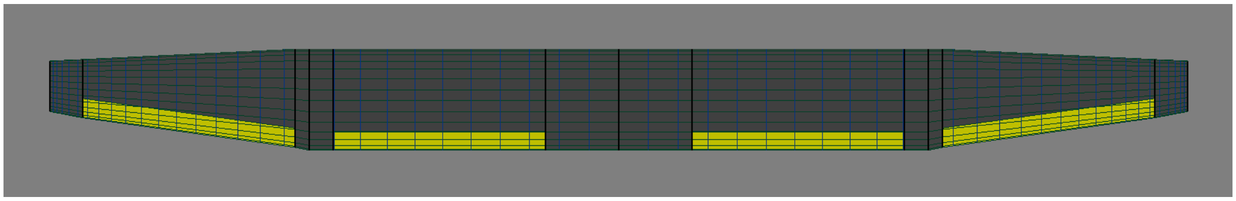


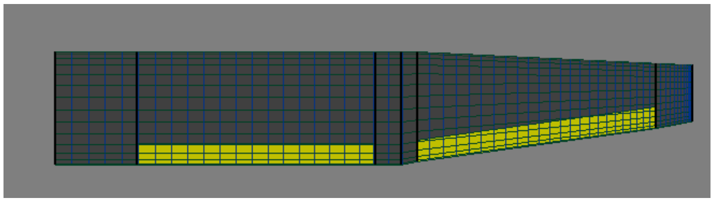
Airfoil: EPPLER 561

	x / c	y / c
1	1.000000	0.000000
2	0.996460	0.001330
3	0.986740	0.005370
4	0.972230	0.011490
5	0.953240	0.018430
6	0.929280	0.025740
7	0.900370	0.033650
8	0.867010	0.042230
9	0.829700	0.051370
10	0.788990	0.060940

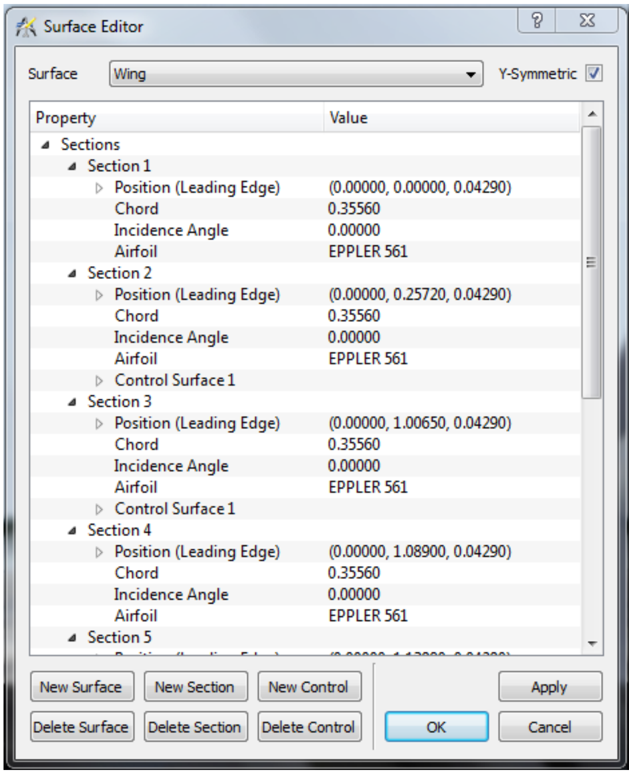
An Eppler 561 dat file was downloaded from http://www.ae.illinois.edu/m-selig/ads/coord_database.html#C. Initially the file was not formatted appropriately for AVLEditor. The file had to be adjusted to the appropriate format outlined in Section 4.1.4.2.

1.3.2. Wing



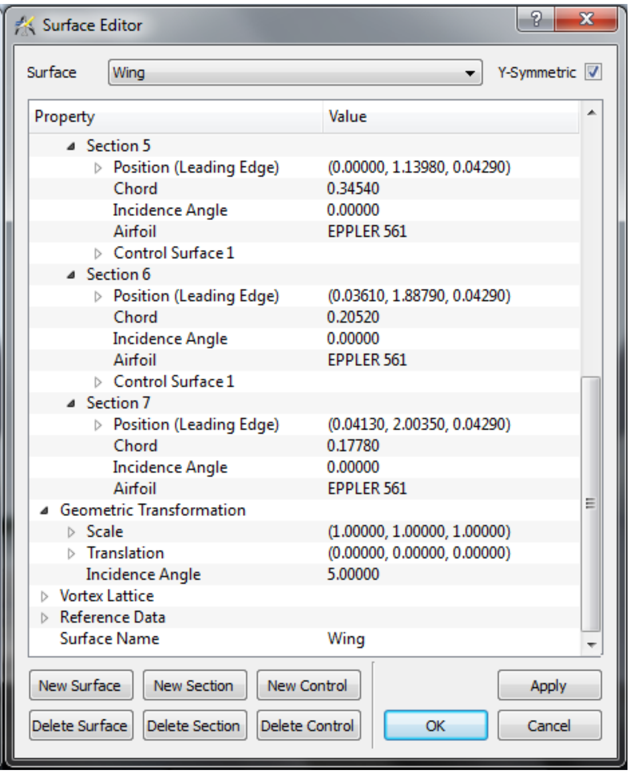


The wing was created with 7 sections, counting the centerline and wing tip. Y-Symmetry was used to mirror the wing on the port side.



Surface: Wing ▼ Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(0.00000, 0.00000, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Section 2	
Position (Leading Edge)	(0.00000, 0.25720, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Section 3	
Position (Leading Edge)	(0.00000, 1.00650, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Section 4	
Position (Leading Edge)	(0.00000, 1.08900, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Section 5	

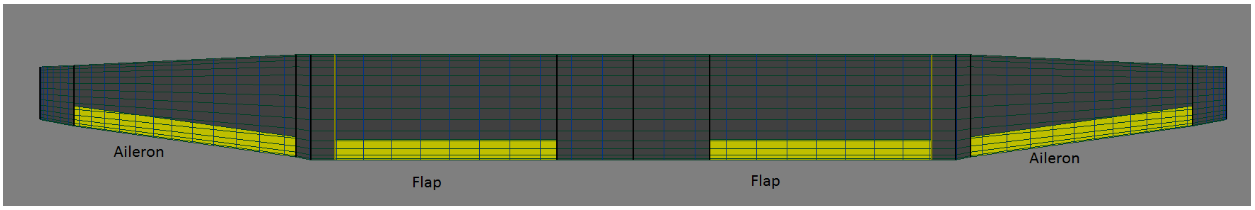


Surface: Wing ▼ Y-Symmetric ☑

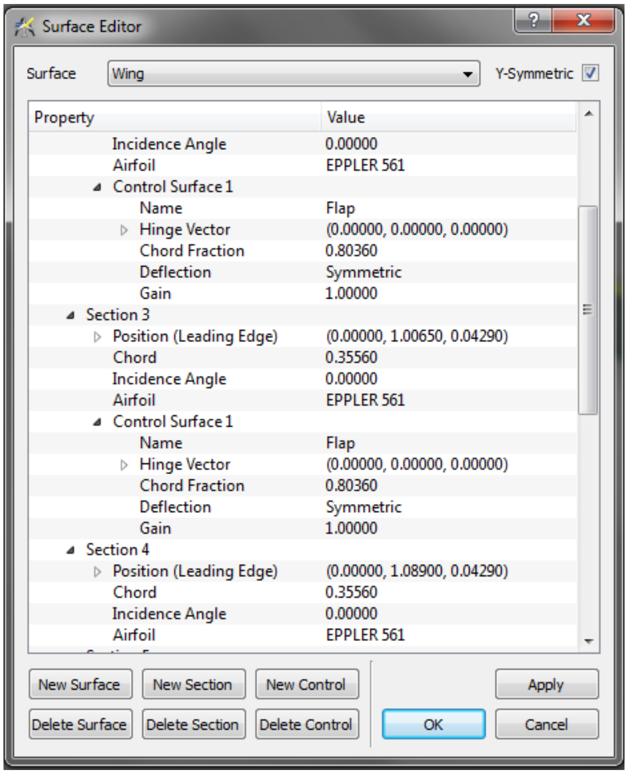
Property	Value
Section 5	
Position (Leading Edge)	(0.00000, 1.13980, 0.04290)
Chord	0.34540
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Section 6	
Position (Leading Edge)	(0.03610, 1.88790, 0.04290)
Chord	0.20520
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Section 7	
Position (Leading Edge)	(0.04130, 2.00350, 0.04290)
Chord	0.17780
Incidence Angle	0.00000
Airfoil	EPPLER 561
Geometric Transformation	
Scale	(1.00000, 1.00000, 1.00000)
Translation	(0.00000, 0.00000, 0.00000)
Incidence Angle	5.00000
Vortex Lattice	
Reference Data	
Surface Name	Wing

The figure above shows the sections and their positions. The surface was named "Wing". Geometric Transformation was used to apply the wing incidence of 5 degrees. The airfoil was assigned to the Eppler 561.

1.3.2.1. Control Surfaces



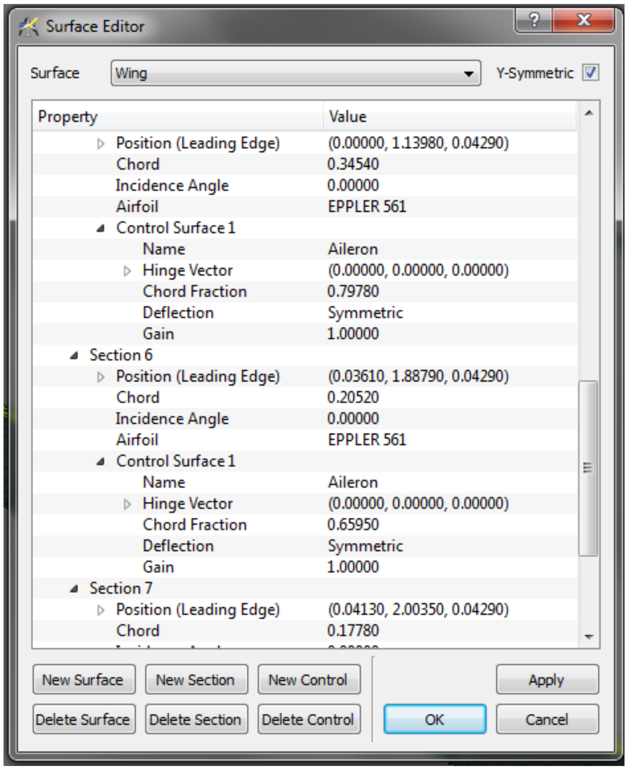
Noctua had two control surfaces on each wing; flaps, and ailerons.



Surface: Wing ☑ Y-Symmetric

Property	Value
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Flap
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80360
Deflection	Symmetric
Gain	1.00000
Section 3	
Position (Leading Edge)	(0.00000, 1.00650, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561
Control Surface 1	
Name	Flap
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.80360
Deflection	Symmetric
Gain	1.00000
Section 4	
Position (Leading Edge)	(0.00000, 1.08900, 0.04290)
Chord	0.35560
Incidence Angle	0.00000
Airfoil	EPPLER 561

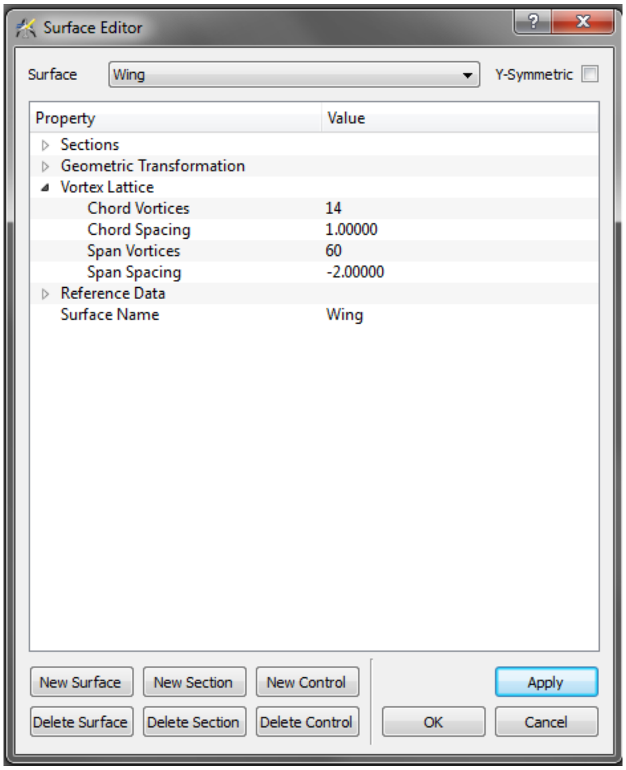
The flaps were added to sections 2 and 3, as shown in the figure above. The control surface was named “Flap”. Y-Symmetry automatically designated the two flaps as separate control surfaces, “LFlap” and “RFlap”. “Deflection” was set at the default setting “Symmetric”. “Gain” was set at the default value of 1.



Property	Value
▷ Position (Leading Edge)	(0.00000, 1.13980, 0.04290)
Chord	0.34540
Incidence Angle	0.00000
Airfoil	EPPLER 561
◢ Control Surface 1	
Name	Aileron
▷ Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.79780
Deflection	Symmetric
Gain	1.00000
◢ Section 6	
▷ Position (Leading Edge)	(0.03610, 1.88790, 0.04290)
Chord	0.20520
Incidence Angle	0.00000
Airfoil	EPPLER 561
◢ Control Surface 1	
Name	Aileron
▷ Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.65950
Deflection	Symmetric
Gain	1.00000
◢ Section 7	
▷ Position (Leading Edge)	(0.04130, 2.00350, 0.04290)
Chord	0.17780

The ailerons were placed between surfaces 5 and 6. The ailerons were named “Aileron.” Y-Symmetry automatically designated the two ailerons as separate control surfaces, “LAileron” and “RAileron”. The chord fraction on section 6 had to be adjusted because the ailerons were constant width, the wing was tapered, and the AVLEditor automatically assumed that the chord fraction stayed constant; therefore, the ailerons were initially tapered. The “Deflection” was set at the default setting “Symmetric”. “Gain” was set at the default value of 1.

1.3.2.2. Vortex Settings

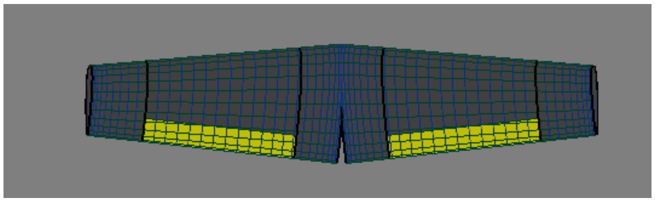


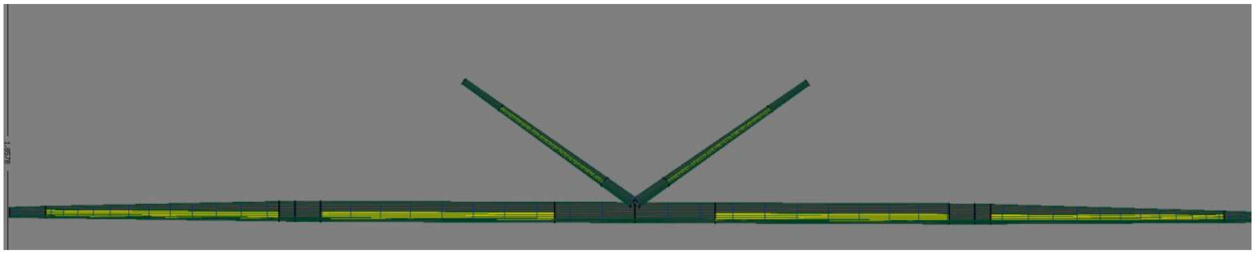
Surface: Wing ▾ Y-Symmetric ☐

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	14
Chord Spacing	1.00000
Span Vortices	60
Span Spacing	-2.00000
▷ Reference Data	
Surface Name	Wing

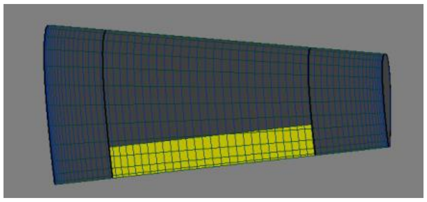
The Vortex settings are shown in the figure above. They were set following the guidelines in Section 4.1.10. The span spacing was set so that there was a vortex line on the sections that began tapering, and so that there were a few vortices bunched at the wing tips. The chord spacing was set so that there was a vortex line where the control surface began and so that there were multiple vortices on the leading and trailing edges.

1.3.3. V-Tail

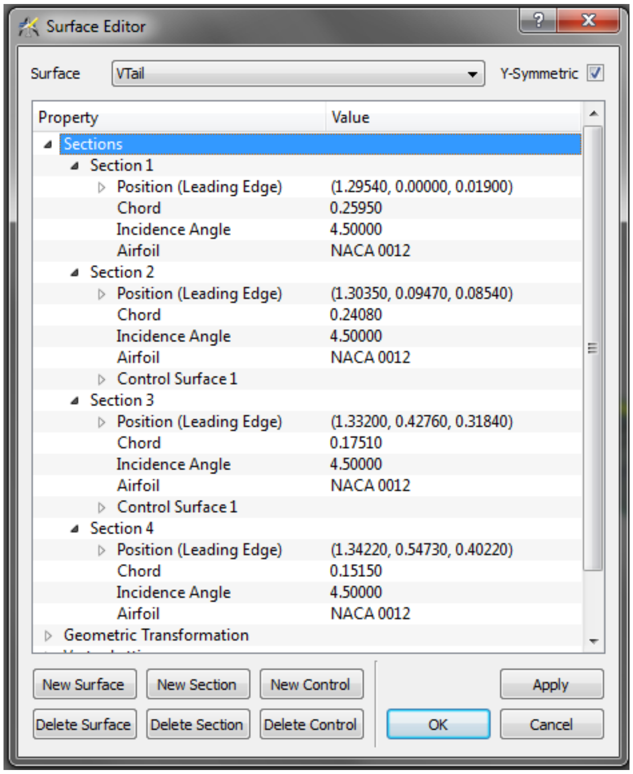




The tail configuration of Noctua was a V-Tail angled at 35 degrees.



The vtail was created with 4 sections, counting the centerline and tips. Y-Symmetry was used to mirror the tail on the port side. The tail had dihedral of 35 degrees.

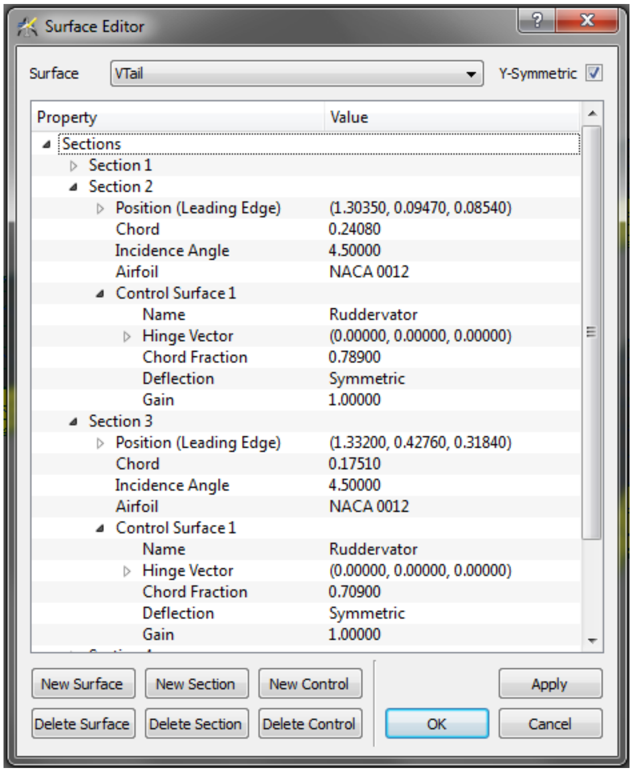


Surface: VTail
Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(1.29540, 0.00000, 0.01900)
Chord	0.25950
Incidence Angle	4.50000
Airfoil	NACA 0012
Section 2	
Position (Leading Edge)	(1.30350, 0.09470, 0.08540)
Chord	0.24080
Incidence Angle	4.50000
Airfoil	NACA 0012
Control Surface 1	
Section 3	
Position (Leading Edge)	(1.33200, 0.42760, 0.31840)
Chord	0.17510
Incidence Angle	4.50000
Airfoil	NACA 0012
Control Surface 1	
Section 4	
Position (Leading Edge)	(1.34220, 0.54730, 0.40220)
Chord	0.15150
Incidence Angle	4.50000
Airfoil	NACA 0012
Geometric Transformation	

The figure above shows the sections and their positions. The surface was named "VTail". When I tried to use Geometric Transformation to add the tail incidence, it did not work very well. It twisted the tail non-uniformly. As a result the incidence was input on each individual section, under "Incidence Angle", instead of using Geometric Transformation. The NACA 0012 airfoil was assigned to each section.

1.3.3.1. Control Surfaces

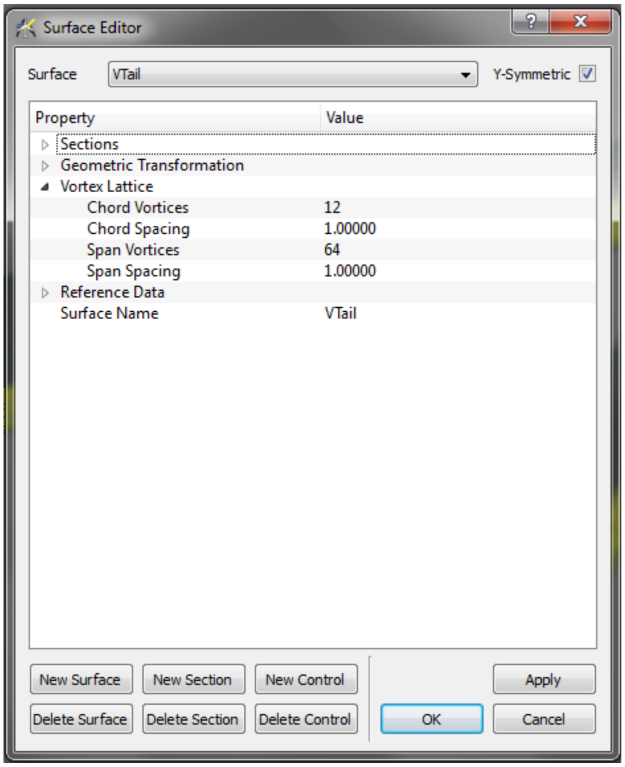


Surface: VTail
Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Section 2	
Position (Leading Edge)	(1.30350, 0.09470, 0.08540)
Chord	0.24080
Incidence Angle	4.50000
Airfoil	NACA 0012
Control Surface 1	
Name	Ruddervator
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.78900
Deflection	Symmetric
Gain	1.00000
Section 3	
Position (Leading Edge)	(1.33200, 0.42760, 0.31840)
Chord	0.17510
Incidence Angle	4.50000
Airfoil	NACA 0012
Control Surface 1	
Name	Ruddervator
Hinge Vector	(0.00000, 0.00000, 0.00000)
Chord Fraction	0.70900
Deflection	Symmetric
Gain	1.00000

There was one control surface on the vtail, ruddervators. The ruddervators acted as an elevator and a rudder. The ruddervators were assigned to sections 2 and 3, and they were named "Ruddervator" as instructed by Section 4.1.6 so that the simulator would recognize that the control surface deflections would be used to pitch and yaw the aircraft. The chord fraction was different so that the control surface thickness would be constant rather than taper. The "Deflection" was set at the default setting "Symmetric". "Gain" was set at the default value of 1.

1.3.3.2. Vortex Settings



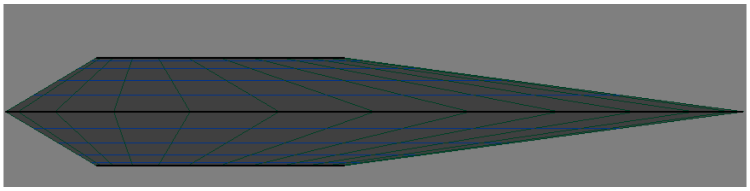
Surface Editor

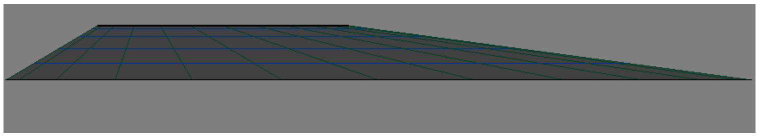
Surface: VTail Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	1.00000
Span Vortices	64
Span Spacing	1.00000
▷ Reference Data	
Surface Name	VTail

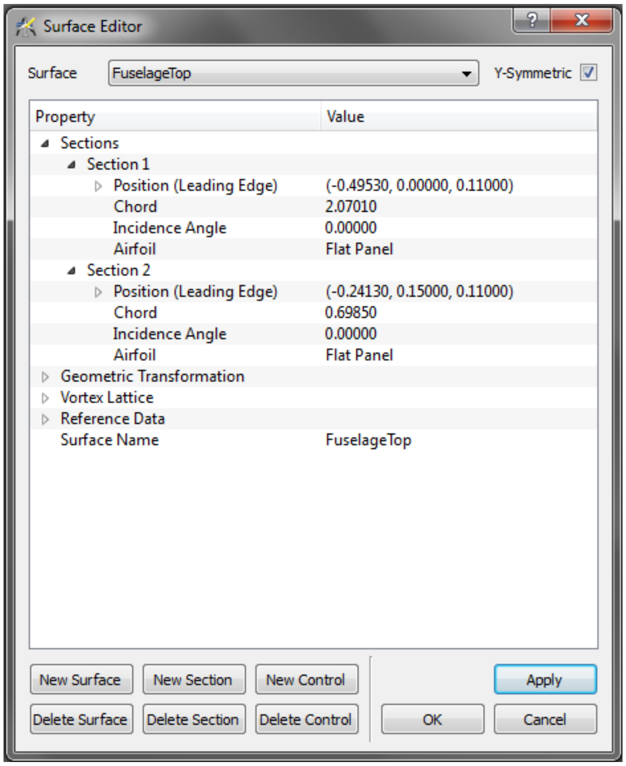
The Vortex settings are shown in the figure above. They were set following the guidelines in Section 4.1.10.

1.3.4. Fuselage Top





The top fuselage surface was made up of 2 sections. One section was at the center of the fuselage and one was at the outside diameter. The top section was located in the x-y plane and was mirrored across the x axis using the Y-Symmetric tool. The length of the surface traveled downstream, along the x – axis.



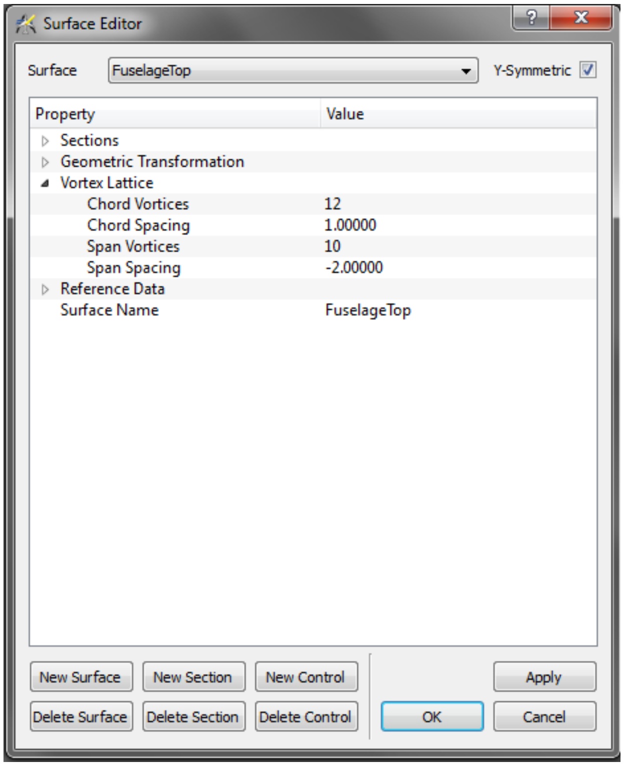
Surface Editor

Surface: FuselageTop   Y-Symmetric ☑

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(-0.49530, 0.00000, 0.11000)
Chord	2.07010
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(-0.24130, 0.15000, 0.11000)
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Name	FuselageTop

The figure above shows the sections and their positions. The surface was named "FuselageTop".   The fuselage surface was treated as a flat panel airfoil as directed in the guidelines, Section 4.1.9.1.

1.3.4.1. Vortex Settings

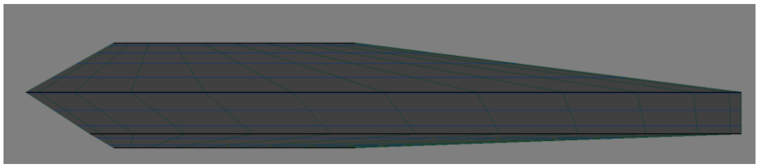


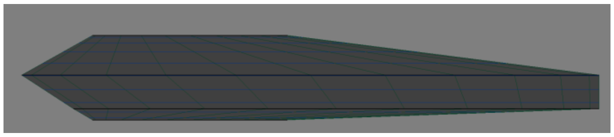
Surface FuselageTop Y-Symmetric ☑

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	1.00000
Span Vortices	10
Span Spacing	-2.00000
▷ Reference Data	
Surface Name	FuselageTop

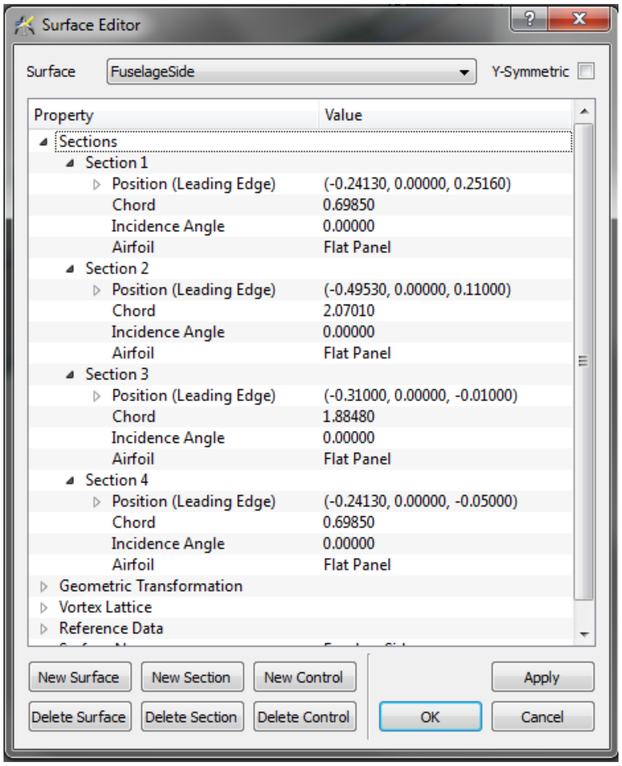
The Vortex settings are shown in the figure above. They were set following the guidelines in Section 4.1.10. As instructed the fuselage surfaces only contained a few vortices.

1.3.5. Fuselage Side





The side fuselage surface was made up of 4 sections. One section was at the center of the fuselage, one at the top diameter of the fuselage, and two at the bottom. The side fuselage surface was located in the x-z plane so it had no y – dimensions; thus, y symmetry was not used. The length of the surface traveled downstream, along the x – axis.

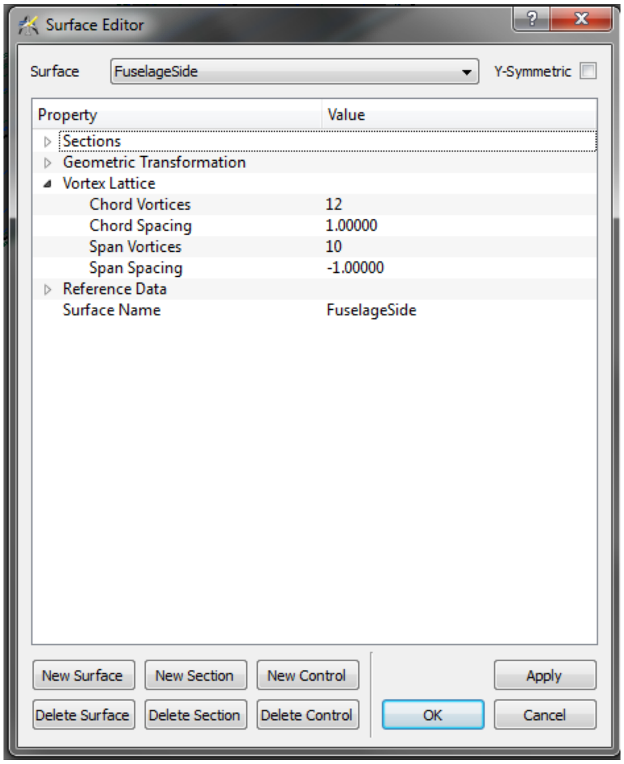


Surface: FuselageSide

Property	Value
Sections	
Section 1	
Position (Leading Edge)	(-0.24130, 0.00000, 0.25160)
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 2	
Position (Leading Edge)	(-0.49530, 0.00000, 0.11000)
Chord	2.07010
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 3	
Position (Leading Edge)	(-0.31000, 0.00000, -0.01000)
Chord	1.88480
Incidence Angle	0.00000
Airfoil	Flat Panel
Section 4	
Position (Leading Edge)	(-0.24130, 0.00000, -0.05000)
Chord	0.69850
Incidence Angle	0.00000
Airfoil	Flat Panel
Geometric Transformation	
Vortex Lattice	
Reference Data	

The figure above shows the sections and their positions. The surface was named “FuselageSide”. The fuselage surface was treated as a flat panel airfoil as directed in the guidelines, Section 4.1.9.2.

1.3.5.1. Vortex Settings

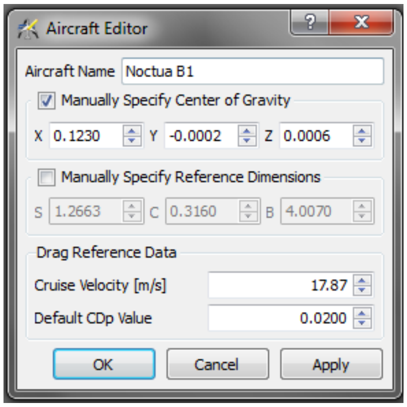


Surface FuselageSide Y-Symmetric □

Property	Value
▷ Sections	
▷ Geometric Transformation	
◢ Vortex Lattice	
Chord Vortices	12
Chord Spacing	1.00000
Span Vortices	10
Span Spacing	-1.00000
▷ Reference Data	
Surface Name	FuselageSide

The Vortex settings are shown in the figure above. They were set following the guidelines in Section 4.1.10. As instructed the fuselage surfaces only contained a few vortices, and the span spacing was set to -1.0.

1.3.6. Aircraft Data



Aircraft Name Noctua B1

☑ Manually Specify Center of Gravity

X 0.1230 Y -0.0002 Z 0.0006

□ Manually Specify Reference Dimensions

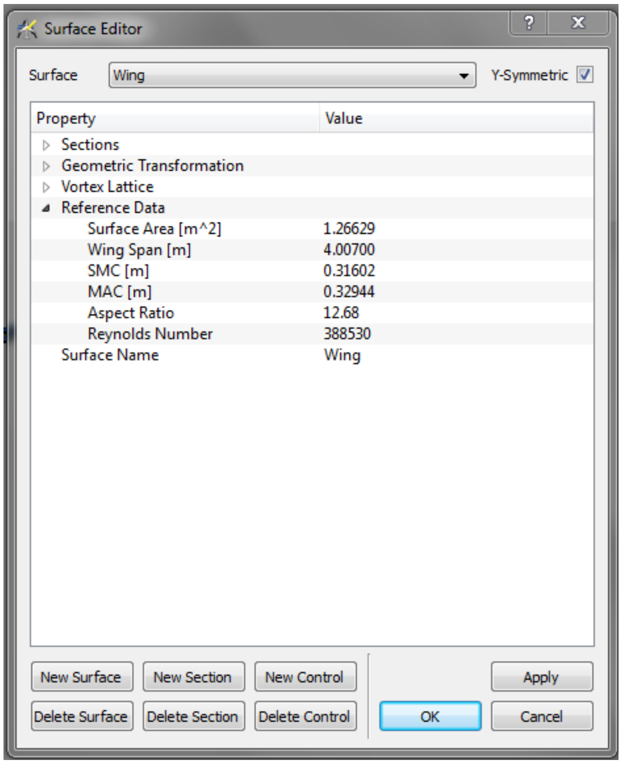
S 1.2663 C 0.3160 B 4.0070

Drag Reference Data

Cruise Velocity [m/s] 17.87

Default CDp Value 0.0200

The figure above depicts the aircraft data that was set for Noctua B1. The aircraft was named “Noctua B1”. The center of gravity was manually specified using the results of the Excel Component Modeling. The center of gravity location was measured from (0,0,0) at the wing tip just as the geometry of the aircraft was. The units in the aircraft editor are in meters.



Surface: Wing
Y-Symmetric ☑

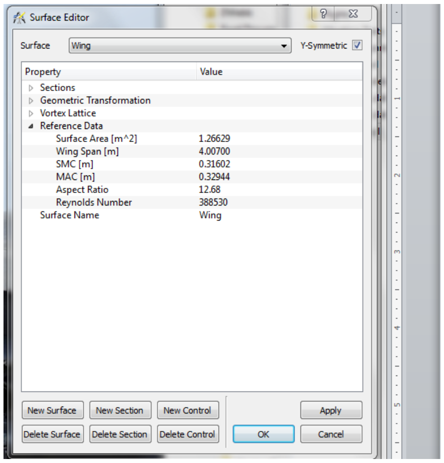
Property	Value
▷ Sections	
▷ Geometric Transformation	
▷ Vortex Lattice	
◢ Reference Data	
Surface Area [m^2]	1.26629
Wing Span [m]	4.00700
SMC [m]	0.31602
MAC [m]	0.32944
Aspect Ratio	12.68
Reynolds Number	388530
Surface Name	Wing

The automatically generated reference dimensions were acceptable. As shown in the figure above the reference dimensions matched those of the Wing which is what the reference dimensions are supposed to be. The cruise velocity for Noctua was designed to be 40 mph (17.9 m/s), so that was the cruise velocity that was entered in the aircraft data. Similarly the design models of Noctua predicted a profile drag of about 0.02 which was also used for the aircraft data.

1.3.7. Xfoil Analysis

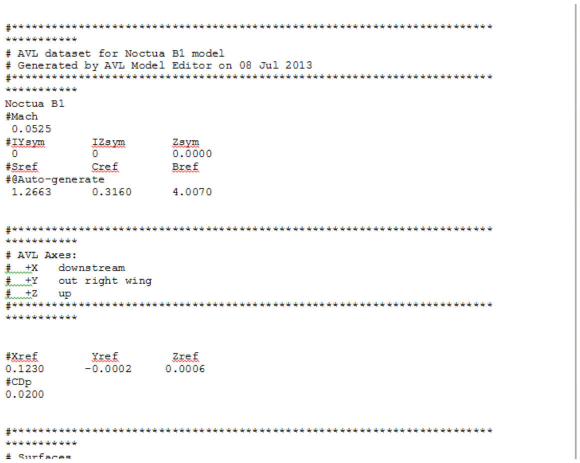
1.3.7.1. Wing

The Eppler 561 airfoil was loaded into Xfoil from the airfoil file that the AVLEditor model used. It initially had only 61 panel nodes so the number of nodes was increased to 160 using the “pane” command in Xfoil.



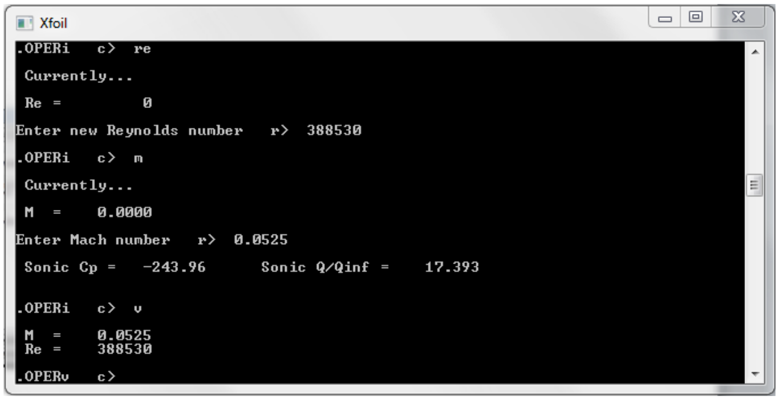
Surface: Wing Y-Symmetric ☑

Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Area [m^2]	1.26629
Wing Span [m]	4.00700
SMC [m]	0.31602
MAC [m]	0.32944
Aspect Ratio	12.68
Reynolds Number	388530
Surface Name	Wing



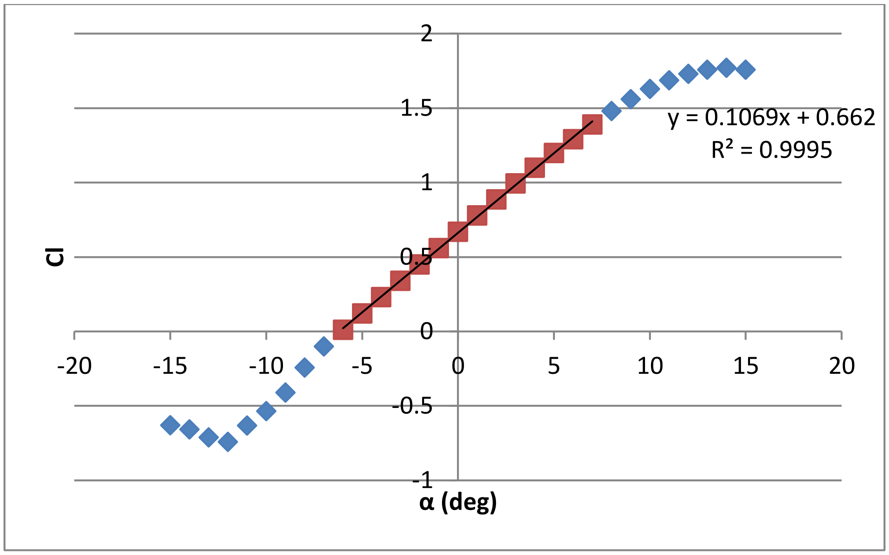
```
#*********************************************************************
***********
# AVL dataset for Noctua B1 model
# Generated by AVL Model Editor on 08 Jul 2013
#*********************************************************************
***********
Noctua B1
#Mach
 0.0525
#IYsym        IZsym        Zsym
 0            0            0.0000
#Sref         Cref         Bref
#@Auto-generate
 1.2663       0.3160       4.0070


#*********************************************************************
***********
# AVL Axes:
#   +X    downstream
#   +Y    out right wing
#   +Z    up
#*********************************************************************
***********


#Xref         Yref         Zref
0.1230       -0.0002      0.0006
#CDp
0.0200


#*********************************************************************
***********
# Surfaces
```

The reynolds and mach numbers were set according to their values from the Surface Editor and the avl model text file. The reynolds number of the wing was 388530, and the mach number was 0.0525.



```
.OPERi   c>  re

 Currently...

 Re  =         0

Enter new Reynolds number   r>  388530

.OPERi   c>  m

 Currently...

 M   =    0.0000

Enter Mach number   r>  0.0525

 Sonic Cp =   -243.96       Sonic Q/Qinf =    17.393


.OPERi   c>  v

 M   =    0.0525
 Re  =    388530

.OPERv   c>
```

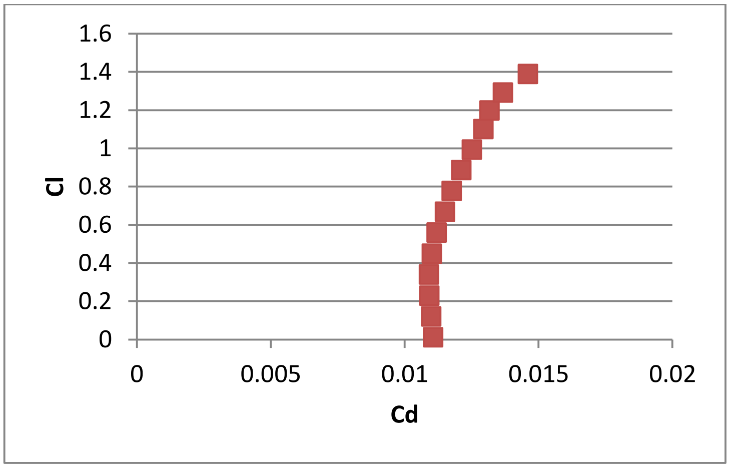
The airfoil was run at -15 to 15 degrees angle of attack at an increment of 1 degree. Initially some of the iterations did not converge so the analysis was run a couple times until all of the iterations converged.

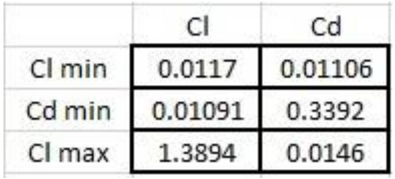


The linear portion of the Cl alpha curve was chosen as between -6 and 7 degrees. The slope was 0.1069 Cl/deg. CLAF was calculated to be 0.9749. The calculation is shown below.

$$C_{l_{\propto}} = 0.1069\,/\deg * \frac{180}{\pi} = 6.1255\,/rad$$

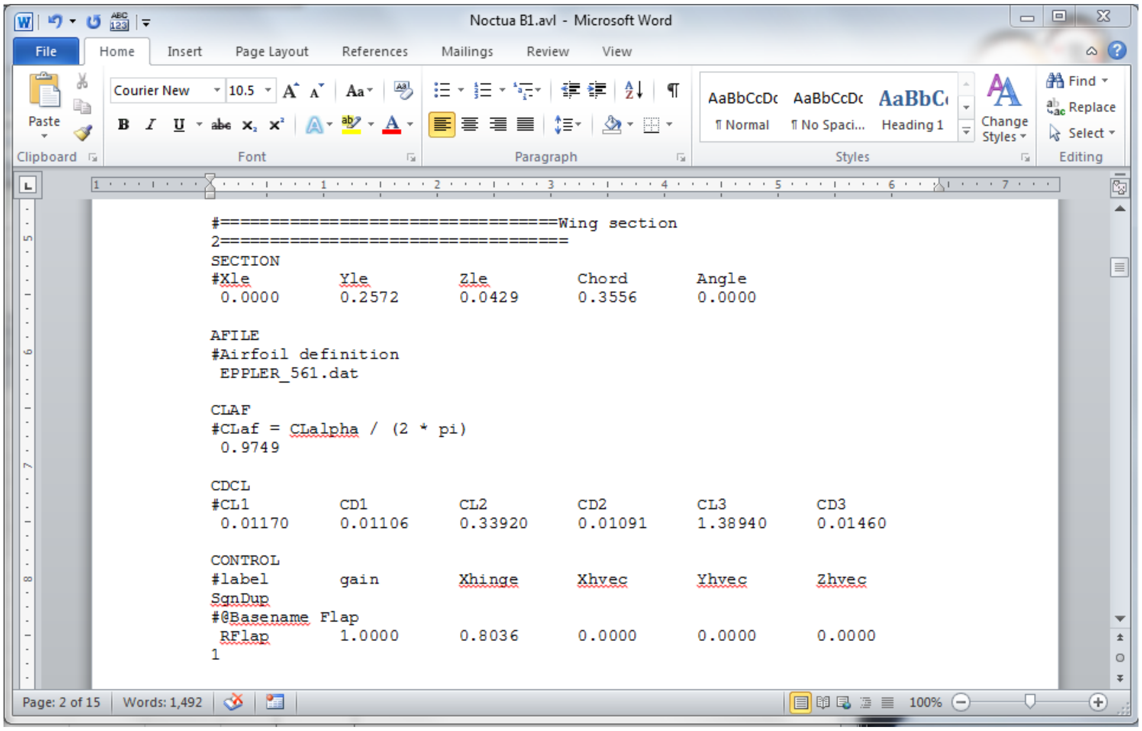
$$CLAF = \frac{C_{l_{\propto}}}{2\pi} = \frac{6.1255}{2\pi} = 0.9749$$





	Cl	Cd
Cl min	0.0117	0.01106
Cd min	0.01091	0.3392
Cl max	1.3894	0.0146

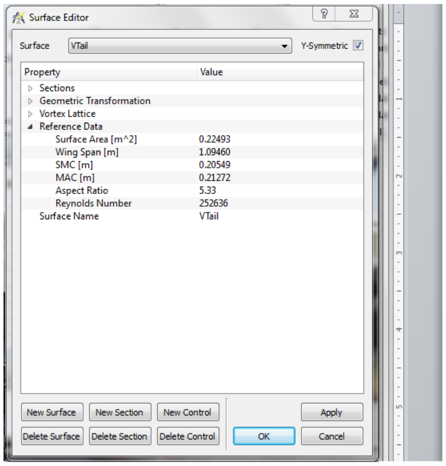
The clcd points were chosen where ClCd1 was Cl min, ClCd2 was Cd min, and ClCd3 was Cl max inside the linear Cl alpha range. All of the values were added to the avl model text file in each section of the wing surface. The figure below depicts one section of the wing.



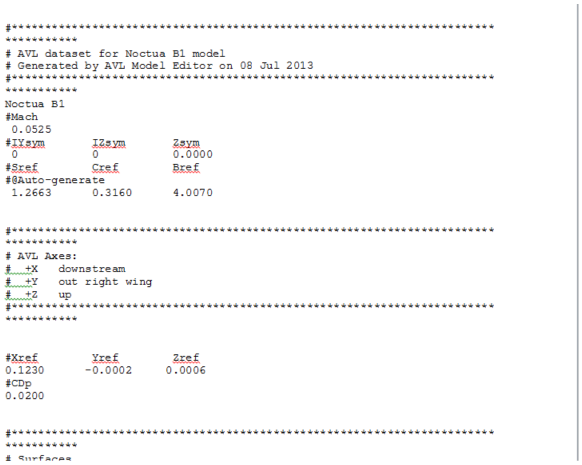
```
#=================================Wing section
2==================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0000      0.2572      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1         CD1         CL2         CD2         CL3         CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460

CONTROL
#label       gain        Xhinge      Xhvec       Yhvec       Zhvec
SgnDup
#@Basename Flap
 RFlap       1.0000      0.8036      0.0000      0.0000      0.0000
1
```

1.3.7.2. V-Tail

The NACA 0012 airfoil was loaded in Xfoil. It was generated by Xfoil so the number of panel nodes was 160 when it was loaded.

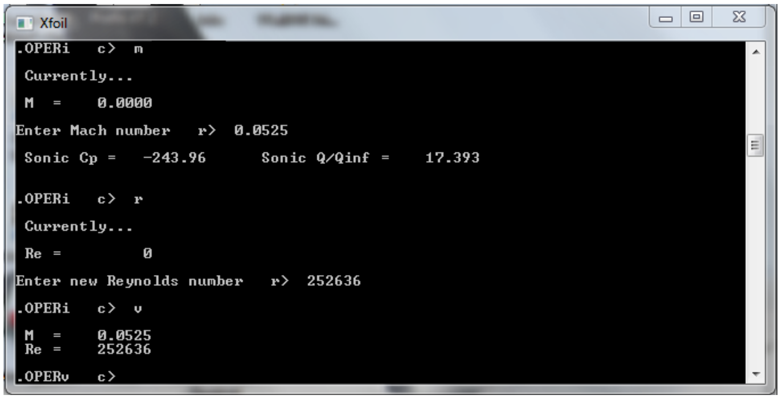


Property	Value
Sections	
Geometric Transformation	
Vortex Lattice	
Reference Data	
Surface Area [m^2]	0.22493
Wing Span [m]	1.09460
SMC [m]	0.20549
MAC [m]	0.21272
Aspect Ratio	5.33
Reynolds Number	252636
Surface Name	VTail



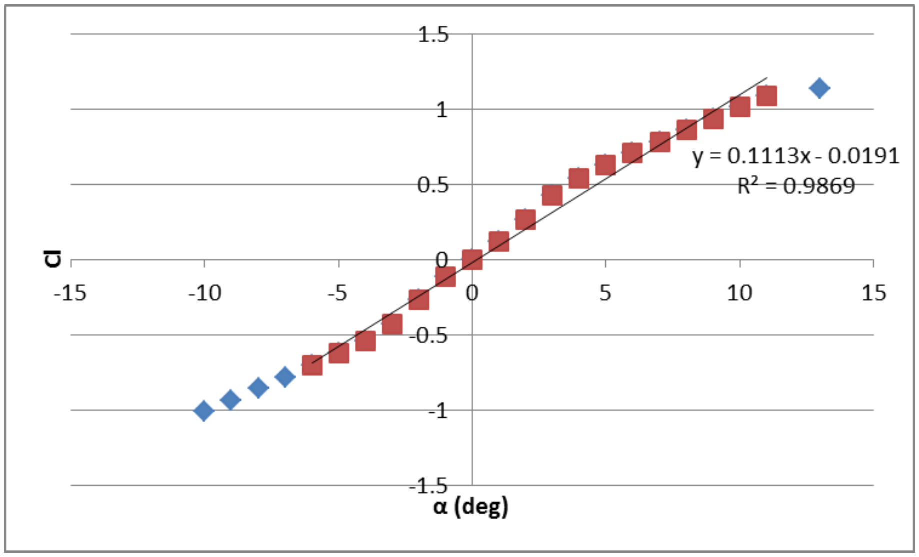
```
#*********************************************************************
***********
# AVL dataset for Noctua B1 model
# Generated by AVL Model Editor on 08 Jul 2013
#*********************************************************************
***********
Noctua B1
#Mach
 0.0525
#IYsym       IZsym       Zsym
 0           0           0.0000
#Sref        Cref        Bref
#@Auto-generate
 1.2663      0.3160      4.0070


#*********************************************************************
***********
# AVL Axes:
#   +X    downstream
#   +Y    out right wing
#   +Z    up
#*********************************************************************
***********


#Xref        Yref        Zref
0.1230      -0.0002     0.0006
#CDp
0.0200


#*********************************************************************
***********
# Surfaces
```

The reynolds and mach numbers were set according to their values from the Surface Editor and the avl model text file. The reynolds number of the tail was 252636, and the mach number was 0.0525.



```
.OPERi   c>  m
 Currently...
 M  =     0.0000
Enter Mach number    r>  0.0525
 Sonic Cp =    -243.96       Sonic Q/Qinf =    17.393

.OPERi   c>  r
 Currently...
 Re =          0
Enter new Reynolds number    r>  252636
.OPERi   c>  v
 M  =     0.0525
 Re =     252636
.OPERv   c>
```

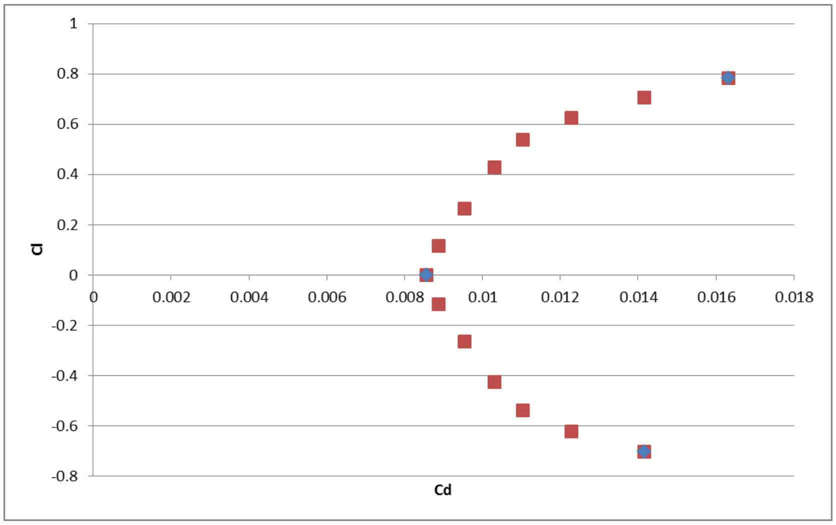
The airfoil was run at -10 to 13 degrees angle of attack at an increment of 1 degree.



The linear portion of the Cl alpha curve was chosen as between -6 and 11 degrees. The sudden jump in Cl at 3 degrees caused the slope to increase significantly if the range ended at 7 degrees. The range was extended so that the slope would more accurately represent the curve. The slope was 0.1113 Cl/deg. CLAF was calculated to be 1.0153. The calculation is shown below.

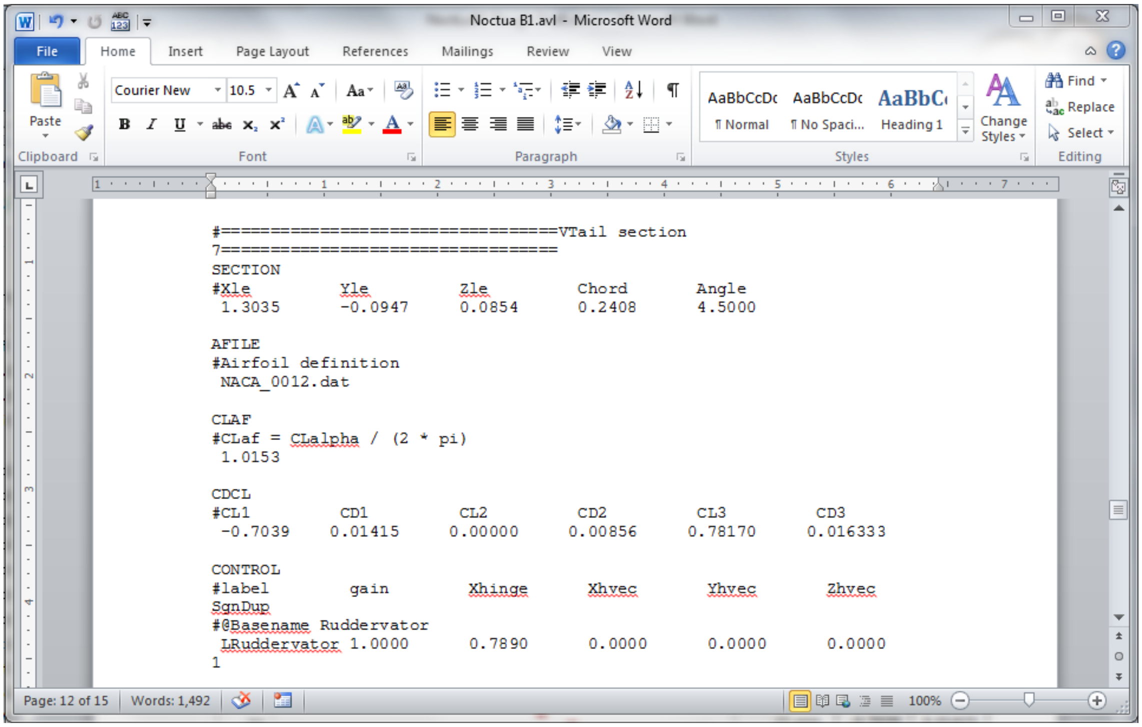
$$C_{l_{\propto}} = 0.1113\ /\ \deg * \frac{180}{\pi} = 6.3796\ /rad$$

$$CLAF = \frac{C_{l_{\propto}}}{2\pi} = \frac{6.3796}{2\pi} = 1.0153$$



	Cl	Cd
Cl min	-0.7039	0.01415
Cd min	0	0.00856
Cl max	0.7817	0.01633

The clcd points were chosen where ClCd1 was Cl min, ClCd2 was Cd min, and ClCd3 was Cl max from the range of -6 to 7 degrees angle of attack. The range ended at 7 degrees because there was a decent amount of flow separation at 7 degrees and a significant amount of flow separation at 11 degrees. All of the values were added to the avl model text file in each section of the vtail surface. The figure below depicts one section of the tail where CLAF and the CDCL data was input.



```
#================================VTail section
7=================================
SECTION
#Xle          Yle          Zle          Chord        Angle
 1.3035       -0.0947      0.0854       0.2408       4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1          CD1          CL2          CD2          CL3          CD3
 -0.7039     0.01415      0.00000      0.00856      0.78170      0.016333

CONTROL
#label         gain         Xhinge       Xhvec        Yhvec        Zhvec
SgnDup
#@Basename Ruddervator
 LRuddervator 1.0000       0.7890       0.0000       0.0000       0.0000
1
```

1.3.8. AVL Aircraft File

```
#************************************************************************
************
# AVL dataset for Noctua B1 model
# Generated by AVL Model Editor on 13 Feb 2014
#************************************************************************
************
Noctua B1
#Mach
 0.0525
#IYsym     IZsym     Zsym
 0          0          0.0000
#Sref      Cref      Bref
#@Auto-generate
 1.2663     0.3160     4.0070


#************************************************************************
************
# AVL Axes:
#  +X   downstream
#  +Y   out right wing
#  +Z   up
#************************************************************************
************
```

```
#Xref        Yref        Zref
0.1230      -0.0002     0.0006
#CDp
0.0200


#************************************************************************
************
# Surfaces
#************************************************************************
************


#========================================Wing====================
====================
#@Yduplicate 7 0.00000 Wing
SURFACE
RWing
#Nchord      Cspace      Nspan       Sspace
 14          1.0000      30          -2.0000

SCALE
#sX          sY          sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX          dY          dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 5.0000

INDEX
#Lsurf
 1


#===================================Wing section
1===================================
SECTION
#Xle         Yle         Zle         Chord       Angle
 0.0000      0.0000      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
```

```
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460


#================================Wing section
2=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000      0.2572      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460

CONTROL
#label      gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Flap
 RFlap       1.0000      0.8036      0.0000      0.0000      0.0000      1


#================================Wing section
3=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000      1.0065      0.0429      0.3556      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460

CONTROL
#label      gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
```

```
#@Basename Flap
 RFlap      1.0000     0.8036     0.0000     0.0000     0.0000     1


#==================================Wing section
4==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     1.0890     0.0429     0.3556     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460


#==================================Wing section
5==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     1.1398     0.0429     0.3454     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460

CONTROL
#label      gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Aileron
 RAileron   1.0000     0.7978     0.0000     0.0000     0.0000     1


#==================================Wing section
6==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
```

```
 0.0361     1.8879     0.0429     0.2052     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460

CONTROL
#label     gain      Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Aileron
 RAileron   1.0000     0.6595     0.0000     0.0000     0.0000     1


#=================================Wing section
7==================================
SECTION
#Xle       Yle       Zle       Chord      Angle
 0.0413     2.0035     0.0429     0.1778     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0033

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460


#==================================Wing
(mirror)==================================
#@Ignore
SURFACE
LWing
#Nchord     Cspace     Nspan     Sspace
 14        1.0000     30        2.0000

SCALE
#sX        sY        sZ
 1.0000     1.0000     1.0000
```

```
TRANSLATE
#dX         dY         dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 5.0000

INDEX
#Lsurf
 1


#==================================Wing section
8==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0413      -2.0035     0.0429      0.1778      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0033

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460


#==================================Wing section
9==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0361      -1.8879     0.0429      0.2052      0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170     0.01106     0.33920     0.01091     1.38940     0.01460

CONTROL
```

```
#label      gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Aileron
 LAileron    1.0000     0.6595     0.0000     0.0000     0.0000     1


#=================================Wing section
10=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     -1.1398    0.0429     0.3454     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460

CONTROL
#label      gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Aileron
 LAileron    1.0000     0.7978     0.0000     0.0000     0.0000     1


#=================================Wing section
11=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     -1.0890    0.0429     0.3556     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460


#=================================Wing section
12=================================
SECTION
```

```
#Xle        Yle        Zle        Chord      Angle
 0.0000     -1.0065    0.0429     0.3556     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460

CONTROL
#label     gain      Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Flap
 LFlap      1.0000     0.8036     0.0000     0.0000     0.0000     1


#=================================Wing section
13=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     -0.2572    0.0429     0.3556     0.0000

AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 0.01170    0.01106    0.33920    0.01091    1.38940    0.01460

CONTROL
#label     gain      Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Flap
 LFlap      1.0000     0.8036     0.0000     0.0000     0.0000     1


#=================================Wing section
14=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 0.0000     0.0000     0.0429     0.3556     0.0000
```

```
AFILE
#Airfoil definition
 EPPLER_561.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 0.9749

CDCL
#CL1      CD1      CL2      CD2      CL3      CD3
 0.01170   0.01106   0.33920   0.01091   1.38940   0.01460


#=====================================VTail==================
===================
#@Yduplicate 4 0.00000 VTail
SURFACE
RVTail
#Nchord    Cspace    Nspan    Sspace
 12        1.0000    32       1.0000

SCALE
#sX       sY       sZ
 1.0000    1.0000    1.0000

TRANSLATE
#dX       dY       dZ
 0.0000    0.0000    0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 2


#================================VTail section
1================================
SECTION
#Xle      Yle      Zle      Chord     Angle
 1.2954    0.0000    0.0190    0.2595    4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153
```

```
CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413     0.00000     0.00856     0.78170     0.01633


#=================================VTail section
2=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.3035      0.0947      0.0854      0.2408      4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413     0.00000     0.00856     0.78170     0.01633

CONTROL
#label       gain       Xhinge     Xhvec       Yhvec       Zhvec       SgnDup
#@Basename Ruddervator
 RRuddervator 1.0000      0.7890      0.0000      0.0000      0.0000      1


#=================================VTail section
3=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.3320      0.4276      0.3184      0.1751      4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413     0.00000     0.00856     0.78170     0.01633

CONTROL
#label       gain       Xhinge     Xhvec       Yhvec       Zhvec       SgnDup
#@Basename Ruddervator
```

```
 RRuddervator 1.0000      0.7090      0.0000      0.0000      0.0000      1


#==================================VTail section
4==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.3422      0.5473      0.4022      0.1515      4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413     0.00000     0.00856     0.78170     0.01633


#==================================VTail
(mirror)==================================
#@Ignore
SURFACE
LVTail
#Nchord     Cspace      Nspan       Sspace
 12          1.0000      32          -1.0000

SCALE
#sX         sY         sZ
 1.0000      1.0000      1.0000

TRANSLATE
#dX         dY         dZ
 0.0000      0.0000      0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 2


#==================================VTail section
5==================================
SECTION
#Xle        Yle        Zle        Chord      Angle
```

```
 1.3422     -0.5473    0.4022     0.1515     4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413    0.00000    0.00856    0.78170    0.01633


#=================================VTail section
6=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.3320     -0.4276    0.3184     0.1751     4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1        CD1        CL2        CD2        CL3        CD3
 -0.70410    0.01413    0.00000    0.00856    0.78170    0.01633

CONTROL
#label       gain       Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Ruddervator
 LRuddervator 1.0000     0.7090     0.0000     0.0000     0.0000     1


#=================================VTail section
7=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.3035     -0.0947    0.0854     0.2408     4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
```

```
 1.0153

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 -0.70410   0.01413    0.00000    0.00856    0.78170    0.01633

CONTROL
#label      gain      Xhinge     Xhvec      Yhvec      Zhvec      SgnDup
#@Basename Ruddervator
 LRuddervator 1.0000     0.7890     0.0000     0.0000     0.0000     1


#=================================VTail section
8=================================
SECTION
#Xle        Yle        Zle        Chord      Angle
 1.2954     0.0000     0.0190     0.2595     4.5000

AFILE
#Airfoil definition
 NACA_0012.dat

CLAF
#CLaf = CLalpha / (2 * pi)
 1.0153

CDCL
#CL1       CD1       CL2       CD2       CL3       CD3
 -0.70410   0.01413    0.00000    0.00856    0.78170    0.01633


#===================================FuselageTop=================
==================
#@Yduplicate 2 0.00000 FuselageTop
SURFACE
RFuselageTop
#Nchord     Cspace     Nspan      Sspace
 12         1.0000     5          -2.0000

SCALE
#sX        sY        sZ
 1.0000     1.0000     1.0000

TRANSLATE
#dX        dY        dZ
 0.0000     0.0000     0.0000

ANGLE
#Ainc
 0.0000
```

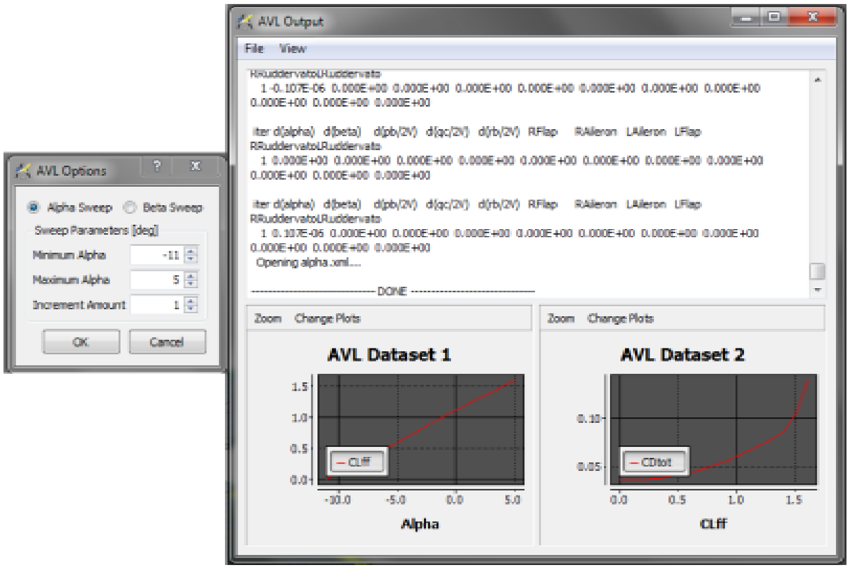
```
INDEX
#Lsurf
 3


#===============================FuselageTop section
1===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.4953    0.0000     0.1100     2.0701     0.0000


#===============================FuselageTop section
2===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.2413    0.1500     0.1100     0.6985     0.0000


#===============================FuselageTop
(mirror)================================
#@Ignore
SURFACE
LFuselageTop
#Nchord     Cspace     Nspan      Sspace
 12         1.0000     5          2.0000

SCALE
#sX         sY         sZ
 1.0000     1.0000     1.0000

TRANSLATE
#dX         dY         dZ
 0.0000     0.0000     0.0000

ANGLE
#Ainc
 0.0000

INDEX
#Lsurf
 3


#===============================FuselageTop section
3===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.2413    -0.1500    0.1100     0.6985     0.0000
```

```
#===============================FuselageTop section
4===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.4953    0.0000     0.1100     2.0701     0.0000


#===================================FuselageSide=================
==================
SURFACE
FuselageSide
#Nchord     Cspace     Nspan      Sspace
 12         1.0000     10         -1.0000

SCALE
#sX         sY         sZ
 1.0000     1.0000     1.0000

TRANSLATE
#dX         dY         dZ
 0.0000     0.0000     0.0000

ANGLE
#Ainc
 0.0000


#==============================FuselageSide section
1===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.2413    0.0000     0.2516     0.6985     0.0000


#==============================FuselageSide section
2===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.4953    0.0000     0.1100     2.0701     0.0000


#==============================FuselageSide section
3===============================
SECTION
#Xle        Yle        Zle        Chord      Angle
 -0.3100    0.0000     -0.0100    1.8848     0.0000


#==============================FuselageSide section
4===============================
SECTION
```

```
#Xle        Yle         Zle         Chord       Angle
 -0.2413     0.0000      -0.0500     0.6985      0.0000
```

1.3.9. Alpha Sweep

The linear range of the wings was determined to be between -6 and 7 degrees; however, the wings were mounted on the aircraft at a 5 degree incidence. Additionally the alpha sweep should run a few degrees above the linear range, somewhere around 10 degrees angle of attack. The range of angle of attack was determined to be between -6 and 10 degrees minus the wing incidence, 5 degrees.



● Alpha Sweep ○ Beta Sweep

Sweep Parameters [deg]

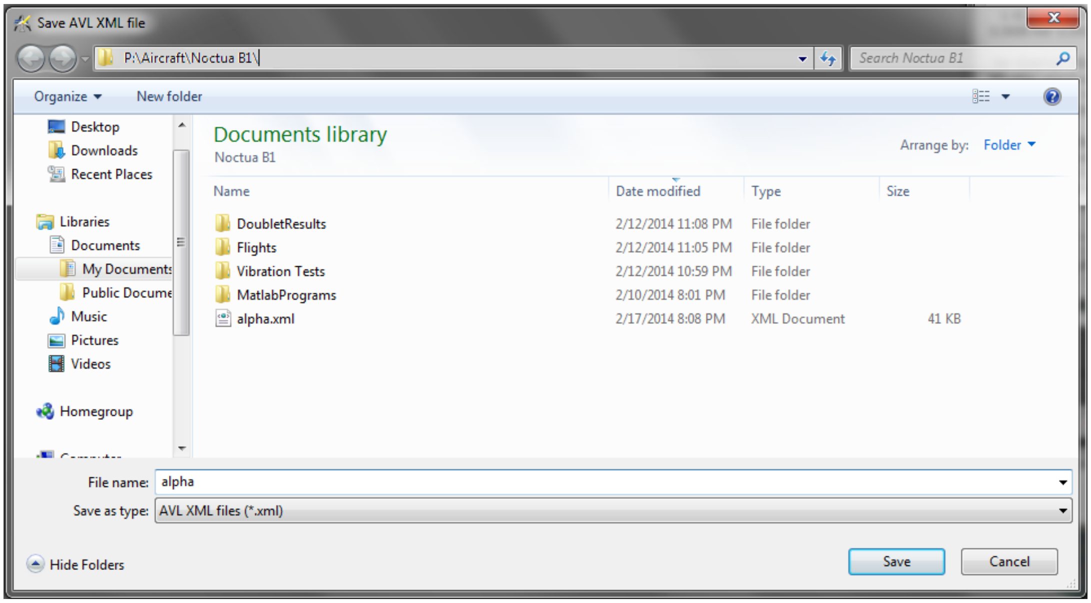
Minimum Alpha: -11

Maximum Alpha: 5

Increment Amount: 1

```
RRuddervatoLRuddervato
  1 -0.107E-06  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00
0.000E+00  0.000E+00  0.000E+00

 iter d(alpha)  d(beta)   d(pb/2V)  d(qc/2V)  d(rb/2V)  RFlap    RAileron  LAileron  LFlap
RRuddervatoLRuddervato
  1  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00
0.000E+00  0.000E+00  0.000E+00

 iter d(alpha)  d(beta)   d(pb/2V)  d(qc/2V)  d(rb/2V)  RFlap    RAileron  LAileron  LFlap
RRuddervatoLRuddervato
  1  0.107E-06  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00  0.000E+00
0.000E+00  0.000E+00  0.000E+00
 Opening alpha.xml....

------------------------------ DONE ------------------------------
```

The alpha sweep was run from -11 degrees to 5 degrees at an increment of 1 degree.



P:\Aircraft\Noctua B1\

Name	Date modified	Type	Size
DoubletResults	2/12/2014 11:08 PM	File folder	
Flights	2/12/2014 11:05 PM	File folder	
Vibration Tests	2/12/2014 10:59 PM	File folder	
MatlabPrograms	2/10/2014 8:01 PM	File folder	
alpha.xml	2/17/2014 8:08 PM	XML Document	41 KB

File name: alpha

Save as type: AVL XML files (*.xml)

The alpha file was saved as “alpha.xml” in the Noctua B1 aircraft folder.

1.3.10. Revisions

1.4. Propeller File

Noctua was designed to use a range of different propellers. At the time that the autopilot was installed Noctua B1 was operating with a 3- blade Mejzlik 22x12 inch propeller. Prior to installing the piccolo tests had been run on the Mejzlik 22x12 to experimentally determine the propeller’s performance. The results were used to create the propeller file for the simulator model. The results were copied into an empty notepad file. The data began with an advance ratio (J) of -1.00 and iterated, by 0.05, to an advance ratio of 2.0. The experimental data did not include efficiency, so efficiency was excluded even though it could have been calculated. The simulator only needed the advance ratio (J), coefficient of power (Cp), and coefficient of thrust (Ct).

#	Prop: Mejzlik 22x12 3-Blade
#	RPM 4500

#	Experimental Data Reduction	
#J	Cp	Ct
-1	-0.0343	0.1403
-0.95	-0.02701825	0.145009
-0.9	-0.020083	0.149186
-0.85	-0.01349425	0.152831
-0.8	-0.007252	0.155944
-0.75	-0.00135625	0.158525
-0.7	0.004193	0.160574
-0.65	0.00939575	0.162091
-0.6	0.014252	0.163076
-0.55	0.01876175	0.163529
-0.5	0.022925	0.16345
-0.45	0.02674175	0.162839
-0.4	0.030212	0.161696
-0.35	0.03333575	0.160021
-0.3	0.036113	0.157814
-0.25	0.03854375	0.155075
-0.2	0.040628	0.151804
-0.15	0.04236575	0.148001
-0.1	0.043757	0.143666
-0.05	0.04480175	0.138799
0	0.0455	0.1334
0.05	0.04585175	0.127469
0.1	0.045857	0.121006
0.15	0.04551575	0.114011
0.2	0.044828	0.106484
0.25	0.04379375	0.098425
0.3	0.042413	0.089834
0.35	0.04068575	0.080711
0.4	0.038612	0.071056
0.45	0.03619175	0.060869
0.5	0.033425	0.05015
0.55	0.03031175	0.038899
0.6	0.026852	0.027116
0.65	0.02304575	0.014801
0.7	0.018893	0.001954
0.75	0.01439375	-0.011425
0.8	0.009548	-0.025336
0.85	0.00435575	-0.039779
0.9	-0.001183	-0.054754
0.95	-0.00706825	-0.070261

1	-0.0133	-0.0863
1.05	-0.01987825	-0.102871
1.1	-0.026803	-0.119974
1.15	-0.03407425	-0.137609
1.2	-0.041692	-0.155776
1.25	-0.04965625	-0.174475
1.3	-0.057967	-0.193706
1.35	-0.06662425	-0.213469
1.4	-0.075628	-0.233764
1.45	-0.08497825	-0.254591
1.5	-0.094675	-0.27595
1.55	-0.10471825	-0.297841
1.6	-0.115108	-0.320264
1.65	-0.12584425	-0.343219
1.7	-0.136927	-0.366706
1.75	-0.14835625	-0.390725
1.8	-0.160132	-0.415276
1.85	-0.17225425	-0.440359
1.9	-0.184723	-0.465974
1.95	-0.19753825	-0.492121
2	-0.2107	-0.5188

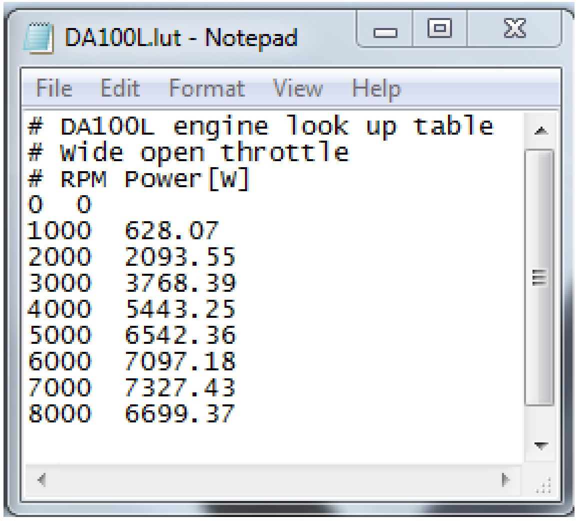
1.5. Motor File

Noctua B1 used a DA100, 2 – cylinder, gasoline engine. The only power curve that was available for the DA100 was table generated by a flying simulator program “FlightSim.” The power table seemed unrealistically high compared to what was observed in ground tests. According to the FlightSim power table at 6000 RPM the motor would output 7097.18W. When the throttle was calibrated, with the aircraft running on a static stand, the maximum rpm obtained was 6000. I used the following propeller performance equation, combined with the properller data at static (J = 0), to calculate the amount of power that was required to spin the Mejzlik 22x12 at 6000 rpm.

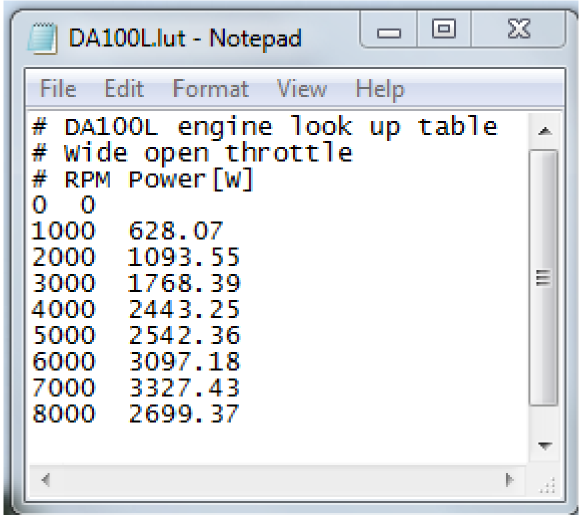
$$P = C_p \rho n^3 D^5$$

$$where\ n = rpm * 1/60$$

The power required was only 2900.5W. Since the simulator used the motor file mainly to determine the maximum rpm and estimate the fuel burn I artificially lowered all of the power data in the motor file until the simulator correctly estimated an rpm of 6000 at full throttle.



```
# DA100L engine look up table
# Wide open throttle
# RPM Power[W]
0   0
1000   628.07
2000   2093.55
3000   3768.39
4000   5443.25
5000   6542.36
6000   7097.18
7000   7327.43
8000   6699.37
```

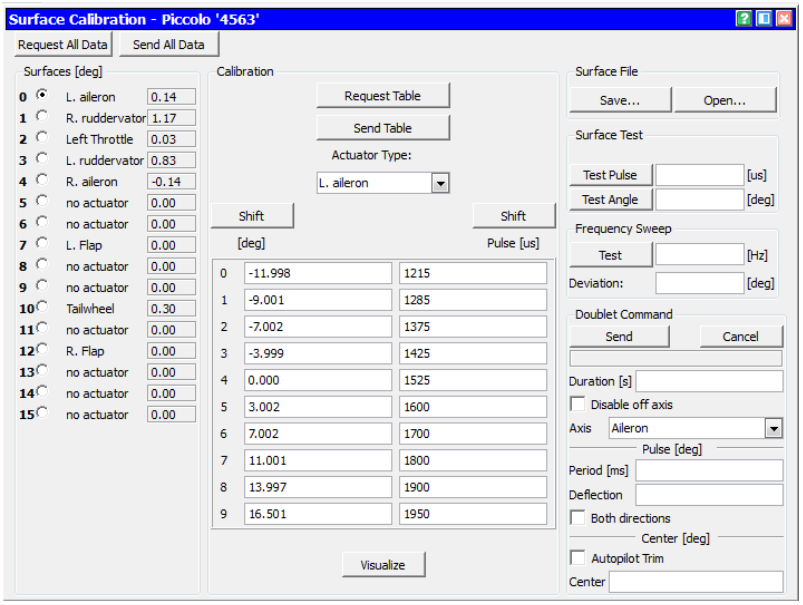


```
# DA100L engine look up table
# Wide open throttle
# RPM Power[W]
0   0
1000   628.07
2000   1093.55
3000   1768.39
4000   2443.25
5000   2542.36
6000   3097.18
7000   3327.43
8000   2699.37
```

The figure above depicts the final motor file that was used for simulations. The power data for 6000 rpm was lowered by 1000W each iteration until full throttle would yield 6000 rpm. The final power setting for 6000 rpm was 3097.18W, 4000W lower than the original value. The power settings at 7000 and 8000 rpm were lowered 4000W as well. The data for 7000 and 8000 rpm was left in the file because the motor file had to have a "knee" in the power curve where the power produced would begin to decrease with an increase in rpm. The power data below 6000 rpm was lowered by 4000, 3000, 2000, and then 1000 watts.

1.6. Control Surfaces Calibration

The control surface deflections were calibrated by commanding test pulse widths to the piccolo and measuring the corresponding surface deflections on the aircraft.


Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	●	L. aileron	0.14
1	○	R. ruddervator	1.17
2	○	Left Throttle	0.03
3	○	L. ruddervator	0.83
4	○	R. aileron	-0.14
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.30
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Surface File

Surface Test

Test Pulse [us]

Test Angle [deg]

Frequency Sweep

Test [Hz]

Deviation: [deg]

Doublet Command

Duration [s]

☐ Disable off axis

Axis Aileron

Pulse [deg]

Period [ms]

Deflection

☐ Both directions

Center [deg]

☐ Autopilot Trim

Center

Specific pulse widths were commanded to the control surfaces with the "Surface Test" function that is available in the Surface Calibration window in PCC.

The control surface deflections of the ailerons, flaps, and ruddervators were measured using a digital Angle Pro. The tailwheel deflections were measured by setting the tailwheel on a secured piece of paper and tracing a straight line on the right outside edge of the tailwheel and calculating the angle between each line and the line at 0 degrees (centered) tailwheel deflection.

The following steps outline how each control surface was measured, except for the throttle.

1) Determined the pulse width for maximum down deflection.

2) Determined the pulse width for maximum up deflection.

3) Determine the pulse width for zero deflection.

4) Attempted to create an even distribution of change in deflection angles to fill in the empty points between 0 and the maximum and minimum deflections.

The throttle calibration was slightly different since throttle is calibrated using rpm as an indicator of power. The first throttle setting that was found was minimum throttle. The motor was a gas engine and at some point a low enough pulse width would cause the motor to die. Since the throttle setting that will cause the engine to die can change depending on varying conditions in air and temperature it was decided to simply set the minimum throttle at a safe rpm instead of attempting to find and set throttle 0. The minimum throttle was determined to be 1800 rpm.

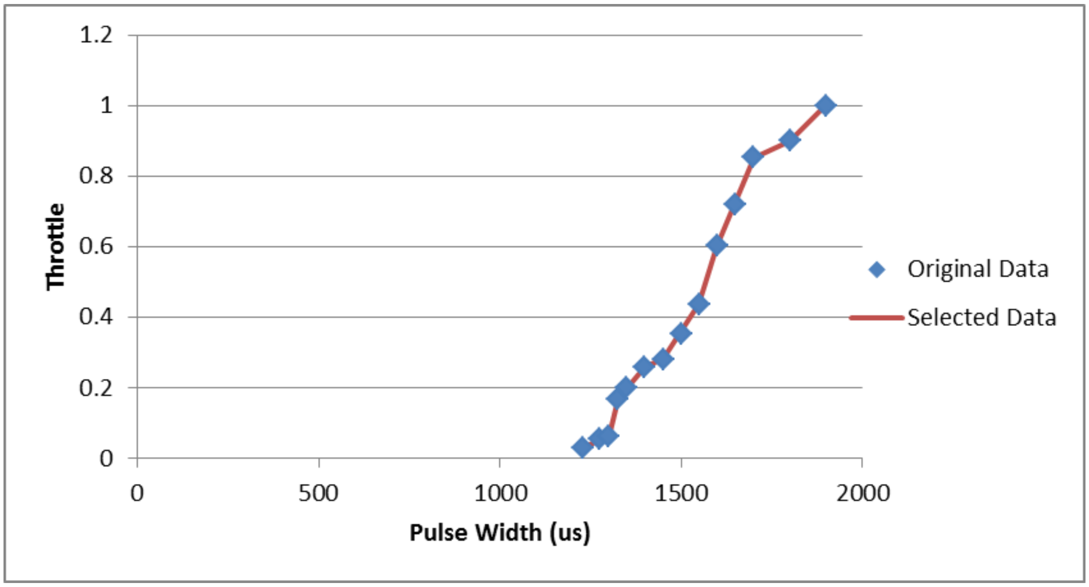
The rpm was measured using the rpm reading that the piccolo received from the external deadman tach board. Once the pulse width was determined for 1800 rpm the pulse width was slowly increased, at a rate of 50 and 100 us, and the rpm recorded until the maximum rpm was reached. It was already known what the expected maximum rpm should be, so the corresponding pulse width with the maximum rpm was treated as full throttle.

pusle width	RPM	RPM^3	Throttle
1230	1800	5832000000	0.029891
1275	2200	1.0648E+10	0.054574
1300	2300	1.2167E+10	0.062359
1325	3200	3.2768E+10	0.167945
1350	3400	3.9304E+10	0.201443
1400	3700	5.0653E+10	0.25961
1450	3800	5.4872E+10	0.281233
1500	4100	6.8921E+10	0.353238
1550	4400	8.5184E+10	0.43659
1600	4900	1.1765E+11	0.602982
1650	5200	1.4061E+11	0.720653
1700	5500	1.6638E+11	0.852715
1800	5600	1.7562E+11	0.900078
1900	5800	1.9511E+11	1

The table above depicts the results of the throttle calibration. The throttle settings were determined using rpm cubed since rpm cubed is proportional to power; thus, percent throttle was equivalent to percent power.

$$Throttle = \frac{RPM^3}{RPM_{max}^3}$$

In total there were 14 points and only 10 could be used. The lowest and highest throttle settings were used and the points in between were selected to best fit the variation of pulse width and rpm that occurred in between maximum and minimum throttle.

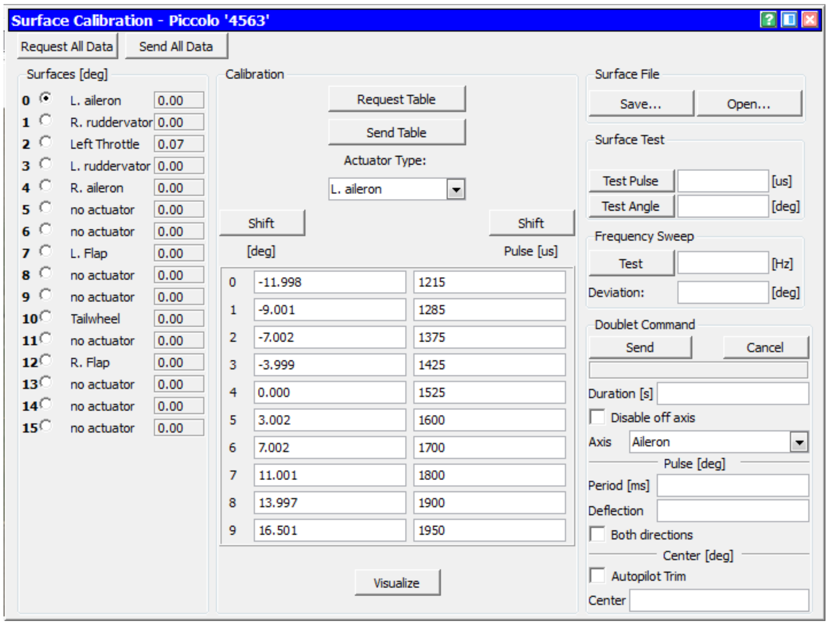


The figure above is a plot that was created to assist in selecting which interior data points to use. The interior data was selected so that the red line in the figure would intersect as many of the original, blue, data points as possible.

pulse width	Throttle
1230	0.029891
1300	0.062359
1325	0.167945
1400	0.25961
1450	0.281233
1550	0.43659
1600	0.602982
1700	0.852715
1800	0.900078
1900	1

The table above contains the selected data points for throttle calibration. There were more points at the low and high end of throttle since there are much higher non linearity’s in the extreme throttle settings.

1.6.1. LAileron



Surface Calibration - Piccolo '4563'

Surfaces [deg]

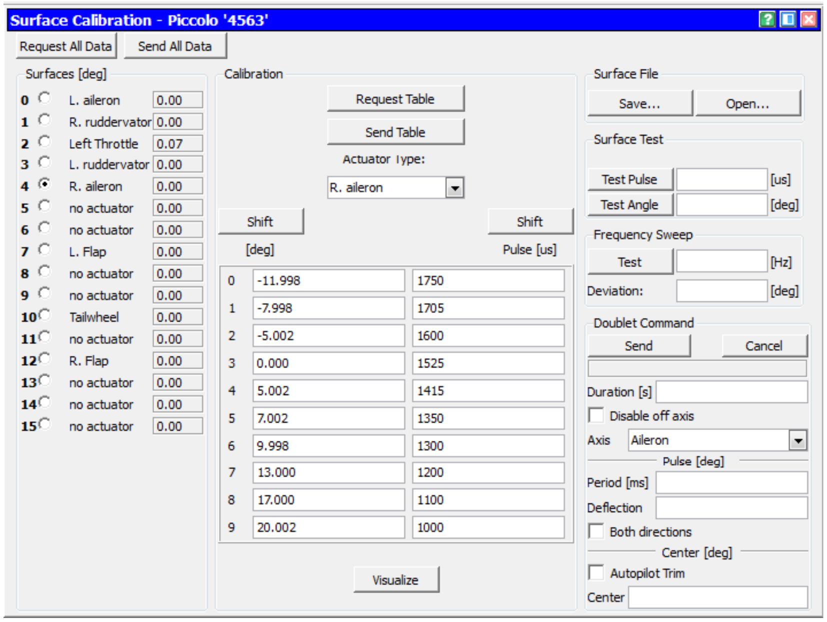
0	●	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: L. aileron

	[deg]	Pulse [us]
0	-11.998	1215
1	-9.001	1285
2	-7.002	1375
3	-3.999	1425
4	0.000	1525
5	3.002	1600
6	7.002	1700
7	11.001	1800
8	13.997	1900
9	16.501	1950

Test Pulse [us]
Test Angle [deg]
Test [Hz]
Deviation: [deg]
Duration [s]
□ Disable off axis
Axis Aileron
Pulse [deg]
Period [ms]
Deflection
□ Both directions
Center [deg]
□ Autopilot Trim
Center

1.6.2. RAileron



Surface Calibration - Piccolo '4563'

Surfaces [deg]

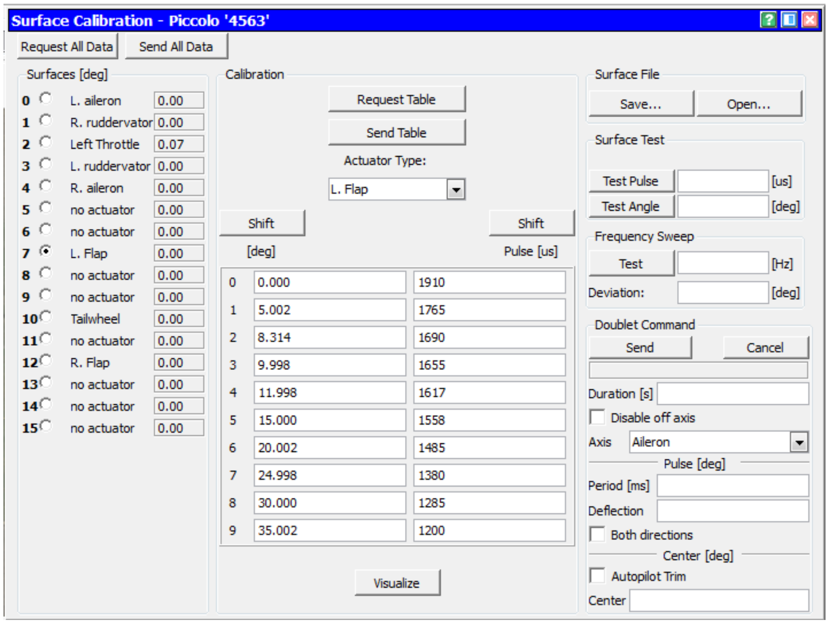
0	○	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	●	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: R. aileron

	[deg]	Pulse [us]
0	-11.998	1750
1	-7.998	1705
2	-5.002	1600
3	0.000	1525
4	5.002	1415
5	7.002	1350
6	9.998	1300
7	13.000	1200
8	17.000	1100
9	20.002	1000

Test Pulse [us]
Test Angle [deg]
Test [Hz]
Deviation: [deg]
Duration [s]
□ Disable off axis
Axis Aileron
Pulse [deg]
Period [ms]
Deflection
□ Both directions
Center [deg]
□ Autopilot Trim
Center

1.6.3. LFlap



Surface Calibration - Piccolo '4563'

Surfaces [deg]

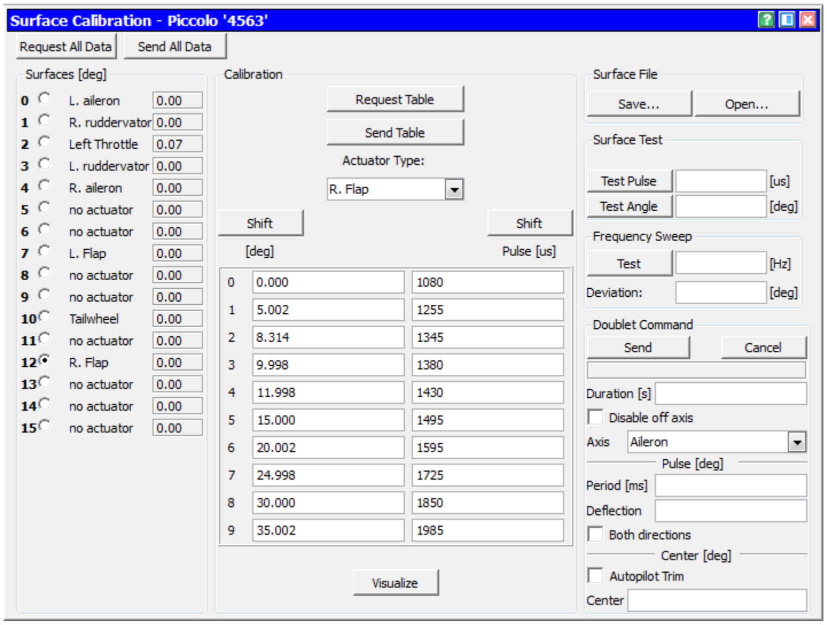
0	○	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	●	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: L. Flap

Axis: Aileron

	[deg]	Pulse [us]
0	0.000	1910
1	5.002	1765
2	8.314	1690
3	9.998	1655
4	11.998	1617
5	15.000	1558
6	20.002	1485
7	24.998	1380
8	30.000	1285
9	35.002	1200

1.6.4. RFlap



Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	○	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	●	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

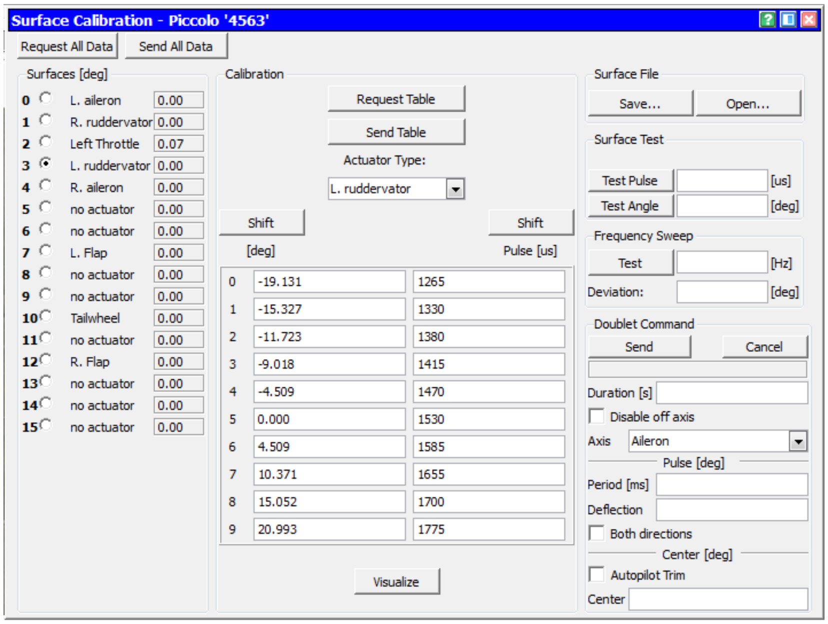
Actuator Type: R. Flap

Axis: Aileron

	[deg]	Pulse [us]
0	0.000	1080
1	5.002	1255
2	8.314	1345
3	9.998	1380
4	11.998	1430
5	15.000	1495
6	20.002	1595
7	24.998	1725
8	30.000	1850
9	35.002	1985

Note that the right flap has to be assigned to one of servos 8-10 for the right flap to be functional in simulations.

1.6.5. LRuddervator



Surfaces [deg]

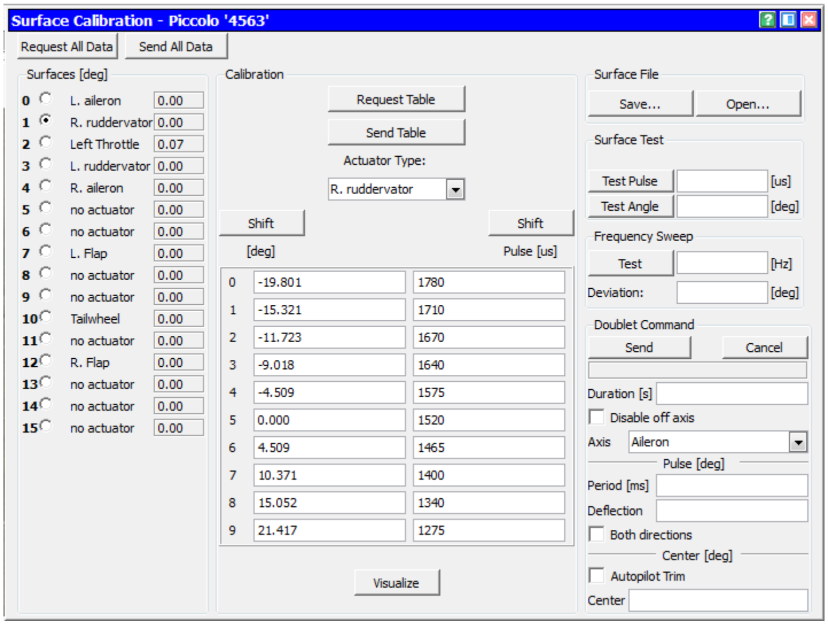
0	○	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	●	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: L. ruddervator

	[deg]	Pulse [us]
0	-19.131	1265
1	-15.327	1330
2	-11.723	1380
3	-9.018	1415
4	-4.509	1470
5	0.000	1530
6	4.509	1585
7	10.371	1655
8	15.052	1700
9	20.993	1775

Axis: Aileron

1.6.6. RRuddervator



Surfaces [deg]

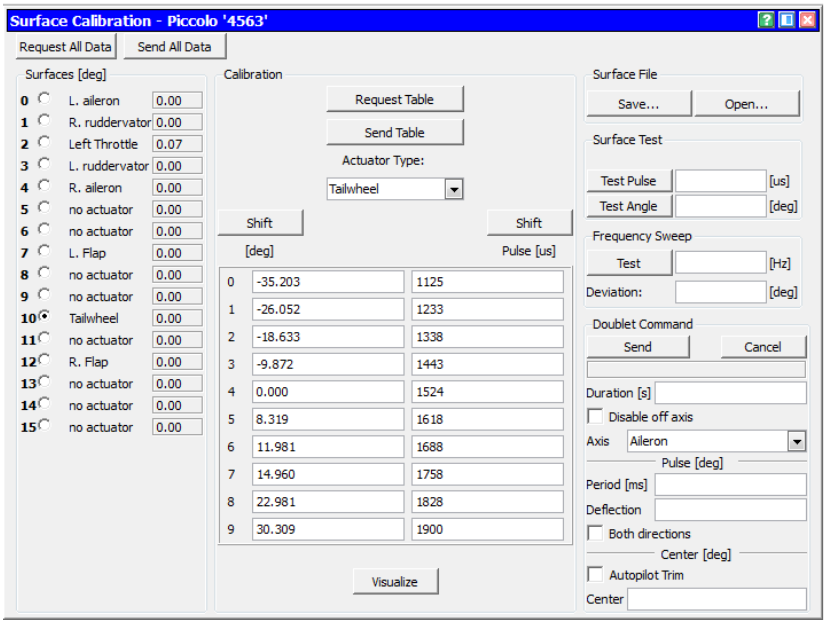
0	○	L. aileron	0.00
1	●	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	○	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Actuator Type: R. ruddervator

	[deg]	Pulse [us]
0	-19.801	1780
1	-15.321	1710
2	-11.723	1670
3	-9.018	1640
4	-4.509	1575
5	0.000	1520
6	4.509	1465
7	10.371	1400
8	15.052	1340
9	21.417	1275

Axis: Aileron

1.6.7. Tailwheel



Surface Calibration - Piccolo '4563'

Surfaces [deg]

0	○	L. aileron	0.00
1	○	R. ruddervator	0.00
2	○	Left Throttle	0.07
3	○	L. ruddervator	0.00
4	○	R. aileron	0.00
5	○	no actuator	0.00
6	○	no actuator	0.00
7	○	L. Flap	0.00
8	○	no actuator	0.00
9	○	no actuator	0.00
10	●	Tailwheel	0.00
11	○	no actuator	0.00
12	○	R. Flap	0.00
13	○	no actuator	0.00
14	○	no actuator	0.00
15	○	no actuator	0.00

Calibration

Actuator Type: Tailwheel

	[deg]	Pulse [us]
0	-35.203	1125
1	-26.052	1233
2	-18.633	1338
3	-9.872	1443
4	0.000	1524
5	8.319	1618
6	11.981	1688
7	14.960	1758
8	22.981	1828
9	30.309	1900

Axis: Aileron

1.7. Simulator File

// Noctua B1// SIMULATOR MODEL

// Oklahoma State University

// Aerodynamics data from AVL

Alpha_sweep_xml_file=alpha.xml

// Mapping of the channel numbers in AVL to channel numbers in the piccolo

// RFlap

// Right flap is really on servo 12, but the simulator wont command flap

// deflections to the right flap if it is on servos 12 - 15

Channel_d1=8

// RAileron

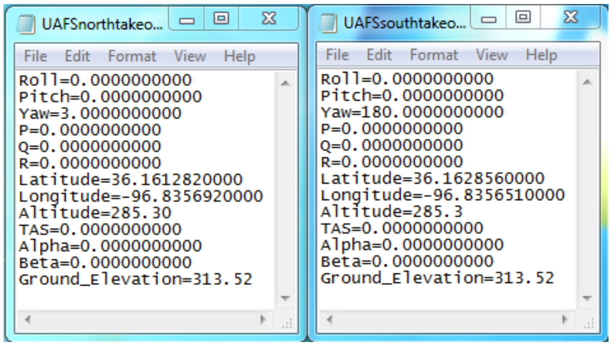
```
Channel_d2=4

// LAileron
Channel_d3=0

// LFlap
Channel_d4=7

// RRuddervator
Channel_d5=1

// LRuddervator
Channel_d6=3

// Servo Response Time
Actuators=StandardActuator.txt


//--------------------  INERTIA  --------------------//

// Gross takeoff mass of the aircraft, in kg
Gross_Mass=32

// Mass of aircraft without fuel, in kg
Empty_Mass=26.6

// Mass Moments of Inertia, in kg*m^2
Roll_Inertia=4.5623
Pitch_Inertia=6.6035
```

```
Yaw_Inertia=10.6141



//--------  Surface Effectiveness Adjustments  --------//


Cmd_scaler_d5=1.05576204

Cmd_scaler_d6=1.05576204

Cnd_scaler_d5=0.62022818

Cnd_scaler_d6=0.62022818



//-------------------- PROPULSION --------------------//


// Gas Motor

Left_Engine_Type=0

Left_Engine_Channel=2

Left_Engine_LUT=DA100L.lut


// Engine-specific fuel consumption

Left_Engine_BSFC=1785


// Propeller is a fixed pitch propeller

Left_Actuator_Type=0



// Propeller diameter, in m

Left_Prop_Diameter=0.5588


// Position of propeller hub wrt to aircraft cg, in m
// Note dimensions are positive if upstream of CG
Left_Prop_X=0.08255
Left_Prop_Y=0.0
Left_Prop_Z=0.02461

//Propeller Angles to Aircraft Frame
Left_Prop_Pan=0
Left_Prop_Tilt=0

//Propeller gear ratio (greater than 1 if rotates slower than the engine)
Left_Prop_GearRatio=1

//Propeller direction of rotation (1 = cw -1 = ccw)
Left_Prop_Sense=1

// Propeller Moment of inertia in kg/m^2
Left_Prop_Inertia=0.005048

// Propeller coefficients look-up table
Left_Prop_LUT=22x12.prd


//---------------  GROUND CONTACT POINTS  ---------------//

//Nose wheel is actually a tail wheel
NoseWheel_Position_X=-0.88011
NoseWheel_Position_Y=0.0
NoseWheel_Position_Z=0.17848
```

```
NoseWheel_RudderWheelRatio=1
NoseWheel_Steering_Channel=10

LeftWheel_Position_X=0.1016
LeftWheel_Position_Y=-0.32753
LeftWheel_Position_Z=0.26738

RightWheel_Position_X=0.1016
RightWheel_Position_Y=0.32753
RightWheel_Position_Z=0.26738

ContactPoint_Nose_Position_X=0.71482
ContactPoint_Nose_Position_Y=0.0
ContactPoint_Nose_Position_Z=-0.012021

ContactPoint_Bottom_Position_X=0.1016
ContactPoint_Bottom_Position_Y=0.0
ContactPoint_Bottom_Position_Z=0.11231

//Contact Point Tail is the point at the bottom of the end of the fuselage
ContactPoint_Tail_Position_X=-1.407541
ContactPoint_Tail_Position_Y=0.0
ContactPoint_Tail_Position_Z=-0.012021

//Wing contact points are the wing tips
ContactPoint_LWing_Position_X=0.12319
ContactPoint_LWing_Position_Y=-2.003
ContactPoint_LWing_Position_Z=0.00068
```

```
ContactPoint_RWing_Position_X=0.12319
ContactPoint_RWing_Position_Y=2.003
ContactPoint_RWing_Position_Z=0.00068

//----------------- AVIONICS MOUNTING -----------------//


// Avionics (IMU sensor) orientation with respect to the aircraft body axes
// Euler angles in deg
IMU_Sensor_Roll_Angle=0.0
IMU_Sensor_Pitch_Angle=-180.0
IMU_Sensor_Yaw_Angle=90.0

// Avionics (IMU sensor) position vector with respect to the aircraft CG, in body axes
// Vector components in m

IMU_Sensor_Position_X=0.09785
IMU_Sensor_Position_Y=0.00024
IMU_Sensor_Position_Z=0.01338



//---------------  GPS ANTENNA MOUNTING  ---------------//


// GPS antenna position with respect to the aircraft CG, in aircraft body axes
GPS_Antenna_Position_X=0.4438
GPS_Antenna_Position_Y=0.0
GPS_Antenna_Position_Z=-0.2509
```

1.8. Simulator State Files

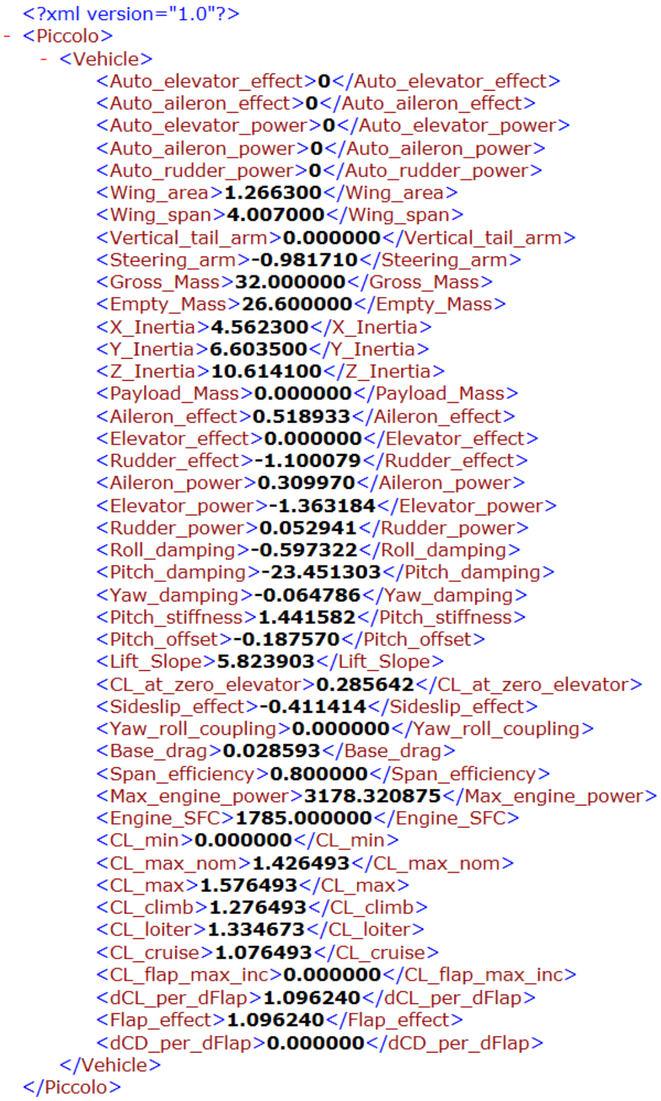


```
Roll=0.0000000000
Pitch=0.0000000000
Yaw=3.0000000000
P=0.0000000000
Q=0.0000000000
R=0.0000000000
Latitude=36.1612820000
Longitude=-96.8356920000
Altitude=285.30
TAS=0.0000000000
Alpha=0.0000000000
Beta=0.0000000000
Ground_Elevation=313.52
```

```
Roll=0.0000000000
Pitch=0.0000000000
Yaw=180.0000000000
P=0.0000000000
Q=0.0000000000
R=0.0000000000
Latitude=36.1628560000
Longitude=-96.8356510000
Altitude=285.3
TAS=0.0000000000
Alpha=0.0000000000
Beta=0.0000000000
Ground_Elevation=313.52
```

The simulator state files were the standard north and south takeoff state files that are in the new aircraft folder. The altitude was set to 285.30 so that the wheels (at 0.26 meters) would be slightly above the ground when simulations start (+0.04 meters).

1.9. Simulator Vehicle File



```
  <?xml version="1.0"?>
- <Piccolo>
  - <Vehicle>
      <Auto_elevator_effect>0</Auto_elevator_effect>
      <Auto_aileron_effect>0</Auto_aileron_effect>
      <Auto_elevator_power>0</Auto_elevator_power>
      <Auto_aileron_power>0</Auto_aileron_power>
      <Auto_rudder_power>0</Auto_rudder_power>
      <Wing_area>1.266300</Wing_area>
      <Wing_span>4.007000</Wing_span>
      <Vertical_tail_arm>0.000000</Vertical_tail_arm>
      <Steering_arm>-0.981710</Steering_arm>
      <Gross_Mass>32.000000</Gross_Mass>
      <Empty_Mass>26.600000</Empty_Mass>
      <X_Inertia>4.562300</X_Inertia>
      <Y_Inertia>6.603500</Y_Inertia>
      <Z_Inertia>10.614100</Z_Inertia>
      <Payload_Mass>0.000000</Payload_Mass>
      <Aileron_effect>0.518933</Aileron_effect>
      <Elevator_effect>0.000000</Elevator_effect>
      <Rudder_effect>-1.100079</Rudder_effect>
      <Aileron_power>0.309970</Aileron_power>
      <Elevator_power>-1.363184</Elevator_power>
      <Rudder_power>0.052941</Rudder_power>
      <Roll_damping>-0.597322</Roll_damping>
      <Pitch_damping>-23.451303</Pitch_damping>
      <Yaw_damping>-0.064786</Yaw_damping>
      <Pitch_stiffness>1.441582</Pitch_stiffness>
      <Pitch_offset>-0.187570</Pitch_offset>
      <Lift_Slope>5.823903</Lift_Slope>
      <CL_at_zero_elevator>0.285642</CL_at_zero_elevator>
      <Sideslip_effect>-0.411414</Sideslip_effect>
      <Yaw_roll_coupling>0.000000</Yaw_roll_coupling>
      <Base_drag>0.028593</Base_drag>
      <Span_efficiency>0.800000</Span_efficiency>
      <Max_engine_power>3178.320875</Max_engine_power>
      <Engine_SFC>1785.000000</Engine_SFC>
      <CL_min>0.000000</CL_min>
      <CL_max_nom>1.426493</CL_max_nom>
      <CL_max>1.576493</CL_max>
      <CL_climb>1.276493</CL_climb>
      <CL_loiter>1.334673</CL_loiter>
      <CL_cruise>1.076493</CL_cruise>
      <CL_flap_max_inc>0.000000</CL_flap_max_inc>
      <dCL_per_dFlap>1.096240</dCL_per_dFlap>
      <Flap_effect>1.096240</Flap_effect>
      <dCD_per_dFlap>0.000000</dCD_per_dFlap>
    </Vehicle>
  </Piccolo>
```

There were two problems with Noctua B1's vehicle file. Both the vertical tail arm and elevator effectiveness were estimated as zero. In order to debug the issues a lot of time was spent changing values in the alpha file and observing how they altered the estimated vehicle parameters to reverse engineer how they were being calculated.

1.9.1. Max Engine Power

Since the motor file used by the simulator was essentially made up to reflect the true maximum rpm of the motor and propeller in real lift the simulator estimated maximum engine power was ignored. The DA100L was rated at a 6 HP max; therefore, 6 HP or 4475 watts, was used as the maximum engine power.

1.9.2. Manual Vertical Tail Arm Estimate

$$-\frac{\sum C_{n_{\delta_r}}}{\sum C_{Y_{\delta_r}}} * b = l_v$$

It was found that the simulator uses the equation above to determine the vertical tail arm (Section 3.6). The simulator will sum up the coefficient terms for each control surface that is interpreted as a rudder control surface. The problem was that since the tail was a ruddervator and ruddervator deflection sign convention was lined up with elevator deflection sign convention the ruddervators had opposite signs for deflecting the same direction. Specifically the left ruddervator would induce a positive yawing moment with a negative rudder deflection while the right ruddervator would induce a positive yawing moment with a positive rudder deflection. The exact same problem also existed with the side force coefficients of the ruddervators. According to the piccolo sign convention trailing edge starboard, or right deflection, for a rudder is considered positive; therefore, a positive rudder deflection should induce a positive yawing moment. The simulator did not change the sign convention for the left ruddervator coefficients for the vertical tail arm calculation; therefore, the $C_{n\delta r}$ of the left ruddervator canceled out the $C_{n\delta r}$ of the right ruddervator as did the $C_{Y\delta r}$ of both surfaces. In the alpha file the left ruddervator was assigned to surface 6 and the right ruddervator was assigned to surface 5.

$$l_v = -\frac{Cnd5(run\ 1) + Cnd6(run\ 1)}{Cyd5(run\ 1) + Cyd6(run\ 1)} * bref = -\frac{0.000462 + -0.000462}{-0.001392 + 0.001392} * 4.007 = 0$$

The calculations above depict the coefficients as they were labeled in the alpha file. Additionally the simulator uses run 1, the very first alpha data point, to calculate the vertical tail arm. The calculation was done manually by changing the sign of the left ruddervator coefficients so that the deflection of the left ruddervator would have the same sign convention as the deflection of the right ruddervator.

$$l_v = -\frac{Cnd5 * (-1) + Cnd6}{Cyd5 * (-1) + Cyd6} = -\frac{0.000462 * (-1) + -0.000462}{-0.001392 * (-1) + 0.001392} * 4.007 = 1.3398m$$

At the time of setting up Noctua B1 these equations had just been backed out, so a physical measurement was made on the aircraft to measure what the vertical tail arm should be. The measurement determined the vertical tail arm to be 1.28 meters. Even though they were similar the measured value of 1.28 meters was used for the vertical tail arm vehicle parameter.

The $C_{n\delta r}$ and $C_{Y\delta r}$ values in the alpha file could have been changed to the proper signs; however, the simulator corrects the sign convention for all other calculations, so if the signs had been changed it would have created many more problems in other areas.

1.9.3. Manual Elevator Effectiveness Estimate

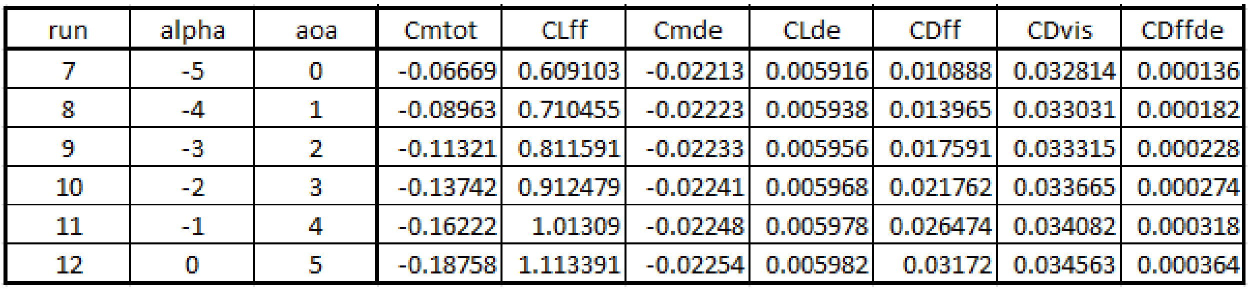
When the simulator calculates elevator effectiveness it will look for values of CL versus alpha in a specific pre-determined range referred to by Cloud Cap as the “linear range.” The simulator defines the linear range as values of CLff in the range of 0.1 to 0.9 at alpha between 0 and 8 degrees.

alpha (deg)	CLff
0	1.113391
1	1.213354
2	1.312948
3	1.412140
4	1.510904
5	1.609207

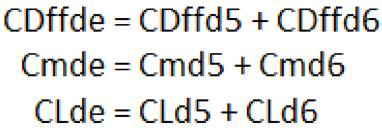
The problem with Noctua B1 was that all of the values of CLff were above 0.9 for all of the alphas in the linear range. As a result the simulator didn't even attempt to calculate a value and simply reported elevator effectiveness as zero. As a result the elevator effectiveness had to be estimated with manual calculations.

The elevator effectiveness estimate was calculated using the method described in Section 3.13 which is nearly exactly how the simulator calculates elevator effectiveness from the alpha file. The calculations went as follows.

Noctua B1 had a wing incidence of 5 degrees. Since the simulator is supposed to calculate elevator effectiveness from angles of attack between 0 and 5 the linear range was determined by choosing the alpha runs that corresponded with 0 to 5 degrees angle of attack. The range was determined to be from alpha -5 to alpha 0. The following data was extracted from Noctua B1's alpha file.



run	alpha	aoa	Cmtot	CLff	Cmde	CLde	CDff	CDvis	CDffde
7	-5	0	-0.06669	0.609103	-0.02213	0.005916	0.010888	0.032814	0.000136
8	-4	1	-0.08963	0.710455	-0.02223	0.005938	0.013965	0.033031	0.000182
9	-3	2	-0.11321	0.811591	-0.02233	0.005956	0.017591	0.033315	0.000228
10	-2	3	-0.13742	0.912479	-0.02241	0.005968	0.021762	0.033665	0.000274
11	-1	4	-0.16222	1.01309	-0.02248	0.005978	0.026474	0.034082	0.000318
12	0	5	-0.18758	1.113391	-0.02254	0.005982	0.03172	0.034563	0.000364



CDffde = CDffd5 + CDffd6
Cmde = Cmd5 + Cmd6
CLde = CLd5 + CLd6

A δ_e and C_{LZ} value was calculated for each alpha run. A sample calculation is shown below of alpha run 7.

$$C_{m\,trim}(run\ 7) = -C_{mtot}(run\ 7) = -(-0.06669)$$

$$\delta_e(run\ 7) = \frac{C_{m\ trim}(run\ 7)}{C_m/_{\delta_e}\ (run\ 7)} = \frac{0.06669}{-0.02213} = -3.01397\ deg$$

$$C_L(run\ 7) = \left(C_Lff(run\ 7) + \frac{C_L}{\delta_e}(run\ 7) * \delta_e(run\ 7)\right)$$

$$= (0.609103 + 0.005916 * -3.01397)$$

$$= 0.591272$$

$$C_D(run\ 7) = \left(C_Dff(run\ 7) + C_Dvis(run\ 7) + \frac{C_Dff}{\delta_e}(run\ 7) * \delta_e(run\ 7)\right)$$

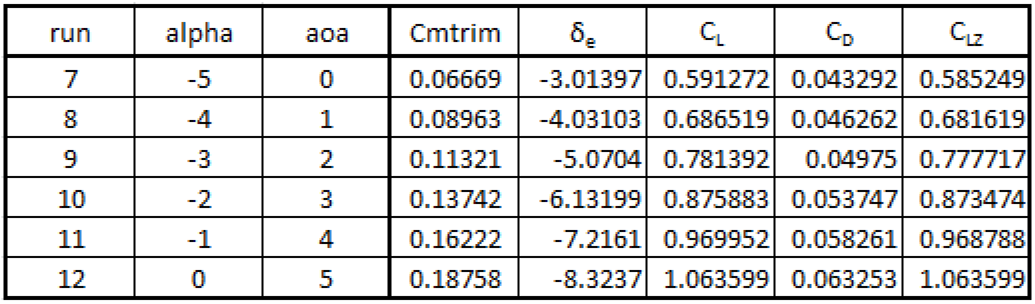
$$= 0.010888 + 0.032814 + 0.000136 * -3.01397$$

$$= 0.043292$$

$$C_{L_Z}(run\ 7) = C_L(run\ 7) * cos(-5) + C_D(run\ 7) * sin(-5)$$

$$= 0.591272 * \cos(-5) + 0.043292 * \sin(-5)$$

$$= 0.585249$$



run	alpha	aoa	Cmtrim	δ_e	C_L	C_D	C_{LZ}
7	-5	0	0.06669	-3.01397	0.591272	0.043292	0.585249
8	-4	1	0.08963	-4.03103	0.686519	0.046262	0.681619
9	-3	2	0.11321	-5.0704	0.781392	0.04975	0.777717
10	-2	3	0.13742	-6.13199	0.875883	0.053747	0.873474
11	-1	4	0.16222	-7.2161	0.969952	0.058261	0.968788
12	0	5	0.18758	-8.3237	1.063599	0.063253	1.063599

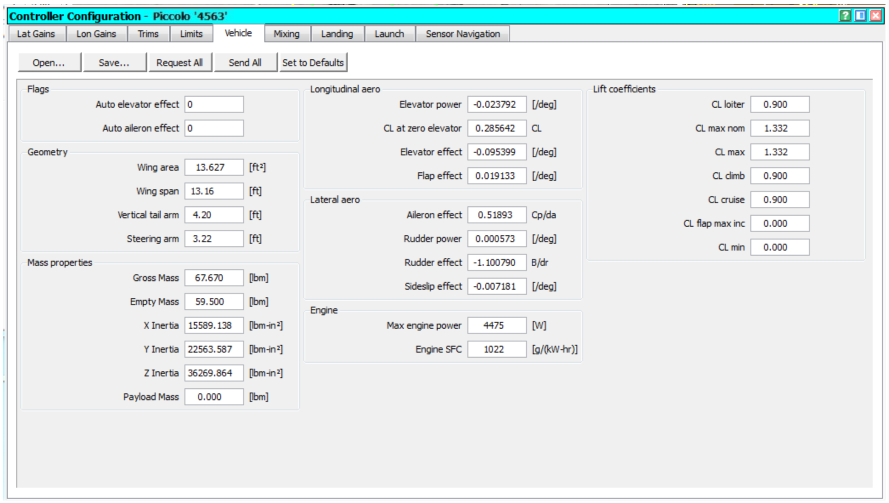
The table above depicts the results of all of the calculations. The final step was to find the largest $\Delta C_{LZ}/\Delta\delta_e$ between all of the alpha runs.

Elevator Effectiveness (/rad)

alpha	-5	-4	-3	-2	-1	0
-5	-					
-4	-5.42894	-				
-3	-5.36247	-5.29743	-			
-2	-5.29631	-5.23211	-5.16816	-		
-1	-5.22951	-5.16583	-5.10208	-5.03738	-	
0	-5.16172	-5.09841	-5.03483	-4.97025	-4.90454	-

The largest occurred between alpha -5 and -4 with an elevator effectiveness of -5.43 /rad.

1.10. Initial Vehicle Parameters



Flags
Auto elevator effect 0
Auto aileron effect 0

Geometry
Wing area 13.627 [ft^2]
Wing span 13.16 [ft]
Vertical tail arm 4.20 [ft]
Steering arm 3.22 [ft]

Mass properties
Gross Mass 67.670 [lbm]
Empty Mass 59.500 [lbm]
X Inertia 15589.138 [lbm-in^2]
Y Inertia 22563.587 [lbm-in^2]
Z Inertia 36269.864 [lbm-in^2]
Payload Mass 0.000 [lbm]

Longitudinal aero
Elevator power -0.023792 [/deg]
CL at zero elevator 0.285642 CL
Elevator effect -0.095399 [/deg]
Flap effect 0.019133 [/deg]

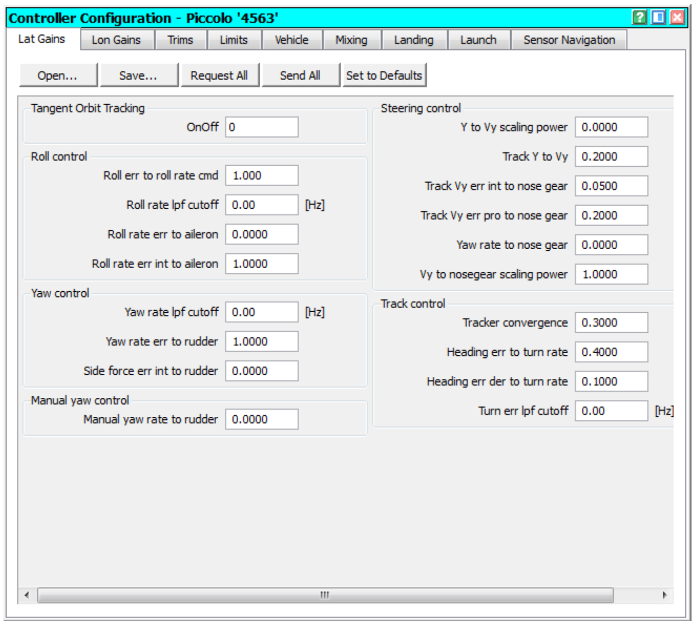
Lateral aero
Aileron effect 0.51893 Cp/da
Rudder power 0.000573 [/deg]
Rudder effect -1.100790 B/dr
Sideslip effect -0.007181 [/deg]

Engine
Max engine power 4475 [W]
Engine SFC 1022 [g/(kW-hr)]

Lift coefficients
CL loiter 0.900
CL max nom 1.332
CL max 1.332
CL climb 0.900
CL cruise 0.900
CL flap max inc 0.000
CL min 0.000

The initial vehicle parameters used matched those that the simulator estimated except for the elevator effectiveness, vertical tail arm, and max engine power parameters. Additionally the lift coefficients were set to match desired airspeeds. CLmax and CLmax nom were set to 1.332 so that the minimum airspeed would be 34 knots. CL loiter, climb, and cruise were set to 0.9 so that the airspeed command would be 41.5 knots.

1.11. Initial Lat Gains



Tangent Orbit Tracking
OnOff 0

Roll control
Roll err to roll rate cmd 1.000
Roll rate lpf cutoff 0.00 [Hz]
Roll rate err to aileron 0.0000
Roll rate err int to aileron 1.0000

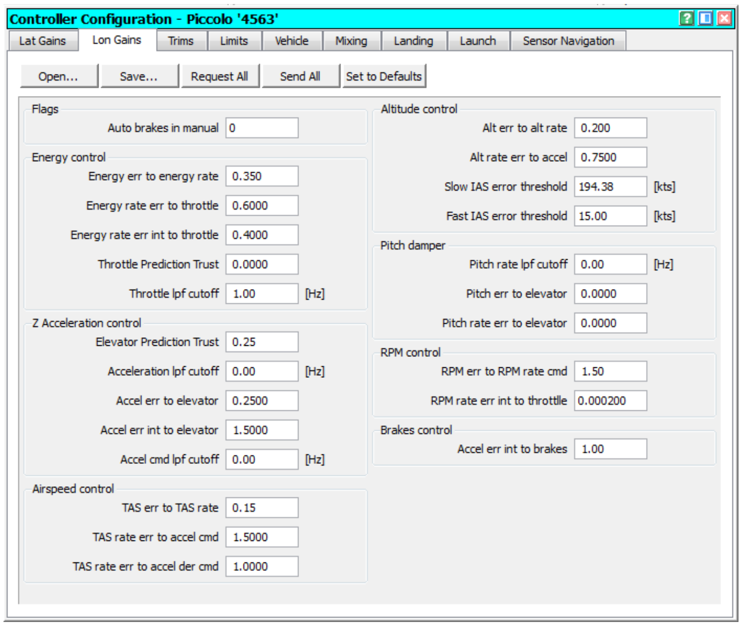
Yaw control
Yaw rate lpf cutoff 0.00 [Hz]
Yaw rate err to rudder 1.0000
Side force err int to rudder 0.0000

Manual yaw control
Manual yaw rate to rudder 0.0000

Steering control
Y to Vy scaling power 0.0000
Track Y to Vy 0.2000
Track Vy err int to nose gear 0.0500
Track Vy err pro to nose gear 0.2000
Yaw rate to nose gear 0.0000
Vy to nosegear scaling power 1.0000

Track control
Tracker convergence 0.3000
Heading err to turn rate 0.4000
Heading err der to turn rate 0.1000
Turn err lpf cutoff 0.00 [Hz]

1.12. Initial Lon Gains



Flags
Auto brakes in manual 0

Energy control
Energy err to energy rate 0.350
Energy rate err to throttle 0.6000
Energy rate err int to throttle 0.4000
Throttle Prediction Trust 0.0000
Throttle lpf cutoff 1.00 [Hz]

Z Acceleration control
Elevator Prediction Trust 0.25
Acceleration lpf cutoff 0.00 [Hz]
Accel err to elevator 0.2500
Accel err int to elevator 1.5000
Accel cmd lpf cutoff 0.00 [Hz]

Airspeed control
TAS err to TAS rate 0.15
TAS rate err to accel cmd 1.5000
TAS rate err to accel der cmd 1.0000

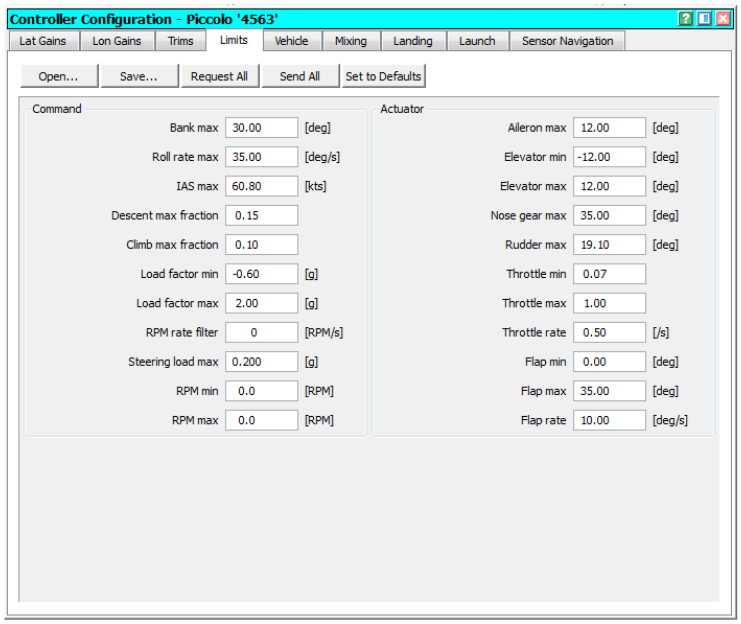
Altitude control
Alt err to alt rate 0.200
Alt rate err to accel 0.7500
Slow IAS error threshold 194.38 [kts]
Fast IAS error threshold 15.00 [kts]

Pitch damper
Pitch rate lpf cutoff 0.00 [Hz]
Pitch err to elevator 0.0000
Pitch rate err to elevator 0.0000

RPM control
RPM err to RPM rate cmd 1.50
RPM rate err int to throttle 0.000200

Brakes control
Accel err int to brakes 1.00

1.13. Limits



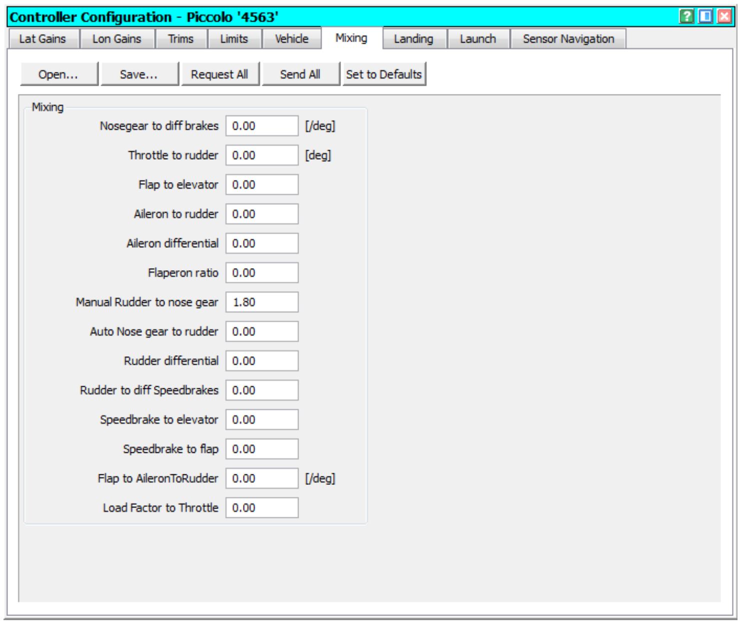
Command

Bank max	30.00	[deg]
Roll rate max	35.00	[deg/s]
IAS max	60.80	[kts]
Descent max fraction	0.15	
Climb max fraction	0.10	
Load factor min	-0.60	[g]
Load factor max	2.00	[g]
RPM rate filter	0	[RPM/s]
Steering load max	0.200	[g]
RPM min	0.0	[RPM]
RPM max	0.0	[RPM]

Actuator

Aileron max	12.00	[deg]
Elevator min	-12.00	[deg]
Elevator max	12.00	[deg]
Nose gear max	35.00	[deg]
Rudder max	19.10	[deg]
Throttle min	0.07	
Throttle max	1.00	
Throttle rate	0.50	[/s]
Flap min	0.00	[deg]
Flap max	35.00	[deg]
Flap rate	10.00	[deg/s]

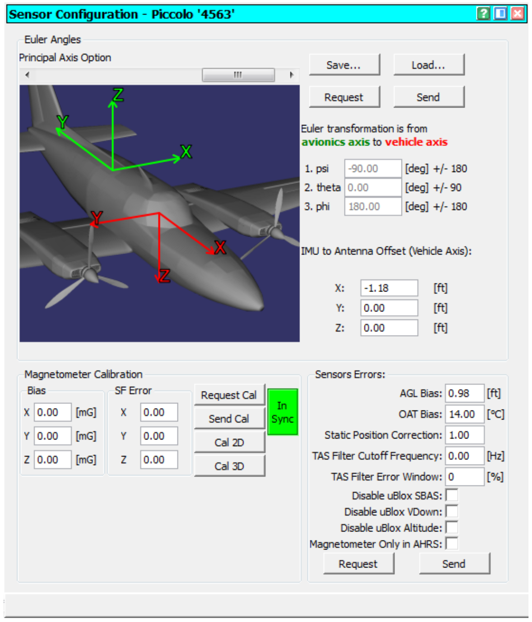
1.14. Initial Mixing



Mixing

Nosegear to diff brakes	0.00	[/deg]
Throttle to rudder	0.00	[deg]
Flap to elevator	0.00	
Aileron to rudder	0.00	
Aileron differential	0.00	
Flaperon ratio	0.00	
Manual Rudder to nose gear	1.80	
Auto Nose gear to rudder	0.00	
Rudder differential	0.00	
Rudder to diff Speedbrakes	0.00	
Speedbrake to elevator	0.00	
Speedbrake to flap	0.00	
Flap to AileronToRudder	0.00	[/deg]
Load Factor to Throttle	0.00	

1.15. Initial Sensor Configuration



Sensor Configuration - Piccolo '4563'

Euler Angles

Principal Axis Option

Euler transformation is from **avionics axis** to **vehicle axis**

1. psi -90.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 180.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]

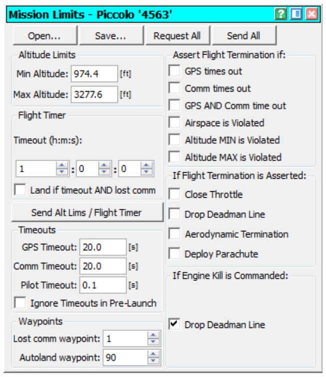
Magnetometer Calibration

Bias			SF Error	
X	0.00	[mG]	X	0.00
Y	0.00	[mG]	Y	0.00
Z	0.00	[mG]	Z	0.00

Sensors Errors:

AGL Bias: 0.98 [ft]
OAT Bias: 14.00 [°C]
Static Position Correction: 1.00
TAS Filter Cutoff Frequency: 0.00 [Hz]
TAS Filter Error Window: 0 [%]
Disable uBlox SBAS: □
Disable uBlox VDown: □
Disable uBlox Altitude: □
Magnetometer Only in AHRS: □

1.16. Initial Mission Limits



Mission Limits - Piccolo '4563'

Altitude Limits

Min Altitude: 974.4 [ft]
Max Altitude: 3277.6 [ft]

Flight Timer

Timeout (h:m:s):

1 : 0 : 0

□ Land if timeout AND lost comm

Timeouts

GPS Timeout: 20.0 [s]
Comm Timeout: 20.0 [s]
Pilot Timeout: 0.1 [s]
□ Ignore Timeouts in Pre-Launch

Waypoints

Lost comm waypoint: 1
Autoland waypoint: 90

Assert Flight Termination if:

□ GPS times out
□ Comm times out
□ GPS AND Comm time out
□ Airspace is Violated
□ Altitude MIN is Violated
□ Altitude MAX is Violated

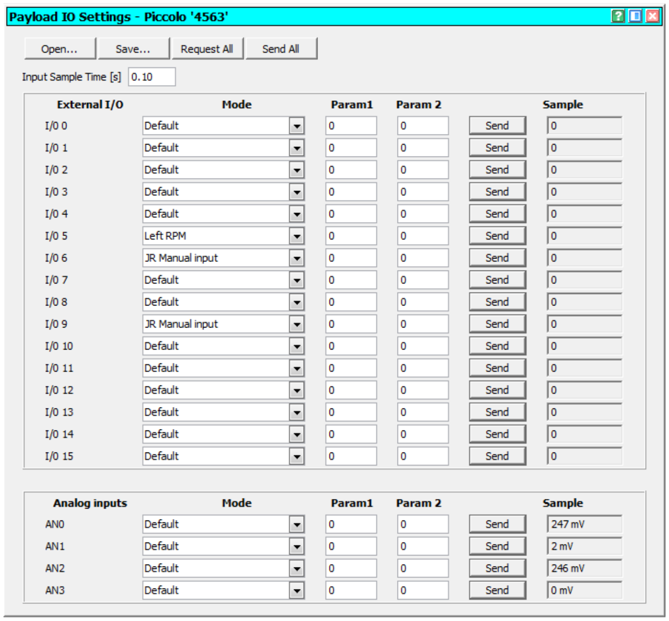
If Flight Termination is Asserted:

□ Close Throttle
□ Drop Deadman Line
□ Aerodynamic Termination
□ Deploy Parachute

If Engine Kill is Commanded:

☑ Drop Deadman Line

1.17. Payload IO Settings



Input Sample Time [s] 0.10

External I/O	Mode	Param1	Param 2		Sample
I/0 0	Default	0	0	Send	0
I/0 1	Default	0	0	Send	0
I/0 2	Default	0	0	Send	0
I/0 3	Default	0	0	Send	0
I/0 4	Default	0	0	Send	0
I/0 5	Left RPM	0	0	Send	0
I/0 6	JR Manual input	0	0	Send	0
I/0 7	Default	0	0	Send	0
I/0 8	Default	0	0	Send	0
I/0 9	JR Manual input	0	0	Send	0
I/0 10	Default	0	0	Send	0
I/0 11	Default	0	0	Send	0
I/0 12	Default	0	0	Send	0
I/0 13	Default	0	0	Send	0
I/0 14	Default	0	0	Send	0
I/0 15	Default	0	0	Send	0

Analog inputs	Mode	Param1	Param 2		Sample
AN0	Default	0	0	Send	247 mV
AN1	Default	0	0	Send	2 mV
AN2	Default	0	0	Send	246 mV
AN3	Default	0	0	Send	0 mV

The RPM signal was wired into pin 30 which was the “Servo_6_PWM” line. Servo 6 corresponded with IO 5; therefore, “I/O 5” was designated as “Left RPM”.

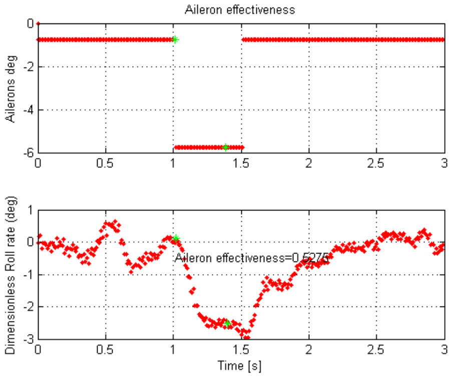
The JR switch board was installed into pins 47 and 26 which were the “Servo_5_PWM” and “Servo_9_PWM” lines. Servo 5 was the “TPU_B” line and servo 9 was the “TPU_A” line. Servo 5 corresponded with IO 6, and servo 9 corresponded with IO 9; therefore, “I/O 6” and “I/O 9” were designated as “JR Manual input.”

1.18. Doublet Results

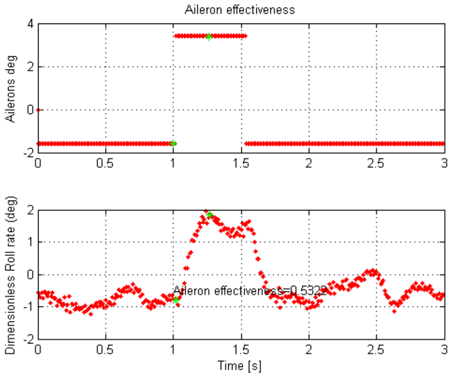
1.18.1. Flight 3

1.18.1.1. Aileron Doublets

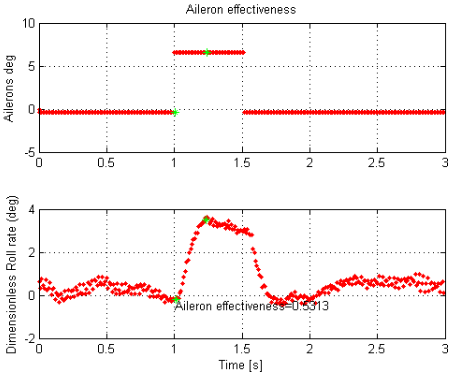
1) Doublet File Method



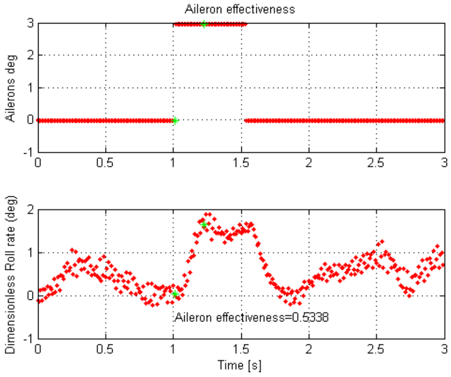
Aileron1



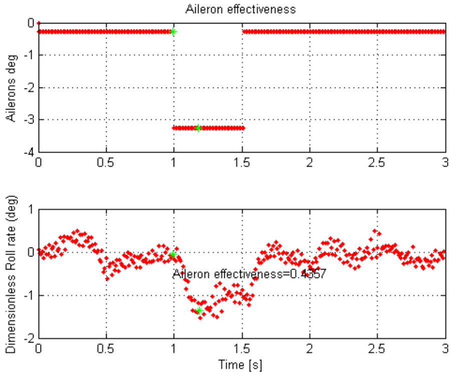
Aileron2



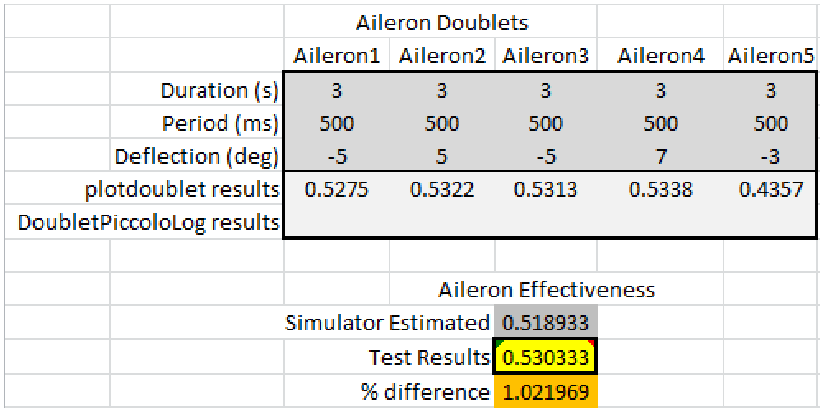
Aileron3



Aileron4



Aileron5

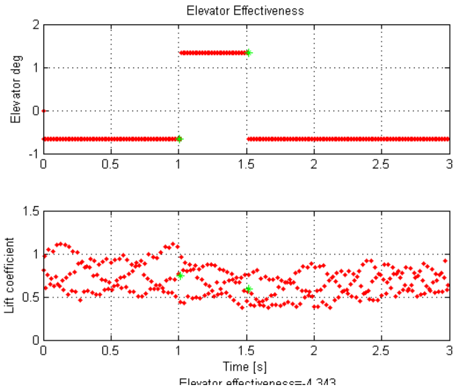


	Aileron Doublets				
	Aileron1	Aileron2	Aileron3	Aileron4	Aileron5
Duration (s)	3	3	3	3	3
Period (ms)	500	500	500	500	500
Deflection (deg)	-5	5	-5	7	-3
plotdoublet results	0.5275	0.5322	0.5313	0.5338	0.4357
DoubletPiccoloLog results					

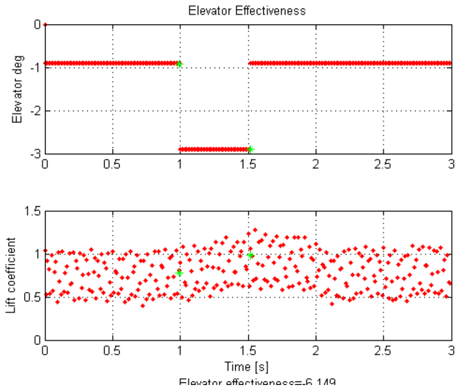
Aileron Effectiveness	
Simulator Estimated	0.518933
Test Results	0.530333
% difference	1.021969

1.18.1.2. Elevator Doublets

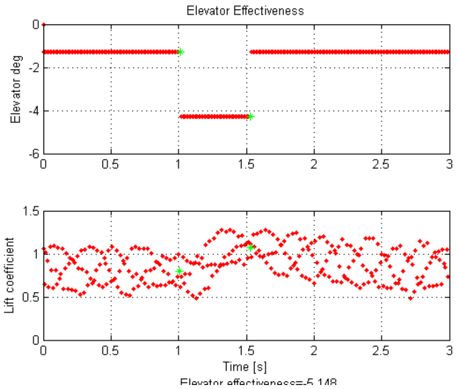
1) Doublet File Method



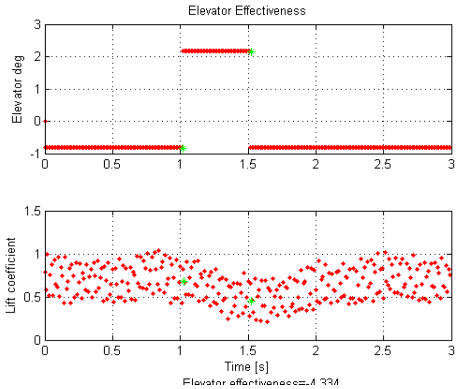
Elevator1



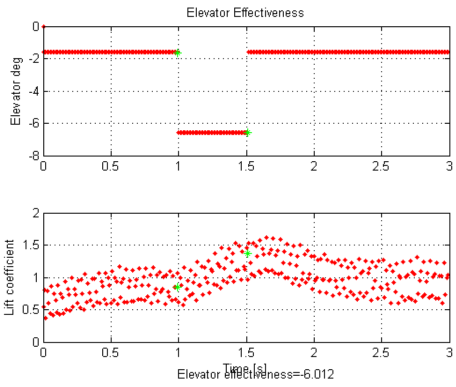
Elevator2



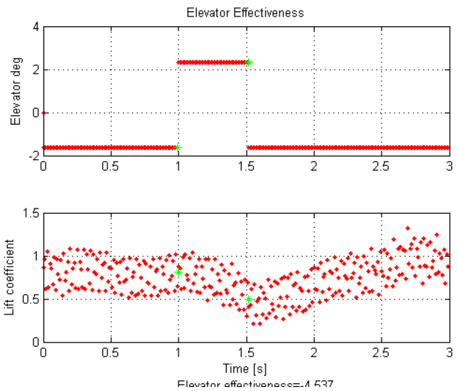
Elevator3



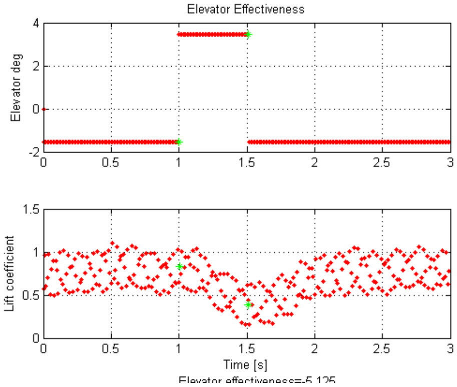
Elevator4



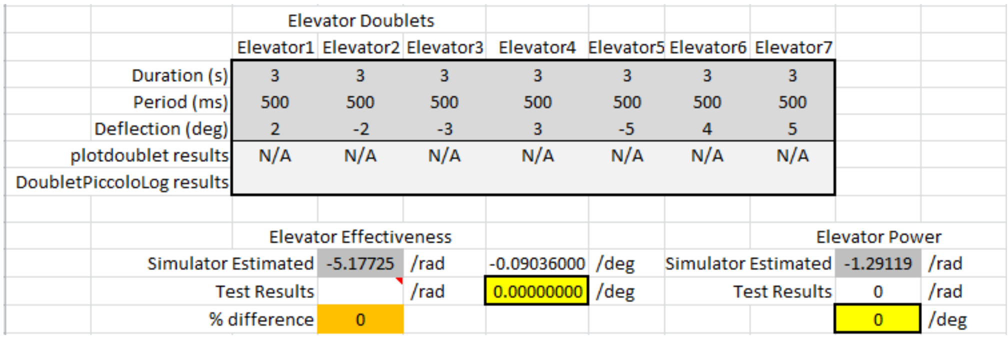
Elevator5



Elevator6



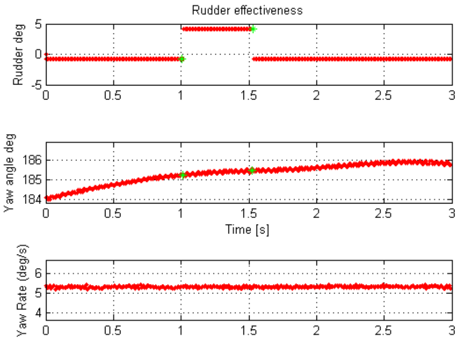
Elevator7



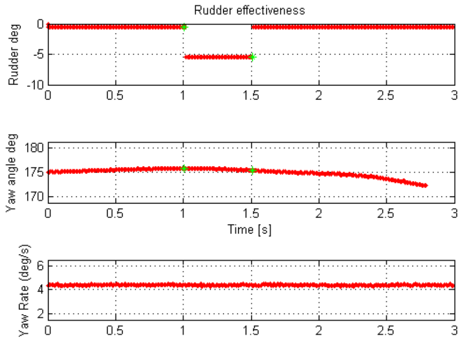
		Elevator Doublets							
	Elevator1	Elevator2	Elevator3	Elevator4	Elevator5	Elevator6	Elevator7		
Duration (s)	3	3	3	3	3	3	3		
Period (ms)	500	500	500	500	500	500	500		
Deflection (deg)	2	-2	-3	3	-5	4	5		
plotdoublet results	N/A	N/A	N/A	N/A	N/A	N/A	N/A		
DoubletPiccoloLog results									
		Elevator Effectiveness						Elevator Power	
	Simulator Estimated	-5.17725	/rad	-0.09036000	/deg		Simulator Estimated	-1.29119	/rad
	Test Results		/rad	0.00000000	/deg		Test Results	0	/rad
	% difference	0						0	/deg

1.18.1.3. Rudder Doublets

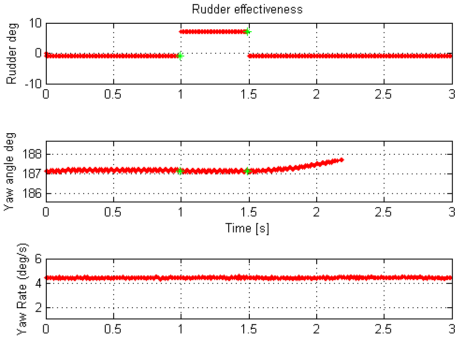
1) Doublet File Method



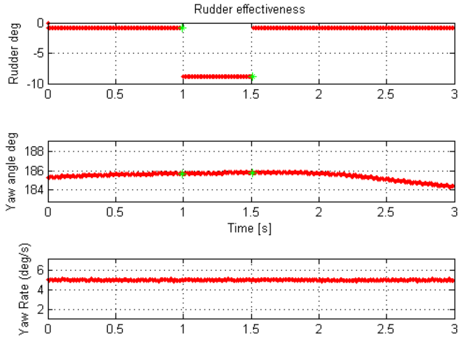
Rudder1



Rudder2

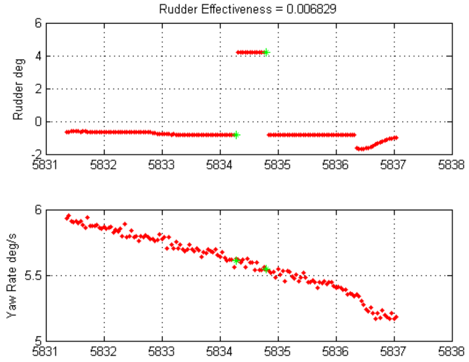


Rudder3

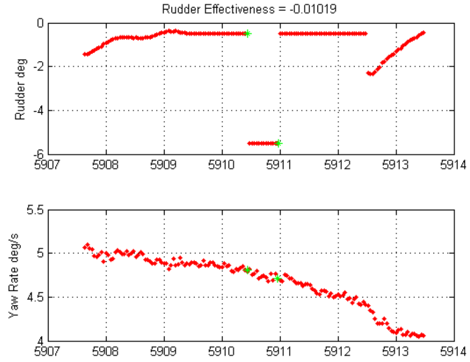


Rudder4

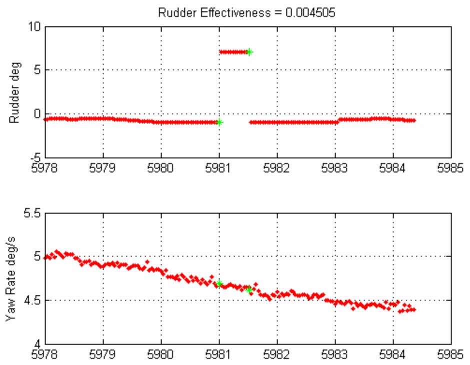
2) Piccolo Log File Method



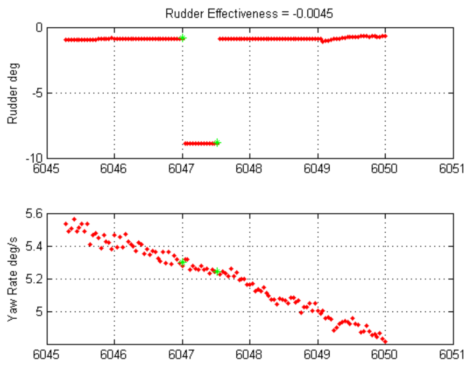
Rudder1



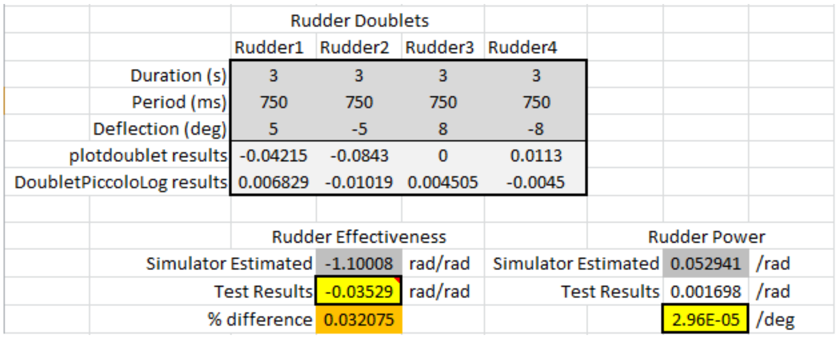
Rudder2



Rudder3



Rudder4



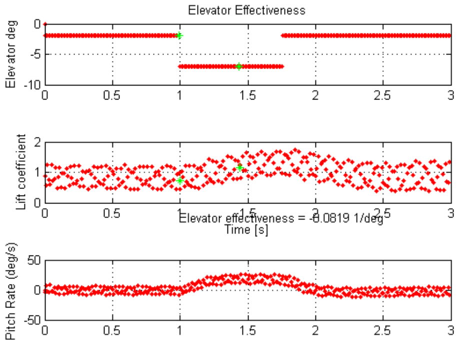
	Rudder Doublets			
	Rudder1	Rudder2	Rudder3	Rudder4
Duration (s)	3	3	3	3
Period (ms)	750	750	750	750
Deflection (deg)	5	-5	8	-8
plotdoublet results	-0.04215	-0.0843	0	0.0113
DoubletPiccoloLog results	0.006829	-0.01019	0.004505	-0.0045

	Rudder Effectiveness			Rudder Power	
Simulator Estimated	-1.10008	rad/rad	Simulator Estimated	0.052941	/rad
Test Results	-0.03529	rad/rad	Test Results	0.001698	/rad
% difference	0.032075			2.96E-05	/deg

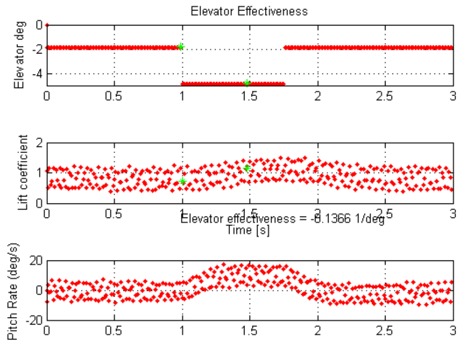
1.18.2. Flight 4

1.18.2.1. Elevator Doublets

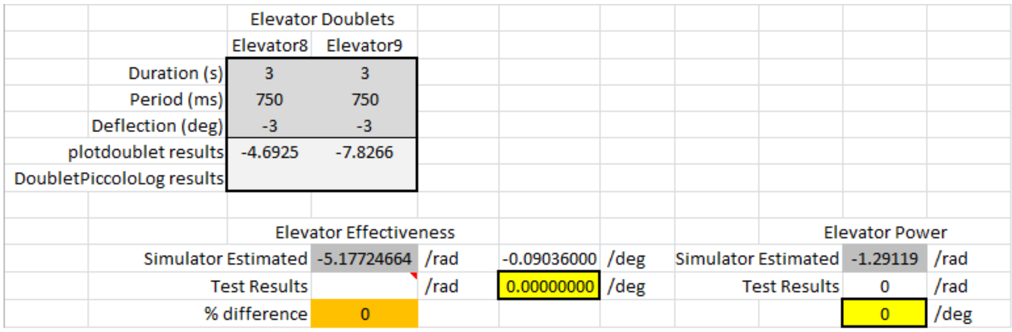
1) Doublet File Method



Elevator8



Elevator9

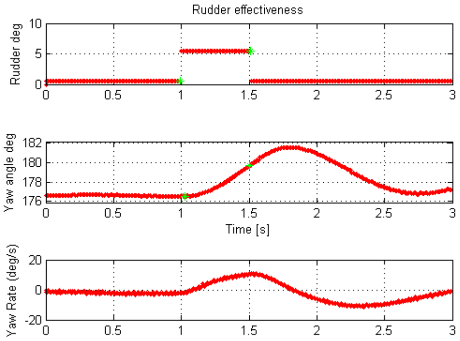


	Elevator Doublets	
	Elevator8	Elevator9
Duration (s)	3	3
Period (ms)	750	750
Deflection (deg)	-3	-3
plotdoublet results	-4.6925	-7.8266
DoubletPiccoloLog results		

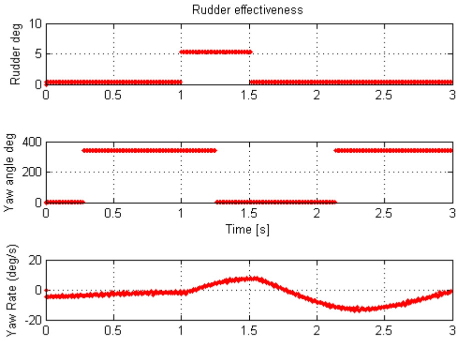
	Elevator Effectiveness					Elevator Power	
Simulator Estimated	-5.17724664	/rad	-0.09036000	/deg	Simulator Estimated	-1.29119	/rad
Test Results		/rad	0.00000000	/deg	Test Results	0	/rad
% difference	0					0	/deg

1.18.2.2. Rudder Doublets

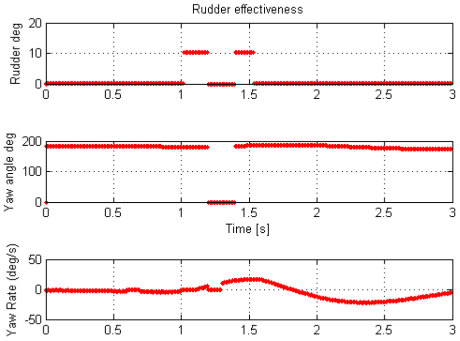
1) Doublet File Method



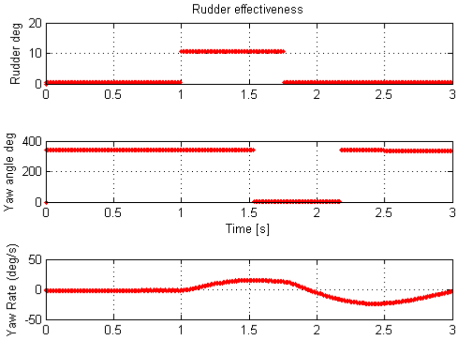
Rudder5



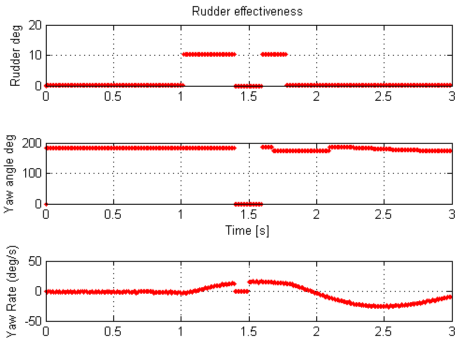
Rudder6



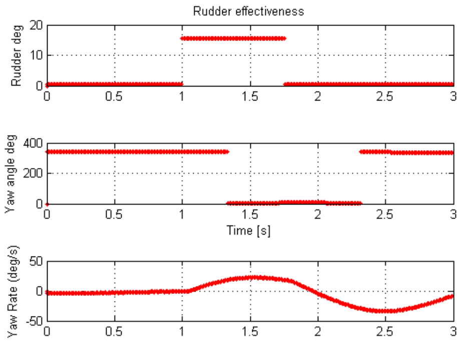
Rudder7



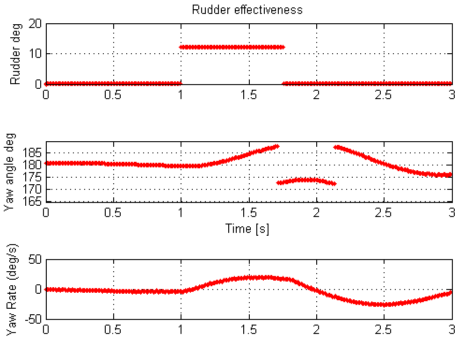
Rudder8



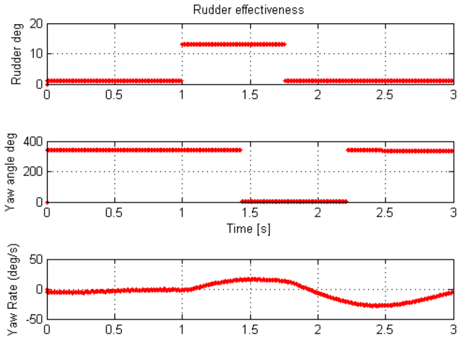
Rudder9



Rudder10

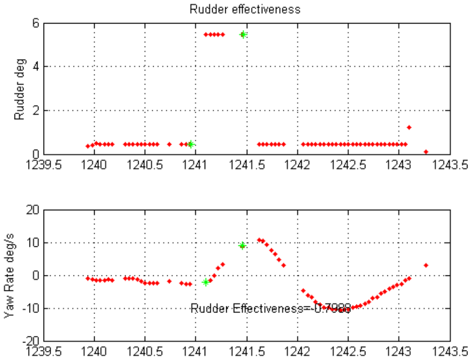


Rudder11

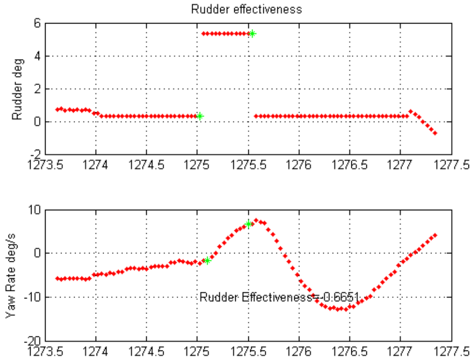


Rudder12

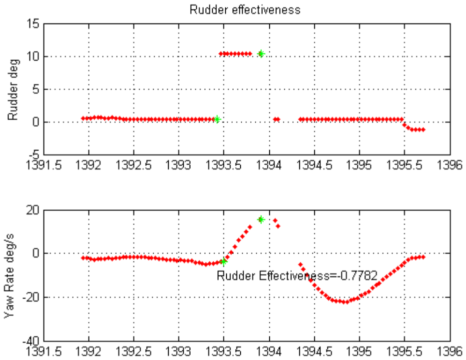
2) Piccolo Log File Method



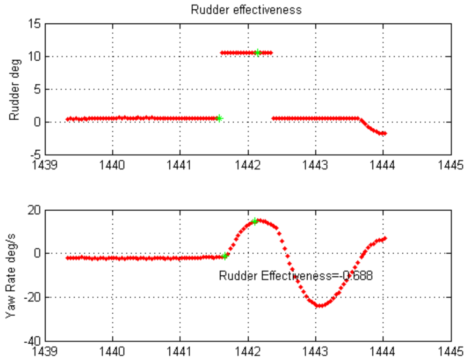
Rudder5



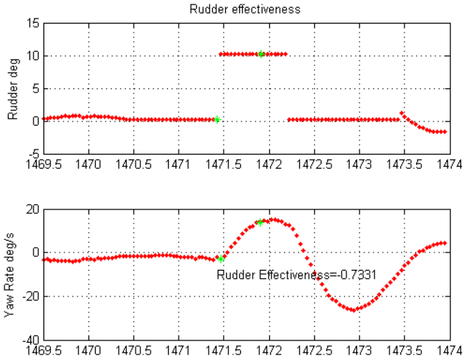
Rudder6



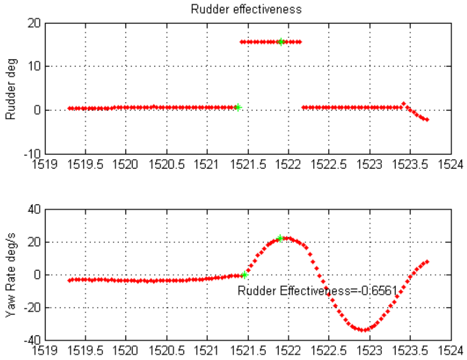
Rudder7



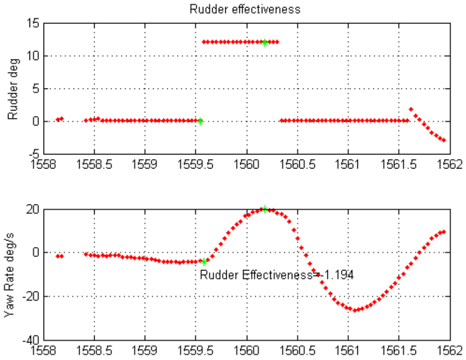
Rudder8



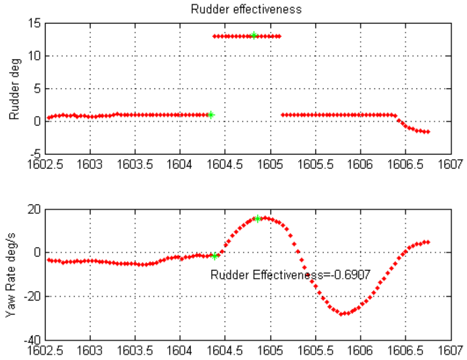
Rudder9



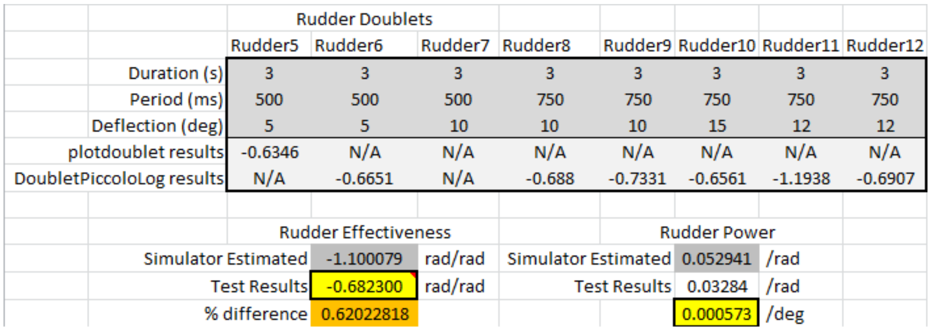
Rudder10



Rudder11



Rudder12

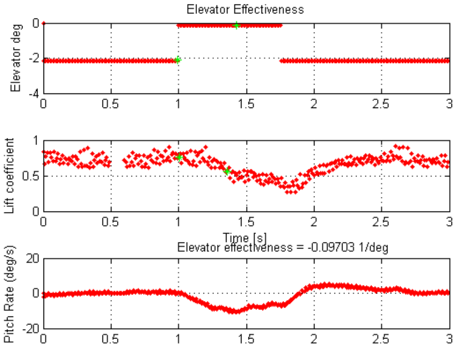


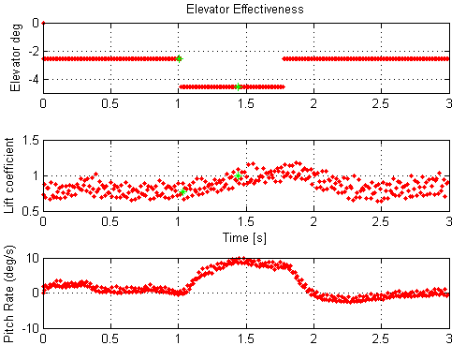
		Rudder Doublets						
	Rudder5	Rudder6	Rudder7	Rudder8	Rudder9	Rudder10	Rudder11	Rudder12
Duration (s)	3	3	3	3	3	3	3	3
Period (ms)	500	500	500	750	750	750	750	750
Deflection (deg)	5	5	10	10	10	15	12	12
plotdoublet results	-0.6346	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DoubletPiccoloLog results	N/A	-0.6651	N/A	-0.688	-0.7331	-0.6561	-1.1938	-0.6907
		Rudder Effectiveness				Rudder Power		
	Simulator Estimated	-1.100079	rad/rad		Simulator Estimated	0.052941	/rad	
	Test Results	-0.682300	rad/rad		Test Results	0.03284	/rad	
	% difference	0.62022818				0.000573	/deg	

1.18.3. Flight 5

1.18.3.1. Elevator Doublets

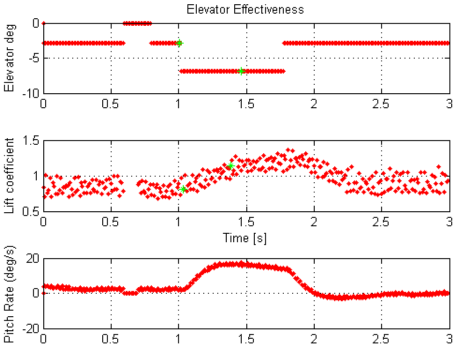
1) Doublet File Method

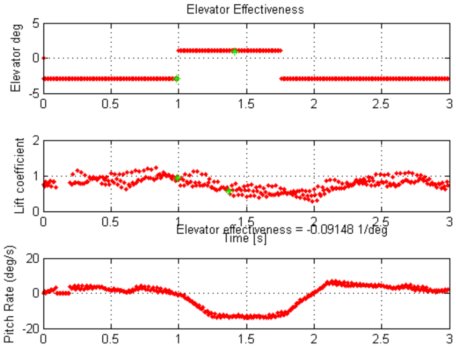




Elevator11

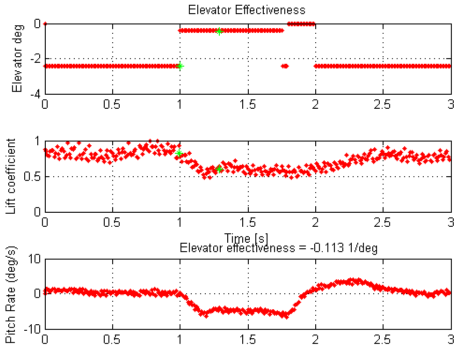
Elevator12



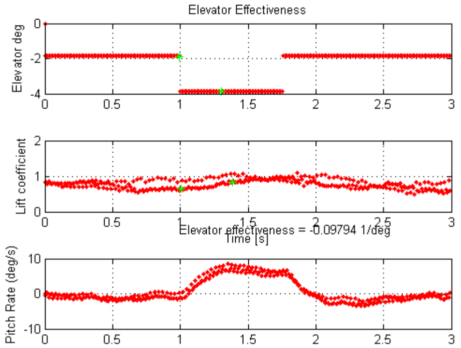


Elevator13

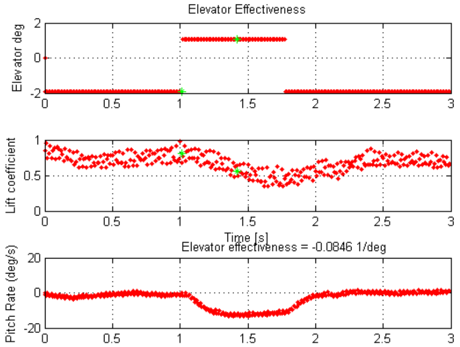
Elevator14



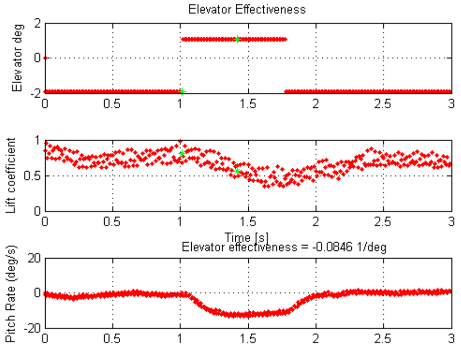
Elevator15



Elevator16

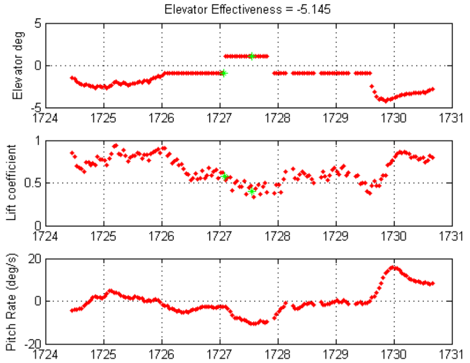


Elevator17

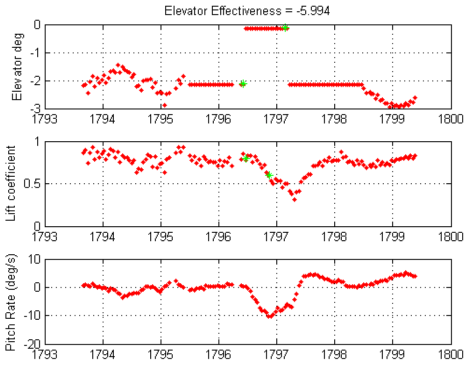


Elevator18

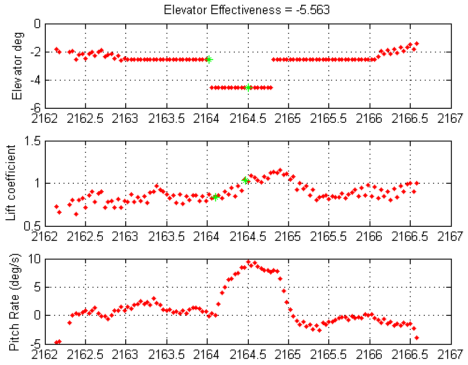
2) Piccolo Log File Method



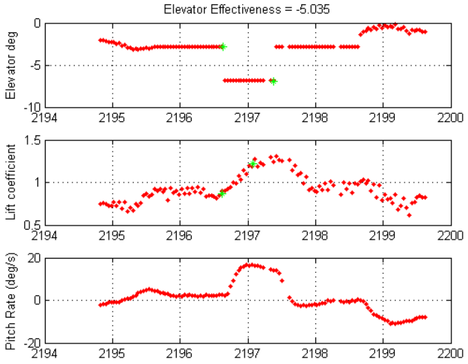
Elevator10



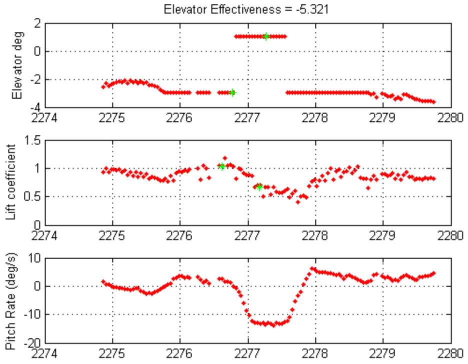
Elevator11



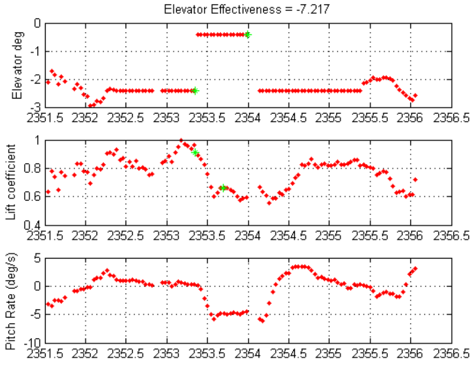
Elevator12



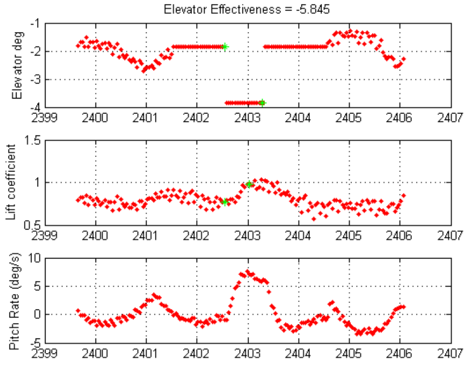
Elevator13



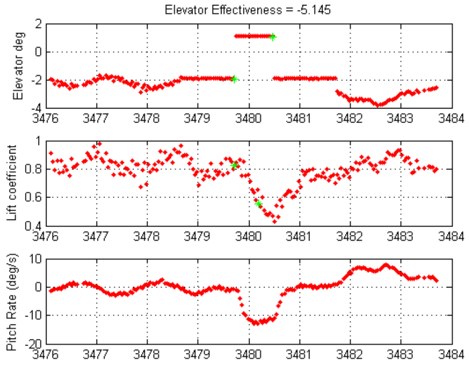
Elevator14



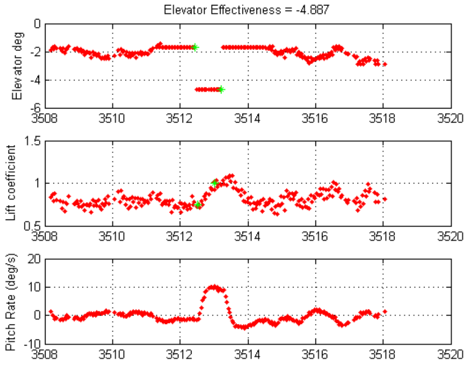
Elevator15



Elevator16



Elevator17



Elevator18

1.19. Flight 2

1.19.1. Flight Plan

1) Ground Comms check

2) Manual R/C pilot flight with JR

    - Check flight comms

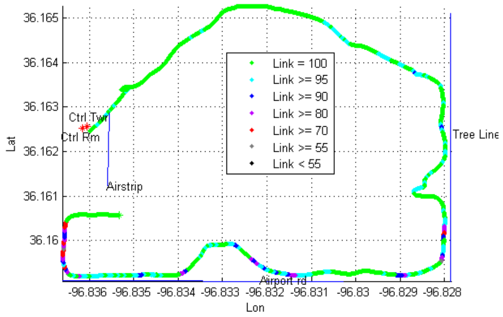
3) Switch to autopilot

4) Perform aileron, elevator, and rudder doublet maneuvers

5) Further gain tuning if necessary

1.19.2. Flight Analysis

Ground comms check



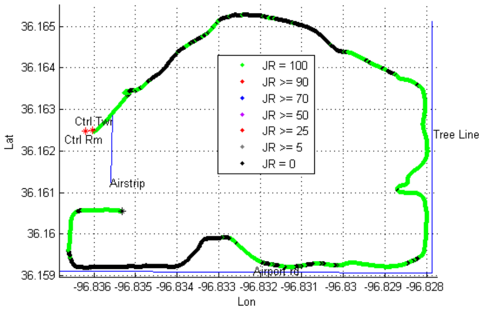
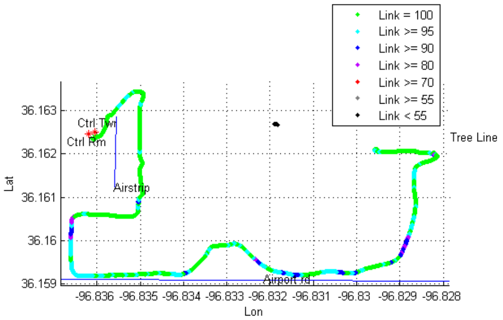


Figure 1

Figure 2

The first ground comms check showed areas of piccolo packet loss. The packet loss wasn't too severe until the aircraft got to the south east end behind the control room. Its likely line of sight was being compromised in that location since the aircraft was sitting, belly down (where the antenna is), inside a gator. The manual JR signal was lost in quite a few places; however, they all coincided with loss of line of sight. The JR receiver wasn't up on the control tower like the piccolo antenna so it lost line of sight quite frequently. It was able to stay connected all the way east to the tree line. Seems odd that the JR signal is either 100% or 0%.

Ground comms check 2



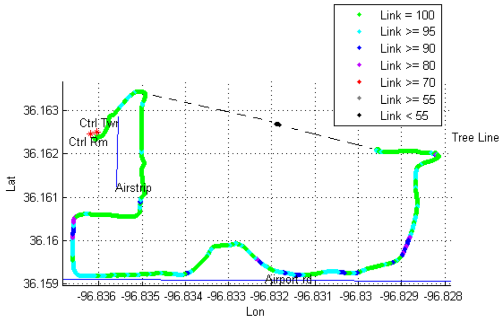


Figure 3

Figure 4

The second ground comms check we evaluated the piccolo signal strength only. There was an area of complete signal loss with the piccolo. The dotted line in Figure 4 shows the path that the aircraft traveled during the period of signal loss. We determined that the likely reason was that the antenna on Noctua had lost line of sight. The packet loss was much less severe south east of the control room the second time.

Flight Analysis

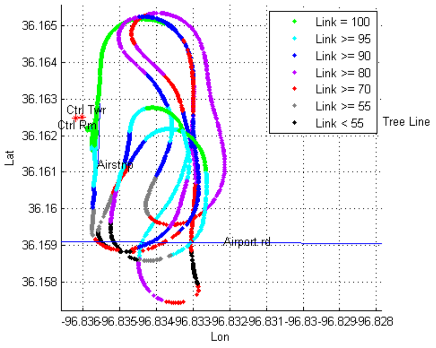


Figure 5

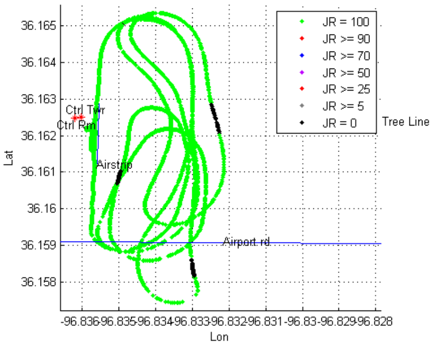


Figure 6

There was significant packet loss with the piccolo signal. As noted by the flight notes there were a couple of instances of JR signal loss.

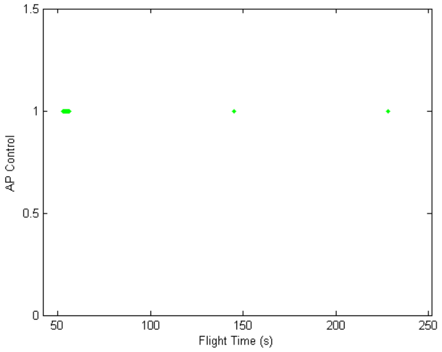


Figure 7

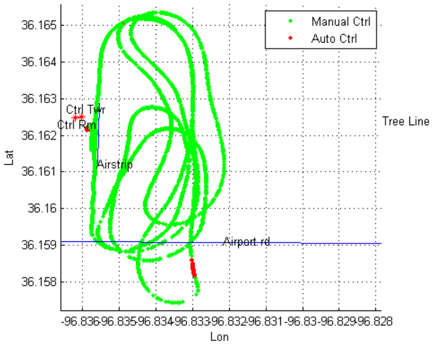
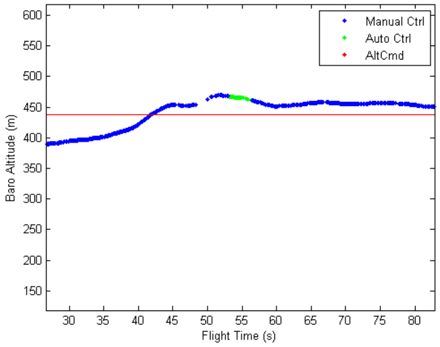


Figure 8

Figure 7 shows that the JR signal was lost for a long enough period that the autopilot actually took control on 3 separate occasions. The first occasion lasted for about 3 seconds and was long enough to be able to analyze the autopilot's actions. The second two occasions lasted only 0.04 seconds each.



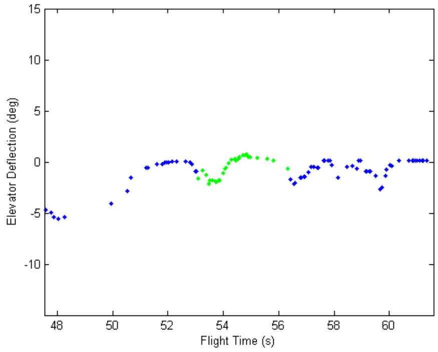


Figure 9

Figure 10

Figure 9 shows that the first time the autopilot took over the aircraft was above the commanded altitude therefore the autopilot should command pitch down. Figure 10 shows that the elevator deflection slowly deflected to pitch up. There wasn't a sudden jump in elevator deflection or oscillating elevator deflections. The deflections appear to have been smooth.

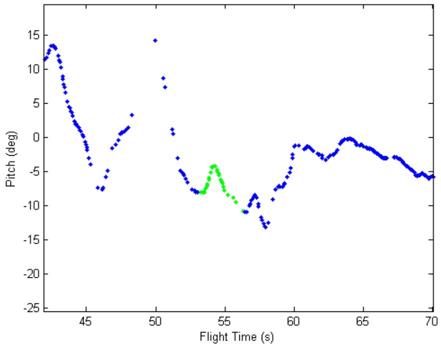


Figure 11

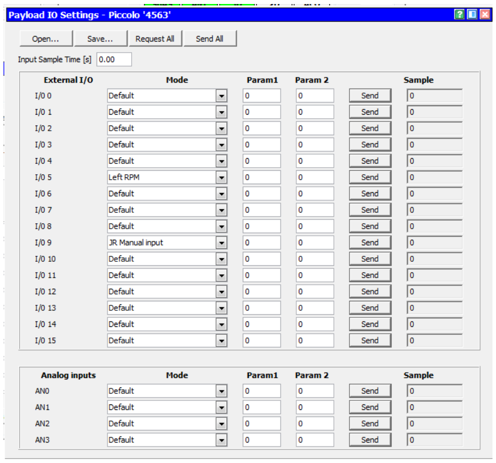
Figure 11 shows that the aircraft initially decreased the pitch of the aircraft and then slowly increased it indicating that the aircraft was actually descending more sharply than the autopilot liked when it took over. Additionally the autopilot did target the lost comms waypoint, waypoint 1.

1.20. Flight 3

1.20.1. Pre – Flight Autopilot Changes

1) After examining the aircraft it was discovered that the ground plane for the piccolo antenna had been taped over. This could account for a significant amount of the piccolo packet loss experienced in Flight 2. The tape was removed.

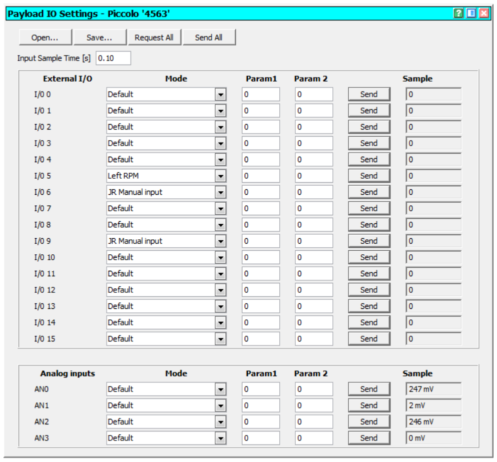
2) Up to this point it was understood that only the I/O line of the TPU_A connection that the JR Manual switch board was plugged into had to be designated as "JR Manual Input" in the Payload IO Settings.



Input Sample Time [s] 0.00

External I/O	Mode	Param1	Param 2		Sample
I/0 0	Default	0	0	Send	0
I/0 1	Default	0	0	Send	0
I/0 2	Default	0	0	Send	0
I/0 3	Default	0	0	Send	0
I/0 4	Default	0	0	Send	0
I/0 5	Left RPM	0	0	Send	0
I/0 6	Default	0	0	Send	0
I/0 7	Default	0	0	Send	0
I/0 8	Default	0	0	Send	0
I/0 9	JR Manual input	0	0	Send	0
I/0 10	Default	0	0	Send	0
I/0 11	Default	0	0	Send	0
I/0 12	Default	0	0	Send	0
I/0 13	Default	0	0	Send	0
I/0 14	Default	0	0	Send	0
I/0 15	Default	0	0	Send	0

Analog inputs	Mode	Param1	Param 2		Sample
AN0	Default	0	0	Send	0
AN1	Default	0	0	Send	0
AN2	Default	0	0	Send	0
AN3	Default	0	0	Send	0



Input Sample Time [s] 0.10

External I/O	Mode	Param1	Param 2		Sample
I/0 0	Default	0	0	Send	0
I/0 1	Default	0	0	Send	0
I/0 2	Default	0	0	Send	0
I/0 3	Default	0	0	Send	0
I/0 4	Default	0	0	Send	0
I/0 5	Left RPM	0	0	Send	0
I/0 6	JR Manual input	0	0	Send	0
I/0 7	Default	0	0	Send	0
I/0 8	Default	0	0	Send	0
I/0 9	JR Manual input	0	0	Send	0
I/0 10	Default	0	0	Send	0
I/0 11	Default	0	0	Send	0
I/0 12	Default	0	0	Send	0
I/0 13	Default	0	0	Send	0
I/0 14	Default	0	0	Send	0
I/0 15	Default	0	0	Send	0

Analog inputs	Mode	Param1	Param 2		Sample
AN0	Default	0	0	Send	247 mV
AN1	Default	0	0	Send	2 mV
AN2	Default	0	0	Send	246 mV
AN3	Default	0	0	Send	0 mV

It was found that this understanding was incorrect, both the TPU_A and TPU_B connection's I/O settings had to be set to "JR Manual Input" for both antennas to work. Only one antenna was functioning in the last flight. Initially only I/O 9 was designated "JR Manual Input". The TPU_B that the JR was plugged into corresponded with I/O 6; therefore, I/O 6 was changed from "Default" to "JR Manual Input".

1.20.2. Flight Plan

1) Manual R/C pilot flight with JR

- Check comms

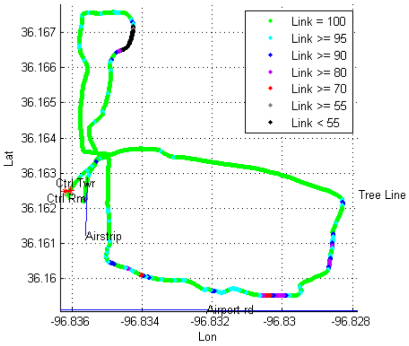
2) Switch to autopilot

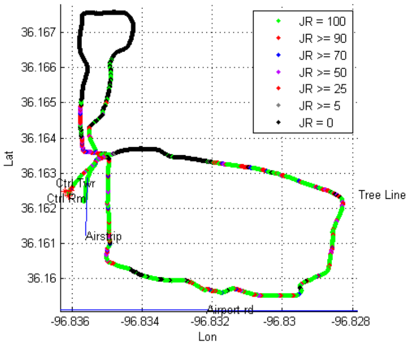
3) Perform aileron, elevator, and rudder doublet maneuvers, Doublet Log File Method

4) Further gain tuning if necessary

1.20.3. Flight Analysis

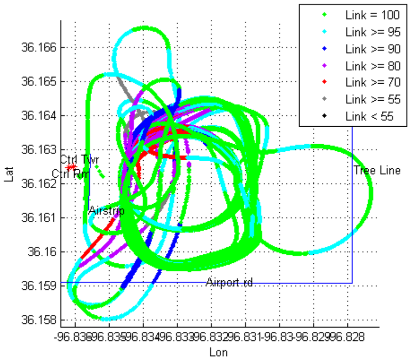
Ground Comms Check

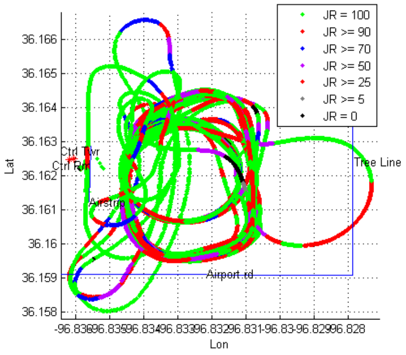




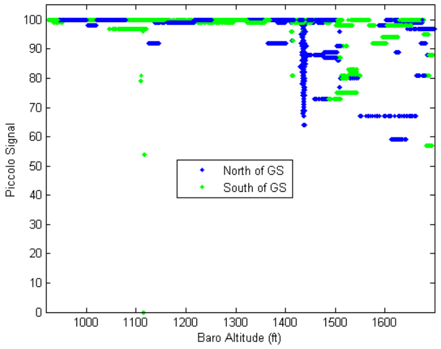
The piccolo packet loss was much improved from Flight 2. The area of lost comms to the north was the result of loss of line of sight. The aircraft was behind a tree line. The manual JR signal exhibited similar characteristics. Now the JR signal is no longer 0% or 100%, there are now in between percentages. The manual remote was setting out on the air strip pointing to the east. A large part of the loss of comms to the north was likely because of the orientation of the transmitter and a loss of line of sight.

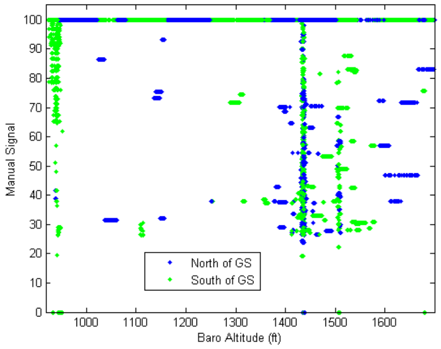
Flight Comms





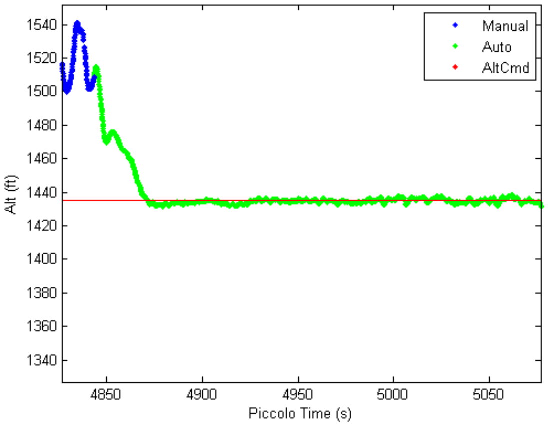
The piccolo signal strengths were much improved from Flight 2. The most packet loss occurred when the aircraft was banking in such a manner that the antenna was being shielded from the control tower by the fuselage and loss of line of sight began to occur. There are still some areas of packet loss that should not occur. It is possible that our fin antenna is not that good. JR had some signal strength losses as well. It is likely that some of the comms loss could be due to the carbon fiber fuselage.



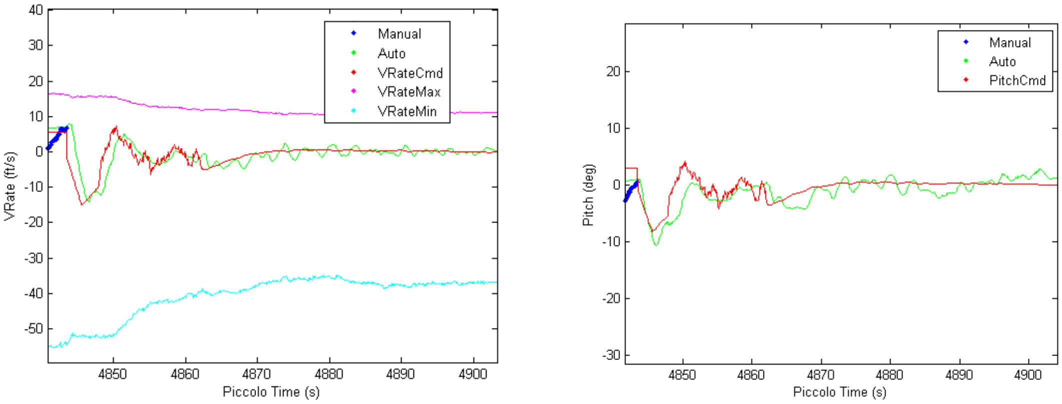


The figures aboe showed that there was no correlation between the altitude and comms loss. The vast majority of the flight occurred at 1435ft, with some flight time at 1510ft and 1700ft.

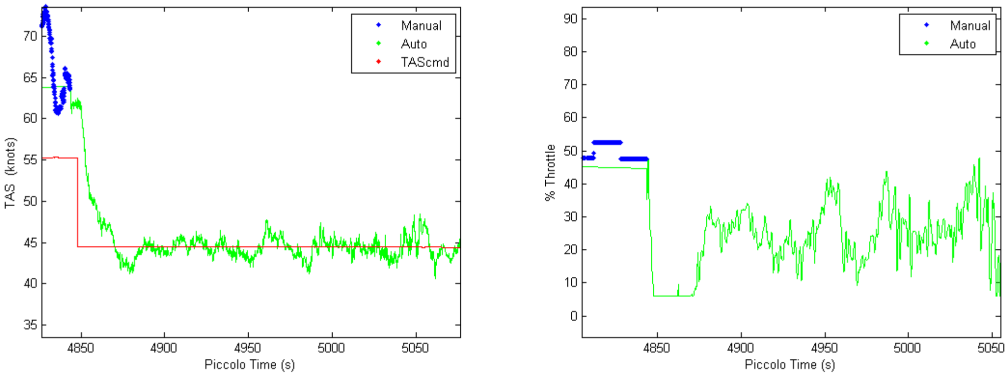
Autopilot Takeover



The figure above shows the altitude when the autopilot first switched over. The aircraft descended down to the commanded altitude and then held altitude without any major oscillations.

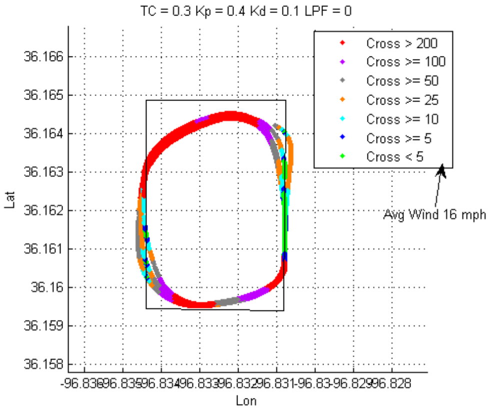


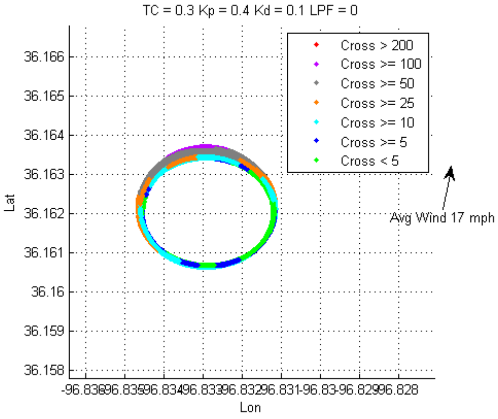
The figure above shows the vertical rate command was a somewhat dramatic descent initially and then the descent rate decreased. Similarly the pitch command behaved in the same manner as it should. The plot on the right shows that the aircraft pitched down to about -9 degrees before it began to decrease the descent rate. There was a time period of quick small VRate command oscillations from about 4848 to 4862 seconds. The actual vertical rate of the aircraft had a lot of small peak oscillations, but it didn’t seem to be affecting overall altitude performance. Altitude control appears to be stable over all.



The figure above shows the airspeed performance on the first descent after switching to autopilot control. Initially the aircraft was traveling over the commanded airspeed by about 7 knots. The autopilot immediately dropped throttle to the minimum throttle setting, 7%. After the aircraft reached the commanded airspeed it dropped to about 41 knots, 4 knots below the commanded airspeed. Looks like energy control can be tuned to reduce the airspeed undershoot. The airspeed did stabilize afterwards though.

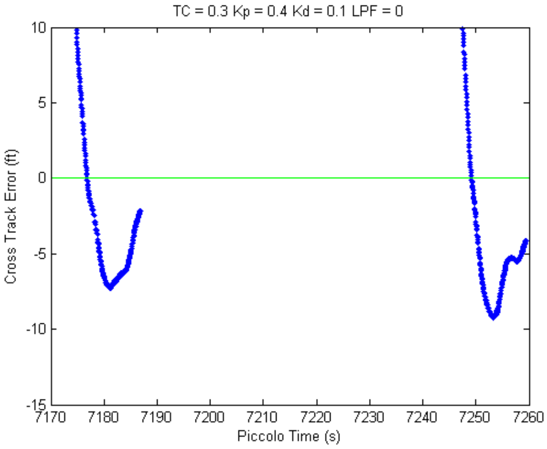
Lateral Tracking





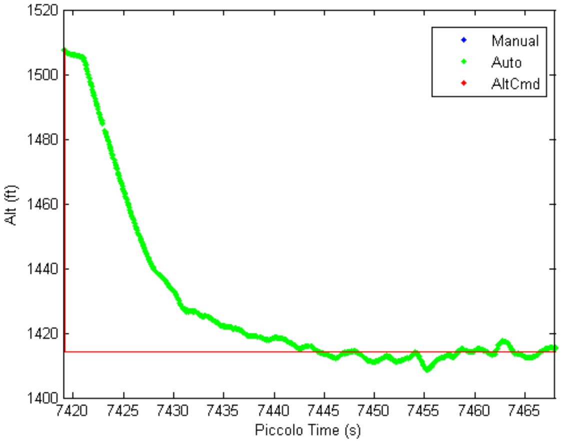
During flight there were no noticeable issues with lateral tracking. The aircraft crabbed to zero the side force flying in the wind and always converged on the track path. The plot on the left shows the aircraft navigating “Box2”. The autopilot cut some corners because of the wind and how short the legs of the flight were. The plot on the right shows the aircraft navigating

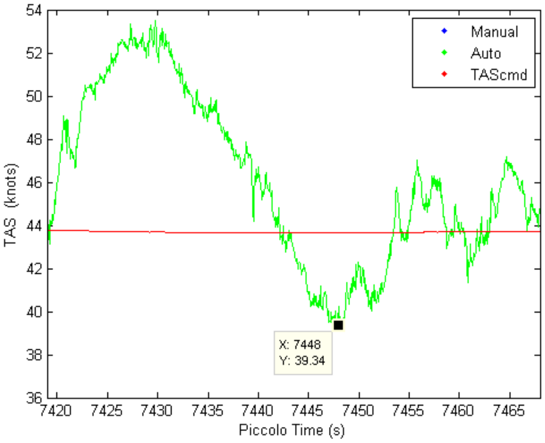
“Loiter1”. As indicated by the wind arrow there was substantial winds. Wind gusted between 15 – 20 mph. The wind accounted for the cross track error at the northwest of the loiter waypoint and caused the early preturns in the box patterns. It is likely that the wind gusts combined with the small radius of the loiter contributed to the loiter tracking being off by as much as 50 feet at one point.



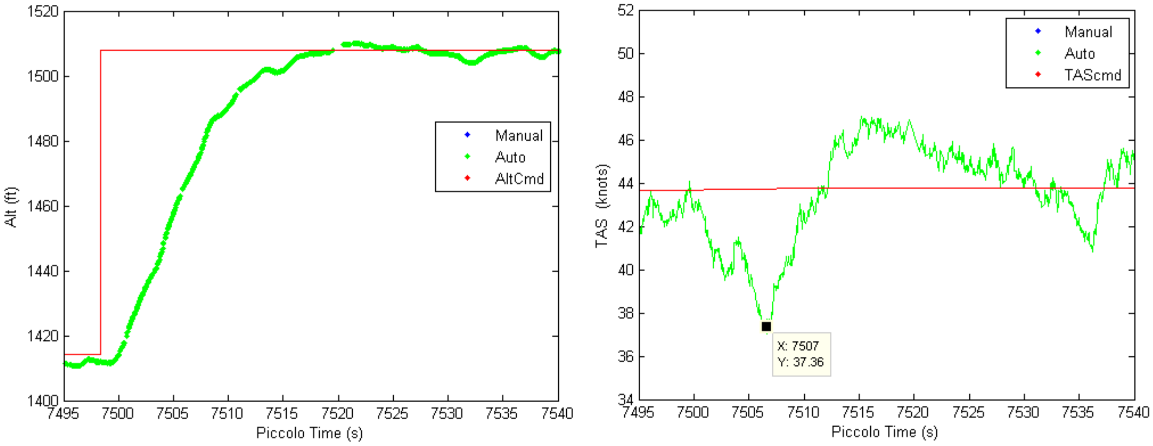
The legs of the flight path weren’t long enough to allow the aircraft to converge to the track path. From this flight tracking control appears to be okay. I plan on flying longer flight paths in the next flight.

Airspeed





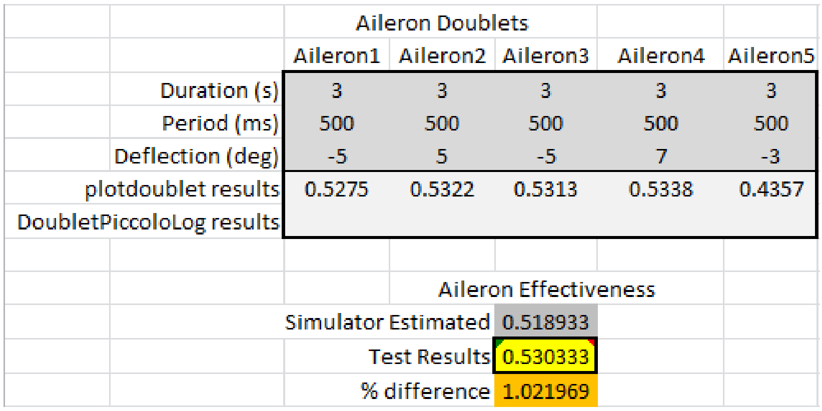
The figure above depicts the airspeed undershoot after a 100 ft descent. After the descent the airspeed dropped to 39 knots, about 4 knots below the commanded airspeed.



The figure above depicts the airspeed undershoot after a 100 ft climb is commanded. In the beginning of the climb the airspeed dropped to 37 knots, about 5 knots below the commanded airspeed. Both figures indicate that energy control can be tuned to increase performance.

Aileron Doublet Analysis

A total of five aileron doublet maneuvers were performed. The doublets used the Doublet Log File method. All five doublets had a period of 500 milliseconds, and a duration of 3 seconds. The doublets all consisted of only one aileron deflection each test. The following deflections were tested: -5, 5, -5, 7, and -3 degrees.



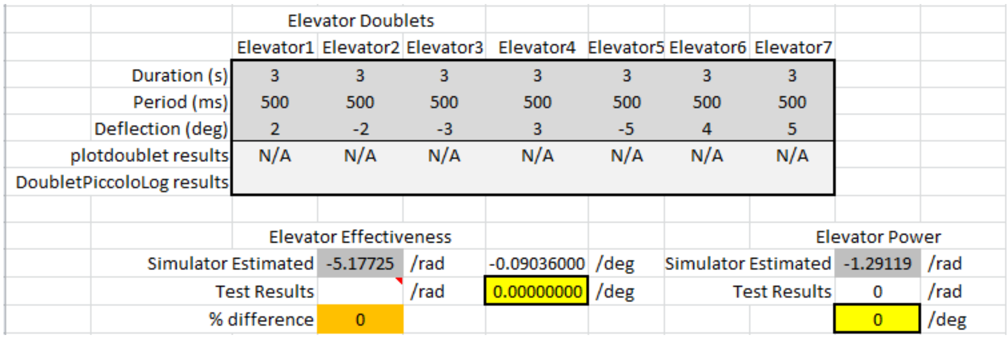
	Aileron Doublets				
	Aileron1	Aileron2	Aileron3	Aileron4	Aileron5
Duration (s)	3	3	3	3	3
Period (ms)	500	500	500	500	500
Deflection (deg)	-5	5	-5	7	-3
plotdoublet results	0.5275	0.5322	0.5313	0.5338	0.4357
DoubletPiccoloLog results					
			Aileron Effectiveness		
		Simulator Estimated	0.518933		
		Test Results	0.530333		
		% difference	1.021969		

The figure above depicts the results of each aileron doublet test. The aileron effectiveness was calculated by removing the highest (0.5338) and lowest (0.4357) values and averaging the rest.

$$Aileron\ Effectiveness = \frac{0.5275 + 0.5322 + 0.5313}{3} = 0.530333\ /rad$$

The final result of 0.530333 was only a 2% difference from the estimated value; therefore, the aileron effectiveness was not changed. The aileron effectiveness vehicle parameter remained at its initial value of 0.518933 /rad.

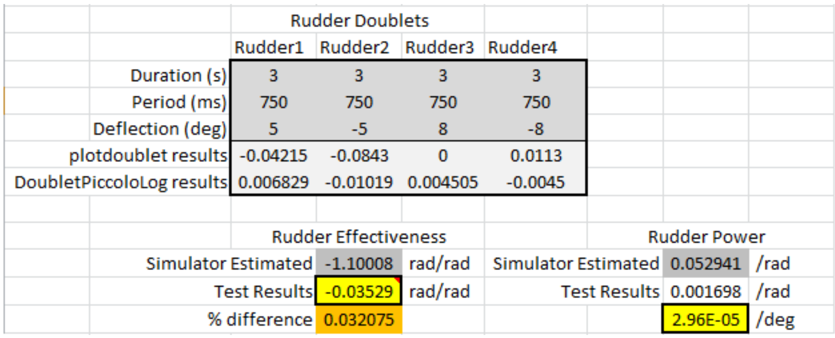
Elevator Doublet Analysis



			Elevator Doublets							
		Elevator1	Elevator2	Elevator3	Elevator4	Elevator5	Elevator6	Elevator7		
	Duration (s)	3	3	3	3	3	3	3		
	Period (ms)	500	500	500	500	500	500	500		
	Deflection (deg)	2	-2	-3	3	-5	4	5		
	plotdoublet results	N/A	N/A	N/A	N/A	N/A	N/A	N/A		
	DoubletPiccoloLog results									
			Elevator Effectiveness						Elevator Power	
		Simulator Estimated	-5.17725	/rad	-0.09036000	/deg		Simulator Estimated	-1.29119	/rad
		Test Results		/rad	0.00000000	/deg		Test Results	0	/rad
		% difference	0						0	/deg

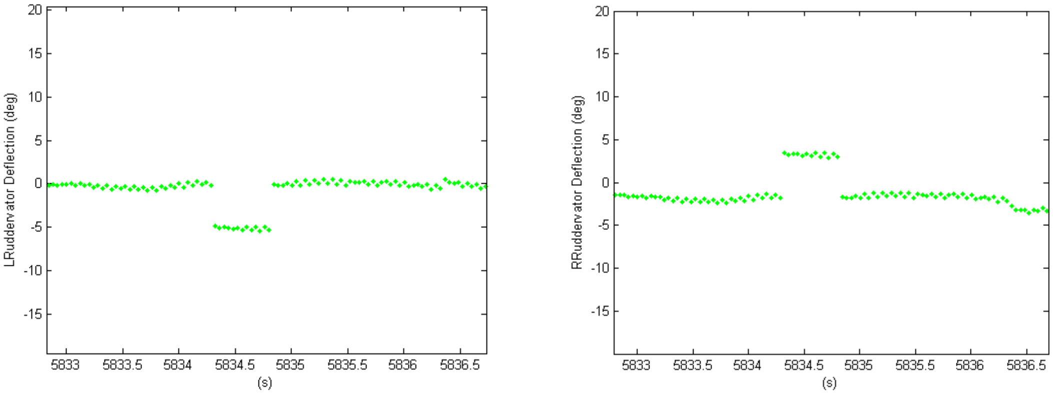
A total of seven elevator doublet maneuvers were performed. The doublets used the Doublet Log File method. All seven doublets had a period of 500 milliseconds and a duration of 3 seconds. The analysis provided no results. There was too much noise in the C_L measurements to be able to determine where to place the 3rd and 4th points on the C_L subplots.

Rudder Doublet Analysis



	Rudder Doublets					
	Rudder1	Rudder2	Rudder3	Rudder4		
Duration (s)	3	3	3	3		
Period (ms)	750	750	750	750		
Deflection (deg)	5	-5	8	-8		
plotdoublet results	-0.04215	-0.0843	0	0.0113		
DoubletPiccoloLog results	0.006829	-0.01019	0.004505	-0.0045		
	Rudder Effectiveness				Rudder Power	
	Simulator Estimated	-1.10008	rad/rad	Simulator Estimated	0.052941	/rad
	Test Results	-0.03529	rad/rad	Test Results	0.001698	/rad
	% difference	0.032075			2.96E-05	/deg

A total of four rudder doublet maneuvers were performed. The doublets used the Doublet File method. All four doublets had a period of 750 milliseconds and a duration of 3 seconds. Analysis of the doublet file data yielded a rudder effectiveness of nearly 0. The doublet plots indicated no change in heading due to rudder deflection. The small changes were simply the trend of the heading rate at the time the maneuvers were conducted. There were no signs of response from rudder deflections. Additionally the doublets were also analyzed via the Piccolo Log File method. The results were essentially the same. Each rudder effectiveness was nearly 0, and the yaw rate did not appear to change as a result of the rudder deflections.

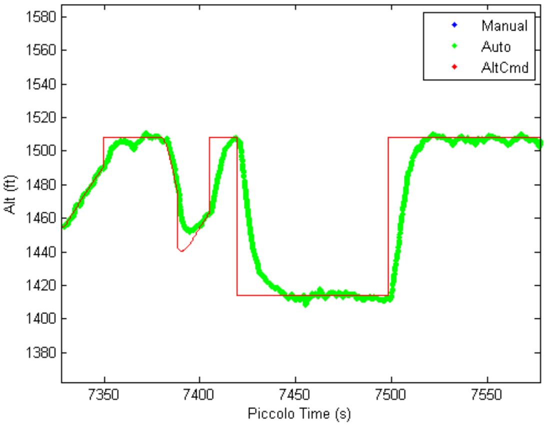


The ruddervator actuator deflections were analyzed to make sure that the rudder deflection commands were properly mixed and the ruddervators weren’t simply deflecting opposite directions and cancelling each other out. The figure above depicts the ruddervator deflections of the Rudder1 which was a 5 degree rudder deflection command. The figures show that the left ruddervator deflected negative while the right ruddervator deflected positive. The deflections were consistent with rudder deflection in ruddervators because the ruddervator deflection sign convention follows that of elevators. The left ruddervator deflected up and the right ruddervator deflected down. Both deflections created a positive yawing moment; thus, they did not cancel each other out.

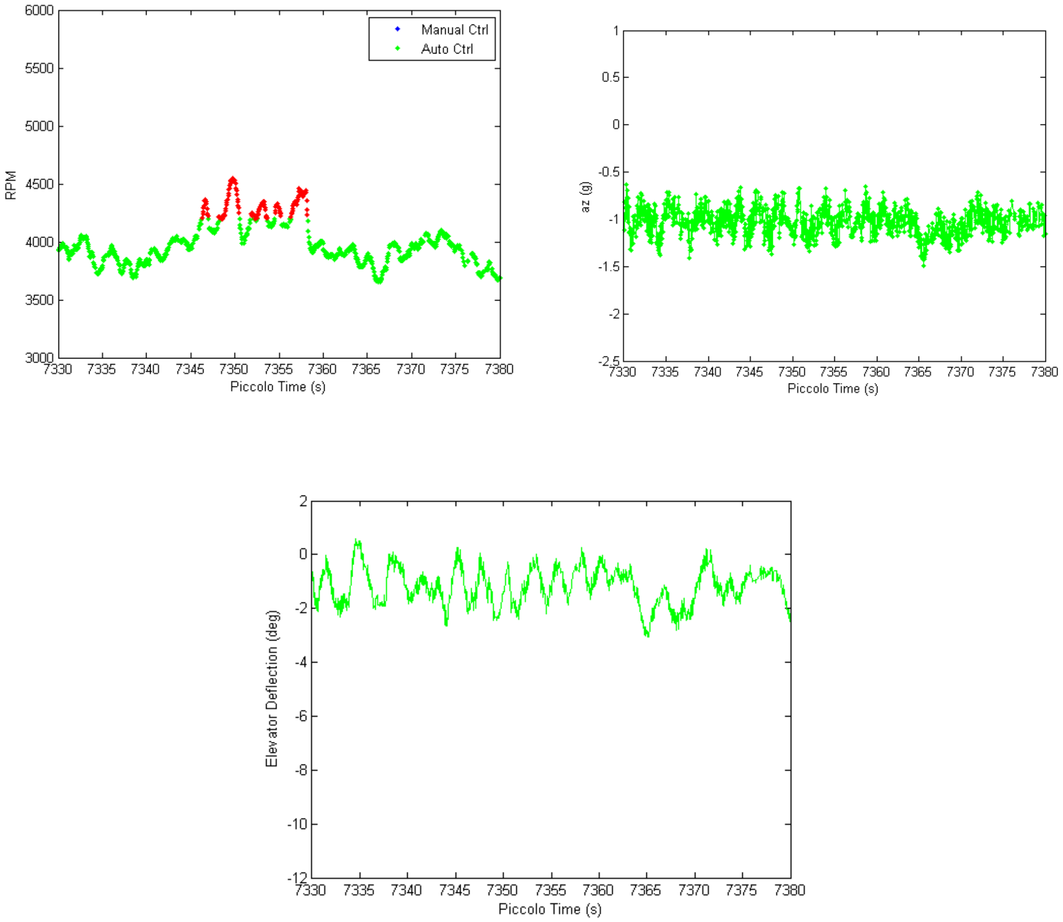
2nd Analysis of Flight 3 Aug 27, 2013

After Flight 4 I realized that we had resonating vibrations in the Z – Accelerometer readings between 4200 – 5200 rpms. Looking back at the climbs from Flight 3 there wasn't any large overshoot that would raise any red flags.

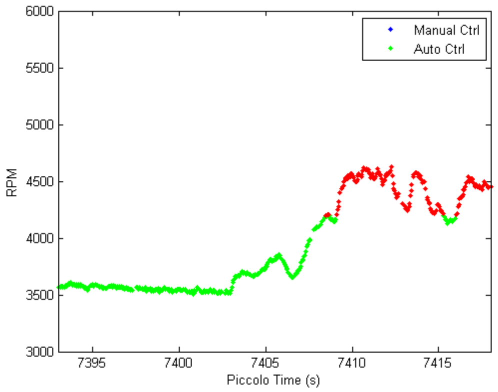
Climb and Descents

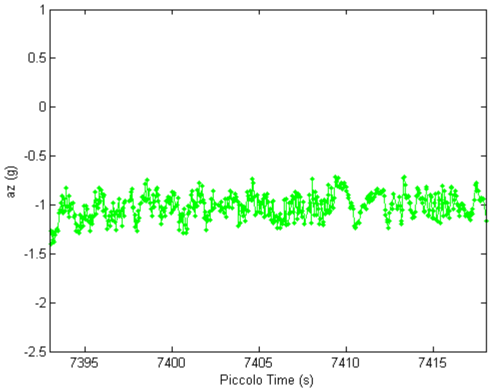


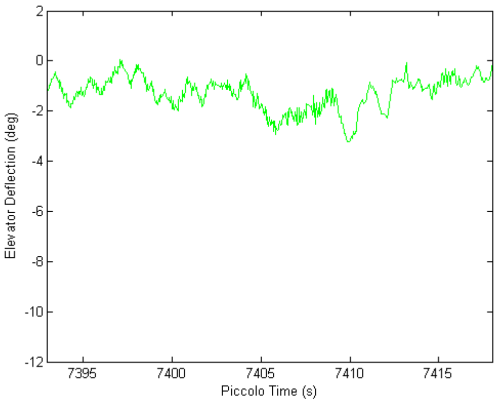
In flight it appeared that the autopilot was able to control altitude throughout the 3 climbs and 2 descents. The first climb, shown in Figure 17, shows that the aircraft descended almost 6 feet under the commanded altitude right after the command altitude was reached at 7375 seconds. After the last climb the aircraft oscillated +- 4 ft over the commanded altitude; however, this is not even close to the 50 foot overshoot from Flight 4.



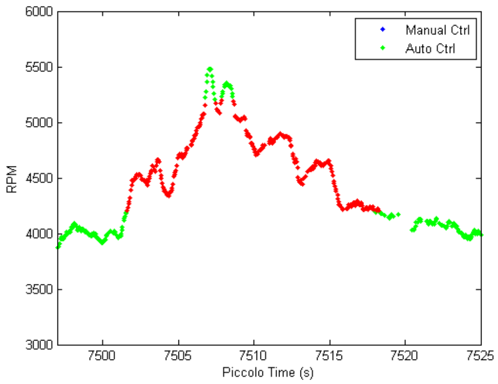
I analyzed all three climbs closely and found that none of the vibration issues that existed in Flight 4 appear to have existed in Flight 3. The figure above shows plots from the first climb. The plots are to the same scale as the plots of Flight 4 that all show heavy resonating vibration issues. The RPM did indeed enter the resonant range of 4200 – 5200 RPMs, and the z accelerometer data is noisy; however, not as bad as Flight 4.

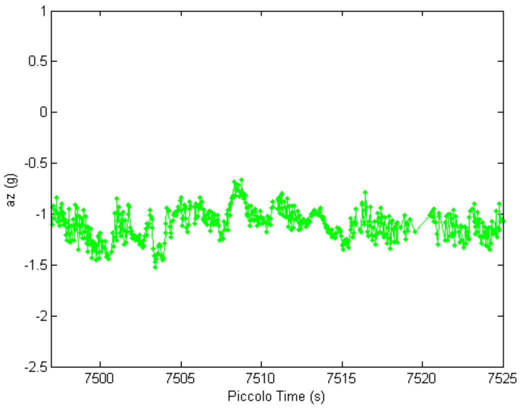


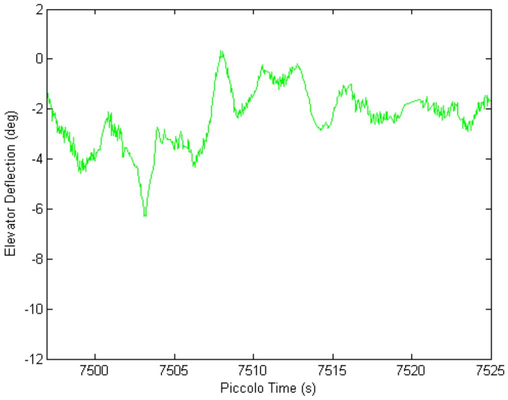




The plots above represent Climb 2, and the plots below represent Climb 3.



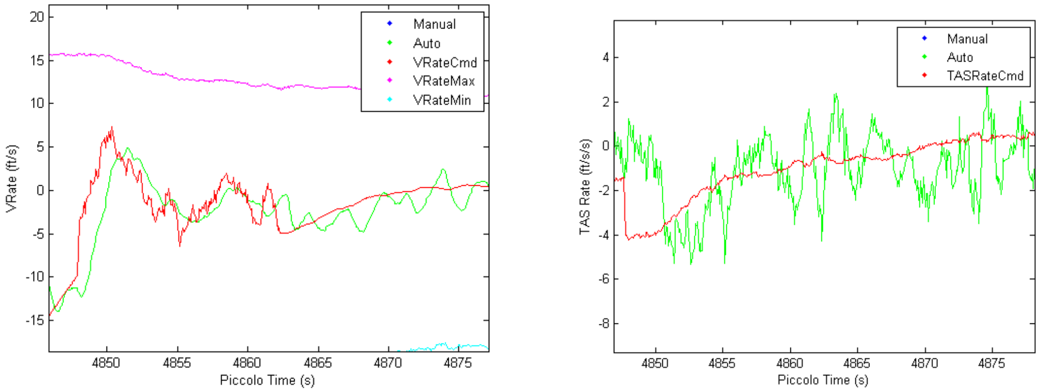




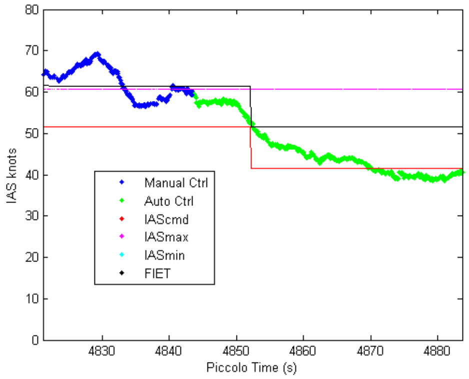
There was no indication from Flight 3 that there were resonant vibration issues.

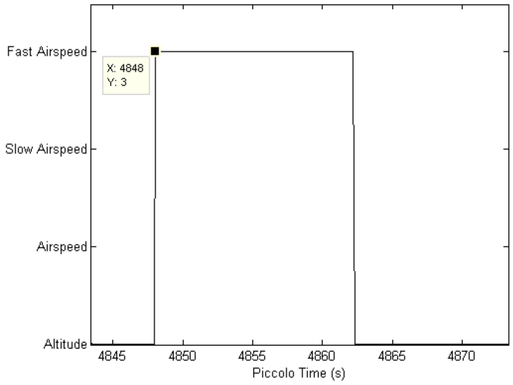
Fast Airspeed Mode

I initially missed the fact that the aircraft had gone into Fast Airspeed Mode, Lon Mode 3, (airspeed control). This flight occurred before I was aware of the existence of Lon Modes. The vertical rate plots showed signs of fast airspeed mode.



The VRate command oscillated just like it does in airspeed control because in airspeed control the VRate command is derived from the TASRate error which certainly oscillates frequently.





Additionally another sign of Fast Airspeed Mode is shown in the IAS plot above. The aircraft violated the Fast IAS error Threshold and caused the autopilot to switch to Lon Mode 3.

1.21. Flight 4

1.21.1. Pre-Flight Autopilot Changes

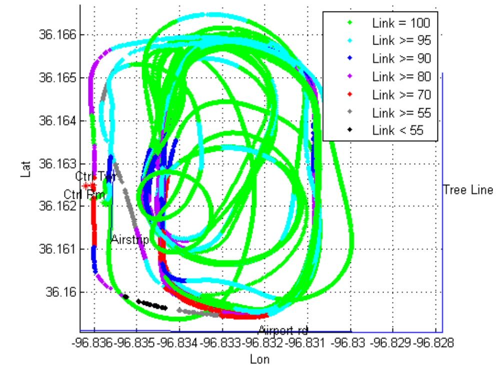
1) The Fast IAS error threshold was increased to 15 knots to attempt to keep the autopilot from entering Fast Airspeed Mode (Lon Mode 3) during descents.

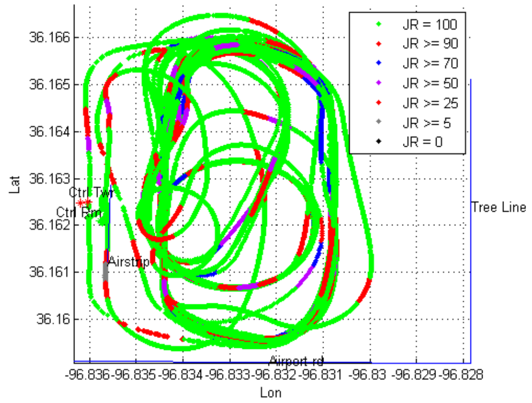
1.21.2. Flight Plan

1) Test CL at zero elevator
2) Elevator Doublets (Doublet Log File Method)
3) Rudder Doublets (Doublet Log File Method)
4) Tune Energy Control
5) Fly patterns for Dr. Gaeta

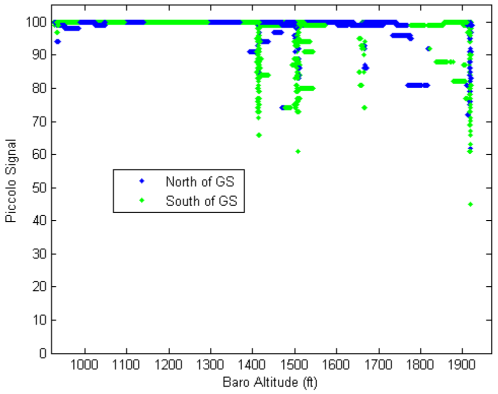
1.22. Flight Analysis

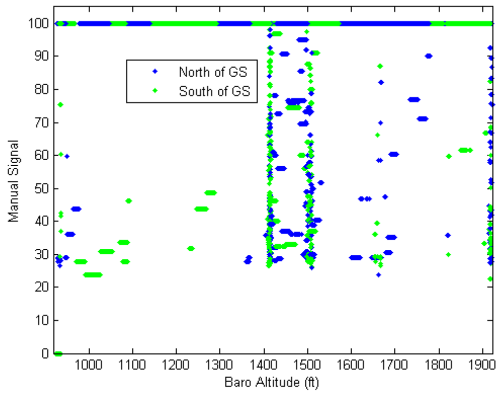
Flight Comms





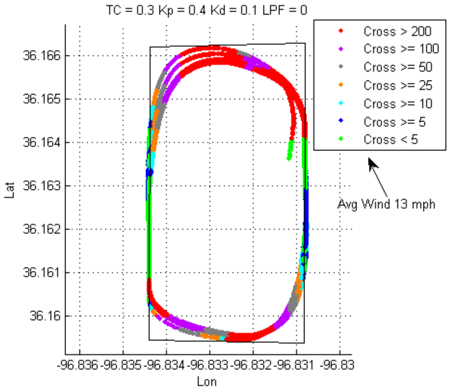
The figure above shows that there was still some piccolo link packet loss. There was significant loss on one of the passes to the south where Noctua flew over the control room. JR had some signal strength losses similar to Flight 3. There really isn't anything else that can be done to the antennas, it must just be carbon fiber structure interference with line of sight.

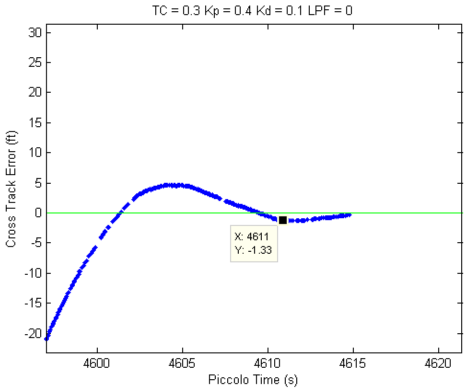




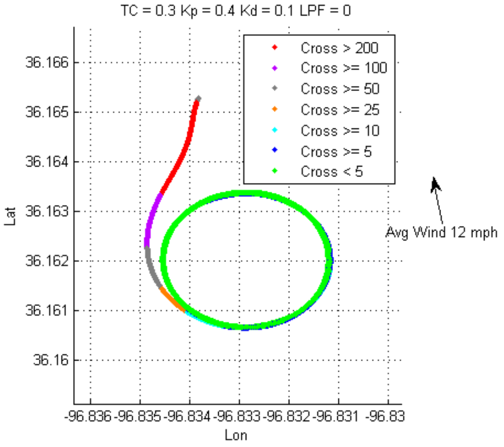
Similar to Flight 3 the figure above shows that there wasn't any correlation between comms loss and altitude. The aircraft flew flight plans at 1400, 1500, 1600, and 1900 feet. 1400 feet was the minimum altitude we flew at so it makes sense that there isn't any comms loss below 1400 feet. The comms plots indicated that the majority of piccolo packet loss occurred south of the control room.

Lateral Tracking



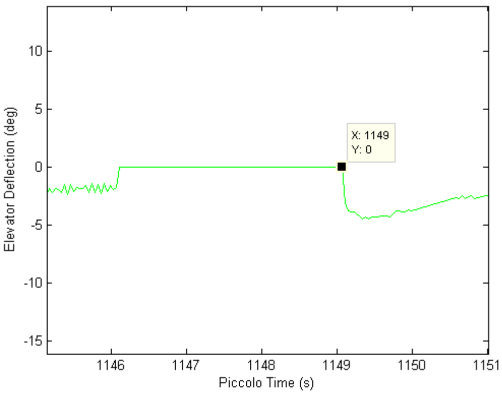


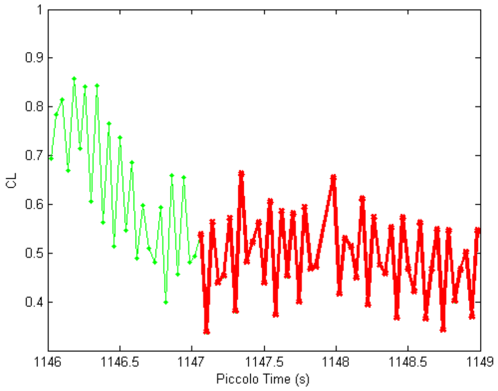
During flight lateral tracking appeared to be working just fine. I extended the north south legs from Flight 3 so the autopilot would have a chance to settle and hold a straight path. The wind arrow from the figure above shows that there was a decent magnitude of wind blowing N, NW. It should be noted that the box pattern was flown counter clockwise so that the aircraft would be flying upwind during the leg closest to the airfield. The significant preturn going into the west leg can be attributed to the wind. The aircraft stayed converged to within 5 feet of the target track path. The figure above shows the response of one of the passes of the west leg. The wind likely contributed to the initial overshoot of about 5 feet. The aircraft traveled to just over 1 foot inside the track path and began to cross the path as it targeted the next waypoint.



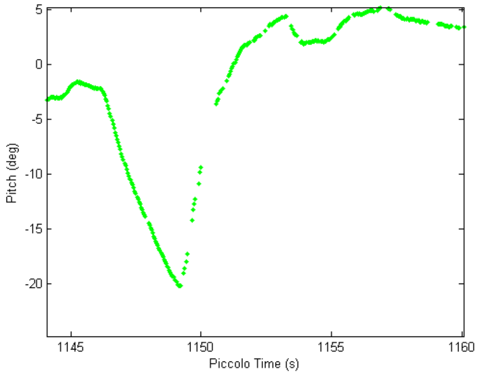
Additionally the loiter tracking, shown in the figure above, was much more accurate than Flight 3. The wind was lower and also did not have large gusts. Once the aircraft converged to less than 5 feet it stayed within 10 feet the entire time. Lateral Tracking is great; there should not be any further need to analyze lateral tracking and there is no need to analyze roll performance.

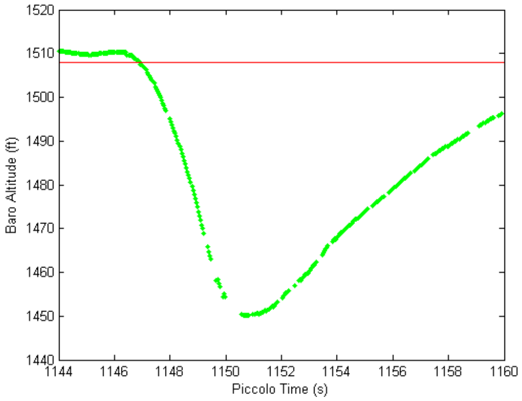
CL at Zero Elevator





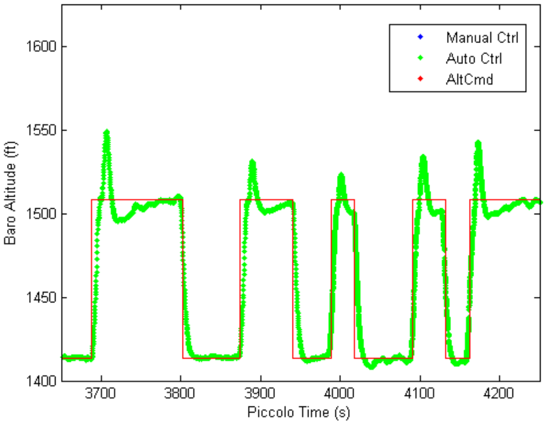
The figure above depicts the 0 degree elevator deflection results. I held 0 degrees elevator deflection for 3 seconds which was nearly too long. As shown below in the figure below the aircraft pitched down to -20 degrees and the aircraft lost 60 feet in altitude.



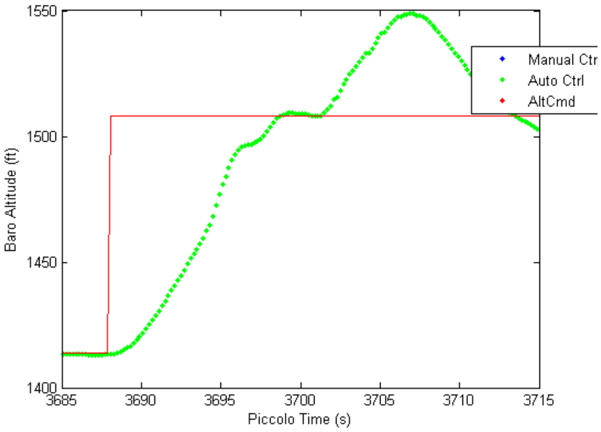


I averaged the CL values that the CL appeared to settle on. They are highlighted red in Figure 8. The average CL was 0.4910.

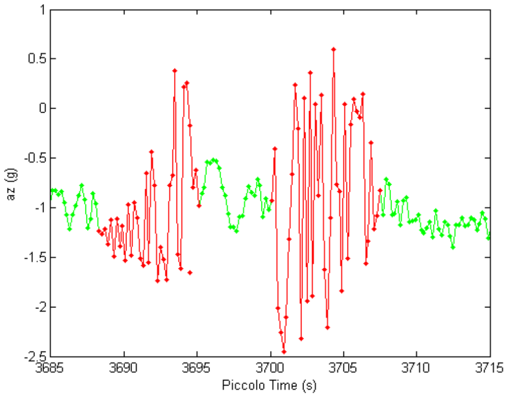
Altitude Control

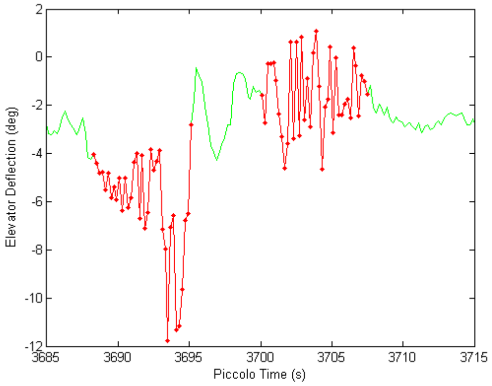


As noted in flight, we seemed to be overshooting the altitude command after climbs. The four climbs in the figure above overshot from 30 – 50 feet. The descents; however, did not exhibit drastic undershoots in altitude and additionally the autopilot was able to maintain a constant altitude. I decided to analyze the largest overshot climb in greater detail.



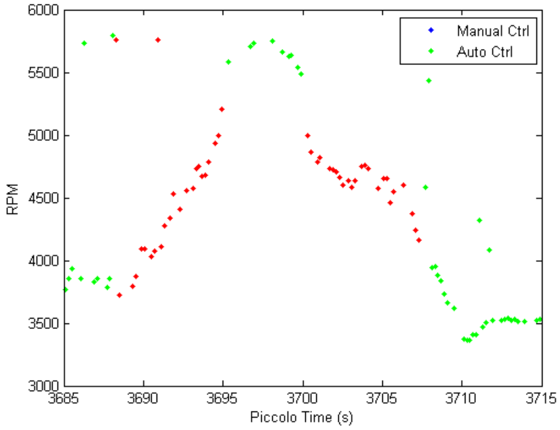
The figure above depicts the first climb. The overshoot went up to 50 feet.





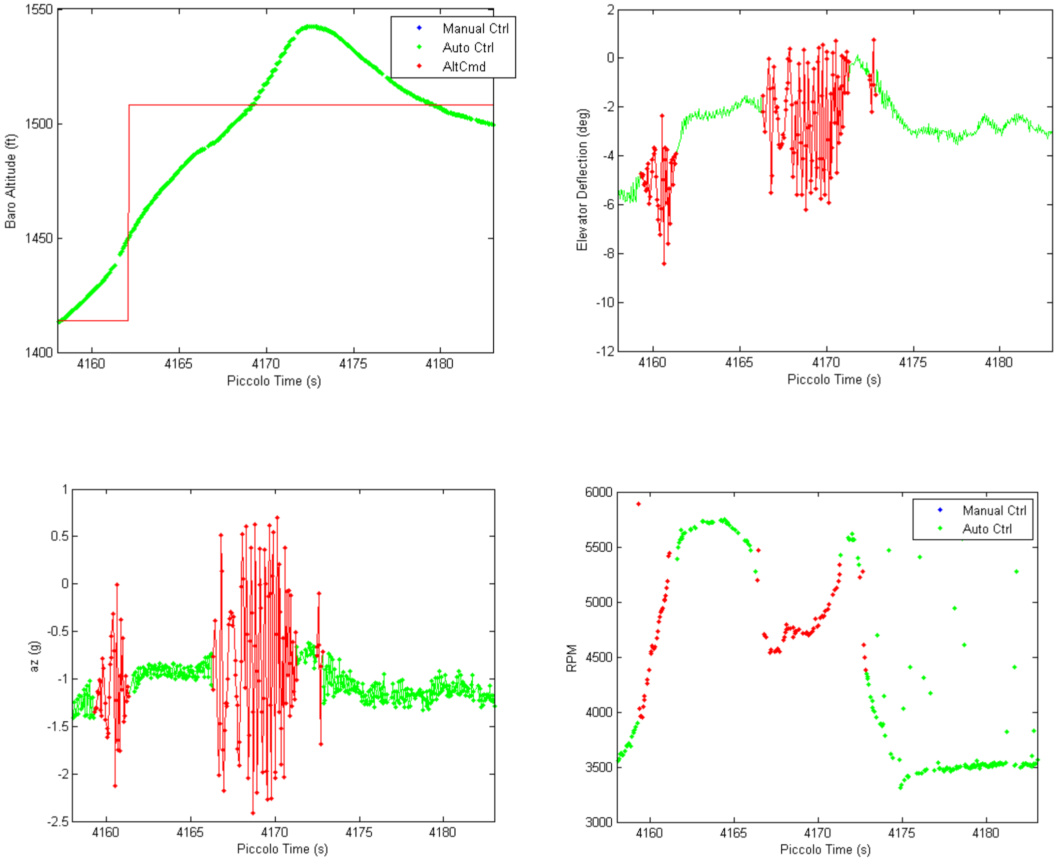
Ultimately Elevator deflections effect altitude performance and elevator deflections are directly commanded by the Z – Acceleration control loop, so I looked at the elevator deflections and z acceleration measurements. The figure above shows two time periods, highlighted in red, where there were hard and quick oscillations in the z acceleration measurement. Similarly the figure above shows the same two time periods had large and quick elevator oscillations.

I checked the descent from climb 1 and the hard z – accelerometer and elevator oscillations that occurred in the climb did not occur on descent. I eventually realized that the only difference between climb and descent was the throttle. The autopilot was throttling up during climbs but not during descents.



The figure above depicts the RPMs during climb 1 where the red data represents the time periods of heavy and quick oscillations. The figure above showed that the oscillations occurred in a corresponding RPM range. The RPM range was approximately 3800 – 5300. Since the RPM data was extremely noisy it was hard to get an exact value for the thresholds as clearly some RPM data was missing.

I looked the fourth climb to analyze the RPM range for that climb.



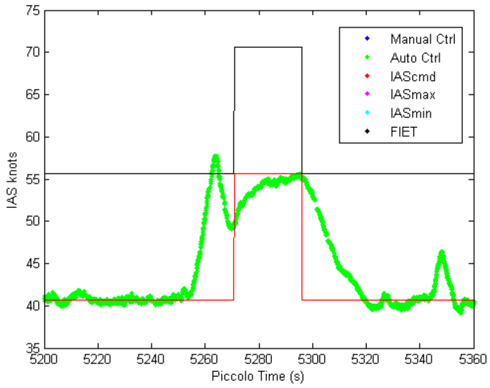
The figure above depicts data from the fourth climb. Again there was an overshoot of nearly 40 feet that coincided with heavy quick oscillations in the z – acceleration measurement and elevator deflection. The RPMs that coincided with the oscillations were in the range of 4000 – 5300.

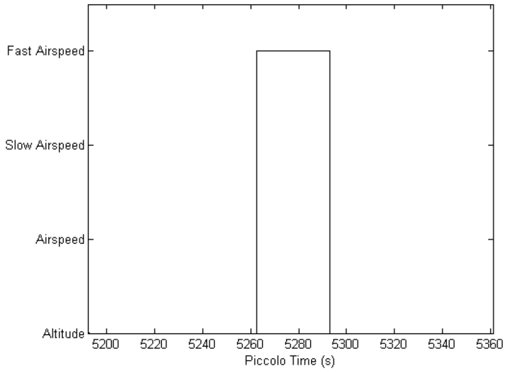
I came to the conclusion that resonating vibrations in the RPM range of 4000 – 5300 RPMs were the reason for the altitude overshoots.

Energy Control

During the attempted energy control tuning the controller telemetry was not enabled, so the EnergyControl GUI could not be used to analyze energy control performance. Additionally the vibrations during climb could have influenced the initial airspeed undershoot going into the climb and the airspeed overshoot after the climbs. Energy Control still needs to be tuned in future flights.

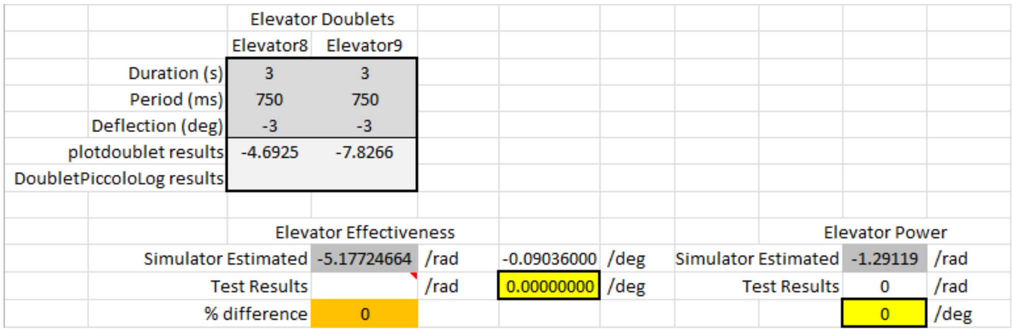
Fast Airspeed Mode





Even though the Fast IAS error Threshold was raised to 15 knots the autopilot still went into Lon Mode 3 during the final descent of the aircraft. I need to either decrease the descent max fraction, increase the Fast IAS error Threshold even more, or accept that it uses fast airspeed mode during descents.

Elevator Doublets

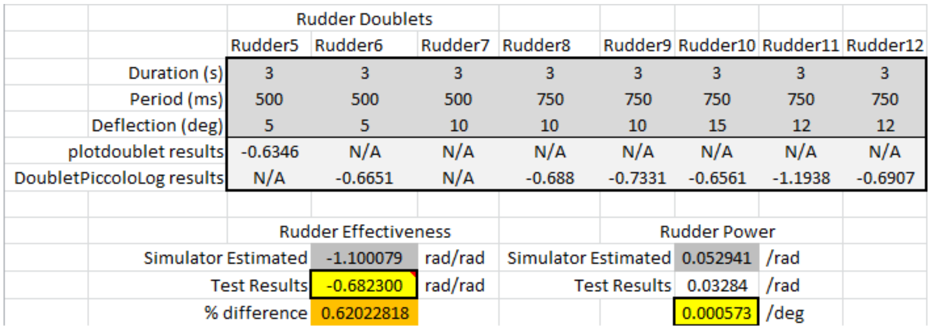


	Elevator Doublets	
	Elevator8	Elevator9
Duration (s)	3	3
Period (ms)	750	750
Deflection (deg)	-3	-3
plotdoublet results	-4.6925	-7.8266
DoubletPiccoloLog results		

	Elevator Effectiveness					Elevator Power	
Simulator Estimated	-5.17724664	/rad	-0.09036000	/deg	Simulator Estimated	-1.29119	/rad
Test Results		/rad	0.00000000	/deg	Test Results	0	/rad
% difference	0					0	/deg

The elevator doublets had too much noise to be able to trust the results. The CL response was determined in the analysis just be eyeballing a mean line that could exist throughout the CL data. The elevator effectiveness calculation was very sensitive to minor changes in the perceived mean line of the CL data. In light of the existing resonant vibration problem and the high error in the calculated results the results were ignored.

Rudder Doublets



	Rudder Doublets							
	Rudder5	Rudder6	Rudder7	Rudder8	Rudder9	Rudder10	Rudder11	Rudder12
Duration (s)	3	3	3	3	3	3	3	3
Period (ms)	500	500	500	750	750	750	750	750
Deflection (deg)	5	5	10	10	10	15	12	12
plotdoublet results	-0.6346	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DoubletPiccoloLog results	N/A	-0.6651	N/A	-0.688	-0.7331	-0.6561	-1.1938	-0.6907

	Rudder Effectiveness			Rudder Power	
Simulator Estimated	-1.100079	rad/rad	Simulator Estimated	0.052941	/rad
Test Results	-0.682300	rad/rad	Test Results	0.03284	/rad
% difference	0.62022818			0.000573	/deg

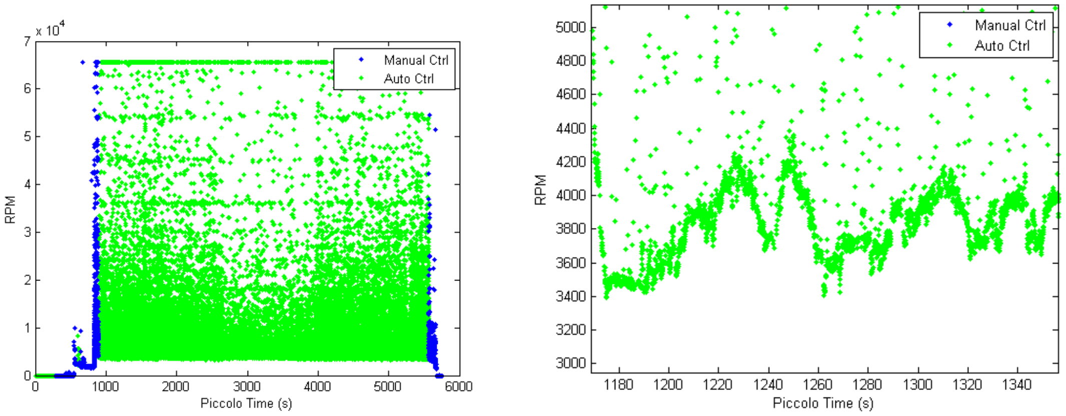
Initially the rudder doublets were meant to be analyzed via the Doublet File method; however, rudder doublets 6 – 12 were performed on north and south legs of the flight pattern so they could not be analyzed with the Doublet File method. Rudder doublets 5 and 7 were performed in an area of high packet loss, so there was not enough data recorded by the piccolo log file to be able to accurately analyze the tests. The test result calculation used the plotdoublet result of rudder 5 with the DoubletPiccoloLog results of the rest of the rudder doublets. The highest and lowest values were dropped, rudder 11 and rudder 10, and the other results were averaged to come calculate the rudder effectiveness as -0.6823 rad/rad.

$$Rudder\ Effectiveness = \frac{-0.6346 \pm 0.6651 \pm 0.688 \pm 0.7331 \pm 0.6907}{5} = -0.6823$$

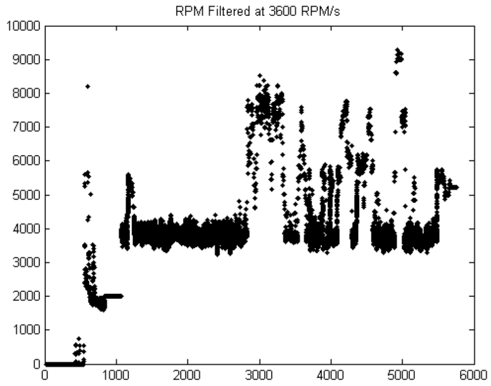
The new rudder power was calculated by multiplying the percent difference of the rudder effectiveness results from the simulator estimated result to the simulator estimated rudder power. The new rudder power was calculated to be 0.000573 /deg.

$$Rudder\ Power = 0.6202282 * 0.052941 = 0.000573\ /deg$$

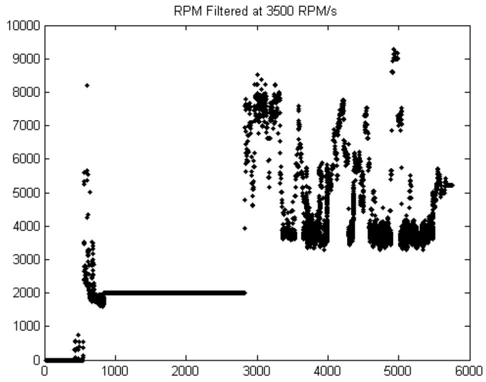
RPM



As depicted by the figure above, there was a lot of noise in the RPM data throughout the flight. The figure above also depicts a zoomed in portion of the RPM data where an RPM trend can be seen; however, observing the trend does not provide numerical data. “PiccoloRPMRateFilter” was utilized to test the whether or not the RPM filter used by the autopilot would be able to accurately filter out the noise.



RPM Filtered at 3600 RPM/s



RPM Filtered at 3500 RPM/s

The figure above depicts filtered RPM data. The plot on the left depicts the RPM data if it were filtered at 3600 RPM/s and the plot on the right depicts the RPM data if it were filtered at 3500 RPM/s. The plot on the left showed that the filter began to flat line at 1000 seconds, but excessively high RPM data still existed. The plot on the right showed that a filter of 3500 RPM/s was too high because it filtered out too much RPM data and created an RPM flat line for nearly 2000 seconds. Additionally 3500 RPM/s did not even filter out all of the noise as RPM data at > 6000 (6000 is the maximum RPM possible) still existed.

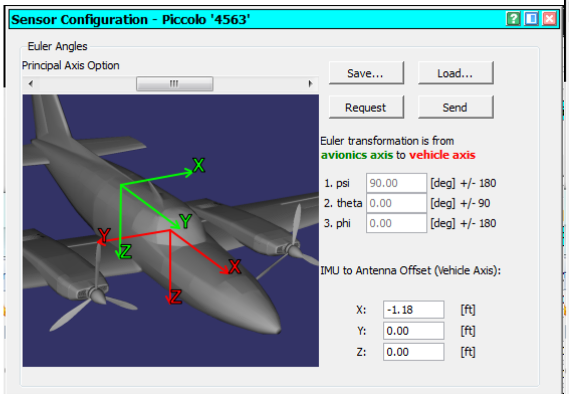
Ultimately we want to use RPM control so we determined that the filter would not work and we needed to determine what was causing the noise in the RPM data. A lot of time was spent running Noctua on the static stand and it was determined that the PMU is the cause of the RPM noise. It was decided to try to create a physical filter to attempt to filter the RPM signal into the Piccolo.

1.23. Flight 5

1.23.1. Pre-Flight Autopilot Changes

1) Re – mounted the piccolo. Added an extra arm for support on the mount to keep it from shaking violently in the z direction.

2) Orientation of the autopilot changed from (90,0,0) to (-90,0,180) (psi, theta, phi).



Sensor Configuration - Piccolo '4563'

Euler Angles

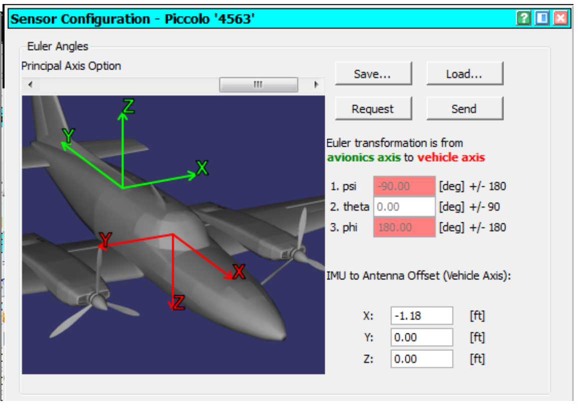
Principal Axis Option

Euler transformation is from avionics axis to vehicle axis

1. psi 90.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 0.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]



Sensor Configuration - Piccolo '4563'

Euler Angles

Principal Axis Option

Euler transformation is from avionics axis to vehicle axis

1. psi -90.00 [deg] +/- 180
2. theta 0.00 [deg] +/- 90
3. phi 180.00 [deg] +/- 180

IMU to Antenna Offset (Vehicle Axis):

X: -1.18 [ft]
Y: 0.00 [ft]
Z: 0.00 [ft]

3) Changed Rudder Effectiveness to -0.6823, which was determined from the rudder doublet test results in Flight 4. Also changed the Rudder Power to 0.000573 /deg.

4) Capacitors were added to the RPM signal line to attempt to filter out the RPM noise.

5) Added a Cnd scaling term to both ruddervator surfaces in the simulator file so that the simulator estimated rudder effectiveness and power would match the results of the doublet tests. The scaling value was the percent difference between the doublet test results and the original simulator estimated results.

$$Cnd\_scaler\_d5 = 0.62022818$$

$$Cnd\_scaler\_d6 = 0.62022818$$

6) The CL at zero elevator was changed to 0.4910, which was determined from the results of the CL at zero elevator deflection test.

1.23.2. Flight Plan

1) Climb and observe climb performance with new mount

2) Perform Elevator doublets

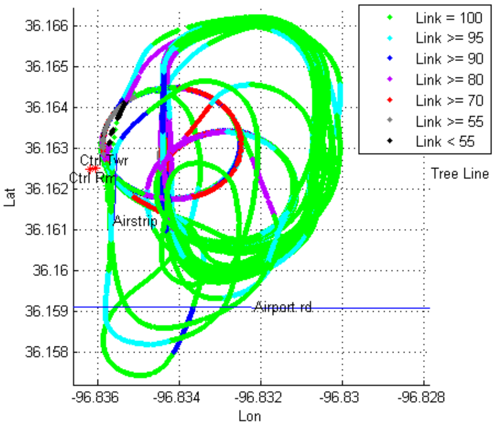
3) Tune Energy Control

4) Fly Loiter pattern for Dr. Gaeta at 500, 750, 1000 ft AGL

5) Fly some test approaches at 150 ft AGL to observe how well the autopilot holds constant altitude and tracks down the runway

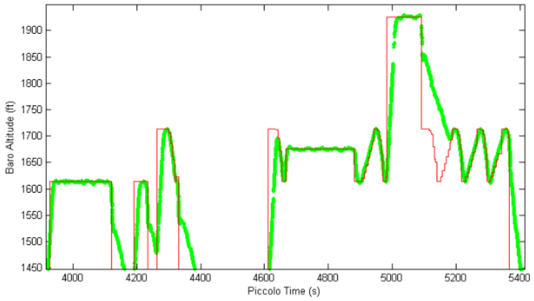
1.23.3. Flight Analysis

Flight Comms

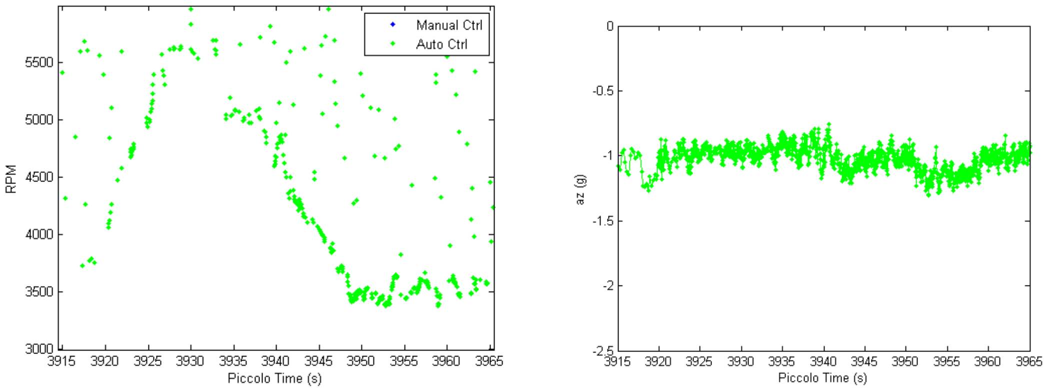


There is still packet loss in the same loiter area as the previous flights. There is much less packet loss flying to the south than there was in Flight 4. There was significant packet loss during the microphone loiters. It seems that just directly above the control tower is a bad spot. The losses must be due to the orientation of the aircraft, and shielding by the fuselage of the line of sight.

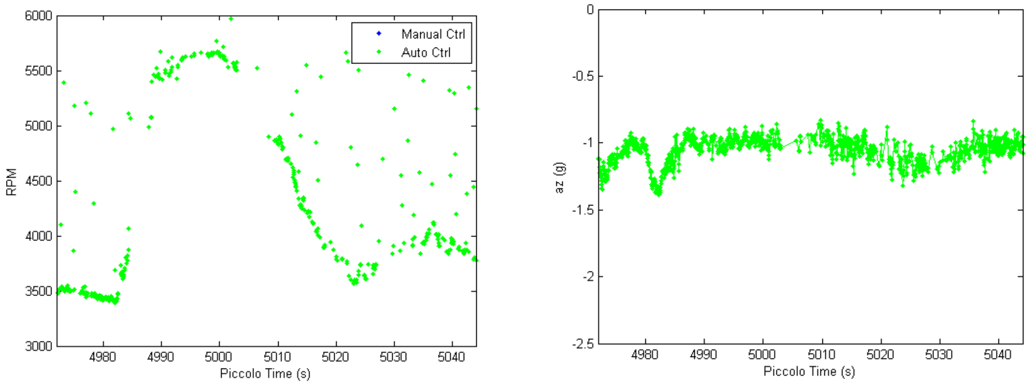
Altitude Control



In each climb there was nothing greater than a 5 foot overshoot which only occurred once. The altitude performance was much improved from Flight 4.



Climb 1

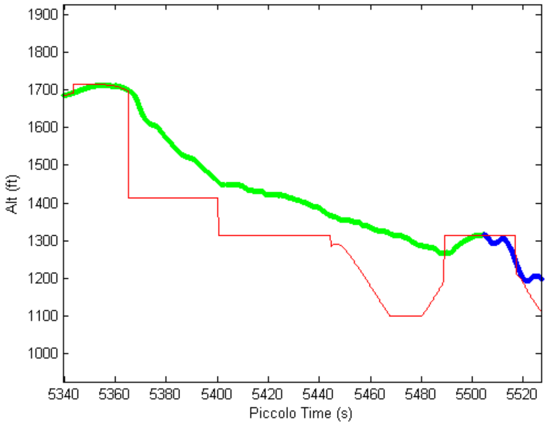


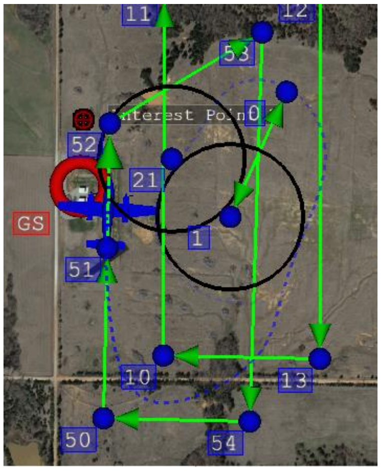
Climb 7

The figures above depict the RPM and Z – Accelerometer telemetry data from 2 of the climbs. In both cases the RPMs went through the resonant vibration range of 3800 – 5200 RPMs and showed no signs of resonant vibrations.

In conclusion the new mount fixed the resonating vibration issues.

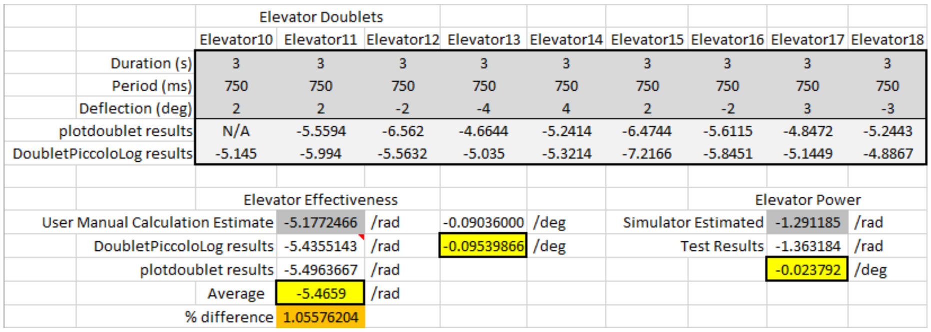
Approach





The autopilot performed the approach appropriately; however, the flight plan did not work very well. The south leg of the approach was too short; thus, the aircraft was still descending and didn't make it to 150 ft. AGL when it flew over the runway. Next flight I will plan on attempting approaches with an extended final approach leg so the aircraft will have a chance at descending before it reaches the runway.

Elevator Doublets



		Elevator Doublets							
	Elevator10	Elevator11	Elevator12	Elevator13	Elevator14	Elevator15	Elevator16	Elevator17	Elevator18
Duration (s)	3	3	3	3	3	3	3	3	3
Period (ms)	750	750	750	750	750	750	750	750	750
Deflection (deg)	2	2	-2	-4	4	2	-2	3	-3
plotdoublet results	N/A	-5.5594	-6.562	-4.6644	-5.2414	-6.4744	-5.6115	-4.8472	-5.2443
DoubletPiccoloLog results	-5.145	-5.994	-5.5632	-5.035	-5.3214	-7.2166	-5.8451	-5.1449	-4.8867
		Elevator Effectiveness						Elevator Power	
	User Manual Calculation Estimate	-5.1772466	/rad	-0.09036000	/deg		Simulator Estimated	-1.291185	/rad
	DoubletPiccoloLog results	-5.4355143	/rad	-0.09539866	/deg		Test Results	-1.363184	/rad
	plotdoublet results	-5.4963667	/rad					-0.023792	/deg
	Average	-5.4659	/rad						
	% difference	1.05576204							

There was much less noise in the CL data than the previous elevator doublet tests due to the new mount. Nevertheless there was still some noise in the Doublet File CL data that did not show up in the piccolo log CL data. The discrepancy is likely due to the higher telemetry rate of the Doublet Files than the piccolo log data. As a result it was decided to analyze the elevator doublets with both the doublet file and the piccolo log file. The CL points in the doublet file analysis were chosen as the mean CL at the time that was determined to be the initial and final times. The DoubletPiccoloLog result was calculated by dropping the highest and lowest two values (-7.2166, -4.8867) and averaging the rest. The DoubletPiccoloLog result was -5.4355143 /rad.

$$Elevator\ Effectiveness$$
$$= \frac{-5.145 + -5.994 + -5.5632 + -5.035 + -5.3214 + -5.8451 + -5.1449}{7}$$
$$= -5.4355143\ /rad$$

The plotpiccolo result was calculated by dropping the highest and lowest two values (-6.562, -4.6644) and averaging the rest. The plotpiccolo result was -5.4963667 /rad.

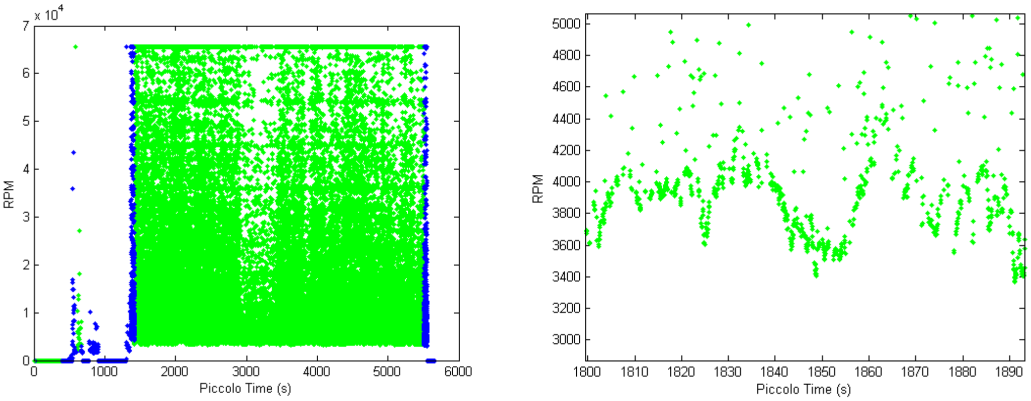
$$Elevator\ Effectiveness$$
$$= \frac{-5.5594 + -5.2414 + -6.4744 + -5.6115 + -4.8472 + -5.2443}{6}$$
$$= -5.4963667\ /rad$$

The final results was determined by averaging the DoubletPiccoloLog and plotpiccolo results. The elevator effectiveness was determined to be -5.4659 /rad.

The elevator power was calculated by multiplying the percent difference of the elevator effectiveness results from the estimated result to the simulator estimated elevator power. The new elevator power was calculated to be 0.000573 /deg.

$$Elevator\ Power = 1.05576204 * -1.291185 = -0.023792\ /deg$$

RPM



The figure above depicts the RPM data. The capacitors did not appear to have filtered any noise.

1.24. Flight 6

1.24.1. Pre-Flight Autopilot Changes

1) Updated the software from 2.1.4g to 2.1.4h. Among other issues the release notes state that they fixed the conflict of the I/O lines 10 and 11 on the SL565 (the unit we are using). Previously we had to click “Send All” on the “Payload I/O Settings” window in order for the tailwheel (plugged into I/O 10) to function. The update fixed this issue. They also changed the way that the SL565 samples accelerometer data. It will be interesting to see how that affects performance given our vibrations that exist in flight.

2) Changed the elevator effectiveness to -5.435514 /rad or -0.09539866 /deg.

3) Added a Cmd scaling term to both ruddervator control surfaces in the simulator file so that the simulator's estimated elevator power would match the results of the elevator doublet tests. The scaling value was the percent difference between the doublet test results and the original simulator estimated results.

$$Cmd\_scaler\_d5 = 1.05576204$$

$$Cmd\_scaler\_d6 = 1.05576204$$

4) It was found out that the PMU that had been flown in Noctua B1 was meant to be used in bench testing, not in actual flight. That being the case it is no wonder that the PMU was creating noise in our RPM data. It was decided to remove the PMU and fly off of battery power alone. An order was placed for a different PMU to test later.

5) The flap rate limit was set to 10 deg/s.

1.24.2. Flight Plan

1) Tune Energy Control

    a) Box to Loiter

2) Assess RPM sensor performance in new configuration without the pmu

3) Test flying with Flaps

    a) 10 degrees

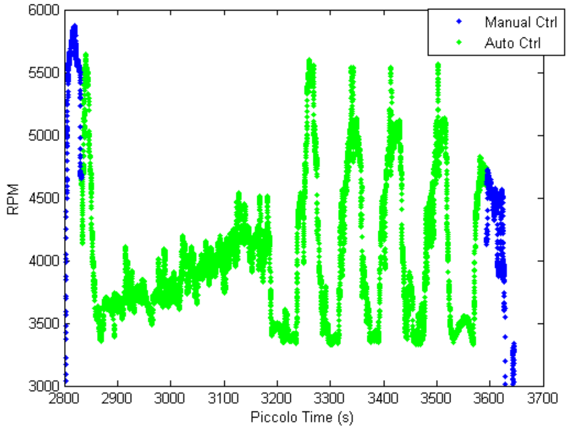
    b) 20 degrees

    c) 35 degrees (full deflection)

4) Fly Approaches at least 100 ft AGL with full flaps if flap performance is adequate.

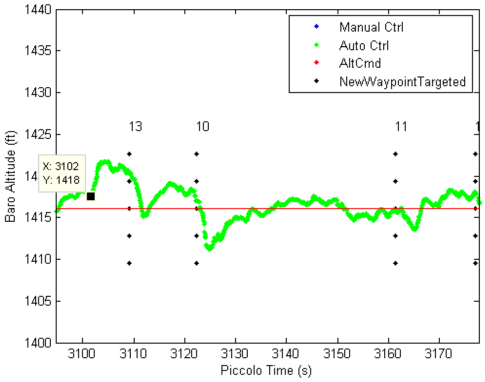
1.24.3. Flight Analysis

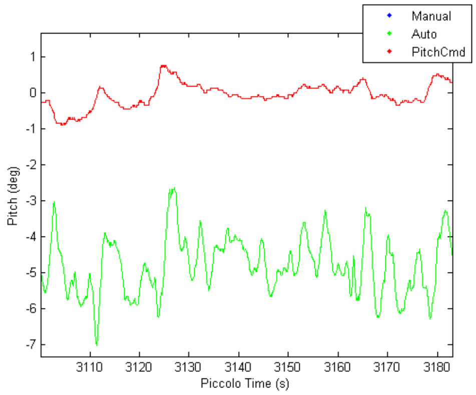
RPM Data



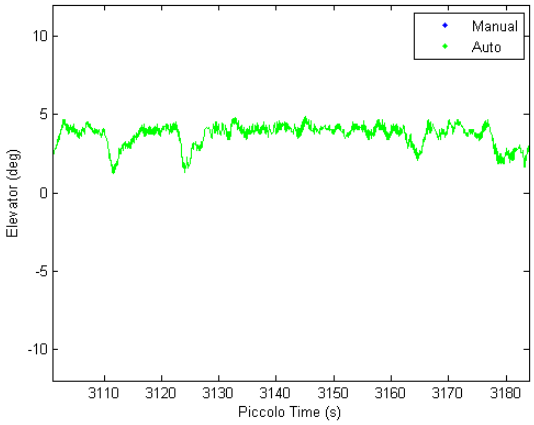
The RPM data contained no noise at all. The data was clean. The PMU was the source of the noise in the RPM signal.

Flap Deflections



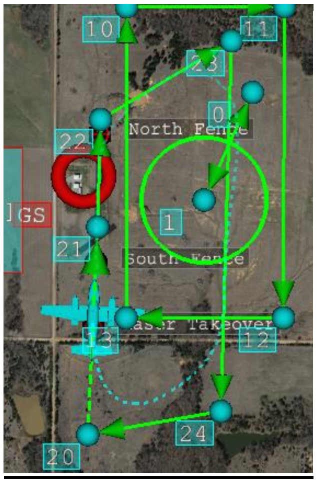


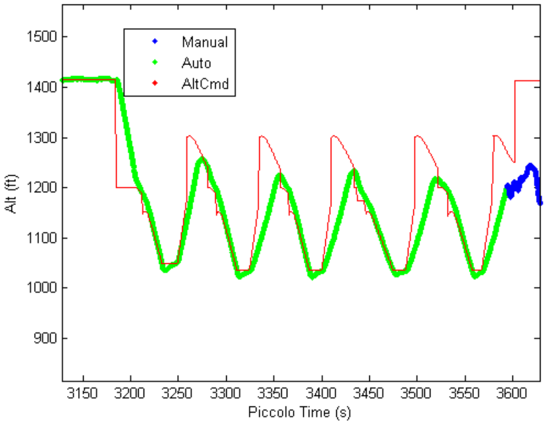
The autopilot was able to maintain altitude throughout the different flap deflections. Each time the flaps were deflected the aircraft initially pitched up and climbed up a few feet. The most dramatic climb was from 20 degrees to 35 degrees, which was also the largest instance of increased flap deflection (15 degrees). The figures above depict the performance immediately after the flap deflection change from 20 to 35 degrees. The flap deflection occurred at 3102 seconds. The aircraft initially pitched up 2.2 degrees and climbed 4 feet in altitude before the autopilot was able to adjust the elevators accordingly. The flight path to waypoint 13 was too short for the autopilot to have time to converge on an altitude; however, the autopilot had no trouble converging onto the altitude command during the flight path to waypoint 10. The overall average pitch of the aircraft was -5 degrees as opposed to -1 degrees which is the average pitch of the aircraft in steady level flight with 0 degrees flap deflection.



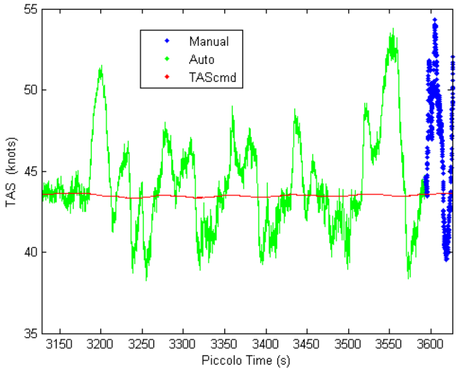
The average elevator deflection to trim with 35 degree flap deflection was +4.1 degrees. The average elevator deflection to trim when there was no flap deflection was about -0.45 degrees. 4.1 is well within the maximum elevator deflection of 12 degrees.

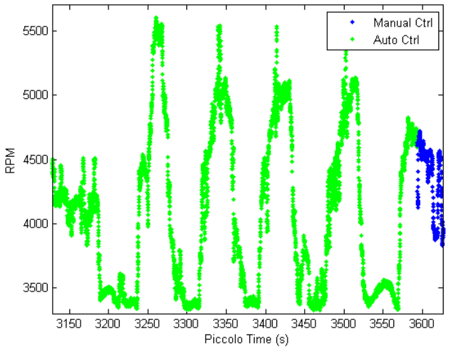
100 ft AGL approachs

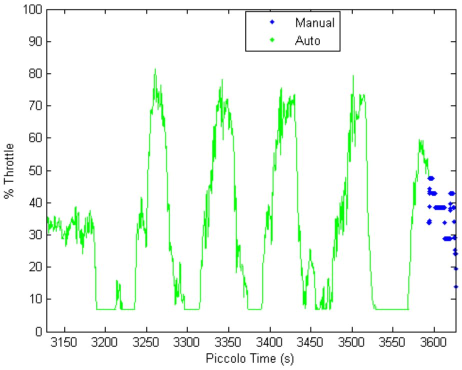




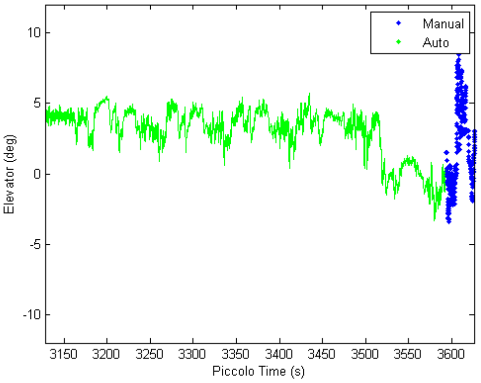
The flight plan for the approaches is shown in the figure above. The flight plan was waypoints 20 – 24. The flight plan didn't actually have a climb out path to a go around waypoint, so there was not enough time for the autopilot to actually climb back up to altitude after each pass. As a result the altitude error was large going into waypoint 23 after the descent. The approach slope was -5 degrees and the climb out slope was 13.10 degrees. The Climb max fraction was originally set at 0.15 for the first approach, and was changed to 0.10 for the remainder of the approaches. The first four approaches had full flaps deployed and the last approach had 0 flap deflection. In all of the climb outs the climbs were restricted by the vertical rate max. The vertical rate command was about 7.2 ft/s for each climb out. After each descent the autopilot undershot altitude by an average of 13 feet. The autopilot was able to climb to just under 1 foot below the commanded altitude during the flights across the runway.

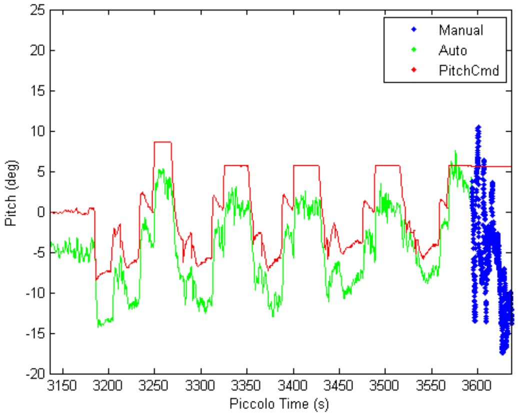






The figures above show the true airspeed and rpm of each approach. The airspeed undershoot after each descent was an average of -4.3 knots. The initial airspeed undershoot after each climb out began was an average of -4 knots. The airspeed did not undershoot after the final approach because the airspeed had a large overshoot during the descent, with no flap deflection. The autopilot commanded up to an average of 80% throttle during the climb outs which lead to 5500 rpm. The throttle command went up to 60%, with an rmp of 4800 for the final climb out with 0 degree flap deflection.





The elevator deflection required to meet the vertical rate commands for the climb outs was an average of +5.5 degrees. The autopilot commanded an elevator deflection of +1.2 degrees for the final climb out with 0 degree flap deflection. The average pitch during the descents of the full flap approaches was -12 degrees. The pitch during the no flap descent was -8.8 degrees.

Conclusions:

1) Deploying flaps with the current settings can cause an initial climb of up to 2 feet in altitude in about 2 seconds before the autopilot trims out.

2) At full flaps in steady level flight the elevator trim changes from -0.45 degrees to +4.1 degrees.

3) At full flaps in steady level flight the pitch changes from -1 degrees to -5 degrees.

4) The full flap climb outs with a vertical rate command of 7.2 ft/s required the following:

   a. 80% Throttle (Max 100%)

   b. 5500 rpm (Max 6000)

   c. 5.5 degrees elevator (Max 12 degrees)

5) Full flaps did not increase or decrease the altitude undershoot that occurred after the approach descents (13ft).

6) Full flaps did not increase or decrease the airspeed undershoot that occurred after the climb outs began (-4 knots).

7) The airspeed undershoot after the approach descents did not occur during the final approach with 0 flap deflection; however, the flaps did not necessarily cause the airspeed undershoot. Due to the absence of flaps the aircraft had a large overshoot in airspeed during the descent; therefore, the airspeed never had a chance to undershoot once the descent was completed because by the time the airspeed made it down to the commanded airspeed the autopilot had already began to climb out.

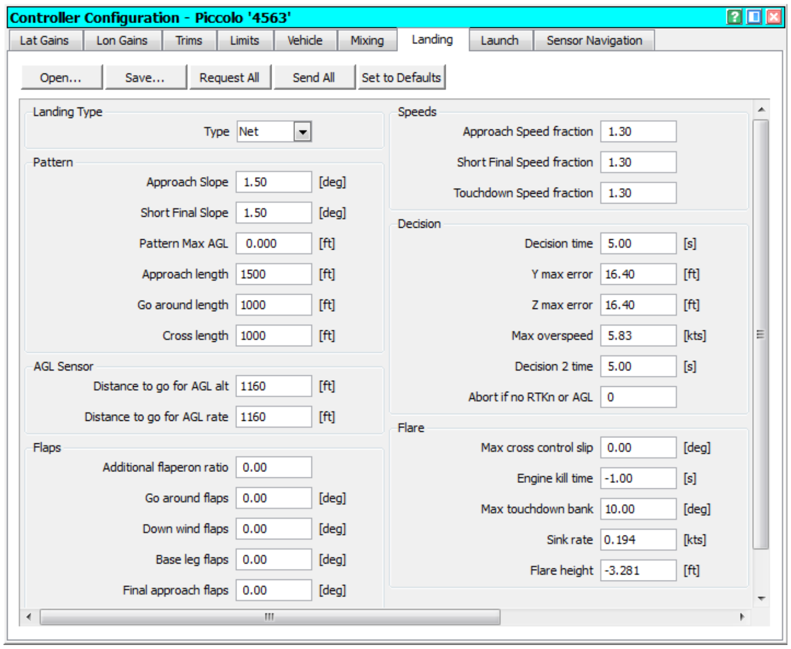
8) The aircraft's pitch to make the -5 descent slope (vertical rate command of -8.8 ft/s) was -12 degrees with full flaps and -8.8 degrees with no flaps.

1.25. Flight 7

1.25.1. Pre-Flight Autopilot Changes

1) Added a laser altimeter. The laser altimeter was wired into the CAN port. As a result no settings had to be changed in the autopilot to get the laser altimeter to function.

2)



Landing Type

Type	Net	

Pattern

Approach Slope	1.50	[deg]
Short Final Slope	1.50	[deg]
Pattern Max AGL	0.000	[ft]
Approach length	1500	[ft]
Go around length	1000	[ft]
Cross length	1000	[ft]

AGL Sensor

Distance to go for AGL alt	1160	[ft]
Distance to go for AGL rate	1160	[ft]

Flaps

Additional flaperon ratio	0.00	
Go around flaps	0.00	[deg]
Down wind flaps	0.00	[deg]
Base leg flaps	0.00	[deg]
Final approach flaps	0.00	[deg]

Speeds

Approach Speed fraction	1.30	
Short Final Speed fraction	1.30	
Touchdown Speed fraction	1.30	

Decision

Decision time	5.00	[s]
Y max error	16.40	[ft]
Z max error	16.40	[ft]
Max overspeed	5.83	[kts]
Decision 2 time	5.00	[s]
Abort if no RTKn or AGL	0	

Flare

Max cross control slip	0.00	[deg]
Engine kill time	-1.00	[s]
Max touchdown bank	10.00	[deg]
Sink rate	0.194	[kts]
Flare height	-3.281	[ft]

Set the "Distance to go for AGL alt" to 1160ft and the "Distance to go for AGL rate" to 1160 so that the autopilot will use the laser altimeter AGL measurements for altitude and vertical rate feedback during the approach of the land plans.

3) Set the "Engine kill time" to -1.0 so that the autopilot will not kill the engine ignition when the aircraft flies through the touchdown waypoints.

4) Set the “Flare height” to -1 so that the autopilot will not perform a flare maneuver during the land plans. PCC automatically changed the flare height from -1 to -3.281, which is -1 meter.

5) Set the landing type to “Net”. Simulations show that the autopilot will continue to fly around the land plan if the aircraft is not captured in net landings.

6) Set all of the flap deflections to 0.

7) Set the approach, short final, and touchdown speed fractions to 1.30, so that the aircraft doesn’t fly too slow during the land plans since we are not actually landing.

1.25.2. Flight Plan

Objectives:

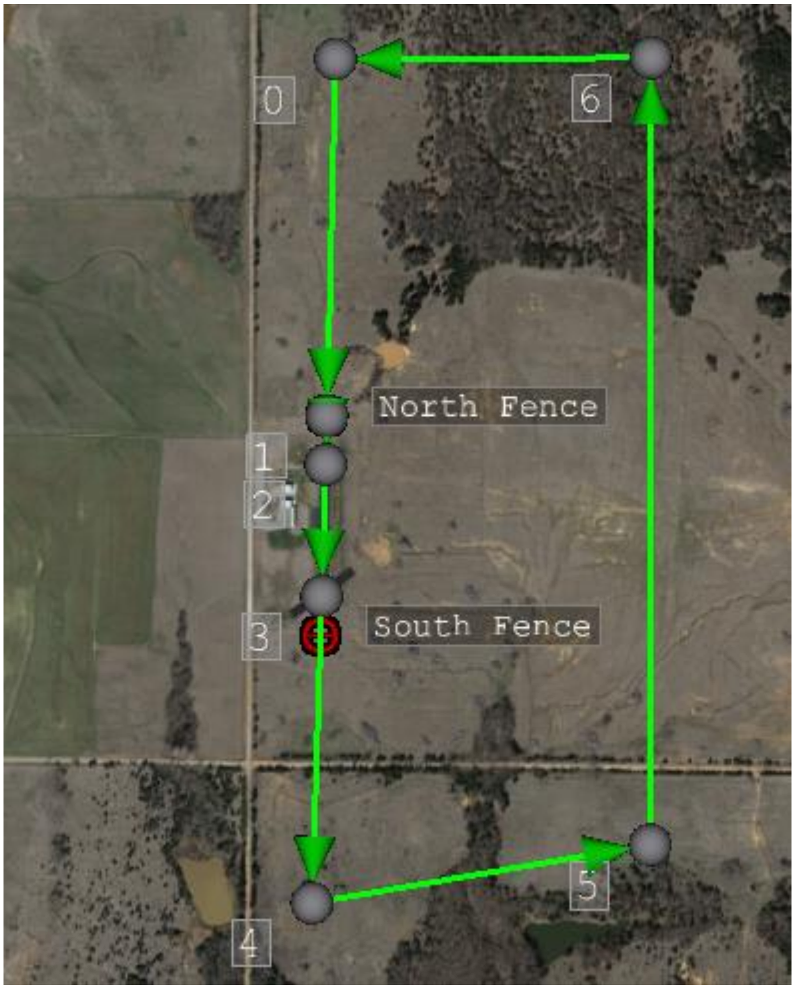
1) Test laser altimeter AGL measurements.

2) Test altitude control with laser altimeter.

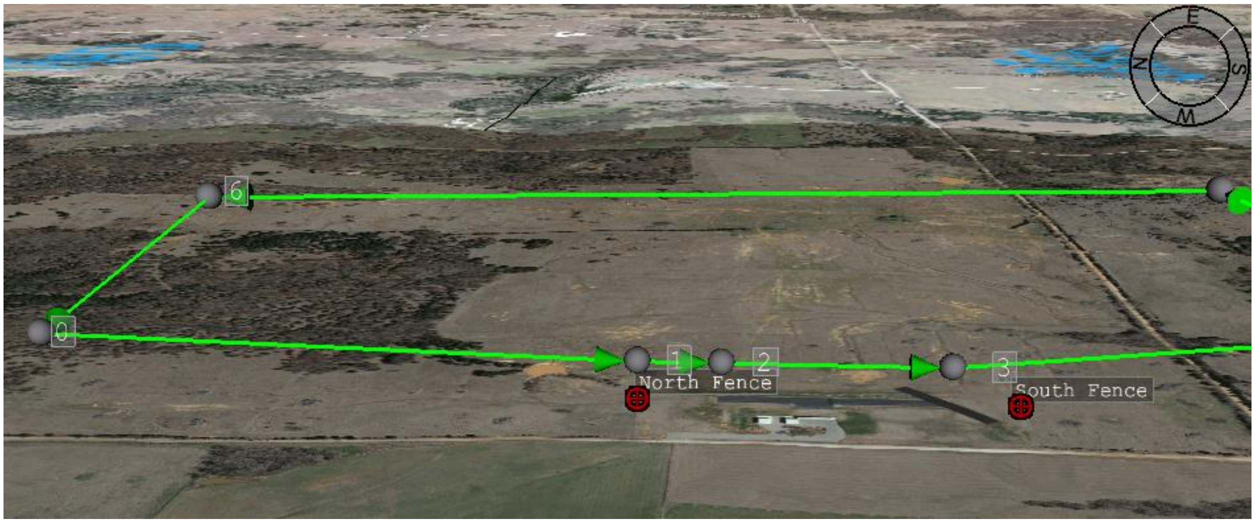
3) Fly approaches as low as possible, and observe the autopilot’s ability to hold constant agl down the runway.

Objective 1:

a) Fly a mock approach, at 100 ft. agl down the runway, that consists of regular waypoints and is not an actual land plan. Additionally all of the waypoints will be defined as “WGS” which means the measurements taken by the laser altimeter will not be used by the autopilot for altitude control.



In the figure waypoints 1, 2, and 3 will be 100ft agl. Waypoint 1 is located at the fence, and waypoint 2 is located at the beginning of the runway.



b) Send Noctua to the lost comms loiter and analyze the mock approach. The laser altimeter measurements at waypoints 1, 2, and 3 will be compared against the agl of the waypoints according to the elevation data.

Objective 2:

a) If the laser altimeter measurements are acceptable fly the same mock approach as before, except this time waypoints 1, 2, and 3 will be designated as “AGL” waypoints. The autopilot will target 100ft agl from waypoints 0 to waypoint 3 and use the laser altimeter measurements in altitude control.

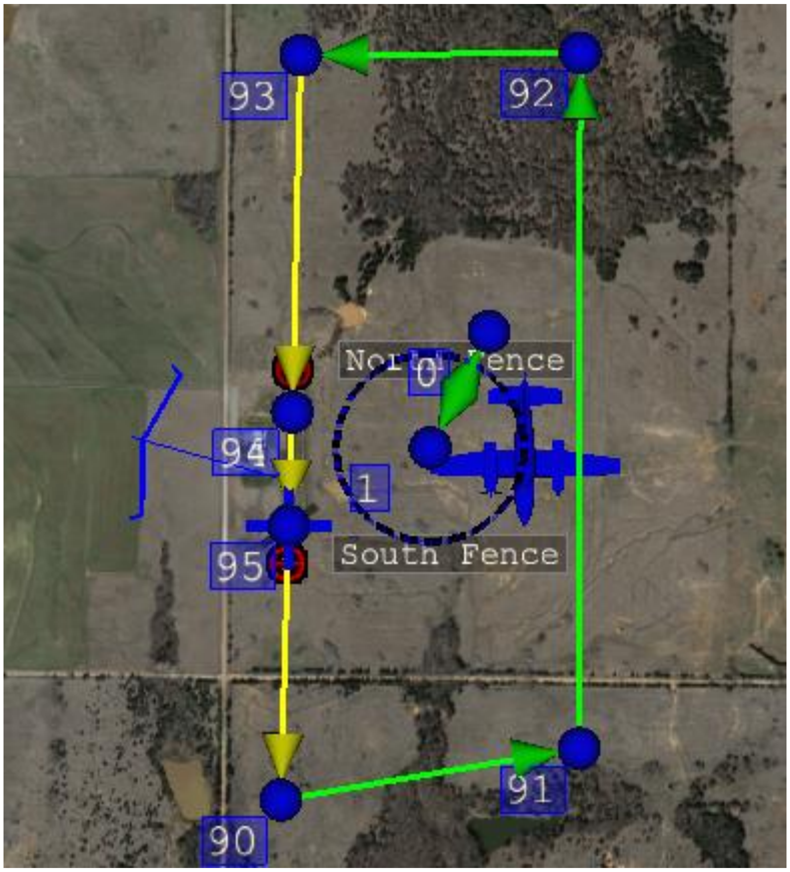
   One issue with this plan is that the autopilot will treat the 100ft agl command as a step input so as soon as Noctua passes through waypoint 0 the autopilot will command as steep of a descent as it is allowed by various limits. The descent max fraction is a limit that has the greatest impact on the maximum descent slope. It will be set to a low value.

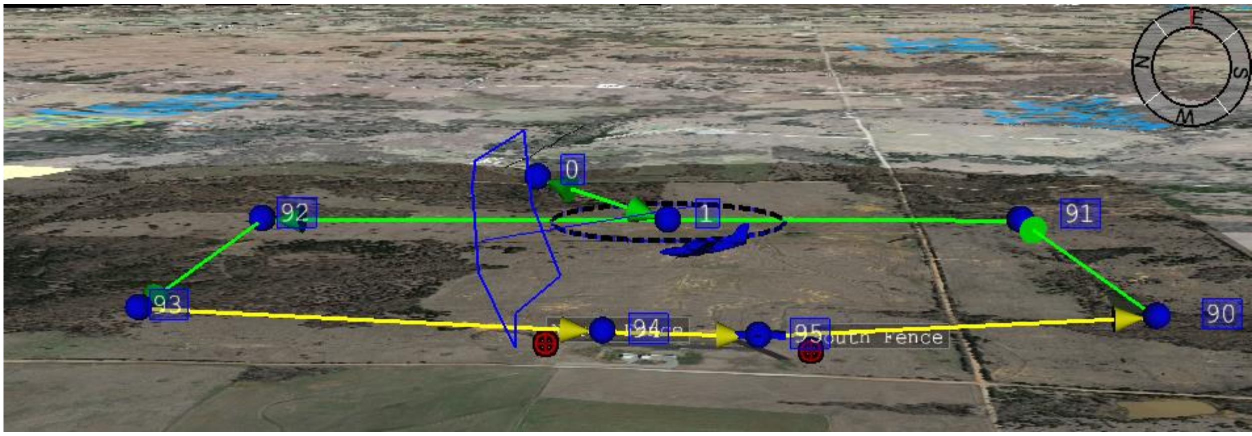
b) Send Noctua to the lost comms loiter and analyze the mock approach. The analysis will focus on the altitude control of Noctua during the AGL stages.

Objective 3:

Before objective 3 is attempted, hardware in the loop simulations will be required in order to test some assumptions that have been made regarding some unknowns of flying actual land plans. The hardware in the loop simulations will be ran at the airfield prior to flying. Simulating land plans with the laser altimeter requires the use of the ‘Piccolo 4563’ unit which is currently in Noctua at the airfield.

The purpose for wanting to fly actual land plans for approaches is to try to avoid the scenario where the autopilot commands a step input for agl waypoints and instead commands a constant descent slope as outlined in the flight path of the land plan.





There are land settings that seem to indicate that the autopilot will use the laser altimeter during approaches to follow the descent slope of the land plan, essentially commanding a ramp input instead of a step input for the altitude command. In the figure the approach would be from waypoints 93 to 95, and the descent slope would be from waypoint 93 to 94. Waypoint 94 is located at the beginning of the runway, and the autopilot should attempt to hold constant altitude down the runway from waypoint 94 to 95. The autopilot's use of the laser altimeter in land plans can be tested in hardware in the loop simulations.

Another unknown is how the autopilot will react when it flies through the "touchdown" waypoint. The touchdown waypoint is waypoint 95 in the figure. It has been discussed that the land plans should be designated as "Net" landings with the assumption that the autopilot will look for x-accelerometer measurements to indicate that the aircraft has been captured and a successful landing has taken place. In "Wheeled" landings the autopilot looks for z – accelerometer measurements to indicate that ground contact and a successful landing has occurred. In software in the loop simulations I have confirmed the following for land plans where the land type is specified as "Net" landing:

1) The autopilot does not look for z-accelerometer measurements to indicate a successful landing has occurred. Ground contact will not transition the autopilot into "rollout" mode (rollout mode is the mode that the autopilot transitions to when it believes it has successfully landed).

2) If the autopilot flies through the touchdown waypoint and a land has not been detected it will kill the engine. By kill the engine I mean it will cutoff power to the engine ignition switch; however, the autopilot will still issue throttle commands and will continue to target the go around waypoint, waypoint 90.

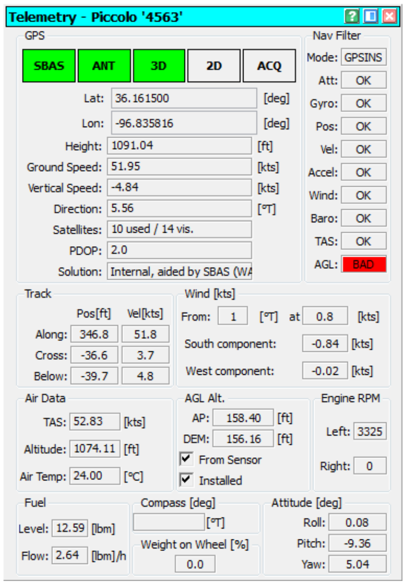
3) The engine kill command that results from passing through the touchdown waypoint can be disabled by setting the "engine kill time" to a negative number in the land settings. In this scenario the autopilot will fly through the touchdown waypoint and continue to the go around waypoint, waypoint 90, and fly the land plan all over again.

These findings need to be confirmed in a full system HiL. A full system HiL is a HiL where all systems on the aircraft are functional, even throttle. It needs to be seen that the engine will indeed continue to run as the aircraft flies through iterations of the land plan.

If the software in the loop findings is confirmed in hardware in the loop simulations and if the autopilot will use the laser altimeter measurements as ramp inputs for constant slope descents on land plans then the test approaches can be flown as actual land plans. Land plan approaches have been made for 50ft, and 20ft AGL. Dan will be able to takeover control at any stage of the approaches. The autopilot should not be able to kill the engine ignition in any scenario as long as the "engine kill time" is a negative number.

1.25.3. Flight Analysis

a**100ft Mock Approach "WGS"**



Lat: 36.161500 [deg]
Lon: -96.835816 [deg]
Height: 1091.04 [ft]
Ground Speed: 51.95 [kts]
Vertical Speed: -4.84 [kts]
Direction: 5.56 [°T]
Satellites: 10 used / 14 vis.
PDOP: 2.0
Solution: Internal, aided by SBAS (WA

Mode: GPSINS
Att: OK
Gyro: OK
Pos: OK
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: BAD

	Pos[ft]	Vel[kts]
Along:	346.8	51.8
Cross:	-36.6	3.7
Below:	-39.7	4.8

From: 1 [°T] at 0.8 [kts]
South component: -0.84 [kts]
West component: -0.02 [kts]

TAS: 52.83 [kts]
Altitude: 1074.11 [ft]
Air Temp: 24.00 [°C]

AP: 158.40 [ft]
DEM: 156.16 [ft]
☑ From Sensor
☑ Installed

Left: 3325
Right: 0

Level: 12.59 [lbm]
Flow: 2.64 [lbm]/h

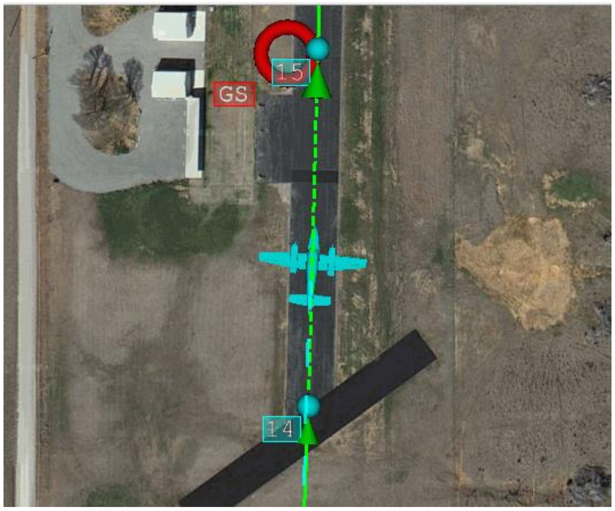
[°T]
0.0

Roll: 0.08
Pitch: -9.36
Yaw: 5.04

The autopilot accepted the laser altimeter measurements for determining aircraft AGL when the laser altimeter was reading AGLs less than 160ft. In the figure the "From Sensor" box was checked when the autopilot was accepting the sensor readings. In the figure AGL is "BAD" because the aircraft was about 60ft above the target altitude. The overshoot was a result of the flight plan, and is not anything to be concerned about.

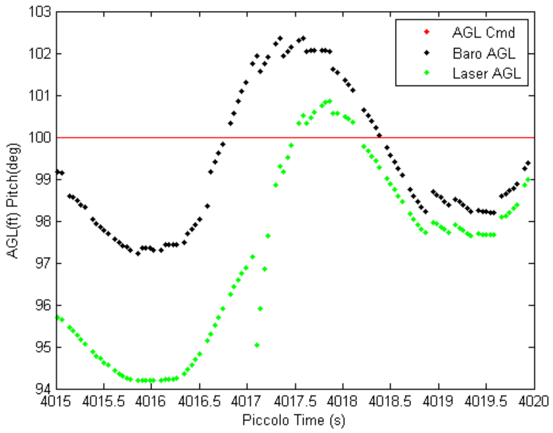
During the mock approach the aircraft was still descending as it flew along the runway so the pitch of the aircraft was -11 degrees. One of the laser altimeter measurements was 158ft. At the same location the barometer AGL was 149ft. The 10ft difference could be due to the pitch of the aircraft, the laser altimeter would have been pointing at an angle rather than straight down.

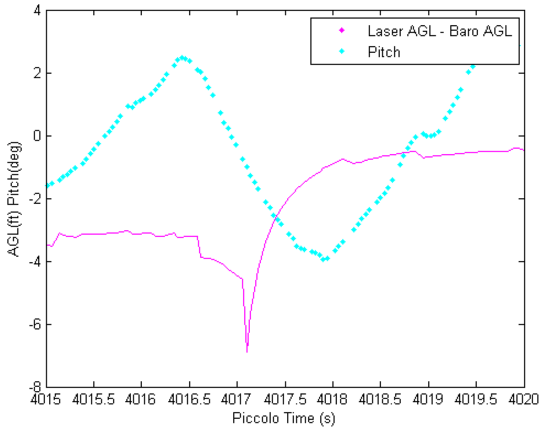
<u>100ft Mock Approach "AGL"</u>



During the mock approaches waypoints 14 and 15 were defined as "AGL" instead of "WGS" so that the autopilot would target AGL altitude commands and use the laser altimeter for aircraft altitude measurements. There were two AGL mock approaches flown. During the first mock approach it was found that the mission limit glitch that showed up in simulations also existed in actual flight. When the autopilot commanded the AGL altitude it interpreted the resulting altitude command as an actual altitude rather than an AGL altitude when it checked the command against the minimum altitude defined in the mission limits. In the case of this flight when the autopilot targeted "100ft AGL" the autopilot incorrectly interpreted that target command as 100ft above sea level when it compared the command to the minimum altitude.

Since the minimum altitude was set to 1043ft the autopilot changed the altitude command to 1043ft, and since the autopilot was using AGL altitude it actually commanded 1043ft AGL. In order to bypass this glitch the minimum altitude in the mission limits window had to be set to less than 100 ft.





The figures depict the AGL measurements during the second mock approach as the aircraft flew down the runway. Just like the simulations the altitude command was a step input and not a slope descent. The barometer AGL indicated that the altitude of the aircraft oscillated about 2ft above and below the commanded altitude. The offset between the laser and barometer measurements began at a magnitude of 3 feet and decreased to about 0.5ft. The barometer AGL is determined from the elevation data of the runway. According to the elevation data the runway is constant elevation, at 935ft; however, the elevation of the south end of the runway is lower than the elevation of the north end of the runway. The larger magnitude of difference between the barometer and laser AGL measurements from 4015 to 4018 seconds could be due to the discrepancy of the elevation data and reality. The largest difference between the laser and barometer AGL measurements was 6ft which occurred just after 4017 seconds where there was a quick 2ft drop in the laser AGL measurement.

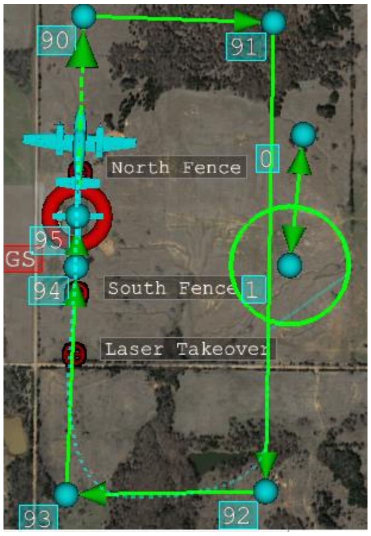
The pitch oscillated between +2.2 degrees and -3.9 degrees during the pass. The error between the barometer and laser AGL measurements does not appear to be directly dependent on the pitch of the aircraft.

Net Land Plans

During pre-flight hardware in the loop simulations and from the patterns flown in flight it was confirmed that the engine will not shut off when the "Engine kill time" is negative and the aircraft flies through the touchdown waypoint of a land plan. Additionally it was also confirmed that the autopilot will continue to command the go around waypoint after the touchdown waypoint if it does not detect that the aircraft has been captured in net land plans. Also, during pre-flight, Cloud Cap confirmed that the autopilot looks for x accelerometer measurements to indicate that a capture has occurred in net landings. It was stated that the x accelerometer measurement value that the autopilot looks for is dependent on various vehicle parameters and it would be impossible for a false capture indication to occur unless the aircraft actually crashed into something.

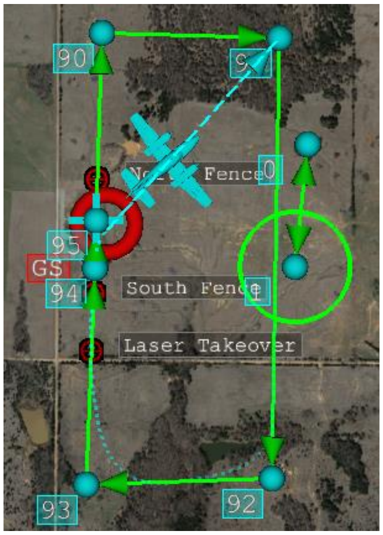
The hardware in the loop simulations with the Piccolo '4653' did not enable the use of AGL sensor in the simulations. As a result the pre-flight simulations were unable to simulate land plans with AGL sensors.

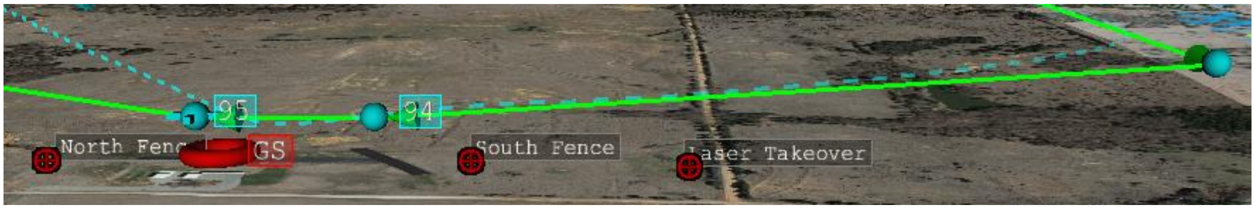
500ft AGL Net Land Plan



The 500ft AGL Net Land Plan confirmed the findings of the simulations that the piccolo will continue to fly through the touchdown waypoint and target the go around waypoint if land is not detected. The figure shows the autopilot targeting the go around waypoint after it flew through the touchdown waypoint.

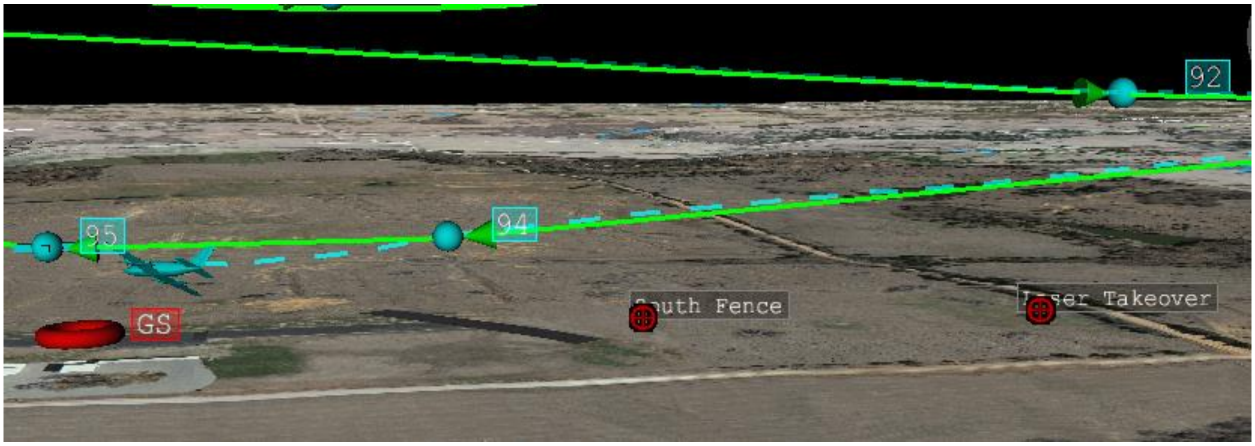
<u>100ft AGL Land Plan Approaches</u>





In total there were 3 land approaches that were attempted. The attempts flew the 100 foot land plan. The 100 foot land plan had all of the waypoints designated as "WGS" altitude and relied on the land settings to use the AGL sensor at a defined maximum distance from the touchdown waypoint. All 3 auto aborted at the short final waypoint, waypoint 94, which was positioned at the south end of the runway. The autopilot aborted because the autopilot perceived that the altitude error was too high. Initially during flight the error seemed odd because the aircraft came down to 124, 104, and 101 feet AGL and the AGL of the short final and touchdown waypoints were set so they would be 100 feet above the ground. Note that the waypoints were not designated as "AGL", their altitudes were designated as "WGS".

The laser takeover point, in the figure above, indicates when the laser took over control. The laser was set to take over control near the approach waypoint, waypoint 93. The autopilot won't use the laser until the laser altitude measurements are within the laser's range (~ 160 ft); therefore, the laser did not take over control until almost half way through the approach descent slope. Due to the AGL Sensor settings in the land settings, the laser provided the feedback for altitude and vertical rate measurements when the autopilot switched to laser control.

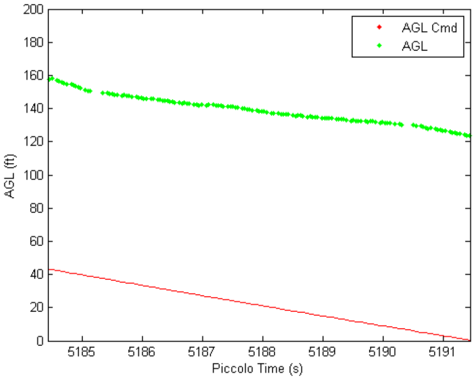


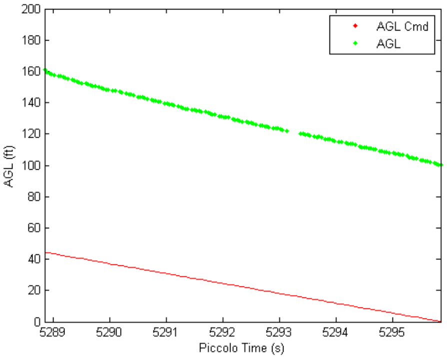
After each auto abort the autopilot targeted waypoint 92 rather than waypoint 90. The abort waypoint for “Net” landings is the crosswind waypoint. The go around waypoint, waypoint 90, would be the target for aborts in “Wheeled” landings. All 3 abort target waypoints were changed manually during flight by the pilot to waypoint 91. During each auto abort by the time the waypoint target was changed to waypoint 91 the autopilot had already targeted waypoint 92 and the aircraft had already banked towards waypoint 92. After each auto abort there were large losses in altitude during the initial banks. It is normal to see an aircraft lose altitude in a bank with the piccolo autopilot because the autopilot’s response to the altitude loss in such a scenario is reactionary; however, the loss of altitude is typically 2 or 3 feet. The land approaches dropped 15 – 28 feet in altitude after the autopilot auto aborted and banked towards waypoint 92.

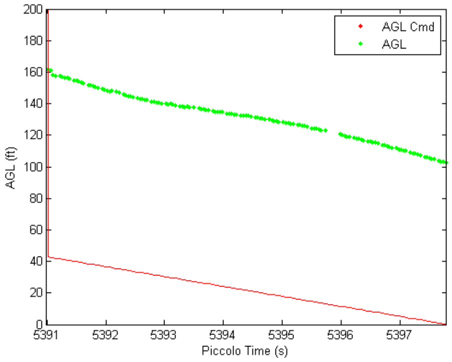
Land Approach	Altitude Loss (ft)	Pitch (deg)	TAS (knots)
1	15	-8.8	52
2	30	-11.75	55
3	28	-11.29	54

The altitude loss that resulted from the auto abort bank commands was extreme because of the large negative pitch of the aircraft. The table above shows the pitch and true airspeed at the instance that the auto aborts occurred, and it shows the altitude loss that resulted after the auto abort bank. There was one mistake that ended up causing the aircraft to pitch down farther than it had to in order to make the descent slope and that was the overshoot in true airspeed. During the land approaches the target airspeed was 45 knots; however, the throttle limit was restricting the autopilot from being able to throttle down to under 3500 rpm. The minimum throttle was set to 15%, during pre-flight, to keep the autopilot from throttling the motor to less than 2000 rpm where the motor would likely die. During flight the motor ran cooler than on the ground, so the rpm response to specific pulse width signals changed in flight as compared to the ground tests. This lead to the overshoots in true airspeed recorded in the table above. Since the airspeed was higher than it was supposed to be the autopilot had to pitch the aircraft down farther than it should have in order to maintain the commanded vertical rate. The data in the table above shows that the higher the airspeed was the lower the pitch was and the larger the altitude loss was.

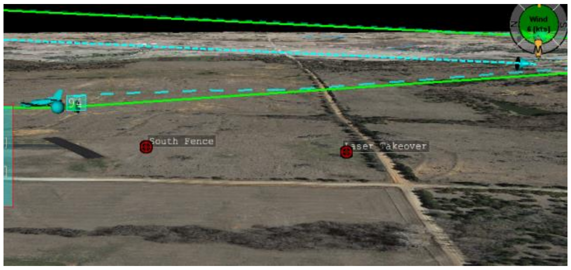
Even though the overshoot in airspeed exacerbated the altitude loss during the auto abort it did not cause the auto abort. It turns out that when the autopilot uses the laser altimeter in land plans to command altitude the laser automatically commands 0 feet AGL regardless of what the altitude of the touchdown waypoint is. As a result the autopilot commanded an AGL of 0, when it switched to the laser altimeter, rather than the planned AGL of 100 feet.

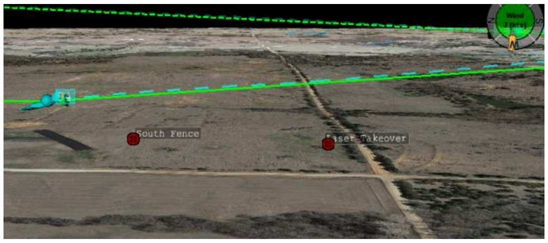


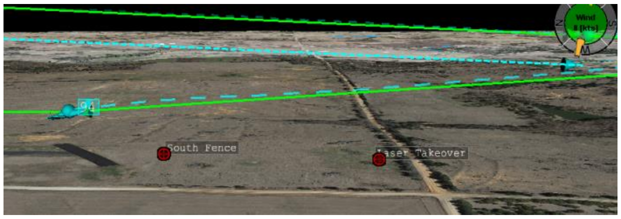




The figures above show the altitude commands during all 3 land approaches when the laser altimeter took control. The altitude command was a slope descent command (ramp input), but it was commanding down to 0 feet AGL. The slope in the land plan was designed for the aircraft to descend to 100 feet AGL, not 0. The figures show that the new altitude command slope created a large altitude error. Recall that the vertical rate command is calculated from altitude errors. As a result the vertical rate command should have increased when the laser took over and the aircraft should have descended steeper than the planned descent slope; however, the actual descent slope of the aircraft did not increase in all 3 land approaches.

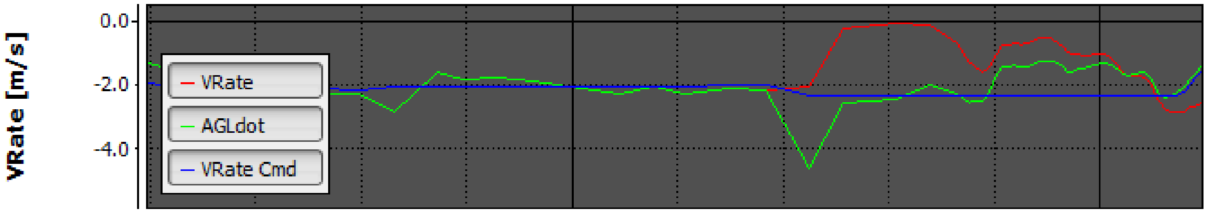


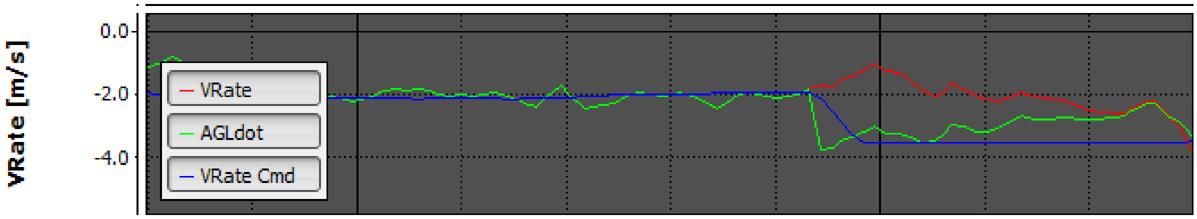


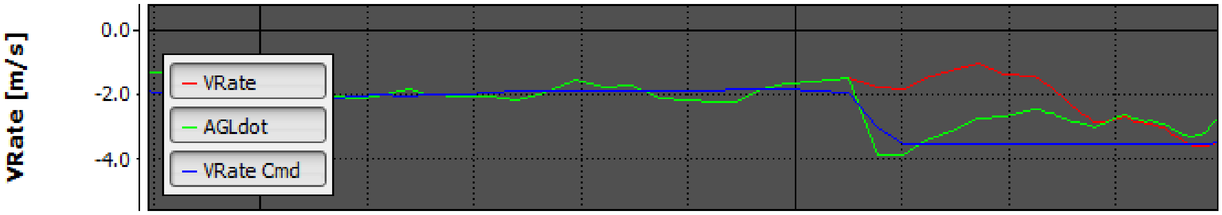


The figures above depict the descent slope of the aircraft during all 3 land approaches. The dotted lines are the "cookie crumb" trail of the aircraft and depict the path that was flown. The figures shows that the descent slope of the aircraft did not increase and in the case of the first land approach the descent slope actually decreased. There was a mistake in the land settings that contributed to offsetting what should have been a steeper descent slope.

The mistake was allowing the autopilot to use the laser altimeter readings to determine the aircraft's vertical rate, rather than the standard GPS/INS solution, when the aircraft was flying over uneven ground. Unless the terrain is completely level it would not be wise to use the laser altimeter to determine the vertical rate of the aircraft because in such a scenario a change in ground elevation would falsely indicate a change in vertical rate of the aircraft. In the case of the 3 land approaches it just happened to be that the elevation of the ground, from the location where the laser took over to the location of the south fence, increased. This means that when the autopilot switched to laser control the vertical descent rate command would have increased in order to descend the aircraft to 0 feet AGL instead of 100 feet AGL; however, the vertical rate feedback, as reported by the change in laser altimeter readings, would have falsely signaled an increase in the vertical descent rate of the aircraft due to the ground elevation increasing. The three figures below show that this is exactly what happened.







The figures are snapshots from the DevInterface plots. Typically "AGLdot" represents the measured vertical rate. During normal flight the "VRate" series does not even exist on the VRate plot. In the case of using the AGL sensor for vertical descent rate measurements "AGLdot" became the laser altimeter vertical descent rate measurements. "VRate" represents the GPS/INS calculated vertical rate. The figures show that when the laser took over control the autopilot commanded the steeper descent to make the 0 foot AGL target, but because of the erroneous vertical rate measurements from the laser altimeter the autopilot falsely perceived that the aircraft's vertical rate was already at the commanded vertical rate. In reality the vertical rate of the aircraft was actually less than what the autopilot thought, as depicted by "VRate" in the plots. Recall that elevator deflection commands are calculated from errors in vertical rate. Since the autopilot did not realize it actually had a positive vertical rate error it did not pitch the aircraft even lower than it already was; thus, the descent slope of the aircraft did not increase. Additionally the aircraft pitched up in the first land approach because the autopilot incorrectly believed there was a negative vertical rate error which caused it to deflect elevators up; thus, the aircraft pitched up. Note that the vertical rate command was not limited by the minimum vertical rate command in all 3 land approaches.

Conclusions

1) The laser altimeter accurately measures aircraft AGL.

2) In order to fly normal waypoints with AGL altitude commands the minimum altitude has to be set to be as low as the AGL altitude command even though they aren't actually the same number.

3) Flying normal waypoints with AGL altitude commands will command altitude as a step input and not a sloped descent.

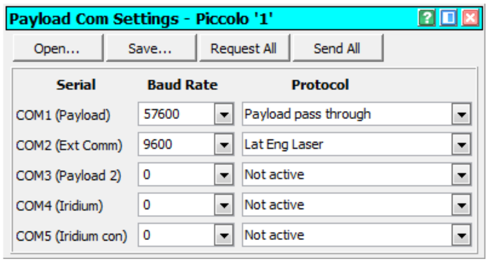
4) The autopilot will repetitively fly net land plans until a capture is detected.

5) The autopilot needs to be capable of throttling down to at least 3000 rpm if not 2500 for the land plans.

6) The "Distance to go for AGL rate" should be set to 0 or at least not used until the aircraft is flying over level ground.

7) The logic of land plans causes the target AGL to automatically target 0 feet AGL when the autopilot is using a laser altimeter.

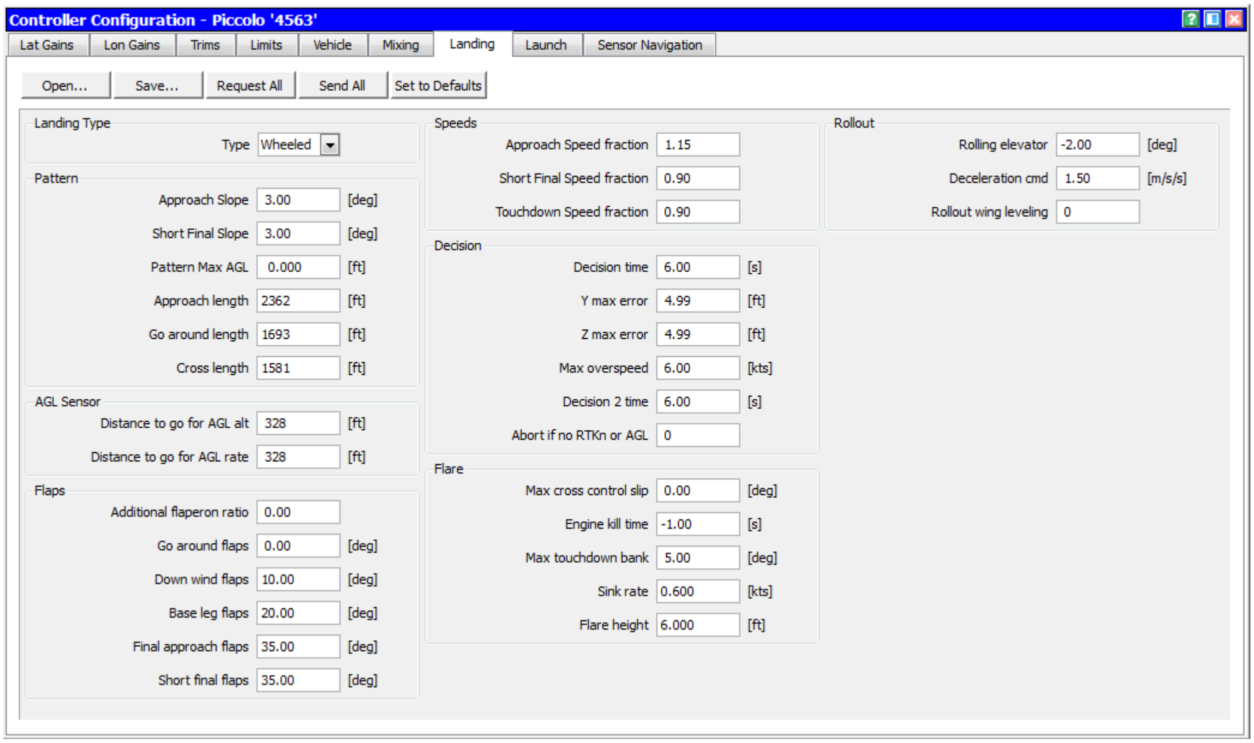
8) The simulator can simulate using a laser altimeter with the piccolo if one of the serial com ports is designated as “Lat Eng Laser”.



Serial	Baud Rate	Protocol
COM1 (Payload)	57600	Payload pass through
COM2 (Ext Comm)	9600	Lat Eng Laser
COM3 (Payload 2)	0	Not active
COM4 (Iridium)	0	Not active
COM5 (Iridium con)	0	Not active

1.26. Flight 8

1.26.1. Pre-Flight Autopilot Changes



Landing Type

Type	Wheeled	

Pattern

Approach Slope	3.00	[deg]
Short Final Slope	3.00	[deg]
Pattern Max AGL	0.000	[ft]
Approach length	2362	[ft]
Go around length	1693	[ft]
Cross length	1581	[ft]

AGL Sensor

Distance to go for AGL alt	328	[ft]
Distance to go for AGL rate	328	[ft]

Flaps

Additional flaperon ratio	0.00	
Go around flaps	0.00	[deg]
Down wind flaps	10.00	[deg]
Base leg flaps	20.00	[deg]
Final approach flaps	35.00	[deg]
Short final flaps	35.00	[deg]

Speeds

Approach Speed fraction	1.15
Short Final Speed fraction	0.90
Touchdown Speed fraction	0.90

Decision

Decision time	6.00	[s]
Y max error	4.99	[ft]
Z max error	4.99	[ft]
Max overspeed	6.00	[kts]
Decision 2 time	6.00	[s]
Abort if no RTKn or AGL	0	

Flare

Max cross control slip	0.00	[deg]
Engine kill time	-1.00	[s]
Max touchdown bank	5.00	[deg]
Sink rate	0.600	[kts]
Flare height	6.000	[ft]

Rollout

Rolling elevator	-2.00	[deg]
Deceleration cmd	1.50	[m/s/s]
Rollout wing leveling	0	

The wheeled land settings were set according to the results of Flight 7 and numerous simulations.

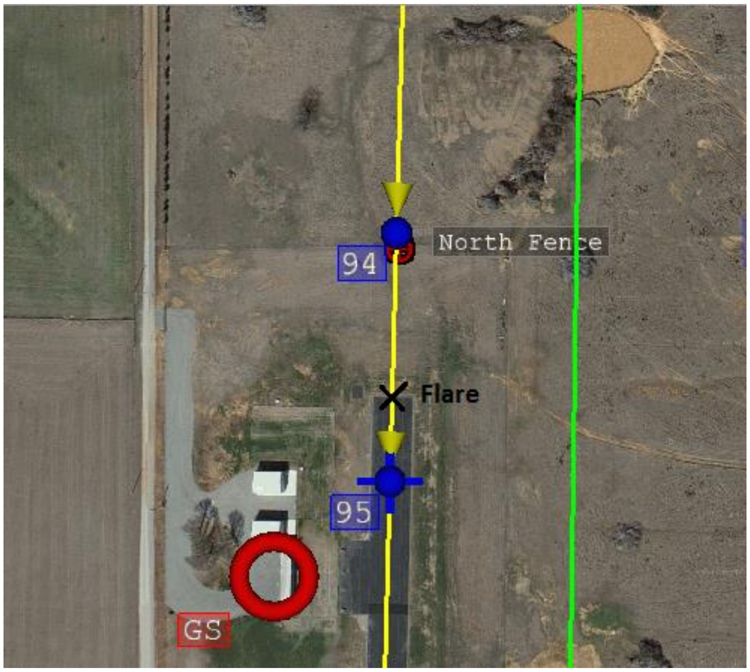
1) The approach length, go around length, and cross lengths were set from the length requirements of the land approaches in Flight 7.

2) The approach speed fraction was set to 1.15 so that the approach IAS command would be around 38 knots.

3) The Short Final and Touchdown speed fractions were set to 0.90 so that the IAS command would be around 28 knots throughout the short final descent and flare.

4) The rolling elevator was set to -2 degrees. The value was estimated so that the tail would slowly come down after touchdown rather than slam to the ground. The deceleration command was left at its default value because it is not applicable to Noctua B1 (no brakes).

5) Set the flap settings to incrementally increase the flap deflection at each leg of the land plan. It was desired to have full flaps throughout the final approach, in addition to the short final, so that the aircraft could shed airspeed and so that the autopilot wouldn’t have to adjust the pitch of the aircraft after the decision waypoint.



6) The decision time was adjusted until the decision waypoint, waypoint 94, was placed just north of the fence line with the touchdown waypoint at about a quarter length of the runway. The time ended up being 6 seconds.

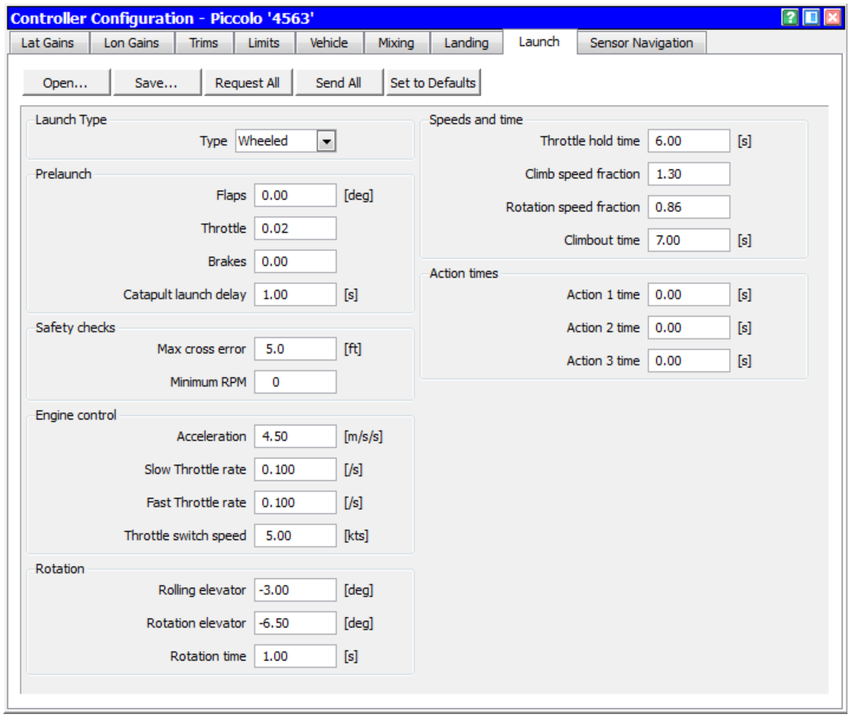
7) The agl altitude and altitude rate settings were set so that the laser altimeter would take control just after the decision waypoint (at 328 feet from the touchdown waypoint). The ground is nearly level from the north fence to the north end of the runway. According to elevation data the ground elevation increases about 3 feet.

8) The flare height was set to 6 feet so that the flare would begin at the beginning of the runway, and close to the touchdown waypoint.

9) The flare sink rate was set to 0.6 knots so that the sink rate would be nearly 1 ft/s.

10) The engine kill time was set to -1.0 so that the autopilot would not kill the ignition.

11) The piccolo comms cable inside Noctua B1 was changed to a new cable, so a ground comms check will have to be performed pre-flight.



Launch Type
Type Wheeled

Prelaunch
Flaps 0.00 [deg]
Throttle 0.02
Brakes 0.00
Catapult launch delay 1.00 [s]

Safety checks
Max cross error 5.0 [ft]
Minimum RPM 0

Engine control
Acceleration 4.50 [m/s/s]
Slow Throttle rate 0.100 [/s]
Fast Throttle rate 0.100 [/s]
Throttle switch speed 5.00 [kts]

Rotation
Rolling elevator -3.00 [deg]
Rotation elevator -6.50 [deg]
Rotation time 1.00 [s]

Speeds and time
Throttle hold time 6.00 [s]
Climb speed fraction 1.30
Rotation speed fraction 0.86
Climbout time 7.00 [s]

Action times
Action 1 time 0.00 [s]
Action 2 time 0.00 [s]
Action 3 time 0.00 [s]

12) Wheeled launch settings were set from analyzing manual pilot takeoffs and trial and error in software in the loop simulations.

13) The acceleration was set to 4.5 m/s2, which was the average acceleration used in all of the previous manual takeoffs.

14) Initially the slow and fast throttle rates were determined from the manual takeoffs and set at 0.01 /s and 0.10 /s. It was found in the software in the loop simulations that the slow throttle rate was too slow and that the autopilot steered better when the throttle rate was quicker; therefore, the slow throttle rate was set to be the same as the fast throttle rate. It appeared that the slow throttle rate allowed the aircraft to slowly deviate from the proper heading and when the throttle began throttling up more quickly the error quickly became large which adversely affected the steering control loop response.

15) The throttle switch speed was initially set at 5 knots; however, since the throttle rates are the same the throttle switch speed is irrelevant.

16) The rotation settings were determined from the averages of the manual takeoffs. In the software in the loop simulations the elevator deflections were too small; however, the simulator's elevator deflections are offset from what is seen in real life by about 4 degrees. It was noticed in the simulations that increasing the rotation elevator can improve the autopilot's steering performance. This effect has also been noticed by the manual pilot in real flights.

17) The throttle hold time was set to 6 seconds so that the autopilot would continue to command full throttle into a decent portion of the climbout period. It was discovered in simulations that the throttle hold time begins as soon as the autopilot enters climbout mode (after liftoff detection via positive vertical rate).

18) The climb speed fraction was set so that the airspeed command would be around 45 knots during the climbout period.

19) The rotation speed fraction was set to 0.86 so that rotation would begin around 26.5 knots. The manual pilot began rotation at an average of 21 knots; however, the target airspeed was increased for safety concerns because 21 knots is well below the minimum airspeed (34.9 knots).

20) The climbout time was set to the minimum value of 7 seconds. The altitudes that we currently fly at are not high enough to need even 7 seconds of climbout time.

1.26.2. Flight Plan

1) Manual pilot stall test with

   a) No Flaps

b) Full Flaps

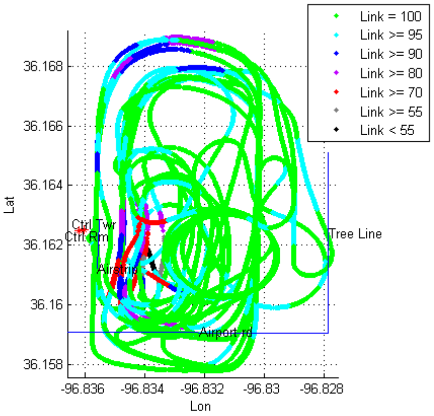
2) Max Climb Rate tests

- Command the autopilot to target a box pattern flight plan
- On the leg closest to the airstrip manually command a positive climb rate through the command loops window
- Observe whether or not it requires the autopilot full throttle to maintain the commanded climb rate
- Re-enable auto altitude control by setting altitude to “Auto” in the command loops window
- Once the aircraft has descended back to altitude repeat the process with a higher climb rate until the maximum climb rate has been determined

3) Attempt wheeled auto lands

1.26.3. Flight Analysis

<u>Flight Comms</u>



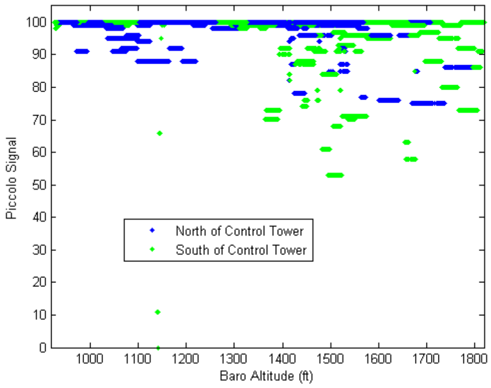


Figure 374

Figure 375

In flight it had been noticed that there was significant packet loss during the climb rate tests. The packet loss was significant enough that it impacted one of the climb tests to where the command to stop climbing could not be sent to the autopilot. Figure 1 depicts the piccolo signal strength versus the location of the aircraft. The climbs were initiated flying south on the west leg of a box pattern. The largest packet losses did occur during the climb tests. Some of the packet loss occurred during the base leg of the land plan and were most likely due to the banking of the aircraft. Figure 2 depicts the piccolo signal strength versus altitude. Most of the packet loss occurred above 1400 ft. In order to properly analyze whether or not location and altitude could be contributing to the packet loss the “LinkPos” plot in “AnalyzePiccolo” was edited so that the user can specify an altitude range for the link versus location plots.

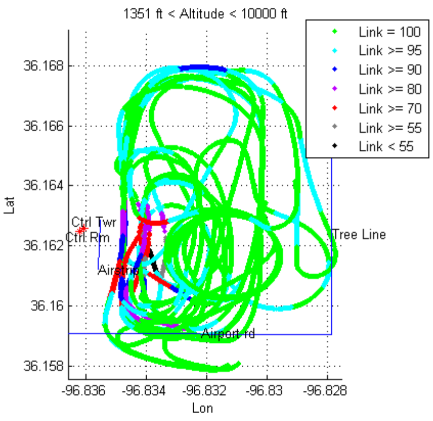


Figure 376

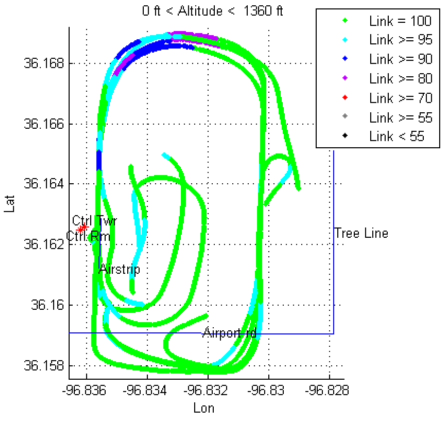


Figure 377

Figure 3 depicts the piccolo signal strength at 1351 feet and above. Figure 4 depicts the piccolo signal strength at 1350 feet and below. Figure 3 shows that altitude alone is not the sole contributor to the packet loss. There was plenty of time that the aircraft was flying above 1350 feet and not experiencing packet loss. The two figures show that when the aircraft was flying in between a longitude of -96.836 and -96.833 there were significant packet losses above 1350 feet.

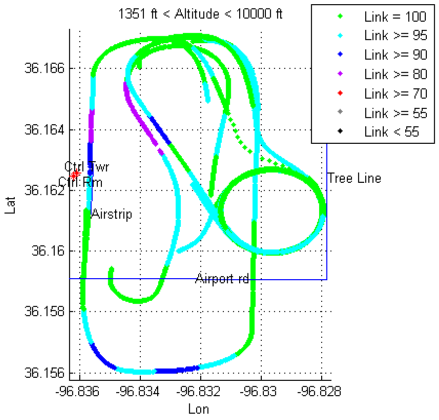


Figure 378 Flight 7

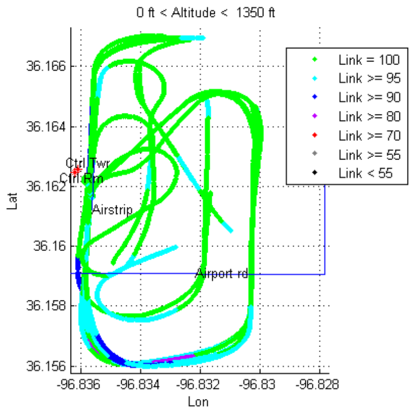
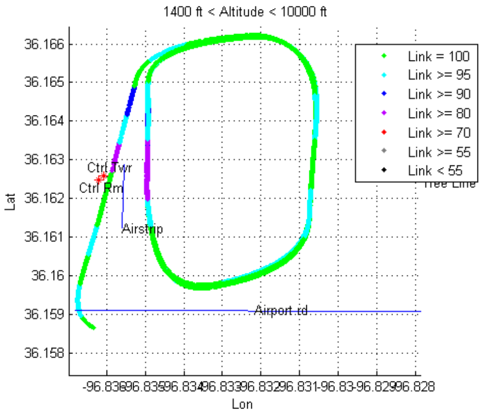


Figure 379 Flight 7

Figures 5 and 6 depict the piccolo packet loss of Flight 7 at varying altitudes. As usual packet loss occurred during banks towards the airstrip. Figure 5 shows that there were moderate packet losses above 1350 feet just east of the airstrip between longitudes -96.836 and -96.833.

Figure 6 shows that the packet loss in the same area and below 1350 feet altitude was nearly nonexistent.



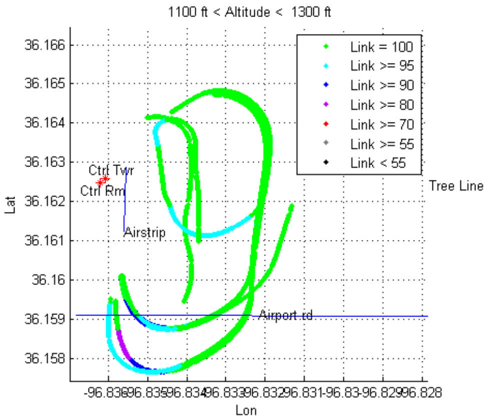
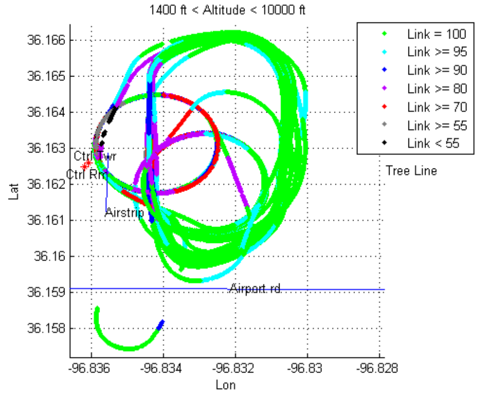


Figure 380 Flight 6

Figure 381 Flight 6

Figures 7 and 8 depict the piccolo packet loss of Flight 6 at varying altitudes. As usual packet loss occurred during banks towards the airstrip. Figure 7 shows that there were moderate packet losses above 1400 feet just east of the airstrip between longitudes -96.836 and -96.834. Figure 8 shows that the packet loss in the same area and below 1400 feet altitude was nearly nonexistent.



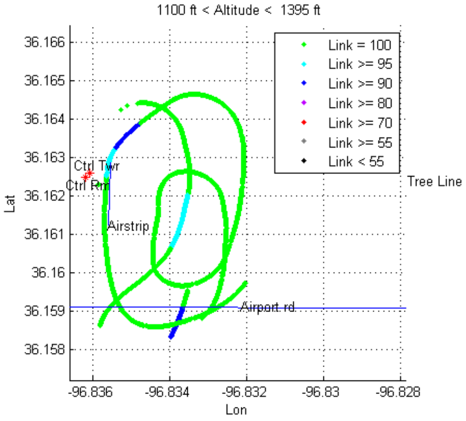


Figure 382 Flight 5

Figure 383 Flight 5

Figures 9 and 10 depict the piccolo packet loss of Flight 5 at varying altitudes. Figure 9 shows that there were significant packet losses above 1400 feet east and west of the airstrip between longitudes -96.836 and -96.832. Figure 10 shows that the packet loss in the same area and below 1400 feet altitude was nearly nonexistent. Flight 5 included altitudes up to 1925 feet.

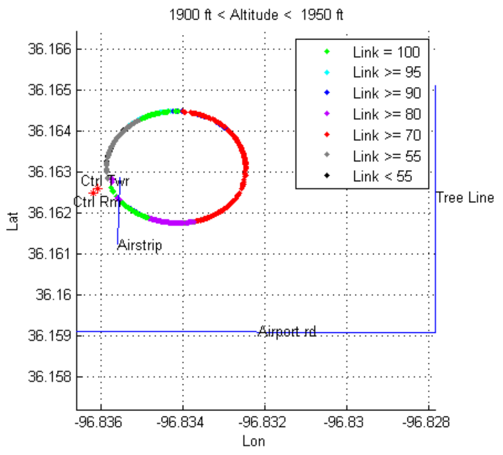
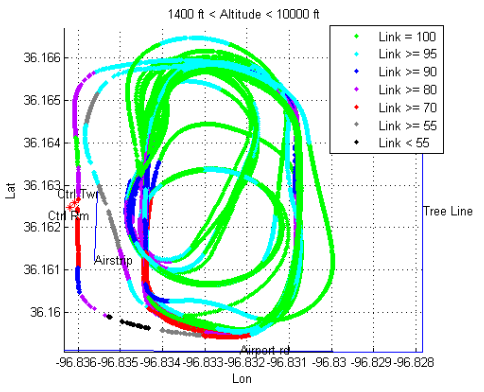


Figure 384 Flight 5

Figure 11 depicts piccolo packet loss from Flight 5 during the loiter at 1925 feet. There were two small areas in the orbit where there were no packet loss; however, most of the orbit contained significant packet loss.



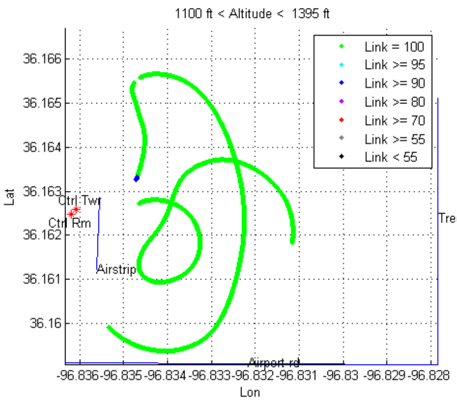


Figure 385 Flight 4 *Figure 386 Flight 4*

Figures 12 and 13 depict the piccolo packet loss of Flight 4 at varying altitudes. Figure 9 shows that there were significant packet losses above 1400 feet east of the airstrip between longitudes -96.836 and -96.832. Figure 10 shows that the packet loss in the same area and below 1400 feet altitude was nearly nonexistent. Figure 12 also shows that there were packet losses on the east leg of the box pattern around -96.831 longitude and also on the south leg.

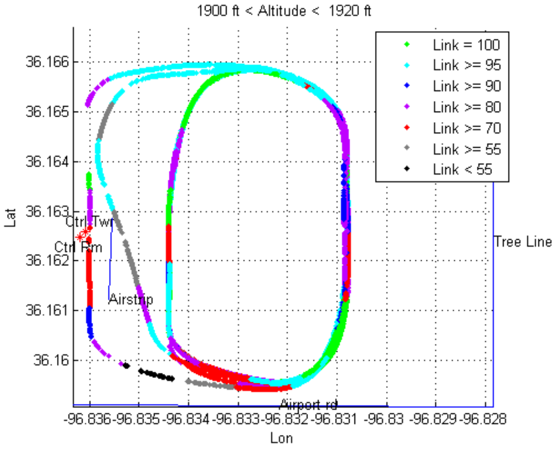


Figure 387 Flight 4

Figure 14 shows the period of Flight 4 that was flown above 1900 feet. The east leg and south leg packet loss occurred at 1920 feet altitude.

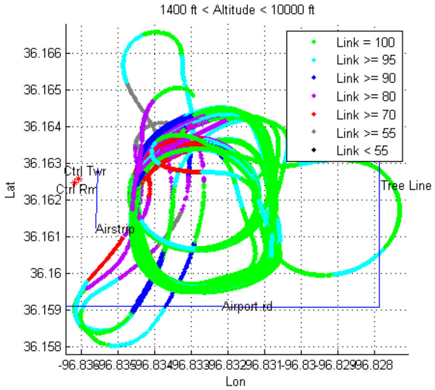


Figure 388 Flight 3

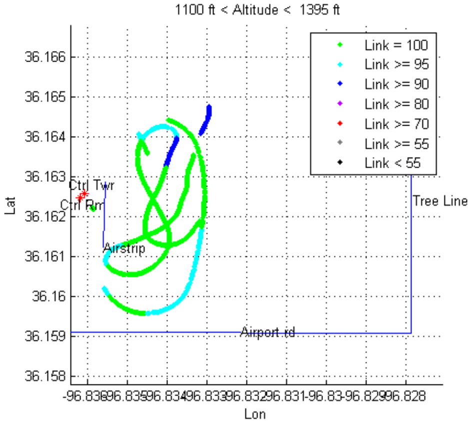


Figure 389 Flight

3

Figures 15 and 16 depict the piccolo packet loss of Flight 3 at varying altitudes. Figure 15 shows that there were significant packet losses above 1400 feet east of the airstrip between longitudes -96.836 and -96.832. Figure 16 shows that the packet loss in the same area and below 1400 feet altitude was nearly nonexistent.

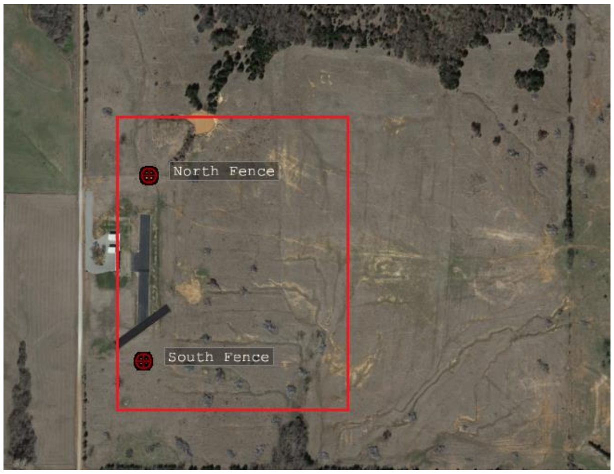
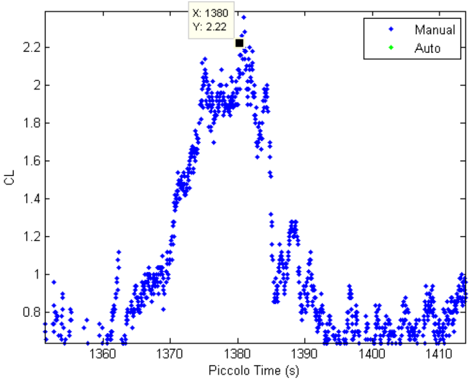


Figure 390

In conclusion there appears to be a dead zone in the piccolo communications as depicted in Figure 17 inside the red box at 1400 feet and above. In the two flights that were flown above

1900 feet there was significant packet loss except for a few small periods of time; therefore, it is concluded that piccolo communications are suspect at 1900 feet and above in all locations.

CLmax



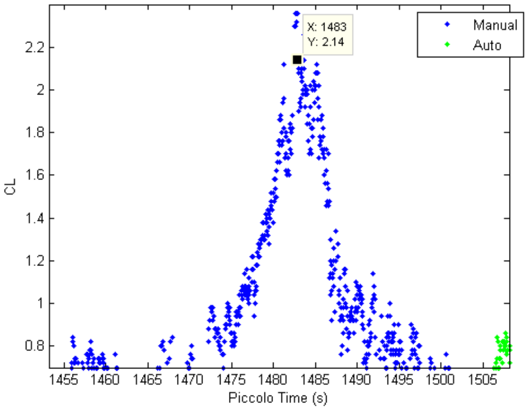
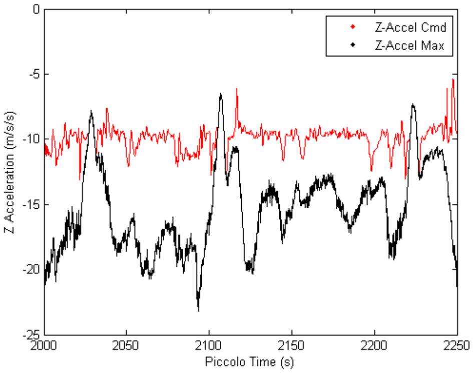


Figure 391 Stall

Figure 392 Stall with Full Flaps

Figures 18 and 19 depict the recorded CL during the stall tests. The CL essentially peaked at 2.2 and 2.14. During design it was determined that the flaps should not increase the maximum CL because the airfoil is a high camber airfoil. Instead of increasing CL max the flaps should shift the CL alpha curve to the left so that the aircraft will reach a higher CL at a lower angle of attack. The results confirm the design hypothesis. Additionally the pitch at the peak of the first stall test was 21 degrees, and the pitch at the peak of the second stall test was 12 degrees, meaning that the angle of attack was lower for the full flap stall test. The airspeed in both tests fell down to 23 knots.

Maximum Climb Rate



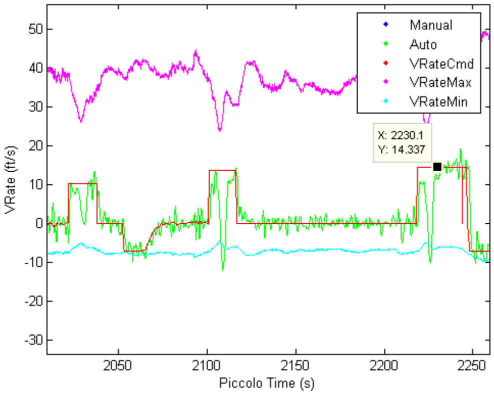


Figure 393 Z-Acceleration

Figure 394 Vertical Rate

There were 3 total climb rate tests. The commanded vertical rates were 6, 8, and 8.5 knots which convert to about 10, 13.5, and 14.3 ft/s. It is interesting to notice how the z-acceleration limiter played a factor in how hard the autopilot attempted to initially climb to make the vertical rate command. Each time that the vertical rate command as sent it was interpreted as a step input, and since the climb max fraction was set to the maximum value (0.50) the autopilot essentially commanded as immediate of a climb possible. Figure 21 shows that in all three climb tests after the autopilot initially climbed to the commanded vertical rate the vertical rate of the aircraft decreased rather dramatically. Figure 20 shows that the decrease in vertical rates corresponded with the z-acceleration command limiter driving the z-acceleration command so as to command elevator down.

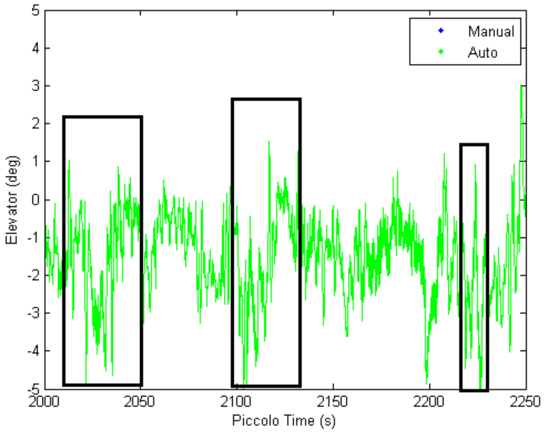


Figure 395 Elevator

Figure 22 highlights the elevator deflections during the climb tests. Each time the z-acceleration limiter drove the z-acceleration command the elevator deflections changed from negative deflection to positive deflection (nose up to nose down). After the initial hard climb attempt the autopilot was able to maintain all three of the vertical rate commands. Ironically the aircraft achieved a higher climb rate after the tests when the climb max fraction was mistakenly left at 0.50. After the climb tests the autopilot was commanded to loiter which was at a higher altitude. Since the climb max fraction was still 0.50 the autopilot commanded a vertical rate of 17 knots (28.6 ft/s).

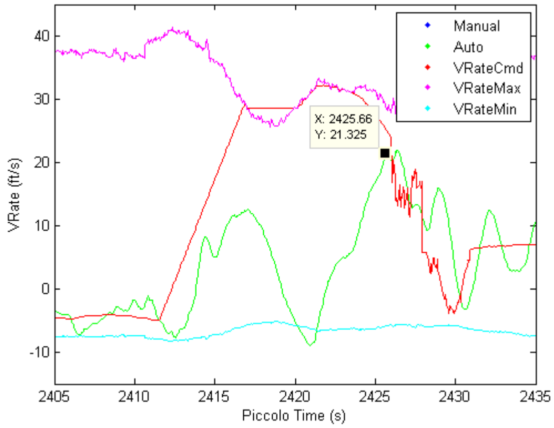
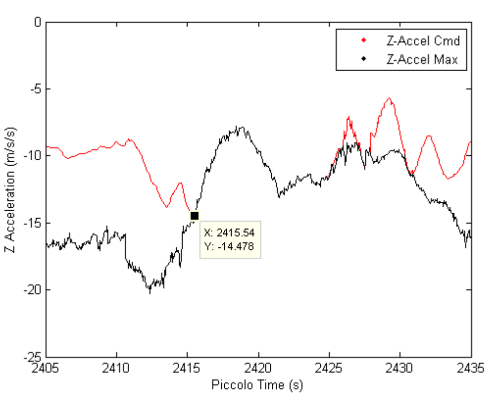


Figure 396 Vertical Rate

Figure 23 shows that Noctua achieved a climb rate of 21.3 ft/s (12.6 knots). At this point in the flight the autopilot transitioned into slow airspeed mode because the throttle was maxed out and the airspeed was below the minimum airspeed. At the time that Noctua peaked at a vertical rate of 21.3 ft/s the airspeed had been decreasing. Just before reaching 21.3 ft/s there was a vertical rate peak at 18.3 ft/s. When the first peak was reached the indicated airspeed began to climb with the throttle at 83%. Since the airspeed was decreasing at the second peak it was concluded that the maximum sustainable vertical rate is 18 ft/s (10.7 knots). It is likely that Noctua could climb at a higher rate since the throttle was not maxed out at 18 ft/s; however, the results of this test do not prove what that climb rate could be.

$$Max\ Climb\ Fraction = \frac{VRate_{max}}{TAS} = \frac{10.7}{34} = 0.31$$

Since the target cruise airspeed is around 34 knots the climb max fraction based on the max climb rate of 18 ft/s was calculated to be 0.31.



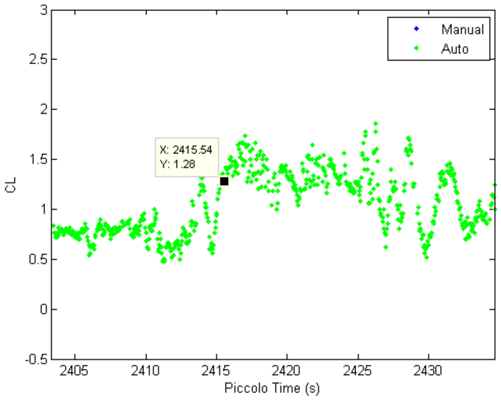
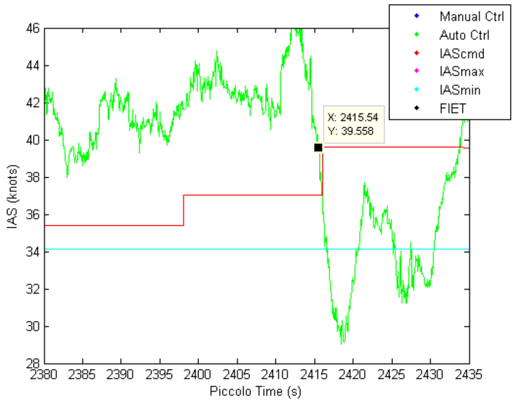


Figure 397 Z-Acceleration

Figure 398 CL

It was interesting to note how the z-acceleration limiter performed during the climb. As the CL approached CLmax (1.332) the z-acceleration limiter began to drive the z-acceleration command; however, the CL still increased up to nearly 1.7.



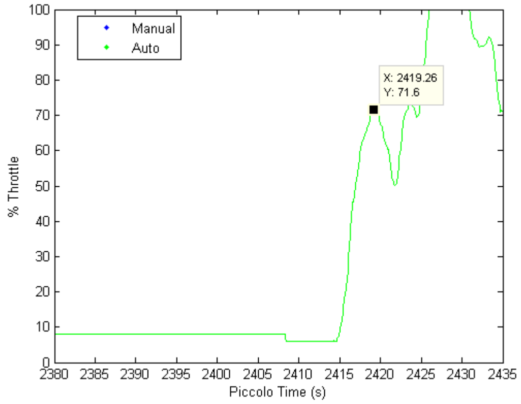


Figure 399 IAS

Figure 400 Throttle

It was also interesting to note that the z-acceleration limiter was driving the z-acceleration command before the airspeed fell below the minimum airspeed. Figure 26 shows that the airspeed was 40 knots when the z-acceleration maximum was reached which was well above the minimum airspeed of 34.1 knots. The autopilot did not switch into slow airspeed mode until 2426 seconds, which was well after the first huge airspeed undershoot where the airspeed fell down to 29 knots. Recall that the throttle plays a role in the autopilot deciding whether or not to switch into slow airspeed mode. During the first airspeed drop the throttle was not at full throttle. Additionally the throttle actually decreased even though the airspeed was 10 knots below the commanded airspeed. It appeared that elevator control responded appropriately but energy control did not. As a result an energy control analysis was performed to assess the performance of the controller.

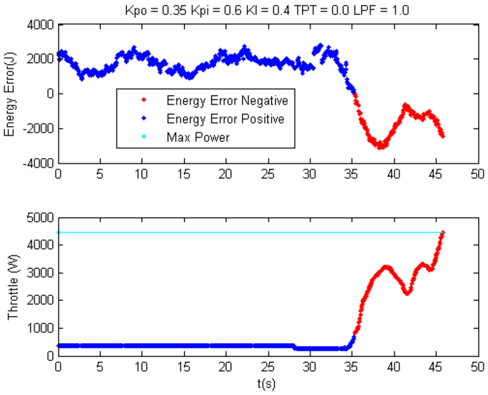


Figure 401 Energy Error

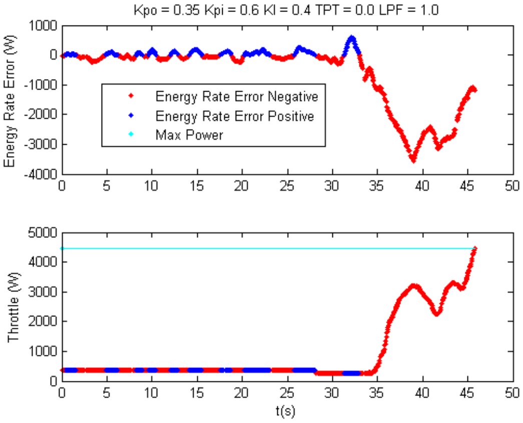


Figure 402 Energy Rate Error

Figure 28 shows that the throttle began to throttle up before there was a negative airspeed and energy error. Figure 29 shows that the throttle did not throttle up until sometime after the energy rate error was negative. Both figures show that the throttle decrease occurred at the same time that the energy and energy rate errors began to decrease. The second time that airspeed fell below the minimum airspeed corresponded with the throttle decrease; however, it only took 0.2 seconds for the controller to max out throttle and switch into slow airspeed mode. The first airspeed drop below the minimum airspeed is more concerning. Energy control could be tuned to quicken the response of the throttle to initial errors. The proportional gains could be too high compared to the integral gain. That would account for the reason as to why the throttle decreased at the same time the energy and energy rate errors began to decrease. Even though there was still a significant negative energy error the proportional gains only respond to instantaneous errors and do not account for the error over time. A larger integral gain could keep the throttle from decreasing so quickly in a similar situation. Additionally the maximum engine power term could be too high. A lower maximum engine power would have caused the throttle to initially increase at a larger magnitude in the scenario in question.

Wheeled Laser Auto Land

The autopilot successfully auto-landed Noctua on the third attempt. There was a small propeller strike just before touchdown. Additionally the aircraft was pitched nose down, -6.35 degrees, and from visual observation Noctua did not appear to flare. It was decided to analyze the time period when the flare should have been commanded.

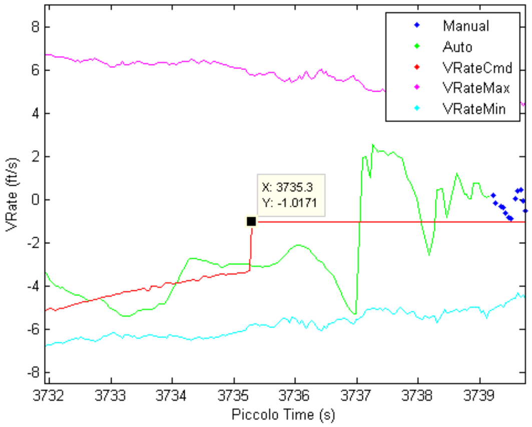
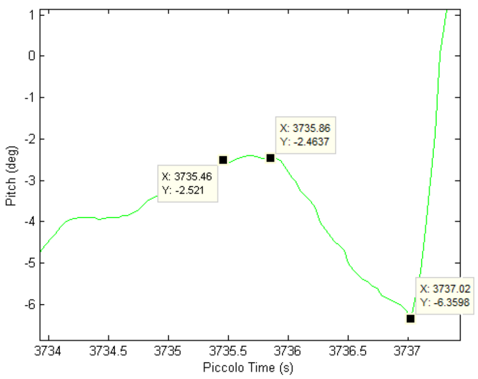


Figure 403 Flare Vertical Rate

Figure 30 shows that the autopilot did indeed command a flare. The vertical rate command increased by 3 ft/s. Figure 30 also shows that in spite of the vertical rate command the actual vertical rate of the aircraft only increased 1.5 ft/s and then began to drop. The negative vertical rate error should have caused the autopilot to command pitch up; however, the vertical rate plot indicated that it did not.



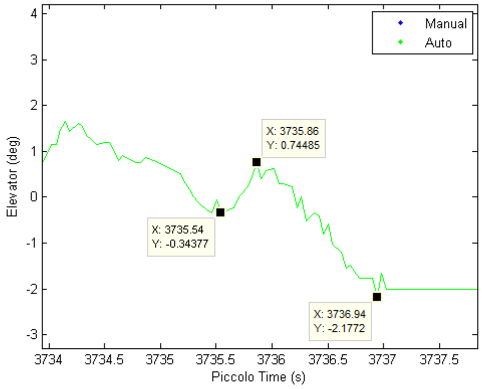
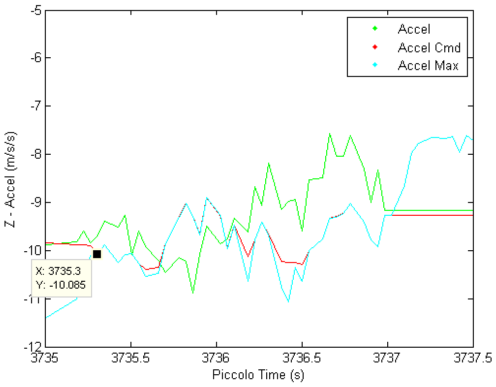


Figure 404 Flare Pitch

Figure 405 Flare Elevator Deflection

Figures 31 and 32 depict the pitch and elevator deflections during the flare maneuver. Figure 32 shows that the elevator deflection initially commanded pitch up, but at 3735.54 seconds the elevator began to command pitch down. Figure 31 shows that the pitch was increasing until the elevator deflection trend changed. At 3735.86 seconds the elevator began to command pitch up again; however, Figure 31 shows that the pitch began to drop quickly and the elevator deflection commands were too little too late as the aircraft pitched down until touchdown at 3737.02 seconds.



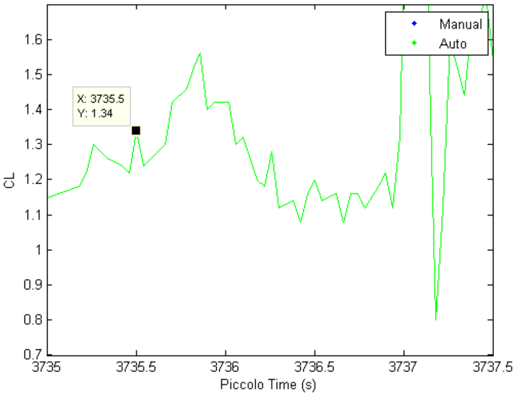
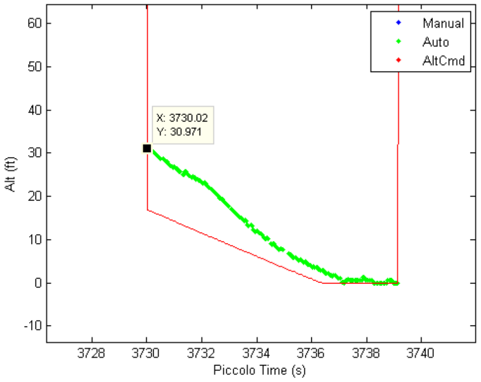


Figure 406 Flare Z – Acceleration

Figure 407 Flare CL

Figures 33 shows that the z-acceleration limiter was driving the z-acceleration command for almost the entirety of the flare. The flare maneuver began at 3735.3 seconds. Figure 34 shows that the CL did not reach CL max until 0.20 seconds after the z-acceleration command became restricted by the z-acceleration limiter. The time period of pitch down elevator deflection (3735.54 – 3737.86 seconds) corresponded to the z-acceleration command being drastically driven positive; therefore, it was concluded that the z-acceleration limiter caused the autopilot to essentially abort the flare maneuver.



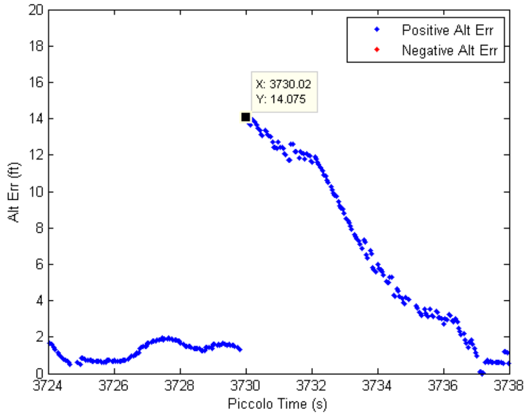


Figure 408 Short Final Laser AGL

Figure 409 Approach Altitude Error

Stopping the flare certainly had a large impact on the pitch orientation of Noctua at touchdown, but it was decided to analyze the entire short final to check for any other errors that could have contributed to the negative pitch angle at touchdown. When the laser AGL altitude was analyzed it was noticed that there was an altitude error of 14 feet when the laser took over (3730.02 seconds). The autopilot was setup to auto abort the land attempt if there was an altitude error of 5 ft. Recall that the laser was setup to take over just after the autopilot made the abort decision; however, the autopilot chose not to abort.

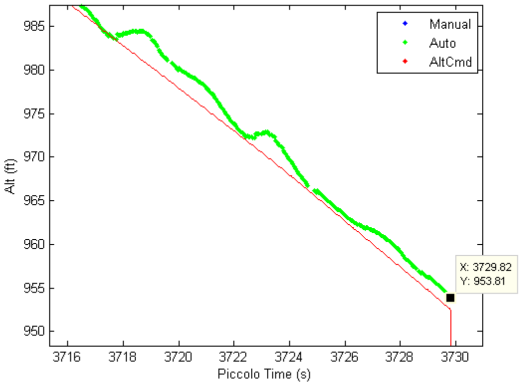


Figure 410 Final Approach Barometer Altitude

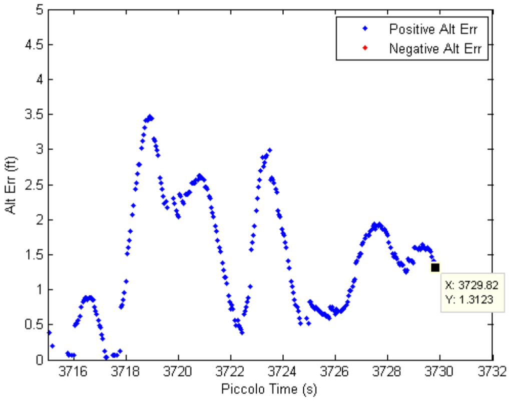


Figure 411 Final Approach Altitude Error

The final approach altitude was analyzed to determine why the autopilot did not auto abort. The data showed that according to the barometer the altitude tracking was well within the limitations set in the land settings. The altitude error at the time the decision waypoint was reached, 3729.82 seconds, was only 1.3 feet.

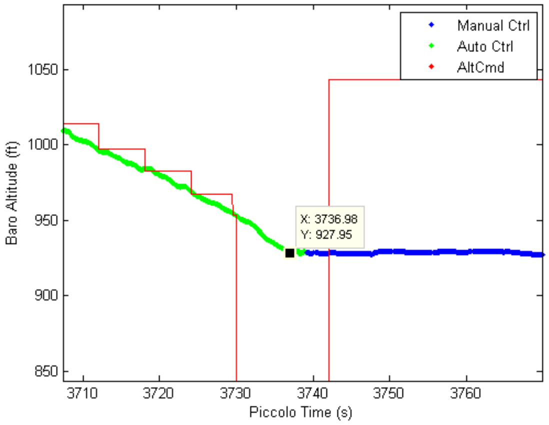
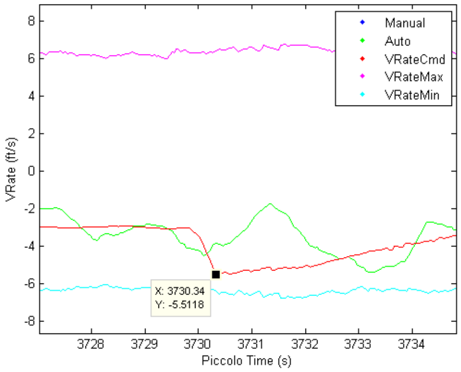


Figure 412 Touchdown Barometer Altitude

A quick analysis of the barometer altitude at touchdown showed that according the barometer the airstrip was at 928 feet; however, the altitude of the air strip is actually 935 feet. At this point it was realized the mistake that caused the altitude discrepancy to occur. The barometer was mistakenly zeroed at 925 feet instead of 935 feet during pre-launch.



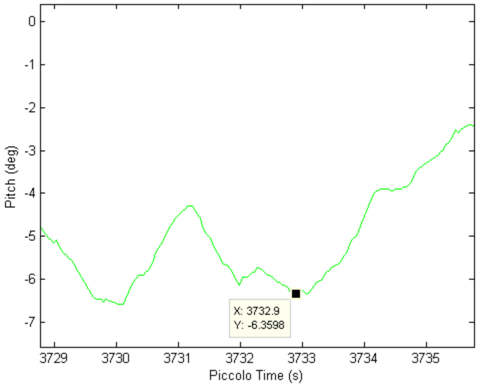


Figure 413 Vertical Rate Short Final

Figure 414 Pitch Short Final

The large altitude error after the decision point caused the autopilot to command a negative vertical rate only seconds before the flare and touchdown were set to occur. The altitude error certainly contributed to the negative pitch of the aircraft at touchdown, even though the pitch increased throughout the short final leading up to the flare.

Laser Altimeter Performance

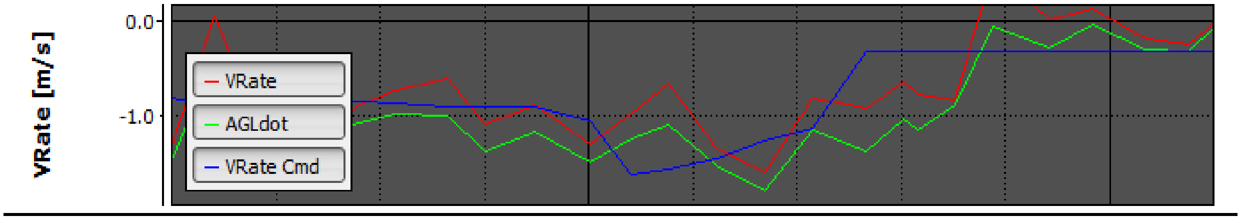
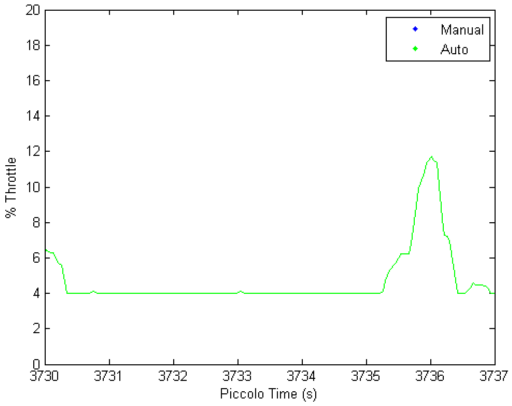


Figure 415 Vertical Rate Short Final

Since the barometer was incorrectly measured it was not possible to compare the laser altimeter AGL to the barometer AGL; however, it was possible to compare the vertical rate measurements of the laser altimeter to those of the GPS based solutions. In Figure 42 the green line is the laser altimeter vertical rate solution and the red line is the GPS based vertical rate solution. The laser vertical rate followed the trend of the GPS vertical rate with a small offset in magnitude. It seems likely that the autopilot uses a combination of both, just weighted towards the laser altimeter measurements.

Throttle Land Performance



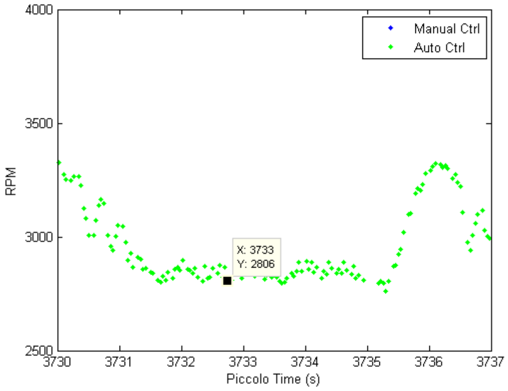


Figure 416 Short Final Throttle *Figure 417 Short Final RPM*

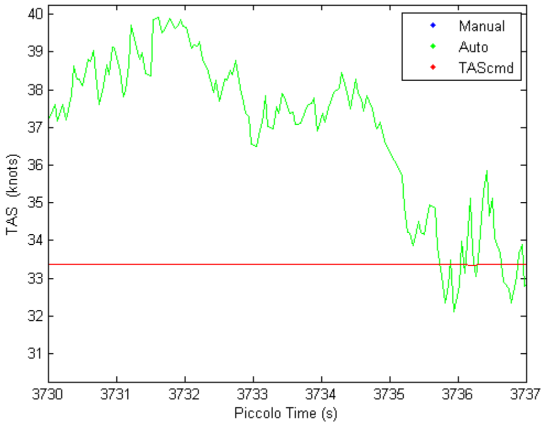


Figure 418 Short Final TAS

The minimum throttle was changed in flight to 4% so that the autopilot would be able to command throttle low enough for less than 3000 rpm. Figure 44 shows that as the throttle command was 4% the rpm was able to get down to 2800 rpm. Figure 45 shows that initially the airspeed was not able to decrease to 33.4 knots, but eventually it was able to get low enough. The drop in airspeed likely had to do with the gusting winds. It was concluded that as long as the autopilot is capable of throttling down to 2800 rpm it should be able to reach the approach airspeed.

APPENDIX D

PICCOLO MECHANICS EXPERIMENTS

1. Aileron Control Experiments

The gain definitions and control descriptions were not enough to create a detailed control schematic of the system. As a result an experiment was conducted in a software in the loop simulation to help determine the unknowns. The unknowns were as follows:

1) Where does the heading command come from?
2) How is the turn rate command converted into a bank angle command?
3) What changes when an aircraft crabs because of wind?
4) Is the roll command, reported by the DevInterface, equal to the bank angle command from the command loops?

A simulation was devised where an aircraft would fly two laps around an elongated box pattern. The first lap would have 0 wind, and the second lap would have a south wind of 5 m/s. The control loop analysis was to analyze the aircraft on the same leg of the flight path during the two separate laps. In between the two laps, before wind was turned on, a constant bank angle command would be issued in order to determine if the roll command was the same as the bank angle command. Even though PCC records the bank angle command, the update rate of command loop commands is delayed in PCC, so the recorded bank angle in PCC could not be compared against the recorded roll angle. The lateral gains were set to their default values except for the track control derivative gain. The derivative gain was set to 0 so that the turn rate command could be easily calculated from the heading error and the track control proportional gain.

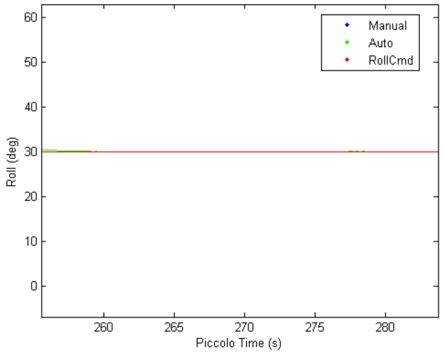
Gains:

Tracker Convergence = 0.30   Heading err to turn rate = 0.40   Heading err der to turn rate = 0.0

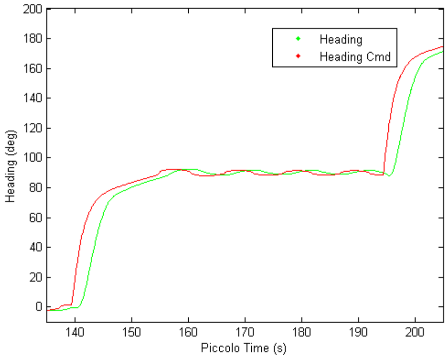
Turn err lpf cutoff = 0.0   Roll err to roll rate cmd = 1.0   Roll rate lpf cutoff = 0.0

Roll rate err to aileron = 0.0   Roll rate err int to aileron = 0

Results:

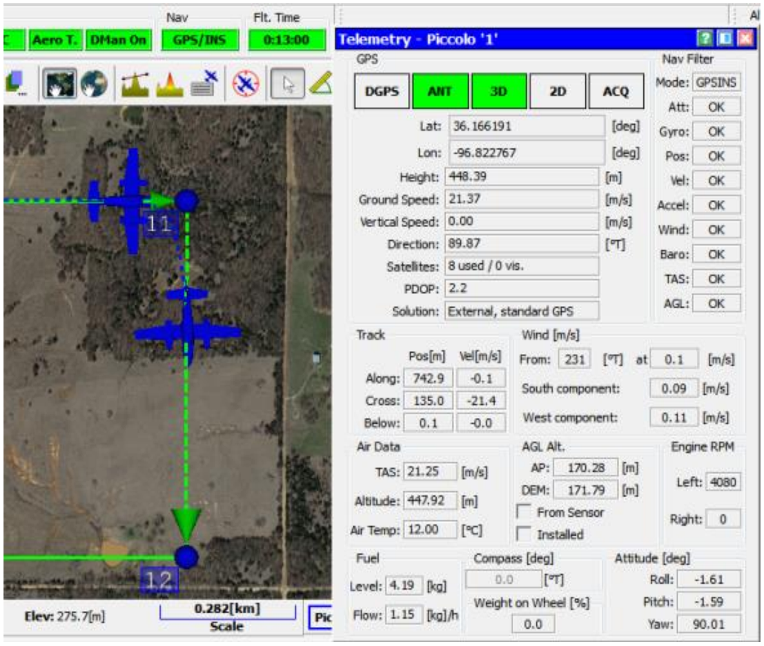


The first analysis focused on the bank angle command. A bank angle command of 30 degrees was issued for about 70 seconds. The figure above shows the roll command that corresponded with the time that the constant bank angle command was issued. The figure showed that the roll command was identical to the bank angle command.



The second analysis focused on where the heading command comes from. It was made clear in the definitions of the track control loop gains that the tracker convergence parameter is used with the true airspeed measurement to plot an elliptical trajectory. It was not clear if the elliptical trajectory was commanded by issuing heading commands and when the elliptical trajectories for flight paths will be set as commands. The figure above depicts the heading command when the autopilot targeted the north leg during the 1st lap of the simulation. The heading commands were clearly elliptical.

It had been noticed in simulations prior that if the tracker convergence gain was high enough and the flight plan was small enough the autopilot would continuously target the next waypoint without leading the aircraft to track to a flight path; hence, it was theorized that the autopilot will use the calculated semi minor axis to determine the distance from the next flight path that the autopilot will command the aircraft to track the next flight path.



Telemetry - Piccolo '1'

GPS

Lat: 36.166191 [deg]
Lon: -96.822767 [deg]
Height: 448.39 [m]
Ground Speed: 21.37 [m/s]
Vertical Speed: 0.00 [m/s]
Direction: 89.87 [°T]
Satellites: 8 used / 0 vis.
PDOP: 2.2
Solution: External, standard GPS

Nav Filter

Mode: GPSINS
Att: OK
Gyro: OK
Pos: OK
Vel: OK
Accel: OK
Wind: OK
Baro: OK
TAS: OK
AGL: OK

Track

	Pos[m]	Vel[m/s]
Along:	742.9	-0.1
Cross:	135.0	-21.4
Below:	0.1	-0.0

Wind [m/s]

From: 231 [°T] at 0.1 [m/s]
South component: 0.09 [m/s]
West component: 0.11 [m/s]

Air Data

TAS: 21.25 [m/s]
Altitude: 447.92 [m]
Air Temp: 12.00 [°C]

AGL Alt.

AP: 170.28 [m]
DEM: 171.79 [m]
□ From Sensor
□ Installed

Engine RPM

Left: 4080
Right: 0

Fuel

Level: 4.19 [kg]
Flow: 1.15 [kg]/h

Compass [deg]

0.0 [°T]

Weight on Wheel [%]

0.0

Attitude [deg]

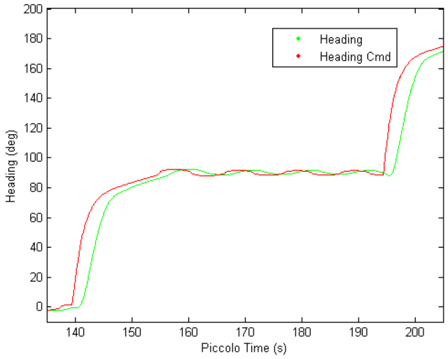
Roll: -1.61
Pitch: -1.59
Yaw: 90.01

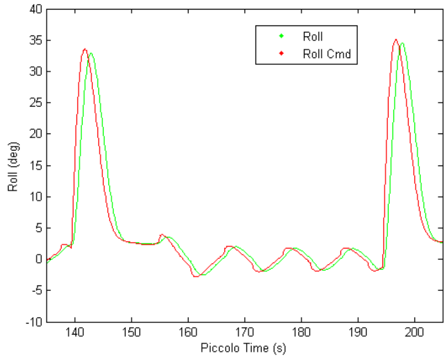
Elev: 275.7[m]
0.282[km]
Scale

The figure above depicts a turn from the first lap of the simulation. The telemetry window displays the telemetry just after the flight path between waypoints 11 and 12 was targeted. The true airspeed was 21.25 m/s and the distance of the aircraft from the newly tracked path, designated as "Cross" in the telemetry window, was 135 meters. The following calculation used the description of the semi minor axis calculation to calculate what the semi minor axis should have been.

$$Semi\ minor\ axis = TAS^2 * TC = 21.25^2 * 0.30 = 135.4\ m$$

The calculated semi minor axis was 135.4 meters, which was 0.4 meters larger than the reported cross track error of 135 meters. Since PCC logged command changes are delayed from when they actually occur it was reasonable to assume that the difference of 0.40 meters in the calculation was due to the delay; thus, it was concluded that the autopilot targets the next flight path when the distance of the aircraft from the next flight path is equal to the length of the semi minor axis.





The third analysis began by analyzing the roll command of the aircraft traveling the north leg of the box pattern in its first lap (no wind). The figure above depicts the roll command and roll in the plot on the right and the heading command and heading in the plot on the left as recorded by the DevInterface. The controller appeared to be calculating roll command from the turn rate command with a pure calculation and no control loop gains. Some research was done into a couple aerodynamics books and the following equation for calculating turn rate from bank angle was found by re-arranging a couple of equations.

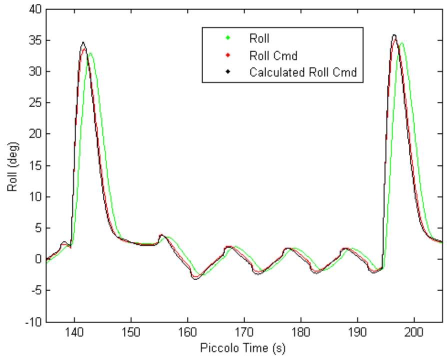
Equation 81 Turn Rate

$$\omega = \frac{g\tan(\varphi)}{V_\infty}$$

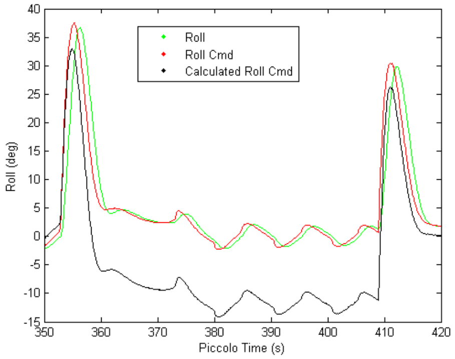
(Anderson 325)

A quick re – arrangement of Equation 81 to solve for the bank angle was made. The turn rate command was calculated from the heading error that occurred throughout the time period and the track control proportional gain.

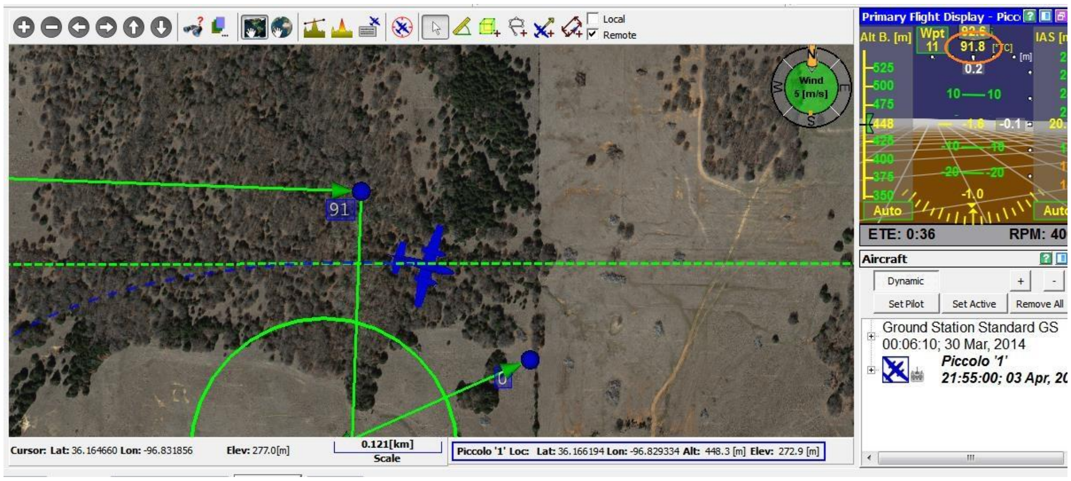
$$Turnrate_{cmd} = Heading_{error} * 0.40 \Rightarrow Roll_{cmd} = \tan^{-1}\left[\frac{Turnrate_{Cmd} * TAS}{g}\right]$$



The figure above depicts the roll command as calculated by the calculations shown above. The roll command calculation was nearly identical throughout the entire plot; however it was not always identical. There seemed to be a small error in the calculations. It was decided to check how the calculated roll command matched up during the pass with wind.



The figure above depicts the roll command calculation during the north leg portion of the 2nd lap. With wind enabled the error in the roll command calculation was amplified. It appeared that something was wrong. Upon reviewing the replay file it was noticed that the reported yaw angle in the primary flight display was not the same as the reported heading angle from the DevInterface data.



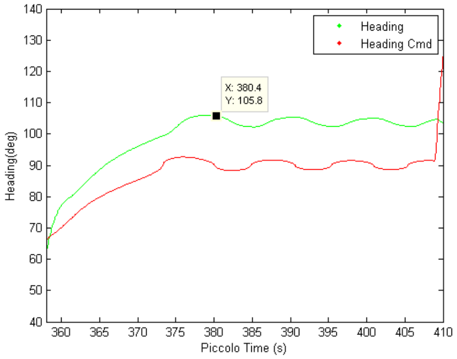
ETE: 0:36 RPM: 40

Ground Station Standard GS
00:06:10; 30 Mar, 2014

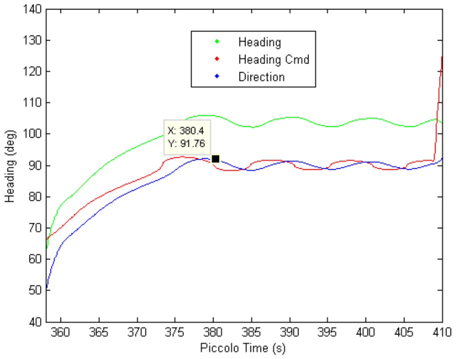
Piccolo '1'
21:55:00; 03 Apr, 20

Cursor: Lat: 36.164660 Lon: -96.831856 Elev: 277.0[m]

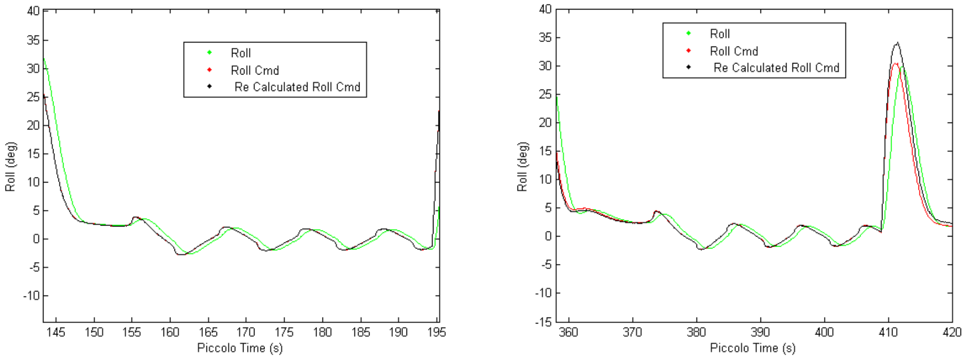
Piccolo '1' Loc: Lat: 36.166194 Lon: -96.829334 Alt: 448.3 [m] Elev: 272.9 [m]



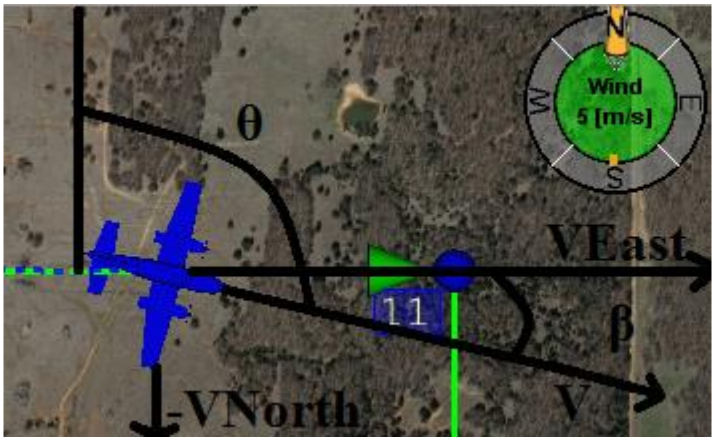
The figures above highlight the reported headings at 380.4 seconds. The heading shown in PCC was 91.8 degrees and the heading recorded by the DevInterface was 105.8 degrees. Additionally the heading recorded by the DevInterface yielded a large offset heading error that did not appear to ever be corrected nor did it appear that the autopilot attempted to correct the error. It was as if the controller was using something else as feedback for heading.



The next step was to analyze the piccolo telemetry data that corresponded to yaw angles or heading. The variable “Direction” was plotted against the heading data from the DevInterface. Direction is shown in the figure above. The direction recorded by PCC appeared to possibly be the feedback term that the controller was using for heading. In order to test that theory a roll command was calculated using the same process as before, except the heading feedback term was changed to direction so that the heading error was the difference between the heading command and direction.



The figure above depicts the re-calculated roll command against the actual roll command for the two laps. The 1st lap roll command and roll command calculations were identical. The small errors had disappeared. The 2nd lap roll command and roll command calculations were nearly identical. There were only small errors that occurred during the turns from the north leg to the east leg and from the east leg to the south leg. In order to investigate further it was decided to determine how the direction angle is calculated.

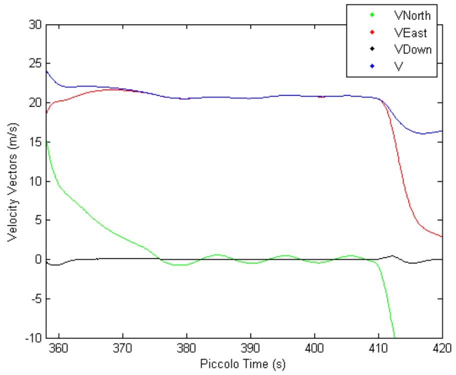


It was immediately assumed that the direction must be calculated using the velocity vectors that PCC records in the piccolo telemetry file. The velocity vectors are recorded as "VNorth", "VEast", and "VDown". It was attempted to calculate the resultant velocity vector and determine the angle between it and the vector that lied on the flight path. The figure above depicts a snapshot of the aircraft traveling on the north leg of the flight plan on the 2nd lap of the simulation. The resultant vector V was calculated along with the angle between V and VEast.

$$\vec{V}_{GroundSpeed} = \sqrt{VEast^2 + VNorth^2 + VDown^2} \;\Rightarrow\; \beta = \cos^{-1}\left[\frac{VEast}{\vec{V}_{GroundSpeed}}\right]$$

$$\theta = \frac{\pi}{2} + \beta$$

At first the calculated angle, beta, was clearly too small to be direction, but it was quickly realized that direction should refer to 0 pointing north. As a result 90 degrees was added to the calculated angle and the results matched perfectly with direction. Additionally it was noticed that the resultant velocity vector was a perfect match with groundspeed as recorded by the piccolo telemetry. It was decided to analyze the velocity vectors to look for any trends or correlations.



The figure above depicts the velocity vectors with the resultant velocity vector. The time periods where the resultant velocity vector, or groundspeed vector, had a large error compared with the VEast velocity vector corresponded to the same time periods that the roll command calculation was off from the actual roll command. It was decided that the error in roll command estimation must be due to some difference in how the autopilot decides what portion of its GPS/INS solution to use to determine the actual state of the aircraft; thus, determining what GPS/INS solution to use for heading feedback. It is possible that the autopilot could use a different solution, other than direction, as heading feedback when there are large errors between the velocity vectors and the resultant vector, or when the aircraft is transitioning between two flight paths.

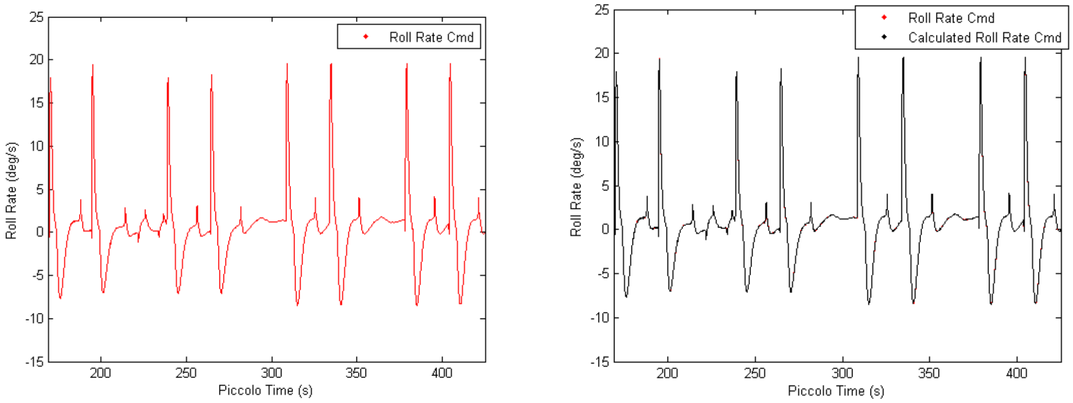
The roll control gain definitions indicated that there are two different paths that issue aileron deflection commands from the roll control loop. The gain definitions implied that the aileron effectiveness parameter is used as a feed forward gain and used to scale the feedback loop gains. In order to test exactly how roll control functions a software in the loop simulation was setup to determine the effects of each gain.

Roll Control Gains		Vehicle Parameters	
Roll err to roll rate cmd	1.0	Wingspan (m)	4.007
Roll rate lpf cutoff	0.0	Aileron Effectiveness	-0.51893
Roll rate err to aileron	1.0		
Roll rate err int to aileron	0.0		

The simulation began with the gains shown in the table above. In the simulation one lap was flown with the initial gains. After the first lap was completed “Roll rate err to aileron” was set to 0 so that only the feed forward path could command aileron deflections.

The first step in the analysis was to determine how the roll rate command is calculated. It seemed straight forward from the definition that the roll rate command is calculated from the feedback of the roll command and the estimated actual roll angle. It was determined that it was most likely that the autopilot was multiplying the roll error by the outer loop proportional gain to calculate the roll rate command. The equation below was used to calculate the roll rate command and the calculated results were compared against the actual roll rate command that existed throughout the entire simulation.

$$Roll\ Rate\ Cmd = (Roll\ rate\ err\ to\ roll\ rate\ cmd) * Roll_{err}\ \ (rad/s)$$



The left plot in the figure above depicts the roll rate command and plot on the right depicts the calculated roll rate command with the actual roll rate command. The calculation produced the same roll rate command that the autopilot commanded. The second analysis focused on the aileron deflections that the feed forward route commanded.

$$Aileron\ Effectiveness = \frac{pb/_{2V}}{\delta_a}$$

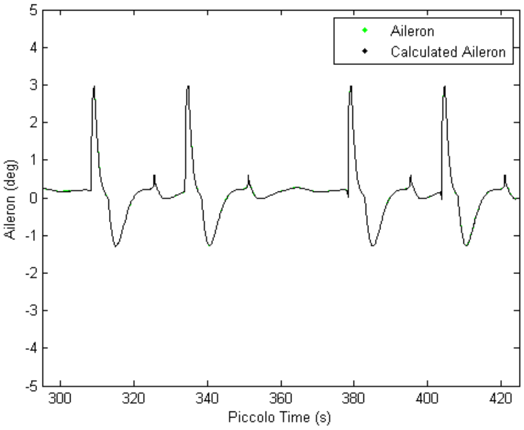
$$where\ p = roll\ rate\ (rad/s)\ \ b\ = wingspan\ (m)\ \ V\ = true\ airspeed\ (m/s)$$

(Tuning piccolo control laws 2.0.x 6)

Aileron effectiveness is defined as the dimensionless roll rate divided by aileron deflection. It was theorized that the autopilot multiplies the roll rate command by the wingspan and divides it by the true airspeed and aileron effectiveness value to calculate aileron deflection commands. Aileron deflection commands were calculated using Equation 82 below and compared against the actual aileron deflection commands in the simulation for the time period that corresponded with the period where the feedback gains were zero.

Equation 82 Aileron Deflection Feed Forward

$$\delta_a = \frac{p_{cmd} * b}{2 * TAS * AE}$$

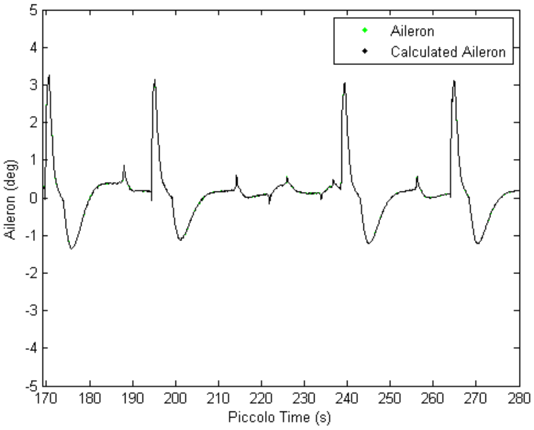


The figure above depicts the calculated aileron deflections with the actual aileron deflections. The results indicated that the calculated aileron deflections were correct. The third analysis focused on the first lap of the simulation when the proportional feedback gain was set to

1.0. It was theorized that similar to the feed forward route the roll rate error is scaled in the same way that the roll rate command was. It was assumed that the roll rate error would be multiplied by the proportional gain value just as the roll error was in the outer loop. Equation 83 below was used to calculate the aileron deflections for the time period that corresponded with the feedback proportional gain.

Equation 83 Aileron Deflection Proportional and Feed Forward Gains

$$\delta_a = \frac{(p_{error})(Roll\ Rate\ err\ to\ aileron) * b}{2 * TAS * AE} + \frac{p_{cmd} * b}{2 * TAS * AE}$$



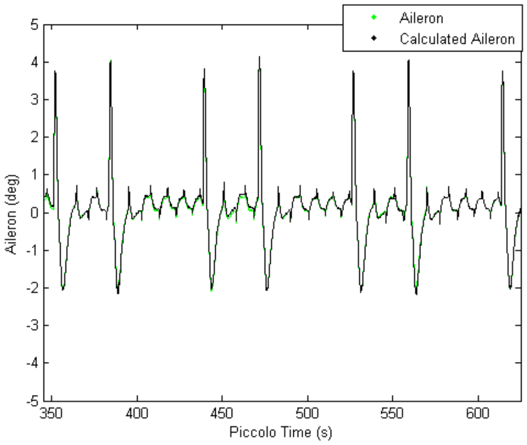
The figure depicts the calculated aileron deflections with the actual aileron deflections. The results indicated that the calculated aileron deflections were correct. It was theorized that the integral gain acts in concert with the inner loop proportional gain to command aileron deflections. In order to test the theory a software in the loop simulation was setup where the inner loop integral and proportional gains would both be utilized by roll control. The table below contains the gains that were used in the simulation. Equation 84 below was used to calculate what the aileron deflections would be.

Roll Control Gains		Vehicle Parameters	
Roll err to roll rate cmd	1.0	Wingspan (m)	4.007
Roll rate lpf cutoff	0.0	Aileron Effectiveness	-0.51893

Roll rate err to aileron	1.0	
Roll rate err int to aileron	1.0	

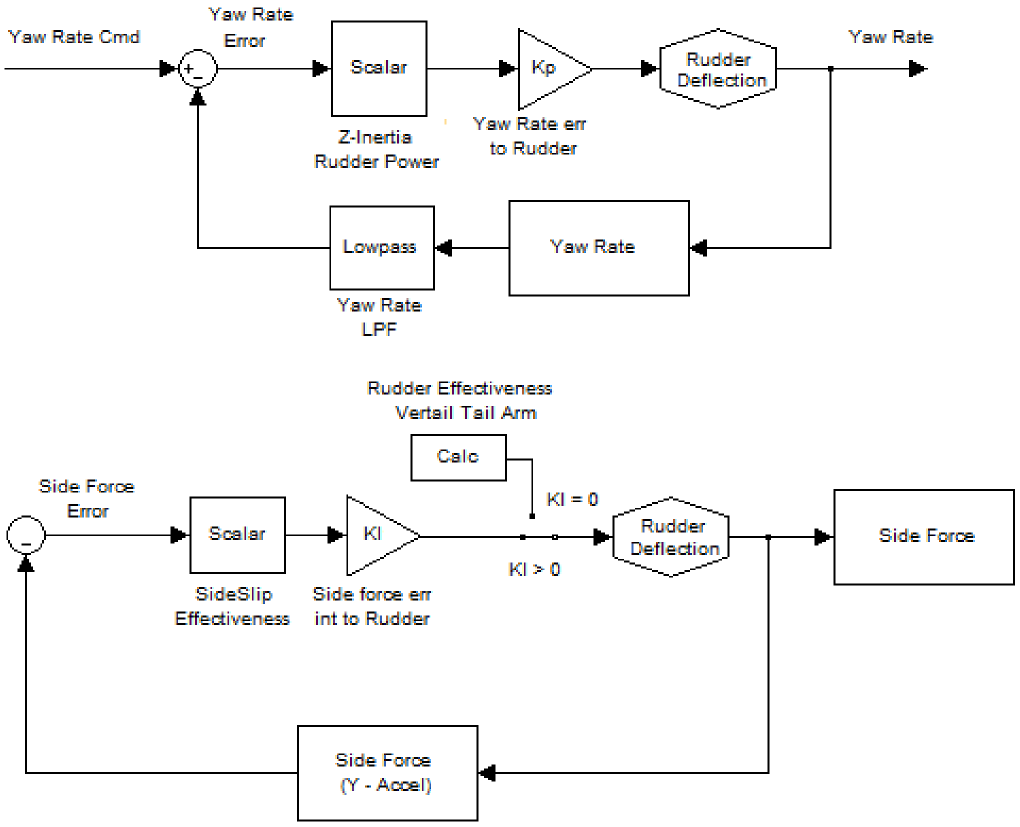
Equation 84 Aileron Deflection with Feedback and Feedforward Gains

$$\delta_a = \frac{\left[p_{error} * K_{prr} + K_I \int p_{error} dt\right] * b}{2 * TAS * AE} + \frac{p_{cmd} * b}{2 * TAS * AE}$$



The figure above depicts the calculated aileron deflections against the actual aileron deflections. The calculated deflections were almost nearly identical. There were time spans where the calculated deflections matched the actual and there were times where it didn't. It was concluded that the error in the calculated results was due to the way that the integral of the roll rate error was being calculated. The roll rate error was integrated by iteratively calculating the area underneath the line between each roll rate error data point and the data point just before it; thus, the calculation was constrained to the telemetry rate. The autopilot likely integrates at a different rate other than the telemetry rate.

2. Rudder Control Experiments

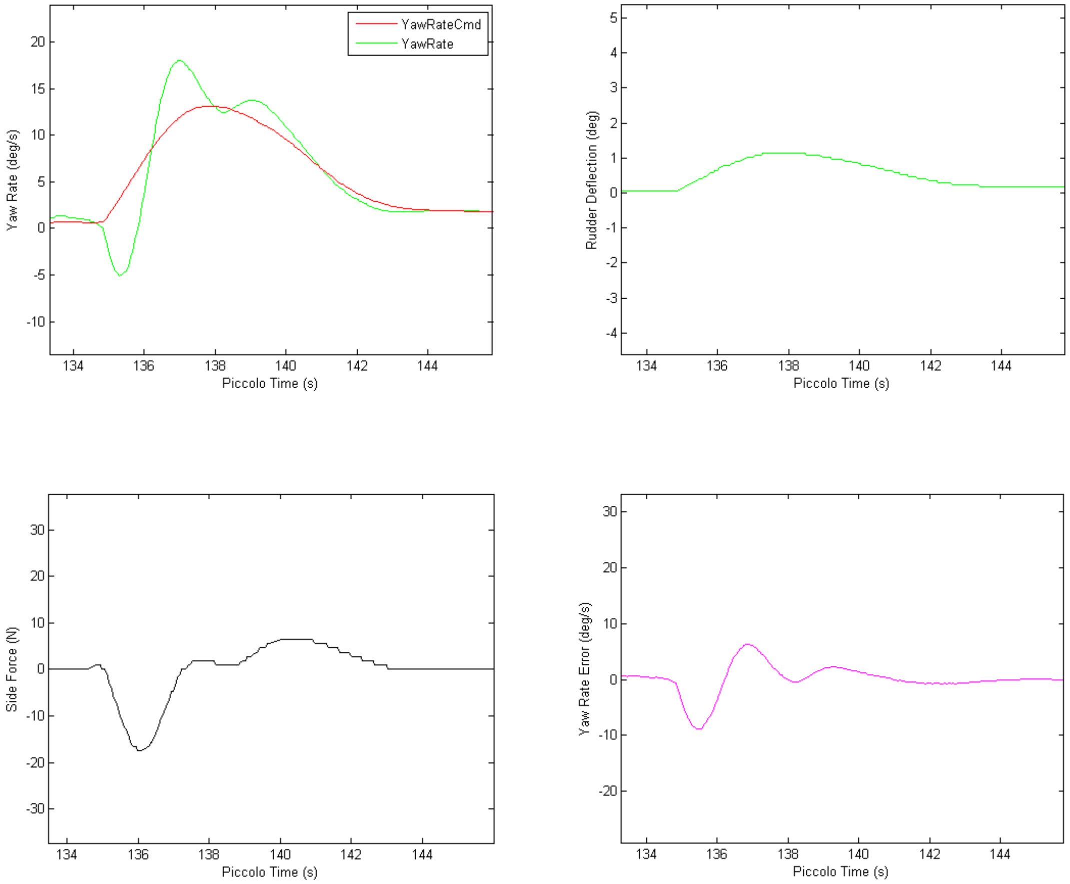


The control loop definitions of the yaw control gains were not enough to fully explain yaw control. The figure above shows yaw control defined purely by the gain definitions. Initially there seemed to be three separate paths that could command rudder deflections. The first question asked about yaw control was the following:

1) What is the input into the Rudder Effectiveness, Vertical Tail Arm calculation?

A software in the loop simulation was devised where an aircraft would fly a box pattern with all of the yaw control feedback gains set equal to zero. In such a scenario only the rudder effectiveness and vertical tail arm could be used to calculate rudder deflection since their gain definitions insinuated that they exist outside of the yaw damper and side force control loops. The table below depicts the gains and vehicle parameters that were used in the simulation.

Yaw Control Gains		Vehicle Parameters	
Yaw Rate lpf cutoff	0.0	Vertical Tail Arm (m)	1.28
Yaw Rate err to Rudder	0.0	Rudder Effectiveness	-0.68230
Side force err int to rudder	0.0	Rudder Power (/deg)	0.000573
		Sideslip Effectiveness (/deg)	-0.007181



The figure above depicts the yaw rate, rudder deflection, side force, and yaw rate error that occurred throughout one of the turns in the simulation. The rudder deflection plot shows that rudder deflections were commanded during the turn even though all of the yaw control loop gains were 0. The rudder deflections did not reflect the behavior of the side force and yaw rate errors as expected since all of the feedback gains were 0. The rudder deflections were proportional to the yaw rate command. It was concluded that the rudder effectiveness and vertical tail arm calculation received its input from the yaw rate command.

The conclusion lead to another question. How is the yaw rate command determined? In order to begin analyzing it was decided to analyze how the rudder effectiveness and vertical tail arm parameters could be used to calculate rudder deflections. It was theorized that figuring out the calculation would lead to insight into what the yaw rate command would have to consist of. It was assumed that the yaw rate command was multiplied by some form of the rudder effectiveness and vertical tail arm parameters in order to command rudder deflections.

A software in the loop simulation was devised to test the effects of zeroing the rudder effectiveness and vertical tail arm parameters when yaw control could only use path 2. The yaw control gains were left at zero to continue to force yaw control to command rudder deflections through path 2 only. The initial gains used were exactly the same as before. PCC would not let the rudder effectiveness parameter be set to 0; however, it was found that entering “0.0000001” would work. Note that PCC rounds parameter values to six decimal places so the final value was indeed 0. When the rudder effectiveness was set to zero the rudder deflection commands oscillated from -19 to +19 which was the rudder max and rudder min. The result indicated that the controller was essentially commanding negative and positive infinity meaning that the rudder effectiveness term is in the denominator. The vertical tail arm was set to 0 while the rudder effectiveness was still 0 and the aircraft was oscillating out of control. As soon as the vertical tail arm parameter was set the rudder deflections went to 0. The result indicated that the vertical tail arm is in the numerator, since zeroing it yielded a rudder deflection of 0. Note that when the vertical tail arm value was set to 0 the digits “01” had to be added to the end of the rudder effectiveness value to keep PCC from automatically setting the rudder effectiveness to a default value. In order to test further the rudder effectiveness term was set back to its proper value of -0.6823. Even with a rudder effectiveness value the rudder deflection commands were 0.

$$RE = rudder\ effectiveness = {}^{\Delta\beta}/_{\Delta\delta_r}$$

(PccUsersGuide 116)

$$\frac{l_v}{RE} = \frac{l_v}{{}^{\Delta\beta}/_{\Delta\delta_r}} \Rightarrow \frac{r_{cmd} * l_v}{{}^{\Delta\beta}/_{\Delta\delta_r}} = \Delta\delta_r$$

Equation 85 Rudder Deflection Feed Forward

The results indicated that the yaw rate command is multiplied by the vertical tail arm and divided by the rudder effectiveness (l_v / RE). It was known from the vehicle gain definitions that the rudder effectiveness is change in sideslip over change in rudder deflection. The results combined with the vehicle parameter definitions lead to the equation shown above. It was determined that in order for the equation to be correct the yaw rate command would have to have the sideslip angle in its numerator and the vertical tail arm in its denominator. Research was done into the equations of motion regarding yawing rate. It was found that the sideslip angle could be defined as the following:

$$\Delta\beta = \frac{-rl_v}{u_0}$$

(Nelson 118)

$$\Rightarrow r = \frac{-\Delta\beta * u_0}{l_v}$$

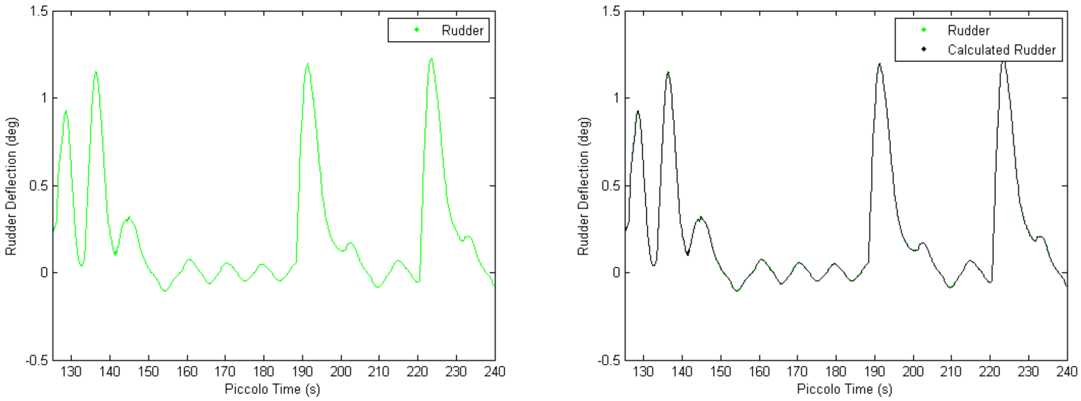
Equation 86 Yaw Rate Defined by Sideslip

It was found that the sideslip equation could be re arranged to define the yaw rate as a function of sideslip angle in the numerator and vertical tail arm in the denominator as shown above in Equation 86.

$$\Rightarrow \quad \Delta\delta_r = \frac{r_{cmd}}{\Delta\beta / \Delta\delta_r} * \frac{l_v}{TAS}$$

Equation 87 Rudder Deflection Feed Forward

The yaw rate and rudder deflection equations were combined so that all of the terms would cancel out to rudder deflection. In order to test Equation 87 another simulation was devised to test the calculated rudder deflections against the actual rudder deflections. The simulation used the exact same gains as before so that only path 2 could be used to command rudder deflections.



The figure above depicts the rudder deflections that occurred throughout the simulation and the calculated rudder deflections. They were an exact match; thus, it was concluded that path 2 calculates rudder deflections using the equation above. Additionally this conclusion lead to the conclusion that the yaw rate command must be calculated with the yaw rate equation stated; thus, the yaw rate command is dependent on the estimated sideslip angle of the aircraft.

The next unknown that was investigated was that of the yaw rate feedback loop. The yaw rate feedback loop is refered to as the yaw damper. The gain definitions indicated that the yaw rate error is routed through the yaw rate proportional gain “yaw rate err to rudder” and then scaled by the rudder power and z – inertia vehicle parameters in order to command rudder deflections. It was assumed that the yaw rate error is multiplied by the yaw rate error proportional gain and then multiplied by the scaling term that includes the rudder power and z – inertia. In order to determine if the vehicle parameters were located in the numerator or denominator a simulation was devised to zero each applicable vehicle parameter and analyze the resulting rudder deflections.

Yaw Control Gains		Vehicle Parameters	
Yaw Rate lpf cutoff	0.0	Vertical Tail Arm (m)	0.0
Yaw Rate err to Rudder	1.0	Rudder Effectiveness	0.0000001
Side force err int to rudder	0.0	Rudder Power (/deg)	0.000573
		Sideslip Effectiveness (/deg)	-0.007181
		Z – Inertia (kg-m^2)	10.614

The table above depicts the gains that were used for the simulation. The vertical tail arm and rudder effectiveness vehicle parameters were zeroed to ensure that path 2 did not influence rudder deflections and to ensure that neither of the parameters were used in path 3. After the simulation had begun the rudder deflections were initially normal with no large oscillations. The first vehicle parameter that was zeroed after the simulation started was the z – inertia. Similar to rudder effectiveness z – inertia had to be set to “0.0000001” to avoid PCC setting it to a default value. The rudder deflection commands became 0 once the z – inertia was set to 0 which indicated that the z – inertia parameter lies in the numerator of the scaling term. The z – inertia was set back to 10.614 and the rudder power parameter was set to “0.0000001”. As soon as the rudder power was zeroed the autopilot began to command rudder deflections oscillating between -19 and +19 degrees. The result indicated that the autopilot was essentially commanding negative and positive infinity; hence, the rudder power term must be in the denominator of the scaling term.

$$Scaling\ Term = \frac{I_z}{RP} \Rightarrow \frac{r_{error} * K_p * I_z}{{\Delta C_n}/{\Delta \delta_r}} = \Delta \delta_r$$

Equation 88 Proposed Rudder Deflection Feedback Calculation

The results indicated that the yaw rate feedback was multiplied by the z – inertia and divided by the rudder power to calculate rudder deflection. It was known from the gain definitions that the rudder power is defined as change in yawing moment coefficient over change in rudder deflection. It was immediately noticed that the units of Equation 88 above did not seem to be correct.

$$\frac{(r_{error} * K_p) * kg * m^2}{{}/{rad}}$$

The units had mass and length combined with radians. It was believable that the yaw rate error combined with the proportional gain could be treated as some type of time constant with regards to units but it seemed hard to believe that the combination could be treated as having units of mass and length. Research was done regarding the derivatives of yawing motion. It was found that Equation 88 nearly resembled the rudder control derivative of yawing motion, Equation 89 as shown below.

Equation 89 Rudder Control Derivative

$$N_{\delta r} = \frac{\left({\Delta C_n}/{\Delta \delta_r}\right) * QSb}{I_z} \ \left(\ {}/{s^2}\right)$$

$$where\ Q = Dynamic\ Pressure\ \left(\frac{N}{m^2}\right)\ \ S = Wing\ Area\ (m^2)\ b = Wingspan\ (m)$$

(Nelson 191)

It seemed reasonable that the yaw rate error feedback term could be treated as the rudder control derivative with units of seconds squared. The equation was re – arranged with the rudder control derivative set as the yaw rate feedback term.

$$N_{\delta r} := (r_{error} * K_p) \Rightarrow (r_{error} * K_p) = \frac{\left({}^{\Delta C_n}/_{\Delta \delta_r}\right) * QSb}{I_z}$$

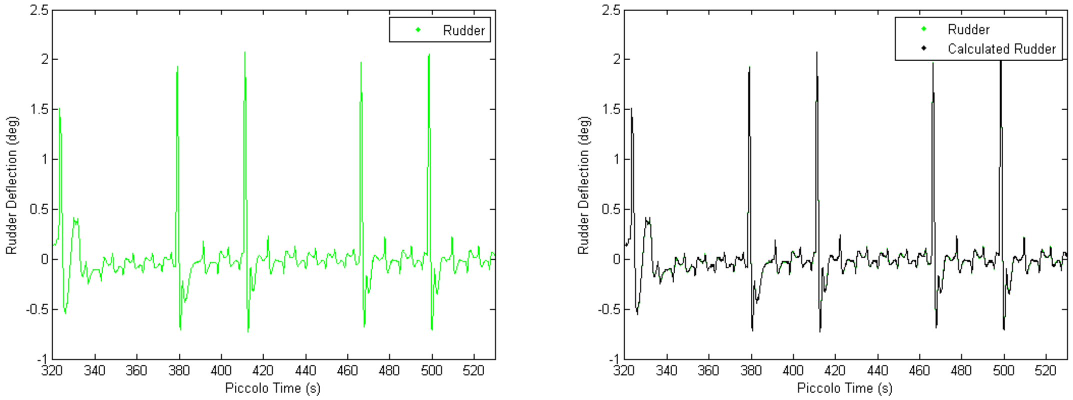
$$\Rightarrow \frac{(r_{error} * K_p) * I_z}{\left({}^{\Delta C_n}/_{\Delta \delta_r}\right) * QSb} = \Delta \delta_r$$

Equation 90 Rudder Deflection Yaw Damper Feedback

$$\Rightarrow \frac{\left(\ /_{s^2}\right)(kg * m^2)}{\left(\ /_{rad}\right)\left(\frac{kg * m}{s^2 * m^2}\right)(m^2)(m)} = rad$$

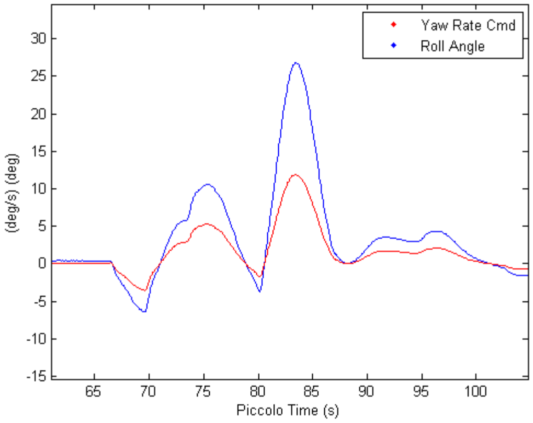
After the equation was re – arranged the units were found to cancel out to radians. It appeared that the equation could be the equation that yaw control uses to calculate rudder deflections through the yaw rate damper. A simulation was devised to test the rudder deflection prediction against the actual rudder deflection commands. The gains were set so that only the yaw rate damper feedback loop could be used to command rudder deflections as shown in the table below.

Yaw Control Gains		Vehicle Parameters	
Yaw Rate lpf cutoff	0.0	Vertical Tail Arm (m)	0.0
Yaw Rate err to Rudder	1.0	Rudder Effectiveness	0.0000001
Side force err int to rudder	0.0	Rudder Power (/deg)	0.000573
		Sideslip Effectiveness (/deg)	-0.007181
		Z – Inertia (kg-m^2)	10.614



The figure above depicts the rudder deflections that occurred throughout the simulation and the calculated rudder deflections. They were an exact match; thus, it was concluded that the yaw rate damper uses Equation 90 to calculate rudder deflection. The results of the test raised another question. If the vertical tail arm is 0 then where is the yaw rate command coming from? According to Equation 86 the yaw rate command is dependent on the vertical tail arm. The gain definition of the vertical tail arm indicated that the parameter is used for turn coordination. The definition also indicated that the vertical tail arm could be set to 0 to disable turn coordination. It seemed reasonable to believe that the yaw rate command would change if turn coordination was disabled.

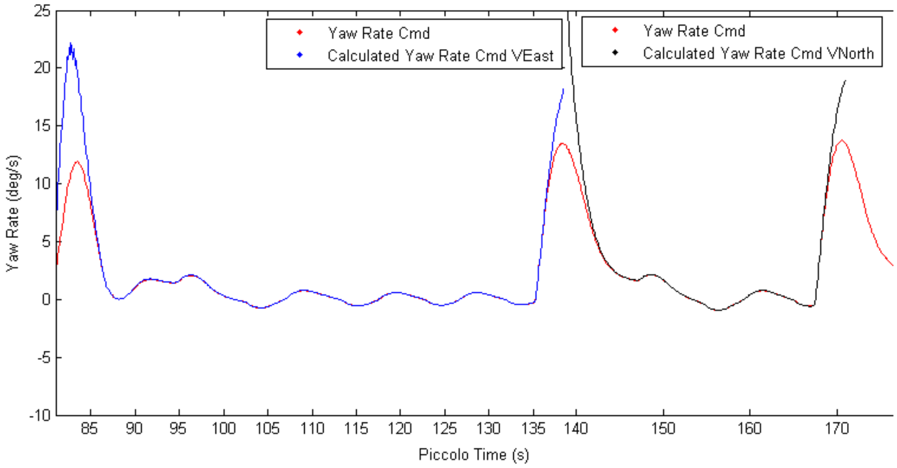
Yaw Control Gains		Vehicle Parameters	
Yaw Rate lpf cutoff	0.0	Vertical Tail Arm (m)	0.0
Yaw Rate err to Rudder	1.0	Rudder Effectiveness	-0.6823
Side force err int to rudder	0.0	Rudder Power (/deg)	0.000573
		Sideslip Effectiveness (/deg)	-0.007181
		Z – Inertia (kg-m^2)	10.614



Another simulation was ran where the vertical tail arm was set to zero to disable turn coordination. The gains that were used are shown in the table above. The yaw rate commands of the simulation were analyzed and compared to other control loop performance parameters. It was discovered that when the vertical tail arm is zero the yaw rate command is exactly proportional to the roll angle, as depicted in the figure above. It appeared that the yaw rate command could come from the turn rate calculated from the actual roll angle of the aircraft. The turn rate equation, Equation 81, was used with different variations of the velocity variable.

$$\omega = \frac{g\tan(Roll)}{V}$$

It was discovered that the yaw rate command is set equal to the turn rate of the aircraft as determined by the roll angle and the velocity vector in the direction of the target flight path. For example if an aircraft was targeting a flight path that traveled east and west the velocity vector used would be "VEast" which is the east west velocity vector recorded by PCC.



The figure above is a combination of two plots from the simulation. From 87 to 137 seconds the autopilot was targeting a flight path that traveled east. From 145 to 168 seconds the autopilot was targeting a flight that traveled south. The figure showed that the yaw rate command calculations were correct when they were using the proper velocity vector. The blue line represented the yaw rate command calculation using VEast as the velocity vector. The black line represented the yaw rate command calculation using VNorth as the velocity vector. Similarly to track control roll commands it appeared that the autopilot uses a different velocity vector during transitional periods to calculate the yaw rate commands.

It seemed reasonable to assume that the side force control loop, via path 1, operates the same way as paths 2 and 3. The gain and vehicle parameter definitions made it clear that the side force control loop is used only when there is a value for the integral gain, “side force err int to rudder”. Additionally the definitions made it clear that path 2 is disabled when path 1 is used. According to the definitions the only vehicle parameter that the side force control loop uses is the sideslip effectiveness, which is used as an error feedback scaling term. A problem arose when the calculations of the side force control loop were analyzed.

$$K_I \int Y_{error} dt * \frac{\Delta C_y}{\Delta \beta}$$

Equation 91 Side Force Feedback based on gain definitions

The sideslip effectiveness is defined as change in side force coefficient over change in sideslip angle. There were not any terms to cancel out the sideslip angle. Additionally no rudder deflection term existed. A simulation was run where the side force control loop was utilized for turn coordination so that other vehicle parameters could be tested for their possible effects on path1 rudder commands. The gains used are shown in the table below.

Yaw Control Gains		Vehicle Parameters	
Yaw Rate lpf cutoff	0.0	Vertical Tail Arm (m)	1.28
Yaw Rate err to Rudder	0.0	Rudder Effectiveness	-0.6823
Side force err int to rudder	1.0	Rudder Power (/deg)	0.000573
		Sideslip Effectiveness (/deg)	-0.007181
		Z – Inertia (kg-m^2)	10.614

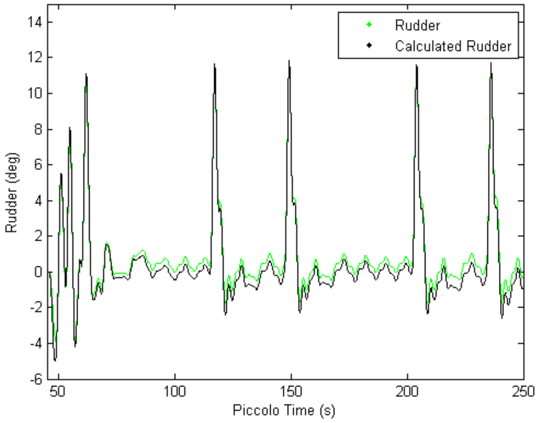
It seemed obvious that the most likely vehicle parameter that could be used is the rudder effectiveness because rudder effectiveness is change in sideslip over change in rudder deflection. After the simulation began the rudder effectiveness parameter was set to “0.000001”. As soon as the command was sent over the autopilot rudder deflection commands oscillated between -19 and +19 degrees which was the minimum and maximum rudder deflections. The autopilot was essentially commanding negative and positive infinity rudder deflection commands. The vertical tail arm was set to 0 to make sure that path 2 was not somehow still affecting the rudder deflection commands. Zeroing the vertical tail arm did not eliminate the oscillating rudder commands; therefore, it was concluded that the rudder effectiveness is included in the denominator of the side force control loop scaling term even though the definition of rudder effectiveness states otherwise. The rudder effectiveness was set back to its original value of -0.6823 and the rudder deflections were allowed to settle back to normal. Once the aircraft was stable the sideslip effectiveness was set to “0.000001”. As soon as the command was sent over the autopilot rudder deflection commands oscillated between -19 and +19 degrees. It appeared that the sideslip effectiveness parameter exists in the denominator of the scaling term as well.

$$\frac{K_I \int Y_{error} dt}{\left({\Delta C_y}/{\Delta\beta}\right)\left({\Delta\beta}/{\Delta\delta_r}\right)} * \frac{m}{QS} = \Delta\delta_r$$

Equation 92 Rudder Deflection Side Force Feedback

$$where\ Q = Dynamic\ Pressure\ \left(\frac{N}{m^2}\right)\ \ S = Wing\ Area\ (m^2)$$

Equation 92 above was determined to be the closest estimation of how the rudder deflections are commanded via path 1. The definition of the side force integral gain clearly states that it is integrating the side force error which it describes as having units of m/s^2 * s; thus, it was concluded that the integral gain is integrating the y acceleration error. The y acceleration measurements needed to be transformed into the coefficient of side force (C_y) so that the units would work out properly; therefore, it was concluded that the scaling term must also include mass divided by dynamic pressure and wing area.



A simulation was run to test the equation against actual rudder deflection commands. The gains were the same as the previous simulation except that no gains were zeroed throughout the simulation. The figure above depicts the rudder deflections versus the calculated rudder deflections of the simulation. The results were nearly identical with an offset that varied in magnitude. The offset was likely due to the way that the integral of the side force error was calculated. The manual integration was a rudimentary estimation that estimated a rectangle area between two points capped by a triangle area at the top. The magnitude of the offset varying indicated that the incorrect variable was changing throughout time; hence, it was concluded that the offset was due to the way that the side force error was integrated manually.

3. Elevator Control Experiments

3.1.1. Z – Acceleration Control Loop

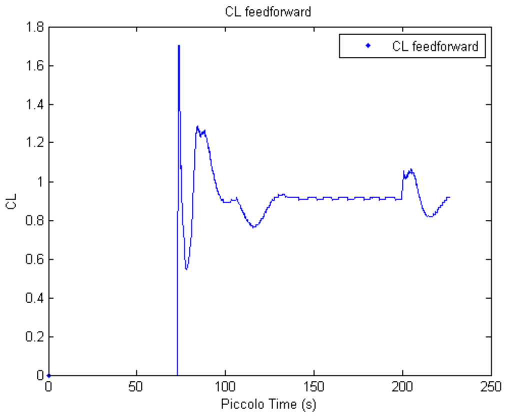
In order to numerically calculate elevator deflection commands from the z – acceleration control loop experiments were conducted in software in the loop simulations. The entire process went through numerous simulations where different gains were set to zero to test the effects on elevator deflections. Unfortunately the ability of the autopilot to maintain altitude was compromised each time z acceleration control gains were disabled so conclusions were not able to be drawn from zeroing gains. Equations were developed as to how the z – acceleration control loop could be calculating elevator deflections from the gain definitions and logical reasoning.

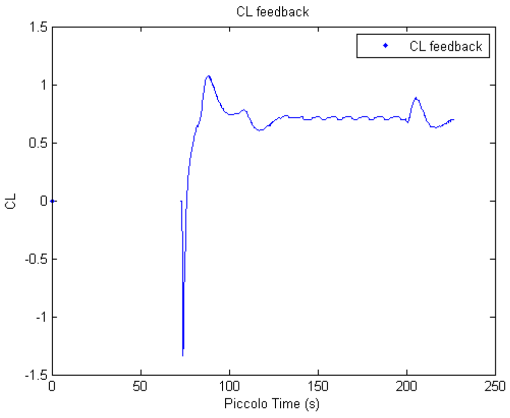
$$C_{L_{feedforward}} = \frac{-Z_{cmd} * m}{q * S_w} \qquad C_{L_{feedback}} = \frac{\left[K_{pz} * Zerror + K_I * \int Zerror\, dt\right] * m}{q * S_w}$$

$$\delta_{e1} = EPT * \left[\frac{C_{L_{feedforward}} - C_{L_{\delta_{e0}}}}{EE}\right] \qquad \delta_{e2} = \left[\frac{C_{L_{feedback}} - C_{L_{\delta_{e0}}}}{EE}\right]$$

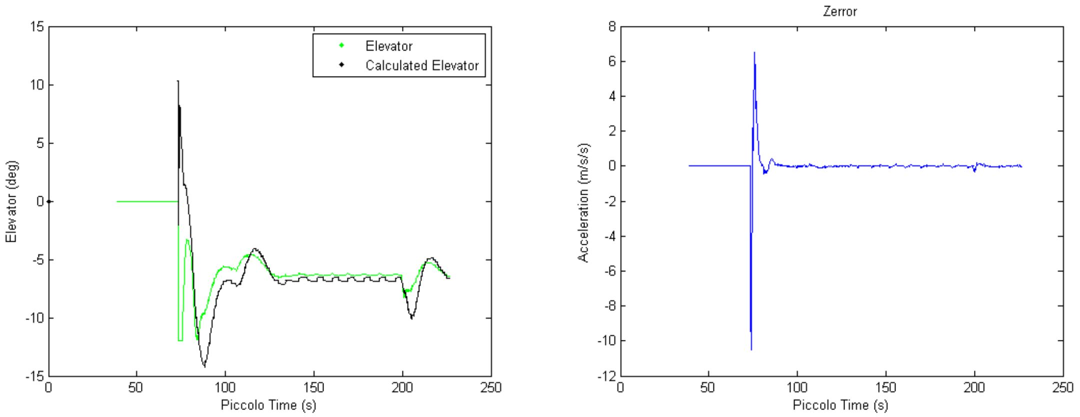
The equations above were tested in a software in the loop simulation. In the simulation the simulator was started with the aircraft suspended in air. The aircraft was commanded to hold an altitude throughout the majority of the simulation. At the end the aircraft was commanded to climb 50ft. The gains that were set are shown in the table below.

Z Acceleration Control Gains		Vehicle Parameters	
Elevator Prediction Trust	1.0	Wing Area (m^2)	1.266
Accel lpf cutoff	0.0	Elevator Effectiveness (/deg)	-0.095399
Accel err to elevator	0.25	Empty Mass (kg)	26.989
Accel err int to elevator	1.5	Gross Mass (kg)	30.695
Accel cmd lpf cutoff	0.0	CL at zero elevator	0.4910





The figure above depicts the calculated CL from the feed forward and feedback loop equations. Both calculated CL commands settled around a CL of 0.8 which was near the CL cruise setting of 0.8. Recall that the CL cruise value is used to calculate the airspeed command; thus, indirectly affecting z acceleration control.



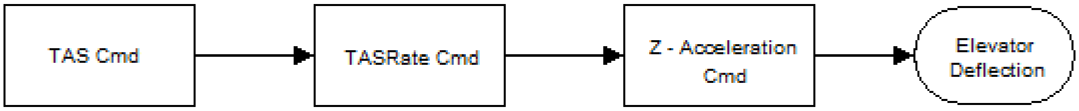
The figure above depicts the calculated elevator deflections versus the actual elevator deflections in the plot on the left and the z error that occurred at the beginning of the simulation where the simulation began with the aircraft suspended in the air in the plot on the right. The largest discrepancy was at the beginning of the simulation. The manual calculations were integrating the z error from the very beginning of the simulation; however, the autopilot did not begin integrating at the beginning. The autopilot's integrator began sometime after the initial negative error. Additionally it is likely that the autopilot integrates the acceleration error differently than how the manual calculations integrated the error. The manual integration was restricted to the controller telemetry rate of 25 Hz. It was concluded that the difference in the estimated and actual elevator deflection commands were due to the integrating error.

3.1.2. Airspeed Control Loop

The gain definitions alone were insufficient in explaining the full functionality of Airspeed Control. Additionally the supplemental description given by the "Tuning piccolo control laws 2.0.x" was inaccurate; therefore, I had to conduct experiments to uncover more detail on Airspeed Control.

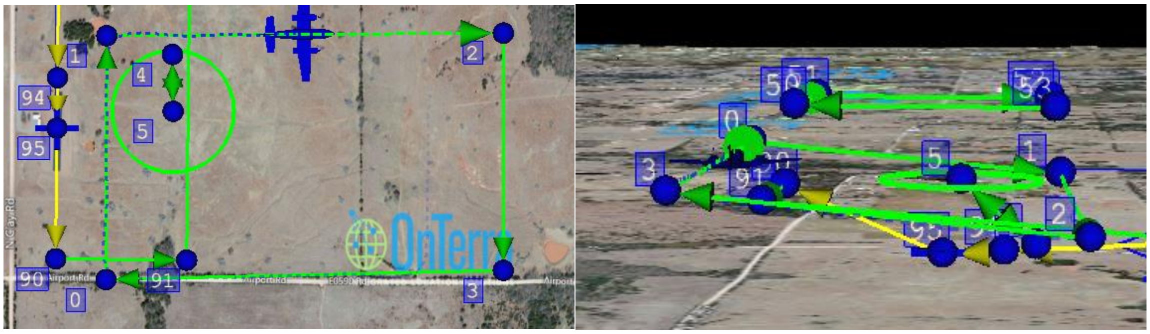
According to Cloud Cap documentation the airspeed control law computes the desired vertical acceleration in order to track the target airspeed (Tuning piccolo control laws 2.0.x 12).

Vertical acceleration is referring to the Z-Acceleration command which commands elevator deflections.



The documentation asserts that, in Airspeed Control, Altitude Control is only able to be controlled by the throttle; however, Airspeed Control has changed (Tuning piccolo control laws 2.0.x 14). If the Z-Acceleration command comes from Airspeed Control only then both the Altitude and VRate command loops should be cut out of the process.

I created an experiment consisting of a series of climbs in a software in the loop simulation to test if either of the Altitude Control Loop gains impact the Z-Acceleration command in Airspeed Control. The flight plan is shown in the figure below (Waypoints 0 – 4).



Note that since the climb was in the flight plan the altitude command during the climb acted as a ramp input for altitude. Initially I set Fast IAS error Threshold to -1 to force the autopilot into Airspeed Control (Lon Mode 1) for the duration of the simulation.

I analyzed the vertical rate and elevator commands to look for indications of whether or not the vertical rate command impacts elevator deflections in Airspeed Control.

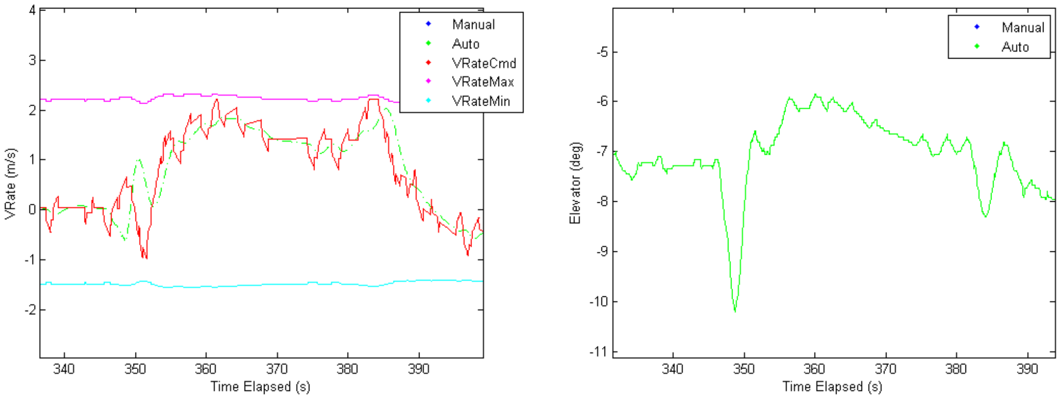
Climb 1: For the first climb I set the Altitude and Airspeed Control Loops gains to their default values to set a benchmark for comparison with the other climbs.

Gains:

Alt err to Alt Rate = 0.20 Alt rate err to accel = 0.75

TAS err to TAS Rate = 0.15 TAS Rate err to Accel cmd = 1.5 TAS Rate err to Accel der cmd = 1.0

Results:



The figure above shows that the VRate command had quick and sharp oscillations. The VRate command oscillations are not characteristic of the VRate command in Altitude Control. That seemed to indicate that the Airspeed Control Loop could have an influence on the VRate command; however, the prime focus in this climb was to set a benchmark to determine if the VRate command impacts Elevator deflections. The figure also shows the Elevator deflections, which were somewhat oscillatory.

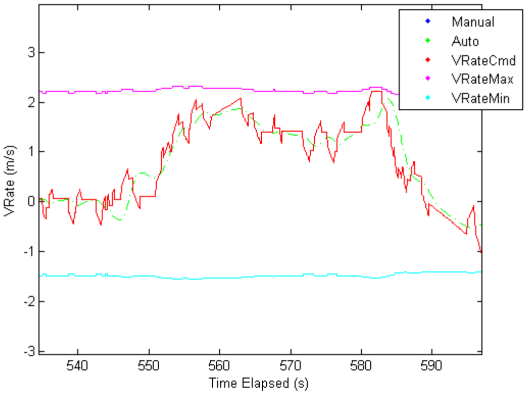
Climb 2: For the second climb I changed the Altitude Control Loop gain "Alt err to Alt Rate" to a high value so that it would induce elevator oscillations if it could impact the Elevator command.

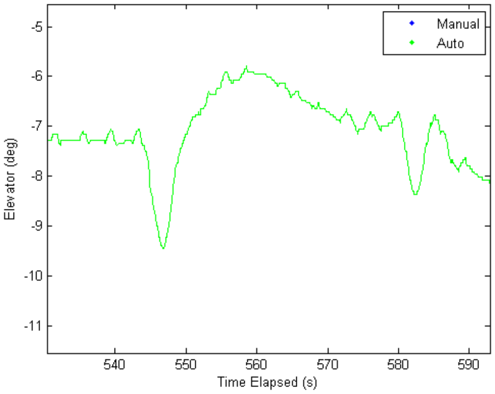
Gains:

Alt err to Alt Rate = 10 Alt rate err to accel = 0.75

TAS err to TAS Rate = 0.15 TAS Rate err to Accel cmd = 1.5 TAS Rate err to Accel der cmd = 1.0

Results:





The figure above shows that the VRate command did not change in behavior compared to Climb 1. Similarly the Elevator deflection commands remained the same as well. These results indicated that the outer loop gain, “Alt err to Alt Rate”, is not utilized in Airspeed Control. There was still no indication of the VRate command having a direct impact on Elevator deflections.

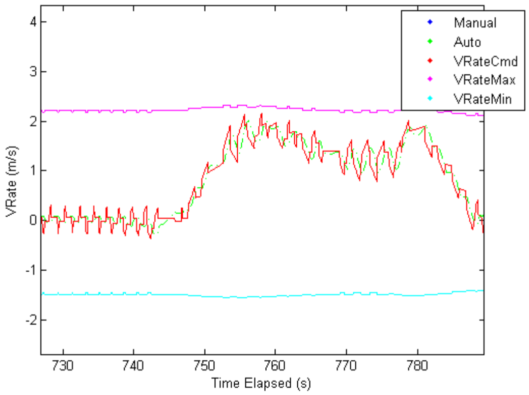
Climb 3: For the third climb I changed the Altitude Control Loop gain “Alt rate err to accel” to a high value so that it would induce elevator oscillations if it could impact the Elevator command. Note that the gain altered is the inner loop feedback gain that impacts the VRate command based on VRate error.

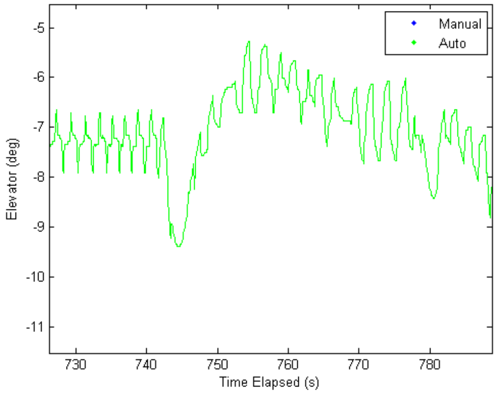
Gains:

Alt err to Alt Rate = 0.20 Alt rate err to accel = 3.0

TAS err to TAS Rate = 0.15 TAS Rate err to Accel cmd = 1.5 TAS Rate err to Accel der cmd = 1.0

Results:





The figure above shows that the VRate command oscillated heavily. In addition to VRate command oscillations the Elevator deflection commands also oscillated heavily.

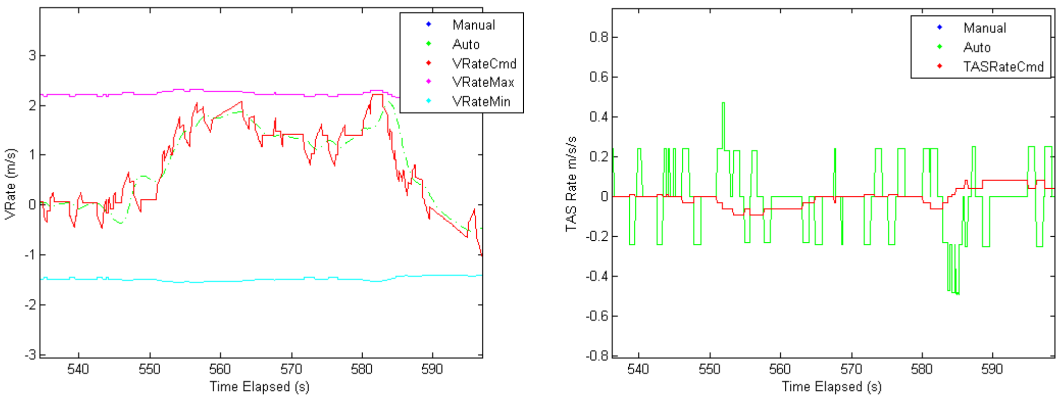
Conclusions:

1) The results indicated that the inner Altitude Control Loop impacts the Elevator deflections in Airspeed Control via the VRate command.

2) The results also indicated that the VRate command is not affected by the Altitude Control outer loop gain “Alt err to Alt Rate”.

3) Conclusion 2 combined with the uncharacteristic oscillations in the VRate command imply that the VRate command could be impacted by one or both of the Airspeed Control Loops .

The Airspeed Control Loops command TASRate based on TAS errors and Z – Acceleration based on TAS Rate errors. In Altitude Control the VRate command is based on the Altitude error and commands Z – Acceleration based on VRate errors. I combined those facts with the conclusions above and further analyzed the results from the previous 3 climbs. I looked for any indication of correlations between the VRate command and the TASRate, since both are used to command Z – Acceleration in their respective control loops. I noticed that the sharp quick oscillations observed in the VRate command are similar to the TAS Rate command, shown in the figure below.



In light of the results I decided to conduct another software in the loop simulation to test if the VRate command is impacted by the Altitude error, and or the TASRate error.

The idea was to have the autopilot perform two identical climbs. The first climb would be conducted in Altitude Control and the second climb would be conducted in Airspeed Control. In both climbs I analyzed the VRate command, TASRate command, and TASRate error. The purpose of the first climb in Altitude Control was to establish a benchmark to compare the VRate command behavior in Altitude Control against the VRate command behavior in Airspeed Control.

The flight pattern was exactly the same as the previous climbs in this section. Once again the purpose of the climb was to create a change in VRate command so the effects could be properly analyzed. The Climb and Descent Max Fractions were set low so as to avoid putting the aircraft in a scenario where the autopilot would switch into one of the Lon Modes during the

Altitude Control portion of the simulation. To analyze the results I plotted VRate, VRate command, TASRate command * 10, and TASRate errors. I multiplied the TASRate command by 10 so it could actually be seen in the same plot as the others.

Climb 1:

The first climb was done in Altitude Control to establish the VRate command behavior in Altitude Control. I set the Fast IAS error Threshold to a large value to avoid the autopilot switching to Airspeed Control. The remaining Altitude and Airspeed Control Loop gains remained at their default values.

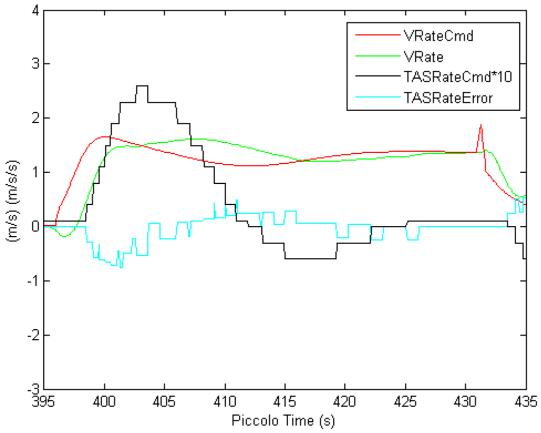
Gains:

Fast IAS error Threshold = 10 m/s Alt err to Alt Rate = 0.20 Alt Rate err to Accel = 0.75

TAS err to TAS Rate = 0.15 TAS Rate err to Accel Cmd = 1.5 TAS Rate err to Accel Der Cmd = 1.0

Climb Max Fraction = 0.10 Descent Max Fraction = 0.15

Results:



The figure above shows that the VRate command was very smooth.

Climb 2:

In the second climb I set the Fast IAS error Threshold to -1 to force the autopilot into Airspeed Control. The rest of the gains remained the same from Climb 1.

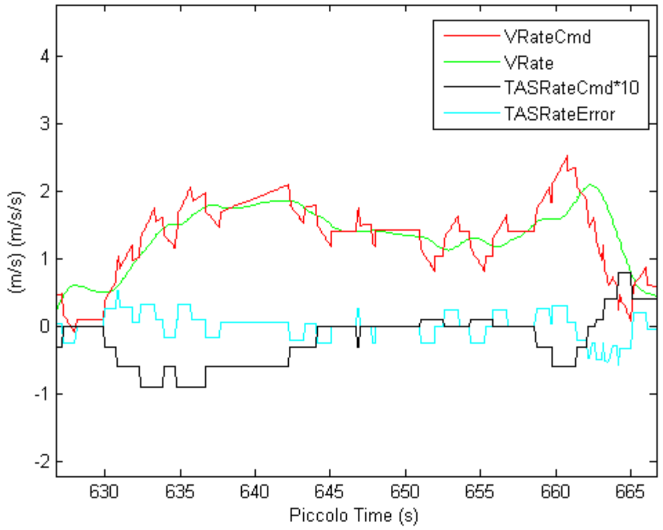
Gains:

Fast IAS error Threshold = -1.0 Alt err to Alt Rate = 0.20 Alt Rate err to Accel = 0.75

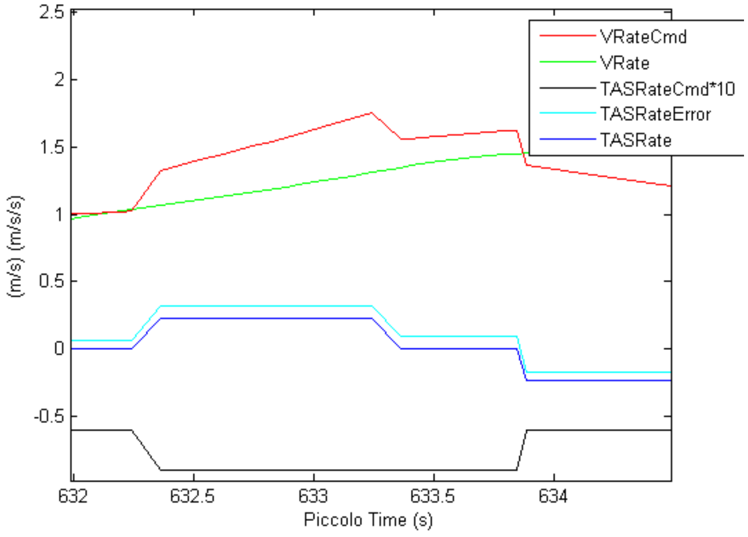
TAS err to TAS Rate = 0.15 TAS Rate err to Accel Cmd = 1.5 TAS Rate err to Accel Der Cmd = 1.0

Climb Max Fraction = 0.10 Descent Max Fraction = 0.15

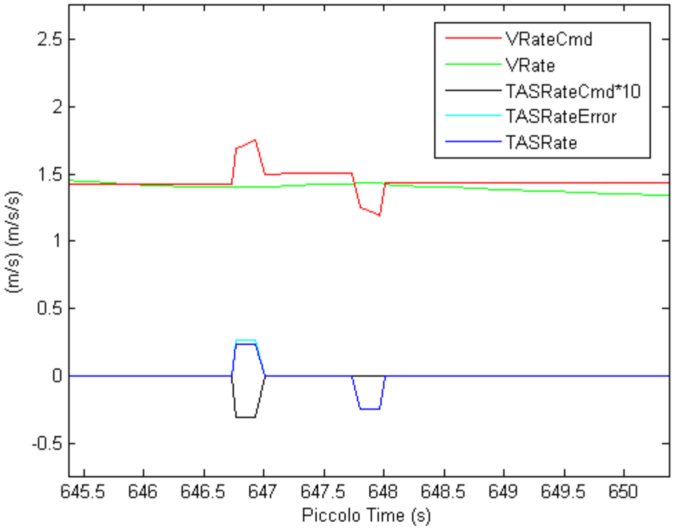
Results:



The figure above shows that the VRate commands oscillated in sync with the TASRate error.

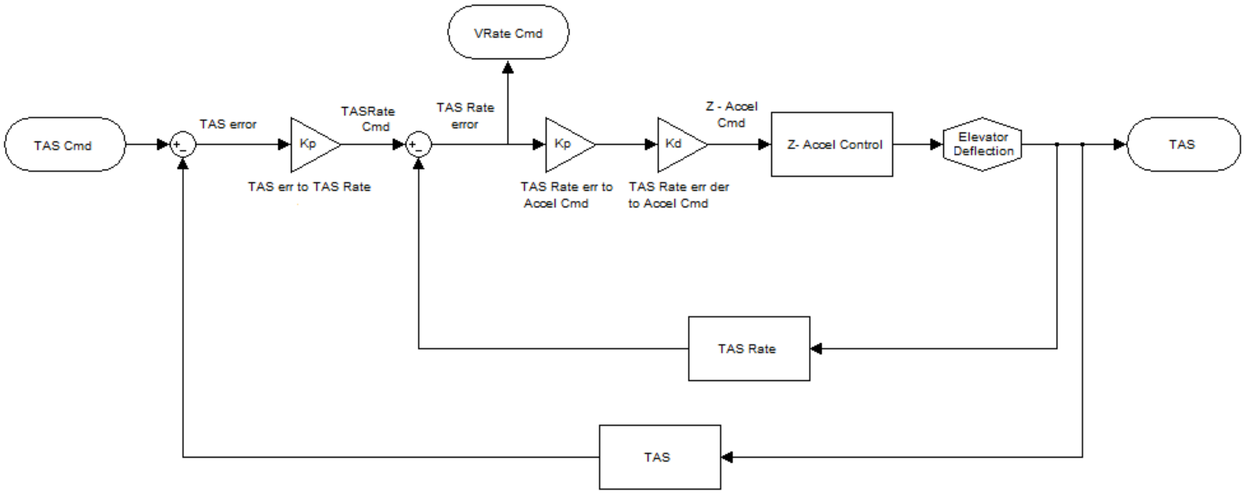


The figure above shows the TAS Rate command decreased at about 632.25 seconds. The TASRate error increased because the actual TAS Rate increased. Initially the VRate command increased proportionally with the TASRate error and inversely proportional to the TASRate command, but as the TAS Rate command and TASRate error remained constant the VRate command climbed. If the TASRate error does command VRate this is the behavior that would be expected because a negative TASRate command would create a positive VRate command in order to decrease the airspeed of the aircraft, and if the actual TASRate remained positive (causing an increase in airspeed as it did) the VRate command would continuously increase in order to decrease the TASRate.



The figure above shows that when the TASRate command and TASRate error were both 0 the VRate command remained constant. This would be expected if the VRate command was coming from the TASRate error because if there was no airspeed error then the VRate command would be 0 to hold airspeed. The increase in VRate happened when the TASRate command decreased, creating a TASRate error. The decrease in VRate occurred when the TASRate error decreased. All of these observations point to the VRate command originating from the TASRate error.

At this point, from the results of all the tests, I assumed that the TASRate error commanded VRate through a separate loop other than the Airspeed Control inner loop as depicted below.



The schematic raised another question. Does the VRate command actually exist inside the Airspeed Control inner loop either before or after the inner loop gains? I devised an experiment to test this question. The idea was if I set the Airspeed Control inner loop gains to zero and the VRate command is inside the Airspeed Control loop, after the inner loop control gains then the VRate command should not change from what it was when the gains were zeroed. I ran a hardware in the loop simulation to test the hypothesis.

Test 1:

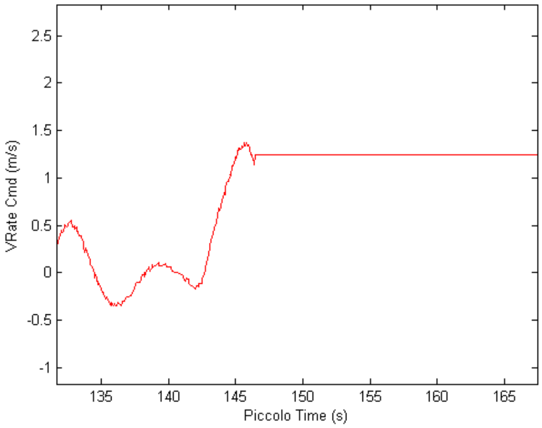
I began the simulation in airspeed control, and zeroed the Airspeed Control inner loop gains after the simulation had already started to observe the VRate command response.

Gains:

Fast IAS error Threshold = -1.0 Alt err to Alt Rate = 0.20 Alt Rate err to Accel = 0.75

TAS err to TAS Rate = 0.15 TAS Rate err to Accel Cmd = 0.0 TAS Rate err to Accel Der Cmd = 0.0

Results:



The figure above shows that when the Airspeed inner loop control gains were set to zero (at 146.5 seconds) the VRate command became constant; therefore the VRate command is located inside the Airspeed Control inner loop. I was not sure if the VRate loop was located before or after the inner loop Airspeed Control gains, so I did another quick test. I set the autopilot in Airspeed Mode, and set the VRate gain “Alt err to Alt Rate” to zero. Then I set the inner loop Airspeed Control gains to very large values (10) in order to induce Elevator oscillations. There were no oscillations; thus, I was able to conclude that the VRate loop lies after the inner loop Airspeed Control gains.

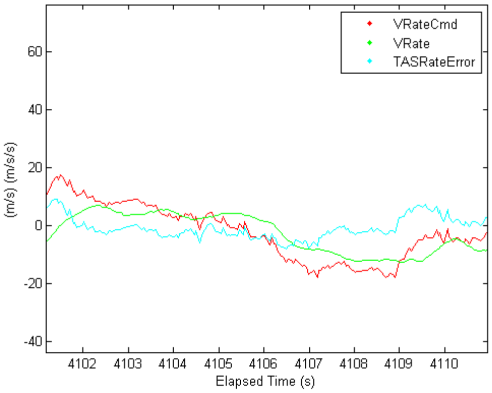
Conclusions:

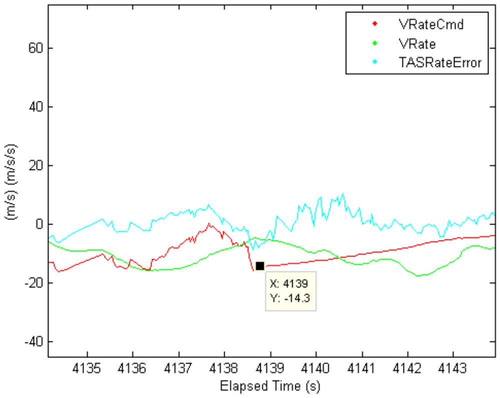
1) In Airspeed Control the VRate command exists inside the Airspeed Control inner loop where it receives input from the TAS Rate error through the Airspeed inner loop control gains to command VRate.

2) The Airspeed Control Z – Acceleration command travels through the VRate loop before being sent to the Z – Acceleration Control Loop.

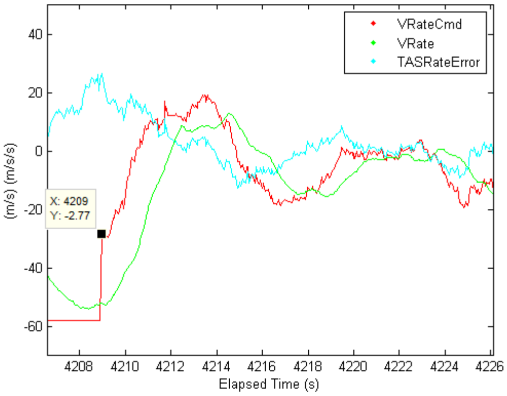
The implication that the VRate command is commanded from the TASRate error seemed strange since Airspeed Control already uses the Airspeed Control inner loop to command Z –

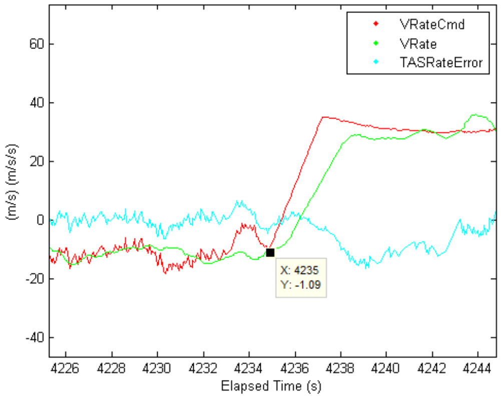
Acceleration. I decided to analyze Airspeed Control even further from actual flight data in case the results were a result of an SiL glitch. In Flight 5 of Noctua – B1 the autopilot went into Airspeed Control (Fast Airspeed Mode actually) multiple times. Below are figures of VRate, and TASRate error at different times in two different descents.





The figure above shows the VRate command during Airspeed Control. Again the VRate command oscillated in sync with the TASRate error. The marker in the plot on the right denotes when the autopilot returned to Altitude Control. Exactly at the moment the autopilot made the switch the VRate command immediately became smooth and did not resemble the TASRate error.





The figure above is from another descent in the same flight. The plot on the left shows the VRate command just before entering Airspeed Control (4209s). As the autopilot entered Airspeed Control the VRate command began oscillating in sync with the TASRate error. The plot on the right shows that the VRate command oscillations stop again right after the switch back to Altitude Control (4235s). The real life flight data results are the same as the simulated results.

4. Throttle Control Experiments

4.1.1. Energy Control

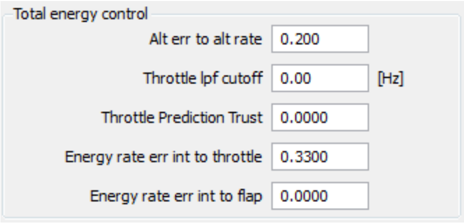
The gain definitions alone were insufficient in explaining the full functionality of Energy Control. The gain definitions did not explain the dependencies, if any, of Energy Control on Altitude and Airspeed Control. Additionally the supplemental description given by the “Tuning piccolo control laws 2.0.x” was for an older version (2.1.2.h); thus, inaccurate for the current version (2.1.4.g). A lot of changes to Energy Control occurred in the update from 2.1.2.h to 2.1.4.a. An outer Energy Control Loop was added to command Energy Rate as a function of Energy Error. In order to fully understand Energy Control I had to start with the explanation of the old system, in “Tuning piccolo control laws 2.0.x”, and try to determine exactly what and how Energy Control changed.

I began with the description of Energy Control from “Tuning piccolo control laws 2.0.x”. “The throttle control law adjusts the throttle to give the correct rate of energy flow into or out of the system. The rate at which energy should go into the system is given by the desired vertical rate and the desired rate of true airspeed according to: $dE\ dt = m(V\ (dV\ dt\ )+ g(dh\ dt\ ))$. The desired vertical rate and rate of true airspeed come from the altitude and airspeed control loops respectively.” (Tuning piccolo control laws 2.0.x 10).

Generally speaking the description describes Energy Control as commanding throttle for the purpose of either adding or removing energy from the system (aircraft). More specifically the description describes Energy Control as a single loop control system where the Energy Rate command comes directly from the Altitude and Airspeed control loops and is converted into a Throttle command via the VRate, TASRate, and TAS commands.

$$Energy\ Rate_{cmd} = \frac{dE}{dt}_{cmd} = m(TAS_{cmd})(TASRate_{cmd}) + mg(VRate_{cmd})$$

The first 10 flights of the Nexstar used version 2.1.1.e which used the very same Energy Control system as described in "Tuning piccolo control laws 2.0.x.pdf." The figure below comes from a snapshot of the Controller Configuration window of a replay of Nexstar Flight 6.



Total energy control

Alt err to alt rate 0.200

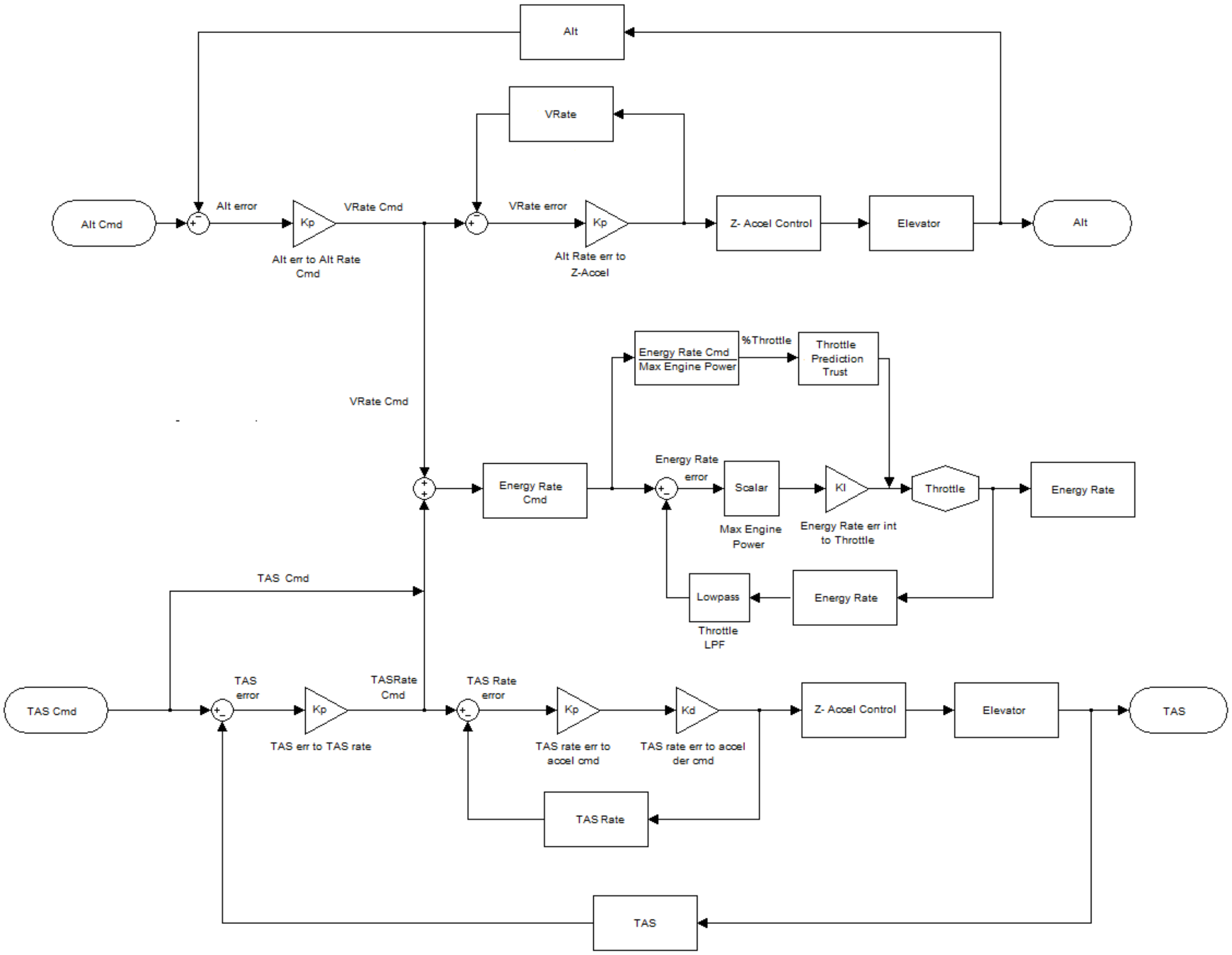
Throttle lpf cutoff 0.00 [Hz]

Throttle Prediction Trust 0.0000

Energy rate err int to throttle 0.3300

Energy rate err int to flap 0.0000

The Altitude Control outer loop gain "Alt err to Alt Rate" was actually listed among the Energy Control gains. "Energy rate err int to flap" seemed to be concerned with using flaps as a means of losing energy. Since the gain does not exists as a gain in Energy Control in 2.1.4.g it is safe to assume that the current energy control laws no longer use flaps to help shed energy. Throttle Prediction Trust and the Throttle lpf cutoff are still used in the same manner as the old control law as their definitions describe the same function. These explanations were used to put together the schematic shown below:

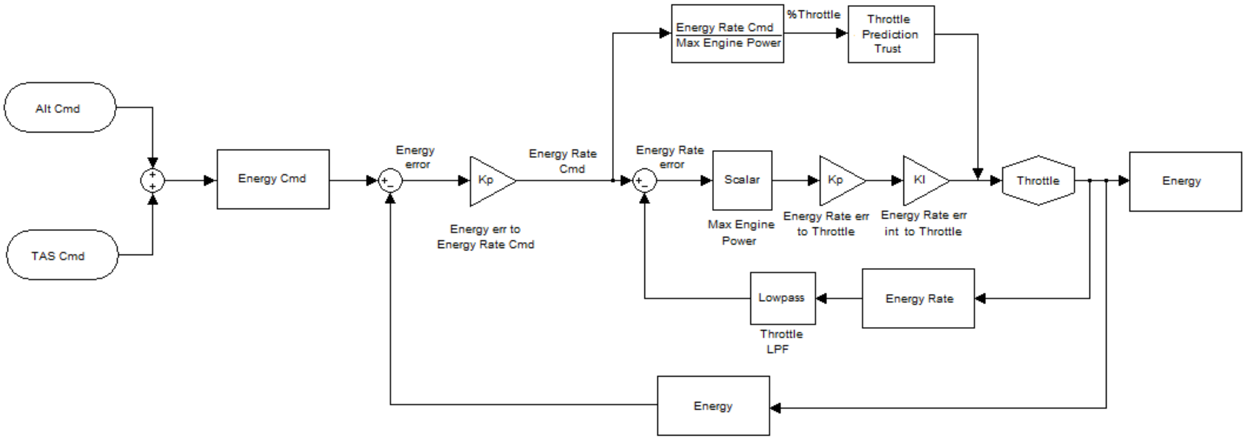


$$where\ Energy\ Rate_{cmd} = \frac{dE}{dt}_{cmd} = m(TAS_{cmd})(TASRate_{cmd}) + mg(VRate_{cmd})$$

From the schematic it is easy to see how the old Energy Control loop was dependent on the Altitude and Airspeed control loops.

According to the Energy Control Loop gain definitions the new outer loop now produces Energy Rate commands. I read through the Cloud Cap release notes ("Release Notes.txt") for version 2.1.4.g to search for more clarity on the changes made to Energy Control. In the section for updating from version 2.1.2.h to version 2.1.4a under "Fixed wing generation 2 autopilot new features" there is a description on the changes to Energy Control. The description of Feature #4 states "Redesigned the throttle control law for better responsiveness. The new law does not have dependencies on the altitude and airspeed control laws. Also updated the default throttle control gains for more performance."

If Energy Control is no longer dependent on Altitude and Airspeed Control, as the excerpt from the Release Notes states, then the VRate, TAS, and TASRate commands should no longer be directly affecting the Energy Rate command. If that is true then the Energy Rate command should come solely from the new outer loop which takes input from the TAS and Altitude commands.



$$where\ Energy_{cmd}(J) = \frac{1}{2}m(TAS_{cmd})^2 + mg(Alt_{cmd})$$

$$Energy\ (J) = \frac{1}{2}m(TAS)^2 + mg(Alt)$$

$$Energy\ Rate\ \left({}^{J}/_{s}\right) = m(TAS)(TASRate) + mg(VRate)$$

The figure above is a schematic of how it seems Energy Control should be operating in 2.1.4.g. I called Cloud Cap to ask them about how Energy Control operates. I asked if Energy Control was no longer dependent on Altitude and Airspeed Control and I was told "No, well not really." I decided I should conduct some experiments of my own to determine if VRate and/or TASRate command an Energy Rate in addition to the outer loop.

The idea was to fly the autopilot in a hardware in the loop simulation with a climb waypoint, so as to induce large Energy and Energy Rate errors, and manipulate the inner and outer loop control gains and observe the Throttle response. The first set of tests were conducted in Altitude Control to avoid any complexities that might arise from switching to Airspeed Control.

I began the simulation with default gains so that the autopilot would have a throttle trim (via the integral gain). After the aircraft was flying steady and level on a constant altitude leg of the flight plan I set all of the Energy Control gains as close to zero as possible. I wanted to make sure that with all the gains disabled the autopilot would only command the Throttle trim that had been set by the integrator, regardless of any positive or negative Energy errors. I also wanted to assess what the Throttle trim value was.

Gains:

Energy err to Energy Rate = 0.0001   Energy Rate err to Throttle = 0.0   Fast IAS error Threshold=10m/s

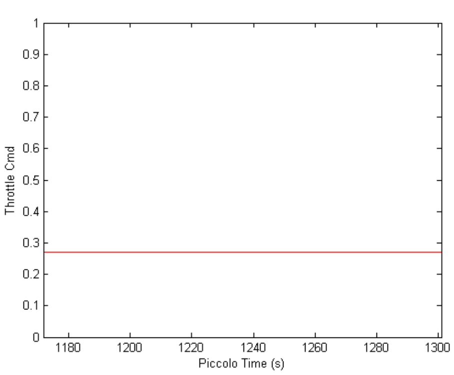
Energy Rate err int to Throttle = 0.00001   TPT = 0.0   Throttle LPF Cutoff = 0 Hz

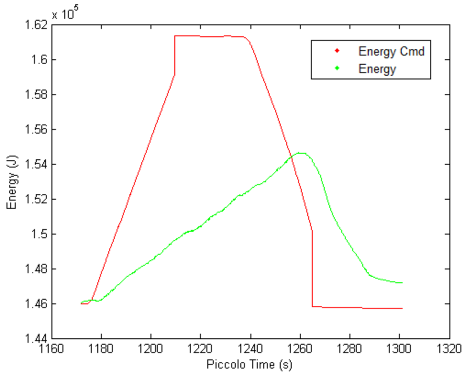
Descent Max Fraction = 0.15   Climb Max Fraction = 0.10   Max Engine Power = 3250 W.

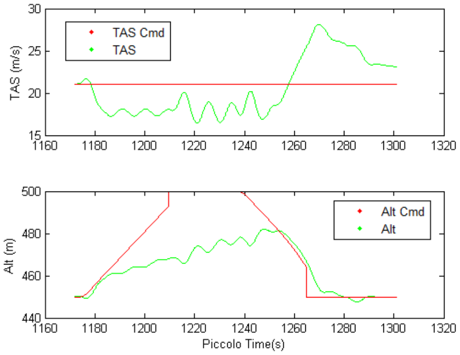
Fast IAS error Threshold = 10 m/s   Max Engine Power = 3250 W

I set the Descent and Climb Max Fractions low to avoid stalling the aircraft and descending too quickly. I also set the Fast IAS error Threshold high to avoid entering Airspeed Control.

Results:







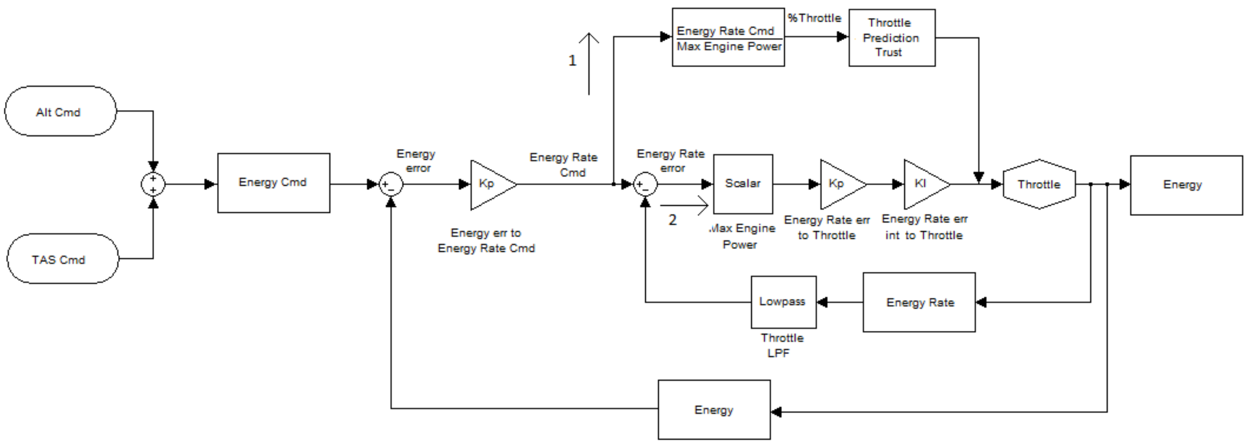
After I set the Energy Control gains the autopilot no longer changed the throttle command and the throttle command remained at the trim setting that Energy Control had set which was about 27% Throttle. The Figures show that the Throttle did not change even with negative and positive Energy errors.

Test 1:

The first test was to confirm that there are only two paths for the Energy Rate command to affect throttle:

1) Through the Throttle Prediction Trust feed forward loop

2) Through the Energy Rate feedback loop



Before I began testing the individual paths I wanted to make sure that no other paths existed so I only turned on the outer loop proportional gain. The idea was if no other paths existed then the throttle command should never change even with the outer loop gain enabled and commanding an Energy Rate.

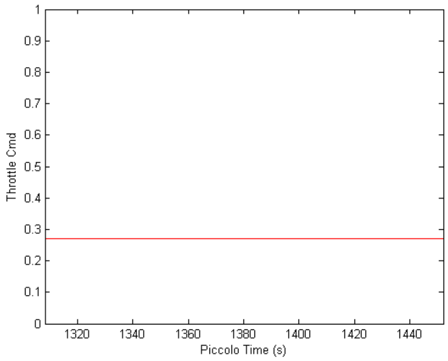
Gains:

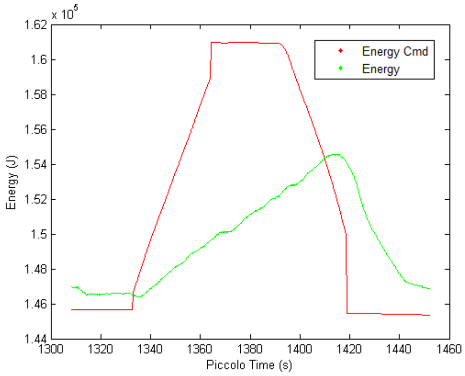
Energy err to Energy Rate = 1.0 Energy Rate err to Throttle = 0.0

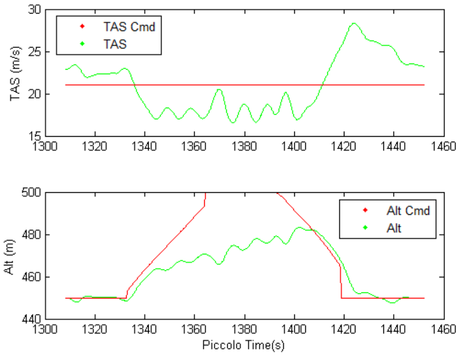
Energy Rate err int to Throttle = 0.00001 TPT = 0.0 Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s Max Engine Power = 3250 W

Results:







The Figures show that there was no change in Throttle even though there were Energy errors; thus confirming that the two paths outlined in the schematic above are the only paths for the Energy Rate command to reach the Throttle command.

To test path 1 I kept the outer loop gain enabled, so as to ensure that there was an Energy Rate command, and enabled the Throttle Prediction Trust. If path 1 does command Throttle then the Throttle command should change as a result of the Energy Rate command changing. The Energy Rate command should change as a result of the changes in Energy Error. Note that because of the Throttle trim the Throttle command would not be able to decrease lower than 27% in the event of a large positive Energy error.

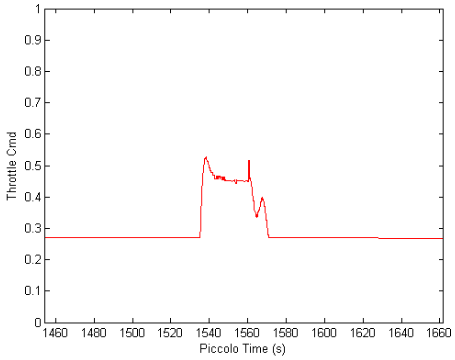
Gains:

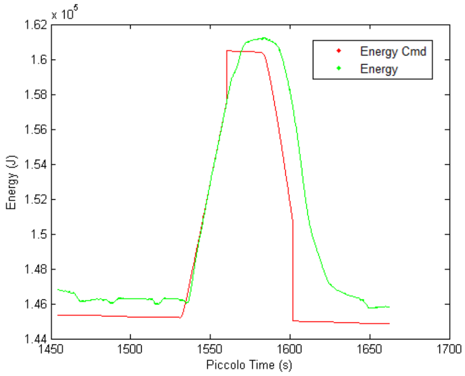
Energy err to Energy Rate = 1.0 Energy Rate err to Throttle = 0.0

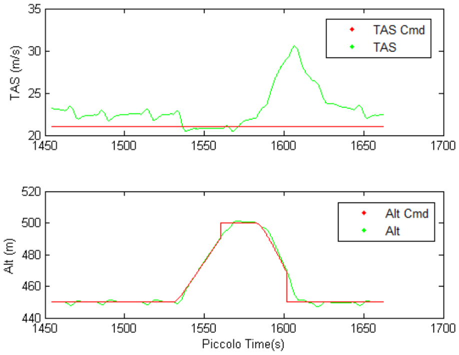
Energy Rate err int to Throttle = 0.00001 TPT = 1.0 Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s Max Engine Power = 3250 W

Results:







The figures show that the Throttle command did indeed increase during the climb to reduce Energy error; thus confirming path 1 as a valid route for commanding Throttle. The Throttle trim at 27% did inhibit the Throttle from further decreasing to shed the positive Energy error.

Path 2 was tested in a similar manner to path 1. The idea was that enabling the inner loop gains would cause the Throttle command to change as a result of the Energy Rate error. The Throttle Prediction Trust was disabled to eliminate path 1 from changing the Throttle command.

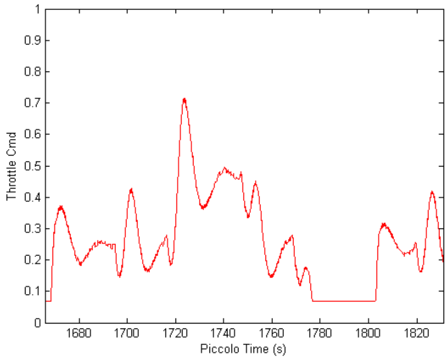
Gains:

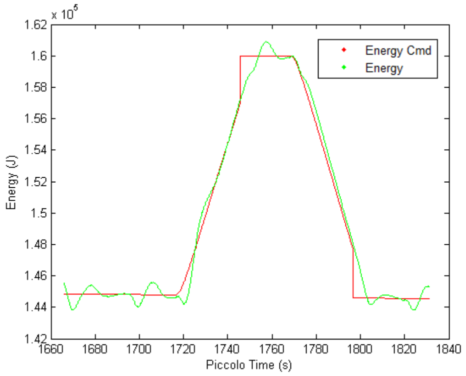
Energy err to Energy Rate = 1.0   Energy Rate err to Throttle = 0.60

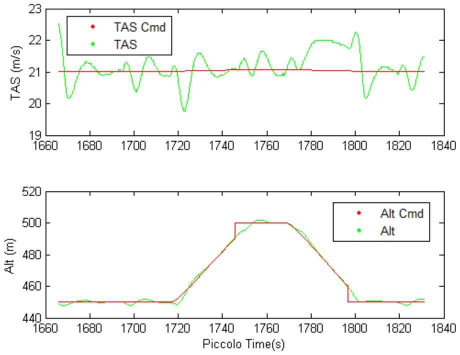
Energy Rate err int to Throttle = 0.40   TPT = 0.0   Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s   Max Engine Power = 3250 W

Results:







The Figures show that the Throttle command increased and decreased, even before the climb, to reduce Energy error; thus confirming path 2 as a valid route for commanding Throttle. Since the integrator was enabled (integral gain) the Throttle trim was altered throughout the test and Throttle was able to command as low as Throttle Min (7%).

Test 2:

The purpose of the second test was to determine if the TASRate and VRate commands are influencing the Energy Rate command (like in the previous version) in any capacity.

I conducted the second test right after the first, in the same simulation. I set the outer loop proportional gain to as near as zero as possible so that the outer loop would no longer be commanding an Energy Rate. Then I disabled the inner loop gains (inner loop integral gain was set as low as possible) so that Energy Rate errors could not influence Throttle commands. I enabled the Throttle Prediction Trust so that any Energy Rate commands that might exist would have a path to alter the Throttle commands.

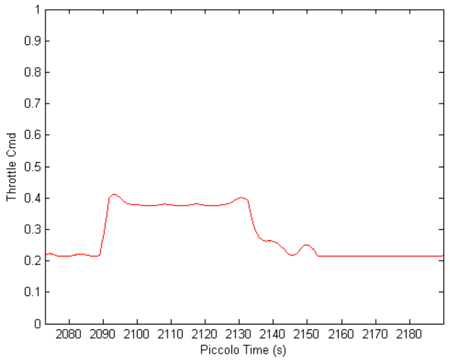
Gains:

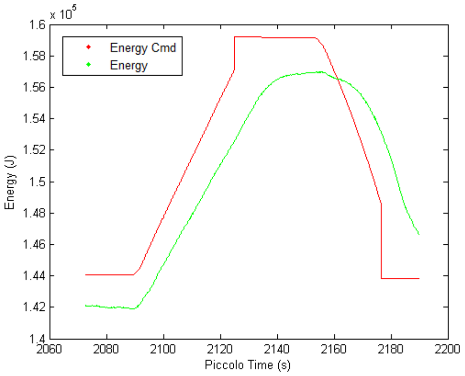
Energy err to Energy Rate = 0.0001 Energy Rate err to Throttle = 0.0

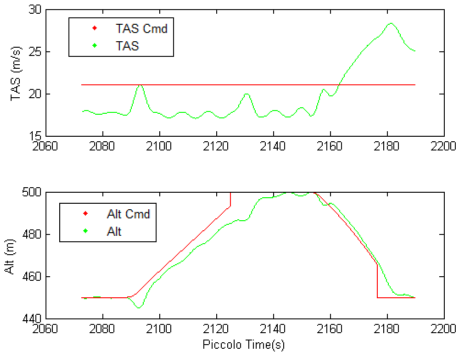
Energy Rate err int to Throttle = 0.00001 TPT = 1.0 Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s Max Engine Power = 3250 W

Results:







The figure above shows that the Throttle trim had been reset to about 21%, but more importantly the figure shows Throttle command changed throughout the test. The change in Throttle was as high as a 20% increase from Throttle trim; thus, there is another source of input commanding Energy Rate.

In light of the new findings I devised another test in a hardware in the loop simulation to determine if the second source comes from the VRate and TASRate commands, similar to the old version.

Test 3:

I setup the simulation to start with only the Throttle Prediction Trust on so that there would not be any throttle trim; thus, the magnitude of the Energy Rate command would be equal to the commanded Throttle. The flight pattern was similar to the previous tests. It contained a climb waypoint to induce large Energy errors. I also kept the Climb Max Fraction low so that the autopilot would not go into Airspeed Control when the autopilot attempted the climbs.

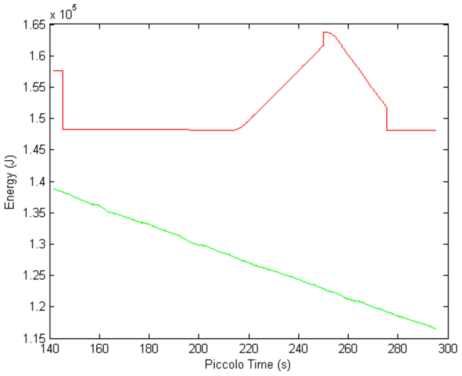
Gains:

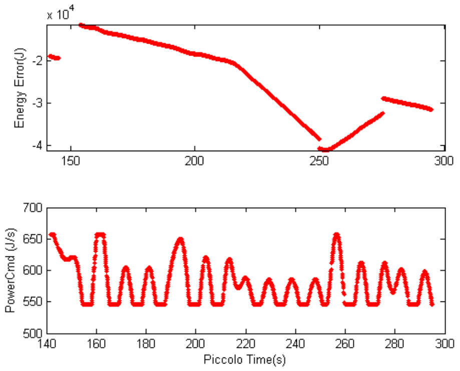
Energy err to Energy Rate = 0.0001   Energy Rate err to Throttle = 0.0

Energy Rate err int to Throttle = 0.00001   TPT = 1.0   Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s   Max Engine Power = 3250 W

Results:

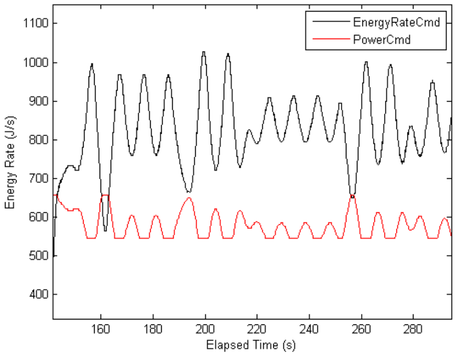




The figure above shows Energy error vs. the Power command where:

PowerCmd = %Throttle * Max Engine Power (J/s)

Both figures show that the Throttle command had no relation to the Energy error; thus, ensuring that the Energy Rate command that was commanding Power was not coming from the Energy Control outer loop.



The figure above represents the Energy Rate command from the equation:

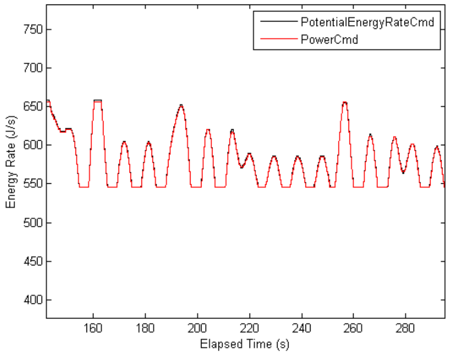
$$Energy\ Rate_{cmd} = \frac{dE}{dt}_{cmd} = m(TAS_{cmd})(TASRate_{cmd}) + mg(VRate_{cmd})$$

Equation 93 Proposed Energy Rate Equation

The figure shows that the Power command did not reflect the Energy Rate command based off of Equation 93. In fact the Energy Rate command was nearly the inverse of the Power command.

In light of the results and the fact that the test occurred in Altitude Control I decided to analyze the Power command versus the VRate command portion of the Energy Rate equation, or the Potential Energy Rate command.

$$Potential\ Energy\ Rate_{cmd} = mg\frac{dh}{dt}_{cmd} = mg(VRate_{cmd})$$



The figure above shows that the Power command was almost an exact match to the Potential Energy Rate command. Only small, negligible, differences occurred at each peak (2.5 J/s at the most). These results indicate that the extra Energy Rate command is coming directly from the VRate command.

<u>Test 4:</u>

In order to get perspective on the impact the VRate command actually has on the Throttle command I ran another hardware in the loop simulation. In the simulation I enabled only the outer loop Energy Control gain and the Throttle Prediction Trust so that I could subtract the Potential Energy Rate command from the Power command; thus calculating how much Energy Rate the outer loop was commanding.

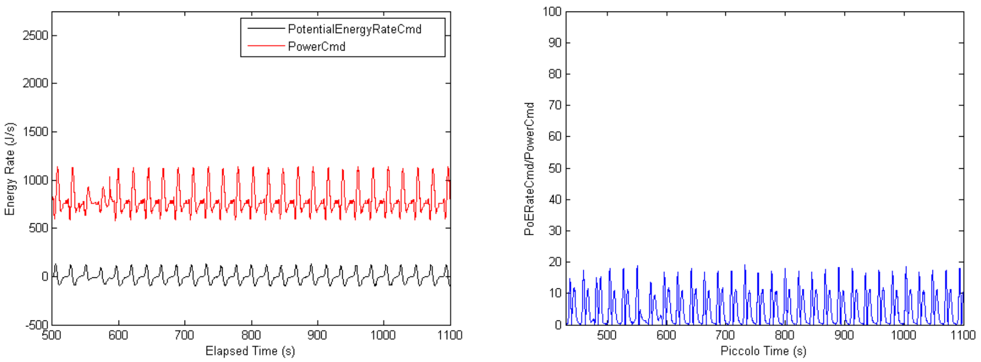
Gains:

Energy err to Energy Rate = 1.0   Energy Rate err to Throttle = 0.0   Max Engine Power = 3250 W

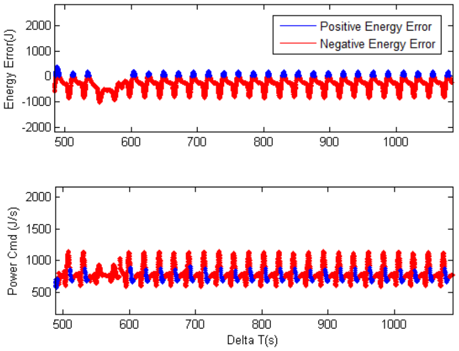
Energy Rate err int to Throttle = 0.00001   TPT = 1.0   Throttle LPF Cutoff = 0 Hz

Fast IAS error Threshold = 10 m/s   Max Engine Power = 3250 W

Results:



The figure above shows that the Power command was much higher than the Potential Energy Rate command. The figure also shows the percentage that the Potential Energy Rate command was of the overall Power command. On average throughout the flight the Potential Energy Rate command made up 5% of the Power command. It is important to note that in the simulation the outer loop energy gain, Energy err to Energy Rate command, was equal to 1.0. At a lower outer loop gain value the Potential Energy Rate command would have a larger impact.



The figure above further elaborates the influence of the outer loop energy control gain on commanding Energy Rate, and thus commanding Throttle, as the Power command oscillations were a near mirror image to the Energy Error. Meaning that when the Energy error was negative the Throttle command increased and vice versa.

Since Energy Control is affected by the VRate command in Altitude Control I decided I should test the response of Energy Control in Airspeed Control.

Test 5:

I setup another hardware in the loop simulation where the autopilot began with only Throttle Prediction Trust enabled, and in Altitude Control. The idea was to create a scenario where the autopilot was commanding Throttle only from the VRate command, via the Potential Energy Rate command, and through the Throttle Prediction Trust. After flying for a few seconds force the autopilot into Airspeed Control and observe how the Throttle responds.

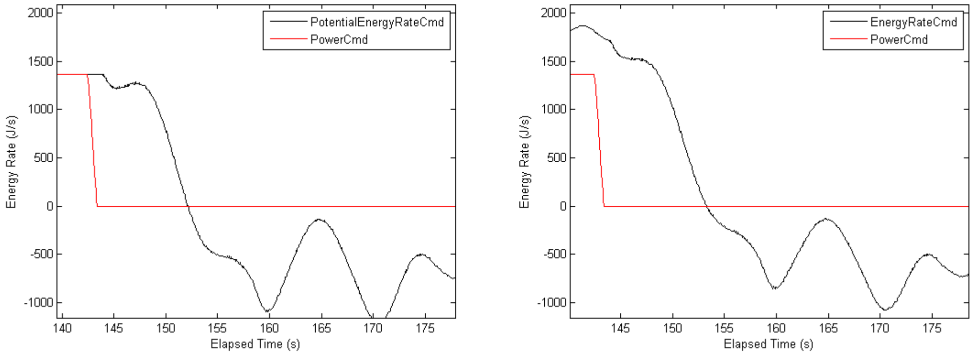
Gains:

Energy err to Energy Rate = 0.0001   Energy Rate err to Throttle = 0.0

Energy Rate err int to Throttle = 0.00001   TPT = 1.0   Throttle LPF Cutoff = 0 Hz

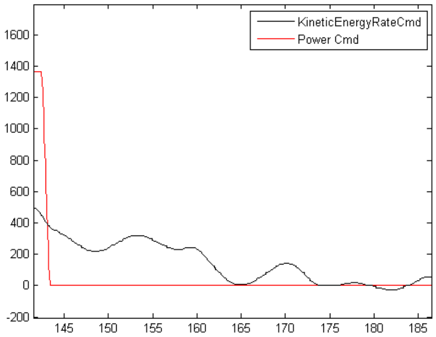
Fast IAS error Threshold = -1 m/s   Throttle Min = 0%

Results:



The figure above shows that initially, in Altitude Control, the Power command came from the Potential Energy Rate cmd. At about 143 seconds the Power Command dropped to zero. The drop in Power command coincided exactly with the switch from Altitude Control to Airspeed Control. The plot to the right shows the Energy Rate command from Equation 93 which includes the Kinetic Energy Rate term in addition to the Potential Energy Rate term. The results are similar to the plot on the left. Both figures show that the Power command was not reflecting the respective Energy Rate commands; therefore, the VRate command does not command Energy Rate in Airspeed Control.

In light of the finding that the VRate command does not affect the Throttle command in Airspeed Control I wanted to check if the TASRate command, via Kinetic Energy Rate command, was affecting Throttle on its own term.



The figure above shows the energy rate command from TASRate where:

$$Kinetic\ Energy\ Rate_{cmd} = \ m(V)\frac{dV}{dt}_{cmd} = m(TAS_{cmd})(TASRate_{cmd})$$

The figure shows no correlation between the Kinetic Energy Rate command and Power command; therefore, the Kinetic Energy Rate command does not exist in Airspeed Control.

The Kinetic Energy Rate brought up another good question about Energy Control. Does the Energy Rate feedback include the Kinetic Energy Rate term, or is it just the Potential Energy Rate term? I devised an experiment to determine if the Energy Rate feedback includes Kinetic Energy Rate. The idea was to eliminate the Energy Control outer loop, by zeroing the proportional outer loop gain “Energy err to Energy Rate command”, and observe whether or not Throttle would respond to changes in Altitude and Airspeed or just changes in Altitude. If the latter were true then the Energy Rate feedback only consists of the Potential Energy Rate.

Test 6:

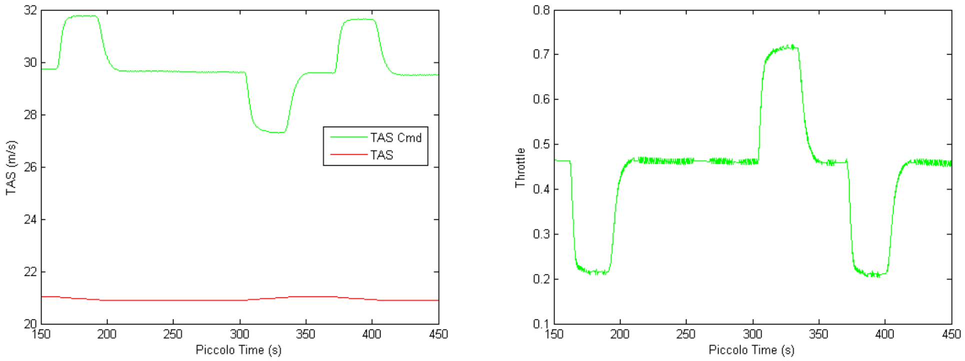
I setup another hardware in the loop simulation where the autopilot began with the Energy Control outer loop disabled. The idea was to create a scenario where the autopilot was commanding Throttle only from the VRate command, via the Potential Energy Rate command, while performing climbs and descents to observe the Throttle response.

Gains:

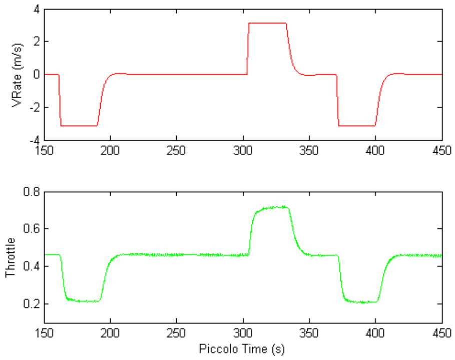
Energy err to Energy Rate = 0.0001   Energy Rate err to Throttle = 0.6

Energy Rate err int to Throttle = 0.4   TPT = 0.0   Throttle LPF Cutoff = 0 Hz

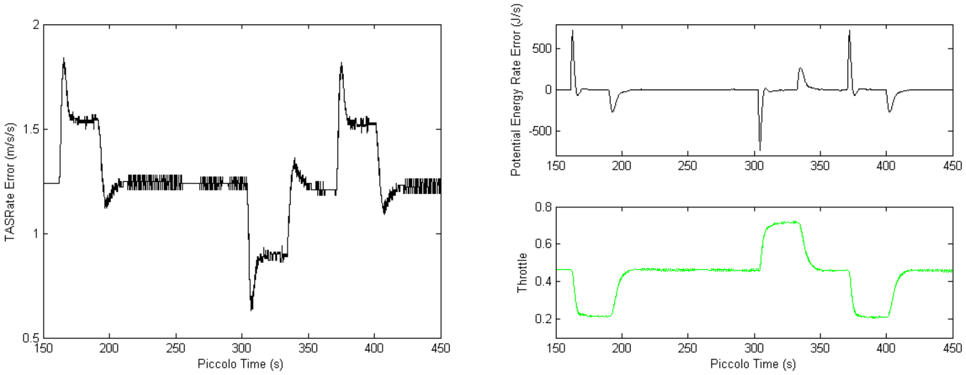
Results:



The figures show that Throttle did not respond to airspeed. The airspeed was above the commanded airspeed throughout the climb and descents. The correlation between TAS and Throttle was a result of the aircraft pitching up and down for climbs and descents. Furthermore the figure below shows that the Throttle commands mirrored the VRate commands.



In this scenario the Energy Rate command was based on the VRate command only. If the Kinetic Energy Rate was included in the Energy Rate feedback the Throttle should have also responded to the TASRate errors. Additionally the figure below shows that the TASRate error was positive throughout the climbs and descents and should have caused a Throttle decrease via the Energy Rate feedback.



The figure above also shows that when the Potential Energy Rate Error increased Throttle decreased. The Throttle stopped decreasing after the Potential Energy Rate Error became negative and Throttle held constant when the Potential Energy Rate Error zeroed. Similarly after the climb was initiated the Potential Energy Rate became negative causing Throttle to increase. All of the evidence points towards the Energy Rate feedback consisting of only the Potential Energy Rate term.

One day while I was messing around with Airspeed Control in a hardware in the loop simulation I noticed that the Throttle did not seem to respond to Airspeed errors. I had been messing with Airspeed Control gains, and the aircraft was flying in a situation where the only changes to Throttle occurred when there was an Altitude error. The constant negative Airspeed error that existed at the time was completely ignored by the Throttle. This discovery lead me to the following questions:

1) Does the Energy Control outer loop respond to errors in Altitude and TAS when the autopilot is in Altitude Control?
2) Does the Energy Control outer loop respond to errors in Altitude and TAS when the autopilot is in Airspeed Control?

Up until this point I had been operating under the assumption that the Energy Control outer loop always responds to errors in both Altitude and TAS.

I conducted a hardware in the loop simulation for each of the two questions. The idea was to manipulate control gains to force the autopilot into a position where only Altitude and then only TAS errors existed in both Altitude and Airspeed control to observe the Throttle response.

Test 7:

For Test 7 I flew the autopilot in Altitude Control to determine how Throttle would respond to Altitude and TAS errors. The idea was to set the Altitude Control outer loop gain "Alt err to Alt Rate" and the Altitude Control inner loop gain "Alt Rate to accel" to zero, at a time where the VRate command was positive, so that the VRate command would remain constant at a positive value. In that scenario I would be able to change the Altitude and Airspeed commands and observe the Throttle response without Elevator deflections trying to correct for Altitude errors.

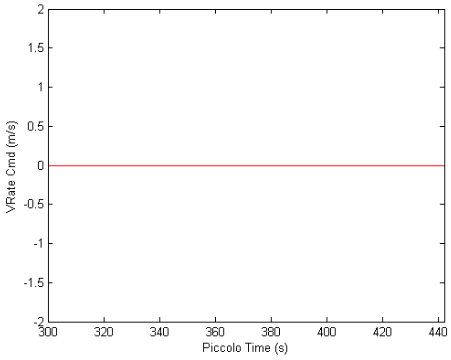
Gains:

Alt err to Alt Rate = 0   Alt Rate err to accel = 0

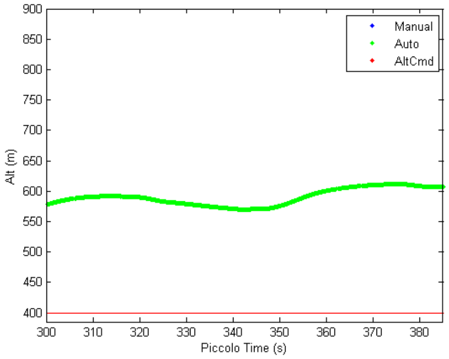
Energy err to Energy Rate = 0.35   Energy Rate err to Throttle = 0.6

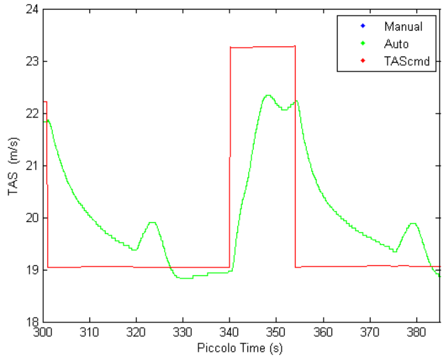
Energy Rate err int to Throttle = 0.4   TPT = 0.0   Throttle LPF Cutoff = 0 Hz

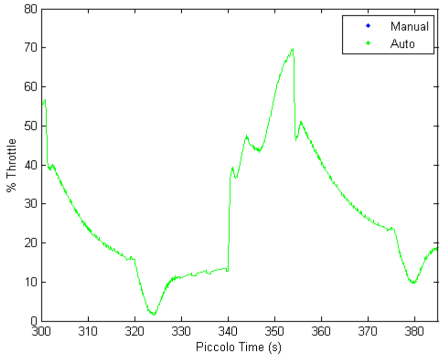
Results:



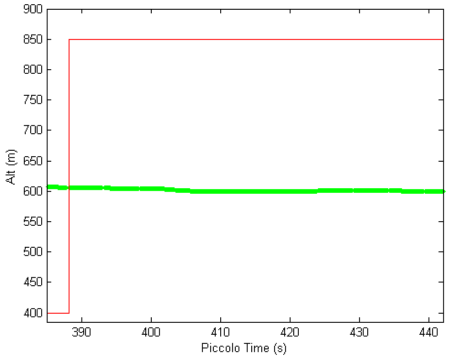
The figure above shows that the VRate command was zero throughout the time period the tests were conducted in the simulation.

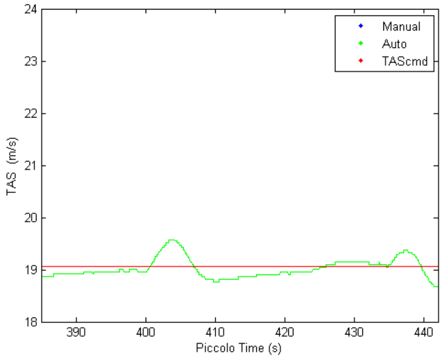


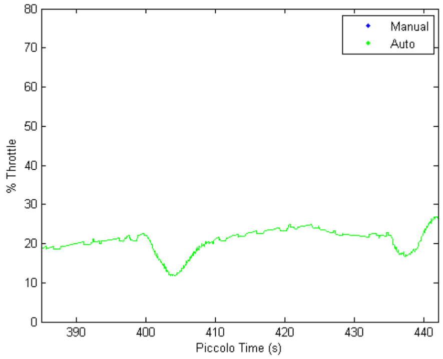




The figure above shows that from 300 seconds to 385 seconds there was a positive Altitude error. The figure also shows that at 300 seconds the TAS command decreased and the Throttle decreased as well. From 330 to 340 seconds the TAS error leveled out, and the Throttle leveled out as well in spite of the positive Altitude error. At 340 seconds the TAS command increased 3 m/s and the Throttle increased as well. Even though there was a positive Altitude error the Throttle was commanding an increase in Energy. The TAS command dropped again at 352 seconds and the Throttle command followed in suit. The Throttle command followed the TAS error throughout the test and ignored the Altitude error.







The figures above depict the period where I altered the Altitude command to create altitude errors. The figure above shows that the there was a -285 meter error in altitude. The figure also shows that the Throttle did not respond at all to the negative altitude error and that the Throttle responded in sync with the TAS error.

The results indicated that, in Altitude Control, the Throttle will respond to TAS errors and not Altitude errors; hence, the Energy Control outer loop only takes input from TAS, presumably via the Kinetic Energy equation.

Test 8:

For Test 8 I flew the autopilot in Airspeed Control to determine how Throttle would respond to Altitude and TAS errors. The idea was to set the Airspeed Control inner loop gains, and the Altitude Control inner loop gain to zero so that Airspeed Control would not be able to command Elevator deflections to influence airspeed. In that scenario I would be able to change the Altitude and Airspeed commands and observe the Throttle response without Elevator deflections trying to correct for TAS errors.

Gains:

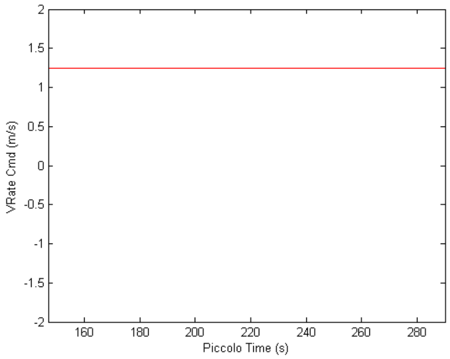
Alt Rate err to accel = 0

TAS rate err to accel cmd = 0   TAS rate err to accel der cmd = 0

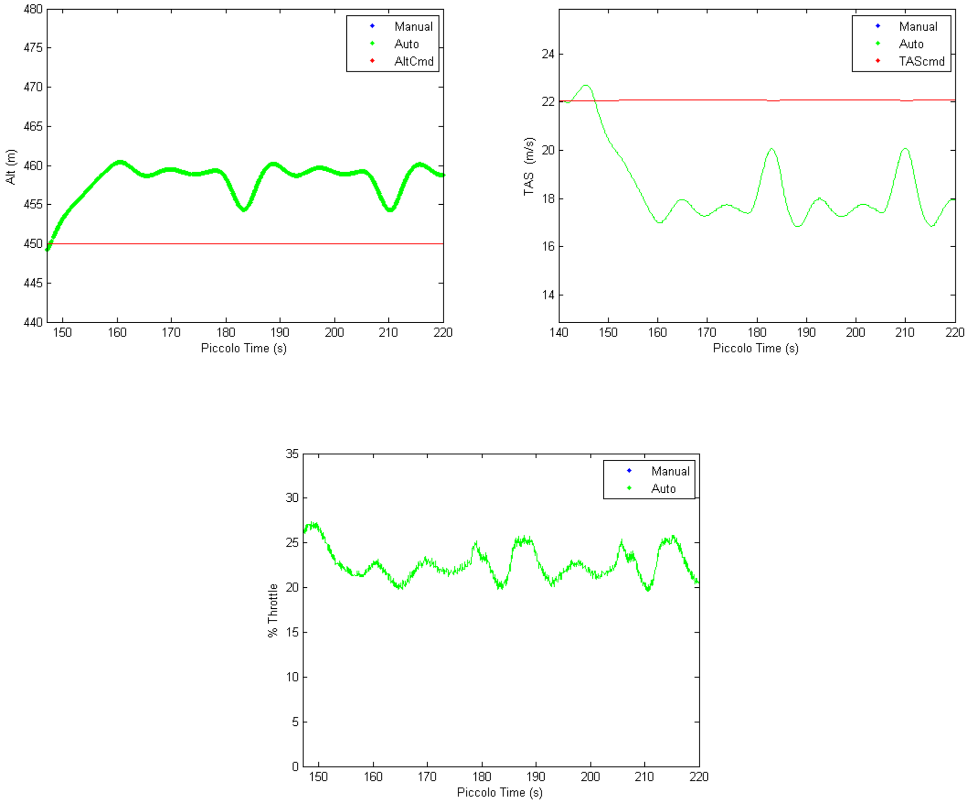
Energy err to Energy Rate = 0.35   Energy Rate err to Throttle = 0.6

Energy Rate err int to Throttle = 0.4   TPT = 0.0   Throttle LPF Cutoff = 0 Hz

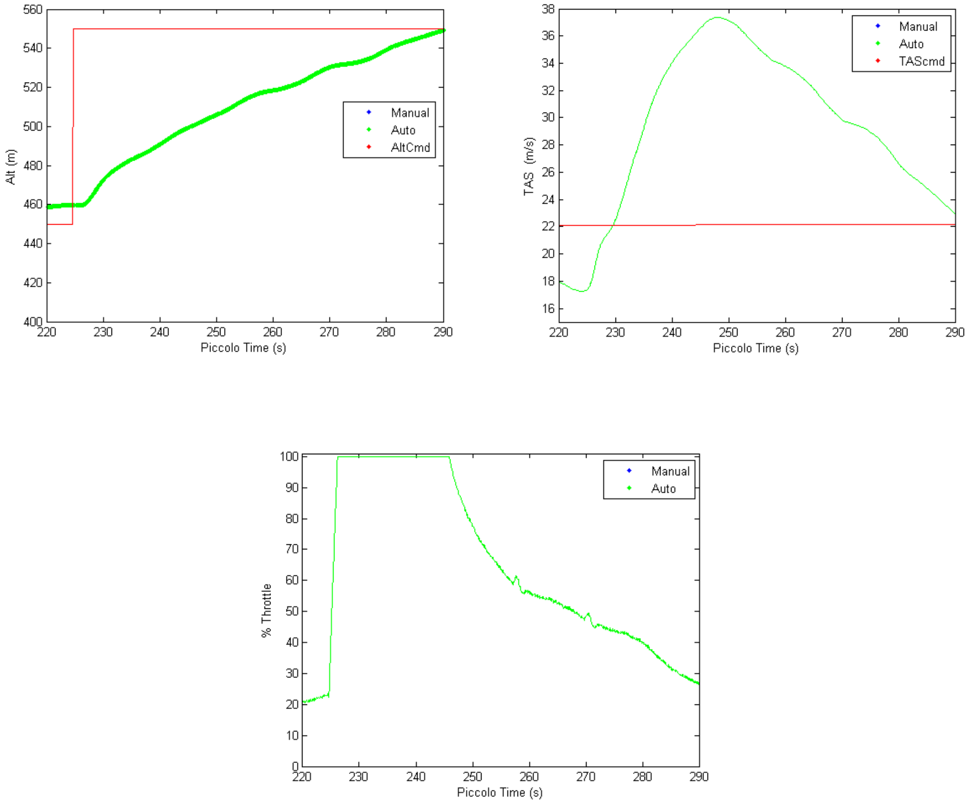
Results:



The figure above shows that the VRate command was zero throughout the time period the tests were conducted in the simulation.



The figure above shows that the positive Altitude error increased in the beginning. The figure also shows that the Throttle was initially on an upward trend (at 150 seconds) but then decreased as the Altitude error grew positive between 150 and 160 seconds. The airspeed decreased throughout the time period of 150-160 seconds, and yet the Throttle command decreased as well. The Throttle oscillated in response to the Altitude error oscillations as the peaks and troughs of the Throttle came about 0.3 seconds (on average) after the Altitude peaks and troughs. TAS oscillated in response to the Throttle changes as the peaks and troughs of TAS came about 0.2 seconds (on average) after the Throttle peaks and troughs.



The figure above depicts the period where I altered the Altitude command to create altitude errors. The figure also shows that the there was a -90 meter error in altitude. Additionall it is shown that the Throttle responded to the sudden negative Altitude error by throttling up all the way to Throttle Max. The Throttle decreased as the Altitude error began to decrease in magnitude. The figure above shows that a large positive TAS error occurred throughout the time period. Even though the TAS error increased the Throttle kept commanding a Throttle increase.

The results indicated that, in Airspeed Control, the Throttle will respond to Altitude errors and not TAS errors; hence, the Energy Control outer loop only takes input from Altitude, presumably via the Potential Energy equation.

The results from Tests 7 and Test 8 both indicated that the outer loop input and feedback of Energy Control is dependent on whether or not the autopilot is in Altitude or Airspeed Control. Note that Altitude errors can still impact Throttle in Altitude Control via the Potential Energy

Rate feedback and the Potential Energy Rate Command that commands the inner loop in addition to the Energy Control outer loop Energy Rate command.

Another unknown is the impact of the Vehicle Parameter “Max Engine Power.” According to gain definitions the “Max Engine Power” does more than just serve as a multiplier for the “Throttle Prediction Trust”, it is also used to scale throttle feedback gains.

If the “Max Engine Power” is used to scale the Throttle feedback gains then it should affect Throttle when the Throttle Prediction Trust is zero. I setup a software in the loop simulation to determine if any changes to the Throttle commands would occur when the “Max Engine Power” was altered.

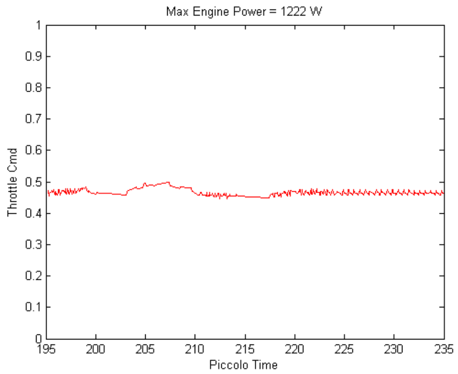
Gains:

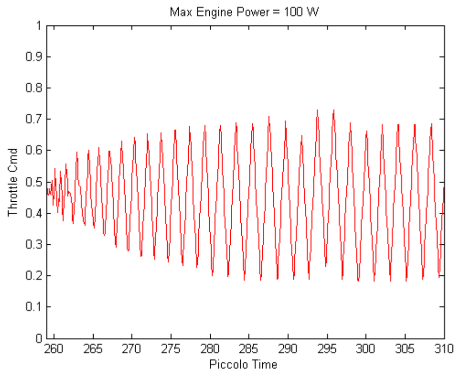
Energy err to Energy Rate = 1.0 Energy Rate err to Throttle = 0.4

Energy Rate err int to Throttle = 0.6 TPT = 0.0 Throttle LPF Cutoff = 0 Hz

Initially I set Max Engine Power = 1222 W. For the second lap I set Max Engine Power = 100 W.

Results:





The figures show that the Throttle command increased when Max Engine Power was decreased; thus, Max Engine Power does affect Throttle without Throttle Prediction Trust. The figures also confirm that Max Engine Power can induce Throttle oscillations.

Conclusions:

1) The Energy Rate command has only 2 paths to effect Throttle:

    a) Through the Throttle Prediction Trust

    b) Through the Energy Rate error feedback loop (Energy Control inner loop)

2) Energy Rate is commanded from more than just the Energy Control outer loop.

3) Energy Control is not independent of Altitude and Airspeed Control.

4) In Altitude Control the VRate command is used to contribute to the Energy Rate command.

5) In Airspeed Control the VRate command is no longer used to contribute to the Energy Rate command.

6) The Energy Rate feedback is actually just the Potential Energy Rate feedback where

$$Potential\ Energy\ Rate\ \left({}^{J}/_{s}\right) = mg(VRate)$$

7) The Energy Command and Energy Feedback terms are:

    a) In Altitude Control

        i) $Energy\ Command\ (J) = \frac{1}{2}m * (TAS_{cmd})^2$

        ii) $Energy\ \ (J) = \frac{1}{2}m * (TAS)^2$

    b) In Airspeed Control

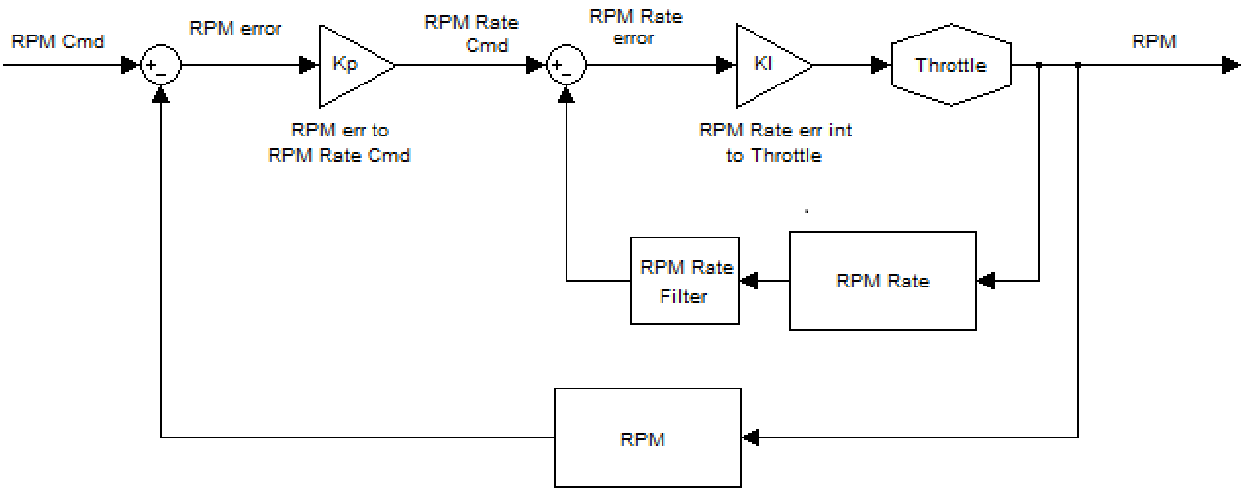
i) $Energy\ Command\ (J) = mg(Alt_{cmd})$

ii) $Energy\ \ (J) = mg * (Alt)$

8) Max Engine Power scales the Throttle feedback gains and alters the commanded Throttle without Throttle Prediction Trust.

4.1.2. RPM Control Experiments

The only documentation that pertains to RPM Control is the gain definitions of the gains in the RPM Control Loop. The gain definitions alone are insufficient in providing a detailed description of RPM Control.



The figure above shows the RPM Control Loop put together from the RPM gain definitions. The schematic raised the following questions:

1) Does RPM Control have full authority over Throttle commands?

2) Where does the RPM command come from?

3) Does RPM control attempt to maintain airspeed within the RPM limits?

I devised a hardware in the loop experiment to attempt to answer these questions. The idea was to have an aircraft fly a simple box pattern, with all the waypoints at the same altitude, and enable RPM Control to observe how it reacted to different airspeed commands. For each test I left the RPM Control gains set to their default values, and I changed RPM Min and RPM Max accordingly.

The only portion of RPM Control that the Dev Interface records is the RPM Rate commands. As a result I had to compare the Throttle commands reported by the PCC telemetry against those recorded by the Dev Interface I used the following facts to assist my analysis of test results. The PCC telemetry data records the Throttle that the autopilot actually commands. The Dev Interface records the Throttle that Energy Control commands. The Dev Interface will record RPM Rate commands only when RPM Control is commanding Throttle.

Test 1:

The purpose of the first test was to determine if RPM Control supersedes Energy Control and commands Throttle on its own. I commanded airspeed so that the airspeed could be maintained within the RPM limits.

Gains:

RPM err to RPM Rate cmd = 1.50   RPM Rate err int to Throttle = 0.0002

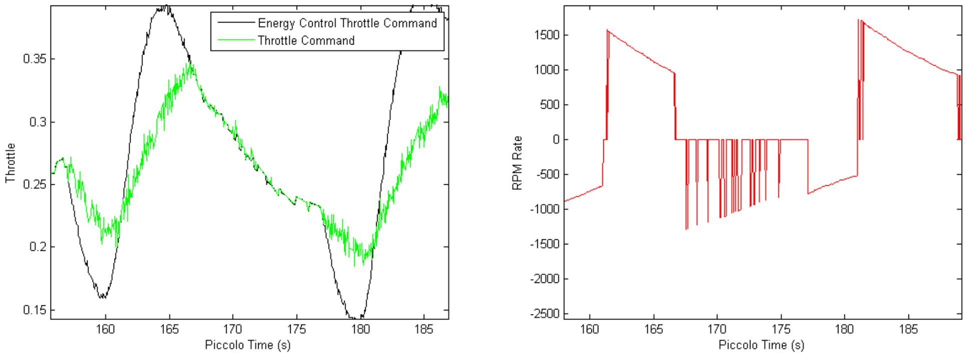
Limits:

IAScmd = 21 m/s

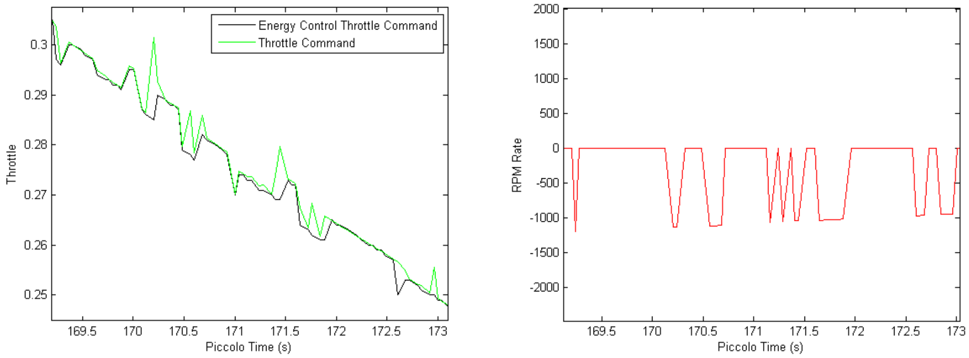
RPM Min = 4000   RPM Max = 5000

Results:

The only portion of RPM Control that the Dev Interface will record is the RPM Rate commands, so there was no way for me to directly observe the RPM Control Throttle commands; however, I noticed throughout my test that the Throttle command displayed by the Dev Interface was not equivalent to the Throttle command that the PCC telemetry reported.



The figure above shows that, at times, the Throttle command is different than the Throttle commanded by Energy Control; thus, RPM Control does occasionally take full command authority of Throttle. The figure also shows that the RPM Rate command is reported by the Dev Interface as greater or less than zero when RPM Control is actually commanding Throttle. I decided to take a closer look at the time period 169 – 173 seconds, where it appears that the Throttle command seemed to switch back and forth between Energy and RPM Control.



The figures show that each time that the Throttle command deviated from the Energy Control Throttle command the Dev Interface recorded an RPM Rate command. This meant that

RPM Control has command authority but only in certain situations. Something caused the controller to switch back and forth between the two.

Test 2:

For the second Test I wanted to test if RPM Control uses airspeed to command Throttle. I tested commanding airspeed inside and outside the ranges of the three different RPM limit scenarios: RPM Min and RPM Max, RPM Max, RPM Min.

1) RPM Min and RPM Max

Gains:

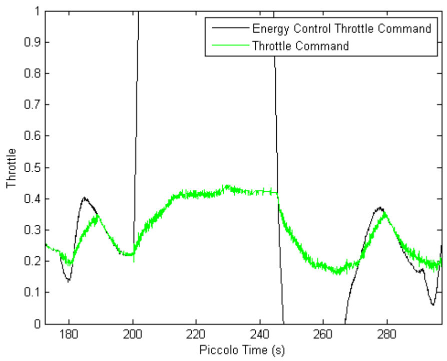
RPM err to RPM Rate cmd = 1.50   RPM Rate err int to Throttle = 0.0002

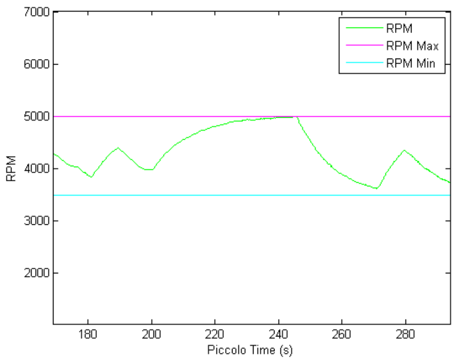
Limits:

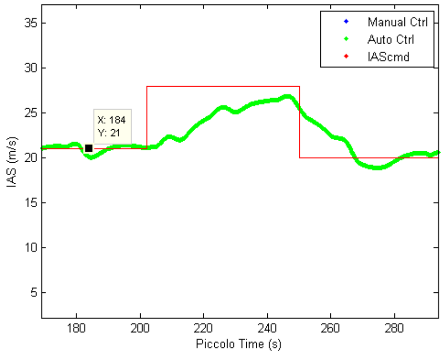
IAScmd = 21, 28, 20 m/s

RPM Min = 3500   RPM Max = 5000

Results:







The figures show that when the IAS command increased, outside of the RPM range, the Throttle command increased until RPM Max was reached. The figures also show that after the IAS command decreased the Throttle command decreased until the airspeed fell below the commanded airspeed where it began to increase Throttle again. The results indicated that when both RPM Max and Min exist RPM Control commands Throttle, within the limits, to maintain airspeed.

2) RPM Max

Gains:

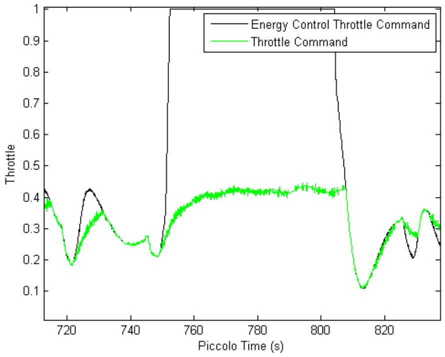
RPM err to RPM Rate cmd = 1.50   RPM Rate err int to Throttle = 0.0002

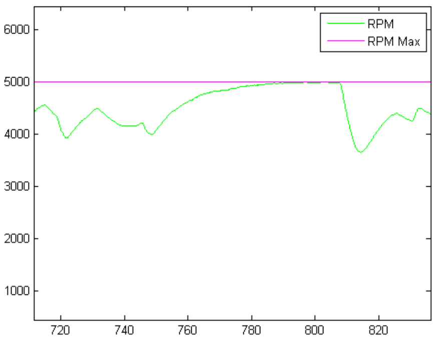
Limits:

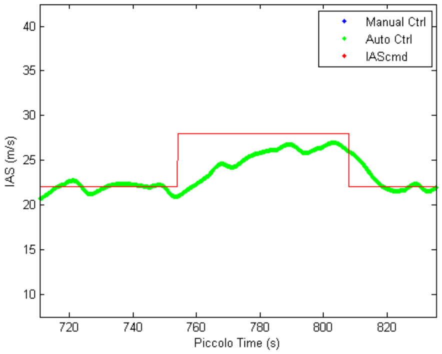
IAScmd = 21, 28, 21 m/s

RPM Min = 0.0   RPM Max = 5000

Results:







The figures show that, when the IAS command increased, the Throttle command increased all the way up to RPM Max to try to reach the airspeed command. After the IAS decreased the Throttle command decreased until the indicated airspeed fell below the commanded airspeed where it increased again. The results indicated that, when only RPM Max exists, RPM Control will command Throttle, within the maximum limit, to maintain airspeed.

3) RPM Min

Gains:

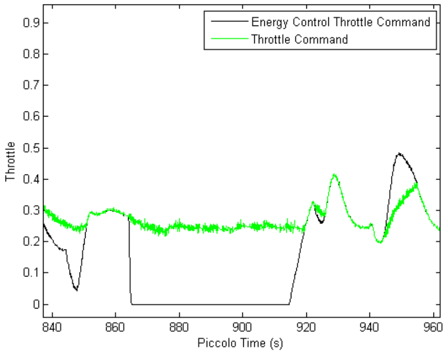
RPM err to RPM Rate cmd = 1.50 RPM Rate err int to Throttle = 0.0002

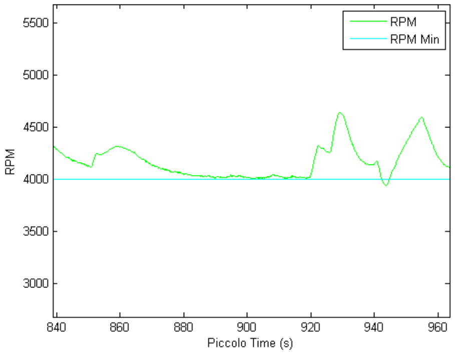
Limits:

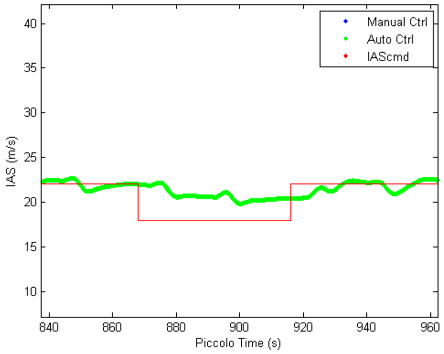
IAScmd = 21, 28, 21 m/s

RPM Min = 4000   RPM Max = 0.0

Results:





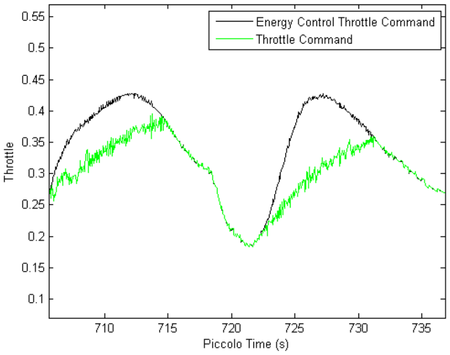


The figures show that, when the airspeed command decreased, the Throttle command and RPMs decreased down to RPM Min. After the IAS command increased the Throttle also increased until the airspeed reached the airspeed command. The results indicated that, when RPM Min exists, RPM Control will command Throttle to maintain airspeed.

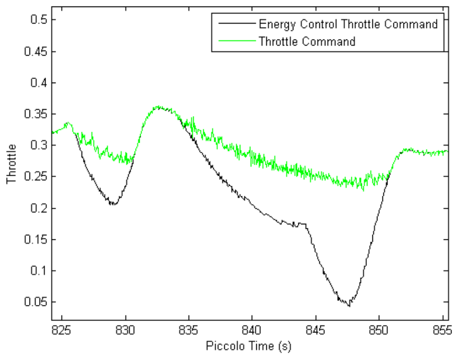
In conclusion Test 2 indicates that when RPM Control is enabled RPM Control will command Throttle to maintain airspeed within the RPM limits regardless of whether or not RPM Min, RPM Max, or both exists.

Test 3:

For the third Test I wanted to determine what will cause the autopilot to give RPM Control Throttle command authority over Energy Control. I had noticed throughout the previous tests that it seemed like it depended on which RPM limits were being used and the magnitude of the Energy Control Throttle command with respect to the RPM Control Throttle command. Referring to the figure below, in the tests where only RPM Max existed I noticed that Energy Control lost Throttle command authority when it increased above the RPM Control Throttle command and regained control once it had seemingly fallen below the RPM Control Throttle command.



This indicated the possibility that Energy Control will lose Throttle command authority when it is higher than the RPM Control Throttle command and RPM Max is the only limit. Similarly I noticed that when only RPM Min existed Energy Control lost Throttle command authority when it decreased below the RPM Control Throttle command depicted in the figure below.



This indicated the possibility that Energy Control will lose Throttle command authority when it is lower than the RPM Control Throttle command and RPM Min is the only limit.

Since I could not observe the RPM Control Throttle commands when RPM Control does not have Throttle command authority I devised a hardware in the loop simulation to test my ideas. The idea behind the test was to fly laps around a simple box pattern and compare the laps with different gain values for “RPM err to RPM Rate cmd”. The first lap would be with default gains, and the following two laps would be with a high and low value for the RPM Control gain. Raising RPM err to RPM Rate cmd should cause RPM Control to command Throttle more aggressively; thus, commanding larger changes in Throttle commands than it would with the default gain value. Similarly lowering RPM err to RPM Rate cmd should cause RPM Control to decrease the magnitude of Throttle commands; thus, commanding smaller changes in Throttle compared to the Throttle commands with the default gains.

If the autopilot determines Throttle command authority based on whether or not the Energy Control Throttle command is above or below (depending on the RPM limits used) the RPM Control Throttle command then in the case of increasing the RPM err to RPM Rate gain the time spent with RPM Control commanding Throttle, throughout one lap, should decrease compared to the lap with the default gain value because RPM Control would be more responsive to changes in airspeed; thus, increasing the allowable range for Energy Control to command

Throttle. In the case of decreasing the RPM err to RPM Rate gain the time spent with RPM Control commanding Throttle, throughout one lap, should increase compared to the lap with the default gain value because RPM Control would be less responsive to changes in airspeed; thus, decreasing the allowable range for Energy Control to command Throttle.

Test 4:

In the fourth test I used RPM Min as the only limit to observe the effects of altering RPM err to RPM Rate cmd" on the amount of time RPM Control receives Throttle command authority throughout one lap.

Gains:

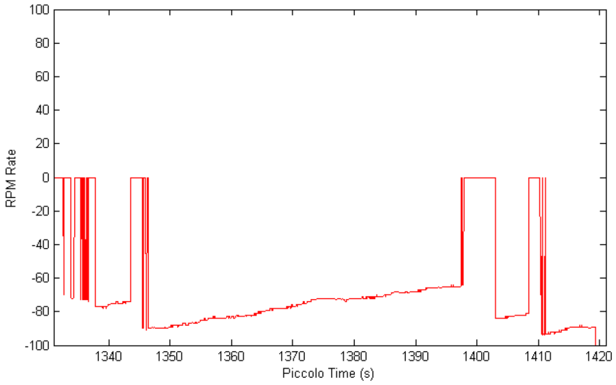
RPM err to RPM Rate cmd = 10, 1.5, 0.10   RPM Rate err int to Throttle = 0.0002

Limits:

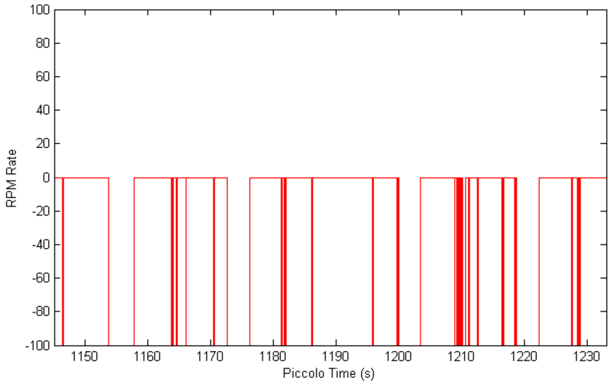
IAScmd = 22 m/s

RPM Min = 3500   RPM Max = 0.0

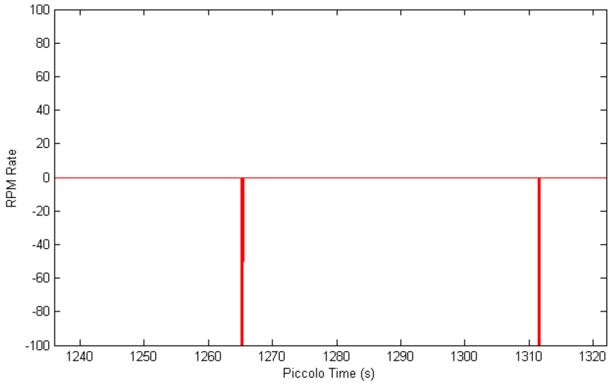
Results:



$$Total\ Lap\ Time = 90.34\ s \quad RPM\ Control\ Time = 79.66s \quad \%RPM\ Control = \frac{79.66}{90.34} = 88.18\%$$



$$Total\ Lap\ Time = 88s \quad RPM\ Control\ Time = 38.04s \quad \%RPM\ Control = \frac{38.04}{88} = 43.23\%$$



$$Total\ Lap\ Time = 86.12s \quad RPM\ Control\ Time = 26s \quad \%RPM\ Control = \frac{26}{86.12} = 30.19\%$$

The results indicated that the autopilot spent more time with RPM Control in command of Throttle at the lowest value of “RPM err to RPM Rate cmd”. The results also indicated that as the RPM err to RPM Rate cmd gain was increased the amount of time the autopilot spent in RPM Control Throttle command authority decreased. The results confirm that, when RPM Min exists, Energy Control can only command Throttle when the Energy Control Throttle command is higher than the RPM Control throttle command.

<u>Test 5:</u>

In the fifth test I used RPM Max as the only limit to observe the effects of altering RPM err to RPM Rate cmd" on the amount of time RPM Control receives Throttle command authority throughout one lap.

Gains:

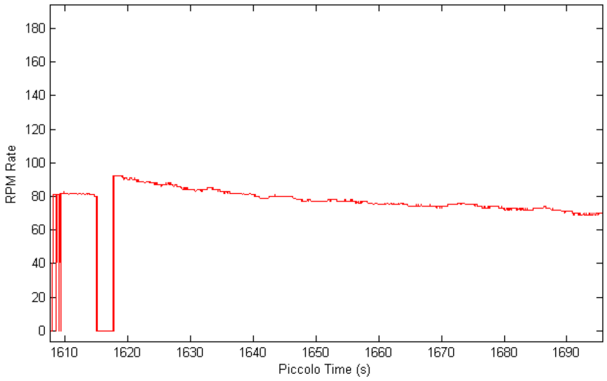
RPM err to RPM Rate cmd = 10, 1.5, 0.10   RPM Rate err int to Throttle = 0.0002

Limits:

IAScmd = 22 m/s

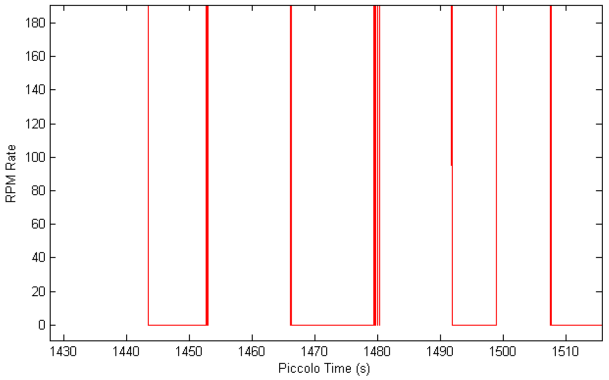
RPM Min = 0.0   RPM Max = 5000

Results:



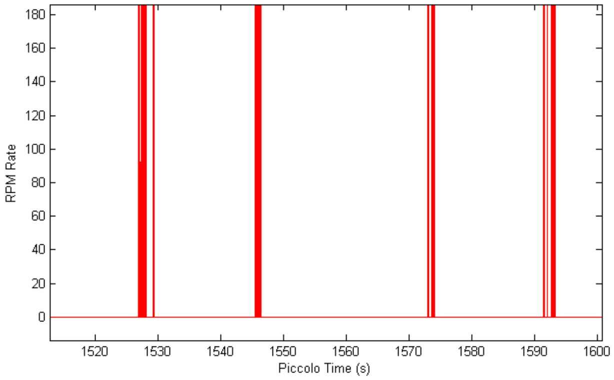
$$Total\ Lap\ Time = 95.04s \quad RPM\ Control\ Time = 93.28s \quad \%RPM\ Control = \frac{93.28}{95.04}$$

$$= 93.28\%$$



$$Total\ Lap\ Time = 81.92s \quad RPM\ Control\ Time = 57.8s \quad \%RPM\ Control = \frac{57.8}{81.92}$$

$$= 70.56\%$$



$$Total\ Lap\ Time = 84.24s \quad RPM\ Control\ Time = 26.32s \quad \%RPM\ Control = \frac{26.32}{84.24}$$

$$= 31.24\%$$

The results indicated that the autopilot spent more time with RPM Control in command of Throttle at the lowest value of “RPM err to RPM Rate cmd”. The results also indicated that as the RPM err to RPM Rate cmd gain was increased the amount of time the autopilot spent in RPM Control Throttle command authority decreased. The results confirm that, when RPM Max exists, Energy Control can only command Throttle when the Energy Control Throttle command is lower than the RPM Control throttle command.

Conclusions:

1) RPM Control attempts maintain the commanded indicated airspeed within the RPM limits; thus, RPM command is calculated in order to maintain airspeed.

2) RPM Control commands Throttle independently of Energy Control and has full Throttle command authority when the Energy Control Throttle command is above or below (depending on the RPM Limits) the RPM Control Throttle command.

5. Lon Mode Experiments

There is no Cloud Cap documentation that describes the specifics of Lon Modes; therefore I had to devise experiments to determine the functionality and logic of Lon Modes. Throughout the experiments I determined that Lon Mode 0 is Altitude Control, and Lon Modes 1-3 are different variations of Airspeed Control. There are a couple mentions of Lon Modes in the Release Notes. I found one mention of Lon Modes under the fixed wing autopilot firmware bug fixes. The specific bug fix reads “Fixed a bug in the logic for transitioning from slow airspeed mode to altitude mode. If the vehicle was exiting slow airspeed mode but the engine power was enough to saturate the climb rate command then the transition back to altitude mode would not happen until the climb completed.” . I also noticed from my experiments that Lon Mode 3 uses Airspeed Control to slow down an aircraft, and Lon Mode 1 is complete Airspeed Control; thus, I determined, and assumed, the following names for Lon Modes:

Lon Mode 0:	Altitude Mode
Lon Mode 1:	Airspeed Mode
Lon Mode 2:	Slow Airspeed Mode
Lon Mode 3:	Fast Airspeed Mode

5.1.1. Lon Mode 0 (Altitude Mode) Experiments

Just from flying some regular hardware in the loop simulations I observed that the autopilot flies in Lon Mode 0 during most steady level flight where no airspeed limitations had been violated. This led me to believe that Lon Mode 0 is more than likely Altitude Control.

I devised an experiment to test whether or not Lon Mode 0 is Altitude Control. I setup a hardware in the loop simulation with a simply box pattern flight plan with climb and descent waypoints. In the simulation I set two airspeed control gains to ridiculously high values so that if the autopilot was in airspeed control the elevator commands would oscillate out of control and cause the aircraft to oscillate in pitch. The two airspeed control gains I manipulated were "TAS rate error to acceleration command" and "TAS rate error derivative to acceleration command". These two were singled out because their definitions specifically state that they are only used in airspeed control. I left the Altitude Control Loop gains at their default values.

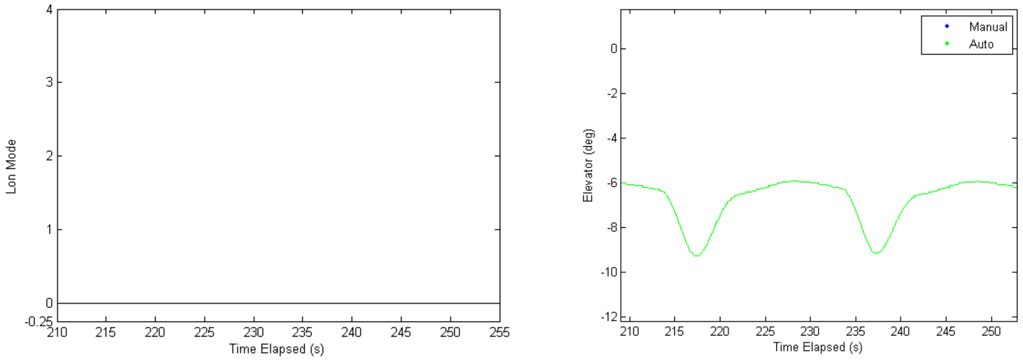
Gains:

Slow IAS error Threshold = 100 m/s Fast IAS error Threshold = 20 m/s

TAS rate err to accel cmd = 10 TAS rate error to accel der cmd = 10

Alt err to Alt Rate = 0.20 Alt rate err to accel = 0.75

Results:



The figure above shows that there were no divergent elevator oscillations; thus, I was able to conclude that Lon Mode 0 is in fact altitude control.

5.1.2. Lon Mode 1 (Airspeed Mode) Experiments

According to "Tuning piccolo control laws 2.0.x" there is one way to force the autopilot into airspeed control. The document states that setting the Fast IAS error Threshold to a negative number will force the autopilot into airspeed control (Tuning piccolo control laws 2.0.x pg. 14).

I setup a hardware in the loop simulation to test the following regarding setting the Fast IAS error Threshold to a negative number:

1) Does the setting force the autopilot into airspeed control? If so will the autopilot remain in airspeed control throughout climbs and descents?

2) Does this correspond to a Lon Mode? If so which one?

For the simulation I setup a simple box pattern without any climbs or descents. After the simulation began I set Fast IAS error Threshold to -1. The idea was to set the airspeed control gains to ridiculously large numbers so that if the autopilot switched to Airspeed Control the autopilot Elevator commands would oscillate I would be able to discern that the autopilot was indeed in Airspeed Control.

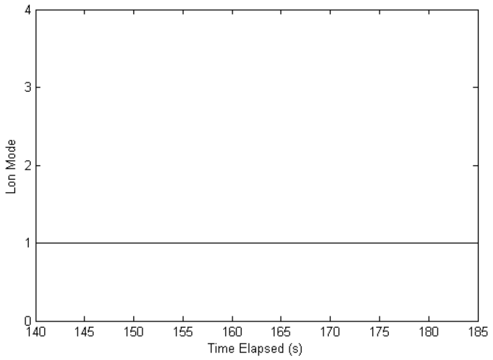
Gains:

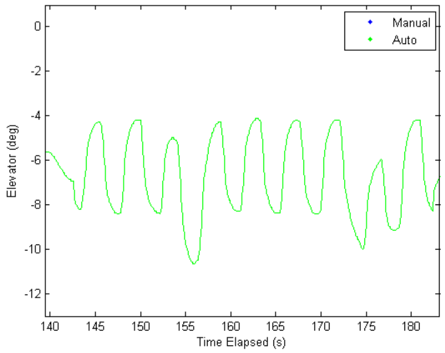
Slow IAS error Threshold = 100 m/s Fast IAS error Threshold = 10 m/s, -1 m/s

TAS rate err to accel cmd = 10 TAS rate error to accel der cmd = 10

Alt err to Alt Rate = 0.20 Alt rate err to accel = 0.75

Results:





The figures show that in Lon Mode 1 the Elevator command oscillated heavily; thus, Lon Mode 1 is Airspeed Control.

In order to test if the autopilot would stay in Airspeed Control throughout climbs and descents I altered the flight plan to include climb and descent waypoints. I also set the Airspeed Control gains back to their default values.

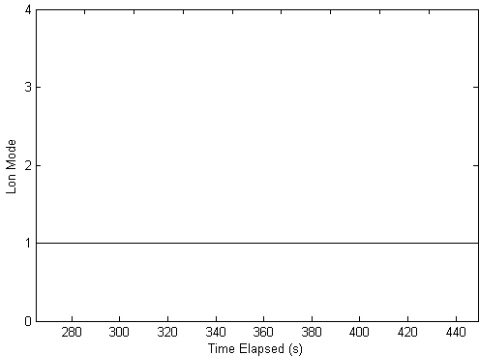
Gains:

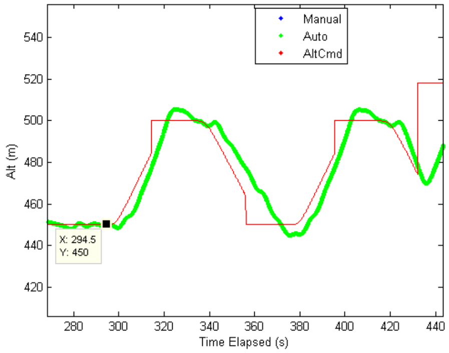
Slow IAS error Threshold = 100 m/s Fast IAS error Threshold = -1 m/s

TAS rate err to accel cmd = 1.5 TAS rate error to accel der cmd = 1.0

Alt err to Alt Rate = 0.20 Alt rate err to accel = 0.75

Results:





The figures show that the autopilot stayed in Lon Mode 1 throughout the climbs and descents.

Conclusions:

1) Setting the Fast IAS error Threshold to -1 will force the autopilot into airspeed control where it will remain until Fast IAS error Threshold is set back to a positive number.

2) Lon Mode 1 corresponds with the autopilot being forced into airspeed control where it will remain.

5.1.3. Lon Mode 2 (Slow Airspeed Mode) Experiments

Lon Mode 2 is the Slow Airspeed Mode. Slow Airspeed Mode is used to prevent the autopilot from stalling an aircraft by switching the autopilot into Airspeed Control

According to “Tuning piccolo control laws 2.0.x” there are several factors that can contribute to the autopilot deciding to switch into airspeed control. One of them is:

- The airspeed is lower than the minimum value, and the throttle integrator has reached 90% of full throttle. This indicates that the engine cannot provide enough power to maintain airspeed under the current conditions (Tuning piccolo control laws 2.0.x 14)

Since the condition is for an older version of the piccolo software it cannot be trusted word for word. That being said it does offer insight into the general logic of the autopilot. The condition indicates that flying too slow below airspeed minimums can cause the autopilot to switch into airspeed control.

The document also asserts the following in regards to exiting airspeed control: "Once the autopilot has switched into airspeed control it will remain there until the conditions given above are removed and when the error in vertical rate has become less than 10% of the allowable vertical rate command" (Tuning piccolo control laws 2.0.x 14).

I simulated a couple of steep climbs to determine what Lon Mode corresponded with flying too slow. The climbs were setup to be steep enough that the aircraft would violate limits concerning minimum airspeed.   I found that the autopilot uses Lon Mode 2 when flying too slowly. To better understand Lon Mode 2 I ended up simulating a total of seven climbs in a hardware in the loop simulation. The climbs were setup between waypoints of two different flight plans so that the autopilot would climb as steeply as allowed by $\text{VRate}_{\text{max}}$. In all seven climbs I analyzed VRate, IAS, Lon Mode, and Throttle. I looked for correlations in each climb that would indicate:

a) What causes the autopilot to switch to Lon Mode 2

b) What causes the autopilot to switch back to Lon Mode 0

In each climb all the limits were set to default values unless otherwise specified. Also I set the Fast IAS error Threshold to 20 m/s for all the climbs because it seemed like Lon Mode 2 was concerned with flying too slowly and I didn't want to chance the autopilot switching to a different Lon Mode other than 2 or 0 during my tests.

Climb 1:

For the first climb I set the following parameters:

Gains:

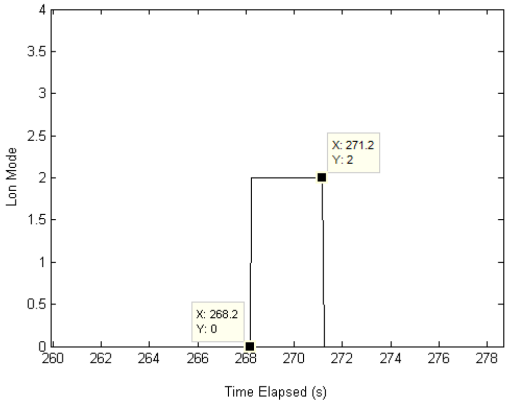
Slow IAS error Threshold = 100 m/s

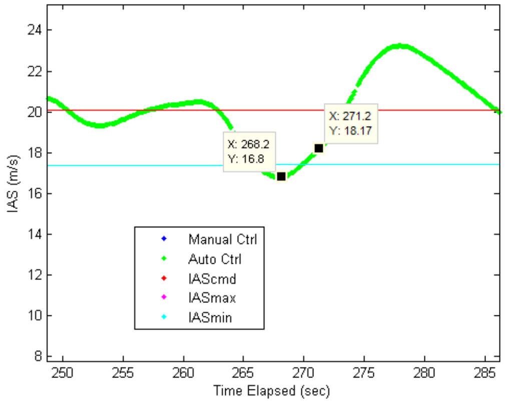
$C_{L\,\max nom} = 1.478 \Rightarrow IAS_{min} = 17.37\ m/s$

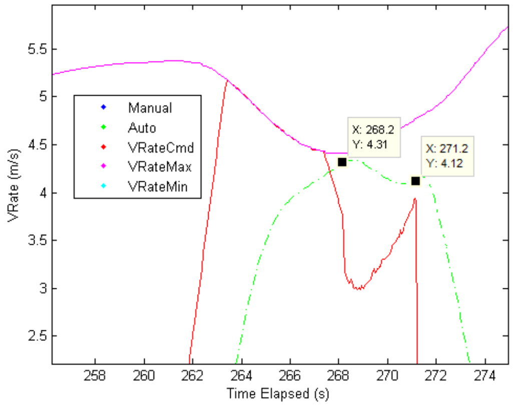
Limits:

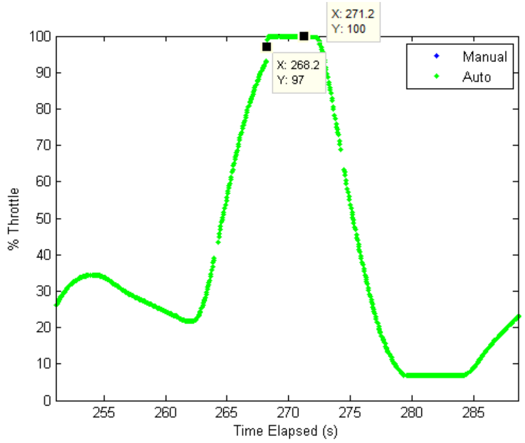
Climb Max Fraction = 0.25

Results:









Analyzing the figures the autopilot went into Lon Mode 2 after the aircraft was below IASmin and the Throttle command was very near Max Throttle (97%). The autopilot exited Lon Mode 2 at an IAS of 18.17 m/s which is approximately equal to 105% of IASmin (17.37 m/s). Also at the moment the autopilot exited Lon Mode 2 the vertical rate error was at 1.47% of the allowable vertical rate range.

$$VRate_{max} = 9.54\frac{m}{s} \quad VRate_{min} = \ -4.78 \ \ Allowable\ Range = 9.54 + 4.78 = 14.32\frac{m}{s}$$

$$VRate = 4.12\frac{m}{s} \quad VRate_{cmd} = 3.91 \ \ VRate\ Error = 4.12 - 3.91 = 0.21\ m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.014665$$

Climb 2:

For Climb 2 I increased VRate$_{max}$ to see how the autopilot responded in a steeper climb, and to determine if the conditions where the autopilot switched to Lon Mode 2 change if the VRate command isn't limited by VRate$_{max}$.

Gains:

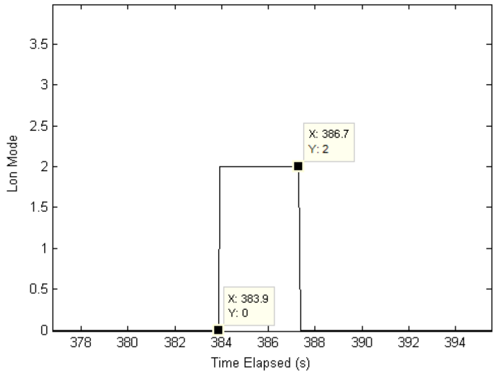
Slow IAS error Threshold = 100 m/s

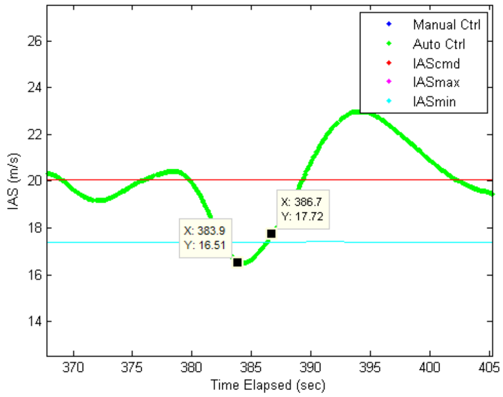
$$C_{L\,\max nom} = 1.478 \;\Rightarrow\; IAS_{min} = 17.37\ m/s$$

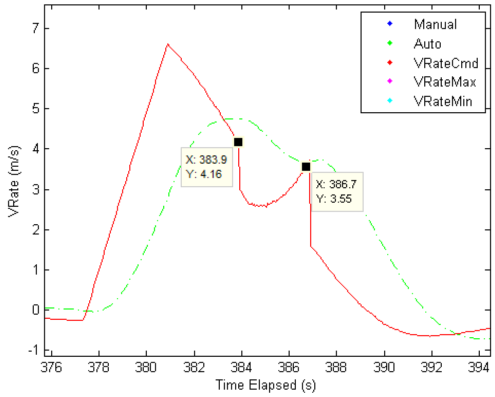
Limits:

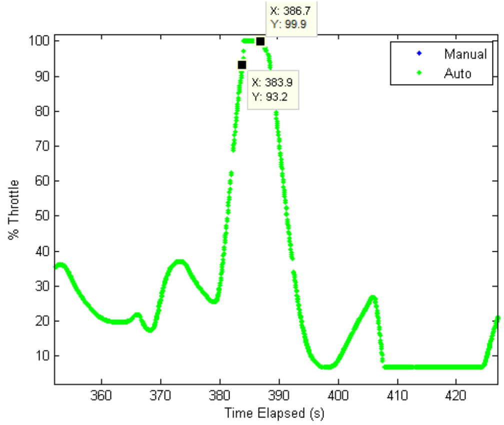
Climb Max Fraction = 0.50

Results:









Analyzing the results the autopilot changed to Lon Mode 2 after IAS was below IAS$_{min}$ and when Throttle had nearly reached Throttle Max (93%). Approximately 0.12 seconds after entering Lon Mode 2 Throttle was equal to Throttle Max (100%). The autopilot switched back to Lon Mode 0 when the indicated airspeed reached 17.72 m/s, 102% of IASmin (17.37 m/s). VRate command never reached VRate$_{max}$. Also at the moment the autopilot exited Lon Mode 2 the vertical rate error was at 0.715% of the allowable vertical rate range.

$$VRate_{max} = 9.325\frac{m}{s} \quad VRate_{min} = \ -4.662 \ \ Allowable\ Range = 9.325 + 4.662 = 13.99\frac{m}{s}$$

$$VRate = 3.68\frac{m}{s} \quad VRate_{cmd} = 3.58 \ \ VRate\ Error = 3.68 - 3.58 = 0.10\ m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.00715$$

Climb 3:

For Climb 3 I wanted to test if Throttle reaching Throttle Max could trigger a switch to Lon Mode 2 independently of indicated airspeed. To test this I increased CL$_{maxnom}$ to decrease IAS$_{min}$ so that IAS$_{min}$ would not be violated.

Gains:

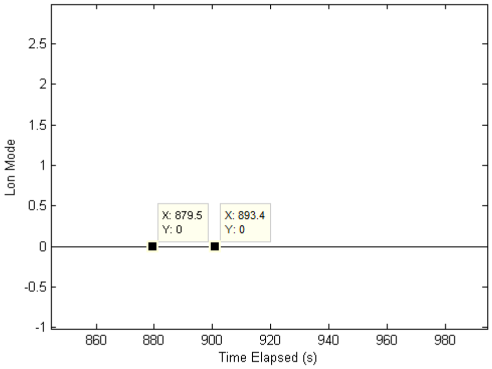
Slow IAS error Threshold = 100 m/s

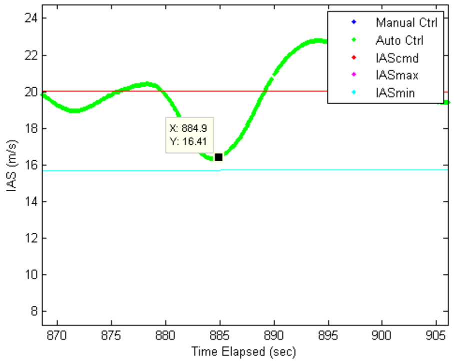
$$C_{L\,\max nom} = 1.80 \;\Rightarrow\; IAS_{min} = 15.69\; m/s$$

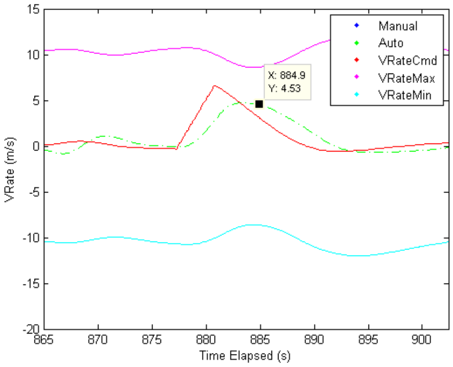
Limits:

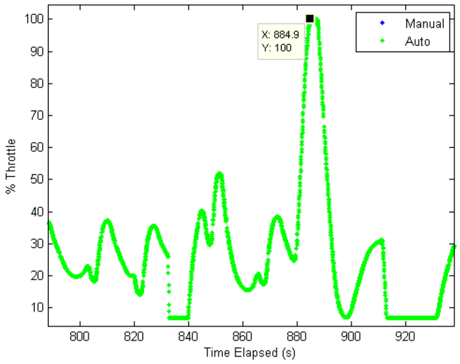
Climb Max Fraction = 0.50

Results:









Analyzing the results the autopilot never switched to Lon Mode 2. The autopilot remained in Lon Mode 0 throughout the entire climb even though it reached Throttle Max and indicated airspeed had decreased well below commanded. Indicated airspeed never fell below IAS$_{\text{min}}$.

Climb 4:

In Climb 4 I increased IAS$_{min}$ to see if the autopilot would go into Lon Mode 2 from violating IAS$_{min}$ alone, independently of reaching Throttle Max.

Gains:

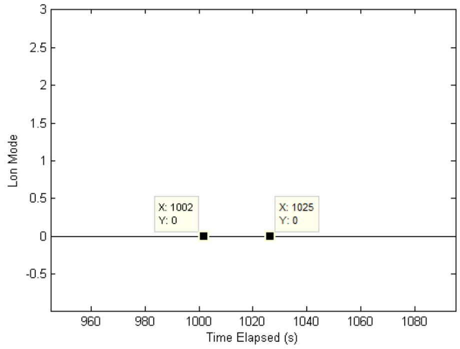
Slow IAS error Threshold = 100 m/s

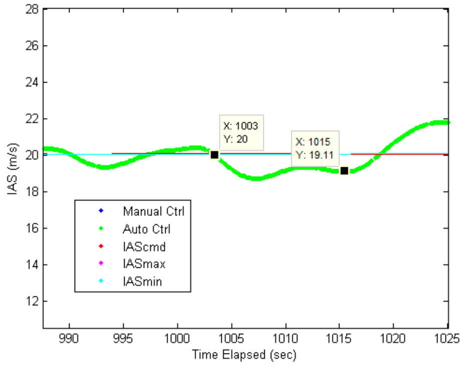
$$C_{L\,\max nom} = 0.9 \;\Rightarrow\; IAS_{min} = 20\ m/s$$

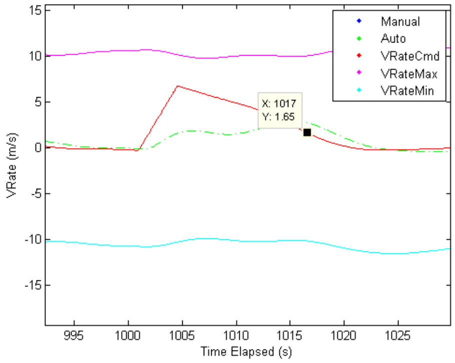
Limits:

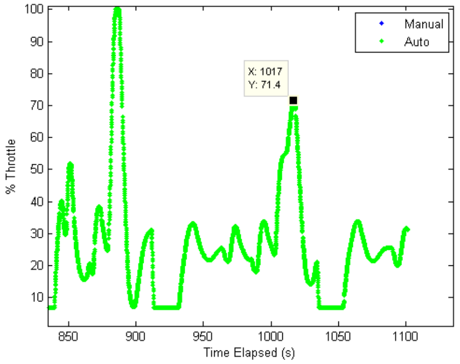
Climb Max Fraction = 0.50

Results:









Analyzing the figures the aircraft almost immediately violated IAS_{min} when the climb was initiated (1002 sec). Throughout the climb the aircraft stays below IAS_{min}. Throttle climbs up to 71.4% at the most, short of Throttle Max. Lon Mode remained in Mode 0 throughout the entire climb.

The results of Climbs 3 and 4 seem to indicate that the autopilot will only switch to Lon Mode 2 when the indicated airspeed is less than IAS_{min} and the Throttle is nearly full.

<u>Climb 5:</u>

For Climb 5 I wanted to test when the autopilot will switch back from Lon Mode 2 to Lon Mode 0 again for repeatability of the previous results with regards to the indicated airspeed reaching nearly 105% of the commanded airspeed. It seemed that the earlier results could have just been coincidence.

Gains:

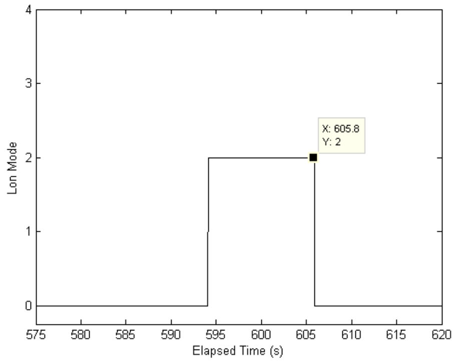
Slow IAS error Threshold = 100 m/s

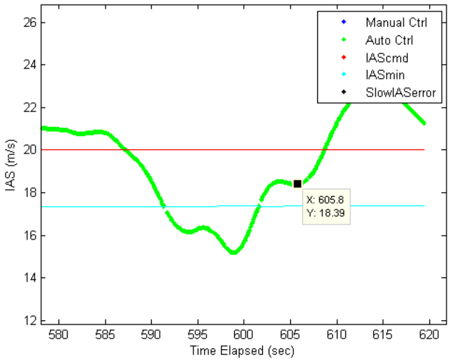
$C_{L\,\max nom} = 1.478 \Rightarrow IAS_{min} = 17.37\ m/s$

Limits:

Climb Max Fraction = 0.15

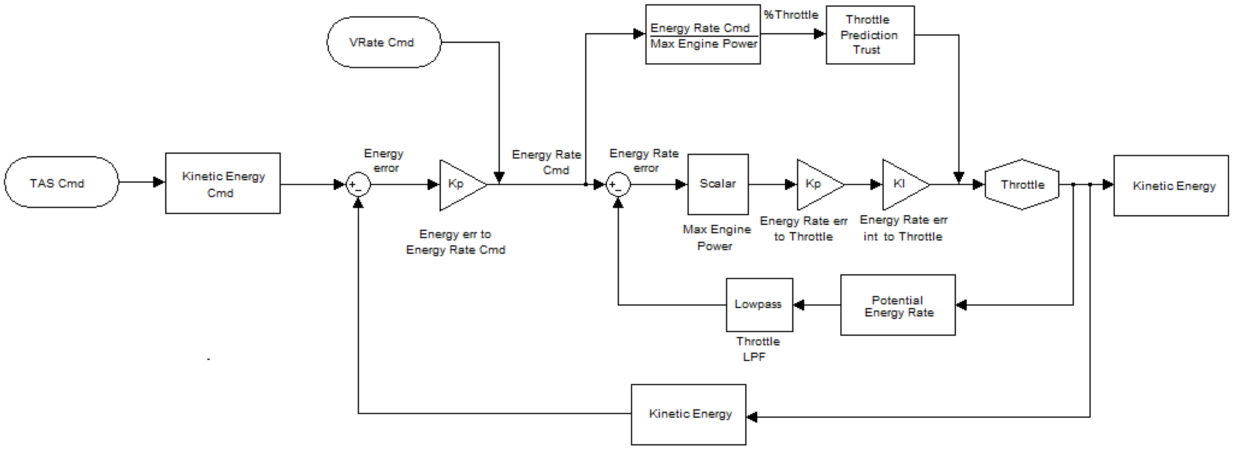
Results:



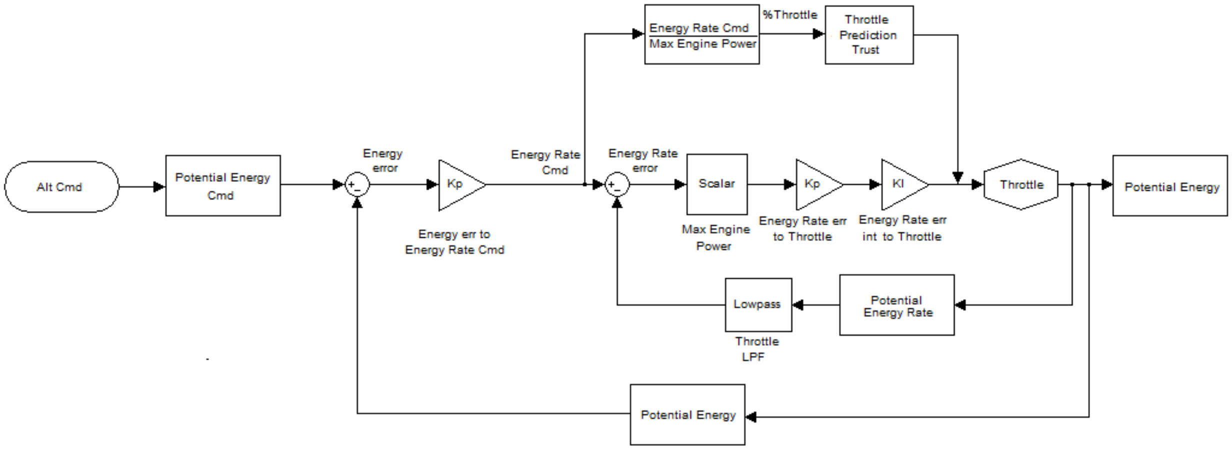


The results, depicted in the figures, indicated that the autopilot switched back to Lon Mode 0 at 605.8s where the indicated airspeed was equal to 18.39 m/s, 105.9% of IAS$_{min}$ (IAS$_{min}$ = 17.37 m/s); however, before this time the airspeed had already reached 18.39 m/s and reached even higher (up to 18.5 m/s) before falling back down. The only real consistency analyzing indicated airspeed is that the indicated airspeed is above IAS$_{min}$ when the autopilot returns to Lon Mode 0.

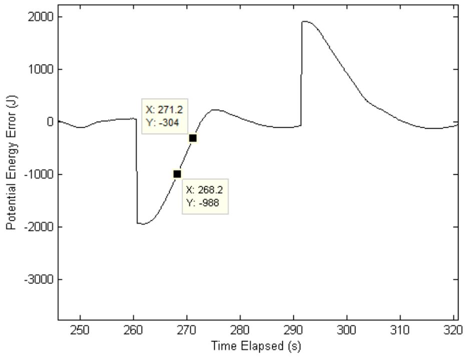
At this point the only real conclusion I had on when the autopilot decides return to Lon Mode 0 is when the autopilot determines that the Throttle is capable of managing airspeed. In order to come up with a more specific explanation I had to re-examine the results of all the tests from the perspective of Energy Control. From the Energy Control results I knew that in Altitude Control the Energy Rate command comes from the Energy Control outer loop, where the Energy error is based on the TAS error via the Kinetic Energy equation, and from the VRate command via the Potential Energy Rate equation.

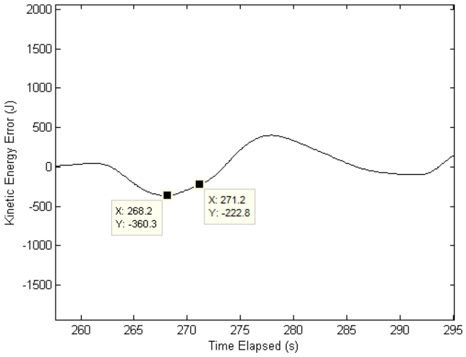


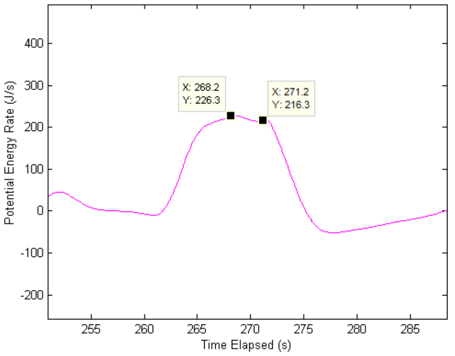
Also from the Energy Control results I knew that in Airspeed Control the Energy Rate command is based only on the Altitude error via the Potential Energy equation.

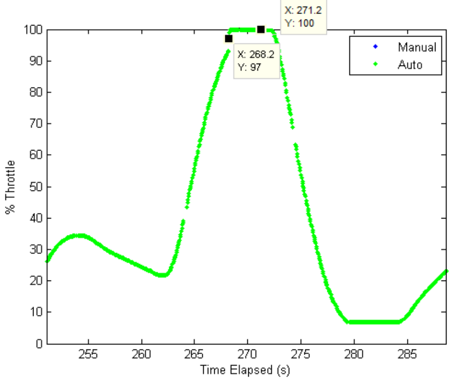


With Energy Control logic in mind I re – analyzed the results of Climbs 1, 2 and 5. Specifically I looked at Kinetic Energy error (TAS error), Potential Energy error (Altitude error), Throttle, and the Potential Energy Rate (VRate). The idea was to follow the logic of Energy Control as it responded throughout the time before the autopilot entered Lon Mode 2, while it was in Lon Mode 2, and when it exited Lon Mode 2.









Analyzing the figures from Climb 1 I stepped through the logic as follows. At about 261 seconds the autopilot began to climb, in Altitude Control. The Kinetic Energy Error became increasingly negative throughout the climb. The negative Energy Error created a positive Energy Rate command (to add Energy to the system) and the Energy Rate command increased as the Energy Error became a larger negative value. VRate grew larger throughout the climb which means that the Energy Rate feedback term became larger; therefore, the Energy Rate command was increasing while the Energy Rate feedback was increasing as a negative value (the feedback is subtracted). This caused a large Energy Rate error which caused the Throttle to climb sharply. At about 268.2 seconds the autopilot went into Lon Mode 2, and the Throttle jumped almost immediately (restricted by the Throttle Rate limit) to Throttle Max. Since the autopilot was in Airspeed Control at that time the Energy error became based on the Potential Energy error (Altitude), and the Throttle trim was set automatically to Throttle Max. During the time period of

Airspeed Control the Kinetic Energy error decreased, as it was controlled by the Elevator to regulate airspeed, the Potential Energy error decreased as the aircraft slowly climbed (caused a decrease in the Energy Rate command), and the Potential Energy Rate decreased. As a result both the Energy Rate command and Energy Rate feedback decreased which means that the Energy Rate error greatly decreased. Since the Kinetic Energy error was not present in the Energy Control logic at that time it seemed reasonable to rule out Kinetic Energy as a reason for leaving Lon Mode 2. After the autopilot returned to Altitude control the Energy Rate command switched back to being based on the Kinetic Energy error and the VRate command. The Throttle remained at Throttle Max shortly until about a second after the Kinetic Energy error and Potential Energy Rate had decreased (caused a decrease in Energy Rate error).

As a result of the analysis I concluded that the autopilot decided to return to Lon Mode 2 because as the Energy Rate error decreased the Throttle integrator would have decreased as well since it is based on the Energy Rate error. Since the Throttle integrator sets the Throttle Trim it seemed reasonable to conclude that at a certain point the Throttle integrator began to lower the Throttle Trim; thus, preparing Energy Control to begin the process of decreasing Throttle. If the inner loop of Energy Control began to command a decrease in Throttle then there would not be any reason to remain in Airspeed Control, because the Throttle would be prepared to throttle down. This lead me to conclude that the autopilot will return to Altitude Control when the Throttle trim is lowered which is caused by a decrease in Altitude error; therefore, with respect to Throttle, the autopilot returns to Altitude Control when the Altitude error has decreased enough to cause the throttle integrator to decrease the Throttle trim.

I analyzed Climbs 2 and 5 and they both showed the exact same behavior as Climb 1.

Climbs 1 – 5 dealt with violating IAS_{min} ;however, there is another minimum airspeed limit that exists. The “Slow IAS error Threshold”, located in the Altitude Control Loop gains section, also sets a minimum standard for airspeed. I decided to test if violating the Slow IAS error Threshold is treated in the same manner as violating IAS$_{min}$.

Climb 6:

For Climb 6 I set IAS$_{min}$ and the Slow IAS error Threshold to low values so that the Slow IAS error Threshold would be violated and not IAS$_{min}$.

Gains:

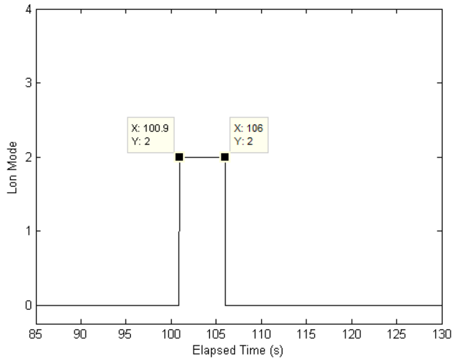
Slow IAS error Threshold = 1.5 m/s

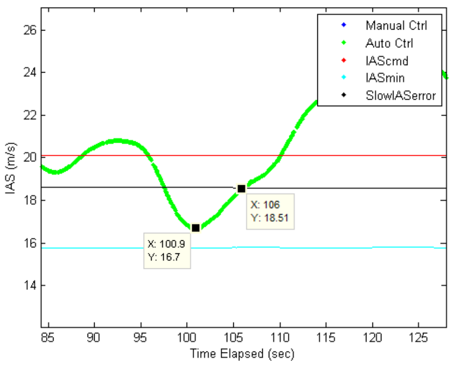
$$C_{L\,\max nom} = 1.80 \;\Rightarrow\; IAS_{min} = 15.69\; m/s$$

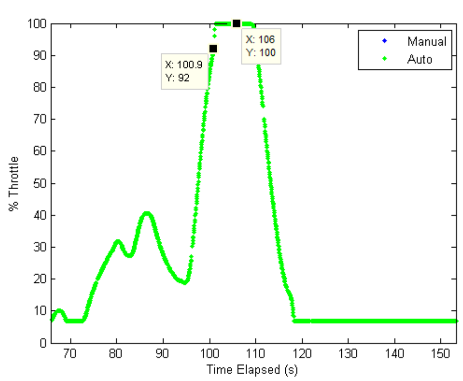
Limits:

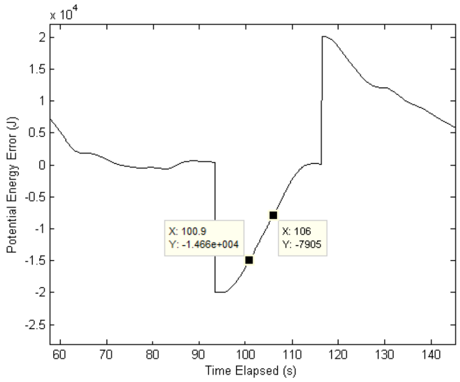
Climb Max Fraction = 0.25

Results:









The figures show that, similar to IAS$_{min}$, the autopilot went into Lon Mode 2 after the Slow IAS error Threshold had been violated. Also the Throttle was at 92% of Throttle Max when the autopilot switched over. The figures also show that the autopilot switched back to Lon Mode 0 as soon as the indicated airspeed became greater than the Slow IAS error Threshold. The figure above shows that the Potential Energy error had decreased significantly. It is likely that the Throttle trim had decreased prior to the time that the airspeed reached the Slow IAS error Threshold.

Climb 7:

I conducted the same climb for repeatability of the results of Climb 6.

Gains:

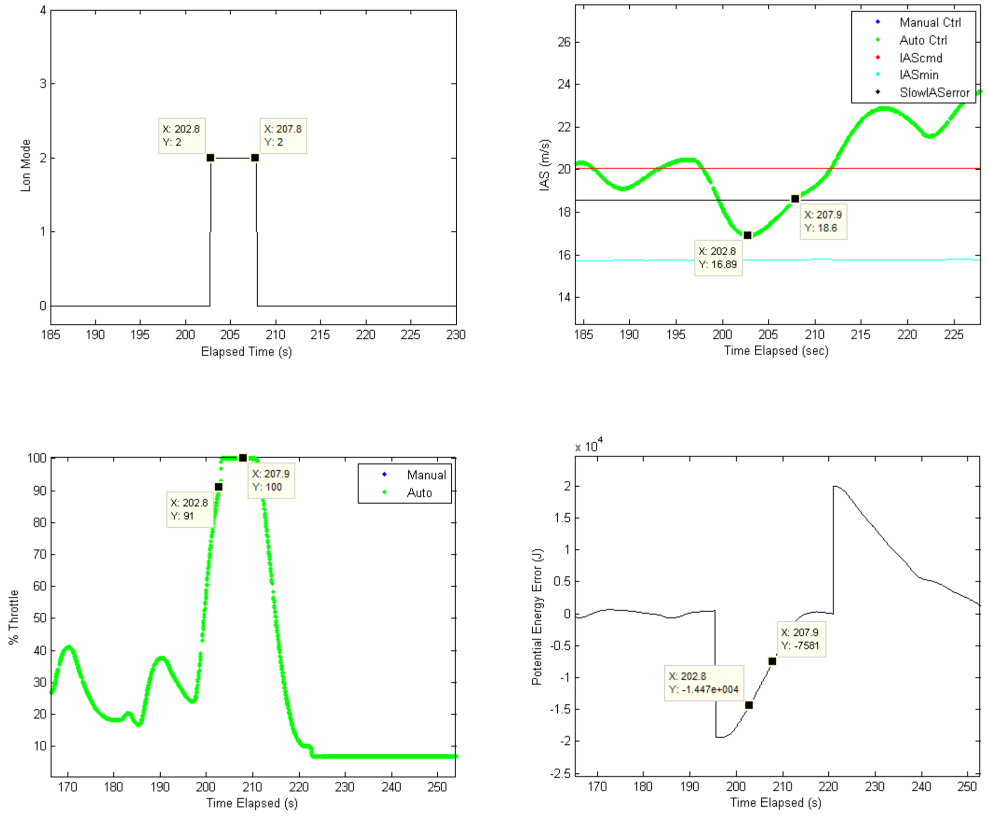
Slow IAS error Threshold = 1.5 m/s

$$C_{L\,\max nom} = 1.80 \;\Rightarrow\; IAS_{min} = 15.69\, m/s$$

Limits:

Climb Max Fraction = 0.25

Results:



The figure shows that the autopilot switched to Lon Mode 2 after violating the Slow IAS error Threshold and at about 91% of Throttle Max. Once again the autopilot switched back to Altitude Control just as the indicated airspeed climbed above the Slow IAS error Threshold. The figure shows the Potential Energy error had decreased dramatically again as well.

Conclusions:

1) The autopilot switches to Slow Airspeed Mode (Lon Mode 2) when **both** of the following conditions are true:

   a) IAS < IAS_{min} or IAS < Slow IAS error Threshold

   b) Throttle >= 90% of Throttle Max

2) The condition of VRate Error being within 10% of the allowable VRate range is no longer a factor for returning to Lon Mode 0.

3) The autopilot will return to Altitude Mode (Lon Mode 0) when **both** of the following conditions are true:

   a) IAS >IAS_{min} or IAS > Slow IAS error Threshold

   b) Altitude error has significantly decreased (enough to cause the Throttle integrator to lower the Throttle trim).

5.1.4. Lon Mode 3 (Fast Airspeed Mode) Experiments

According to "Tuning piccolo control laws 2.0.x" there are several factors that can contribute to the autopilot deciding to switch into airspeed control. One of the conditions concerns flying too fast and reads as follows:

- The airspeed has exceeded the commanded speed by the "Fast IAS error Threshold" and the throttle integrator is less than 5% of full throttle. This indicates that the airframe cannot remove energy fast enough under the current conditions (Tuning piccolo control laws 2.0.x 14).

Similar to the slow airspeed mode the condition stated is for an older version of the piccolo software so it cannot be trusted word for word. The condition indicates that flying too fast, above the "Fast IAS error Threshold" and possibly flying above "IAS_{max}", can cause the autopilot to switch into airspeed control.

The document also asserts the following in regards to exiting airspeed control: "Once the autopilot has switched into airspeed control it will remain there until the conditions given above are removed and when the error in vertical rate has become less than 10% of the allowable vertical rate command" (Tuning piccolo control laws 2.0.x 14).

I simulated a couple of steep descents to determine what Lon Mode corresponded with flying too fast. The descents were setup to be steep enough that the aircraft would violate limits concerning maximum airspeed. I found that the autopilot uses Lon Mode 3 when flying too fast. To better understand Lon Mode 3 I ended up simulating a total of four descents in a hardware in the loop simulation. The descents were setup between waypoints of two different flight plans so that the autopilot would descend as steeply as allowed by $\mathrm{VRate}_{\mathrm{max}}$. In all four descents I analyzed VRate, IAS, Lon Mode, and Throttle. I looked for correlations in each climb that would indicate:

a) What causes the autopilot to switch to Lon Mode 3

b) What causes the autopilot to switch back to Lon Mode 0

c) Does the return condition of VRate Error being within the allowable vertical rate range still apply?

In each descent all the longitudinal gains were set to default values unless otherwise specified. For all the descents I also used a manual IAS command because of a weird condition called IAS "float". There appears to be some extra function that attempts to delay switching to Airspeed Control because the IAS command will automatically increase in some descents causing the Fast IAS error Threshold to increase. I noticed a section in the Release Notes that detailed removing IAS float from climb conditions because it was not function appropriately, so that would explain why I did not experience any IAS float during my simulated Climbs.

Similarly to Lon Mode 2 I analyzed the Potential and Kinetic Energy errors. The logic for Lon Mode 3 should be different since the autopilot is concerned about losing Energy rather than adding Energy; therefore, I looked for the possibility of the Potential Energy error causing the Energy Control to want to increase Throttle.

Descent 1:

For the first descent I wanted to test violating the Fast IAS error Threshold. I set IAS_{max} to a very high value to eliminate that possible condition.

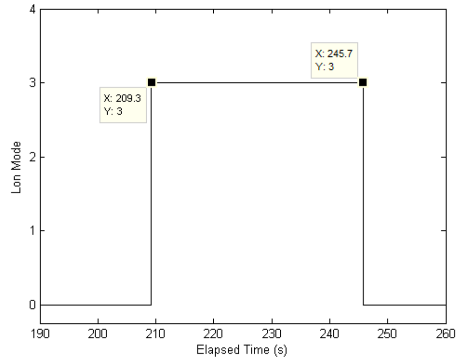
Gains:

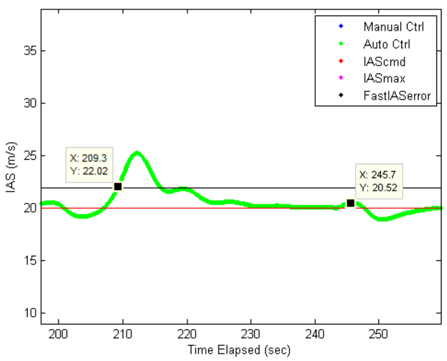
Fast IAS Error Threshold = 2 m/s   Manual IAS_{cmd} = 20 m/s

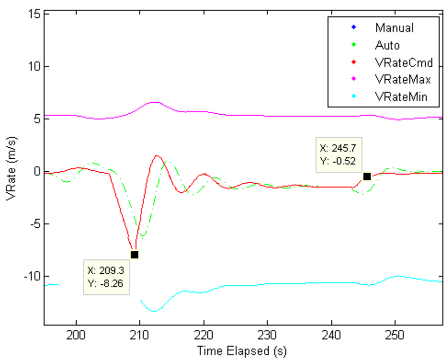
Limits:

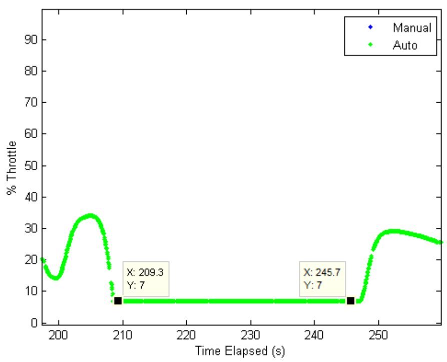
Descent Max Fraction = 0.50 IAS_{max} = 60 m/s   Throttle Min = 7%

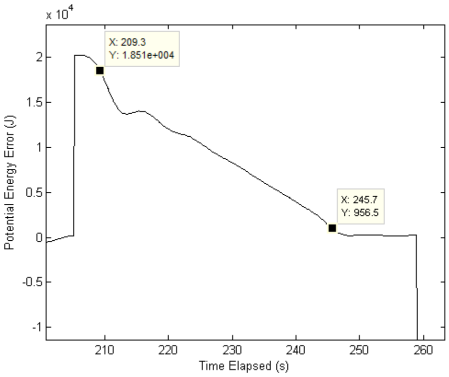
Results:











The figure above shows that the autopilot switched to Fast Airspeed Mode as soon as the indicated airspeed reached the Fast IAS error Threshold. The throttle at the time of the switch was equal to Throttle Min (7%). The vertical rate began to climb to a positive vertical rate in Fast Airspeed Mode; thus, the plane pitched up and decreased its descent in order to lose energy. The autopilot didn't return to Altitude Control until well after the indicated airspeed had fallen below the Fast IAS error Threshold. The figure also shows that the Energy Error had decreased dramatically just prior to returning to Lon Mode 0. Additionally the figure shows that the return to Lon Mode 0 occurred just before the autopilot commanded an increase in Throttle. The vertical rate error was 8.5% of the allowable vertical rate range at the time of return to Altitude Control. The vertical rate error was nearly zero for 5 seconds before hand.

$$VRate_{max} = 5.39\frac{m}{s} \quad VRate_{min} = \ -10.78 \ \ Allowable\ Range = 5.39 + 10.78 = 16.17\frac{m}{s}$$

$$VRate = -1.93\frac{m}{s} \quad VRate_{cmd} = -0.55 \ \ VRate\ Error = -1.93 - -0.55 = -1.38\, m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.08534$$

<u>Descent 2:</u>

For the second descent I increased the Fast IAS error Threshold to observe if the autopilot behaved differently than the results from Descent 1.

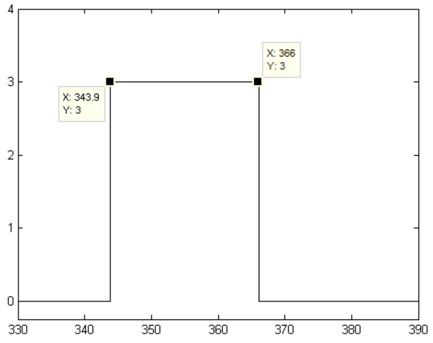
Gains:

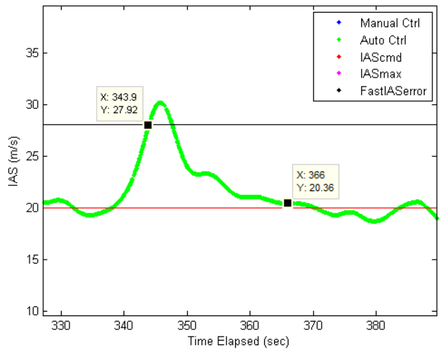
Fast IAS Error Threshold = 8 m/s   Manual IAS_{cmd} = 20 m/s

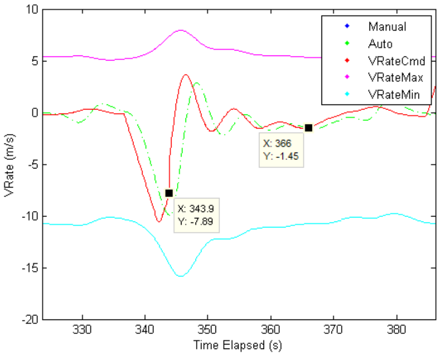
Limits:

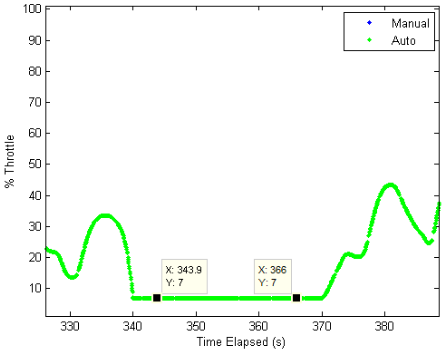
Descent Max Fraction = 0.50 IAS_{max} = 60 m/s   Throttle Min = 7%

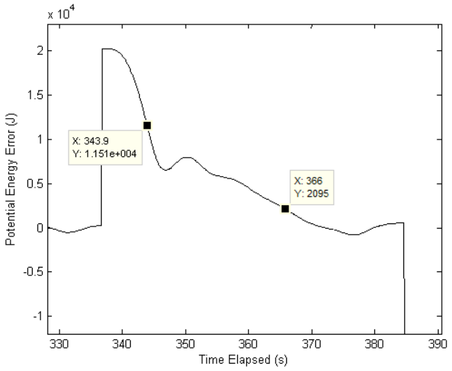
Results:











The figure above shows that the autopilot switched to Fast Airspeed Mode as soon as the indicated airspeed reached the Fast IAS error Threshold. The throttle at the time of the switch was equal to Throttle Min (7%). The vertical rate had already begun to climb to a positive vertical rate prior to Fast Airspeed. The vertical rate command increased the slope to climb more quickly to a positive vertical rate once the autopilot entered Fast Airspeed Mode. The autopilot didn't return to Altitude Control again until well after the indicated airspeed had fallen below the Fast IAS error Threshold. The figure also shows that the Energy Error had decreased dramatically just prior to returning to Lon Mode 0. Additionally the figure shows that the return to Lon Mode 0 occurred before the autopilot commanded an increase in Throttle. The vertical rate error was 0.187% of the allowable vertical rate range at the time of return to Altitude Control.

$$VRate_{max} = 5.35\frac{m}{s} \quad VRate_{min} = \ -10.70 \ \ Allowable\ Range = 5.35 + 10.70 = 16.05\frac{m}{s}$$

$$VRate = -1.45\frac{m}{s} \quad VRate_{cmd} = -1.42 \ \ VRate\ Error = -1.45 - -1.42 = -0.03\ m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.00187$$

Descent 3:

For the third descent I wanted to test how the autopilot would react if the aircraft violated IASmax.

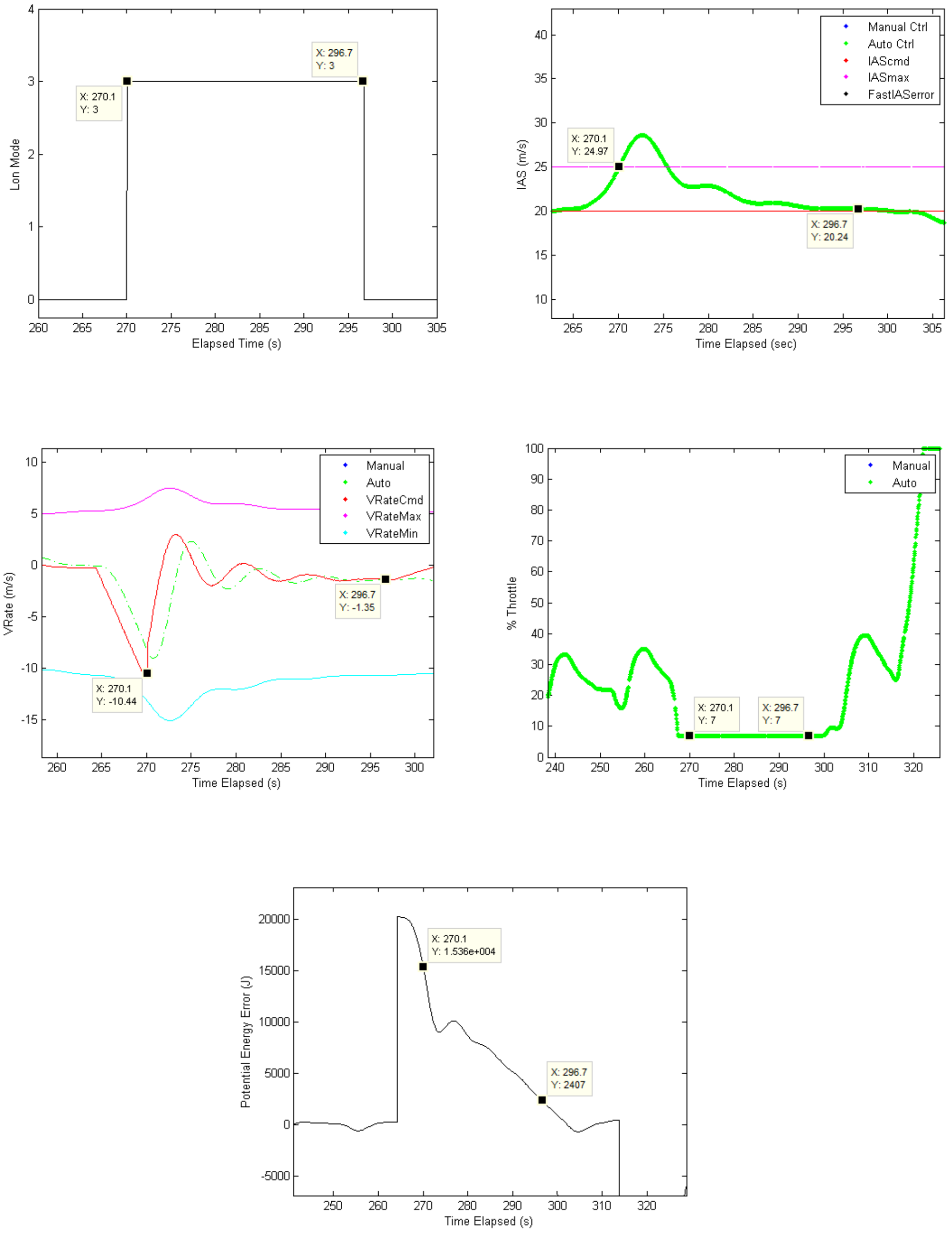
Gains:

Fast IAS Error Threshold = 30 m/s Manual IAS$_{cmd}$ = 20 m/s

Limits:

Descent Max Fraction = 0.50 IAS$_{max}$ = 25 m/s Throttle Min = 7%

Results:



The figure above shows that the autopilot switched to Fast Airspeed Mode as soon as the indicated airspeed reached IAS$_{max}$. The throttle at the time of the switch was equal to Throttle Min (7%). The autopilot didn't return to Altitude Control again until well after the indicated airspeed had fallen below IAS$_{max}$. The figure also shows that the Energy Error had decreased dramatically just prior to returning to Lon Mode 0. Additionally the figure shows that the return to Altitude Control occurred before the autopilot commanded an increase in Throttle. The vertical rate error was 0.188% of the allowable vertical rate range at the time of return to Altitude Control.

$$VRate_{max} = 5.32\frac{m}{s} \quad VRate_{min} = \ -10.64 \ \ Allowable\ Range = 5.32 + 10.64 = 15.96\frac{m}{s}$$

$$VRate = -1.32\frac{m}{s} \quad VRate_{cmd} = -1.35 \ \ VRate\ Error = -1.32 - -1.35 = 0.03\ m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.00188$$

<u>**Descent 4:**</u>

For the fourth descent I decreased the Descent Max Fraction so the descent would be shallower to observe if the conditions of switching to Lon Mode 3 were different than Descent 3.

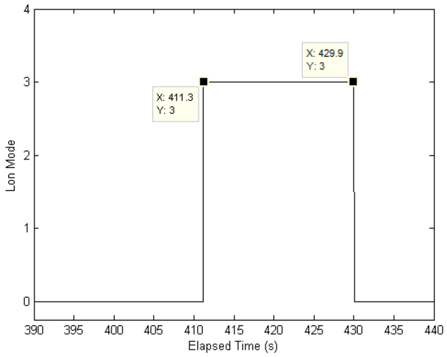
Gains:

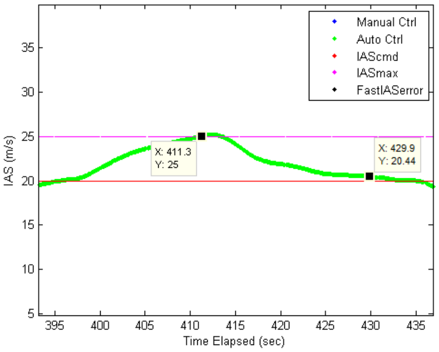
Fast IAS Error Threshold = 30 m/s   Manual IAS_{cmd} = 20 m/s

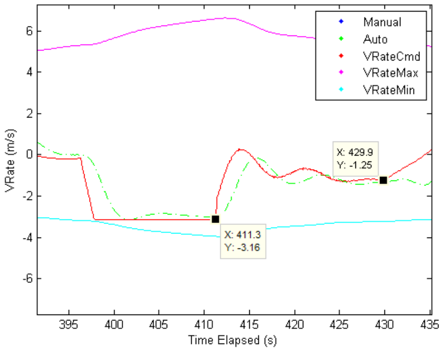
Limits:

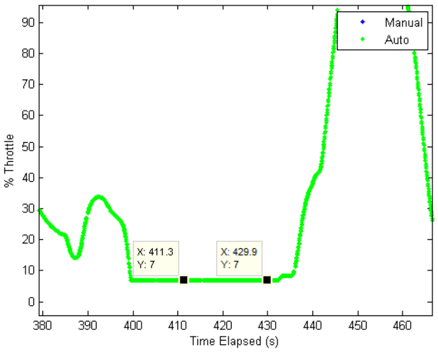
Descent Max Fraction = 0.15 IAS_{max} = 25 m/s   Throttle Min = 7%

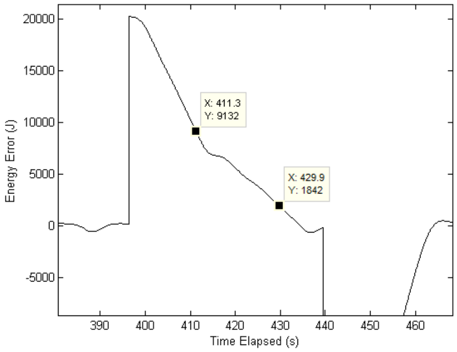
Results:











Similar to the last descent the autopilot switched to Fast Airspeed Mode as soon as the indicated airspeed reached IAS$_{max}$. The throttle at the time of the switch was equal to Throttle Min (7%). The autopilot didn't return to Altitude Control again until well after the indicated airspeed had fallen below IAS$_{max}$. The figure above shows that the Energy Error had decreased just prior to the return to Lon Mode 0. The figure also shows that the return to Altitude Control occurred before the autopilot commanded an increase in Throttle. The vertical rate error was 0.248% of the allowable vertical rate range at the time of return to Altitude Control.

$$VRate_{max} = 5.37\frac{m}{s} \quad VRate_{min} = \ -10.74 \ \ Allowable\ Range = 5.37 + 10.74 = 16.11\frac{m}{s}$$

$$VRate = -1.29\frac{m}{s} \quad VRate_{cmd} = -1.25 \ \ VRate\ Error = -1.29 - -1.25 = -0.04\,m/s$$

$$\frac{abs(VRate\ Error)}{Allowable\ Range} = 0.00248$$

Conclusions:

1) The autopilot will switch to Fast Airspeed Mode (Lon Mode 3) when both of the following are true:

    a) $IAS > IAS_{max}$ or IAS > Fast IAS error Threshold

    b) Throttle = Throttle Min

2) The autopilot will return to Altitude Control when both of the following are true:

    a) $IAS < IAS_{max}$ or IAS < Fast IAs error Threshold

    b) Altitude error has significantly decreased

3) The condition of VRate Error being within 10% of the allowable VRate range is not a factor for returning to Lon Mode 0.

APPENDIX E

MATLAB CODE

1. CreatePiccoloMatFile

```
%By Anton Mornhinweg
%Uses Cloud Cap's 'loadlogfilepiccolo.m' script to import data from a
%telemetry log file into a matlab workspace structure with a variable
for
%each column of data in the log file.

%User select Piccolo .log file
[filename,pathname]=uigetfile({'*.log','Piccolo Log Files
(*.log)'},'Select a Piccolo Log File');

%If statement prevents errors if the user clicks cancel on uigetfile
if pathname ~= 0

    %assigns the variables output by 'loadlogfilepiccolo.m' to the
    %structure 'dat'.
    dat = loadlogfilepiccolo([pathname filename]);

    %creates the variable 'tClock' to represent the piccolo time in
seconds
    tClock = dat.Clock/1000;

    %Allows the user to name the mat file
    fname = char(inputdlg('What do you want to name the mat file?'));

    %If statement saves the mat file only if fname is not empty
    if ~isempty(fname)

    %Saves the workspace as a mat file in the same location as the log
file
    %Only saves the 'dat' structure and 'tClock' variable.
    save([pathname fname],'dat','tClock')

    end

    %clears the workspace
    clear

else
```

```
    clear

end
```

2. AnalyzePiccolo

```
%By Anton Mornhinweg
%This gui uses the .mat workspace file created from
'loadpiccolologfile.m' provided by Cloud
%Cap. Its a userinterface for plotting various telemetry data.
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = AnalyzePiccolo(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AnalyzePiccolo_OpeningFcn, ...
                   'gui_OutputFcn',  @AnalyzePiccolo_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before AnalyzePiccolo is made visible.
function AnalyzePiccolo_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for AnalyzePiccolo
handles.output = hObject;

%Defaults units to ft, ft/s, seconds, degrees, lb
handles.TimeScale = '(s)';
handles.TimeScaleValue = 1;
handles.DistanceValue = 0.3048;
handles.DistanceUnits = ' (ft)';
handles.SpeedValue = 0.51444;
handles.SpeedUnits = ' knots';
handles.DeflectionUnits = ' (deg)';
handles.DeflectionValue = 180/pi;
handles.AccelValue = 1;
handles.AccelUnits = ' (m/s2)';
handles.InertialValue = 180/pi;
handles.InertialUnits = ' (deg)';
handles.TempUnits = ' (C)';
handles.MassUnits = ' (lb)';
handles.MassValue = 0.4535924;
handles.mass = '';
handles.Sw = '';
handles.ElevRange = '';
handles.CtrlRoom = [36.162476;-96.836197];
handles.CtrlTower = [36.162587;-96.836087];
handles.UnMix1 = '';
handles.UnMix2 = '';
handles.MotorType = 'Gas';
handles.time = 'Piccolo';
handles.TakeoffTimeVal = '';
handles.TimeLabel = 'Piccolo Time';
handles.StartSec = '';
handles.EndSec = '';
handles.StartMin = '';
handles.EndMin = '';
handles.ShowWaypoint = 'no';

%Turns on/off radiobuttons that correspond to the defaul units
%These have to be defined everytime the gui is opened because if the
user
%enters a new name for a surface the gui saves itself which can change
%the default saved status of the radio buttons
set(handles.TASmps,'Value',0)
set(handles.Minutes,'Value',0)
set(handles.DistanceMeters,'Value',0)
set(handles.DeflectionDeg,'Value',1)
set(handles.accelms2,'Value',1)
set(handles.InertialDeg,'Value',1)
set(handles.Celsius,'Value',1)
set(handles.TASmph,'Value',0)
set(handles.TASknots,'Value',1)
set(handles.TASfps,'Value',0)
set(handles.Seconds,'Value',1)
set(handles.DistanceFeet,'Value',1)
set(handles.DeflectionRad,'Value',0)
set(handles.accelg,'Value',0)
set(handles.accelfts2,'Value',0)
set(handles.InertialRad,'Value',0)
set(handles.Fhrnht,'Value',0)
set(handles.MassInput,'String','')
set(handles.SwInput,'String','')
set(handles.UnMix1Input,'String','')
set(handles.UnMix2Input,'String','')
set(handles.Masskg,'Value',0)
set(handles.Masslb,'Value',1)
set(handles.MassN,'Value',0)
set(handles.Electric,'Value',0)
set(handles.Gas,'Value',1)
set(handles.PiccoloTime,'Value',1)
set(handles.TakeoffTime,'Value',0)
set(handles.StartSec,'String','')
set(handles.EndSec,'String','')
set(handles.StartMin,'String','')
set(handles.EndMin,'String','')
set(handles.CLendSec,'String','')
set(handles.CLstartSec,'String','')
set(handles.CLendMin,'String','')
set(handles.CLstartMin,'String','')
set(handles.Waypoint,'Value',0)

%saves the handle variables values
guidata(hObject, handles);

%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument
Error!');
        return;
    end
end

%Launches the load listbox function
load_listbox(initial_dir,handles)


% --- Outputs from this function are returned to the command line.
function varargout = AnalyzePiccolo_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Listbox Functions

```
function load_listbox(dir_path,handles)
```

```
%The listbox is designed to dislplay piccolo mat files in the current
%directory that can be selected to load data from.

%Clears the listbox in case it contains strings that were saved in the
gui
set(handles.listbox1, 'String', '');

%Defines the handles associated with loading mat files into the listbox
handles.file_names = {''};
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles)

handles.flightfolder = dir_path;

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

%Loop searches for piccolo mat files in the current directory
count = 0; %count adds each time the loop skips a file
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %Looks for the variable 'tClock'
        if ~isempty(whos('-file',name,'tClock*'))

            %Assigns the filename to the listbox by listbox index
number
            handles.file_names(i-count,1)=sorted_names(i);
            handles.sorted_index(i-count,1) = sorted_index(i);

        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

if ~isfield(handles,'file_names')

    msgbox('There arent any .mat plotpiccolo files in the current
directory')
```

```
else

    %Sets the directory to the current folder
    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)

    %Assigns the piccolo mat file names to the listbox
    set(handles.listbox1,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)

%Selects first listbox item as the current piccolo mat file
file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};

%Removes the extension from the file name so that the name of all the
%plots can begin with the flight name only, no extension.
filename4 = handles.flight;
remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

%Saves the selected file into a handle 'handles.flight'
if ~strcmp(handles.flightname,'')

    index_selected = get(handles.listbox1,'Value');
    file_list = get(handles.listbox1,'String');
    handles.flightfilename = file_list{index_selected};

    %Flight handle includes the directory path and filename
    handles.flight = [handles.flightfolder '\' handles.flightfilename];

    %Saves just the filename, w/o extension, to a handle
'handles.flightname'
    filename4 = handles.flightfilename;
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```

Telemetry Plots

```
% --- Executes on button press in Height.
function Height_Callback(hObject, eventdata, handles)
% hObject    handle to Height (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Height is the GPS altitude

%If statement prevents the user from attempting to plot without a mat
file
%selected
if strcmp(handles.flight,'')

    msgbox('Select a plotpiccolo .mat file')

else

%Loads data from the piccolo mat file
load(handles.flight)

%Sets up having different series for autopilot ctrl being on or off
apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

%Determines what time scale to use
if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%Includes the target waypoints in the plot as vectors if radio button
is
%selected
if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    %Creates waypoint vectors to be 2 points below and 2 points above
the
    %telemetry value
    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            %x and y vectors store the location to print the waypoint#
            %'Waypoint' stores the new target waypoint number
            count = count + 1;
            x(count,1) = (tClock(i)/handles.TimeScaleValue-
1/handles.TimeScaleValue);
            y(count,1) = (dat.LoopTarget1(i)+3)/handles.DistanceValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            %Loop creates 5 points at the same time span
            for j = 1:5

                %'turn' is the y values for new waypiont data points
                %'turntime' is the x values for new waypoitn data
points
                turn(counter+j,1) = dat.LoopTarget1(i)-3 + j;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')

    %Plots the new waypoint vectors

plot(turntime/handles.TimeScaleValue,turn/handles.DistanceValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end

 %Plots the telemetry data versus time with the appropriate scale and
units

plot(tClock(apon)/handles.TimeScaleValue,dat.Height(apon)/handles.Dista
nceValue,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Height(apoff)/handles.Dis
tanceValue,'b.')
 hold on

plot(tClock/handles.TimeScaleValue,dat.LoopTarget1/handles.DistanceValu
e,'r')
 set(gcf,'Name',[handles.flightname 'GPS Altitude']) %Name of the
figure includes the flight name
 ylabel('GPS Altitude','string',['GPS Altitude' handles.DistanceUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

%Adds Legend if Legend radio button is selected
legendval = get(handles.Legend,'Value');
if isequal(legendval,1)
```

```
   h1=plot(0,0,'b.');
   h2=plot(0,0,'g.');
   h3=plot(0,0,'r.');

   %Adds legend for new waypoints
   if strcmp(handles.ShowWaypoint,'yes')


h4=plot(turntime(1)/handles.TimeScaleValue,turn(1)/handles.DistanceValu
e,'.k');
       legend([h1 h2 h3 h4],'Manual Ctrl','Auto
Ctrl','AltCmd','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

   else

      legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','AltCmd')

   end

end
end

% --- Executes on button press in TAS.
function TAS_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.TAS(i)+2)/handles.SpeedValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.TAS(i)-1.5 + j/2;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')

    plot(turntime/handles.TimeScaleValue,turn/handles.SpeedValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end


plot(tClock(apon)/handles.TimeScaleValue,dat.TAS(apon)/handles.SpeedVal
ue,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.TAS(apoff)/handles.SpeedV
alue,'b.')
 set(gcf,'Name',[handles.flightname 'TAS'])
 ylabel('TAS','string',['TAS' handles.SpeedUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');

    if strcmp(handles.ShowWaypoint,'yes')


h3=plot(turntime(1)/handles.TimeScaleValue,turn(1)/handles.DistanceValu
e,'.k');
       legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

   else

    legend([h1 h2],'Manual Ctrl','Auto Ctrl')

    end
```

```
end
end

% --- Executes on button press in IAS.
function IAS_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else

load(handles.flight)
apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%IAS is not recorded, it has to be calculated
%Piccolo calculates IAS using measured dynamic pressure and sea
%level density
IAS = zeros(length(dat.Dynamic),1);
IAS(dat.Dynamic>0) = sqrt(2*dat.Dynamic(dat.Dynamic>0)/1.225); %Avoids
sqrt of negative values

%Asks variables necessary to calculate minimum airspeed
CLmaxnom = str2double(inputdlg('If IAS min is not desired then leave
blank otherwise, what is CL Max Nom?','Minimum IAS'));
CLmax = str2double(inputdlg('If IAS min is not desired then leave blank
otherwise, what is CL Max?','Minimum IAS During Land'));

%If CLmaxnom is left blank the rest of the IAS min variables are
skipped
if ~isnan(CLmaxnom)

    Sw = str2double(inputdlg('What is the wing area in
ft^2?'))*0.3048^2;
    EmptyWeight = str2double(inputdlg('What is the empty weight in
lb?'));
    mass = dat.Fuel + EmptyWeight*0.4535924; %Converts units to kg

    IASmin1 = sqrt(2*mass*9.81./(CLmaxnom*Sw*1.225)*1.1);

end

%If CLmaxnom is left blank the rest of the IAS min variables are
skipped
if ~isnan(CLmax)

    %Sw = double(Sw);
    %EmptyWeight = double(EmptyWeight);

    if exist('Sw','var')

        if ischar(Sw)

        Sw = str2double(inputdlg('What is the wing area in
ft^2?'))*0.3048^2;

        end

    else

        Sw = str2double(inputdlg('What is the wing area in
ft^2?'))*0.3048^2;

    end

    if exist('EmptyWeight','var')

        if ischar(EmptyWeight)

        EmptyWeight = str2double(inputdlg('What is the empty weight in
lb?'));

        end

    else

        EmptyWeight = str2double(inputdlg('What is the empty weight in
lb?'));

    end

    mass = dat.Fuel + EmptyWeight*0.4535924; %Converts units to kg

    IASmin2 = sqrt(2*mass*9.81./(CLmax*Sw*1.225)); %m/s

end

%Asks variabels necessary to display max limits on IAS
FIET = str2double(inputdlg('What is fast IAS error threshold in
knots?','FIET'));
IASmax = str2double(inputdlg('What is max IAS in knots?','IASmax'));

%Converts the units to m/s
if ~isnan(IASmax)

    IASmax = IASmax * 0.514;

end

if ~isnan(FIET)

    FIET = FIET * 0.514;

end
```

```
figure

plot(tClock(apon)/handles.TimeScaleValue,IAS(apon)/handles.SpeedValue,'
-g')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,IAS(apoff)/handles.SpeedValue
,'b.')
 hold on

plot(tClock/handles.TimeScaleValue,dat.LoopTarget0/handles.SpeedValue,'
-r')

 %Following conditions only plot the limits that were given variables
for
 if ~isnan(CLmaxnom)
 hold on
 plot(tClock/handles.TimeScaleValue,IASmin1/handles.SpeedValue,'-c')
 end

 if ~isnan(CLmax)
 hold on
 plot(tClock/handles.TimeScaleValue,IASmin2/handles.SpeedValue,'-y')
 end

 if ~isnan(IASmax)
 hold on
 plot(tClock/handles.TimeScaleValue,IASmax/handles.SpeedValue,'-m')
 end

 if ~isnan(FIET)
 hold on

plot(tClock/handles.TimeScaleValue,(dat.LoopTarget0+FIET)/handles.Speed
Value,'-k')
 end

 set(gcf,'Name',[handles.flightname 'IAS'])
 ylabel('IAS','string',['IAS' handles.SpeedUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    h3=plot(0,0,'r.');
    h4=plot(0,0,'c.');
    h5=plot(0,0,'y.');
    h6=plot(0,0,'m.');
    h7=plot(0,0,'k.');

    legend([h1 h2 h3 h4 h5 h6 h7],'Manual Ctrl','Auto Ctrl',
'IAScmd','CLmaxnom','CLmax','IASmax','FIET')

end
```

```
end

% --- Executes on button press in StaticPressure.
function StaticPressure_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else

load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
 plot(tClock(apon)/handles.TimeScaleValue,dat.Static(apon),'-g')
 hold on
 plot(tClock(apoff)/handles.TimeScaleValue,dat.Static(apoff),'-b')
 set(gcf,'Name',[handles.flightname 'Static Pressure'])
 ylabel('a','string','Static Pressure (Pa)')
 xlabel('Time Elapsed','string',handles.TimeScale)

end

% --- Executes on button press in DynamicPressure.
function DynamicPressure_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
 plot(tClock(apon)/handles.TimeScaleValue,dat.Dynamic(apon),'-g')
 hold on
 plot(tClock(apoff)/handles.TimeScaleValue,dat.Dynamic(apoff),'-b')
 set(gcf,'Name',[handles.flightname 'Dynamic Pressure'])
 ylabel('a','string','Dynamic Pressure')
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in AGL.
function AGL_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.AGL(apon)/handles.Distance
Value,'-g')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.AGL(apoff)/handles.Distan
ceValue,'b.')
 set(gcf,'Name',[handles.flightname 'AGL'])
 ylabel('Altitude','string',['AGL Altitude' handles.DistanceUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'.b');
    h2=plot(0,0,'.g');
    legend([h1 h2],'Manual Ctrl','Auto Ctrl')

 end
end

% --- Executes on button press in BaroAlt.
function BaroAlt_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
```

```
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = 
(tClock(i)/handles.TimeScaleValue/handles.TimeScaleValue);
            y(count,1) = (dat.LoopTarget1(i)+3)/handles.DistanceValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.LoopTarget1(i)-3 + j;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')


plot(turntime/handles.TimeScaleValue,turn/handles.DistanceValue,'.k')
    set(gcf,'DefaulttextClipping','on')
    text(x,y,Waypoint)
    hold on

 end
```

```
plot(tClock(apon)/handles.TimeScaleValue,dat.Alt(apon)/handles.Distance
Value,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Alt(apoff)/handles.Distan
ceValue,'b.')
 hold on

plot(tClock/handles.TimeScaleValue,dat.LoopTarget1/handles.DistanceValu
e,'r')
 set(gcf,'Name',[handles.flightname 'Barometer Altitude'])
 ylabel('Baro Altitude','string',['Baro Altitude'
handles.DistanceUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    h3=plot(0,0,'r.');
    if strcmp(handles.ShowWaypoint,'yes')

h4=plot(turntime(1)/handles.TimeScaleValue,turn(1)/handles.DistanceValu
e,'.k');
    legend([h1 h2 h3 h4],'Manual Ctrl','Auto
Ctrl','AltCmd','NewWaypointTargeted')
    clear('x','y','turn','turntime','Waypoint')
    else
    legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','AltCmd')
    end

 end
end

% --- Executes on button press in GroundStationAlt.
function GroundStationAlt_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end
```

```
figure

plot(tClock(apon)/handles.TimeScaleValue,dat.GSHeight(apon)/handles.Dis
tanceValue,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.GSHeight(apoff)/handles.D
istanceValue,'b.')
 set(gcf,'Name',[handles.flightname 'Ground Station Altitude (GPS)'])
 ylabel('GS Altitude','string',['GS Altitude' handles.DistanceUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end


% --- Executes on button press in GPSvBaro.
function GPSvBaro_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Alt(apon)/handles.Distance
Value,'b.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Alt(apoff)/handles.Distan
ceValue,'b.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Height(apoff)/handles.Dis
tanceValue,'r.')
 hold on

plot(tClock(apon)/handles.TimeScaleValue,dat.Height(apon)/handles.Dista
nceValue,'r.')
 set(gcf,'Name',[handles.flightname 'GPS vs. Barometer'])
 ylabel('Altitude','string',['Altitude' handles.DistanceUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'r.');
    legend([h1 h2],'Baro Alt','GPS Alt')

 end
end

% --- Executes on button press in ax.
function ax_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Xaccel(apon)/handles.Accel
Value,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Xaccel(apoff)/handles.Acc
elValue,'b.')
 set(gcf,'Name',[handles.flightname 'ax'])
 ylabel('a','string',['ax' handles.AccelUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in ay.
function ay_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')
```

```
    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Yaccel(apon)/handles.Accel
Value,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Yaccel(apoff)/handles.Acc
elValue,'b.')
 set(gcf,'Name',[handles.flightname 'ay'])
 ylabel('a','string',['ay' handles.AccelUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in az.
function az_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.Zaccel(i)+3)/handles.AccelValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};
            for j = 1:5

                turn(counter+j,1) = dat.Zaccel(i)-3 + j;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')

    plot(turntime/handles.TimeScaleValue,turn/handles.AccelValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end


plot(tClock(apon)/handles.TimeScaleValue,dat.Zaccel(apon)/handles.Accel
Value,'.-g')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Zaccel(apoff)/handles.Acc
elValue,'b.')
 set(gcf,'Name',[handles.flightname 'az'])
 ylabel('a','string',['az' handles.AccelUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in ag.
function ag_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;
```

```
end

figure
    subplot(3,1,1)

plot(tClock(apon)/handles.TimeScaleValue,dat.Xaccel(apon)/handles.Accel
Value,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Xaccel(apoff)/handles.Acc
elValue,'b.')
    ylabel('a','string',['ax' handles.AccelUnits])

    subplot(3,1,2)

plot(tClock(apon)/handles.TimeScaleValue,dat.Yaccel(apon)/handles.Accel
Value,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Yaccel(apoff)/handles.Acc
elValue,'b.')
    ylabel('a','string',['ay' handles.AccelUnits])

    subplot(3,1,3)

plot(tClock(apon)/handles.TimeScaleValue,dat.Zaccel(apon)/handles.Accel
Value,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Zaccel(apoff)/handles.Acc
elValue,'b.')
    set(gcf,'Name',[handles.flightname 'ax, ay, az'])
    ylabel('a','string',['az' handles.AccelUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in PQR.
function LMN_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end
```

```
figure
    subplot(3,1,1)

plot(tClock(apon)/handles.TimeScaleValue,dat.Roll(apon)*handles.Inertia
lValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Roll(apoff)*handles.Inert
ialValue,'b.')
    ylabel('a','string',['Roll' handles.InertialUnits])

    subplot(3,1,2)

plot(tClock(apon)/handles.TimeScaleValue,dat.Pitch(apon)*handles.Inerti
alValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Pitch(apoff)*handles.Iner
tialValue,'b.')
    ylabel('a','string',['Pitch' handles.InertialUnits])

    subplot(3,1,3)

plot(tClock(apon)/handles.TimeScaleValue,dat.Yaw(apon)*handles.Inertial
Value,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Yaw(apoff)*handles.Inerti
alValue,'b.')
    set(gcf,'Name',[handles.flightname 'Roll, Pitch, Yaw'])
    ylabel('a','string',['Yaw' handles.InertialUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in PQR.
function PQR_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end
```

```
%Instead of making new radio button for units of PQR I just force them
to
%be the equivalent of the inertial units, rad or deg.
if strcmp(handles.InertialUnits,' (rad)')

    units = '(rad/s)';

else

    units = '(deg/s)';

end

figure
    subplot(3,1,1)

plot(tClock(apon)/handles.TimeScaleValue,dat.P(apon)*handles.InertialVa
lue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.P(apoff)*handles.Inertial
Value,'b.')
    ylabel('a','string',['Roll rate ' units])

    subplot(3,1,2)

plot(tClock(apon)/handles.TimeScaleValue,dat.Q(apon)*handles.InertialVa
lue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Q(apoff)*handles.Inertial
Value,'b.')
    ylabel('a','string',['Pitch rate ' units])

    subplot(3,1,3)

plot(tClock(apon)/handles.TimeScaleValue,dat.R(apon)*handles.InertialVa
lue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.R(apoff)*handles.Inertial
Value,'b.')
    set(gcf,'Name',[handles.flightname 'Roll, Pitch, Yaw Rates'])
    ylabel('a','string',['Yaw Rate ' units])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in GroundSpeed.
function GroundSpeed_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.GroundSpeed(apon)/handles.
SpeedValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.GroundSpeed(apoff)/handle
s.SpeedValue,'b.')
    set(gcf,'Name',[handles.flightname 'Ground Speed (GPS)'])
    ylabel('Ground Speed','string',['Ground Speed' handles.SpeedUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end
end

% --- Executes on button press in RPM.
function RPM_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else

load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')
```

```
    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.LeftRPM(i)+350);
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.LeftRPM(i)-300 + j*100;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')

    plot(turntime/handles.TimeScaleValue,turn,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end

    plot(tClock(apon)/handles.TimeScaleValue,dat.LeftRPM(apon),'.g')
    hold on
    plot(tClock(apoff)/handles.TimeScaleValue,dat.LeftRPM(apoff),'b.')
    set(gcf,'Name',[handles.flightname ' RPM'])
    ylabel('RPM')
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');

    if strcmp(handles.ShowWaypoint,'yes')


h3=plot(turntime(1)/handles.TimeScaleValue,turn(1)/handles.DistanceValu
e,'.k');
       legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

    else

        legend([h1 h2],'Manual Ctrl','Auto Ctrl')

    end

 end
end

% --- Executes on button press in RPMFiltered.
function RPMFiltered_Callback(hObject, eventdata, handles)

%Plots the RPM variable created by 'RPMFilter.m'

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else    load(handles.flight)

%Checks for the filtered RPM variable 'RPM'
if ~exist('RPM','var')

    msgbox('There is no filtered rpm data. Run "RPMFilter.m"
first','Error')

else

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
    plot(tClock(apon)/handles.TimeScaleValue,RPM(apon),'g.')
    hold on
    plot(tClock(apoff)/handles.TimeScaleValue,RPM(apoff),'b.')
    set(gcf,'Name',[handles.flightname ' Filtered RPM'])
    ylabel('RPM')
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end
 end
end

% --- Executes on button press in Roll.
function Roll_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.Roll(i)+5/180*pi)*handles.InertialValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.Roll(i)-4/180*pi +
j/180*pi+1/180*pi;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')
```

```
plot(turntime/handles.TimeScaleValue,turn*handles.InertialValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end


plot(tClock(apon)/handles.TimeScaleValue,dat.Roll(apon)*handles.Inertia
lValue,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Roll(apoff)*handles.Inert
ialValue,'b.')
 hold on

plot(tClock(apon)/handles.TimeScaleValue,dat.LoopTarget2(apon)*handles.
InertialValue,'.r')
 set(gcf,'Name',[handles.flightname 'Roll'])
 ylabel('a','string',['Roll' handles.InertialUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    h3=plot(0,0,'r.');

    if strcmp(handles.ShowWaypoint,'yes')


h4=plot(turntime(1)/handles.TimeScaleValue,turn(1)*handles.InertialValu
e,'.k');
       legend([h1 h2 h3 h4],'Manual Ctrl','Auto Ctrl','Bank
Cmd','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

    else

        legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','Bank Cmd')

    end

end
end

% --- Executes on button press in Pitch.
function Pitch_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)
```

```
apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;

    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.Pitch(i)+5/180*pi)*handles.InertialValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.Pitch(i)-4/180*pi +
j/180*pi+1/180*pi;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')


plot(turntime/handles.TimeScaleValue,turn*handles.InertialValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end


plot(tClock(apon)/handles.TimeScaleValue,dat.Pitch(apon)*handles.Inerti
alValue,'g.')
```

```
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Pitch(apoff)*handles.Iner
tialValue,'b.')
 set(gcf,'Name',[handles.flightname 'Pitch'])
 ylabel('a','string',['Pitch' handles.InertialUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');

    if strcmp(handles.ShowWaypoint,'yes')


h3=plot(turntime(1)/handles.TimeScaleValue,turn(1)*handles.InertialValu
e,'.k');
       legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

    else

        legend([h1 h2],'Manual Ctrl','Auto Ctrl')

    end

end
end

% --- Executes on button press in Yaw.
function Yaw_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.ShowWaypoint,'yes')

    counter = 0;
    count = 0;
    for i = 2:length(dat.Clock)

        if dat.TrackerTarget(i,1) ~= dat.TrackerTarget(i-1,1)

            count = count + 1;
            x(count,1) = tClock(i)/handles.TimeScaleValue;
            y(count,1) = (dat.Yaw(i)+35/180*pi)*handles.InertialValue;
            Waypoint(count,1) = {dat.TrackerTarget(i)};

            for j = 1:5

                turn(counter+j,1) = dat.Yaw(i)-30/180*pi +
j*10/180*pi+1/180*pi;
                turntime(counter+j,1) = tClock(i);

            end

            counter = counter + 5;

        end
    end
end

figure

 if strcmp(handles.ShowWaypoint,'yes')


plot(turntime/handles.TimeScaleValue,turn*handles.InertialValue,'.k')
    set(gcf,'DefaulttextClipping','on') %limits the text to be
displayed only in the plot area
    text(x,y,Waypoint)
    hold on

 end


plot(tClock(apon)/handles.TimeScaleValue,dat.Yaw(apon)*handles.Inertial
Value,'g.')
 hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Yaw(apoff)*handles.Inerti
alValue,'b.')
 set(gcf,'Name',[handles.flightname 'Yaw'])
 ylabel('a','string',['Yaw' handles.InertialUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');

    if strcmp(handles.ShowWaypoint,'yes')
```

```
h3=plot(turntime(1)/handles.TimeScaleValue,turn(1)*handles.InertialValu
e,'.k');
       legend([h1 h2 h3],'Manual Ctrl','Auto Ctrl','NewWaypointTarget')
       clear('x','y','turn','turntime','Waypoint')

    else

        legend([h1 h2],'Manual Ctrl','Auto Ctrl')

    end

end
end

% --- Executes on button press in Heading.
function Heading_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

%Heading command is recorded as +/- 180 degrees
headingpos=find(dat.LoopTarget4>=0);
headingneg=find(dat.LoopTarget4<0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

 plot(tClock/handles.TimeScaleValue,dat.Yaw*handles.InertialValue,'b.')
 hold on

plot(tClock/handles.TimeScaleValue,dat.Direction*handles.InertialValue,
'g.')
 hold on

 %plot heading command as 0 to 360 degrees

plot(tClock(headingpos)/handles.TimeScaleValue,dat.LoopTarget4(headingp
os)*handles.InertialValue,'r.')
 hold on

plot(tClock(headingneg)/handles.TimeScaleValue,(dat.LoopTarget4(heading
neg)+2*pi)*handles.InertialValue,'r.')
 set(gcf,'Name',[handles.flightname 'Heading'])
 ylabel('a','string',['Heading' handles.InertialUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])
```

```
legendval = get(handles.Legend,'Value');
if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    h3=plot(0,0,'r.');
    legend([h1 h2 h3],'Yaw','Direction','Heading Cmd')

end
end

% --- Executes on button press in MagHdg.
function MagHdg_Callback(hObject, eventdata, handles)
if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure

 plot(tClock/handles.TimeScaleValue,dat.MagHdg*handles.InertialValue,'-
k')
 hold on
 set(gcf,'Name',[handles.flightname 'Magnetometer Heading'])
 ylabel('a','string',['Heading' handles.InertialUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])


end
```

GPS Location Telemetry Plots

```
% --- Executes on button press in LinkPos.
function LinkPos_Callback(hObject, eventdata, handles)

%Plots the lat and lon GPS locations with different colors to represent
%different signal strengths

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

rad2deg = 180/pi;
deg2rad = pi/180;

if strcmp(handles.time,'Piccolo')
```

```
    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
plot(tClock,dat.AckRatio,'.k')
set(gcf,'Name',[handles.flightname 'Piccolo Comms Packet Loss vs
Time'])
xlabel('Piccolo Time (s)')
ylabel('Link')

%Allows user to specify an altitude range for the plot
minalt = str2double(inputdlg('What is minimum altitude for plot?
(ft)'))*0.3048;
maxalt = str2double(inputdlg('What is maximum altitude for plot?
(ft)'))*0.3048;

figure
 latlonfig=gcf;

 %Defines the plot boundaries as the max and min latitude and
longitudes
 %that the aircraft flew and adds a little space to the boundaries
 a1 = min(dat.Lon(find(dat.PosGood ==1)))-0.000004;
 a2 = max(dat.Lon(find(dat.PosGood ==1)))+0.000004;
 b1 = min(dat.Lat(find(dat.PosGood ==1)))-0.000004;
 b2 = max(dat.Lat(find(dat.PosGood ==1)))+0.000004;

 %defines the location of the treeline and airport rd at the OSU UAFS.
 treeline = 96.827858*deg2rad;
 airportrd = 36.159065*deg2rad;

%if statements make sure the max and min axes are big enough to at
least include
%the tree line, airport rd, and the control room
 if abs(a2) > treeline

     a2 = -96.827856*deg2rad;

 end

 if b1 > airportrd

     b1 = airportrd;

 end

 if abs(a1) < abs(handles.CtrlRoom(2,1)*deg2rad)

     a1 = handles.CtrlRoom(2,1)*deg2rad - 0.000007;

 end
 %The following lines were copied from Cloud Cap's 'plotpiccolo.m'
lines
 %427-502 with some modifications
 hold on
 xlabel('Lon')
 ylabel('Lat')
 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);
 grid
 set(gcf,'Name',[handles.flightname 'Piccolo Comms Packet Loss'])

 red=[1 0 0];
 yell = [1 1 0];
 grn=[0 1 0];
 blu = [0 0 1];
 orange = [1 1/2 0];
 lblu = [0 1 1];
 purpl = [3/4 0 1];
 grey=[1/2 1/2 1/2];
 blk=[0 0 0];

 %Plot RSSI as colors..
 if (max(dat.RSSI)-min(dat.RSSI) == 0),
     r1=zeros(length(dat.RSSI),1);
 else
%     r1=(dat.RSSI-max(dat.RSSI))/(max(dat.RSSI)-min(dat.RSSI))+1;
%goes from 0 (rssi -115) to 1 (best RSSI)
    %Color code by AckRatio instead..
    r1=(dat.AckRatio-max(dat.AckRatio))/(max(dat.AckRatio)-
min(dat.AckRatio))+1;   %goes from 0 to 1 (best AckRatio)
 end
 red2=[ones(length(r1),1) zeros(length(r1),1) zeros(length(r1),1)];
 rssicolor=red2+r1*(grn-red);

 %Loop defines colors for different signal strengths
 rssicolor=[];
 for i=1:length(dat.AckRatio),
     AckRatio=dat.AckRatio(i);
     if (AckRatio==100),
         col=grn;
     elseif (AckRatio>=95),
         col=lblu;
     elseif (AckRatio>=90),
         col=blu;
     elseif (AckRatio>=80),
         col=purpl;
     elseif (AckRatio>=70),
         col=red;
     elseif (AckRatio>=55),
         col=grey;
     elseif (AckRatio<55),
         col=blk;
     end
     rssicolor=[rssicolor; col];
 end
```

```
%If there is a time period specified in the CL section the link plot
will
%only plot within that time period.
selectperiod = 'yes'; %Used to determine if a specified range has been
given

if strcmp(handles.StartSec,'')

    selectperiod = 'no';

end

if strcmp(handles.StartMin,'')

    selectperiod = 'no';

end

if strcmp(handles.EndSec,'')

    selectperiod = 'no';

end

if strcmp(handles.EndMin,'')

    selectperiod = 'no';

end

if strcmp(selectperiod,'yes')

 starttime = (handles.StartMin*60+handles.StartSec);
 endtime = (handles.EndMin*60+handles.EndSec);
 range = find(dat.Clock/1000 >= starttime & dat.Clock/1000 <= endtime);

else

 range = 0;
 range(1,1) = 1;
 range(2,1) = length(dat.Clock);

end

if isnan(minalt)

    minalt = 0;

end

if isnan(maxalt)

    maxalt = 3048; %m

end

 for i=range(1,1):range(end,1)
```

```
     if dat.Alt(i) >= minalt && dat.Alt(i) <= maxalt


plot(dat.Lon(i)*rad2deg,dat.Lat(i)*rad2deg,'.','Color',rssicolor(i,:))
        hold on

     end

 end

 if minalt > 0

     title(sprintf(['%5.5g ft < Altitude < %5.5g
ft'],minalt/0.3048,maxalt/0.3048));

 elseif maxalt < 3048

     title(sprintf(['%5.5g ft < Altitude < %5.5g
ft'],minalt/0.3048,maxalt/0.3048));

 end

 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);

 %Defines geo locations that provide a frame of reference for viewing
 %flight data at the OSU UAFS
 plot(handles.CtrlTower(2,1),handles.CtrlTower(1,1),'r*');
 text('Position',[handles.CtrlTower(2,1)-0.0004
handles.CtrlTower(1,1)+0.0002],'string','Ctrl Twr')

 plot(handles.CtrlRoom(2,1),handles.CtrlRoom(1,1),'r*');
 text('Position',[handles.CtrlRoom(2,1)-0.0005 handles.CtrlRoom(1,1)-
0.0002],'string','Ctrl Rm')

 plot([-96.827858 -96.827858],[36.165109 36.159065])
 text('Position',[-96.8278147 36.162385],'string','Tree Line')

 plot([-96.836808 -96.827858],[36.159101 36.159065])
 text('Position',[-96.832164 36.159100],'string','Airport rd')

 plot([-96.835598 -96.835546],[36.161234 36.162863])
 text('Position',[-96.835612 36.161213],'string','Airstrip')

legendval = get(handles.Legend,'Value');
if isequal(legendval,1)
 h1=plot(0,0,'.');
 h2=plot(0,0,'.');
 h3=plot(0,0,'.');
 h4=plot(0,0,'.');
 h5=plot(0,0,'.');
 h6=plot(0,0,'.');
 h7=plot(0,0,'.');

 set(h1,'Color',grn);
 set(h2,'Color',lblu);
 set(h3,'Color',blu);
 set(h4,'Color',purpl);
 set(h5,'Color',red);
 set(h6,'Color',grey);
 set(h7,'Color',blk);

 legend([h1 h2 h3 h4 h5 h6 h7],'Link = 100','Link >= 95','Link >=
90','Link >= 80','Link >= 70', 'Link >= 55', 'Link < 55')
end
end

% --- Executes on button press in ManualPilotConnection.
function ManualPilotConnection_Callback(hObject, eventdata, handles)

%Exactly the same funciton as 'LinkPos' except this function plots the
%manual pilot signal strength which is applicable for systems that
utilize
%the JR manual remote control

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

rad2deg = 180/pi;
deg2rad = pi/180;

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
plot(tClock,dat.PilotPrcnt,'.k')
set(gcf,'Name',[handles.flightname 'JR Signal Strength'])
xlabel('Piccolo Time (s)')
ylabel('JR Signal')

%Allows user to specify an altitude range for the plot
minalt = str2double(inputdlg('What is minimum altitude for plot?
(ft)'))*0.3048;
maxalt = str2double(inputdlg('What is maximum altitude for plot?
(ft)'))*0.3048;

figure
 latlonfig=gcf;

 a1 = min(dat.Lon(find(dat.PosGood ==1)))-0.000004;
 a2 = max(dat.Lon(find(dat.PosGood ==1)))+0.000004;
 b1 = min(dat.Lat(find(dat.PosGood ==1)))-0.000004;
 b2 = max(dat.Lat(find(dat.PosGood ==1)))+0.000004;
 treeline = 96.827858*deg2rad;
 airportrd = 36.159065*deg2rad;

 if abs(a2) > treeline

     a2 = -96.827856*deg2rad;

 end

 if b1 > airportrd

     b1 = airportrd;

 end

 if abs(a1) < abs(handles.CtrlRoom(2,1)*deg2rad)

     a1 = handles.CtrlRoom(2,1)*deg2rad - 0.000007;

 end

 %The following lines were copied from Cloud Cap's 'plotpiccolo.m'
lines
 %427-502 with some modifications
 hold on
 xlabel('Lon')
 ylabel('Lat')
 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);
 grid
 set(gcf,'Name',[handles.flightname 'JR Signal Strength'])

 red=[1 0 0];
 yell = [1 1 0];
 grn=[0 1 0];
 blu = [0 0 1];
 orange = [1 1/2 0];
 lblu = [0 1 1];
 purpl = [3/4 0 1];
 grey=[1/2 1/2 1/2];
 blk=[0 0 0];

 %Plot RSSI as colors..
 if (max(dat.RSSI)-min(dat.RSSI) == 0),
     r1=zeros(length(dat.RSSI),1);
 else
%     r1=(dat.RSSI-max(dat.RSSI))/(max(dat.RSSI)-min(dat.RSSI))+1;
%goes from 0 (rssi -115) to 1 (best RSSI)
    %Color code by AckRatio instead..
    r1=(dat.AckRatio-max(dat.AckRatio))/(max(dat.AckRatio)-
min(dat.AckRatio))+1;   %goes from 0 to 1 (best AckRatio)
 end
 red2=[ones(length(r1),1) zeros(length(r1),1) zeros(length(r1),1)];
 rssicolor=red2+r1*(grn-red);


 rssicolor=[];
 for i=1:length(dat.PilotPrcnt),
     PilotPrcnt=dat.PilotPrcnt(i);
     if (PilotPrcnt==100),
         col=grn;
     elseif (PilotPrcnt==90),
         col=red;
     elseif (PilotPrcnt>=70),
         col=blu;
     elseif (PilotPrcnt>=50),
         col=purpl;
     elseif (PilotPrcnt>=25),
         col=red;
     elseif (PilotPrcnt>=5),
         col=grey;
     elseif (PilotPrcnt==0),
         col=blk;
     end
     rssicolor=[rssicolor; col];
 end

%If there is a time period specified in the CL section the JR signal
plot will
%only plot within that time period.
selectperiod = 'yes'; %Used to determine if a specified range has been
given

if strcmp(handles.StartSec,'')

    selectperiod = 'no';

end

if strcmp(handles.StartMin,'')

    selectperiod = 'no';

end

if strcmp(handles.EndSec,'')

    selectperiod = 'no';

end

if strcmp(handles.EndMin,'')

    selectperiod = 'no';

end

if strcmp(selectperiod,'yes')

 starttime = (handles.StartMin*60+handles.StartSec);
 endtime = (handles.EndMin*60+handles.EndSec);
 range = find(dat.Clock/1000 >= starttime & dat.Clock/1000 <= endtime);

else
```

```
 range = 0;
 range(1,1) = 1;
 range(2,1) = length(dat.Clock);

end

if isnan(minalt)

    minalt = 0;

end

if isnan(maxalt)

    maxalt = 3048; %m

end

 for i=range(1,1):range(end,1)

     if dat.Alt(i) >= minalt && dat.Alt(i) <= maxalt


plot(dat.Lon(i)*rad2deg,dat.Lat(i)*rad2deg,'.','Color',rssicolor(i,:))
        hold on

     end

 end

 if xor(minalt > 0,maxalt < 3048)

     title(sprintf(['%5.5g ft < Altitude < %5.5g
ft'],minalt/0.3048,maxalt/0.3048));

 end

 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);

 plot(handles.CtrlTower(2,1),handles.CtrlTower(1,1),'r*');
 text('Position',[handles.CtrlTower(2,1)-0.0004
handles.CtrlTower(1,1)+0.0002],'string','Ctrl Twr')

 plot(handles.CtrlRoom(2,1),handles.CtrlRoom(1,1),'r*');
 text('Position',[handles.CtrlRoom(2,1)-0.0005 handles.CtrlRoom(1,1)-
0.0002],'string','Ctrl Rm')

 plot([-96.827858 -96.827858],[36.165109 36.159065])
 text('Position',[-96.8278147 36.162385],'string','Tree Line')

 plot([-96.836808 -96.827858],[36.159101 36.159065])
 text('Position',[-96.832164 36.159100],'string','Airport rd')

 plot([-96.835598 -96.835546],[36.161234 36.162863])
 text('Position',[-96.835612 36.161213],'string','Airstrip')
```

```
legendval = get(handles.Legend,'Value');
if isequal(legendval,1)
 h1=plot(0,0,'.');
 h2=plot(0,0,'.');
 h3=plot(0,0,'.');
 h4=plot(0,0,'.');
 h5=plot(0,0,'.');
 h6=plot(0,0,'.');
 h7=plot(0,0,'.');

 set(h1,'Color',grn);
 set(h2,'Color',red);
 set(h3,'Color',blu);
 set(h4,'Color',purpl);
 set(h5,'Color',red);
 set(h6,'Color',grey);
 set(h7,'Color',blk);

 legend([h1 h2 h3 h4 h5 h6 h7],'JR = 100','JR >= 90','JR >= 70','JR >=
50','JR >= 25', 'JR >= 5', 'JR = 0')
end
end

% --- Executes on button press in AP_Mode.
function AP_Mode_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else

load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
    plot(tClock/handles.TimeScaleValue,dat.AP_Mode,'.k')
    set(gcf,'Name',[handles.flightname 'AP Mode'])
    axis([0 max(tClock/handles.TimeScaleValue) 0 11.5])
    set(gca,'YTick',[0 1 2 3 4 5 6 7 8 9 10 11])

set(gca,'YTickLabel','Prelaunch|Transition|Liftoff|Climbout|Flying|Land
ing|Downwind|Base|Final Approach|Short Final|Touchdown|Rollout|')
    xlabel('Piccolo Time','string',handles.TimeScale)

end
```

```
% --- Executes on button press in APcontrol.
function APcontrol_Callback(hObject, eventdata, handles)

%Plots the gps location of the aircraft where green represents that the
%auto pilot was in control and blue represents that the manual r/c
pilot was
%in control
%Also plots autopilot/manual control vs. time

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

rad2deg = 180/pi;
deg2rad = pi/180;
apon = find(dat.AP_Global == 1);
apoff = find(dat.AP_Global == 0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
 latlonfig=gcf;
 a1 = min(dat.Lon(find(dat.PosGood ==1)))-0.000004;
 a2 = max(dat.Lon(find(dat.PosGood ==1)))+0.000004;
 b1 = min(dat.Lat(find(dat.PosGood ==1)))-0.000004;
 b2 = max(dat.Lat(find(dat.PosGood ==1)))+0.000004;
 treeline = 96.827858*deg2rad;
 airportrd = 36.159065*deg2rad;

 if abs(a2) > treeline

     a2 = -96.827856*deg2rad;

 end

 if b1 > airportrd

     b1 = airportrd;

 end

 if abs(a1) < abs(handles.CtrlRoom(2,1)*deg2rad)

     a1 = handles.CtrlRoom(2,1)*deg2rad - 0.000007;

 end
 hold on
 xlabel('Lon')
 ylabel('Lat')
 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);
 grid
 set(gcf,'Name','Lat-Lon')

 red=[1 0 0];
 yell = [1 1 0];
 grn=[0 1 0];
 blu = [0 0 1];
 orange = [1 1/2 0];
 lblu = [0 1 1];
 purpl = [3/4 0 1];
 grey=[1/2 1/2 1/2];
 blk=[0 0 0];

 %Plot RSSI as colors..
 if (max(dat.RSSI)-min(dat.RSSI) == 0),
     r1=zeros(length(dat.RSSI),1);
 else
%     r1=(dat.RSSI-max(dat.RSSI))/(max(dat.RSSI)-min(dat.RSSI))+1;
%goes from 0 (rssi -115) to 1 (best RSSI)
    %Color code by AckRatio instead..
    r1=(dat.AckRatio-max(dat.AckRatio))/(max(dat.AckRatio)-
min(dat.AckRatio))+1;   %goes from 0 to 1 (best AckRatio)
 end
 red2=[ones(length(r1),1) zeros(length(r1),1) zeros(length(r1),1)];
 rssicolor=red2+r1*(grn-red);


 rssicolor=[];
 for i=1:length(dat.AP_Global),
     AP_Global=dat.AP_Global(i);
     if (AP_Global==0),
         col=blu;
     elseif (AP_Global==1),
         col=grn;
     end
     rssicolor=[rssicolor; col];
 end

 %If there is a time period specified in the CL section the link plot
will
%only plot within that time period.
selectperiod = 'yes'; %Used to determine if a specified range has been
given

if strcmp(handles.StartSec,'')

    selectperiod = 'no';

end

if strcmp(handles.StartMin,'')
```

```
    selectperiod = 'no';

end

if strcmp(handles.EndSec,'')

    selectperiod = 'no';

end

if strcmp(handles.EndMin,'')

    selectperiod = 'no';

end

if strcmp(selectperiod,'yes')

 starttime = (handles.StartMin*60+handles.StartSec);
 endtime = (handles.EndMin*60+handles.EndSec);
 range = find(dat.Clock/1000 >= starttime & dat.Clock/1000 <= endtime);

else

 range = 0;
 range(1,1) = 1;
 range(2,1) = length(dat.Clock);

end

 for i=range(1,1):range(end,1)

plot(dat.Lon(i)*rad2deg,dat.Lat(i)*rad2deg,'.','Color',rssicolor(i,:))
     hold on
 end

 axis('equal')
 axis([a1 a2 b1 b2]*rad2deg);

 plot(handles.CtrlTower(2,1),handles.CtrlTower(1,1),'r*');
 text('Position',[handles.CtrlTower(2,1)-0.0004
handles.CtrlTower(1,1)+0.0002],'string','Ctrl Twr')

 plot(handles.CtrlRoom(2,1),handles.CtrlRoom(1,1),'r*');
 text('Position',[handles.CtrlRoom(2,1)-0.0005 handles.CtrlRoom(1,1)-
0.0002],'string','Ctrl Rm')

 plot([-96.827858 -96.827858],[36.165109 36.159065])
 text('Position',[-96.8278147 36.162385],'string','Tree Line')

 plot([-96.836808 -96.827858],[36.159101 36.159065])
 text('Position',[-96.832164 36.159100],'string','Airport rd')

 plot([-96.835598 -96.835546],[36.161234 36.162863])
 text('Position',[-96.835612 36.161213],'string','Airstrip')

legendval = get(handles.Legend,'Value');
```

```
if isequal(legendval,1)
 h1=plot(0,0,'.');
 h2=plot(0,0,'.');

 set(h1,'Color',blu);
 set(h2,'Color',grn);

 legend([h1 h2 ],'Manual Ctrl','Auto Ctrl')
end

if strcmp(handles.time,'Piccolo')
    tClock = dat.Clock/1000;
elseif strcmp(handles.time, 'TakeoffTime')
    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;
end

figure
plot(tClock(apon)/handles.TimeScaleValue, dat.AP_Global(apon),'.g')
hold on
plot(tClock(apoff)/handles.TimeScaleValue, dat.AP_Global(apoff),'.b')
set(gcf,'Name',[handles.flightname 'Autopilot vs. Manual Control'])
xlabel([handles.TimeLabel ' ' handles.TimeScale])
ylabel('Control')
axis([min(tClock/handles.TimeScaleValue)
max(tClock/handles.TimeScaleValue) 0 1.5])

end

% --- Executes on button press in LinkAlt.
function LinkAlt_Callback(hObject, eventdata, handles)

%Plots piccolo signal strength vs. altitude
%Also plots two different colors depending on whether or not the
aircraft
%was north or south of the OSU UAFS control room

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

deg2rad = pi/180;

figure
    grn=[0 1 0];
    blu = [0 0 1];
    rssicolor=[];

    for i=1:length(dat.Lat),
        Lattitude=dat.Lat(i);
        if (Lattitude>=handles.CtrlTower(1,1)*deg2rad),
            col=blu;
        elseif (Lattitude<handles.CtrlTower(1,1)*deg2rad),
            col=grn;
        end
        rssicolor=[rssicolor; col];
    end
    for i=1:length(dat.Lat),

plot(dat.Alt(i)/handles.DistanceValue,dat.AckRatio(i),'.','Color',rssic
olor(i,:))
     hold on
    end

    a1=0; %Link min = 0%
    a2=105; %Link max = 100%
    b1=280/handles.DistanceValue; %Altitude of OSU UAFS in meters
    b2=max(dat.Alt)/handles.DistanceValue;

    axis([b1 b2 a1 a2]);
    set(gcf,'Name',[handles.flightname 'Link vs. Altitude'])
    ylabel('Piccolo Signal')
    xlabel(['Baro Altitude' handles.DistanceUnits])

    h1=plot(100,285,'b.');
    h2=plot(100,285,'g.');

    legend([h1 h2],'North of Control Tower','South of Control Tower')
end

% --- Executes on button press in ManualConnectionAlt.
function ManualConnectionAlt_Callback(hObject, eventdata, handles)
%Plots manual signal strength vs. altitude
%Also plots two different colors depending on whether or not the
aircraft
%was north or south of the OSU UAFS control room

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

deg2rad = pi/180;

figure
    grn=[0 1 0];
    blu = [0 0 1];
    rssicolor=[];

    for i=1:length(dat.Lat),
        Lattitude=dat.Lat(i);
        if (Lattitude>=handles.CtrlTower(1,1)*deg2rad),
            col=blu;
        elseif (Lattitude<handles.CtrlTower(1,1)*deg2rad),
            col=grn;
        end
        rssicolor=[rssicolor; col];
    end

    for i=1:length(dat.Lat),

plot(dat.Alt(i)/handles.DistanceValue,dat.PilotPrcnt(i),'.','Color',rss
icolor(i,:))
```

```
         hold on
        end

        a1=0; %Manual Pilot min = 0%
        a2=105; %Manual Pilot max = 100%
        b1=280/handles.DistanceValue; %Altitude of OSU UAFS in meters
        b2=max(dat.Alt)/handles.DistanceValue;

        axis([b1 b2 a1 a2]);
        set(gcf,'Name',[handles.flightname 'Manual Signal vs. Altitude'])
        ylabel('Manual Signal')
        xlabel(['Baro Altitude' handles.DistanceUnits])

        h1=plot(100,285,'b.');
        h2=plot(100,285,'g.');

        legend([h1 h2],'North of Control Tower','South of Control Tower')
    end
```

Unit Definitions and Time Scale Selections

```
function TASmph_Callback(hObject, eventdata, handles)

%Changes velocity units to mph

units = get(hObject,'Value'); %returns toggle state of radiobutton

if units == 1

    handles.SpeedUnits = ' (mph)'; %used for ylabels
    handles.SpeedValue = 0.44704; %unit conversion

    %De-select the other airspeed unit radiobuttons
    set(handles.TASmps,'Value',0)
    set(handles.TASknots,'Value',0)
    set(handles.TASfps,'Value',0)

else

    %Keeps error from occuring where the radiobutton is de-selected by
the
    %user but the units do not change
    %The only way to change units is to select a unit rather than
    %de-selecting one to avoid this error
    set(handles.TASmph,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASmps.
function TASmps_Callback(hObject, eventdata, handles)

%Changes velocity units to m/s

units = get(hObject,'Value');

if units == 1
```

```
    handles.SpeedUnits = ' (m/s)';
    handles.SpeedValue = 1;
    set(handles.TASknots,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASfps,'Value',0)

else

    set(handles.TASmps,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASknots.
function TASknots_Callback(hObject, eventdata, handles)

%Changes velocity units to knots

units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (knots)';
    handles.SpeedValue = 0.51444;
    set(handles.TASmps,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASfps,'Value',0)

else

    set(handles.TASknots,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASfps.
function TASfps_Callback(hObject, eventdata, handles)

%Changes velocity units to ft/s

units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (ft/s)';
    handles.SpeedValue = 0.304785;
    set(handles.TASmps,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASknots,'Value',0)

else

    set(handles.TASfps,'Value',1)
```

```
end

guidata(hObject,handles)

% --- Executes on button press in DistanceMeters.
function DistanceMeters_Callback(hObject, eventdata, handles)

%Changes length units to meters

units = get(hObject,'Value');

if units == 1

    handles.DistanceUnits = ' (m)';
    handles.DistanceValue = 1;
    set(handles.DistanceFeet,'Value',0)

else

    set(handles.DistanceMeters,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DistanceFeet.
function DistanceFeet_Callback(hObject, eventdata, handles)

%Changes length units to ft

units = get(hObject,'Value');

if units == 1

    handles.DistanceUnits = ' (ft)';
    handles.DistanceValue = 0.3048;
    set(handles.DistanceMeters,'Value',0)

else

    set(handles.DistanceFeet,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Minutes.
function Minutes_Callback(hObject, eventdata, handles)

%Changes time units to minutes

units = get(hObject,'Value');

if units == 1

    handles.TimeScale = '(min)';
    handles.TimeScaleValue = 60;
    set(handles.Seconds,'Value',0)

else

    set(handles.Minutes,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Seconds.
function Seconds_Callback(hObject, eventdata, handles)

%Changes time units to seconds

units = get(hObject,'Value');

if units == 1

    handles.TimeScale = '(s)';
    handles.TimeScaleValue = 1;
    set(handles.Minutes,'Value',0)

else

    set(handles.Seconds,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Legend.
function Legend_Callback(hObject, eventdata, handles)

% --- Executes on button press in DeflectionRad.
function DeflectionRad_Callback(hObject, eventdata, handles)

%Changes surface deflection units to radians

units = get(hObject,'Value');

if units == 1

    handles.DeflectionUnits = ' (rad)';
    handles.DeflectionValue = 1;
    set(handles.DeflectionDeg,'Value',0)

else

    set(handles.DeflectionRad,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DeflectionDeg.
function DeflectionDeg_Callback(hObject, eventdata, handles)
```

```
%Changes surface deflection units to degrees

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.DeflectionUnits = ' (deg)';
    handles.DeflectionValue = 180/pi;
    set(handles.DeflectionRad,'Value',0)

else

    set(handles.DeflectionDeg,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in accelms2.
function accelms2_Callback(hObject, eventdata, handles)

%Changes accelerometer measurement units to m/s^2

units = get(hObject,'Value');

if units == 1

    handles.AccelValue = 1;
    handles.AccelUnits = ' (m/s2)';
    set(handles.accelg,'Value',0)
    set(handles.accelfts2,'Value',0)
else

    set(handles.accelms2,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in accelg.
function accelg_Callback(hObject, eventdata, handles)

%Changes accelerometer measurement units to g's

units = get(hObject,'Value');

if units == 1

    handles.AccelValue = 9.81;
    handles.AccelUnits = ' (g)';
    set(handles.accelms2,'Value',0)
    set(handles.accelfts2,'Value',0)
else

    set(handles.accelg,'Value',1)
```

```
end

guidata(hObject,handles)

% --- Executes on button press in accelfts2.
function accelfts2_Callback(hObject, eventdata, handles)

%Changes accelerometer measurement units to ft/s^2

units = get(hObject,'Value');

if units == 1

    handles.AccelValue = 0.3048;
    handles.AccelUnits = ' (ft/s2)';
    set(handles.accelg,'Value',0)
    set(handles.accelms2,'Value',0)
else

    set(handles.accelfts2,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialRad.
function InertialRad_Callback(hObject, eventdata, handles)

%Changes attitude angle units to radians

units = get(hObject,'Value');
if units == 1

    handles.InertialValue = 1;
    handles.InertialUnits = ' (rad)';
    set(handles.InertialDeg,'Value',0)

else

    set(handles.InertialRad,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialDeg.
function InertialDeg_Callback(hObject, eventdata, handles)

%Changes attitude angle units to degrees

units = get(hObject,'Value');

if units == 1

    handles.InertialValue = 180/pi;
    handles.InertialUnits = ' (deg)';
    set(handles.InertialRad,'Value',0)
```

```
else

    set(handles.InertialDeg,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Celsius.
function Celsius_Callback(hObject, eventdata, handles)

%Changes temperature units to celsius

units = get(hObject,'Value');

if units == 1

    handles.TempUnits = ' (C)';
    set(handles.Fhrnht,'Value',0)

else

    set(handles.Celsius,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Fhrnht.
function Fhrnht_Callback(hObject, eventdata, handles)

%Changes temperature units to fahrenheit

units = get(hObject,'Value');

if units == 1

    handles.TempUnits = ' (F)';
    set(handles.Celsius,'Value',0)

else

    set(handles.Fhrnht,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Masslb.
function Masslb_Callback(hObject, eventdata, handles)

%Changes mass units to pounds

units = get(hObject,'Value');

if units == 1
```

```
    handles.MassUnits = ' (lb)';
    handles.MassValue = 0.4535924;
    set(handles.Masskg,'Value',0)
    set(handles.MassN,'Value',0)

else

    set(handles.Masslb,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Masskg.
function Masskg_Callback(hObject, eventdata, handles)

%Changes mass units to kilograms

units = get(hObject,'Value');

if units == 1

    handles.MassUnits = ' (kg)';
    handles.MassValue = 1;
    set(handles.MassN,'Value',0)
    set(handles.Masslb,'Value',0)

else

    set(handles.Masskg,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in MassN.
function MassN_Callback(hObject, eventdata, handles)

%Changes mass units to newtons

units = get(hObject,'Value');

if units == 1

    handles.MassUnits = ' (N)';
    handles.MassValue = 1/9.81;
    set(handles.Masskg,'Value',0)
    set(handles.Masslb,'Value',0)

else

    set(handles.MassN,'Value',1)

end

guidata(hObject,handles)
```

```
% --- Executes on button press in PiccoloTime.
function PiccoloTime_Callback(hObject, eventdata, handles)

%Sets time scale to piccolo time
handles.time = 'Piccolo';
handles.TimeLabel = 'Piccolo Time';

%Makes sure the time scale radiobuttons are correct
set(handles.PiccoloTime,'Value',1)
set(handles.TakeoffTime,'Value',0)

guidata(hObject,handles)

% --- Executes on button press in TakeoffTime.
function TakeoffTime_Callback(hObject, eventdata, handles)

%Sets time scale to takeoff time
load(handles.flight)

%First time selecting will plot altitude vs. piccolo time for the user
to
%determine when takeoff time is
if strcmp(handles.TakeoffTimeVal,'')

    figure
    plot(dat.Clock/1000, dat.Height,'.k')
    xlabel('Piccolo Time (s)')
    ylabel('GPS Alt (m)')

    figure
    plot(dat.Clock/1000, dat.AP_Mode,'.k')
    xlabel('Piccolo Time (s)')
    ylabel('AP Mode')
    handles.TakeoffTimeVal = str2double(input('What time is takeoff
time? ','s'));

end

if ~isnan(handles.TakeoffTimeVal)

    handles.time = 'TakeoffTime'; %Used by plot function to determine
time scale
    handles.TimeLabel = 'Flight Time'; %Used by plot functions for the
x-axis label

    %Makes sure the time scale radiobuttons are correct
    set(handles.PiccoloTime,'Value',0)
    set(handles.TakeoffTime,'Value',1)

else

    set(handles.TakeoffTime,'Value',0)
    set(handles.PiccoloTime,'Value',1)
    msgbox('Takeoff time must be a number','Error')

end
```

```
guidata(hObject,handles)

% --- Executes on button press in Waypoint.
function Waypoint_Callback(hObject, eventdata, handles)
%Sets the option of plotting waypoint changes in plots
%Only utilized in some of the telemetry and inertial plots

units = get(hObject,'Value');

if units == 1

    handles.ShowWaypoint = 'yes';

else

    handles.ShowWaypoint = 'no';

end

guidata(hObject,handles)
```

Control Surface Deflections

```
% --- Executes on button press in Surface0.
function Surface0_Callback(hObject, eventdata, handles)

%Plots deflections of Surface0
%The title of the surfaces are defined by the user in the input boxes
next
%to each surface

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1); %time range that the autopilot is in
control
apoff=find(dat.AP_Global==0); %time range that the manual pilot is in
control

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%Retrieves surface name from the value of the corresponding input box
surface = char(get(handles.Sfc0Name,'string'));

%autopilot deflections are plotted green, manual pilot deflections are
%plotted blue
figure
```

```
plot(tClock(apon)/handles.TimeScaleValue,dat.Surface0(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface0(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface1.
function Surface1_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc1Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface1(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface1(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface2.
function Surface2_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
```

```
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

figure
    plot(tClock(apon)/handles.TimeScaleValue,dat.Surface2(apon)*100,'-
g')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface2(apoff)*100,'b.')
    set(gcf,'Name',[handles.flightname ' Throttle'])
    ylabel('Throttle (%)')
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in Surface3.
function Surface3_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end
surface = char(get(handles.Sfc3Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface3(apon)*handles.Def
lectionValue,'g.')
    hold on
```

```
plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface3(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface4.
function Surface4_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc4Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface4(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface4(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface5.
function Surface5_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc5Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface5(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface5(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface6.
function Surface6_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc6Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface6(apon)*handles.Def
lectionValue,'g.')
    hold on
```

```
plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface6(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface7.
function Surface7_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc7Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface7(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface7(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface8.
function Surface8_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc8Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface8(apon)*handles.Def
lectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface8(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface9.
function Surface9_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc9Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface9(apon)*handles.Def
lectionValue,'g.')
    hold on
```

```
plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface9(apoff)*handles.D
eflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface10.
function Surface10_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc10Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface10(apon)*handles.De
flectionValue,'-g')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface10(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface11.
function Surface11_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc11Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface11(apon)*handles.De
flectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface11(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface12.
function Surface12_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc12Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface12(apon)*handles.De
flectionValue,'g.')
    hold on
```

```
plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface12(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface13.
function Surface13_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc13Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface13(apon)*handles.De
flectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface13(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface14.
function Surface14_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc14Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface14(apon)*handles.De
flectionValue,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface14(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end

% --- Executes on button press in Surface15.
function Surface15_Callback(hObject, eventdata, handles)
if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

surface = char(get(handles.Sfc15Name,'string'));

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Surface15(apon)*handles.De
flectionValue,'g.')
    hold on
```

```
plot(tClock(apoff)/handles.TimeScaleValue,dat.Surface15(apoff)*handles.
DeflectionValue,'b.')
    set(gcf,'Name',[handles.flightname ' ' surface])
    ylabel('Rudder','string',[surface ' Deflection'
handles.DeflectionUnits])
    xlabel('Time Elapsed', 'string',handles.TimeScale)

end
```

Control Surface Names

```
function Sfc0Name_Callback(hObject, eventdata, handles)

%Control Surface Names are saved to handles from the input boxes

name = get(hObject,'String');
set(handles.Sfc0Name,'String',name) %sets the input box value to the
user input value
hgsave('AnalyzePiccolo') %saves the gui with the control surface name
as the input box value
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc0Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc1Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc1Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc1Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc2Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc2Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc2Name_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc3Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc3Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc3Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc4Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc4Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc4Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc5Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc5Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc5Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc6Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc6Name,'String',name)
```

```
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc6Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc7Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc7Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc7Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc8Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc8Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc8Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc9Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc9Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc9Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function Sfc10Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc10Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc10Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc11Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc11Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc11Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc12Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc12Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc12Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc13Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc13Name,'String',name)
hgsave('AnalyzePiccolo')
```

```
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc13Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc14Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc14Name,'String',name)
hgsave('AnalyzePiccolo')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc14Name_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Sfc15Name_Callback(hObject, eventdata, handles)

name = get(hObject,'String') ;
set(handles.Sfc15Name,'String',name)
hgsave('AnalyzePiccolo')
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Sfc15Name_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Unmix Upright Ruddervators

```
function UnMix1Input_Callback(hObject, eventdata, handles)

%Sets left control surface number to be unmixed
%Needs to be saved as a string to utilize the eval command in the
unmixing
handles.UnMix1 = get(hObject,'String');

%Makes sure the input is not text.  If it is text the if statement will
%reset everything to blanks to avoid a matlab error
a = str2double(get(hObject,'String'));

if isnan(a)

    set(handles.UnMix1Input,'String','')
    handles.UnMix1 = '';
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function UnMix1Input_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function UnMix2Input_Callback(hObject, eventdata, handles)

%Sets right control surface number to be unmixed
%Needs to be saved as a string to utilize the eval command in the
unmixing
handles.UnMix2 = get(hObject,'String');

a = str2double(get(hObject,'String'));

if isnan(a)

    set(handles.UnMix2Input,'String','')
    handles.UnMix2 = '';

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function UnMix2Input_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in UnMixElevator.
function UnMixElevator_Callback(hObject, eventdata, handles)

%If statements make sure that the needed parameters have been given
if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
elseif strcmp(handles.UnMix1,'')
    msgbox('Need a Surface Number for LRuddervator')
elseif strcmp(handles.UnMix2,'')
    msgbox('Need a Surface Number for RRuddervator')
else

load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')
```

```
    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%Unmixes elevator deflection
%The surface numbers have to be text in order to utilize the eval
function
LRuddervator = eval(['dat.Surface' handles.UnMix1]);
RRuddervator = eval(['dat.Surface' handles.UnMix2]);

Elevator = (LRuddervator + RRuddervator)/2;

figure

plot(tClock(apon)/handles.TimeScaleValue,Elevator(apon)*handles.Deflect
ionValue,'-g')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,Elevator(apoff)*handles.Defle
ctionValue,'b.')
    set(gcf,'Name',[handles.flightname ' Elevator'])
    ylabel('Rudder','string',['Elevator Deflection'
handles.DeflectionUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in UnMixRudder.
function UnMixRudder_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
elseif strcmp(handles.UnMix1,'')
    msgbox('Need a Surface Number for LRuddervator')
elseif strcmp(handles.UnMix2,'')
    msgbox('Need a Surface Number for RRuddervator')
else

load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end
```

```
%Unmixes rudder deflection
LRuddervator = eval(['dat.Surface' handles.UnMix1]);
RRuddervator = eval(['dat.Surface' handles.UnMix2]);

Rudder = (-LRuddervator + RRuddervator)/2;

figure

plot(tClock(apon)/handles.TimeScaleValue,Rudder(apon)*handles.Deflectio
nValue,'-g')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,Rudder(apoff)*handles.Deflect
ionValue,'b.')
    set(gcf,'Name',[handles.flightname ' Rudder'])
    ylabel('Rudder','string',['Rudder Deflection'
handles.DeflectionUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

end

% --- Executes on button press in TimeCorrection.
function TimeCorrection_Callback(hObject, eventdata, handles)
%If the piccolo is reset it will reset the recorded time values so
there wont be
%overlapping data
%This corrects the time in dat.Clock and saves the mat file
%Works for an infinite number of resets

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

count = 0;
tadjust(1,1) = 0;

time = dat.Clock;

%Loop looks for resets
for i = 2:length(time)

    if time(i,1) < time(i-1,1)

        count = count + 1;
        tadjust(count,1) = i; %row numbers where resets occur

    end
end

%Sets the end of the time span of the flight
tadjust(count+1,1) = length(time);
time2 = time;

%Adjusts the times after there are resets
if count > 0
```

```
    cc = 1;
    for i = 2:length(time)

        if xor(i == tadjust(cc,1), i > tadjust(cc,1))

            if i >= tadjust(cc+1,1)

                cc = cc + 1;

            end

            if cc == length(tadjust)

                %Occurs at the last point
                time2(i,1) = round(time(i,1) + time2(tadjust(cc-1,1)-
1,1));

            else

                time2(i,1) = round(time(i,1) + time2(tadjust(cc,1)-
1,1));

            end

        end

    end

    %Corrects time data in 'dat.Clock'
    dat.Clock = time2;
    flight = handles.flight;

clearvars('cc','count','i','tadjust','handles','hObject','eventdata','t
ime','time2')
    save(flight)

else

    msgbox('No Resets Detected')

end
end
```

Define Directory

```
function CurrentDirectory_Callback(hObject, eventdata, handles)

%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

%Changes directory for the listbox to load piccolo mat files

pathname = uigetdir; %User select directory from a dialog box

%If statement avoids error in case user clicks cancel
if ~pathname==0

    handles.flightfolder = pathname;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(hObject,handles)

end

%Launches reload listbox function
reload_listbox(handles)

function reload_listbox(handles)

%Reloads the listbox in the event of changing folders for the listbox
to
%load piccolo mat files from

%Clear the listbox file handles
handles.file_names = {''};
handles.flightname = '';

%Prepares all files in the directory to be loaded into the listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
count = 0;

%Assigns only files with extension '.mat' to listbox handles
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'tClock*'))

        handles.file_names(i-count,1)=sorted_names(i);
        handles.sorted_index(i-count,1) = sorted_index(i);

        else

            count = count +1;

        end
    else
        count = count + 1;

    end

end

handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox1,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)
%Resets the takeoff time in case a takeoff time had been specified for
the
%previous flight
handles.TakeoffTimeVal = '';
handles.time = 'Piccolo';
set(handles.PiccoloTime,'Value',1)
set(handles.TakeoffTime,'Value',0)

%Selects first listbox item

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)

    msgbox('There arent any .mat plotpiccolo files in the current
directory')

else

    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);
```

CL Section

```
function MassInput_Callback(hObject, eventdata, handles)

%Assigns input mass to a handle
handles.mass = str2double(get(hObject,'String'));

%If the mass input box is blank or a string the mass handle will be set
to
%an empty character
if isnan(handles.mass)

    handles.mass = '';
    set(handles.MassInput,'String','') %clears the input box on the gui
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function MassInput_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function SwInput_Callback(hObject, eventdata, handles)

handles.Sw = str2double(get(hObject,'String'));

if isnan(handles.Sw)

    handles.Sw = '';
    set(handles.SwInput,'String','')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function SwInput_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CLendSec_Callback(hObject, eventdata, handles)

handles.EndSec = str2double(get(hObject,'String'));

if isnan(handles.EndSec)

    handles.EndSec = '';
    set(handles.CLendSec,'String','')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function CLendSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CLendMin_Callback(hObject, eventdata, handles)
```

```
handles.EndMin = str2double(get(hObject,'String'));

if isnan(handles.EndMin)

    handles.EndMin = '';
    set(handles.CLendMin,'String','')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function CLendMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CLstartSec_Callback(hObject, eventdata, handles)

handles.StartSec = str2double(get(hObject,'String'));

if isnan(handles.StartSec)

    handles.StartSec = '';
    set(handles.CLstartSec,'String','')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function CLstartSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CLstartMin_Callback(hObject, eventdata, handles)

handles.StartMin = str2double(get(hObject,'String'));

if isnan(handles.StartMin)

    handles.StartMin = '';
    set(handles.CLstartMin,'String','')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function CLstartMin_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in RunCL.
function RunCL_Callback(hObject, eventdata, handles)

%Calculates and plots CL based on the specified time range, empty
weight,
%and wing area. If no time range is given the function will plot CL for
%the entire flight.
%This function will use the corrected fuel mass variable to determine
the
%aircraft's mass if the variable exists otherwise it uses the fuel mass
%estimated by the piccolo in flight ('dat.Fuel').

%If statements make sure that all of the required parameters have been
%given number values
if strcmp(handles.flight,'')

    msgbox('Select a plotpiccolo .mat file')

elseif ischar(handles.mass)

    msgbox('Need a value for Mass','Error')

elseif ischar(handles.Sw)

    msgbox('Need a value for Wing Area','Error')

else

selectperiod = 'yes'; %Used to determine if a specified range has been
given

if strcmp(handles.StartSec,'')

    selectperiod = 'no';

end

if strcmp(handles.StartMin,'')

    selectperiod = 'no';

end

if strcmp(handles.EndSec,'')

    selectperiod = 'no';

end

if strcmp(handles.EndMin,'')

    selectperiod = 'no';
```

```
end

load (handles.flight) %loads the piccolo mat file selected in the
listbox

CL=zeros(length(tClock),1);

mass = handles.mass*0.4535924; %converts mass to kg

%Loop calculates CL
for i= 1:length(tClock)

    if strcmp(handles.MotorType,'Gas') %for electric motors there is no
'dat.Fuel' variable

        if ~exist('CorrectedFuel','var') %determines if the corrected
fuel mass variable exists

            %if corrected fuel variable does not exist
            CL(i,1) =
(mass+dat.Fuel(i,1))*dat.Zaccel(i,1)/(dat.Dynamic(i,1)*handles.Sw)*-1;

        else

            CL(i,1) =
(mass+CorrectedFuel(i,1))*dat.Zaccel(i,1)/(dat.Dynamic(i,1)*handles.Sw)
*-1;

        end

    else

        CL(i,1) = mass*dat.Zaccel(i,1)/(dat.Dynamic(i,1)*handles.Sw)*-
1;

    end

end

%the input time is defined on the piccolo time scale
time = dat.Clock/1000;

if strcmp(selectperiod,'no')

    %sets time period to be the entire flight
    idx = zeros(length(tClock),1);
    idx(1:length(tClock),1) = 1:length(tClock);

else

    %sets time period to be within the bounds of the start and end
times
    idx = find(time > (handles.StartMin*60+handles.StartSec) & time <
(handles.EndMin*60+handles.EndSec));

end
```

```
%Plots CL vs. time at the specified range
figure
    plot(time(idx)/handles.TimeScaleValue, CL(idx),'-g.')
    hold on
    set(gcf,'Name',[handles.flightname 'CL'])
    ylabel('Y','string','CL')
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

    %Variables are displayed in the command window
    CLAverage = mean(CL(idx))
    CLMax = max(CL(idx))
    CLMin = min(CL(idx))

end

% --- Executes on button press in MassEstimateCorrected.
function MassEstimateCorrected_Callback(hObject, eventdata, handles)

%Plots mass estimate with corrected fuel mass

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%If statements make sure that the corrected fuel variable and empty
mass
%exist
if ~exist('CorrectedFuel','var')

    msgbox('Run Fuel Correction first','Error')

elseif ischar(handles.mass)

    msgbox('Need a value for Empty Mass','Error')

else

    mass = CorrectedFuel + handles.mass*0.4535924;

    figure
     plot(tClock/handles.TimeScaleValue,mass/handles.MassValue,'-k')
     set(gcf,'Name',[handles.flightname 'Corrected Mass Estimate'])
     ylabel('a','string',['Mass' handles.MassUnits])
     xlabel([handles.TimeLabel ' ' handles.TimeScale])
```

```
end
end

% --- Executes on button press in FuelMassCorrected.
function FuelMassCorrected_Callback(hObject, eventdata, handles)

%Plots the corrected fuel mass

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.MotorType,'Electric')

    msgbox('There is no fuel mass for Electric Motors')

%If statement makes sure that the corrected fuel variable exists
elseif ~exist('CorrectedFuel','var')

    msgbox('Run Fuel Correction first')

else

    figure

plot(tClock(apon)/handles.TimeScaleValue,CorrectedFuel(apon)/handles.Ma
ssValue,'.g')
     hold on

plot(tClock(apoff)/handles.TimeScaleValue,CorrectedFuel(apoff)/handles.
MassValue,'.b')
     set(gcf,'Name',[handles.flightname 'Corrected Fuel Mass
Estimate'])
     ylabel('a','string',['Fuel Mass' handles.MassUnits])
     xlabel([handles.TimeLabel ' ' handles.TimeScale])

end
end

% --- Executes on button press in MassEstimate.
function MassEstimate_Callback(hObject, eventdata, handles)
```

```
%Plots the mass estimate based on the piccolo estimated fuel mass and
user
%input empty mass

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%Makes sure a value for empty mass exists
if ischar(handles.mass)

    msgbox('Need a value for Empty Mass','Error')

else

%If electric the function won't add fuel
if strcmp(handles.MotorType,'Gas')

    mass = dat.Fuel + handles.mass*0.4535924;

else

    mass = handles.mass*0.4535924;

end

figure
 plot(tClock/handles.TimeScaleValue,mass/handles.MassValue,'-k')
 set(gcf,'Name',[handles.flightname 'Mass Estimate'])
 ylabel('a','string',['Mass' handles.MassUnits])
 xlabel([handles.TimeLabel ' ' handles.TimeScale])

end
end

% --- Executes on button press in FuelMass.
function FuelMass_Callback(hObject, eventdata, handles)

%Plots the fuel mass based on the piccolo fuel mass estimate

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)

apon=find(dat.AP_Global==1);
```

```
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

%If electric the function won't add fuel
if strcmp(handles.MotorType,'Electric')

    msgbox('There is no fuel mass for Electric Motors')

else

    figure

plot(tClock(apon)/handles.TimeScaleValue,dat.Fuel(apon)/handles.MassVal
ue,'.g')
     hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.Fuel(apoff)/handles.MassV
alue,'.b')
     set(gcf,'Name',[handles.flightname 'Fuel Mass Estimate'])
     ylabel('a','string',['Fuel' handles.MassUnits])
     xlabel([handles.TimeLabel ' ' handles.TimeScale])

end
end

% --- Executes on button press in Gas.
function Gas_Callback(hObject, eventdata, handles)

handles.MotorType = 'Gas';
set(handles.Gas,'Value',1)
set(handles.Electric,'Value',0)

guidata(hObject,handles)

% --- Executes on button press in Electric.
function Electric_Callback(hObject, eventdata, handles)
handles.MotorType = 'Electric';
set(handles.Gas,'Value',0)
set(handles.Electric,'Value',1)

guidata(hObject,handles)
```

Temperature Data

```
% --- Executes on button press in Temp.
function Temp_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
```

```
else
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.TempUnits,' (C)')

    units1 = 1;
    units2 = 0;

else

    units1 = 9/5;
    units2 = 32;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.BoxTemp(apon)*units1+units
2,'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.BoxTemp(apoff)*units1+uni
ts2,'b.')
    set(gcf,'Name',[handles.flightname ' Piccolo Temperature'])
    ylabel('Rudder','string',['Temp' handles.TempUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end

end

% --- Executes on button press in CylinderTempA.
function CylinderTempA_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
```

```
load(handles.flight)

apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.TempUnits,' (C)')

    units1 = 1;
    units2 = 0;

else

    units1 = 9/5;
    units2 = 32;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.CHT_A(apon)*units1+units2,
'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.CHT_A(apoff)*units1+units
2,'b.')
    set(gcf,'Name',[handles.flightname ' Outside Air Temperature'])
    ylabel('a','string',['Temp' handles.TempUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end

end

% --- Executes on button press in CylinderTempB.
function CylinderTempB_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)
```

```
apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.TempUnits,' (C)')

    units1 = 1;
    units2 = 0;

else

    units1 = 9/5;
    units2 = 32;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.CHT_B(apon)*units1+units2,
'g.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.CHT_B(apoff)*units1+units
2,'b.')
    set(gcf,'Name',[handles.flightname ' Piccolo Temperature'])
    ylabel('a','string',['Temp' handles.TempUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end

end

% --- Executes on button press in OutsideTemp.
function OutsideTemp_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')
    msgbox('Select a plotpiccolo .mat file')
else
load(handles.flight)
```

```
apon=find(dat.AP_Global==1);
apoff=find(dat.AP_Global==0);

if strcmp(handles.time,'Piccolo')

    tClock = dat.Clock/1000;

elseif strcmp(handles.time, 'TakeoffTime')

    tClock = dat.Clock/1000 - handles.TakeoffTimeVal;

end

if strcmp(handles.TempUnits,' (C)')

    units1 = 1;
    units2 = 0;

else

    units1 = 9/5;
    units2 = 32;

end

figure

plot(tClock(apon)/handles.TimeScaleValue,dat.OAT(apon)*units1+units2,'g
.')
    hold on

plot(tClock(apoff)/handles.TimeScaleValue,dat.OAT(apoff)*units1+units2,
'b.')
    set(gcf,'Name',[handles.flightname ' Piccolo Temperature'])
    ylabel('a','string',['Temp' handles.TempUnits])
    xlabel([handles.TimeLabel ' ' handles.TimeScale])

 legendval = get(handles.Legend,'Value');
 if isequal(legendval,1)

    h1=plot(0,0,'b.');
    h2=plot(0,0,'g.');
    legend([h1 h2],'Maunal Ctrl','Auto Ctrl')

 end

end
```

3. AnalyzeDevInterface

```
%By Anton Mornhinweg
%This gui uses the DevInterface recorded telemetry data for flight
analysis
%Also imports some data from piccolo mat files
```

```
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = AnalyzeDevInterface(varargin)
% ANALYZEDEVINTERFACE MATLAB code for AnalyzeDevInterface.fig
%      ANALYZEDEVINTERFACE, by itself, creates a new
ANALYZEDEVINTERFACE or raises the existing
%      singleton*.
%
%      H = ANALYZEDEVINTERFACE returns the handle to a new
ANALYZEDEVINTERFACE or the handle to
%      the existing singleton*.
%
%      ANALYZEDEVINTERFACE('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in ANALYZEDEVINTERFACE.M with the given
input arguments.
%
%      ANALYZEDEVINTERFACE('Property','Value',...) creates a new
ANALYZEDEVINTERFACE or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before AnalyzeDevInterface_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to AnalyzeDevInterface_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
AnalyzeDevInterface

% Last Modified by GUIDE v2.5 02-Feb-2014 15:15:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AnalyzeDevInterface_OpeningFcn,
...
                   'gui_OutputFcn',  @AnalyzeDevInterface_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before AnalyzeDevInterface is made visible.
function AnalyzeDevInterface_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AnalyzeDevInterface (see
VARARGIN)

% Choose default command line output for AnalyzeDevInterface
handles.output = hObject;

%Defaults units to feet, ft/s, ft/s^2, seconds, degrees
handles.TimeScale = 'Piccolo Time (s)';
handles.TimeScaleValue = 1/1000;
handles.DistanceValue = 0.3048;
handles.DistanceUnits = ' (ft)';
handles.SpeedValue = 0.3048;
handles.SpeedUnits = ' (ft/s)';
handles.InertialValue = 180/pi;
handles.InertialUnits = ' (deg)';
handles.InertialRateUnits = ' (deg/s)';
handles.TASRateUnits = ' ft/s/s';

%Turns on/off radiobuttons that correspond to the default units
set(handles.TASmps,'Value',0)
set(handles.TASRatemps,'Value',0)
set(handles.Minutes,'Value',0)
set(handles.DistanceMeters,'Value',0)
set(handles.DeflectionDeg,'Value',1)
set(handles.InertialDeg,'Value',1)
set(handles.InertialRateDeg,'Value',1)
set(handles.TASmph,'Value',0)
set(handles.TASknots,'Value',0)
set(handles.TASfps,'Value',1)
set(handles.Seconds,'Value',1)
set(handles.DistanceFeet,'Value',1)
set(handles.DeflectionRad,'Value',0)
set(handles.InertialRad,'Value',0)
set(handles.InertialRateRad, 'Value', 0)
set(handles.TASRatemph, 'Value', 0)
set(handles.TASRateknots, 'Value', 0)
set(handles.TASRatefps,'Value',1)

% Update handles structure
guidata(hObject, handles);
%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument
Error!');
        return;
    end
end

%Launches the load listbox function
load_listbox1(initial_dir,handles)

% --- Outputs from this function are returned to the command line.
function varargout = AnalyzeDevInterface_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Dev mat files

```
% --- Executes on button press in CreateDevFile.
function CreateDevFile_Callback(hObject, eventdata, handles)
%Creates a Dev mat file from the Dev interface and piccolo telemetry
logs

%User select Dev .log file
[filename2,pathname2]=uigetfile({'*.log','Dev Log Files
(*.log)'},'Select a Dev Log File',[handles.flightfolder]);

%User select Plotpiccolo .mat file corresponding to the same flight as
the Dev .log file
[filename3,pathname3]=uigetfile({'*.mat','Piccolo mat Files
(*.mat)'},'Select the corresponding Piccolo Data mat
file',[handles.flightfolder]);

%Loads Dev .log file name to be used for Dev .mat file name
[~,fname] = fileparts(filename2);

%Imports the first line of the log file
tline = fgetl(fopen([pathname2 filename2],'r'));

%Finds all of the spaces in the first line
s0idx=strfind(tline,' ');

%Finds all of the brackets, [, in the first line
```

```
s1idx=strfind(tline,'[');

%creates a variable where each column on the first row is the name of
the
%variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
%This loop works by assigning the characters, according to their
character
%number on the first line, in between each space and '[' as a variable
name
vars=[];
for i=1:length(s1idx),

    vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

end

%This loop deals with the fact that the CL and Lon Modes don't have
units
%so they are seperated by only spaces instead of a space and a '['.
for i = length(s1idx)+1:(length(s0idx)-1)

    vars{i}=tline(s0idx(i):s0idx(i+1));

end

%The variable 'data' will contain all of the contents of the Dev Log
file
%except for the header line.  Each variable in the log file will have
its
%own column in 'data'.

%old matlab versions import different
%data = importdata([pathname2 filename2]);

%MatLab 2013 and on
data2 = importdata([pathname2 filename2]);
data = data2.data;

%This loop takes each column of the variable 'data' and assigns it to a
%variable where the variable name is the corresponding column heading
for i=1:length(vars)

    %form command to evaluate:   dat.variable=a(:,i);
    cmdstr=['DevData.' vars{i} '=' 'data(:,' sprintf('%d',i) ');'];
    %evaluate assignment
    eval(cmdstr);

end

%Load plotpiccolo .mat file
load ([pathname3 filename3])

%Sets up variable 'combine'.
%'combine' imports the variable AP_Global from the
%plotpiccolo replay file.  AP_Global tells if the ap is in control or
the
```

```
%manual pilot.  The data recorded by the controller and in the replay
file
%are not always at the same rate, which means the data for AP_Global
does
%not match the data recorded by the dev interface.  'combine' corrects
this
%by removing the data points in AP_Global that don't have corresponding
%sampled data in the dev log file.

combine = zeros(length(data),4);
i = 1:length(data);
combine(i,3) = data(i,1);
ap1 = find(dat.Clock >= data(1,1));
ap = dat.AP_Global(ap1(1,1));
combine(1,2) = ap;
combine(1,1) = dat.Clock(1,1);
c = 0;

for j = ap1(1,1):length(dat.Clock);

    a = isequal(dat.AP_Global(j,1),ap);
    if a==0
        c = c +1;
        ap = dat.AP_Global(j,1);
        combine(1+c,1) = dat.Clock(j,1);
        combine(1+c,2) = ap;

    end

end

a = combine(1,1);
c = 1;
for k = 1:length(combine);

    a = isequal(combine(c+1,1),0);
    if a==0

        if combine(k,3) > combine(c,1)

            if combine(k,3) < combine(c+1,1)

                combine(k,4) = combine(c,2);

            else

                combine(k,4) = combine(c+1,2);
                c = c +1;
            end

        end

    else

        combine(k,4) = combine(c,2);

    end
```

```
end

piccolotimestart = combine(1,1);
APGlobal = zeros(length(data),1);

i = 1:length(data);
APGlobal(i,1) = combine(i,4);

%Creates different series for auto/manual control
apon=find(APGlobal==1);
apoff=find(APGlobal==0);

save ([handles.flightfolder '\' fname],'DevData','apon','apoff')

reload_listbox(handles)
```

Listbox

```
% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

if ~strcmp(handles.flightname,'')
index_selected = get(handles.listbox1,'Value');
file_list = get(handles.listbox1,'String');
handles.flightfilename = file_list{index_selected};
handles.flight = [handles.flightfolder '\' handles.flightfilename];
%Saves just the filename, w/o extension, to a handle
'handles.flightname'
filename4 = handles.flightfilename;
remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function load_listbox1(dir_path,handles)

handles.file_names = {''};
set(handles.listbox1, 'String', '');
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles)
handles.flightfolder = dir_path;
```

```
%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
count = 0;

%Assigns only files with extension '.mat' to listbox handles
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statement searches the .mat file for the structure DevData.
If
        %DevData does not exist then the .mat file will not be loaded
into
        %the listbox bc presumably it is not a proper Dev.mat file

        if ~isempty(whos('-file',name,'DevData*'))

            if ~isempty(whos('-file',name,'apon*'))

            handles.file_names(i-count,1)=sorted_names(i);
            handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count + 1;
            end
        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

if ~isfield(handles,'file_names')

    %msgbox('There arent any Dev .mat files in the current directory')

else

    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)
    set(handles.listbox1,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)
%Selects first listbox item on startup

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;
remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

pathname = uigetdir([handles.flightfolder]);
if ~pathname==0
handles.flightfolder = pathname;
set(handles.CurrentDirectory,'String',handles.flightfolder)
guidata(hObject,handles)
end
reload_listbox(handles)

function reload_listbox(handles)

%clear the listbox
handles.file_names = {''};
handles.flightname = '';
handles.flight = '';

%Prepares all files in directory to be loaded into listbox

dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
count = 0;

%Assigns only files with extension '.mat' to listbox handles
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'DevData*'))

            if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'apon*'))
```

```
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count + 1;
            end

        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox1,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)
%Selects first listbox item

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)

    msgbox('There arent any .mat Dev files in the current directory')

else

    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);

function CurrentDirectory_Callback(hObject, eventdata, handles)

%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
```

```
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CurrentDirectory (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Units

```
% --- Executes on button press in TASmph.
function TASmph_Callback(hObject, eventdata, handles)

%Changes Velocity Units to MPH

%Checks whether or not the button is selected. If you click it and it's
%already been selected this will make sure that it stays highlighted.
units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (mph)';
    handles.SpeedValue = 0.44704;
    set(handles.TASmps,'Value',0)
    set(handles.TASknots,'Value',0)
    set(handles.TASfps,'value',0)

else

    set(handles.TASmph,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASmps.
function TASmps_Callback(hObject, eventdata, handles)

%Changes Velocity Units to m/s

units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (m/s)';
    handles.SpeedValue = 1;
    set(handles.TASknots,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASfps,'value',0)
else

    set(handles.TASmps,'Value',1)
```

```
end

guidata(hObject,handles)

% --- Executes on button press in TASknots.
function TASknots_Callback(hObject, eventdata, handles)

%Changes Velocity Units to Knots

units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (knots)';
    handles.SpeedValue = 0.51444;
    set(handles.TASmps,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASfps,'value',0)

else

    set(handles.TASknots,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DistanceMeters.
function DistanceMeters_Callback(hObject, eventdata, handles)

%Changes Length Units to Meters

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.DistanceUnits = ' (m)';
    handles.DistanceValue = 1;
    set(handles.DistanceFeet,'Value',0)

else

    set(handles.DistanceMeters,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DistanceFeet.
function DistanceFeet_Callback(hObject, eventdata, handles)

%Changes Length Units to ft

units = get(hObject,'Value');

if units == 1
```

```
    handles.DistanceUnits = ' (ft)';
    handles.DistanceValue = 0.3048;
    set(handles.DistanceMeters,'Value',0)

else

    set(handles.DistanceFeet,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASRatemps.
function TASRatemps_Callback(hObject, eventdata, handles)

%Changes Velocity Rate Units to m/s/s

units = get(hObject,'Value');

if units == 1

    handles.TASRateUnits = ' (m/s/s)';
    handles.SpeedValue = 1;
    set(handles.TASRateknots,'Value',0)
    set(handles.TASRatemph,'Value',0)
    set(handles.TASRatefps,'Value',0)

else

    set(handles.TASRatemps,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASRatefps.
function TASRatefps_Callback(hObject, eventdata, handles)

%Changes Velocity Rate Units to ft/s/s

units = get(hObject,'Value');

if units == 1

    handles.TASRateUnits = ' (ft/s/s)';
    handles.SpeedValue = 0.3048;
    set(handles.TASRatemps,'Value',0)
    set(handles.TASRatemph,'Value',0)
    set(handles.TASRateknots,'Value',0)

else

    set(handles.TASRatefps,'Value',1)

end

guidata(hObject,handles)
```

```
% --- Executes on button press in TASRatemph.
function TASRatemph_Callback(hObject, eventdata, handles)

%Changes Velocity Rate Units to mph/s

units = get(hObject,'Value');

if units == 1

    handles.TASRateUnits = ' (mph/s)';
    handles.SpeedValue = 0.44704;
    set(handles.TASRatemps,'Value',0)
    set(handles.TASRatefps,'Value',0)
    set(handles.TASRateknots,'Value',0)

else

    set(handles.TASRatemph,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASRateknots.
function TASRateknots_Callback(hObject, eventdata, handles)

%Changes Velocity Rate Units to mph/s

units = get(hObject,'Value');

if units == 1

    handles.TASRateUnits = ' (knots/s)';
    handles.SpeedValue = 0.51444;
    set(handles.TASRatemps,'Value',0)
    set(handles.TASRatefps,'Value',0)
    set(handles.TASRatemph,'Value',0)

else

    set(handles.TASRateknots,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in TASfps.
function TASfps_Callback(hObject, eventdata, handles)

%Changes Velocity Units to ft/s

units = get(hObject,'Value');

if units == 1

    handles.SpeedUnits = ' (ft/s)';
    handles.SpeedValue = 0.3048;
    set(handles.TASmps,'Value',0)
    set(handles.TASmph,'Value',0)
    set(handles.TASknots,'value',0)

else

    set(handles.TASfps,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialRad.
function InertialRad_Callback(hObject, eventdata, handles)

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.InertialValue = 1;
    handles.InertialUnits = ' (rad)';
    set(handles.InertialDeg,'Value',0)

else

    set(handles.InertialRad,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialDeg.
function InertialDeg_Callback(hObject, eventdata, handles)

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.InertialValue = 180/pi;
    handles.InertialUnits = ' (deg)';
    set(handles.InertialRad,'Value',0)

else

    set(handles.InertialDeg,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialRateRad.
function InertialRateRad_Callback(hObject, eventdata, handles)

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1
```

```
    handles.InertialValue = 1;
    handles.InertialRateUnits = ' (rad/s)';
    set(handles.InertialRateDeg,'Value',0)

else

    set(handles.InertialRateRad,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in InertialRateDeg.
function InertialRateDeg_Callback(hObject, eventdata, handles)

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.InertialValue = 180/pi;
    handles.InertialRateUnits = ' (deg/s)';
    set(handles.InertialRateRad,'Value',0)

else

    set(handles.InertialRateDeg,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Minutes.
function Minutes_Callback(hObject, eventdata, handles)

%Changes Time Units to Minutes

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.TimeScale = 'Piccolo Time (min)';
    handles.TimeScaleValue = 1/1000/60;
    set(handles.Seconds,'Value',0)

else

    set(handles.Minutes,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in Seconds.
function Seconds_Callback(hObject, eventdata, handles)

%Changes Time Units to Seconds
```

```
units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.TimeScale = 'Piccolo Time (s)';
    handles.TimeScaleValue = 1/1000;
    set(handles.Minutes,'Value',0)

else

    set(handles.Seconds,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DeflectionRad.
function DeflectionRad_Callback(hObject, eventdata, handles)

units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.InertialUnits = ' (rad)';
    handles.InertialValue = 1;
    set(handles.DeflectionDeg,'Value',0)

else

    set(handles.DeflectionRad,'Value',1)

end

guidata(hObject,handles)

% --- Executes on button press in DeflectionDeg.
function DeflectionDeg_Callback(hObject, eventdata, handles)
units = get(hObject,'Value'); %returns toggle state of radiobutton1

if units == 1

    handles.InertialUnits = ' (deg)';
    handles.InertialValue = 180/pi;
    set(handles.DeflectionRad,'Value',0)

else

    set(handles.DeflectionDeg,'Value',1)

end

guidata(hObject,handles)
```

VRate Limits

```
function ClimbMax_Callback(hObject, eventdata, handles)
```

```
%Saves the Climb Max Fraction input into the gui
Max = str2double(get(hObject,'String'));

if isnan(Max)

    Max = '';

end

set(handles.ClimbMax,'String',Max)
hgsave('AnalyzeDevInterface')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function ClimbMax_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function DescentMax_Callback(hObject, eventdata, handles)

%Saves the Descent Max Fraction input into the gui
Max = str2double(get(hObject,'String'));
if isnan(Max)

    Max = '';

end

set(handles.DescentMax,'String',Max)
hgsave('AnalyzeDevInterface')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function DescentMax_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Dev Controller Telemetry Plots

```
% --- Executes on button press in TAS.
function TAS_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
```

```
plot(DevData.Time(apon)*handles.TimeScaleValue,DevData.TAS(apon)/handle
s.SpeedValue,'-g')
    hold on

plot(DevData.Time(apoff)*handles.TimeScaleValue,DevData.TAS(apoff)/hand
les.SpeedValue,'b.')
    hold on

plot(DevData.Time*handles.TimeScaleValue,DevData.TASCmd/handles.SpeedVa
lue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'TAS'])
    ylabel('TAS','string',['TAS ' handles.SpeedUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','TAScmd')

end

% --- Executes on button press in TASRate.
function TASRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure

plot(DevData.Time(apon)*handles.TimeScaleValue,DevData.TASRate(apon)/ha
ndles.SpeedValue,'-g')
    hold on

plot(DevData.Time(apoff)*handles.TimeScaleValue,DevData.TASRate(apoff)/
handles.SpeedValue,'b.')
    hold on

plot(DevData.Time*handles.TimeScaleValue,DevData.TASRateCmd/handles.Spe
edValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'TAS Rate'])
    ylabel('TAS','string',['TAS Rate' handles.TASRateUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','TASRateCmd')
end

% --- Executes on button press in VRate.
function VRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

ClimbMaxFraction = str2double(get(handles.ClimbMax,'String'));
DescentMaxFraction = str2double(get(handles.DescentMax,'String'));
VRateMax = DevData.TAS*ClimbMaxFraction;
VRateMin = -DevData.TAS*DescentMaxFraction;

figure

plot(DevData.Time(apon)*handles.TimeScaleValue,DevData.VRate(apon)/hand
les.SpeedValue,'-g')
    hold on

plot(DevData.Time(apoff)*handles.TimeScaleValue,DevData.VRate(apoff)/ha
ndles.SpeedValue,'b.')
    hold on

plot(DevData.Time*handles.TimeScaleValue,DevData.VRateCmd/handles.Speed
Value,'r')

    if ~isnan(VRateMax)
    hold on

plot(DevData.Time*handles.TimeScaleValue,VRateMax/handles.SpeedValue,'-
m')
    end
    if ~isnan(VRateMin)
    hold on

plot(DevData.Time*handles.TimeScaleValue,VRateMin/handles.SpeedValue,'-
c')
    end

    hold on
    set(gcf,'Name',[handles.flightname 'VRate'])
    ylabel('V','string',['VRate' handles.SpeedUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');
    h4 = plot(0,0,'m.');
    h5 = plot(0,0,'c.');

    legend([h1 h2 h3 h4 h5],'Manual','Auto','VRateCmd','VRateMax',
'VRateMin')

end

% --- Executes on button press in Roll.
function Roll_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Roll(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Roll(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue,
DevData.RollCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Roll'])
    ylabel('R','string',['Roll' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','RollCmd')

end

% --- Executes on button press in Pitch.
function Pitch_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
```

```
    plot(DevData.Time(apon)*handles.TimeScaleValue, 
DevData.Pitch(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue, 
DevData.Pitch(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue, 
DevData.PitchCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Pitch'])
    ylabel('P','string',['Pitch' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','PitchCmd')

end

% --- Executes on button press in Heading.
function Heading_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue, 
DevData.Hdg(apon)*handles.InertialValue,'g.')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue, 
DevData.Hdg(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue, 
DevData.HdgCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Heading'])
    ylabel('H','string',['Heading' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','HeadingCmd')

end
```

```
% --- Executes on button press in RollRate.
function RollRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.RollRate(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.RollRate(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue,
DevData.RollRateCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Roll Rate'])
    ylabel('R','string',['Roll Rate' handles.InertialRateUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','RollRateCmd')

end

% --- Executes on button press in PitchRate.
function PitchRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.PitchRate(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.PitchRate(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue,
DevData.PitchRateCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Pitch Rate'])
    ylabel('P','string',['Pitch Rate' handles.InertialRateUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','PitchRateCmd')

end

% --- Executes on button press in YawRate.
function YawRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.YawRate(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.YawRate(apoff)*handles.InertialValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue,
DevData.YawRateCmd*handles.InertialValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Yaw Rate'])
    ylabel('Y','string',['Yaw Rate' handles.InertialRateUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','YawRateCmd')

end

% --- Executes on button press in Altitude.
function Altitude_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else
```

```
load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Alt(apon)/handles.DistanceValue,'g.')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Alt(apoff)/handles.DistanceValue,'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue,
DevData.AltCmd/handles.DistanceValue,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Altitude'])
    ylabel('A','string',['Alt' handles.DistanceUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','AltCmd')

end

% --- Executes on button press in AileronDefl.
function AileronDefl_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Aileron(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Aileron(apoff)*handles.InertialValue,'b.')
    hold on
    set(gcf,'Name',[handles.flightname 'Aileron Deflection'])
    ylabel('Y','string',['Aileron' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end
```

```
% --- Executes on button press in ElevatorDefl.
function ElevatorDefl_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Elevator(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Elevator(apoff)*handles.InertialValue,'b.')
    hold on
    set(gcf,'Name',[handles.flightname 'Elevator Deflection'])
    ylabel('Y','string',['Elevator' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end

% --- Executes on button press in ThrottleCmd.
function ThrottleCmd_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Throttle(apon)*100,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Throttle(apoff)*100,'b.')
    hold on
    set(gcf,'Name',[handles.flightname '% Throttle'])
    ylabel('% Throttle')
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end

% --- Executes on button press in RudderDefl.
function RudderDefl_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Rudder(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Rudder(apoff)*handles.InertialValue,'b.')
    hold on
    set(gcf,'Name',[handles.flightname 'Rudder Deflection'])
    ylabel('Y','string',['Rudder' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end

% --- Executes on button press in FlapDefl.
function FlapDefl_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Flap(apon)*handles.InertialValue,'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Flap(apoff)*handles.InertialValue,'b.')
    hold on
    set(gcf,'Name',[handles.flightname 'Flap Deflection'])
    ylabel('Y','string',['Flap' handles.InertialUnits])
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end

% --- Executes on button press in RunCL.
function RunCL_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.CL(apon),'g.')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.CL(apoff),'b.')
    set(gcf,'Name',[handles.flightname 'CL'])
    ylabel('CL')
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');

    legend([h1 h2],'Manual','Auto')

end

% --- Executes on button press in RunZaccel.
function RunZaccel_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.Accel(apon),'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.Accel(apoff),'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue, DevData.AccelCmd,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'Z - Acceleration'])
    ylabel('Z - Accel (m/s/s)')
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','AccelCmd')

end

% --- Executes on button press in LonMode.
function LonMode_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.LonMode(apon),'-k')
    set(gcf,'Name',[handles.flightname 'Lon Mode'])
    axis([0 max(DevData.Time(apon)*handles.TimeScaleValue) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('Piccolo Time','string',handles.TimeScale)

end

% --- Executes on button press in RPMRate.
function RPMRate_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select a DevInterface .mat file')

else

load(handles.flight)

figure
    plot(DevData.Time(apon)*handles.TimeScaleValue,
DevData.RPMRate(apon),'-g')
    hold on
    plot(DevData.Time(apoff)*handles.TimeScaleValue,
DevData.RPMRate(apoff),'b.')
    hold on
    plot(DevData.Time*handles.TimeScaleValue, DevData.RPMRateCmd,'r')
    hold on
    set(gcf,'Name',[handles.flightname 'RPM Rate'])
    ylabel('RPM/s')
    xlabel('Piccolo Time','string',handles.TimeScale)

    hold on

    h1 = plot(0,0,'b.');
    h2 = plot(0,0,'g.');
    h3 = plot(0,0,'r.');

    legend([h1 h2 h3],'Manual','Auto','RPMRateCmd')

end
```

4. AltitudeControl

```
%By Anton Mornhinweg
%This gui uses the Dev log file to produce variables.
%Its a userinterface for plotting relevant controller telemetry that is
%relevant to tuning Altitude Control gains.
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = AltitudeControl(varargin)
% ALTITUDECONTROL MATLAB code for AltitudeControl.fig
%      ALTITUDECONTROL, by itself, creates a new ALTITUDECONTROL or
raises the existing
%      singleton*.
%
%      H = ALTITUDECONTROL returns the handle to a new ALTITUDECONTROL
or the handle to
%      the existing singleton*.
%
%      ALTITUDECONTROL('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in ALTITUDECONTROL.M with the given
input arguments.
%
%      ALTITUDECONTROL('Property','Value',...) creates a new
ALTITUDECONTROL or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before AltitudeControl_OpeningFcn gets
called.  An
```

```
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to AltitudeControl_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help AltitudeControl

% Last Modified by GUIDE v2.5 06-May-2014 21:37:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AltitudeControl_OpeningFcn, ...
                   'gui_OutputFcn',  @AltitudeControl_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before AltitudeControl is made visible.
function AltitudeControl_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AltitudeControl (see VARARGIN)

% Choose default command line output for AltitudeControl
handles.output = hObject;

%Sets initial handles on startup
handles.TimeSelection = 'yes';
handles.StartMin = '';
handles.StartSec = '';
handles.EndMin = '';
handles.EndSec = '';
handles.quit = 'no';
handles.gains.Kpa = '';
handles.gains.Kpv = '';
handles.gains.Kpz = '';
handles.gains.KI = '';
handles.gains.EPT = '';
handles.gains.LPF = '';
handles.gains.CmdLPF = '';
handles.EmptyMass = str2double(get(handles.InputEmptyMass,'String'));
handles.MotorType = 'Gas';
handles.Sw = str2double(get(handles.InputSw,'String'))/0.3048/0.3048;
handles.CLmax = str2double(get(handles.InputCLmax,'String'));
handles.CLmn = str2double(get(handles.InputCLmn,'String'));
handles.CLmin = str2double(get(handles.InputCLMin,'String'));
handles.LFMin = str2double(get(handles.InputLFMin,'String'));
handles.LFMax = str2double(get(handles.InputLFMax,'String'));

%Selects the input time radio button on startup
set(handles.InputTime,'Value',1)

%Sets the motor type selection to gas
set(handles.Gas,'Value',1)
set(handles.Electric,'Value',0)

%Update handles structure
guidata(hObject, handles);

%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument
Error!');
        return;
    end
end

%Launches the load listbox function on startup
load_listbox1(initial_dir,handles)

% --- Outputs from this function are returned to the command line.
function varargout = AltitudeControl_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Listbox Functions

```
function load_listbox1(dir_path,handles)
```

```
%Loads AltitudeCtrl mat files from the current directory into the
listbox

%Clears the listbox and handles that are associated with file names
handles.file_names = {''};
set(handles.listbox1, 'String', '');
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles)
handles.flightfolder = dir_path;

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for AltitudeCtrl mat files in the current directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variables 'apon' and 'time' in order to discern what is an
altitude file
        if ~isempty(whos('-file',name,'DevData*'))

            %apon and time should not be a variables if the mat file is
an altitude
            %file.  Time is included to weed out energy files.
            if isempty(whos('-file',name,'apon')) && isempty(whos('-
file',name,'time'))

                %Assigns the filename to the listbox by listbox index
number
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;
            end

        else

            count = count +1;

        end
    else

        count = count + 1;

    end

end

if ~isfield(handles,'file_names')

    msgbox('There arent any AltitudeCtrl.mat files in the current
directory')

else

    %Sets the directory to the current folder
    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)

    %Assigns the AltitudeCtrl mat file names to the listbox
    set(handles.listbox1,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)

%Selects first listbox item on startup and assigns the required handles

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

%Saves the selected file name into a handle 'handles.flight'
if ~strcmp(handles.flightname,'')

    index_selected = get(handles.listbox1,'Value');
    file_list = get(handles.listbox1,'String');
    handles.flightfilename = file_list{index_selected};

    %Flight handle includes the directory path and filename
    handles.flight = [handles.flightfolder '\' handles.flightfilename];

    %Saves just the filename, w/o extension, to a handle
'handles.flightname'
```

```
    filename4 = handles.flightfilename;
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CurrentDirectory_Callback(hObject, eventdata, handles)

%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CurrentDirectory (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

%Changes directory for the listbox to load AltitudeCtrl mat files

pathname = uigetdir; %User select directory from a dialog box

%If statement avoids error in case user clicks cancel
if ~pathname==0

    handles.flightfolder = pathname;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(hObject,handles)

end

%Launches reload listbox
reload_listbox(handles)

function reload_listbox(handles)
```

```
%Reloads the listbox in the event of changing folders for the listbox
to
%load AltitudeCtrl mat files from

%Clear the listbox file handles
handles.file_names = {''};
handles.flightname = '';

%Prepares all files in directory to be loaded into listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for AltitudeCtrl mat files in the new directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variables 'apon' and 'time' in order to discern what is an
AltitudeCtrl file
        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'DevData*'))

            %apon and time should not be a variables if the mat file is
an AltitudeCtrl
            %file.  Time is included to weed out energy files.
            if isempty(whos('-file',[handles.flightfolder '\' name
ext],'apon')) && isempty(whos('-file',[handles.flightfolder '\' name
ext],'time'))

                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;

            end

        else

            count = count +1;

        end

    else
        count = count + 1;

    end
```

```
end

%Sets the directory handle to the new directory
handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox1,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)
%Selects first listbox item

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)

    msgbox('There arent any .mat altitude files in the current
directory')

else

    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);
```

AltitudeCtrl .mat files

```
% --- Executes on button press in CreateAltitudeFile.
function CreateAltitudeFile_Callback(hObject, eventdata, handles)
%The AltitudeCtrl file is designed to import data from the dev log file

%User select Dev .log file
[filename2,pathname2]=uigetfile({'*.log','Dev Log Files
(*.log)'},'Select a Dev Log File',[handles.flightfolder]);

%Loads Dev .log file name to be used for AltitudeCtrl .mat file name
[~,fname] = fileparts(filename2);

%Imports the first line of the log file
tline = fgetl(fopen([pathname2 filename2],'r'));

%Find all < symbols.
s0idx=strfind(tline,' ');

%Find all > symbols
s1idx=strfind(tline,'[');

%Creates a variable where each column on the first row is the name of
the
```

```
%variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
%This loop works by assigning the characters, according to their
character
%number on the first line, in between each space and '[' as a variable
name
vars=[];
for i=1:length(s1idx),

    vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

end

%This loop deals with the fact that the CL and Lon Modes don't have
units
%so they are seperated by only spaces instead of a space and a '['.
for i = length(s1idx)+1:(length(s0idx)-1)

    vars{i}=tline(s0idx(i):s0idx(i+1));

end

%The variable 'data' will contain all of the contents of the Dev Log
file
%except for the header line.  Each variable in the log file will have
its
%own column in 'data'.

%Old matlab versions import differently
%data = importdata([pathname2 filename2]);

%MatLab 2013 and on
data2 = importdata([pathname2 filename2]);
data = data2.data;

%This loop takes each column of the variable 'data' and assigns it to a
%variable where the variable name is the corresponding column heading
for i=1:length(vars)

    %form command to evaluate:   dat.variable=a(:,i);
    cmdstr=['DevData.' vars{i} '=' 'data(:,' sprintf('%d',i) ');'];
    %evaluate assignment
    eval(cmdstr);

end

%If the motor type is gas the user will be required to select the
%corresponding piccolo telemetry log file to import the fuel mass data
if strcmp(handles.MotorType,'Gas')

    %User select PCC .log file
    [filename3,pathname3]=uigetfile({'*.log','PCC Log Files
(*.log)'},'Select a PCC Log File',pathname2);

    %Imports the first line of the log file
    tline = fgetl(fopen([pathname3 filename3],'r'));
    %Find all < symbols.
    s0idx=strfind(tline,'<');

    %Find all > symbols
    s1idx=strfind(tline,'>');

    %Creates a variable where each column on the first row is the name
of the
    %variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
    %This loop works by assigning the characters, according to their
character
    %number on the first line, in between each space and '[' as a
variable name
    vars=[];
    for i=1:length(s0idx),

        vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

    end

    %Imports the entire piccolo log file into 'data'.
    data = importdata([pathname3 filename3]);

    %Defines the names of the variables, as their corresponding header
labels,
    %to be taken from the piccolo log file data
    variables = {'Clock','Fuel','Dynamic'};

    %Using "importdata" MATLAB imports piccolo log data into the
variable "data.textdata"
    %along with the column headers. As a result the following is
necessary to
    %extract the data seperately from its column header
    PClock =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{1}))));
    Fuel =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{2}))));
    q =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{3}))));

    %In case the start fuel level was not properly entered into PCC
    if isempty(Fuel)

        FuelMass(1:length(PClock),1) = 0;

    end

    FuelMass = Fuel;

else

    FuelMass(1:length(time),1) = 0;
```

```
end

%Removes Dev from the filename
remove = 'Dev';
fname = regexprep(fname,remove,'');

PClock = PClock/1000;

%Saves the filename with the addition of 'AltitudeCtrl' to the flight
name
save ([handles.flightfolder '\' 'AltitudeCtrl'
fname],'DevData','FuelMass','q','PClock')
reload_listbox(handles)
```

Gain Values

```
%The gain values are only used to be printed in the title of the
figures
function Kpa_Callback(hObject, eventdata, handles)

%Assigns the input value of alt err to alt rate to a handle

a = get(hObject,'String');

if isempty(a) %If the input box is blank the variable a will be empty

    handles.gains.Kpa = ''; %Sets gain handle to ''

else

    handles.gains.Kpa = ['Kpa = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Kpa_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Kpv_Callback(hObject, eventdata, handles)

%Assigns the input value of alt rate err to accel to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.Kpv = '';

else

    handles.gains.Kpv = ['Kpv = ' a];
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Kpv_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CmdLPF_Callback(hObject, eventdata, handles)

%Assigns the input value of accel cmd lpf cutoff to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.CmdLPF = '';

else

    handles.gains.CmdLPF = ['CmdLPF = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function CmdLPF_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function KI_Callback(hObject, eventdata, handles)

%Assigns the input value of accel err int to elevator to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.KI = '';

else

    handles.gains.KI = ['KI = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
```

```
function KI_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Kpz_Callback(hObject, eventdata, handles)

%Assigns the input value of accel err to elevator to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.Kpz = '';

else

    handles.gains.Kpz = ['Kpz = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Kpz_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function LPF_Callback(hObject, eventdata, handles)

%Assigns the input value of acceleration lpf cutoff to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.LPF = '';

else

    handles.gains.LPF = ['LPF = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function LPF_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

function EPT_Callback(hObject, eventdata, handles)

%Assigns the input value of elevator prediction trust to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.EPT = '';

else

    handles.gains.EPT = ['EPT = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function EPT_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Time Range

```
%The user has the option to input a time range for the initial plots to
%select from.
function InputTime_Callback(hObject, eventdata, handles)
%Radio button that is used to determine if the user wants to input a
time
%range

% --- Executes on button press in InputTime.
function InputStartMin_Callback(hObject, eventdata, handles)

%Assigns input start minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    handles.StartMin = '';

else

    handles.StartMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartMin_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputEndMin_Callback(hObject, eventdata, handles)

%Assigns input end minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndMin = '';

else

    handles.EndMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEndMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputStartSec_Callback(hObject, eventdata, handles)

%Assigns input start seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.StartSec = '';

else

    handles.StartSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function InputEndSec_Callback(hObject, eventdata, handles)

%Assigns input end seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndSec = '';

else

    handles.EndSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEndSec_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Vibration Analysis

```
% --- Executes on button press in ZaccelVibe.
function ZaccelVibe_Callback(hObject, eventdata, handles)
%Plots Z accelerometer data versus Elevator, RPM, and Throttle
%The RPM data comes from the corresponding piccolo telemetry file
if strcmp(handles.flight,'')

    msgbox('Select an AltitudeCtrl .mat file')

else

load(handles.flight)

%If input time is selected if statements make sure that all the
required
%parameters have been entered
if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';
    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

    handles.quit = 'no';

    end

end

if strcmp(handles.quit,'no')

%Analyzing Zaccel Vibrations requires RPM data so the function must
import
%data from piccolo mat files in addition to the altitude mat file
%User select PCC .log file
[filename3,pathname3]=uigetfile({'*.log','PCC Log Files
(*.log)'},'Select a PCC Log File',[handles.flightfolder]);

%Imports the first line of the log file
tline = fgetl(fopen([pathname3 filename3],'r'));

%Find all < symbols.
s0idx=strfind(tline,'<');

%Find all > symbols
s1idx=strfind(tline,'>');

%Creates a variable where each column on the first row is the name of
the
%variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
%This loop works by assigning the characters, according to their
character
%number on the first line, in between each space and '[' as a variable
name
vars=[];
for i=1:length(s0idx),

    vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

end

%Imports the entire piccolo log file into 'data'.
data = importdata([pathname3 filename3]);

%Defines the names of the variables, as their corresponding header
labels,
%to be taken from the piccolo log file data
variables = {'Clock','LeftRPM'};

%Using "importdata" MATLAB imports piccolo log data into the variable
"data.textdata"
```

```
%along with the column headers. As a result the following is necessary
to
%extract the data seperately from its column header
PClock =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{1}))));
RPM =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{2}))));

%Sets time to seconds
time = DevData.Time/1000;
Ptime = PClock / 1000; %Ptime is the time that the RPM data is logged
on

Alt = DevData.Alt;
AltCmd = DevData.AltCmd;

%If a time range is input the initial plots will use the time period
%otherwise the initial plots will use the entire time period of the log
file
if isequal(get(handles.InputTime,'Value'),1)

    %Sets the period to the time inputs
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    %Plots altitude for the time period for the user to select from
    figure
    plot(time(period),Alt(period),'.g')
    hold on
    plot(time(period),AltCmd(period),'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (m)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot ranges to the selected area on the plot
    idx = find(time >= x1 & time <= x2);
    idx2 = find(Ptime >= x1 & Ptime <= x2);

else

    %Plots initial altitude plot
    figure
    plot(time,Alt,'.g')
    hold on
    plot(time,AltCmd,'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (m)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot ranges to the selected area on the plot
    idx = find(time >= x1 & time <= x2);
    idx2 = find(Ptime >= x1 & Ptime <= x2);

end

%Define all of the variables as a function of the selected time range
Zaccel = DevData.Accel(idx)/9.81;
```

```
ZaccelCmd = DevData.AccelCmd(idx)/9.81;
Selectiontime = time(idx);
Elevator = DevData.Elevator(idx)*180/pi;
Selectiontime2 = Ptime(idx2);
RPM = RPM(idx2); %RPMs are off the piccolo mat file telemetry rate
Throttle = DevData.Throttle(idx);
Pitch = DevData.Pitch(idx)*180/pi;
PitchCmd = DevData.PitchCmd(idx)*180/pi;
VRate = DevData.VRate(idx);
VRateCmd = DevData.VRateCmd(idx);

%Plot figures for analysis
figure
plot(Selectiontime, Zaccel,'-g')
hold on
plot(Selectiontime, ZaccelCmd,'-r')
xlabel('Piccolo Time(s)')
ylabel('Z Accel (g)')

figure
subplot(2,1,1)
plot(Selectiontime, Zaccel,'-g')
hold on
plot(Selectiontime, ZaccelCmd,'-r')
ylabel('Z Accel (g)')

subplot(2,1,2)
plot(Selectiontime, Elevator,'-k')
ylabel('Elevator (deg)')
xlabel('Piccolo Time (s)')

figure
subplot(2,1,1)
plot(Selectiontime, Zaccel,'-g')
hold on
plot(Selectiontime, ZaccelCmd,'-r')
ylabel('Z Accel (g)')

subplot(2,1,2)
plot(Selectiontime2, RPM,'-k')
ylabel('RPM')
xlabel('Piccolo Time (s)')

figure
subplot(2,1,1)
plot(Selectiontime, Zaccel,'-g')
hold on
plot(Selectiontime, ZaccelCmd,'-r')
ylabel('Z Accel (g)')

subplot(2,1,2)
plot(Selectiontime, Throttle,'-k')
ylabel('Throttle')
xlabel('Piccolo Time (s)')

figure
subplot(3,1,1)
```

```
plot(Selectiontime, VRate,'-g')
hold on
plot(Selectiontime, VRateCmd,'-r')
ylabel('VRate (m/s)')

subplot(3,1,2)
plot(Selectiontime, Pitch,'-g')
hold on
plot(Selectiontime, PitchCmd,'-r')
ylabel('Pitch (deg)')

subplot(3,1,3)
plot(Selectiontime, Zaccel,'-g')
hold on
plot(Selectiontime, ZaccelCmd,'-r')
ylabel('Z accel (g)')
xlabel('Piccolo Time (s)')

end
end
```

Outer and Inner Loop Plots

```
% --- Executes on button press in PlotKpa.
function PlotKpa_Callback(hObject, eventdata, handles)

%Analyzes the outer loop of altitude control
%Plots altitude error versus VRate error and Elevator deflections
if strcmp(handles.flight,'')

    msgbox('Select an AltitudeCtrl .mat file')

else

load (handles.flight)

%Checks if the user specified a time range
if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';
    else

        %The quit handle is used to cancel the run if not all of the
time values were input
        handles.quit = 'no';

    end

end

if strcmp(handles.quit,'no')

%Variables for the initial plots
time = DevData.Time/1000;
Alt = DevData.Alt;
AltCmd = DevData.AltCmd;

%If a time range is input the initial plots will use the time period
%otherwise the initial plots will use the entire time period of the log
file
if isequal(get(handles.InputTime,'Value'),1)

    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    plot(time(period),Alt(period)*3.28084,'-g') %Alt converted from m
to ft
    hold on
    plot(time(period),AltCmd(period)*3.28084,'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (ft)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

else

    %Plots the initial plots for no input time period
    figure
    plot(time,Alt*3.28084,'-g') %Alt converted from m to ft
    hold on
    plot(time,AltCmd*3.28084,'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (ft)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

end

%Define all of the variables as a function of the selected time range
'idx'
Selectiontime = time(idx) - time(idx(1));
Elevator = DevData.Elevator(idx)*180/pi;
```

```
VRate = DevData.VRate(idx);
VRateCmd = DevData.VRateCmd(idx);
VRateError = VRate - VRateCmd;
VREneg = find(VRateError < 0); %Negative VRate Error
VREpos = find(VRateError >= 0); %Positive VRate Error
AltError = Alt(idx) - AltCmd(idx);
AEneg = find(AltError < 0); %Negative Altitude Error
AEpos = find(AltError >= 0); %Positive Altitude Error

%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime(AEneg), AltError(AEneg)*3.28084,'.r') %Alt converted
from m to ft
hold on
plot(Selectiontime(AEpos), AltError(AEpos)*3.28084,'.b')
title([handles.gains.Kpa ' ' handles.gains.Kpv ' ' handles.gains.Kpz '
' handles.gains.KI ' ' handles.gains.EPT ' ' handles.gains.LPF ' '
handles.gains.CmdLPF])
ylabel('Alt Error (ft)')

subplot(2,1,2)
plot(Selectiontime(AEneg), VRateCmd(AEneg)*3.28084,'.r') %VRate
converted from m/s to ft/s
hold on
plot(Selectiontime(AEpos), VRateCmd(AEpos)*3.28084,'.b')
ylabel('VRateCmd (ft/s)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(AEneg), AltError(AEneg)*3.28084,'.r') %Alt converted
from m to ft
hold on
plot(Selectiontime(AEpos), AltError(AEpos)*3.28084,'.b')
title([handles.gains.Kpa ' ' handles.gains.Kpv ' ' handles.gains.Kpz '
' handles.gains.KI ' ' handles.gains.EPT ' ' handles.gains.LPF ' '
handles.gains.CmdLPF])
ylabel('Alt Error (ft)')

subplot(2,1,2)
plot(Selectiontime(AEneg), VRateError(AEneg)*3.28084,'.r') %VRate
converted from m/s to ft/s
hold on
plot(Selectiontime(AEpos), VRateError(AEpos)*3.28084,'.b')
ylabel('VRate Error (ft/s)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(AEneg), AltError(AEneg)*3.28084,'.r') %Alt converted
from m to ft
hold on
plot(Selectiontime(AEpos), AltError(AEpos)*3.28084,'.b')
title([handles.gains.Kpa ' ' handles.gains.Kpv ' ' handles.gains.Kpz '
' handles.gains.KI ' ' handles.gains.EPT ' ' handles.gains.LPF ' '
handles.gains.CmdLPF])
```

```
ylabel('Alt Error (ft)')

subplot(2,1,2)
plot(Selectiontime(AEneg), Elevator(AEneg),'.r')
hold on
plot(Selectiontime(AEpos), Elevator(AEpos),'.b')
ylabel('Elevator (deg)')
xlabel('t(s)')

%Checks the selected time for any of the Airspeed Modes
LonMode = DevData.LonMode(idx);
AirMode = find(LonMode > 0,1);

if ~isempty(AirMode)

    msgbox('Warning, part of the selected time occured in Airspeed
Control')

    figure
    plot(Selectiontime,DevData.LonMode(idx),'-k')
    set(gcf,'Name',[handles.flightname 'Lon Mode'])
    axis([0 max(Selectiontime) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('t(s)')

end

end
end

% --- Executes on button press in PlotVRateZaccel.
function PlotVRateZaccel_Callback(hObject, eventdata, handles)
%Analysis of the VRate and Z acceleration control loops
%Plots VRate, VRate error, Pitch, Elevator, Zaccel, and Zaccel Error
if strcmp(handles.flight,'')

    msgbox('Select an AltitudeCtrl .mat file')

else

load (handles.flight)

if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)
```

```
        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

        handles.quit = 'no';

    end

end

if strcmp(handles.quit,'no')

time = DevData.Time/1000;
VRate = DevData.VRate;
VRateCmd = DevData.VRateCmd;
AltCmd = DevData.AltCmd;
Alt = DevData.Alt;

%If a time range is input the initial plots will use the time period
%otherwise the initial plots will use the entire time period of the log
file
if isequal(get(handles.InputTime,'Value'),1)

    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    subplot(2,1,1)
    plot(time(period),VRate(period)*3.28084,'-g') %VRate converted from
m/s to ft/s
    hold on
    plot(time(period),VRateCmd(period)*3.28084,'-r')
    set(gcf,'Name', 'VRate')
    xlabel('Piccolo Time (s)')
    ylabel('VRate (ft/s)')

    subplot(2,1,2)
    plot(time(period),Alt(period)*3.28084,'-g') %Alt converted from m
to ft
    hold on
    plot(time(period),AltCmd(period)*3.28084,'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (ft)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);
    idx2 = find(PClock >= x1 & PClock <= x2);

else

    %Plots the initial plots
    figure
    plot(time,VRate*3.28084,'-g') %VRate converted from m/s to ft/s
    hold on
    plot(time,VRateCmd*3.28084,'-r')
    set(gcf,'Name', 'VRate')
    xlabel('Piccolo Time (s)')
    ylabel('VRate (ft/s)')

    figure
    plot(time,Alt*3.28084,'-g') %Alt converted from m to ft
    hold on
    plot(time,AltCmd*3.28084,'-r')
    set(gcf,'Name', 'Alt')
    xlabel('Piccolo Time (s)')
    ylabel('Alt (ft)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Altitude plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');
    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);
    idx2 = find(PClock >= x1 & PClock <= x2);

end

%Define all of the variables as a function of the selected time range
Selectiontime = time(idx) - time(idx(1));
Selectiontime2 = PClock(idx2) - PClock(idx2(1));
Pitch = DevData.Pitch(idx)*180/pi;
PitchCmd = DevData.PitchCmd(idx)*180/pi;
Elevator = DevData.Elevator(idx)*180/pi;
VRateError = VRate(idx) - VRateCmd(idx);
VREneg = find(VRateError < 0);
VREpos = find(VRateError >= 0);
Zaccel = DevData.Accel(idx);
ZaccelCmd = DevData.AccelCmd(idx);
ZaccelError = Zaccel - ZaccelCmd;
ZaEneg = find(ZaccelError < 0);
ZaEpos = find(ZaccelError >= 0);
mass = handles.EmptyMass*0.4535924 + FuelMass;
grey=[1/2 1/2 1/2];
orange = [1 1/2 0];


%Define Limit Variables
if ~isnan(handles.LFMax)

    ZaccelMax1(1:length(idx2)) = handles.LFMax * -9.81;

end

if ~isnan(handles.LFMin)

    ZaccelMin1(1:length(idx2)) = handles.LFMin*-9.81;

end

if ~isnan(handles.CLmn)
```

```
    if ~isnan(handles.Sw)

        ZaccelMax2 = -handles.CLmn*handles.Sw*q(idx2)./mass(idx2);

    end

end

if ~isnan(handles.CLmax)

    if ~isnan(handles.Sw)

        ZaccelMax3 = -handles.CLmax*handles.Sw*q(idx2)./mass(idx2);

    end

end

if ~isnan(handles.CLmin)

    if ~isnan(handles.Sw)

        ZaccelMin2 = -handles.CLmin*handles.Sw*q(idx2)./mass(idx2);

    end

end

%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime,AltCmd(idx)*3.28084,'-r') %Alt converted from m to
ft
hold on
plot(Selectiontime, Alt(idx)*3.28084,'-g')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('Alt (ft)')

subplot(2,1,2)
plot(Selectiontime, Elevator,'-g')
ylabel('Elevator (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime,PitchCmd,'-r')
hold on
plot(Selectiontime, Pitch,'-g')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('Pitch (deg)')

subplot(2,1,2)
plot(Selectiontime, Elevator,'-g')
ylabel('Elevator (deg)')
```

```
xlabel('t(s)')

figure
subplot(2,1,1)
h1 = plot(Selectiontime,ZaccelCmd,'-r');
hold on
h2 = plot(Selectiontime, Zaccel,'-g');
h3 = plot(Selectiontime2, ZaccelMax1,'-k');
h4 = plot(Selectiontime2, ZaccelMax2,'-m');
h5 = plot(Selectiontime2, ZaccelMax3,'-c');
h6 = plot(Selectiontime2, ZaccelMin1,'-');
h7 = plot(Selectiontime2, ZaccelMin2,'-');
set(h7,'Color',orange);
set(h6,'Color',grey);

title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('Zaccel (m/s/s)')

legend([h1 h2 h3 h4 h5 h6 h7],'Zaccel
Cmd','Zaccel','LFmax','CLmaxnom','CLmax','LFmin','CLmin');

subplot(2,1,2)
plot(Selectiontime, Elevator,'-g')
ylabel('Elevator (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(ZaEneg),ZaccelError(ZaEneg),'.r')
hold on
plot(Selectiontime(ZaEpos), ZaccelError(ZaEpos),'.b')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('Zaccel Error (m/s/s)')

subplot(2,1,2)
plot(Selectiontime(ZaEneg), Elevator(ZaEneg),'.r')
hold on
plot(Selectiontime(ZaEpos), Elevator(ZaEpos),'.b')
ylabel('Elevator (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(VREneg), VRateError(VREneg)*3.28084,'.r') %VRate
converted from m/s to ft/s
hold on
plot(Selectiontime(VREpos), VRateError(VREpos)*3.28084,'.b')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('VRate Error (ft/s)')

subplot(2,1,2)
plot(Selectiontime(VREneg), Elevator(VREneg),'.r')
hold on
plot(Selectiontime(VREpos), Elevator(VREpos),'.b')
```

```
ylabel('Elevator (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(VREneg), VRateError(VREneg)*3.28084,'.r') %VRate
converted from m/s to ft/s
hold on
plot(Selectiontime(VREpos), VRateError(VREpos)*3.28084,'.b')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('VRate Error (ft/s)')

subplot(2,1,2)
h1 = plot(Selectiontime(VREneg), ZaccelCmd(VREneg),'.r');
hold on
h2 = plot(Selectiontime(VREpos), ZaccelCmd(VREpos),'.b');
h3 = plot(Selectiontime2, ZaccelMax1,'-k');
h4 = plot(Selectiontime2, ZaccelMax2,'-m');
h5 = plot(Selectiontime2, ZaccelMax3,'-c');
h6 = plot(Selectiontime2, ZaccelMin1,'-');
h7 = plot(Selectiontime2, ZaccelMin2,'-');
set(h7,'Color',orange);
set(h6,'Color',grey);

ylabel('ZaccelCmd (m/s/s)')
xlabel('t(s)')

legend([h1 h2 h3 h4 h5 h6 h7],'Zaccel
Cmd','Zaccel','LFmax','CLmaxnom','CLmax','LFmin','CLmin');

figure
subplot(2,1,1)
plot(Selectiontime, VRateCmd(idx),'-r')
hold on
plot(Selectiontime, VRate(idx),'-g')
ylabel('VRate (m/s)')
title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])

subplot(2,1,2)
h1 = plot(Selectiontime,ZaccelCmd,'-r');
hold on
h2 = plot(Selectiontime, Zaccel,'-g');
h3 = plot(Selectiontime2, ZaccelMax1,'-k');
h4 = plot(Selectiontime2, ZaccelMax2,'-m');
h5 = plot(Selectiontime2, ZaccelMax3,'-c');
h6 = plot(Selectiontime2, ZaccelMin1,'-');
h7 = plot(Selectiontime2, ZaccelMin2,'-');
set(h7,'Color',orange);
set(h6,'Color',grey);

ylabel('Zaccel (m/s/s)')
xlabel('t(s)')

legend([h1 h2 h3 h4 h5 h6 h7],'Zaccel
Cmd','Zaccel','LFmax','CLmaxnom','CLmax','LFmin','CLmin');
```

```
figure
h1 = plot(time(idx),ZaccelCmd,'-r');
hold on
h2 = plot(time(idx), Zaccel,'-g');
h3 = plot(PClock(idx2), ZaccelMax1,'-k');
h4 = plot(PClock(idx2), ZaccelMax2,'-m');
h5 = plot(PClock(idx2), ZaccelMax3,'-c');
h6 = plot(PClock(idx2), ZaccelMin1,'-');
h7 = plot(PClock(idx2), ZaccelMin2,'-');
set(h7,'Color',orange);
set(h6,'Color',grey);

title([handles.gains.Kpv ' ' handles.gains.Kpz ' ' handles.gains.KI ' '
handles.gains.EPT ' ' handles.gains.LPF ' ' handles.gains.CmdLPF])
ylabel('Zaccel (m/s/s)')
xlabel('Piccolo Time (s)')

legend([h1 h2 h3 h4 h5 h6 h7],'Zaccel
Cmd','Zaccel','LFmax','CLmaxnom','CLmax','LFmin','CLmin');

%Checks the selected time for any of the Airspeed Modes
LonMode = DevData.LonMode(idx);
AirMode = find(LonMode > 0,1);

if ~isempty(AirMode)

    msgbox('Warning, part of the selected time occured in Airspeed
Control')

    figure
    plot(Selectiontime,DevData.LonMode(idx),'-k')
    set(gcf,'Name',[handles.flightname 'Lon Mode'])
    axis([0 max(Selectiontime) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('t(s)')

end

end
end
```

Input Aircraft Data and Limits

```
function InputEmptyMass_Callback(hObject, eventdata, handles)

%Assigns empty mass to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.EmptyMass = '';
    set(handles.InputEmptyMass,'String',handles.EmptyMass)

else
```

```
    handles.EmptyMass = a;
    hgsave('AltitudeControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEmptyMass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputEmptyMass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in Gas.
function Gas_Callback(hObject, eventdata, handles)

handles.MotorType = 'Gas';
set(handles.Gas,'Value',1)
set(handles.Electric,'Value',0)

guidata(hObject,handles)

% --- Executes on button press in Electric.
function Electric_Callback(hObject, eventdata, handles)

handles.MotorType = 'Electric';
set(handles.Gas,'Value',0)
set(handles.Electric,'Value',1)

guidata(hObject,handles)

function InputLFMax_Callback(hObject, eventdata, handles)

%Assigns the input value of load factor max to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.LFMax = '';
    set(handles.InputLFMax,'String',handles.LFMax)

else

    handles.LFMax = a;
    hgsave('AltitudeControl')

end
```

```
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputLFMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputLFMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function InputLFMin_Callback(hObject, eventdata, handles)

%Assigns the input value of load factor min to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.LFMin = '';
    set(handles.InputLFMin,'String',handles.LFMin)

else

    handles.LFMin = a;
    hgsave('AltitudeControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputLFMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputLFMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputCLmn_Callback(hObject, eventdata, handles)

%Assigns the input value of CL max nom to a handle

a = str2double(get(hObject,'String'));
```

```
if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.CLmn = '';
    set(handles.InputCLmn,'String',handles.CLmn)

else

    handles.CLmn = a;
    hgsave('AltitudeControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputCLmn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputCLmn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputCLmax_Callback(hObject, eventdata, handles)

%Assigns the input value of CL max to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.CLmax = '';
    set(handles.InputCLmax,'String',handles.CLmax)

else

    handles.CLmax = a;
    hgsave('AltitudeControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputCLmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputCLmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputCLMin_Callback(hObject, eventdata, handles)

%Assigns the input value of CL min to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    %handles.CLmin = '';
    set(handles.InputCLMin,'String',handles.CLmin)

else

    handles.CLmin = a;
    hgsave('AltitudeControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputCLMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputCLMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputSw_Callback(hObject, eventdata, handles)

%Assigns the input value of Sw to a handle

handles.Sw = str2double(get(hObject,'String'));

%If the Sw input box is blank or a string the Sw handle will be set to
%an empty character
if isnan(handles.Sw)

    handles.Sw = '';
    set(handles.InputSw,'String','') %clears the input box on the gui

else

    handles.Sw = handles.Sw*0.3048*0.3048;
    hgsave('AltitudeControl')
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputSw_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputSw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

5. AirspeedControl

```
%By Anton Mornhinweg
%This gui uses the Dev log file to produce variables.
%Its a userinterface for plotting controller telemetry that is
%relevant to tuning Airspeed Control gains for systems that operate
%in Altitude Control.
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = AirspeedControl(varargin)
% AIRSPEEDCONTROL MATLAB code for AirspeedControl.fig
%      AIRSPEEDCONTROL, by itself, creates a new AIRSPEEDCONTROL or
raises the existing
%      singleton*.
%
%      H = AIRSPEEDCONTROL returns the handle to a new AIRSPEEDCONTROL
or the handle to
%      the existing singleton*.
%
%      AIRSPEEDCONTROL('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in AIRSPEEDCONTROL.M with the given
input arguments.
%
%      AIRSPEEDCONTROL('Property','Value',...) creates a new
AIRSPEEDCONTROL or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before AirspeedControl_OpeningFcn gets
called.  An
```

```
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to AirspeedControl_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help AirspeedControl

% Last Modified by GUIDE v2.5 04-Feb-2014 14:44:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AirspeedControl_OpeningFcn, ...
                   'gui_OutputFcn',  @AirspeedControl_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before AirspeedControl is made visible.
function AirspeedControl_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AirspeedControl (see VARARGIN)

% Choose default command line output for AirspeedControl
handles.output = hObject;

%Sets initial handles on startup
handles.StartMin = '';
handles.StartSec = '';
handles.EndMin = '';
handles.EndSec = '';
handles.quit = 'no';
handles.gains.Kpo = '';
handles.gains.Kpi = '';
handles.gains.Kd = '';

%Selects the input time radio button on startup
```

```
set(handles.InputTime,'Value',1)

% Update handles structure
guidata(hObject, handles);

%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument
Error!');
        return;
    end
end

%Launches the load listbox function on startup
load_listbox1(initial_dir,handles)

% --- Outputs from this function are returned to the command line.
function varargout = AirspeedControl_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Listbox Functions

```
function load_listbox1(dir_path,handles)

%Loads AltitudeCtrl mat files from the current directory into the
listbox
%AirspeedControl uses the same mat file as AltitudeCtrl

%Clears the listbox and handles that are associated with file names
handles.file_names = {''};
set(handles.listbox1, 'String', '');
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles) %saves handles
handles.flightfolder = dir_path;

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
```

```
count = 0; %count adds each time the loop skips a file

%Loop searches for AltitudeCtrl mat files in the current directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variables 'apon' and 'time' in order to discern what is an
AltitudeCtrl file
        if ~isempty(whos('-file',name,'DevData*'))

            %apon and time should not be a variables if the mat file is
an altitude
            %file.  Time is included to weed out energy files.
            if isempty(whos('-file',name,'apon')) && isempty(whos('-
file',name,'time'))

                %Assigns the filename to the listbox by listbox index
number
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;
            end

        else

            count = count +1;

        end

    else

        count = count + 1;

    end

end

if ~isfield(handles,'file_names')

    msgbox('There arent any AltitudeCtrl.mat files in the current
directory')

else

    %Sets the directory to the current folder
    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)

    %Assigns the AltitudeCtrl mat file names to the listbox
    set(handles.listbox1,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)

%Selects first listbox item on startup and assigns the required handles

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

%Saves the selected file name into a handle 'handles.flight'
if ~strcmp(handles.flightname,'')

    index_selected = get(handles.listbox1,'Value');
    file_list = get(handles.listbox1,'String');
    handles.flightfilename = file_list{index_selected};

    %Flight handle includes the directory path and filename
    handles.flight = [handles.flightfolder '\' handles.flightfilename];

    %Saves just the filename, w/o extension, to a handle
'handles.flightname'
    filename4 = handles.flightfilename;
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function CurrentDirectory_Callback(hObject, eventdata, handles)

%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

%Changes directory for the listbox to load AltitudeCtrl mat files

pathname = uigetdir; %User select directory from a dialog box

%If statement avoids error in case user clicks cancel
if ~pathname==0

    handles.flightfolder = pathname;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(hObject,handles)

end

%Launches reload listbox
reload_listbox(handles)

function reload_listbox(handles)

%Reloads the listbox in the event of changing folders for the listbox
to
%load AltitudeCtrl mat files from

%Clear the listbox
handles.file_names = {''};
handles.flightname = '';

%Prepares all files in directory to be loaded into listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for AltitudeCtrl mat files in the new directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));
    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variables 'apon' and 'time' in order to discern what is an
AltitudeCtrl file
        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'DevData*'))

            %apon and time should not be a variables if the mat file is
an AltitudeCtrl
            %file.  Time is included to weed out energy files.
            if isempty(whos('-file',[handles.flightfolder '\' name
ext],'apon')) && isempty(whos('-file',[handles.flightfolder '\' name
ext],'time'))

                %Assigns the filename to the listbox by listbox index
number
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;

            end

        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

%Sets the directory handle to the new directory
handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox1,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)
%Selects first listbox item

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)
```

```
    msgbox('There arent any AirspeedControl.mat files in the current
directory')

else

    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);
```

AltitudeCtrl .mat files

```
% --- Executes on button press in CreateAltitudeFile.
function CreateAltitudeFile_Callback(hObject, eventdata, handles)
%The AltitudeCtrl file is designed to import data from the dev log file
%Exactly the same as the 'CreateAltitudeFile' function in
'AltitudeControl.m'

%User select Dev .log file
[filename2,pathname2]=uigetfile({'*.log','Dev Log Files
(*.log)'},'Select a Dev Log File',[handles.flightfolder]);

%Loads Dev .log file name to be used for AltitudeCtrl .mat file name
[~,fname] = fileparts(filename2);

%Imports the first line of the log file
tline = fgetl(fopen([pathname2 filename2],'r'));

%Find all < symbols.
s0idx=strfind(tline,' ');

%Find all > symbols
s1idx=strfind(tline,'[');

%Creates a variable where each column on the first row is the name of
the
%variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
%This loop works by assigning the characters, according to their
character
%number on the first line, in between each space and '[' as a variable
name
vars=[];
for i=1:length(s1idx),

    vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

end

%This loop deals with the fact that the CL and Lon Modes don't have
units
%so they are seperated by only spaces instead of a space and a '['.
for i = length(s1idx)+1:(length(s0idx)-1)

    vars{i}=tline(s0idx(i):s0idx(i+1));
```

```
end

%The variable 'data' will contain all of the contents of the Dev Log
file
%except for the header line.  Each variable in the log file will have
its
%own column in 'data'.

%Old matlab versions import differently
%data = importdata([pathname2 filename2]);

%MatLab 2013 and on
data2 = importdata([pathname2 filename2]);
data = data2.data;

%This loop takes each column of the variable 'data' and assigns it to a
%variable where the variable name is the corresponding column heading
for i=1:length(vars)

    %form command to evaluate:   dat.variable=a(:,i);
    cmdstr=['DevData.' vars{i} '=' 'data(:,' sprintf('%d',i) ');'];
    %evaluate assignment
    eval(cmdstr);

end

%Removes Dev from the filename
remove = 'Dev';
fname = regexprep(fname,remove,'');

%Saves the filename with the addition of 'AltitudeCtrl' to the flight
name
save ([handles.flightfolder '\' 'AltitudeCtrl' fname],'DevData')
reload_listbox(handles)
```

Gain Values

```
%The gain values are only used to be printed in the title of the
figures
function Kd_Callback(hObject, eventdata, handles)

%Assigns the input value of "TAS rate err der to accel cmd" to a handle

a = get(hObject,'String');

if isempty(a) %If the input box is blank the variable a will be empty

    handles.gains.Kd = ''; %Sets gain handle to ''

else

    handles.gains.Kd = ['Kd = ' a];

end

guidata(hObject,handles)
```

```
% --- Executes during object creation, after setting all properties.
function Kd_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Kpo_Callback(hObject, eventdata, handles)

%Assigns the input value of "TAS err to TAS rate cmd" to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.Kpo = '';

else

    handles.gains.Kpo = ['Kpo = ' a];

end

guidata(hObject,handles)


% --- Executes during object creation, after setting all properties.
function Kpo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Kpi_Callback(hObject, eventdata, handles)

%Assigns the input value of "TAS rate err to accel cmd" to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.Kpi = '';

else

    handles.gains.Kpi = ['Kpi = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Kpi_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```

Time Range

```
%The user has the option to input a time range for the initial plots to
%select from.
function InputTime_Callback(hObject, eventdata, handles)
%Radio button that is used to determine if the user wants to input a
time
%range

function InputStartMin_Callback(hObject, eventdata, handles)

%Assigns input start minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    handles.StartMin = '';

else

    handles.StartMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputEndMin_Callback(hObject, eventdata, handles)

%Assigns input end minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndMin = '';

else

    handles.EndMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEndMin_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputStartSec_Callback(hObject, eventdata, handles)

%Assigns input start seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.StartSec = '';

else

    handles.StartSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputEndSec_Callback(hObject, eventdata, handles)

%Assigns input end seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndSec = '';

else

    handles.EndSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEndSec_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Airspeed Control Inner Loop Plots

```
% --- Executes on button press in PlotInnerLoop.
function PlotInnerLoop_Callback(hObject, eventdata, handles)
%Analyzes the inner loop of the airspeed control loops

if strcmp(handles.flight,'')

    msgbox('Select an AltitudeCtrl .mat file')

else

load (handles.flight)

%Checks if the user specified a time range
if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

        %The quit handle is used to cancel the run if not all of the
time values were input
        handles.quit = 'no';

    end

end

if strcmp(handles.quit,'no')

%Variables for the initial plots
time = DevData.Time/1000;
TAS = DevData.TAS;
TAScmd = DevData.TASCmd;
LonMode = DevData.LonMode;

%If a time range is input the initial plots will use the time period
%otherwise the initial plots will use the entire time period of the log
file
if isequal(get(handles.InputTime,'Value'),1)
```

```
    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    subplot(2,1,1)
    plot(time(period),TAS(period)*1.9438445,'-g') %TAS converted from
m/s to knots
    hold on
    plot(time(period),TAScmd(period)*1.9438445,'-r')
    set(gcf,'Name', 'Initial Plots')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    subplot(2,1,2)
    plot(time(period), LonMode(period),'-k')
    axis([0 max(time(period)) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('Piccolo Time (s)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Airspeed plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

else

    %Plots the initial plots for no input time period
    figure
    plot(time,TAS*1.9438445,'-g') %TAS converted from m/s to knots
    hold on
    plot(time,TAScmd*1.9438445,'-r')
    set(gcf,'Name', 'TAS Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    figure
    plot(time, LonMode,'-k')
    set(gcf,'Name', 'Lon Mode Initial Plot')
    axis([0 max(time) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('Piccolo Time (s)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Airspeed plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

end

%Define all of the variables as a function of the selected time range
'idx'
Selectiontime = time(idx) - time(idx(1));
TASrate = DevData.TASRate(idx);
TASrateCmd = DevData.TASRateCmd(idx);
Elevator = DevData.Elevator(idx)*180/pi;
TASrateError = TASrate - TASrateCmd;
TREneg = find(TASrateError < 0); %Negative TASrate Error
```

```
TREpos = find(TASrateError >= 0); %Positive TASrate Error
VRateCmd = DevData.VRateCmd(idx);
Pitch = DevData.Pitch(idx)*180/pi;
TASerror = TAS(idx) - TAScmd(idx);
TEneg = find(TASerror < 0);
TEpos = find(TASerror >= 0);
Alt = DevData.Alt(idx);
AltCmd = DevData.AltCmd(idx);

%Plot figures for analysis
figure
subplot(3,1,1)
plot(Selectiontime,TASrateCmd,'-r')
hold on
plot(Selectiontime, TASrate,'-g')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS Rate (m/s/s)')

subplot(3,1,2)
plot(Selectiontime, Elevator,'-g')
axis([min(Selectiontime) max(Selectiontime) min(Elevator)
max(Elevator)])
ylabel('Elevator (deg)')

subplot(3,1,3)
plot(Selectiontime, Pitch,'.g')
axis([min(Selectiontime) max(Selectiontime) min(Pitch) max(Pitch)])
ylabel('Pitch (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime,AltCmd*3.28084,'-r') %Alt converted from m to ft
hold on
plot(Selectiontime, Alt*3.28084,'-g')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('Alt (ft)')

subplot(2,1,2)
plot(Selectiontime(TEneg), TASerror(TEneg)*1.9438445,'.r') %TAS
converted from m/s to knots
hold on
plot(Selectiontime(TEpos), TASerror(TEpos)*1.9438445,'.b')
ylabel('TAS Error (knots)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(TREneg), TASrateError(TREneg),'.r')
hold on
plot(Selectiontime(TREpos), TASrateError(TREpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS Rate Error (m/s/s)')

subplot(2,1,2)
plot(Selectiontime(TREneg), VRateCmd(TREneg)*3.28084,'.r') %VRate
converted from m/s to ft/s
```

```
hold on
plot(Selectiontime(TREpos), VRateCmd(TREpos)*3.28084,'.b')
ylabel('VRate (m/s)')
xlabel('t(s)')

figure
subplot(3,1,1)
plot(Selectiontime(TREneg), TASrateError(TREneg),'.r')
hold on
plot(Selectiontime(TREpos), TASrateError(TREpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS Rate Error (m/s/s)')

subplot(3,1,2)
plot(Selectiontime(TREneg), Elevator(TREneg),'.r')
hold on
plot(Selectiontime(TREpos), Elevator(TREpos),'.b')
axis([min(Selectiontime) max(Selectiontime) min(Elevator)
max(Elevator)])
ylabel('Elevator (deg)')

subplot(3,1,3)
plot(Selectiontime(TREneg), Pitch(TREneg),'.r')
hold on
plot(Selectiontime(TREpos), Pitch(TREpos),'.b')
axis([min(Selectiontime) max(Selectiontime) min(Pitch) max(Pitch)])
ylabel('Pitch (deg)')
xlabel('t(s)')

%Checks the selected time for Airspeed and Altitude Modes
LonMode = DevData.LonMode(idx);
LonModes = find(LonMode == 0 & LonMode == 1);

if ~isempty(LonModes)

    msgbox('Warning, part of the selected time did not occur in Fast or
Slow Airspeed Mode')

end

end
end
```

Airspeed Control Outer Loop plots

```
% --- Executes on button press in PlotKpo.
function PlotKpo_Callback(hObject, eventdata, handles)

%Analysis of the VRate and Z acceleration control loops
%Plots VRate, VRate error, Pitch, Elevator, Zaccel, and Zaccel Error

if strcmp(handles.flight,'')

    msgbox('Select an AltitudeCtrl .mat file')

else

load (handles.flight)
```

```
%Checks if the user specified a time range
if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

        %The quit handle is used to cancel the run if not all of the
time values were input
        handles.quit = 'no';

    end

end

if strcmp(handles.quit,'no')

%Variables for the initial plots
time = DevData.Time/1000;
TAS = DevData.TAS;
TAScmd = DevData.TASCmd;
LonMode = DevData.LonMode;

%If a time range is input the initial plots will use the time period
%otherwise the initial plots will use the entire time period of the log
file
if isequal(get(handles.InputTime,'Value'),1)

    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    subplot(2,1,1)
    plot(time(period),TAS(period)*1.9438445,'-g') %TAS converted from
m/s to knots
    hold on
    plot(time(period),TAScmd(period)*1.9438445,'-r')
    set(gcf,'Name', 'Initial Plots')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    subplot(2,1,2)
    plot(time(period), LonMode(period),'-k')
    set(gcf,'Name', 'Lon Mode Initial Plot')
    axis([0 max(time(period)) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('Piccolo Time (s)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Airspeed plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

else

    %Plots the initial plots for no input time period
    figure
    plot(time,TAS*1.9438445,'-g') %TAS converted from m/s to knots
    hold on
    plot(time,TAScmd*1.9438445,'-r')
    set(gcf,'Name', 'TAS Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (m/s)')

    figure
    plot(time, LonMode,'-k')
    set(gcf,'Name', 'Lon Mode Initial Plot')
    axis([0 max(time) 0 3.5])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')
    xlabel('Piccolo Time (s)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use the Airspeed plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

end

%Define all of the variables as a function of the selected time range
'idx'
Selectiontime = time(idx) - time(idx(1));
TASrateCmd = DevData.TASRateCmd(idx);
Elevator = DevData.Elevator(idx)*180/pi;
TAS = DevData.TAS(idx);
TAScmd = DevData.TASCmd(idx);
TASerror = TAS - TAScmd;
TEneg = find(TASerror < 0);
TEpos = find(TASerror >= 0);
VRateCmd = DevData.VRateCmd(idx);
VRate = DevData.VRate(idx);
Pitch = DevData.Pitch(idx)*180/pi;
Alt = DevData.Alt(idx);
AltCmd = DevData.AltCmd(idx);

%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime,TAScmd*1.9438445,'-r') %TAS converted from m/s to
knots
```

```
hold on
plot(Selectiontime, TAS*1.9438445,'-g')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS (knots)')

subplot(2,1,2)
plot(Selectiontime, VRateCmd*3.28084,'-r') %VRate converted from m/s to
ft/s
hold on
plot(Selectiontime, VRate*3.28084,'-g')
ylabel('VRate (ft/s)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime,AltCmd*3.28084,'-r') %Alt converted from m to ft
hold on
plot(Selectiontime, Alt*3.28084,'-g')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('Alt (ft)')

subplot(2,1,2)
plot(Selectiontime(TEneg), TASerror(TEneg)*1.9438445,'.r') %TAS
converted from m/s to knots
hold on
plot(Selectiontime(TEpos), TASerror(TEpos)*1.9438445,'.b')
ylabel('TAS Error (knots)')
xlabel('t(s)')

figure
subplot(3,1,1)
plot(Selectiontime(TEneg), TASerror(TEneg)*1.9438445,'.r') %TAS
converted from m/s to knots
hold on
plot(Selectiontime(TEpos), TASerror(TEpos)*1.9438445,'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS Error (knots)')

subplot(3,1,2)
plot(Selectiontime(TEneg), Elevator(TEneg),'.r')
hold on
plot(Selectiontime(TEpos), Elevator(TEpos),'.b')
axis([min(Selectiontime) max(Selectiontime) min(Elevator)
max(Elevator)])
ylabel('Elevator (deg)')

subplot(3,1,3)
plot(Selectiontime(TEneg), Pitch(TEneg),'.r')
hold on
plot(Selectiontime(TEpos), Pitch(TEpos),'.b')
axis([min(Selectiontime) max(Selectiontime) min(Pitch) max(Pitch)])
ylabel('Pitch (deg)')
xlabel('t(s)')

figure
subplot(2,1,1)
```

```
plot(Selectiontime(TEneg), TASerror(TEneg)*1.9438445,'.r') %TAS
converted from m/s to knots
hold on
plot(Selectiontime(TEpos), TASerror(TEpos)*1.9438445,'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.Kd])
ylabel('TAS Error (knots)')

subplot(2,1,2)
plot(Selectiontime(TEneg), TASrateCmd(TEneg),'.r')
hold on
plot(Selectiontime(TEpos), TASrateCmd(TEpos),'.b')
ylabel('TAS Rate Cmd (m/s/s)')
xlabel('t(s)')

%Checks the selected time for Airspeed and Altitude Modes
LonMode = DevData.LonMode(idx);
LonModes = find(LonMode == 0 & LonMode == 1);

if ~isempty(LonModes)

    msgbox('Warning, part of the selected time did not occur in Fast or
Slow Airspeed Mode')

end

end
end
```

6. EnergyControl

```
%By Anton Mornhinweg
%This gui uses the piccolo telemetry and Dev log files to produce
variables.
%Its a userinterface for plotting controller telemetry that is
%relevant to tuning Energy Control gains for systems that operate
%in Altitude Control.
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = EnergyControl(varargin)
% ENERGYCONTROL M-file for EnergyControl.fig
%      ENERGYCONTROL, by itself, creates a new ENERGYCONTROL or raises
the existing
%      singleton*.
%
%      H = ENERGYCONTROL returns the handle to a new ENERGYCONTROL or
the handle to
%      the existing singleton*.
%
```

```
%      ENERGYCONTROL('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in ENERGYCONTROL.M with the given input
arguments.
%
%      ENERGYCONTROL('Property','Value',...) creates a new
ENERGYCONTROL or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before EnergyControl_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to EnergyControl_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help EnergyControl

% Last Modified by GUIDE v2.5 01-Apr-2014 10:46:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @EnergyControl_OpeningFcn, ...
                   'gui_OutputFcn',  @EnergyControl_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before EnergyControl is made visible.
function EnergyControl_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to EnergyControl (see VARARGIN)

% Choose default command line output for EnergyControl
handles.output = hObject;
```

```
%Sets initial handles on startup
handles.StartMin = '';
handles.StartSec = '';
handles.EndMin = '';
handles.EndSec = '';
handles.gains.Kpo = '';
handles.gains.Kpi = '';
handles.gains.KI = '';
handles.gains.TPT = '';
handles.gains.LPF = '';
handles.TimeSelection = 'yes';
handles.quit = 'no';
handles.EmptyMass = str2double(get(handles.InputEmptyMass,'String'));
handles.MEP = str2double(get(handles.InputMEP,'String'));
handles.MotorType = 'Gas';

%Selects the input time radio button on startup
set(handles.InputTime,'Value',1)

%Sets the motor type selection to gas
set(handles.Gas,'Value',1)
set(handles.Electric,'Value',0)

% Update handles structure
guidata(hObject, handles);

%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input 
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument 
Error!');
        return;
    end
end

%Launches the load listbox function on startup
load_listbox1(initial_dir,handles)

% --- Outputs from this function are returned to the command line.
function varargout = EnergyControl_OutputFcn(hObject, eventdata, 
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
```

```
varargout{1} = handles.output;
```

Listbox Functions

```
function load_listbox1(dir_path,handles)

%Loads Energy mat files from the current directory into the listbox

%Clears the listbox and handles that are associated with file names
handles.file_names = {''};
set(handles.listbox1, 'String', '');
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles) %saves handles
handles.flightfolder = dir_path;

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for Energy mat files in the current directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variable 'FuelMass' in order to discern what is an Energy
file
        if ~isempty(whos('-file',name,'DevData*'))

            %FuelMass should be a variable. If the mat file does not
have
            %FuelMass it is excluded from the listbox
            if ~isempty(whos('-file',name,'FuelMass'))

                %Assigns the filename to the listbox by listbox index
number
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;
            end

        else

            count = count +1;
        end

    else

        count = count + 1;

    end

end

if ~isfield(handles,'file_names')

    msgbox('There arent any Energy.mat files in the current directory')

else

    %Sets the directory to the current folder
    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)

    %Assigns the AltitudeCtrl mat file names to the listbox
    set(handles.listbox1,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)

%Selects first listbox item on startup and assigns the required handles

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

%Saves the selected file name into a handle 'handles.flight'
if ~strcmp(handles.flightname,'')

    index_selected = get(handles.listbox1,'Value');
    file_list = get(handles.listbox1,'String');
    handles.flightfilename = file_list{index_selected};

    %Flight handle includes the directory path and filename
    handles.flight = [handles.flightfolder '\' handles.flightfilename];
    %Saves just the filename, w/o extension, to a handle
'handles.flightname'
    filename4 = handles.flightfilename;
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CurrentDirectory_Callback(hObject, eventdata, handles)
%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

%Changes directory for the listbox to load Energy mat files

pathname = uigetdir; %User select directory from a dialog box

%If statement avoids error in case user clicks cancel
if ~pathname==0

    handles.flightfolder = pathname;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(hObject,handles)

end

%Launches reload listbox
reload_listbox(handles)

function reload_listbox(handles)

%Reloads the listbox in the event of changing folders for the listbox
to
%load Energy mat files from

%Clear the listbox
```

```
handles.file_names = {''};
handles.flightname = '';

%Prepares all files in directory to be loaded into listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for Energy mat files in the new directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %If statements search the .mat file for the structure 'DevData'
and
        %the variable 'FuelMass' in order to discern what is an Energy
file
        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'DevData*'))

            %FuelMass should be a variable. If the mat file does not
have
            %FuelMass it is excluded from the listbox
            if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'FuelMass'))

                %Assigns the filename to the listbox by listbox index
number
                handles.file_names(i-count,1)=sorted_names(i);
                handles.sorted_index(i-count,1) = sorted_index(i);

            else

                count = count +1;

            end

        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

%Sets the directory handle to the new directory
```

```
handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox1,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)
%Selects first listbox item

file_list = get(handles.listbox1,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)

    msgbox('There arent any Energy.mat files in the current directory')

else

    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);
```

Energy .mat files

```
% --- Executes on button press in CreateEnergyFile.
function CreateEnergyFile_Callback(hObject, eventdata, handles)
%The Energy file consists of data from the Dev log file and can contain
%data from the piccolo log file (gas motors)
%Loads dev log file first

%User select Dev .log file
[filename2,pathname2]=uigetfile({'*.log','Dev Log Files
(*.log)'},'Select a Dev Log File',[handles.flightfolder]);

%Loads Dev .log file name to be used for Energy .mat file name
[~,fname] = fileparts(filename2);

tline = fgetl(fopen([pathname2 filename2],'r'));

%Find all < symbols.
s0idx=strfind(tline,' ');

%Find all > symbols
s1idx=strfind(tline,'[');

%Creates a variable where each column on the first row is the name of
the
%variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
%This loop works by assigning the characters, according to their
character
```

```
%number on the first line, in between each space and '[' as a variable
name
vars=[];
for i=1:length(s1idx),

    vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

end

%This loop deals with the fact that the CL and Lon Modes don't have
units
%so they are seperated by only spaces instead of a space and a '['.
for i = length(s1idx)+1:(length(s0idx)-1)

    vars{i}=tline(s0idx(i):s0idx(i+1));

end

%Older MATLAB versions (< R2013) import text files differently
%data = importdata([pathname2 filename2]);

%MATLAB, version R2013, will import the dev log file data into the data
variable seperate
%from the column headers in textdata so the following is necessary.
data2 = importdata([pathname2 filename2]);
data = data2.data;

for i=1:length(vars)

    %form command to evaluate:   dat.variable=a(:,i);
    cmdstr=['DevData.' vars{i} '=' 'data(:,' sprintf('%d',i) ');'];
    %evaluate assignment
    eval(cmdstr);

end

%time defaults to seconds
time = DevData.Time/1000;

%If the motor type is gas the user will be required to select the
%corresponding piccolo telemetry log file to import the fuel mass data
if strcmp(handles.MotorType,'Gas')

    %User select PCC .log file
    [filename3,pathname3]=uigetfile({'*.log','PCC Log Files
(*.log)'},'Select a PCC Log File',pathname2);

    %Imports the first line of the log file
    tline = fgetl(fopen([pathname3 filename3],'r'));

    %Find all < symbols.
    s0idx=strfind(tline,'<');

    %Find all > symbols
    s1idx=strfind(tline,'>');
    %Creates a variable where each column on the first row is the name
of the
    %variable that was recorded i.e. 'Time', 'Alt', etc. w/o units and
symbols
    %This loop works by assigning the characters, according to their
character
    %number on the first line, in between each space and '[' as a
variable name
    vars=[];
    for i=1:length(s0idx),

        vars{i}=tline(s0idx(i)+1:s1idx(i)-1);

    end

    %Imports the entire piccolo log file into 'data'.
    data = importdata([pathname3 filename3]);

    %Defines the names of the variables, as their corresponding header
labels,
    %to be taken from the piccolo log file data
    variables = {'Clock','Fuel'};

    %Using "importdata" MATLAB imports piccolo log data into the
variable "data.textdata"
    %along with the column headers. As a result the following is
necessary to
    %extract the data seperately from its column header
    PClock =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{1}))));
    Fuel =
str2double(data.textdata(2:length(data.textdata),find(strcmp(vars,varia
bles{2}))));

    %In case the start fuel level was not properly entered into PCC
    if isempty(Fuel)

        Fuel(1:length(PClock),1) = 0;

    end

    %Loop adjusts the Fuel data to match the time span and time rate of
the dev
    %telemetry log file
    i = 0;
    cc = 1;
    j = 0;
    for i = 1:length(PClock)


        Pt1 = PClock(i);

        if PClock(i) >= DevData.Time(cc)

            for j = 0:length(DevData.Time)
```

```
                if DevData.Time(cc+j) <= PClock(i)

                    FueltAdjusted(cc+j,1) = Fuel(i);

                else
                    cc = cc + j;
                    break
                end

                if cc == length(DevData.Time)
                    break
                end

            end

        end

    end

    FuelMass = FueltAdjusted;

else

    FuelMass(1:length(time),1) = 0;

end

%Removes Dev from the filename
remove = 'Dev';
fname = regexprep(fname,remove,'');

%Saves the filename with the addition of 'Energy' to the flight name
save ([handles.flightfolder '\' 'Energy'
fname],'DevData','time','FuelMass')
reload_listbox(handles)
```

Gain Values

```
%The gain values are used to be printed in the title of the figures and
%they are used to calculate the Energy Rate Cmd which is not provided
by
%the DevInterface
function InputKpo_Callback(hObject, eventdata, handles)

%Assigns the input value of "Energy err to Energy Rate cmd" to two
handles

a = get(hObject,'String');

if isempty(a) %If the input box is blank the variable a will be empty

    handles.gains.Kpo = ''; %Sets gain text handle to ''

else

    handles.gains.Kpo = ['Kpo = ' a]; %Used to print on plots (text
variable)
    b = str2double(a);
    handles.gains.KpoVal = b; %Used in calculations of Energy Rate Cmd

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputKpo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputKpi_Callback(hObject, eventdata, handles)

%Assigns the input value of "Energy Rate err to Throttle" to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.Kpi = '';

else

    handles.gains.Kpi = ['Kpi = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputKpi_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputKI_Callback(hObject, eventdata, handles)

%Assigns the input value of "Energy Rate err int to Throttle" to a
handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.KI = '';

else

    handles.gains.KI = ['KI = ' a];

end
```

```
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputKI_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputTPT_Callback(hObject, eventdata, handles)

%Assigns the input value of "Elevator Prediction Trust" to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.TPT = '';

else

    handles.gains.TPT = ['TPT = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputTPT_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputLPF_Callback(hObject, eventdata, handles)

%Assigns the input value of "Throttle LPF Cutoff" to a handle

a = get(hObject,'String');

if isempty(a)

    handles.gains.LPF = '';

else

    handles.gains.LPF = ['LPF = ' a];

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputLPF_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Time Range

```
%The user has the option to input a time range for the initial plots to
%select from.
function InputTime_Callback(hObject, eventdata, handles)
%Radio button that is used to determine if the user wants to input a
time
%range

function InputStartMin_Callback(hObject, eventdata, handles)

%Assigns input start minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    handles.StartMin = '';

else

    handles.StartMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputEndMin_Callback(hObject, eventdata, handles)

%Assigns input end minutes to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndMin = '';

else

    handles.EndMin = str2double(get(hObject,'String'));

end

guidata(hObject,handles)
```

```
% --- Executes during object creation, after setting all properties.
function InputEndMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputStartSec_Callback(hObject, eventdata, handles)

%Assigns input start seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.StartSec = '';

else

    handles.StartSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputStartSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputEndSec_Callback(hObject, eventdata, handles)

%Assigns input end seconds to a handle

a = str2double(get(hObject,'String'));

if isnan(a)

    handles.EndSec = '';

else

    handles.EndSec = str2double(get(hObject,'String'));

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEndSec_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
```

Input Aircraft Data

```
function InputEmptyMass_Callback(hObject, eventdata, handles)

%Assigns empty mass to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    handles.EmptyMass = '';
    set(handles.InputEmptyMass,'String','')

else

    handles.EmptyMass = a;
    hgsave('EnergyControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputEmptyMass_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function InputMEP_Callback(hObject, eventdata, handles)

%Assigns max engine power to a handle

a = str2double(get(hObject,'String'));

if isnan(a) %str2double of a string, or blank, results in NaN

    handles.MEP = '';
    set(handles.InputMEP,'String','')

else

    handles.MEP = a;
    hgsave('EnergyControl')

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function InputMEP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputMEP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Energy Control Gain Plots

```
% --- Executes on button press in AnalyzeERateCmd.
function AnalyzeERateCmd_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select an Energy .mat file')

elseif strcmp(handles.gains.Kpo,'')

    msgbox('Need a value for Energy err to Energy Rate Cmd')

elseif strcmp(handles.InputEmptyMass,'')

    msgbox('Need a value for Empty Mass')

elseif strcmp(handles.InputMEP,'')

    msgbox('Need a value for Max Engine Power')

else

load (handles.flight)

%Variables for the initial plots
Alt = DevData.Alt;
TAS = DevData.TAS;
Altcmd = DevData.AltCmd;
TAScmd = DevData.TASCmd;
LonMode = DevData.LonMode;

if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)
```

```
        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

    %If a time range is input the initial plots will use the time
period
    %otherwise the initial plots will use the entire time period of the
log file
    handles.quit = 'no';

    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    subplot(3,1,1)
    plot(time(period),Alt(period)*3.28084,'-g') %Alt converted from m
to ft
    hold on
    plot(time(period),Altcmd(period)*3.28084,'-r')
    set(gcf,'Name', 'Initial Plots')
    %xlabel('Piccolo Time (s)')
    ylabel('Altitude (m)')

    subplot(3,1,2)
    plot(time(period), LonMode(period),'-k')
    axis([min(time(period)) max(time(period)) -0.25 3.25])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')

    %figure
    subplot(3,1,3)
    plot(time(period),TAS(period)*1.9438445,'-g') %TAS converted from
m/s to knots
    hold on
    plot(time(period),TAScmd(period)*1.9438445,'-r')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use either the Altitude or TAS plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

    end

else

    %Plots the initial plots for no input time period
    figure
    plot(time,Alt*3.28084,'-g') %Alt converted from m to ft
    hold on
    plot(time,Altcmd*3.28084,'-r')
    set(gcf,'Name', 'Altitude Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('Altitude (m)')

    figure
    plot(time, LonMode,'-k')
    set(gcf,'Name', 'Lon Mode Initial Plot')
    axis([min(time) max(time) -0.25 3.25])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')

    figure
    plot(time,TAS*1.9438445,'-g') %TAS converted from m/s to knots
    hold on
    plot(time,TAScmd*1.9438445,'-r')
    set(gcf,'Name', 'TAS Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use either the Altitude or TAS plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

end

%If the input time is missing values the GUI will quit
if strcmp(handles.quit,'yes')

else

%Define all of the variables as a function of the selected time range
'idx'
VRate = DevData.VRate(idx);
VRatecmd = DevData.VRateCmd(idx);
TASRate = DevData.TASRate(idx);
TASRatecmd = DevData.TASRateCmd(idx);
TAS = DevData.TAS(idx);
TAScmd = DevData.TASCmd(idx);
Alt = DevData.Alt(idx);
Altcmd = DevData.AltCmd(idx);
Selectiontime = time(idx)-time(idx(1));
Throttle = DevData.Throttle(idx);

%Calculate variables
Power = Throttle*handles.MEP;
MaxPower(1:length(idx)) = handles.MEP;
TASRateError = TASRate - TASRatecmd;
VRateError = VRate - VRatecmd;
AltError = Alt - Altcmd;
TASError = TAS - TAScmd;
Eerrorneg = find(TASError<0);
Eerrorpos = find(TASError>=0);
mass = handles.EmptyMass*0.4535924+FuelMass(idx); %empty mass is
converted from lb to kg
Energy = mass*0.5.*(TAS.^2); %Kinetic Energy equation E = 1/2mV^2
Energycmd = mass*0.5.*(TAScmd.^2);
EnergyError = Energy - Energycmd;
PotentialEnergyRatecmd = mass*9.81.*abs(VRatecmd); %Potential Energy
Rate dE/dt = mg(dh/dt)
EnergyRate = mass*9.81.*abs(VRate);

%Calculate Energy Rate Command that comes from the outer loop
```

```
Kpo = handles.gains.KpoVal;
ERateCmd = -Kpo*EnergyError; %Outer loop Energy Rate Command
ERateCmd(find(ERateCmd < 0)) = 0;
%The Energy Rate error the inner loop sees includes the Potential
Energy Rate Cmd
ERateError = EnergyRate - ERateCmd-PotentialEnergyRatecmd;
ERateErrorNeg = find(ERateError < 0);
ERateErrorPos = find(ERateError >= 0);

%Calculate Energy Rate Command that comes from the VRate Cmd
ERateCmdV = mass*9.81.*VRatecmd;

%Plot figures for analysis
figure
subplot(2,1,1)
plot(Selectiontime, TAS*1.9438445, '-g') %TAS converted from m/s to
knots
hold on
plot(Selectiontime, TAScmd*1.9438445,'-r')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('TAS (knots)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg),EnergyError(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),EnergyError(Eerrorpos),'.b')
ylabel('Energy Error(J)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(Eerrorneg),EnergyError(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),EnergyError(Eerrorpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('Energy Error(J)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg),Power(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),Power(Eerrorpos),'.b')
hold on
plot(Selectiontime,MaxPower,'-c')
ylabel('Throttle (W)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(ERateErrorNeg),ERateError(ERateErrorNeg),'.r')
hold on
plot(Selectiontime(ERateErrorPos), ERateError(ERateErrorPos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('Energy Rate Error (W)')
```

```
subplot(2,1,2)
plot(Selectiontime(ERateErrorNeg), Power(ERateErrorNeg),'.r')
hold on
plot(Selectiontime(ERateErrorPos), Power(ERateErrorPos),'.b')
hold on
plot(Selectiontime,MaxPower,'-c')
ylabel('Throttle (W)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime(Eerrorneg),EnergyError(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),EnergyError(Eerrorpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('Energy Error (J)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg),ERateCmd(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos),ERateCmd(Eerrorpos),'.b')
ylabel('Outer Loop Energy Rate Cmd (W)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime,VRatecmd,'-r')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('VRate Cmd (m/s)')

subplot(2,1,2)
plot(Selectiontime(ERateCmdV < 0),ERateCmdV(ERateCmdV <0),'.r')
hold on
plot(Selectiontime(ERateCmdV >= 0),ERateCmdV(ERateCmdV >= 0),'.b')
ylabel('VRate Energy Rate Cmd (W)')
xlabel('t(s)')

figure
plot(Selectiontime, ERateCmd + PotentialEnergyRatecmd,'-r')
hold on
plot(Selectiontime, EnergyRate,'-g')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('Energy Rate (W)')
xlabel('t(s)')

h1 = plot(Selectiontime(1),ERateCmd(1) +
PotentialEnergyRatecmd(1),'.r');
h2 = plot(Selectiontime(1),EnergyRate(1),'.g');

legend([h1 h2],'Energy Rate Cmd','Energy Rate')

figure
subplot(2,1,1)
plot(Selectiontime(Eerrorneg),TASError(Eerrorneg),'.r')
```

```
hold on
plot(Selectiontime(Eerrorpos),TASError(Eerrorpos),'.b')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('TAS Error (m/s)')

subplot(2,1,2)
plot(Selectiontime(ERateErrorNeg),ERateError(ERateErrorNeg),'.r')
hold on
plot(Selectiontime(ERateErrorPos),ERateError(ERateErrorPos),'.b')
ylabel('Energy Rate Error(W)')
xlabel('t(s)')

figure
plot(Selectiontime,VRatecmd,'-r')
hold on
plot(Selectiontime,VRate,'-g')
ylabel('VRate (m/s')
xlabel('t(s)')

h1 = plot(Selectiontime(1),VRatecmd(1),'.r');
h2 = plot(Selectiontime(1),VRate(1),'.g');

legend([h1 h2],'VRate Cmd','VRate')

%Checks the selected time for any of the Airspeed Modes
LonMode = DevData.LonMode(idx);
AirMode = find(LonMode > 0,1);

if ~isempty(AirMode)

    msgbox('Warning, part of the selected time occured in Airspeed
Control. The Energy Calculations are invalid','Error')

end

end
end


% --- Executes on button press in TASvsThrottle.
function TASvsThrottle_Callback(hObject, eventdata, handles)

if strcmp(handles.flight,'')

    msgbox('Select an Energy .mat file')

else

load (handles.flight)

%Variables for the initial plots
Alt = DevData.Alt;
TAS = DevData.TAS;
Altcmd = DevData.AltCmd;
TAScmd = DevData.TASCmd;
LonMode = DevData.LonMode;
```

```
if isequal(get(handles.InputTime,'Value'),1)

    if isempty(handles.StartMin)

        msgbox('Need a value for Start Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.StartSec)

        msgbox('Need a value for Start Seconds')
        handles.quit = 'yes';

    elseif isempty(handles.EndMin)

        msgbox('Need a value for End Minutes')
        handles.quit = 'yes';

    elseif isempty(handles.EndSec)

        msgbox('Need a value for End Seconds')
        handles.quit = 'yes';

    else

    %If a time range is input the initial plots will use the time
period
    %otherwise the initial plots will use the entire time period of the
log file
    handles.quit = 'no';

    %Plots the initial plots within the input time period
    starttime = handles.StartMin*60+handles.StartSec;
    endtime = handles.EndMin*60+handles.EndSec;
    period = find(time >= starttime & time <= endtime);

    figure
    subplot(3,1,1)
    plot(time(period),Alt(period)*3.28084,'-g') %Alt converted from m
to ft
    hold on
    plot(time(period),Altcmd(period)*3.28084,'-r')
    set(gcf,'Name', 'Initial Plots')
    xlabel('Piccolo Time (s)')
    ylabel('Altitude (ft)')

    subplot(3,1,2)
    plot(time(period), LonMode(period),'-k')
    axis([min(time(period)) max(time(period)) -0.25 3.25])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')

    subplot(3,1,3)
    plot(time(period),TAS(period)*1.9438445,'-g') %TAS converted from
m/s to knots
    hold on
    plot(time(period),TAScmd(period)*1.9438445,'-r')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use either the Altitude or TAS plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

    end

else

    %Plots the initial plots for no input time period
    figure
    plot(time,Alt*3.28084,'-g') %Alt converted from m to ft
    hold on
    plot(time,Altcmd*3.28084,'-r')
    set(gcf,'Name', 'Altitude Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('Altitude (ft)')

    figure
    plot(time, LonMode,'-k')
    set(gcf,'Name', 'Lon Mode Initial Plot')
    axis([min(time) max(time) -0.25 3.25])
    set(gca,'YTick',[0 1 2 3 4])
    set(gca,'YTickLabel','Altitude|Airspeed|Slow Airspeed|Fast
Airspeed|')

    figure
    plot(time,TAS*1.9438445,'-g') %TAS converted from m/s to knots
    hold on
    plot(time,TAScmd*1.9438445,'-r')
    set(gcf,'Name', 'TAS Initial Plot')
    xlabel('Piccolo Time (s)')
    ylabel('TAS (knots)')

    %Allows the user time to zoom and pan on the altitude plot before
having to
    %select an area
    disp('Use either the Altitude or TAS plot to')
    disp('Select a range for analysis')
    disp('Hit Enter when ready to begin')
    disp('')
    i = input('');

    %Allows the user to select a time range by clicking and dragging on
the
    %plot
    k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);

    %Sets the plot range to the selected area on the plot
    idx = find(time >= x1 & time <= x2);

end

%If the input time is missing values the GUI will quit
if strcmp(handles.quit,'yes')

else

%Define all of the variables as a function of the selected time range
'idx'
TAS = DevData.TAS(idx);
TAScmd = DevData.TASCmd(idx);
Alt = DevData.Alt(idx);
Altcmd = DevData.AltCmd(idx);
Throttle = DevData.Throttle(idx);
Selectiontime = time(idx) - time(idx(1));
TASError = TAS - TAScmd;
Eerrorneg = find(TASError<0);
Eerrorpos = find(TASError>=0);

%Plot figures for analysis
figure
```

```
plot(Selectiontime, Alt*3.28084, '-g') %Alt converted from m to ft
hold on
plot(Selectiontime, Altcmd*3.28084,'-r')
ylabel('Altitude (ft)')
xlabel('t(s)')

figure
subplot(2,1,1)
plot(Selectiontime, TAS*1.9438445, '-g') %TAS converted from m/s to
knots
hold on
plot(Selectiontime, TAScmd*1.9438445,'-r')
title([handles.gains.Kpo ' ' handles.gains.Kpi ' ' handles.gains.KI ' '
handles.gains.TPT ' ' handles.gains.LPF])
ylabel('TAS (knots)')

subplot(2,1,2)
plot(Selectiontime(Eerrorneg), Throttle(Eerrorneg),'.r')
hold on
plot(Selectiontime(Eerrorpos), Throttle(Eerrorpos),'.b')
ylabel('Throttle')
xlabel('t(s)')

%Checks the selected time for any of the Airspeed Modes
LonMode = DevData.LonMode(idx);
AirMode = find(LonMode > 0,1);

if ~isempty(AirMode)

    msgbox('Warning, part of the selected time occured in Airspeed
Control')

end

end
end


% --- Executes on button press in Gas.
function Gas_Callback(hObject, eventdata, handles)

handles.MotorType = 'Gas';
set(handles.Gas,'Value',1)
set(handles.Electric,'Value',0)

guidata(hObject,handles)


% --- Executes on button press in Electric.
function Electric_Callback(hObject, eventdata, handles)

handles.MotorType = 'Electric';
set(handles.Gas,'Value',0)
set(handles.Electric,'Value',1)

guidata(hObject,handles)
```

7. LateralTracking

```
%By Anton Mornhinweg
%This gui uses the piccolo mat file cross track error and gps location
data
%with flight plan files to plot lateral tracking plots
%I used portions of the Lat-Lon plots code came from Cloud Cap's
'plotpiccolo.m'
%The function 'varargout' was generated automatically by MATLAB when I
%created the GUI
%All of the 'Create_Fcn' functions were generated by MATLAB when I
created
%the objects in GUIDE
```

MATLAB generated GUI functions

```
function varargout = LateralTracking(varargin)
% LATERALTRACKING M-file for LateralTracking.fig
%      LATERALTRACKING, by itself, creates a new LATERALTRACKING or
raises the existing
%      singleton*.
%
%      H = LATERALTRACKING returns the handle to a new LATERALTRACKING
or the handle to
%      the existing singleton*.
%
%      LATERALTRACKING('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in LATERALTRACKING.M with the given
input arguments.
%
%      LATERALTRACKING('Property','Value',...) creates a new
LATERALTRACKING or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before LateralTracking_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to LateralTracking_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help LateralTracking

% Last Modified by GUIDE v2.5 07-Feb-2014 18:11:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @LateralTracking_OpeningFcn, ...
                   'gui_OutputFcn',  @LateralTracking_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before LateralTracking is made visible.
function LateralTracking_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LateralTracking (see VARARGIN)

% Choose default command line output for LateralTracking
handles.output = hObject;

%Input time is selected on startup
set(handles.InputTime,'Value',1)

% Update handles structure
guidata(hObject, handles);

%Opening for the list box. Makes sure its loading the current directory
if nargin == 3,
    initial_dir = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
        else
            errordlg('Input argument must be a valid directory','Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument
Error!');
        return;
    end
end

%Launches the load listbox function
load_FlightPlans(initial_dir,handles)
load_listbox2(initial_dir,handles)

% --- Outputs from this function are returned to the command line.
```

```
function varargout = LateralTracking_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Piccolo mat file Listbox Functions (listbox2)

```
function load_listbox2(dir_path,handles)

%Loads piccolo mat files from the current directory into the listbox

%Clears the listbox and handles that are associated with file names
handles.file_names = {''};
set(handles.listbox2, 'String', '');
handles.flight = '';
handles.flightname = '';
guidata(handles.figure1,handles)
handles.flightfolder = dir_path;

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for .mat files in the current directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %Looks for the variable 'tClock'
        if ~isempty(whos('-file',name,'tClock*'))

            %Assigns the filename to the listbox by listbox index
number
            handles.file_names(i-count,1)=sorted_names(i);
            handles.sorted_index(i-count,1) = sorted_index(i);

        else

            count = count +1;

        end

    else
        count = count + 1;
    end

end

%Checks if theres any .mat files in the directory
if ~isfield(handles,'file_names')

    msgbox('There arent any plotpiccolo .mat workspace files to load')

else

    %Sets the directory to the current folder
    handles.is_dir = [dir_struct.isdir];
    handles.flightfolder = dir_path;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(handles.figure1,handles)

    %Assigns the piccolo mat file names to the listbox
    set(handles.listbox2,'String',handles.file_names,'Value',1)

    %Calls Initialize
    initialize_gui(handles);

end

function initialize_gui(handles)

%Selects first listbox item on startup

file_list = get(handles.listbox2,'String');
handles.flight = file_list{1};

filename4 = handles.flight;
remove = '.mat(\w*)';
handles.flightname = regexprep(filename4,remove,' ');

guidata(handles.figure1, handles);

% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)

%Saves the selected file into a handle 'handles.flight'
if ~strcmp(handles.flightname,'')

    index_selected = get(handles.listbox2,'Value');
    file_list = get(handles.listbox2,'String');
    handles.flightfilename = file_list{index_selected};

    %Flight handle includes the directory path and filename
    handles.flight = [handles.flightfolder '\' handles.flightfilename];

    %Saves just the filename, w/o extension, to a handle
'handles.flightname'
    filename4 = handles.flightfilename;
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CurrentDirectory_Callback(hObject, eventdata, handles)

%Current directory is initially the same as the matlab directory
%This inputbox displays the current directory
set(handles.CurrentDirectory,'String',handles.flightfolder)

% --- Executes during object creation, after setting all properties.
function CurrentDirectory_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in ChangeDirectory.
function ChangeDirectory_Callback(hObject, eventdata, handles)

%Changes directory for the listbox to load piccolo mat files

pathname = uigetdir; %User select directory from a dialog box

%If statement avoids error in case user clicks cancel
if ~pathname==0

    handles.flightfolder = pathname;
    set(handles.CurrentDirectory,'String',handles.flightfolder)
    guidata(hObject,handles)

end

%Launches reload listbox
reload_listbox2(handles)

function reload_listbox2(handles)

%Reloads the mat file listbox

%Clear the listbox
handles.file_names = {''};
handles.flightname = '';

%Prepares all files in directory to be loaded into listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
```

```
count = 0; %count adds each time the loop skips a file

%Loop searches for .mat files in the new directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.mat')

        %Looks for the variable 'tClock'
        if ~isempty(whos('-file',[handles.flightfolder '\' name
ext],'tClock*'))

            %Assigns the filename to the listbox by listbox index
number
            handles.file_names(i-count,1)=sorted_names(i);
            handles.sorted_index(i-count,1) = sorted_index(i);

        else

            count = count +1;

        end

    else
        count = count + 1;

    end

end

%Sets the directory handle to the new directory
handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.listbox2,'String',handles.file_names,'Value',1)

%Calls reinitialize
reinitialize_gui(handles);

function reinitialize_gui(handles)

%Selects first listbox item

file_list = get(handles.listbox2,'String');
handles.flight = file_list{1};
filename4 = handles.flight;

if isempty(filename4)

    msgbox('There arent any piccolo mat files in the current
directory')

else
```

```
    remove = '.mat(\w*)';
    handles.flightname = regexprep(filename4,remove,' ');
    handles.flight = [handles.flightfolder '\' filename4];

end

guidata(handles.figure1, handles);

%Launches the flight plan file reload
reload_FlightPlans(handles)

function reload_FlightPlans(handles)
%Reloads the flight plan listbox in the event of changing folders

handles.file_names = {''};

%Prepares all files in directory to be loaded into listbox
dir_struct = dir(handles.flightfolder);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for .fp files in the new directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.fp')

        %Assigns the filename to the listbox by listbox index number
        handles.file_names(i-count,1)=sorted_names(i);
        handles.sorted_index(i-count,1) = sorted_index(i);

    else

        count = count + 1;

    end

end

%Sets the directory handle to the new directory
handles.is_dir = [dir_struct.isdir];
guidata(handles.figure1,handles)
set(handles.FlightPlans,'String',handles.file_names,'Value',1)
```

Input Time

```
% --- Executes on button press in InputTime.
function InputTime_Callback(hObject, eventdata, handles)
%Radio button that is used to determine if the user wants to input a
time
%range
%The input time uses the gps time that piccolo telemetry records
```

```
function StartHour_Callback(hObject, eventdata, handles)

%Assigns input start hour to a handle

handles.Hstart = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function StartHour_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function StartMin_Callback(hObject, eventdata, handles)

%Assigns input start minute to a handle

handles.Mstart = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function StartMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function StartSec_Callback(hObject, eventdata, handles)

%Assigns input start second to a handle

handles.Sstart = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function StartSec_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function EndHour_Callback(hObject, eventdata, handles)

%Assigns input end hour to a handle

handles.Hend = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function EndHour_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function EndMin_Callback(hObject, eventdata, handles)

%Assigns input end minute to a handle

handles.Mend = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function EndMin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function EndSec_Callback(hObject, eventdata, handles)

%Assigns input end second to a handle

handles.SecEnd = str2double(get(hObject,'String'));
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function EndSec_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Gain Values

```
%The gain values are only used to be printed in the title of the
figures
function Track1_Callback(hObject, eventdata, handles)

%Assigns the input value of tracker convergence to a handle

a = get(hObject,'String');
handles.gains.TC = ['TC = ' a];

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Track1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Track2_Callback(hObject, eventdata, handles)

%Assigns the input value of heading error to turn rate to a handle

a = get(hObject,'String');
```

```
handles.gains.HETR = ['Kp = ' a];

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Track2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Track3_Callback(hObject, eventdata, handles)

%Assigns the input value of heading error der to turn rate to a handle

a = get(hObject,'String');
handles.gains.HEDTR = ['Kd = ' a];

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Track3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Track4_Callback(hObject, eventdata, handles)

%Assigns the input value of turn err lpf cutoff to a handle

a = get(hObject,'String');
handles.gains.LPF = ['LPF = ' a];

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Track4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Waypoint Functions

```
function Lat_Callback(hObject, eventdata, handles)

%Assigns the input value of waypoint latitude to a handle

handles.lat = str2double(get(hObject,'String'));

if isnan(handles.lat)

    handles.lat = '';
    set(handles.Lat,'String','') %clears the input box on the gui
```

```
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Lat_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Lon_Callback(hObject, eventdata, handles)

%Assigns the input value of waypoint longitude to a handle

handles.lon = str2double(get(hObject,'String'));

if isnan(handles.lon)

    handles.lon = '';
    set(handles.Lon,'String','') %clears the input box on the gui

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function Lon_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function WaypointNumb_Callback(hObject, eventdata, handles)

%Assigns the input value of waypoint number to a handle

handles.numb = str2double(get(hObject,'String'));

if isnan(handles.numb)

    handles.numb = '';
    set(handles.WaypointNumb,'String','') %clears the input box on the
gui

end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function WaypointNumb_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

% --- Executes on button press in AddWaypoint.
function AddWaypoint_Callback(hObject, eventdata, handles)

%Adds waypoint locations entered manually to handles

%If conditions make sure the input values are numbers
if isempty(handles.numb)

    msgbox('Waypoint number must be a number','Error')

elseif isempty(handles.lat)

    msgbox('Latitude must be a number','Error')

elseif isempty(handles.lon)

    msgbox('Longitude must be a number','Error')

else

%Adds waypoint data to handles
if ~isfield(handles,'latitude')

    %Creates the handles if its the first waypoint
    handles.latitude(1,1) = handles.lat;
    handles.longitude(1,1) = handles.lon;
    handles.waypointnumber(1,1)= handles.numb;

else

    %Assigns waypoint data to handles
    [m,n]=size(handles.latitude);
    i = m;

    if isequal(n,0)
        handles.latitude(i,1) = handles.lat;
        handles.longitude(i,1) = handles.lon;
        handles.waypointnumber(i,1) = handles.numb;
    else
    handles.latitude(i+1,1) = handles.lat;
    handles.longitude(i+1,1) = handles.lon;
    handles.waypointnumber(i+1,1) = handles.numb;
    end

end

%Update the waypoint listbox
update_listbox(handles)

end

guidata(hObject,handles)

function update_listbox(handles)
```

```
%Sets the waypoint listbox to list the waypoint numbers
set(handles.listbox1,'String',handles.waypointnumber)

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

%Lists waypoint numbers of the flight plan file that has been loaded or
%waypoints that have been entered manually

index_selected = get(handles.listbox1,'Value');
set(handles.Lat,'String',num2str(handles.latitude(index_selected),10));
set(handles.Lon,'String',num2str(handles.longitude(index_selected),10))
;
set(handles.WaypointNumb,'String',handles.waypointnumber(index_selected
));

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in Remove.
function Remove_Callback(hObject, eventdata, handles)

%Removes the highlighted waypoint from the waypoint listbox and handles

index_selected = get(handles.listbox1,'Value');

for i = index_selected:length(handles.waypointnumber)

    if isequal(i,length(handles.waypointnumber))

        handles.waypointnumber(i) = '';
        handles.latitude(i) = '';
        handles.longitude(i) = '';

    else

       handles.waypointnumber(i,1) = handles.waypointnumber(i+1,1);
       handles.latitude(i,1) = handles.latitude(i+1,1);
       handles.longitude(i,1) = handles.longitude(i+1,1);

    end

end

set(handles.listbox1,'Value',1);
guidata(hObject,handles)
update_listbox(handles)

% --- Executes on selection change in FlightPlans.
function FlightPlans_Callback(hObject, eventdata, handles)
```

```
%Loads the flight plan file that is selected in the flight plan file
listbox

index_selected = get(handles.FlightPlans,'Value');
file_list = get(handles.FlightPlans,'String');
handles.flightplan = file_list{index_selected};

%Imports the data from the fp text file
flightplandata = importdata([handles.flightfolder '\'
handles.flightplan],' ',2);

%Loop assigns data to the waypoint handles
for i = 1:size(flightplandata.data,1)

    handles.latitude(i,1) = flightplandata.data(i,3);
    handles.longitude(i,1) = flightplandata.data(i,4);
    handles.waypointnumber(i,1) = flightplandata.data(i,1);

end

%If the flight plan loaded previously had more waypoints this loop
removes
%the excess waypoints
difference = length(handles.waypointnumber)-
size(flightplandata.data,1);
if difference > 0

    for i = 1:difference

        handles.latitude(length(handles.waypointnumber)) = '';
        handles.longitude(length(handles.waypointnumber)) = '';
        handles.waypointnumber(length(handles.waypointnumber)) = '';

    end

end

%Load flight plan listbox
update_listbox(handles)

set(handles.WaypointNumb,'String','')
set(handles.Lat,'String','')
set(handles.Lon,'String','')

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function FlightPlans_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function load_FlightPlans(dir_path,handles)
```

```
%Loads flight plan files into the flight plan file listbox

%Prepares all files in directory to be loaded into listbox
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');

count = 0; %count adds each time the loop skips a file

%Loop searches for .fp files in the directory
for i=1:length(sorted_index)

    %Fileparts will return the extension of the file
    %sorted_names had to be converted to a character to work in
fileparts
    [pathstr, name, ext]=fileparts(char(sorted_names(i)));

    if strcmp(ext,'.fp')

        %Assigns the filename to the listbox by listbox index number
        handles.file_names(i-count,1)=sorted_names(i);
        handles.sorted_index(i-count,1) = sorted_index(i);

    else

        count = count + 1;

    end

end

%Checks if theres any .fp files in the directory
if ~isfield(handles,'file_names')

    msgbox('There arent any FlightPlan .fp files to load')

else

    handles.is_dir = [dir_struct.isdir];
    guidata(handles.figure1,handles)
    set(handles.FlightPlans,'String',handles.file_names,'Value',1)

end

% --- Executes on button press in Preview.
function Preview_Callback(hObject, eventdata, handles)

%Plots the flight path of the waypoint data

if isfield(handles,'waypointnumber')
if
xor(isnan(handles.waypointnumber),isequal(length(handles.waypointnumber
),1))
else

figure
for i = 1:length(handles.waypointnumber)
```

```
    if i < length(handles.waypointnumber)

            x2 = handles.longitude(i+1);
            y2 = handles.latitude(i+1);

    else

        x2 = handles.longitude(1);
        y2 = handles.latitude(1);

    end

    x1 = handles.longitude(i);
    y1 = handles.latitude(i);


    plot([x1 x2],[y1 y2])
    hold on

end
end
end

% --- Executes on button press in Run.
function Run_Callback(hObject, eventdata, handles)

%Begins the Lat Lon lateral tracking plot process

if strcmp(handles.flight,'')

    msgbox('Select a piccolo mat file')

else

graphpath(handles)

%Plots the flight plan if waypoints exist
if isfield(handles,'waypointnumber')
if
xor(isnan(handles.waypointnumber),isequal(length(handles.waypointnumber
),1))
else

for i = 1:length(handles.waypointnumber)

    if i < length(handles.waypointnumber)

            x2 = handles.longitude(i+1);
            y2 = handles.latitude(i+1);

    else

        x2 = handles.longitude(1);
        y2 = handles.latitude(1);

    end
    x1 = handles.longitude(i);
    y1 = handles.latitude(i);

    plot([x1 x2],[y1 y2],'-k')
    hold on

end
end
end
end


function graphpath(handles)

%Performs the Lat Lon lateral tracking plots

%Checks if gain values were input
%If they are blanks they wont be printed on the plots
if ~isfield(handles,'gains')

    handles.gains.LPF = '';

end

if ~isfield(handles.gains, 'LPF')

    handles.gains.LPF = '';

end

if ~isfield(handles.gains, 'HETR')

    handles.gains.HETR = '';

end

if ~isfield(handles.gains, 'HEDTR')

    handles.gains.HEDTR = '';

end

if ~isfield(handles.gains, 'TC')

    handles.gains.TC = '';

end

load(handles.flight)

rad2deg = 180/pi;
deg2rad = pi/180;

%Sets the start and endtimes for the analysis
if isequal(get(handles.InputTime,'Value'),1)
```

```
    %Checks that all of the time parameters have been input
    if ~isfield(handles,'Hstart')

        msgbox('Must input start hours')

    elseif ~isfield(handles,'Mstart')

        msgbox('Must input start minutes')

    elseif ~isfield(handles, 'Sstart')

        msgbox('Must input start seconds')

    elseif ~isfield(handles,'Hend')

        msgbox('Must input end hours')

    elseif ~isfield(handles,'Mend')

        msgbox('Must input end minutes')

    elseif ~isfield(handles,'SecEnd')

        msgbox('Must input end seconds')

    else

    %Determines beginning and ending row numbers inside the given time
period
    for i=1:length(dat.Lon)
       if dat.Hours(i) == handles.Hstart
        if dat.Minutes(i) == handles.Mstart
            if xor(dat.Seconds(i) == handles.Sstart,dat.Seconds(i-1) <
handles.Sstart && dat.Seconds(i) > handles.Sstart)

                starttime = i; %represents the start row number

            end
        end
       end

       if dat.Hours(i) == handles.Hend
        if dat.Minutes(i) == handles.Mend
            if xor(dat.Seconds(i) == handles.SecEnd,dat.Seconds(i-1) <
handles.SecEnd && dat.Seconds(i) > handles.SecEnd)

                endtime = i; %represents the end row number

            end
        end
       end

    end
    end

else
```

```
%If there is no input time the period spans the entire flight
starttime = 1;
endtime = length(dat.Clock);

end

%If statements make sure that the start and end time variables were
%assigned values
if ~exist('starttime','var')

        msgbox('Start and end times not valid')

elseif ~exist('endtime','var')

        msgbox('Start and end times not valid')

else

%Plots GPS Latitude vs Longitude
figure
    latlonfig=gcf;

    %Defines the plot boundaries as the max and min latitude and
longitudes
    %that the aircraft flew and adds a little space to the boundaries
    a1 = min(dat.Lon(find(dat.PosGood ==1)))-0.000004;
    a2 = max(dat.Lon(find(dat.PosGood ==1)))+0.000004;
    b1 = min(dat.Lat(find(dat.PosGood ==1)))-0.000004;
    b2 = max(dat.Lat(find(dat.PosGood ==1)))+0.000004;

    %Lines 1003 - 1050 were copied from Cloud Cap's 'plotpiccolo.m'
lines
    %427-502 with some modifications
    hold on
    xlabel('Lon')
    ylabel('Lat')
    axis('equal')
    axis([a1 a2 b1 b2]*rad2deg);
    grid
    set(gcf,'Name',[handles.flightname])

    red=[1 0 0];
    yell = [1 1 0];
    grn=[0 1 0];
    blu = [0 0 1];
    orange = [1 1/2 0];
    lblu = [0 1 1];
    purpl = [3/4 0 1];
    grey=[1/2 1/2 1/2];
    blk=[0 0 0];


    %Loop defines colors for different cross track errors
    rssicolor=[];
    for i=starttime:endtime,
        TrackError=abs(dat.Track_Y(i)*3.28084); %track error converted
to ft
```

```
        if (TrackError>200),
            col=red;
        elseif (TrackError>=100),
            col=purpl;
        elseif (TrackError>=50),
            col=grey;
        elseif (TrackError>=25),
            col=orange;
        elseif (TrackError>=10),
            col=lblu;
        elseif (TrackError>=5),
            col=blu;
        elseif (TrackError<5),
            col=grn;
        end
        rssicolor=[rssicolor; col];
    end
    for (i=starttime:endtime),

plot(dat.Lon(i)*rad2deg,dat.Lat(i)*rad2deg,'.','Color',rssicolor(i-
starttime+1,:))
        hold on
    end

    axis('equal')
    axis([a1 a2 b1 b2]*rad2deg);
    title([handles.gains.TC ' ' handles.gains.HETR ' '
handles.gains.HEDTR ' ' handles.gains.LPF])

    hold on

    %Lines 1056 - 1089 create a wind arrow to provide the average wind
    %speed and direction during the time period
    WindSouth = mean(dat.WindSouth(starttime:endtime)); %Positive wind
south is wind blowing north
    WindWest = mean(dat.WindWest(starttime:endtime)); %Positive wind
west is wind blowing east
    angle = atan(WindSouth/WindWest);
    WindSpeed = round(sqrt(WindSouth^2 + WindWest^2)/0.44704); %Wind
speed is converted to mph

    x1 = 0.85; %Sets start point of wind vector at 85% of the figure in
the x - direction
    y1 = 0.50; %Sets start point of wind vector at 50% of the figure in
the y - direction

    %If statements determine which direction the arrow should point
    %Each quadrant has its own equations for calculating x2 and y2
    if WindSouth < 0
        if WindWest < 0
           x2 = x1 - cos(angle)*0.1;
           y2 = y1 - sin(angle)*0.1;
        else
           angle = angle*-1;
           x2 = x1 + cos(angle)*0.1;
           y2 = y1 - sin(angle)*0.1;
        end
    end

    if WindSouth > 0
        if WindWest < 0
            angle = angle * -1;
            x2 = x1 - cos(angle)*0.1;
            y2 = y1 + sin(angle)*0.1;
        else
            x2 = x1 + cos(angle)*0.1;
            y2 = y1 + sin(angle)*0.1;
        end
    end

    %Prints the arrow with text on the figure
    annotation('textarrow',[x1 x2],[y1 y2],'String',sprintf('Avg Wind
%g mph',WindSpeed))

    %Defines legend
    h1=plot(0,0,'.');
    h2=plot(0,0,'.');
    h3=plot(0,0,'.');
    h4=plot(0,0,'.');
    h5=plot(0,0,'.');
    h6=plot(0,0,'.');
    h7=plot(0,0,'.');

    set(h1,'Color',red);
    set(h2,'Color',purpl);
    set(h3,'Color',grey);
    set(h4,'Color',orange);
    set(h5,'Color',lblu);
    set(h6,'Color',blu);
    set(h7,'Color',grn);

    legend([h1 h2 h3 h4 h5 h6 h7],'Cross > 200','Cross >= 100','Cross
>= 50','Cross >= 25','Cross >= 10', 'Cross >= 5', 'Cross < 5')
end

% --- Executes on button press in GraphSelection.
function GraphSelection_Callback(hObject, eventdata, handles)

%Plots the user selected area from the lat lon plot as cross track vs
time
if strcmp(handles.flight,'')

    msgbox('Select a piccolo mat file')

else

load(handles.flight)

%Checks if gain values were input
%If they are blanks they wont be printed on the plots
if ~isfield(handles,'gains')

    handles.gains.LPF = '';
```

```
end

if ~isfield(handles.gains, 'LPF')

    handles.gains.LPF = '';

end

if ~isfield(handles.gains, 'HETR')

    handles.gains.HETR = '';

end

if ~isfield(handles.gains, 'HEDTR')

    handles.gains.HEDTR = '';

end

if ~isfield(handles.gains, 'TC')

    handles.gains.TC = '';

end

disp('')
disp('Select flight path leg for further examination')
disp('Make sure the selection begins with')
disp('the waypoint being targeted for the leg')
disp('Make sure that the selected area ends after the next waypoint has
been targeted')
disp('')

%Allows the user to select a latitude and longitude range to define a
leg
%of the flight for analysis
k = waitforbuttonpress;
    point1 = get(gca,'CurrentPoint');    % button down detected
    finalRect = rbbox;                   % return figure units
    point2 = get(gca,'CurrentPoint');    % button up detected
    point1 = point1(1,1:2);              % extract x and y
    point2 = point2(1,1:2);
    p1 = min(point1,point2);             % calculate locations
    offset = abs(point1-point2);         % and dimensions
    x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
    y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];
    hold on
    axis manual
    x1=min(x);
    x2=max(x);
    y1 = min(y);
    y2 = max(y);

%Sets the start and endtimes for the analysis
if isequal(get(handles.InputTime,'Value'),1)
```

```
    %Checks that all of the time parameters have been input
    if ~isfield(handles,'Hstart')

        msgbox('Must input start hours')

    elseif ~isfield(handles,'Mstart')

        msgbox('Must input start minutes')

    elseif ~isfield(handles, 'Sstart')

        msgbox('Must input start seconds')

    elseif ~isfield(handles,'Hend')

        msgbox('Must input end hours')

    elseif ~isfield(handles,'Mend')

        msgbox('Must input end minutes')

    elseif ~isfield(handles,'SecEnd')

        msgbox('Must input end seconds')

    else

    %Determines starttime and endtime again
    for (i=1:length(dat.Lon)),
       if dat.Hours(i) == handles.Hstart
        if dat.Minutes(i) == handles.Mstart
            if xor(dat.Seconds(i) == handles.Sstart,dat.Seconds(i-1) <
handles.Sstart && dat.Seconds(i) > handles.Sstart)

                starttime = i;

            end
        end
       end

       if dat.Hours(i) == handles.Hend
        if dat.Minutes(i) == handles.Mend
            if xor(dat.Seconds(i) == handles.SecEnd,dat.Seconds(i-1) <
handles.SecEnd && dat.Seconds(i) > handles.SecEnd)

                endtime = i;

            end
        end
       end

    end
    end
else

%If there is no input time the period spans the entire flight
starttime = 1;
```

```
endtime = length(dat.Clock);

end

%If statements make sure that the start and end time variables were
%assigned values
if ~exist('starttime','var')

        msgbox('Start and end times not valid')

elseif ~exist('endtime','var')

        msgbox('Start and end times not valid')

else

    %Searches for the data points that fall within the latitude and
    %longitude area in addition to the selected time period
    idx = find(tClock >= tClock(starttime) & tClock <= tClock(endtime)
& dat.Lon >= x1*pi/180 & dat.Lon <= x2*pi/180 & dat.Lat >=y1*pi/180 &
dat.Lat <= y2*pi/180);

    count2 = 0;
    count = 0;
    for i = 2:length(idx)

        %Calculates the average settling offset
        if dat.TrackerTarget(idx(i)) ~= dat.TrackerTarget(idx(i-1))

            if dat.TrackerTarget(idx(i-1)) == dat.TrackerTarget(idx(1))

                count = count + 1;

                TrackAvg(count,1) = abs(dat.Track_Y(idx(i-1)));

            end

        end

        %Defines x and y variables for the cross track error vs time
plot
        if dat.TrackerTarget(idx(i)) == dat.TrackerTarget(idx(1))

            count2 = count2 + 1;
            timetoplot(count2,1) = tClock(idx(i));
            Ytoplot(count2,1) = dat.Track_Y(idx(i));

        end
    end

    %Convert units from meters to feet
    Ytoplot = Ytoplot / 0.3048;
    %TrackAvg = TrackAvg / 0.3048;

    %Sets max and mins for the plot axes
    b1 = min(Ytoplot);
    b2 = max(Ytoplot);
    a1 = min(timetoplot);
    a2 = max(timetoplot);

    if b1 < -100

        b1 = -100;

    end

    if b2 > 100

        b2 = 100;

    end

    %Plots cross track error vs time
    figure
    plot(timetoplot,Ytoplot,'b.')
    hold on
    plot([a1 a2],[0 0],'g')
    axis([a1 a2 b1 b2]);
    title([handles.gains.TC ' ' handles.gains.HETR ' '
handles.gains.HEDTR ' ' handles.gains.LPF])

    %My axes got flipped when i used the 'axis' command. So i just
swapped
    %their labels
    ylabel('Cross Track Error (ft)')
    xlabel('Piccolo Time (s)')
    disp('The Average Convergence Offset')
    %disp(sprintf('of the selected leg = %1g ft',mean(TrackAvg)))

end
end
```

8. FuelCorrection

```
%By Anton Mornhinweg
%Fuel Correction loads the estimated fuel mass from the plotpiccolo mat
%file and corrects it based on how much fuel was actually burned in
flight
%Additionally it also calculates the ESFC for the flight and saves both
%variables in the plotpiccolo mat file

%User select Plotpiccolo .mat file
[filename3,pathname3]=uigetfile({'*.mat','Piccolo mat Files
(*.mat)'},'Select the Flight Piccolo Mat file');

%Loads the mat file
load ([pathname3 filename3])

%User input variables to calculate mass of fuel burned
MaxPower = str2double(inputdlg('What is Max Power(w)?','Fuel
Correction'))/1000; %converts units from W to kW
```

```
GTOW = str2double(inputdlg('What is takeoff weight(lb)?','Fuel
Correction'))*453.5924; %convert units from lb to g
EmptyWeight = str2double(inputdlg('What is empty weight(lb)?','Fuel
Correction'))*453.5924;
FinalWeight = str2double(inputdlg('What is final weight(lb)?','Fuel
Correction'))*453.5924;

%Assign variables
FuelBurn = (GTOW - FinalWeight); %Fuel Burn is in grams because ESFC is
g/kWhr
Throttle = dat.Surface2; %Change surface number if throttle is not
'Surface2'
t = dat.Clock/1000/3600; %Time units is hours

%Loop calculates the kWhr as %Throttle * MaxPower(kW)* t(hr)
kWhr(1,1) = 0;
for i = 2:length(Throttle)

    kWhr(i,1) = Throttle(i-1,1)*MaxPower*(t(i,1) - t(i-1));

end

ESFC = FuelBurn/sum(kWhr); % g /(kW-hr)

CorrectedFuel(1,1) = (GTOW - EmptyWeight)/1000; %units back to kg as
the piccolo records fuel estimate in kg

for i = 2:length(dat.Clock)

    CorrectedFuel(i,1) = CorrectedFuel(i-1,1)-ESFC*kWhr(i,1)/1000;

end

a = inputdlg ('Save? y or n');

%If the user wants to save will save the new variables in the piccolo
mat file
if strcmp(a,'y')

    %Determines if piccolo mat file has corrected fuel variables
    if exist('RPMFiltered','var')

        %Saves over the original piccolo mat file in the same location
it was loaded
        save([pathname3
filename3],'dat','tClock','ESFC','CorrectedFuel','RPMFiltered','RPMRate
Max')

    else

        %Saves over the original piccolo mat file in the same location
it was loaded
        save([pathname3
filename3],'dat','tClock','ESFC','CorrectedFuel')

    end
```

```
end

%Clear the excess variables so the user can view the new variables
clearvars('-except','ESFC','CorrectedFuel')
```

9. PiccoloRPMRateFilter

```
%By Anton Mornhinweg
%Filters RPM data by the same method that the piccolo RPM rate filter
uses
%The RPM rate filter value is treated as the max frequency for rpm data
so
%any changes in rpm that exceed this limit are ignored and removed from
the
%data

%User select Plotpiccolo .mat file
[filename3,pathname3]=uigetfile({'*.mat','Piccolo mat Files
(*.mat)'},'Select the corresponding Piccolo Data mat file');

load([pathname3 filename3])

%Define variables from dat structure
RPM = dat.LeftRPM;
time = dat.Clock/1000;
c = 'n';
%clearvars('-except','RPM','time')

figure (1)
plot(time, RPM,'-k')
title('RPM Data')

%First loop defines the number of times that the rpm data is filtered
so
%that the user can change the rate filter to view how the rpm data
would be
%filtered
for i = 1:1000

    if i > 1

        c = input('Continue? y/n ','s');

        if ~strcmp(c,'y')
            break
        end

    end

    RPMRateMax = input('What is RPM Rate limit (RPM/s)? ');

    %Loop removes RPM data that violates the max rate
    RPMFiltered = RPM;
    for j = 2:length(RPM)
```

```
        if abs(RPMFiltered(j) - RPMFiltered(j-1))/(time(j) - time(j-1))
> RPMRateMax

            RPMFiltered(j) = RPMFiltered(j-1);

        end

    end

    %Plots the new filtered RPM
    figure
    plot(time, RPMFiltered,'.k')
    set(gcf,'Name',['RPM rate filter = ' num2str(RPMRateMax) ' RPM/s'])
    title(['RPM Filtered at ' num2str(RPMRateMax) ' RPM/s'])

end

a = inputdlg ('Save? y or n');

%If the user wants to save will save the new variables in the piccolo
mat file
if strcmp(a,'y')

    %Determines if piccolo mat file has corrected fuel variables
    if exist('ESFC','var')

        %Saves over the original piccolo mat file in the same location
it was loaded
        save([pathname3
filename3],'dat','tClock','ESFC','CorrectedFuel','RPMFiltered','RPMRate
Max')

    else

        %Saves over the original piccolo mat file in the same location
it was loaded
        save([pathname3
filename3],'dat','tClock','RPMFiltered','RPMRateMax')

    end

end

clear
```

APPENDIX F

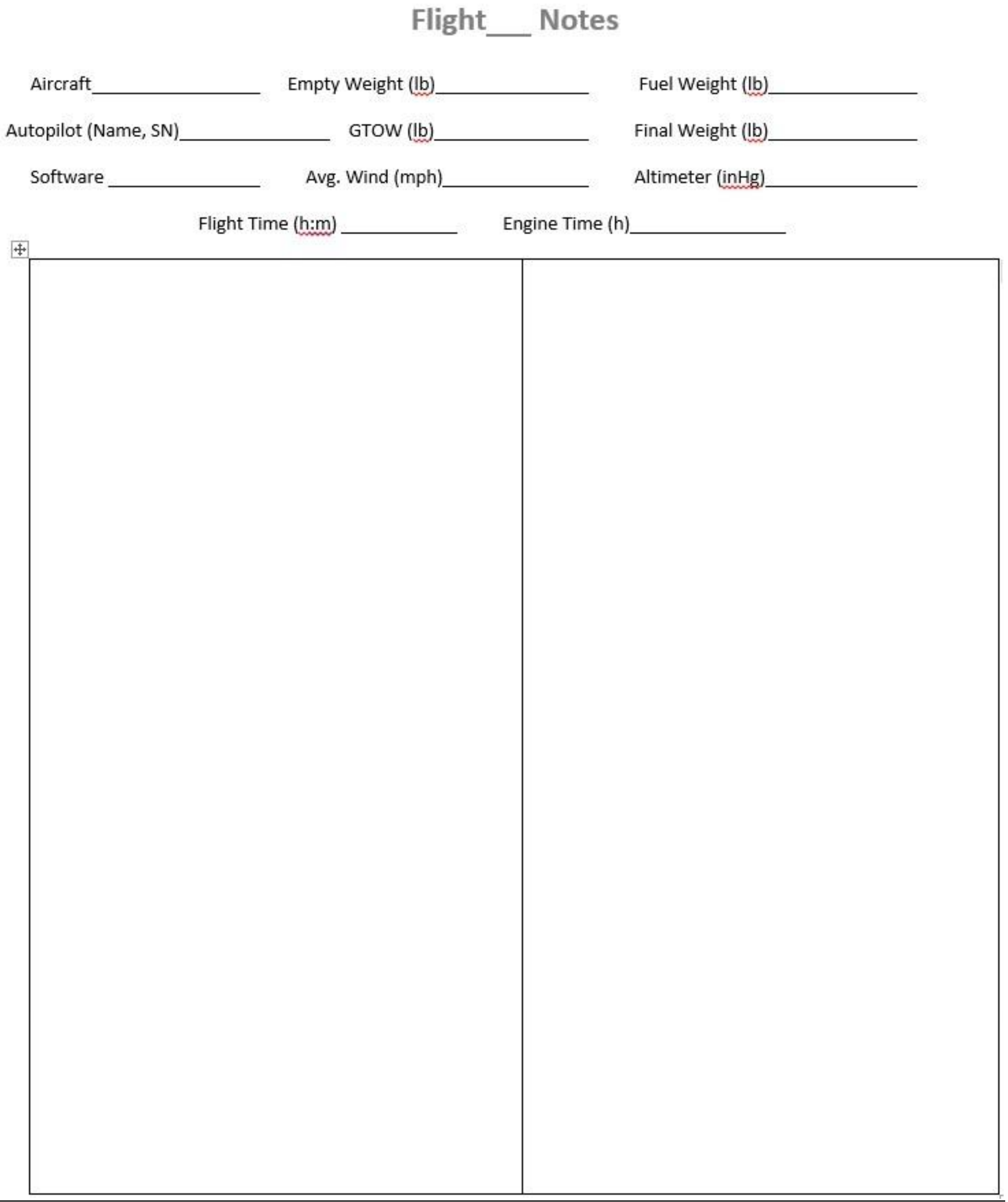
FLIGHT SHEETS

1. Pre – Flight Checklist

Before Powering the motor	
	Set Telemetry bandwidth to 1Hz
	Check Piccolo and Servo Voltages and Current draw (Piccolo System window)
	Make sure Autopilot is in Pre-Launch Mode
	Check Piccolo Radio Settings (915, 1, 1)
	Check Ground Station Radio Settings (915,1,1)
	Check Link and Pilot Sample
	Verify Limits and Gains are Correct - Load each gain and limit tabs in the controller configuration window from known saved configuration files
	Verify Land Settings - Load appropriate land settings from known saved configuration file
	Verify Launch Settings - Load appropriate launch settings from known saved configuration file
	Verify Surface Calibration - Load known control surfaces configuration file - Click on each control surface to make sure that none of the pulse width deflection values have changed
	Verify Payload IO Settings - Load known configuration file
	Verify Payload Com Settings - Load known configuration file
	Verify Sensor Configuration - Load known configuration file
	Verify Command Loops AUTO
	Verify Mission Limits
	Uncheck GPS box to set Barometer for Altitude Control
	Set Local Pressure (Pa) in Altimeter Box

	Cup Pitot Tube and Zero Air Data
	Test Autopilot Surface Deflections - Command full deflections in Pre – Flight window - Check surface telemetry for commanded deflections - Have someone out by the plane calling back the surfaces and direction they were deflected
	Set Gross and Empty Mass
	Check IMU (plane dance)
	Check GPS satellites - Make sure the solution is in 3D mode
	Verify Flight Plans - Load in any new flight plans - Check the current flight plans
	Verify Barometer Altitude - Check that the altitude reported by the barometer is the altitude of the runway - Re – zero air data if needed
	Start Engine
	AP Full Throttle Test
	Manual Control Test
	Test Engine Kill Switch
	Check Link and Pilot Sample
	Enable Controller Telemetry
	Set all telemetry to 25 Hz
	Ready for Takeoff

2. Flight Notes



Flight___ Notes

Aircraft________________ Empty Weight (lb)_______________ Fuel Weight (lb)______________

Autopilot (Name, SN)______________ GTOW (lb)_______________ Final Weight (lb)______________

Software _______________ Avg. Wind (mph)_______________ Altimeter (inHg)______________

Flight Time (h:m) ___________ Engine Time (h)_______________

3. Doublet Maneuvers

Aircraft____________________

Flight______________________

Doublet	Duration (s)	Period (ms)	Deflection (deg)

4. Airspeed Control

Gain	Definition	Too High	Too Low
TAS err to TAS Rate	Gain relating true air speed error [m/s] to true air speed rate command [m/s/s]. Increasing this gain will increase the bandwidth with which the autopilot tries to control airspeed. This gain must not be zero. Gain that sets the desired true airspeed rate based upon the true airspeed error. This gain is the airspeed outer loop which simply sets the desired rate of change of true airspeed based upon the true airspeed error. The default value for this gain is 0.15. Increasing this gain will produce more aggressive airspeed tracking and vice versa. If the gain is too high a pitch oscillation will result.	Pitch oscillations	
TAS Rate error to Accel Cmd	Gain relating true air speed rate error [m/s/s] to Z acceleration command [m/s/s]. If the vehicle is not changing airspeed at the desired rate this gain causes the flight path trajectory to curve up or down as needed to correct this problem. This gain must not be zero. This gain is only used when the autopilot is in airspeed mode. Most of the time the autopilot is in altitude mode, unless it detects a power problem. Gain that computes the vertical acceleration command (or Z – Acceleration command) based upon the airspeed rate error. The theory behind this gain is that if the airspeed is not changing at the rate we want then the vehicles trajectory should be curved – i.e. a vertical acceleration should be commanded. Hence if we are speeding up too quickly command a vertical acceleration which will cause the vehicle to pull up so that gravity can slow us down and vice versa. The default value for this gain is 1.5. Since the acceleration command appears directly on the elevator (via the elevator prediction trust) making this gain too high can cause pitch oscillations. Making this gain too small will impair the ability of the autopilot to regulate airspeed.	Pitch oscillations	No vrate response to TAS Rate errors
TAS Rate err to Accel der Cmd	Gain relating true air speed rate error derivative [m/s/s/s] to Z acceleration command [m/s/s]. This gain can be zero but slightly better airspeed control can be achieved if it is not zero. This gain is only used when the autopilot is in airspeed mode. Most of the time the autopilot is in altitude mode, unless it detects a power problem.		

Flight

TAS error to TAS rate command	TAS rate error to accel cmd	TAS rate err to accel der cmd

Start Time (M:S)	

End Time (M:S)	

Notes

5. Altitude Control

Alt err to alt rate	Gain relating altitude error (m) to commanded altitude rate (m/s). Must not be zero.	Too High	Too Low
Alt rate err to accel	Gain relating altitude rate error (m/s) to Z acceleration command (m/s/s). This gain sets the bandwidth with which the vehicle tries to achieve the desired vertical rate. It must not be zero. In most cases this gain must be at least as large as "Alt err to alt rate". The theory behind this gain is that an error in vertical rate is corrected by accelerating up or down. The gain has a default value of 0.75, hence most of the vertical rate command should be achieved in something like 2 seconds. Increasing this gain will increase the bandwidth of the altitude controller. Since the acceleration command appears directly on the elevator (via the elevator prediction trust) making this gain too high can cause pitch oscillations. Making this gain too small will impair the ability of the autopilot to regulate altitude.	Pitch oscillations	No elevator response to vrate errors
Elevator Prediction Trust	Ratio (0.0 – 1.0) describing how much to trust the elevator prediction from vehicle parameters, from 0.0 (no trust) to 1.0 (full trust). Lower numbers are safer, higher numbers perform better. When using high elevator prediction trust values the Z acceleration error integral to elevator must be strong enough to overcome errors in prediction, otherwise the vehicle could diverge due to miss predicting elevator motion. Gain that determines how much of the predicted elevator should actually be used. The autopilot computes the amount of elevator required to achieve a given acceleration based upon the elevator effectiveness, the measured dynamic pressure, the current mass estimate, and the wing area of the vehicle. The elevator effectiveness is like a proportional gain. However sometimes the vehicle plant dynamics require that the proportional gain be smaller than the elevator effectiveness implies. This is due to the time delay that occurs between elevator command and change in acceleration. For those cases the elevator predictive trust can be reduced. The elevator predictive trust can go from 0.0 (no prediction) to 1.0 (full prediction). For most vehicles the best performance occurs when the predictive trust is at 1.0. For vehicles with large time delays in the elevator actuation (high altitude, slow actuators, strange aerodynamics) the predictive trust should be reduced. If fast oscillations in pitch are observed, and the elevator effectiveness and mass estimate are correct, then reduce the predictive trust.	Fast pitch oscillations	
Acceleration lpf cutoff	Z acceleration low pass filter cutoff frequency [Hz]. Zero to disable. This filter can be used to remove noise on the z-acceleration measurement. Enabling this filter will reduce the vertical rate control bandwidth.		
Accel err to elevator	Gain relating the Z acceleration error [[m/s/s]*s] to elevator. When elevator prediction trust is used this gain can be zero. Increase this gain to improve elevator responsiveness and reduce overshoot on acceleration control due to integral wind up.		
Accel err int to elevator	Gain relating the integral of Z acceleration error [[m/s/s]*s] to elevator. This is the primary gain that moves the elevator and must not be zero. In particular this gain must be strong enough to overcome elevator prediction errors. This gain should be as high as practical in order to maximize the bandwidth of the vertical acceleration and vertical rate control. Gain that computes the elevator deflection as the time integral of the acceleration error. The acceleration error integrator is used to find the errors in	Fast pitch oscillation	Speed Divergence

	the elevator prediction. The default value for this gain is 1.5. Increasing this gain will compensate for errors in the elevator predictor and will improve the altitude and airspeed performance. Too much gain will cause a fast pitch oscillation. Vehicles with low elevator bandwidth may need to reduce this gain. If this gain is too low, relative to the predictive trust, the vehicle may enter a speed divergence.		

5.1.1. Altitude Control Loop

Flight	Start Time (M:S)		End Time (M:S)		Alt err to alt rate

Notes

5.1.2. Z – Acceleration Control Loop

Flight

Start Time (M:S)	

End Time (M:S)	

Alt rate err to accel	Elevator Prediction Trust	Acceleration lpf cutoff	Accel err to elevator	Accel err int to elevator	Accel cmd lpf cutoff

Notes

6. Energy Control

Gain	Definition	Too High	Too Low
Energy err to Energy Rate	Gain relating energy error to energy rate command. Increasing this gain increases how aggressively the autopilot moves the throttle to maintain energy.	Throttle oscillations	
Energy Rate err to Throttle	Gain relating the energy rate error to throttle.	Throttle oscillations	
Energy Rate err int to Throttle	Gain relating the integral of energy rate error to throttle. This gain must not be zero since it sets the throttle trim.		
Throttle Prediction Trust	Ratio (0.0 - 1.0) describing how much to trust the predicted throttle form vehicle parameters. Lower numbers are safer, higher numbers usually perform better. When prediction trust is 1.0 the throttle will instantly respond to changes in required power, according to power predicted from vehicle parameters. When prediction trust is 0.0 the throttle will only move in response to feedback errors. Gain that determines how much of the predicted throttle should actually be used. The autopilot computes the amount of throttle required to achieve a given energy rate. The resulting throttle, times the throttle prediction trust, is effectively a proportional feed forward gain. The difficultly associated with predicting the actual engine power is such that the default value for this gain is zero. However you may be able to get better performance or responsiveness by increasing this value, up to 1.0. Larger throttle prediction trust values will cause the throttle to move more, and may cause excessive throttle motion. This is why the default value is set to zero. If maximum engine performance is desired than it is good to "linearize" the throttle actuation table. The autopilot assumes that the throttle is basically a power lever. However the typical small engine has a highly nonlinear relationship between throttle position and engine power. You can improve the performance by adjusting the throttle actuator table such that the power is in fact linear with the throttle command.		
Throttle LPF cutoff	Throttle low pass filter cutoff frequency (Hz). Zero to disable. This filter can be used to quiet engine transients caused by sensor noise.		

Flight

Flight Plans

Energy Err to Energy Rate (Kpo)	Energy Rate Error to throttle (Kpi)	Energy Rate Error Integral to Throttle (KI)	Throttle Prediction Trust	Throttle lpf cuttoff

Start Time (M:S)	

IAScmd	IAS	Undershoot

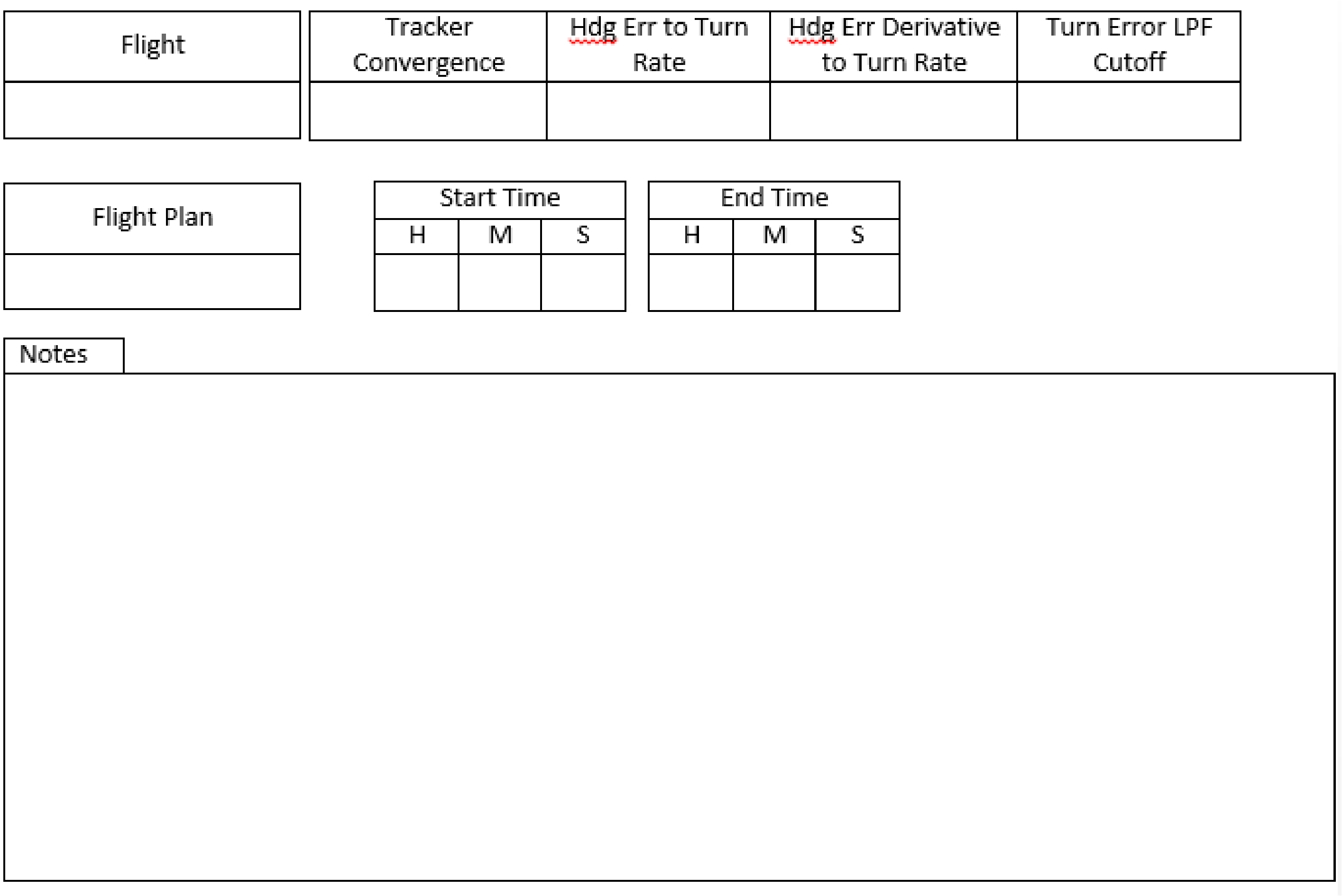
End Time (M:S)	

Notes

7. Lateral Track Control

Lateral Tracking

Gain	Definition	Too High	Too Low
Tracker Convergence	Decreasing this number causes the vehicle to try to fly more closely to the track. Making this value too small will cause track oscillations. Tracker convergence is a lateral gain that gives a dimensionless number setting the size of the elliptical trajectory used to guide the vehicle onto track. The track controller plots an elliptical trajectory to guide the vehicle onto the desired track line. The ellipse is defined by a semi-major axis (a) which is four times the semi-minor axis. The semi-minor axis is computed with the square of the speed and the tracker converge parameter, which defaults to 0.35. Making this value smaller will cause the aircraft to more rapidly converge onto a track segment. Making it too small could exceed the bandwidth of the track controller, causing track oscillations	Pre turn before converging on flight path	Quick oscillations in and out of a flight path
Heading err to turn rate	Gain relating heading error (rad) to turn rate (rad/s). Used to provide the primary steering input from either the heading controller or the track controller. The available gain depends on the bandwidth of the inner loop lateral controller. Heading err to turn rate is a lateral gain that sets the desired turn rate (rad/s) as a function of the error in heading (rad). This gain sets how hard the vehicle attempts to turn in response to an error in heading. Since turn rate is really the derivative of heading than this gain basically sets the time constant required to null a heading error. Increasing this gain will improve tracker bandwidth, but going too far will cause heading (bank angle) oscillations.	Bank oscillations	
Heading err der to turn rate	Gain relating the derivative of heading error (rad) to turn rate (rad/s). For vehicles with poor inner loop lateral control bandwidth this gain can help reduce track oscillations, otherwise this gain can be zero. Heading err der to turn rate is a lateral gain that sets the desired turn rate (rad/s) as a function of the rate of change of error in heading (rad/s). The derivative term helps to slow the track controller down as it gets close to the desired heading. This can be useful for systems that have lag in their bank angle control. Finding the optimal value for this gain is challenging. The default value is 0.1. Too much gain will cause fast bank angle oscillations, but too little gain may cause slow oscillations due to inner loop lag. For vehicles that have fast bank angle control it may be best to set this gain to zero.	Fast bank oscillations	Slow oscillations from inner loop lag
Turn error lpf cutoff	A low pass filter used to estimate the vehicle asymmetry, which is the turn rate that results at zero bank angle. Set to zero to disable. Enabling this allows the autopilot to correct for asymmetrical vehicles when tracking on a flight plan.		



Flight

Tracker Convergence	Hdg Err to Turn Rate	Hdg Err Derivative to Turn Rate	Turn Error LPF Cutoff

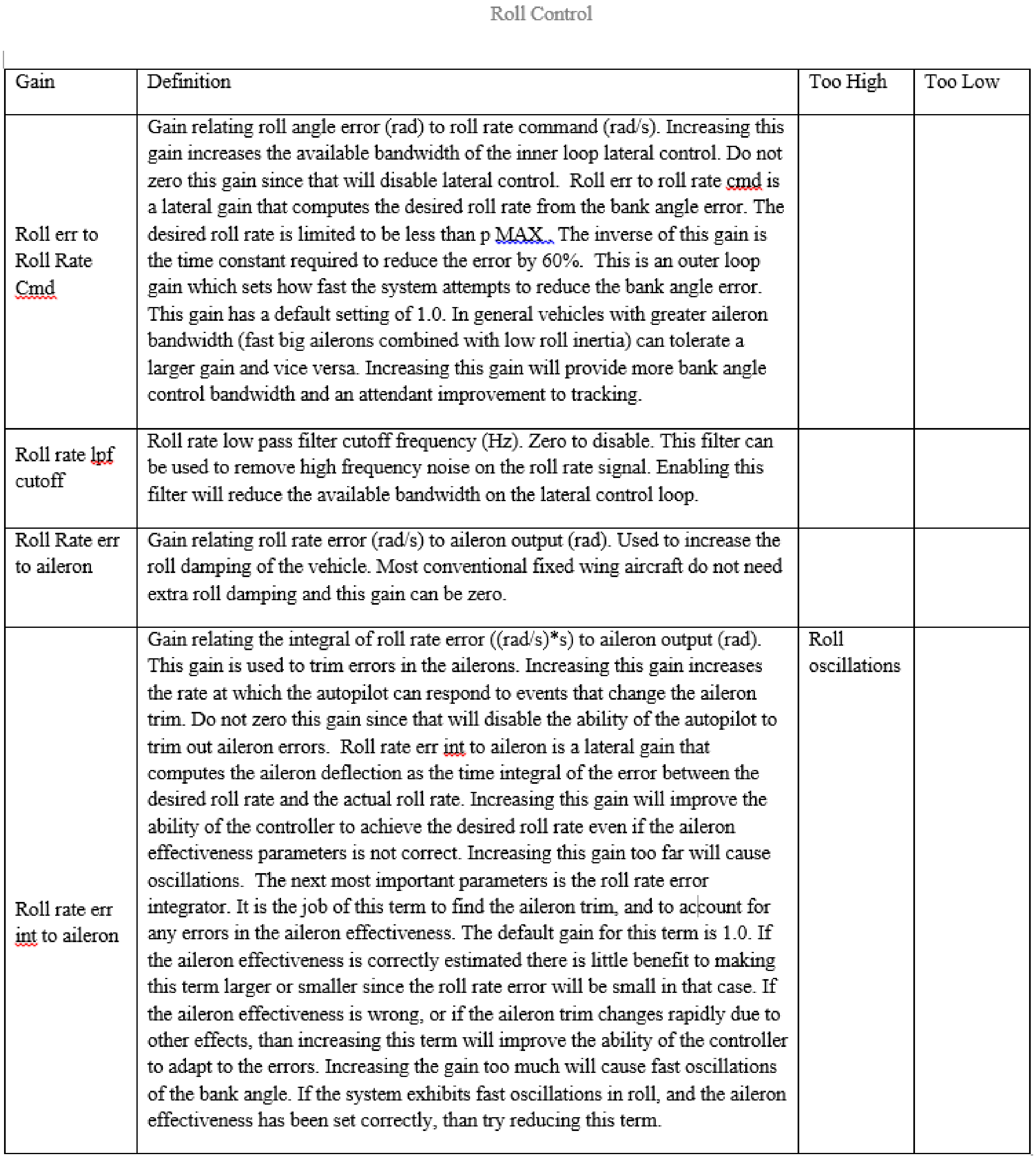
Flight Plan

Start Time		
H	M	S

End Time		
H	M	S

Notes

8. Roll Control



Roll Control

Gain	Definition	Too High	Too Low
Roll err to Roll Rate Cmd	Gain relating roll angle error (rad) to roll rate command (rad/s). Increasing this gain increases the available bandwidth of the inner loop lateral control. Do not zero this gain since that will disable lateral control. Roll err to roll rate cmd is a lateral gain that computes the desired roll rate from the bank angle error. The desired roll rate is limited to be less than p MAX. The inverse of this gain is the time constant required to reduce the error by 60%. This is an outer loop gain which sets how fast the system attempts to reduce the bank angle error. This gain has a default setting of 1.0. In general vehicles with greater aileron bandwidth (fast big ailerons combined with low roll inertia) can tolerate a larger gain and vice versa. Increasing this gain will provide more bank angle control bandwidth and an attendant improvement to tracking.		
Roll rate lpf cutoff	Roll rate low pass filter cutoff frequency (Hz). Zero to disable. This filter can be used to remove high frequency noise on the roll rate signal. Enabling this filter will reduce the available bandwidth on the lateral control loop.		
Roll Rate err to aileron	Gain relating roll rate error (rad/s) to aileron output (rad). Used to increase the roll damping of the vehicle. Most conventional fixed wing aircraft do not need extra roll damping and this gain can be zero.		
Roll rate err int to aileron	Gain relating the integral of roll rate error ((rad/s)*s) to aileron output (rad). This gain is used to trim errors in the ailerons. Increasing this gain increases the rate at which the autopilot can respond to events that change the aileron trim. Do not zero this gain since that will disable the ability of the autopilot to trim out aileron errors. Roll rate err int to aileron is a lateral gain that computes the aileron deflection as the time integral of the error between the desired roll rate and the actual roll rate. Increasing this gain will improve the ability of the controller to achieve the desired roll rate even if the aileron effectiveness parameters is not correct. Increasing this gain too far will cause oscillations. The next most important parameters is the roll rate error integrator. It is the job of this term to find the aileron trim, and to account for any errors in the aileron effectiveness. The default gain for this term is 1.0. If the aileron effectiveness is correctly estimated there is little benefit to making this term larger or smaller since the roll rate error will be small in that case. If the aileron effectiveness is wrong, or if the aileron trim changes rapidly due to other effects, than increasing this term will improve the ability of the controller to adapt to the errors. Increasing the gain too much will cause fast oscillations of the bank angle. If the system exhibits fast oscillations in roll, and the aileron effectiveness has been set correctly, than try reducing this term.	Roll oscillations	

VITA

Anton Mornhinweg

Candidate for the Degree of

Master of Science

Thesis: A PRACTICAL GUIDE TO THE PICCOLO AUTOPILOT

Major Field: Mechanical and Aerospace Engineering

Education:

Completed the requirements for the Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2014.

Completed the requirements for the Bachelor of Science in Aerospace Engineering at Oklahoma State Universiy, Stillwater, Oklahoma in July, 2009.