DEVELOPMENT OF A SOFTWARE IN THE LOOP SIMULATION APPROACH FOR RISK MITIGATION IN UNMANNED AERIAL SYSTEM DEVELOPMENT

By

CHARLES PRESTON JOHNSON

Bachelor of Science in Mechanical Engineering

Bachelor of Science in Aerospace Engineering

Oklahoma State University

Stillwater, Oklahoma

2018

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
December, 2020

DEVELOPMENT OF A SOFTWARE IN THE LOOP SIMULATION APPROACH FOR RISK MITIGATION IN UNMANNED AERIAL SYSTEM DEVELOPMENT

Thesis Approved:

Dr. Andy Arena

_____

Thesis Adviser

Dr. Rick Gaeta

_____

Dr. Imraan Faruque

_____

ACKNOWLEDGEMENTS

I would first like to thank my parents started me on the path to becoming an engineer and have helped me and encouraged me to achieve something more each step of the way. My brother has always been there and without him many of the electrical or coding knowledge we learned would have been lost on me. They also took on the task of helping me form clearer messages so that the reader might have an easier time understanding what I am trying to say.

The wonderful and awesome members of my committee for listening to my questions and for pushing the bar just a little bit higher. Dr. Arena continues to push for the best and expects even more leading to discovering incredibly rewarding adventures. Dr. Gaeta and Dr. Faruque have taught many lessons that will stay with me for a long time to come and broadened my knowledge to new horizons during every lecture and conversation.

My fellow students that have helped me in some way are too numerous for me to list them all. Jeff, Aron, Thomas, Colton, and Zac who were there at every moment to listen to me babble about computers and code. Our pilot Marc who took time out of his way for a short series of flights with one small aircraft. Collin who keeps the world turning and all of us in line.

A special thank you goes to Dr. Tridgell who graciously answered every question I asked about ArduPilot. Thanks also needs to be given to the vibrant community of users and developers that believe open source can be as good or better than commercial options.

Finally, a thank you to Pat “Moondog” Hardage for showing that education should be wildly diverse every day of the week.

Name: CHARLES PRESTON JOHNSON

Date of Degree: DECEMBER, 2020

Title of Study: DEVELOPMENT OF A SOFTWARE IN THE LOOP SIMULATION APPROACH FOR RISK MITIGATION IN UNMANNED AERIAL SYSTEM DEVELOPMENT

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: A common method to reduce risk during the development of new designs is simulation and estimation. The extent to which these simulation and estimation techniques can be relied upon for small Unmanned Aerial Systems (sUAS) is unknown. Combining the autopilot together with a simulator for Software in the Loop (SITL) allows designers to tune and observe autopilot behavior before the design is finished. This thesis extends the tools provided through SITL to present methodology that can both provide early insight to the handling qualities of the aircraft and validation of the simulator.

ArduPilot and X-Plane 11 are used as the autopilot and simulator. New features were developed to extend the functionality of ArduPilot. Additional software was developed to assist in both identification of aircraft modes and validation of simulation.

With mixed results from validation of X-Plane, the need to perform flight testing of real aircraft is still more desirable for precision tuning of sUAS for more desirable handling qualities. What can be gained from SITL is risk mitigation from unconventional additions to sUAS and detailed analysis of failsafe behavior of the autopilot.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Current development of small unmanned aerial systems (sUAS) includes many high-risk operations to both the aircraft and the operators. For novel designs, there are limited options beyond development of technology demonstrators to test the new designs. A designer who is provided with tools that can accurately simulate the design will be able to develop faster, more efficiently, and reach a completed flight-testing phase earlier than a designer without these tools. Currently, there are many tools available to these designers, but the question of how accurate the simulations are has yet to be answered. Comparisons of flight behavior between simulators and real aircraft have been done but without well-defined methods that would allow comparison across a suite of aircraft and simulators. This begs the question: how can such comparisons be made repeatedly, safely, and with lower pilot experience requirements?

For more costly aircraft, the risks of flight testing also come with a much higher development cost and thus adds to the many risks that must be taken on to develop novel aircraft. Hamish Willee noted as such in the ArduPilot software in the loop (SITL) documentation in 2016, “Crashing software planes is a lot cheaper than crashing real ones!” Many of these risks could be better mitigated with SITL testing and robust development procedures. Including SITL testing provides early opportunities in the design process to tune controllers and verify design decisions. The extent to which SITL testing can be relied upon is unknown. To verify where in the design process of a sUAS a reliable model can be acquired, a study of the efficacy of the models must be

done. Due to the variation in dynamics of sUAS this method might not be applicable for all sUAS, but methodology to check the efficacy for a flight behavior can still be invaluable.

This thesis will present work that demonstrates that the ArduPilot portion of SITL is the same as that used by the hardware autopilots. This lays the groundwork for determining that any differences in measured flight behavior, while the autopilot is providing a stability augmentation system or a control augmentation system, is the results of the flight dynamics model rather than differences in autopilot output signals. After this is established a set of example maneuvers will be down that will assist in tuning a virtual aircraft before a physical aircraft is flown. The qualify and confidence of the virtual tuning process will depend exclusively on the quality of the flight dynamics model.

GOAL AND OBJECTIVES

First and foremost, an overview of the autopilot and the flight simulation software will be explored here. These are the core tools of the apparatus which enable the ability to conduct repeatable and safe validation of simulation tools. This includes any modification or new features that are added to the autopilot to make adaptation of the methodology easier and more attainable for future work. These software solutions will be able to be used for both simulation and flight-testing methods.

Established analytical methods have been used across the industry for decades to provide insight to the stability and handling qualities of aircraft. These methods will be used as a starting point for determination of frequency and damping of the aircraft's flight modes. Computational Fluid Dynamics will also be explored going so far as to determine how well previously explored methods to determine aircraft stability performed by others can be used by more accessible tools today. The characteristics found from these methods will provide guidance on frequency used for the input signal for mode identification in simulation and flight-testing.

Flight testing maneuvers that can exhibit typical aircraft mode identification will also be explored. Software will be developed which executes safe and repeatable maneuvers for the purpose of aircraft mode identification. This software takes advantage of features created earlier in this work. Comparison of these aircraft modes will provide insight to how accurately the dynamic system of the aircraft has been estimated by the analytical methods and by the simulator. This will be done with a form of system identification. The comparison will be made easier by using the same input signals for simulations and real aircraft flight testing.

A case study of a situation where SITL testing could have prevented catastrophe will be presented in the final section. This work was done with fixed wing sUAS aircraft in mind. The methodology could feasibly be applied to many other classes of aircraft for the purposes of simulation efficacy.

- Conduct a literature review on ArduPilot, X-Plane, MAVLink, MAVProxy, SOLIDWORKS Flow Simulation, and aircraft testing maneuvers for parameter estimation. Each of these building blocks adds to mitigation of risk through knowledge of the tools that are used.
- Perform parameter estimations of the test aircraft using analytical methods. These methods will be used the baseline measure to determine to what extent each method can be relied on to predict the modes of the aircraft.
- Develop and test a software solution to command maneuvers to a SITL aircraft. This should be able to create a repeatable servo output and be easily adaptable to apply any input signal.
- Develop and test a software solution to command maneuvers to a real aircraft with identical inputs to SITL. This can be the same as the software developed in the previous objective with additional debugging through ground testing.
- Analyze data from SITL maneuvers and real aircraft maneuvers and identify aircraft modes from each utilizing methods found through the literature review.

- Demonstrate that improper risk mitigation can lead to catastrophe through lack of testing and prior knowledge. Provided detailed examples from past flight testing where past methodology surrounding the autopilot produced dangerous or unexpected results.

AUTOPILOT STRUCTURE

The term autopilot must first be defined. The traditional definition is typically taken from the perspective of a controls engineer. The autopilot is the system that provides guidance, navigation, and control for the aircraft. This definition includes all hardware and software needed to make this goal happen. When looking from the current perspective of sUAS development, the term has come to refer to either the hardware, excluding sensors, or the software that is running on the hardware. The term is used both ways throughout the documentation. The autopilot, hardware and software together, communicates with Ground Control Software (GCS) using a protocol common to both autopilot and GCS called MAVLink. Knowing what is in the stack of parts that allows the autopilot to perform its job of flying an aircraft mitigates risk from choosing improper parts or incompatible parts.

A brief overview of some of the parts of the autopilot and related components will be presented. This is by no means comprehensive and should only be considered a very short gist of the ArduPilot documentation. In order to really understand the nuances of the autopilot, the reader will find the reference to the ArduPilot documentation and forums invaluable

MAVLink is a data transfer protocol that is designed to operate with minimal data structure size and with reliable data packet transfer while also being secure. There have been numerous published articles that look at the pieces of the current generation autopilot systems. Several have looked at the MAVLink protocol and its security vulnerabilities [1]. A few have looked at autopilot systems that use the MAVLink protocol to communicate both to a ground control station (GCS) and to other aircraft [2].

There are two major open source autopilot software packages being actively developed as of this writing for traditional and complex fixed wing aircraft. Much of the creation was born out of development focused on using hardware from the open source Arduino project. PX4 and ArduPilot have become the go-to open source autopilot projects on which many in the community have focused on developing. Each has pros and cons which are detailed, in a somewhat older perspective, by Hood [3]. ArduPilot has emerged as the most feature rich autopilot system with many new sensors and vehicle types being added on a regular basis [4]. Chao et al. did a survey of closed source and open source autopilots in 2016 that did not include PX4 nor ArduPilot [5]. They did cover the basics of how this class of autopilot does guidance, navigation, and control (GNC).

The ArduPilot open source project is constantly updating with support for new hardware and new features. Currently the project supports several vehicle types as well as several autopilot boards. The architecture of the autopilot is large compared to simple stabilization systems or simpler autopilots [6]. Much of the architecture of the controller has remained the same as more features such as Lua scripting and Independent Watchdog (IWDG) have been added to take advantage of ever improving small scale hardware. An overview of the structure of a typical implementation of an ArduPilot supported hardware and software stack is shown in Figure 1. ArduPilot's ArduPlane version 4.0.5 will be used for all examples in this work. A fork of the ArduPilot master branch will be used for flight testing [7]. Any and all modifications to the firmware will be explicitly communicated where it is relevant and also viewable in thesis branch of the fork.

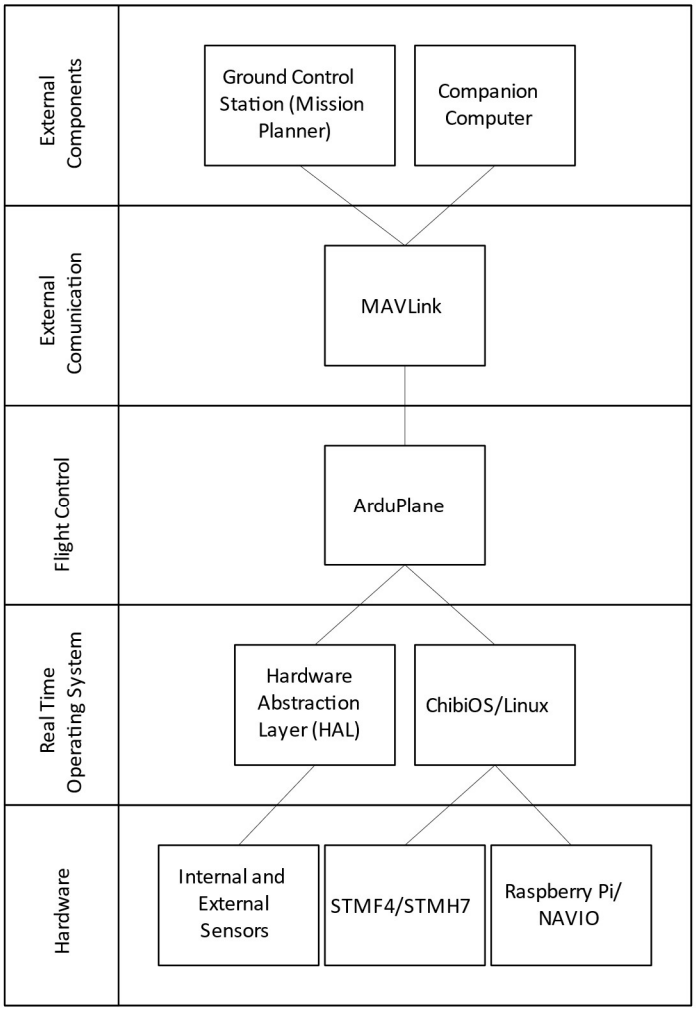


Figure 1: ArduPilot software and hardware structure

Autopilot hardware has seen many rapid development advancements in recent years thanks to increased demands from other consumer electronics companies pushing for smaller, faster, and lower power consumption components. ProfiCNC designs and manufactures, in partnership HEX Technology, the CubePilot series of autopilots. Other supported autopilot hardware designs are available from companies such as Matek, CUAV, mRobotics, and Holybro. While these other options are supported and have good features, many of the latest features and redundancy checks are available first and foremost on the fully featured Cube series autopilots. Many other open source

autopilot hardware designs are also available on the market with many included in a 2018 review of autopilot hardware available on the market at the time by Ebeid et al. [8]. Since the publication of that article, ArduPilot has added support for the STM32H7xx series of processors and new pieces of hardware from Global Navigation Satellite System (GNSS) sensors to laser rangefinders. This has opened the doors for many memory-intensive features to be added to the autopilots.

Software in the loop testing (SITL) with ArduPilot is a good way to tune an aircraft quickly and cheaply for stabilized flight. As the ArduPilot developer documentation says, "crashing virtual vehicles is a lot cheaper than crashing real ones!" [6]. There are several flight dynamics models (FDM) that are supported and detailed in the ArduPilot documentation. Together a flight dynamics model, autopilot, and ground control software form what is called SITL. Figure 2 shows how each piece is connected and what information is transmitted between each piece. This set up is not unlike how the autopilot would be used with a real aircraft, autopilot, and ground control software.

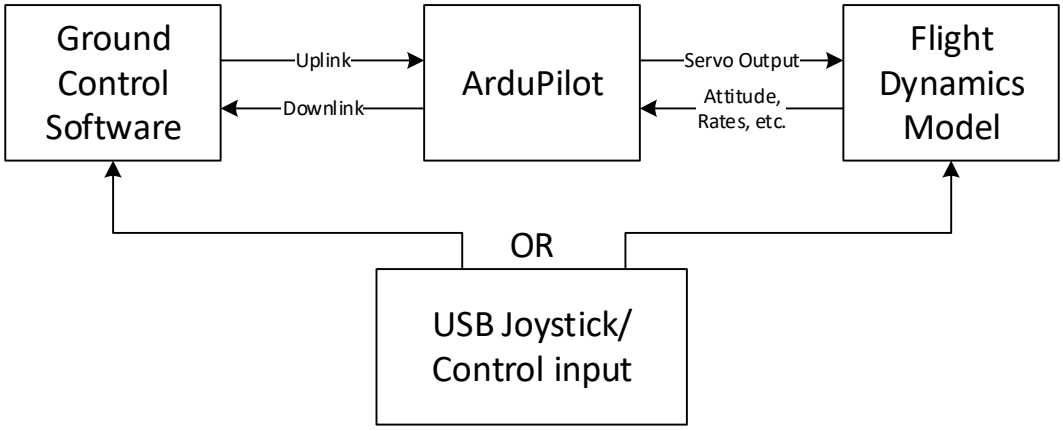


Figure 2: SITL structure block diagram

Hardware in the Loop (HIL) is not currently supported by ArduPilot. The benefit of this style of simulation is the ability to check the behavior of the aircraft with the hardware that would be used during a flight test. Support for HIL was dropped for several reasons. Bringing back support for HIL has been a frequent request on the ArduPilot forums [9]. One of the biggest limitations of HIL is the latency and consistency of USB. High Speed USB would not solve this problem though.

After a discussion with Dr. Andrew Tridgell, the issue lies with the operating system of the host computer, specifically Windows, not parsing data from the drivers for time gaps of up to 20 ms. Due to this latency, the full ArduPilot sensor estimation methods, the Extended Kalman Filter (EKF), cannot be used. This diminishes the distinctions between useful cases where HIL can perform actions that SITL cannot. As Dr. Tridgell explained, the issue is not a lack of bandwidth from the USB technology used by the majority of the autopilot hardware supported by ArduPilot.

ARDUPLANE SYSTEM FAILURE ACTIONS

Risk mitigation from knowing what to expect when something does go wrong with the aircraft allows for operators to prepare standard operating procedures proactively rather than be caught off guard. There are several categories of failure that can cause unexpected behavior from the aircraft. The main categories being software failure and hardware failure.

Many of the software failures that ArduPilot could encounter in flight are compartmentalized to small groups of features. Such as if the software for a sensor gets an error, the most likely outcome is that the sensor will be dropped from contributing to the EKF. However, if ArduPilot is unable to continue operating for any reason, a WATCHDOG_RESET event occurs and reboots the autopilot inflight. During the time that the autopilot is still booting, the IOMCU chip that is present on some of the autopilots allows the aircraft to continue to be controlled in MANUAL mode. This failure behavior, as well as many others, can be tested trough tools provided by MAVProxy, pymavlink, and RC switch options. Hardware failure behavior can be tested with SITL as well. Many of the question that start with what if can be answered by changing parameters that start with SIM_. GPS, IMU, RC, and GCS failure, drift, and erroneous sensing can all be tested using SIM_ parameters in SITL.

During a recent manufacturing issue by an autopilot hardware vendor, it was found that ArduPlane was not always applying the proper Inertial Measurement Unit (IMU) failure behavior. Thanks to renewed attention to the failover mechanics of the autopilot, the intended behavior of

triple redundant IMU sensors was reimplemented. With the new update, the GCS will display a warning to the user of the failure. An example of the dialog is shown in Figure 3. A feature to test this behavior was added through a radio control (RC) input channel option. When the pilot commands the RC channel from a low pulse width modulation (PWM) to a high PWM, the selected IMU data is stopped from being used by the autopilot [4]. This RC input channel option can be done by anyone without any custom-built firmware or modification to the hardware.

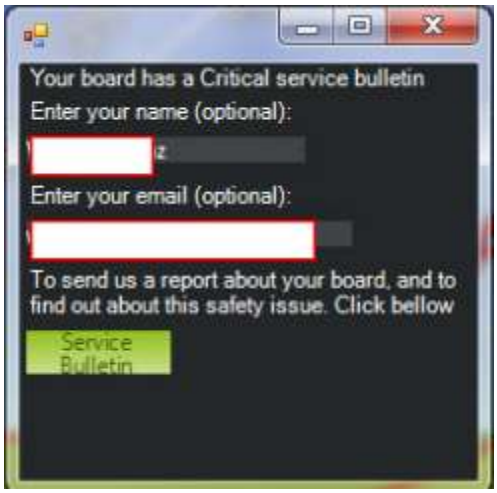


Figure 3: Mission Planner critical service bulletin

This issue as well a few others over the course of 2019 have led to the proposal presented at the 2020 ArduPilot Unconference for a critical alert bulletin being sent to all users that are impacted by the fault [10]. A Github repository has since been created to log each of the critically important bulletins that should be presented to users by GCS.

ARDUPLANE CONTROL STRUCTURE

In continuing to better understand how ArduPilot works and where the root of the tuning process must focus, an overview of the attitude and GNC controllers is presented. When applying methods to reduce risk during controller tuning, knowing what part of the controller needs to be changed can be made easier by know how the controller is designed.

The control structure for ArduPlane follows the general guidelines for typical proportional-integral-derivative (PID) control used by many textbooks for attitude control of fixed wing aircraft [11] [12] [13]. There are a few details that should be considered when tuning the aircraft that are not readily apparent on the surface but do become apparent when looking at the control diagrams for the roll and pitch controllers. The code for these control loops can be found in the APM_Control folder [4]. For the pitch control loop,

$$k_P = (\text{PTCH2SRV\_P} - \text{PTCH2SRV\_I} * \text{PTCH2SRV\_TCONST}) * \text{PTCH2SRV\_TCONST} - \text{PTCH2SRV\_D} \quad (1)$$

$$k_I = \text{PTCH2SRV\_I} * \text{PTCH2SRV\_TCONST} \quad (2)$$

This is such that k_{FF} is proportional to the error of the pitch rate. The k_P and k_I of the roll controller are calculated in similar manners.

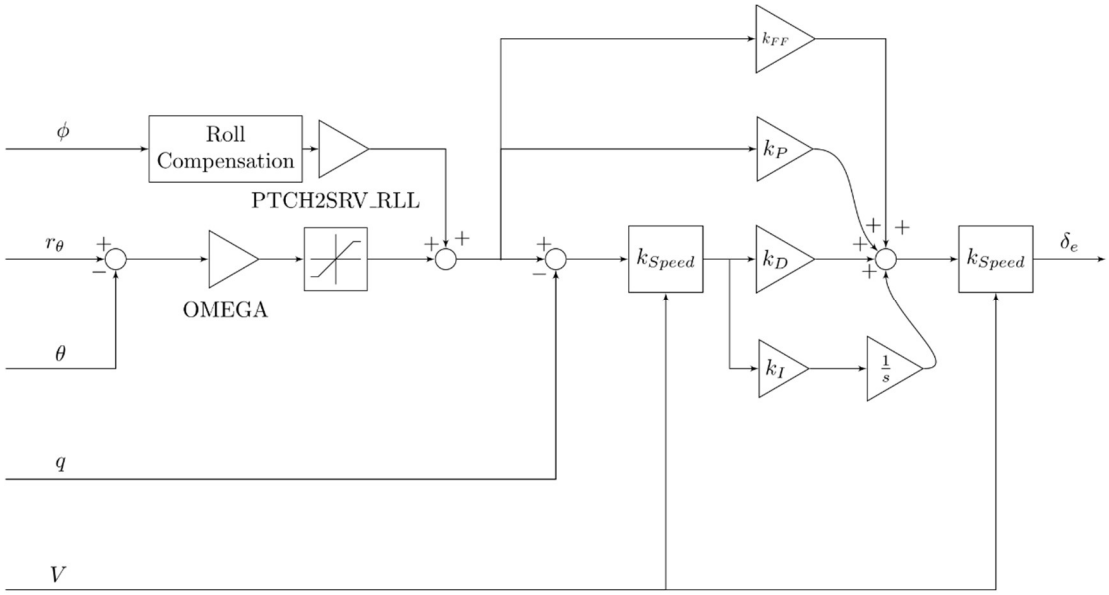


Figure 4: ArduPlane pitch control loop

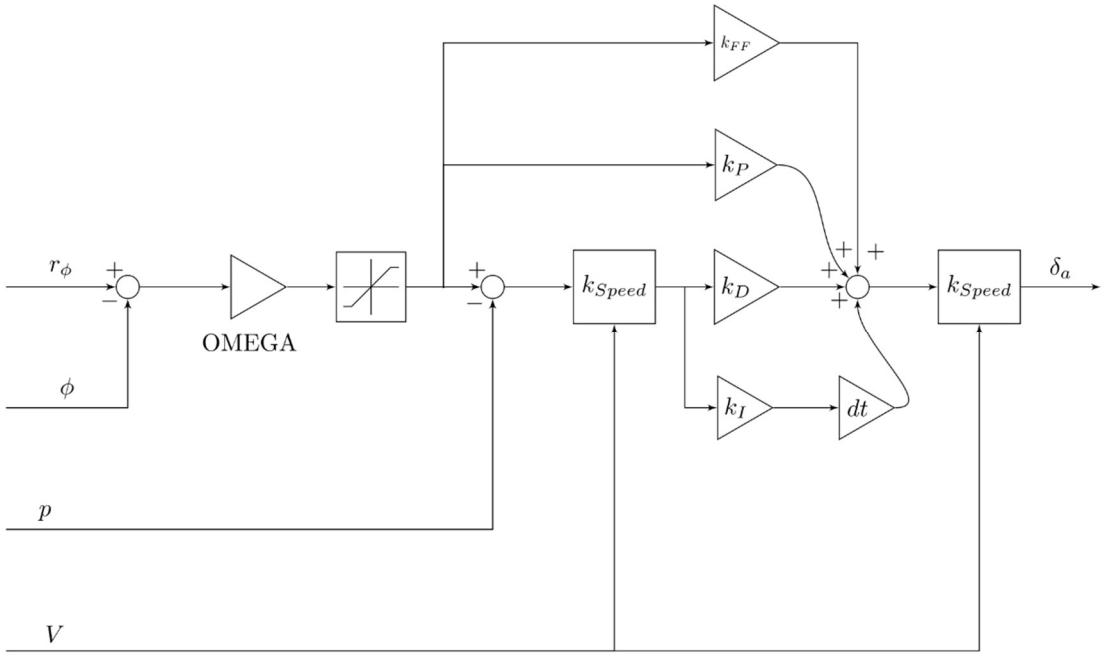


Figure 5: ArduPlane roll control loop

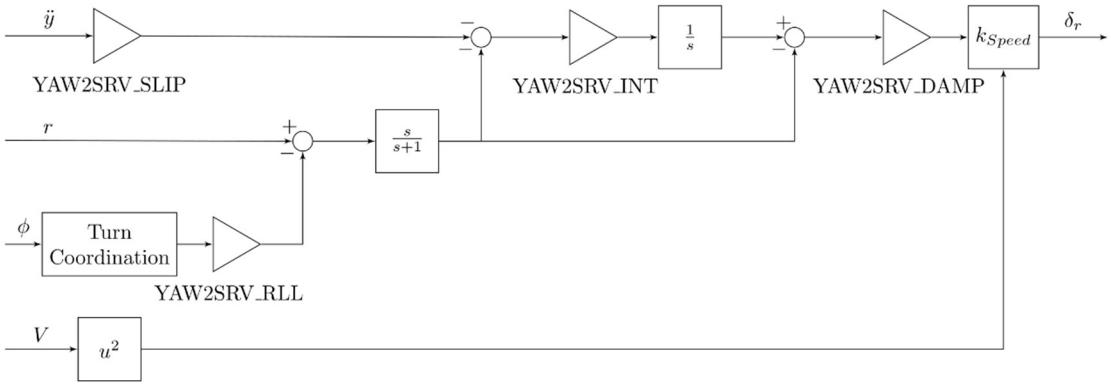


Figure 6: ArduPlane yaw control loop

ArduPlane includes a feature to use the airspeed of the vehicle, either from synthetic estimate or differential pressure measurement from a pitot probe, to scale the proportional, derivative, and integral gains. The scaling equation that is used is based on the inverse of the airspeed.

$$k_{speed} = \frac{V}{SCALING\_SPEED} \tag{3}$$

This can be very beneficial as control surfaces can have the same effectiveness at higher speeds with smaller deflections. There is more detail to the speed scaling that can be gathered from looking at the ArduPilot source code file ArduPlane/Attitude.cpp [4].

```
if (aspeed > auto_state.highest_airspeed) {
    auto_state.highest_airspeed = aspeed;
}
if (aspeed > 0.0001f) {
    speed_scaler = g.scaling_speed / aspeed;
} else {
    speed_scaler = 2.0;
}
// ensure we have scaling over the full configured airspeed
float scale_min = MIN(0.5, (0.5 * aparm.airspeed_min) / g.scaling_speed);
float scale_max = MAX(2.0, (1.5 * aparm.airspeed_max) / g.scaling_speed);
speed_scaler = constrain_float(speed_scaler, scale_min, scale_max);
```

Of note from this piece of code is the inclusion of minimum and maximum bounds on the scaler. The parameter that has the largest impact on this part of the autopilot is aptly named SCALING_SPEED. A graphical visualization of the impact this parameter has on the scaling of the PID values can be seen in Figure 7. For this plot ARSPD_FBW_MAX is 50 m/s and ARSPD_FBW_MIN is 20 m/s.

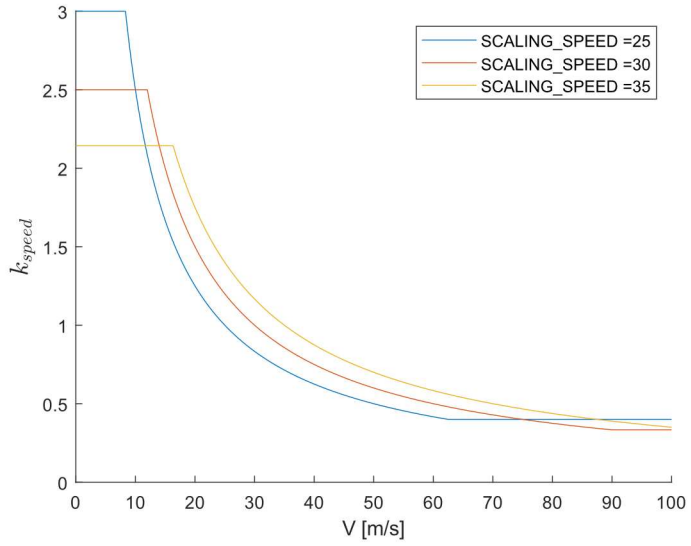


Figure 7: ArduPlane scaling speed

For airspeed and altitude control ArduPilot uses the Total Energy Control System (TECS) and for navigation the L1 controller is used. TECS has many parameters to that can be tuned to control longitudinal behavior. As the name TECS implies, the controller trades kinetic energy from airspeed and potential energy from altitude to achieve the desired flight condition. ArduPilot provides a way to use each of the control loops in an "added layer" sort of method. The ArduPilot documentation shows the control loops that are used by each flight mode. This can be useful for selectively tuning each of the controllers. TECS also includes parameters for deepstall and glideslope landing methods.

CURRENT ARDUPLANE TUNING METHODS

The documentation for ArduPilot provides methods for tuning aircraft by in-flight methods such as Autotune mode or manual parameter changes. These methods are excellent for traditional aircraft with predictable flight characteristics. When looking at aircraft that might be higher risk or require takeoff with a control loop turned on, these flight-testing methods no longer work. Using

these same methods in SITL can be done, but the question, “How well does a simulator work?” arises.

Internally at the Oklahoma State University Department of Mechanical and Aerospace Engineering, we have developed a method to estimate tunes for aircraft control loops without SITL and before maiden flight. This method arose from a lack of having the information this thesis is searching for as well as multiple issues with understanding the scope of the impact of improper PID tuning. Many of our aircraft require at least some level of stability augmentation to assist the Pilot in Command. Most of our aircraft are designed to operate with full control authority at landing and takeoff as dictated by the Code of Federal Regulation (CFR) Part 25 and as guided by countless other authorities of aircraft controls.

The standard process: Set the ArduPilot SCALING_SPEED to approach speed. Either disable the use of the airspeed sensor or apply a consistent flow to the airspeed sensor to achieve approach speed. Adjust the P gain such that full deflection of the control surface in Manual mode is also applied in FBWA mode. This ensures that full deflection of the control surface is possible during landing approaches in control loop enabled modes. The contribution from the integral gain can be watched from the steady behavior of the control surfaces after holding the aircraft at a non-level attitude and returning to level pitch and roll. Derivative gain can be tuned by observing the aircraft moving at consistent rates. This method has provided adequate results for maiden flights and additional tuning has been done by looking at the flight logs for the PID controller outputs.

There have been a few cases where this method has proven to be inadequate for flight outside of the linear regime. For example, a swept delta-wing aircraft entering a high alpha vortex lift maneuver exhibited a pitch oscillation during flight testing after the initial tuning procedure. This can be observed in Figure 8 from the logs for the PID pitch controller and the pitch attitude from the log stored on the internal SD card called the dataflash log. As the θ of the aircraft reaches near 45 degrees, the oscillations begin. This aircraft had previously flown with an Aura 8 rate

controller with no oscillations in high alpha. After further tuning during flight testing, the aircraft did not exhibit the same oscillations in high alpha.

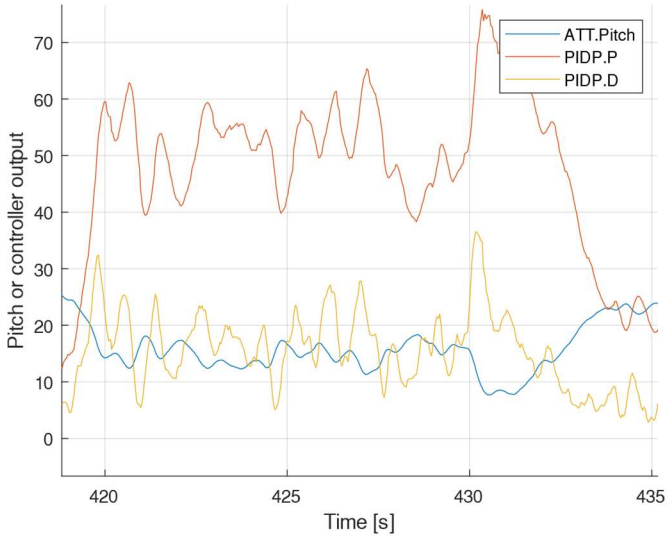


Figure 8: Oscillations from PID controller in high alpha flight

AIRCRAFT MODE IDENTIFICATION BACKGROUND

To reduce comparison of flight characteristics between flight testing and simulation to just aerodynamic modeling, aircraft modes will be excited and measured. These modes can be measured at the same flight conditions in simulation and flight testing by allowing the autopilot to do most of the hard work including trimming the aircraft and deflecting the control surface. To effectively tune an attitude controller prior to flight testing the real aircraft, predictions of the flight characteristics of the aircraft must be done. These predictions come from analytical methods and the flight characteristics are shown through aircraft modes. Having estimations of modes available can better prepare the designer for flight testing of the aircraft.

There are five modes for fixed wing aircraft: phugoid, short period, roll, Dutch roll, and spiral. These modes are defined by the system dynamics of the aircraft and can be estimated as well as measured. For linear flight regimes these modes can be estimated based on the geometry of the aircraft. Geometry based estimation details are covered by Nelson and McCormick based on

methods presented in the USAF Stability and Control DATCOM [12] [14] [15]. There has been extensive research into identification of flight modes from flight test data using various techniques that fit the intended flight regime of interest.

MathCAD programs for the method presented by Nelson were created by the author and Dr. Andy Arena for the purpose of streamlining analytically based mode identification methods for various aircraft. Nearly all the inputs necessary to find the stability derivatives can be done without charts from other sources. Places where manual lookups need to be done were noted next to the equations with the location in the text and information needed to find an answer from the chart. This method was used with the intention of presenting what many aerospace engineers use as a first glance to classify the handling qualities of an aircraft. The outputs of this program were compared to those presented in the textbook as well as another MathCAD program created by Dr. Andrew Arena called 6DoF. 6DoF, six degrees of freedom, takes the derivatives as inputs and outputs longitudinal and lateral roots.

Arena also created two MathCAD programs that went back to the original equations presented by McCormick and Nelson in their textbooks that made assumptions of typically negligible terms and made calculations for those terms [14]. These methods were separated into two programs, StabCon-S and StabCon-D, which perform similar tasks but provide different information in the end. StabCon-S provides information about aerodynamic center, neutral point, and trim conditions with and without propulsion effects, whereas StabCon-D provides information about the dynamic behavior of the aircraft including handling qualities and dynamic response to inputs.

There are many variations of input signals for identifying aircraft flight modes. Babcock explored many options when testing them in STARS [16]. Their use of the DC chirp was more geared towards wide range identification of frequency response. Many flight programs use the singlet or doublet signal to identify short period and Dutch roll modes [17]. Phugoid, roll, and spiral are typically found from constant inputs signals rather than time dependent inputs. Other signals

with varying frequency have also been used in previous experiments. O'Neill analyzed several options for his work and ended up using a DC chirp for his work frequency sweep work [18]. This thesis uses a doublet input signal as the goal is excitation of modes for identification of short period and Dutch roll. Considerations to reduce putting the aircraft out of trim will be done.

Identification of aircraft parameters from flight data has been a long-studied topic with several methods published for various flight-testing techniques. Dorobantu et al. and O'Neill used frequency information from flight data to find where the roots of the aircraft's dynamic system [19] [18]. Their information gathered does provide many good insights to the behavior of the aircraft but requires an input signal that covers a larger range of frequencies. Seamans et al. showed a method that also used a Fourier Transform [17]. Their method relied on discretizing the data to fit into the memory space of their mechanical calculator. The 1974 conference on "Parameter estimation techniques and application in aircraft flight testing" included methods using root mean square, Kalman Filter, and maximum likelihood techniques for nonlinear models [20]. While very informative for the historical significance for the field, these methods are not fully applicable to this thesis. Kimberlin provides methods to determine natural frequency and damping from each of the aircraft modes [21]. These methods are very applicable to this thesis assuming that data of the aircraft attitude can be captured at a high enough rate. Casiano shows several methods for determining the damping ratio from output signals including the common log decrement method [22].

Using a combination of frequency domain techniques as well as time domain techniques from Kimberlin and Casiano will provide a best of both worlds option. Since ArduPilot can log attitude at up to 25 Hz by enabling ATTITUDE_FAST, the Nyquist frequency from the dataflash log is 12.5 Hz. The typical short period and Dutch roll frequencies are about one tenth of the Nyquist frequency making capturing of the desired output signal within the capabilities of ArduPilot.

SIMULATOR TYPES

Choosing an appropriate simulator for the aircraft being designed can be critical to effective SITL testing. Use of a simulator that oversimplifies aircraft characteristics can lead to misguided ideas of how the real aircraft might behave.

Each FDM that is supported by ArduPilot has a purpose that is defined when the simulator is created. Some are geared heavily towards inertial models for rotary wing vehicles while others make largely linear fixed wing flight assumptions. The ideal simulator would be a lock step simulator where the simulation time steps could be controlled to account for the time to process instructions by the microprocessor. This would effectively simulate the time taken for each processor instruction and provide insight to better control the aircraft by processing lag. This would also be ideal from the perspective of the ArduPilot scheduler. Since the default main loop rate is 50 Hz for ArduPlane, a lock step simulator would allow for precise time steps between control outputs. For the Extended Kalman Filter (EKF) this would be able to provide data at the expected rates from real hardware. Since most FDMs are not easily capable of lock step simulation, ArduPilot created different EKF types do handle less consistent data streams and time steps.

For rapid development and testing of new features, ArduPilot includes a built-in simulator. This built-in FDM is used by default when running the "sim_vehicle.py" script and is adequate for simple modeling of the supported vehicle types. At the top of the file "SIM_Plane.cpp" is a code comment that states, "[V]ery simple plane simulator class. Not aerodynamically accurate, just enough to be able to debug control logic for new frame types." The aerodynamics of the built-in FDM are based on the last_letter project [23]. The aircraft used by default is a Skywalker 2013. Other aircraft models can be added with ease.

Another free and open source software FDM is the open source simulator JSBSim [24]. JSBSim modeling is based on the textbook "Aircraft Control and Simulation" by Stevens and Lewis as well as other papers published by AIAA, NASA, and the FAA. FlightGear, another open source

project, can be used to visualize what is happening inside the JSBSim FDM. As with the built-in FDM, a few example aircraft are provided in the ArduPilot repository and more can be easily added.

There are two commercial flight simulators that are of interest to rapid prototyping, autopilot development, and the scope of this thesis. RealFlight is a simulator specifically marketed as a sUAS simulator. The developers of RealFlight, Knife Edge Software, provide many high fidelity, commercially popular model aircraft. The second is X-Plane. X-Plane is used by many hobbyist flight simulator users as well as a pilot training tool. There are several common general aviation aircraft provided at full scale models. X-Plane does have its own peculiarities, of which static margin and inertia will be discussed. Typical aircraft define an inertia matrix based on the second area moment of inertia, I. This can also be expressed as a radius of gyration, R_g.

$$R_g = \sqrt{\frac{I}{m}} \tag{4}$$

where m is the mass of the object.

Some recent work has been done to find methods to quantify efficacy of FDMs. This work is usually relevant to what are typically referred to as replay methods. Replay used when the inputs from that flight are replayed in a simulator to observe efficacy, sensor variations, and sources of error. Nguyen et al. used this method with X-Plane using a scale model of a KLA-100 [25]. Their flight was to show X-Plane response to inputs from a standard flight. No comparison of aircraft modes was done. Kamal and Aly used a Tiger-Trainer model aircraft and compared characteristics of their own FDM with results from Digital DATCOM [26]. In 2014 Dorobantu et al. showed a method using the Theil inequality coefficient (TIC) to quantify differences between linear time invariant system outputs. Previously in 2013, Dorobantu et al. showed TIC being used to verify controller performance between a flight test inputs and the simulated results [27] in a replay format.

COMPUTATIONAL FLUID DYNAMICS AS A DESIGN TOOL

Computational Fluid Dynamics (CFD) has been used to identify and tune minute details of aircraft design. This allows for quick feedback in the design loop without spending time creating prototypes or models to be tested in wind or water tunnels. SOLIDWORKS Flow Simulation is a widely used and highly regarded numerical solving program that has been extensively validated [28]. Static derivatives are relatively easy to find and are reliable enough to be used for validation [16]. Getting dynamic stability derivatives with an aeroservo method is not possible in SOLIDWORKS Flow Simulation, but a step towards them from a design point is with quasi-stability. In theory, simulating the quasi-steadiness can be done with rotation of the flow to simulate, for example, a constant pitch rate, q.

After an extensive search, no validation work for rotating flow regions around airfoils in SOLIDWORKS Flow Simulation has been found. Since this level of validation work is outside the scope of this thesis, a general overview of what SOLIDWORKS Flow Simulation offers for rotational flow is given. More information about the options for rotational flows can be found in the SOLIDWORKS Flow Simulation Technical Reference [29]. Babcock used the simulation package STARS, which is capable of aeroservoelastic simulation, and described methods for finding the quasi-steady derivatives of aircraft [16]. Babcock used ALE2D in STARS with an NACA 0012 with varying magnitude of flow rotation and position of the center of rotation. This provided excellent insights into how well the data from ALE2D matches the data found through X-Foil. While SOLIDWORKS Flow Simulation does have many advanced features and can handle a large number of different problem styles appropriate for the $k-\epsilon$ (pronounced k-epsilon)method, it does not have a good way to provide timed inputs to the mesh to mimic control surfaces like STARS does. This would provide a much clearer picture of the unsteady aerodynamics as well as dynamic stability..

CFD simulation runs to determine static stability derivatives is done by changing angle of attack, α, and sideslip angle, β, in the inputs to the flow parameters. Where

$$\alpha \equiv \tan^{-1}\frac{w}{u} \tag{5}$$

and

$$\beta \equiv \sin^{-1}\frac{u}{V} \tag{6}$$

where

$$V \equiv \|\langle u, v, w\rangle\| \tag{7}$$

where $\langle u, v, w\rangle$ are the velocities of the aircraft in the body frame.

Velocity for the flow is usually selected near the trim or cruise condition of the aircraft, and the same is used for analytical parameter estimation methods. Derivatives with respect to velocity can be approximated from changes in angle of attack, α, or sideslip angle, β, within the linear flight dynamics region.

Velocity [kts]	Alpha [degrees]	Beta [degrees]	Aileron [degrees]	Rudder [degrees]	Elevator [degrees]
30	-5	0	0	0	0
30	-2	0	0	0	0
30	0	0	0	0	0
30	2	0	0	0	0
30	5	0	0	0	0
30	10	0	0	0	0
30	15	0	0	0	0
30	0	5	0	0	0
30	0	10	0	0	0
30	0	15	0	0	0
30	0	0	5	0	0
30	0	0	10	0	0
30	0	0	15	0	0
30	0	0	0	5	0
30	0	0	0	10	0
30	0	0	0	15	0
30	0	0	0	0	-10
30	0	0	0	0	-5
30	0	0	0	0	5
30	0	0	0	0	10

Table 1: Simulation runs choosen for static stability

As can be noted in Table 1, computation time can usually be saved by the port-starboard symmetry of the aircraft for changes in β. Derivatives with respect to control surface deflection can be done with model configurations in SOLIDWORKS. As with the velocity based CFD runs, computation time can be saved by using the symmetry of the ailerons and rudder deflections.

The preferred approach to analyzing dynamic stability in CFD would be to begin a time dependent flow and allow the parameters to come to a steady state before applying the deflection of the control surface. A quasi form of this can be done in STARS where the area of the control surface can apply a local velocity to the flow. This change in flow approximates a control surface deflection and is done in a time stepped solver. SOLIDWORKS CFD has a similar set of features, but time dependent parameter functions for deflection of a control surface are not available. Quasi-steady cases can be used in place of the fully time dependent solver. Setting up a run for rotational

flow about the center of gravity (CG) of the aircraft can approximate a change in a body rotation rate. This is not fully representative of the unsteady aerodynamics of an aircraft, but as a design tool quasi-steady cases can be helpful to point out possible issues with the aircraft.

Validation for 2D airfoils in SOLIDWORKS with $k-\epsilon$ has been previously explored by Wallace [30]. Wallace concluded that if more computation power were available, convergence could be found for a wider range of conditions. Continuing that work briefly with more computing resources will be presented later in this thesis.

One of the critical parameters for turbulent models such as the $k-\epsilon$ solver used by SOLIDWORK CFD is y^+. This parameter gives the position within a boundary layer and provides insight as to what part of the boundary layer is being resolved by the solver. From this information the height of the first cell away from a wall can be found. From White's textbook, the height of the first cell away from the wall can be found from

$$\Delta s = \frac{y^+ \mu}{U_{fric} \rho} \tag{8}$$

where

$$U_{fric} = \sqrt{\frac{\tau_{wall}}{\rho}} \tag{9}$$

where

$$\tau_{\text{wall}} = \frac{C_f \rho U_\infty^2}{2} \tag{10}$$

where

$$C_f = \frac{0.026}{Re_x^{1/7}} \tag{11}$$

where

$$Re_x = \frac{\rho U_\infty L}{\mu} \tag{12}$$

for a flat plate boundary layer [31]. ρ is the density of the fluid, Re_x is the Reynolds number, C_f is the coefficient of friction from an approximation of the Moody diagram, τ_{wall} is the shear stress of the fluid, and Δs is the size of the cell. Assuming that the upper or lower surfaces of an airfoil are approximately a flat plate, these equations hold true. This closeness and size of cells is referred to as the Law of the Wall. Each type of solver has a range of y^+ values that the solver can resolve. SOLIDWORKS Flow Simulation uses boundary layer approximations when the y^+ of the cell is below the minimum for $k-\epsilon$ [29]. Other more computationally expensive solvers can solve the boundary layer close to the wall and achieve more accurate results.

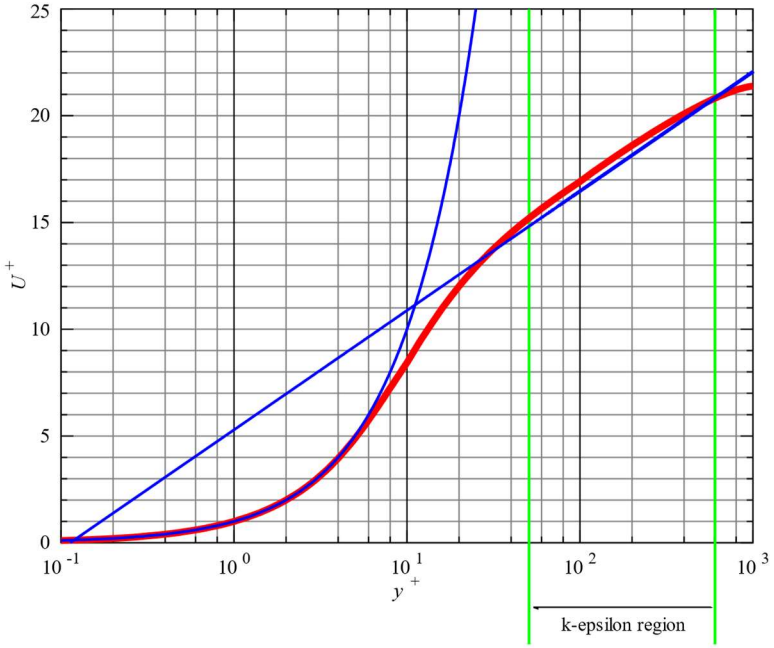


Figure 9: Law of the Wall for $k-\epsilon$

Another important parameter for turbulent models is the free-stream turbulence intensity, Tu. Those who are familiar with the program X-Foil will know of or have seen the parameter N_{crit} at some point. Coder provided a link between these 2 parameters, based on previous work by Drela, to facilitate similarity of inputs between the different solvers they were focused on [32]. Coder

states that the relationship is highly dependent on the mechanics of the chosen solver, but for this case of exploration it is a convenient starting point.

CHAPTER II

APPARATUS AND METHODOLOGY

AIRCRAFT USED FOR EXPERIMENTS

The aircraft that was used for testing is an Edge 540 1300 mm model from the manufacturer FMS. Stock servos, batteries, and linkages were used. The manual for the airframe states that the center of gravity (CG) of the model should be located at 80 mm from the leading edge. The location of the CG was held constant as other components were added to the airframe. The stock 3948 760 KV motor with a 2 blade APC 11x5.5E propeller was used for propulsion. Table 2 shows the inertia matrix of the aircraft in the modified configuration as measured using a bifilar pendulum method [33]. Coupled terms were assumed to be negligible and are not used by X-Plane.

Axis	Inertia [slug-ft^2]	Radius of Gyration [ft]
X	9E-03	0.262
Y	3.60E-02	0.524
Z	2.60E-02	0.445

Table 2: Inertia matrix for FMS Edge 540

Figure 10: Image of Edge 540 model used

A Cube Orange with a standard ADSB carrier board was used as the autopilot. An RFD 900x was used for telemetry connection to the ground control station (GCS). 128-bit AES encryption was turned on and power output was set to 20 dB. Stock quarter wave antennas were used in the aircraft. The RFD 900x modem used firmware version 3.15.

A SOLIDWORKS model of the Edge 540 was created using simple features as would be used for a preliminary model for a new aircraft. Control surfaces were placed and hinged along the outside surface of the skin.

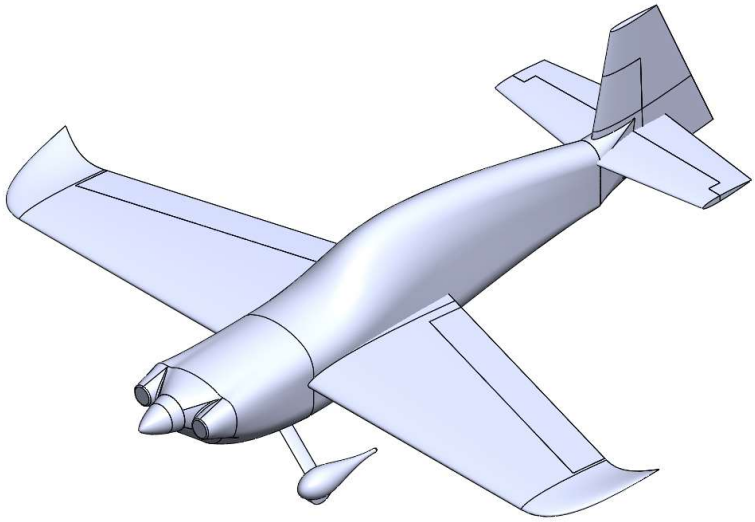


Figure 11: SOLIDWORKS CAD model of Edge 540

This model was used to quickly gather measurements to be used in equations for mode estimation methods and in SOLIDWORKS Flow Simulation. A standard 3-view drawing of the model is included in the Appendix. SOLIDWORKS 2019 SP 4.0 was used to model the aircraft. SOLIDWORKS 2019 SP 5.0 was used for CFD as there were a few stability fixes.

An X-Plane model of the aircraft was also created. This model included dimensionally accurate versions of the control surfaces, motor, and propeller. The fuselage was held approximate as the shape of the fuselage does not have a significant role in the FDM beyond calculating skin friction and frontal area drag. There is a consideration for fitness ratio for transonic flight, but this aircraft will not be flown outside of the incompressible region below Mach 0.3.

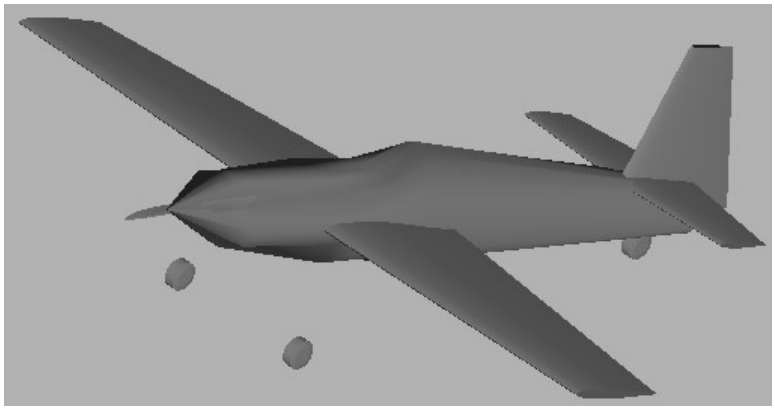


Figure 12: X-Plane model of Edge 540

SOFTWARE USED FOR FLIGHT TESTING

Ground control software (GCS) is used to connect to vehicles to receive telemetry and transmit commands. Since ArduPilot uses the MAVLink protocol to send and receive data from ground stations and other aircraft, there are a few options for ground control software. Mission Planner was selected as the GCS for flight testing because of the author's familiarity with the interface. MAVProxy is a ground control software that is primarily used for development of features and rapid testing of the autopilot software. MAVProxy can be used to access features not yet available in third party GCS programs. Mission Planner version 1.3.72 and MAVProxy version 1.8.19 were used.

SOFTWARE FOR VALIDATING ARDUPILOT SITL

A rigid body simulator was created using MATLAB and the ArduPilot JSON interface in order to assist in validation of the attitude controllers. The purpose was to make use of the attitude controller with a rotation error and track the servo output from ArduPilot. This can be used to show that ArduPlane SITL running on an x86 processor provides the same outputs as running ArduPlane on an STM32 processor. Initially, a procedure of tiling the physical autopilot on a wood block with two flat faces cut at an angle was used to match what the MATLAB simulation was used. After a discussion with Andrew Tridgell and Peter Barker, a procedure where just the AHRS_TRIM

parameters needed to be changed in order to measure the validation outputs was created. Adjusting the AHRS_TRIM parameters would also be used when performing a typical pre-flight procedure of an aircraft that is too large to be picked up and rotated by hand. The default parameters for SITL include some amount of noise and drift for the sensors. Normally these parameters are helpful for attempting to imitate what happens with physical sensors, but for validation it can cause issues. These features were turned off via various parameters starting with "SIM_". This was done for consistent values during the validation process between the virtual and the real.

For physically testing the same procedure, the autopilot must have a 3D GPS lock. This is because of the code path taken to determine what gains to use for the attitude controllers. If no GPS is connected, then no airspeed estimate is calculated. The fakegps module from MAVProxy was used to speed up the testing process as waiting for a GPS position to be reported from a physical module takes longer. The speed scalar without any airspeed estimate is different from the speed scalar with an airspeed of zero. During the discussion with some of the ArduPilot developers, Peter Barker added the airspeed scalar value to the AETR dataflash log message.

SIMULATION SOFTWARE

The software in the loop (SITL) configuration used is partially documented on the ArduPilot wiki [6]. As X-Plane 11 has received updates, parts of the documentation for X-Plane 10 have to be used creatively to work correctly. With a little bit of inferring on where the settings from X-Plane 11 have been moved in the menus, the X-Plane 10 instructions can be used without issue. Windows Subsystem for Linux version 1 with Ubuntu 18.04 and the build environment set up by the bash script in the ArduPilot repository was used to compile ArduPlane and to start the SITL environment. The ArduPilot developer section of the online documentation includes thorough instructions on how to setup an environment for SITL testing. X-Plane version 11.33 was used. The software Steam by Valve provides ways to roll back to previous versions of software should that be necessary.

Part of the background operations of X-Plane is called datarefs. These datarefs provide many parameters that change controls, throttles, landing gear, and many other parts of the aircraft being modeled. Normal operation of ArduPilot SITL for X-Plane does not use these datarefs. This is because the dataref for throttle, as an example, contains 8 different indices for throttle. From a user perspective, the index of the throttle that is used for the propeller might not be the one that is expected, but it would still work as intended if control is set for all throttles rather than a singular throttle. As an alternative, generalized commands to control the aircraft can be sent, but are applied to all parts of the aircraft under the same category. For aircraft with multiple throttles, multiple ailerons, and custom controls, datarefs are the way to control those with ArduPilot SITL and X-Plane running together. These unique controls can be added to the SIM_XPlane.cpp file and mapped to controls output from ArduPilot. Tridgell created mappings from Mixture3 and Mixture4 controlled in X-Plane to RC channels 6 and 7, respectively. Small snippets of code were added as well as changing parameters in ArduPlane to map Mixture3 to the SERVO6 output channel. This allowed the control of a second engine in X-Plane while SITL was running.

CFD for the Edge 540 model aircraft provided static stability parameters. More than enough CFD runs were performed with runs taking between 10 and 20 hours to reach convergence. An Intel i9 9900K with 64 GB of 3200 MHz RAM was used to run the CFD simulations. Typically, convergence of the flow parameters for a full aircraft happened around 7 to 10 million cells with some cases requiring 14 million cells when near a flow separation condition. 6 to 12 hours were needed to complete each case. Table 3, Table 4, and Table 5 detail the settings used for SOLIDWORKS Flow Simulation.

General Settings							
Analysis Type		Fluids		Wall Conditions		Turbulence Parameters	
		Project Fluid	Flow Type	Thermal Condition	Roughness	Intensity	Length
External	Not time-dependent	Air	Laminar and Turbulent	Adiabatic	0 µin	0.10%	0.01 in

Table 3: SOLIDWORKS Flow Simulation General Settings

Computational Domain						Global Mesh	
Type	Forward	Aft	Span	Upper	Lower	Initial	Ratio
3D	3b	7b	5b	5c	5c	4	1

Table 4: SOLIDWORKS Flow Simulation 2D Computational Domain settings

Calculation Control Options						
Finishing				Refinement		
Criterion to stop	Goals	Refinements	Analysis Interval	Global Domain	Maximum Cells	Strategy
All	All	6	1.5 Travels	6	14 million	Periodic

Table 5: SOLIDWORKS Flow Simulation Calculation Contorl Options

FLIGHT TESTING METHODS

Flight and safety standard operating procedures for the Edge 540 model follow procedures used for many other aircraft at the OSU MAE department. The aircraft undergoes a complete system check before being placed on the flight line and handed over to the Pilot in Command. A detailed list of preflight checks follows.

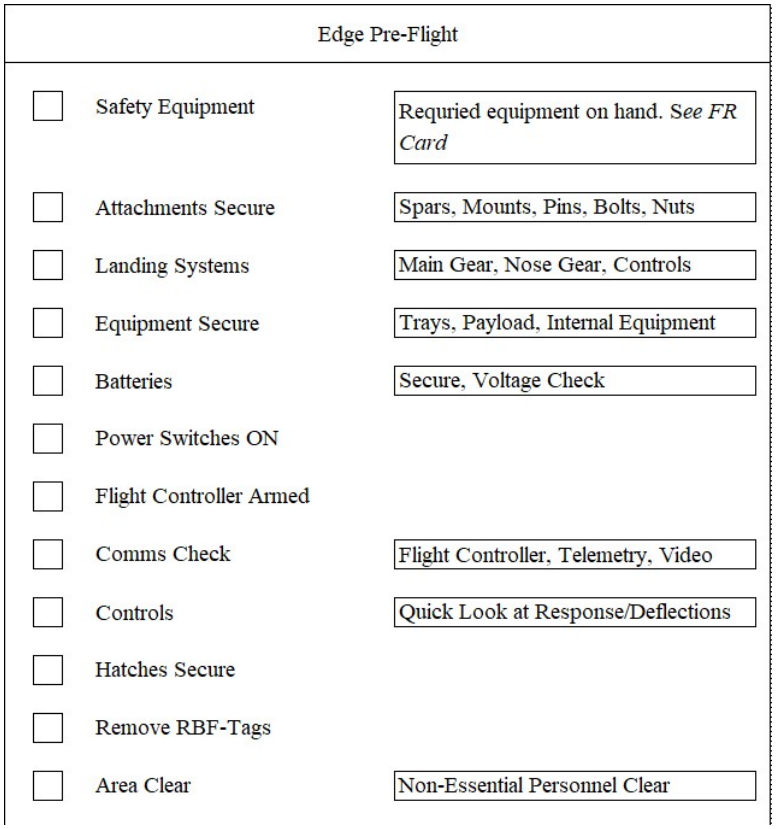


Figure 13: Pre-Flight checklist for Edge 540

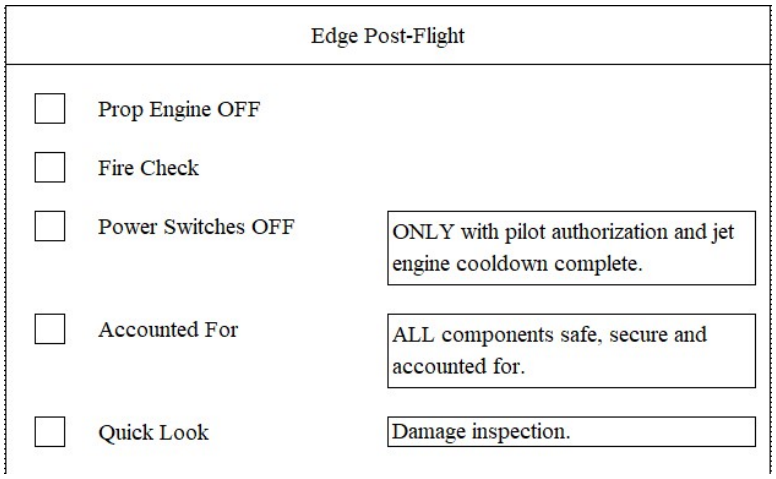


Figure 14: Post-Flight checklist for Edge 540

Just as with testing manned aircraft, the test points that will be targeted during the flight testing are provided in flight data card format. These include many of the details for what conditions

allow the test point to be performed safely and what data is important [21]. Transparent communication from designers to pilot is critical for reducing the risk during flight. Awareness of what actions the autopilot will take as well as what actions to take should the autopilot fail or act unexpectedly will be shown to have enormous potential to mitigate risk.

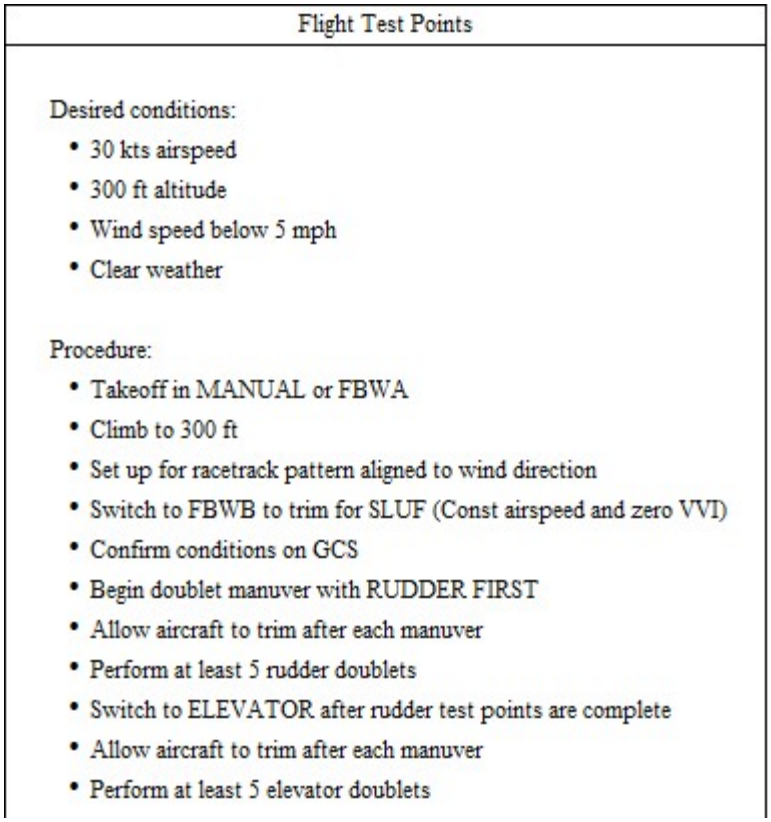


Figure 15: Flight test points card with procedures

These flight test conditions were decided based on data provided by the analytical methods. The aircraft was trimmed at 30 kts airspeed for both simulations and flight testing. The altitude of 300 feet was chosen as a safety precaution should the aircraft enter an unsafe state. The altitude would provide at least a longer amount of time to attempt recovery of the aircraft.

Table 6: Test points

	Axis	Mode	Parameter set	Purpose
1	Pitch	MANUAL	N/A	Mode identification
2	Yaw and Roll			
3	Pitch	FBWA	1	Current controller gains
4	Yaw			
5	Pitch		2	Previous controller gains
6	Yaw			
7	Pitch		3	Controller oscillation gains
8	Yaw			
9	Pitch		4	Integrator build up
10	Pitch	FBWB		

Table 7: Attitude controller parameter set 1

Parameter Name	Value
PTCH2SRV_D	0.07
PTCH2SRV_FF	0
PTCH2SRV_I	0.1
PTCH2SRV_IMAX	1000
PTCH2SRV_P	1.5
PTCH2SRV_RLL	1
PTCH2SRV_RMAX_DN	0
PTCH2SRV_RMAX_UP	0
PTCH2SRV_TCONST	0.5
RLL2SRV_D	0.06
RLL2SRV_FF	0
RLL2SRV_I	0.09
RLL2SRV_IMAX	1000
RLL2SRV_P	0.8
RLL2SRV_RMAX	0
RLL2SRV_TCONST	0.5
YAW2SRV_DAMP	0.05
YAW2SRV_IMAX	1500
YAW2SRV_INT	0
YAW2SRV_RLL	1
YAW2SRV_SLIP	1

Table 8: Attitude controller parameter set 2

Parameter Name	Value
PTCH2SRV_D	0.05
PTCH2SRV_FF	0
PTCH2SRV_I	0.12
PTCH2SRV_IMAX	3000
PTCH2SRV_P	1
PTCH2SRV_RLL	1
PTCH2SRV_RMAX_DN	0
PTCH2SRV_RMAX_UP	0
PTCH2SRV_TCONST	0.5
RLL2SRV_D	0.06
RLL2SRV_FF	0
RLL2SRV_I	0.09
RLL2SRV_IMAX	1000
RLL2SRV_P	0.8
RLL2SRV_RMAX	0
RLL2SRV_TCONST	0.5
YAW2SRV_DAMP	0.1
YAW2SRV_IMAX	1500
YAW2SRV_INT	0
YAW2SRV_RLL	1
YAW2SRV_SLIP	1

Table 9: Attitude controller parameter set 3

Parameter Name	Value
PTCH2SRV_D	0.07
PTCH2SRV_FF	0
PTCH2SRV_I	0.1
PTCH2SRV_IMAX	1000
PTCH2SRV_P	1.5
PTCH2SRV_RLL	1
PTCH2SRV_RMAX_DN	0
PTCH2SRV_RMAX_UP	0
PTCH2SRV_TCONST	0.5
RLL2SRV_D	0.06
RLL2SRV_FF	0
RLL2SRV_I	0.09
RLL2SRV_IMAX	1000
RLL2SRV_P	0.8
RLL2SRV_RMAX	0
RLL2SRV_TCONST	0.5
YAW2SRV_DAMP	0.3
YAW2SRV_IMAX	1500
YAW2SRV_INT	0
YAW2SRV_RLL	1
YAW2SRV_SLIP	1

Table 10: Attitude controller parameter set 4

Parameter Name	Value
PTCH2SRV_D	0.4
PTCH2SRV_FF	0
PTCH2SRV_I	0.15
PTCH2SRV_IMAX	3000
PTCH2SRV_P	1
PTCH2SRV_RLL	1
PTCH2SRV_RMAX_DN	0
PTCH2SRV_RMAX_UP	0
PTCH2SRV_TCONST	0.5
RLL2SRV_D	0.06
RLL2SRV_FF	0
RLL2SRV_I	0.09
RLL2SRV_IMAX	1000
RLL2SRV_P	0.8
RLL2SRV_RMAX	0
RLL2SRV_TCONST	0.5
YAW2SRV_DAMP	0.05
YAW2SRV_IMAX	1500
YAW2SRV_INT	0
YAW2SRV_RLL	1
YAW2SRV_SLIP	1

Each test point was chosen to test characteristics of the aircraft and partially demonstrate the procedures proposed in this thesis. Test points 1 and 2 are for gathering data on the characteristics of the short period and Dutch roll of the aircraft. Test points 3 through 9 use the FBWA mode with different sets of gains to demonstrate how the controllers behave in comparison between virtual and real. The parameter set 1 values are not yet perfect, but they are much closer than those provided as the defaults for ArduPlane and have been handed tuned for the desired handling qualities. Parameter set 2 a set of gains that was used previously in the process of tuning the aircraft. This is to show a different set of outputs with the same inputs and in the same mode. Test points 5 and 6 were chosen to demonstrate pitfalls that can be encountered during the tuning

process. Parameter set 3 will attempt to show oscillation produced by the controller and parameter set 4 will attempt to show sluggish vehicle controlled due to too much integrator control.

AIRCRAFT MODE IDENTIFICATION METHODS

Hood hypothesized during the analysis of his results that, “a pristine input function is not necessary as long as enough energy is placed evenly into the band of frequencies that are intended for system identification.” He also noted that it would be, “extremely difficult to get a clean control input without resorting to programming an autopilot’s flight control routine to command one.” [3] His coherence data showed that this hypothesis was true, but there might still some room for improvement by having the autopilot command a pristine input function. Programming the autopilot to do this is now well within reach and no longer extremely difficult.

To excite the desired Dutch Roll flight mode, a rudder doublet control input was used. For the short period mode, an elevator doublet was used. Seamans et. al. covered some methods of static and dynamic control inputs for traditional, linearly dependent aircraft [17]. Kimberlin details control inputs and flight data analysis methods to determine frequency and damping of aircraft modes [21]. Kimberlin also provides insight to how to isolate modes for easier identification from flight test logs by using a doublet rather than something that would have a greater disturbance on the trimmed flight of the aircraft.

ArduPilot recently added the ability to write Lua scripts to perform many functions that have typically been done by devices other than the autopilot. Tasks such as payload data gathering, vehicle state reactions, more advanced failsafe, and non-native device drivers have all been delegated to a companion computer, external microcontroller, or a script on a ground station. With Lua scripting available in ArduPilot version 4.0 and newer, many of these tasks can now be done internally on the autopilot. Lua scripts are run as the lowest priority task in the scheduler as their originally intended usage is non-flight-critical tasks.

One feature of ArduPilot Lua scripting that is very useful for mode identification is the ability to set servo outputs at precise times and positions. This can be used to input a precise and repeatable control surface deflection or other PWM driven action. The problem with the feature as it is implemented in ArduPlane 4.0 is that to move an elevator servo a parameter that defines the parameter that defines the function of the servo output as an elevator must be changed to a script output and then returned to an elevator after the Lua script is done. If the script were to fail by an internal-to-Lua error or external factor from the autopilot, the aircraft would be left without control of the elevator. As a workaround to this, a script from a GCS could be used to sense for status text messages signaling the start and end of servo function reassignment in a Lua script. This GCS script could then send correct parameters to the aircraft if the Lua script fails. This requires that a high quality, high speed telemetry link with low packet loss be maintained and that there is enough bandwidth available to send and parse the now emergency priority MAVLink commands.

Adding a new function to the ArduPilot source code and Lua script bindings to access the new function were done. This new feature allows ArduPilot to set a servo position for a set amount of time and then returning to the intended function provides a safer way to control any servo output of the autopilot. After discussing the changes with the ArduPilot developers during their weekly online call-in meeting and fixing some bugs, the code for this change was merged into the ArduPilot master branch with pull request #14366 [4]. Documentation for the new scripting bindings was also included in the ardupilot_wiki repository.

```
void SRV_Channels::set_output_pwm_chan_timeout(uint8_t chan, uint16_t value,
uint16_t timeout_ms)
{
    WITH_SEMAPHORE(_singleton->override_counter_sem);

    if (chan < NUM_SERVO_CHANNELS) {
        const uint32_t loop_period_us = AP::scheduler().get_loop_period_us();
```

```
        // round up so any non-zero requested value will result in at least
one loop
        const uint32_t loop_count = ((timeout_ms * 1000U) + (loop_period_us -
1U)) / loop_period_us;
        override_counter[chan] = constrain_int32(loop_count, 0, UINT16_MAX);
        channels[chan].set_override(true);
        channels[chan].set_output_pwm(value,true);
    }
}
```

This was quickly put to use by Dr. Andrew Tridgell for demonstrating control of a quadruped limb in the pybullet simulator [34]. Using this new tool in ArduPilot, a Lua script was written to perform consistent control surface inputs. The same script was used for SITL and flight testing. The complete version of the script is included in the Appendix.

Just as a pilot would have the flexibility of providing different inputs to the aircraft, the Lua script can be changed to provide different inputs. Parameters for doublet or singlet inputs such as magnitude and period can be quickly changed and uploaded to the aircraft before flight. To command the start of the maneuver, the PWM of an RC input channel is switched to higher than 1700 μs. As a safety measure, if the RC channel is switched to a lower PWM before the maneuver is completed servo override timers are cleared and the vehicle is returned to a safe flight mode from which the PIC may command the aircraft as normal.

ANALYSIS

Several methods of determining aircraft parameters from flight exist. Seamans et al. detail a triangular graphing method that captures a Fourier Transform which they also used with a mechanical calculator designed for Fourier Transforms [17]. These methods proved to be very valuable for the 1950s and laid the groundwork for more advanced methods to be created.

Additional advanced methods have been developed since then that include the use of the Fast Fourier Transform (FFT), the Discrete FFT (DFFT), and Kalman Filters [20].

Typical visualizations for input-output systems are the Bode plot and Root-Locus plot. Typically, these plots are acquired a priori from a model of the system. The Root-Locus plot allows quick identification of the roots, or modes, of the system. This provides information such as frequency and damping for each mode.

With the ArduPilot dataflash logging system, the time between data points can change. The approximate frequency of attitude data in ArduPilot is 50 Hz. Variability in the logging rate does have an impact on DFFT methods and leads to the need for a Non-Uniform FFT (NUFFT). The MATLAB function `nufft` provide implementations of this variation of the Fast Fourier Transform. Methods such as measuring periods of local maxima and minima from the attitude of the aircraft are also used. Damping ratio can be calculated from decay of the peaks or from methods described by Kimberlin [21].

LIMITATIONS

The servos used on the model aircraft have limitation such as update rate and movement speed. This will not be shown in the input or output data as no sensor to measure the angle of deflection of the control surface was used.

The main scheduler loop for ArduPlane runs at 50 Hz by default. This can be adjusted with the parameter SCHED_LOOP_RATE, but for stable aircraft higher loop rates are not necessary. Lua scripts in ArduPilot run with a minimum callback time of 1 ms with some background processing time or about 1000 Hz. Both loops have built in sleep or delay timers that allow other flight critical tasks which the autopilot is expected to perform the necessary time to complete.

Only stick fixed actions can be taken with this current apparatus. There is no telemetry link from the servos to the autopilot nor any way to measure the load being applied to the servo. Future

work could be done to create a sUAV system that is capable of stick free dynamics, but that is outside of the scope of this thesis.

UNCERTAINTY OF GYROSCOPES

Definitions of uncertainty will be taken from Figliola and Beasley as well as Kline [35] [36]. Further details specific to micro-electromechanical system (MEMS) gyroscope uncertainty is taken from Woodman as well as El-Sheimy et al. [37] [38].

The primary sensor that will be used at the time scales for short period and Dutch roll are the gyroscopes. The Cube Orange has an ICM-20948 from TDK Invensense as its primary source of attitude rates. As with many rate gyros of this class, the primary source of error is the noise. ArduPilot samples the gyroscopes at 8 kHz and the accelerometers at 4 kHz before applying a 1 pole low pass filter at 188 Hz and downsampling the data to 1 kHz. The relevant code for this can be seen in "AP_InertialSensor/AP_InertialSensor_Invensensev2.cpp" [4]. The full sampling rate of the inertial measurement unit (IMU) can be written to an SD card by ArduPilot using the inertial navigation system (INS) batch sampler feature. These batches will be contiguous within a set of samples, but there can be time gaps between sets due to limitations with the data logger used by ArduPilot.

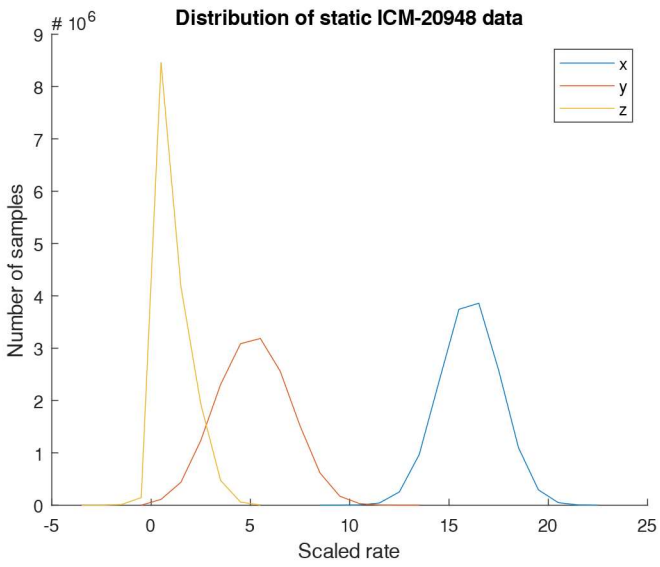


Figure 16: ISC-20948 static sampling data

A method that has been widely used to measure the noise and other parameters of IMUs is the Allan Variance [37] [39] [40]. From this method information such as random walk and bias instability can be found using long term data. A recommendation of several hours is made in several literature sources. The Allan Variance also provides insight to the variance from the noise that would be present after downsampling.

15 million samples of data of each axis were taken. This equates to about 32 minutes of gyroscope data and 1 hour of accelerometer data. The limitation to logging additional data is the size of the dataflash file on the SD card. ArduPilot uses the FAT32 file system which has a maximum file size of about 4 gigabytes (GB). IMU data must be logged for both the accelerometer and gyroscope at the same time in ArduPilot as well as some other logging data that cannot be disabled by a parameter change. To maximize the current logging abilities of ArduPilot, all other options in the LOG_BITMASK parameter were disabled. A custom firmware could be created to make better use of the 4 GB file size limit.

Figure 16 shows that the data for the x and y axes follow a general normal distribution while the z axis has a different kurtosis and skewness than a normal distribution. From Figure 18 it

can be found that the expected variance from the noise of the gyroscopes downsampled at 1 kHz is 5.8E-3 $°^2/s^2$ and a standard deviation of 0.076 °/s. All other information from the datasheet is explicitly or assumed to be reported at about 3σ or about 99% probability. This makes the uncertainty from the Allan Variance ±0.228 °/s (P99%). Since the noise is closely Gaussian, the Extended Kalman Filter that ArduPilot uses is a good fit to filter the data, but the uncertainty will still be present.

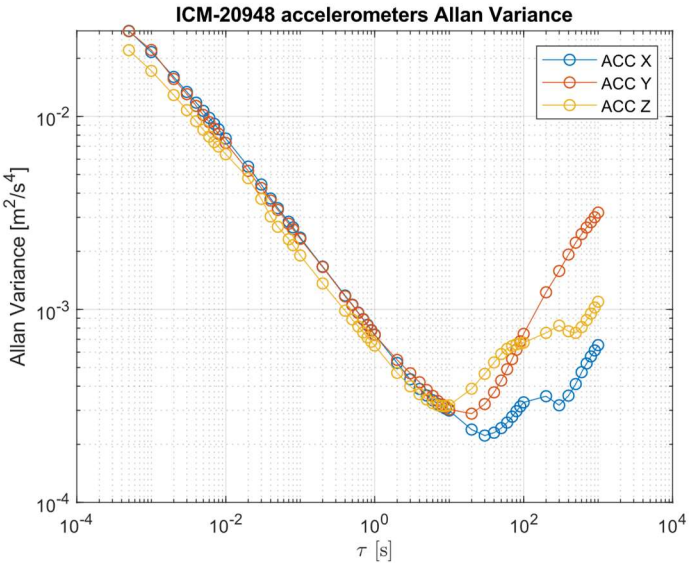


Figure 17: Allan variance of accelerometers in the ICM-20948 IMU

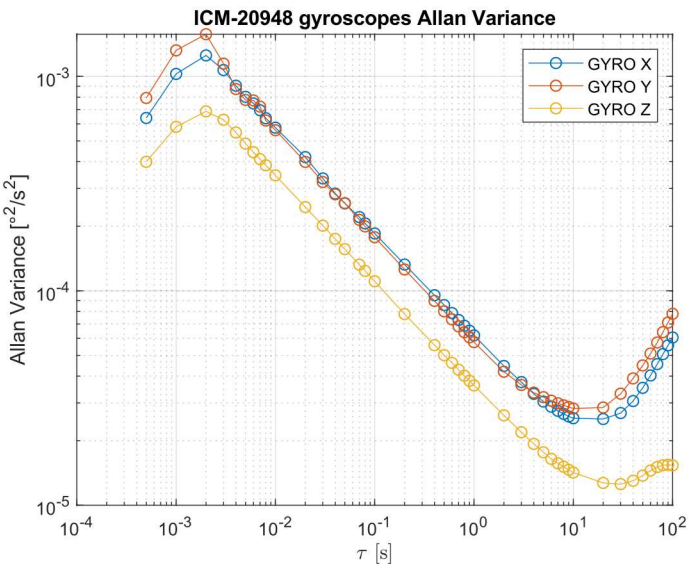


Figure 18: Allan variance of gyroscopes in the ICM-20948 IMU

The autopilot is equipped with a heater and maintains a constant temperature of 45 °C. The autopilot is also rebooted after reaching a constant temperature so that any bias from temperature effects is learned. The sensor is setup by ArduPilot at boot to have a range of ±2000 °/s. Quantization error for the ICM-20948 is based on the 16-bit sensor output and is 0.031 °/s (P99%). This is less than half of the uncertainty from the Allan Variance.

UNCERTAINTY FROM AN EXTENDED KALMAN FILTER

Using a Kalman Filter to determine states of an aircraft from sensors has been done for many years [20]. This is the uncertainty of a measurement from a single or a group of measurement sources rather than uncertainty as defined by Kline. Since part of the Kalman Filter deals with statistical variation of the sensors, the uncertainty of the measurement can be found from the filter. ArduPilot provides the one standard deviation uncertainty from the Extended Kalman Filter in the dataflash logs under the message NKF4 which has most of the EKF2 variance information. The data is logged as the square root of the variance which is equal to the standard deviation. As Eichstaedt shows, this is a form of uncertainty under the Internal Organization for Standardization's

Guides to the expression of uncertainty in measurement (ISO GUM) [41]. Therefore, taking YCS from the NKY0 dataflash log message provides the uncertainty of the yaw measurement. For roll and pitch the answer is more ambiguous for EKF2. The value of errRP in the NKF4 message combines the roll and pitch axes. EKF3 provides the same errRP data as well as the full variances from the EKF states in the XKV messages. Since EKF3 is still considered to be in an experimental stage of testing and intended for tailsitter or other non-traditional aircraft, it will not be used.

CHAPTER III

RESULTS

STATIC ANALYSIS

SOLIDWORKS Flow Simulation static analysis was used to verify data found using the analytical methods. Information such as coefficient of lift and coefficient of moment closely matched 2D to 3D approximations used with StabCon-S. Static Margin was also found from the CFD data and is included in Table 14. As should be expected, the linear region of the coefficient of lift curve matches very closely between theoretical data. As the aircraft stalls, there is a divergence in the coherence of the data. While not desirable to see, it is expected as stall is a very complex characteristic. Detailed characterization of the stall characteristics is outside of the scope of this thesis.

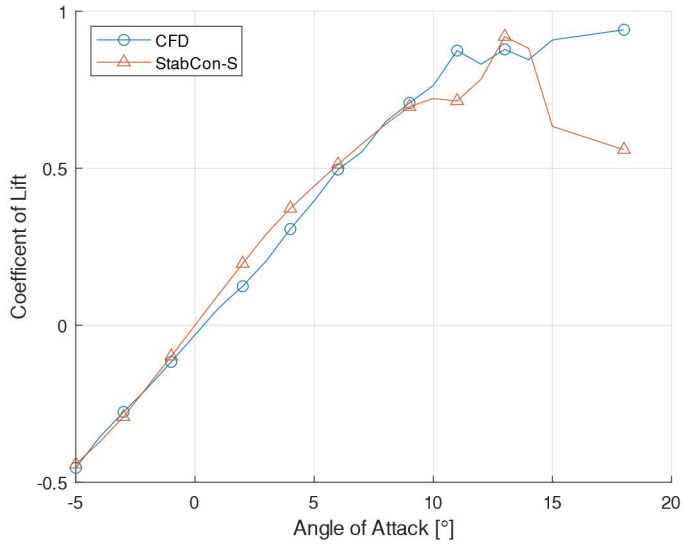


Figure 19: Coefficient of Lift from CFD and StabCon-S

For the coefficient of moment data from CFD, there is again clear agreement in a linear region between -2° and 3° angles of attack. The coefficient of moment from each is for the full aircraft. There are no solid hypothesizes for why the CFD data appears to have a larger linear region.

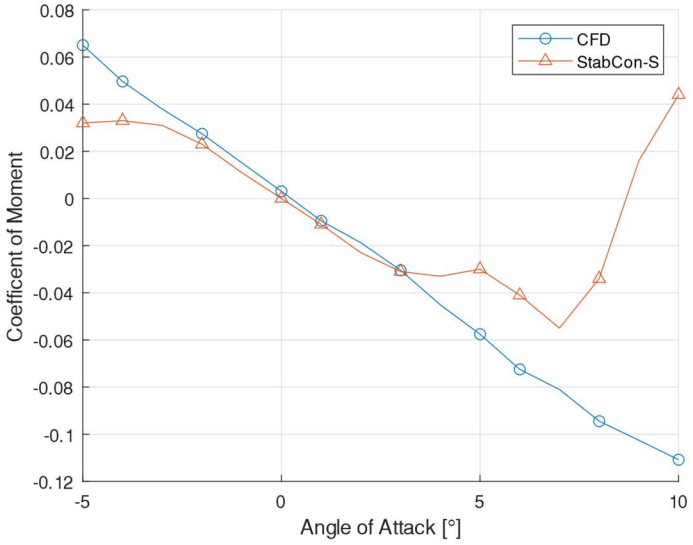


Figure 20: Coefficient of Moment from CFD and StabCon-S

For determination of the static margin from the CFD data the derivative of coefficient of lift with respect to alpha and the derivative of coefficient of moment with respect to alpha were found from the slopes of the linear regions of the previously presented data. The static margin from the CFD was found to be 14% and 3% from StabCon-S. A comparison of static margin from the various other methods is presented in Table 14.

ANALYTICAL METHODS MODE IDENTIFICATION

The full method followed to create the GeometricStabCon program is detailed by Nelson [12]. The methods used in Arena's StabCon-D program come from McCormick's textbook [14]. The longitudinal roots are visualized in Figure 21 and lateral roots in Figure 22. The identified mode characteristics for short period and Dutch roll are included in Table 11 and Table 12, respectively.

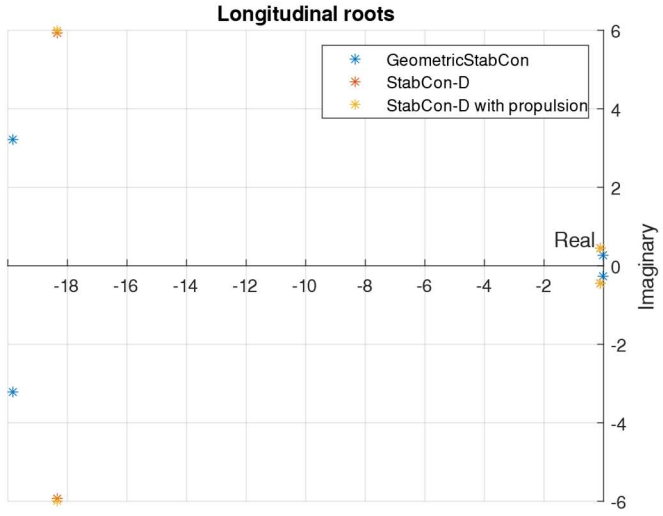


Figure 21: Roots of longitudinal modes from estimates

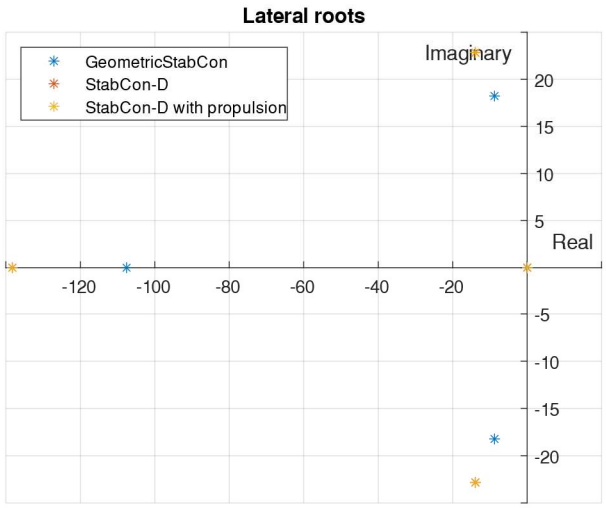


Figure 22: Roots of lateral modes from estimates

VALIDATION OF ARDUPILOT ATTITUDE CONTROLLER

The first set of plots, Figure 23 and Figure 24, show the issue of testing without suing the same code paths for the airspeed scalar. This is presented as an example of what experiments should be wary of while working with complex and deep code paths. As can be seen in Figure 23 and Figure 24, the pitch and roll controllers are providing an output to the servo rail due to and error in the desired angle and the actual angle. These results did not have an estimate for the airspeed of the vehicle. The value that is shown in these plots is not the same as the servo output value that is shown in Figures 25-28. This is due to the lack of a GPS position and has been discussed further in a previous section. Having a GPS position is important because it allows ArduPilot to estimate airspeed of the vehicle. Even if the airspeed estimated from GPS is zero rather than unknown due to lack of GPS, a different speed scalar value is used. By default, in SITL testing the GPS position is taken from the FDM. With physical testing however, either a physical GPS must be connected with a position lock or GPS data needs to be provided over MAVLink. In the spirit of repeatability and faster testing, the fakegps module provided with MAVProxy was used to provide the GPS data for the physical validation testing.

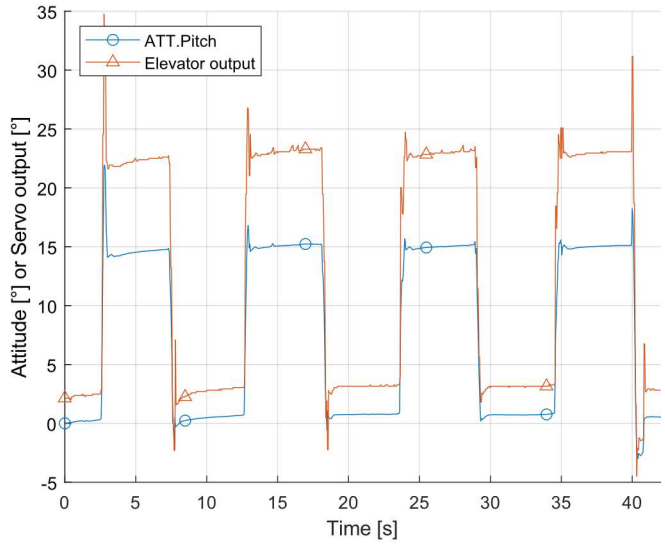


Figure 23: Incorrect pitch SITL controller validation on physical hardware

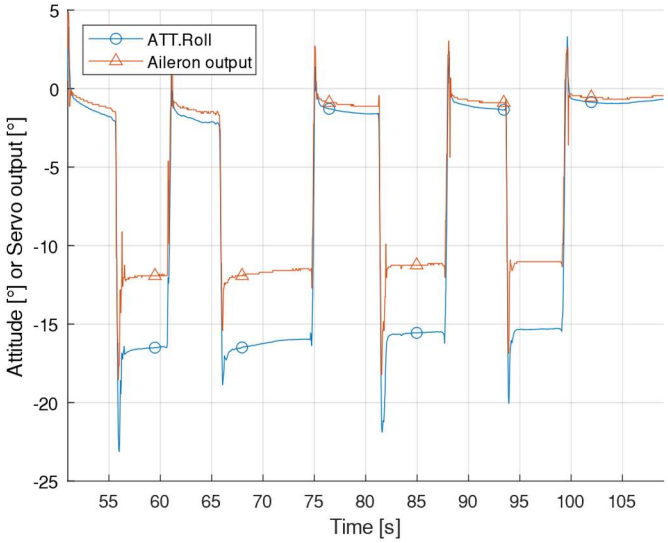


Figure 24: Incorrect roll SITL controller validation on physical hardware

The SITL simulator, using either the MATLAB simulator or the ArduPilot built-in simulator, produced the following outputs in Figure 25 and Figure 26. These results were created by adjusting the AHRS_TRIM parameters, but the simulator created in MATLAB would also

provide validating results. The physical corollary for the virtual SITL tests is shown in Figure 27 and Figure 28. Both the input and output values are the same for each axis.

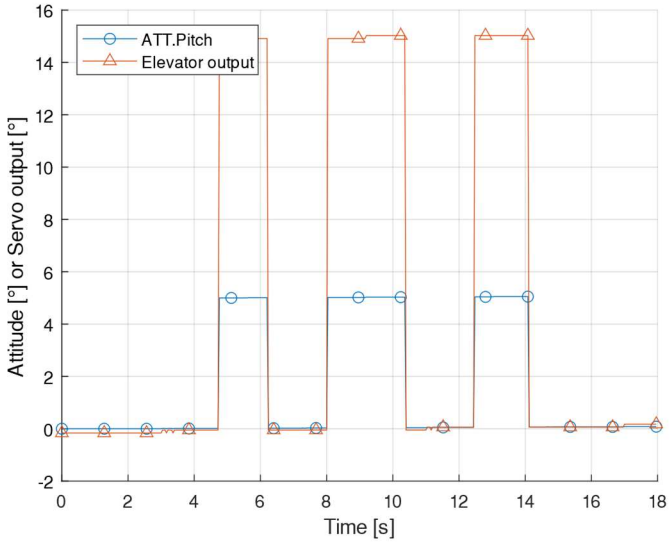


Figure 25: SITL pitch controller validation

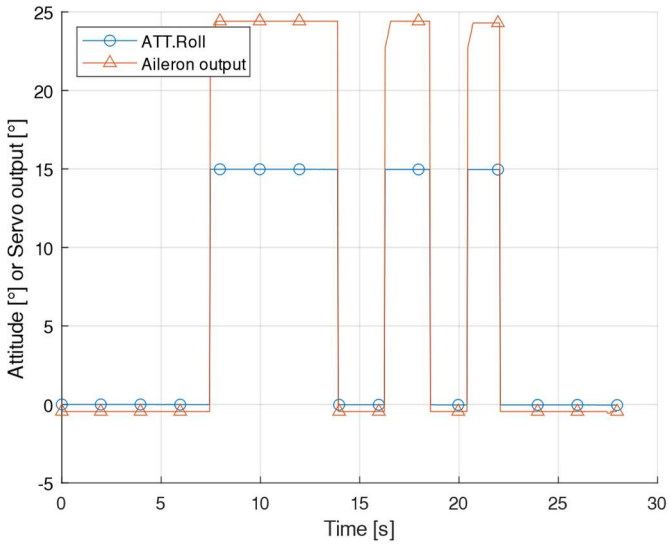


Figure 26: SITL roll controller validation

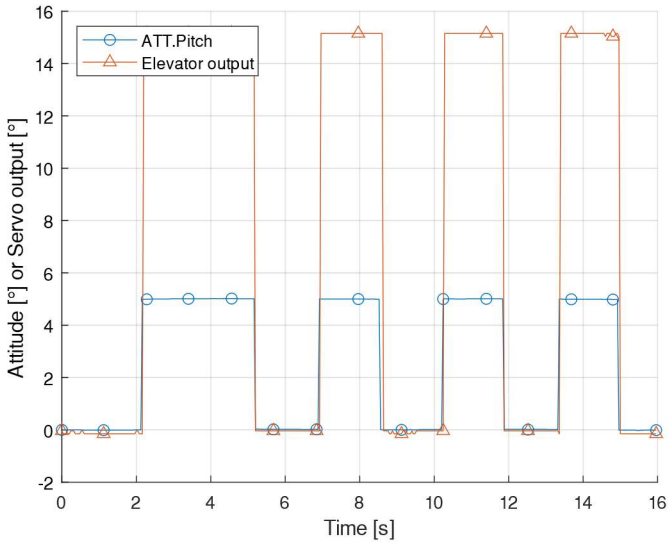


Figure 27: Physical pitch controller validation

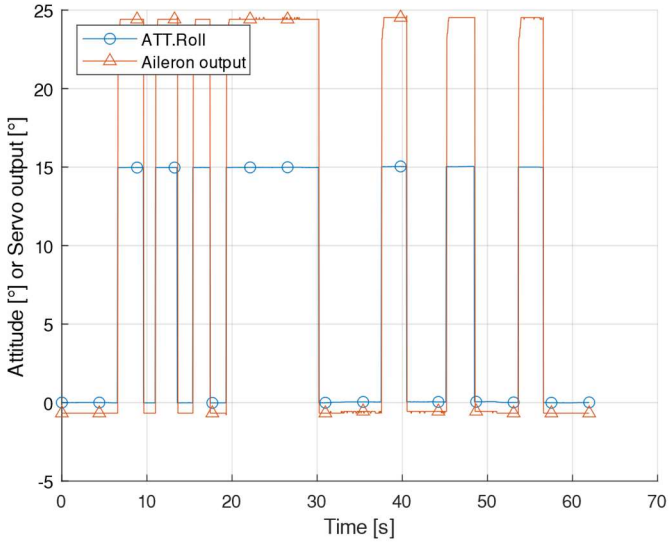


Figure 28: Physical roll controller validation

From this validation result, the extent of code utilized for the attitude controllers and speed scalar produce the same results from the same inputs and therefore are the same between a physical ArduPilot autopilot and ArduPilot SITL. This is significant as this provides confidence that any differences seen in later results are not due to differences in mathematics in the attitude controllers.

FLIGHT TEST POINTS

Each test point will be presented separately. The X-Plane 11 version of the maneuver will be shown first and the physical flight test data second. This is to allow for easier comparison of the characteristics of the FDM and actual flight dynamics.

Before getting too deep into the experiments, it was found that better data logging might be more useful. The first series of X-Plane testing with test point 1 was done using the default logging rate of attitude data at 25 Hz. This resulted is a lower number of data points for analyzing the period of any oscillation peaks and troughs than was desirable. In Figure 29 the data from a rudder doublet in X-Plane logged at a rate of 25 Hz is shown.

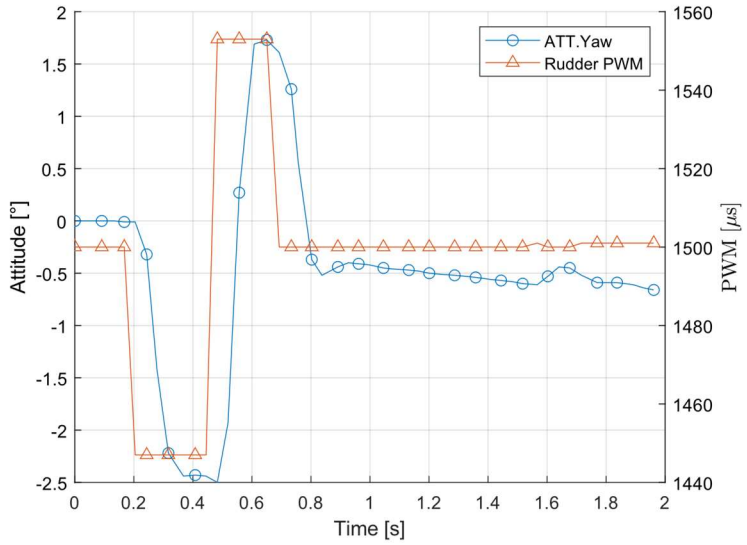


Figure 29: X-Plane test point 1 in the time domain (25 Hz)

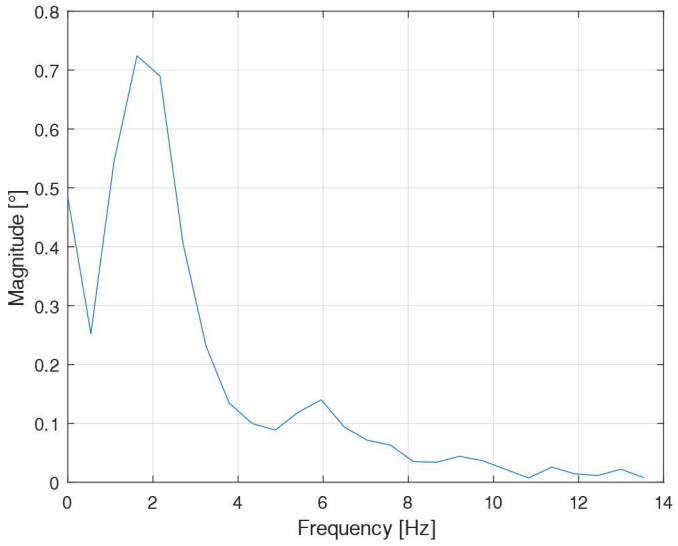


Figure 30: X-Plane test point 1 in the frequency domain (25 Hz)

The scheduler rate for the `Log_Write_Task` function was changed from 25 Hz to 50 Hz. This had no impact on the autopilot performance and is well below the default 400 Hz logging that is used on the same hardware by ArduCopter. This provided much clearer peaks and troughs in the time domain and more data points for the frequency domain.

TEST POINT 1 (SHORT PERIOD IDENTIFICATION)

For this test case the full SITL stack is running: X-Plane is the FDM, ArduPilot is the autopilot, and the doublet Lua script is running. The aircraft is allowed to trim using FBWB mode between trials and that trim position is used as the stead state value after the doublet input is completed. For most of these plots the attitude data will be on the left axis and the PWM value of the servo output will be on the right axis. The value of the PWM should not be considered significant as during a doublet the deflection of the control surface from the trim point is the same between the virtual and the real.

Figure 31 shows the multiple trials done for test point 1 and Figure 32 shows a detailed view of one of those trials. From these figures it can be seen that there is hardly any evidence of an

underdamped short period and leans heavily towards being critically damped. Figure 33 shows the frequency domain for the pitch output signal which does not have much significance.

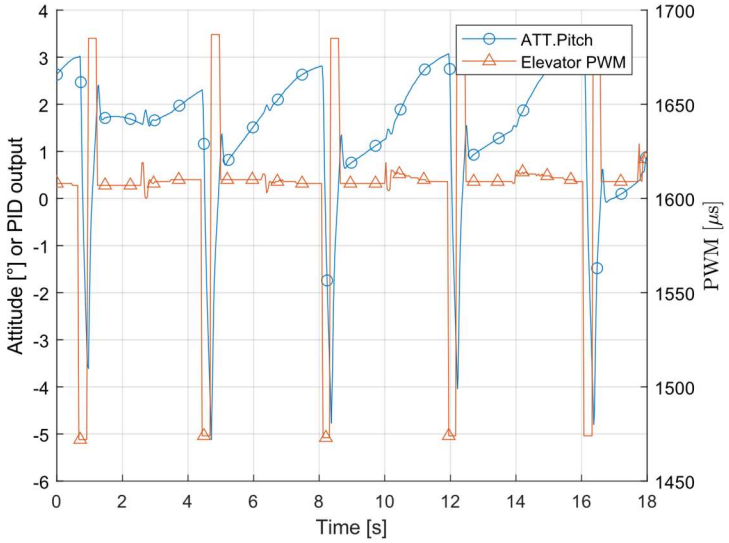


Figure 31: X-Plane test point 1 multiple trials

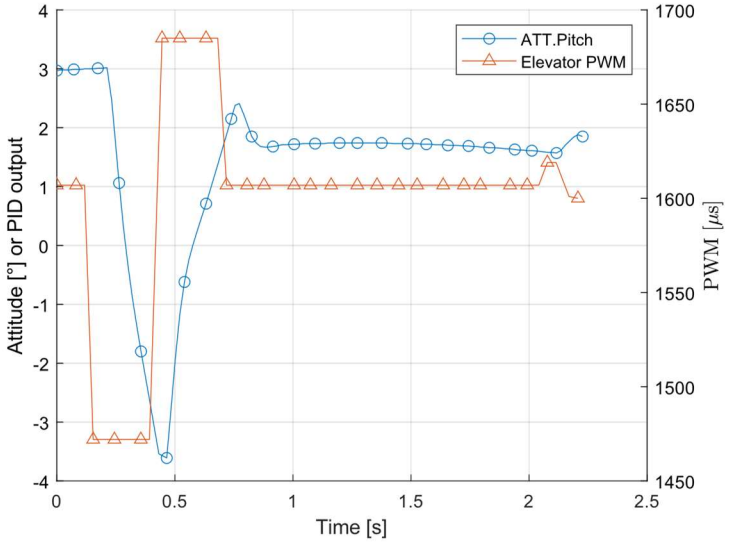


Figure 32: One trial of X-Plane test point 1 in the time domain

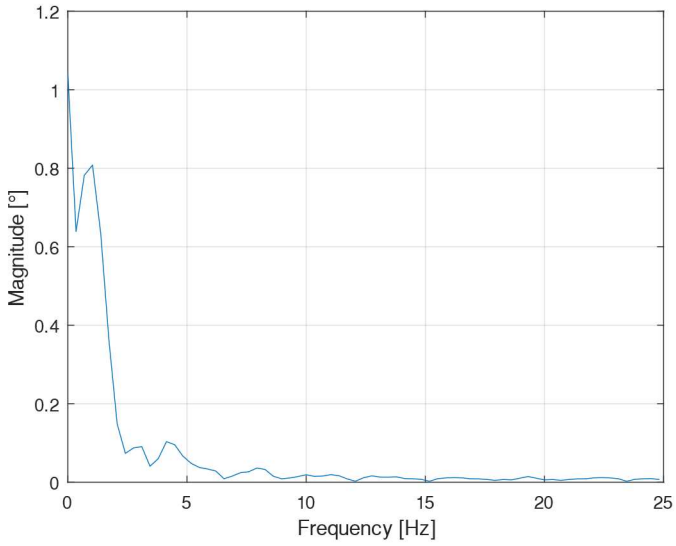


Figure 33: X-Plane test point 1 in the frequency domain

For the physical testing of test point 1, Figure 34 shows several of the trials that were done for test point 1 and Figure 35 shows a detailed view of one of these trials. From this plot it can be seen that there is no clear evidence of an underdamped short period. Figure 36 shows the frequency domain of the physical trails of test point 1 with an expected low significance.

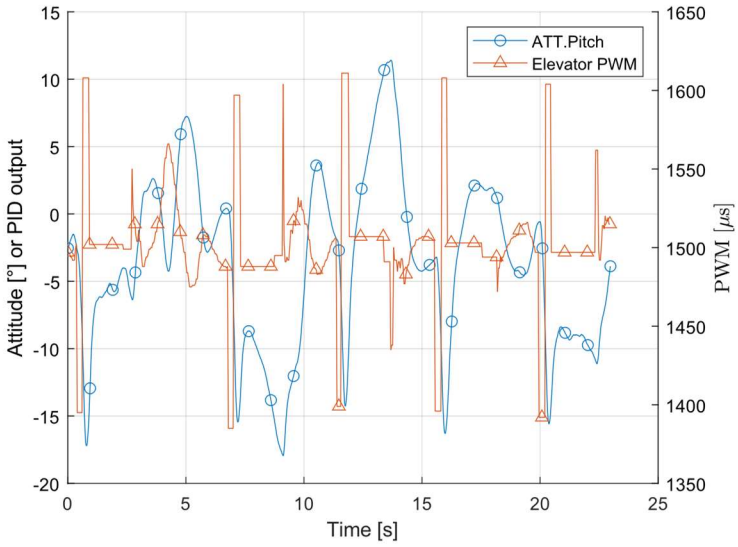


Figure 34: Physical test point 1 multiple trials

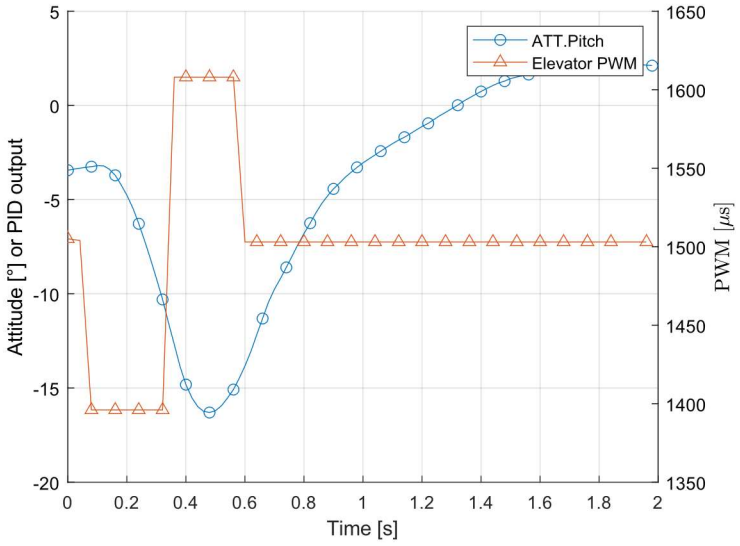


Figure 35: One trial physical test point 1 in the time domain

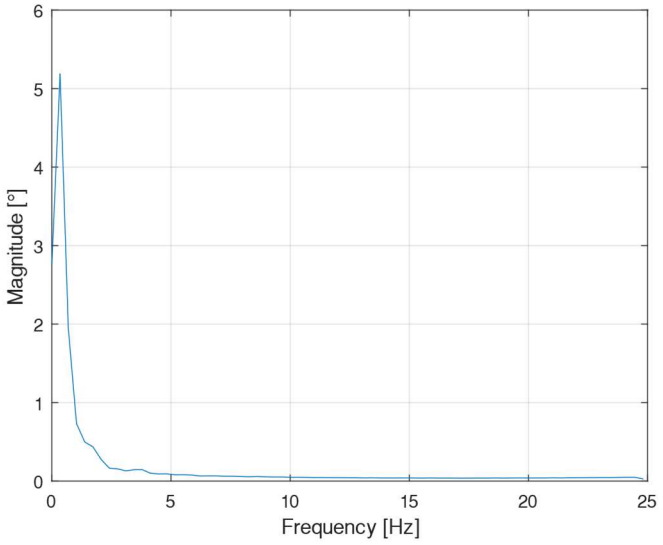


Figure 36: Physical test point 1 in the frequency domain

One curious part of the results of this test point is the sharpness of the change of pitch attitude at the start of the doublet response. This is predicted to be due to differences in the inertia model that X-Plane uses. This is troubling because the radius of gyration was measured on the real

aircraft and input into X-Plane. This seems to indicate that a deeper analysis needs to be done of the closed source FDM X-Plane.

TEST POINT 2 (DUTCH ROLL IDENTIFICATION)

Test point 2 is a very similar set up to test point 1 with the major difference being the change in axes from pitch to yaw. This was done to capture the Dutch roll behavior of the aircraft just as test point 1 was done the capture the short period behavior.

Figure 37 shows some of the trials done in X-Plane done for test point 2 and Figure 38 highlights one of the responses. In these plots there is a yaw dominant response which does hold constant for both X-Plane and physical testing. Figure 39 does not show anything too interesting in the frequency domain. As was seen in test point 1, the inertia model for X-Plane seems to be oddly different from that of the real aircraft. The same evidence from the sharpness of the start of the response is shown in this test point as well.

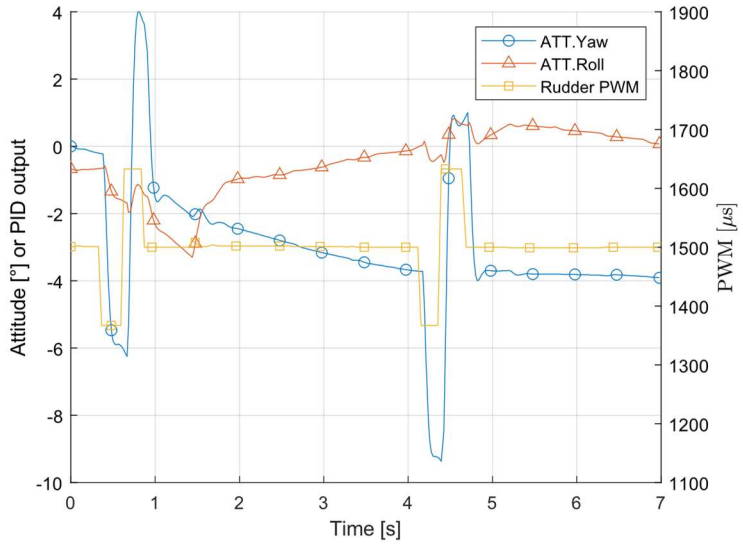


Figure 37: X-Plane test point 2 multiple trials

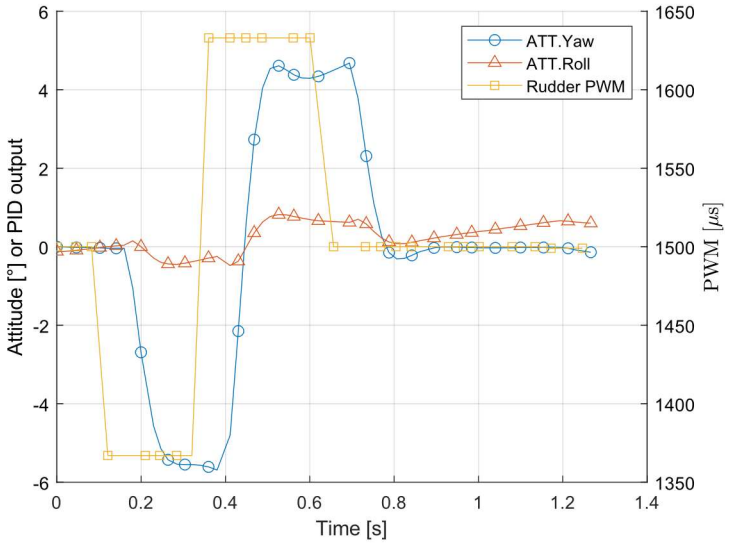


Figure 38: One trial of X-Plane test point 2 in the time domain

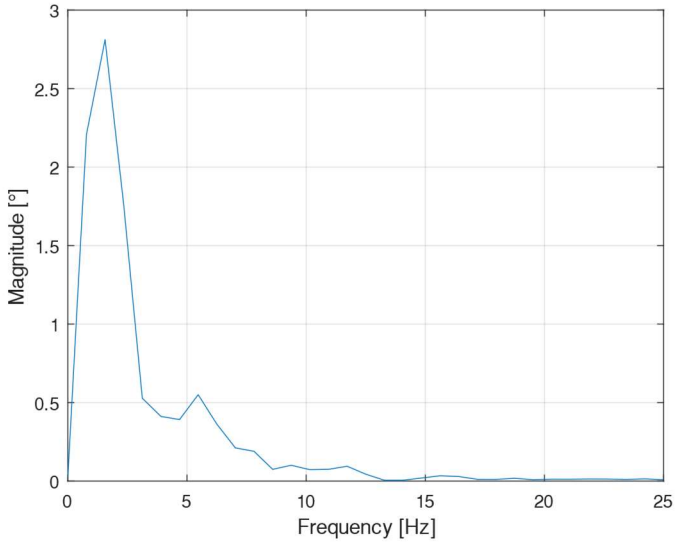


Figure 39: X-Plane test point 2 in the frequency domain

For the rudder deflections there was a clearly observable oscillation from the Dutch roll mode in the physical trials. Several trials are shown in Figure 40 and a detailed view of one trials in Figure 41. This is a textbook, classic Dutch roll response though. Clear peaks and troughs with a decaying response.

In Figure 42, the frequency of the response is shown to assist in determining the dominant frequencies. The frequency domain was only partially useful as determining what was from the input and what was from the response was difficult to determine with such a large delta between frequency points. For this test point it was also easy to measure the period of the response to assist in determining the frequency. Damping was determined by a log decrement method.

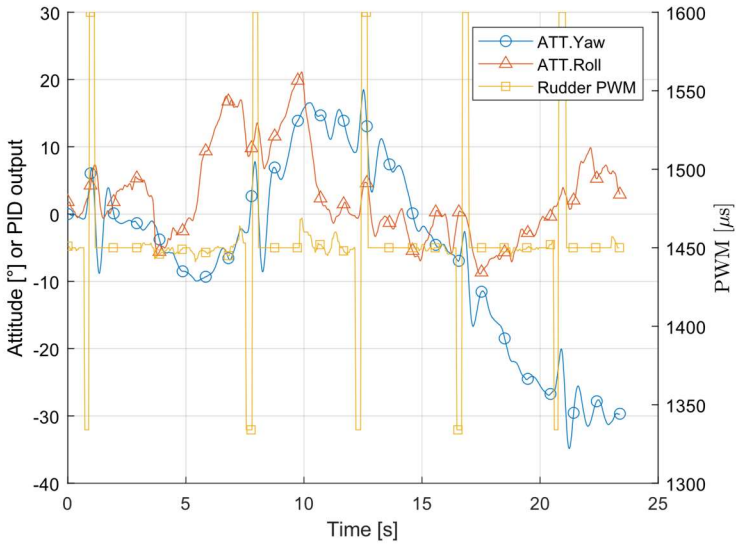


Figure 40: Physical test point 2 multiple trials

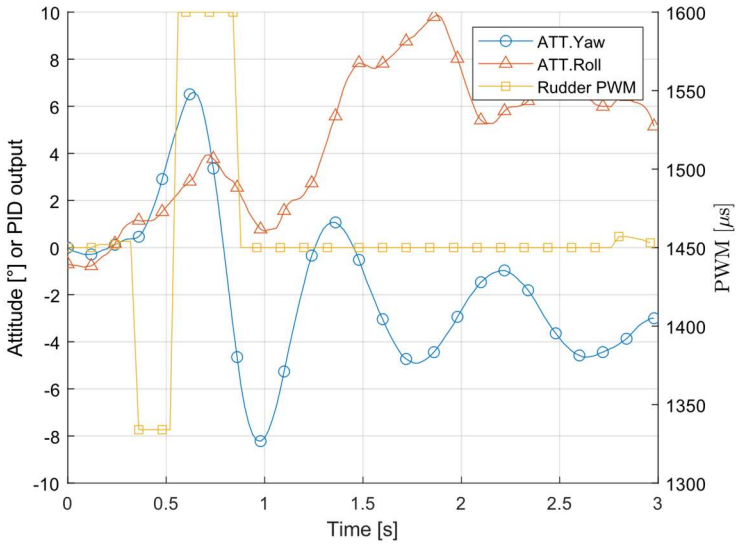


Figure 41: One trial of test point 2 in the time domain

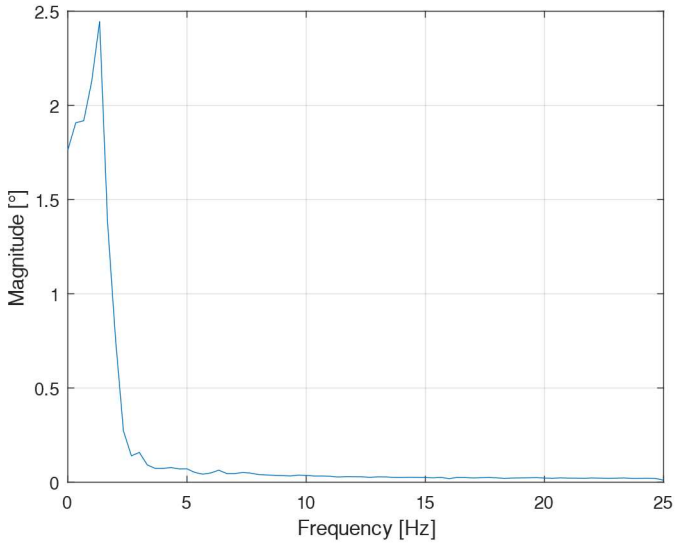


Figure 42: Test point 2 in the frequency domain

TEST POINT 3 (FBWA PITCH GAIN TUNING)

Test points 3 and beyond are done with the ArduPlane attitude controller running. The same input signal is used to better evaluate the controller response for tuning purposes. The input signal, ATT.DesPitch, is the target angle for the aircraft and the that is shown in the attitude of the aircraft. If there is a need to tune the output of the servos that information could be plotted as well, but for most cases tuning the attitude of the aircraft is the priority. The parameters shown in Table 7 are used for test point 3 and test point 4.

Figure 43 again shows the sharps changes in attitude in the X-Plane simulator. It also shows a clean response that could be considered well-tuned for the pitch controller. Figure 44 shows that the tuning carries over to the real aircraft well with a few small points that could be improved on. What improvements to make and how they were spotted is at the end of this section.

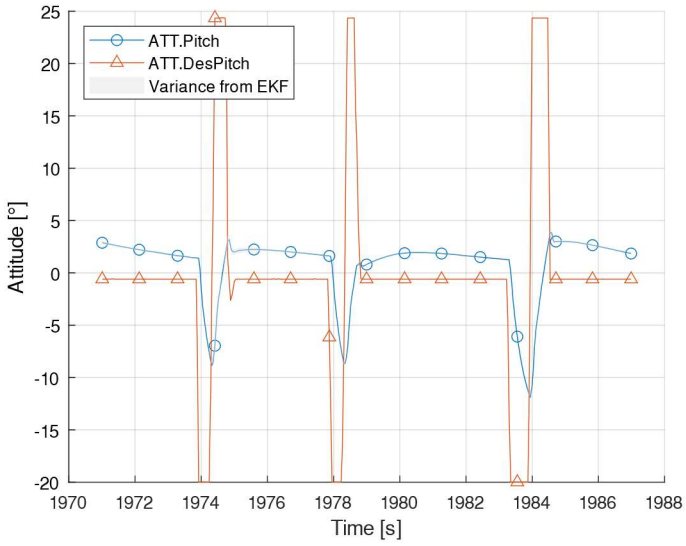


Figure 43: X-Plane test point 3

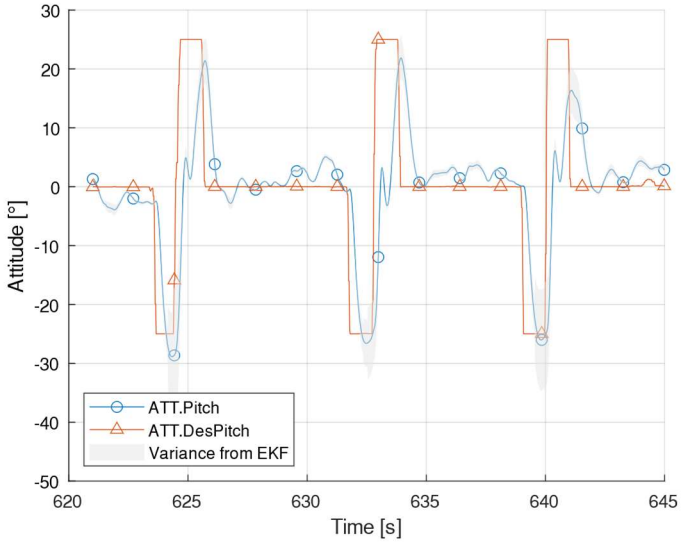


Figure 44: Physical test point 3

Looking at each of these figures in particular the impact that atmospheric turbulence has on the attitude of the aircraft between trials of the test point. X-Plane does have a turbulence model than can be turned on, but for the purposes of this work it is not very relevant.

The X-Plane result appears to suggest that the flap effectiveness of the elevator is much lower than that of the physical aircraft. The deflection and size of the elevator of the X-Plane model was verified to match the physical aircraft as well. This could also be a difference in the estimation of the propeller wash by X-Plane 11.33. The 11.50 update for X-Plane included a change to the modeling of control surfaces, but the update was released while experiments for this thesis were already underway.

In the response to the input for the physical aircraft, there are oscillations when the aircraft is in the process of pitching up and near level pitch. These were tracked down to the derivative gain which is to be expected as the tuning of the aircraft can continue to be tweaked to reach more desirable outputs. This tune would be adequate for moving on to tuning other control loops that wrap around the pitch attitude controller, but if acrobatic flight using this controller is desired then more tuning will be needed. In particular, Figure 45 shows that the D gain for the pitch controller is oscillating which could be solved by decreasing that gain slightly. For most flights, this level of handing qualities would be more than acceptable for most pilots. Figure 45 also shows that the P gain is behaving as expected and that the aircraft is well trimmed because of the small I gain.

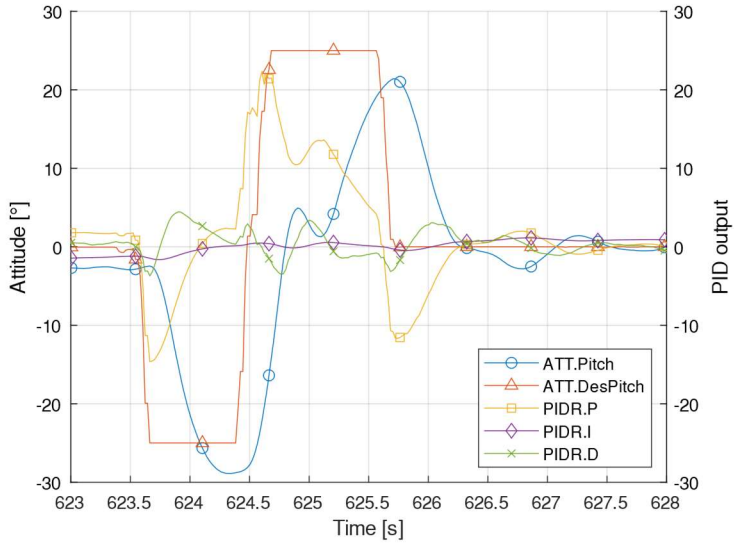


Figure 45: Example of PID break down for tuning

TEST POINT 4 (FBWA YAW GAIN TUNING)

In a very similar vein as test point 3, this test point is looking at the tuned response of the yaw attitude controller. Figure 46 shows the X-Plane response to several yaw doublets. The shape of the response is interesting for this case as it indicates greater differences with the inertia model in X-Plane as well as the

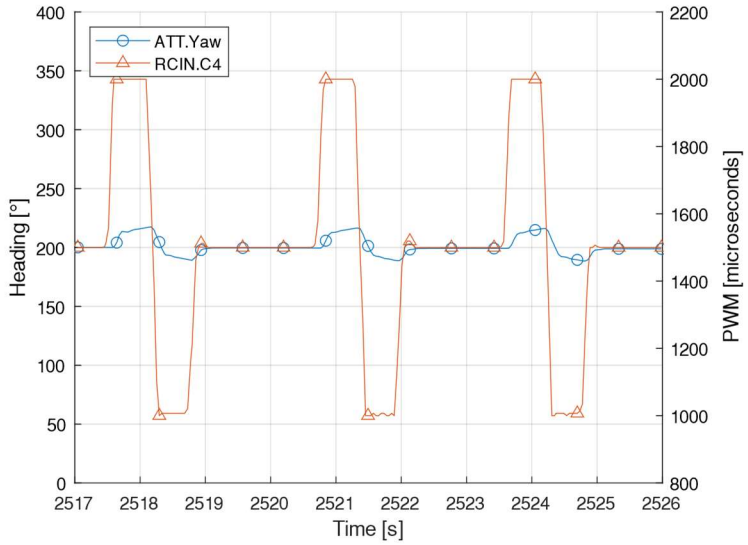


Figure 46: X-Plane test point 4

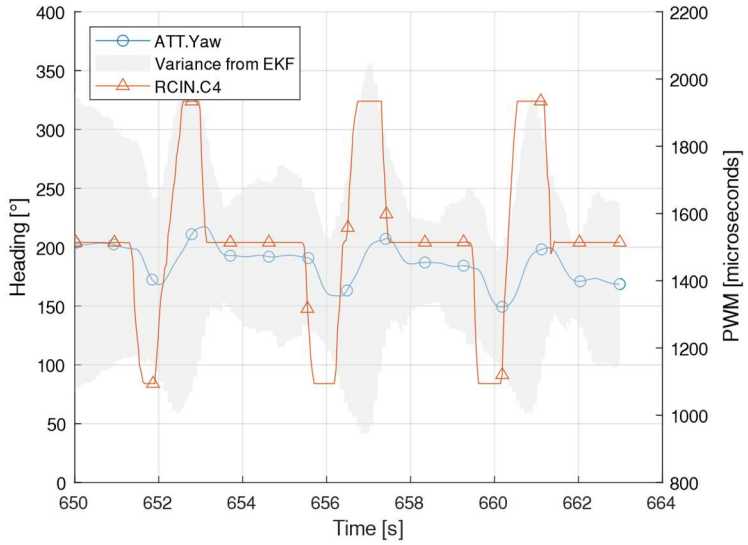


Figure 47: Physical test point 4

TEST POINT 5 (FBWA PITCH PREVIOUS GAIN TUNING)

This test point is the same as test point 3 with the only difference being a different set of gains that are listed in Table 8.

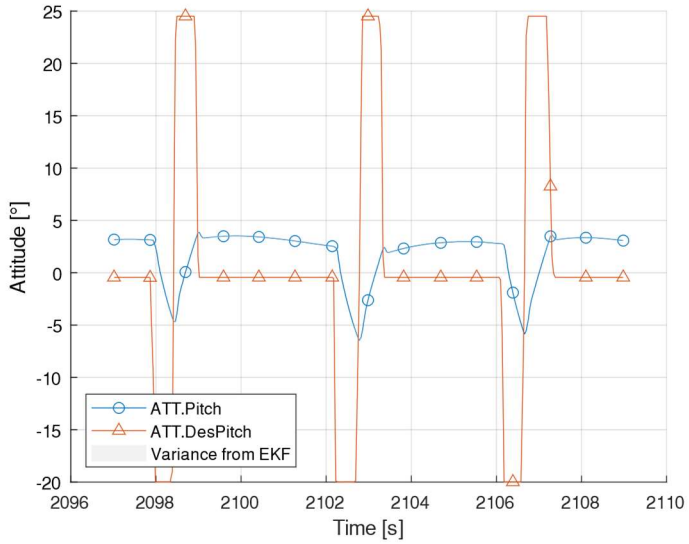


Figure 48: X-Plane test point 5

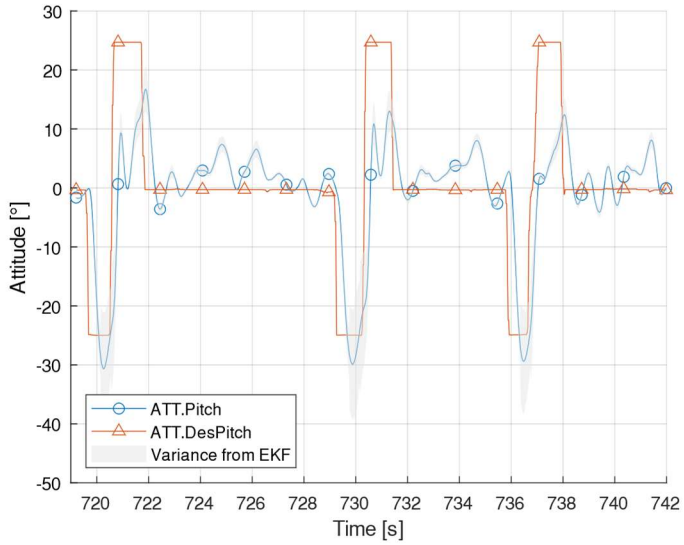


Figure 49: Physical test point 5

The results of this test point can be compared to the results shown in test point 3. This result shows a similar oscillation from the derivative controller in Figure 49, but also does a less than desirable job of reaching the positive pitch command. There is also excessive overshoot on the nose down pitch command that should be tuned out of the pitch controller. Comparing the results of this test point to the results of test point 3, especially from the X-Plane result in Figure 48, shows that the pitch rate of the aircraft is lower for this test point. This is expected as the P gain from parameter set 2 is lower than the gain used in parameter set 1.

TEST POINT 6 (FBWA YAW PREVIOUS GAIN TUNING)

This test point does not have much difference shown in the tuning of the yaw controller for a yaw doublet input. Differences in the gains for the yaw controller from parameter set 2 to parameter set 1 were due to targeting more coordination. Comparing Figure 50 and Figure 51 to their counter parts in test point 4 shows only small differences.

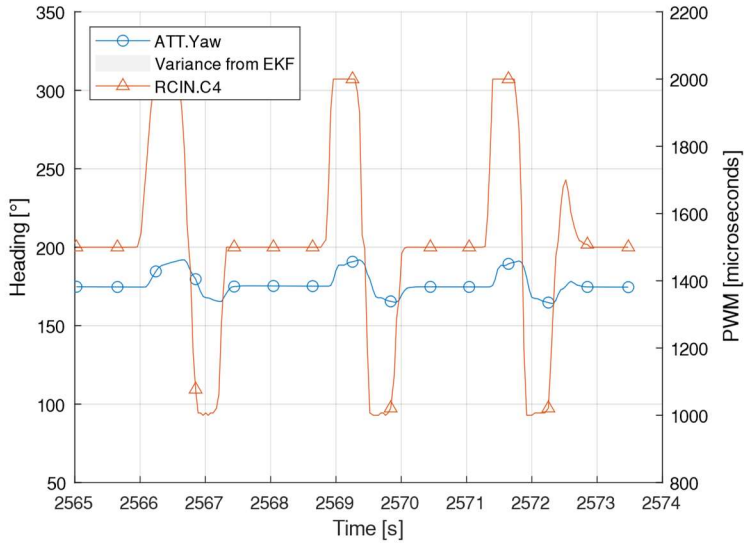


Figure 50: X-Plane test point 6

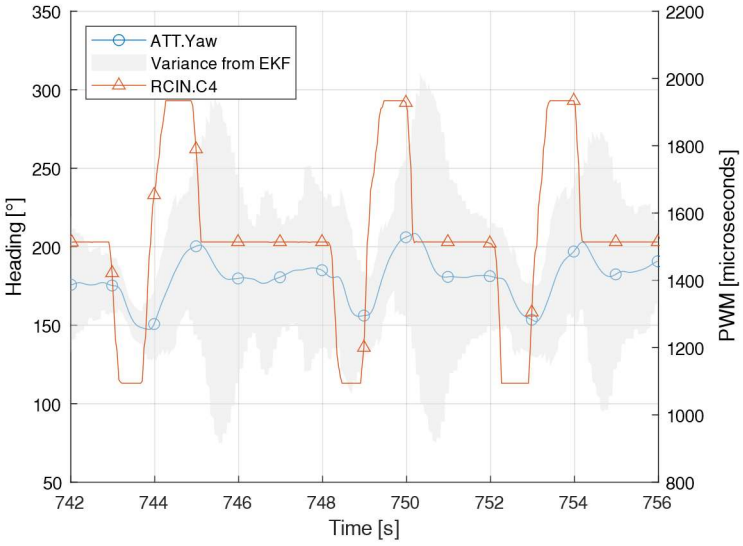


Figure 51: Physical test point 6

TEST POINT 7 (FBWA PITCH HIGH GAIN TUNING)

This test point and the following test point 8 were done with parameter set 3, listed in Table 9, as an attempt to show a different set of results that could come up while tuning a fixed wing aircraft. Figure 53 shows the need for the pitch D gain to be lowered. As has continued to be seen throughout the results, the affect of the X-Plane inertia model can be seen in Figure 52.

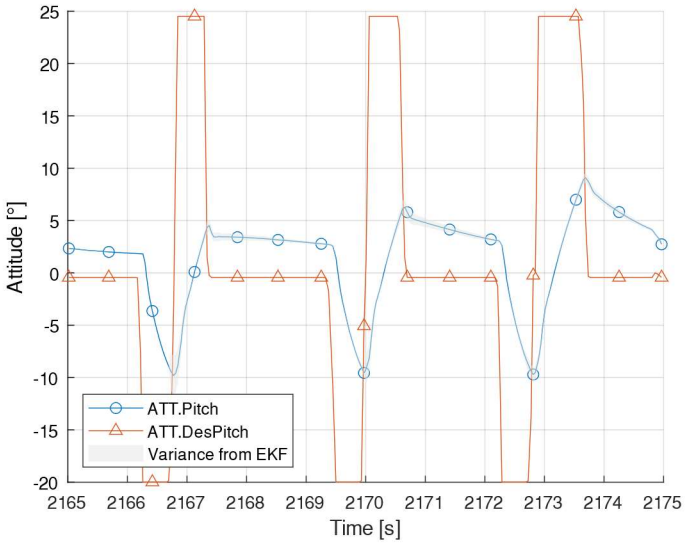


Figure 52: X-Plane test point 7

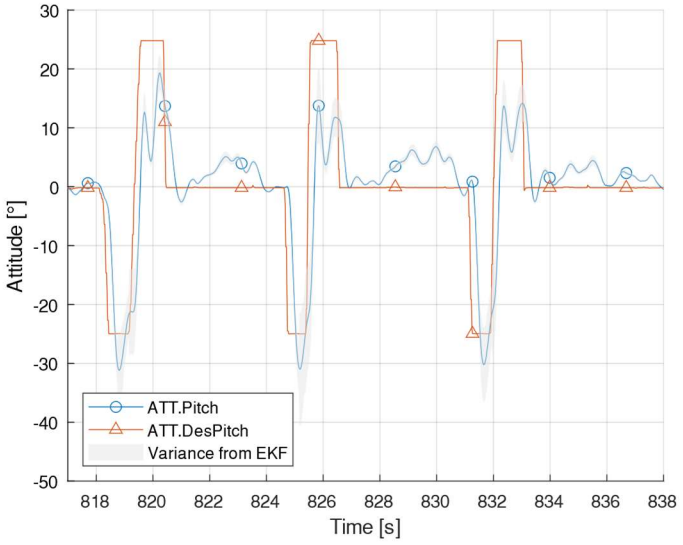


Figure 53: Physical test point 7

TEST POINT 8 (FBWA YAW HIGH GAIN TUNING)

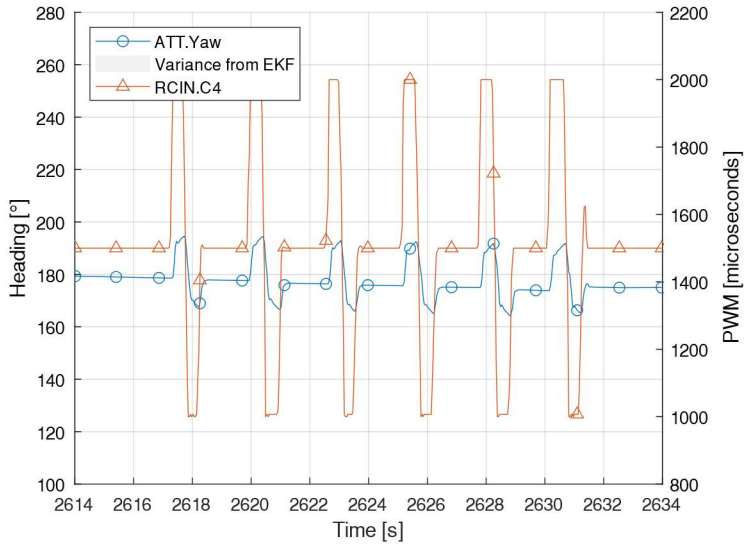


Figure 54: X-Plane test point 8

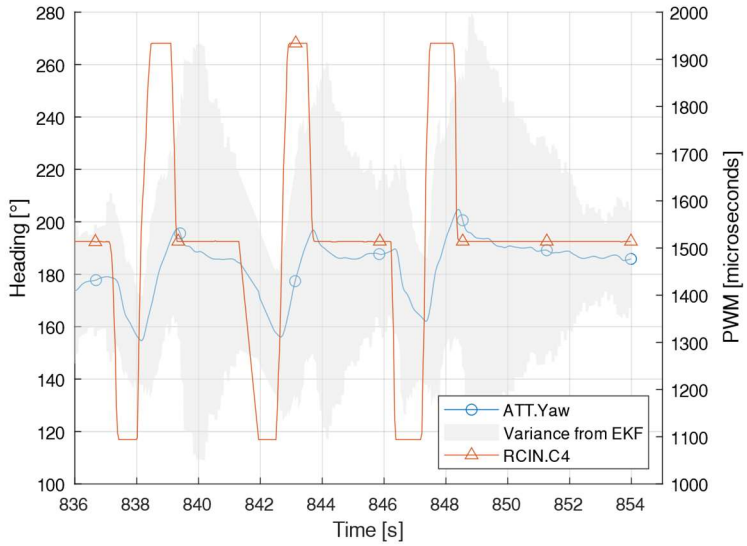


Figure 55: Physical test point 8

For this test point there is a small, but consistently present, oscillation present in the X-Plane result that is not present in the physical result. There is also a difference in the rate induced

by the control surface deflection. This might again be related to the differences in how flap forces are calculated by X-Plane.

TEST POINT 9 (FBWA PITCH INTEGRATOR)

This test point was done to try to show the impact that having too much integrator wind up has on reaching target attitudes. Figure 56 from X-Plane does not have much difference from the other test points. Figure 57 shows that there is not much impact for the rapid doublet that is the input signal. Choosing a different input signal for this test point or having a different test point with a different input signal might have been a better choice. This was inadvertently done at the end of the flight and will be shown after test point 10.

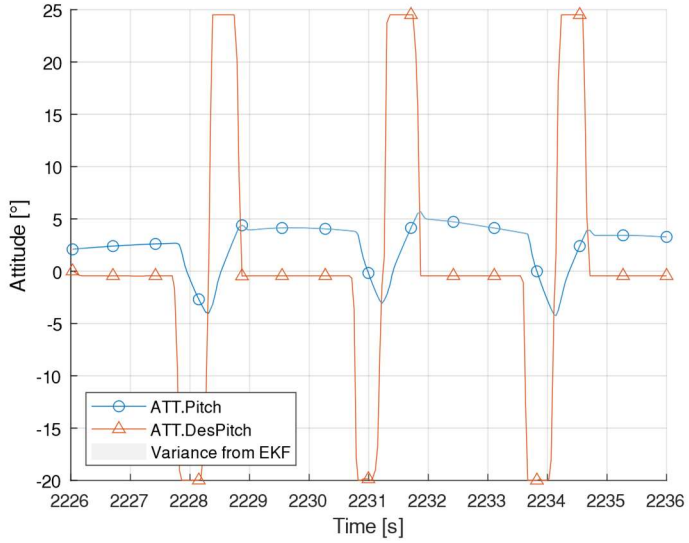


Figure 56: X-Plane test point 9

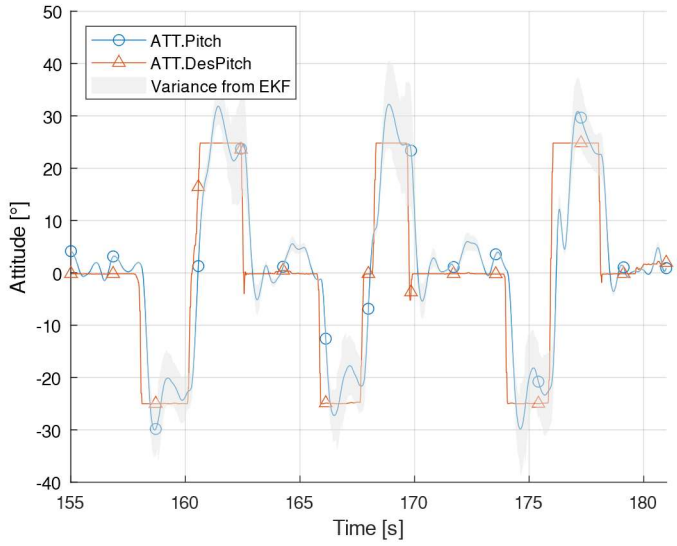


Figure 57: Physical test point 9

TEST POINT 10 (FBWB PITCH INTEGRATOR)

This test point was done in an attempt to show sluggish attitude tracking due to dominant control from the integrator. While that goal was not explicitly shown this test point does show that the aircraft is well trimmed and tuned to be used with the TECS. This sort of well-tuned behavior is exemplified in Figure 59 where the pilot is inputting doublets to the commanded climb and descent rates. This causes the TECS to choose a target attitude and throttle to achieve the command. This is also shown as an example of adding on additional control loops and tuning using those after the lower level loops are tuned. This stacked tuning is critical for more advanced, reliable, and trustworthy control of the aircraft in AUTO mode.

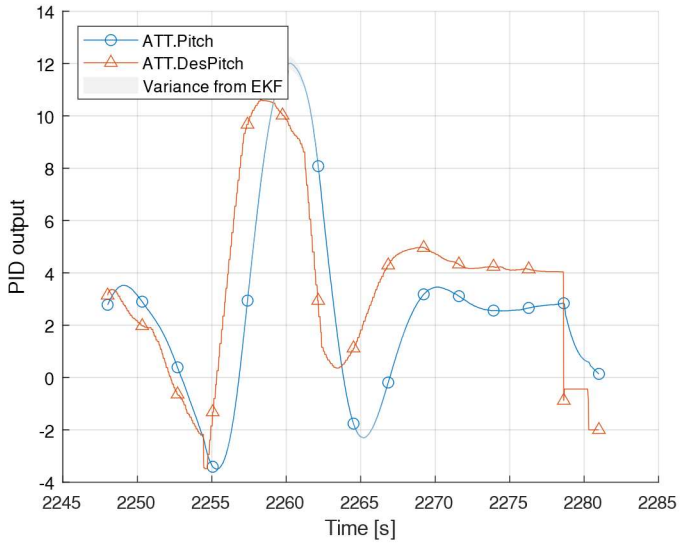


Figure 58: X-Plane test point 10

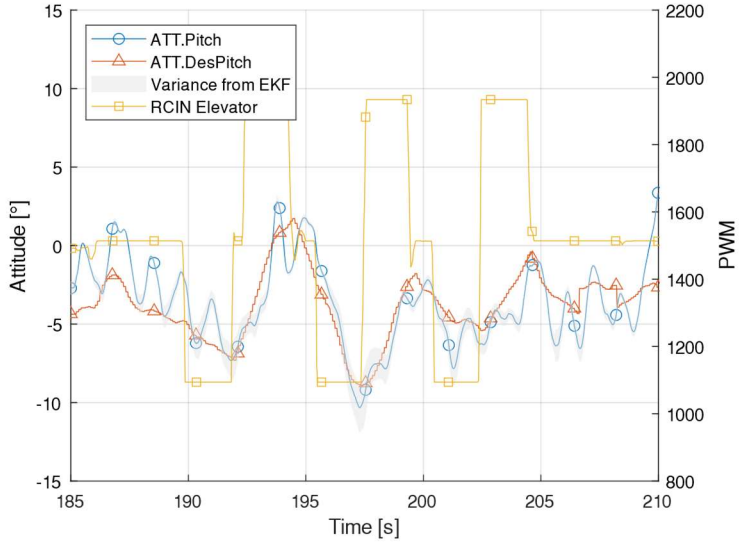


Figure 59: Physical test point 10

ADDITIONAL INTEGRATOR BUILD UP EXAMPLE

This test point was inadvertently done at the end of the flight and was not planned, but actually showed a key importance to limiting the integration of control surface controllers. In order to remain consistent with the presentation of other test points, the X-Plane simulator results is

shown before the physical test results. For this example though the physical results, Figure 61, occurred first and the X-Plane test, Figure 60, was done to try to replicate what a pilot would do in a similar scenario and to show where problems with how the aircraft is operated with default parameters might lead to undesirable results.

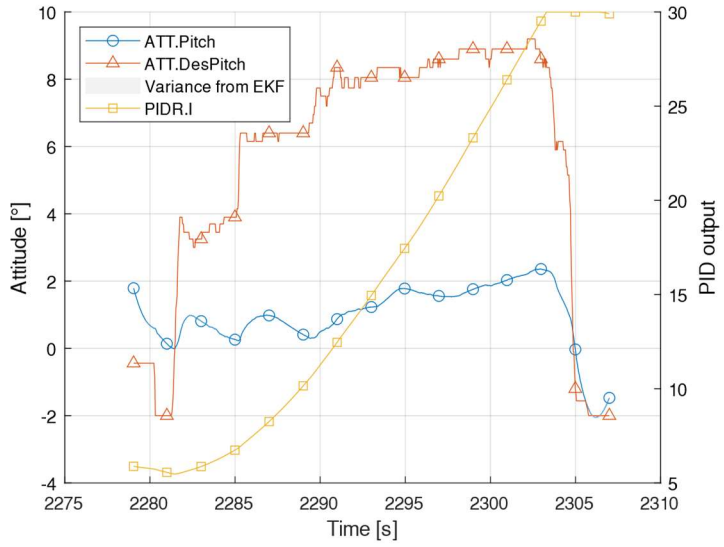


Figure 60: X-Plane replication of integrator build up

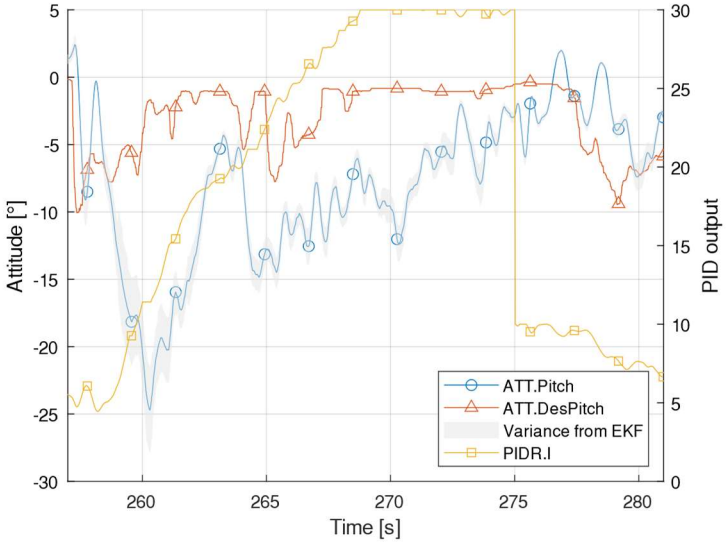


Figure 61: Physical instance of integrator build up

After the pilot completed each test point, they started setting up for a landing. Before the tuned set of parameters was able to be sent to the autopilot to make landing as easy as possible. The pilot dropped the throttle down in order to drop altitude to hit the standard downwind a turn base point. The autopilot attempted to hold the attitude of the aircraft level which lead to an increase in the pitch controller integrator. As previously shown in the parameters used for test point 10, the IMAX parameter for the pitch controller was set to the default value of 30. This would not be too big of a problem if the aircraft actually did need that much trim on the elevator in a long distance cruise, but for doing short and maneuverable flights this much integrator build up will only cause issues with allowing the proportional and derivative parts of the pitch controller from being expressed within the hard limits of the controller output. After the correct parameters were sent to the aircraft, the value of integrator immediately dropped to

A similar scenario was set up in X-Plane just to demonstrate that this scenario could be checked and tested before flying a physical aircraft. The X-Plane version showed a similar rapid increase in the pitch controller integrator that also hit the integrator limit of 30 out of 45. Had the pilot attempted to do a doublet or other rapid maneuver while the integrator was maxed out at 30, the other parts of the controller would not be strongly expressed, and the output attitude changes would be sluggish.

MODE IDENTIFICATION FROM ATTITUDE DATA

Previously described methods of flight data mode identification were used to extract information about the frequency and damping of the short period and Dutch roll where applicable. Data from multiple excitations of each mode were taken an averaged together to get the clearest picture of what the damping and response frequency were. The flight-testing data for the short period exhibited a deadbeat behavior. This was not entirely unexpected given the results from the analytical methods being only slightly underdamped and very close to being critically damped. This could be considered a desirable characteristic for a highly maneuverable, pylon racing aircraft

that frequently uses step inputs on the elevator when flying around a course. For those kinds of maneuvers any oscillation that is not quickly dampened out would require more work be done by the pilot to counter the oscillations and less predictability of the ending steady state pitch of the aircraft after a maneuver.

Method	Real	Imaginary	wn	zeta	wn%diff	zeta%diff
GeometricStabCon	-19.832	3.216	20.09	0.99	Baseline	
StabCon-D	-18.342	5.932	19.28	0.95	4.1	3.7
StabCon-D with propulsion	-18.345	5.992	19.30	0.95	4.0	3.8
X-Plane	-	-	Deadbeat		-	-
Flight Test	-	-	Deadbeat		-	-

Table 11: Comparisoin of short period mode

The Dutch roll results show that the analytical solutions are closer in agreement to each other. Comparing the analytical methods to the simulation and flight testing however, larger differences in the natural frequency can be seen.

Method	Real	Imaginary	wn	zeta	wn%diff	zeta%diff
GeometricStabCon	-8.75	18.23	20.22	0.43	Baseline	
StabCon-D	-13.90	22.82	26.72	0.52	27.7	18.3
StabCon-D with propulsion	-13.90	22.85	26.75	0.52	27.8	18.3
X-Plane	-30.14	30.75	43.05	0.7	72.2	47.2
Flight Test	-12	29.971	32.28	0.37	38.9	15.2

Table 12: Comparison of Dutch roll mode

CASE STUDY OF LOCUST AIRCRAFT

A case study of detailing the importance of SITL testing using the aircraft that Caster wrote his thesis about will be made [42]. This aircraft includes a turbojet mounted vertically to augment lift for Very Short Takeoff and Landing (VSTOL). The aircraft has successfully flown many times, but some aspects of the ArduPlane control structure were part of the design process. The aircraft was flown in AUTOTUNE mode to get initial parameters for the ArduPlane PID controllers. These parameters were tuned slightly after successive flights to improve the handling qualities of the aircraft. SITL was not used with this aircraft prior this thesis. An X-Plane model did exist but was

only flown in what ArduPilot considers MANUAL mode as there was no autopilot involved in the control of the aircraft.

There had been a few flights that had resulted in crashes prior to the one that will be explored in detail. The cause of these crashes seemed to indicate power failures and having the center of gravity (CG) in the wrong location. These faults were corrected, and the program continued with several more successful flights. During pre-flight checks of the aircraft one day, a member of the flight team noted that the ailerons were deflecting nearly three quarters of the max deflecting when the aircraft was bumped slightly. The concern was brought up with the other flight team members and a course of action was chosen to correct for the apparent fault. Table 13 shows the changes to the parameters that were done to correct the observed behavior. No other parameter changes were made.

Parameter	Before	After
PTCH2SRV_P	1.6716	0.8716
RLL2SRV_P	0.807	0.257

Table 13: Parameter changes before Locust flight

The aircraft took off normally and entered a bank while lowering the throttle on the turbojet. The pilot in command (PIC) then noted that the aircraft was not coming out of the bank while even when commanding an opposite bank command. The standard operating procedure (SOP) for this aircraft stated to fly the aircraft in Fly-By-Wire-A (FBWA). There was no mention of changing modes if something with the controls was not correct during flight. The aircraft remained in the bank and overshot the bank limit leading to a spiral mode. The aircraft was also unresponsive in the pitch axis. In an attempt to gain lift from the wings the throttle of the propeller was increased. The aircraft crashed into the ground at a reported airspeed of 90 knots. The dataflash log ends on impact with the ground.

After analyzing the dataflash log, a full learning process for the entire flight team was started to better understand the control structure of ArduPlane. Focus was given to the scaling of

the PID gains based on airspeed and to the PID control structure that ArduPilot uses. The fault of the crash was a runaway integrator due to the proportional gain being too low. The SOP was also found to be at fault as changing to MANUAL mode would have increased the chances that the PIC would be able to return the aircraft to a safe operating condition.

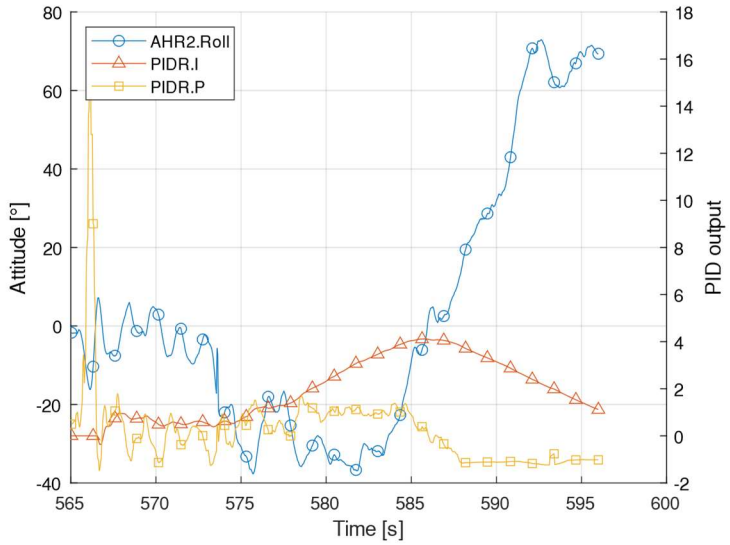


Figure 62: Flight log of Locust crash

To correct the faults, changes to the gains were reverted and tuned again using the method described previously. The parameters for the maximum limit of the integrator were also changed. For example, the default value for PTCH2SRV_IMAX is 3000 centidegrees. This translates to 30° out of 45° of travel. Internally, ArduPilot considers 45° to be equal to the maximum deflection of the servo and 0° equal to the trim position. Therefore, the default is for 66% of the deflection of the servo can be used to trim the aircraft. Changing the IMAX parameters to a more reasonable 20% deflection of the servo prevents the integrator from running away with a large trim value.

To demonstrate the impact that the parameter change had, Locust was recreated in X-Plane 11 and flown in SITL with ArduPlane version 3.9.4. The throttle of the vertical lift jet was controlled manually and separately from the front propeller. To better demonstrate the impact of the moment produced by the thrust from the turbojet, the jet was positioned 1 inch off the center

line of the aircraft. This has a massive impact on integral control. The model was flown in steady, level, unaccelerated flight (SLUF) in FBWA with the turbojet on at maximum thrust and an airspeed of 70 kts. The turbojet throttle was then dropped to minimum and the commanded pitch and roll angles kept at zero. The same parameters were used as the day of the crash. Figure 63 shows that the roll control has a large amount of overshoot and is very underdamped.

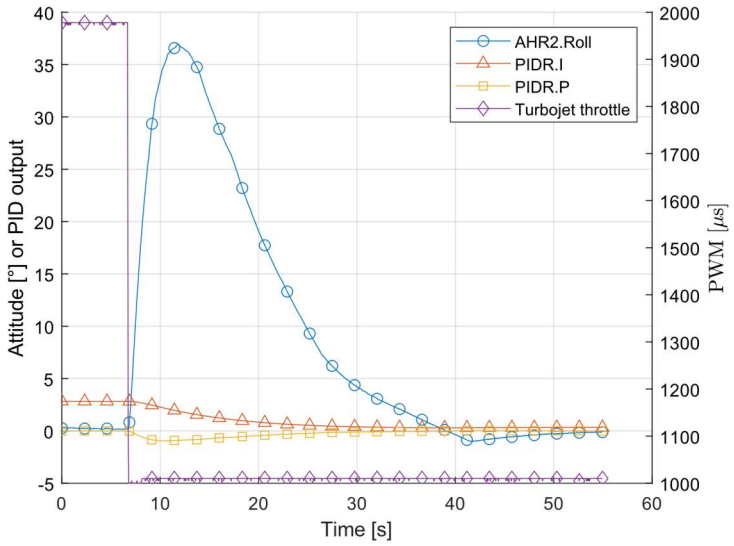


Figure 63: Testing Locust crash parameters in X-Plane with ArduPilot SITL

Another SITL test was done using the parameters from before the change and without integrator control. Figure 64 shows that the system would have a much more desirable response with the higher proportional controller value and does not change the trim of the aircraft. There is a steady state error in the roll angle due to other moments on the aircraft such as the propeller because there is no trimming being done by the autopilot towards a roll angle of zero with the ailerons applying a counter moment.

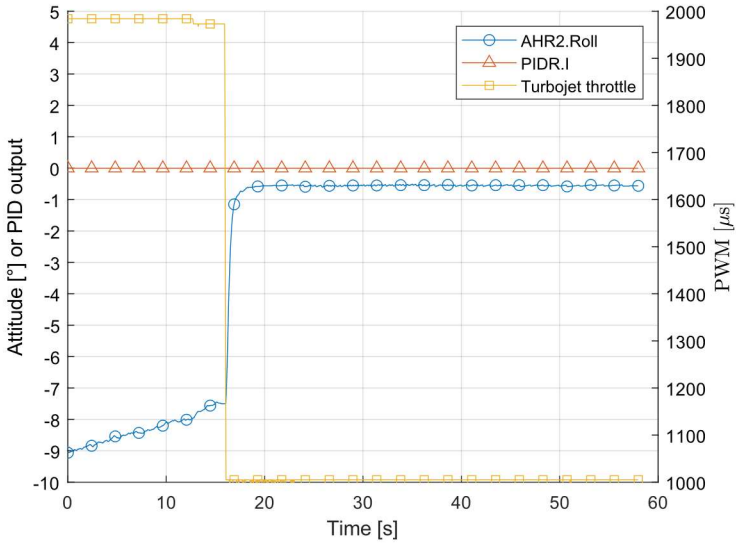


Figure 64: SITL testing pre-change Locust parameters

CHAPTER IV

CONCLUSIONS

VARIATIONS IN MODE ESTIMATION METHODS

While looking through the many files that X-Plane creates after running any aircraft, it was found that several useful and interesting pieces of information were near the bottom of the "Cycle Dump.txt" file. They include the radius of gyrations used for the aircraft that was loaded as well as the static margin. Neither of these values were close to the values they should be after creating the model in X-Plane. The radius of gyration error is easy enough to fix since that can be input for the model in the Plane Maker program. However, the static margin value is a very different case. Through a series of iterations, the position of the CG that achieved the correct static margin value in the output file was behind the trailing edge of the mean aerodynamic chord (MAC). This configuration still flies in X-Plane, but with a much more sensitive control as should be expected from decreasing the static margin. The aircraft handles closer to what is expected in X-Plane with the CG in the position specified by the manufacturer of the physical Edge 540 model. A comparison of calculated static margins of the FMS Edge 540 can be seen in Table 14.

The same issue could be seen with the model of the Cessna 172 Skyhawk that ships with X-Plane. For the 172, the output file reported a static margin of 59% MAC while the actual value of the static margin is near 20% MAC. This leads to the conclusion that there most likely a different method is being used to calculate the static margin output value in X-Plane and is this is not a valid output measure to rely upon.

Method	Static Margin [MAC]
GeometricStabCon	0.09
StabCon-S	0.03
StabCon-S with propulsion	0.02
StabCon-D	0.12
StabCon-D with propulsion	0.12
CFD	0.14
X-Plane	0.61

Table 14: Static margin reported by each method

The difference in the reported moments of inertia are more of a curious note as all methods used for estimation allow for the radius of gyration or moment of inertia to be input to replace the approximated values. Each of assumed radius of gyration values is reported in Table 15. StabCon-S and StabCon-D use the same method to approximate the moments of inertia

Method	Radius of Gyration [ft]		
	X	Y	Z
StabCon-S/D	0.87	0.99	2.75
X-Plane	0.52	1.08	1.18

Table 15: Radius of gyration assumed by each method

With regard to using SOLIDWORKS Flow Simulation for determining dynamic stability, there is currently no way to repeat the processes that Babcock and O'Neil used in STARS for dynamic parameter estimation and therefore aircraft mode estimation. Verification of static parameters is possible and has been demonstrated through static margin, coefficient of lift, and coefficient of moment. This agrees with previous work done by Babcock and O'Neil.

EFFICACY OF SITL AND FDM WITH REGARD TO RISK MITIGATION

As can be seen in tables for the short period and Dutch roll modes, there are strong indicators where different estimation methods are very accurate and others where the estimations are not as accurate. While GeometricStabCon is well within the ballpark for short period, there is

a is a more significant difference in the estimation of Dutch roll. An explanation of why this might be the case for Dutch roll is not within the scope of this thesis.

For the short period of this aircraft, there were two good indicators that the response would be nearly critically damped. The predicted damping behavior was clearly seen in both X-Plane and flight testing. For Dutch roll, X-Plane is nearly dead on for the frequency and damping ratio shown in flight testing. This is a good indication of the accuracy of the model created in X-Plane as well as the accuracy of the lateral model used by the simulator.

Using this data of aircraft flight modes, it can be determined that X-Plane is only somewhat valid as an FDM for sUAS aerodynamic mode identification. This is not comprehensive of all aircraft configurations or modes and X-Plane or other FDM software could be shown to be very accurate at predicting other sUAS flight modes. It is merely stating that the lowest level of controllers for roll, pitch, and yaw can be tuned to acceptable handling qualities prior to any real aircraft taking to the skies. The major failings that can be seen in the results is the differences in the rate that the control surface deflections on the aircraft. This could be due to differences in the way that X-Plane calculates the forces of flap deflection, propeller wash, and inertia. A few of the major benefits of using virtual testing of aircraft with ArduPilot SITL are shown in the results to be prevention of excessive integrator build up, reducing possible overshoot, and prediction of risky flight conditions.

The SITL portion of ArduPilot runs with nearly the same code as a hardware autopilot. The biggest difference is the approximation form the EKF. This did not have a significant nor noticeable impact on the behavior of the autopilot during the portions of the flight that were under the control of the autopilot. Flight dynamics wise, using SITL does provide the opportunity to catch problems with the parameters and might have prevented the crash of the Locust aircraft. Parameter adjustments relevant to the failsafe actions or run-away integration could be effectively simulated by using SITL. It is recommended to at least fly the aircraft in a simulator for the purpose of rough estimation of PID gains and testing changes to behavior parameters. Using a suite of tools and

methods, such as those used here, to predict aircraft behavior can lead to better understanding of where the unexpected might happen.

Having SITL available for testing and debugging Lua scripts was invaluable during the work of this thesis. Most functionality available to scripting could be tested with visual results in the FDM or GCS. As stated previously, with X-Plane these visual indications could be extended with the utilizations of built-in or custom datarefs. Of note, using SITL to test Lua scripts does not currently allow for testing memory usage of the script. A script that would overflow the hardware memory assigned to scripting would not overflow the memory space in SITL. Any script that is used in flight must be carefully debugged and each code path tested for runtime errors.

For the case of Locust there are many places where risks could have been mitigated. Both methods of risk mitigation discussed in this thesis, knowledge and simulation, might have played significant roles in preventing the Locust crash. Showing what would have been found through using SITL after the fact demonstrates that there are methods of discovering problematic parameters without placing an expensive aircraft in harm's way. Practicing risk mitigation in every aspect of the design process has been proven time and time again to create safer products that everyone involved with the project can feel more confident in.

SERVO OVERRIDE IMPACT ON CONTROLLERS

Since the control loop calculations are paused in modes where the controllers are not used, it is the author's recommendation to use modes that will not continue calculating an overridden control surface. For this work, MANUAL mode was used as all controls were overridden to be kept at trim or constant positions while the doublet was performed. While the autopilot is in MANUAL mode, PID calculations are not performed. In some cases, switching in or out of a mode that uses a control loop can reset some integrators to zero. These cases are more applicable to quadplane modes where integration build up could cause undesirable behavior when starting or stopping transitions between hovering and forward flight.

As an additional example, if a new version of altitude control wanted to be tested using the servo override method, putting the aircraft into STABILIZE or FBWA would reduce the chances of causing the elevator integrator to ramp up during the time that the

2D COMPUTATIONAL FLUID DYNAMICS IN SOLIDWORKS

While the 3D aircraft CFD data has strong correlations to theoretical data, the 2D CFD does not show good convergence of data. Wallace's recommendations for more computing resources to be applied to the low Reynolds number 2D simulation in SOLIDWOKRS Flow Simulation were explored. An NACA 2412 airfoil was used with the same solver parameters that Wallace used. The simulation reached 3 million cells, but the same issues which Wallace observed, namely oscillations and non-converged solutions, continued. The cell size was well below the expected minimum y^+ value for a $k-\epsilon$ solver to resolve the boundary layer. At this point it can be hypothesized that the $k-\epsilon$ solver used by SOLIDWORKS Flow Simulation cannot provide ideal answers for nuanced analysis of airfoils and no amount of additional computation resources used will help to converge the solution. This hypothesis could change in the future if different boundary layer estimation techniques are introduced to the solver.

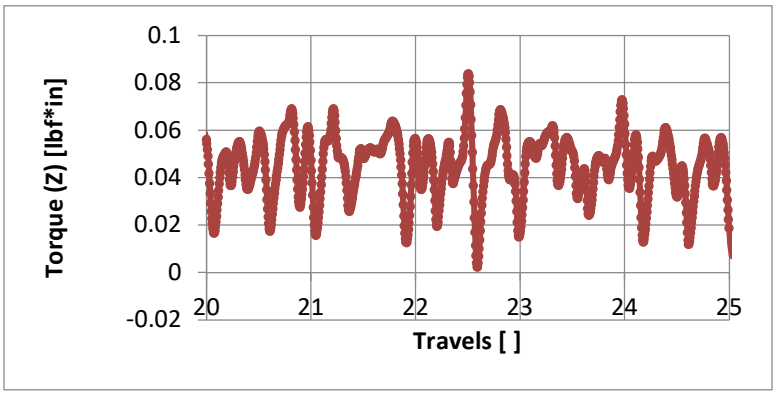


Figure 65: Oscillations in CFD solution

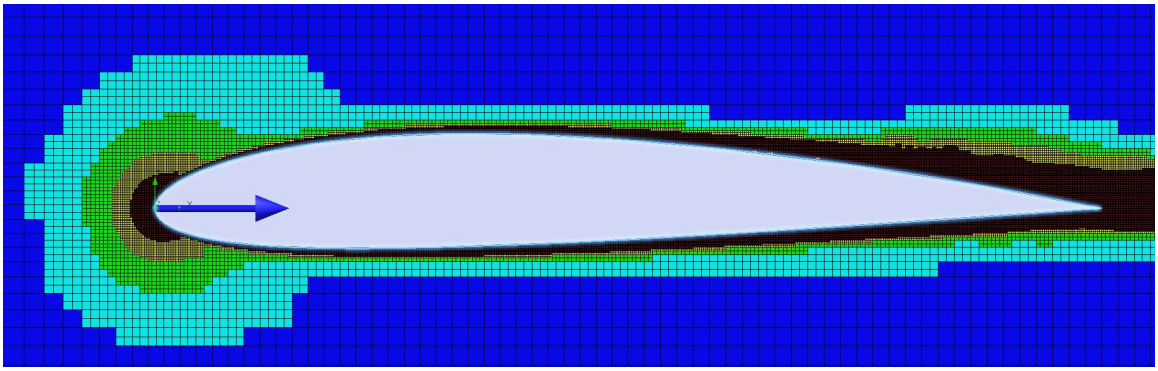


Figure 66: 2D slice of mesh for NACA 2412

SUMMARY OF SOFTWARE CREATED

This is a summary of the pieces of software that have been created or modified for this thesis. All of these are freely available on GitHub and have the most important pieces included with this thesis.

The first piece of software that was created was a modification to ArduPilot. This was created with a lot of helpful assistance from Dr. Andrew Tridgell. This allowed for any servo output channel to be overridden with a specified PWM for a fixed amount of time in milliseconds. This also added a bug that was caught by Peter Hall where the emergency stop for the overridden channel no longer worked. After this was quickly fixed and the feature has since been shown being used by several others using ArduPilot for other projects.

For their Google Summer of Code 2020 project, Peter Hall created a JavaScript Object Notation (JSON) interface for ArduPilot SITL. This allowed for an external simulator to receive the servo outputs that ArduPilot provided and for the simulator to send back sensor data. Hall created examples using MATLAB and Python. These were very helpful in creating a new custom simulator for validating the ArduPilot attitude controllers. This new simulator could be easily modified to add more validation points for other controllers in ArduPilot.

The last piece of software that was created for this thesis was the doublet control input script. This was created to perform a consistent doublet maneuver safely and quickly. Additional work could be done to create any signal desired as a control input.

FUTURE WORK

Further development of the control signal input structure used in this thesis could be used to create a SYSID mode similar to one used in ArduCopter for ArduPlane. This could allow a next generation of autotune methods to come about for ArduPlane.

Further investigation could be done with other FDMs using the same aircraft and resources created by this thesis. This would provide a better picture for where some FDMs provide a higher fidelity model for sUAS fixed wing aircraft.

ArduPilot does include a procedure to tune the lateral navigation controller (L1) and the total energy control system (TECS), but it does not have the same level of performance-based tuning that quadcopters in the ArduCopter documentation. Having a set of accurate models of a variety of aircraft could provide that level of tuning guidance for ArduPlane without a large monetary investment in aircraft or materials.

Considering the continued work with SOLIDWORKS Flow Simulation, a different flow solver might be worth exploring assuming that a simple workflow to quickly go from a SOLIDWORKS model to a mesh can be achieved. This could provide more accuracy in the early stages of development when flow interactions might play a significant role in the stability of the aircraft.

REFERENCES

[1] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith and M. Khalgui, "Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey," *IEEE Access, 2019,* 22 6 2019.

[2] S. Arnold, "Towards an Open Instrumentation Platform: Getting the Most From MAVLink, ArduPilot, and BeagleBone," 2017.

[3] S. Hood, "Development of a Flight Data Acquisition System for Small Unmanned Aircraft," 2014.

[4] A. Tridgell, R. Mackay, P. Barker, L. D. Marchi, WickedShell, P. Riseborough, T. Pittenger, Jason4short, P. Kancir, Jschall, L. Hall, G. J. D. Sousa, M. Oborne, J. Walser, P. Hickey, A. Lucas, S. B. Purohit, R. Lefebvre, k. murata, F. Ferreira, P. Hall, C. M. D. O. Filho, G. Staroselskii, G. Morphett, A. Piper, M. Denecke, M. Whitehorn, P. J. Pereira, E. Shamaev and J. BERAUD, *ArduPilot master,* Github, 2020.

[5] H. Chao, Y. Cao and Y. Chen, "Autopilots for small unmanned aerial vehicles: A survey," *International Journal of Control, Automation and Systems,* vol. 8, p. 36–44, 2 2010.

[6] A. Tridgell, *ArduPilot documentation,* https://ardupilot.org/ardupilot/, 2020.

[7] A. Tridgell, R. Mackay, P. Barker, L. D. Marchi, WickedShell, P. Riseborough, T. Pittenger, Jason4short, P. Kancir, Jschall, L. Hall, G. J. D. Sousa, M. Oborne, J. Walser, P. Hickey, A. Lucas, S. B. Purohit, R. Lefebvre, k. murata, F. Ferreira, P. Hall, C. M. D. O. Filho, G. Staroselskii, G. Morphett, A. Piper, M. Denecke, M. Whitehorn, P. J. Pereira, E. Shamaev, J. BERAUD and C. Johnson, *ArduPilot fork,* Github, 2020.

[8] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen and U. P. Schultz, "A Survey of Open-source UAV Flight Controllers and Flight Simulators," *Microprocessors and Microsystems,* vol. 61, p. 11–20, 9 2018.

[9] *ArduPilot forums,* https://discuss.ardupilot.org/, 2020.

[10] J. Pattison and S. Dade, *Managing safety notices,* 2020.

[11] R. Beard, Small Unmanned Aircraft : Theory and Practice, Princeton: Princeton University Press, 2012.

[12] R. C. Nelson, Flight Stability and Automatic Control, McGraw-Hill Education - Europe, 1997.

[13] B. Stevens, Aircraft control and simulation : dynamics, controls design, and autonomous systems, Hoboken, New: Wiley, 2015.

[14] McCormick, Aerodynamics, Aeronautics, and Flight Mechanics, John Wiley & Sons, 1994.

[15] D. E. Hoak and R. D. Finck, "USAF Stability and Control DATCOM," 1978.

[16] D. A. Babcock, "Aircraft Stability Derivative Estimation from Finite Element Analysis," 2002.

[17] R. C. Seamans, B. P. Blasingame and G. C. Clementson, "The Pulse Method for the Determination of Aircraft Dynamic Performance," *Journal of the Aeronautical Sciences,* vol. 17, p. 22–38, 1 1950.

[18] C. R. O'Neill, "Higher Order and Dynamic CFD for Aeroelastic Simulations," 2011.

[19] A. Dorobantu, A. Murch, B. Mettler and G. Balas, "System Identification for Small, Low-Cost, Fixed-Wing Unmanned Aircraft," *Journal of Aircraft,* vol. 50, p. 1117–1130, 7 2013.

[20] "Parameter estimation techniques and application in aircraft flight testing," 1974.

[21] R. Kimberlin, Flight testing of fixed-wing aircraft, Reston, VA: American Institute of Aeronautics and Astronautics, 2003.

[22] M. J. Casiano, "Extracting damping ratio from dynamic data and numerical solutions," 2016.

[23] G. Zogopoulos-Papaliakos, *https://github.com/Georacer/last_letter,* 2019.

[24] J. S. Berndt, *https://sourceforge.net/projects/jsbsim/,* 2020.

[25] L. T. Nguyen, W. N. Tun, N. V. A. N. Nguyen, M. Tyan, S. Kim and J.-W. Lee, "Evaluation on X-Plane Simulator Using Flight Test Data of Light Aircraft KLA-100 Scaled Model," in *The Korean Society for Aeronautical & Space Sciences*, 2016.

[26] A. Kamal, A. M. Aly and A. Elshabka, "Modeling, Analysis and Validation of a Small Airplane Flight Dynamics," in *AIAA Modeling and Simulation Technologies Conference*, 2015.

[27] A. Dorobantu, P. J. Seiler and G. J. Balas, "Validating Uncertain Aircraft Simulation Models Using Flight Test Data," in *AIAA Atmospheric Flight Mechanics (AFM) Conference*, 2013.

[28] D. A. V. Ivanov, T. V. Trebunskikh and V. V. Platonovich, "Validation Methodology for Modern CAD-Embedded CFD Code: from Fundamental Tests to Industrial Benchmarks," 2017.

[29] S. Corporation, "Technical Reference: SOLIDWORKS Flow Simulation," 2019.

[30] J. S. Wallace, "Investigation of Solidworks Flow Simulation As a Valid Tool for Analyzing Airfoil Performance Characteristics in Low Reynolds Number Flows," 2019.

[31] F. White, Fluid mechanics, 5 ed., New York: McGraw-Hill, 1979.

[32] J. G. Coder, "Further Development of the Amplification Factor Transport Transition Model for Aerodynamic Flows," in *AIAA Scitech 2019 Forum*, 2019.

[33] P. E. Klopsteg, *THE BIFILAR PENDULUM,* vol. 1, AIP Publishing, 1930, p. 3–8.

[34] A. Tridgell, *OpenDog demo with ArduPilot and Lua control,* 2020.

[35] R. S. Figliola and D. E. Beasley, Theory and Design for Mechanical Measurements, John Wiley & Sons Inc, 2014.

[36] S. J. Kline, "The Purposes of Uncertainty Analysis," *Journal of Fluids Engineering,* vol. 107, no. 2, pp. 153-160, 1985.

[37] O. J. Woodman, "An introduction to inertial navigation," 2007.

[38] N. El-Sheimy, H. Hou and X. Niu, "Analysis and Modeling of Inertial Sensors Using Allan Variance," *IEEE Transactions on Instrumentation and Measurement,* vol. 57, p. 140–149, 1 2008.

[39] R. Gonzalez and P. Dabove, "Performance Assessment of an Ultra Low-Cost Inertial Measurement Unit for Ground Vehicle Navigation," *Sensors,* vol. 19, p. 3865, 9 2019.

[40] R. Gonzalez, J. I. Giribet and H. D. Patino, "NaveGo: a simulation framework for low-cost integrated navigation systems," *Journal of Control Engineering and Applied Informatics,* vol. 17, p. 110–120, 2015.

[41] S. Eichstädt, N. Makarava and C. Elster, "On the evaluation of uncertainties for state estimation with the Kalman filter," 4 5 2016.

[42] G. O. Castor, "Design and Development of a Controllable Wing Loading Unmanned Aerial System," 2017.

APPENDICIES

Doublet Lua Script

```
-- This script will perform a control surface doublet
-- Charles Johnson, OSU 2020

local DOUBLET_ACTION_CHANNEL = 9 -- RCIN channel to start a doublet when high (>1700)
local DOUBLET_CHOICE_CHANNEL = 10 -- RCIN channel to choose elevator (low) or rudder (high)
local DOUBLET_FUCNTION = 19 -- which control surface (SERVOx_FUNCTION) number will have a doublet happen
-- A (Servo 1, Function 4), E (Servo 2, Function 19), and R (Servo 4, Function 21)
local DOUBLET_MAGNITUDE = 6 -- defined out of 45 deg used for set_output_scaled
local DOUBLET_TIME = 500 -- period of doublet signal in ms


-- flight mode numbers for plane https://mavlink.io/en/messages/ardupilotmega.html
local MODE_MANUAL = 0
local MODE_FBWA = 5
local MODE_FBWB = 6
local MODE_RTL = 11
local K_AILERON = 4
local K_ELEVATOR = 19
local K_THROTTLE = 70
local K_RUDDER = 21

-- store the info between callbacks
-- set at the start of each doublet
local start_time = -1
local end_time = -1
local now = -1

-- store information about the vehicle
local doublet_srv_chan = SRV_Channels:find_channel(DOUBLET_FUCNTION)
local doublet_srv_min = param:get("SERVO" .. doublet_srv_chan + 1 .. "_MIN")
```

```
local doublet_srv_max = param:get("SERVO" .. doublet_srv_chan + 1 ..
"_MAX")
local doublet_srv_trim = param:get("SERVO" .. doublet_srv_chan + 1 ..
"_TRIM")
local pre_doublet_mode = vehicle:get_mode()
local current_hagl = ahrs:get_hagl()

function retry_set_mode(mode)
    if vehicle:set_mode(mode) then
        -- if the mode was set successfully, carry on as normal
        return doublet, 1
    else
        -- if the mode was not set successfully, try again ASAP
        return retry_set_mode, 1
    end
end

function doublet()
    local callback_time = 100
    if arming:is_armed() == true and rc:get_pwm(DOUBLET_ACTION_CHANNEL)
> 1700 and end_time ==-1 then
        callback_time = DOUBLET_TIME / 10
        -- start a quick doublet based on some math/logic
        now = millis()
        if start_time == -1 then
            -- this is the time that we started
            start_time = now
            -- notify the gcs that we are starting a doublet
            gcs:send_text(6, "STARTING DOUBLET " .. DOUBLET_FUCNTION)
            -- get info about the doublet channel
            doublet_srv_chan                                        =
SRV_Channels:find_channel(DOUBLET_FUCNTION)
            doublet_srv_min = param:get("SERVO" .. doublet_srv_chan + 1
.. "_MIN")
            doublet_srv_max = param:get("SERVO" .. doublet_srv_chan + 1
.. "_MAX")
            doublet_srv_trim = param:get("SERVO" .. doublet_srv_chan + 1
.. "_TRIM")
            pre_doublet_mode = vehicle:get_mode()
            -- are we doing a doublet on elevator or rudder? set the
other controls to trim
            local                  doublet_choice_pwm                =
rc:get_pwm(DOUBLET_CHOICE_CHANNEL)
            local trim_funcs = {}
            local                pre_doublet_elevator                =
SRV_Channels:get_output_pwm(K_ELEVATOR)
            if doublet_choice_pwm < 1500 then
                -- doublet on elevator
                DOUBLET_FUCNTION = K_ELEVATOR
                trim_funcs = {K_AILERON, K_RUDDER}
```

```
                DOUBLET_MAGNITUDE = 12
                DOUBLET_TIME = 500
                doublet_srv_trim = pre_doublet_elevator
            else
                -- doublet on rudder
                DOUBLET_FUCNTION = K_RUDDER
                trim_funcs = {K_AILERON}
                DOUBLET_MAGNITUDE = 15
                DOUBLET_TIME = 500
                -- pin elevator to current position. This is most likely
different than the _TRIM value

SRV_Channels:set_output_pwm_chan_timeout(SRV_Channels:find_channel(K_EL
EVATOR), pre_doublet_elevator, DOUBLET_TIME * 4)
            end
            -- set the channels that need to be still to trim until the
doublet is done
            for i = 1, #trim_funcs do
                local                    trim_chan                    =
SRV_Channels:find_channel(trim_funcs[i])
                local trim_pwm = param:get("SERVO" .. trim_chan + 1 ..
"_TRIM")
                SRV_Channels:set_output_pwm_chan_timeout(trim_chan,
trim_pwm, DOUBLET_TIME * 2)
            end
            -- get the current throttle PWM and pin it there until the
doublet is done
            local                pre_doublet_throttle                =
SRV_Channels:get_output_pwm(K_THROTTLE)
            SRV_Channels:set_output_pwm_chan_timeout(
                SRV_Channels:find_channel(K_THROTTLE),
                pre_doublet_throttle,
                DOUBLET_TIME * 3
            )
            -- enter manual mode
            retry_set_mode(MODE_MANUAL)
        end
        -- split time evenly between high and low signal
        if now < start_time + (DOUBLET_TIME / 2) then
            down  =  doublet_srv_trim  -  math.floor((doublet_srv_trim  -
doublet_srv_min) * (DOUBLET_MAGNITUDE / 45))
            SRV_Channels:set_output_pwm_chan_timeout(doublet_srv_chan,
down, DOUBLET_TIME / 2 + 100)
        elseif now < start_time + DOUBLET_TIME then
            up  =  doublet_srv_trim  +  math.floor((doublet_srv_max  -
doublet_srv_trim) * (DOUBLET_MAGNITUDE / 45))
            SRV_Channels:set_output_pwm_chan_timeout(doublet_srv_chan,
up, DOUBLET_TIME / 2 + 100)
        elseif now < start_time + (DOUBLET_TIME * 2) then
            -- stick fixed at pre doublet trim position
```

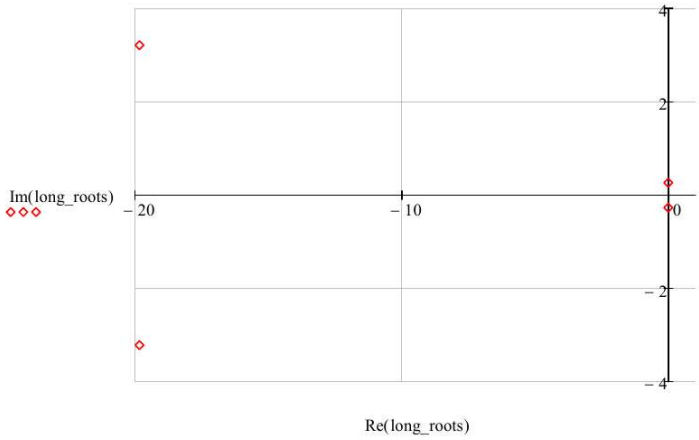
```
            SRV_Channels:set_output_pwm_chan_timeout(doublet_srv_chan,
doublet_srv_trim, DOUBLET_TIME * 2)
        elseif now > start_time + (DOUBLET_TIME * 2) then
            -- notify GCS
            end_time = now
            gcs:send_text(6, "DOUBLET FINISHED")
        else
            gcs:send_text(6, "this should not be reached")
        end
    elseif end_time ~= -1 and rc:get_pwm(DOUBLET_ACTION_CHANNEL) > 1700
then
        -- wait for RC input channel to go low
        gcs:send_text(6, "RC" .. DOUBLET_ACTION_CHANNEL .. " still high")
        callback_time = 100 -- prevents spamming messages to the GCS
    elseif now ~= -1 and end_time ~= -1 then
        gcs:send_text(6, "RETURN TO PREVIOUS FLIGHT MODE")
        now = -1
        end_time = -1
        start_time = -1
        -- clear all of the timeouts
        control_functions   =   {K_AILERON,   K_ELEVATOR,   K_THROTTLE,
K_RUDDER}
        for i = 1, 4 do
            local                       control_chan                    =
SRV_Channels:find_channel(control_functions[i])
            SRV_Channels:set_output_pwm_chan_timeout(control_chan,
param:get("SERVO" .. control_chan + 1 .. "_TRIM"), 0)
        end
        retry_set_mode(pre_doublet_mode)
        callback_time = 100 -- don't need to rerun for a little while
    elseif now ~= -1 then
        -- stopped before finishing. recover to level attitude
        gcs:send_text(6, "FBWA RECOVER")
        now = -1
        end_time = -1
        start_time = -1
        -- clear all of the timeouts
        control_functions   =   {K_AILERON,   K_ELEVATOR,   K_THROTTLE,
K_RUDDER}
        for i = 1, 4 do
            local                       control_chan                    =
SRV_Channels:find_channel(control_functions[i])
            SRV_Channels:set_output_pwm_chan_timeout(control_chan,
param:get("SERVO" .. control_chan + 1 .. "_TRIM"), 0)
        end
        retry_set_mode(MODE_FBWA)
        callback_time = 100
    end
    return doublet, callback_time
end
```

```
gcs:send_text(6, "doubletv3.lua is running")
return doublet(), 500
```

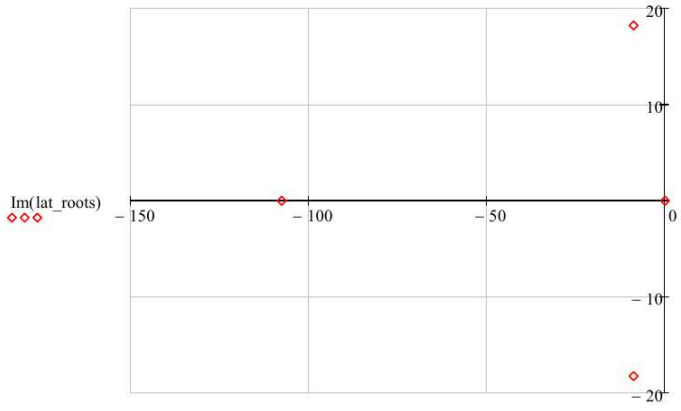
GEOMETRICSTABCON

$$\text{long\_roots} := \text{eigenvals}\left(A_{\text{long}}\right) = \begin{pmatrix} -19.832 + 3.216i \\ -19.832 - 3.216i \\ -0.019 + 0.268i \\ -0.019 - 0.268i \end{pmatrix}$$

Longitudinal Stability Coefficients



Re(long_roots)



Re(lat_roots)

Modes

Phugoid

$$Z_u := \frac{-1\cdot\left(C_{L.u} + 2\cdot C_{L.0.w}\right)\cdot Q\cdot S_w}{mass\cdot u_{0.u}} = -0.17\,\frac{ft}{s^2} \qquad Z_{wdot} := C_{z.\alpha dot}\cdot\frac{c_{bar.w}}{2\cdot u_{0.u}}\cdot\frac{Q\cdot S_w}{u_{0.u}\cdot mass} = -0.024\,\frac{ft^2}{s^2}$$

$$Z_{\alpha dot} := u_{0.u}\cdot Z_{wdot} = -1.203\,\frac{ft^2}{s^2}$$

$$Z_w := \frac{-1\cdot\left(C_{L.\alpha} + C_{D.0}\right)\cdot Q\cdot S_w}{mass\cdot u_{0.u}} = -5.566\,\frac{ft}{s}\cdot\frac{1}{s}$$

$$Z_\alpha := V_{inf}\cdot Z_w = -281.848\,\frac{ft}{s}\cdot\frac{ft}{s^2} \qquad Z_{\delta.e} := C_{z.\delta.e}\cdot\frac{Q\cdot S_w}{mass} = -41.441\,\frac{ft}{s^2}$$

$$Z_q := C_{z.q}\cdot\frac{c_{bar.w}}{2\cdot u_{0.u}}\cdot\frac{Q\cdot S_w}{mass} = -2.883\,\frac{ft^2}{s^2}$$

$$M_u := C_{m.u}\cdot\frac{Q\cdot S_w}{u_{0.u}\cdot I_{yy}} = 4.063\times 10^{-4}\,\frac{1}{s^2\cdot ft}$$

$$M_w := C_{m.\alpha}\cdot\frac{Q\cdot S_w\cdot c_{bar.w}}{u_{0.u}\cdot I_{yy}} = -5.325\,\frac{1}{s^2} \qquad M_{wdot} := C_{m.\alpha dot}\cdot\frac{c_{bar.w}}{2\cdot u_{0.u}}\cdot\frac{Q\cdot S_w\cdot c_{bar.w}}{u_{0.u}\cdot I_{yy}} = -0.198\,\frac{ft}{s^2}$$

$$M_\alpha := V_{inf}\cdot M_w = -269.645\,\frac{ft}{s^3} \qquad M_{\alpha dot} := u_{0.u}\cdot M_{wdot} = -10.038\,\frac{ft}{s^2}$$

$$M_q := C_{m.q}\cdot\left(\frac{c_{bar.w}}{2\cdot u_{0.u}}\right)\cdot\frac{Q\cdot S_w\cdot c_{bar.w}}{I_{yy}} = -24.056\,\frac{ft}{s^2} \qquad M_{\delta.e} := C_{m.\delta.e}\cdot\frac{Q\cdot S_w\cdot c_{bar.w}}{I_{yy}} = -345.73\,\frac{1}{s^2}$$

Remove the Units

$$M_q\cdot 0.288\,\frac{rad}{s} = -6.928\,\frac{ft}{s^3}$$

$$X_u := X_u\cdot\frac{s^2}{ft} = -0.042 \qquad X_w := X_w\cdot\frac{s^2}{ft} = -0.053$$

$$Z_u := Z_u\cdot\frac{s^2}{ft} = -0.17 \qquad M_w := M_w\cdot s^2 = -5.325$$

$$Z_w := Z_w\cdot\frac{s^2}{ft} = -5.566 \qquad M_{wdot} := M_{wdot}\cdot\frac{s^2}{ft} = -0.198$$

$$Z_\alpha := Z_\alpha\cdot\frac{s^3}{ft^2} = -281.848$$

$$V_{inf} := V_{inf}\cdot\frac{s}{ft} = 50.634$$

$$M_\alpha := M_\alpha\cdot\frac{s^3}{ft} = -269.645$$

$$M_q := M_q\cdot\frac{s^2}{ft} = -24.056 \qquad M_u := M_u\cdot s^2\cdot ft = 4.063\times 10^{-4}$$

$$A_{long} := \begin{pmatrix} X_u & X_w & 0 & -32.2 \\ Z_u & Z_w & V_{inf} & 0 \\ M_u + M_{wdot}\cdot Z_u & M_w + M_{wdot}\cdot Z_w & M_q + M_{wdot}\cdot u_{0.u} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$A_{long} = \begin{pmatrix} -0.042 & -0.053 & 0 & -32.2 \\ -0.17 & -5.566 & 50.634 & 0 \\ 0.034 & -4.222 & -34.093 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$C_{z.q} := -2 \cdot \eta_t \cdot C_{L.\alpha.t} \cdot V_H = -7.207$$

$$C_{z.\delta.e} := \left(\frac{-S_t}{S_w}\right) \cdot (dCLt\_d\delta e) = -0.734$$

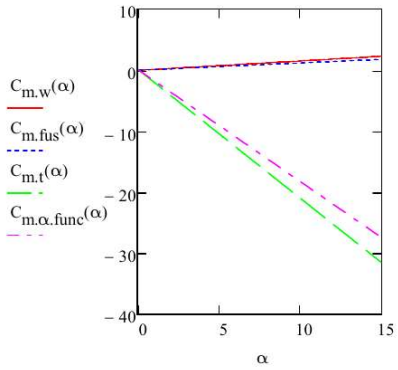
$$\alpha := 0, .1 .. 15$$

$$C_{m.w}(\alpha) := \alpha \cdot C_{L.\alpha} \cdot \left(\frac{x_{cg}}{c_{bar.w}} - \frac{x_{ac.w}}{c_{bar.w}}\right)$$

$$C_{m.fus}(\alpha) := \alpha \cdot C_{m.\alpha.fus}$$

$$C_{m.t}(\alpha) := \alpha \cdot \left[-1 \eta_t \cdot V_H \cdot C_{L.\alpha.t} \cdot (1 - d\varepsilon\_d\alpha)\right]$$

$$C_{m.\alpha.func}(\alpha) := C_{m.\alpha} \cdot \alpha$$



$$x_{NP} := \frac{x_{ac.w}}{c_{bar.w}} - \frac{C_{m.\alpha.fus}}{C_{L.\alpha}} + \eta_t \cdot C_{L.\alpha.t} \cdot V_H \cdot \frac{1}{C_{L.\alpha}} \cdot (1 - d\varepsilon\_d\alpha) = 0.687$$

Longitudinal Stability Derivatives

$$X_u := \frac{-1 \cdot \left(C_{D.u} + 2 \cdot C_{D.0}\right) \cdot Q \cdot S_w}{mass \cdot u_{0.u}} = -0.042 \frac{ft}{s} \cdot \frac{1}{s} \qquad X_w := \frac{-1 \cdot \left(C_{D.\alpha} - C_{L.0.w}\right) \cdot Q \cdot S_w}{mass \cdot u_{0.u}} = -0.053 \frac{ft}{s^2}$$

$L_{\delta.a} := L_{\delta.a} \cdot s^2 = 2.748 \times 10^3$

$g_u := g \cdot \frac{s^2}{ft} = 32.174 \qquad u_{0.u} := u_0 \cdot \frac{s}{ft} = 50.634$

$$A_{lat} := \begin{bmatrix} \frac{Y_\beta}{u_{0.u}} & \frac{Y_p}{u_{0.u}} & -\left(1 - \frac{Y_r}{u_{0.u}}\right) & \frac{g_u \cdot \cos(\theta_0)}{u_{0.u}} \\ L_\beta & L_p & L_r & 0 \\ N_\beta & N_p & N_r & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{pmatrix} -0.66 & 0 & -0.971 & 0.635 \\ 0 & -107.578 & 3.604 & 0 \\ 409.289 & 0 & -16.823 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$lat\_roots := eigenvals(A_{lat}) = \begin{pmatrix} -107.579 \\ -8.752 + 18.228i \\ -8.752 - 18.228i \\ 0.021 \end{pmatrix}$$

$$B_{lat} := \begin{pmatrix} 0 & \frac{Y_{\delta.r}}{u_{0.u}} \\ L_{\delta.a} & L_{\delta.r} \\ N_{\delta.a} & N_{\delta.r} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0.528 \\ 2.748 \times 10^3 & 64.957 \\ 0 & -303.211 \\ 0 & 0 \end{pmatrix}$$

$$C_{lat} := \begin{pmatrix} 0 & 0 & \frac{180}{\pi} & 0 \\ 0 & 0 & 0 & \frac{180}{\pi} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 57.296 & 0 \\ 0 & 0 & 0 & 57.296 \end{pmatrix} \qquad D_{lat} := \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Lateral Stability Coefficients

Longitudinal Stability Coefficients

Assume drag properties from NACA0010 wing

$C_{L.u} := 0 \qquad C_{D.u} := 0 \qquad C_{m.\alpha.fus} := 0.12 \qquad C_{D.\alpha} := 0.1239$

$C_{m.u} := 0.0001$

$$C_{m.\alpha} := C_{L.\alpha} \cdot \left(\frac{x_{cg}}{c_{bar.w}} - \frac{x_{ac.w}}{c_{bar.w}}\right) + C_{m.\alpha.fus} - \eta_t \cdot V_H \cdot C_{L.\alpha.t} \cdot (1 - d\varepsilon\_d\alpha) = -1.827$$

$$C_{m.\alpha dot} := -2 \cdot \eta_t \cdot C_{L.\alpha.t} \cdot V_H \cdot \frac{l_t}{c_{bar.w}} \cdot d\varepsilon\_d\alpha = -9.598$$

$$C_{m.q} := -2 \cdot \eta_t \cdot C_{L.\alpha.t} \cdot V_H \cdot \frac{l_t}{c_{bar.w}} = -23.002$$

$C_{m.\delta e} := -\eta_t \cdot V_H \cdot dCLt\_d\delta e = -2.342$

$C_{z.\alpha dot} := -2 \cdot \eta_t \cdot C_{L.\alpha.t} \cdot V_H \cdot d\varepsilon\_d\alpha = -3.007$

$$C_{n.p} := -1 \cdot \frac{C_L}{8} = 0$$

$$C_{n.r} := -1 \cdot 2 \cdot \eta_v \cdot V_v \cdot \frac{l_v}{b_w} \cdot C_{L.\alpha.v} = -0.324$$

$$C_{y.\delta.r} := \frac{S_v}{S_w} \cdot \tau_r \cdot C_{L.\alpha.v} = 0.474$$

$$C_{n.\delta.r} := -1 \cdot V_v \cdot \eta_v \cdot \tau_r \cdot C_{L.\alpha.v} = -0.248$$

$$C_{l.\delta.r} := \frac{S_v}{S_w} \cdot \frac{z_v}{b_w} \cdot \tau_r \cdot C_{L.\alpha.v} = 0.018$$

Lateral Stability Derivatives

$$Y_\beta := Q \cdot S_w \cdot \frac{C_{y.\beta.tail}}{mass} = -33.432 \frac{ft}{s^2}$$

$$Y_p := \frac{Q \cdot S_w \cdot b_w \cdot C_{y.p}}{2 \cdot mass \cdot u_0} = 0$$

$$L_p := \frac{Q \cdot S_w \cdot b_w^{\,2} \cdot C_{l.p}}{2 \cdot I_{xx} \cdot u_0} = -107.578 \frac{1}{s}$$

$$Y_r := \frac{Q \cdot S_w \cdot b_w \cdot C_{y.r}}{2 \cdot mass \cdot u_0} = 1.484 \frac{ft}{s}$$

$$L_r := \frac{Q \cdot S_w \cdot b_w^{\,2} \cdot C_{l.r}}{2 \cdot I_{xx} \cdot u_0} = 3.604 \frac{1}{s}$$

$$Y_{\delta.a} := \frac{Q \cdot S_w \cdot C_{y.\delta.a}}{mass} = 0$$

$$N_{\delta.a} := \frac{Q \cdot S_w \cdot b_w \cdot C_{n.\delta.a}}{I_{zz}} = 0$$

$$L_{\delta.a} := \frac{Q \cdot S_w \cdot b_w \cdot C_{l.\delta.a}}{I_{xx}} = 2.748 \times 10^3 \frac{1}{s^2}$$

$$N_\beta := \frac{Q \cdot S_w \cdot b_w \cdot C_{n.\beta}}{I_{zz}} = 409.289 \frac{1}{s^2}$$

$$L_\beta := \frac{Q \cdot S_w \cdot b_w \cdot C_{l.\beta}}{I_{xx}} = 0$$

$$N_p := \frac{Q \cdot S_w \cdot b_w^{\,2} \cdot C_{n.p}}{2 \cdot I_{zz} \cdot u_0} = 0$$

$$N_r := \frac{Q \cdot S_w \cdot b_w^{\,2} \cdot C_{n.r}}{2 \cdot I_{zz} \cdot u_0} = -16.823 \frac{1}{s}$$

$$Y_{\delta.r} := \frac{Q \cdot S_w \cdot C_{y.\delta.r}}{mass} = 26.745 \frac{ft}{s^2}$$

$$N_{\delta.r} := \frac{Q \cdot S_w \cdot b_w \cdot C_{n.\delta.r}}{I_{zz}} = -303.211 \frac{1}{s^2}$$

$$L_{\delta.r} := \frac{Q \cdot S_w \cdot b_w \cdot C_{l.\delta.r}}{I_{xx}} = 64.957 \cdot \frac{1}{s^2}$$

Remove the units

$$Y_\beta := Y_\beta \cdot \frac{s^2}{ft} = -33.432$$

$$Y_p := Y_p \cdot \frac{s}{ft} = 0$$

$$L_p := L_p \cdot s = -107.578$$

$$Y_r := Y_r \cdot \frac{s}{ft} = 1.484$$

$$L_r := L_r \cdot s = 3.604$$

$$Y_{\delta.a} := Y_{\delta.a} \cdot \frac{s^2}{ft} = 0$$

$$N_{\delta.a} := N_{\delta.a} \cdot s^2 = 0$$

$$N_\beta := N_\beta \cdot s^2 = 409.289$$

$$L_\beta := L_\beta \cdot s^2 = 0$$

$$N_p := N_p \cdot s = 0$$

$$N_r := N_r \cdot s = -16.823$$

$$Y_{\delta.r} := Y_{\delta.r} \cdot \frac{s^2}{ft} = 26.745$$

$$N_{\delta.r} := N_{\delta.r} \cdot s^2 = -303.211$$

$$L_{\delta.r} := L_{\delta.r} \cdot s^2 = 64.957$$

drag build up.

$C_{m.\delta.e} := -\eta_t \cdot V_H \cdot dCLt\_d\delta e$

Aircraft Aerodynamic Properties

Lateral Stability Coefficients

From Nelson Figure 2.29 and 2.30

$R_{l.f} := \frac{V \cdot l_f}{\nu} = 4.889 \times 10^5 \frac{lb \cdot ft}{A \cdot s^2}$ $\quad k_{Rl} := 1.8$

$\frac{x_m}{l_f} = 0.071$ $\quad \frac{l_f^2}{S_{fs}} = 5.396$ $\quad \left(\frac{h_{fl}}{h_{f2}}\right)^{0.5} = 0.911$ $\quad \frac{h_f}{w_f} = 1.397$ $\quad k_n := 0.0022$

$\eta_v := 1$ Assume ratio of dynamic pressure on vertical tail to dynamic pressure on wing

$$OnePlusd\sigma\_d\beta := \frac{0.724 + 3.06 \cdot \frac{\frac{S_v}{S_w}}{1 + \cos(\Lambda_{c\_4.w})} + 0.4 \cdot \frac{z_w}{d_f} + 0.009 \cdot AR_w}{\eta_v} = 1.088$$

$C_{n.\beta.wf} := -1 \cdot k_n \cdot k_{Rl} \cdot \frac{S_{fs} \cdot l_f}{S_w \cdot b_w} = -4.171 \times 10^{-5} \cdot \frac{1}{deg}$

Empirical factor for C.n.da from Nelson Figure 3.11

$K := -0.00018 \cdot \frac{1}{deg^2}$ This is C.l.β/Γ $\quad \Delta C_{l.\beta} := 0 \cdot \frac{1}{rad}$

$C_{l.\beta} := K \cdot \Gamma_w + \Delta C_{l.\beta} = 0$ (see Nelson Figure 3.11)

$C_{y.\beta.tail} := -\eta_v \cdot C_{L.\alpha.v} \cdot \frac{S_v}{S_w} = -0.592$ $\quad C_{l.p} := -1 \cdot \frac{C_{L.\alpha}}{12} \cdot \frac{1 + 3 \cdot \lambda_w}{1 + \lambda_w} = -0.718$

$C_{y.r} := -2 \cdot \frac{l_v}{b_w} \cdot C_{y.\beta.tail} = 0.62$ $\quad C_{y.p} := C_L \cdot \frac{AR_w + \cos(\Lambda_{c\_4.w})}{AR_w + 4 \cdot \cos(\Lambda_{c\_4.w})} \tan(\Lambda_{c\_4.w}) = 0$

$C_{l.r} := \frac{C_L}{4} - 2 \cdot \frac{l_v}{b_w} \cdot \frac{z_v}{b_w} \cdot C_{y.\beta.tail} = 0.024$ $\quad C_{y.\delta.a} := 0$

$$C_{l.\delta.a} := 2 \cdot C_{L.\alpha} \cdot \frac{\tau_a}{S_w \cdot b_w} \cdot \int_0^{\frac{b_w}{2}} c_{r.w} \cdot \sqrt{1 - 4 \cdot \left(\frac{y}{b_w}\right)^2} \cdot y \, dy = 0.778$$

$C_{n.\delta.a} := 2 \cdot K \cdot C_L \cdot C_{l.\delta.a} = 0$

$C_{n.\beta} := C_{n.\beta.wf} + \eta_v \cdot V_v \cdot C_{L.\alpha.v} \cdot OnePlusd\sigma\_d\beta = 0.335$

$x_m := x_{cg} = 2.75 \cdot in$ $S_{fs} := 280 \cdot in^2$ $l_f := 38.87in$

$w_f := 6.5in$ maximum fuselage width

$h_f := 9.08in$ maximum fuselage height

$\frac{l_f}{4} = 9.717 \cdot in$ location for h.1 $h_{f1} := 6.32in$

$\frac{3l_f}{4} = 29.152 \cdot in$ location for h.2 $h_{f2} := 7.62in$

$d_f := 9.08in$ maximum fuselage depth. Not defined well in Nelson.

Aircraft deminsional properties

Aircraft Aerodynamic Properties

2-D

$C_{l.\alpha.w} := 2\pi \cdot \frac{1}{rad}$ $C_{m.ac.w} := -0.1$ $C_{l.0.w} := 0.076$

$C_{l.\alpha.v} := 2\pi \cdot \frac{1}{rad}$ $C_{l.\alpha.t} := 2\pi \cdot \frac{1}{rad}$

3-D

$C_{L.0.w} := C_{l.0.w}$ $\eta_t := 1$ Assume η for the horizontal tail

$C_{L.\alpha} := \frac{C_{l.\alpha.w}}{1 + \left(\frac{C_{l.\alpha.w}}{\pi \cdot AR_w}\right)} = 4.972 \cdot \frac{1}{rad}$ 2 pi corrected for AR

$C_{L.\alpha.v} := \frac{C_{l.\alpha.v}}{1 + \left(\frac{C_{l.\alpha.v}}{\pi \cdot AR_v}\right)}$

$C_L := C_{L.\alpha} \cdot \alpha$

$C_{m.cg.w} := C_{m.ac.w} + C_L \cdot \left(\frac{x_{cg}}{c_{bar.w}} - \frac{x_{ac.w}}{c_{bar.w}}\right)$

$d\varepsilon\_d\alpha := 2 \cdot \frac{C_{L.\alpha}}{\pi \cdot AR_w}$

$C_{L.\alpha.t} := \frac{C_{l.\alpha.t}}{1 + \left(\frac{C_{l.\alpha.t}}{\pi \cdot AR_t}\right)}$ $dCLt\_d\delta e := C_{L.\alpha.t} \cdot \tau_e$

$C_{L.\delta.e} := \frac{S_t}{S} \cdot \eta_t \cdot dCLt\_d\delta e$ $C_{D.0} := 0.019$ Assume drag as a reasonable number to avoid a component

Flight Condition Properties

$\rho := 1.225\frac{kg}{m^3}$ $V_{inf} := 30knot$ $\alpha := 0deg$ $u_0 := V_{inf}$ $\theta_0 := 0deg$

$Q := \frac{1}{2}\cdot\rho\cdot V_{inf}^{\ 2} = 0.021\cdot psi$ $\nu := 1.5723\cdot 10^{-4}\cdot\frac{ft^2}{s}$ $\rho = 1.376\times 10^{-6}\cdot\frac{slug}{in^3}$

Flight Condition Properties

Aircraft deminsional properties

$mass := 1.914kg$ $x_{cg} := 2.75in$ $g_c := 32.2\frac{lb\cdot ft}{lbf\cdot s^2}$

$I_{xx} := 0.009slug\cdot ft^2$ $I_{yy} := 0.036slug\cdot ft^2$ $I_{zz} := 0.026slug\cdot ft^2$

Wing

$S_w := 418in^2$ $b_w := 51.5275in$ $c_{t.w} := 7.375in$ $c_{r.w} := 12.75in$ $\Gamma_w := 0deg$ $\Lambda_{c\_4.w} := -4.15deg$

$c_{bar.w} := 8.61in$ $\alpha_w := 0deg$

$i_w := 0deg$ $x_{LE.w} := 11.37in$ $x_{ac.w} := 2.485in$

$AR_w := \frac{b_w^{\ 2}}{S_w} = 7.586$ $\lambda_w := \frac{c_{t.w}}{c_{r.w}} = 0.578$ $z_w := -0.5in$ $S_a := 97.11in^2$

$x_{cg} = 2.75\cdot in$

$\frac{x_{cg}}{c_{bar.w}} = 0.319$

$\frac{S_a}{S_w} = 0.277$ use Nelson Figure 2.21 $\tau_a := 0.5$

$\frac{x_{ac.w}}{c_{bar.w}} = 0.289$

H Tail

$S_t := 102.8057in^2$ $b_t := 18in$ $\alpha_t := 0deg$ $i_t := 0deg$ $x_{LE.t} := 38.8in$ $x_{ac.t} := 40.55in$

$S_e := 44.94in^2$ $l_t := 27.48in$ $AR_t := \frac{b_t^{\ 2}}{S_t}$ $V_H := \frac{l_t\cdot S_t}{S_w\cdot c_{bar.w}} = 0.937$

$\frac{S_e}{S_t} = 0.437$ use Nelson Figure 2.21 $\tau_e := 0.65$

V Tail

$l_v := 26.97in$ leading edge of wing to leading edge of verticle tail

$S_v := 72.5\cdot in^2$ $z_v := 2in$ $b_v := 11in$ $S_r := 53.63in^2$

$V_v := \frac{S_v\cdot l_v}{S_w\cdot b_w} = 0.108$ $AR_v := \frac{b_v^{\ 2}}{S_v} = 1.669$

$\frac{S_r}{S_v} = 0.74$ use Nelson Figure 2.21 $\tau_r := 0.8$

Fuselage

$$\omega_{n\_phug} := \left|long\_roots_3\right| \cdot \frac{1}{2\cdot\pi} = 0.043$$

$$\zeta_{phug} := -\cos\left(\arg\left(long\_roots_3\right)\right) = 0.07$$

$$t_{2\_phug} := \frac{-6.93}{Re\left(long\_roots_3\right)} = 370.105 \quad [sec]$$

$$P_{phug} := \frac{2\cdot\pi}{\left|Im\left(long\_roots_3\right)\right|} = 23.442 \quad [sec]$$

$$N_2 := 0.110 \cdot \frac{\left|Im\left(long\_roots_3\right)\right|}{\left|Re\left(long\_roots_3\right)\right|} = 1.575 \quad [cycles]$$

Short Period

$$\omega_{n\_SP} := \left|long\_roots_0\right| = 20.091$$

$$\zeta_{SP} := -\cos\left(\arg\left(long\_roots_0\right)\right) = 0.987$$

$$\arg\left(long\_roots_0\right) = 2.981$$

$$t_{2\_SP} := \frac{-6.93}{Re\left(long\_roots_0\right)} = 0.349 \quad [sec]$$

$$P_{SP} := \frac{2\cdot\pi}{\left|Im\left(long\_roots_0\right)\right|} = 1.954 \quad [sec]$$

$$N_2 := 0.110 \cdot \frac{\left|Im\left(long\_roots_0\right)\right|}{\left|Re\left(long\_roots_0\right)\right|} = 0.018 \quad [cycles]$$

Dutch Roll

$$\omega_{n\_DR} := \left|lat\_roots_0\right| \cdot \frac{1}{2\cdot\pi} = 17.122$$

$$\zeta_{DR} := -\cos\left(\arg\left(lat\_roots_1\right)\right) = 0.433$$

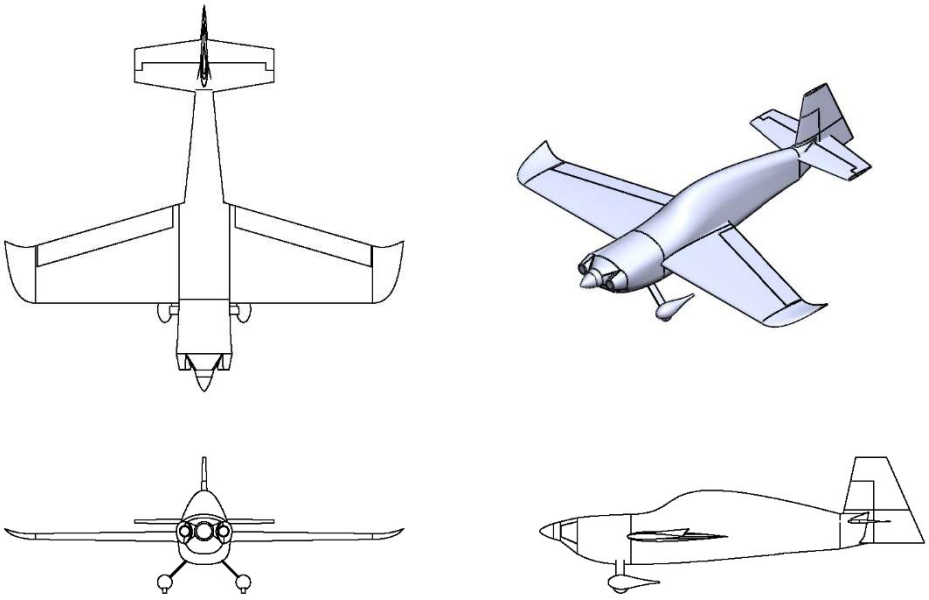
$$t_{2\_DR} := \frac{-6.93}{Re\left(lat\_roots_1\right)} = 0.792 \quad [sec]$$

$$P_{DR} := \frac{2\cdot\pi}{\left|Im\left(lat\_roots_1\right)\right|} = 0.345 \quad [sec]$$

$$N_2 := 0.110 \cdot \frac{\left|Im\left(lat\_roots_1\right)\right|}{\left|Re\left(lat\_roots_1\right)\right|} = 0.229 \quad [cycles]$$

Modes

3-VIEW DRAWING OF FMS EDGE 540 1300 MM MODEL



OKLAHOMA STATE UNIVERSITY

TITLE

FMS EDGE 540 1300 mm

APPROVED	N/A	SIZE A	CAGE CODE N/A	DWG NO 1	REV A
DESIGNED	CPJ	SCALE 1:15	4.22 LBS	SHEET 1 OF 1	

VITA

Charles Preston Johnson

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A SOFTWARE IN THE LOOP SIMULATION APPROACH FOR RISK MITIGATION IN UNMANNED AERIAL SYSTEM DEVELOPMENT

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in December, 2020.

Completed the requirements for the Bachelor of Science in Mechanical Engineering at Oklahoma State University, Stillwater, Oklahoma in 2018.

Completed the requirements for the Bachelor of Science in Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in 2018.

Experience:

Graduate Research Assistant to Dr. Andy Arena, Department of Mechanical and Aerospace Engineering, Oklahoma State University, June 2018 – Present. Responsibilities include designing, manufacturing, and testing composite UAS. Specializing in electrical and autopilot systems.

Professional Memberships:

Member of Sigma Gamma Tau

Member of American Institute of Aeronautics and Astronautics