SMALL UNMANNED AIRCRAFT SYSTEMS OPERATIONAL AND TRAFFIC MANAGEMENT CONSIDERATIONS

By

ZACHARY P. BARBEAU

Bachelor of Science in Mechanical & Aerospace
Engineering
Oklahoma State University
Stillwater, OK
2014

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
JULY, 2017

SMALL UNMANNED AIRCRAFT SYSTEMS OPERATIONAL AND TRAFFIC MANAGEMENT CONSIDERATIONS

Thesis Approved:

Dr. Jamey Jacob

Thesis Advisor

Dr. James Kidd

Dr. Brian Elbing

ACKNOWLEDGMENTS

First and foremost, I would like thank Dr. Jamey Jacob for his mentorship and guidance throughout my studies at Oklahoma State University. I'll always be grateful for the countless opportunities to grow and develop as a researcher and engineer. I would also like to acknowledge my colleagues for all the help provided during this thesis project. I would like to thank Seabrook Whyte and Marc Hartman for the time spent manually flying aircraft for tuning and also serving as safety pilots. Thanks to Taylor Mitchell and Fred Keating for all of the operational guidance. As a flight team we have developed a highly regarded reputation for safe and professional UAS operations. Finally, I would like to thank my Mom, Dad, and Brother for all of their support throughout my studies.

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: Zachary P. Barbeau Date of Degree: JULY, 2017

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: SMALL UNMANNED AIRCRAFT SYSTEMS OPERATIONAL AND TRAFFIC MANAGEMENT CONSIDERATIONS

Major Field: Mechanical and Aerospace Engineering

A substantial growth in the number of unmanned aircraft systems (UAS) operating within U.S. national airspace is projected through the next two decades. Regulations for small UAS weighing under 55 pounds have been enacted per Part 107. Small UAS are restricted from operating outside Part 107 rules primarily due to see and avoid (SAA) criteria that exist in the Federal Aviation Regulations, in addition to key UAS traffic management milestones. Several research questions are addressed relevant to higher density, beyond visual line of sight, small UAS scenarios including operational takeoff and landing procedures, separation, and avionics architecture. Flight test architecture was developed to evaluate fixed wing small UAS autonomous approach and landing. Modifications were developed for the command and control software enabling point and click traffic pattern generation and real time performance logging. The accuracy of multi-rotor auto landing capability with different sensor configurations was also examined. Recommendations are formed from flight test observations and provided for future small UAS operations in the national airspace including a case study examining operation of small UAS from an existing general aviation airport.

ADVISOR'S APPROVAL: Dr. Jamey Jacob

TABLE OF CONTENTS

Chapter Page

1 Introduction 1

1.1 Motivation 1

1.2 Research Questions and Objectives 3

1.2.1 Objectives 4

1.3 Outline 5

2 Previous Work 6

2.1 Small UAS Part 107 and General Aviation Pilotage 6

2.2 UAS Integration into the National Airspace System 11

2.2.1 NASA UAS Traffic Management (UTM) 11

2.2.2 Small UAS Flight Tests 13

2.2.3 Pixhawk IMU Characterization and Flight Test 16

3 Methodology 17

3.1 Autopilot Architecture and Ecosystem 17

3.1.1 ArduPilot 19

3.1.2 PX4 Pixhawk 19

3.2 Flight Test Technique 22

3.2.1 Uncertainty and Error 22

3.2.2 Taking the Human Out of the Loop 24

3.2.3 Command and Control, Auto Land Traffic Pattern Script . . . 25

3.3 UAS Platforms and Test Overview 47

3.3.1 Nominal Flight Plan 50
3.4 Autopilot Takeoff and Landing 52
3.4.1 Critical Autopilot Parameters 52
3.4.2 Auto Land Logic 56

4 Results **60**
4.1 Fixed-Wing Flight Test 60
4.2 Multi-Rotor Flight Test 67
4.3 Wake Vortex Considerations 68

5 Conclusions **72**
5.1 Flight Test Conclusions 72
5.2 General Aviation Infrastructure Case Study 74
5.2.1 Elements of Small UAS Operation at Stillwater Regional . . . 77

A Appendix **95**
A.1 Autoland Traffic Pattern Script 95
A.2 Operator Checklists 106
A.3 Flight Time Log and Configuration Management 113
A.4 KSWO Preflight Briefing Outline 116

BIBLIOGRAPHY **118**

LIST OF FIGURES

Figure Page

2.1 Airspace Profile [5] 8
2.2 Example Class E Airport UAS Facility Map [6] 9
2.3 Standard Left Hand Traffic Pattern [3] 10
2.4 Methods for Traffic Pattern Entry [3] 11
2.5 UTM Technical Capability Levels (TCL) [7] 13
2.6 Small UAS, General Aviation Encounter [13] 15

3.1 General UAS architecture [15] 18
3.2 General Pixhawk v1 component info graphic [18] 21
3.3 Mission Planner Architecture, Adapted from [24] 27
3.4 Mission Planner Flight Data Overview 27
3.5 Nominal Left Hand Traffic Pattern 30
3.6 Traffic Pattern Landing Script State Flow 31
3.7 Mission Planner Status Page 36
3.8 Horizontal Plane (North-East) Angular and Vector Relationships [31] 38
3.9 Auto Land Traffic Pattern Script Generated Right Hand Pattern . . . 46
3.10 RMRC Anaconda and Multi-Rotors with Approximate Relative Sizes 48
3.11 Anaconda SF11 Laser Altimeter Configuration 49
3.12 Landing Target View, DJI Mavic Camera 51
3.13 Landing Performance Definitions 51

4.1 Wind Rose for Fixed Wing Landing Approaches, Vehicle Estimate at Short Final 60

4.2 Anaconda Auto Land Performance, Wheel Stop Locations 61
4.3 Anaconda Auto Land Performance, EKF Height Estimate and Laser Rangefinder State Time Trace [meters] 62
4.4 Anaconda Auto Land Performance, Trajectory 63
4.5 Anaconda Auto Land Performance, Pitch Desired/Actual Trace . . . 64
4.6 Anaconda Auto Land Performance, Roll Desired/Actual Trace 64
4.7 Anaconda Auto Land Performance, Flare Initiation 65
4.8 Anaconda Auto Land Performance, Short Final Airspeed Histogram . 65
4.9 Anaconda Auto Land Performance, Short Final Altitude Histogram . 66
4.10 Anaconda Auto Land Performance, Wheel Stop Target Cumulative Probability 66
4.11 Wake vortex encounter scenario [35] 69
4.12 Wake Avoidance ADS-B Concept 71

5.1 Grand Sky UAS Business & Aviation Park [42] 75
5.2 Penguin-B [39] 79
5.3 Pixhawk 2.1 Redundant Power Distribution Architecture, Mouch Electronic 80
5.4 Elements of UAS CONOP, adapted from [12] 82
5.5 KSWO UA Flight Plan 85
5.6 Flight Plan Card or Flight Test Card Sample 87
5.7 Before Landing Checklist, Anaconda 88

A.1 Anaconda Operator Checklist, Preflight Part 1 106
A.2 Anaconda Operator Checklist, Preflight Part 2 107
A.3 Anaconda Operator Checklist, Autopilot Configuration GPS Guided . 108
A.4 Anaconda Operator Checklist, Autopilot Configuration Takeoff and Landing, Part 1 109

A.5 Anaconda Operator Checklist, Autopilot Configuration Takeoff and Landing, Part 2 110
A.6 Anaconda Operator Checklist, Before Takeoff 111
A.7 Anaconda Operator Checklist, Before Landing 112
A.8 Anaconda Flight Log 113
A.9 Anaconda Configuration Tracking 114
A.10 Anaconda Configuration Tracking, Detailed Entry Sample 115
A.11 KSWO Detailed Preflight Brief Outline, Adapted from OSU UAS Flight Test and Certification Graduate Course, Part 1 116
A.12 KSWO Detailed Preflight Brief Outline, Adapted from OSU UAS Flight Test and Certification Graduate Course, Part 2 117

NOMENCLATURE

$\overline{V}^{ned}$	Inertial velocity vector
$\overline{V}_n$	North inertial velocity
$\overline{V}_e$	East inertial velocity
$\overline{V}_{air}^{ned}$	Inertial relative airspeed vector
$\overline{V}_w^{ned}$	Inertial wind vector
$\overline{V}_w^n$	North inertial wind velocity
$\overline{V}_w^e$	East inertial wind velocity
ψ	Euler yaw angle
θ	Euler pitch angle
ϕ	Euler roll angle
δ	Magnetic declination
N_{mag}	Magnetic north
Ψ	True body yaw angle
β	Aerodynamic side slip angle
χ	Inertial velocity heading
χ_{crab}	Wind-induced crab angle
χ_w	Wind heading
ϕ_{lat}	Latitude
$\lambda_{lng,2}$	Longitude
θ_{nav}	Navigation bearing
δ	Angular distance
D	Waypoint leg distance

R	Radius of Earth
Γ	Circulation strength
$C_{L\alpha}$	3D lift curve slope
U_∞	Aircraft velocity
W	Aircraft weight
b	Aircraft wingspan
b'	Effective vortex span
c	Wing chord
S	Wing area
τ	Flap effectiveness constant
δ_a	Aileron deflection
C_l	Roll moment coefficient
C_{l_v}	Vortex induced roll moment coefficient
$C_{l_{\delta a}}$	Roll control power, aileron
$C_{l_v}/C_{l_{\delta a}}\delta_a$	Roll control ratio

CHAPTER 1

Introduction

1.1 Motivation

An exponential growth in the number of unmanned aircraft systems (UAS) operating within U.S. national airspace is projected through the next two decades. Federal Aviation Administration (FAA) Administrator Michael Huerta announced, in February 2016, that the total number of registered UAS operators eclipsed the number of manned aircraft pilots. The FAA's 2017-2037 Aerospace Forecast estimates as many as 1.6 million commercial small UAS in the national airspace system (NAS) by 2021 [1]. The Teal Group, a contributor to the forecast, acknowledges a commercial UAS forecast is volatile and highly dependent on market reaction to present and future regulations. In fact, before the official small UAS Code of Federal Regulations (CFR) Part 107 regulation was released, The Department of Transportation's Volpe Center released a 2013 report forecasting UAS demand from 2015-2035 [2]. Volpe estimated 175,000 commercial UAS by 2035. Four years later, with data from the FAA Part 107 database, 44,000 commercial small UAS were registered in 2016. 420,000 active commercial small UAS are now expected to be operational as a baseline scenario by 2021 as shown in Table 1.1. Forecasts and market projections will continue to fluctuate in the near term, but UAS operations will grow at a substantial rate within the next two decades.

Table 1.1: Million Small UAS Units, FAA Baseline Commercial UAS Forecast [1]

	2016	**2017**	**2018**	**2019**	**2020**	**2021**
Hobbyist	1.10	2.15	2.80	3.20	3.40	3.55
Commercial	0.042	0.108	0.167	0.242	0.327	0.422

At the time of this study, small UAS weighing under 55 pounds can operate in uncontrolled airspace below 400 feet as part of FAA Part 107 criteria. Small UAS will be initially restricted from integrating with controlled air traffic primarily due to see and avoid (SAA) criteria that exists in the Federal Aviation Regulations, in addition to several key UAS traffic management milestones. Integration into controlled airspace will follow once small and medium sized UAS reliably satisfy SAA criteria and are coordinated through a robust traffic management system. The National Aeronautics and Space Administration (NASA) has played a key role in researching and developing the crewed Next Generation Transportation System (NextGen) NAS architecture in conjunction with the FAA. NASA Aeronautics must now apply decades of experience developing technology and procedures for crewed aircraft towards UAS Traffic Management (UTM) concepts.

NASA UTM is a near term research initiative with several Technical Capability Levels (TCL) defined over a five year period that are critical to far term UAS NAS integration standards that could inform future implementation by the FAA. TCL 1 and 2 were completed in August 2015 and October 2016, respectively. Most recently, TCL 2 focused on beyond visual line of sight enabling technologies in sparsely populated areas and dynamic airspace contingencies. TCL 3 and 4 will leverage the previous milestones and focus on developing technology necessary to integrate UAS in high density, controlled airspace. A fundamental traffic management paradigm of slow moving VFR and fast moving IFR aircraft is a robust set of procedural rules, sequencing, and separation standards. UTM and the eventual FAA implementation must include the same set of standards applied to UAS.

A key principle of any safe integrated air traffic control concept is that aircraft must remain well clear of each other. As large numbers of UAS are integrated into the NAS, this directive becomes more challenging. In addition to near misses and mid-air collision, wake vortex encounters could pose a threat to UAS. Notwithstanding vortex incidents, according to the current FAA Airplane Flying Handbook, 45% of all general aviation accidents occur during the approach and landing phases, with over 90% of these accidents caused by some pilot error—including loss of control [3]. Nonetheless, this introduction aims to provide relevant background and motivation to support the investigation of several small UAS NAS integration research questions relating to inevitable higher volume UAS operations.

1.2 Research Questions and Objectives

Answering the selected research questions will contribute towards the removal of fundamental barriers preventing large scale UAS operations. It is important to note that the questions represent a small part of the larger research gap to fill before a large scale UTM system can be implemented. Basic takeoff and landing procedures, sequencing and separation, avionics architecture, and overall concept of operation will be the focus. There are limited airspace procedures and operational requirements for small UAS, and examining existing airspace procedures is a first step that will quickly highlight any deficiencies to be further examined. The selected research questions are particularly applicable for higher volume, beyond visual line of sight scenarios. As more operations are staged from a single point of departure and arrival, a set of procedural rules, similar to general aviation, need to be implemented. Included in the set of procedures is the basic structure of an autonomous takeoff, traffic pattern, and landing maneuver.

1. What is relevant for small UAS operations regarding existing airspace departure and arrival procedures, "rules of the sky"?

(a) What is the nominal approach procedure for both fixed-wing and rotary small UAS?

(b) How are contingency situations handled? Missed approach, wave off, power loss, lost communications, etc.

2. What is the minimum avionics architecture and equipage necessary to enable precision departure and arrival of small UAS?

1.2.1 Objectives

Several objectives are listed below that complement the overall research questions.

1. Develop small UAS operational flight test technique and crew training resources:

   (a) Flight test performance logging capability via an automated tool

   (b) Aircraft hardware and software configuration management

   (c) Flight and test planning quick reference documentation

   (d) Systems checklists for relevant stages of flight: preflight, start, before take-off, and before landing

2. Develop flight test architecture:

   (a) Autopilot takeoff, navigation, and landing capability

   (b) Instrumented aircraft equipped with following sensors: Global Navigation Satellite System (GNSS), airspeed, Inertial Measurement Unit (IMU), laser or sonar altimeter

3. Develop recommendations based on flight test conclusions for future small to medium class UAS operations in the NAS

1.3 Outline

The following Chapter is a literature review of airspace rules, FAA regulations, UAS integration into the NAS considerations and UTM, and relevant small to medium class UAS studies. Chapter 3 will outline the methodology and flight test architecture. As part of the Chapter 3, detailed operational procedures for small UAS command and control are provided with the Pixhawk flight management unit and ArduPlane flight stack used as the hardware and software in the UAS platform, respectively. Chapter 4 will detail results from flight test sessions and highlight potential small UAS wake vortex hazards. The final chapter will summarize key flight test conclusions and discuss a local airport small UAS Concept of Operation (CONOP) as a recommendation for future work.

CHAPTER 2

Previous Work

2.1 Small UAS Part 107 and General Aviation Pilotage

The United States National Airspace System (NAS) encompasses all infrastructure and information necessary to facilitate air travel in the United States. It is one of the world's most complex and integrated airspaces that includes airports, navigational aids, air traffic control services, aeronautical charts, technical information, rules, regulations, procedures, personnel, and a mix of commercial, military, general aviation, agriculture, and sport aircraft. For the purpose of this study existing certification, airspace rules, and operational flight procedures will be discussed. A pilot certificate is required to legally operate an aircraft within the NAS and there are seven pilot certificates in order of increasing training requirements: remote, student, sport, recreational, private, commercial, and airline transport. In certain circumstances, an individual can operate an ultralight vehicle without any formal certification or training under CFR Part 103—Ultralight Vehicles. However, remote pilot, is the only credential granted with no flight training component.

In June of 2016 the small UAS rule, commonly referred to as Part 107, was published [4]. Small UAS are defined as weighing less than 55 pounds on takeoff and operated without an ability to intervene from within or on the aircraft. The designated remote pilot-in-command (PIC) must maintain knowledge of, at all times, the unmanned aircraft (UA) location, attitude, altitude, and direction of flight through either direct line of sight or a visual observer. If a visual observer (VO) is used, direct communication with the PIC is required. A single remote PIC can operate one vehi-

cle at time and only during daylight hours without a waiver. Additional operational highlights are detailed as follows:

1. For an in-flight emergency requiring immediate action, a remote PIC may deviate from any rule in Part 107 necessary to meet that emergency
2. No operation in controlled airspace without prior authorization from FAA
3. No flight in a manner that interferes with operations and traffic patterns at any airport, heliport, or seaplane base
4. Groundspeed may not exceed 87 knots (100 miles per hour)
5. Maximum altitude of 400 feet above ground level
6. Minimum flight visibility of 3 statute miles
7. No operation of a small UAS so close to another aircraft as to create a collision hazard
    (a) Yield right of way to all aircraft
8. No flight over people unless directly involved in operation
9. No operation from a moving aircraft or vehicle
    (a) Operation from a land or sea borne vehicle allowed only over sparsely populated areas

There are two categories of airspace in the NAS, regulatory and nonregulatory (usually military). Within both categories are four types: controlled, uncontrolled, special use, and other airspace. A top down hierarchy of regulatory airspace classification is used with Class A airspace being the most restrictive and Class G being the least restrictive. FAA Air Traffic Control (ATC) has authority and responsibility for controlled airspace (Class A-E) and pilots flying in uncontrolled airspace

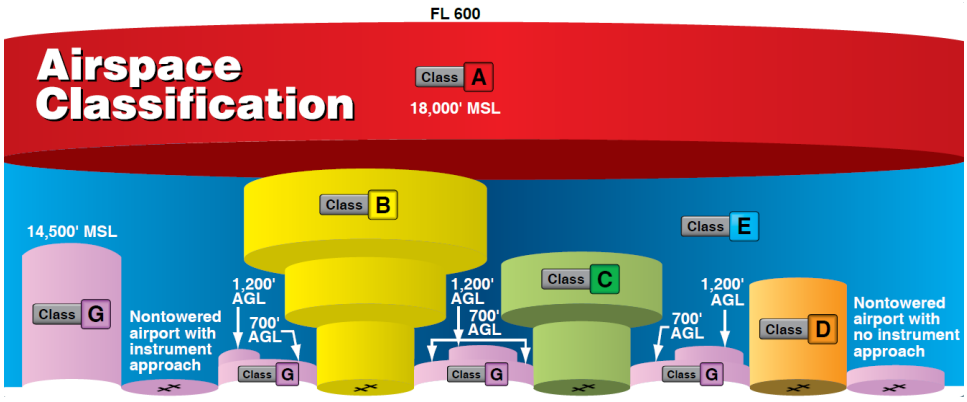

Figure 2.1: Airspace Profile [5]

(Class G) must observe VFR minimums. Figure 2.1 shows a generic airspace profile. Class B, C, and D airspace usually have individually configured areas. Prior authorization through a waiver is required before operations in controlled airspace can be conducted. The FAA has begun to streamline this process for operations near small airports designated as surface class E airspace with designated UAS facility maps.

UA facility maps depict where the FAA may authorize Part 107 UAS operations without additional safety analysis. The first set of published maps at the time of this study are for Class E airports, although the FAA has indicated that it will publish select maps for airports within more congested airspace such as Class D airports [6]. See Figure 2.2 for an example of a Class E airport with specific locations and altitudes where UA can potentially operate. The UA facility maps are designed to speed up the current waiver process, but the FAA does grant UA specialized access to higher density controlled airspace given enough justification and safety analysis. It is important to note that operations immediately surrounding the airport are still prohibited without a waiver.

A standard general aviation traffic pattern is typically flown at 1,000 feet above runway elevation and consists of four legs: crosswind, downwind, base, and final. Unless specifically noted for a particular airport, or an airport with parallel runways, a nominal traffic pattern is left handed as shown in Figure 2.3. Standard procedure

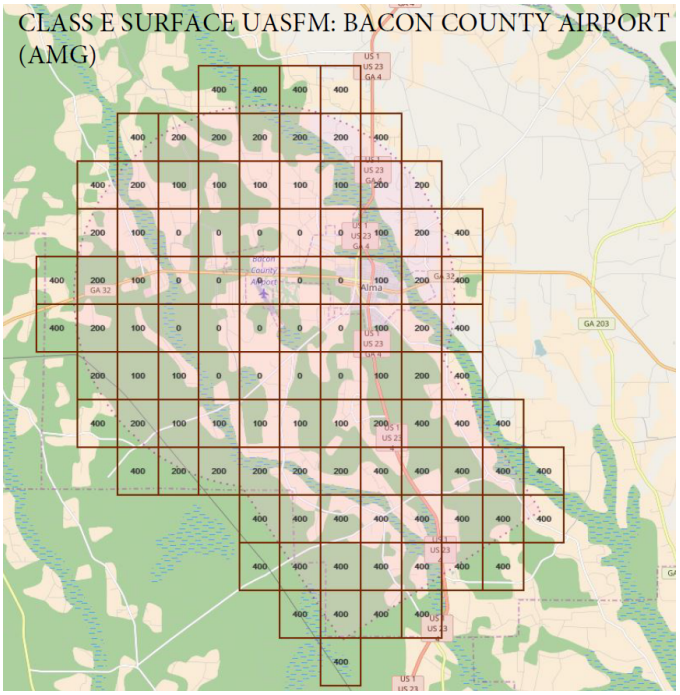

Figure 2.2: Example Class E Airport UAS Facility Map [6]

to enter a traffic pattern involves crossing the midfield point at least 500 feet above pattern altitude, descending to pattern altitude, and then entering the downwind leg while maintaining a 45° intersection course. Helicopters fly the same traffic pattern, unless directed by a control tower, but at a pattern altitude of 500 feet and turns executed to the right to avoid the flow of fixed-wing traffic.

The standard entry procedure, along with an alternate method is shown graphically in Figure 2.4. The alternate method is not advisable when the pattern is busy. Aircraft should yield to other traffic established for pattern entry or already on the downwind leg. Pattern altitude should be maintained until the base leg. A gradual descent is started during the base leg and the aircraft is set up to turn to final approach. A stabilized approach, a constant glide path to a targeted landing point, or aim point is the objective. The aim point is beyond the runway threshold, but before the first third of total runway distance. A stabilized approach requires a pilot to constantly manage the aircraft configuration (airspeed, power, pitch) and utilize visual references to maintain glide path to the aim point. The aircraft flares before impact-

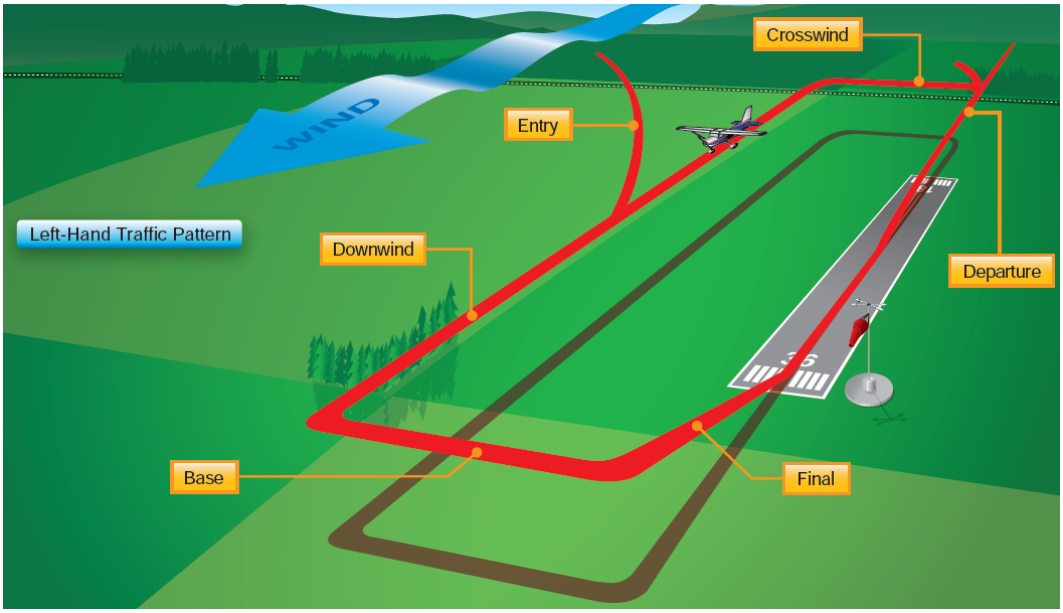

Figure 2.3: Standard Left Hand Traffic Pattern [3]

ing the aim point, reducing descent rate and bleeding airspeed before touchdown. If a landing can not be executed, a go-around should be initiated.

Go-arounds or rejected landings are not emergency procedures. It is a normal maneuver that is also used during an emergency scenario. A landing approach can be rejected for any number of reasons, but the most common reason for a go-around is an unstable approach that is unlikely to hit the desired aim point. The unstable approach can be due to pilot, environmental, or mechanical factors. A go-around is not necessarily hazardous if executed properly. Indecisiveness leading up to or at the instant of a go-around are hazardous. Maximum power must be applied smoothly to execute a go-around and aircraft attitude should be maintained until sufficient airspeed allows a climb. After airspeed has stabilized, a max power climb is initiated to 500 feet above runway elevation to begin the traffic pattern.

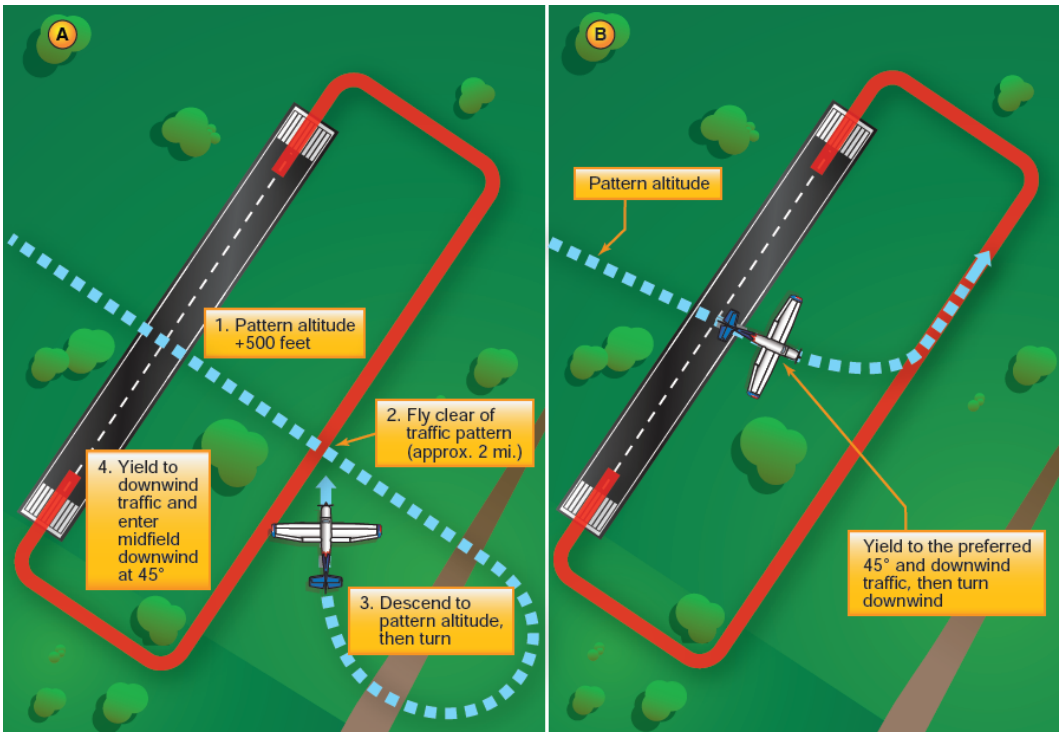

Figure 2.4: Methods for Traffic Pattern Entry [3]

2.2 UAS Integration into the National Airspace System

2.2.1 NASA UAS Traffic Management (UTM)

NASA UAS Traffic Management (UTM) is a multi-year research program to identify services, roles, responsibilities, architecture, infrastructure, and performance requirements to enable management of multiple, beyond visual line of sight (BVLOS) UAS operations in low-altitude, uncontrolled airspace [7]. NASA's UTM concept is not a system for traditional air traffic controllers to actively "control" UAS. It is envisioned as a system utilized by the FAA to monitor operations in real time and dynamically issue clearances, advisories, constraints, and airspace corridors. Table 2.1 details the fundamental principles and services provided by a UTM concept. Selected research and development focal areas from a vehicle perspective include: tracking via ADS-B or similar, reliability of the autopilot system, and safe, autonomous takeoff and landing within the first/last 50 feet of a mission.

Table 2.1: NASA UTM Principles and Services [7]

Principles	**UAS Services**
Operate in authenticated airspace	Authorization/Authentication
UAS stay clear of other UAS	Airspace config. and geofencing
UAS/manned aircraft stay clear	Track and locate
UAS operator situational awareness	Command and control (spectrum)
Public safety UAS have priority	Weather and wind sensing, prediction
	Conflict avoidance
	Demand/capacity management
	Large scale contingency management

UTM's development and implementation road map is primarily driven by four Technical Capability Levels (TCL), as seen in Figure 2.5. TCL 1 and 2 have been completed at the time of this study. TCL 1 collected state data (conducted across multiple states) for operations, weather conditions, and demonstrated the initial UTM software framework to include scheduling and planning of authorized airspace. TCL 2 analysis is still ongoing, but several initial lessons learned have been formed. Overall as the operational range (BVLOS) and density increased, it was apparent that an altitude standard was needed. Also, wind and weather sensing factored in to the ability to provide tasking and airspace for UAS to remain "well clear." Overall, better forecasting, or reporting of wind data would be beneficial. The concluding remark was to "expect the unexpected" in reference to contingency management [7].

NASA UTM research, development, and testing not only provides validated requirements to enable core TCL objectives, but also provides several key technology transfer deliverables. At the conclusion of the TCL 4 milestone, NASA Aeronautics plans to begin technology transfer of the UTM prototype, architecture of services, and associated requirements over to the FAA. Near term objectives include a UTM pilot program with FAA through 2019, after which the FAA will make operational adjustments for a target implementation in the early 2020s.

Capability 1: Demonstrated Multiple Operations Under Constraints	Capability 3: Focus on Enabling Multiple Heterogeneous Operations
• Notification of area of operation • Over unpopulated land or water • Minimal general aviation traffic in area • Contingencies handled by UAS pilot • **Product: Overall concept of operations, architecture, and roles (user/regulator)**	• Beyond visual line of sight • Over moderately populated land • Some interaction with manned aircraft • Tracking, vehicle to vehicle, and vehicle to UTM connection • **Product: Requirements for heterogeneous operations**
Capability 2: Demonstrated Expanded Multiple Operations	Capability 4: Demonstrated How to Enable Multiple Operations Under Constraints
• Beyond visual line of sight (BVLOS) • Tracking and low density operations • Sparsely populated areas • Procedures and "rules of the road" • Longer range applications • **Product: Requirements for multiple BVLOS operations**	• Notification of area of operation • Over unpopulated land or water • Minimal general aviation traffic in area • Contingencies handled by UAS pilot • **Product: Overall concept of operations, architecture, and roles (user/regulator)**

Figure 2.5: UTM Technical Capability Levels (TCL) [7]

2.2.2 Small UAS Flight Tests

The majority of small UAS flight testing with respect to NAS integration has primarily focused on maturing tracking and locating technology with low SWAP requirements. An emerging technology is low SWAP Automatic Dependent Surveillance Broadcast (ADS-B) systems. ADS-B out, or ability to broadcast GNSS data, is required by January 2020 to fly in most controlled airspace in the United States, by FAA mandate. If all aircraft, both manned and unmanned, operating in the NAS utilized ADS-B, probability of conflict could be reduced. ADS-B technology has been certified, implemented operationally, and maintains independence from other critical safety systems such as Traffic Alert and Collision Avoidance Systems (TCAS). Thus, it appears to be a logical solution for small UAS tracking and locating within the NAS.

There are numerous successful case studies regarding ADS-B and small UAS, but these studies were executed in operating environments that may or may not be similar

to an actual future airspace scenario. There is concern that ADS-B frequencies could quickly become saturated based on the predicted numbers of small UAS operating in low altitude airspace. MITRE Corporation staff performed simulations of varying small UAS and general aviation densities, along with transmit power [8]. MITRE concluded that existing legacy general aviation traffic densities have little impact on frequency congestion, but small UAS density does. A balance of small UAS density and transmit power is necessary to ensure safe and reliable coverage, but MITRE acknowledged that it appears feasible.

Industry, military, and research institutions have all successfully utilized small ADS-B equipment to fly UAS within the NAS. NASA Langley Research Center's subscale flight dynamics research aircraft recently completed several beyond visual line of sight flights equipped with ADS-B [9]. NASA Langley's flights were conducted under a FAA certification of authorization (COA) and small SWAP ADS-B capability was a key technology necessary to enable BVLOS flight. In addition to providing basic separation confidence, ADS-B has been explored as a data source to other alerting systems, such as a wake advisory system. Handley describes a framework for generating wake turbulence advisory corridors using ADS-B [10]. Another example of ADS-B use on small UAS is Utah State University's AggieAir group. AggieAir has flown a proprietary integrated UTM framework using commercial off the shelf small SWAP ADS-B equipment, described in literature [11].

In addition to UTM development, AggieAir is also a proponent of developing robust flight operations, documentation, and certification standards necessary to conduct small UAS flights. A concept of operations (CONOPS) is referenced by AggieAir [12]. It is still unknown what elements of a similar CONOPS will be regulated as part of a future FAA UTM system that allows BVLOS flight. It is possible that operators will be certified for BVLOS and required to keep an operation structured similar to AggieAir[12]. A relevant objective of this thesis is to examine elements that fall under

the flight operations and operator certification focal points.

Oklahoma State University is also conducting concurrent research investigating the visibility of small UA to general aviation pilots under visual meteorological conditions (VMC) [13]. ADS-B was utilized by the safety pilot to monitor the target UAS position with respect to the general aviation aircraft. The experimental flights were executed with a general aviation aircraft on an intercept course with small UA vertically separated (fixed-wing and multi-rotor) from the manned aircraft. The general aviation aircraft pilot was made acutely aware of the presence of small UAS and asked to detect the UA along the controlled intercept course. Even with warning, a 40% detection rate (48 intercept runs) was observed. In summary, the fixed-wing aircraft was easier to identify and based on distance at first contact, a manned aircraft pilot should have time to avoid a conflict based on the FAA's 12.5 second model for conflict processing. However, it is unlikely that a conflict could be avoided with a small multi-rotor. More recent tests on small rotary wing UA show a consistently low rate of detection, less than 5%. Figure 2.6 shows an image of a fixed-wing encounter from both the general aviation aircraft and ground perspective.

Figure 2.6: Small UAS, General Aviation Encounter [13]

2.2.3 Pixhawk IMU Characterization and Flight Test

The UAS flight test system architecture for this study includes Pixhawk autopilot hardware and ArduPlane flight software. Hood performed a detailed characterization of the Pixhawk during the development of an instrumentation package suitable for small UAS flight test research [14]. Critical sensors including the Pixhawk's inertial measurement unit, barometer, temperature, and airspeed sensor were tested as part of an aircraft system identification flight test campaign. Hood concluded that all critical sensors were acceptable for research use. Each individual sensor and its representative noise and error are shown in Table 2.2. Bias error is shown for sensors that were quantifiable. An interesting note also made by Hood is the difficulty of executing flight test maneuvers from a ground based, remote pilot perspective. Autonomous flight test methodology was developed for this study to increase repeatability and reduce variability in test data by removing the human pilot.

Table 2.2: Pixhawk Sensor Characterization

Sensor	**Random Noise**	**Bias Error**	**Total Error**
Accelerometer	± 0.0145 m/s^2	0.05 m/s^2	± 0.0545 m/s^2
Gyroscope	± 0.0384 °/s	0.005 °/s	± 0.223 °/s
ADC	± 0.00115 V	N/A	± 0.00115 V
Barometric Sensor, Temperature	± 0.0216 °C	N/A	± 0.0388 °C
Barometric Sensor, Pressure	± 0.0760 hPa	2.49 hPa	± 1.32 hPa
Airspeed, IAS	± 0.0483 m/s	N/A	± 0.318 m/s
Outside Air Temperature	± 0.156 °C	2.99 °C	± 1.65 °C

CHAPTER 3

Methodology

3.1 Autopilot Architecture and Ecosystem

UAS autopilot technology over the past decade has rapidly progressed due to hardware size, weight, and power (SWAP) reduction in two critical areas—processing and sensors. Microprocessors the size of coinage are now capable of executing real time navigation code. However, the enabler of *small* UAS autopilot technology is undoubtedly the decrease in the SWAP of micro-electromechanical system (MEMS) sensors. Accelerometers, gyroscopes, barometers, and magnetometers that make up the inertial measurement unit (IMU) are now both cost effective and physically acceptable for integration in UAS with an operating weight under 20 pounds. A decade ago UAS were limited by the SWAP of the IMU, which translated directly back to cost. Autopilot solutions can be separated into three general categories: military, industrial, and consumer grade. Today there are numerous non-military, commercially available autopilot solutions. There are several open source options that are very conducive to quick modification and customization.

Most commercially available solutions use a MEMS based IMU coupled with a global navigation satellite system (GNSS) receiver to estimate the aircraft state. For example, the open source ArduPilot software executes a full inertial navigation system with GNSS and IMU measurements fused into an extended Kalman filter with the appropriate external sensors. Fifteen years ago, GNSS augmented inertial navigation existed only on large military UAS with costs over six figures. The aforementioned capability can be applied to most any ground or air vehicle for under $500 USD using

open source hardware and software architecture, e.g. Pixhawk and ArduPlane, respectively. A generalized UAS architecture block diagram is shown in Figure 3.1. The three path diagram blocks, also referred to as the primary navigation subroutine, can be executed entirely on-board the vehicle with no operator interaction or computed off-line and sent remotely to the vehicle while in flight. The most common configuration is an off-line graphical user interface (GUI) path planner and manager software that an operator manipulates to send commands remotely to the on-board navigation subroutine—the path follower. The navigation subroutine executes along with the state estimator and control loops on the same microprocessor. In this configuration, the UAS flies solely based on given commands and restrictions. It is somewhat of a misnomer to imply a UAS is completely autonomous because most do not possess the ability to sense and avoid obstacles such as terrain, buildings, or other aircraft. There are primitive failsafe systems and local look ahead terrain databases, but small UAS still lack the robustness of the "human sensor."

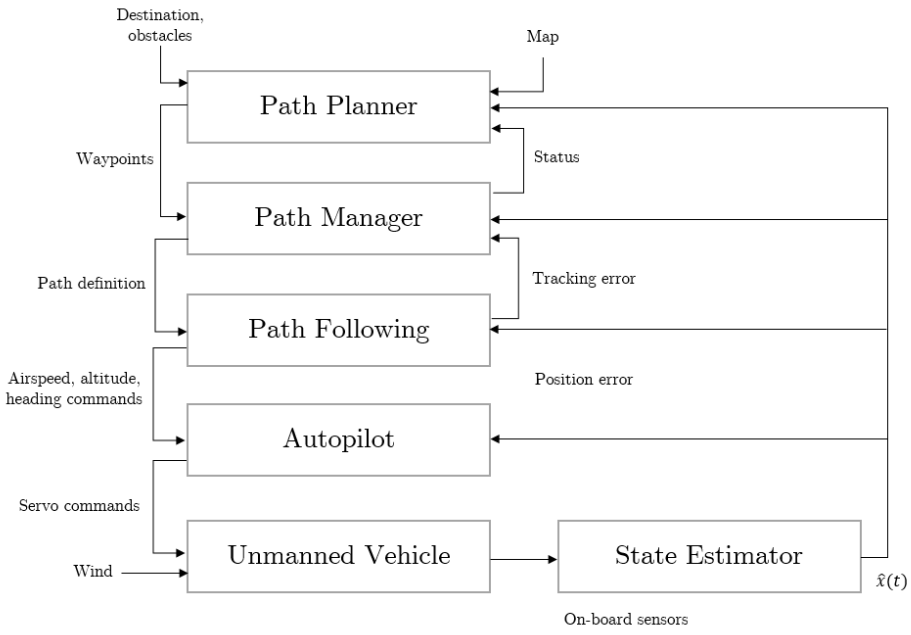

Figure 3.1: General UAS architecture [15]

3.1.1 ArduPilot

For the purpose of this study, ArduPilot provides all of the necessary flight features with the ability to examine the underlying programming [16, 17]. It is an open source platform that was created in 2007 for use on fixed-wing aircraft, multi-rotors, traditional helicopters, and ground vehicles. The original code was written for Arduino based hardware using the Arduino programming language, hence the Ardu prefix. The current releases are written in C++ and the project is officially called ArduPilot and APM: Plane, Copter, or Rover distinguish between the different types of vehicles. The APM source code is incorporated onto a hardware abstraction layer (HAL), a driver, that communicates between the software and hardware. The HAL structure allows users a flexible software solution that can be ported to many different hardware options. There are currently 16 different hardware platforms supported by the ArduPilot development team, although anyone can add support for custom hardware via the HAL.

3.1.2 PX4 Pixhawk

The Pixhawk v1 is the hardware platform of choice for this study as it has been throughly tested for thousands of hours by the open source community. The hardware design was developed by the PX4 team who also produce their own flight stack software similar to ArduPilot. The Pixhawk was manufactured by 3D Robotics through a licensing agreement with the PX4 project. 3D Robotics has ended their agreement with the PX4 project and the developers are now preparing the Pixhawk v2 for manufacture with a different partner at the time of this study.

The Pixhawk v1 consists of two separate hardware components–the flight management unit or FMU and the input/output (IO) module. The IO module controls all of the incoming and outgoing commands to the flight vehicle in addition to managing power input and output, failsafe processes, servo outputs, and sensor inputs. The

IO module is mated to the FMU which houses the main microprocessor and IMU. The IMU has redundant accelerometers and gyroscopes. The FMU also includes a separate barometer and magnetometer. The specifications of the Pixhawk v1 are shown in Table 3.1. The general peripheral layout is shown in Figure 3.2. Figure 3.2 is illustrated for a multi-rotor vehicle, but the general layout is functionally identical to a fixed-wing aircraft. Instead of servo rail outputs to multiple motors, there are separate servo outputs for the various fixed-wing control surfaces in addition to the propulsion motor. Relevant component layouts for the flight test UAS will be discussed in Section 3.3.

Table 3.1: Pixhawk v1 general specifications

Processor
Primary 32-bit 168 Mhz ARM Cortex M4 with floating point unit
Failsafe 32-bit 24 Mhz ARM Cortex M3
256 kilobytes RAM, 2 MB flash storage
Sensors
Primary 16-bit 3D MEMS MPU6000 accelerometer and gyroscope
Failsafe 16-bit 3D MEMS L3DG20 gyroscope
Internal 14-bit 3D MEMS accelerometer and compass (magnetometer)
Internal MEMS MS5611 barometer
Power
Diode controller with automatic failover
Servo rail, 7 volt high power and high current ready
Outputs over current protected, all inputs ESC protected
Interfaces
5x UART serial ports, 1 high power capable, 2x with hardware flow control
Spektrum DSM/DSM2/DSM-X Satellite input
Futaba S.BUS input
PPM sum signal
RSSI (PWM or voltage) input
I2C, SPI, 2x CAN, USB
3.3 and 6.6 ADC inputs
Dimensions
Weight 1.3 oz (38 g)
Width 2 inches (50 mm)
Height 0.6 inches (15.5 mm)
Length 3.2 inches (81.5 mm)

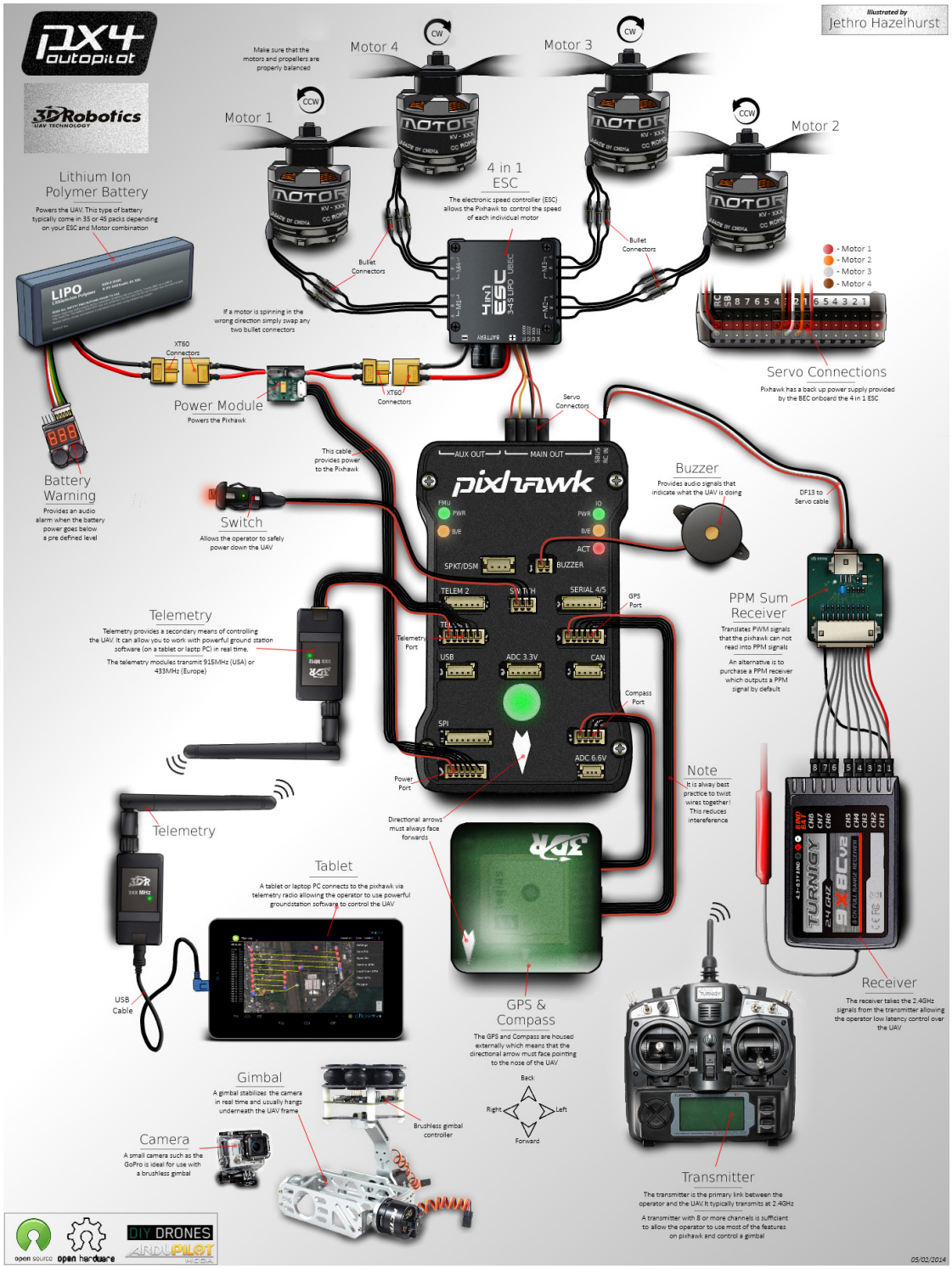

Figure 3.2: General Pixhawk v1 component info graphic [18]

3.2 Flight Test Technique

3.2.1 Uncertainty and Error

Traditional manned aircraft flight testing is a long, arduous process that contributes significantly to a program's overall schedule and budget mainly due to the great lengths taken to minimize uncertainty in test data. Error is defined by the standard notion of a measured quantity's deviation from the perceived true input, or value. The definition of experimental uncertainty may also be taken as the *possible value* the error may have–analogous to the error magnitude. Uncertainty and error magnitude are used interchangeably, but uncertainty is separate from the general definition of error as stated above. Unfortunately, aircraft flight testing encompasses nearly all potential error sources relating to experimental methods. This realization makes the task of determining the uncertainty much more challenging as the various error sources are difficult to isolate.

Modern flight test programs primarily utilize sophisticated data acquisition systems and rely less on hand recording techniques. One of the more common hand recording errors was due to indirect viewing perspective of a flight instrument, sometimes called parallax error. This has mostly been eliminated in modern glass avionics cockpit and does not necessarily apply to small UAS either as the ground station operator has digital displays. The embedded Pixhawk data acquisition system used for this study is a digital device. Analog recording of dynamic responses like aircraft flight are preferred, but even an analog sensor today has its output digitized by a data acquisition system (DAS). There are analog to digital converters used on the Pixhawk for external sensors that will be discussed in later sections. However, the advantages of digital data acquisition far outweigh the disadvantages as the primary error due to discretization of signals can be reduced by higher frequency data rates with modest microprocessors. Also, most small UAS have some portion of active electromagnetic

interference (EMI) unless specifically characterized and shielded accordingly. Digital equipment is preferred for such a scenario. Oklahoma State University has developed several small UAS flight test data acquisition systems; more recently a custom Pixhawk solution as mentioned previously. It was concluded that the critical sensors used by the Pixhawk are suitable for small UAS flight test research [14].

Significant sources of other error include systematic error from instrumentation, random error from atmospheric conditions and pilot technique, gross blunder error due to data recording technique, haphazard test planning, or pilot violation of established techniques. Systematic error of a sensor can usually be accounted for by closely characterizing bias and hysteresis. This characterization is relatively simple to complete in a laboratory setting, but the physics of flight often complicate the matter. Most instrumentation is sensitive to position error on the aircraft itself. For example, the standard pitot-static system.

The perceived accuracy of a pitot-static system is greatly influenced by the location of the static pressure port which is used in conjunction with the total pressure port to determine dynamic pressure during flight. But, the location of the static port is often determined by some sort of wind tunnel pressure distribution test—subject to all the same uncertainty. Another, slightly different example is the placement of an IMU. If the IMU is not mounted close enough to the center of gravity, the readings are prone to error in each principal axis due to the offset distance. How is the acceptable offset threshold determined? Usually by more tests.

There are also physical variations and uncertainties introduced due to the various aircraft subsystems (propulsion or flight control). The tolerances are further exacerbated in small UAS, mainly in an effort reduce cost. Manufacturing inconsistency is common for small internal combustion engines along with installation factors. As a result, many small UA propulsion systems are over sized to account for these losses. Unless significant effort (cost) is dedicated to ensuring servo torque, linkage geometry,

and control surface hinging remain consistent it is difficult to provide a single, global autopilot gain set for a common type of UA. Instead, a conservative autopilot gain is used across a fleet and UA must be individually tuned if additional responsiveness is required.

Uncertainty due to pilot technique is the primary reason flight test is schedule and budget intensive. Coupled with random atmospheric effects, and performance becomes difficult to predict. From the United States Air Force (USAF) and United States Navy (USN) flight test manuals, "It is neither possible nor practical to make exact predictions or corrections of takeoff and landing data. It is only possible to estimate the approximate capabilities of an aircraft within broad limits. Individual pilot technique is probably the factor causing the greatest variation in takeoff [and landing] data. It cannot be quantified and mathematical corrections are impossible."

As this study is primarily focused on the takeoff and landing portions of flight, the term "experiment" is avoided because the outcome is not a response with a given confidence interval. Instead, rather broad operational guidelines will be developed for small UAS based on flight test *observations.* The nominal approach involves the combination of minimizing systematic uncertainty as much as possible with a series of common sense criteria: consistency, theory, and correlation. This study is based on traditional flight test techniques from several established references including the USAF and USN flight test manuals, flight test engineering, and experimental methods texts [19, 20, 21, 22]. However, small UAS present a unique platform where traditional flight test techniques can be evaluated, adapted, or discarded.

3.2.2 Taking the Human Out of the Loop

Small UAS pilot technique is more variable when compared to manned aircraft pilot technique. The most challenging aspect, and an obvious difference between the two, is analogous to "parallax" error mentioned in the previous section. A UAS pilot is not

physically controlling the vehicle from the body frame of reference perspective, but rather a stationary Earth, or inertial, frame of reference. The parallax error can be compensated for by adding an on-board heads up display (HUD) video stream that allows the pilot to control the vehicle from the body frame of reference perspective, but even gimball stabilized full motion video does not compare to manned aircraft situational awareness. NASA's Ikhana (Predator-B/Reaper) UAS research test pilot, Mark Pestana [23], aptly summarizes the experience of piloting UAS, "[It's like trying to fly] with only ONE of my five-senses. [The] view lacks three dimensions, depth perception, and peripheral vision. In essence, the pilot has ONE eye, looking down a pipe, allowing just a 30-degree field of view!"

Consider an example landing approach of a small UAS starting at a 200 foot final approach fix altitude. The pilot must capture the final approach fix altitude, maintain the required glide slope (typically 10° or less; manned aircraft usually fly a 3° glide slope), centerline heading, and speed until the flare point all while controlling the vehicle from a stationary 800-1000 foot lateral separation point at the top of the approach profile—usually under the distress of random wind gusts. This is the necessary precision for any semblance of consistency during flight test, which is a core objective of the study. Table 3.2 shows some of the typical factors influencing takeoff and landing performance data. It is important to note that the parameters in Table 3.2 are all under the discretion of the pilot. Uncertainty cannot be eliminated in this study, however it is possible to reduce uncertainty of factors that *most* significantly affect takeoff and landing performance. Detailed autopilot operational procedures were developed to transition the human operator to the backup, safety pilot role.

3.2.3 Command and Control, Auto Land Traffic Pattern Script

Mission Planner is the primary ground control software for the ArduPilot flight stack. It supports each development of ArduPilot: Plane, Copter, and Rover. Mission Plan-

Table 3.2: Takeoff and Landing Pilot Uncertainties

Takeoff	**Approach and Landing**
Nose wheel steering/rudder deflection	Power handling
Number & amplitude of directional control inputs	Altitude of flare initiation
Aileron & elevator position during acceleration	Rate of rotation in flare
Airspeed at rotation	Length of hold-off time
Pitch rate during rotation	Touchdown speed
Angle of attack at liftoff	Rate of braking

ner is open source, developed in C#, and runs only on the Windows operating system. ArduPilot uses the MAVLink protocol for command and control of the vehicle. Telemetry, configuration parameters, and navigation fixes can be transmitted real time during flight. MAVLink is also an open source protocol and thus there are several options for ground control software that can interface with ArduPilot. Examples include QGroundControl, APM Planner, DroidPlanner, and MAVProxy. Mission Planner is preferred for ArduPlane applications because it not only combines the real time command and control of the vehicle, but is the most mature solution for setup and configuration of ArduPlane augmented fixed wing UAS. A high level Mission Planner/MAVLink architecture is shown in Figure 3.3. The user interacts with the UAS through four primary displays within Mission Planner.

The flight data screen (example shown in Figure 3.4) overlays all current vehicle status and information on the electronic flight instrumentation system and heads up display. The flight data screen also allows the user to command single direct fly-to waypoints, flight modes, and mission segments. The flight plan page is an interface for real time creation and editing of navigation fixes. Different actions can also be defined such as initiating takeoff or landing sequences, which will be discussed further. Vehicle calibration and firmware installation is managed through the setup page with detailed autopilot parameters configured within the tuning page. The Mission Planner developer site and documentation describes features of the software [25].

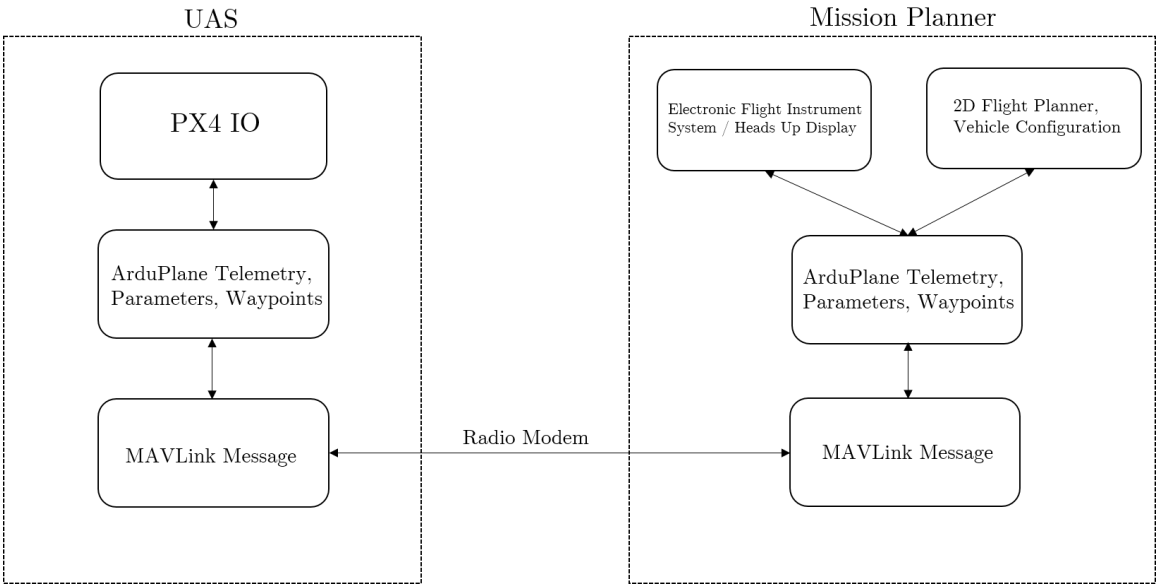

Figure 3.3: Mission Planner Architecture, Adapted from [24]

Figure 3.4: Mission Planner Flight Data Overview

Mission Planner's public release is a versatile, powerful application for small UAS, but is designed to be fairly broad in function. However, since the software is open source, functionality can be added to further enhance the command and control of a small UAS. To facilitate the rapid development of additional features, Mission Planner supports Python scripting and interfacing via an internal implementation of IronPython. IronPython is designed to integrate tightly with .NET programming languages, including C#. IronPython supports most Python 2.7 libraries. At the time of this study, the Mission Planner public release lacks real time point, click autonomous approach and landing. There are commercial developments of Mission Planner that implement this type of capability, but are configured to support a specific platform [26]. Referring to Figure 3.1 again, these features are part of the top two blocks of a small UAS architecture–path planning and path management. An autonomous traffic pattern and landing script was developed to quickly allow the flight test engineer to generate consistent landing approaches under varying flight conditions. Because the Python script is modular and integrated with the path planner and manager, it can be used for any fixed wing platform with ArduPlane executing the path following.

Several Mission Planner classes are exposed for direct interaction using Python variables. These include `Script, CurrentState`, and `MAVLink` located within their respective C# files, `Script.cs, CurrentState.cs, and MAVLink.cs`. The current release of Mission Planner can be located under the ArduPilot repository on GitHub [27]. The relevant excerpt of the Mission Planner class, `Script`, that generates the Python variables is shown in Listing 1. A customized version of Mission Planner can be recompiled to add more classes for interaction with the Python scripting engine. For example, the syntax would be similar to lines 19-23 in Listing 1, `scope.SetVariable(PythonVariableName, MissionPlannerClassInstance)`. However, modifying Mission Planner source code and then recompiling the program for

use is much more complicated when compared to writing a Python script for the same functionality. There are some dependencies that will be discussed, but installation and use of the traffic pattern and landing script is relatively straight forward.

```
1           public Script(bool redirectOutput = false)
2           {
3               Dictionary<string, object> options = new
                ↪   Dictionary<string, object>();
4               options["Debug"] = true;
5
6               if (engine != null)
7                   engine.Runtime.Shutdown();
8
9               engine = Python.CreateEngine(options);
10
11              var paths = engine.GetSearchPaths();
12              paths.Add(Settings.GetRunningDirectory() + "Lib.zip");
13              engine.SetSearchPaths(paths);
14
15              scope = engine.CreateScope();
16
17              var all =
                ↪   System.Reflection.Assembly.GetExecutingAssembly();
18              engine.Runtime.LoadAssembly(all);
19              scope.SetVariable("MAV", MainV2.comPort);
20              scope.SetVariable("cs", MainV2.comPort.MAV.cs);
21              scope.SetVariable("Script", this);
22              scope.SetVariable("mavutil", this);
23              scope.SetVariable("Joystick", MainV2.joystick);
24
25              engine.CreateScriptSourceFromString("print 'hello world
                ↪   from python'").Execute(scope);
26              engine.CreateScriptSourceFromString("print
                ↪   cs.roll").Execute(scope);
27          ...}
```

Listing 1: Mission Planner C# Script Class Code Excerpt

The traffic pattern used for the auto land script is similar to a standard manned aircraft traffic pattern as shown in Figure 3.5. Four legs of the maneuver are programmed: downwind, base, final, short final. Each leg is assigned a distance based on the aircraft type and required glide slope. The approach is generated after the script estimates the wind direction and velocity. The user selects the touch down point or landing zone. Short final, final, base, and the downwind legs are generated and populated on the mission flight plan screen in reverse succession starting from the landing zone. The latitude of the downwind leg is extracted from the user's desired landing zone coordinate latitude in degrees. The script also generates a real time summary of relevant approach data and has logic to re-check an existing approach for wind

direction and velocity. If the wind direction has changed significantly, the user will be prompted to select another landing zone and the pattern will be adjusted for the new wind direction. In addition to confirming the wind velocity and direction remain suitable for the planned approach, the script detects a timely approach in progress and will not prompt the user to re-select the landing zone and instead will proceed directly into the logging mode.

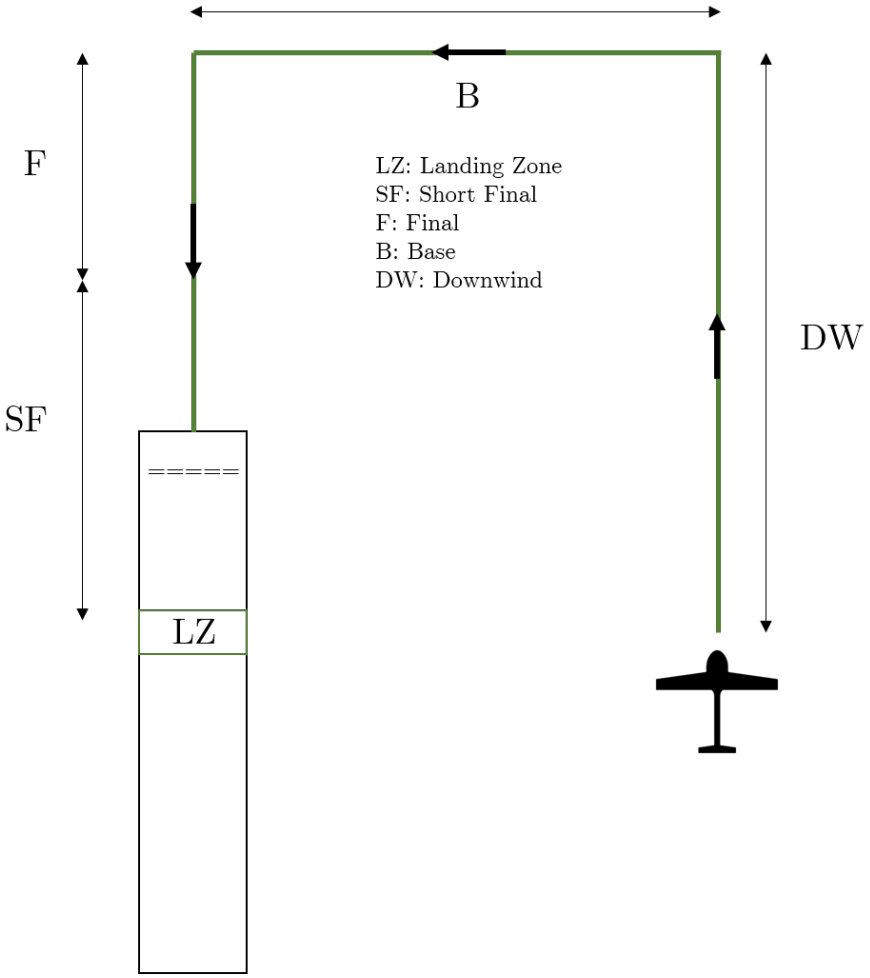

Figure 3.5: Nominal Left Hand Traffic Pattern

A state flow diagram of the auto land traffic pattern script is shown in Figure 3.6. As the script initializes, all required Python modules are loaded. The only external Python library that is required is Py AutoHotKey. PyAHK is used to automate tasks that cannot be directly programmed using publicly declared classes of

Mission Planner. PyAHK can be installed to a Python 2.7 distribution by entering `pip install pyahk` into the Python command line. Continuum Analytics Python 2.7 distribution, Anaconda, is recommended [28]. To utilize PyAHK, a specialized AutoHotKey .dll needs to be in the same folder as the PyAHK Python library. The appropriate .dll depends on the system executing the script. Both the 32 bit and 64 bit AutoHotKey H .dll versions have been tested with success [29]. The script was primarily developed and tested on a 64 bit machine with actual field use on a 32 bit ground control station laptop. The core features of the auto land traffic pattern script are written using Python functions that are called during the execution of the main conditional sections of the script. The functions are the boxed portions of Figure 3.6. Python is a scripting language, but by following simple programming best practices with functions the code retains modularity and can be updated to include additional features as necessary. The non function section of the script only amounts to approximately 50 lines of code.

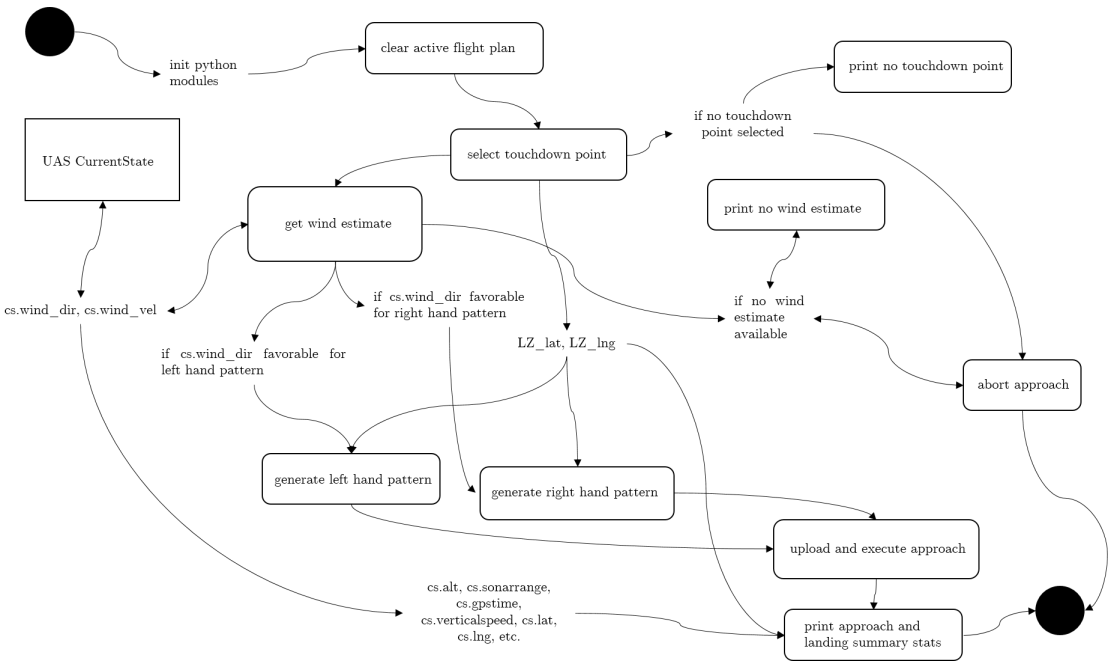

Figure 3.6: Traffic Pattern Landing Script State Flow

The conditional sections of the script will be briefly described in the following listings. Listing 2 shows the first section of the "main" logic portion of the script. Line 4 assigns the variable, `dir_path`, to the current working directory where the script is executed. Next, the wind estimation function is called and the current working directory is checked for any existing approach summary logs. Lines 9-12 initialize a count variable and loop through each file extension in the current working directory. If any files are found with a .txt extension, the count is increased. Listing 2 and 3 are the primary conditional sections of the script. Lines 4-6 of Listing 3 search for the most recent log file to load into a memory mapped file object using the Python library, `mmap`. The memory mapped file object is useful because it behaves like a string in Python. The most recent log file contents are mapped to a string in line 6 of Listing 3 and a trivial subset character search is performed immediately after in lines 9 and 10.

```
1  #starting script
2
3  #getting file path for summary text file
    ↪  output
4  dir_path =
    ↪  os.path.dirname(os.path.realpath(__file__))
5
6  get_wind_estimate()
7
8  #checking for existing logs, counting number
    ↪  of .txt log files
9  log_count = 0
10 for file in os.listdir(dir_path):
11     if file.endswith(".txt"):
12             log_count = log_count + 1
13
14 print "Log Count: %d" % log_count
```

Listing 2: Auto Land Traffic Pattern Script "Main", Log Check

The purpose of the log file string match is to determine the most recent type of approach, a standard left or right hand traffic pattern. The conditional statements in lines 11 and 17 of Listing 3 check the output of the wind estimate function for either a left or right hand traffic pattern, respectively. If the most recent logged approach pattern remains favorable with respect to current wind conditions, the script proceeds

into logging mode and uses the existing approach. IronPython's implementation in Mission Planner does not have a command prompt and thus all scripts execute identically from run to run. Listings 2 and 3 highlight logic that the flight test engineer can manipulate to some degree. By moving log files out of the current working directory the script will run a full traffic pattern generation process—this is interpreted as either the first approach of the testing session, or that the landing zone needs to be changed.

```
#checking current approach for pattern validity
if log_count >= 1:
        print 'Timely autoland approaches detected, checking latest traffic pattern for wind correction'
        current_approach_log = max(glob.iglob(dir_path + "\*.txt"), key=os.path.getctime)
        current_approach_log_file = open(current_approach_log, 'r')
        s = mmap.mmap(current_approach_log_file.fileno(), 0, access=mmap.ACCESS_READ)
        print current_approach_log

        match1 = s.find('Left')
        match2 = s.find('Right')
        if match1 != -1 and traffic_pattern_flag == 0:
                print 'No wind correction needed, proceeding into approach logging'
                LZ_lat = MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lat
                LZ_lng = MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lng
                approach_summary(LZ_lat,LZ_lng)
                print 'done baby done'
        elif match2 != -1 and traffic_pattern_flag == 1:
                print 'No wind correction needed, proceeding into approach logging'
                LZ_lat = MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lat
                LZ_lng = MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lng
                approach_summary(LZ_lat,LZ_lng)
                print 'done baby done'
        else:
                print 'Traffic pattern update needed for changing wind conditions, standby for LZ selection'
                select_LZ(traffic_pattern_flag)
                print 'done baby done'
else:
        print 'No timely autoland approaches detected, standby for traffic pattern generation'
        select_LZ(traffic_pattern_flag)
        print 'done baby done'
```

Listing 3: Auto Land Traffic Pattern Script "Main", conditional logic

The first function that is called in the conditional section of the script is the wind estimate function, `get_wind_estimate`, shown in Listing 4. As described above, Mission Planner exposes several classes for direct interaction in Python. The Mission Planner class, `CurrentState.cs`, allows for any parameter reported by the status subpage on the main flight data screen (see Figure 3.7) to be accessed with the Python variable, `cs`. Three variables of interest—`cs.wind_dir, cs.wind_vel,` and `cs.airspeed` are initialized in lines 2-4. The function looks for a valid output with a "not a number" check on line 5. If the check passes, the script records ten seconds of the wind velocity and heading data reported to the ground control station by the autopilot in lines 8-12. An average of the ten second data download is performed and the conditional checks begin on line 19. The conditional statements assign a traffic pattern flag, or approach type according to the reported wind heading. The script avoids tail wind approach scenarios, but defaults to a left hand traffic pattern with a large cross wind component.

```
def get_wind_estimate():
        wind_dir = []
        wind_vel = []
        arspd = []
        if math.isnan(cs.wind_vel) or
         ↪  math.isnan(cs.wind_dir) == True:
                print 'No wind estimate available from
                 ↪  vehicle'
        else:
                t_end = time.time() + 10
                while time.time() < t_end:
                        wind_dir.append(cs.wind_dir)
                        wind_vel.append(cs.wind_vel)
                        arspd.append(cs.airspeed)

        wind_dir_est = sum(wind_dir)/len(wind_dir)
        wind_vel_est = sum(wind_vel)/len(wind_vel)
        arspd_avg = sum(arspd)/len(arspd)
        wind_dir_est_integer = int(wind_dir_est)

        if wind_dir_est_integer in range(123,236):
                print 'Left hand traffic pattern
                 ↪  recommended'
                traffic_pattern_flag = 0 #flag as left
                 ↪  hand pattern
                pattern = "Left"
        elif wind_dir_est_integer in range(303,360) or
         ↪  wind_dir_est_integer in range (0,65):
                print 'Right hand traffic pattern
                 ↪  recommended'
                traffic_pattern_flag = 1 #flag as right
                 ↪  hand pattern
                pattern = "Right"
        else:
                print 'Help me Tom Cruise'
                traffic_pattern_flag = 0 #default to
                 ↪  left
                pattern = "Left"
        print "Wind Direction Estimate [deg]: %d" %
         ↪  wind_dir_est
        Script.Sleep(1000)
        print "Wind Velocity Estimate [kts]: %d" %
         ↪  wind_vel_est
        Script.Sleep(1000)
        print "Average Airspeed Estimate [kts]: %d" %
         ↪  arspd_avg
        return traffic_pattern_flag, pattern
```

Listing 4: Auto Land Traffic Pattern Script, `get_wind_estimate` Function

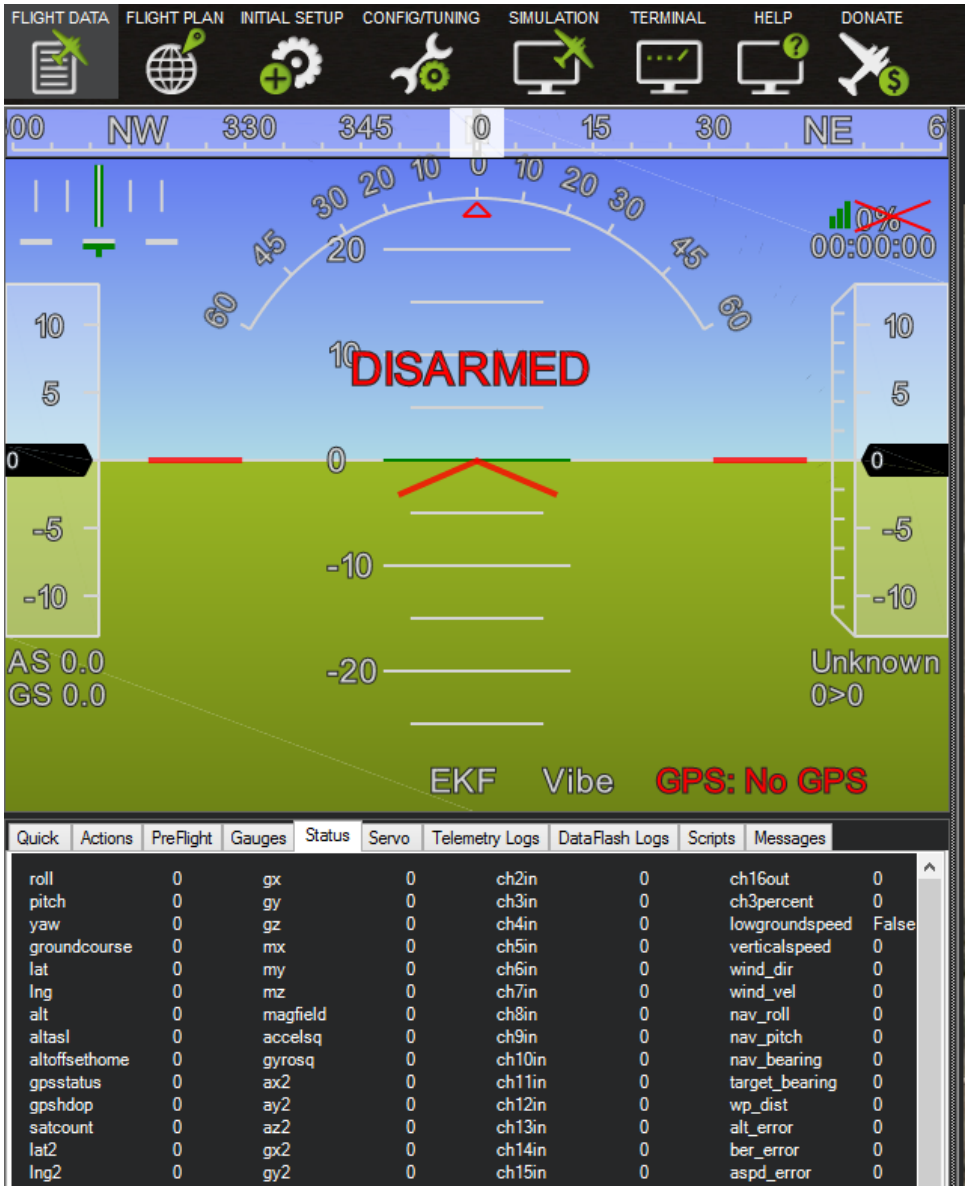

Figure 3.7: Mission Planner Status Page

ArduPlane estimates the real time, two dimensional wind vector using airspeed, GNSS, and IMU measurements. The estimate is adequate for use in the auto land traffic pattern script because pattern direction is based on conservative wind direction sectors. The wind vector is also recorded during the approach to characterize performance with respect to the wind conditions. During automatic approach and landing the 2D wind vector estimate is used to compensate for crosswind slide slip angle and headwind airspeed adjustments. The 2D wind vector estimate is also fused each time step by the Extended Kalman filter. The aircraft's inertial velocity vector, $\overline{V}^{ned}$, is the vector sum of the relative airspeed vector, $\overline{V}^{ned}_{air}$, and the wind vector, $\overline{V}^{ned}_{w}$, as shown in Figure 3.8. The wind estimate in ArduPlane is three lines of code shown first in Equations 3.1, 3.2, and 3.3.

$$|V^{ne}_{w}| = \sqrt{V^2_n + V^2_e} - |V^{ne}_{air,pitot}| \tag{3.1}$$

$$V^{n}_{w} = |V^{ne}_{w}| \cdot \cos\psi \tag{3.2}$$

$$V^{e}_{w} = |V^{ne}_{w}| \cdot \sin\psi \tag{3.3}$$

The C++ implementation is shown in Listing 5, lines 6-8, respectively. Source code can be found on the ArduPilot GitHub repository [17]. The magnitude of the wind vector is inertial GNSS velocity magnitude subtracted from axial airspeed. Axial airspeed is measured using a pitot-static system. Scalar quantities of the wind vector are then calculated using the Euler yaw angle, ψ. ArduPlane converts the attitude quaternion to Euler angles for this operation. The aircraft must be changing attitude with the inertial GNSS velocity also changing. ArduPlane's method of wind estimate is rudimentary and is referenced as direct estimation in literature [30, 31]. However,

other methods for estimating the wind vector have been proposed, including a fusion of the direct method and predictive methods [32].

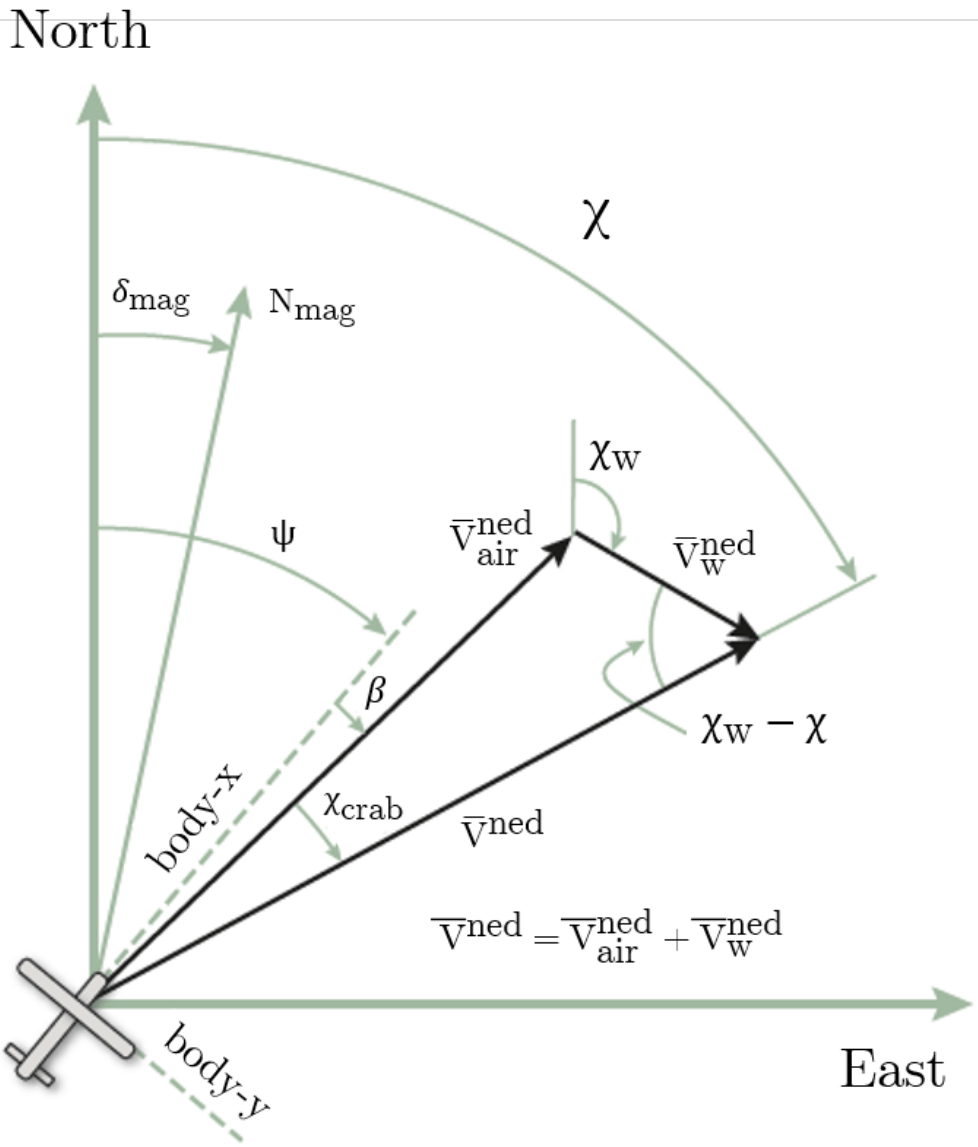

Figure 3.8: Horizontal Plane (North-East) Angular and Vector Relationships [31]

```
if (yawAlignComplete && useAirspeed()) {
// if we have airspeed and a valid heading, set the wind states to the reciprocal of the vehicle heading
// which assumes the vehicle has launched into the wind
  Vector3f tempEuler;
  stateStruct.quat.to_euler(tempEuler.x, tempEuler.y, tempEuler.z);
  float windSpeed =  sqrtf(sq(stateStruct.velocity.x) + sq(stateStruct.velocity.y)) - tasDataDelayed.tas;
  stateStruct.wind_vel.x = windSpeed * cosf(tempEuler.z);
  stateStruct.wind_vel.y = windSpeed * sinf(tempEuler.z);
// set the wind sate variances to the measurement uncertainty
  for (uint8_t index=22; index<=23; index++) {
        P[index][index] =
         sq(constrain_float(frontend->_easNoise, 0.5f, 5.0f)
         * constrain_float(_ahrs->get_EAS2TAS(), 0.9f,
         10.0f));
  }
}
```

Listing 5: ArduPlane EKF2 Wind State Observation

The direction and magnitude of the 2D wind vector are sent to Mission Planner via MAVLink Telemetry message as `cs.wind_dir` and `cs.wind_vel`, respectively. Magnitude was shown previously in Equation 3.1 and Listing 5. The subset of code in Listing 5 is called by the Attitude and Heading Reference System (AHRS) during each predictor, update time step of the Kalman filter. Wind heading, χ_w, is expressed as the inverse tangent of the scalar components of the wind vector in Equation 3.4. Wind heading as reported to Mission Planner, `cs.wind_dir`, is shown implemented in the GCS library of ArduPlane (see Listing 6).

$$\chi_w = \arctan\left(\frac{V_w^e}{V_w^n}\right) \tag{3.4}$$

```
void Plane::send_wind(mavlink_channel_t chan)
{
    Vector3f wind = ahrs.wind_estimate();
    mavlink_msg_wind_send(
        chan,
        degrees(atan2f(-wind.y, -wind.x)), // use negative, to give
                                           // direction wind is coming from
        wind.length(),
        wind.z);
}
```

Listing 6: ArduPlane GCS MAVLink Wind Estimate Output

Returning to Listing 3, after the wind estimate function is used to determine if the approach pattern needs to be adjusted for current wind conditions, the script goes into logging mode or generates a new approach. Before the logging mode is discussed, the process for generating an approach will be covered. The function call for `select_LZ`, first shown in line 25 of Listing 3, contains the logic for generating a traffic pattern on the Mission Planner flight plan screen based on the flight test engineer's desired landing point. The function, `select_LZ`, is shown in Listing 7. Line 3 of `select_LZ` calls for a separate function, `clear_flight_plan_active`, to execute a series of AutoHotKey commands within Python to switch to the flight plan screen and clear any existing missions and waypoints.

After the flight plan is cleared, the flight test engineer has seven seconds to select the desired landing point. The conditional statement on line 8 of Listing 7 checks if the flight test engineer has selected a landing point before proceeding, if not, the script aborts on line 27. Line 8 accesses the number of waypoints that are currently populated on the Mission Planner flight plan screen. Although the statement in line 8 is not directly supported with a Python variable through the Mission Planner `Script` class, any publicly declared functionality elsewhere in the Mission Planner source code can be accessed with the correct syntax. The `FlightPlanner.cs` C# source code file contains all of the functionality for Mission Planner's flight plan screen [27]. Most of the `FlightPlanner` class is privately declared, but there are several key features that are publicly declared. If these features were not publicly declared, the C# source code would probably have to be modified to include the desired functionality.

The latitude and longitude of the landing point on the flight plan screen are accessed with the syntax `FlightPlanner.pointlist[1].Lat/Lng` and assigned to the Python variables, `LZ_lat` and `LZ_lng` in lines 10 and 11 of Listing 7. After the landing point coordinates are assigned to their respective variables, the landing point is cleared from the display with the another series of AutoHotKey Python commands in

the function, `clear_flight_plan_user_LZ`. Once the flight plan screen is cleared, the function compares the `traffic_pattern_flag` and calls for the respective approach pattern function in lines 19-22.

```
1  def select_LZ(traffic_pattern_flag):
2          #clearing active flight plan
3          clear_flight_plan_active()
4          print 'Select the LZ'
5          #user has 7 seconds to select a landing point
6          Script.Sleep(7000)
7          #make sure LZ is selected, explicitly checking if a waypoint
            ↪  was added on the flight plan map--the desired "LZ",
            ↪  reason > 1 is because home counts as 0, but does not
            ↪  appear as an entry on the datagrid
8          if
            ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist.Count
            ↪  > 1:
9                 print 'Received LZ, generating approach'
10                LZ_lat =
                   ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[1].Lat
11                LZ_lng =
                   ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[1].Lng
12                Script.Sleep(1000)
13                print "LZ Lat: %f" % LZ_lat
14                Script.Sleep(1000)
15                print "LZ Lng: %f" % LZ_lng
16                # #clearing LZ selection, prep for approach pattern
17                clear_flight_plan_user_LZ()
18                # generating traffic pattern based on wind estimate
19                if traffic_pattern_flag == 1:
20                        right_hand_traffic_pattern(LZ_lat,LZ_lng)
21                elif traffic_pattern_flag == 0:
22                        left_hand_traffic_pattern(LZ_lat,LZ_lng)
23                else:
24                        print 'Help me Tom Cruise'
25                approach_summary(LZ_lat,LZ_lng)
26           else:
27                sys.exit("LZ not selected, aborting approach")
```

Listing 7: Auto Land Traffic Pattern Script, `select_LZ` Function

There are two approach traffic pattern functions, a standard left and right hand pattern. The logic is the same between the two, but the calculation of each leg of the approach pattern is different. The left hand traffic pattern function is shown in Listing 9. The arguments for the function are the coordinates of the desired landing point. Pattern legs are generated in reverse succession starting from the desired landing point, short final, final, base, and the downwind or initial approach fix. The function, `newpos`, calculates a new set of latitude and longitude coordinates on a great circle path given a distance and bearing from the initial point. Equations 3.5 and 3.6

$$\phi_{lat,2} = \arcsin\left(\sin\phi_{lat,1} \cdot \cos\delta + \cos\phi_{lat,1} \cdot \sin\delta \cdot \cos\theta_{nav}\right) \tag{3.5}$$

$$\lambda_{lng,2} = \lambda_{lng,1} + \arctan\left(\frac{\sin\theta_{nav} \cdot \sin\delta \cdot \cos\phi_{lat,1}}{\cos\delta - \sin\phi_{lat,1} \cdot \sin\phi_{lat,2}}\right) \tag{3.6}$$

are derived from the spherical law of cosines where ϕ is latitude, λ is longitude, θ is bearing, and δ is angular distance, $\frac{D}{R}$. D is the desired distance between the two locations and R is the Earth's radius. The implementation of these formulae in Python are shown in Listing 8. Python maps floating-point calculations to IEEE-754 standard, commonly known as doubles. IEEE-754 "doubles" contain 53 bits of precision which makes Equations 3.5 and 3.6 reasonably accurate down to offset distances as small as a few meters. The smallest offset distance demanded in the nominal traffic pattern is 200 meters. As will be shown in the sample approach patterns, subjectively, floating-point error does not manifest as problematic.

```
1  def newpos(bearing,distance,lat,lng):
2          lat1 = math.radians(lat)
3          lon1 = math.radians(lng)
4          brng = math.radians(bearing)
5          dr = distance / 6378100.0 # / radius of earth in
            ↪  meters
6
7          lat2 = math.asin(math.sin(lat1) * math.cos(dr) +
            ↪  math.cos(lat1) * math.sin(dr) *
            ↪  math.cos(brng))
8          lon2 = lon1 + math.atan2(math.sin(brng) *
            ↪  math.sin(dr) * math.cos(lat1), math.cos(dr) -
            ↪  math.sin(lat1) * math.sin(lat2))
9
10         lat_out = math.degrees(lat2)
11         lng_out = math.degrees(lon2)
12         return lat_out,lng_out
```

Listing 8: Auto Land Traffic Pattern Script, `newpos` Function

The left hand traffic pattern approach legs are each generated with `newpos` in lines 7-13 of Listing 9. For example, short final on line 7 is bearing 0°, 255 meters from the desired landing point, `LZ_lat/LZ_lng`. Each successive leg is generated with a Python tuple element output from `newpos` that contains the latitude and longitude of the previous approach leg as an argument. Each approach leg is populated on the

flight plan display with the syntax, `FlightPlanner.InsertCommand` on lines 17-25. The full command with the prefix `MissionPlanner.MainV2.instance` is shortened for formatting in Listing 9. The C# method in the `FlightPlanner.cs` source code is shown in Listing 10. As mentioned previously, this method is usable in the Mission Planner Python script engine because it is declared public. The argument structure follows standard MAVLink protocol of seven parameter, command fields. A common list of MAVLink commands and command field structure for fixed wing aircraft can be found on the ArduPlane documentation website [33].

```
1  #generate the left hand traffic pattern from LZ selection
2  def left_hand_traffic_pattern(LZ_lat,LZ_lng):
3
4          # 0 - lat, 1 - lng
5          #left hand traffic pattern
6          #short final
7          short_final = newpos(0,255,LZ_lat,LZ_lng)
8          #final
9          final = newpos(0,200,short_final[0],short_final[1])
10         #base
11         base = newpos(90,200,final[0],final[1])
12         #downwind/IAF
13         downwind = newpos(180,445,base[0],base[1])
14
15         #adding to FP - distance argument is in units as selected on
            ↪  MP
16         #downwind/IAF
17         ...FlightPlanner.InsertCommand(1,MAVLink.MAV_CMD.WAYPOINT
            ↪  ,0,0,0,0,downwind[1],downwind[0],200)
18         #base
19         ...FlightPlanner.InsertCommand(2,MAVLink.MAV_CMD.WAYPOINT
            ↪  ,0,0,0,0,base[1],base[0],150)
20         #final
21         ...FlightPlanner.InsertCommand(3,MAVLink.MAV_CMD.WAYPOINT
            ↪  ,0,0,0,0,final[1],final[0],125)
22         #short final
23         ...FlightPlanner.InsertCommand(4,MAVLink.MAV_CMD.WAYPOINT
            ↪  ,0,0,0,0,short_final[1],short_final[0],80)
24         #LZ
25         ...FlightPlanner.InsertCommand(5,MAVLink.MAV_CMD.LAND
            ↪  ,50,0,0,0,LZ_lng,LZ_lat,0)
26         ...FlightPlanner.InsertCommand(6,MAVLink.MAV_CMD.CONTINUE_AND_CHANGE_ALT
            ↪  ,1,0,0,0,0,0,75)
27         ...FlightPlanner.InsertCommand(7,MAVLink.MAV_CMD.DO_JUMP
            ↪  ,1,-1,0,0,0,0,0)
```

Listing 9: Auto Land Traffic Pattern Script, `left_hand_traffic_pattern` Function

```
1  public void InsertCommand(int rowIndex, MAVLink.MAV_CMD cmd,
    ↪  double p1, double p2, double p3, double p4, double x,
    ↪  double y,
2              double z, object tag = null)
3          {
4              if (Commands.Rows.Count <= rowIndex)
5              {
6                  AddCommand(cmd, p1, p2, p3, p4, x, y, z, tag);
7                  return;
8              }
9
10             Commands.Rows.Insert(rowIndex);
11
12             this.selectedrow = rowIndex;
13
14             FillCommand(this.selectedrow, cmd, p1, p2, p3, p4,
                ↪  x, y, z, tag);
15
16             writeKML();
17         }
```

Listing 10: Mission Planner InsertCommand Method, `FlightPlanner.cs`

Four MAVLink commands are used in the traffic pattern: `MAV_CMD.WAYPOINT`, `MAV_CMD.LAND`, `MAV_CMD.CONTINUE_AND_CHANGE_ALT`, and `MAV_CMD.DO_JUMP`. In Listing 10, the first two arguments are the mission command index and the name of the MAVLink command. The remaining seven arguments, or command fields, are specific to the MAVLink command that is desired. `MAV_CMD.WAYPOINT` has four usable command fields—two, five, six, and seven. The command fields specify waypoint radius, target longitude, target latitude, and target altitude, respectively. All other fields are ignored by ArduPlane. `MAV_CMD.LAND` has three usable command fields. Field one specifies the abort altitude to climb to if the approach is waved off. Fields five and six are the target longitude and latitude of the landing point. During an approach, the flight test engineer has three options to trigger a go around or wave off once the terminal landing sequence command, `MAV_CMD.LAND` is initiated. If the manual transmitter throttle is raised above 90%, a flight mode change, or an abort command is initiated from the GCS, the wave off logic will execute. The default wave off logic does not require any pre-planning and will follow the same protocol—maximum throttle climb out at 10° pitch up to a specified target altitude, or default to 100 feet. If `MAV_CMD.LAND` is followed by a `MAV_CMD.CONTINUE_AND_CHANGE_ALT`

command, the mission index will increment to execute any additional altitude change. `MAV_CMD.CONTINUE_AND_CHANGE_ALT` can be followed by any mission command behavior, but `MAV_CMD.DO_JUMP` is used to reset the mission index to the downwind, or initial approach fix, to attempt another landing.

Once the traffic pattern is generated and populated, the flight test engineer uploads and executes the approach when ready. A sample right hand traffic pattern output is shown in Figure 3.9. The script pauses for a set time and proceeds into a logging mode. Returning to Listing 3 and the "main" conditional section of the auto land traffic pattern script, the only remaining function is `approach_summary`. The `approach_summary` function outputs .txt log files that characterize the real time performance of the approach and landing. ArduPlane and Pixhawk's on-board SD card dataflash log files are notoriously difficult to post process. In recent releases of Mission Planner, options have been added to convert dataflash logs to MATLAB .mat files which are useful for examining higher frequency IMU data. However, for general GNSS, altitude, and attitude performance; logging the real time telemetry stream is ideal. It is less time intensive than exporting dataflash logs, exporting to MATLAB, and post-processing. The real time telemetry logging also gives the flight test engineer instant feedback on the approach performance and promotes efficiency during a flight test session.

The structure of the `approach_summary` function is a series of comparator statements inside a while loop. The while loop executes continuously as long as the flight mode remains in "auto." Short final to touchdown and subsequently roll out are of interest. There are five comparator statements; four of which are associated with a discrete event during the approach. Each statement is scanning for a specific telemetry condition and generating .txt logs when satisfied. Access to relevant telemetry is done by calling the Python variable `cs.` The `approach_summary` function can be found in Appendix A as part of the full code block.

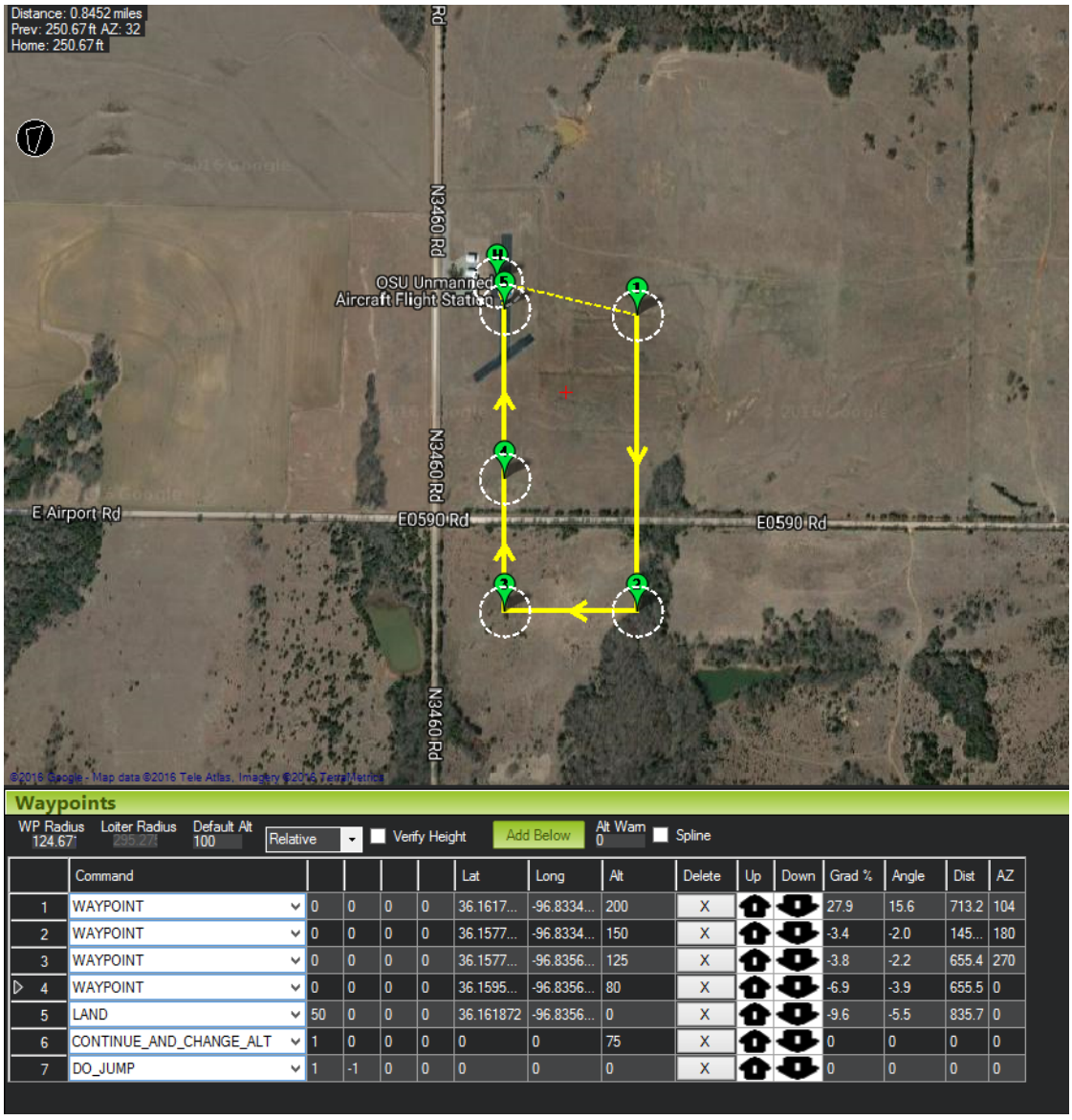

Figure 3.9: Auto Land Traffic Pattern Script Generated Right Hand Pattern

3.3 UAS Platforms and Test Overview

The platforms used for this study can be classified as either a fixed-wing or multi-rotor vehicle. The fixed-wing platforms are under 55 pounds gross takeoff weight (GTOW) and the multi-rotor vehicles do not exceed 5 pounds GTOW. Multi-rotors were included in the study because these types of vehicles have not only proliferated into the mainstream population, but also provide certain utility that can be applied to many commercial operational scenarios. Primary fixed-wing flight test was conducted using the ReadyMadeRC Anaconda—a medium sized group one UAS (DoD definition) with a GTOW less than 15 pounds. Multi-rotor flight test was performed using the 3DR Solo and DJI Mavic. The basic specifications for each platform can be found in Table 3.3 and Figure 3.10 shows the approximate relative sizes of each vehicle.

Table 3.3: UAS Platform General Specifications

Parameter	**RMRC Anaconda**	**3DR Solo**	**DJI Mavic**
Vehicle Type [F-W or M-R]	F-W	M-R	M-R
GTOW [lbs]	10	3.9	1.6
Wing Span [ft]	6	2	1
Length [ft]	5	2	1
Payload Capacity [lbs]	3	0.6	N/A
Propulsion	Electric	Electric	Electric
Autopilot	ArduPlane	ArduCopter	DJI
Max Speed [kts]	60	30	35
Loiter Speed [kts]	30	0-10	0-10
Endurance [hrs]	0.5-0.75	0.15-0.2	0.5

The vehicle subsystems for the RMRC Anaconda include structure, propulsion and power, avionics and control, and external sensors. The Anaconda has two main gear wheels fixed to a 0.25 inch thick aluminum landing strut. The steerable nose gear wheel is attached to a torsion spring strut. The primary lifting surfaces were reinforced with rectangular carbon fiber rods. The propulsion system includes an 800

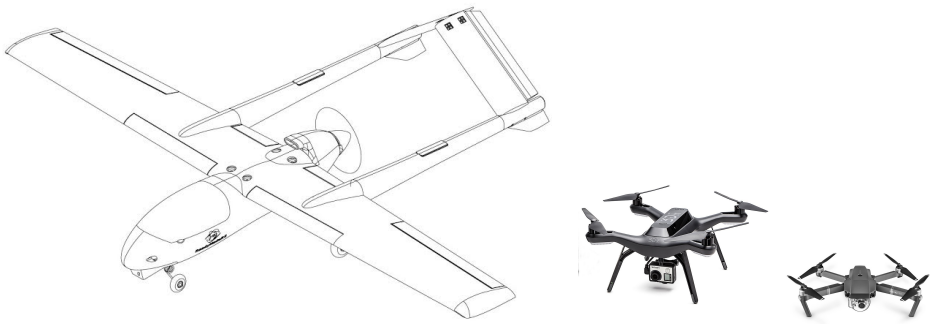

Figure 3.10: RMRC Anaconda and Multi-Rotors with Approximate Relative Sizes

kV brushless outrunner electric motor and a 15x4E pusher propeller. Two, four cell lithium polymer batteries wired in parallel provide 13,200 milli-amp hour through an 80 amp electronic speed controller. Electrical power is distributed to external sensors and flight control servos via the Pixhawk's power module and servo rail. Standard external sensors include a uBlox GPS GNSS module and a Measurement Specialties 4525DO differential pressure pitot-static system. A LightWare SF11-C laser altimeter is mounted near the main landing gear. The laser altimeter is activated only when the aircraft crosses the short final reference altitude and waypoint. Table 3.4 contains specifications for the SF11-C laser altimeter and Figure 3.11 details the installation on the fixed-wing aircraft.

Table 3.4: SF-11C Specifications

Parameter	
Weight	35 [g]
Dimensions	30 x 56.5 x 50 [mm]
Range	0.1-120 [m]
Resolution	1 [cm]
Accuracy	±0.1 [m]
Outputs	Serial, I2C, Analog
Supply Voltage	5.0 [VDC]
Supply Current	200 [mA] max
Laser Power	20 [W] peak, 15 [mW] average
Optical Aperture	51 [mm]
Beam Divergence	0.2°

The Anaconda uses the ArduPlane 3.7.1 flight stack on the PX4 Pixhawk v1 with command and control on the 915 Mhz frequency. The 3DR Solo is configured with a PX4 Pixhawk v2. PX4 firmware is Solo specific; version 1.3.1 Sensors include GPS GNSS and compass module. Power is provided by one three cell 5100 mAH battery. The Solo was in factory hardware configuration. Command and control hardware includes WiFi RC transmitter and a ground station laptop running Mission Planner. The DJI Mavic was in the factory hardware and software configurations and is equipped with several additional sensors in comparison to the 3DR Solo. Dual band GNSS (GPS/GLONASS) is standard on the Mavic with a downward facing sonar and computer vision array. DJI claims that these systems increase the horizontal accuracy to ± 1 foot when landing. 2.4 Ghz and 5.8 Ghz are used for both command and control and full motion video through the Mavic's integrated camera gimbal.

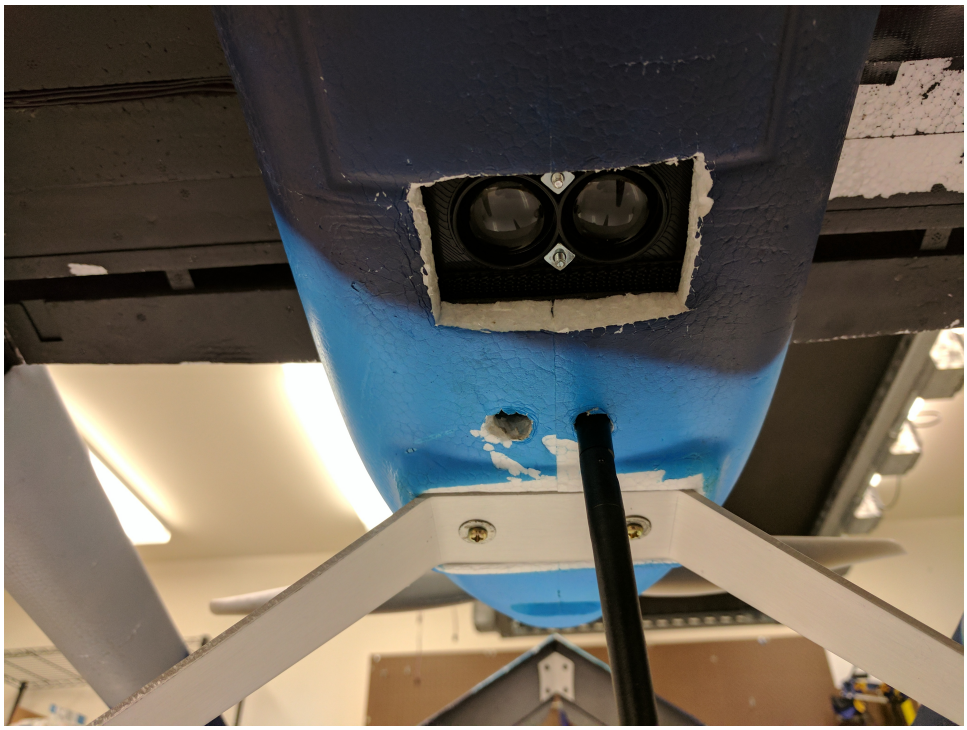

Figure 3.11: Anaconda SF11 Laser Altimeter Configuration

3.3.1 Nominal Flight Plan

Two test plans were executed for both the fixed wing and multi-rotor. The overall goal for both test plans was to characterize the performance of each vehicle's autonomous landing capability in regards to external sensor equipage. Future higher density operations scenarios will not have operators directly controlling vehicles. For example, one multi-rotor (3DR Solo) was only equipped with barometric altitude and GPS GNSS for position while the Mavic has dual band GNSS and a computer vision system (landing target in Figure 3.12). A notional multi-rotor test plan is also shown in Figure 3.12. The fixed wing aircraft test plan was to fly a standard traffic pattern based on wind conditions, as previously in Figure 3.5.

Landing definitions are adapted from the USN Test Pilot School flight test manual [20]. Landing final approach reference altitude is usually 50 feet, but for this study the altitude is increased to 80 feet. Landing performance, Figure 3.13, is broken into two phases, air phase and ground phase. The air phase, S_3, is taken to be the distance from short final at 80 feet AGL to touchdown. After touchdown, landing roll out begins. The total distance to wheel stop after touchdown is defined as the ground phase, S_4. Airspeed at the 80 foot reference altitude and touchdown are V_{80} and V_{TD}, respectively. Real time telemetry based logging of the relevant landing performance data was developed as part of a traffic pattern generation script.

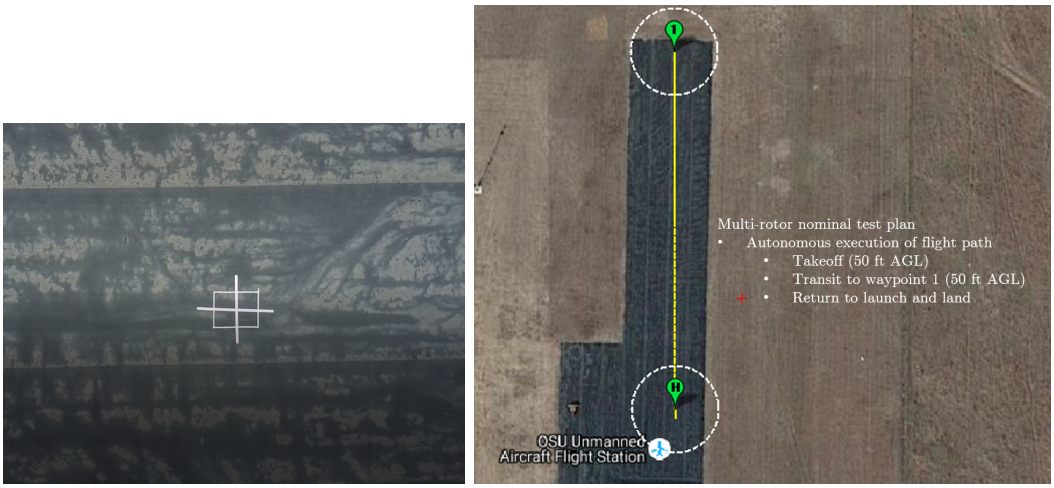

Figure 3.12: Landing Target View, DJI Mavic Camera

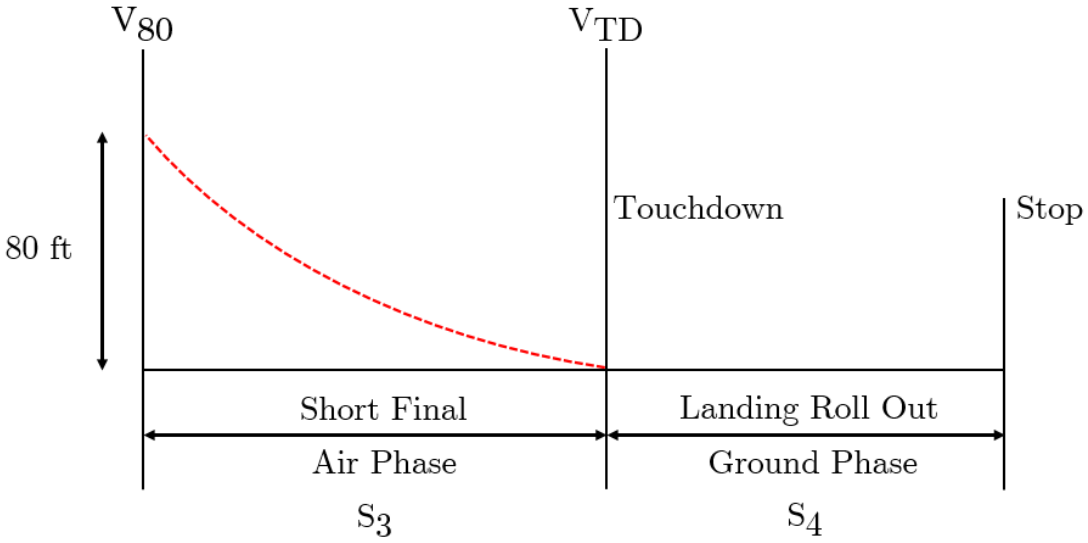

Figure 3.13: Landing Performance Definitions

3.4 Autopilot Takeoff and Landing

3.4.1 Critical Autopilot Parameters

ArduPlane fixed-wing parameters necessary for GNSS guided waypoint flight, auto takeoff and landing will be briefly discussed. GNSS augmented inertial navigation capability is critical to waypoint guided flight modes in small UAS COTS autopilot systems as the MEMS IMU does not have low enough position drift error sufficient for navigation. ArduPlane has a Extended Kalman Filter (EKF) that when combined with the MEMS IMU can only reliably provide navigation quality position estimates for 1-2 minutes if GNSS capability is completely lost. GNSS coverage degradation is common, but a complete loss can be caused by hardware failure or signal interference. ArduPlane also supports redundant GNSS hardware configurations to guard against hardware failure. Benchmark performance for current ArduPlane compatible GNSS hardware is dual band constellation capability, differential correction at ground level, with a horizontal dilution of precision (HDOP) below 1.0. Parameters in Table 3.5 are critical to GNSS flight. ARMING_CHECK initializes all pre-arm checks of all flight critical sensors including IMU, GNSS, airspeed, magnetometer, and barometer. AHRS_GPS_USE ensures that the Attitude Heading Reference System (AHRS) utilizes GNSS position estimates. EK2_ENABLE and AHRS_EKF_TYPE enable the latest version of the EKF and its use in the AHRS. The final parameter, INITIAL_MODE, boots the autopilot in a manual mode. The boot mode can be modified if configured to fly in an auto flight mode without a manual RC transmitter.

Table 3.6 lists parameters necessary for auto takeoff capability. There are no parameter values listed as they are variable depending on aircraft. ArduPlane utilizes the common Proportional, Integral, Derivative (PID) controller for flight control response. There is a separate ground steering PID controller that is active during

Table 3.5: ArduPlane GNSS Critical Parameters

Parameter	**Value**
ARMING_CHECK	1
AHRS_GPS_USE	1
EK2_ENABLE	1
AHRS_EKF_TYPE	2
INITIAL_MODE	0

auto takeoff. The first five parameters listed in Table 3.6 tune the ground steering PID gains. STEER2SRV_MINSPD is the minimum ground speed before the ground steering control loop is activated. This value is dependent on a reliable GNSS ground speed estimate. More capable GNSS hardware allows this value to be set lower and thus achieve better low speed ground handling performance during the initial stages of auto takeoff. TKOFF_THR_SLEW and TKOFF_THR_MAX set the rate at which throttle is applied and maximum available throttle during auto takeoff, respectively. Throttle slew rate should be conservative as too much throttle input quickly can cause small vehicles to oscillate and torque during initial ground roll. Maximum available throttle during takeoff is dependent on aircraft capability. Aircraft with significant excess power due to propulsion system or payload will most likely be configured to limit power or throttle in normal flight (not have 100% available). This prevents unnecessary throttle surging to maximum and increases endurance. In these scenarios, choosing a TKOFF_THR_MAX slightly higher than normal throttle limits is recommended for auto takeoff. TKOFF_ROTATE_SPD and TECS_PITCH_MAX set the initial rotate airspeed and maximum pitch angle during takeoff, respectively. These parameters are aircraft dependent. For example, the RMRC Anaconda has a large propeller in a pusher configuration. Thus, lower pitch angles are required during take off and landing to prevent propeller strikes. The final parameter, GROUND_STEER_ALT sets the altitude threshold when the ground steering loop terminates or activates.

Table 3.7 details parameters for auto landing. A well executed auto land is highly dependent on a stabilized approach, as discussed in Chapter 2. The same

Table 3.6: ArduPlane Auto Takeoff Parameters

Parameter
STEER2SRV_P
STEER2SRV_I
STEER2SRV_D
STEER2SRV_IMAX
STEER2SRV_TCONST
STEER2SRV_MINSPD
TKOFF_THR_SLEW
TKOFF_THR_MAX
TKOFF_ROTATE_SPD
TECS_PITCH_MAX
GROUND_STEER_ALT

general concepts that apply to large commercial aircraft Category III auto land systems apply to small UAS. Management of the glide path and thus airspeed, power, and pitch are critical to hitting a consistent aim point, flare, and touchdown. TECS_LAND_ARSPD and TECS_LAND_SPDWGT control the approach airspeed and error weighting with respect to airspeed and altitude. Both parameters are aircraft dependent. TECS_LAND_ARSPD should be above stall speed, but low enough that the aircraft can maintain the desired glide path to the aim point. TECS_LAND_SPDWGT manages the error priority of airspeed and altitude. For example, the default value of 1.0 places equal emphasis on maintaining airspeed and altitude targets and generally results in a stabilized approach for a variety of weather and approach conditions. A value closer to 2.0 gives airspeed priority over altitude and could be applicable for an approach close to stall speed. TECS_LAND_SPDWGT can be set to the special value of -1 for a well tuned aircraft. Error will be scaled during approach so that airspeed is maintained at top of the approach and traded for altitude closer to the aim point, if necessary, to ensure an accurate flare and touchdown. In practice, TECS_LAND_SPDWGT = -1 is highly dependent on the performance of the flight control loops and Total Energy Control System (TECS). Using TECS_LAND_SPDWGT = -1 without first tuning the aircraft for a broad

spectrum of flight conditions will result in oscillation and unstable approaches as the aircraft tries to dynamically correct deviations during approach.

Table 3.7: ArduPlane Auto Land Parameters

Parameter
TECS_LAND_ARSPD
TECS_LAND_SPDWGT
LAND_FLARE_SEC
LAND_FLARE_ALT
TECS_LAND_SINK
LAND_PITCH_CD
TECS_PITCH_MAX
LEVEL_ROLL_LIMIT
THR_MIN
TECS_LAND_DAMP
LAND_ABORT_THR
LAND_DISARMDELAY

LAND_FLARE_SEC sets the flare point as a function of vertical speed, or sink rate. For example, LAND_FLARE_SEC = 1.5 sets the flare point 1.5 seconds before impact at the current vertical speed. This parameter allows the aircraft to flare early or late depending on sink rate and achieve the desired vertical speed at touchdown, TECS_LAND_SINK. LAND_FLARE_ALT is the secondary parameter setting an altitude threshold at which the aircraft flares, regardless of vertical speed. LAND_PITCH_CD is the minimum pitch angle during flare. This is generally a smaller value, but lower than TECS_PITCH_MAX. LAND_PITCH_CD and TECS_PITCH_MAX are aircraft dependent. The RMRC Anaconda has conservative flare settings that generally result in a three point landing to prevent propeller strikes due to excessive pitch angles and hold off during flare. A configuration with no propulsion clearance limitations can flare more aggressively closer to the surface and achieve touchdown on rear main landing gear. ArduPlane also supports non conventional landing configurations such as VTOL, belly land/skid, or deep stall. LEVEL_ROLL_LIMIT locks the roll limit during flare to prevent wing strikes.

LAND_ABORT_THR is the power or throttle setting applied during a go around or aborted landing and LAND_DISARMDELAY is a timer to disarm the aircraft once ground speed reaches a certain threshold. It is important to note that for larger UAS that land conventionally with higher ground speed, the ground steering loop will be active during roll out to maintain a straight heading projected from the desired landing location. Table 3.8 details settings for adding a rangefinder. These settings are specific to the type of rangefinder used, but RNGFND_LANDING globally enables a rangefinder for use during approach and landing. Specifics for supported rangefinders and also further background on parameters discussed in this section can be found within ArduPlane documentation [34].

Table 3.8: ArduPlane Rangefinder Parameters

Parameter
RNGFND_LANDING
RNGFND_MAX_CM
RNGFND_PIN_
RNGFND_SCALING
RNGFND_TYPE
RNGFND_RMETRIC

3.4.2 Auto Land Logic

Edited code excerpts are shown in the following listings describing one iteration of ArduPlane's auto land loop. As the approach and landing are dependent on GNSS inertial position estimates and altitude estimates, the auto land logic is part of the main navigation loop which runs at 10 Hz. The C++ source code for ArduPlane's auto land logic can be found on the development GitHub repository under the path: ardupilot/libraries/AP_Landing/ [17]. The two source files discussed are `AP_Landing.cpp` and `AP_Landing_Slope.cpp`, the landing and glide slope logic handlers, respectively. The main landing logic handler begins by first checking for a land mission command, initializing glide slope, and verifying initial approach conditions. Listing 11 shows a

portion of the `verify_land` boolean which is called upon initial approach and also during final flare. Waypoint targets, current position, altitude, vertical speed, and a series of checks are called–including verification that the rangefinder is within operational altitude limits. The `verify_land` boolean never returns as true, or complete. It is used to constantly monitor and adjust the current action, i.e. approach and landing, unless a GCS command is initiated. Line 4 in Listing 11 is the standard glide slope case and line 5 calls `type_slope_verify_land`, part of the glide slope logic handler, `AP_Landing_Slope.cpp`.

```
1  bool AP_Landing::verify_land(...)
2  {
3  ...
4   case TYPE_STANDARD_GLIDE_SLOPE:
5          success = type_slope_verify_land(prev_WP_loc,
           ↪  next_WP_loc, current_loc,
6                 height, sink_rate, wp_proportion,
                  ↪  last_flying_ms, is_armed, is_flying,
                  ↪  rangefinder_state_in_range);
7          break;
8  ...
9  }
```

Listing 11: `AP_Landing.cpp` boolean, `verify_land`

Although not explicitly referenced in this section, `type_slope_verify_land` is constantly calculating the stage of approach and landing with each iteration of the loop. There are four stages: normal, approach, preflare, and final. Normal stage is before the aircraft crosses the waypoint before the terminal landing waypoint. Approach stage is activated when the aircraft is lined up on heading and altitude is below the previous waypoint at the top of the glide path. When the final stage is activated, the flare is triggered under three scenarios: altitude within LAND_FLARE_ALT, vertical speed within LAND_FLARE_SEC, or flying past landing target without rangefinder data. Preflare stage is active if configured via operator and prompts the aircraft to bleed excess speed and slow closer to stall before flare. `Type_slope_verify_land` also keeps the current L1 navigation waypoint 200 meters ahead of the aircraft to prevent sudden changes in direction if the landing target is overshot.

After verification of initial approach stage, `setup_landing_glide_slope(...)` is called. The following listings are edited to show relevant portions of the algorithm and its process for generating the glide slope. First, total horizontal distance is calculated using the landing target waypoint and the preceding waypoint in line 2 of Listing 12. Next, total altitude delta to the landing target altitude is computed in line 4. Using total horizontal distance and current ground speed, time to descend to landing target altitude in calculated line 9. Vertical speed necessary for approach and landing is then calculated in line 13. Aim altitude for flare is calculated by multiplying the desired flare time (LAND_FLARE_SEC) by current vertical speed, shown in line 14. The first glide slope is then calculated in the following operation, line 18.

```
1  {...
2      float total_distance = get_distance(prev_WP_loc,
       ↪  next_WP_loc);
3      }
4      float sink_height = (prev_WP_loc.alt -
       ↪  next_WP_loc.alt)*0.01f;
5      float groundspeed = ahrs.groundspeed();
6      if (groundspeed < 0.5f) {
7          groundspeed = 0.5f;
8      }
9      float sink_time = total_distance / groundspeed;
10     if (sink_time < 0.5f) {
11         sink_time = 0.5f;
12     }
13     float sink_rate = sink_height / sink_time;
14     float aim_height = flare_sec * sink_rate;
15     if (aim_height <= 0) {
16         aim_height = flare_alt;
17     bool is_first_calc = is_zero(slope);
18     slope = (sink_height - aim_height) / total_distance;
19     if (is_first_calc) {
20         GCS_MAVLINK::send_statustext_all(MAV_SEVERITY_INFO,
           ↪  "Landing glide slope data...",);
21     }
22 ... }
```

Listing 12: `AP_Landing_Slope.cpp`, `type_slope_setup_landing_glide_slope`, part 1

After the first iteration of the glide slope calculation, time before flare is calculated using the target aim altitude and landing sinking rate, line 2 in Listing 13. Horizontal distance remaining to flare is calculated by multiplying ground speed and time before flare, line 3. During approach and landing only, the algorithm generates a target projected through the desired landing point. Erratic pitch behavior is prevented

using this method as the aircraft approaches the landing target and ground plane. This can be visualized by taking the calculated linear glide slope and projecting it through the ground plane some additional distance. The flare aim height remains the same and additional sections of code are in place to anticipate the flare, limit roll angle, and reduce throttle.

```
{...
    float flare_time = aim_height /
     ↪  SpdHgt_Controller->get_land_sinkrate();
    float flare_distance = groundspeed * flare_time;
    if (flare_distance > total_distance/2) {
        flare_distance = total_distance/2;
    }
   ...
...}
```

Listing 13: `AP_Landing_Slope.cpp`, `type_slope_setup_landing_glide_slope`, part 2

Excess altitude offset due to barometer drift is also accounted for by comparing the initial glide slope calculation to a glide slope estimate utilizing the rangefinder correction. If the glide slope difference crosses a certain threshold, the glide slope will be recalculated to account for the barometer drift. If the glide slope error passes an operator defined parameter, the landing will abort, store the barometer offset, and execute go around procedures to line up for another approach. This logic is handled by a separate function, `type_slope_adjust_landing_slope_for_rangefinder_bump`. The remaining logic within `AP_Landing.cpp` and `AP_Landing_Slope.cpp` manages airspeed targets, heading adjustments for wind compensation, abort and restart of the landing sequence, and disarming of the aircraft.

CHAPTER 4

Results

4.1 Fixed-Wing Flight Test

System characterization flights were executed with the RMRC Anaconda to evaluate auto land capability. Approaches were generated and logged using the auto land traffic pattern script. A total of 45 approaches and landings were recorded across five test sessions. A wind rose plot is shown in Figure 4.1 for all recorded approach and landings. Maximum estimated wind speed was 16 knots and the maximum demonstrated crosswind component was 9 knots.

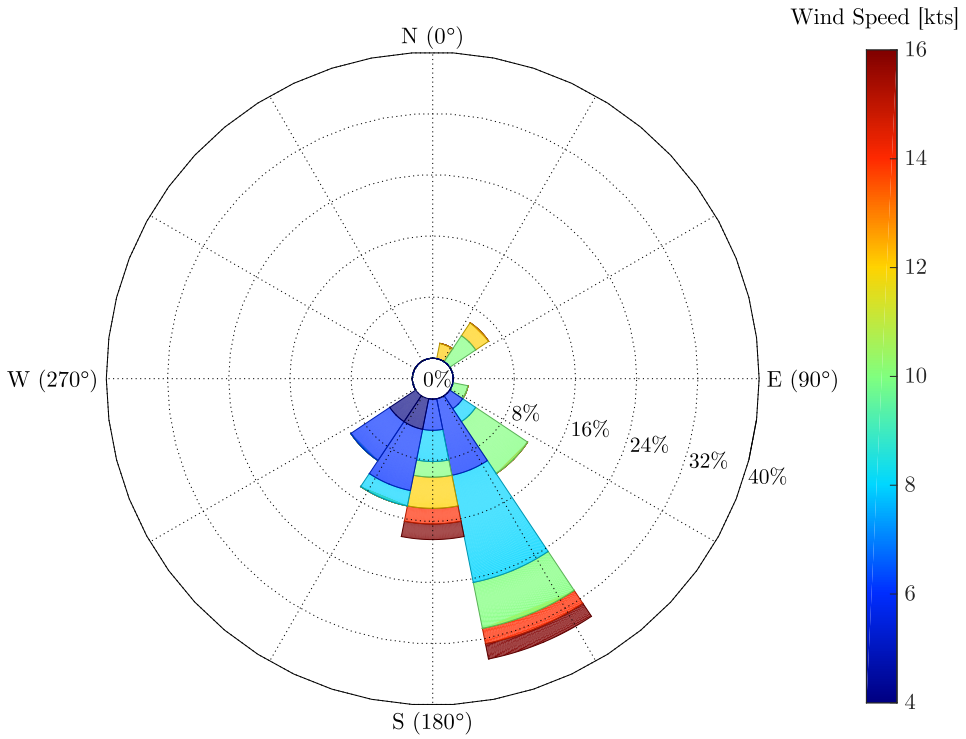

Figure 4.1: Wind Rose for Fixed Wing Landing Approaches, Vehicle Estimate at Short Final

A visualization of the wheel stop location, zero ground speed, for all recorded landings is shown in Figure 4.2. No target landing locations are shown in Figure 4.2, but distinct grouping can be observed for several desired landing points. Each approach and landing was logged by the auto land script for several parameters, including the wheel stop distance with respect to distance from the desired landing point. All landings were performed with a GNSS module capable of receiving only US GPS satellite signals. All 45 approach and landings had active differential GPS correction at the runway threshold. Mean self reported horizontal dilution of precision (HDOP) was 0.81 ± 0.18. At two standard deviations, HDOP was 0.99, excellent for a single constellation GNSS setup.

Figure 4.2: Anaconda Auto Land Performance, Wheel Stop Locations

Figure 4.3 and Figure 4.4 represent two approach trajectory trends. At short final the laser altimeter consistently reported higher altitude than barometric altitude as shown in Figure 4.3. The green trace in Figure 4.3 is the laser altimeter measurement

and the red trace is the EKF altitude state estimate. The step jump is the point when the laser altimeter is activated. The EKF altitude state snaps to the laser altimeter measurement because the filter innovation (difference between predicted and measured value) is nearly zero throughout the approach.

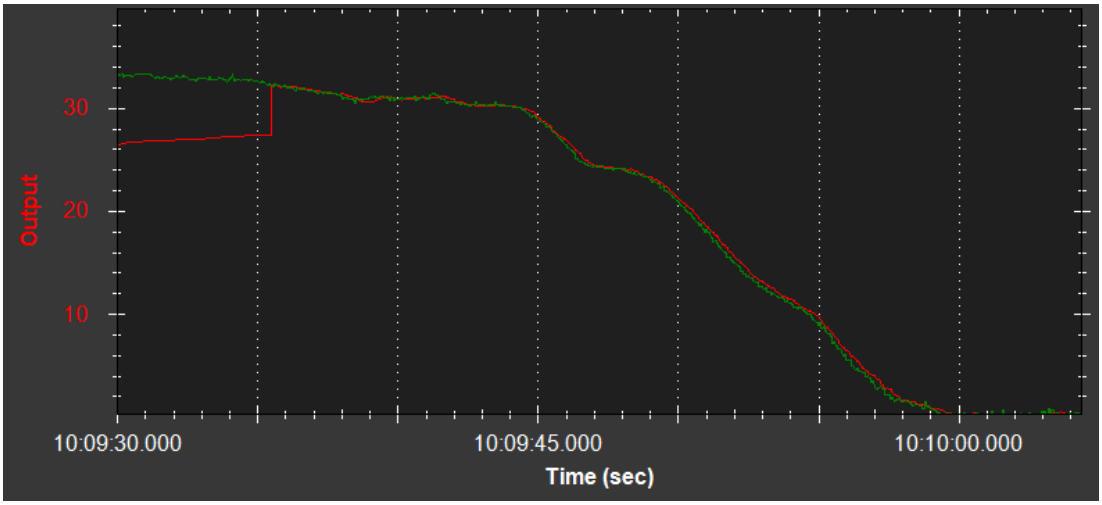

Figure 4.3: Anaconda Auto Land Performance, EKF Height Estimate and Laser Rangefinder State Time Trace [meters]

Once landed, barometer drift was typically less than 5 feet. The difference is likely due to a gradual decrease in elevation at the short final waypoint with respect to the ramp staging area where the barometer was armed. The two trajectories are either stabilized with only a few minor deviations (Figure 4.3) or exhibited oscillatory behavior as shown in Figure 4.4. Figure 4.4 is data from the auto land script and the 5.1° line represents the initial barometric altitude based glide slope. These larger bumps are primarily due to the parameter controlling weighting between airspeed and altitude error along the approach. TECS_LAND_SPDWGT was set to the default value of 1 for all approaches, which prioritizes the errors equally. At TECS_LAND_SPDWGT = 1 the autopilot can correct airspeed errors using pitch. Setting TECS_LAND_SPDWGT to a value closer to zero would prompt the autopilot to primarily maintain airspeed target using throttle and could stabilize the glide slope more.

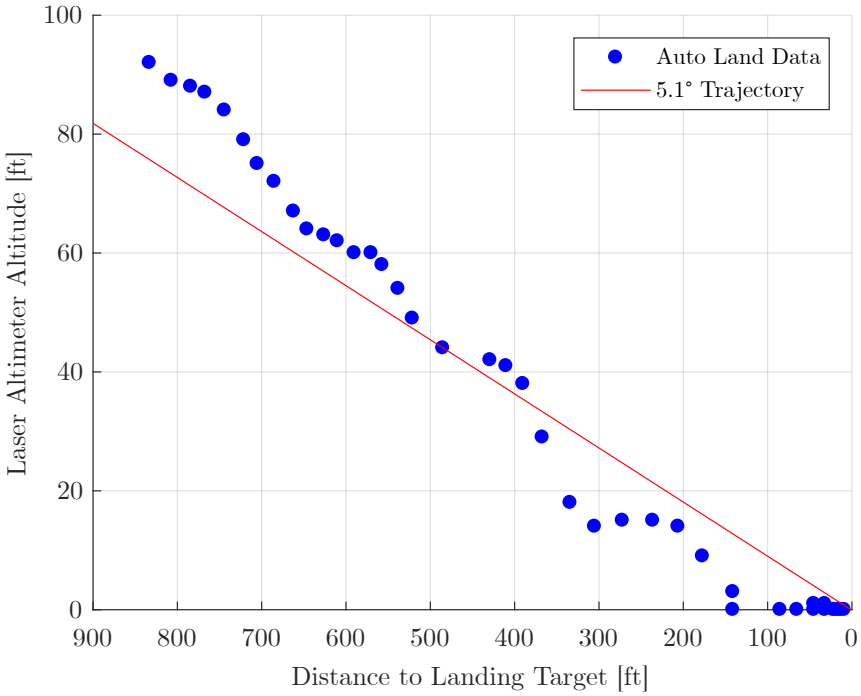

Figure 4.4: Anaconda Auto Land Performance, Trajectory

Attitude response during approach and landing is shown in Figure 4.5 and Figure4.6. In both Figures, the green trace is the desired attitude and the red trace is the measured attitude. Pitch and roll response in general is acceptable, but pitch overshoot was common was near the end of the approach as shown in Figure 4.5. In normal flight, the primary flight control PID gains performed outstanding considering the amount of time spent tuning gains. ArduPlane features an auto tune capability where the flight controller "learns" the response of the aircraft and adjusts PID gains by monitoring pilot input versus attitude response. Over the course of a 10-15 minute auto tune session most fixed wing aircraft have an 80-85% gain solution that facilitates GNSS guided waypoint flight in most flight conditions. Auto tune is one of ArduPlane's most impressive features and eliminates a notoriously difficult aspect of setting up a new aircraft. However, a precision approach and landing could benefit from the extra 10-15% response performance that manual tuning can accomplish.

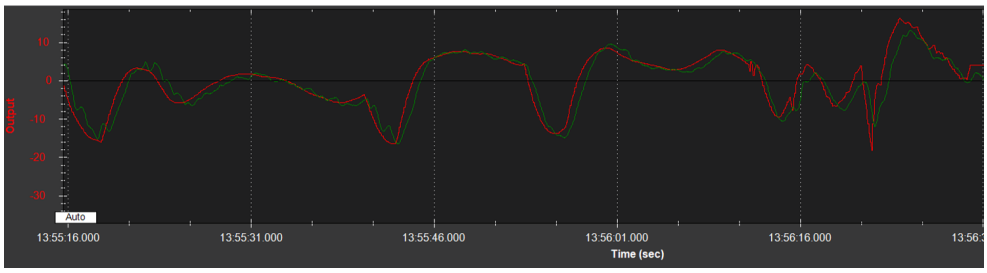

Figure 4.5: Anaconda Auto Land Performance, Pitch Desired/Actual Trace

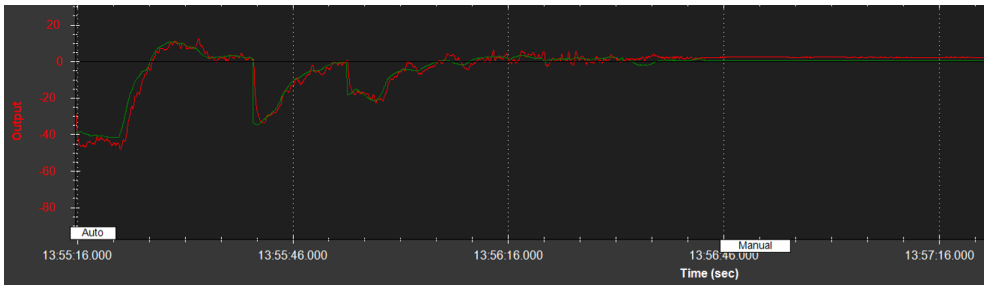

Figure 4.6: Anaconda Auto Land Performance, Roll Desired/Actual Trace

Consistent flare initiation at the aim point produced repeatable wheel stop locations as shown in Figure 4.7. On the majority of approaches flare is initiated consistently between 200 and 150 feet to the landing target. The consistent flare at the desired aim point resulted in the greatest number of wheel stop distances within 50 feet. Factors discussed above such as reducing airspeed correction via pitch input and further tuning of autopilot gains could increase trajectory tracking consistency to the aim point. Figures 4.8 and 4.9 are histograms of airspeed and altitude performance at short final. The target for all runs was 28 knots and 80 feet, respectively.

In summary, despite variability and off condition airspeed or altitude (Figures 4.8 and 4.9), the aircraft still consistently lands adequately. A cumulative probability function was generated using recorded wheel stop distances with respect to the desired landing point and is shown in Figure 4.10. The dashed lines represent the 95% confidence bounds. Conservatively, using the lower 95% confidence bound, there is an 80% probability that the fixed wing aircraft will stop within 100 feet of the desired point.

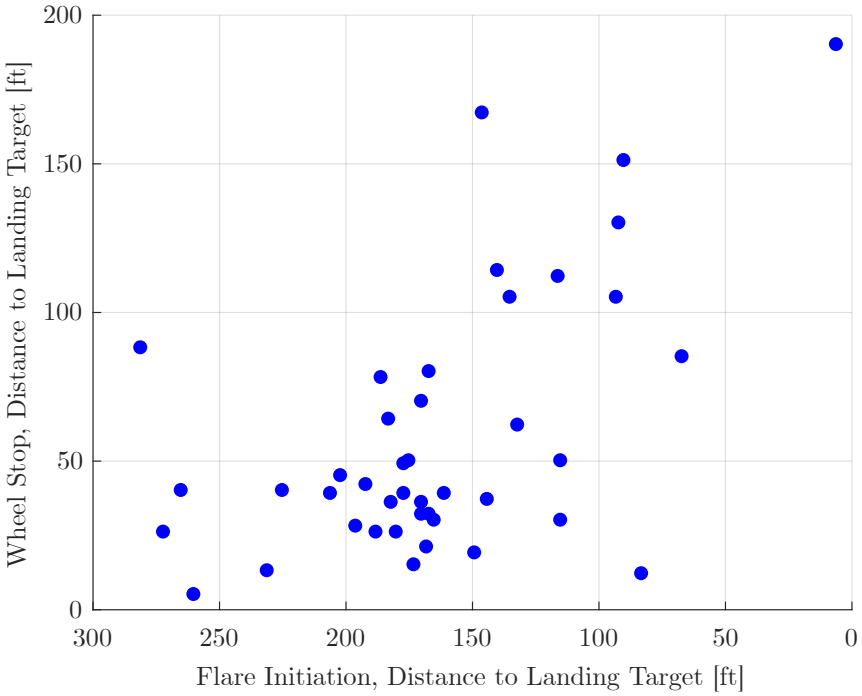

Figure 4.7: Anaconda Auto Land Performance, Flare Initiation

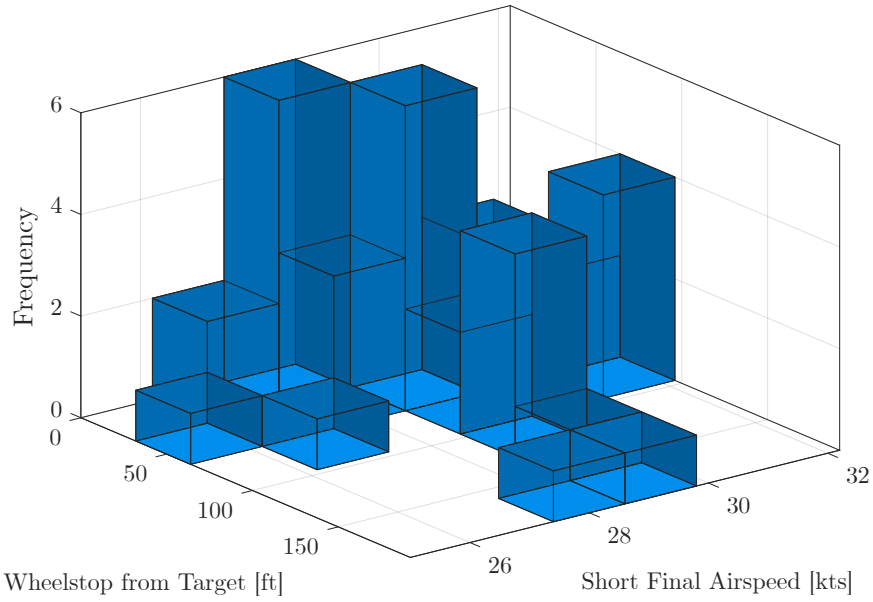

Figure 4.8: Anaconda Auto Land Performance, Short Final Airspeed Histogram

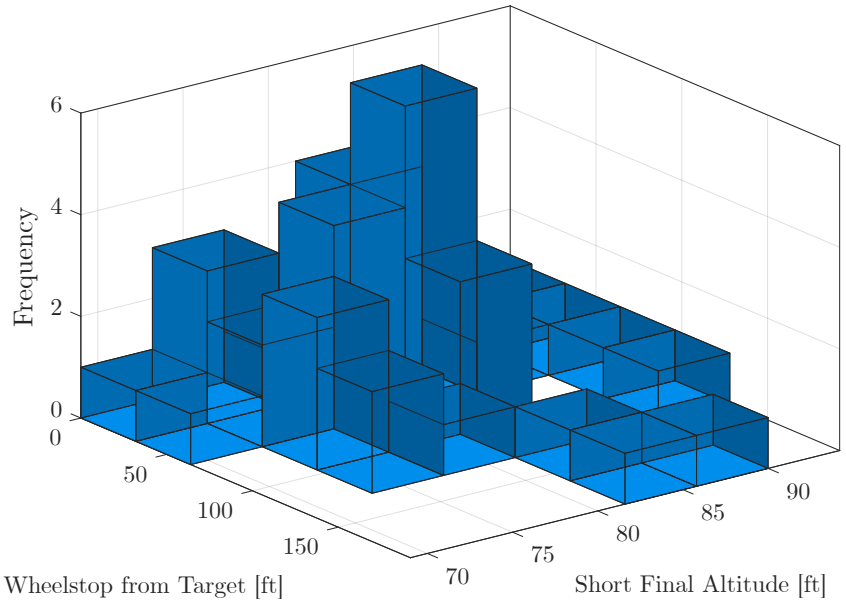

Figure 4.9: Anaconda Auto Land Performance, Short Final Altitude Histogram

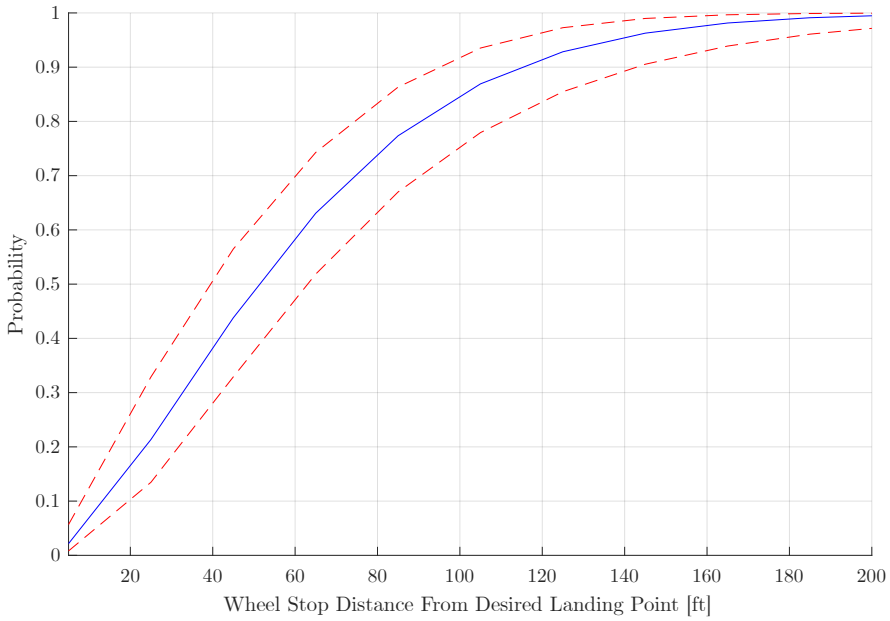

Figure 4.10: Anaconda Auto Land Performance, Wheel Stop Target Cumulative Probability

4.2 Multi-Rotor Flight Test

The multi-rotor test plan was executed six times for each vehicle. Figure 4.1 and Figure 4.2 show results for the DJI Mavic and 3DR Solo, respectively. Measurements were made after landing with respect to initial launch position. Measurements were made with a hand tape from the approximate center of mass of each vehicle. Wind conditions were 5 knots and 6 knots, SSE, over a 10 minute average for the Solo and Mavic tests, respectively. The 3DR Solo average accuracy with respect to initial launch position was 35 inches with a standard deviation of 11 inches. The DJI Mavic performed better, as expected, with its additional sensors. Average error from launch point was 5 inches with a standard deviation of 3 inches.

Table 4.1: Mavic Precision Landing Tests

Target Lat	**Target Lng**	**Reported Lat**	**Reported Lng**	**Target Delta [in]**
36.16217	-96.83567	36.16217	-96.83567	9
36.16217	-96.83567	36.16217	-96.83567	2
36.16215	-96.83567	36.16215	-96.83567	7
36.16215	-96.83567	36.16215	-96.83567	5
36.16214	-96.83567	36.16214	-96.83567	4
36.16214	-96.83567	36.16214	-96.83567	3

Table 4.2: Solo Precision Landing Tests

Target Lat	**Target Lng**	**Reported Lat**	**Reported Lng**	**Target Delta [in]**
36.16217	-96.83566	36.16217	-96.83566	24
36.16217	-96.83565	36.16217	-96.83566	48
36.16217	-96.83565	36.16217	-96.83565	38
36.16217	-96.83566	36.16217	-96.83565	48
36.16217	-96.83566	36.16217	-96.83566	27
36.16216	-96.83565	36.16216	-96.83565	24

4.3 Wake Vortex Considerations

A key motivator to maintain "well clear" of other aircraft is not only to avoid a collison, but also to prevent wake vortex encounters and subsequently, hazardous effects. As lift varies across an aircraft wingspan, circulation is shed as a vortex sheet that starts at the trailing edge and progresses downstream. The vortex wake formation begins as the initial high-pressure swirl from the lower surface of the wing creates the tip vortex which then sucks in more of the trailing edge vortex sheet further downstream in a process commonly referred to as vortex rollup. The result is a well-defined pair of oppositely signed vortices—usually completely and distinctly formed several wingspans downstream. Aircraft wings normally have finite length discontinuities throughout the span like flaps and ailerons. These surfaces also create varying strength tip vortices and sheets where each distinct cross-section of vorticity distorts over time combining into the final vortex wake structure downstream.

A primary objective of early wake vortex research was to formulate analytical closed form solutions to quantify worst case scenarios for a wake vortex encounter. A common result of intercepting a strong wake vortex is an induced roll moment that could exceed available roll control. Other effects depend on the orientation of the following aircraft and the wake vortex. For example, flying perpendicularly into a vortex core will impart large structural loads that could excite aero-elastic modes resulting in failure of the structure. A more common scenario is simply flying into the downwash area before full vortex wake rollup. These scenarios are shown in Figure 4.11.

Rossow presents several early closed form solutions, but many are based on knowledge of the vortex geometry with respect to the aircraft wing [35]. In most cases, the vortex core radius and orientation is unknown and must be assumed. Hallock presents a simplified metric based on the roll moment coefficient, C_l, induced by a point vortex located at the center of a wing (fuselage), see Equation 4.1 [36].

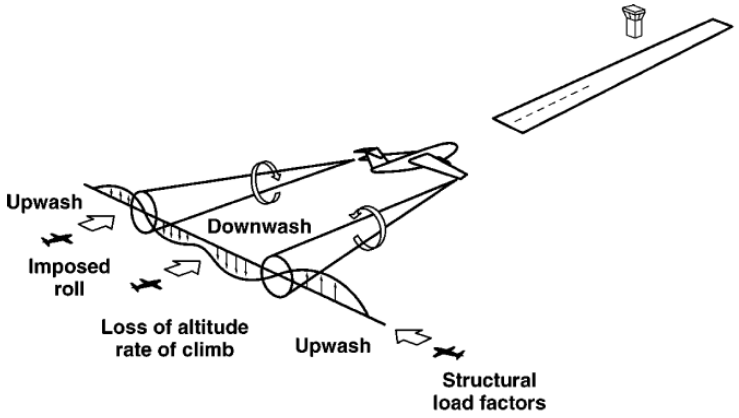

Figure 4.11: Wake vortex encounter scenario [35]

$$C_l = \frac{C_{L\alpha}\Gamma}{2\pi U_\infty b} \tag{4.1}$$

$C_{L\alpha}$ is the 3-D lift curve slope, Γ is vortex circulation, U_∞ is aircraft speed, and b is the aircraft wingspan. Zheng and Ash also state that the maximum roll moment coefficient occurs when the following wing center is located at the vortex core [37].

Although not experimentally tested in this study, a brief analysis of potential wake vortex hazards was performed using conservative calculation methods. Induced roll is typically a hazard for aircraft in a leader follower formation. Normally, an incremental change in roll moment is provided via an aileron or similar control surface that modifies the spanwise lift distribution of a wing. Roll control power, $C_{l_{\delta a}}$, can be approximated using a strip integration method as shown in Equation 4.2. The same method can be used to calculate roll moment coefficient due to a vortex encounter [36, 38]. An expression for the strength of the vortex can be calculated assuming an elliptic lift distribution for a generating aircraft, shown in Equation 4.3, where W is the weight, U_∞ is the flight speed, and b' is the effective span of the vortices ($\frac{\pi}{4}b$). The expression for roll moment coefficient induced by a vortex in Equation 4.4 assumes a point vortex at the center of a wing with no diffusion or decay. $C_{L_{\alpha w}}$ in both expressions is the 3D lift curve slope corrected for aspect ratio (AR $= b^2/S$) where y is the control surface dimension.

$$C_{l_{\delta a}} = \frac{2C_{L_{\alpha w}}\tau}{Sb}\int_{y_1}^{y_2} \mathrm{c}y \ \mathrm{d}y \tag{4.2}$$

$$\Gamma = \frac{W}{\rho U_\infty b'} \tag{4.3}$$

$$C_{l_v} = \frac{C_{L_{\alpha w}}\Gamma}{2\pi U_\infty b} \tag{4.4}$$

Assume an Anaconda encounters another similarly sized aircraft and then a vehicle similar to the Penguin-B, which is on the larger side of the small UAS spectrum [39]. For the first scenario, the following Anaconda has approximately 70% available roll control after a direct encounter with a similarly sized vehicle. The RMRC Anaconda used in this study has rectangular wing planform with large aileron surfaces capable of large deflections. For the second scenario, an Anaconda encountering a Penguin-B sized vehicle, about 50% roll control remains for counter control. This assumes roll is input instantaneously and that the circulation strength of the vortex does not decay. Although the method is conservative, similar approaches are used in conjunction with LIDAR measurements to provide vortex circulation estimates of generating aircraft and characterization of potential roll hazards.

A popular concept for an active system to prevent wake vortex encounters utilizes ADS-B out data from surrounding traffic to create a fast time wake vortex model [10]. The predicted hazard corridor is compared to the predicted UA flight path. If there is a potential conflict, an avoidance maneuver is commanded. The general concept block diagram is shown in Figure 4.12. Another method for avoiding multi-rotor downwash is presented by Yeo, et. al. [40]. Yeo details an active pressure sensing probe system mounted on a multi-rotor vehicle coupled with a estimation algorithm that detects, localizes, and avoids a vertical disturbance via a path planner.

Table 4.3: Small UAS Vortex Encounter Scenarios

Anaconda/Anaconda Scenario	
Lead Aircraft Flight Speed	40 [ft/s]
Lead Aircraft Weight	10 [lb]
Lead Aircraft Vortex Circulation	20 [ft^2/s]
Following Aircraft Flight Speed	40 [ft/s]
Following Aircraft Roll Control Power, $C_{l_{\delta a}}$	0.5 [rad^{-1}]
Following Aircraft Induced Vortex Roll Coefficient, C_{l_v}	0.06
Following Aircraft Roll Control Ratio, $(C_{l_v}/C_{l_{\delta a}}\delta_a)$	0.26
Penguin-B/Anaconda Scenario	
Lead Aircraft Flight Speed	55 [ft/s]
Lead Aircraft Weight	45 [lb]
Lead Aircraft Vortex Circulation	41 [ft^2/s]
Following Aircraft Flight Speed	40 [ft/s]
Following Aircraft Roll Control Power, $C_{l_{\delta a}}$	0.5 [rad^{-1}]
Following Aircraft Induced Vortex Roll Coefficient, C_{l_v}	0.12
Following Aircraft Roll Control Ratio, $(C_{l_v}/C_{l_{\delta a}}\delta_a)$	0.53

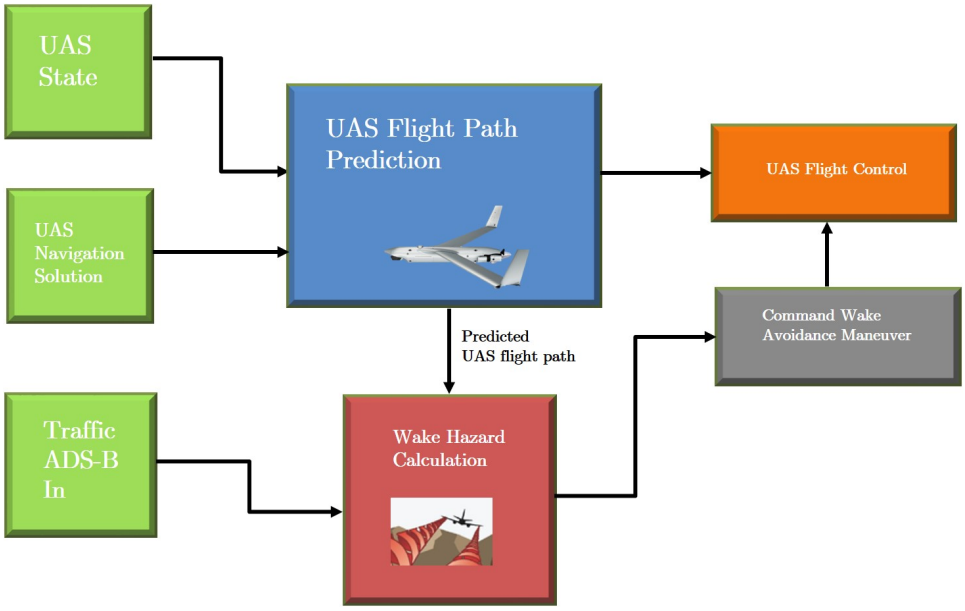

Figure 4.12: Wake Avoidance ADS-B Concept

.

CHAPTER 5

Conclusions

5.1 Flight Test Conclusions

An autonomous traffic pattern landing script was developed and tested in conjunction with auto land capable fixed wing small UAS. The script allows the operator to select a desired touchdown point and generate a traffic pattern based on wind conditions. This flight planning and command and control capability is recommended for all fixed wing aircraft in the cases when landing approaches must be adjusted quickly. The ability of the fixed wing UAS to report a real time estimate of wind speed and direction enabled the pattern to be flown with respect to standard procedure of landing into the wind.

The Autopilot is able to compensate target airspeed for the estimated head winds and crosswinds. Auto landing performance was demonstrated in winds as high as 16 knots, with maximum crosswind components approaching 10 knots. The wind estimate from the vehicle is not necessarily meteorological grade, but is more than sufficient to provide these basic functions to augment fixed wing landing performance and increase operator situational awareness. As discussed in Chapter **??** wind and weather sensing were highlighted as areas to improve upon for large scale airspace and trajectory planning in an eventual UTM system. Most small UAS, even multi-rotors, can estimate a 2D wind vector with reasonable accuracy. A rich wind vector data set could be available to more complex, stochastic models within a UTM framework that can receive estimates from all connected vehicles in real time; similar to the method proposed by Salazar et. al [32].

50 plus approaches (45 landings for record) were conducted during the flight test campaign with no mishaps. A single landing was waved off due to an unstable approach, but the autopilot triggered a go-around at its pre-determined glide path error threshold and successfully rejoined the traffic pattern to land during the next attempt, autonomously. Performance was sufficient for a 600 foot long runway surface, specifically for the fixed wing aircraft. Flight test data showed an 80% likelihood of landing within 100 feet of the desired point. Consistent flare initiation resulted in best landing performance. Further tuning of autopilot gains and autopilot prioritization of throttle control for airspeed correction would likely increase trajectory tracking consistency to the aim point. Demonstrated fixed wing landing performance appears to agree with the general rule of thumb of general aviation approach and landing technique: aim for the first third of the runway, allow the second two thirds for flare and roll out. A 3,000 foot runway should accommodate all small UAS up to 55 pounds.

Given a maximum altitude of 400 feet AGL dictated by part 107 regulations, a standard small UAS fixed wing traffic pattern should be flown starting at 200 feet AGL downwind. A loiter transition down to pattern altitude is preferred. Glide slope is dependent on obstacle height along the intended approach path. A short final waypoint fix, to engage the precision portion of the approach, can be added to allow the vehicle to stabilize once clear of obstacles. However, a short final fix too close to the runway threshold can introduce excess barometer/laser altimeter error that requires the vehicle to readjust glide path too quickly with respect to the landing target. Additionally, the navigation waypoint radius threshold should be dynamically adjusted based on wind conditions to ensure stable cross track during each maneuver leg of the traffic pattern. Finally, multi-rotor aircraft should fly traffic patterns opposite of fixed wing traffic at 100 feet AGL, similar to existing general aviation best practice. For example, if a fixed wing aircraft is entering the left hand

pattern at 200 ft AGL, a multi-rotor should execute a right hand pattern at 100 ft AGL to remain well clear of the fixed wing traffic.

In conclusion, hardware and software are currently available with acceptable SWAP that enable precision approach and landing of both fixed-wing and multi-rotor vehicles. The systems characterized in this paper are on the smaller side of the 55 pound small UAS category, but are inherently scalable. ArduPlane provides a robust software architecture, advanced flight features, and adaptability suited to larger UAS via intuitive tuning and configuration. Hardware wise, fixed-wing single constellation GNSS module HDOP values less than 1.0 at 2 standard deviations were observed and all flight test landings fell within the 55 foot wide boundaries of the runway surface. Redundant dual band GNSS capability should virtually eliminate concerns of hardware failure or coverage dropout. Precision range finding devices on both fixed-wing and multi-rotor vehicles are critical. Laser altimeters provide high quality measurements at altitudes necessary for fixed wing precision approaches and landings while vision or sonar based systems drastically improve multi-rotor precision landing capabilities.

5.2 General Aviation Infrastructure Case Study

Initial concepts and development of higher volume UAS facilities are co-located at the regional airport scale and below. This includes local airports that account for 38% of all National Plan of Integrated Airport Systems (NPIAS) Airports. NPIAS Airports have been deemed important to the national airspace system by the FAA and thus eligible for federal funding under the Airport Improvement Program (AIP). The FAA defines regional airports as non-primary airports (some have commercial service) serving a metropolitan urban core population of at least 50,000 or a micropolitan urban core population of 10,000 to 50,000 [41]. Regional airports have high levels of activity and support both multi-engine and jet operations. Local airports support

mainly piston aircraft and are located near larger population areas, but not always as part of a metro or micropolitan demographic. One of the first planned higher volume UAS facilities is the Grand Sky UAS Business & Aviation Park in North Dakota. The facility is attached to Grand Forks Air Force Base with direct runway access. Figure 5.1 shows the conceptual layout of the facility. As of April 2017 the facility began operating flights beyond visual line of sight within a 60 nautical mile radius under FAA waiver.

Figure 5.1: Grand Sky UAS Business & Aviation Park [42]

Grand Sky is intended to be a UAS research, testing, and training facility capable of supporting high altitude, long endurance UAS operations. However, a similar approach of using existing regional and local airport infrastructure to stage higher volume UAS operations will continue across the United States. Small UAS, as currently defined by the FAA, have a maximum takeoff weight of 55 pounds; the same as the Department of Defense (DoD) definition of Group II tactical UAS. Commercial off the shelf (COTS) internal combustion powered fixed wing UAS in this category are capable of flight times exceeding 12 hours depending on payload configuration [39]. Assuming UAS operators will mimic existing general aviation aircraft missions with capable fixed wing small UAS operating beyond visual line of sight (BVLOS), a back of the envelope expectation of flight hours can be estimated using FAA general aviation survey data.

The FAA collects annual data on aircraft use cases, type, and flight hours [43]. Instructional, aerial agriculture/application, and aerial observation are three use cases from the general aviation survey with a strong likelihood of being augmented via UAS operations. For a conservative lower bound estimate, UAS flight hours are assumed to be a percentage of general aviation flight hours in each category. Using this heuristic approach and 2015 survey data—523,000 annual UAS flight hours are expected as shown in Table 5.1, about 1% of total GA flight hours. Similar to general aviation, instructional flight markets and businesses will develop for UAS operations as BVLOS training requirements are mandated. Aerial agriculture typically includes application of fertilizers and pesticides, but the payload capacity of small UAS will reduce adoption of the mission set in the United States under limited circumstances, considering the 55 pound weight limit.

Table 5.1: UAS Flight Hour Estimate (rounded to nearest thousand) from 2015 FAA General Aviation (GA) Survey [43]

	Instructional	**Aerial Agriculture**	**Aerial Observation**
GA Hours	4,648,000	941,000	1,412,000
UAS hours	232,000 (5%)	9,000 (1%)	282,000 (20%)
Total			523,000 UAS Flight Hours

Excluding spraying and application, precision agriculture missions still include surveying, mapping fields, and livestock monitoring. These missions are likely to be based out of local airport scale infrastructure or separate rural staging. The remote sensing portion of precision agriculture falls within the third category, along with majority of potential use cases for small UAS, aerial observation. In 2015, 13% of total general aviation flight hours were flown by public use aircraft [43]. Nearly all search and rescue operations, disaster relief, homeland security, and law enforcement missions are performed by public use aircraft and fall under the aerial observation category. In addition, aerial observation encompasses industrial inspection and monitoring of national infrastructure—manufacturing facilities, pipelines, electrical grids,

roads, dams, bridges, etc. The aforementioned mission sets are rather diverse and well suited for capable small UAS operations based out of existing general aviation scale infrastructure, which was designed to support a variety of use cases.

The time horizon for 500,000 UAS flight hours staged at the regional and local airport scale is unknown. Continuing the heuristic discussion, several regulatory issues need be addressed, primarily BVLOS. The time line for BVLOS is also uncertain, but it is conservative to expect regulations within the next 10 years. Flight hours are assumed to accumulate rather quickly once in place. For comparison, the 500,000 small UAS flight hour milestone could be easily be exceeded within the first year of FAA Part 107 operations if each commercially registered UAS flew slightly more than 10 hours annually. There are no direct source methods for gathering Part 107 UAS operational flight hour data unless the FAA begins to conduct surveys, similar to GA. Current operations fall under established Part 107 rules and primarily are conducted within visual line of sight, point launch and recovery. However, over 6,800 Part 107 waivers for operations in controlled airspace were granted in a 2016; along with 20% of total waivers requesting operations beyond visual line of sight [1].

5.2.1 Elements of Small UAS Operation at Stillwater Regional

Stillwater Regional Airport (KSWO) is a public use city owned airport located in Stillwater, Oklahoma. The airport has two runways that are 7,401 feet and 5,004 feet in length. During tower service hours, 0800-2000, KSWO is Class D airspace. In 2016, American Airlines began daily scheduled service to Dallas Fort Worth International Airport using a 50-75 seat class commuter jet. KSWO sees a mix of traffic, mostly general aviation. Oklahoma State University operates a FAA part 141 flight school out of KSWO. Larger narrow body, single aisle class aircraft are chartered by Oklahoma State University during athletic seasons and the airport also services military traffic typically consisting of student pilots training at nearby Vance Air Force Base. KSWO

fits the demographic described in the previous section and is a potential candidate to stage small UAS operations serving the local economy. A concept of operation working towards a KSWO flight demo is presented that focuses on a systematic crawl, walk, run test method. Considerations and recommendations for limited KSWO small UAS operation complying with FAA Part 107 waiver guidelines are presented in conclusion.

Operating Rules and Equipment For Controlled Airspace Entry

There are five operating rules and requirements for Class D airspace [5]. First, there are no specific manned pilot certifications required for operation in Class D airspace. However, the UAS pilot in command (PIC) will, at a minimum, posses a commercial pilot certificate. This ensures that the PIC has experience with all aspects of normal operations and radio communication. No transponder is required for operation in Class D airspace, but two way radio contact is required. Arrival and entry rules for Class D airspace state radio contact before entering the airspace. Since all operations will be conducted within KSWO Class D airspace, PIC will comply with standard ground and tower radio procedures—clearance for taxi, hold for takeoff, clear for takeoff, depart heading, etc.

Aircraft speed must be below 200 knots and there are no separation services provided to VFR aircraft. The distinction between VFR or IFR UAS should be avoided unless it is clear that the PIC will be filing an instrument flight plan and receiving radar vector instructions from regional ATC centers. A current UAS IFR scenario is typically reserved for high altitude long endurance UAS, like NASA's Ikhana. For KSWO UAS operations, the PIC is ultimately responsible for utilizing all information available via telemetry and observers (ground visual and tower controller) to maintain safe separation and flight, a pseudo UAS VFR condition.

UAS Configuration and Equipment

The recommended UA for KSWO operations is the UAV Factory Penguin-B shown in Figure 5.2. Penguin-B is a aerodynamic and structurally efficient "small" UA platform with a wing span of 11 ft and maximum take off weight approaching 50 lbs. It can be equipped with an internal combustion engine capable of providing more than enough endurance for the KSWO UAS demo. In the base configuration, a 28cc engine provides multiple hours of flight. For reference, Penguin-B UAS have flight proven 20 hour endurance with electronic fuel injection.

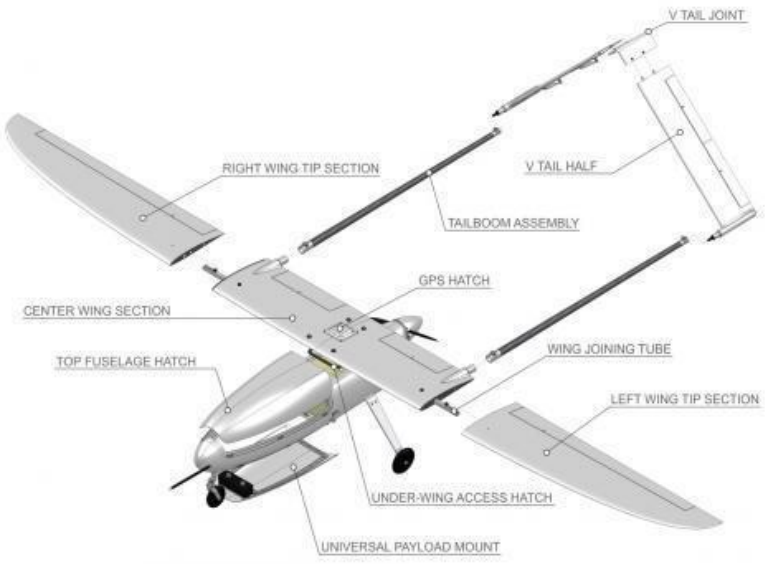

Figure 5.2: Penguin-B [39]

Avionics hardware and software architecture is recommended to be similar to this study, but components added for safety and redundancy. Pixhawk 2.1 offers a triple redundant, vibration isolated IMUs, dual redundant GNSS, and dual redundant power distribution. Dual GNSS is supported through both hardware, two physical receivers, and software, via EKF position blending. Pixhawk 2.1 supports redundant power distribution architecture as shown in Figure 5.3. Not only does the architec-

ture protect against a single point of failure from a battery or electrical hardware component, but a high quality power distribution board will also passively reduce risk by supporting much greater power demand from heated IMUs, 2x GNSS, flight servos, engine ignition, high power telemetry radios, navigation lights, laser altimeter, ADS-B, etc.

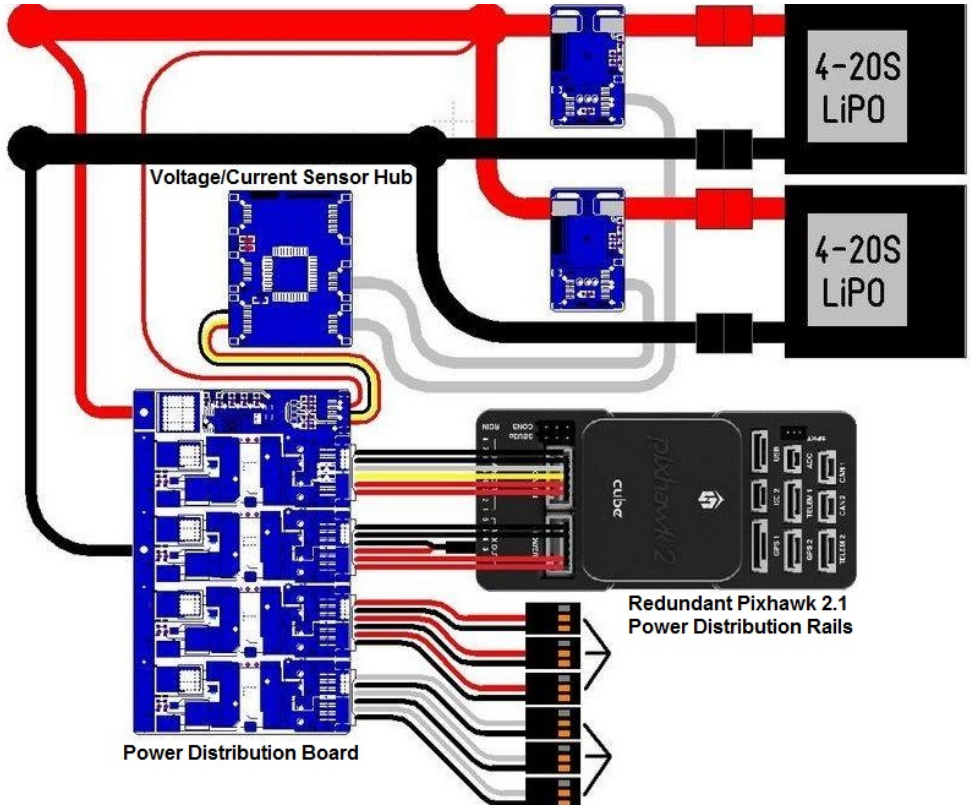

Figure 5.3: Pixhawk 2.1 Redundant Power Distribution Architecture, Mouch Electronic

The standard for small UAS ADS-B capability is uAvionix's Ping 2020 transceiver. The Ping2020 is ADS-B in/out capable, direct integrates with Pixhawk, ArduPlane, and Mission Planner and is similar in size to a SD card. Nominal transmit power is 20W, which is more than sufficient to broadcast ADS-B out messages to surrounding KSWO traffic. uAvionix has also released ADS-B in capability integrated with general aviation electronic flight bag applications for a mere $199. Laser altimeters from LightWare remain the quality standard for small UAS with respect to size, weight,

power, and performance. A laser altimeter provides critical auto landing capability demonstrated in this study. To increase visibility, beacon and navigation strobe lights are recommended.

Taxi will be controlled by the PIC, but all flight plans will be autopilot from takeoff to landing roll out. A manual override and flight control is available to the PIC via a high power 1W transmitter system. HD full motion video capability provides increased PIC situational awareness during taxi maneuvers and flight. HD video transmission range in excess of 2 nautical miles can be achieved by the compact DJI LightBridge system, the standard for cost effective long distance small UAS HD video transmission. Upgraded command and control data links are necessary to ensure reliable communication with the aircraft at further ranges. RFDesign Pty Ltd offers off the shelf encrypted Pixhawk/ArduPlane compatible telemetry radios with an effective range exceeding 3 nautical miles with moderate radio line of sight. Performance is greatly increased by adding directional antenna tracking capability, which ArduPlane and Mission Planner support natively. An antenna tracker setup can also incorporate directional video antennas. Table 5.2 summarizes recommended avionics components.

Table 5.2: KSWO Penguin-B Avionics Components

Component	**Model or Vendor**
Autopilot Hardware and Software	Pixhawk 2.1, ArduPlane
GNSS	2x Pixhawk 2.1 Here GNSS modules
Air Data	MRobotics Pixhawk 2.1 NextGen Airspeed Sensor
Integrated fuel flow sensor	Aero Telemetry SS-FFS-350 or similar
Power Distribution System	Mouch Electronic Pixhawk 2.1 Power Cube
Navigation Lights	North American Survival Systems DS-30 or similar
Telemetry C2	RFD 900x Encrypted MAVLink Radios
Directional Antenna Tracker	Pan-tilt directional mount, various vendors or custom
OR an Antenna Mast	Blue Sky Mast
PIC Radio Control	DragonLink V3
Laser Altimeter	LightWare SF20
ADS-B In/Out	uAvionix Ping2020
Full Motion Video	DJI Lightbridge 2

Concept of Operations

The concept of operations proposed for limited small UAS operation at KSWO is a systematic craw, walk, run approach. UA platform and avionics architecture are proposed for performance standards and redundancy. Thorough initial flight testing and checkout of UA and rehearsal of *all planned* KSWO operations is conducted in uncontrolled airspace under normal Part 107 rules before even scheduling KSWO operations. There are several elements to successful UAS operations first discussed in Chapter 2, but shown here for convenience in Figure 5.4. This proposal aims to adhere to key operational elements including UA airworthiness and safety, flight operations best practices, and robust operator experience, training, and certification.

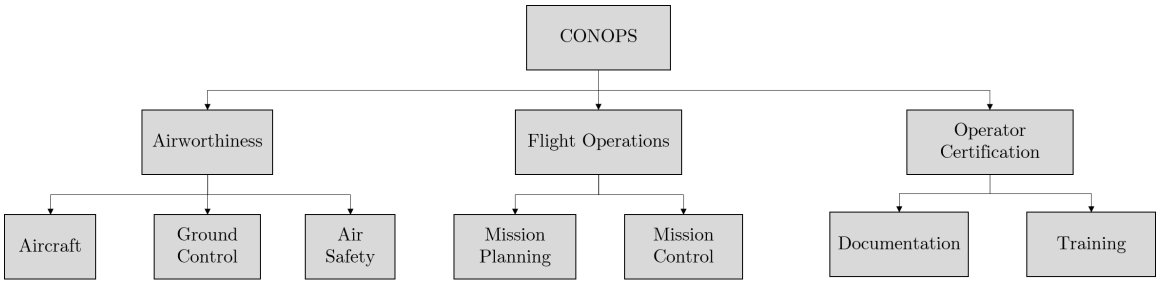

Figure 5.4: Elements of UAS CONOP, adapted from [12]

The following outline provides recommendations for personnel and responsibilities:

1. UA Pilot-in-Command

    (a) Ultimately responsible for UA during operations and sole individual communicating on KSWO tower control radio frequency

2. 2x UA Flight Test Engineers

    (a) Primarily responsible for generating and maintaining data compendium detailing airworthiness and performance capability of UA throughout test program

(b) Responsible for autopilot configuration, monitoring UA status, communicating status to flight crew and PIC, and initiation of autopilot flight maneuvers during operations

3. UA Crew Chief

   (a) Responsible for UA platform logistics and configuration, ramp start procedures and checks, and taxi marshalling

4. 3x Visual Observers

   (a) Responsible for relying available visual status of UAS when prompted by any flight crew member, also monitors KSWO tower frequency and alerts flight crew to any visually spotted traffic not previously identified as cooperative

5. KSWO Tower Operator

   (a) Considered "flight director" with the ultimate authority to terminate operations if necessary (unexpected traffic conditions/workload/weather/etc.)

   (b) Provides guidance to any potential manned aircraft in vicinity and also gives direction and clearance to UA for flight maneuvers

UA PIC, one Flight Test Engineer, and the UA Crew Chief will be co-located at the main ramp staging area and ground control station. One Flight Test Engineer will be located in the KSWO Control Tower with tower operations personnel. The remote Flight Test Engineer will have a remote ground control station configured to receive multi-point capable telemetry (same data stream as primary control station) from the UA in flight and also a real time full motion video slave receiver. This will allow tower operators to monitor UA status in real time. Three visual observers will be placed in sectors promoting greatest visual coverage of intended UA flight

area and surrounding airspace. Communication hierarchy is simple. PIC, Flight Test Engineers, Crew Chief, and Visual Observers are all communicating on a Multi-Use Radio Service (MURS) UHF channel. PIC, Flight Test Engineers, and Visual observers are also monitoring KSWO frequencies, but *only* PIC is communicating on KSWO frequencies. Although the hierarchy is simple, flawless communication is one of the most critical aspects of operation and must be practiced.

As previously mentioned a full autonomous takeoff, traffic pattern, and landing approach is well into the "run" phase of the demonstration. It is essential to rehearse all logistics, communication, UA setup, configuration, start up, and ground handling first. Progression to full flight is also relatively simple in theory, but not in practice. As mentioned above all procedures in this section are first drilled extensively under normal Part 107 rules at an acceptable facility. First, every ground maneuver procedure is rehearsed multiple times: UA configuration and ramp start, ramp taxi, and runway taxi. Once initial ground handling tests and communication strategy is practiced, high speed taxi tests can be conducted. These tests are conducted under full manual PIC control and allow the flight crew and KSWO tower operators to adjust to an increasing UA operational tempo. Once high speed taxi is complete, final feedback from flight crew and KSWO personnel is used to prepare and plan for flight.

It can not be stressed enough that actual flight is the final step in a long workup process. Invaluable experience and lessons learned are gained working through the initial stages previously outlined. The proposed flight plan is shown in Figure 5.5. The red polygon boundary is the GNSS geofence designed to keep the UA inside an intended flight area away from people and structures. The geofence boundary is approximately 2 nautical miles from the ramp location at its furthest point. The required geofence boundary to keep the UA flight path away from people and structures makes approach and landing viable using only runway 17. Conditions must be favorable for a right hand traffic pattern to runway 17. The traffic pattern is exactly the

same as demonstrated in this study. Initial pattern altitude is 200 feet when crossing the transition to final over an approach corridor free of obstacles. A 3° glide slope is recommended to the desired landing target. PIC has necessary tools and telemetry to determine if the approach needs to be modified on first attempt via methods similar to the auto land traffic pattern script developed and demonstrated as part of this study. The purple waypoints in Figure 5.5 are example common rally points that can be used by any member of the flight crew to quickly reference instructions. Each rally point is assigned a phonetic call sign, i.e. Alpha, Bravo, etc. For example, KSWO tower could then provide instructions for PIC to hold at rally point Alpha to allow fixed wing traffic to pass or land. The boxed blue areas in Figure 5.5 are locations of the remote visual observers. Examples of nominal operational procedural flow is shown in the next paragraph.

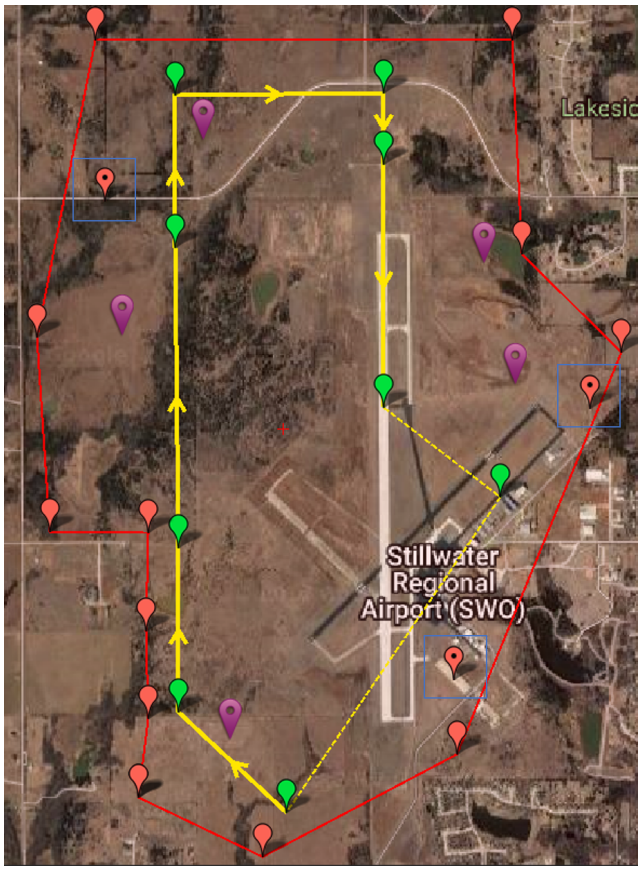

Figure 5.5: KSWO UA Flight Plan

The nominal procedural flow developed for small UAS flight operations is a detail oriented, collaborative effort modeled and adapted from existing manned aircraft flight operations. Flight logs and configuration management of both hardware and software is necessary to document airworthiness and incremental changes of the proposed Penguin-B UA as its test program progresses towards KSWO operations. For each configuration entry, a separate document should contain details as to how and why the change was performed, its intended impact on the system, and the change authority.

A detailed day of operations preparatory briefing is outlined in the Appendix, this is mandatory before any operations, even for the first electrical and data connection test of the UA at KSWO. The flight or test card should be briefed by the PIC to the flight crew or test director prior to takeoff. A flight/test card is a one to two page document that organizes all relevant information pertaining to the flight session. Test objectives should be listed along with current weather conditions, flight limits or go, no go criteria, aircraft configuration, weight and balance, and test procedure. A sample test card is shown in Figure 5.6.

After the test card is briefed, the flight crew moves to all necessary preflight setup and checklists. Five operating checklists were developed for the Anaconda in this study: preflight, autopilot configuration, after start, before takeoff, and before landing. The checklists are modeled after call and response crewed aircraft checklists. This style requires the operators to "be in the loop" at all times and promotes cross checking of critical actions prior to key stages of the flight. For example, the before landing checklist is shown in Figure 5.7. Sample Anaconda documentation shown in the Appendix is suitable as a baseline for Penguin-B KSWO documentation development.

Test Sortie:	AN2 Auto Takeoff / Landing Baseline		
Aircraft: AN2 SF11	**Date/Location:** 13 MAY 17 UAFS	**Flight #:** 1	**Supervising Test Engineer:** ZPB

Test Objectives:	**Weather:**	**Limits:**
1. Autopilot Auto Takeoff/Auto Landing Baseline	Wind: Temperature: Barometric Pressure:	1. < 400 ft. AGL 2. < 13 kt winds 3. TOT < 25 minutes

Configuration 002:	**Notes:**
1. ArduPlane 3.7.1 firmware 2. Pixhawk PX4 3. GPS 4. Airspeed Sensor 5. SF11 Laser Altimeter 6. Manual, FBW-A, RTL, AUTO	NOTE START/END BATTERY VOLTAGES REFER TO PATTERN FLIGHT PLAN ATTACHED ATTACH FLIGHT LOGS TO THIS TEST CARD POST-FLIGHT **RECORD TAKEOFF / LANDING DATA AS SPECIFIED BY TEST CARD ATTACHMENT**
Frequencies: PIC RF: 2.4Ghz GCS RF: 915Mhz	**Weight & Balance:** **TOW lbs: 9.4** CG Measured (est.): 25% CG Calculated: 28%

PROCEDURE

Start Time:		**End Time:**
1.	Verify **PRE-FLIGHT for auto takeoff / land,**	
2.	Engage **AUTO mode for takeoff roll**, mark start time on ground roll	
3.	Monitor ground roll, execute **ABORT S_1 CRITERIA if necessary**	
4.	Monitor transition to **CROSSWIND LEG, HARDECK > 75 ft AGL**	
5.	Monitor **AUTO** Waypoint Course (1 complete traffic pattern)	
6.	Monitor traffic pattern to **FINAL approach**	
7.	Verify 'stable' approach, PIC and Test Director discretion for go around	
8.	Call landing before at or before 50 ft. AGL	
9.	Monitor **S_3 ABORT CRITERIA** and telemetry output for auto land	

# Takeoff / Landing	**Comments**	**Issues**

Figure 5.6: Flight Plan Card or Flight Test Card Sample

ANACONDA FCOM – REV 3 20161120

BEFORE PATTERN ENTRY

MISSION PLANNER

DISTANCE UNITS ..... SET FEET

SPEED UNITS ..... SET KNOTS

ALT MODE ..... SET RELATIVE

RALLY POINT ..... SET +50FT PATTERN ALT

FAILSAFE ..... CONFIGURED

LANDING PATTERN ..... VERIFY & RECEIVED

TRAFFIC ..... CLEAR

AIRCRAFT STATUS

GNSS ..... VERIFY 3D FIX

AIRSPEED ..... VERIFY HEALTHY

COMPASS ..... VERIFY HEALTHY

TELEMETRY ..... VERIFY C2 LINK QUALITY

ARTIFICIAL HORIZON ..... VERIFY HEALTHY

EKF ..... VERIFY HEALTHY

BEFORE LANDING

CROSSING SHORT FINAL FIX ..... +/- 10 FT ALT

BARO BUMP ..... STABLE

LASER ALTIMETER ..... VERIFY HEALTY

APPROACH ..... ON CENTERLINE, GLIDEPATH STABLE

RUNWAY ..... CLEAR

CONTINGENCY ..... GO AROUND

INITIATE GO AROUND

GLIDEPATH DEVIATION THRESHOLD ..... AUTOPILOT TRIGGER

C2 ACTION ..... ABORT LANDING

***OR* THR POSITION ..... >90%**

***OR* FLT MODE ..... TOGGLE**

Figure 5.7: Before Landing Checklist, Anaconda

FAA Part 107 Waivers

Under current FAA Part 107 Small UAS rules, operation in Class D airspace is prohibited without a relevant waiver exempting the controlled airspace restriction. As mentioned in Chapter 2, UAS facility maps will be published surrounding certain low volume controlled airspace. However, these facility maps are intended to quickly approve low altitude operations that maintain a stand off radius from an active airport facility. Several UTM commercial partners have also announced direct integration with FAA and airports to grant real time low altitude airspace authorization [44, 45]. The KSWO UAS demo involves direct use on ramps, taxiways, runways, and flight within the traffic pattern. Consequently, much more risk reduction, justification, and planning will be necessary to facilitate two Part 107 waivers for operation at KSWO.

There are explicit FAA guidelines for each waiverable section of Part 107 that provide recommendations for applicants [46]. The methodology outlined in this proposal is specifically designed to increase UAS flight safety and decrease operational risk. Close coordination between small UAS operators, KSWO airport management, and tower controllers is essential to proving the concept. Flight operations will not be conducted without detailed planning, feedback, and endorsement from KSWO. Part §107.41 and §107.31 are aggressive measures designed to reduce risk for small UAS operations. Section §107.41 prohibits flight in controlled airspace. Section §107.31 specifies that an operator or visual observer must maintain unaided visual line of sight of the small UAS and determine its altitude, heading, and attitude. Operations will be in compliance with all other provisions of Part 107. The specific waiver guidelines are described below and addressed with proposed justification and mitigation strategy.

1. §107.31 Visual line of sight aircraft operation

   (a) Provide the method by which the remote pilot will be able to continuously know and determine the position, altitude, attitude, and movement of their small unmanned aircraft and ensure the aircraft remains in the area of intended operation without exceeding the performance capabilities of the command and control link.

      i. **UAS will return to a predetermined rally point inside flight area if geofence boundary is breached**

      ii. **Telemetry command and control data link has demonstrated additional performance margin 2 nautical miles beyond intended geofence flight area**

      iii. **If telemetry command and control data link is lost, aircraft will return to a predetermined rally point inside intended flight area**

      iv. **If manual override command and control data link is lost, UA will return to a predetermined rally point inside intended flight area**

      v. **If both telemetry and manual override data links are lost, UA will automatically enter traffic pattern and land**

      vi. **ADS-B also provides position data as a backup, but is primarily used to avoid conflict with manned aircraft traffic**

      vii. **Visual observers will be placed along the anticipated traffic pattern flight plan for risk reduction and contingency, but are not primary method for determining aircraft altitude, heading, and attitude.**

   (b) Provide a method for the remote pilot to avoid other aircraft, flying over/into

people on the ground, and ground-based structures and obstacles at all times.

i. **Geofence flight area avoids ground based structures and people**

ii. **UA flight operations conducted during pre-arranged low traffic density windows for complete cooperation of KSWO management and tower controllers**

iii. **KSWO Class D airspace NOTAM for limited UAS operational windows**

iv. **KSWO tower will periodocally broadcast countdown alerts on control frequency before commencing and at conclusion of UAS operations**

v. **KSWO tower will alert traffic of UAS operational status upon initial radio contact**

vi. **UAS PIC will acknowledge KSWO tower instructions over frequency per standard procedure and will also announce flight intentions and position over frequency when possible (avoid radio clutter)**

vii. **UAS broadcasts ADS-B out messages well beyond KSWO Class D airspace boundary**

viii. **UAS ADS-B In and automatic avoidance capability**

ix. **KSWO tower controllers manage control frequency and provide instructions to vector UA and any manned traffic**

x. **Ground based visual observers can also alert PIC to traffic**

(c) Provide a method to increase conspicuity of the small unmanned aircraft to be seen at a distance of at least 3 statute miles unless a system is in

place that can avoid all non-participating aircraft.

i. **Class D airspace requires radio contact prior to entry**

ii. **UA operations conducted with tower control staffing**

iii. **UAS ADS-B In/Out and automatic avoidance capability**

iv. **UAS equipped with daytime navigation and strobe lights visible to 3 statute miles**

(d) Provide a method by which the remote pilot is alerted of a degraded small unmanned aircraft system function.

i. **PIC and flight team monitor telemetry alerts regarding flight critical systems such as engine fuel flow, battery voltages, and autopilot status**

ii. **Ground based visual observers and KSWO tower spotters can alert PIC to degraded system function, i.e. engine loss**

(e) Provide a method to assure all required persons participating in the operation have relevant knowledge of all aspects of operating a small unmanned aircraft that is not in visual line of sight of the remote pilot.

i. **PIC is a commercially rated manned aircraft pilot with *XX.XX* number of flight hours and *XX.XX* number of UAS flight hours**

ii. **Flight test engineers specialize in UAS systems engineering, have remote pilot certificates, and *XX.XX* number of UAS operational flight hours**

iii. **UA Crew Chief specializes in UAS systems engineering, has remote pilot certificate, and *XX.XX* number of UAS operational flight hours**

iv. **Ground based visual observers have at a minimum, remote pilot certificates and *XX.XX* number of UAS operational flight hours**

v. **KSWO Control Tower Operators hold Control Tower Operator Certificates**

2. §107.41 Operation in certain airspace

(a) Provide a method to ensure the small unmanned aircraft will operate safely and efficiently within the specified controlled airspace without obtaining prior authorization from Air Traffic Control.

i. **Overall plan and strategy designed to mitigate risk as much as possible and facilitate safe and efficient flight, but is only conducted under direct guidance from KSWO Tower Control.**

(b) Provide contact instructions for ATC in case the operation needs to be terminated.

i. **KSWO control tower has authority to terminate operations at any time for any reason**

ii. **UAS PIC and KSWO control tower in constant two way radio communication**

APPENDIX A

Appendix

A.1 Autoland Traffic Pattern Script

```
# -*- coding: utf-8 -*-
"""
Created on July 5 13:21 2017

@author: Notsure
"""

import os
import datetime
import socket
import sys
import math

from math import sqrt
import clr
import time
import re, string
import MissionPlanner
clr.AddReference("MissionPlanner.Utilities") #includes the Utilities class
clr.AddReference("MAVLink") # includes the Utilities class
import MAVLink
from MissionPlanner.Utilities import Locationwp
from MissionPlanner.Utilities import PointLatLngAlt
from MissionPlanner.GCSViews import FlightPlanner
#adding external library locations
#RH workstation
#sys.path.append("C:\Anaconda\Lib\site-packages")

#notsure
sys.path.append("D:\Anaconda2\Lib\site-packages")
#AutoHotKey for the functionality behind private classes that cannot be
 ↪  accessed by Python
import ahk
import csv
import glob
```

```
35 import mmap
36
37 #getting file path for summary text file output
38 dir_path = os.path.dirname(os.path.realpath(__file__))
39
40
41
42
43 #functions ================================================
44
45
46 #extrapolate latitude/longitude given a heading and distance
47 #http://www.movable-type.co.uk/scripts/latlong.html
48 def newpos(bearing,distance,lat,lng):
49         lat1 = math.radians(lat)
50         lon1 = math.radians(lng)
51         brng = math.radians(bearing)
52         dr = distance / 6378100.0 # / radius of earth in meters
53
54         lat2 = math.asin(math.sin(lat1) * math.cos(dr) + math.cos(lat1) *
          ↪  math.sin(dr) * math.cos(brng))
55         lon2 = lon1 + math.atan2(math.sin(brng) * math.sin(dr) *
          ↪  math.cos(lat1), math.cos(dr) - math.sin(lat1) * math.sin(lat2))
56
57         lat_out = math.degrees(lat2)
58         lng_out = math.degrees(lon2)
59         return lat_out,lng_out
60
61 #clear active flight plan -using AHK
62 def clear_flight_plan_active():
63         MissionPlanner.MainV2.instance.FlightPlanner.Activate() y
64         print 'Start approach procedure, clearing flight plan screen'
65         Script.Sleep(3000) #sleep in ms
66
67         #ahk call to clear flight plan
68         #initialize the ahk script thread
69         autohotkey_script = ahk.Script()
70         #activating mission planner window
71         autohotkey_script.click()
72         Script.Sleep(350)
73         #switching to flight plan screen
74         autohotkey_script.send("{F3}","Send")
75
76         Script.Sleep(350)
77         #right click flight plan map
78         autohotkey_script.click("right",1,889,147)
79         Script.Sleep(350)
80         #clear flight plan
```

```
81          autohotkey_script.click("",1,971,356)
82          Script.Sleep(350)
83
84          MissionPlanner.MainV2.instance.FlightPlanner.fullpointlist.Clear()
85
86  #clearing user LZ selection
87  def clear_flight_plan_user_LZ():
88
89          autohotkey_script = ahk.Script()
90          #right click flight plan map
91          autohotkey_script.click("right",1,889,147)
92          Script.Sleep(500)
93          #clear flight plan
94          autohotkey_script.click("",1,971,356)
95          Script.Sleep(500)
96
97  #generate the left hand traffic pattern from LZ selection
98  def left_hand_traffic_pattern(LZ_lat,LZ_lng):
99
100         # 0 - lat, 1 - lng
101         #left hand traffic pattern
102         #short final
103         short_final = newpos(0,255,LZ_lat,LZ_lng)
104         #final
105         final = newpos(0,200,short_final[0],short_final[1])
106         #basegen
107         base = newpos(90,200,final[0],final[1])
108         #downwind/IAF
109         downwind = newpos(180,445,base[0],base[1])
110
111         #adding to FP - distance argument is in units as selected on MP
112         #use feet because no one uses meters in aviation/aeronautics...
113         #downwind/IAF
114         #Syntax is shortened for inclusion into appendix
115         #As follows:
            ↪  MissionPlanner.MainV2.instance.FlightPlanner.InsertCommand
116         InsertCommand(1,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,downwind[1],downwind[0],200)
117         #base
118         InsertCommand(2,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,base[1],base[0],150)
119         #final
120         InsertCommand(3,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,final[1],final[0],125)
121         #short final
122         InsertCommand(4,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,short_final[1],short_final[0],80)
123         #LZ
124         InsertCommand(5,MAVLink.MAV_CMD.LAND,0,0,0,0,LZ_lng,LZ_lat,0)
125         InsertCommand(6,MAVLink.MAV_CMD.CONTINUE_AND_CHANGE_ALT,1,0,0,0,0,0,100)
126         InsertCommand(7,MAVLink.MAV_CMD.DO_JUMP,1,-1,0,0,0,0,0)
127
```

```
128 #generate the right hand traffic pattern from LZ selection
129 def right_hand_traffic_pattern(LZ_lat,LZ_lng):
130
131         # 0 - lat, 1 - lng
132         #right hand traffic pattern
133         #short final
134         short_final = newpos(180,255,LZ_lat,LZ_lng)
135         #final
136         final = newpos(180,200,short_final[0],short_final[1])
137         #base
138         base = newpos(90,200,final[0],final[1])
139         #downwind/IAF
140         downwind = newpos(0,445,base[0],base[1])
141
142         #adding to FP - distance argument is in units as selected on MP
143         #use feet because no one uses meters in aviation/aeronautics...
144         #downwind/IAF
145         #Syntax is shortened for inclusion into appendix
146         #As follows:
             ↪  MissionPlanner.MainV2.instance.FlightPlanner.InsertCommand
147         InsertCommand(1,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,downwind[1],downwind[0],200)
148         #base
149         InsertCommand(2,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,base[1],base[0],150)
150         #final
151         InsertCommand(3,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,final[1],final[0],125)
152         #short final
153         InsertCommand(4,MAVLink.MAV_CMD.WAYPOINT,0,0,0,0,short_final[1],short_final[0],80)
154         #LZ
155         InsertCommand(5,MAVLink.MAV_CMD.LAND,0,0,0,0,LZ_lng,LZ_lat,0)
156         InsertCommand(6,MAVLink.MAV_CMD.CONTINUE_AND_CHANGE_ALT,1,0,0,0,0,0,100)
157         InsertCommand(7,MAVLink.MAV_CMD.DO_JUMP,1,-1,0,0,0,0,0)
158
159 #generate comprehensive approach summary data
160 def approach_summary(LZ_lat,LZ_lng):
161         #getting file path for summary text file output
162         #dir_path = os.path.dirname(os.path.realpath(__file__))
163         #start approach summary
164         Script.Sleep(8000)
165         count = 0
166         count2 = 0
167         count3 = 0
168         count4 = 0
169         glide_sys_time = datetime.datetime.now().strftime("Autoland
             ↪  glideslope log %Y %m %d %H %M %S")
170         glide_file_path = "%s\%s.txt" % (dir_path,glide_sys_time)
171         glide_file = open(glide_file_path,"w")
172         glide_file.write("Traffic Pattern, %s" % pattern)
173         glide_file.write("\n")
```

```
174         glide_file.close()
175         #glide_target_alt = []
176         #glide_alt = []
177         #glide_sonarrange = []
178         while cs.mode == 'Auto':
179                 if cs.wpno == 5:
180                         glide_file = open(glide_file_path,"a")
181                         glide_file.write("WP_dist, %d," % cs.wp_dist)
182                         glide_file.write("Laser_alt, %d," % cs.sonarrange)
183                         glide_file.write("Baro_alt, %d" % cs.alt)
184                         glide_file.write("\n")
185                         glide_file.close()
186                         #glide_target_alt.append(cs.targetalt)
187                         #glide_alt.append(cs.alt)
188                         #glide_sonarrange.append(cs.sonarrange)
189
190
191                 if cs.wpno == 5 and count == 0:
192                         count = count + 1
193                         sys_time =
                         ↪  datetime.datetime.now().strftime("Autoland log
                         ↪  %Y %m %d %H %M %S")
194                         gpstime_short_final = cs.gpstime
195                         alt_short_final = cs.alt
196
197                         #getting target wp's for summary
198                         short_final_target_lat =
                         ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[4].Lat
199                         short_final_target_lng =
                         ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[4].Lng
200                         short_final_actual_lat = cs.lat
201                         short_final_actual_lng = cs.lng
202
203                         short_final_wind_dir = cs.wind_dir
204                         short_final_wind_vel = cs.wind_vel
205                         #gpstime_short_final = cs.gpstime
206                         file_path = "%s\%s.txt" % (dir_path,sys_time)
207                         f = open(file_path, "w")
208                         f.write("GPS Time Short Final, %s" %
                         ↪  gpstime_short_final)
209                         f.write("\n")
210                         f.write("Traffic Pattern Type, %s" % pattern)
211                         f.write("\n")
212                         f.write("Target Lat Short Final, %f" %
                         ↪  short_final_target_lat)
213                         f.write("\n")
214                         f.write("Target Lng Short Final, %f" %
                         ↪  short_final_target_lng)
```

```
                    f.write("\n")
                    f.write("Actual Lat Short Final, %f" %
                    ↪  short_final_actual_lat)
                    f.write("\n")
                    f.write("Actual Lng Short Final, %f" %
                    ↪  short_final_actual_lng)
                    f.write("\n")
                    f.write("Altitude Short Final, %d" %
                    ↪  alt_short_final)
                    f.write("\n")
                    f.write("Wind Direction Short Final, %d" %
                    ↪  short_final_wind_dir)
                    f.write("\n")
                    f.write("Wind Velocity Short Final, %d" %
                    ↪  short_final_wind_vel)
                    f.close()

                if cs.ch3percent <= 5 and count2 == 0 and cs.wpno == 5 and
                ↪  cs.sonarrange <= 5 and cs.alt <= 5:
                    count2 = count2 + 1
                    arspd_flare_target = cs.targetairspeed
                    arspd_flare_actual = cs.airspeed
                    flare_wind_dir = cs.wind_dir
                    flare_wind_vel = cs.wind_vel
                    sink_rate_flare = cs.verticalspeed
                    alt_flare = cs.alt
                    throttle_flare = cs.ch3percent
                    alt_flare_laser = cs.sonarrange
                    LZ_flare_lat = cs.lat
                    LZ_flare_lng = cs.lng
                    gpstime_flare = cs.gpstime
                    flare_stats = cs.messageHigh
                    f2 = open(file_path, "a")
                    f2.write("\n")
                    f2.write("\n")
                    f2.write("GPS Time Flare, %s" % gpstime_flare)
                    f2.write("\n")
                    f2.write("Flare Baro Alt, %d" % alt_flare)
                    f2.write("\n")
                    f2.write("Flare Laser Alt, %d" % alt_flare_laser)
                    f2.write("\n")
                    f2.write("Throttle percent flare, %d" %
                    ↪  throttle_flare)
                    f2.write("\n")
                    f2.write("Flare Lat, %f" % LZ_flare_lat)
                    f2.write("\n")
                    f2.write("Flare Lng, %f" % LZ_flare_lng)
                    f2.write("\n")
```

```
256                         f2.write("Flare Target Airspeed, %d" %
                            ↪  arspd_flare_target)
257                         f2.write("\n")
258                         f2.write("Flare Actual Airspeed, %d" %
                            ↪  arspd_flare_actual)
259                         f2.write("\n")
260                         f2.write("Flare Wind Direction, %d" %
                            ↪  flare_wind_dir)
261                         f2.write("\n")
262                         f2.write("Flare Wind Velocity, %d" %
                            ↪  flare_wind_vel)
263                         f2.write("\n")
264                         f2.write("Sink Rate, %d" % sink_rate_flare)
265                         f2.write("\n")
266                         f2.write("ArduPlane returned landing stats, %s" %
                            ↪  flare_stats)
267                         f2.close()
268
269                 if cs.sonarrange <= 1.5 and cs.wpno == 5 and count3 == 0:
270                         count3 = count3 + 1
271                         gpstime_LZ = cs.gpstime
272                         arspd_lz_target = cs.targetairspeed
273                         arspd_lz_actual = cs.airspeed
274                         LZ_sinkrate = cs.verticalspeed
275                         LZ_actual_lat = cs.lat
276                         LZ_actual_lng = cs.lng
277                         LZ_laser_alt = cs.sonarrange
278                         LZ_alt = cs.alt
279                         LZ_vibez = cs.vibez
280                         f3 = open(file_path, "a")
281                         f3.write("\n")
282                         f3.write("\n")
283                         f3.write("GPS Time Touchdown, %s" % gpstime_LZ)
284                         f3.write("\n")
285                         f3.write("Touchdown Lat, %f" % LZ_actual_lat)
286                         f3.write("\n")
287                         f3.write("Touchdown Lng, %f" % LZ_actual_lng)
288                         f3.write("\n")
289                         f3.write("Target Lat, %f" % LZ_lat)
290                         f3.write("\n")
291                         f3.write("Target Lng, %f" % LZ_lng)
292                         f3.write("\n")
293                         f3.write("Laser Alt, %d" % LZ_laser_alt)
294                         f3.write("\n")
295                         f3.write("Baro Alt, %d" % LZ_alt)
296                         f3.write("\n")
297                         f3.write("VibeZ, %d" % LZ_vibez)
298                         f3.write("\n")
```

```
299                         f3.write("Airspeed Target, %d" % arspd_lz_target)
300                         f3.write("\n")
301                         f3.write("Airspeed Touchdown, %d" %
                             ↪  arspd_lz_actual)
302                         f3.write("\n")
303                         f3.write("Sink rate at Touchdown, %d" %
                             ↪  LZ_sinkrate)
304                         f3.close()
305
306                 if cs.landed == 'True' and cs.groundspeed <= 0.2 and count4
                     ↪  == 0:
307                         count4 = count4 + 1
308                         gpstime_rollout = cs.gpstime
309                         LZ_groundspeed = cs.groundspeed
310                         Rollout_lat = cs.lat
311                         Rollout_lng = cs.lng
312                         f4 = open(file_path, "a")
313                         f4.write("\n")
314                         f4.write("\n")
315                         f4.write("GPS Time Wheelstop, %s" %
                             ↪  gpstime_rollout)
316                         f4.write("\n")
317                         f4.write("Groundspeed at Wheelstop, %d" %
                             ↪  LZ_groundspeed)
318                         f4.write("\n")
319                         f4.write("Lat Wheelstop, %f" % Rollout_lat)
320                         f4.write("\n")
321                         f4.write("Lng Wheelstop, %f" % Rollout_lng)
322                         f4.close()
323
324 def select_LZ(traffic_pattern_flag):
325         #clearing active flight plan
326         clear_flight_plan_active()
327         print 'Select the LZ'
328         Script.Sleep(7000)
329         #make sure LZ is selected, explicitly checking if a waypoint was
             ↪  added on the flight plan map
330         #the desired "LZ", reason > 1 is because home counts as 0, but does
             ↪  not appear as an entry on the datagrid
331         if MissionPlanner.MainV2.instance.FlightPlanner.pointlist.Count >
             ↪  1:
332                 print 'Reading LZ, generating approach'
333                 LZ_lat =
                     ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[1].Lat
334                 LZ_lng =
                     ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[1].Lng
335                 Script.Sleep(1000)
336                 print "LZ Lat: %f" % LZ_lat
```

```
337                 Script.Sleep(1000)
338                 print "LZ Lng: %f" % LZ_lng
339                 # #clearing LZ selection, prep for approach pattern
340                 clear_flight_plan_user_LZ()
341                 # generating traffic pattern based on wind estimate
342                 if traffic_pattern_flag == 1:
343                         right_hand_traffic_pattern(LZ_lat,LZ_lng)
344                 elif traffic_pattern_flag == 0:
345                         left_hand_traffic_pattern(LZ_lat,LZ_lng)
346                 else:
347                         print 'Help me Tom Cruise'
348                 approach_summary(LZ_lat,LZ_lng)
349           else:
350                 sys.exit("LZ not selected, aborting approach")
351
352
353
354 # end functions ================================================
355 #starting script
356 #wind vector estimate from vehicle, 30 second average
357 print 'Start script, downloading wind estimate from vehicle'
358 #get_wind_estimate()
359 wind_dir = []
360 wind_vel = []
361 arspd = []
362 if math.isnan(cs.wind_vel) or math.isnan(cs.wind_dir) == True:
363         print 'No wind estimate available from vehicle'
364 else:
365         t_end = time.time() + 10
366         while time.time() < t_end:
367                 wind_dir.append(cs.wind_dir)
368                 wind_vel.append(cs.wind_vel)
369                 arspd.append(cs.airspeed)
370
371 wind_dir_est = sum(wind_dir)/len(wind_dir)
372 wind_vel_est = sum(wind_vel)/len(wind_vel)
373 arspd_avg = sum(arspd)/len(arspd)
374 wind_dir_est_integer = int(wind_dir_est)
375
376 if wind_dir_est_integer in range(123,236):
377         print 'Left hand traffic pattern recommended'
378         traffic_pattern_flag = 0 #flag as left hand pattern
379         pattern = "Left"
380 elif wind_dir_est_integer in range(303,360) or wind_dir_est_integer in
    ↪  range (0,65):
381         print 'Right hand traffic pattern recommended'
382         traffic_pattern_flag = 1 #flag as right hand pattern
383         pattern = "Right"
```

```
384 else:
385         print 'Help me Tom Cruise'
386         traffic_pattern_flag = 0 #default to left
387         pattern = "Left"
388
389 print "Wind Direction Estimate [deg]: %d" % wind_dir_est
390 Script.Sleep(1000)
391 print "Wind Velocity Estimate [kts]: %d" % wind_vel_est
392 Script.Sleep(1000)
393 print "Average Airspeed Estimate [kts]: %d" % arspd_avg
394
395 txt_count = 0
396 for file in os.listdir(dir_path):
397     if file.endswith(".txt"):
398             txt_count = txt_count + 1
399
400 print "Log Count: %d" % txt_count
401
402 if txt_count >= 1:
403         print 'Timely autoland approaches detected, checking latest traffic
          ↪  pattern for wind correction'
404         current_approach_log = max(glob.iglob(dir_path + "\*.txt"),
          ↪  key=os.path.getctime)
405         current_approach_log_file = open(current_approach_log, 'r')
406         s = mmap.mmap(current_approach_log_file.fileno(), 0,
          ↪  access=mmap.ACCESS_READ)
407         print current_approach_log
408         match1 = s.find('Left')
409         match2 = s.find('Right')
410         if match1 != -1 and traffic_pattern_flag == 0:
411                 print 'No wind correction needed, proceeding into approach
                  ↪  logging'
412                 LZ_lat =
                  ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lat
413                 LZ_lng =
                  ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lng
414                 approach_summary(LZ_lat,LZ_lng)
415                 print 'done baby done'
416         elif match2 != -1 and traffic_pattern_flag == 1:
417                 print 'No wind correction needed, proceeding into approach
                  ↪  logging'
418                 LZ_lat =
                  ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lat
419                 LZ_lng =
                  ↪  MissionPlanner.MainV2.instance.FlightPlanner.pointlist[5].Lng
420                 approach_summary(LZ_lat,LZ_lng)
421                 print 'done baby done'
422         else:
```

```
423                 print 'Traffic pattern update needed for changing wind
                     ↪  conditions, standby for LZ selection'
424                 select_LZ(traffic_pattern_flag)
425                 print 'done baby done'
426 else:
427         print 'No timely autoland approaches detected, standby for traffic
             ↪  pattern generation'
428         select_LZ(traffic_pattern_flag)
429         print 'done baby done'
430
431 # done baby done
```

A.2 Operator Checklists

ANACONDA FCOM – REV 3 20161120

PREFLIGHT

PROPULSION

PROPELLER	SECURED
MOTOR	SECURED
ESC	LEADS SECURE
BATTERY VOLTAGES	16.8V

FLIGHT CONTROL SURFACES AND SERVOS

SERVOS	SECURE
CONTROL PUSHRODS	FREE
CONTROL CLEVIS	SECURE
CONTROL SURFACES	NO SLOP
STRUCTURAL CONTROL SURFACE HINGES	NO VISIBLE WEAR

LANDING GEAR

STRUCTURE	NO VISIBLE WEAR
ATTACHMENT POINTS	SECURE
WHEELS	FREE

AVIONICS AND PAYLOAD

FLIGHT CONTROLLER	SECURE & POSITIONED
AIRSPEED SENSOR	SECURE & POSITIONED
GPS/COMPASS	SECURE & POSITIONED
LASER ALTIMETER	SECURE & POSITIONED
PITOT-STATIC	SECURE & POSITIONED
RF COMMS	SECURE & POSITIONED

FLIGHT DIRECTOR

PERSONNEL BRIEF	COMPLETED
RANGE AND WEATHER	GO / NO GO

PROCEED WITH CRITICAL PARAMS FCOM ACCORDING TO MISSION OBJECTIVES

GCS	CONFIGURED

Figure A.1: Anaconda Operator Checklist, Preflight Part 1

ANACONDA FCOM – REV 3 20161120

PROCEED WITH ALL NOMINAL AVIONICS CALIBRATION BEFORE CLOSEOUT DURING INITIAL START

NOMINAL CALIBRATIONS BEFORE CLOSEOUT

IMU ..... ALIGNED

COMPASS ..... ALIGNED

MANUAL TX ..... CALIBRATED & RANGE CHECKED

FLIGHT MODES ..... CONFIGURED & CHECKED

FAILSAFE ..... CONFIGURED

AIRSPEED ..... CONFIGURED

LASER ALTIMETER ..... CONFIGURED

CLOSEOUT

CONTROL SURFACE SERVOS ..... CONNECTED

WING AND TAIL STRUCTURE ..... SECURED

PROPULSION BATTERY ..... STRAP & SECURE

AVIONICS BATTERY ..... STRAP & SECURE

TAKEOFF WEIGHT AND CG ..... VERIFY

GROUND START

VEHICLE & PROPELLER ..... CLEAR

TX ..... POWER ON, MANUAL FLT MODE

PROPULSION BATTERY ..... CONNECTED

AVIONICS WARMUP ..... VEHICLE STATIC FOR 60-120 SECONDS

TELEMETRY ..... CONNECTED & RECEIVING

IMU AND GYRO ..... HEALTHY

GPS / COMPASS ..... 3D FIX

EKF AND VIBE ..... HEALTHY

AIRSPEED ..... HEALTHY

BARO ..... HEALTHY

LASER ALTIMETER ..... HEALTHY

BATTERY MONITOR ..... HEALTHY

Figure A.2: Anaconda Operator Checklist, Preflight Part 2

ANACONDA FCOM – REV 3 20161120

CRITICAL PARAMETERS FOR GPS GUIDED AUTO FLT MODE MISSIONS

MISSION PLANNER

DISTANCE UNITS ..... SET FEET

SPEED UNITS ..... SET KNOTS

ALTITUDE MODE ..... SET RELATIVE

RALLY POINT ..... SET 200 FT

FAILSAFE ..... CONFIGURED

FLIGHT PLAN ..... VERIFY GPS COORD. & ALT

MAIN TUNING PARAMETERS

AIRSPEED ..... ENABLED & SET FOR FLT CONTROL

CRUISE / MIN / MAX SPEEDS ..... VERIFY (14 / 11 / 20)

THROTTLE RANGES ..... VERIFY

PID GAINS ..... VERIFY

MIN / MAX CLIM & DESCENT RATES ..... VERIFY

MIN / MAX PITCH & ROLL ANGLES ..... VERIFY

AMPLIFIED PARAMETERS

ARMING CHECKS (ARMING_CHECK) ..... SET 1

AHRS USE GPS (AHRS_GPS_USE) ..... SET 1

ENABLE EKF2 (EK2_ENABLE) ..... SET 1

AHRS EKF2 (AHRS_EKF_TYPE) ..... SET 2

INITIAL FLIGHT MODE (INITIAL_MODE) ..... SET 0

WAYPOINT RADIUS (WP_RADIUS) ..... SET 100

WAYPOINT LOITER RADIUS (WP_LOITER_RAD) ..... SET 200

RTL ALTITUDE (ALT_HOLD_RTL) ..... SET 4575

AUTOMATIC STALL PREVENTION (STALL_PREVENTION) ..... SET 1

TARGET AIRSPEED (TRIM_ARSPD_CM) ..... SET 1400

THROTTLE CRUISE PERCENT (TRIM_THROTTLE) ..... SET 45

THROTTLE NUDGE ENABLE (THROTTLE_NUDGE) ..... SET 1

THROTTLE SLEW RATE (THR_SLEWRATE) ..... SET 100

CRASH DETECTION (CRASH_DETECT) ..... SET 0

Figure A.3: Anaconda Operator Checklist, Autopilot Configuration GPS Guided

ANACONDA FCOM – REV 3 20161120

CRITICAL PARAMS FOR AUTO TAKEOFF / AUTO LAND (CMPLTE GPS WYPT & FS FCOM)

MISSION PLANNER

DISTANCE UNITS ..... SET FEET

SPEED UNITS ..... SET KNOTS

ALTITUDE MODE ..... SET RELATIVE

RALLY POINT ..... SET 200 FT

FLIGHT PLAN

NOMINAL FLT PLAN ..... AS REQUIRED

TAKEOFF PITCH ..... VERIFY 10 DEGREES

TAKEOFF ALTITUDE ..... SET 75 FEET

TOUCHDOWN POINT ..... SET

GLIDE SLOPE ..... VERIFY <= 10%

GO AROUND CRITERIA ..... BRIEF

GO AROUND TRIGGER ..... GCS ABORT / MANUAL THROTTLE > 95%

GO AROUND LOGIC ..... SET CONTINUE_AND_CHANGE_ALT 75 FEET

GO AROUND LOGIC CONT'D ..... SET DO_JUMP WAYPOINT

AUTOLAND TRAFFIC PATTERN SCRIPT ..... AS REQUIRED

AMPLIFIED PARAMETERS TAKEOFF

GROUND STEERING LOOP (STEER2SRV_P) ..... SET 1.8

GROUND STEERING LOOP (STEER2SRV_I) ..... SET 0.2

GROUND STEERING LOOP (STEER2SRV_D) ..... SET 0.005

GROUND STEERING LOOP (STEER2SRV_IMAX) ..... SET 1500

GROUND STEERING LOOP (STEER2SRV_MINSPD) ..... SET 1

GROUND STEERING LOOP (STEER2SRV_TCONST) ..... SET 0.75

GROUND STEERING LOOP (GROUND_STEER_ALT) ..... SET 5

TAKEOFF CONTROL LOOP (TKOFF_THR_SLEW) ..... SET 35

TAKEOFF CONTROL LOOP (TKOFF_THR_MAX) ..... SET 100

VERIFY TECS THROTTLE MAX (THR_MAX) ..... SET AS REQUIRED

TAKEOFF CONTROL LOOP (TKOFF_ROTATE_SPD) ..... SET 12.5

CLIMB OUT PITCH ANGLE (TECS_PITCH_MAX) ..... SET 20

Figure A.4: Anaconda Operator Checklist, Autopilot Configuration Takeoff and Landing, Part 1

ANACONDA FCOM – REV 3 20161120

AMPLIFIED PARAMETERS LANDING

LASER ALTIMETER (RNGFND_LANDING).....SET 1

LASER ALTIMETER (RNGFND_MAX_CM).....SET 4000

LASER ALTIMETER (RNGFND_PIN).....SET 14

LASER ALTIMETER (RNGFND_SCALING).....SET 12.12

LASER ALTIMETER (RNGFND_TYPE).....SET 1

LASER ALTIMETER (RNGFND_RMETRIC).....SET 0

LANDING CONTROL LOOP (LAND_ABORT_THR).....SET 1

LANDING CONTROL LOOP (LAND_FLARE_SEC).....SET 1.5

LANDING CONTROL LOOP (LAND_FLARE_ALT).....SET 2

LANDING CONTROL LOOP (TECS_LAND_ARSPD).....SET 13.5

LANDING CONTROL LOOP (TECS_LAND_SPDWGT).....SET 1

LANDING CONTROL LOOP (THR_MIN).....SET 0

LANDING CONTROL LOOP (TECS_LAND_SINK).....SET 0.25

LANDING CONTROL LOOP (LAND_PITCH_CD).....SET 275

LANDING CONTROL LOOP (TECS_LAND_DAMP).....SET 0.5

LANDING CONTROL LOOP (LEVEL_ROLL_LIMIT).....SET 5 DEG

LANDING CONTROL LOOP (LAND_DISARMDELAY).....SET 20

Figure A.5: Anaconda Operator Checklist, Autopilot Configuration Takeoff and Landing, Part 2

ANACONDA FCOM – REV 3 20161120

AFTER START

GCS

RALLY POINT AND GPS HOME LOCATION ..... SET

FLIGHT PLAN ..... RECEIVED

FLIGHT CONTROL SURFACES AND SERVOS

PERSONNEL ..... **CLEAR**

MANUAL FLT MODE ..... **SET**

GCS TELEM ..... CHECK MANUAL FLT MODE

SAFETY SWITCH ..... **ARM**

ESC ..... ARMED & HEALTHY

CONTROL SURFACES ..... CHECK TRAVEL

ROLL, L/R ..... CHECK

PITCH, U/D ..... CHECK

YAW ..... CHECK

NOSE STEERING ..... CHECK

THROTTLE ..... **CHECK**

FBW-A FLT MODE ..... **SET**

ROLL, L/R ..... CHECK

PITCH, U/D ..... CHECK

YAW, RUDDER CORRECTS ROLL ..... CHECK

THROTTLE ..... **CHECK**

MANUAL FLT MODE ..... **SET**

SAFETY SWITCH ..... **DISARM**

AIRSPEED AND BARO ..... DO ACTION PREFLIGHT CALIBRATION

BEFORE TAKEOFF

RANGE AND WEATHER ..... GO / NO GO

FLIGHT DIRECTOR ..... GO / NO GO

SAFETY SWITCH ..... **ARM**

GCS HUD SCAN ..... CLEAR & HEALTHY

Figure A.6: Anaconda Operator Checklist, Before Takeoff

ANACONDA FCOM – REV 3 20161120

BEFORE PATTERN ENTRY

MISSION PLANNER

DISTANCE UNITS	SET FEET
SPEED UNITS	SET KNOTS
ALT MODE	SET RELATIVE
RALLY POINT	SET +50FT PATTERN ALT
FAILSAFE	CONFIGURED
LANDING PATTERN	VERIFY & RECEIVED
TRAFFIC	CLEAR

AIRCRAFT STATUS

GNSS	VERIFY 3D FIX
AIRSPEED	VERIFY HEALTHY
COMPASS	VERIFY HEALTHY
TELEMETRY	VERIFY C2 LINK QUALITY
ARTIFICIAL HORIZON	VERIFY HEALTHY
EKF	VERIFY HEALTHY

BEFORE LANDING

CROSSING SHORT FINAL FIX	+/- 10 FT ALT
BARO BUMP	STABLE
LASER ALTIMETER	VERIFY HEALTY
APPROACH	ON CENTERLINE, GLIDEPATH STABLE
RUNWAY	CLEAR
CONTINGENCY	**GO AROUND**

INITIATE GO AROUND

GLIDEPATH DEVIATION THRESHOLD	**AUTOPILOT TRIGGER**
C2 ACTION	**ABORT LANDING**
***OR* THR POSITION**	**>90%**
***OR* FLT MODE**	**TOGGLE**

Figure A.7: Anaconda Operator Checklist, Before Landing

A.3 Flight Time Log and Configuration Management

Flight Log

Aircraft: **ANACONDA (N538UA)**

Date	Location	# T/O & Ldg	TOW (lbs)	Conditions	Configuration	PIC	Flight Time (HH:MM)
160216	UAFS	2	8.36	8-11 kts.WNW	#000 (AN1)	ZPB/SW	00:5
160219	UAFS	7	8.36	0-8 kts. NNW	#001 (AN1)	ZPB/SW	00:15
160225	UAFS	4	8.36	8-13 kts. NNW	#002 (AN1)	ZPB/SW	00:20
160304	UAFS	7	9.2	8.5-17 kts. S	#003 (AN1)	ZPB/SW	00:20
160712	UAFS	5	9.2	10 kts S	#003 (AN1)	TM/SW	00:45
160929	UAFS	7	9.3	8 kts SSE	#001 (AN2)	ZPB/SW	00:30
161006	UAFS	13	9.3	6 kts S	#001 (AN2)	ZPB/SW	00:40
161110	UAFS	14	9.3	5 kts S	#001 (AN2)	ZPB	01:00
161116	UAFS	18	9.2	7-9 kts S	#001 (AN2)	ZPB	01:30
161204	UAFS	2	9.3	5-8 kts S	#001 (AN2)	ZPB	00:25
170314	UAFS	6	9.3	5-8 kts S/SSE	#001 (AN2)	ZPB/SW	01:20
170513	UAFS	5	9.3	4-7 kts S/SSE	#001 (AN2)	ZPB/MH	00:30
170522	UAFS	6	9.3	8-12 kts S	#001 (AN2)	ZPB/MH	00:35
Cumulative Flight Time This Sheet (HH:MM)							**08:15**

Figure A.8: Anaconda Flight Log

Configuration Change Tracking

Change No.	Date	Description	Weight Change	Authorized	Signature
#000 (AN1)	160216	Initial Prototype	8.36	ZPB	ZPB
#001 (AN1)	160219	Removed Lidar Lite	8.35	ZPB	ZPB
#002 (AN1)	160224	Swapped PX4, telem. cable	8.35	ZPB	ZPB
#003 (AN1)	160302	Installed SF02 Laser Altimeter, RFD900+ GCS C2 Link, 15x4E prop	9.2	ZPB	ZPB
#004 (AN1)	161110	3DR GCS C2 Link, 15x4E pusher prop, PID gain tune, autoland param tune	9.2	ZPB	ZPB
#001 (AN2)	160331	Initial Config, SF-11C serial laser altimeter slightly heavier due to paint and additional wiring harnesses	9.3	ZPB	ZPB

Figure A.9: Anaconda Configuration Tracking

#004 160302 – 3DR C2 GCS Link, 15x4E APC prop, PID gain tune, autoland param tune

3DR GCS C2 link reinstated resulting in poor performance. Bermuda triangle / null zone approximately 1000 feet north of UAFS during a nominal left hand traffic pattern landing to the south. C2 link consistently degraded and sometimes completely dropped at the same location, turn to base. Without diagnosing airframe internal interference, antenna configuration, etc. recommend to switch back to higher RF power RFD900+ GCS C2 link.
15x4E pusher prop installed. 2/3 motor leads were swapped to change the direction of the motor to accommodate pusher prop.

PID gain tune using AUTOTUNE flight mode was performed. Vehicle exhibited oscillations during approach and landing using recent gain tune from AN2. There are very slight differences in weight and control surface authority between each airframe—AN1 and AN2. These slight differences manifested as undamped oscillatory behavior during higher wind speed approaches >= 10 kts. After gains were re-tuned, aircraft performance looked, subjectively, much more stable. Recommend to AUTOTUNE each and every individual airframe, even of the same type.

Autoland specific parameters were tuned. In high wind speed scenarios, the aircraft is constantly adjusting power and attitude to stay on glide slope. In most situations, the most stable approach configuration is to weight airspeed error as static, TECS_SPDWEIGHT = 1.0, without dynamic weighting enabled (-1). TECS_SPDWEIGHT = ~0.2 to force airspeed target management using throttle not pitch to reduce oscillatory pitch response. To use the dynamically weighted setting or throttle airspeed error correction (0.2), the PID loops must be tuned manually for more performance under low speed, high dynamic environments. Flare pitch should be kept low to prevent prop strikes and close to the ground which results in LAND_FLARE_TIME = 1.0 sec and LAND_PITCH_CD = 275 centi-degrees. This configuration keeps the speed up slightly and flares less aggressively and lower to the ground to prevent ballooning. In general, a three point landing results with the adequate performance in regards to desired touchdown point.

Figure A.10: Anaconda Configuration Tracking, Detailed Entry Sample

A.4 KSWO Preflight Briefing Outline

PRE-FLIGHT

General/Admin:

- Date/Flight no./test no.
- Time hack.
- Purpose of test.
- Roll call/Crew Info/call sign(s) (including chase crew and telemetry (TM) room).
  - Pilot-in-command and rules for in-flight changes.
  - Test Director.
  - Flight Test Engineers, FTE(s)/specialists (instrumentation, photographer, etc.).
  - Visual Observer(s).
- Emergency duties.
- Newcomers to aircraft? Arrange safety briefing.
- Crew rest, crew duty day (flight crew and TM room participants).
- Crew currency and qualification.
- Personal safety equipment
- Ground personnel (including TM room) responsibilities.
- Crash recovery personnel and responsibilities.
- ATC/Range Coordination

Safety Review Board completed (if applicable).

Review takeoff time, crew show-time, chase check-in time, range time.

Test aircraft (A/C) configuration and status:

- A/C info (type/model/serial number/registration/…).
- Instrumentation requirements and status special equipment (e.g., smoke generator, CO monitor, portable O2, cone, load banks, ice probes).
- Inoperative systems for special test configurations (e.g single engine.).
- Open maintenance items
- Temporary operating limitations.
- Instrument calibrations (e.g., pitot-static).
- Changes since last flight (e.g., maintenance, instrumentation, software, cg).
- Weight and Balance:
  - Takeoff & target gross weight & cg for test.
  - Ballast configuration and movement.
- Fuel on board.
- Thrust rating.

Figure A.11: KSWO Detailed Preflight Brief Outline, Adapted from OSU UAS Flight Test and Certification Graduate Course, Part 1

Local Info:

- Aircraft performance versus takeoff conditions.
- Communications: primary/secondary/emergency/test frequencies.
- Mission profile.
- Weather:
  - Current.
  - Go/no-go and/or requirements.
  - Forecast for test area and destination/alternate.
- NOTAMS.
- Fuel requirements (return to base/min on deck).
- Recovery and landing.
- Expected landing time.

Test Condition, Flight Plan Details:

- Flight test plan or flight plan reviewed.
- Previous lessons learned reviewed.
- Previous flight test reports reviewed.
- Detailed review of flight plan cards:
  - Who will fly and exchange of control.
  - Initial conditions/set-up.
  - Limits (Airspeed, Altitude, GW/cg, Wind, etc.).
  - Review of flight test technique or flight plan procedures.
  - Special test limitations.
  - Buildup to final conditions.
  - Test predictions.
  - Review of company tests.
  - Instrumentation/data requirements.
  - Knock-it-off criteria and procedures (including ground personnel).
  - Crew Resource Management (Who is watching what. Who makes what calls).
  - Review unique recovery/emergency procedures.
  - ***Review risk assessment.***

Emergencies/Contingencies:

- Emergency recovery procedures (primary/secondary) (e.g., engine loss, comm link loss, minimum, call-outs, non-cooperative traffic).
- Emergency personnel and equipment (fire truck, EMS, etc.)
- Crew Resource Management during contingencies

Figure A.12: KSWO Detailed Preflight Brief Outline, Adapted from OSU UAS Flight Test and Certification Graduate Course, Part 2

BIBLIOGRAPHY

[1] Federal Aviation Administration Forecasts and Performance Analysis Division (APO-100), "FAA Aerospace Forecast Fiscal Year 2017-2037," *TC17-0002*, 2017.

[2] U.S. Department of Transportation, John A. Volpe National Transportation Systems Center, "Unmanned Aircraft System (UAS) Service Demand 2015-2035: Literature Review and Projections of Future Usage," *DOT-VNTSC-DoD-13-01*, 2013.

[3] Federal Aviation Administration, "Airplane Flying Handbook," *FAA-H-8083-3B*, 2016.

[4] U.S. Department of Transportation, Federal Aviation Administration, "Part 107—Small Unmanned Aircraft Systems," *Code of Federal Regulations*, vol. Title 14, Chapter I, Subchapter F, 2017.

[5] U.S. Department of Transportation, Federal Aviation Administration, "Aeronautical Information Manual, Official Guide to Basic Flight Information and ATC Procedures," 2017.

[6] Federal Aviation Administration, "UAS Facility Maps." `https://www.faa.gov/uas/request_waiver/uas_facility_maps/`, April 2017.

[7] P. H. Kopardekar, "Unmanned Aircraft Systems Traffic Management Safely Enabling UAS Operations in Low-Altitude Airspace," *NASA Technical Reports Document ID 20170001573*, 2017.

[8] M. Guterres, S. Jones, G. Orrell, and R. Strain, "ADS-B Surveillance System Performance with Small UAS at Low Altitudes," in *AIAA SciTech Forum*, AIAA, 2017.

[9] K. Cunningham, D. E. Cox, J. V. Foster, S. E. Riddick, and S. A. Laughter, "Air STAR Beyond Visual Range UAS Description and Preliminary Test Results," *NASA Technical Reports Document ID 20160010790*, 2016.

[10] W. A. Handley, "Two NextGen Air Safety Tools: An ADS-B Equipped UAV and a Wake Turbulence Estimator," Master's thesis, University of Washington, 2016.

[11] C. Coopmans, N. V. Hoffer, A. M. Jensen, and D. J. Robinson, "AggieAir Unmanned Aerial System Traffic Integration Management: A Case Study with ADS-B Out," in *Digital Avionics Systems Conference (DASC)*, IEEE/AIAA, 2016.

[12] B. Stark, C. Coopmans, and Y. Chen, "Concept of Operations for Personal Remote Sensing Unmanned Aerial Systems," *Journal of Intelligent and Robotic Systems*, vol. 69, no. 1, pp. 5–20, 2013.

[13] J. M. Loffi, R. J. Wallace, J. D. Jacob, and J. C. Dunlap, "Seeing the Threat: Pilot Visual Detection of Small Unmanned Aircraft Systems in Visual Meteorological Conditions," *International Journal of Aviation, Aeronautics, and Aerospace*, vol. 3, no. 3, 2016.

[14] S. R. Hood, "Development of a flight data acquisition system for small unmanned aircraft," Master's thesis, Oklahoma State University, 2014.

[15] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft, Theory and Practice*, pp. 3–5. Princeton University Press, 2012.

[16] ArduPilot Development Team, "ArduPilot Development Site." http://ardupilot.org/ardupilot/, October 2016.

[17] ArduPilot Development Team, "GitHub - ArduPilot/ardupilot: ArduPlane, ArduCopter, ArduRover source." https://github.com/ArduPilot/ardupilot, July 2017.

[18] J. Hazelhurst, "Advanced Pixhawk Quadcopter Wiring Chart." http://ardupilot.org/copter/docs/advanced-pixhawk-quadcopter-wiring-chart.html, March 2016.

[19] USAF Test Pilot School, Edwards AFB, California, *Volume I, Performance Phase*, 1993.

[20] G. L. Gallagher, L. B. Higgins, L. A. Khinoo, and P. W. Pierce, *U.S. Naval Test Pilot School Flight Test Manual*, 1992.

[21] J. Holman, *Experimental Methods for Engineers*. McGraw-Hill, 2012.

[22] R. D. Kimberlin, *Flight Testing of Fixed-Wing Aircraft*. AIAA, 2003.

[23] M. E. Pestana, "Flying Unmanned Aircraft: A Pilots Perspective," in *Infotech@Aerospace 2011*, AIAA, 2011.

[24] L. Meier, A. Tridgell, and J. Goppert, "MAVLink Onboard Integration Tutorial." http://qgroundcontrol.org/dev/mavlink_onboard_integration_tutorial, July 2017.

[25] Mission Planner Development Team, "Mission Planner Development Site." http://ardupilot.org/planner/docs/mission-planner-overview.html, October 2016.

[26] Event 38 Unmanned Systems, "E386 Mapping Drone." https://event38.com/fixed-wing/e386-mapping-drone/, October 2016.

[27] Michael Oborne, "GitHub - ArduPilot/MissionPlanner: Mission Planner Ground Control Station (c# .net)." https://github.com/ArduPilot/MissionPlanner/, October 2016.

[28] Continuum Analytics, "Anaconda Python 2.7 Distribution." https://www.continuum.io/downloads, October 2016.

[29] Steve Gray, Chris Mallet, and AutoIt Team, "AutoHotkey H v1 download package." http://hotkeyit.github.io/v2/, November 2016.

[30] J. W. Langelaan, N. Alley, and J. Neidhoefer, "Wind Field Estimation for Small Unmanned Aerial Vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 4, pp. 1016–1030, 2011.

[31] J. D. Barton, "Fundamentals of Small Unmanned Aircraft Flight," *Johns Hopkins APL Technical Digest*, vol. 31, no. 2, pp. 132–149, 2012.

[32] L. R. Salazar, J. A. Cobano, and A. Ollero, "Small UAS-Based Wind Feature Identification System Part 1: Integration and Validation," *Sensors*, vol. 17, no. 1, 2017.

[33] ArduPilot Development Team, "MAVLink Mission Commands MAV CMD — Plane Documentation." http://ardupilot.org/plane/docs/common-mavlink-mission-command-messages-mav_cmd.html, November 2016.

[34] ArduPilot Development Team, "Automatic Landing – Plane Documentation." http://ardupilot.org/plane/docs/automatic-landing.html, July 2017.

[35] V. J. Rossow, "Lift-generated vortex wakes of subsonic transport aircraft," *Progress in Aerospace Sciences*, vol. 35, pp. 507–660, 1999.

[36] J. N. Hallock, G. C. Greene, J. A. Tittsworth, P. D. Strande, and F. Y. Wang, "Use of Simple Models to Determine Wake Vortex Categories for New Aircraft," in *AIAA Aviation*, AIAA, 2015.

[37] Z. C. Zheng and R. L. Ash, "Study of Aircraft Wake Vortex Behavior Near the Ground," *AIAA Journal*, vol. 34, no. 3, pp. 580–589, 1996.

[38] R. C. Nelson, *Flight Stability and Automatic Control*, pp. 81–95. McGraw Hill, 1998.

[39] UAV Factory, "UAV Factory Unmanned Platforms and Subsystems — Penguin B." `http://www.uavfactory.com/product/46`, April 2017.

[40] D. W. Yeo, N. Sydney, D. A. Paley, and D. Sofge, "Downwash Detection and Avoidance with Small Quadrotor Helicopters," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 3, pp. 692–701, 2017.

[41] U.S. Department of Transportation, Federal Aviation Administration, "National Plan of Integrated Airport Systems (NPIAS) 2017-2021," 2016.

[42] Grand Sky Development Company, "America's First UAS Business & Aviation Park — Grand Sky." `http://grandskynd.com/`, April 2017.

[43] U.S. Department of Transportation, Federal Aviation Administration, "General Aviation and Part 135 Activity Surveys - CY 2015," *2015 GA Survey Chapter 3 Tables 16SEP2016V2*, 2016.

[44] F. A. Administration, "Request to Operate in Controlled Airspace." `https://www.faa.gov/uas/request_waiver/request_operate_controlled_airspace/`, July 2017.

[45] AirMap, "Automated Airspace Authorization at U.S. Airports." `https://www.airmap.com/50-airports-airspace-authorization-laanc/`, July 2017.

[46] F. A. Administration, "Waiver Safety Explanation Guidelines for Part 107 Waiver Applications." https://www.faa.gov/uas/request_waiver/waiver_safety_explanation_guidelines/, July 2017.

VITA

Zachary P. Barbeau

Candidate for the Degree of

Master of Science

Thesis: SMALL UNMANNED AIRCRAFT SYSTEMS OPERATIONAL AND TRAFFIC MANAGEMENT CONSIDERATIONS

Major Field: Mechanical and Aerospace Engineering

Biographical:

Received B.S. degree from Oklahoma State University, Stillwater, OK, 2014, in Aerospace Engineering. Completed the requirements for the degree of Master of Science with a major in Mechanical and Aerospace Engineering at Oklahoma State University in July 2017.