NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ENGINEERING OF FAST AND ROBUST ADAPTIVE CONTROL FOR FIXED-WING UNMANNED AIRCRAFT

by

Ryan G. Beall

June 2017

Advisor:	Oleg A. Yakimenko
Co-Advisor:	Vladimir N. Dobrokhodov
Second Reader:	Fotis A. Papoulias

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2017	Master's Thesis

4. TITLE AND SUBTITLE

ENGINEERING OF FAST AND ROBUST ADAPTIVE CONTROL FOR FIXED-WING UNMANNED AIRCRAFT

5. FUNDING NUMBERS

6. AUTHOR(S)

Ryan G. Beall

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

As the demand for Unmanned Aerial System (UAS) technology increases, the current guidance, navigation, and control (GNC) algorithms will scale poorly to meet the demand because currently, significant resources are required to certify flight controllers on an individual platform basis. As different airframes are introduced to meet the expanding mission requirements, the resources required to sustain the GNC certification demand will become a limiting factor in scalability. The feasibility of replacing conventional GNC techniques with modern adaptive control theory was conducted on a commercial-off-the-shelf (COTS) open-source autopilot. This enabled rapid prototyping and integration of an adaptive controller. The adaptive controller architecture was designed to be aircraft non-specific. This ensures the controller easily integrates into any aircraft, therefore minimizing the resource burden of tuning and certification. The adaptive controller tested in this research improved performance over the baseline controller and was rapidly integrated on multiple various airframes with minimal resources. Improved performance over classical feedback was achieved with fast and robust adaptation in multiple regimes of flight.

14. SUBJECT TERMS

L1 Adaptive Control, Fixed-wing Modern Control, Fast and Robust Adaptive Control

15. NUMBER OF PAGES 121

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

ENGINEERING OF FAST AND ROBUST ADAPTIVE CONTROL FOR FIXED-WING UNMANNED AIRCRAFT

Ryan G. Beall
Lieutenant, United States Navy
B.S., United States Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2017

Approved by: Oleg A. Yakimenko
Advisor

Vladimir N. Dobrokhodov
Co-Advisor

Fotis A. Papoulias
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

As the demand for Unmanned Aerial System (UAS) technology increases, the current guidance, navigation, and control (GNC) algorithms will scale poorly to meet the demand because currently, significant resources are required to certify flight controllers on an individual platform basis. As different airframes are introduced to meet the expanding mission requirements, the resources required to sustain the GNC certification demand will become a limiting factor in scalability. The feasibility of replacing conventional GNC techniques with modern adaptive control theory was conducted on a commercial-off-the-shelf (COTS) open-source autopilot. This enabled rapid prototyping and integration of an adaptive controller. The adaptive controller architecture was designed to be aircraft non-specific. This ensures the controller easily integrates into any aircraft, therefore minimizing the resource burden of tuning and certification. The adaptive controller tested in this research improved performance over the baseline controller and was rapidly integrated on multiple various airframes with minimal resources. Improved performance over classical feedback was achieved with fast and robust adaptation in multiple regimes of flight.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

API	application program interface
APM	ArduPilotMega / Multi-Platform Autopilot
CG	center of gravity
COTS	commercial-off-the-shelf
DOD	Department of Defense
EKF	extended Kalman filter
FIR	finite impulse response
GCS	ground control station
GNC	guidance navigation and control
GPS	global positioning system
GUI	graphical user interface
IIR	infinite impulse response
IMU	inertial measurement unit
LTI	linear time invariant
mAh	milli amp hour
MAV	mirco aerial vehicle
MEMS	microelectromechanical systems
MIMO	multiple input multiple output
MRAC	model reference adaptive control

NED	north-east-down
ODE	ordinary differential equation
OS	operating system
PCHIP	piecewise cubic hermite interpolating polynomial
PID	proportional integral derivative
PWM	pulse width modulation
RC	remote controlled
SISO	single input single output
SITL	software in the loop
TF	transfer function
UAS	unmanned aerial system
VTOL	vertical take off and landing
ZOH	zero order hold

Executive Summary

The primary objective of this research was to determine if modern control techniques could provide engineering cost and schedule savings for DOD/NAVY autonomous systems. A waterfall systems engineering technique was utilized to evaluate the use of adaptive control on fixed-wing unmanned aircraft. The growing demand for unmanned systems will inherit the costs associated with guidance, navigation, and control. With the use of modern control techniques, these costs could potentially be reduced—if not eliminated—as well as gaining improved performance over the classical methods.

The field of adaptive control offers techniques for increasing performance and robustness in numerous settings and applications. Adaptive control is different from traditional feedback in that it provides a mechanism for adjusting the controller's parameters to reduce plant uncertainty. Classical feedback control utilizes parameters, which are specified by the engineer to optimize an ideal use case, which often requires extensive tuning and testing. Adaptive controllers adjust their control parameters using various intelligent mechanisms designed to increase robustness to plant variation or unanticipated disturbances. Adaptive control has many applications in the aerospace domain to include control strategies when aerodynamic coefficients are unknown or are non-constant, actuator failure, airframe damage, etc. This research evaluates fixed wing UAS controller performance and robustness using the $\mathcal{L}_1$ adaptive control architecture.

Successful integration of the $\mathcal{L}_1$ adaptive control algorithm was achieved utilizing discretization techniques on the Pixhawk 1 commercial autopilot. This algorithm was then prototyped and tested utilizing MavProxy and X-Plane10 as a complete software in the loop tool-chain. This enabled rapid prototyping and testing of the $\mathcal{L}_1$ algorithm in a high fidelity environment ensuring successful integration onto prototype aircraft.

Two different prototype aircraft were constructed to perform flight tests of the $\mathcal{L}_1$ adaptive controller with respect to the main objective of reducing the engineering demand required to achieve successful flight. Multiple flights were conducted across both prototype aircraft and achieved a drastic reduction the effort to achieve fully autonomous capabilities that also outperformed conventional PID controllers.

Overall, the $\mathcal{L}_1$ adaptive controller implemented in this research met the primary objective of reducing the engineering demand required of guidance navigation and control architectures. This key performance benefit is realizable through the use of modern control techniques, which also resulted in increased controller performance as well as improved battle damage tolerance as ancillary byproducts of the improved architecture. Utilizing modern adaptive control techniques has been shown as a reliable method for improved performance and robust control which could result in immediate cost savings to the DOD/NAVY.

Acknowledgments

First, I would like to thank my beautiful wife. Her continued support and unwavering love continue to enable me to achieve goals far beyond my own capability. My parents, Nolan and Charmian Beall, continue to keep me grounded and remind me to always remember what is important in life.

I would like to thank Oleg Yakimenko for his guidance and mentorship. His published acumen is unparalleled and set a high goal for me to achieve some day.

Additionally, I would like to thank Vladimir Dobrokhodov who is an expert in all things adaptive control. His numerous hours dedicated to discussing adaptive control were crucial in solidifying my understanding.

Finally, I would like to thank Andrew Tridgell and the ArduPilotMega Development team for their support and guidance on how to integrate into the APM infrastructure.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 DOD / NAVY Autonomous Roadmap

The Department of Defense (DOD) conducted an analysis of the role of autonomy, which outlined technology gaps and predicted advancements required to meet the growing performance demands [1]. The study amplifies the fact that autonomy is a challenging field and that it is arguably in its infancy. The roadmap attempts to guide decision makers in ensuring capitalization of under-utilized technology and succinct awareness of technical challenges limiting the current state of the art. Figure 1.1 outlines these elements at increasing scope of control comprised of various technology portfolios. The language referenced in the

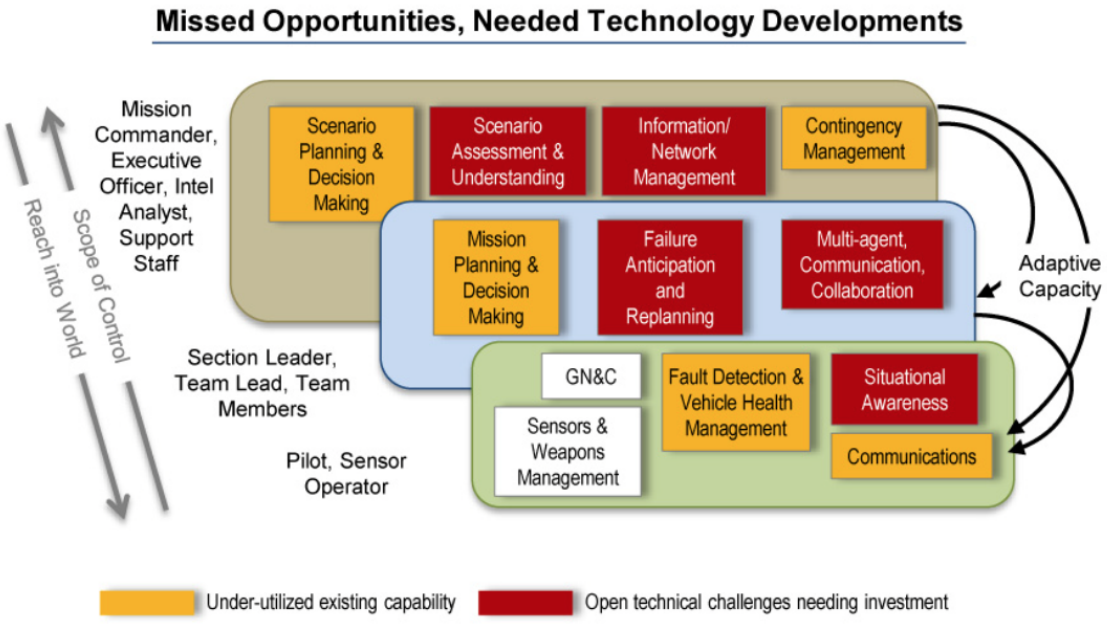

Figure 1.1. DOD Autonomy Roadmap. Source [1].

roadmap often recommends that machine learning and/or artificial intelligence is needed for elements such as "Fault Detection." On the lowest level, the roadmap annotates the use of guidance navigation and control (GNC) as neither needing improvement or current technology being underutilized. An in-depth look at the current state of the art with respect

to GNC reveals that GNC is still very costly and arguably antiquated. Research of adaptive control as applied to GNC offers new strategies offering improved performance and future reduced cost.

1.2 Challenges in Designing Versatile Controllers

The unmanned aerial system (UAS) has evolved tremendously over the past decade. Miniature autopilots have gotten smaller and cheaper with more sensitive and redundant sensor packages largely due to the cellular phone industry accelerating microelectromechanical systems (MEMS) technology. The ability to manufacture these autonomous systems at fractions of the cost enables the advancement in multiple cooperative UAS applications including swarming capability. This ability to mass-produce large quantities of UAS's poses an interesting challenge. Even though the price has gone down and the performance has gone up, there still exists a significant amount of man-hours dedicated to sensor calibration and autopilot control law configuration and tuning for optimal performance.

Over the past decade UAS avionics have drastically improved, but the fundamental control law algorithms have not changed. The proportional integral derivative (PID) architecture found its origin in automatic ship steering applications in 1922 [2]. Conventional control law architectures for UASs predominately still use PID controllers. Their architecture offers a well understood and predictable behavior for the class of linear systems. For this reason, it is well suited for the aviation application. The detriment of PID control is that its application is mostly constrained by its use on a linear plant and most aerospace applications are non-linear and time varying. An aircraft's control authority that increases proportionally to dynamic pressure is one example of significant changes in aerodynamic non-linear control behavior. In this case, the PID controller's robustness to changes in velocity and/or density altitude is not guaranteed and for most aircraft has to be delicately handled with lookup tables produced from hours of flight test for given configurations.

Conventional controllers (PID) are difficult to tune and achieving an adequately tuned controller requires a significant amount of time and resources. These difficulties can arise because of many uncertainties with respect to the aircraft as outlined in the following three subsections.

1.2.1 Unknown Constant Airframe Parameters

In the case where airframes are assembled by the same manufacturer, all aircraft still require a tedious quality assurance check. Physical aspects of the airframes such as center of gravity (CG), control surface deflection/calibration/speed, and airframe alignment all can drastically vary within the same delivered batch of airframes. Additionally, these miniature UASs experience hard landings, crashes, and/or damage in transportation, which all can affect aerodynamic handling qualities.

1.2.2 Unknown Non-Constant Airframe Parameters

Aircraft with large airspeed envelopes or various configurations (vertical take off and landing (VTOL), flaps, retractable landing gear), exhibit changing dynamics that challenge conventional PID robustness. Because these dynamics are changing drastically, the PID controllers either underperform or have to be over designed with ad-hoc gain scheduling techniques which significantly increases the controller's already complex tuning process.

1.2.3 Fault Tolerance

If an aircraft exhibits airframe faults or battle damage, the desirable outcome results in the controller achieving robust performance. Classical PID controllers have limited to no ability to cope with these types of failures. Conventional methods require a priori knowledge of the failure scenario. Reconfiguration schemes are developed for specific failures, further complicating designs.

1.3 Problem Formulation and Thesis Organization

This thesis addresses a problem of shortening the time for developing and fielding a new autonomous aerial vehicle by utilizing an adaptive controller, which relaxes some of the traditional constraints and assures better robustness in the case of airframe configuration variation, various uncertainties, and faults.

To address the formulated problem, this thesis is organized as follows.

Chapter 2 provides an overview of modern adaptive control. The brief history of adaptive control is introduced to further amplify the specific use case for aerospace applications.

The adaptive control architecture is compared with conventional feedback controllers (PID) to clarify the distinction between the two approaches for stable and robust control. The model reference adaptive control (MRAC) architecture is defined in order to articulate the difference between the MRAC architecture and the specific modifications utilized in this research to formulate the $\mathcal{L}_1$ adaptive controller.

Chapter 3 outlines the proposed adaptive controller architecture used instead of conventional less-robust controllers. It addresses why the architecture must take the form of indirect adaptive control vice direct adaptive control. The definition of estimated parameters are defined with respect to the aerodynamics model derived in Appendix B. The filter section of the controller used to bandwidth limit the controller is defined with respect to the specific three parameter model chosen for this research. The $\mathcal{L}_1$ adaptive controller discretization process is reviewed with respect to methods implemented to achieve integration on a commercial-off-the-shelf (COTS) autopilot, which is described in Chapter 4.

Chapter 4 discusses the COTS autopilot and flight stack code chosen for this research. It also covers the ground control station (GCS) and software in the loop (SITL) infrastructure which was heavily utilized for initial code development, testing, and validation. Chapter 4 follows with the major steps undertake to prototype two airframes to be used in the flight testing of the proposed controller.

Chapter 5 and 6 describe the results of computer simulations and flight tests, respectively. Chapter 6 also outlines the shortcomings of the COTS autopilot architecture and improvements specific to the $\mathcal{L}_1$ adaptive controller.

The thesis concludes with Chapter 7 providing conclusions.

CHAPTER 2:
Overview of Modern Control Techniques

This chapter is a general review of the history of adaptive control to include its use cases and previous pitfalls. A brief overview of algorithm differences between conventional feedback control and MRAC architectures is also covered.

2.1 Adaptive Control History

Adaptive control saw its early debut in the NASA North American X-15 hypersonic rocket-powered X-plane experimental aircraft. The X-15's performance envelope exceeded Mach 6.0 and 300,000 feet [3]. Engineers realized early on that the linear controllers performed well only at one dynamic pressure, but nowhere near the entire flight envelope. Scheduling the controller gains with respect to dynamic pressure (gain scheduling) was one method used to help ensure robustness; the method is still widespread in commercial aviation due to its robustness but requires a significant effort to explore the entire flight envelope. These non-trivial efforts were what encouraged the exploration of the benefits of adaptive control.

The X-15 program started in 1959 and continued to 1968 flying nearly 200 successful flights [4]. It was considered one of NASA's most successful programs. The benefit of adaptive control to the X-15 was that the adaptive controller was supposed to adjust the gain parameters online automatically. If the controller was self-tuning, it could potentially offer increased performance while reducing complexity. Honeywell implemented the MH-96 adaptive controller in the X-15-3 as a fly-by-wire controller designed to adaptively adjust the damping in pitch and roll with respect to the desired model response. The goal was to achieve consistent aircraft response regardless of dynamic pressure and other variables. Dydek et. al comments that during test flights of the MH-96 adaptive control, increased performance was observed especially in the dynamic phases of re-entry over that of the linear fixed gain damping system [5]. These early breakthroughs in adaptive control proved the benefits could be viable aerospace solutions. However, on November 15, 1967, there was a fatal accident caused by the adaptive controller. NASA documented that the adaptive controller created an out of control flight situation resulting in dynamic pressures exceeding

the structural limits and subsequent breakup of the airframe at 65,000 feet [4].

The turbulent start of adaptive control, as implemented on the X-15 program, was due in large part to the early naive understanding of robustness. Contemporary robust adaptive control strives to encapsulate these deficiencies of robustness in studies and proofs using Lyapunov stability analysis. In addition to the developments of rigorous stability analysis tools, a number of unique techniques have also been implemented to increase controller robustness. One such technique utilizes dead band limits on the model adaptation process to avoid system/measurement noise from causing the un-learning of the states and is called "dead-zone modification" as proposed by B.B. Peterson and K.S. Narendra [6]. Lavretsky and Wise also reference the "σ and e modifications" which adds damping to the adaptation process [7]. The $\mathcal{L}_1$ adaptive control algorithm utilizes a technique which seeks to decouple the adaptation rate from robustness by "low-pass filtering" the contribution of the fast estimator under the premise that estimating the entire frequency spectrum is overly ambitious and should be limited to the bandwidth of the actuator-plant combination. Many advances have been made in the adaptive control field over the past few decades, and this research sets out to evaluate a small subset of these techniques in the unforgiving aerospace environment.

2.2 Classical Feedback vs Adaptive Control

Control of a system can be categorized into two required elements; the requirement to stabilize the system in the presence of:

1. disturbances that affect the controlled states and outputs (pitch rate perturbation caused by environmental effects)
2. disturbances that affect the dynamics of the open loop plant (pitch rate effectiveness with respect to dynamic pressure)

Classical feedback control seeks to resolve disturbances that affect the tracking with respect to state perturbation and assume that the plant dynamics are constant. This form of control is meticulously tuned to achieve the desired overshoot and settling time for example. The important assumption that is made by classical feedback controllers is that the underlying plant/system dynamics are not changing. For example, the cruise control that maintains a vehicle's speed assumes that the available horsepower of the car is fixed. This is a

fairly good assumption as the horsepower with respect to rpm available at sea level and 5,000 feet for an internal combustion engine is constant enough that a fixed gain feedback controller would perform well at maintaining the speed of the vehicle in both environments. In the case of an airplane, the dynamic pressure is proportional to velocity squared and can drastically change the performance of the aircraft. In this case, the constant system performance assumption can cause a fixed gain classical feedback controller to go unstable at higher dynamic pressures (higher airspeeds). Conversely, adaptive controllers assume that the system performance is unknown and is likely to vary with time. Adaptive control seeks to ensure a system's performance with respect to characteristics, such as damping ratio and settling time, which are kept constant regardless of a plant's dynamics that may be unknown and time varying. For both classical and adaptive control, there exists some form of error which drives the controller. In the case of classical feedback, the error is calculated between the command and the feedback state of the plant. In adaptive control (in general), the error is calculated between the outputs of the desired reference model and real plant's measured performance.

Figure 2.1 outlines the decision making process a controls engineer makes when deciding the type of controller needed for a given circumstance.

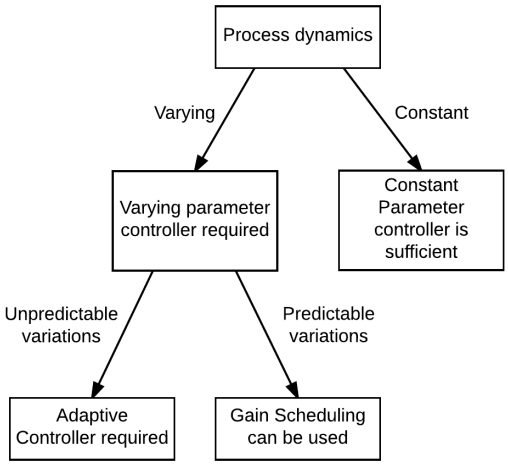

Figure 2.1. Determine If Adaptive Control Should Be Used. Adapted from [8].

2.3 Model Reference Adaptive Control

MRAC establishes the foundation for most of modern, robust adaptive control. Its structure is intuitive in nature and seeks to define a system's response to a command signal with a reference model. Unlike traditional feedback where the error signal is generated with respect to state or output error, MRAC's objective is to minimize the error between the performance characteristics of a chosen reference model and the plant. The error between the model response and the system response generates error for an "adjustment mechanism" to learn the unknown model parameters.

Figure 2.2 illustrates a topology where a traditional feedback controller is established as an inner loop and the "Reference Model" and "Adjustment Mechanism" is established as an outer loop. The outer loop attempts to minimize the error between the reference model

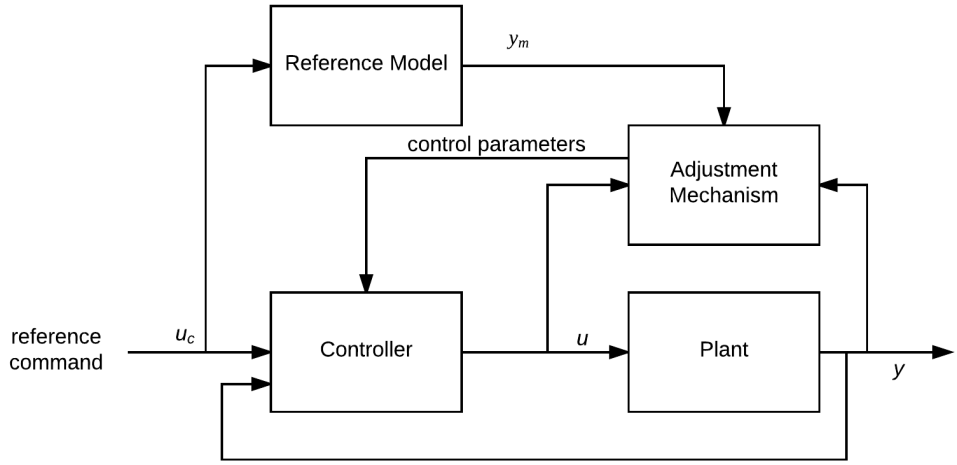

Figure 2.2. Traditional Model Reference Adaptive Control (MRAC) Architecture. Adapted from [8].

output and the plant output. Two primary methods for using this error to learn the system parameters are: gradient descent and Lyapunov stability theory.

2.3.1 MIT Rule - Gradient Descent

One of the first approaches to the design of MRAC controllers was implemented at the Instrument Labs at MIT (now known as Draper Labs). The gradient descent based method

was called the "MIT Rule" for this reason [8]. This method attempts to learn some unknown parameter by descending the gradient of the error between the reference model and the plant output.

Given the simple first-order system $G(s)$:

$$G(s) = k_{dc}\frac{1}{s+1} \tag{2.1}$$

where k_{dc} is some unknown feedforward gain. In the case of the MIT rule, k_{dc} is the parameter to be learned and is defined as θ. The first step in the MIT rule is to establish a cost (or loss) function. One example of a cost function $J(\theta)$ is:

$$J(\theta) = \frac{1}{2}e^2 \tag{2.2}$$

$$e = y - y_m \tag{2.3}$$

where e is error, y is the plant output, and y_m is the model output.

In order for the cost function to be minimized, the negative gradient of the cost function is calculated and used to correct the a priori estimate. This method takes the following form where γ is the adaptation gain:

$$\frac{d\theta}{dt} = -\gamma\frac{\partial J}{\partial \theta} = -\gamma e\frac{\partial e}{\partial \theta} \tag{2.4}$$

The stability of this method is very system dependent and heavily relies on trial and error to ensure the adaptation gain (γ) is not too high. This usually requires low adaptation rates for most systems and may not produce adequate results. It should also be noted that this method relies on adequate persistence of excitation [8]. In simple terms, the persistence of excitation condition guarantees some functional correspondence between the error and the unknown parameters θ. If the error signal is sufficiently rich, then the partial derivative is explicitly evaluated by the adaptation process running recursively. Additionally, gradient descents are subject to local extreme convergence for non-convex manifolds. Therefore, this method does not explicitly guarantee parameter convergence even though correct steady-state gain

is achieved for the closed loop system.

With the differences between classical feedback control and adaptive control being discussed, the next chapter applies analytical tools to develop a specific adaptive control architecture to address the requirements outlined in Chapter 1.

CHAPTER 3:
Engineering of Adaptive Control

This chapter outlines the pertinent aspects of the $\mathcal{L}_1$ adaptive control architecture used for this research. The distinction between direct and indirect adaptive control is made in order to illustrate the importance of why the indirect architecture is critical for the $\mathcal{L}_1$ algorithm. This chapter also defines the parameter estimates provided by the $\mathcal{L}_1$ algorithm with respect to the aerodynamic model simplifications outlined in Appendix B. Finally, the $\mathcal{L}_1$ adaptive control filter is explained and the methods used for discretizing the algorithm for autopilot integration are discussed. It should be noted that the versions of the $\mathcal{L}_1$ adaptive controller have been patented and proper licensing should be considered if commercial use is desired.

3.1 $\mathcal{L}_1$ Adaptive Control

The $\mathcal{L}_1$ adaptive controller is an evolution of the concepts implemented by MRAC. They are similar approaches designed to control a system with unknown parameters. The estimates of unknown parameters are adjusted to achieve the desired outcome of the error between the actual plant (system) and the referenced system model (state predictor) to asymptotically approach zero. Adaptive control attempts to estimate the plant's unknown parameters in situ. Parameter estimation is done using either direct or indirect architecture. The indirect architecture attempts to estimate the system's parameters and can be compared to online system identification. Alternately, the easier to implement direct architecture estimates the controller parameters explicitly. These architectures can be seen in Figures 3.1 and 3.2.

3.1.1 Reference Model versus Companion Model

Traditional MRAC controllers often refer to the system objective function as the "Reference Model." In this case, the engineer designs a reference model response, and it is from this model response that the error state is calculated directly. Because the $\mathcal{L}_1$ adaptive control implements the use of a filter in conjunction with a model objective function, the model is often referred to as the "Companion Model." This subtle distinction is necessary because the engineer must be aware that the system response will be with respect to the filter plus

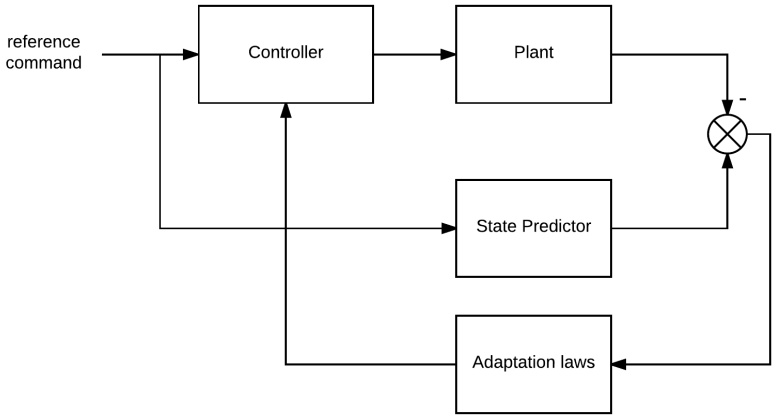

Figure 3.1. Direct MRAC Architecture

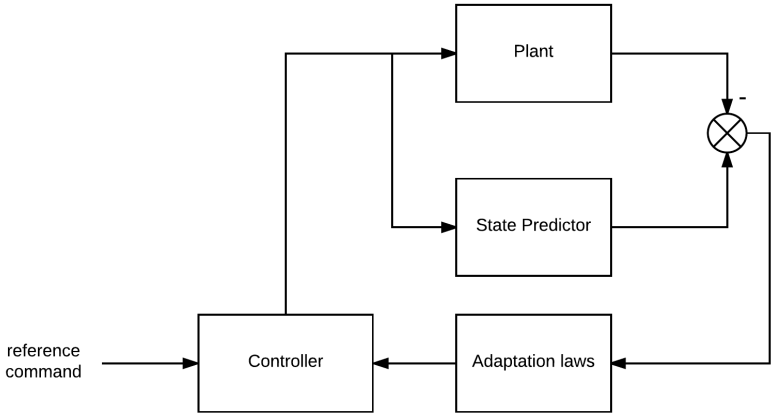

Figure 3.2. Indirect MRAC Architecture

the companion model in series. In other words, the plant will not mimic the companion model; it will mimic the companion model plus the filter section.

3.1.2 $\mathcal{L}_1$ Architecture

The $\mathcal{L}_1$ adaptive control algorithm asserts that trying to contorl the plant uncertainties outside of the control actuators' bandwidth is overly ambitious. The system's actuator bandwidth and the slow dynamics of the plant are most commonly the system's limiting factors, and the estimator's robustness/stability could be in question if unmodeled high-

frequency content exists in the plant. The $\mathcal{L}_1$ adaptive control constrains the objective function by using a low-pass filter (first or second-order) to limit the frequency response to meet robustness specifications. This low-pass filter should be tuned to a frequency response commensurate with the actuator's frequency response. Through inspection the low-pass filter placement in the controller topology, it becomes clear that the indirect architecture is the only candidate. Figures 3.3 and 3.5 illustrate the placement of the low-pass filter and its implication on the closed loop model.

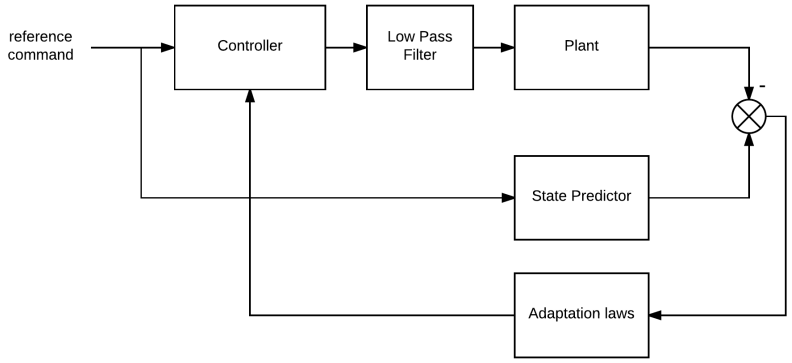

Figure 3.3. Direct MRAC Architecture with Low-Pass Filter

It can be seen in Figure 3.3 that the direct architecture implementation of the low-pass filter introduces difficulties. The companion model (state predictor) is defined using aerodynamic model and the stability is verified for a Lyapunov candidate function. However, these underpinnings designed in the companion model become nonsensical in the implementation found in Figure 3.4 because the filter is applied to the plant and not modeled in the companion model. This specific filter placement is non-subtractable when the error state is calculated because it is not applied in both signal paths (plant and state predictor).

Conversely, the indirect approach offers an implementation which ensures the low-pass filter is applied to both the companion model (state predictor) and the plant ensuring that the interaction of the filter is subtractable when calculating the error state.

It can be seen that the low-pass filter in the direct architecture inherently changes the structure of the plant with the cascading of the low-pass filter and plant block diagrams. This change mathematically is not mirrored in the companion model (state predictor) and therefore is not subtractable. This means that the state predictor is modeling incomplete

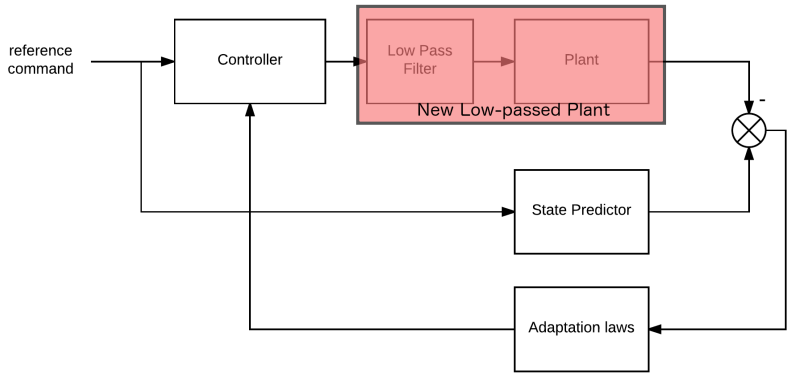

Figure 3.4. Non-Subtractable Low-Pass Implementation - Direct Architecture

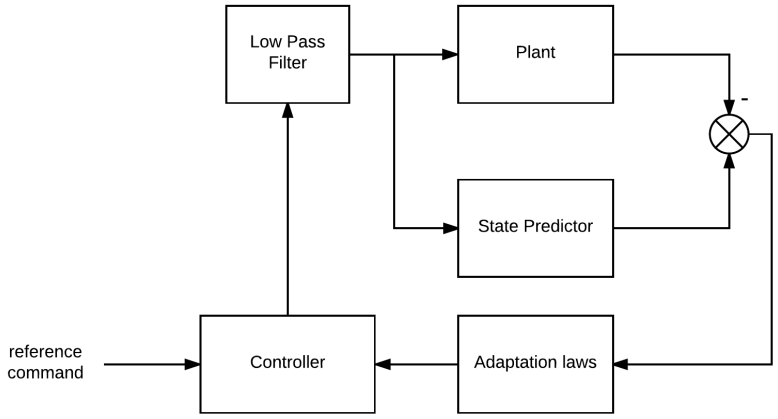

Figure 3.5. Indirect MRAC Architecture with Low-Pass Filter

information and therefore will incorrectly characterize the system. However, in the indirect case, the structure of the model is kept intact, and the low-pass filter is applied to both the plant and the state predictor. This ensures that the low-pass filter is subtractable when calculating the error state and the model's structure is kept intact.

Many variations of the $\mathcal{L}_1$ adaptive architectures have been derived for various use cases [9]. Some of the following forms were studied for viability in the fixed wing UAS use case with respect to state parameters (θ, ω, σ) as described in Figure 3.6 and Section 3.2:

- single input single output (SISO) with constant but unknown state parameters

- SISO with time variant and/or nonlinear unknown bounded state parameters
- multiple input multiple output (MIMO) with constant but unknown state parameters
- MIMO with time variant and/or nonlinear unknown bounded state parameters

MIMO control algorithms would potentially afford the controller more ability to cope with system coupling if present. Fixed wing UAS equations of motion, as seen in Equation B.11, exhibit coupled behavior both in the aerodynamic and inertial coupling. However, MIMO was not chosen due to the added complexity required to architect the algorithm into source code. Matched uncertainty are terms that can be factored from the control matrix (B) and can be time invariant. Unmatched uncertainties are typically modeled as process noise impacting the state ($\dot{x}$) directly regardless of control. The unmatched uncertainty architecture offers a more appealing solution for fixed wing use cases (asymmetric actuator failure, aerodynamic coefficients scaled by dynamic pressure), but adds a significant amount of complexity to the architecture. In summary, the SISO architecture with matched uncertainty was chosen for this research.

The SISO controller with matched uncertainty was selected to control pitch rate (q) and roll rate (p) of the aircraft using two separate controllers. The simplified equations in Equation B.13 assume that there is no aerodynamic or inertial coupling in order to simplify the model. This simplification was chosen to make the controller as airframe agnostic as possible while enabling very simple applications in code. Utilizing this simplified architecture may be at the detriment of controller performance, but as seen by the flight test results in Section 5.2, the baseline performance is adequate to achieve performance which is better than PID in some cases. In this implementation of the $\mathcal{L}_1$ adaptive controller, the desired state x to be controlled was an individual body rate (e.g., q, p).

As seen in Figure 3.6, the generalized $\mathcal{L}_1$ architecture in block diagram form and the following elements can be identified

k_g - feed forward input gain
k - feedback gain
$D(s)$ - user described filter (second-order low-pass plus integrator)
$\hat{\eta}$ - $\mathcal{L}_1$ controller state
$\dot{x}$ - first-order differential equation of state model
$\hat{x}$ - state estimate

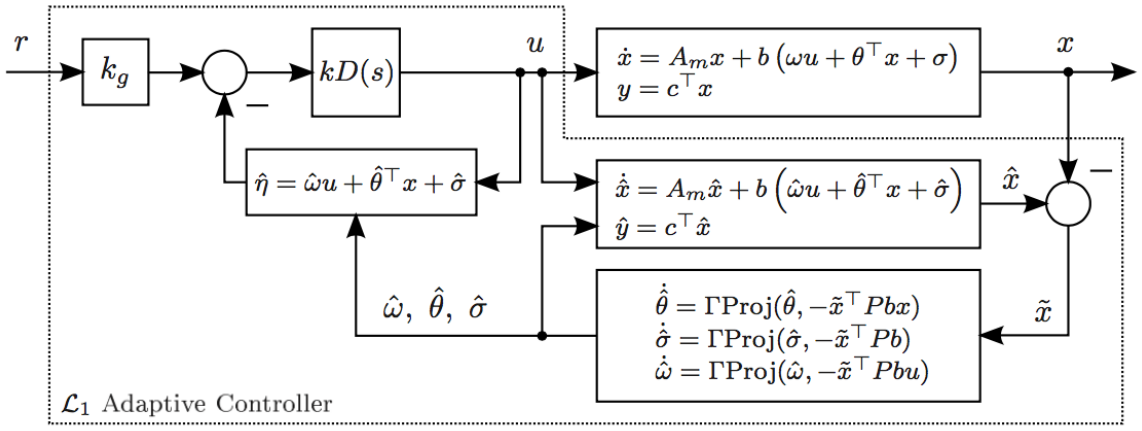

Figure 3.6. $\mathcal{L}_1$ Architecture with Matched Uncertainty Block Diagram. Source [9].

$\tilde{x}$ - state error

u - controller output

r - reference input

A_m - Hurwitz matrix

b - input matrix

$\hat{\omega}$ - unknown input gain coefficient

$\hat{\theta}$ - unknown constant state coefficient

$\hat{\sigma}$ - unknown disturbance estimate

Γ - adaptation gain

Pb - solution to the Lyapunov stability equation

It should also be noted that the architecture presented in Figure 3.6 includes the use of a projection operator. The estimation dynamics of $\dot{\hat{\omega}}$, $\dot{\hat{\theta}}$, and $\dot{\hat{\sigma}}$ are projection based adaptation laws. This ensures that the adaptation stays bounded around the feasible region of parameter space. The Lyapunov stability proofs for this architecture rely on this method to guarantee stability [9]. More discussion on the specific implementation of this operator can be found in Appendix E.

One of the main benefits of using the SISO architecture is in its simplicity so that the solution to the Lyapunov stability equation (Pb) utilized in the projection based adaptation laws is greatly simplified.

In this case, Pb reduces to:

$$Pb = \frac{1}{2\omega_n} \tag{3.1}$$

where ω_n is the natural frequency in rad/s for the first-order companion model in discrete recursive form assuming DC gain of 1. The proof for this can be found in Appendix F.4.

3.2 $\mathcal{L}_1$ Parameter Estimation

Three model parameters were evaluated in this research. The first model parameter θ establishes the baseline architecture and Lyapunov stability proof. Each additional model parameter added to the architecture adds complexity to the architecture and its required stability proofs. The general progressive build up of the three parameter architecture is outlined in the nomenclature for the estimation of a system with unknown constant parameters, input/output disturbances, and an unknown system input gain.

The $\mathcal{L}_1$ adaptive control algorithm is primarily used to estimate unknown constant system parameters. These system parameters are defined as θ as seen in the following model:

$$\dot{x}(t) = Ax(t) + b(u(t) + \theta^\top(t)x(t)) \tag{3.2}$$

The second adaptive element is the estimation of the input/output disturbances (σ). This additional parameter is implemented in this research as any unmodeled transient dynamics such as aerodynamic/inertial coupling. This model takes the form

$$\dot{x}(t) = Ax(t) + b(u(t) + \theta^\top(t)x(t) + \sigma(t)) \tag{3.3}$$

The last adaptive element used for this research was the estimation of unknown system input gain (ω). Estimating the unknown system input gain offers the controller the ability to estimate the actuator effectiveness for changes in dynamic pressure or failed control surfaces. This model takes the form

$$\dot{x}(t) = Ax(t) + b(\omega(t)u(t) + \theta^\top(t)x(t) + \sigma(t)) \tag{3.4}$$

The model in Equation 3.4 can be paralleled to the aircraft model derived in Equation B.14. As a result, the roll rate model takes the similar form

$$\dot{\hat{p}} = A_p\hat{p} + b_p\left(\hat{\omega}_p\delta_a + \hat{\theta}_p p + \hat{\sigma}_p\right) \tag{3.5}$$

This final model, with the inclusion of all three estimated parameters ($\hat{\omega}, \hat{\theta}, \hat{\sigma}$), established the architecture tested in this research. The fundamental assumption that the aerodynamic and inertial coupling was negligible and set to zero as outlined in Equation B.13 is agreeably a gross assumption which may prove to be inadequate for stable flight. These assumptions presume that the estimated input/output disturbance ($\hat{\sigma}$) would be adequate to compensate for these unmodeled dynamics and were validated by successful flight test in Section 5.2.

These derivations of the $\mathcal{L}_1$ algorithm guarantees that the error between the model and plant asymptotically approaches zero, but this does not imply the constraint that the estimated parameters are in fact converging to their real values. The algorithm only guarantees that the parameters are bounded and therefore it is common to observe the parameters never reaching steady-state. The engineer must ensure that the bounds set on the parameters are sufficient for the controlled system to not become unstable. In the discrete form, the algorithm can have numerical floating point instability if these projection operator bounds are too high and not thoroughly tested in simulation.

3.3 $\mathcal{L}_1$ Filter - $C(s)$

One of the key features of the $\mathcal{L}_1$ adaptive controller is that the robustness of the controller is decoupled from the adaptation rate. This is handled in the filter section of the $\mathcal{L}_1$ architecture which is annotated as $C(s)$. To ensure guaranteed stability of the $\mathcal{L}_1$ algorithm, $C(s)$ must be verified as strictly-proper stable. With the architecture that includes the system input gain (ω), one cannot simply apply a stand alone filter. The inclusion of ω in the architecture block diagram in Figure 3.6 slightly modifies the signal output and takes the following form

$$u(s) = -kD(s)(\hat{\eta}(s) - k_g r(s)) \tag{3.6}$$

where $D(s)$ is the new user defined filter and $C(s)$ now takes the form

$$\begin{aligned} \omega \in \Omega_0 &\triangleq [\omega_{l_0}, \omega_{u_0}] \\ C(s) &= \frac{\omega k D(s)}{1 + \omega k D(s)} \end{aligned} \tag{3.7}$$

In the case which the user defined function $D(s)$ is a simple integrator, $C(s)$ takes the form

$$\begin{aligned} D(s) &= \frac{1}{s} \\ C(s) &= \frac{\omega k}{s + \omega k} \end{aligned} \tag{3.8}$$

The feedback gain k should not be assumed to be 1 in this case. As seen in Equation 3.8, the $C(s)$ transfer function is a function of k that can have significant influence on the controller output. In actual implementation, k was found to be one of the most influential gains in the architecture and extremely critical in specifying the transition between robustness and performance. The feedback gain k must be set to low enough to prevent high frequencies to the actuators, and high enough to ensure the $\mathcal{L}_1$ norm stability conditions [9].

3.4 $\mathcal{L}_1$ Discrete Time Implementation

Implementing any algorithm on actual autopilot hardware will inevitably force some if not all parts of the algorithm to be discretized. Autopilots like the Pixhawk operate at some scheduled loop rate for executing the litany of subprograms that measure sensors, calculate navigation commands, and much more. In the case of the Pixhawk autopilot, the main loop can run up to 400 Hz. At a 400 Hz sampling rate, there is a significant insurance that the vehicle's dynamics bandwidth will be completely defined. However, the ArduPilotMega / Multi-Platform Autopilot (APM) flight stack records all logged parameters also at this loop rate and can create log files larger than are reasonably desired. There are a myriad of other reasons why the engineer would not want to run at high loop rates, but successful flight at the lowest (default of 50 Hz) is desired if adequate performance of the adaptive control can be achieved. Failures in early adaptive control were largely in part due to a very naive understanding of robustness. Brian Anderson concludes that "it is clear that the identification time scale needs to be faster than the plant variation time scale, else

identification cannot keep up" [10].

The $\mathcal{L}_1$ architecture does not require the algorithm to run in sync with the sensor measurements. Ideally, the autopilot main loop rate would remain at its default value (50 Hz), and the adaptation section of the $\mathcal{L}_1$ controller would run as fast as the CPU would allow. However, the APM architecture does not lend itself well to this scheme in its current configuration without significant modification to the code base. Therefore, the initial tests presumed that the $\mathcal{L}_1$ algorithm is running at the same frequency as the main loop of the autopilot. The fundamental derivation of the $\mathcal{L}_1$ algorithm proves that performance and adaptation gain are maximized as the loop time (dt) approaches zero. Increased performance would come from higher adaptation loop rates, but the primary focus was to implement the $\mathcal{L}_1$ algorithm with minimal or no detrimental effects to the current APM codebase.

3.4.1 Digital Bi-Quad Filter

The $\mathcal{L}_1$ adaptive control algorithm utilizes two specific elements that will require careful discretization; the companion model and the low-pass filter. The digital bi-quad filter offers a very versatile and straightforward method for accurately implementing the companion model and the low-pass filter discretely using its recursive nature. It is a second-order filter which uses a finite impulse response (FIR) front end and an infinite impulse response (IIR) back end requiring four total memory blocks. This topology allows the designer to create numerous types of filters (low-pass, high-pass, band-pass) simply by choosing appropriate coefficients. If a first-order filter is needed, then the higher order FIR/IIR terms can be set to zero. Figure 3.7 illustrates this filter's topology where the FIR structure is the left two memory blocks and the IIR structure is the right two memory blocks.

A bilinear Z transform is used to convert the desired S-domain (continuous time domain) filter/model into the Z-domain (discrete time domain) to determine the structure of the coefficients.

This derivation for the second-order low-pass model is

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1} \tag{3.9}$$

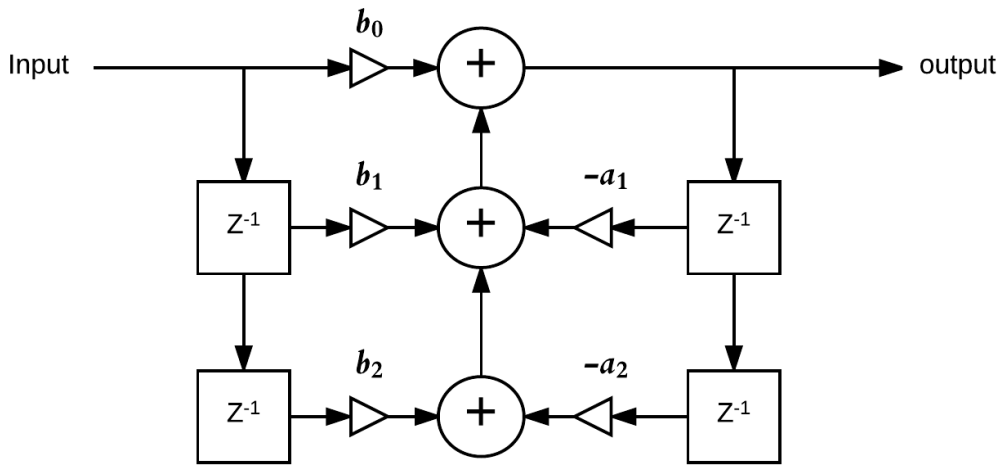

Figure 3.7. Digital Bi-quad Filter Architecture

where the bi-linear transform converts s to z via

$$s = \left(\frac{1}{K}\right)\left(\frac{z-1}{z+1}\right) \tag{3.10}$$

K is the "pre-warping" factor which accounts for the transition of the vertical s-plane into the circular z-plane as seen in Figure 3.8.

where ωT is

$$\omega T = 2\pi\left(\frac{F_c}{F_s}\right) \tag{3.11}$$

$$\begin{aligned} K &= tan\left(\frac{\omega T}{2}\right) \\ &= tan\left(\pi\frac{F_c}{F_s}\right) \end{aligned} \tag{3.12}$$

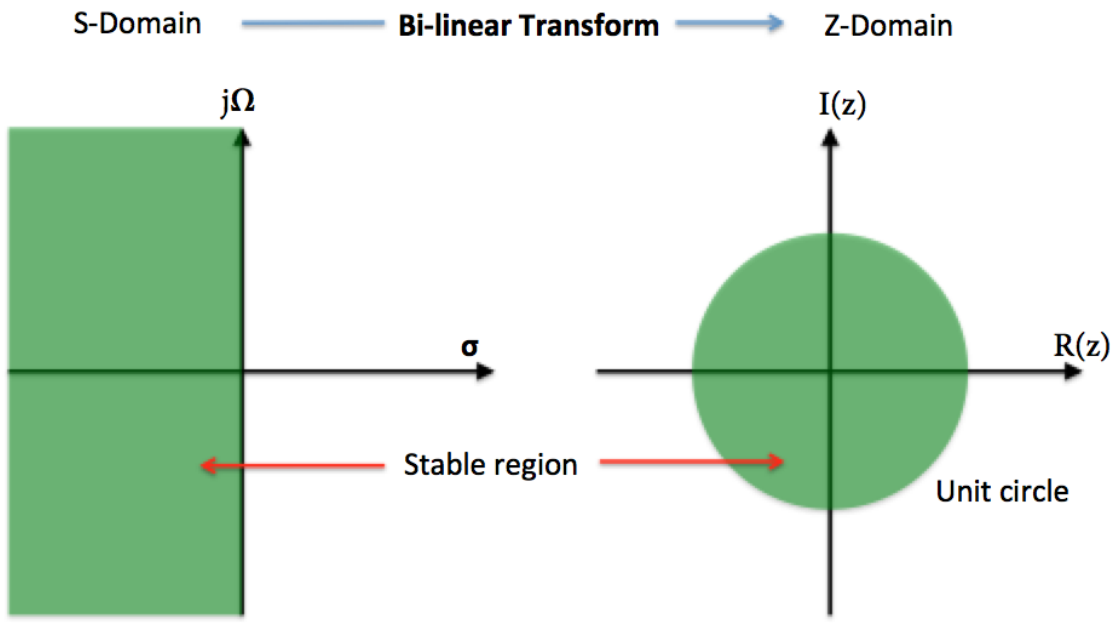

Figure 3.8. Bi-linear Transform

F_c is the desired corner frequency of the filter and F_s is the sampling rate (or loop rate of the autopilot). This "pre-warping" is critical to ensure that the continuous time cutoff frequency desired is correctly established in the discrete implementation. It is the engineer's discretion if pre-warping is required for the appropriate application, but the general guidance is to pre-warp the Z-domain coefficients if the desired cut-off frequency is close to Nyquist–Shannon sampling theorem frequency ($\frac{F_s}{2}$). It was chosen for this application to always pre-warp the coefficients even though the error is small for corner frequencies which are fairly distant from Nyquist–Shannon sampling theorem frequency. Continuous calculation of the pre-warp coefficient was chosen because calculating the $tan()$ function real time on the CPU adds negligible computational strain but offers ease of tuning for the engineer.

Applying the bi-linear transform to the continuous time second-order low-pass filter results in

$$H(z) = \frac{1}{\left[\left(\frac{1}{K}\right)\left(\frac{z-1}{z+1}\right)\right]^2 + \frac{\left(\frac{1}{K}\right)\left(\frac{z-1}{z+1}\right)}{Q} + 1} \tag{3.13}$$

The desired form is

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{3.14}$$

Reducing Equation 3.13 to match the form in Equation 3.14 results in the following coefficients

$$\begin{aligned} a_0 &= 1 \\ a_1 &= \frac{2(K^2 - 1)}{K^2 + \frac{K}{Q} + 1} \\ a_2 &= \frac{K^2 - \frac{K}{Q} + 1}{K^2 + \frac{K}{Q} + 1} \\ b_0 &= \frac{K^2}{K^2 + \frac{K}{Q} + 1} \\ b_1 &= 2b_0 \\ b_2 &= b_0 \end{aligned} \tag{3.15}$$

The bandwidth of the filter Q can be set by the engineer. For example, if the pass-band of the filter is desired to be flat (Butterworth) then Q can be set equal to $\frac{1}{\sqrt{2}}$. For this research, the following C++ code segments were used to explicitly calculate the bi-quad low-pass filter implementation [11]:

```
void DigitalBiquadFilter<T>::compute_params(float sample_freq,
float cutoff_freq, biquad_params &ret) {
    ret.cutoff_freq = cutoff_freq;
    ret.sample_freq = sample_freq;

    float fr = sample_freq/cutoff_freq;
    float K = tanf(M_PI/fr);  //Pre-Warp calculation
    float c = 1.0f+2.0f*cosf(M_PI/4.0f)*K + K*K;
```

```
    ret.b0 = K*K/c;
    ret.b1 = 2.0f*ret.b0;
    ret.b2 = ret.b0;
    ret.a1 = 2.0f*(K*K-1.0f)/c;
    ret.a2 = (1.0f-2.0f*cosf(M_PI/4.0f)*K+K*K)/c;
}
```

```
T DigitalBiquadFilter<T>::apply(const T &sample,
const struct biquad_params &params) {

    T delay_element_0 = sample - _delay_element_1 * params.a1
     - _delay_element_2 * params.a2;

    T output = delay_element_0 * params.b0
     + _delay_element_1 * params.b1
     + _delay_element_2 * params.b2;

    _delay_element_2 = _delay_element_1;
    _delay_element_1 = delay_element_0;

    return output;
}
```

This implementation can be employed as the $\mathcal{L}_1$ low-pass filter and as the companion model. It can be seen in the code segment that K, the pre-warp factor, is explicitly calculated every iteration.

3.4.2 Simplified Bi-quad First-order Model

In the case of the companion model, a first-order response may be desired. As described in equations A.2 and A.4, the discrete first-order model can be derived from a simplified Bi-quad as seen in Figure 3.9. It can be seen that the first coefficient of the IIR filter is kept from this topology.

The first-order model can be specified by either its time constant (time in seconds to reach 63% of steady-state) or its -3dB corner frequency. The system takes the form as seen in Equation 3.16 when defined by its corner frequency

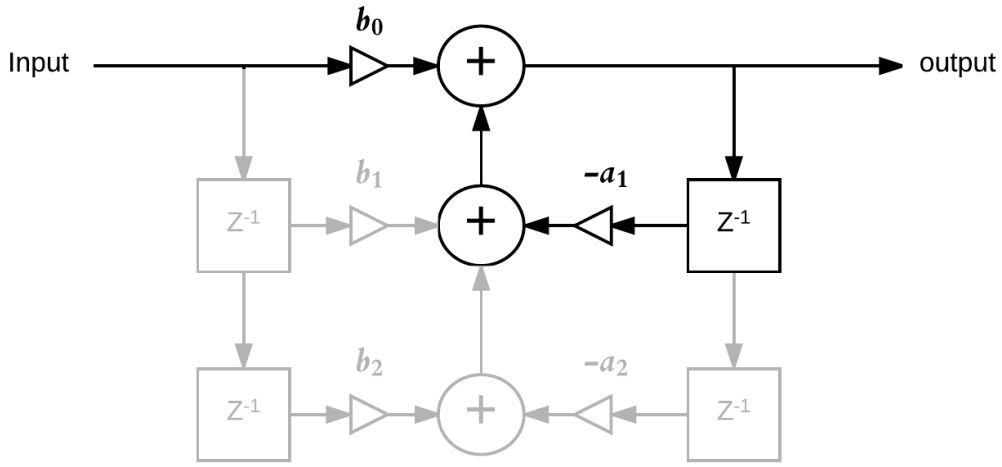

Figure 3.9. Digital Bi-quad Simplified First-Order Low-Pass Filter

$$H(s) = \frac{\omega_n}{s + \omega_n} \tag{3.16}$$

Therefore, the explicit calculation of the Bi-quad coefficients in this case becomes

$$\begin{aligned} a_1 &= e^{\left(\frac{-\omega_n}{F_s}\right)} \\ b_0 &= 1 - a_1 \end{aligned} \tag{3.17}$$

where ω_n is the -3dB corner frequency in radians per second and F_s is the sampling frequency measured in Hertz.

Therefore, the discrete recursive form of the first-order model becomes

$$y_{i+1} = a_1 y_{i-1} + b_0 y_i \tag{3.18}$$

Another form designed to optimize for speed that is commonly seen in software is

```
float b_0=exp(-f_c/f_s);
float out+=(in-out)*b_0;
```

3.4.3 Euler vs. Trapezoid Rule

The model estimate, as well as the parameter estimates for the $\mathcal{L}_1$ algorithm, are both numerically estimated using discrete integration. The Euler method is a numerical procedure for solving ordinary differential equations. The Euler method as applied to discrete integration is the fundamental method for recursively integrating a digital signal. The algorithm takes the form:
where h is the uniform step size,

$$y_{i+1} = y_i + hf(t_i, y_i) \tag{3.19}$$

The recursive trapezoidal method takes the form

$$\begin{aligned} \tilde{y}_{i+1} &= y_i + hf(t_i, y_i) \\ y_{i+1} &= y_i + \frac{h}{2}[f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})] \end{aligned} \tag{3.20}$$

Comparing the accuracy of the two numerical methods for discretely calculating the integral of $y = e^t$ can be seen in Figure 3.10

As seen in Equation 3.20, the recursive trapezoidal integration method only adds one more line of complexity to the algorithm for a significant gain in accuracy and therefore will be the chosen method applied for all discrete numerical integration in this research and takes the form

```
float trap_integration(float y0, float y1_dot, float dt, float &y0_dot)
{
    float y1 = y0 + (dt/2)*(y0_dot+y1_dot);
    y0_dot = y1_dot;

    return y1;
}
```

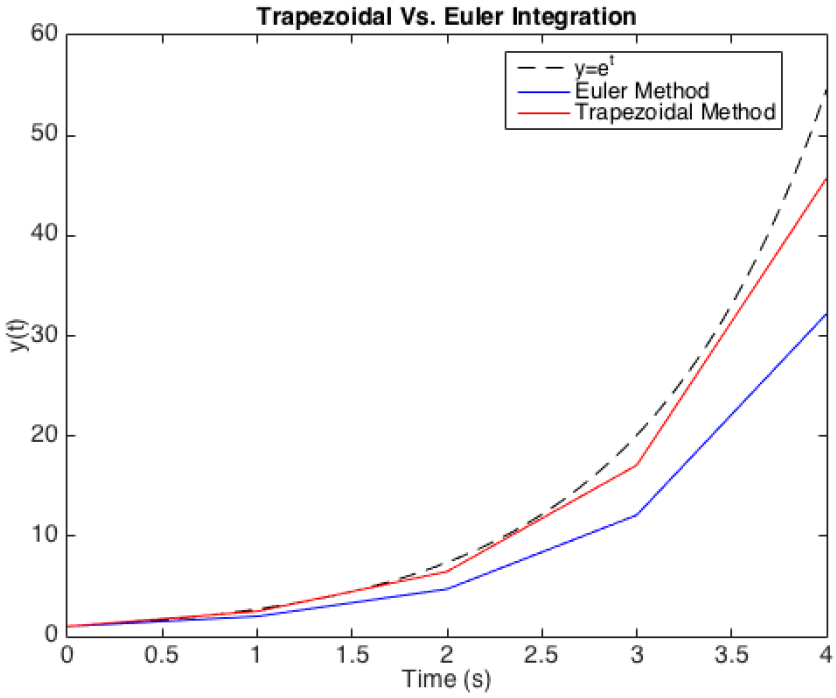

Figure 3.10. Euler vs Trapezoidal Integration Error

With the $\mathcal{L}_1$ adaptive control algorithm defined in discrete form utilizing the architecture in Figure 3.6 and the discretization methods outlined in Section 3.4 enabled the simulation testing found in Chapter 5. Flight testing was also desired and therefore multiple test aircraft were designed and built as seen in Chapter 4 with integrated COTS autopilots.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Design of Experimental Platform

This chapter presents the COTS autopilots used in this research as well as outlining the Linux toolchain used to conduct |acSITL and GCS operations. This chapter also introduces the airframes that were prototyped for testing specific performance characteristics discussed in Chapter 5.

4.1 Pixhawk Autopilot

The Pixhawk autopilot is a collaborative project among open-source engineers which resulted in a high-performance autopilot which is capable of controlling aircraft, ground vehicles, and many others. The primary reason this autopilot was chosen was because of the vast amount of support in the developer community. The Pixhawk 1 autopilot hardware is effectively obsolete at the time of this writing, but the open-source code base is extremely flexible and continues to be ported to new hardware as it becomes available. This has been the case for various Raspberry Pi autopilots as well as the Pixhawk 2. The Pixhawk autopilot operates using two flight stacks (code base/operating systems); the PX4 flight stack and the APM flight stack. This research was implemented on the APM flight stack primarily because the author's familiarity with the developer team which offers unparalleled assistance to the academic community. The APM codebase also offers a litany of open-source tools such as a Linux based GCS, SITL simulator, log analysis tools, and an application program interface (API) for the RealFlight 7.5 simulator (high fidelity airframe simulation for small aircraft).

Figure 4.1 illustrates the connection diagram for the Pixhawk 1 autopilot.

4.1.1 Key Features

The Pixhawk 1 Autopilot has the following features as found on [12]:

- 168 MHz / 252 MIPS Cortex-M4F

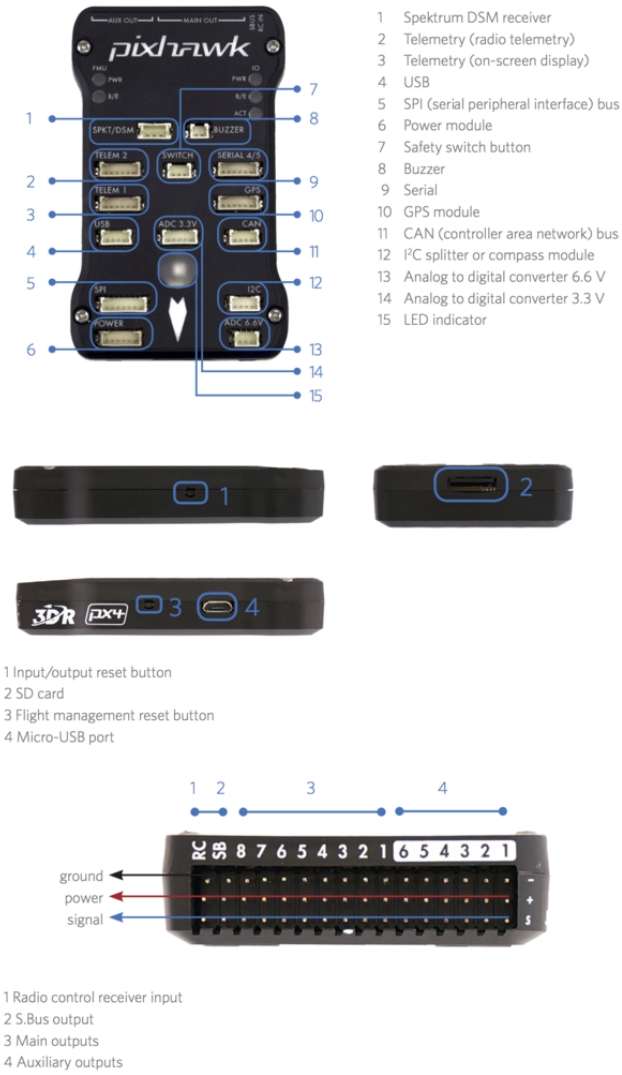

Figure 4.1. Pixhawk 1 Autopilot Connection Diagram. Source [12].

- 14 PWM / Servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
- Abundant connectivity options for additional peripherals (UART, I2C, CAN)
- Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply (fixed-wing use)
- Backup system integrates mixing, providing consistent autopilot and manual override mixing modes (fixed wing use)
- Redundant power supply inputs and automatic failover

- External safety switch
- Multicolor LED main visual indicator
- High-power, multi-tone piezo audio indicator
- microSD card for high-rate logging over extended periods of time

4.1.2 Specifications

The Pixhawk 1 Autopilot has the following specifications as found on [12]:

Processor

- 32bit STM32F427 Cortex M4 core with FPU
- 168 MHz
- 256 KB RAM
- 2 MB Flash
- 32 bit STM32F103 failsafe co-processor

Sensors

- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- Invensense MPU 6000 3-axis accelerometer/gyroscope
- MEAS MS5611 barometer

Interfaces

- 5x UART (serial ports), one high-power capable, 2x with HW flow control
- 2x CAN (one with internal 3.3V transceiver, one on expansion connector)
- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
- Futaba S.BUS® compatible input and output
- PPM sum signal input
- RSSI (PWM or voltage) input
- I2C
- SPI
- 3.3 and 6.6V ADC inputs
- Internal microUSB port and external microUSB port extension

4.2 Ground Control Station

The GCS used for this research was MAVproxy [13]. It is an open-source python based GCS which provides flexible communication and command with any autopilot utilizing the MAVlink protocol [14]. Even though MAVproxy is written in Python (operating system (OS) agnostics language), it was found to be cumbersome to operate the GCS on any other platform other than Linux. This is primarily because the source code updates quite rapidly to support new features and the core developer (Andrew Tridgell) exclusively utilizes MAVproxy in Linux. A significant amount of external libraries are utilized which results in a moderate amount of compatibility debugging for other OS's if desired.

4.2.1 MAVproxy Features

The following are summaries which proved to be extremely useful for this research [15]:

- command-line, console based application. Plugins included in MAVProxy provide a basic graphical user interface (GUI).
- network capable and run over any number of computers.
- portable; capable of running on any POSIX OS with Python, pyserial, and select() function calls, which means Linux, OS X, Windows, and others.
- light-weight design; runs on small netbooks.
- tab-completion of commands.

4.3 Simulation

The APM environment offers three versions of SITL simulations. The lowest fidelity SITL is provided by MAVproxy, which is a simple 6-degree of freedom kinematics model with no environment or actuator modeling. This proved to be adequate for initial testing but resulted in poorly tuned algorithms when actual flight tests were conducted. The MAVproxy simulator was used for basic code debugging but nothing else.

The second SITL offered in the APM environment is X-plane 10. This is a much higher fidelity simulation, which includes actuator models and environmental modeling. X-plane 10 is primarily used for simulating full-scale aircraft and therefore is difficult to find models, which accurately represent the dynamics of small fixed-wing UAS. The open-source

community provided model called the “maxi-swift” was similar enough to the airframe in this research that it provided adequate SITL modeling which ensured robust flight test.

The last SITL simulation tested under the APM environment was the RealFlight 7.5 API. The RealFlight remote controlled (RC) airplane simulator offers some of the industry’s highest fidelity simulations for small aircraft. This product requires an API key to hook into MAVproxy. This capability is not yet on the market as of the time of this research, but the APM core developers were supportive of this research and ran multiple experiments with the RealFlight SITL for early validation. The screenshot in Figure 4.2 was captured while conducting testing utilizing the RealFlight SITL.

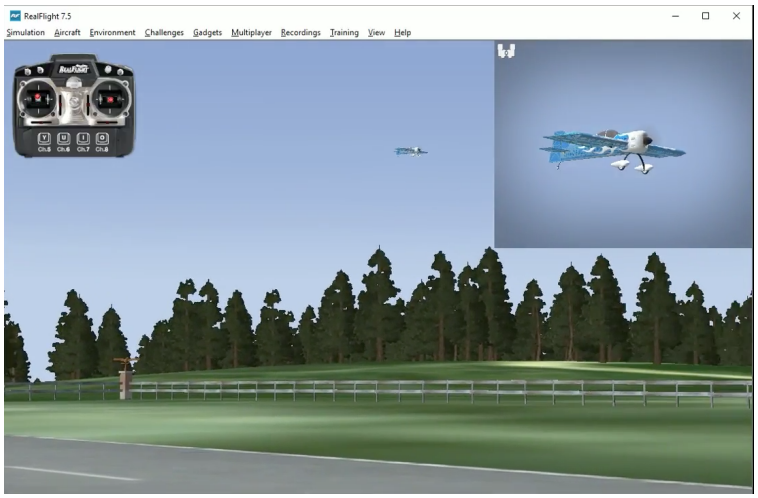

Figure 4.2. High Fidelity RealFlight 7.5 Software in the Loop (SITL)

4.4 Airframe

The aircraft used for this research was the Flitetest Spear and the Flitetest Explorer [16]. The Spear airframe was chosen for its endurance capability of greater than 45 minutes of flight time and its large capacity fuselage. The flying-wing architecture keeps the actuation requirement to a minimum of two servos by utilizing an elevon configuration.

Figures 4.3 and 4.4 are example photos from the instructional build website [16]. Figure 4.5 illustrates the process of building the Spear aircraft. The large blunt nose provides adequate space for two 2,200 mAh (12.6volts) lithium polymer batteries wired in parallel. The remaining cargo space was used for accommodating the Pixhawk autopilot.

Figure 4.3. Spear Airframe. Source [16].

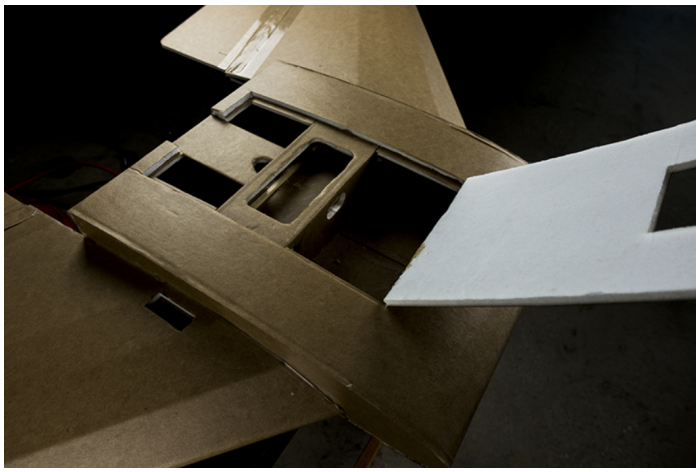

Figure 4.4. Spear Cargo Capacity. Source [16].

This plane was constructed out of craft foam board. The plans were downloaded from flitetest.com [16] and converted to CorelDraw vector files for use in a laser cutter. These files were then cut out of four sheets of foam board using the laser cutter. The wing halves were joined with standard box tape and hot glue. This provided a cheap and rapid construction process which was achievable under four hours of build time.

4.4.1 Spear Specifications

- weight without battery: 1.45 lbs (658 g)
- center of gravity: 3 – 3.5" (76 – 89 mm) in front of firewall
- control surface throws: 16°deflection – Expo 30%
- wingspan: 41 inches (1041 mm)

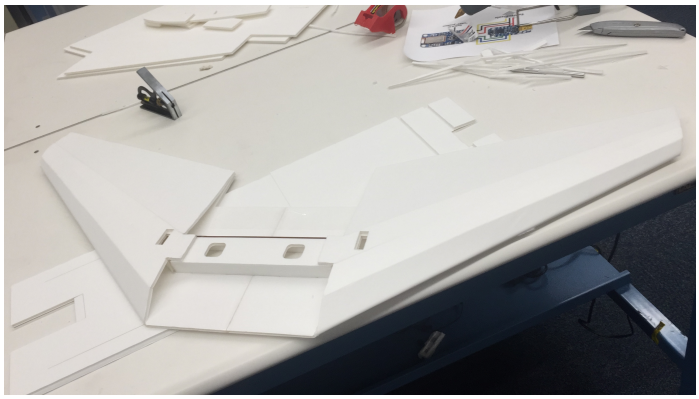

Figure 4.5. Spear Build Process

- motor: 425 sized, 1200 kv minimum
- prop: 9 x 4.5 CW (reverse) prop
- electronic speed control (ESC): 30 amp minimum
- battery: (2) 2200 mAH 12.6 volt minimum
- servos: (2) 9 gram servos

The Explorer airframe was chosen because it is a conventional airframe with highly coupled aerodynamics which can be configured in multiple different failure modes. The sport wing provided in the plans was modified to have independently actuated flaps, and ailerons for a combination of software enabled failure modes. This aircraft was also configured with a rudder for testing the lateral aerodynamics coupling effects on the adaptive controller. Figure 4.6 illustrates the completed Explorer airframe used in this research. Figures 4.7 and 4.8 are examples of the Explorer build process and autopilot integration.

4.4.2 Explorer Specifications

- weight without battery: 1.08 lbs (493 g)
- center of gravity: 2.25” (57 mm) from leading edge of wing
- control surface throws: 12°deflection – Expo 30%
- wingspan: 57 inches (1447 mm)
- motor: 425 sized, 1000 kv minimum
- prop: 9 x 6 CW (reverse) prop
- ESC: 30 amp minimum
- battery: (1) 2200 mAH 12.6 volt minimum

- servos: (6) 9 gram servos

Figure 4.6. FliteTest Explorer. Source [16].

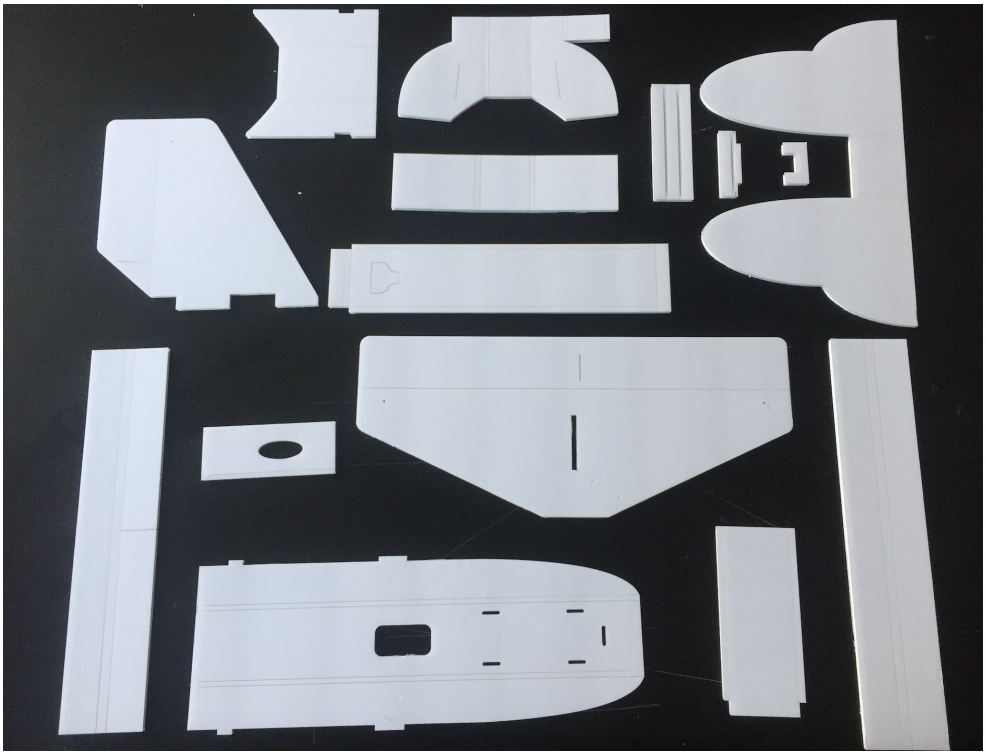

Figure 4.7. Explorer Cut Foamboard Parts

The previously discussed COTS autopilot, Linux software tool chain for SITL and GCS, and prototype aircraft were utilized to present the results found in Chapter 5.

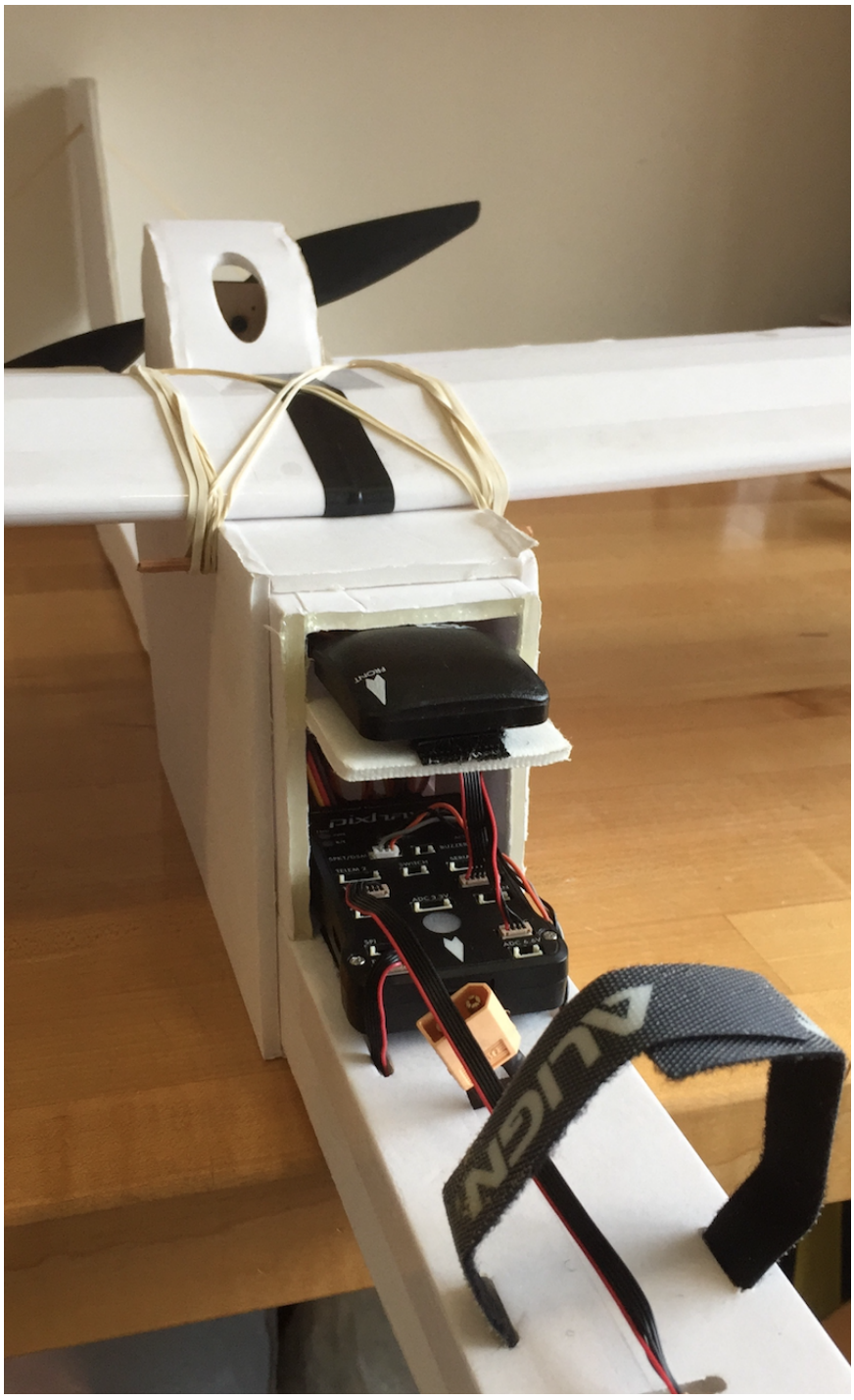

Figure 4.8. Pixhawk Autopilot Installed on Explorer

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Flight Testing and Performance Evaluation

This chapter utilizes the APM SITL architecture as well as prototype aircraft to tune and test the $\mathcal{L}_1$ adaptive control algorithm to highlight performance parameters which meet or exceed the underlying requirements established in the problem proposal in Chapter 1. For all graphs in Chapter 5, the green background identifies when the adaptive control algorithm was being utilized.

5.1 Simulation Results

Simulation results were captured utilizing the APM SITL using X-Plane10. The aircraft model chosen was an open-source flying wing model called the maxi-swift as seen in Figure 5.1, a simulation photo.

Figure 5.1. Maxi-Swift Flying Wing Model Used in X-Plane10

This plane did not afford proper testing of elevon mixing, but the fidelity in the model provided utilizing X-Plane10 was surprisingly accurate with respect to the FT-Spear aircraft used for actual flight test. This drastically increased confidence that anything tested in SITL would have a high probability of success in actual flight test.

5.1.1 Pitch Attitude Performance

It can be seen in Figure 5.2 that there exists some steady state error when under PID control. This phenomenon is more pronounced when the desired pitch attitude is negative. This steady state error is due to the increase in lift caused by the increasing airspeed. The PID controller underperforms in this regime. However, the adaptive controller is able to compensate for these dynamics fast enough to track the desired attitude with no steady state error. The green background in Figure 5.2 identifies when the adaptive control was being utilized and the white background was utilizing the PID control.

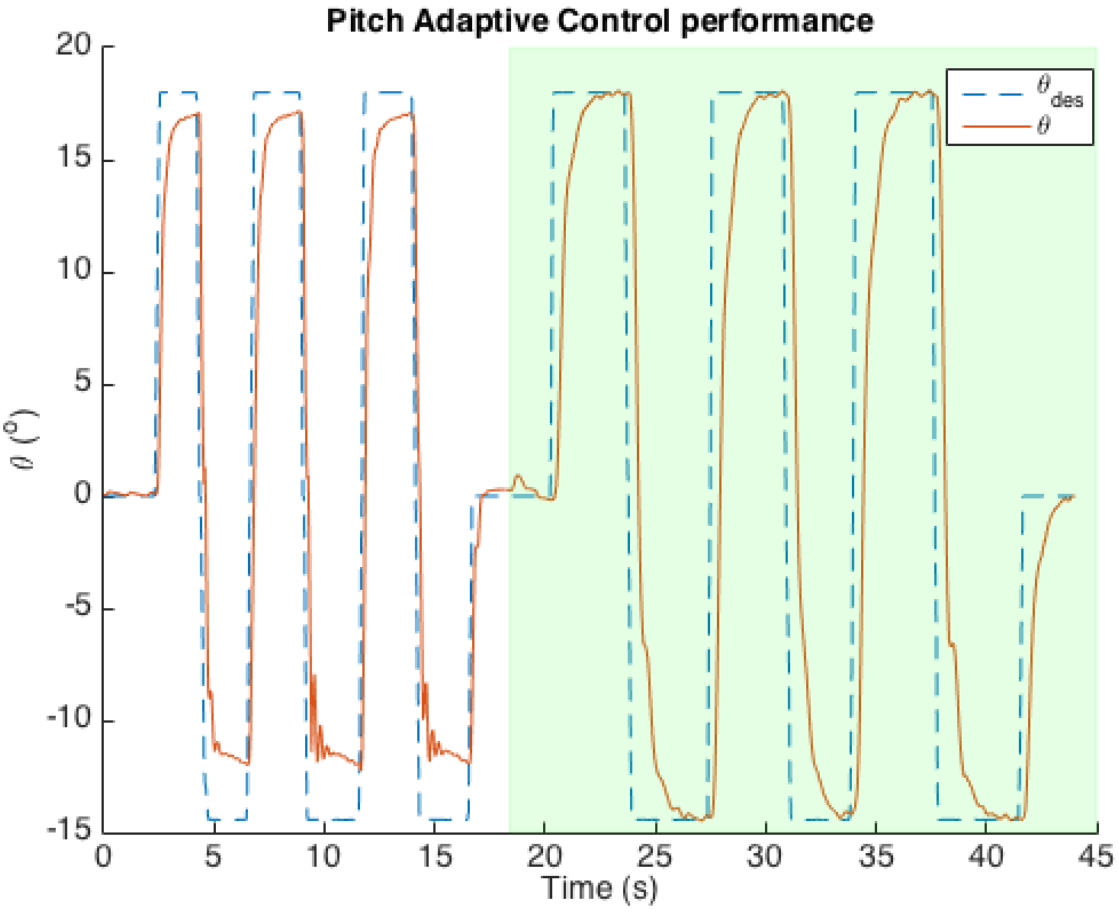

Figure 5.2. PID vs $\mathcal{L}_1$ Adaptive Control Pitch Performance

5.1.2 Roll Induced Pitch Disturbance

When rapidly rolling the maxi-swift aircraft in simulation, there was a noticeable coupling in the pitch axis. The PID controller struggles to correct this discrepancy because the time constant of the integral error simply cannot be increased high enough to achieve satisfactory compensation. As seen in Figure 5.3, the $\mathcal{L}_1$ adaptive controller significantly reduced the pitching disturbance due to rapid rolling.

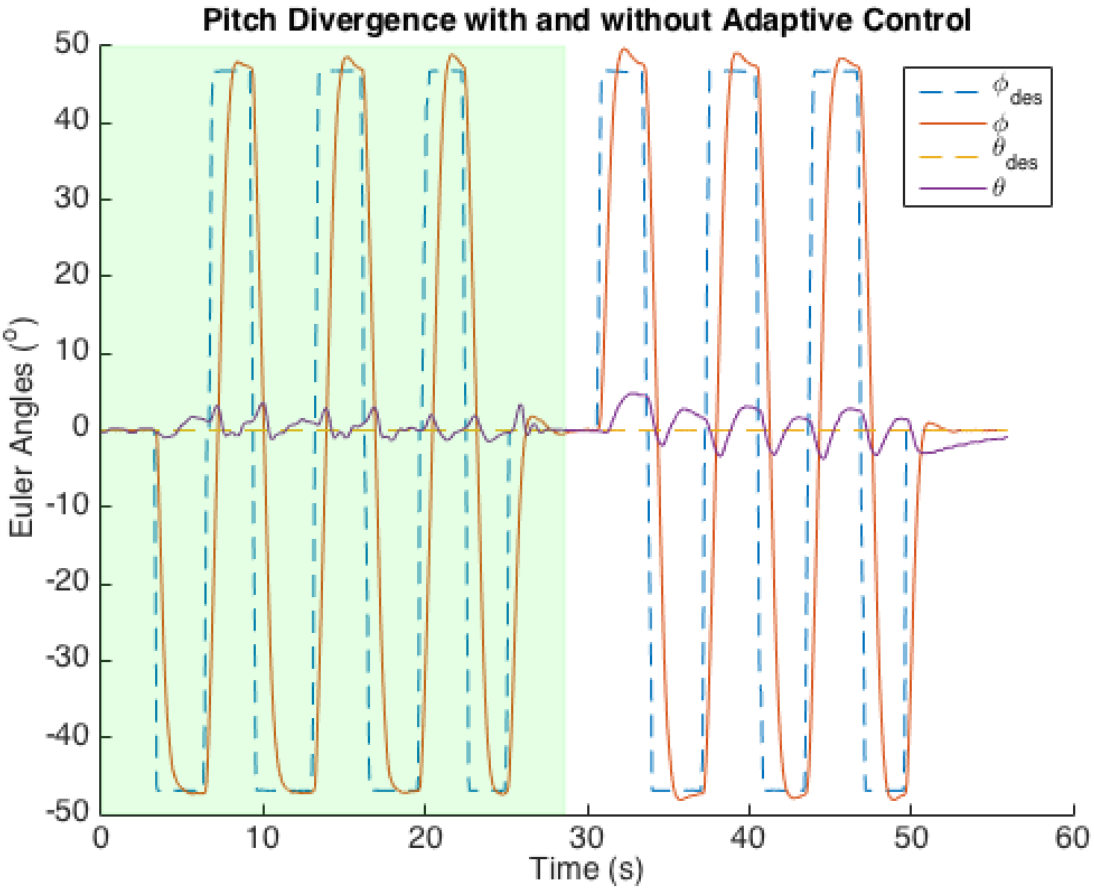

Figure 5.3. PID vs $\mathcal{L}_1$ Adaptive Control Coupled Pitch Response Performance

5.1.3 Pitch due to Landing Gear and Flaps

Lowering the landing gear and flaps entering the landing phase of flight causes uncommanded deviation in pitch. The F4U Corsair (see Figure 5.4) in X-Plane10 was used to evaluate the attitude hold retention performance. This un-modeled aerodynamics can cause

the integrator in a PID controller to saturate or wind up. Figure 5.5 illustrates the pitch performance during the actuation of flaps and gear with and with out adaptive control.

Figure 5.4. F4U Corsair Model in X-Plane10

The $\mathcal{L}_1$ adaptive controller significantly reduces the attitude excursion due to flaps and landing gear.

5.1.4 Roll Performance

Adaptive control offered little improvement to roll performance with respect to the nominal attitude retention. The roll axis for multiple aircraft was typically seen to have higher cutoff frequencies with respect to pitch and therefore required higher values for the $D(s)$ cut off frequencies. Figure 5.6 illustrates that the adaptive controller is capable of achieving equal or improved performance over the PID controller.

5.1.5 Yaw to Roll Coupling

The roll dynamics are coupled to the yaw dynamics as seen in Equation B.11. In the case of an actuator/servo hardover in the yaw channel, the coupling causes unwanted rolling moments. Figure 5.7 compares a left and right yaw servo hard over for both PID and the $\mathcal{L}_1$ controller. The adaptive controller significantly outperforms the PID controller in maintaining the desired roll. It could be argued that the PID controller's integrator time constant could be re-tuned to be comparable. It was evident for each of the simulation

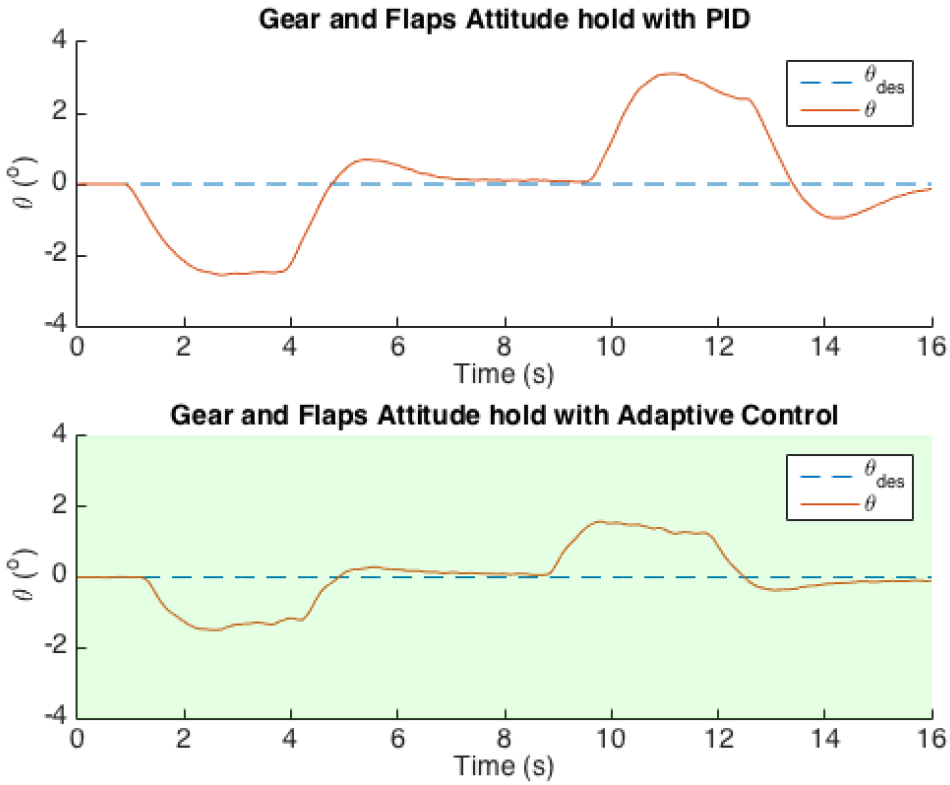

Figure 5.5. Pitch Attitude Response Due to Gear and Flaps

tests that tuning the PID for every scenario would likely have been possible individually to achieve similar performance to the $\mathcal{L}_1$. However, the $\mathcal{L}_1$ was not tuned between each of these tests and outperformed PID in most, if not all cases.

5.1.6 Actuator Miscalibration

The $\mathcal{L}_1$ adaptive control provided fast learning of airframe actuator miscalibrations. The autopilot parameter SERVOX_TRIM was used to offset the ailerons by 15% of their travel to evaluate how quickly the controller was capable of adapting to the new offset. This was tested first on the PID controller and then on the $\mathcal{L}_1$ as seen in Figure 5.8.

It can be seen in Figure 5.8 and 5.9 that the new trim value achieves the reference command in about 0.5 seconds for the $\mathcal{L}_1$. The $\mathcal{L}_1$ is slightly faster than PID, but the $\mathcal{L}_1$ exhibits some overshoot. It is important to note that this rate of learning is perfectly adequate for learning the miscalibrations in flight, but is insufficient for learning on take off. In the

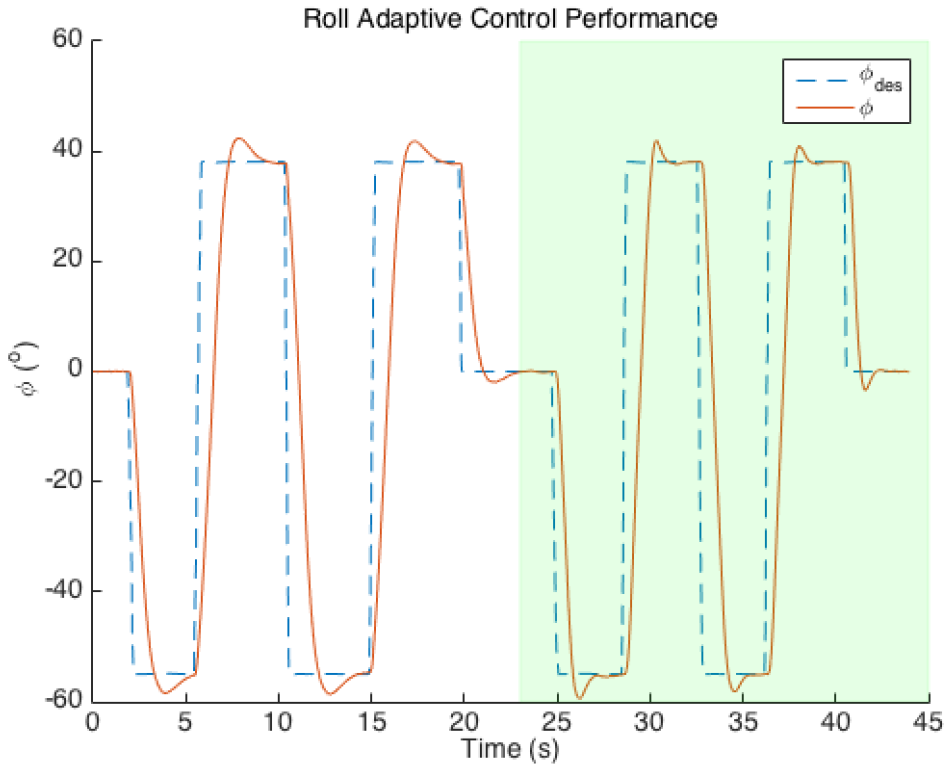

Figure 5.6. Roll Attitude Performance

takeoff circumstance, the algorithm saturates the actuator if biases exist between desired and achieved. Because the aircraft has no dynamic pressure (it is not flying), the desired cannot be achieved. This causes controller saturation, which cannot be unwound fast enough for take-off (specifically for tail-dragger configuration). This has to be handled with ad-hoc heuristics very delicately. One could choose to speed scale the learned parameters to help with learning rate, but it was not chosen for this research because this controller is also utilized in tail-sitter configurations where the zero airspeed would cause the controller to refuse to learn when in VTOL mode.

5.2 Flight Test Results

After successful results were achieved utilizing the SITL environment across multiple aircraft, similar test techniques were conducted on the prototype aircraft described in Chapter 4.

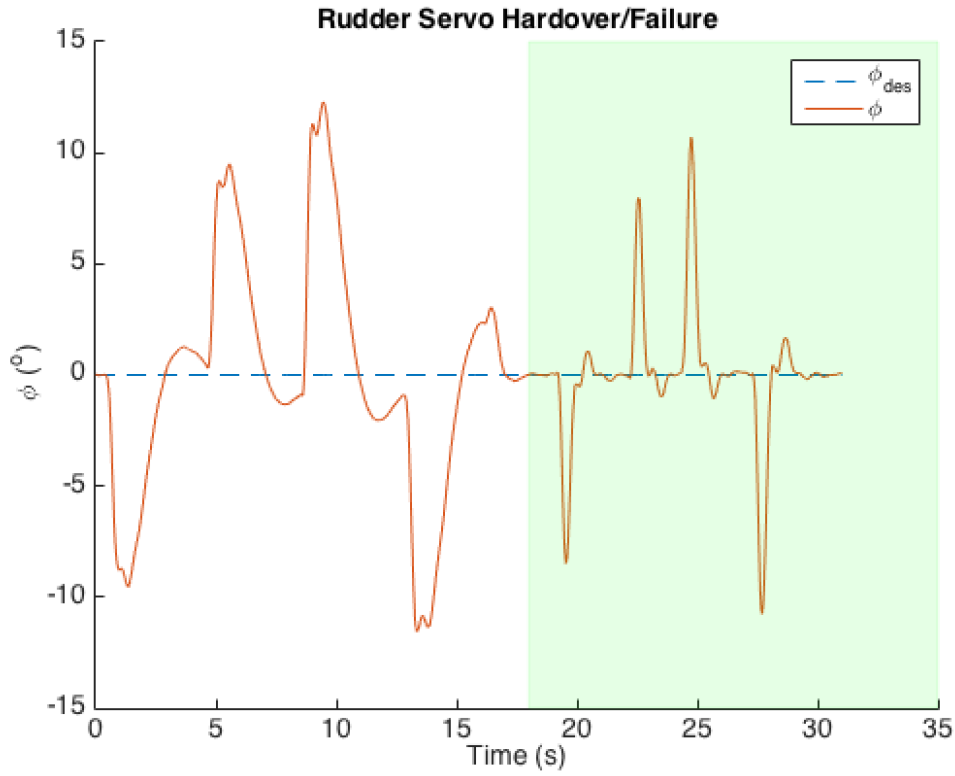

Figure 5.7. Roll Response to Rudder Servo Hardover/Failure

5.2.1 General Observations

The FT Spear aircraft was used in all initial flight tests to prove the algorithm's initial robustness and facilitate testing of various tuning scenarios. Choosing this aircraft proved to be beneficial because it provided 30 or more minutes of endurance. The flying wing architecture had a slight forward CG when configured with two 2200 milli amp hour (mAh) batteries. This resulted in the aircraft being anemic in pitch response but still adequate for testing the $\mathcal{L}_1$ algorithm. Various iterations of the algorithm were tested and compared to the results observed in SITL as the development progressed.

The first observation was that the algorithm, even when poorly tuned, was bounded in response and never approached instability. The gains found in SITL when applied to actual flight test always resulted in stable flight. As discussed in Section 6.1, the first flight tests of gains found in SITL either produced low-frequency oscillations or high-frequency oscillations. Low-frequency oscillations occurred when the $\mathcal{L}_1$ filter cutoff bandwidth was set too low. High-frequency oscillations occurred when the $\mathcal{L}_1$ filter cutoff bandwidth was

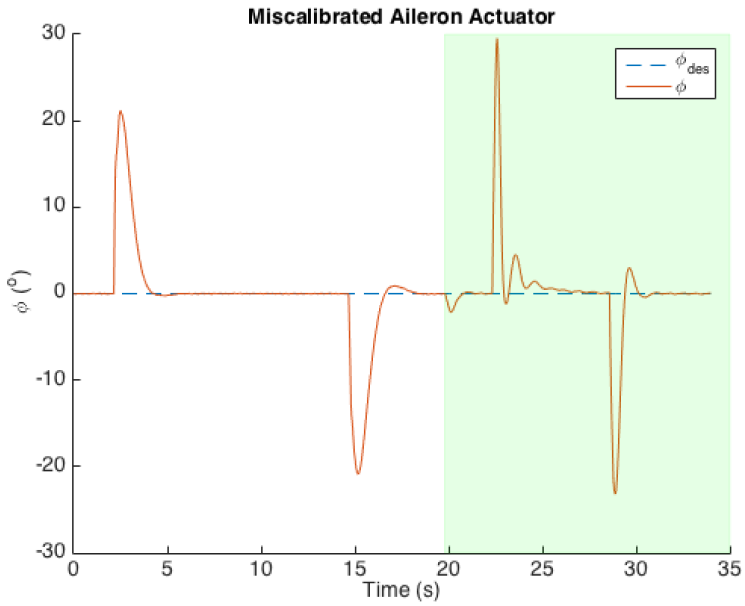

Figure 5.8. Roll Response to Miscalibrated Aileron Servo

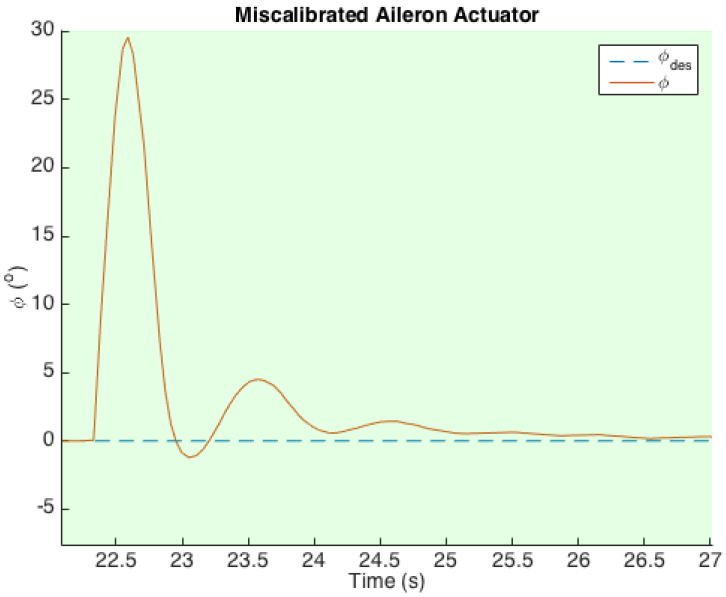

Figure 5.9. $\mathcal{L}_1$ Fast Adaptation to Unknown Miscalibration

set too high. Tuning the filter to achieve adequate response somewhere between the low and high-frequency oscillations was easily achievable within three to four iterations of setting the filter cutoff frequency. Finding the optimal filter setting is arguably difficult to see on

the low refresh rate telemetry, but achieving similar or better performance than PID was easily attained with very minimal effort.

Once the algorithm was well understood and properly implemented in code, flight tests were then conducted on the FT Explorer aircraft. This aircraft provided more test opportunities because multiple aircraft configurations could be tested with various combinations of flaps and rudder.

5.2.2 Pitch Performance

As demonstrated in Figure 5.10, the simulation results almost identically matched the flight test results for pitch response. As the aircraft pitches down, the airspeed builds and causes a building steady state error that the integrator time constant struggles to out pace. This was observed both in simulation and in flight test. The $\mathcal{L}_1$ adaptive controller's fast adaptation was sufficient to counteract this unmodeled pitching moment. These results drastically increased the confidence both in the SITL fidelity as well as the $\mathcal{L}_1$ adaptive controller.

5.2.3 Pitch due to Landing Gear and Flaps

The $\mathcal{L}_1$ adaptive controller performed well when compensating for the change in aircraft configuration when lowering the flaps. Figure 5.11 illustrates that the tracking error from lowering and raising the flaps under adaptive control is almost imperceptible. However, PID resulted in a sharp peak both when lowering and raising the flaps.

5.2.4 Roll Induced Pitch Disturbance

The aircraft max roll angle was set to ±45°to achieve rapid roll while trying to maintain a zero pitch attitude. This rapid roll maneuver caused significant excursions in pitch when under PID. However, the $\mathcal{L}_1$ controller maintains the pitch attitude within ±3°. It can be noticed in Figure 5.12, that the adaptive controller oscillates around zero pitch attitude, while the PID has more random excursions. The slight sinusoidal behavior or the $\mathcal{L}_1$ controller is likely due to the cutoff frequency of the filter being slightly too low. This maneuver may possibly be utilized to fine tune the $\mathcal{L}_1$'s filter cutoff frequency for pitch.

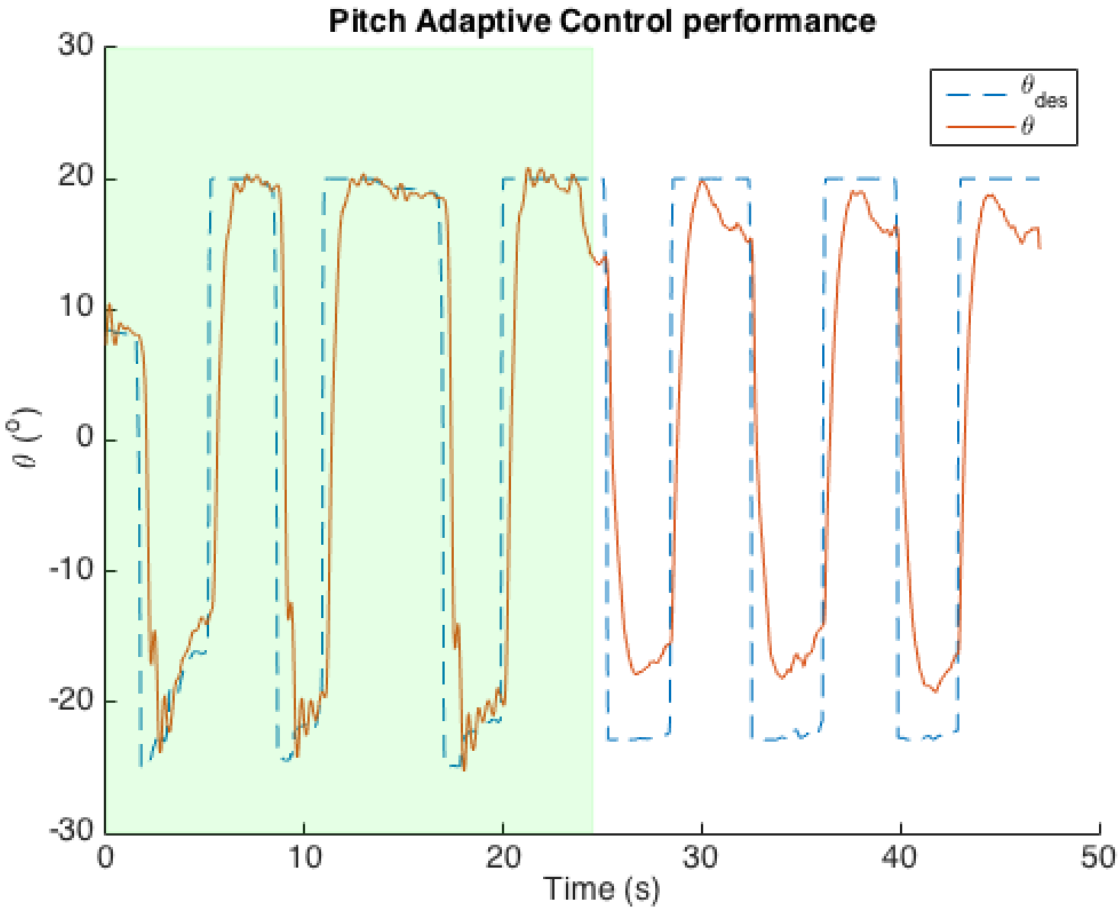

Figure 5.10. PID vs $\mathcal{L}_1$ Adaptive Control Pitch Performance

5.2.5 Roll Performance

To test the nominal flight path stability and roll attitude performance, a rectangular flight plan was established. Figure 5.13 illustrates the $\mathcal{L}_1$ adaptive controller while maintaining the left hand pattern. The noise in the roll channel was observed in actual flight as turbulence. The filter cutoff frequency could be lowered to reduce performance if desired, but these results were not oscillatory which suggested that the filter cutoff frequency was not too high. These results compare to the noise found on the PID channel so the threat of damaging servo by over driving them was not expected.

Asymmetric flaps were configured by only allowing one servo to lower the left flap while leaving the right flap stationary. This failure was designed to test the unmodeled roll response from a very asymmetric mechanical failure. Unfortunately, the roll and pitch

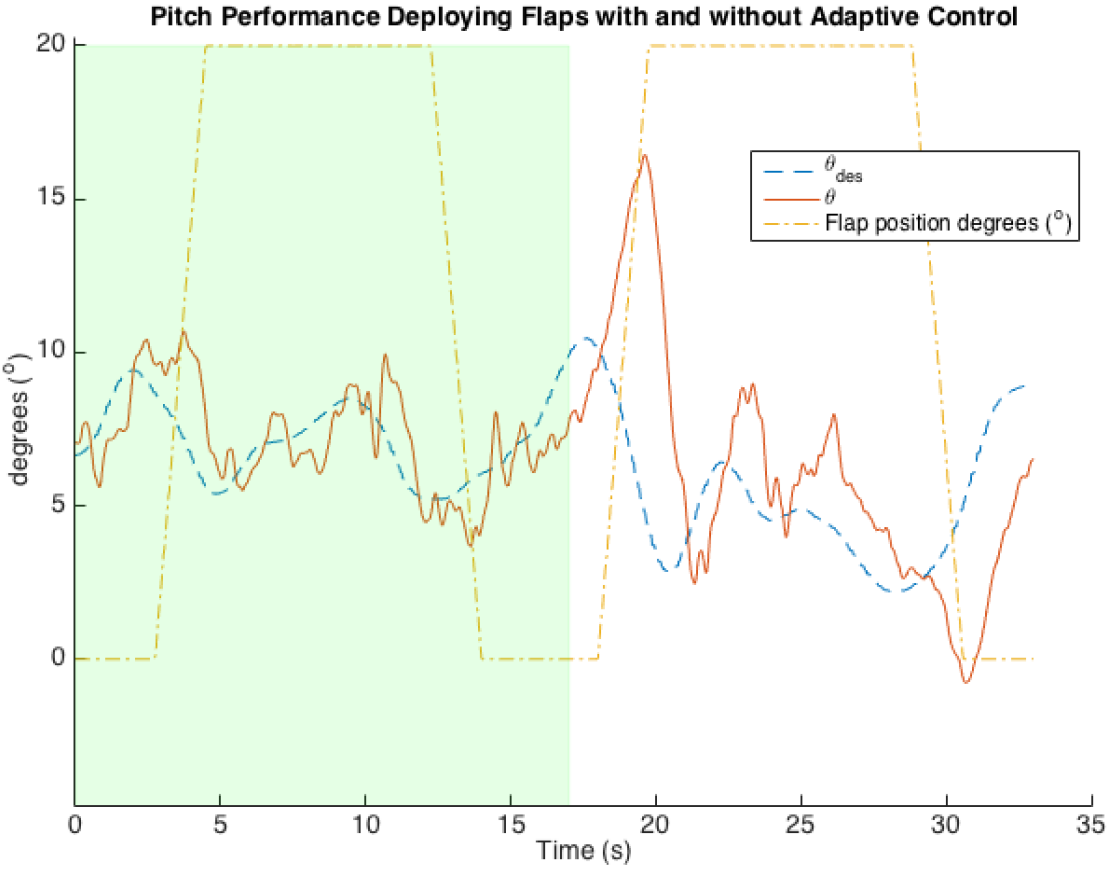

Figure 5.11. Pitch Attitude Response Due to Gear and Flaps

excursions caused by this failure were marginal. Both PID and the $\mathcal{L}_1$ handled this failure with adequate reliability.

The overall result of the performance testing of the $\mathcal{L}_1$ algorithm was quite successful. The results from SITL provided confidence in the expected results from actual flight test and proved to be robust representations of the results achieved in flight test. Flight test were significantly more difficult to conduct, but the mirrored performance between the SITL and flight tests proved that more time can be spent utilizing SITL to drastically reduce development cost and schedule.

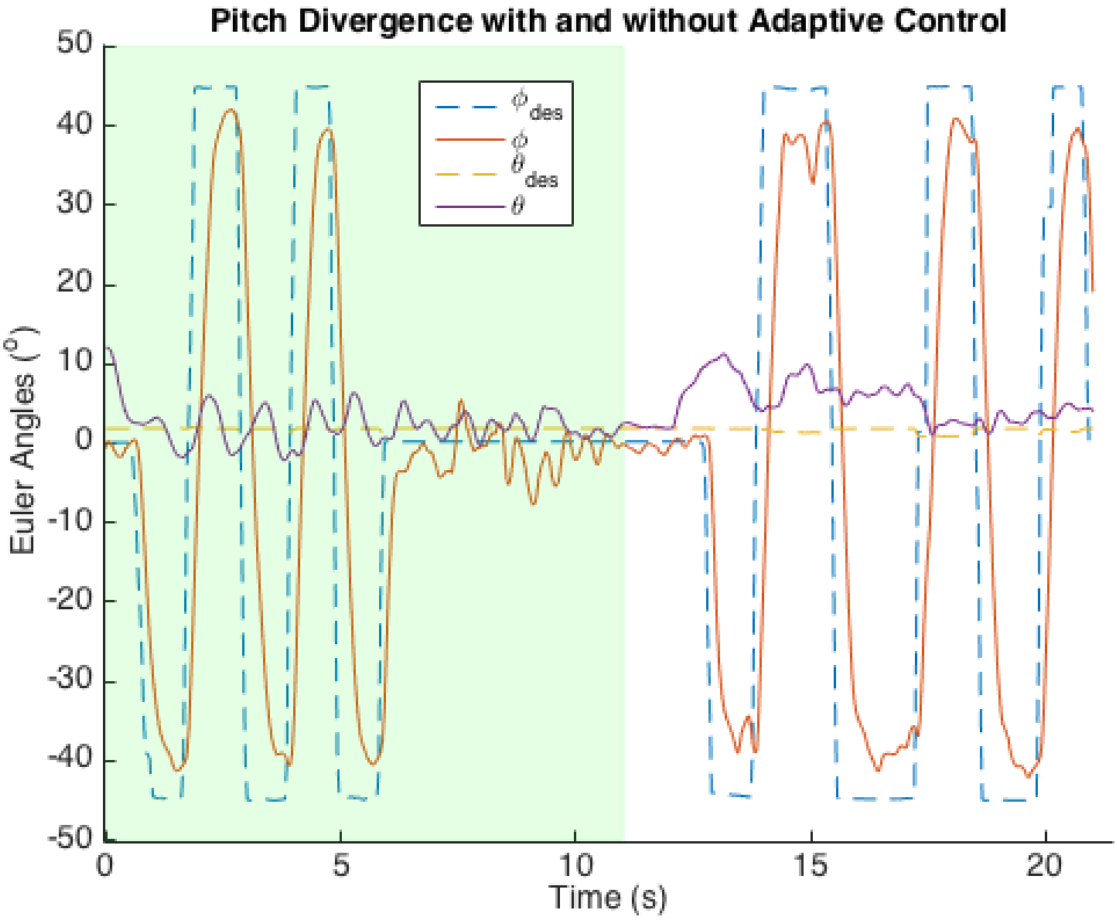

Figure 5.12. PID vs $\mathcal{L}_1$ Adaptive Control Coupled Pitch Response Performance

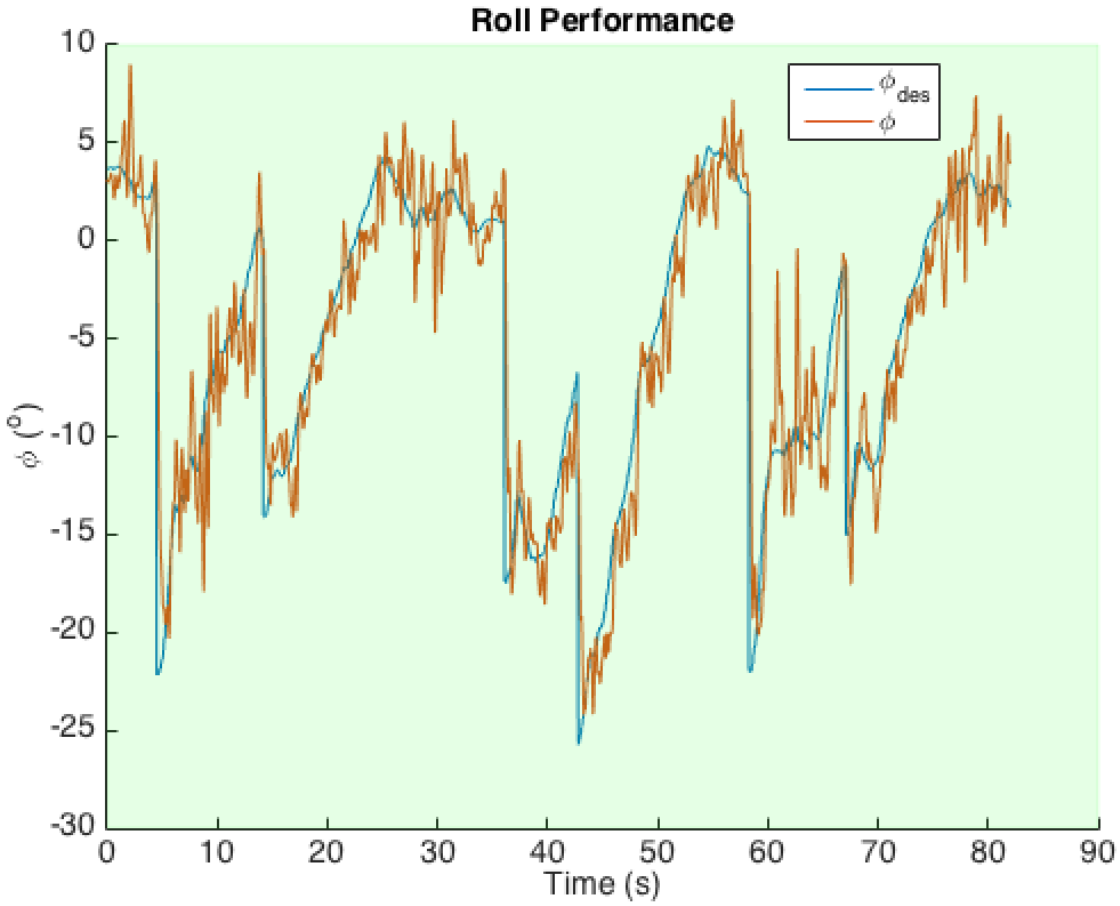

Figure 5.13. $\mathcal{L}_1$ Adaptive Control Roll Performance

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Recommendation

This chapter provides various recommendations pertinent to the integration of the $\mathcal{L}_1$ adaptive control algorithm. These recommendations may be more qualitative than quantitative, but provide experience based guidance for the engineer to better understand the nuances of the $\mathcal{L}_1$ adaptive control algorithm. Prior to this research, there was very little experientially based guidance on implementation of this algorithm which proved to be the largest barrier for implementing this modern controller.

6.1 $\mathcal{L}_1$ Adaptive Control Algorithm Tuning

The primary goal of this research was to reduce the complexity of tuning expertise required to get an unknown airframe airborne successfully. The amount of time required to get the adequate airframe performance using the $\mathcal{L}_1$ algorithm is significantly reduced. Three primary features need tuning: the adaptation gain, the controller filter, and the companion model.

6.1.1 Tuning the Adaptation gain

Tuning the adaptation gain is fairly intuitive. The adaptation gain is a function of the loop frequency at which the filter is run so it may only need to be tuned for a given autopilot at a given loop rate and not for every specific aircraft. Anecdotally, the adaptation gain of 10,000 was used on the Pixhawk1 autopilot running the scheduled loop rate at 100Hz across multiple aircraft without needing to modify. The primary feedback to the user for tuning this gain resides in the desired movement of the estimated states. The adaptation gain was set low (1-10) while watching the parameters (θ, ω, σ) adapt. The adaptation gain is slowly increased until the desired rate of adaptation as seen in the real time monitoring of the parameters is adequate. Another approach for tuning the adaptation rate is to increase the gain until parameters start to oscillate between the bounds and then reduce it. This bang-bang response in the parameter adaptation will show up in the performance of the controller as increased peaks away from zero error between desired and achieved. In the case of this research, this noise can be hard to identify in the rate control itself and therefore

is why monitoring the attitude error helps find the appropriate adaptation gain which is extremely high but still not injecting attitude error spikes.

6.1.2 Tuning the Controller Filter

The adaptation feedback gain (k), as discussed in Section 3.3 was by far the most influential gain to tune. The simplicity of tuning the $\mathcal{L}_1$ algorithm resides in the fact that the majority of lay users could adequately tune an airframe with this stand-alone gain. Adequate values for this gain ranged from 0.3 for responsive aircraft and 2.0 for very sluggish aircraft. This value was set for both the roll and pitch axis independently. As seen in Equation 3.8, this value is establishing the cutoff frequency of the control filter that separates the bandwidth limited control channel output and the high bandwidth adaptation. The default value assigned in the source code was 0.45, which proved to be a good starting point. If the default value for k is not correct, then the control channel will produce either low-frequency oscillations or high-frequency oscillations. The low-frequency oscillations are produced because the bandwidth of the control is too low and there should be a perceptible lag between the desired state and the achieved state. High-frequency oscillations occur when the control filter bandwidth is set higher than the plant's bandwidth, and the aircraft is incapable of achieving the desired rates. Unlike PID control, the $\mathcal{L}_1$ control never exhibited unstable performance with incorrect gains. The controller simply oscillates with extremely poor performance. This was a remarkable feature because poorly tuned PID gains can cause an aircraft to depart controlled flight rapidly. Whereas, the $\mathcal{L}_1$ controller maintained bounded flight performance as the theory suggests.

6.1.3 Tuning the Companion Model Cutoff Frequency

System identification was conducted as seen in Appendix C in order to ascertain the bandwidth of various airframes and their actuators. Figure 6.1 is an example of second-order models for two aircraft's roll dynamics compared to second-order models of RC actuators (servos). As to be expected, the bandwidth for the actuators is slightly higher than the airframe dynamics.

These rough approximations were used to then place the cutoff frequencies of the companion model with the expectation that the companion model cannot achieve higher bandwidths than that of the airframe. Conservatively, the companion model cutoff frequency was

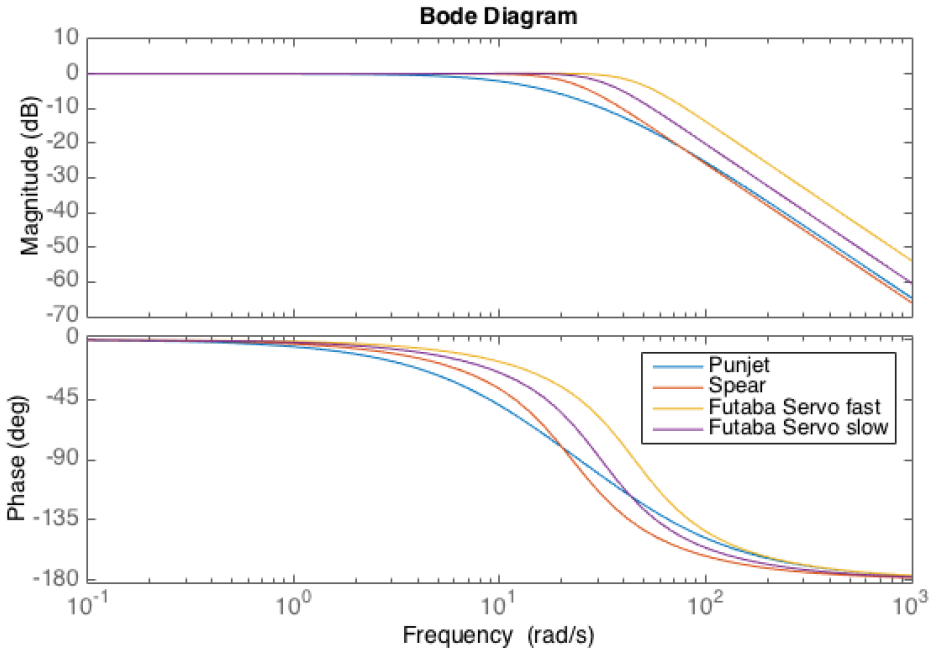

Figure 6.1. Aircraft Frequency Analysis

typically set 2-5 rad/s lower than the expected max performance of the airframe. After the other algorithm gains are tuned, and satisfactory performance is achieved, the companion model cutoff frequency can then be increased to achieve higher performance.

6.2 Improved Recursive Architecture

The speed of adaptation and accuracy of the discretized $\mathcal{L}_1$ algorithm is drastically improved with increased loop frequency. The algorithm was written as one recursive architecture that updates at the Pixhawk's scheduled loop rate. As previously discussed, the scheduled loop rate ideally should run at lower frequencies to prevent excessive log file size and added strain on the CPU. However, the $\mathcal{L}_1$ architecture only requires the adaptation loop be run at faster rates. With this specific performance enhancement in mind, the APM architecture could be modified to accept an independent loop specifically for the $\mathcal{L}_1$ adaptation update. The sensors measurements and extended Kalman filter (EKF) update can run significantly lower with only increased performance of the algorithm. The higher adaptation loop would enable higher adaptation gains (Γ) and consequently produce faster adaptation of the system.

6.3 Integrator Windup Issue

The $\mathcal{L}_1$ controller architecture exhibits a similar response to integrator wind up in PID control when the aircraft is not flying. If the controller was enabled before takeoff, the control surfaces would move to counteract any slight deviation from input to output attempting to zero the error. Because the aircraft was not flying, the controller would continue to increase the control output until the actuators reached saturation. The baseline code was configured to ensure that the parameter estimates would not continue to integrate after saturation was reached. This technique is standard practice when writing control laws that utilize actuators that have saturation limitations. However, the anti-windup feature that this offers only occurs when the actuators are saturated. This is inadequate for the takeoff scenario. The flight surfaces quickly saturate while waiting for takeoff and cause a crash immediately upon takeoff. It is standard practice to also disable the integrator action of controllers if it is known that the aircraft is not flying using any combination of airspeed, throttle position, inertial measurement unit (IMU) estimated velocity, or global positioning system (GPS) ground speed. In the case of the $\mathcal{L}_1$ algorithm, the integrator utilized in $D(s)$ is essential to the entire architecture and presented challenges in how to use ad-hoc methods to prevent the integrator windup issue even though the "non-flying" state was calculable onboard the autopilot. The initial experiments were to see if the adaptation rate was fast enough to un-learn the aircraft's saturated state, but there were no combinations of filter gains which resulted in learning rates adequate to ensure safe takeoffs. Further research in this area is required to completely replace the PID controller with the $\mathcal{L}_1$. All takeoffs were conducted either in manual control or with the PID controller enabled until safely airborne.

In summary, no one manual has been created for this type of controller's implementation and integration. The recommendations provided may only apply to this specific implementation of the $\mathcal{L}_1$ adaptive controller, but it offers guidance where none previously was articulated in contemporary literature.

CHAPTER 7:
Conclusion

The primary objective of this thesis was to evaluate a modern control technique to determine if the advanced controller could reduce the Navy's growing GNC demand on engineering resources.

The $\mathcal{L}_1$ adaptive control algorithm suggested and developed in this thesis proved to be a successful alternative to the conventional PID control strategy which drastically reduced the cost and time requirements to achieve robust flight. In addition to meeting the objective of lowering the engineering demand of the aircraft GNC algorithms, the $\mathcal{L}_1$ adaptive control could also provide battle damage tolerance and achieve more accurate control with the utilization of fast and robust adaptation.

Conversion of the continuous time $\mathcal{L}_1$ algorithm proved to be difficult with limited precedent literature but proved to be quite achievable with very little limitation posed by contemporary embedded processors. The APM flight stack open source project proved to be a very versatile software base that enable rapid prototyping and testing of the $\mathcal{L}_1$ algorithm.

The use of SITL was a critical tool in the development of this algorithm into source code. Follow on research or implementation of aircraft GNC in the simulation environment drastically reduces the project risk at little to no expense using contemporary high fidelity aircraft simulation tools.

Flight test of the $\mathcal{L}_1$ algorithm highlighted many capabilities gained through the use of fast and robust adaptation. The primary goal of reducing the GNC engineering demand required to achieve successful robust flight of UAS's was evident even on the first test flights. Achieving adequate flight performance was now attainable within a matter of minutes instead of multiple tests spread across multiple flights potentially spread across multiple days. With the use of SITL integration, a robust model of the airframe, and more dedicated research, successful first flights with no tuning required are now within the realm of possibility.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
Transfer Functions

A.1 Transfer Functions

This research utilizes the transfer function (TF) representation of aircraft flight dynamics that is typical of linear time invariant (LTI) systems. A transfer function is a very useful approach to describe the relationship between inputs and outputs of LTI systems. Both analytically and numerically, the TF approach has significant benefits in continuous and discrete time domains as its construct is based on well-developed properties and primitives of polynomials. These polynomial representations in the s or z domain map to aerodynamic and inertial coefficients through equations of motion. What is unknown or partially known a priori, are the numerical values of coefficients for those polynomials. Therefore the tools from the areas of online estimation such as regression are utilized to solve for them.

Transfer functions take the form

$$H(s) = \frac{Y(s)}{X(s)} \tag{A.1}$$

where

Y(s) is the Laplace transform of the output
X(s) is the Laplace transform of the input

Standard physics models of first and second order form are well understood and seen in many model derivations. The first-order model takes the form

$$H(s) = \frac{k_{dc}}{\tau s + 1} = \frac{\omega_n}{s + \omega_n} \tag{A.2}$$

where

k_{dc} is the DC gain

τ is the system time constant (time in seconds to reach 63% of steady state)
ω_n is the natural frequency of the first order system

Similarly the standard form for a second-order system takes the form

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2} \tag{A.3}$$

where

ω_0 is the system natural frequency in radians per second
ζ is the system damping ratio

The modeling of a system can also be converted to a system of first-order differential equations also known as state-space modeling. In this case, the first-order system model can be represented as

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{A.4}$$

where $\dot{x}$ is the time derivative of the state, A is the state transition matrix with all its eigenvalues chosen negative (Hurwitz), B is the input matrix, and u is the input vector.

APPENDIX B:
Fixed Wing Aircraft Dynamics Model

B.1 Fixed Wing Aircraft Dynamics Model

The following is the nomenclature used to describe the fixed-wing kinematic equations. Euler angles for pitch (θ), roll (ϕ), and yaw (ψ) have the units of radians. Figure B.1 illustrates the north-east-down (NED) reference frame definitions used for body rotational rates about the x axis (p), y axis (q), and the z axis (r), as well as the body velocities in the x axis (u), y axis (v), and the z axis (w).

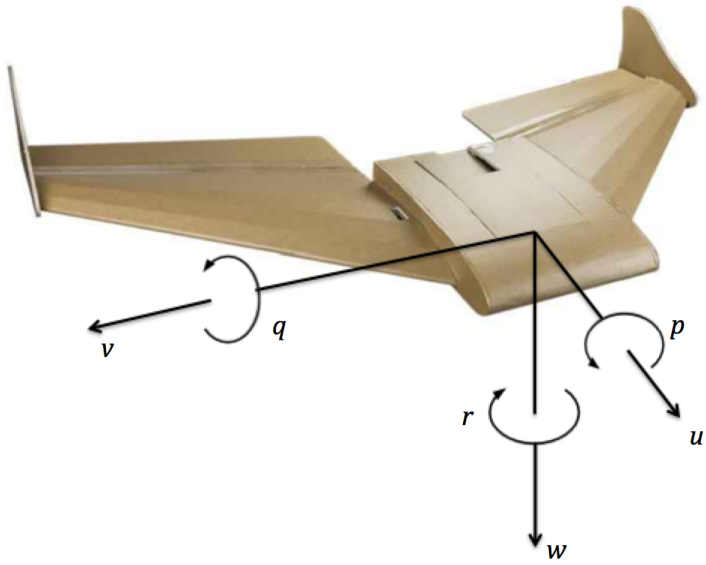

Figure B.1. Reference Frame - Body Rates and Velocities. Adapted from [16].

Newton's second law as it pertains to rotational motion can be stated as

$$\tau = J\frac{d\omega}{dt_i} \tag{B.1}$$

where τ is the torques applied to the body, J is the moment of inertia, and $\frac{d\omega}{dt_i}$ is the angular

acceleration of the body with respect to the inertial frame.

Equation B.1 can be rewritten in the body reference frame as follows:

$$\tau^b = J\dot{\omega}^b_{b/i} + \omega^b_{b/i} \times \left(J\omega^b_{b/i}\right) \tag{B.2}$$

The expression $\dot{\omega}^b_{b/i}$ is the angular acceleration in the body frame as viewed in the body frame

$$\dot{\omega}^b_{b/i} = \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} \tag{B.3}$$

The equation can then be rewritten with respect to $\dot{\omega}^b_{b/i}$ in Equation B.3

$$\dot{\omega}^b_{b/i} = J^{-1}\left[-\omega^b_{b/i} \times \left(J\omega^b_{b/i}\right) + \tau^b\right] \tag{B.4}$$

where J is the inertia matrix as follows:

$$J = \begin{pmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{pmatrix} \tag{B.5}$$

The moments of inertia, or the diagonal terms, are always non-zero for any rigid body. The products of inertia, or the off-diagonal terms, are terms which describe the inertial coupling between axis. For a traditional fixed wing aircraft, the natural symmetry will simplify the inertia matrix in the off-diagonal terms as follows:

$$J = \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix} \tag{B.6}$$

The inverse of J can be found to be

$$J^{-1} = \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \tag{B.7}$$

where

$$\Gamma = J_x J_z - J_{xz}^2 \tag{B.8}$$

Aircraft nomenclature for torques are defined $\tau \triangleq (l, m, n)^T$, and therefore the combined equations derived from first principles take the form

$$\begin{aligned} \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} + \begin{pmatrix} l \\ m \\ n \end{pmatrix} \right] \\ &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \left[\begin{pmatrix} J_{xz}pq + (J_y - J_z)qr \\ J_{xz}(r^2 - p^2) + (J_z - J_x)pr \\ (J_x - J_y)pq - J_{xz}qr \end{pmatrix} + \begin{pmatrix} l \\ m \\ n \end{pmatrix} \right] \\ &= \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr + \Gamma_3 l + \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{1}{J_y} m \\ \Gamma_7 pq - \Gamma_1 qr + \Gamma_4 l + \Gamma_8 n \end{pmatrix} \end{aligned} \tag{B.9}$$

where

$$
\begin{aligned}
\Gamma_1 &= \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} \\
\Gamma_2 &= \frac{J_z(J_z - J_y) + J_{xz}^2}{\Gamma} \\
\Gamma_3 &= \frac{J_z}{\Gamma} \\
\Gamma_4 &= \frac{J_{xz}}{\Gamma} \\
\Gamma_5 &= \frac{J_z - J_x}{J_y} \\
\Gamma_6 &= \frac{J_{xz}}{J_y} \\
\Gamma_7 &= \frac{J_x(J_x - J_y) + J_{xz}^2}{\Gamma} \\
\Gamma_8 &= \frac{J_x}{\Gamma}
\end{aligned}
\tag{B.10}
$$

The aerodynamic torques (excluding propulsive torques) can be found to be

$$
\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} b\left[C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2Va}p + C_{l_r}\frac{b}{2V_a}r + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r\right] \\ c\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q + C_{m_{\delta_e}}\delta_e\right] \\ b\left[C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2Va}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r\right] \end{pmatrix} \tag{B.11}
$$

where $C_{l_0}, C_{l_\beta}, C_{l_p}, C_{l_r}, C_{l_{\delta_a}}, C_{l_{\delta_r}}, C_{n_0}, C_{n_\beta}, C_{n_p}, C_{n_r}, C_{n_{\delta_a}}, C_{n_{\delta_r}}$ are the lateral aerodynamic coefficients

and $C_{m_0}, C_{m_\alpha}, C_{m_q}, C_{m_{\delta_e}}$ are the longitudinal aerodynamic coefficients.

Substituting the aerodynamic torques found in Equation B.11 into Equation B.9 results

in [17]

$$\begin{aligned}
\dot{p} &= \Gamma_1 pq - \Gamma_2 qr + \frac{1}{2}\rho V_a^2 Sb\left[C_{p_0} + C_{p_\beta}\beta + C_{p_p}\frac{bp}{2V_a} + C_{p_r}\frac{br}{2V_a} + C_{p_{\delta_a}}\delta_a + C_{p_{\delta_r}}\delta_r\right] \\
\dot{q} &= \Gamma_5 pr - \Gamma_6(p^2 - r^2) + \frac{1}{2}\rho V_a^2 Sc\frac{1}{J_y}\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{cq}{2V_a} + C_{m_{\delta_e}}\delta_e\right] \\
\dot{r} &= \Gamma_7 pq - \Gamma_1 qr + \frac{1}{2}\rho V_a^2 Sb\left[C_{r_0} + C_{r_\beta}\beta + C_{r_p}\frac{bp}{2V_a} + C_{r_r}\frac{br}{2V_a} + C_{r_{\delta_a}}\delta_a + C_{r_{\delta_r}}\delta_r\right]
\end{aligned} \tag{B.12}$$

Simplifying Equation B.12 assuming no inertial or aerodynamic coupling results in

$$\begin{aligned}
\dot{p} &= \frac{1}{2}\rho V_a^2 Sb\left[C_{p_{\delta_a}}\delta_a + C_{p_p}\frac{bp}{2V_a} + C_{p_0}\right] \\
\dot{q} &= \frac{1}{2}\rho V_a^2 Sc\frac{1}{J_y}\left[C_{m_{\delta_e}}\delta_e + C_{m_q}\frac{cq}{2V_a} + C_{m_0}\right] \\
\dot{r} &= \frac{1}{2}\rho V_a^2 Sb\left[C_{r_{\delta_r}}\delta_r + C_{r_r}\frac{br}{2V_a} + C_{r_0}\right]
\end{aligned} \tag{B.13}$$

The equations in B.13 are then slightly modified to fit the first-order ordinary differential equation (ODE) model as described in Equation A.4

$$\begin{aligned}
\dot{p} &= A_p\hat{p} + b_p\left(\hat{\omega}_p\delta_a + \hat{\theta}_p p + \hat{\sigma}_p\right) \\
\dot{q} &= A_q\hat{q} + b_q\left(\hat{\omega}_q\delta_e + \hat{\theta}_q q + \hat{\sigma}_q\right) \\
\dot{r} &= A_r\hat{r} + b_r\left(\hat{\omega}_r\delta_r + \hat{\theta}_r r + \hat{\sigma}_r\right)
\end{aligned} \tag{B.14}$$

where ω is the input gain coefficient, θ is the constant state coefficient, and σ is the disturbance estimate.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C:
System Identification

C.1 System Identification

System identification was performed to scope the author's expectation for the bandwidth of small fixed wing unmanned airframes. This was conducted in order to properly establish cutoff frequencies for the companion model. These results are also mirrored in Figure 6.1 and discussed in Chapter 6.

C.1.1 Data Collection

The Pixhawk autopilot was used to capture roll and pitch rates ($\dot{p}, \dot{q}$) for the test vehicle as well as the pilot's command inputs. These outputs and inputs were the essential building blocks for creating pitch rate and roll rate models for the test vehicle. The autopilot is capable of logging data at 50-400 Hz that is represented as a discrete time domain signal. This data should ultimately be manipulated into the s-domain. The mathematics for this procedure are well defined, and numerous tools can be used to simplify this process [18].

It is crucial to ensure there is sufficient frequency content in the data recorded. Exciting multiple frequencies in the time domain ensures the regression techniques have an adequate persistence of excitation to resolve polynomial coefficients with higher certainty.

To ensure sufficient frequency content was obtained from the aircraft, a linear chirp was chosen and implemented into the Pixhawk source code as follows:

$$x(t) = sin\left[\phi_0 + 2\pi\left(f_0 t + \frac{k}{2}t^2\right)\right] \tag{C.1}$$

where

ϕ_0 is the initial phase of the chirp at t=0 (nominally zero degrees)
f_0 is the initial frequency at t=0

k is the chirp rate

t is time in seconds

An example of this method can be seen in Figure C.1.

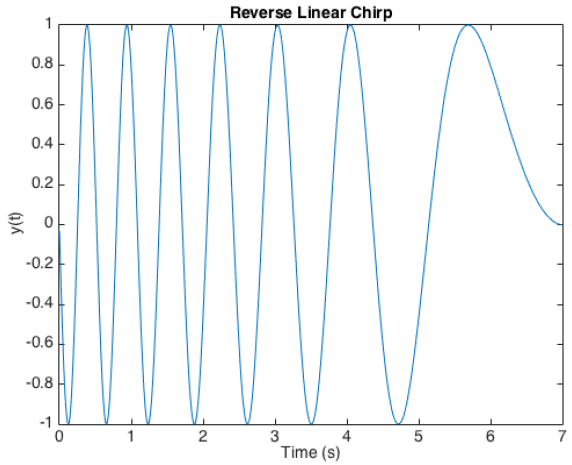

Figure C.1. Reverse Linear Chirp Example

Specifically for this research, the reverse formulation of the linear chirp was used because it ensured the aircraft wouldn't exceed max roll or pitch limits within the first few cycles of the output. The reverse chirp was applied in open loop while in "training mode" which would constrain the angle of bank or pitch by switching into attitude hold mode upon an attitude exceedance.

C.1.2 z-Domain to s-Domain

The logged input and output data in discrete form requires shaping to convert cleanly into an s-domain representation. The first step is ensuring that the data is of constant sampling rate. In other words, the time between samples is uniform from sample to sample. The data provided from the Pixhawk autopilot does not have a uniform sampling rate. The sample rate is a user-defined rate (50-400Hz) but has a slight amount of jitter ($\pm 0.1\%$). piecewise cubic hermite interpolating polynomial (PCHIP) interpolation was used to interpolate the data into a uniform sampling rate.

After the data is shaped correctly with a uniform sample rate, the discrete data is transformed into a continuous approximation using a zero order hold (ZOH) technique. Taking the

Laplace transform of the continuous input/output data will convert it into the s-domain, and finally, a non-linear least squares minimization algorithm can be run to find the polynomial coefficients which best fit the data.

The order of the regression (number of polynomials to estimate) is at the discretion of the engineer and their intuition of system's physical representation. Higher order models will better fit the data, but in most cases, they tend to overfit the data; therefore some tradeoffs should be considered to simplify the model. Most aircraft models reasonably limit the system to an LTI and second-order. These fundamental aerodynamics models divide the modeling into longitudinal and lateral dynamics. Each axis of the aircraft is assigned two-second order responses. Pitch, for example, has a second order response in the pitch damping mode (also known as the short period) and also has a second order response in the transition of kinetic energy to potential energy (also known as the long-period or phugoid). Both first order and second order models were estimated for comparison sake.

Results were collected from two flight test events. The first flight test was conducted under manual control without utilizing the chirp. The pilot attempted to increase frequency of the input signal manually. The second set of data collected was via the reverse linear chirp method previously described.

The manually piloted acquired data was expected to have insufficient frequency content in the signal. However, the manual flight test provided moderate results for modeling the aircraft as seen in Figure C.2 .

The results in Figure C.2 demonstrates the utility of this technique even if data can only be acquired from manual pilot inputs. It can be seen that the second order model starts to misrepresent the data at higher frequencies. Figure C.3 illustrates the data recorded from the reverse chirp experiment run at 50Hz. The δ pulse width modulation (PWM) input channel data clearly does not represent the software calculated chirp. The highest frequency designed to be outputted during this experiment was 10Hz. 10 Hz is significantly less than the frequency required to meet Nyquist sampling criterion (25Hz) for this data sampling rate. However, there are clear patterns of aliasing in the input signal as recorded by the data flash logger.

The chirp response was physically observed on pre-flight, in actual flight, and in the data-

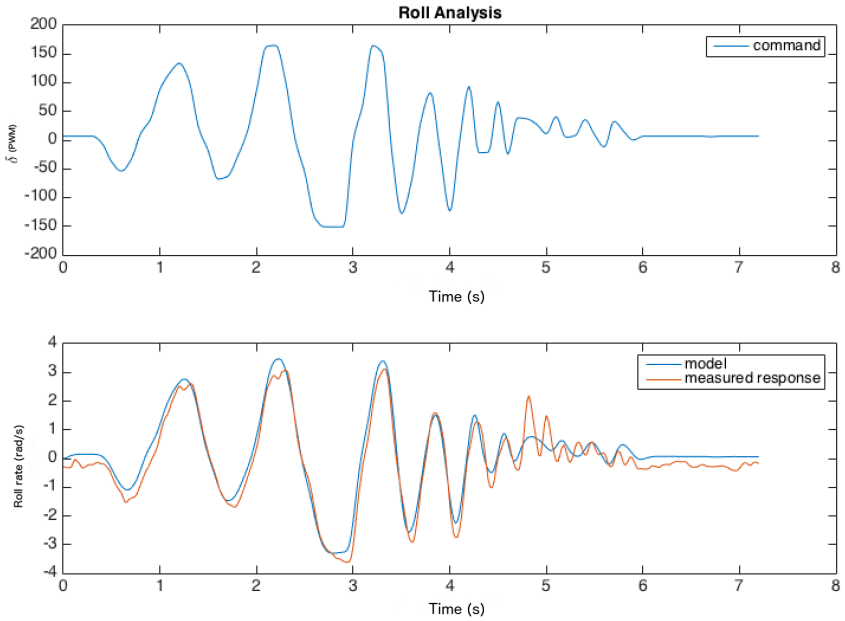

Figure C.2. Roll Model Regression with Manual Inputs

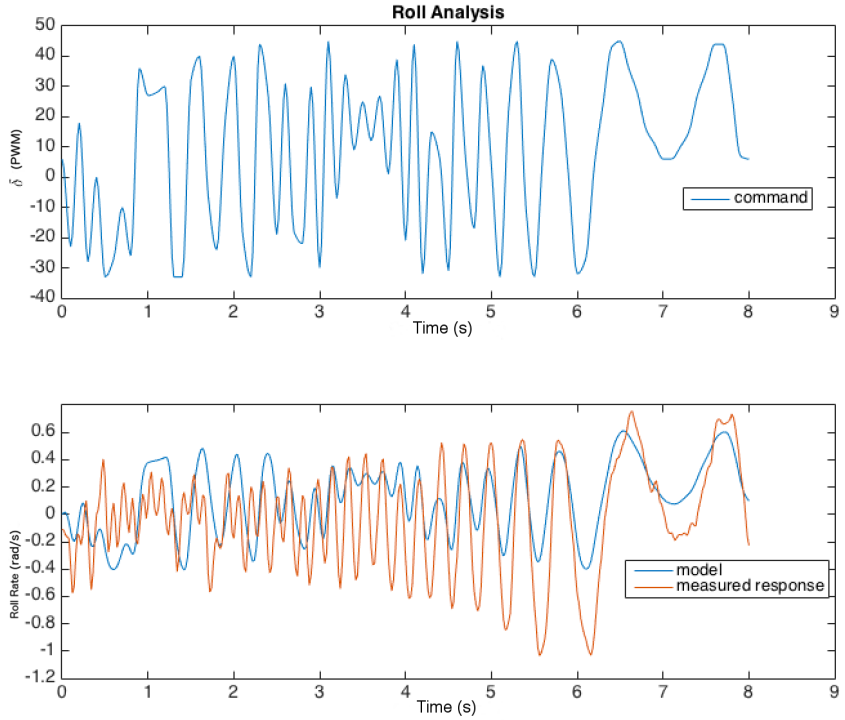

Figure C.3. Roll Model Regression with Reverse Linear Chirp

flash logged body rate of the aircraft. However, the aliased input channel was arbitrarily biasing the regression result. The significant aliasing in the logged input was not due to insufficient sampling rate as previously discussed. After inspection, the peculiar aliasing issue was hardware specific to the Pixhawk 1 autopilot in how the main CPU sends servo commands to the auxiliary I/O CPU. The most recent version of firmware, at the time of this test, improperly logs the PWM through an aliased prone signal path. The main and I/O CPU both run at 50Hz with some appreciable clock drift. This generates a noticeable beat frequency and delay when the actual values in registry are saved for PWM values are sent back round trip to the main CPU. The implication of logging the PWM values at the very end of the digital transmission line seems valuable in principle because the values being logged are the undeniable values being sent to the actuators. However, the cost of logging these values in this manner on the Pixhawk architecture incurs significant aliasing at almost all frequencies. Logging the commanded PWM values before being sent to I/O CPU solved the aliasing discrepancy and produced very frequency rich models.

The manually piloted acquired data provided viable data source for the models even though it is a very simplistic approach. There were two separate manual tests run on the same aircraft on the same flight, and the following are the results using this regression technique to model a second-order system:

$$H(s) = \frac{10.39}{s^2 + 31.26s + 504.9} \tag{C.2}$$

and

$$H(s) = \frac{10.61}{s^2 + 29.77s + 498.7} \tag{C.3}$$

Converting to standard form as described in Equation A.3 yields

$$H(s) = \frac{0.0206 * 22.47^2}{s^2 + 2 * 0.69 * 22.47s + 22.47^2} \tag{C.4}$$

and

$$H(s) = \frac{0.0213 * 22.33^2}{s^2 + 2 * 0.67 * 22.33s + 22.33^2} \tag{C.5}$$

It is important to note that this system identification technique run on separate sets of data has produced two models with very similar values for ω_n and ζ.

This produces the average values of

$\omega_n = 22.4 rad/s$
$k = 0.0209$
$\zeta = 0.681$

With the aliasing removed from the chirped input command signals as previously described, the model is drastically improved and produces the following results:

$$H(s) = \frac{4.409}{s^2 + 27.11s + 430.6} \tag{C.6}$$

and

$$H(s) = \frac{3.295}{s^2 + 18.82s + 296.5} \tag{C.7}$$

Converting to standard form as described in Equation A.3 results in

$$H(s) = \frac{0.0102 * 20.75^2}{s^2 + 2 * 0.65 * 20.75s + 20.75^2} \tag{C.8}$$

and

$$H(s) = \frac{0.0111 * 17.21^2}{s^2 + 2 * 0.54 * 17.21s + 17.21^2} \tag{C.9}$$

The comparison of the model to recorded results is seen in Figure C.4.

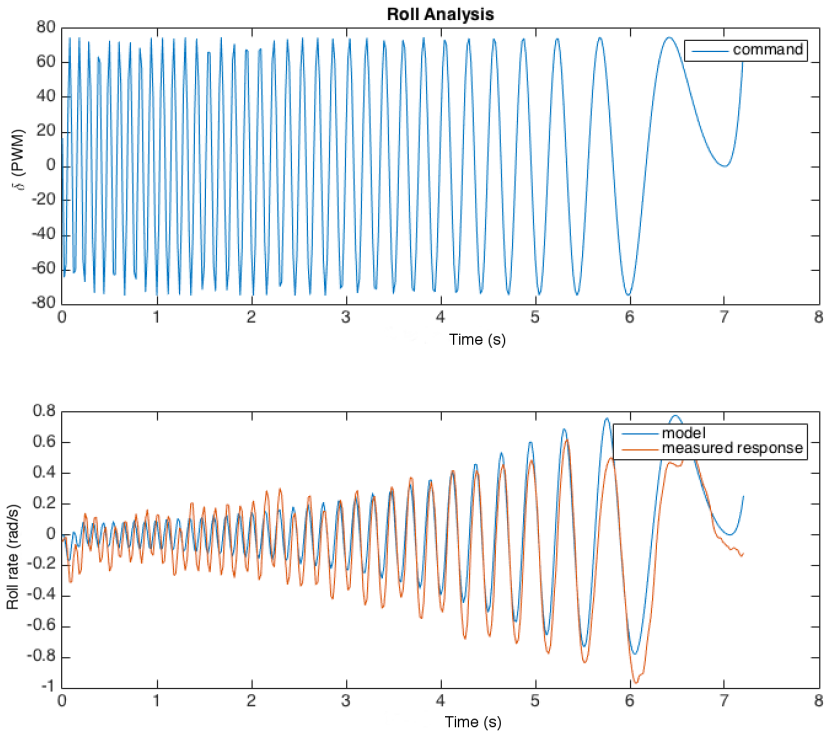

Figure C.4. Non-Aliased Reverse Chirp Model Example

This produces the average values of

$\omega_n = 18.98 rad/s$
$k = 0.010$
$\zeta = 0.598$

In the author's experience, these values are reasonable values for this size and weight of airframe based on multiple system identification experiments in SITL across multiple airframes as well as actual test flights of three different airframes. This regression technique has shown potential to create realistic models from actual flight test data. The data must be properly shaped. The reverse chirp method has the potential to increase the fidelity of the high-frequency response of the aircraft if the aliasing issue can be resolved on the command input channel.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D:
Lyapunov Stability Definition

Aerospace designs tend to use linear controllers as reference models for their simplicity and well-understood robustness characteristics. This is despite the fact that the applications of these linear controllers are applied to a non-linear dynamical system such as attitude control with varying dynamic pressure. Adaptive Controllers are non-linear and may offer performance benefits to non-linear systems as seen in aforementioned aerospace applications. However, non-linear controller's stability properties need to be evaluated. The Lyapunov stability criteria offer methods of evaluating these controller's boundedness and robustness behavior.

Aleksandr Lyapunov was a Russian mathematician who's work was published in 1892 [19] concerning the behavior of non-linear systems close to equilibrium without having to rigorously find the unique solutions to difficult differential equations used to model the system. His work was largely overlooked until the Cold War when aerospace solutions required a more rigorous approach to analyzing non-linear control robustness. Modern non-linear control engineers extensively utilize Lyapunov's techniques to design and evaluate non-linear controllers.

D.1 Lyapunov Stability Theory

According to Lyapunov, the stability properties of a system can be classified as stable, asymptotically stable, and exponentially stable.

Given the following autonomous nonlinear system:

$$\dot{x}(t) = f(x(t)), \qquad x(0) = x_0 \tag{D.1}$$

where f has equilibrium at x_e :

$$f(x_e) = 0 \tag{D.2}$$

then the equilibrium is said to be:

1. **Lyapunov Stable**
   for every $\epsilon > 0$ there exists a $\delta > 0$ such that, if $\|x(0) - x_e\| < \delta$, then for every $t \geq 0$ we have $\|x(t) - x_e\| < \epsilon$
2. **Asymptotically Stable**
   if the system is Lyapunov stable and there exists a $\delta > 0$ such that if $\|x(0) - x_e\| < \delta$, then $\lim_{t\to\infty} \|x(t) - x_e\| = 0$
3. **Exponentially Stable**
   if the system is asymptotically stable and there exists $\alpha > 0, \beta > 0, \delta > 0$ such that if $\|x(0) - x_e\| < \delta$, then $\|x(t) - x_e\| \leq \alpha \|x(0) - x_e\| \, e^{-\beta t}$, for all $t \geq 0$

Being Lyapunov stable infers that if a system is near equilibrium, then it will indefinitely remain near equilibrium. If the system if found to be asymptotically stable then it eventually will achieve equilibrium as $t \to \infty$ and being exponentially stable implies it reaches equilibrium with a rate of convergence β.

Lyapunov's Second Method

Lyapunov's second method is also known as Lyapunov stability criteria. This method offers a less tenuous method for evaluating mathematically non-ideal systems. Lyapunov analysis of the linearized system around equilibrium can be cumbersome when the eigenvalues are purely imaginary. In this case, the solutions can rapidly depart to infinity or approach zero with little perturbation to the eigenvalues. Lyapunov's second method offers an alternative approach for classifying a system's stability using a concept that is similar to how total energy is defined in classical mechanics.

Conceptually, Lyapunov's second method can be compared to the evaluation of mechanical system similar to the modeling of energy in a vibrating spring mass system. The energy of the unforced spring mass system will dissipate energy due to friction and or damping. This trend of energy leaving the system towards some "attractor" is evidence of the system's stability characteristics and identifies that there will be some stable end state. Likewise, Lyapunov's second method specifies this with the use of a Lyapunov candidate function $V(x)$ which implicitly characterizes the total energy of the system. It is important to note that Lyapunov realized that the candidate function could be any positive definite and radially

unbounded function. It is then said to be Lyapunov stable if any candidate function is found and meets the stability criteria. This means that it is only incumbent upon the engineer to find one candidate function to meet the criteria. This can be an iterative process of trying various energy like equations. A common approach is to model the Lyapunov candidate equation as kinetic energy ($\frac{1}{2}u^2$). Lyapunov realized that characterizing the energy of a nonlinear system could be almost impossible for some cases, but this approach could prove stability without the rigorous knowledge of the true system's energy.

Lyapunov's second method defines a system as Lyapunov Stable for a system $\dot{x} = f(x)$ having an equilibrium point at $x = 0$ where the Lyapunov candidate function $V(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ such that:

- $V(x) = 0$ if and only if $x = 0$
- $V(x) > 0$ if and only if $x \neq 0$
- $\dot{V}(x) = \frac{d}{dt}V(x) = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i} f_i(x) \leq 0$, for all values of $x \neq 0$

if $\dot{V}(x) < 0$ for $x \neq 0$ then system is asymptotically stable.

Additionally, it is required to demonstrate the condition of radial unboundedness to ensure the system is globally stable [20].

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E:
Projection Operator

E.1 Derivation of Projection Operator $Proj(\theta, y)$

In this section, the projection operator will be defined. Adaptive controllers often use the projection operator in their adaptive laws to ensure uniform boundedness of the system error. This can aid in faster adaptation and ensure controller closed-loop stability. The projection operator attempts to mathematically achieve two objectives; ensure the Lyapunov function time derivative remains negative semi-definite, and to keep the estimated parameters uniformly bounded. Using the projection operator ensures that θ is locally Lipchitz continuous even though the input y is piecewise continuous. The projection operator as utilized in this research is defined as

$$Proj(\theta, y) \triangleq \begin{cases} y & \text{if } f(\theta) > 0, \\ y & \text{if } f(\theta) \leq 0 \text{ and } \nabla f^\top y \geq 0, \\ y - \frac{\nabla f}{\|\nabla f\|} \left\langle \frac{\nabla f}{\|\nabla f\|}, y \right\rangle f(\theta) & \text{if } f(\theta) \leq 0 \text{ and } \nabla f^\top y < 0. \end{cases} \tag{E.1}$$

where $\epsilon > 0$

$$f(\theta) = -\frac{\theta^2 - \theta_{\max}^2}{\epsilon \theta_{\max}^2} \tag{E.2}$$

$$\nabla f^\top = -\frac{2\theta}{\epsilon \theta_{\max}^2} \tag{E.3}$$

The projection function chosen for this research is parabolic and has inflection points at user defined maximum/minimum bands. Equation E.2 is plotted in Figure E.1 with example maximum/minimum of 0.65 and various values for ϵ. The engineer must set the value for ϵ to achieve the desired slope of the projection at the maximum values. This slope should be steep enough to capture the highest expected error given one recursion through the algorithm.

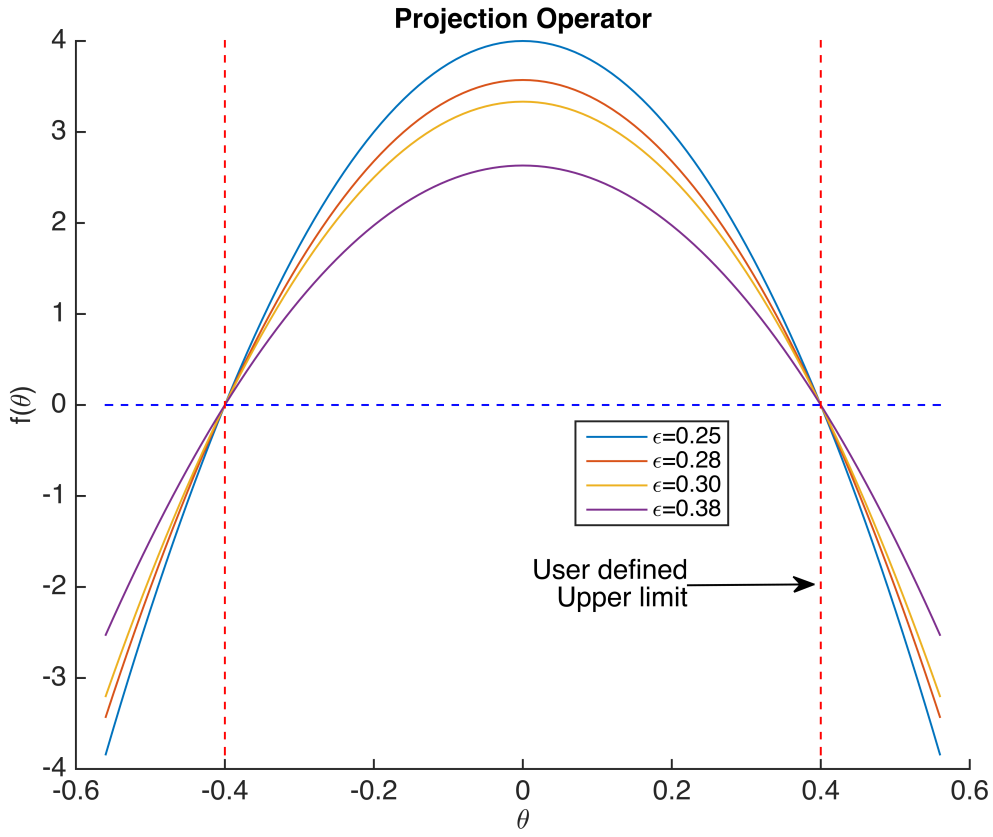

Figure E.1. Projection Operator - $f(\theta)$

There may exists systems which need to have projection bounds which are not symmetrical about zero. This was found to be the case for this research and therefore the following projection function was used to assist the engineer tuning the algorithm:

where $\epsilon > 0$

$$f(\theta) = -\frac{4(\theta_{\min} - \theta)(\theta_{\max} - \theta)}{\epsilon(\theta_{\max} - \theta_{\min})^2} \tag{E.4}$$

$$\nabla f^{\top} = \frac{4(\theta_{\min} + \theta_{\max} - 2\theta)}{\epsilon(\theta_{\max} - \theta_{\min})^2} \tag{E.5}$$

Equation E.4 is plotted in Figure E.2 with example maximum of 0.65, minimum of 0.25, and various values for ϵ.

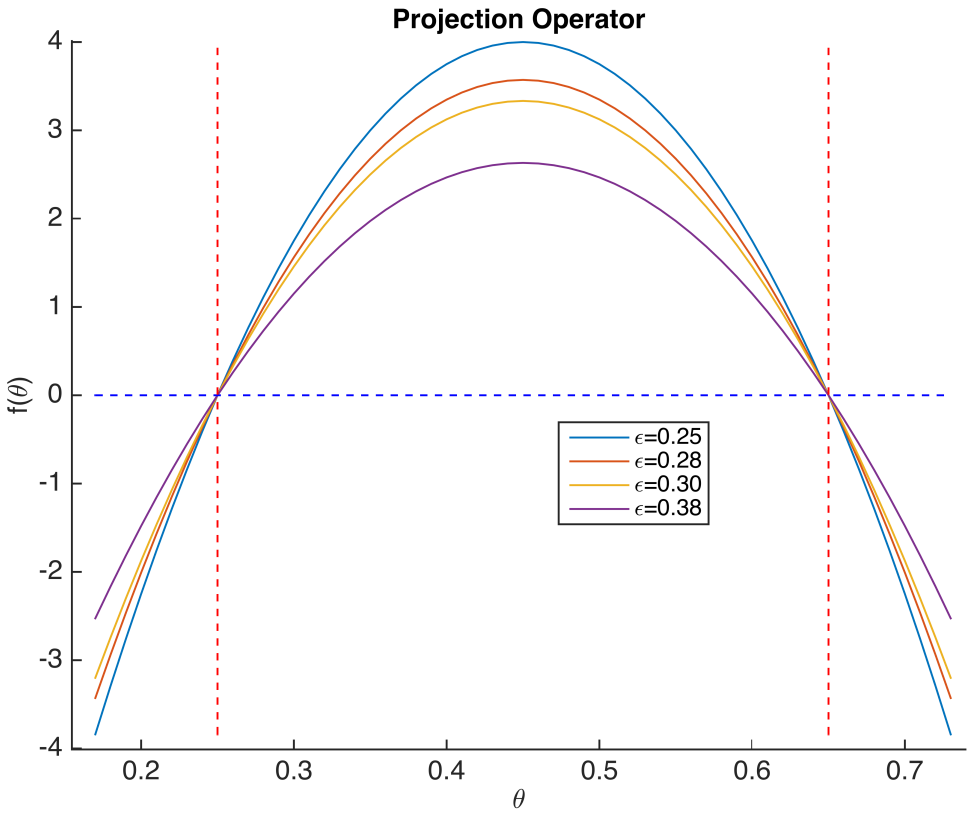

Figure E.2. Projection Operator with Offset Limits

E.2 C++ Implementation

```
float projection_operator(float theta,float y,float epsilon,
float theta_max,float theta_min)
{

        // Calculate convex function
        float f_theta = (-4*(theta_min-theta)*(theta_max-theta))
        /(epsilon*(theta_max-theta_min)^2);
        float f_theta_dot = (4*(theta_min+theta_max-(2*theta)))
        /(epsilon*(theta_max-theta_min)^2);

        float projection_out = y;

        if (f_theta <=0 && f_theta_dot*y < 0)
        {
```

```
                projection_out = y*(f_theta+1); // y-(y*-f_theta);
        }

        return projection_out;
}
```

APPENDIX F:
Matlab Code

F.1 Euler vs Trapezoidal Method

```
clear, format long, clc, close all
dt = 1;

t0 = [0:.01:4];

y0 = exp(t0);

%Euler integration

t1 = [0:4];
y1 = exp(0);
for i=1:4
    y1(i+1)=y1(i) + dt*exp(i-1);
end

%Trapezoidal integration
t2 = [0:4];
y2 = exp(0);

for i=1:4
    y2(i+1)=y2(i) + dt*exp(i-1);
    y2(i+1)=y2(i) + (dt/2)*(exp(i-1)+y2(i+1));
end

plot(t0,y0,'k--',t1,y1,'b',t2,y2,'r')
legend('y=e^t','Euler Method','Trapezoidal Method')
```

F.2 SISO Lyapunov Solution Proof

```
clear,clc, format compact, close all
%% Find Pb
wn = [5:0.1:10]; %rad/s
```

```
for i=1:length(wn)
    b = [wn(i)];
    a = [1,wn(i)];
    [A,B,C,D] = tf2ss(b,a)

    Pb(i) = lyap(A,1);
end
Pb_test = 1./(2*wn);

plot(wn,Pb_test,'o',wn,Pb)

%in this simple 1x1 matrix case Pb is easy to calc by hand in code
% Pb ends up being: Pb=1/2*wn;
```

F.3 Reverse Linear Chirp

```
clear, format compact, clc, close all

fs = 1/200;
t = [0:fs:7];
f0= 0.01;%Hz
f1= 10; %Hz
k = (f1-f0)/(t(end));

phi_0 = 0.0;

sample_delay = 0.02;

out = sin(phi_0+2*pi*(f0*(7-(t+sample_delay))+(k/2).*(7-(t+sample_delay)
    ).^2));
%out = 1500 + out*10;

out = out*10;

plot(t,out)
```

F.4 Projection Operator Example Plots

```
clear, clc, format compact, close all
```

```
2
3 %% Plot Projection
4 epsilon = [0.25,0.28,0.3,0.38];
5 theta_max = 0.65;
6 theta_min = 0.25;
7 center = (theta_max+theta_min)/2;
8
9 span = (theta_max-theta_min)*1.4;
10 theta = [center-(span/2):0.01:center+(span/2)];
11
12 for j=1:length(epsilon)
13     for i=1:length(theta)
14         %f_theta(i,j) = -(theta(i).^2-theta_max^2)/(epsilon(j)*theta_max
               ^2);
15         %f_theta_dot(i,j) = -(2*theta(i))/(epsilon(j)*theta_max^2);
16         f_theta(i,j) = -(theta_min-theta(i))*(theta_max-theta(i))./(
               epsilon(j));
17         f_theta_dot(i,j) = (theta_min+theta_max-(2*theta(i)))/(epsilon(j
               ));
18     end
19 end
20
21 figure
22 for j=1:length(epsilon)
23 hold on
24 plot(theta,f_theta(:,j))
25 end
26 line([min(theta),max(theta)],[0,0],'color','blue','linestyle','--')
27 line([theta_min,theta_min],[min(f_theta(:,1)),max(f_theta(:,1))],'color'
    ,'red','linestyle','--')
28 line([theta_max,theta_max],[min(f_theta(:,1)),max(f_theta(:,1))],'color'
    ,'red','linestyle','--')
29 legend('\epsilon=0.25','\epsilon=0.28','\epsilon=0.30','\epsilon=0.38')
30 ylabel('f(\theta)')
31 xlabel('\theta')
32 title('Projection Operator')
33 hold off
```

F.5 System Identification

```
clear, clc, format compact, close all

% Import data file for analysis
[filename, pathname] = uigetfile('*.m','Choose first MATLAB file');
run(filename)
%%
% Clean tigger PWMs
for r=1:length(RCIN.data(:,9))
    if RCIN.data(r,8) < 1700 %channel 6
        RCIN.data(r,8) = 1000;
    end
    if RCIN.data(r,9) < 1700 %channel 7
        RCIN.data(r,9) = 1000;
    end
end

%Trigger Channel Data
roll_trigger_data = RCIN.data(:,9);
pitch_trigger_data = RCIN.data(:,8);
trigger_data_time = RCIN.data(:,2);
%Command Data
roll_command_data = ((RCOU.data(:,3)-1500)+(RCOU.data(:,4)-1500))+1526;
pitch_command_data = ((RCOU.data(:,3)-1500)-(RCOU.data(:,4)-1500))+1460;
command_data_time = RCOU.data(:,2);
%IMU Data
p = IMU2.data(:,3);
q = -IMU2.data(:,4);
IMU_time = IMU2.data(:,2);

% Plot to show sectioned data
plot(RCIN.data(:,2)*10^-6,RCIN.data(:,8)-1500,IMU2.data(:,2)*10^-6,p
    *(180/pi),...
      RCIN.data(:,2)*10^-6,RCIN.data(:,9)-1500,IMU2.data(:,2)*10^-6,q
          *(180/pi))
legend('ch6','p','ch7','q')
xlabel('time (seconds)')
ylabel('pwm counts')

[start_index, stop_index] = find_trigger(roll_trigger_data, 1700);
fprintf('Which test would you like to analyze? [Pick a number between 1
```

```
    and %i]\n',length(start_index));
num_of_test_desired = input('');

%[ input, output, time ] = section_triggered_data( input_data,
    input_data_time, output_data, output_data_time, trigger_data,
    trigger_time, num_of_test_desired )
[ input, output, time ] = section_triggered_data( roll_command_data,
    command_data_time, p, IMU_time, roll_trigger_data, trigger_data_time
    , num_of_test_desired );

%Center controls for SysID and plotting
input = input-1500; %center stick ouputs 1500

% Theoretical Chirp
fs = 1/50;
t = [0:fs:7];
f0= 0.01;%Hz
f1= 10; %Hz
k = (f1-f0)/(t(end));
phi_0 = 0;

input_theoretical = sin(phi_0+2*pi*(f0*(7-t)+(k/2).*(7-t).^2));
input_theoretical = input_theoretical*75;

figure
plot(time,input,'o', time, output*(180/pi),t,input_theoretical)
legend('roll cmd', 'roll rate')

%% System Identification
loop_rate = mean(diff(IMU_time))*10^-6;
time_clean = [0:loop_rate:(length(time)-1)*loop_rate]';
output_clean = interp1(time,output,time_clean,'pchip');
%input_clean = interp1(time,input,time_clean,'pchip');
input_clean = interp1(t,input_theoretical,time_clean,'pchip');


figure
plot(time, input_clean, time, output_clean*(180/pi))
legend('roll cmd', 'roll rate')

```

```
74 [y,t,x] = estimate_model(input_clean, output_clean, time_clean);
75
76 figure
77 subplot(2,1,1)
78 plot(time_clean,input_clean)
79 legend('command')
80 xlabel('time (seconds)')
81 ylabel('\delta (pwm)')
82 title('Roll Analysis')
83 subplot(2,1,2)
84 plot(t,y,time_clean,output_clean)
85 legend('model','measured response')
86 xlabel('Time (s)')
87 ylabel('Roll rate (rad/s)')
```

APPENDIX G:
$\mathcal{L}_1$ Adaptive Controller Source Code

G.1 $\mathcal{L}_1$ Adaptive Control Source Code

```
/*
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
/*
  Adaptive controller by Ryan Beall
  See
  http://naira-hovakimyan.mechse.illinois.edu/l1-adaptive-control-
      tutorials/
  for mathematical basis
 */

#include <AP_HAL/AP_HAL.h>
#include <GCS_MAVLink/GCS.h>
#include "ADAP_Control.h"
#include <stdio.h>

extern const AP_HAL::HAL& hal;

const AP_Param::GroupInfo ADAP_Control::var_info[] = {
    // adaptive control parameters
    AP_GROUPINFO_FLAGS("CH", 1, ADAP_Control, enable_chan, 0,
        AP_PARAM_FLAG_ENABLE),
    AP_GROUPINFO("AL",       2, ADAP_Control, alpha, 24),
```

```
31      AP_GROUPINFO("GAMT",      3, ADAP_Control, gamma_theta, 1000),
32      AP_GROUPINFO("GAMW",      4, ADAP_Control, gamma_omega, 1000),
33      AP_GROUPINFO("GAMS",      5, ADAP_Control, gamma_sigma, 1000),
34      AP_GROUPINFO("THU",       6, ADAP_Control, theta_max, 2.0),
35      AP_GROUPINFO("THL",       7, ADAP_Control, theta_min, 0.5),
36      AP_GROUPINFO("THE",       8, ADAP_Control, theta_epsilon, 5925),
37      AP_GROUPINFO("OMU",       9, ADAP_Control, omega_max, 2.0),
38      AP_GROUPINFO("OML",      10, ADAP_Control, omega_min, 0.5),
39      AP_GROUPINFO("OME",      11, ADAP_Control, omega_epsilon, 5925),
40      AP_GROUPINFO("SIGU",     12, ADAP_Control, sigma_max, 0.1),
41      AP_GROUPINFO("SIGL",     13, ADAP_Control, sigma_min, -0.1),
42      AP_GROUPINFO("SIGE",     14, ADAP_Control, sigma_epsilon, 203),
43      AP_GROUPINFO("W0",       15, ADAP_Control, w0, 25),
44      AP_GROUPINFO("K",        16, ADAP_Control, k, 0.45),
45      AP_GROUPINFO("KG",       17, ADAP_Control, kg, 1.0),
46
47      AP_GROUPEND
48  };
49
50  /*
51    return true when enabled
52   */
53  bool ADAP_Control::enabled(void) const
54  {
55      // Enables Adaptive Controller instead of PID if rc PWM value is
            above 1700 milli seconds
56      return (enable_chan > 0 && hal.rcin->read(enable_chan-1) >= 1700);
57  }
58
59
60  /*
61    reset to startup values
62  */
63  void ADAP_Control::reset(uint16_t loop_rate_hz)
64  {
65      x_m = x;
66      u = 0.0;
67      u_lowpass = 0.0;
68      u_sp = 0.0;
69      theta = 1.0;
```

```
70      omega = 1.0;
71      sigma = 0.0;
72      integrator = 0.0;
73
74      float u_cutoff_hz = w0/(2*M_PI); //convert cutoff freq from rad/s to
            hz
75      u_filter.set_cutoff_frequency(loop_rate_hz, u_cutoff_hz);
76      u_filter.reset();
77
78      float r_cutoff_hz = (alpha-2)/(2*M_PI); //convert cutoff freq from
            rad/s to hz
79      r_filter.set_cutoff_frequency(loop_rate_hz, r_cutoff_hz);
80      r_filter.reset();
81  }
82
83
84  /*
85    trapezoidal integration helper function
86   */
87  float ADAP_Control::trapezoidal_integration(float y0, float y1_dot,
        float dt, float &y0_dot)
88  {
89      float y1 = y0 + (dt/2)*(y0_dot+y1_dot);
90      y0_dot = y1_dot;
91
92      return y1;
93  }
94
95  /*
96    adaptive control update. Given a target rate in radians/second and a
97    current sensor rate on the same axis in radians/second, return an
98    actuator value from -1 to 1
99   */
100 float ADAP_Control::update(uint16_t loop_rate_hz, float target_rate,
        float sensor_rate, float scaler, float aspeed)
101 {
102     float dt;
103     const float u_limit = radians(45);
104
105     x = sensor_rate;
```

```
106     r = target_rate;
107     r = r_filter.apply(r);
108 
109     //reset reference model at initialization
110     uint64_t now = AP_HAL::micros64();
111     if (last_run_us == 0 || now - last_run_us > 200000UL) {
112         reset(loop_rate_hz);
113         last_run_us = now;
114         return 0;
115     }
116 
117     dt = (now - last_run_us) * 1.0e-6;
118     last_run_us = now;
119 
120 
121     // u (controller output to plant)
122     eta = theta*x + omega*u_lowpass + sigma;
123     u_sp = eta; // u to state predictor
124 
125     u = constrain_float(eta-(kg*r),-radians(90)/dt, radians(90)/dt);
126 
127     // Check Controller Saturation from previous time step
128     bool saturated = ((u < 0 && u_lowpass >= 0.99*u_limit) ||
129                       (u > 0 && u_lowpass <= -0.99*u_limit));
130 
131     // kD(s) (cascaded second order low pass + simple integrator)
132     u_lowpass = u_filter.apply(u);
133 
134     if (!saturated) {
135         integrator = trapezoidal_integration(integrator, u_lowpass, dt,
            out1);
136       u_lowpass = constrain_float(-k*integrator,-u_limit,u_limit);
137     }
138 
139     // State Predictor (first order single pole recursive filter)
140     // Reference/Companion Model
141     float alpha_filt = exp(-alpha*dt); //alpha in rad/s
142     alpha_filt = constrain_float(alpha_filt, 0.0, 1.0);
143     float beta_filt = 1-alpha_filt;
144 
```

```
145     x_m = alpha_filt*x_m + beta_filt*(u_sp);
146
147     x_error = x_m - x;
148
149
150     // Constrain error to +-300 deg/s
151     x_error = constrain_float(x_error,-radians(300), radians(300));
152
153     // Calculate solution to Lyapunov 1x1 matrix
154     float Pb = 1/(2*alpha);
155
156     // Projection Operator
157     theta_dot = projection_operator(theta,-gamma_theta*x_error*Pb*x,
            theta_epsilon,theta_max,theta_min);
158     omega_dot = projection_operator(omega,-gamma_omega*x_error*Pb*
            u_lowpass,omega_epsilon,omega_max,omega_min);
159     sigma_dot = projection_operator(sigma,-gamma_sigma*x_error*Pb,
            sigma_epsilon,sigma_max,sigma_min);
160
161     // Parameter Update using Trapezoidal integration
162     if (!saturated) {
163         theta = trapezoidal_integration(theta, theta_dot, dt, theta1);
164         omega = trapezoidal_integration(omega, omega_dot, dt, omega1);
165         sigma = trapezoidal_integration(sigma, sigma_dot, dt, sigma1);
166     }
167
168     theta = constrain_float(theta, theta_min, theta_max);
169     omega = constrain_float(omega, omega_min, omega_max);
170     sigma = constrain_float(sigma, sigma_min, sigma_max);
171
172     // Log Data to flash
173     DataFlash_Class::instance()->Log_Write(log_msg_name, "TimeUS,Dt,
            Atheta,Aomega,Asigma,Aeta,Axm,Ax,Ar,Axerr,AuL", "Qfffffffff",
174                                            now,
175                                            dt,
176                                            theta,
177                                            omega,
178                                            sigma,
179                                            eta,
180                                            degrees(x_m),
```

```
                                                degrees(x),
                                                degrees(r),
                                                degrees(x_error),
                                                degrees(u_lowpass));


    return constrain_float(u_lowpass/u_limit, -1, 1);
}

float ADAP_Control::projection_operator(float Theta, float y, float
    epsilon, float th_max, float th_min) const
{

        // Calculate convex function
        // Nominal un-saturated value is above zero line on a parabolic
            curve
        // Steepness of curve is set by epsilon
        float f_diff2 = (th_max-th_min)*(th_max-th_min);
        float f_theta = (-4*(th_min - Theta) * (th_max - Theta))/(
            epsilon*f_diff2);
        float f_theta_dot = (4*(th_min + th_max - (2*Theta)))/(epsilon*
            f_diff2);

        float projection_out = y;

        if (f_theta <= 0 && f_theta_dot*y < 0)
        {
                projection_out = y*(f_theta+1); //y-(y*(-1*f_theta));
        }

           return projection_out;
}

/*
  send ADAP_TUNING message
*/
void ADAP_Control::adaptive_tuning_send(mavlink_channel_t chan, uint8_t
    axis)
{
        if (!enabled() || !HAVE_PAYLOAD_SPACE(chan, ADAP_TUNING)) {
```

```
216         return;
217     }
218     mavlink_msg_adap_tuning_send(chan, axis,
219                                  r,
220                                  x,
221                                  x_error,
222                                  theta,
223                                  omega,
224                                  sigma,
225                                  theta_dot,
226                                  omega_dot,
227                                  sigma_dot,
228                                  x_m,
229                                  u_lowpass,
230                                  u);
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] DoD (Department of Defense), "The role of autonomy in dod systems," Defense Science Board, Washington, D.C. 20301-3140, Tech. Rep., July 2012.

[2] N. Minorsky, "Directional stability of automatically steered bodies," *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, 1922.

[3] D. R. Jenkins. (2000). Hypersonics before the shuttle: A concise history of the X-15 research airplane. NASA. [Online]. Available: https://spaceflight.nasa.gov/outreach/SignificantIncidents/assets/hypersonics-before-the-shuttle.pdf

[4] NASA Armstrong Fact Sheet: X-15 Hypersonic Research Program. (2015, Aug. 13). NASA. [Online]. Available: https://www.nasa.gov/centers/armstrong/news/FactSheets/FS-052-DFRC.html

[5] Z. T. Dydek, A. M. Annaswamy, and E. Lavretsky, "Adaptive control and the NASA X-15-3 flight revisited," *IEEE Control Systems*, vol. 30, no. 3, pp. 32–48, 2010.

[6] B. Peterson and K. Narendra, "Bounded error adaptive control," *IEEE Transactions on Automatic Control*, vol. 27, no. 6, pp. 1161–1168, dec 1982.

[7] E. Lavretsky and K. A. Wise, *Robust Adaptive Control*, M. J. Grimble and M. A. Johnson, Eds. New York, Dordrecht, Heidelberg, London: Springer, 2013.

[8] K. J. Åström and B. Wittenmark, *Adaptive control*, 2nd ed. Mineola, New York: Dover Publications, INC., 1995.

[9] N. Hovakimyan and C. Cao, *L1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. Philadelphia: Society for Industrial and Applied Mathematics, 2010.

[10] Anderson, B. et al., "Failures of adaptive control theory and their resolution," *Communications in Information & Systems*, vol. 5, no. 1, pp. 1–20, 2005.

[11] ArduPlane, ArduCopter, ArduRover Source. (2017). ArduPilot Development Team. [Online]. Available: https://github.com/ArduPilot/ardupilot

[12] ArduPilot Autopilot Suite. (2016). ArduPilot Development Team. [Online]. Available: http://ardupilot.org/ardupilot/

[13] A. Tridgell. (2017). MAVLink proxy and command line ground station. ArduPilot Development Team. [Online]. Available: https://github.com/ArduPilot/MAVProxy

[14] L. Meier. (2014). Marshalling / communication library for drones. PX4 Development Team. [Online]. Available: https://github.com/mavlink/mavlink/

[15] A. Tridgell. (2017). MAVProxy. ArduPilot Development Team. [Online]. Available: http://ardupilot.github.io/MAVProxy/html/index.html

[16] Articles. (2017). FliteTest. [Online]. Available: https://www.flitetest.com/articles

[17] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton, New Jersey: Princeton university press, 2012.

[18] Transfer function estimation. (2017, May). The Mathworks, Inc. [Online]. Available: https://www.mathworks.com/help/ident/ref/tfest.html

[19] A. M. Lyapunov, "The general problem of motion stability," *Annals of Mathematics Studies*, vol. 17, 1892.

[20] H. K. Khalil, *Noninear Systems*. Upper Saddle River, New Jersey 07458: Prentice Hall, 2002.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California