Adaptive Control of Micro Air Vehicles

Joshua Stephen Matthews
Brigham Young University - Provo

ADAPTIVE CONTROL OF MICRO AIR VEHICLES

by

Joshua Stephen Matthews

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Electrical and Computer Engineering

Brigham Young University

December 2006

ABSTRACT

ADAPTIVE CONTROL OF MICRO AIR VEHICLES

Joshua Stephen Matthews

Electrical and Computer Engineering

Master of Science

Although PID controllers work well on Miniature Air Vehicles (MAVs), they require tuning for each MAV. Also, they quickly lose performance in the presence of actuator failures or changes in the MAV dynamics. Adaptive control algorithms that self tune to each MAV and compensate for changes in the MAV during flight are explored. However, because the autopilots on MAVs are small, many of the adaptive control algorithms like those that employ least squares estimation may take too much code space, memory, and/or computing power. In this thesis we develop several Lyapunov-based model reference adaptive control (MRAC) schemes that are both simple and efficient with the MAV autopilot resources. Most notable are the $\mathcal{L}_1$ controllers that have all the benefits of traditional MRACs but have reduced high frequency content to the actuators. The schemes control both roll and pitch through aileron and elevator commands. Flight test results for the schemes are also compared.

ACKNOWLEDGMENTS

I would like to give thanks to Nathan Knobel and Stephen Osborne whose help in designing, implementing, and testing the algorithms made this thesis possible. I must also thank my wife, Rachel, and friend, Dean Anderson, for proof reading this entire thesis. Finally, I would also like to thank Dr. Randal Beard for his guidance and help with this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem

Historically, unmanned aircraft (UA) have been used primarily for military purposes such as reconnaissance, weapons delivery, battlefield communications, and anything that is too "dull, dirty, or dangerous" for human pilots [1]. Examples of UAs are the MQ-1 Predator (reconnaissance and weapons delivery), the RQ-4 Global Hawk (reconnaissance), and the RQ-2B Pioneer (reconnaissance).

Micro air vehicles (MAV) are a type of UA that is becoming popular on the battlefield. MAVs are easily transported and deployed by soldiers in the field. Because MAVs have become smaller, cheaper, and easy to transport, non-military applications are gaining interest. As non-military uses become more popular, creating user-friendly MAVs is essential for commercial viability. MAVs are finding their way into law enforcement, crop surveillance, fire fighting, communication relays, search and rescue, etc. However, most commercial autopilots must be hand-tuned when installed in an aircraft. This typically involves an experienced remote control (RC) pilot flying the MAV to tune proportional-integral-derivative control (PID) gains and trim the airplane. Such a process can be expensive and time intensive. Furthermore, due to inexact manufacturing processes, temperature sensitivities, crashes, and changing atmospheric conditions, the tuned values vary from aircraft to aircraft and sometimes on each airplane throughout the day. For many applications, tuning the autopilot can range from annoying to unacceptable. Also, even though MAVs are decreasing in price, the loss of an aircraft can still be expensive. Thus, an autopilot should be self tuning and fault tolerant. Adaptive control schemes show promise for fulfilling these requirements.

The focus of this thesis is adaptive control of fixed-wing MAVs. Experience has shown that PID values are different for each MAV, even if they have the same physical geometry. Adaptive control is an approach at allowing one autopilot to control every MAV while requiring minimal tuning. Because of the physical size of the MAVs, a small autopilot is used that has limited code space, memory, and computational resources. Also, the aerodynamic parameters for the MAVs are unknown. Thus the adaptive autopilot code must have a small footprint, require little computation and memory to run, and be able to adapt itself to any fixed-wing MAV.

1.2 Adaptive Control

Adaptive control is a term used to describe a large class of control systems. Simply put, adaptive control algorithms adapt to unknown parameters in a plant. This does not mean that an adaptive controller is the optimal controller for a system. Instead, adaptive controllers are able to control plants with parameters that are unknown or changing. To illustrate this idea let us look at traditional PID control.

PID control is the swiss army knife of control theory. It can control a large set of plants, and its intent is to drive the error between a desired reference signal and the output of the plant to zero. It does this by operating on that error and passing the result to the plant's input. The proportional part amplifies the error, it is used to drive the error to zero. The integral part amplifies the integral of the error, it is used to eliminate steady-state error. And the derivative part amplifies the derivative of the error, it is used to reduce oscillations caused by the previous two parts. These three signals are added together and passed to the plant's input. PID gain controllers are manually tuned for each system to satisfaction of the operator. However, as the plant's parameters change, the PID controller may need to be retuned. Parameter variation can be caused by changes in environmental conditions, state changes (i.e. airplane dynamics changing as a result of airspeed, angle of attach, and sideslip angle), time progression, etc. For MAVs, this means that a PID controller that is tuned in one flight regime may not work as well or become unstable under another flight regime, thus requiring re-tuning. Adaptive control typically does not suffer from this problem.

The goal of adaptive control is to adjust to unknown or changing plant parameters. This is accomplished by either changing parameters in the controller to minimize error, or using plant parameter estimates to change the control signal. Therefore, there are many different approaches to adaptive control. In this thesis three different types of adaptive control are discussed–least square estimation adaptive control, dynamic inversion using neural networks, and model reference adaptive control (MRAC). Least squares adaptive control uses least squares estimation to perform online system identification. The parameter estimates for the system are then used in the controller. Dynamic inversion is the process of inverting the dynamics of the system to make control design easier. When neural networks are used in conjunction with dynamic inversion, the neural network can learn to invert the dynamics of the plant on-the-fly, thereby adapting to changing parameters. Finally, MRACs use a reference model to update parameter estimates. These estimates are used to help drive the system error to zero.

1.3 Contributions

This thesis has two main contributions. They are the application of two varieties of MRAC to MAVs. The first type of MRAC is a Lyapunov based MRAC. Lyapunov based MRACs are not new, but do not seem to have been applied to MAVs. The second is an extension to Lyapunov MRACs, known as $\mathcal{L}_1$ adaptive controllers. $\mathcal{L}_1$ controllers are relatively new and are actively being researched. To the author's knowledge, this is the first real-world implementation of an $\mathcal{L}_1$ adaptive flight controller appearing in the literature, [2–4]. In this thesis, four adaptive schemes will be derived for each type of MRAC. The results of these schemes will be compared and contrasted.

1.4 Thesis Outline

This thesis is divided into six chapters. Chapter 2 introduces notation, describes the system model for MAVs, presents important theories for this thesis, and discusses different type of adaptive control. Chapter 3 derives the different adaptive schemes tested in this thesis. Chapter 4 describes the experimental platforms. Chapter 5 give the results of simulation and flight testing of the adaptive algorithms. Finally, Chapter 6 is the thesis summary and conclusion.

Chapter 2

Preliminaries

2.1 Notation

This thesis adopts the following notations. Lower case letters are scalars, while bold lower case letters and lower case Greek letters are vectors. Matrices are indicated by capital Roman or capital Greek letters. Estimates of states or parameters are indicated by carets. A tilde indicates an error signal (i.e. $\tilde{\phi} = \phi - \phi_m$). We use $|\cdot|$ as the absolute value of a scalar while we use $||\cdot||$ as the vector norm or the matrix norm (i.e. $||\mathbf{x}||$ is a vector norm and $||\mathrm{A}||$ is a matrix norm). Because we are only concerned with the continuous case, states, inputs, and parameters that depend on time will be in the form of $x(t)$ unless context allows the argument to be dropped. In some instances frequency and time domains are mixed using the convention $C(s)\{f(t)\}$ to mean that the time domain function $f(t)$ is operated on by the frequency domain function $C(s)$. Table 2.1 defines some of the notation used in this thesis.

Throughout this thesis, the projection operator is used. The projection operator ensures that the parameter estimates remain within a certain bound. To define the projection operator, let $g(\mathbf{b}) = 0$ be a function that defines the surface of the set S such that $S = \{\mathbf{b} \in \mathcal{R}^n | g(\mathbf{b}) \leq 0\}$. The normal of the surface is $\bigtriangledown g(\mathbf{b})$. Furthermore, the set S_i is the interior of the set S, and S_o is the boundary of S. The initial parameter estimate, $\mathbf{b}_0$, is chosen such that $\mathbf{b}_0 \in S$. The operator is defined as

$$\mathrm{Proj}_S(\mathbf{a}, \mathbf{b}) = \begin{cases} \mathbf{a} & \mathbf{b} \in S_i \text{ or } \mathbf{b} \in S_o \text{ and } \mathbf{a}^T \bigtriangledown g(\mathbf{b}) \leq 0 \\ \left(I - \frac{\bigtriangledown g(\mathbf{b}) \bigtriangledown g(\mathbf{b})^T}{\bigtriangledown g(\mathbf{b})^T \bigtriangledown g(\mathbf{b})}\right)\mathbf{a} & \mathbf{b} \in S_o \text{ and } \mathbf{a}^T \bigtriangledown g(\mathbf{b}) > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{2.1}$$

Table 2.1: State Definitions

ϕ, θ, ψ	roll, pitch, yaw angles
ϕ^d,θ^d	desired roll and pitch angles
ϕ_m, θ_m	reference model roll and pitch angles
p, q, r	roll, pitch, yaw rates
α, β	angle of attack, sideslip angle
χ, γ	ground track, climb angle
δ_e, δ_a, δ_r	elevator, aileron, rudder commands
V	airspeed
S, b, $\bar{c}$	wing area, wing span, average cord length
J_{ij}	inertial moments about i and j axes
C_*	aerodynamic coefficients
$\hat{J}_*$	Constants that depend on J

The projection operator reduces to

$$\mathrm{Proj}_S\left(a, b\right) = \begin{cases} a & \underline{b} < b < \bar{b} \\ a & b = \bar{b} \text{ and } a \leq 0 \text{ or } b = \underline{b} \text{ and } a \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

in the scalar case. Further discussion about the projection operator can be found in [5].

2.2 Mathematical Model

The mathematic model for the MAVs is a fixed-wing aircraft model that assumes a symmetry in the body frame (see Figure 2.1) $x - z$ plane, being either a flying-wing or fuselage design. This implies that the inertia matrix is of the form

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

We use this matrix to develop the equations of motion for our model. Notice the symmetry across the diagonals.

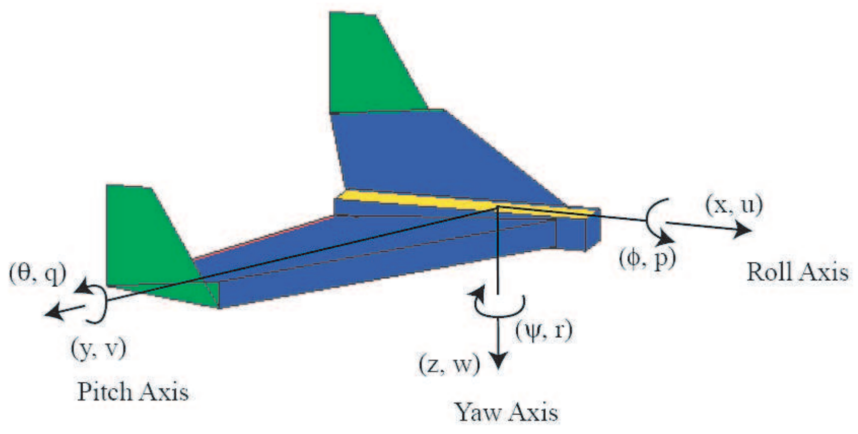


Figure 2.1: The body frame coordintes for the MAV model.

The translational kinematics and dynamics of our six degree-of-freedom model (6-DOF) are given by

$$\begin{aligned}
\dot{p}_n =& \cos\theta\cos\psi u + (\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi)v \\
&+ (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)w, && (2.3)\\
\dot{p}_e =& \cos\theta\sin\psi u + (\sin\phi\sin\theta\sin\psi - \cos\phi\cos\psi)v \\
&+ (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)w, && (2.4)\\
\dot{h} =& u\sin\theta - v\sin\phi\cos\theta - w\cos\phi\cos\theta, && (2.5)\\
\dot{u} =& rv - qw - g\sin\theta + C_X(\mathbf{x},\delta), && (2.6)\\
\dot{v} =& pw - ru + g\cos\theta\sin\phi + C_Y(\mathbf{x},\delta), \text{ and} && (2.7)\\
\dot{w} =& qu - pv + g\cos\theta\cos\phi + C_Z(\mathbf{x},\delta), && (2.8)
\end{aligned}$$

where

- p_n is the position north,
- p_e is the position east,
- h is the altitude,
- u is the velocity out the nose,

- v is the velocity out the right wing,
- w is the velocity out the belly,
- x is the state vector, and
- δ is the control vector.

Also, the aerodynamic forces are

$$\begin{aligned}
C_X(\mathbf{x},\delta) =& \frac{\rho V_{air}^2 S}{2m}\left[C_{X_0} + C_{X_\alpha}\alpha + C_{X_q}\frac{\bar{c}q}{V_{air}} + C_{X_{\delta_e}}\delta_e\right], \\
&+ \frac{\rho S_{prop}}{2m} C_{prop}\left[(k\delta_t)^2 - V_{air}^2\right], \quad (2.9)\\
C_Y(\mathbf{x},\delta) =& \frac{\rho V_{air}^2 S}{2m}\left[C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{bp}{2V_{air}} + C_{Y_r}\frac{br}{2V_{air}} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r\right], \text{ and} \quad (2.10)\\
C_Z(\mathbf{x},\delta) =& \frac{\rho V_{air}^2 S}{2m}\left[C_{Z_0} + C_{Z_\alpha}\alpha + C_{Z_q}\frac{\bar{c}q}{V_{air}} + C_{Z_{\delta_e}}\delta_e\right], \quad (2.11)
\end{aligned}$$

and are assumed to be linear. This should be an appropriate model as the MAVs in this thesis are not high-performance aircraft. The rotational kinematics and dynamics of the MAVs are

$$\begin{aligned}
\dot{\phi} =& p + q\sin\phi\tan\theta + r\cos\phi\tan\theta, \quad (2.12)\\
\dot{\theta} =& q\cos\phi - r\sin\phi, \quad (2.13)\\
\dot{\psi} =& q\sin\phi\sec\theta + r\cos\phi\sec\theta, \quad (2.14)\\
\dot{p} =& \hat{J}_1 pq - \hat{J}_2 qr + \frac{1}{2}\rho V^2 S\frac{b}{2}\bigg[C_{p_0} + C_{p_\beta}\beta \\
&+ C_{p_p}\frac{bp}{2V} + C_{p_r}\frac{br}{2V} + C_{p_{\delta_a}}\delta_a + C_{p_{\delta_r}}\delta_r\bigg], \quad (2.15)\\
\dot{q} =& \frac{J_{xz}}{J_y}(r^2 - p^2) + \frac{J_z - J_x}{J_y}pr \\
&+ \frac{1}{2J_y}\rho V^2 \bar{c} S\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{\bar{c}q}{V} + C_{m_{\delta_e}}\delta_e\right], \quad (2.16)
\end{aligned}$$

$$\dot{r} = \hat{J}_3 pq - \hat{J}_4 qr + \frac{1}{2}\rho V^2 S \frac{b}{2}\Big[C_{r_0} + C_{r_\beta}\beta + C_{r_p}\frac{bp}{2V} + C_{r_r}\frac{br}{2V} + C_{r_{\delta_a}}\delta_a + C_{r_{\delta_r}}\delta_r\Big], \tag{2.17}$$

$$\ell = \frac{1}{4}b\rho V_{air}^2\left(C_{\ell_0} + C_{\ell_\beta}\beta + C_{\ell_p}\frac{bp}{2V} + C_{\ell_r}\frac{br}{2V} + C_{\ell_{\delta_a}}\delta_a + C_{\ell_{\delta_r}}\delta_r\right), \tag{2.18}$$

$$m = \frac{1}{2}\bar{c}\rho V_{air}^2\left(C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{\bar{c}q}{2V} + C_{m_{\delta_e}}\delta_e\right), \text{ and} \tag{2.19}$$

$$n = \frac{1}{4}b\rho V_{air}^2\left(C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{bp}{2V} + C_{n_r}\frac{br}{2V} + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r\right), \tag{2.20}$$

where

$$\begin{aligned}
C_{p_0} &= \hat{J}_3 C_{\ell_0} + \hat{J}_4 C_{n_0}, & C_{p_\beta} &= \hat{J}_3 C_{\ell_\beta} + \hat{J}_4 C_{n_\beta},\\
C_{p_p} &= \hat{J}_3 C_{\ell_p} + \hat{J}_4 C_{n_p}, & C_{p_r} &= \hat{J}_3 C_{\ell_r} + \hat{J}_4 C_{n_r},\\
C_{p_{\delta_a}} &= \hat{J}_3 C_{\ell_{\delta_a}} + \hat{J}_4 C_{n_{\delta_a}}, & C_{p_{\delta_r}} &= \hat{J}_3 C_{\ell_{\delta_r}} + \hat{J}_4 C_{n_{\delta_r}},\\
C_{r_0} &= \hat{J}_4 C_{\ell_0} + \hat{J}_3 C_{n_0}, & C_{r_\beta} &= \hat{J}_4 C_{\ell_\beta} + \hat{J}_3 C_{n_\beta},\\
C_{r_p} &= \hat{J}_4 C_{\ell_p} + \hat{J}_3 C_{n_p}, & C_{r_r} &= \hat{J}_4 C_{\ell_r} + \hat{J}_3 C_{n_r},\\
C_{r_{\delta_a}} &= \hat{J}_4 C_{\ell_{\delta_a}} + \hat{J}_3 C_{n_{\delta_a}}, & C_{r_{\delta_r}} &= \hat{J}_4 C_{\ell_{\delta_r}} + \hat{J}_3 C_{n_{\delta_r}},\\
\hat{J}_1 &= \tfrac{J_{xz}(J_x - J_y + J_z)}{J_x J_z - J_{xz}^2}, \text{ and} & \hat{J}_2 &= \frac{J_z(J_z - J_y) + J_{xz}^2}{J_x J_z - J_{xz}^2}.
\end{aligned}$$

Furthermore, we make the following assumptions:

A1: V, θ, ϕ, p, r, q can be measured or estimated by on-board sensors.

A2: The pitch angle is limited to $-\bar{\theta} \leq \theta \leq \bar{\theta}$ where $\bar{\theta} < \pi/2$.

A3: The roll angle is limited to $-\bar{\phi} \leq \phi \leq \bar{\phi}$.

A4: The pitch angle $\theta = \gamma + \alpha$.

A5: The sideslip angle $\beta = \psi - \chi$.

The control algorithms in this thesis deal only with pitch and roll attitude hold. Therefore, attention is restricted to pitch angle, roll angle, pitch rate, and roll rate equations. All other states are treated as measured states when used in the pitch and roll equations and are otherwise ignored. The exceptions are α and β as they cannot be measured on the MAVs.

2.3 Lyapunov's Second Method

When working with nonlinear or time-varying systems, it can be difficult or impossible to prove stability of an equilibrium point using classic control theory. The Russian mathematician, Aleksandr Lyapunov, developed an alternative method to determine stability when conventional methods fail. His method is based on the idea that all solutions starting at points surrounding a stable equilibrium point stay near the equilibrium point. The equilibrium point is unstable otherwise [6]. Lyapunov functions can be thought of as energy-like functions. A system that is not gaining energy is stable, whereas, a system that is gaining energy is unstable. We introduce the following theorems as they will be used in this work. The theorems use the idea of class $\mathcal{K}$ and class $\mathcal{KL}$ functions found in [6]. The following theorems are restated as found in [6]. For all theorems, let $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$.

Theorem 2.3.1 *Let $\boldsymbol{x} = \boldsymbol{0}$ be an equilibrium point for the nonautonomous system $\dot{\boldsymbol{x}} = f(t, \boldsymbol{x})$ and $\boldsymbol{D} \subset \boldsymbol{R}^n$ be a domain containing $\boldsymbol{x} = \boldsymbol{0}$. Let $V : [0, \infty) \times \boldsymbol{D} \rightarrow \boldsymbol{R}$ be a continuously differentiable function such that*

$$W_1(\boldsymbol{x}) \leq V(t, \boldsymbol{x}) \leq W_2(\boldsymbol{x}) \quad \textit{and} \tag{2.21}$$

$$\dot{V} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \boldsymbol{x}} f(t, \boldsymbol{x}) \leq 0, \tag{2.22}$$

$\forall t \geq 0$ and $\forall \boldsymbol{x} \in \boldsymbol{D}$ where W_1 and W_2 are continuous positive definite functions on $\boldsymbol{D}$. Then, $\boldsymbol{x} = 0$ is uniformly stable.

Proof: see [6].

This theorem is applied by first defining a Lyapunov function over some domain $\boldsymbol{D}$ that is bounded by positive definite functions. If the derivative of the Lyapunov function is nonincreasing, the system is stable. In other words, if the energy-like function is not increasing in energy it is stable, but convergence to the equilibrium point is not guaranteed.

Example: Take the system

$$\begin{aligned} \dot{x}_1 &= -x_1 \text{ and} \\ \dot{x}_2 &= 2x_1 - x_2, \end{aligned} \tag{2.23}$$

and define the Lyapunov candidate function

$$V = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2.$$

This Lyapunov function satisfies the first part of Theorem 2.3.1. Taking its derivative yields

$$\begin{aligned} \dot{V} &= x_1\dot{x}_1 + x_2\dot{x}_2, \\ &= -x_1^2 + 2x_1x_2 - x_2^2, \text{ and} \\ &= -\left(x_1 - x_2\right)^2 \leq 0. \end{aligned} \tag{2.24}$$

Equation (2.24) completely satisfies Theorem 2.3.1, therefore the system (2.23) is uniformly stable.

Theorem 2.3.2 *Suppose the assumptions of Theorem 2.3.1 are satisfied with inequality (2.22) strengthened to*

$$\dot{V} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \boldsymbol{x}} f\left(t, \boldsymbol{x}\right) \leq -W_3(\boldsymbol{x}),$$

$\forall t \geq 0$ *and* $\forall \boldsymbol{x} \in \boldsymbol{D}$ *where* W_3 *is a continuous positive definite function on* $\boldsymbol{D}$. *Then,* $\boldsymbol{x} = 0$ *is asymptotically uniformly stable. Moreover, if* r *and* c *are chosen such that the ball,* $\boldsymbol{B}_r$, *is defined as* $\boldsymbol{B}_r = \{||\boldsymbol{x}|| \leq r\} \subset \boldsymbol{D}$ *and* $c \leq \min_{||\boldsymbol{x}||=r} W_1$, *then every trajectory starting in* $\{\boldsymbol{x} \in \boldsymbol{B}_r | \, W_2\left(\boldsymbol{x}\right) \leq c\}$ *satisfies*

$$||\boldsymbol{x}(t)|| \leq \beta\left(||\boldsymbol{x}(t_0)||\,, t - t_0\right),$$

for some $\mathcal{KL}$ function β. Finally, if $\boldsymbol{D} = \boldsymbol{R}^n$ and $W_1(\boldsymbol{x})$ is radially unbounded, then $\boldsymbol{x} = 0$ is globally uniformly asymptotically stable.

Proof: see [6].

Theorem 2.3.2 simply extends Theorem 2.3.1 such that if the energy-like function is decreasing, the system is converging to the equilibrium point asymptotically. Furthermore, if the domain encompasses all of $\boldsymbol{R}^n$, the convergence holds globally.

Example: Take the system

$$\begin{aligned} \dot{x}_1 &= -x_1 + x_2 \text{ and} \\ \dot{x}_2 &= -x_1 - x_2, \end{aligned} \tag{2.25}$$

and define the Lyapunov candidate function

$$V = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2.$$

This Lyapunov function satisfies the first part of Theorem 2.3.1. Taking its derivative we get

$$\begin{aligned} \dot{V} &= x_1\dot{x}_1 + x_2\dot{x}_2, \\ &= -x_1^2 + x_1x_2 - x_1x_2 - x_2^2, \text{ and} \\ &= -x_1^2 - x_2^2. \end{aligned} \tag{2.26}$$

Equation (2.26) satisfies Theorem 2.3.2 over the domain $\boldsymbol{R}^n$, therefore the system (2.25) is globally uniformly asymptotically stable.

Lemma 2.3.3 *(Barbalat's Lemma) Let $\phi : \boldsymbol{R} \to \boldsymbol{R}$ be a uniformly continuous function on $[0, \infty)$. Suppose that $\lim_{t\to\infty} \int_0^t \phi(\tau)\, d\tau$ exists and is finite. Then,*

$$\phi(t) \to 0 \text{ as } t \to \infty.$$

Proof: see [6].

Theorem 2.3.4 *Let $\boldsymbol{D} \subset \boldsymbol{R}$ be a domain containing $\boldsymbol{x} = \boldsymbol{0}$ and suppose $f(t, \boldsymbol{x})$ is piecewise continuous in t and locally Lipschitz in $\boldsymbol{x}$, uniformly in t, on $[0, \infty) \times \boldsymbol{D}$. Furthermore, suppose $f(t, \boldsymbol{0})$ is uniformly bounded for all $t \geq 0$. Let $V : [0, \infty) \times \boldsymbol{D} \rightarrow \boldsymbol{R}$ be a continuous differentiable function such that*

$$\begin{aligned} W_1(\boldsymbol{x}) &\leq V(t, \boldsymbol{x}) \leq W_2(\boldsymbol{x}) \ \ \textit{and} \\ \frac{\partial V}{\partial t} &+ \frac{\partial V}{\partial \boldsymbol{x}} f(t, \boldsymbol{x}) \leq -W(\boldsymbol{x}), \end{aligned}$$

$\forall t \geq 0$, $\forall \boldsymbol{x} \in \boldsymbol{D}$, *where $W_1(\boldsymbol{x})$ and $W_1(\boldsymbol{x})$ are continuously positive definite functions and $W(\boldsymbol{x})$ is a continuous positive semidefinite function on $\boldsymbol{D}$. Choose $r > 0$ such that the ball $\boldsymbol{B}_r \subset D$ and let $\rho < \min_{||\boldsymbol{x}||=r} W_1(\boldsymbol{x})$. Then, all solutions of $\dot{\boldsymbol{x}} = f(t, \boldsymbol{x})$ with $\boldsymbol{x}_0 \in \{\boldsymbol{x} \in \boldsymbol{B}_r | W_2(\boldsymbol{x}) \leq \rho\}$ are bounded and satisfy*

$$W(\boldsymbol{x}(t)) \rightarrow 0 \text{ as } t \rightarrow \infty.$$

Moreover, if all the assumptions hold globally and $W_1(\boldsymbol{x})$ is radially unbounded, the statement is true for all $\boldsymbol{x}(t_0) \in \boldsymbol{R}^n$.

Proof: see [6].

Therefore, if solutions start in the ball, $\mathbf{B}_r$, then $\mathbf{x}(t) \rightarrow \{\mathbf{x} \in \boldsymbol{D} \mid W(\mathbf{x}) = 0\}$ as $t \rightarrow \infty$. This, unfortunately, only guarantees convergence to a set that satisfies $W_3(\mathbf{x}) = 0$ and not convergence to the equilibrium point. However, it may be possible to show that some, or all, of the states go to zero in order for $\mathbf{x}(t)$ to be a member of the above set.

Example: Let

$$\dot{x} = -x - x\mathrm{e}^{-t} \tag{2.27}$$

be a system with a Lyapunov candidate function

$$V = \frac{1}{2}x^2.$$

Taking derivative of the Lyapunov function derivative gives us

$$\begin{aligned}\dot{V} &= x\dot{x},\\ &= -x^2 - x^2\mathrm{e}^{-t}, \text{ and}\\ &= -x^2\left(1+\mathrm{e}^{-t}\right) \leq 0.\end{aligned}$$

We define

$$W(x(t)) = x^2$$

so that

$$\dot{V} = -x^2\left(1+\mathrm{e}^{-t}\right) \leq -W(x(t)). \tag{2.28}$$

If we integrate Equation (2.28), we get

$$\begin{gathered}\int_{t_0}^{t} \dot{V}(t, x(\tau))d\tau \leq -\int_{t_0}^{t} W(x(\tau))d\tau,\\ -\int_{t_0}^{t} \dot{V}(t, x(\tau))d\tau \geq \int_{t_0}^{t} W(x(\tau))d\tau, \text{ and}\\ V(t_0, x(t_0)) - V(t, x(t_0) \geq \int_{t_0}^{t} W(x(\tau))d\tau.\end{gathered}$$

Because $V(t,x)$ is decreasing, the left hand side of the equation is finite. Because of Lemma 2.3.3, this implies

$$W(x(t)) \to 0 \text{ as } t \to \infty,$$

which implies

$$x(t) \to \left\{x \in \boldsymbol{D} \mid W(x(t)) = 0\right\}.$$

If we let $r = \infty$, all solutions of $\dot{x} = -x - x\mathrm{e}^{-t}$ are bounded. But since $W(x) = x^2$, the set where $W(x) = 0$ contains only the equilibrium point. Thus,

$$x(t) \to 0 \text{ as } t \to \infty.$$

Theorem 2.3.5 *Let $\boldsymbol{D} \subset \boldsymbol{R}$ be a domain containing $\boldsymbol{x} = \boldsymbol{0}$ and suppose $f(t, \boldsymbol{x})$ is piecewise continuous in t and locally Lipschitz in $\boldsymbol{x}$ for all $t \geq 0$ and $x \in \boldsymbol{D}$. Let $\boldsymbol{x} = \boldsymbol{0}$ be an equilibrium point for $\dot{\boldsymbol{x}} = f(t, \boldsymbol{x})$ at $t = 0$. Let $V : [0, \infty) \times \boldsymbol{D} \rightarrow \boldsymbol{R}$ be a continuously differentiable function such that*

$$W_1(\boldsymbol{x}) \leq V(t, \boldsymbol{x}) \leq W_2(\boldsymbol{x}),$$
$$\dot{V}(t, \boldsymbol{x}) = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \boldsymbol{x}} f(t, \boldsymbol{x}) \leq 0, \text{ and}$$
$$V(t+\delta, \phi(t+\delta; t, \boldsymbol{x})) - V(t, \boldsymbol{x}) \leq -\lambda V(t, \boldsymbol{x}), \; 0 < \lambda < 1,$$

$\forall t \geq 0$, $\forall \boldsymbol{x} \in \boldsymbol{D}$, for some $\delta > 0$, where $W_1(\boldsymbol{x})$ and $W_1(\boldsymbol{x})$ are continuously positive definite functions on $\boldsymbol{D}$ and $\phi(\tau; t, \boldsymbol{x})$ is the solution of the system that starts at $(t, \boldsymbol{x})$. Then, the origin is uniformly asymptotically stable. If all the assumptions hold globally and $W_1(\boldsymbol{x})$ is radially unbounded, then the origin is globally uniformly asymptotically stable. If

$$W_1(\boldsymbol{x}) \geq k_1 \|\boldsymbol{x}\|^c, W_2(\boldsymbol{x}) \leq k_2 \|\boldsymbol{x}\|^c, \; k_1 > 0, \; k_2 > 0, \; c > 0$$

then the origin is exponentially stable.

Proof: see [6].

Theorem 2.3.5 states that asymptotic or exponential stability can be inferred as long as the Lyapunov function is decreasing. This holds even if the Lyapunov function's derivative is negative semi-definite.

Example: Looking at the previous example $\dot{V} \leq 0$. And because V is decreasing,

$$V(t+\delta, \phi(t+\delta; t, x)) - V(t, x) \leq -\lambda V(t, x).$$

Moreover, by letting $k_1 = \frac{1}{2}$ and $k_2 = 2$, V is bounded above and below such that

$$\frac{1}{2} \|x\|^2 \leq V(t, x) \leq 2 \|x\|^2.$$

Therefore by using Theorem 2.3.5 we can determine that this system is globally exponentially stable.

For the following theorem, we are considering the system

$$\dot{\mathbf{x}} = f(t, \mathbf{x}), \tag{2.29}$$

where $f : [0, \infty) \times \mathbf{D} \to \mathbf{R}^n$ is piecewise continuous in t and locally Lipschitz in $\mathbf{x}$ on $[0, \infty) \times \mathbf{D}$, and $\mathbf{D} \subset \mathbf{R}^n$ is a domain that contains the origin [6].

Theorem 2.3.6 *Let $\boldsymbol{D} \subset \boldsymbol{R}^n$ be a domain that contains the origin and $V : [0, \infty) \times \boldsymbol{D} \to \boldsymbol{R}$ be a continuously differentiable function such that*

$$\begin{gathered}\alpha_1(||\boldsymbol{x}||) \leq V(t, \boldsymbol{x}) \leq \alpha_2(||\boldsymbol{x}||) \ \textit{and} \\ \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \boldsymbol{x}} f(t, \boldsymbol{x}) \leq -W_3(\boldsymbol{x}), \quad \forall \ ||\boldsymbol{x}|| \geq \mu > 0,\end{gathered}$$

$\forall\ t \geq 0$ and $\forall\ \boldsymbol{x} \in \boldsymbol{D}$, where α_1 and α_2 are class $\mathcal{K}$ functions and $W_3(\boldsymbol{x})$ is a continuous positive definite function. Take $r > 0$ such that $\boldsymbol{B}_r \subset \boldsymbol{D}$ and suppose that

$$\mu < \alpha_2^{-1}(\alpha_1(r))$$

Then, there exists a class $\mathcal{KL}$ function β and for every initial state $\boldsymbol{x}(t_0)$, satisfying $||\boldsymbol{x}(t_0)|| \leq \alpha_2^{-1}(\alpha_1(r))$, there is $T \geq 0$ (dependent on $\boldsymbol{x}(t_0)$ and μ) such that the solution of (2.29) satisfies

$$||\boldsymbol{x}(t)|| \leq \beta(||\boldsymbol{x}(t_0)||, t - t_0), \quad \forall\ t_0 \leq t \leq t_0 + T \ \textit{and} \tag{2.30}$$

$$||\boldsymbol{x}(t)|| \leq \alpha_1^{-1}(\alpha_2(\mu)), \quad \forall\ t \geq t_0 + T, \tag{2.31}$$

Moreover, if $\boldsymbol{D} = \boldsymbol{R}^n$ and α_1 belongs to class $\mathcal{K}_\infty$, then (2.30) and (2.31) hold for any initial state $\boldsymbol{x}(t_0)$, with no restriction on how large μ is.

Proof: see [6].

This theorem states that if a ball, $\mathbf{B}_r$, is selected such that the Lyapunov candidate function is decreasing for $||\mathbf{x}|| > \mu$, where and $\mu \in \mathbf{B}_r$, then ultimate boundedness can be determined. This theorem is useful when the derivative of the Lyapunov candidate function is indeterminate around the equilibrium. The cause of indeterminance maybe a positive definite term or a term that is indeterminate or unknown but has an upper bound.

Example: Define the system as

$$\begin{aligned} \dot{x}_1 &= M - x_1 \text{ and} \\ \dot{x}_2 &= M - x_2, \end{aligned} \tag{2.32}$$

where M is an unknown positive constant. We define the Lyapunov candidate function as

$$V = \frac{1}{2}\mathbf{x}^T\mathbf{x},$$

which is bound above and below by class $\mathcal{K}$ functions such that

$$\left(\alpha_1(||\mathbf{x}||) = \frac{1}{2}\left|\left|\mathbf{x}\right|\right|^2\right) \leq V(||\mathbf{x}||^2) \leq \left(\alpha_2(||\mathbf{x}||) = 2\left|\left|\mathbf{x}\right|\right|^2\right).$$

Taking the derivative yields

$$\begin{aligned} \dot{V} &= \mathbf{x}^T\dot{\mathbf{x}}, \\ &= \mathbf{x}^T\left(-\mathbf{x} + M\begin{bmatrix}1\\1\end{bmatrix}\right), \\ &= -\left|\left|\mathbf{x}\right|\right|^2 + M\mathbf{x}^T\mathbf{y}, \text{ where } \mathbf{y} = \begin{bmatrix}1 & 1\end{bmatrix}^T, \text{ and} \\ &\leq -\left|\left|\mathbf{x}\right|\right|^2 + M\sqrt{2}\left|\left|\mathbf{x}\right|\right|, \end{aligned}$$

where we have used the relation $\mathbf{x}^T\mathbf{y} \leq ||\mathbf{x}||\,||\mathbf{y}||$. Observe that the derivative is negative as long as $-\left|\left|\mathbf{x}\right|\right|^2$ dominates $M\sqrt{2}\,||\mathbf{x}||$. Setting $\mu = M\sqrt{2}\,||\mathbf{x}||$ and applying Theorem 2.3.6 we get that $||\mathbf{x}||$ is bounded by $||\mathbf{x}|| \leq M\sqrt{2}$.

Lemma 2.3.7 *Let*

$$f(\theta, \phi, \psi) \;\; = \;\; \theta - \tilde{\psi}^T \left[\phi + \dot{\psi}\right]. \tag{2.33}$$

be a piecewise continuous function, where $\tilde{\psi} = \hat{\psi} - \psi$*. Furthermore, let* θ *be a piecewise continuous, time-varying function and* ψ *and* ϕ *be time-varying vectors of size* a*. Also, let each entry in* $\psi_i \in S_i$*, and define* $\hat{\psi}$ *to be the estimate of* ψ*. If we define the update law for* ψ *to be*

$$\dot{\psi} = Proj_S\left(-\phi, \psi\right), \tag{2.34}$$

where we are using the projection operator defined by Equation (2.1)*, then,*

$$f(\theta, \phi, \psi) \;\; \leq \;\; \theta. \tag{2.35}$$

Proof: Using the definition of the projection operator in (2.1) we have

$$\psi_i \in S_i, \;\; 0 \leq i \leq a. \tag{2.36}$$

Equation (2.34) and Equation (2.36) imply the following three cases.

Case 1: One or more entries are $|\psi_i| < \bar{\psi}_i$.

Each entry defined by this case has the derivative $\dot{\psi}_i = -\phi_i$, and therefore are canceled out by counterparts in ϕ. Therefore, $f(\theta, \phi, \psi)$ reduces to

$$f(\theta, \phi, \psi) = \theta.$$

Case 2: $\psi_i = \bar{\psi}_i$ for one or more entries.

Entries where $\psi_i = \bar{\psi}_i$ and $-\phi$ is negative are set to $\dot{\psi}_i = -\phi_i$ by the projection operator, which cancels with the appropriate ϕ_i terms. However, the entries where $\psi_i = \bar{\psi}_i$ and $-\phi$ is positive, the projection operator sets $\dot{\psi} = 0$. Equation (2.33) becomes

$$f(\theta, \phi, \psi) = \theta + \tilde{\psi}^T \phi. \tag{2.37}$$

But since $\psi_i \in S_i$, we have $\hat{\psi}_i \geq \psi_i$, which means that $\tilde{\psi}$ is positive and ϕ is negative. Thus, the terms for this case are negative. Equation (2.37) becomes

$$f(\theta, \phi, \psi) = \theta + \tilde{\psi}^T \phi \leq \theta.$$

Case 3: $\psi_i = -\bar{\psi}_i$ for one or more entries.

For the entries where $\psi_i = -\bar{\psi}_i$ and $-\phi$ is negative, the projection operator sets $\dot{\psi} = 0$. Equation (2.33) becomes

$$f(\theta, \phi, \psi) = \theta + \tilde{\psi}^T \phi. \tag{2.38}$$

But since $\psi_i \in S_i$, we have $\hat{\psi}_i \leq \psi_i$, which means that $\tilde{\psi}$ is negative and ϕ is positive. Thus, the terms for this case are negative. Equation (2.38) becomes

$$f(\theta, \phi, \psi) = \theta + \tilde{\psi}^T \phi \leq \theta.$$

The combined result of these three cases is

$$f(\theta, \phi, \psi) \leq \theta.$$

2.4 Forms of Adaptive Control

In the following sections we will discuss different types of adaptive control, namely least squares estimation, dynamic inversion with neural networks, and model reference adaptive control. Least squares estimation, dynamic inversion, and the MIT rule are included for completeness. A brief mathematical development of each is presented and their advantages and disadvantages are discussed.

2.4.1 Least Squares Adaptive Control

Least squares estimation uses the outputs of a system and estimates the plant parameters to minimize the squared error. A controller can then use the parameter estimates to adapt. Reference [7] is used for the derivation below. When using least squares estimation we minimize the cost function

$$J = \frac{1}{2}\left|\left|\mathbf{y}(t) - \Phi(t)\hat{\theta}\right|\right|^2, \tag{2.39}$$

where

$$\begin{aligned}
\vartheta^T(t) &= [\vartheta_1(t)\ \vartheta_2(t) \cdots \vartheta_n(t)], \\
\Phi^T(t) &= [\vartheta(1)\ \vartheta(2) \cdots \vartheta(t)], \\
\hat{\theta}^T(t) &= \left[\hat{\theta}_1(t)\ \hat{\theta}_2(t) \cdots \hat{\theta}_n(t)\right], \text{ and} \\
\mathbf{y}^T(t) &= [y(1)\ y(2) \cdots y(t)], \qquad (2.40)
\end{aligned}$$

and $\Phi(t)$ is a matrix of inputs, $\vartheta(t)$ is a vector of inputs, $\hat{\theta}$ is a vector of parameters, and $\mathbf{y}(t)$ is a vector of outputs. From calculus we know that the minima occurs when $\Phi(t)\hat{\theta} = \mathbf{y}(t)$. If Φ is an m×n matrix, $\hat{\theta}$ is n×1, and $\mathbf{y}$ is m×1 we can solve for the minimum by

$$\begin{aligned}
& \Phi^T(t)\Phi(t)\hat{\theta}(t) = \Phi^T(t)\mathbf{y}(t), \\
\Rightarrow\quad & \left(\Phi^T(t)\Phi(t)\right)^{-1}\Phi(t)^T\Phi(t)\hat{\theta}(t) = \left(\Phi^T(t)\Phi(t)\right)^{-1}\Phi^T(t)\mathbf{y}(t), \text{ and} \\
\Rightarrow\quad & \hat{\theta}(t) = \left(\Phi^T(t)\Phi(t)\right)^{-1}\Phi^T(t)\mathbf{y}(t), \qquad (2.41)
\end{aligned}$$

given that $\Phi(t)^T\Phi(t)$ is invertible. Obviously, every time there is new information, this equation must be rerun to compute the parameters. This can be impractical as datasets become large and computationally intensive. Also, memory will be needed to store the entire dataset, which is impractical. These issue can be addressed using a recursive form. To find the recursive form we define

$$\begin{aligned}
P(t) &= \left(\Phi^T(t)\Phi(t)\right)^{-1} \text{ and} \\
P(t) &= \left(\sum_{i=1}^{t}\left(\vartheta(i)\vartheta^T(i)\right)\right)^{-1} \qquad (2.42)
\end{aligned}$$

so that Equation (2.41) becomes

$$\hat{\theta}(t) = P(t)\Phi^T(t)\mathbf{y}(t). \tag{2.43}$$

It should be noted that the dimension of $P(t)$ will always be n×n. Using Equation (2.42) we get

$$\begin{aligned}
P^{-1}(t) &= \left(\sum_{i=1}^{t}\left(\vartheta(i)\vartheta^T(i)\right)\right), \\
P^{-1}(t) &= \left(\sum_{i=1}^{t-1}\left(\vartheta(i)\vartheta^T(i)\right)\right) + \vartheta(t)\vartheta^T(t), \text{ and} \\
P^{-1}(t) &= P^{-1}(t-1) + \vartheta(t)\vartheta^T(t). \qquad (2.44)
\end{aligned}$$

The above equation gives us a recursive way to compute the matrix $P^{-1}(t)$. To initialize, we simply use Equation (2.42) at $t = 0$ and Equation (2.44) at $t > 0$. If we plug Equation (2.42) into Equation (2.41) we get

$$\hat{\theta}(t) = P(t)\Phi^T(t)\mathbf{y}(t). \tag{2.45}$$

Taking this equation at $t = t - 1$ and substituting Equation (2.44) gives us the following formulation of a recursive least squares algorithm

$$\begin{aligned}
& \hat{\theta}(t-1) = P(t-1)\Phi^T(t-1)\mathbf{y}(t-1), \\
\Rightarrow \quad & P(t-1)^{-1}\hat{\theta}(t-1) = \Phi^T(t-1)\mathbf{y}(t-1), \text{ and} \\
\Rightarrow \quad & P(t)^{-1}\hat{\theta}(t-1) - \vartheta(t)\vartheta^T(t)\hat{\theta}(t-1) = \Phi^T(t-1)\mathbf{y}(t-1). \qquad (2.46)
\end{aligned}$$

Plugging (2.46) into (2.44) gives

$$\begin{aligned}
& \hat{\theta}(t) = P(t)\left(\Phi^T(t-1)\mathbf{y}(t-1) + \vartheta(t)y(t)\right), \\
\Rightarrow \quad & \hat{\theta}(t) = P(t)\left(P(t)^{-1}\hat{\theta}(t-1) - \vartheta(t)\vartheta^T(t)\hat{\theta}(t-1) + \vartheta(t)y(t)\right), \text{ and} \\
\Rightarrow \quad & \hat{\theta}(t) = \hat{\theta}(t-1) + P(t)\vartheta(t)\left(y(t) - \vartheta^T(t)\hat{\theta}(t-1)\right), \qquad (2.47)
\end{aligned}$$

which is the recursive update equation. To summarize,

$$\begin{aligned} P^{-1}(t) &= P^{-1}(t-1) + \vartheta(t)\vartheta^T(t) \text{ and} \\ \hat{\theta}(t) &= \hat{\theta}(t-1) + P(t)\vartheta(t)\left(y(t) - \vartheta^T(t)\hat{\theta}(t-1)\right). \end{aligned} \tag{2.48}$$

A version that does not require any inverses can be found in [8]. Used in the current form, least squares can estimate a parameter to be used by the controller. Unfortunately, the covariance matrix $P(t)$ can become singular. This can be rectified by resetting the covariance matrix or introducing a forgetting factor [9]. Adaptive control using least squares has the advantage of fast convergence to the parameters [9]. However, least squares estimation can be computationally intensive, particularly on small autopilots.

Least squares estimation has been used in the following papers. In [10], the authors use least squares estimation with a forgetting factor to estimate the states of an RC airplane. This airplane is flown with a pilot in control commanding pitch and roll rates. During the flight, half of the elevator is released. The least squares computes the parameter estimates and aircraft stability is maintained.

Several different approaches to adaptive control are explored in [11]. Dynamic inversion was performed using Modified Sequential Least Squares (MSLS) to identify the parameters of the aircraft. This approach was found to do well.

Paper [12] deals with the problem of ARMA models that lose their co-primeness. In this paper, least squares is used to estimate the parameters of an ARMA model. These estimates are used in an adaptive digital PID controller. The effectiveness of the controller is shown for an F-16.

2.4.2 Dynamic Inversion

Dynamic inversion is a technique where the error dynamics between the reference input and plant output are inverted to simplify regulator design. This technique can be applied to both linear and nonlinear systems, but it finds its greatest use in nonlinear systems. The derivation for dynamic inversion discussed below comes from [13].

We take an autonomous system with affine inputs as described by the state equation

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \text{ and} && (2.49) \\ y &= h(x). && (2.50) \end{aligned}$$

Defining an error system

$$e = r(t) - y(t) \tag{2.51}$$

and taking its derivative, we get

$$\dot{e} = \dot{r}(t) - \dot{y}(t). \tag{2.52}$$

Expanding $\dot{y}$ gives

$$\begin{aligned} \dot{y} &= \frac{\partial h}{\partial x}\dot{x}, \\ &= \frac{\partial h}{\partial x}f(x) + \frac{\partial h}{\partial x}g(x)u, \text{ and} \\ &= F(x) + G(x)u, \end{aligned}$$

where

$$F(x) \triangleq \frac{\partial h}{\partial x}f(x) \text{ and } G(x) \triangleq \frac{\partial h}{\partial x}g(x).$$

By picking

$$u = G^{-1}(x)\left[-F(x) + \dot{r} + v\right] \tag{2.53}$$

the error system becomes

$$\dot{e} = -v.$$

The input v can be viewed as an outer loop, while $G^{-1}(x)\left[-F(x) + \dot{r}\right]$ acts like an inner loop. The outer loop can be replaced by a variety of control schemes.

Looking at the final controller, some things become apparent. First, $G(x)$ must be invertible. Also, the controller needs a model of the plant or a lookup table for $F(x)$ and $G^{-1}(x)$. Having a model can be computationally expensive, while a lookup table requires memory. Furthermore, neither method is robust to changes in the plant. However, this last problem can be overcome using neural networks.

When used in dynamic inversion, neural networks eliminate the need for internal models or lookup tables, account for modeling error, and can adapt to changes in the plant. In an adaptive controller, typically two neural networks are employed. The first neural network is trained off-line from prior flight, wind tunnel, or simulated data. The second neural network is trained continuously during flight to account for modeling error and to adapt to changes in plant dynamics resulting from damages or actuator failures.

The main advantage of this design technique is that it deals directly with the nonlinearities of the plant. This translates into a wide operating range without using gain scheduling. Also, the outer loop is simple to design since the inner loop accounts for the plant dynamics [13]. However, this technique is not without its disadvantages.

One disadvantage is that dynamic inversion needs full-state feedback for the inner loop. On some aircraft it may be impractical or impossible to estimate all states, such as measuring angle-of-attack on RC-style aircraft. Another disadvantage is that the processor on a MAV may not be able to compute all of the nodes of the neural network fast enough to keep the aircraft stable.

Using dynamic inversion in conjunction with neural networks appears to be an important development in adaptive control. The following papers implement this combination. References [11, 14–16] use neural networks trained offline to invert the aircraft dynamics. An online neural network is then used to compensate for model error. In [17], the author performs a statistical analysis of the previous scheme, finding the one presented in the paper to have a high level of confidence. However, in [18] dynamic inversion is used to invert the dynamics of a tailless fighter aircraft without using neural networks. Instead, a neural network in conjunction with a PD controlller is used to adapt to failures on an aircraft.

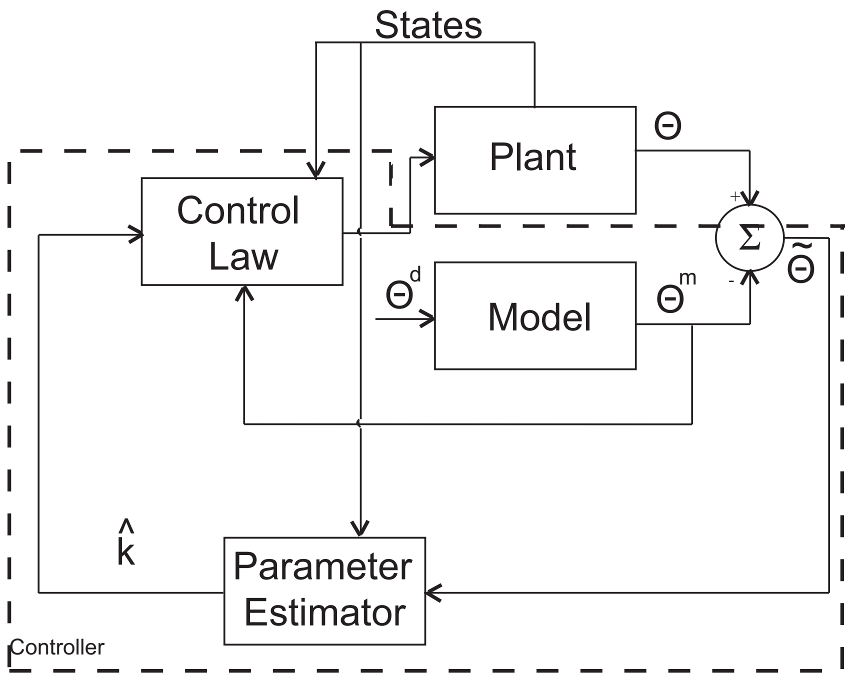


Figure 2.2: Block diagram of Lyapunov MRAC structure.

2.4.3 Model Reference Adaptive Control

In model reference control, the output of a plant is compared against the output of a model that is being driven by a reference signal. The error between the model output and plant output is then used to drive the plant to the desired reference input. Model reference adaptive control (MRAC) is a similar approach except instead of using the error between the model and the plant to drive the system to a reference input, the error is used to update the parameter estimates of the plant. Through updating the parameters, the tracking error is driven to zero. Essentially, the goal is to force the plant into behaving like the model. Figure 2.2 shows a block diagram of an MRAC.

In this subsection, three different types of MRAC are presented. The first part explains the gradient based method also known as the MIT rule. The second part describes a Lyapunov based MRAC, which uses stability concepts to develop its adaptive law. Finally, the last part lays out an extension to the Lyapunov MRACs called $\mathcal{L}_1$ controllers.

Gradient Based MRAC

The gradient based approach is one of the oldest forms of MRAC. It was developed by researchers at MIT [7]. This method employs gradient descent to update the parameters of the controller. Define the cost function J as

$$J = \frac{1}{2}e^2,$$

where e is the error between the actual plant output and the model output. On MAVs e might be the error in pitch or roll angles. Our goal is to minimize J, so we update the parameter estimates in the opposite direction of the gradient. Letting θ be the vector of parameters, the update law would be

$$\frac{d\theta}{dt} = -\gamma \frac{\partial J}{\partial \theta} = -\gamma e \frac{\partial e}{\partial \theta}. \tag{2.54}$$

The MIT rule is not guaranteed to find the true parameters of the system; to find the true parameters, persistent excitation is needed [7]. Persistent excitation is further elaborated on in [5]. Also, the MIT rule only works if changes in the parameters are small. This means that the adaptive gains are limited by the magnitude of the input signal and the process gain. The following modification is used to compensate for the previous limitation.

$$\frac{d\theta}{dt} = -\gamma \mathrm{sat}\left(\frac{e\frac{\partial e}{d\theta}}{\alpha + \left(\frac{\partial e}{\partial \theta}\right)^T \left(\frac{\partial e}{\partial \theta}\right)}, \beta \right), \tag{2.55}$$

where the saturation operator is defined as

$$\mathrm{sat}\left(\alpha, \beta\right) = \begin{cases} \alpha & -\beta \leq \alpha \leq \beta \\ \beta & \alpha > \beta \\ -\beta & \alpha < -\beta \end{cases}. \tag{2.56}$$

Unfortunately, we may still have an unstable system, e.g. γ is too high making the update equation "stiff". Normalization and proofs for stability in the sense of

Lyapunov are found in [5]. Conditions for global stability with proofs can be found in [19].

Lyapunov Stability

Lyapunov stability is based on an energy-like function constructed from the states of the system. If the derivative of the energy equation is nonincreasing, we can say the system is stable. Section 2.3 discusses the Lyapunov stability criteria in more detail. Lyapunov stability can be used to create MRACs that have certain stability guarantees. MRACs based on Lyapunov stability have similar update equations to gradient based MRACs, but differ on the signals used in the update equations. To see the difference we define the system

$$\dot{x} = \theta_1 x + \theta_2 u, \tag{2.57}$$

where θ_1 is an unknown constant, θ_2 is a positive, unknown constant, x is the state variable, and u is the input. Furthermore, let the model be

$$\dot{x}_m = -kx_m + kx^d, \tag{2.58}$$

where x_m is the model state variable and x^d is the reference signal. Defining the error system to be $e = x - x_m$ and taking its derivative we get

$$\begin{aligned} \dot{e} &= \dot{x} - \dot{x}_m \text{ and} \\ &= \theta_1 x + \theta_2 u + k(x_m - x^d). \end{aligned}$$

Defining the Lyapunov function candidate

$$V = \frac{1}{2}e^2,$$

and taking its derivative gives

$$\dot{V} = \quad e\dot{e} = \quad e\left(\theta_1 x + \theta_2 u + k(x_m - x^d)\right).$$

If we could pick

$$u = \vartheta^T \mathbf{k},$$

where

$$\vartheta = \begin{pmatrix} -\gamma e - k(x_m - x^d) \\ -x \end{pmatrix} \text{ and } \mathbf{k} = \begin{pmatrix} \frac{1}{\theta_2} \\ \frac{\theta_1}{\theta_2} \end{pmatrix}, \tag{2.59}$$

then $\dot{V} = -\gamma e^2$ which means the system is stable. However, we do not know θ_1 or θ_2 and so we will use an estimate of $\mathbf{k}$, $\hat{\mathbf{k}}$, to get

$$u = \vartheta^T \hat{\mathbf{k}} \tag{2.60}$$

instead. Replacing u in the Lyapunov function derivative gives us

$$\dot{V} = e\left(\theta_1 x + \theta_2 \vartheta^T \hat{\mathbf{k}} + k(x_m - x^d)\right).$$

Adding and subtracting $\vartheta^T \mathbf{k}$ from $\vartheta^T \hat{\mathbf{k}}$ gives

$$\begin{aligned} \dot{V} &= e\left(\theta_1 x + \theta_2 \vartheta^T (\hat{\mathbf{k}} - \mathbf{k} + \mathbf{k}) + k(x_m - x^d)\right), \\ &= e\left(\theta_1 x + \theta_2 \vartheta^T \mathbf{k} + \theta_2 \vartheta^T \tilde{\mathbf{k}} + k(x_m - x^d)\right), \text{ where } \tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}, \text{ and} \\ &= -\gamma e^2 + \theta_2 e \vartheta^T \tilde{\mathbf{k}}, \end{aligned}$$

which is indefinite. Therefore, we define a new Lyapunov function candidate

$$V_1 = V + \frac{\theta_2}{2} \tilde{\mathbf{k}}^T \Gamma^{-1} \tilde{\mathbf{k}},$$

whose derivative is

$$\dot{V}_1 \quad = \quad -\gamma e^2 + \theta_2 e \vartheta^T \tilde{\mathbf{k}} + \theta_2 \dot{\hat{\mathbf{k}}}^T \Gamma^{-1} \tilde{\mathbf{k}} \text{ and}$$

$$= \; -\gamma e^2 + \theta_2 \left(e\vartheta^T \; + \dot{\hat{\mathbf{k}}}^T \Gamma^{-1} \right) \tilde{\mathbf{k}}.$$

To update our estimate of $\mathbf{k}$ we let

$$\dot{\hat{\mathbf{k}}} = -e\Gamma\vartheta,$$

which will cancel out the rightmost term. Using theorems defined later in this thesis, it can be shown that if $\hat{\mathbf{k}}$ is bounded, then the system is exponentially stable. This update equation is similar to the gradient based update equation from the previous section. The difference being that the gradient based equation uses a sensitivity derivative, while the Lyapunov based equation uses the regressor vector, ϑ.

There are a few advantages to this adaptive control method. One is that the Lyapunov algorithms are easy to implement. Another advantage is that the implementation does not involve large matrices with inverses. Finally, we should have a stable system since this adaptive mechanism was derived from stability theory.

This adaptive method has some disadvantages that should be considered. The first disadvantage is that it requires a good mathematical representation of the plant or the controller will be unstable. Also, noise causes this method to "unlearn" the plant parameters, reducing performance. Another issue is that parameter convergence is not guaranteed. Finally, this adaptive method causes high frequency chatter on the output signal; high frequency oscillations on the output can result in destabilization as unmodeled dynamics are excited. Moreover, there is a tradeoff between adaptive gains and oscillations on the control signal. In some cases, oscillations can be eliminated by reducing the adaptive gain, but performance and adaptability are reduced. The converse is also true. By increasing the adaptive gain, oscillations increase and the plant may become unstable.

The following are applications of Lyapunov MRACs. In [20] a morphing aircraft is presented that warps its wings by using multiple actuators. To compensate for failures of individual actuators, the authors use a Lyapunov MRAC. Reference [21] uses a Lyapunov MRAC to compensate for failures of actuator in a MIMO system.

The system presented has multiple actuators for similar functions. The article ends with an aircraft simulation controlling roll and yaw.

Like birds that fly in formation, aircraft that fly in a “V” expend less fuel in flight. Because the effects of the vortex coming off the lead aircraft are unknown, [22] derives an adaptive extremum seeking controller. The authors use an adaptive gain for neural network to control throttle and aileron movements.

In this thesis, a backstepping Lyapunov MRAC is derived. Reference [23] also uses backstepping to derive an aircraft controller. However, the authors are using a different model. The authors of this article also consider saturation in their derivation, whereas we do not.

$\mathcal{L}_1$ Controllers

In the previous section, we discussed MRACs based on Lyapunov stability criteria. Although they are guaranteed to asymptotically converge to the reference signal, the control signal may contain high frequency chatter. High frequency chatter can be detrimental as it may lead to the excitation of unmodeled dynamics, thereby causing instability. Moreover, chatter can also cause actuator failure. The hallmark of the $\mathcal{L}_1$ controller is that it has the benefits of Lyapunov MRACs but has high adaptive gains and reduced high frequency oscillations, while maintaining adequate stability margins.

In a sense, $\mathcal{L}_1$ controllers are a modification of Lyapunov based MRACs. They are called $\mathcal{L}_1$ controllers because the $\mathcal{L}_1$ gain of the system between input and output is guaranteed to be bounded. $\mathcal{L}_1$ controllers have all the benefits of traditional MRAC controllers except they can have high adaptive gains and reduced chatter on the output. This is accomplished by low-pass filtering the parameter estimates and replacing the model with a companion model [2, 3]. Notice in Figure 2.3 how the MRAC from Figure 2.2 has been modified with the addition of a low-pass filter to make it an $\mathcal{L}_1$ controller.

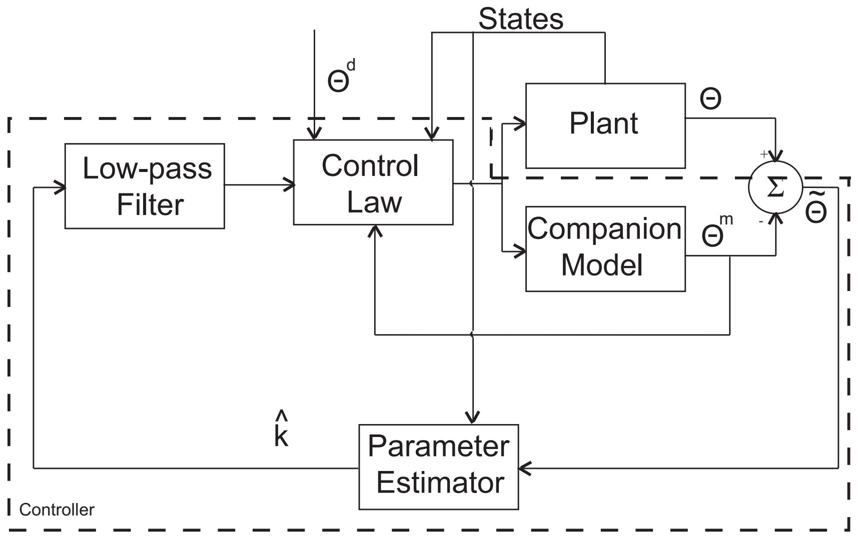


Figure 2.3: Block diagram of $\mathcal{L}_1$ controller structure. A low-pass filter is added to the parameter estimates to reduce oscillations on the output signal.

To explain companion models, let us look to Equation (2.58) of the last section. If we modify this equation we get the following companion model,

$$\dot{x}_{cm} = -kx_{cm} + \hat{\theta}_2 \left(u + \frac{kx_{cm} + \gamma e}{\hat{\theta}_2} + \frac{\hat{\theta}_1}{\hat{\theta}_2} x \right).$$

Inserting (2.60) into this model yields the original model. Thus, using companion models is equivalent to using standard models in MRAC [2]. However, this is not the case if we filter the estimates. Instead, the companion model takes on the high frequency content of the parameters. Meanwhile, the input variable u only gets the low-pass content of the parameters. This is how the $\mathcal{L}_1$ controller reduces chatter to the plant's input but still preserves all the information of the system.

Thus, to design an $\mathcal{L}_1$ controller, we begin by designing a traditional MRAC. After the MRAC is designed, the model is modified to become a companion model. Then the adaptive gains are adjusted to give fast parameter convergence. Finally, a low-pass filter is tuned for the parameter estimates until chatter is acceptable without sacrificing performance.

There are several advantages to the $\mathcal{L}_1$ adaptive controller. First, all of the benefits of Lyapunov controllers are realized. The shared benefits are a stability guarantee, simple computation without matrices, and easy implementation. Beyond this, $\mathcal{L}_1$ controllers also significantly reduce chatter that other MRAC methods produce. They also guarantee perfect tracking of the model as the adaptive gain goes to infinity [3].

Like Lyapunov based MRACs, $\mathcal{L}_1$ controllers must have good mathematical representations of the plant structure to be stable. Also, due to noise, they can unlearn parameters when given a constant reference signal. Finally, they are not guaranteed to converge to the true values of the parameters.

Chapter 3

Adaptive Control Derivation

This chapter develops two different types of algorithms for use in MAVs–Lyapunov based MRACs and $\mathcal{L}_1$ adaptive controllers. This chapter is divided into two sections. The first develops four Lyapunov based MRAC pitch and roll controllers, while the second develops four $\mathcal{L}_1$ pitch controllers.

3.1 Lyapunov Based MRAC Controllers

In this section, four different MRAC schemes are derived. All four schemes are based on Lyapunov stability arguments and use the same reference pitch and roll models. The pitch model is

$$\dot{\theta}_m = k_\theta \left(\theta^d - \theta_m\right), \tag{3.1}$$

and the roll model is

$$\dot{\phi}_m = k_\phi \left(\phi^d - \phi_m\right), \tag{3.2}$$

where θ^d and ϕ^d are the desired pitch and roll, θ_m and ϕ_m are the model pitch and roll, and k_θ and k_ϕ are the model gains for pitch and roll respectively. However, the schemes differ on their approach to how much is lumped into the regressor's bias term. "Lumping" the parameters will give us computationally simpler algorithms that free more processor time for other essential functions. The first scheme will employ backstepping to handle derivatives on angular rates. The second does not use backstepping, but instead solves for q and p, reducing the roll and pitch equations to pseudo first order systems. The third and fourth schemes further simplify the system by lumping more signals into the bias term.

All four schemes will use the following definition for tracking error. For pitch we have

$$\begin{aligned}\tilde{\theta} &= \theta - \theta_m \text{ and} \\ \dot{\tilde{\theta}} &= \dot{\theta} - \dot{\theta}_m, \end{aligned} \tag{3.3}$$

and for roll we have

$$\tilde{\phi} = \phi - \phi_m \text{ and} \tag{3.4}$$

$$\dot{\tilde{\phi}} = \dot{\phi} - \dot{\phi}_m. \tag{3.5}$$

3.1.1 MRAC Scheme A

Two backstepping controllers will be developed in this section. The first controller regulates pitch, while the second controller regulates roll. The boundedness of both controllers will be proven.

Pitch Attitude Hold

Our objective is to have the plant track the model (3.1). Because δ_e comes into the system through the q channel, the tracking error, $\tilde{\theta}$, is driven to zero by using backstepping. To derive the pitch controller, Equation (3.3) is applied to Equation (2.13) giving

$$\begin{aligned}\dot{\tilde{\theta}} &= q\cos\phi - r\sin\phi - k_\theta(\theta^d - \theta_m), \\ &= -\lambda_\theta\tilde{\theta} + \lambda_\theta\tilde{\theta} - k_\theta(\theta^d - \theta_m) - r\sin\phi + q\cos\phi, \\ &= -\lambda_\theta\tilde{\theta} + \cos\phi\left\{\frac{\lambda_\theta\tilde{\theta} - k_\theta(\theta^d - \theta_m) - r\sin\phi}{\cos\phi} + q\right\}, \text{ and} \\ &= -\lambda_\theta\tilde{\theta} + \tilde{q}\cos\phi, \end{aligned} \tag{3.6}$$

where $\tilde{q} = q - q^{des}$ and the backstepping variable is defined as

$$q^{des} \triangleq \frac{-\lambda_\theta \tilde{\theta} + k_\theta(\theta^d - \theta_m) + r\sin\phi}{\cos\phi}. \tag{3.7}$$

The variable, q^{des}, is chosen such that if $q = q^{des}$ Equation (3.6) reduces to

$$\dot{\tilde{\theta}} = -\lambda_\theta \tilde{\theta},$$

which is asymptotically stable.

Differentiating $\tilde{q}$ and use Assumption A4 from section 2.2 and Equation (2.16) yields

$$\dot{\tilde{q}} = \frac{J_{xz}}{J_y}(p^2 - r^2) + \frac{J_z - J_x}{J_y}pr + \frac{\frac{1}{2}\rho V^2 S\bar{c}}{J_y}\Big[C_{m_0} + C_{m_\alpha}(\theta - \gamma) + C_{m_q}\frac{\bar{c}q}{V} + C_{m_{\delta_e}}\delta_e\Big] - \dot{q}^{des}.$$

Taking the Lyapunov function candidate as

$$\mathcal{V}_1 = \frac{1}{2}\tilde{\theta}^2 + \frac{1}{2}\tilde{q}^2$$

and differentiating, gives

$$\begin{aligned}
\dot{\mathcal{V}}_1 &= -\lambda_\theta\tilde{\theta}^2 + \tilde{q}\Big(\tilde{\theta}\cos\phi + \dot{\tilde{q}}\Big),\\
&= -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \tilde{q}\Big(\lambda_q\tilde{q} + \tilde{\theta}\cos\phi + \dot{\tilde{q}}\Big),\\
&= -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)V^2\tilde{q}\Bigg\{\left(\frac{\lambda_q\tilde{q} + \tilde{\theta}\cos\phi - \dot{q}^{des}}{V^2}\right)\left(\frac{J_y}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right)\\
&\quad + \left(\frac{r^2 - p^2}{V^2}\right)\left(\frac{J_{xz}}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right) + \left(\frac{pr}{V^2}\right)\left(\frac{J_z - J_x}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right) + (1)\left(\frac{C_{m_0} - C_{m_\alpha}\gamma}{C_{m_{\delta_e}}}\right)\\
&\quad + (\theta)\left(\frac{C_{m_\alpha}}{C_{m_{\delta_e}}}\right) + \left(\frac{q}{V}\right)\left(\frac{C_{m_q}\bar{c}}{C_{m_{\delta_e}}}\right) + \delta_e\Bigg\}, \text{ and}\\
&= -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)V^2\tilde{q}\Big\{\delta_e - \vartheta^T\mathbf{k}\Big\},
\end{aligned}$$

where

$$\vartheta \triangleq \begin{pmatrix} \frac{-\lambda_q\tilde{q}-\tilde{\theta}\cos\phi+\dot{q}^{des}}{V^2} \\ \frac{p^2-r^2}{V^2} \\ -\frac{pr}{V^2} \\ -1 \\ -\theta \\ -\frac{q}{V} \end{pmatrix} \quad \text{and} \quad \mathbf{k} \triangleq \begin{pmatrix} k_1 = \left(\frac{J_y}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right) \\ k_2 = \left(\frac{J_{xz}}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right) \\ k_3 = \left(\frac{J_z-J_x}{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}\right) \\ k_4 = \left(\frac{C_{m_0}-C_{m_\alpha}\gamma}{C_{m_{\delta_e}}}\right) \\ k_5 = \left(\frac{C_{m_\alpha}}{C_{m_{\delta_e}}}\right) \\ k_6 = \left(\frac{C_{m_q}\bar{c}}{C_{m_{\delta_e}}}\right) \end{pmatrix}. \tag{3.8}$$

If we knew the vector $\mathbf{k}$, setting $\delta_e = \vartheta^T\mathbf{k}$ yields

$$\dot{\mathcal{V}} = -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2.$$

According to Theorem 2.3.2, asymptotic convergence of both $\tilde{\theta}$ and $\tilde{q}$ is guaranteed as long as ϑ is bounded. However, we do not know $\mathbf{k}$ and so instead use

$$\delta_e = \vartheta^T\hat{\mathbf{k}}, \tag{3.9}$$

where $\hat{\mathbf{k}}$ is the adaptive estimate of $\mathbf{k}$ and define $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$ to get

$$\dot{\mathcal{V}}_1 = -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)V^2\tilde{q}\vartheta^T\tilde{\mathbf{k}}.$$

Modifying the Lyapunov function candidate as

$$\mathcal{V} = \frac{1}{2}\tilde{\theta}^2 + \frac{1}{2}\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)\frac{1}{2}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}},$$

differentiation gives

$$\begin{aligned} \dot{\mathcal{V}} &= -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)\tilde{\mathbf{k}}^T\left[\Lambda^{-1}\dot{\tilde{\mathbf{k}}} + V^2\tilde{q}\vartheta\right] \text{ and} \\ &= -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)\tilde{\mathbf{k}}^T\left[\Lambda^{-1}\dot{\hat{\mathbf{k}}} - \Lambda^{-1}\dot{\mathbf{k}} + V^2\tilde{q}\vartheta\right]. \end{aligned} \tag{3.10}$$

The update equation for $\hat{\mathbf{k}}$ is selected as

$$\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_1}\left(-\Lambda V^2 \tilde{q}\vartheta, \hat{\mathbf{k}}\right). \tag{3.11}$$

Theorem 3.1.1 If

H1: $\theta^d, p, r \in \mathcal{L}_\infty$ and are continuous signals,

H2: $V \in [V_{min}, V_{max}]$,

H3: $\phi \in [-\bar{\phi}, \bar{\phi}]$ where $\bar{\phi} < \pi/2$ and ϕ is continuous,

H4: $|\dot{k}_4| \leq n$,

H5: $\mathbf{k} \in S_1$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.1), (3.7), (3.8), (3.9), and (3.11) is bounded and $\tilde{\theta}$ and $\tilde{q}$ are bounded such that

$$\lim_{t\to\infty} |\tilde{\theta}^2 + \tilde{q}^2| \leq \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{\lambda_{min}\lambda_4 J_y}\right) mn,$$

whose bound is tightened by increasing the adaptive gain, λ_4, or the smaller of the two control gains, λ_θ and λ_q.

Proof: We need to show that all states are bounded. If Hypothesis **H5** and Lemma 2.3.7 are applied to Equations (3.10) and (3.11), we get

$$\dot{\mathcal{V}} \leq -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 - \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\mathbf{k}}.$$

However, since $\frac{C_{m_0} - C_{m_\alpha}\gamma}{C_{m_{\delta_e}}}$ is the only term in $\mathbf{k}$ that is time varying, the above equation reduces to

$$\dot{\mathcal{V}} \leq -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 - \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{J_y}\right)\tilde{k}_4\lambda_4^{-1}\dot{k}_4.$$

Applying Hypotheses **H4** and **H5** gives

$$\dot{\mathcal{V}} \leq -\lambda_\theta\tilde{\theta}^2 - \lambda_q\tilde{q}^2 + \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{\lambda_4 J_y}\right) mn,$$

where M is the upper bound for $|\tilde{k}_4|$. Therefore, the Lyapunov function derivative is negative definite as long as $|\lambda_\theta \tilde{\theta}^2 + \lambda_q \tilde{q}^2| \geq \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{\lambda_4 J_y}\right) mn$. Defining $\lambda_{min} \triangleq \min(\lambda_\theta, \lambda_q)$ and applying Theorem 2.3.6, we have the upper bound for $|\tilde{\theta}^2 + \tilde{q}^2|$ as follows:

$$\begin{aligned} \lambda_{min}|\tilde{\theta}^2 + \tilde{q}^2| & \leq |\lambda_\theta \tilde{\theta}^2 + \lambda_q \tilde{q}^2| \leq \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{\lambda_4 J_y}\right) mn \text{ and} \\ \Rightarrow \qquad & |\tilde{\theta}^2 + \tilde{q}^2| \leq \left(\frac{\frac{1}{2}\rho S\bar{c}C_{m_{\delta_e}}}{\lambda_{min}\lambda_4 J_y}\right) mn. \end{aligned} \tag{3.12}$$

Therefore, either increasing the adaptive gain λ_4 or the control gain λ_{min} reduces the tracking error. However, the control signal acquires high frequency oscillations as these gains are increased. High frequency content in the control signal may negatively impact the performance by exciting unmodeled dynamics or damaging actuators.

Equation (3.12) implies that

$$\tilde{\theta}, \tilde{q} \in \mathcal{L}_\infty. \tag{3.13}$$

Because **H1** states that $\theta^d \in \mathcal{L}_\infty$,

$$\theta_m \in \mathcal{L}_\infty, \tag{3.14}$$

since the model is BIBO stable. Hypotheses **H1**, **H2**, **H3**, **H5**, and Equation (3.14) imply that

$$q^{des}, \dot{q}^{des} \in \mathcal{L}_\infty. \tag{3.15}$$

Equations (3.13) and (3.15) imply that $q \in \mathcal{L}_\infty$. Equations (3.6), (3.14), (3.15), and (3.13) imply that

$$\dot{\tilde{\theta}}, \dot{\tilde{q}} \in \mathcal{L}_\infty. \tag{3.16}$$

Furthermore, Equations (3.3), (3.13), and (3.14) imply that $\theta \in \mathcal{L}_\infty$. Finally, applying this result, **H1**, **H2**, and **H3** to (3.8) implies that $\vartheta \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 1 Backstepping MRAC Scheme A for pitch attitude hold.

```
1: Obtain $V$ $\phi$, $p$, $r$, $\alpha$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\theta}_m = k_\theta\left(\theta^d - \theta_m\right)$.
3: Compute: $\dot{q}^{des} = \frac{-\lambda_\theta\tilde{\theta}+k_\theta(\theta^d-\theta_m)+r\sin\phi}{\cos\phi}$.
4: Compute: $\vartheta = \begin{pmatrix} \frac{-\lambda_q\tilde{q}-\tilde{\theta}\cos\phi+\dot{q}^{des}}{V^2} \\ -\frac{p^2-r^2}{V^2} \\ -\frac{pr}{V^2} \\ -1 \\ -\theta \\ -\frac{q}{V} \end{pmatrix}$.
5: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \text{Proj}\left(-\Lambda V^2\tilde{q}\vartheta, \hat{\mathbf{k}}\right)$.
6: Compute the elevator command: $\delta_e = \vartheta^T\mathbf{k}$.
```

Roll Attitude Hold

The desire is for the MAV to track the roll model (3.2). However, the control input comes in through the p channel and so backstepping is employed. The backstepping roll controller is derived by differentiating (3.5) to get

$$\begin{aligned}\dot{\tilde{\phi}} &= p + q\sin\phi\tan\theta + r\cos\phi\tan\theta - k_\theta(\theta^d - \theta_m), \\ &= -\lambda_\phi\tilde{\phi} + \Big\{p + \lambda_\phi\tilde{\phi} + q\sin\phi\tan\theta \\ &\qquad + r\cos\phi\tan\theta - k_\phi(\phi^d - \phi_m)\Big\}, \text{ and} \\ &= -\lambda_\phi\tilde{\phi} + \tilde{p}, \end{aligned} \tag{3.17}$$

where $\tilde{p} = p - p^{des}$ and the backstepping variable is

$$p^{des} \triangleq -\lambda_\phi\tilde{\phi} - q\sin\phi\tan\theta - r\cos\phi\tan\theta + k_\phi(\phi^d - \phi_m) \tag{3.18}$$

Differentiating $\tilde{p}$ and substituting (2.15) in for $\dot{p}$ results in

$$\begin{aligned}\dot{\tilde{p}} = \hat{J}_1pq - \hat{J}_2qr + \frac{1}{2}\rho V^2 S\frac{b}{2}\Big[&C_{p_0} + C_{p_\beta}\beta + C_{p_p}\frac{bp}{2V} \\ &+ C_{p_r}\frac{br}{2V} + C_{p_{\delta_a}}\delta_a + C_{p_{\delta_r}}\delta_r\Big] - \dot{p}^{des}.\end{aligned} \tag{3.19}$$

Notice that although δ_r is a control input, it is being treated like a measured state–with the assumption being that the rudder to the MAV (if present) is used to regulate sideslip. The experimental MAV platforms used in this thesis lack a rudder and therefore $\delta_r = 0$. As a matter of implementation, this value should be obtained from a sensor. If a sensor is not present to determine the current rudder position, previously commanded rudder values should be used. Although less optimal than sensor values, using previous rudder commands is preferred to rudder commands not yet sent to the rudder actuators. Non-delayed commanded values may incorrectly weight the rudder during adaptation since the states that are directly effected by the rudder, namely roll and yaw, will not correlate with the rudder value being used for adaptation. This problem may or may not cause instability, but performance will probably be effected.

In order to determine a control law for δ_a, the Lyapunov function candidate

$$\mathcal{V}_1 = \frac{1}{2}\tilde{\phi}^2 + \frac{1}{2}\tilde{p}^2$$

is differentiated to obtain

$$\begin{aligned}\dot{\mathcal{V}}_1 &= -\lambda_\phi\tilde{\phi}^2 + \tilde{p}\left(\tilde{\phi} + \dot{\tilde{p}}\right) \text{ and}\\ &= -\lambda_\phi\tilde{\phi}^2 - \lambda_p\tilde{p}^2 + \tilde{p}\left(\tilde{\phi} + \lambda_p\tilde{p} + \dot{\tilde{p}}\right).\end{aligned}$$

This derivative in its current form is indefinite and therefore we cannot say anything about the stability of the system. However, by substituting Equation (3.19) in place of $\dot{\tilde{p}}$ yields the following derivative,

$$\begin{aligned}\dot{\mathcal{V}}_1 = -\lambda_\phi\tilde{\theta}^2 - \lambda_p\tilde{q}^2 + \left(\frac{1}{2}\rho S\frac{b}{2}C_{p_{\delta_a}}\right)V^2\tilde{p}\Bigg(&\left(\frac{\lambda_p\tilde{p} + \tilde{\phi} - \dot{p}^{des}}{V^2}\right)\left(\frac{1}{\frac{1}{2}\rho S\frac{b}{2}C_{p_{\delta_a}}}\right)\\ &+ \left(\frac{pq}{V^2}\right)\left(\frac{\hat{J}_1}{\frac{1}{2}\rho S\frac{b}{2}C_{p_{\delta_a}}}\right) + \left(\frac{-qr}{V^2}\right)\left(\frac{\hat{J}_2}{\frac{1}{2}\rho S\frac{b}{2}C_{p_{\delta_a}}}\right) + (1)\left(\frac{C_{p_0} + C_{p_\beta}\vartheta}{C_{p_{\delta_a}}}\right)\\ &+ (-\chi)\left(\frac{C_{p_\beta}}{C_{p_{\delta_a}}}\right) + \left(\frac{p}{V}\right)\left(\frac{C_{p_p}\frac{b}{2}}{C_{p_{\delta_a}}}\right) + \left(\frac{r}{V}\right)\left(\frac{C_{p_r}\frac{b}{2}}{C_{p_{\delta_a}}}\right) + (\delta_r)\left(\frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\right) + \delta_a\Bigg),\end{aligned}$$

$$\dot{\mathcal{V}}_1 = -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 + \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) V^2 \tilde{p}\left(\delta_a - \vartheta^T \mathbf{k}\right),$$

where

$$\vartheta \triangleq \begin{pmatrix} \frac{-\lambda_p \tilde{p} - \tilde{\phi} + \dot{p}^{des}}{V^2} \\ \frac{-pq}{V^2} \\ \frac{qr}{V^2} \\ -1 \\ \chi \\ \frac{-p}{V} \\ \frac{-r}{V} \\ -\delta_r \end{pmatrix} \text{ and } \mathbf{k} \triangleq \begin{pmatrix} k_1 = \left(\frac{1}{\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}}\right) \\ k_2 = \left(\frac{\hat{J}_1}{\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}}\right) \\ k_3 = \left(\frac{\hat{J}_2}{\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}}\right) \\ k_4 = \left(\frac{C_{p_0} + C_{p_\beta}\vartheta}{C_{p_{\delta_a}}}\right) \\ k_5 = \left(\frac{C_{p_\beta}}{C_{p_{\delta_a}}}\right) \\ k_6 = \left(\frac{C_{p_p}\frac{b}{2}}{C_{p_{\delta_a}}}\right) \\ k_7 = \left(\frac{C_{p_r}\frac{b}{2}}{C_{p_{\delta_a}}}\right) \\ k_8 = \left(\frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\right) \end{pmatrix}. \tag{3.20}$$

Since we do not know $\mathbf{k}$, let $\hat{\mathbf{k}}$ be the estimate of $\mathbf{k}$ and set

$$\delta_a = \vartheta^T \hat{\mathbf{k}}. \tag{3.21}$$

Substituting $\vartheta^T \hat{\mathbf{k}}$ in for δ_a gives us

$$\dot{\mathcal{V}}_1 = -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 + \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) V^2 \tilde{p} \vartheta^T \tilde{\mathbf{k}},$$

where we have used the relation, $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$. Modifying the Lyapunov function candidate to be

$$\mathcal{V} = \frac{1}{2}\tilde{\phi}^2 + \frac{1}{2}\tilde{p}^2 + \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) \frac{1}{2} \tilde{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}},$$

differentiation gives

$$\dot{\mathcal{V}} = -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 + \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) \tilde{\mathbf{k}}^T \left[\Lambda^{-1}\dot{\hat{\mathbf{k}}} - \Lambda^{-1}\dot{\mathbf{k}} + V^2 \tilde{p} \vartheta\right]. \tag{3.22}$$

Define the update equation

$$\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_1}\left(-\Lambda V^2 \tilde{p} \vartheta, \hat{\mathbf{k}}\right). \tag{3.23}$$

Theorem 3.1.2 If

H1: $\phi^d, q, r \in \mathcal{L}_\infty$ are continuous signals,

H2: $V \in [V_{min}, V_{max}]$,

H3: $\theta \in [-\bar{\theta}, \bar{\theta}]$ where $\bar{\theta} < \pi/2$ and θ is continuous,

H4: $|k_4| \le n$,

H5: $\mathbf{k} \in S_1$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.5),(3.18), (3.20), (3.21), and (3.23) is stable with ultimate boundedness

$$\lim_{t\to\infty} |\tilde{\phi}^2 + \tilde{p}^2| \le \left(\frac{1}{2}\rho S \frac{b}{2\lambda_{min}\lambda_4} C_{p_{\delta_a}}\right) mn.$$

Proof: We need to show all that states are bounded. Applying Hypothesis **H5** and Lemma 2.3.7 to Equations (3.22) and (3.23), we get

$$\dot{\mathcal{V}} \le -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 - \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) \tilde{\mathbf{k}}^T \Lambda^{-1} \dot{\mathbf{k}}. \tag{3.24}$$

However, since k_4 is the only term in $\mathbf{k}$ that is time varying, the above equation reduces to

$$\dot{\mathcal{V}} \le -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 - \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) \tilde{k}_4 \lambda_4^{-1} \dot{k}_4.$$

Applying Hypotheses **H4** and **H5** gives

$$\dot{\mathcal{V}} \le -\lambda_\phi \tilde{\phi}^2 - \lambda_p \tilde{p}^2 - \left(\frac{1}{2}\rho S \frac{b C_{p_{\delta_a}}}{2\lambda_4}\right) mn.$$

where m is the upper bound for $|\tilde{k}_4|$. Therefore, the Lyapunov function derivative is negative definite as long as $|\lambda_\phi \tilde{\phi}^2 + \lambda_p \tilde{p}^2| \ge \left(\frac{1}{2}\rho S \frac{b}{2} C_{p_{\delta_a}}\right) mn$. If we define $\lambda_{min} \triangleq \min(\lambda_\phi, \lambda_p)$, and apply Theorem 2.3.6, we have the upper bound for $|\tilde{\phi}^2 + \tilde{p}^2|$ as follows:

$$\begin{aligned} \lambda_{min}|\tilde{\phi}^2 + \tilde{p}^2| \leq \quad & |\lambda_\phi \tilde{\phi}^2 + \lambda_p \tilde{p}^2| \leq \left(\frac{1}{2}\rho S \frac{b}{2\lambda_4} C_{p_{\delta_a}}\right) mn \text{ and} \\ \Rightarrow \qquad & |\tilde{\phi}^2 + \tilde{p}^2| \leq \left(\frac{1}{2}\rho S \frac{b}{2\lambda_{min}\lambda_4} C_{p_{\delta_a}}\right) mn. \end{aligned} \tag{3.25}$$

Therefore, either increasing the adaptive gain λ_4 or the control gain λ_{min} reduces the tracking error. However, the control signal gains high frequency oscillations as these gains are increased. High frequency content in the control signal may negatively impact the performance by exciting unmodeled dynamics or damaging actuators. Also, this upper bound further implies that

$$\tilde{\phi}, \tilde{p} \in \mathcal{L}_\infty. \tag{3.26}$$

Because **H1** states that $\phi^d \in \mathcal{L}_\infty$,

$$\phi_m \in \mathcal{L}_\infty, \tag{3.27}$$

since the model is BIBO stable. All Hypotheses, (3.27), and (3.26) applied to (3.18) imply that

$$p^{des}, \dot{p}^{des} \in \mathcal{L}_\infty. \tag{3.28}$$

Equations (3.26) and (3.28) imply that $p \in \mathcal{L}_\infty$. Using (3.17), (3.26), (3.27), and (3.28) implies that

$$\dot{\tilde{\phi}}, \dot{\tilde{p}} \in \mathcal{L}_\infty. \tag{3.29}$$

Furthermore, Equations (3.5), (3.26), and (3.27) imply that $\phi \in \mathcal{L}_\infty$. The previous result used with the Hypotheses imply that $\vartheta \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

3.1.2 MRAC Scheme B

Two MRAC controllers will be developed in this section. Unlike the previous section, backstepping is not used to create the two controllers. Instead, both roll and

Algorithm 2 MRAC Scheme B for roll attitude hold.

```
1: Obtain $V$, $\phi$, $p$, $r$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\phi}_m = k_\phi\left(\phi^d - \phi_m\right)$.
3: Compute: $\dot{p}^{des} = -\lambda_\phi\tilde{\phi} - q\sin\phi\tan\theta - r\cos\phi\tan\theta + k_\phi(\phi^d - \phi_m)$.
4: Compute: $\vartheta = \begin{pmatrix} \frac{-\lambda_p\tilde{p} - \tilde{\phi} + \dot{p}^{des}}{V^2} \\ -\frac{pq}{V^2} \\ -\frac{qr}{V^2} \\ -1 \\ \chi \\ -\frac{p}{V} \\ -\frac{r}{V} \\ -\delta_r \end{pmatrix}$.
5: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_1}\left(-\Lambda\tilde{p}V\vartheta, \hat{\mathbf{k}}\right)$.
6: Compute the aileron command: $\delta_a = \vartheta^T\hat{\mathbf{k}}$.
```

pitch equations will be reduced to pseudo first-order systems where the roll and pitch rate derivatives are treated as measured states. The first controller regulates pitch, while the second controller regulates roll. The boundedness of both controllers will be proven.

Pitch Attitude Hold

MRAC Scheme B uses the same definition of tracking error for pitch as MRAC A but does not employ backstepping. Instead, solving for q in Equation (2.16) and apply it to Equation (3.5) to reduces the plant from a second order system to a pseudo first-order system. Again, the desire is to pick δ_e such that for $\lambda_\theta > 0$ we have

$$\dot{\tilde{\theta}} = -\lambda_\theta\tilde{\theta} \tag{3.30}$$

Solving for q yields

$$\begin{aligned} q =& \frac{V}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - \left(I_z - I_x\right)pr]\right. \\ & \left. - C_{m_0} + C_{m_\alpha}\gamma - C_{m_\alpha}\theta - C_{\delta_e}\delta_e\right). \end{aligned} \tag{3.31}$$

$\dot{q}$ is included in this equation as a measurable state, since q is known. If the equation for q is plugged this into equation (3.30) the result is

$$\begin{aligned}\dot{\tilde{\theta}} =& \frac{V\cos\phi}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - (J_z - J_x)\,pr]\right.\\ &\left. - C_{m_0} + C_{m_\alpha}\gamma - C_{m_\alpha}\theta - C_{m_{\delta_e}}\delta_e\right) - r\sin\phi - \dot{\theta}_m.\end{aligned} \tag{3.32}$$

If the parameters were known we could set δ_e to be

$$\delta_e^{des} = \vartheta^T\mathbf{k},$$

where

$$\vartheta = \begin{pmatrix} \frac{\lambda_\theta\tilde{\theta} - k_\theta(\theta^d - \theta_m) - r\sin\phi}{V\cos\phi} \\ \dot{q} \\ 1 \\ p^2 - r^2 \\ -pr \\ -\theta \end{pmatrix} \quad \mathbf{k} = \begin{pmatrix} \left(k_1 = \frac{\bar{c}C_{m_p}}{C_{m_{\delta_e}}}\right) \\ \left(k_2 = \frac{J_y}{\frac{1}{2}S\bar{c}C_{m_{\delta_e}}}\right) \\ \left(k_3 = -\frac{C_{m_o}}{C_{m_{\delta_e}}} + \frac{C_{m_\alpha}}{C_{m_{\delta_e}}}\gamma\right) \\ \left(k_4 = \frac{J_{xz}}{\frac{1}{2}V^2 S\bar{c}C_{m_{\delta_e}}}\right) \\ \left(k_5 = \frac{J_z - J_x}{\frac{1}{2}V^2 S\bar{c}C_{m_{\delta_e}}}\right) \\ \left(k_6 = \frac{C_{m_\alpha}}{C_{m_{\delta_e}}}\right) \end{pmatrix}. \tag{3.33}$$

Direct substitution of δ_e^{des} for δ_e in Equation (3.32) would yield (3.30). However, since $\mathbf{k}$ is not known an estimate, $\hat{\mathbf{k}}$, is used instead such that

$$\delta_e = \vartheta^T\hat{\mathbf{k}}. \tag{3.34}$$

By adding and subtracting δ_e^{des} in equation (3.32), we get

$$\begin{aligned}\dot{\tilde{\theta}} =& \frac{V\cos\phi}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}\left[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - (J_z - J_x)\,pr\right] - C_{m_0}\right.\\ &\left. + C_{m_\alpha}\gamma - C_{m_\alpha}\theta - C_{m_{\delta_e}}\left(\delta_e^{des} + \delta_e - \delta_e^{des}\right)\right) - r\sin\phi - \dot{\theta}_m \text{ and}\\ =& -\lambda_\theta\tilde{\theta} - \frac{V\cos\phi C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\vartheta^T\tilde{\mathbf{k}},\end{aligned} \tag{3.35}$$

where $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$.

Defining the Lyapunov function candidate

$$V = \frac{1}{2}\tilde{\theta}^2 - \frac{C_{m_{\delta_e}}}{2C_{m_q}\bar{c}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}},$$

(C_{m_q} is assumed to be negative) and taking its derivative we get

$$\begin{aligned}\dot{V} =& \tilde{\theta}\dot{\tilde{\theta}} - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\tilde{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}},\\ =& -\lambda_\theta\tilde{\theta}^2 - \frac{\tilde{\theta}V\cos\phi}{C_{m_q}C_{m_{\delta_e}}\bar{c}}\vartheta^T\tilde{\mathbf{k}} - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}} + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \text{ and}\\ =& -\lambda_\theta\tilde{\theta}^2 - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\left(\tilde{\theta}V\cos\phi\vartheta^T + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\right)\tilde{\mathbf{k}} + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}.\end{aligned} \tag{3.36}$$

We define the update law as

$$\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_3}\left(-\Lambda\tilde{\theta}V\cos\phi\vartheta, \hat{\mathbf{k}}\right). \tag{3.37}$$

Theorem 3.1.3 If

H7: $\theta^d, p, r, \dot{q} \in \mathcal{L}_\infty$ are bounded signals,

H8: $V \in [V_{min}, V_{max}]$,

H9: $\phi \in [-\bar{\phi}, \bar{\phi}]$ where $\bar{\phi} < \pi/2$ and ϕ is continuous,

H10: $|k_3| \leq n$,

H11: $\mathbf{k} \in S_3$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.1),(3.33), (3.34), and (3.37) is stable and $\tilde{\phi}$ is ultimately bounded by

$$\lim_{t\to\infty}|\tilde{\theta}| \leq \sqrt{\left|\frac{C_{m_{\delta_e}}}{\lambda_\theta\lambda_3 C_{m_q}\bar{c}}\right| mn}. \tag{3.38}$$

Proof: We need to show all that states are bounded. Applying Equations (3.36) and

(3.37), Hypothesis **H10**, and Lemma 2.3.7 we get

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}} \dot{\tilde{\mathbf{k}}}^T \Lambda^{-1} \tilde{\mathbf{k}}.$$

However, as k_3 is the only parameter that is time-varying in $\mathbf{k}$, the above equation reduces to

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}} \dot{\tilde{k}}_3^T \lambda_3^{-1} \tilde{k}_3. \tag{3.39}$$

Invoking Hypotheses **H10** and **H11** gives

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \left| \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}} \right| \lambda_3^{-1} mn,$$

where m is the upper bound for $|k_3|$. The Lyapunov function derivative is negative definite as long as -$\lambda_\theta \tilde{\theta}^2$ dominates the rightmost term. Thus, applying Theorem 2.3.6 we determine that the bound on $\tilde{\theta}$ is

$$\begin{aligned} \lambda_\theta \tilde{\theta}^2 &\leq \left| \frac{C_{m_{\delta_e}}}{\lambda_3 C_{m_q}\bar{c}} \right| mn \text{ and} \\ \Rightarrow |\tilde{\theta}| &\leq \sqrt{\left| \frac{C_{m_{\delta_e}}}{\lambda_\theta \lambda_3 C_{m_q}\bar{c}} \right| mn} \end{aligned} \tag{3.40}$$

Therefore, increasing either the adaptive gain, λ_3 or the control gain, λ_θ reduces the tracking error. Furthermore, Equation (3.40) implies that

$$\tilde{\theta} \in \mathcal{L}_\infty. \tag{3.41}$$

Hypothesis **H7** states that $\theta^d \in \mathcal{L}_\infty$, therefore

$$\theta_m \in \mathcal{L}_\infty, \tag{3.42}$$

since the model is BIBO stable. Equations (3.3), (3.41), (3.42) imply that $\theta \in \mathcal{L}_\infty$. Finally, using (3.42), (3.41), (3.32), all the Hypothesis, (3.42), and (3.32) implies that $\vartheta, \dot{\tilde{\theta}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 3 MRAC Scheme B pitch attitude hold.

```
1: Obtain $\phi$, $p$, $r$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\theta}_m = k_\phi\left(\theta^d - \theta_m\right)$.
3: Compute:
   $\vartheta = \begin{pmatrix} \frac{\lambda_\theta\tilde{\theta} - k_\phi(\theta^d - \theta_m) - r\sin\phi}{V\cos\phi} & \dot{q} & 1 & p^2 - r^2 & -pr & -\theta \end{pmatrix}^T$.
4: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_3}\left(-\Lambda\tilde{\theta}V\cos\phi\vartheta, \hat{\mathbf{k}}\right)$.
5: Compute the elevator command: $\delta_e = \vartheta^T\hat{\mathbf{k}}$.
```

Roll Attitude Hold

An MRAC roll controller will be developed without using backstepping in this section. The second order roll equations are reduced to a pseudo first-order system to avoid backstepping. Therefore, solving for p in Equation (2.15) yields

$$\begin{aligned} p = & \frac{8}{\rho V S b^2 C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho V S b^2 C_{p_p}}pq + \frac{8\Lambda_2}{\rho V S b^2 C_{p_p}}qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta \\ & - \frac{C_{p_r}}{C_{p_p}}r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\delta_a - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r. \end{aligned} \tag{3.43}$$

Substituting this equation in equation (3.5) give us

$$\begin{aligned} \dot{\tilde{\phi}} = & \frac{8}{\rho V S b^2 C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho V S b^2 C_{p_p}}pq + \frac{8\Lambda_2}{\rho V S b^2 C_{p_p}}qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta \\ & - \frac{C_{p_r}}{C_{p_p}}r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\delta_a - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m. \end{aligned} \tag{3.44}$$

But since the hope is for $\dot{\tilde{\phi}} = -\lambda_\phi\tilde{\phi}$, the desired aileron response is set to be

$$\begin{aligned} \delta_a^{des} = & \frac{bC_{p_p}}{2VC_{p_{\delta_a}}}\left(\lambda_\phi\dot{\tilde{\phi}} + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m\right) - \frac{C_{p_0}}{C_{p_{\delta_a}}} - \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\beta \\ & - \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\delta_r - \frac{C_{p_r}}{2VC_{p_{\delta_a}}}r + \frac{4}{\rho V^2 S b}\dot{p} - \frac{4\Lambda_1}{\rho V^2 S b}pq + \frac{4\Lambda_2}{\rho V^2 S b}qr \text{ and} \end{aligned} \tag{3.45}$$

$$\begin{aligned} = & \frac{bC_{p_p}}{2VC_{p_{\delta_a}}}\left(\lambda_\phi\dot{\tilde{\phi}} + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m\right) - \frac{C_{p_0}}{C_{p_{\delta_a}}} - \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\chi \\ & + \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\psi - \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\delta_r - \frac{C_{p_r}}{2VC_{p_{\delta_a}}}r + \frac{4}{\rho V^2 S b}\dot{p} - \frac{4\Lambda_1}{\rho V^2 S b}pq + \frac{4\Lambda_2}{\rho V^2 S b}qr, \end{aligned} \tag{3.46}$$

where the assumption that $\beta \approx \chi - \psi$ has been employed. Grouping the knowns and unknowns together reduces the above equation to $\delta_a^{des} = \vartheta^T \mathbf{k}$, where

$$\vartheta = \begin{pmatrix} \frac{\lambda_\phi \tilde{\phi} + q\sin\theta\tan\theta + r\cos\theta\tan\theta - k_\phi\left(\phi^d - \phi_m\right)}{V} \\ \frac{\dot{p}}{V^2} \\ -\frac{pq}{V^2} \\ \frac{qr}{V^2} \\ -1 \\ -\chi \\ -\frac{r}{V} \\ -\delta_r \end{pmatrix} \text{ and } \mathbf{k} = \begin{pmatrix} \left(k_1 = \frac{bC_{p_p}}{2C_{p_{\delta_a}}}\right) \\ \left(k_2 = \frac{4}{\rho S b}\right) \\ \left(k_3 = \frac{4\Lambda_1}{\rho S b}\right) \\ \left(k_4 = \frac{4\Lambda_2}{\rho S b}\right) \\ \left(k_5 = \psi\frac{C_{p_\beta}}{C_{p_{\delta_a}}} + \frac{C_{p_0}}{C_{p_{\delta_a}}}\right) \\ \left(k_6 = \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\right) \\ \left(k_7 = \frac{C_{p_r}}{2C_{p_{\delta_a}}}\right) \\ \left(k_8 = \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\right) \end{pmatrix}. \tag{3.47}$$

However, $\mathbf{k}$ is unknown so the aileron command is defined as

$$\delta_a = \vartheta^T \hat{\mathbf{k}}, \tag{3.48}$$

where $\hat{\mathbf{k}}$ is an estimate of the parameters. Using the above definition for δ_a and adding and subtracting δ_a^{des} from Equation (3.44) gives us

$$\begin{aligned} \dot{\tilde{\phi}} =& \frac{8}{\rho V S b^2 C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho V S b^2 C_{p_p}}pq + \frac{8\Lambda_2}{\rho V S b^2 C_{p_p}}qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta \\ &- \frac{C_{p_r}}{C_{p_p}}r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\delta_a + \delta_a^{des} - \delta_a^{des}\right) - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r + q\sin\phi\tan\phi \\ &+ r\cos\phi\tan\phi - \dot{\phi}_m, \\ =& -\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\delta_a - \delta_a^{des}\right), \\ =& -\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\vartheta^T\hat{\mathbf{k}} - \vartheta^T\mathbf{k}\right), \text{ and} \\ =& -\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\vartheta^T\tilde{\mathbf{k}}, \end{aligned} \tag{3.49}$$

where $\tilde{\mathbf{k}}$ is the parameter error defined as $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$.

By using the Lyapunov function candidate,

$$V = \frac{1}{2}\tilde{\phi}^2 - \frac{C_{p_{\delta_a}}}{bC_{p_p}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \tag{3.50}$$

(C_{p_p} is assumed to be negative) and take its derivative gives

$$\begin{aligned}
\dot{V} =& \tilde{\phi}\dot{\tilde{\phi}} - \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\tilde{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \\
=& \tilde{\phi}\left(-\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\vartheta^T\tilde{\mathbf{k}}\right) - \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}} + \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \\
=& -\lambda_\phi\tilde{\phi}^2 - \frac{2V\tilde{\phi}C_{p_{\delta_a}}}{bC_{p_p}}\vartheta^T\tilde{\mathbf{k}} - \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}} + \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \text{ and} \\
=& -\lambda_\phi\tilde{\phi}^2 - \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\left(V\tilde{\phi}\vartheta^T + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\right)\tilde{\mathbf{k}} + \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}.
\end{aligned} \tag{3.51}$$

Let the parameter update law be

$$\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_4}\left(-\Lambda V\tilde{\phi}\vartheta, \hat{\mathbf{k}}\right). \tag{3.52}$$

Theorem 3.1.4 If

H10: $\phi^d, q, r, \dot{p} \in \mathcal{L}_\infty$ are continuous signals,

H11: $V \in [V_{min}, V_{max}]$,

H12: $\theta \in [-\bar{\theta}, \bar{\theta}]$ where $\bar{\theta} < \pi/2$ and θ is continuous,

H13: $|k_5| \leq n$,

H14: $\mathbf{k} \in S_4$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.2),(3.47), (3.48), and (3.52) is stable and $\tilde{\phi}$ is bounded by

$$\lim_{t\to\infty}|\tilde{\phi}| \leq \sqrt{\left|\frac{2C_{p_{\delta_a}}}{\lambda_\phi\lambda_5 bC_{p_p}}\right| mn}.$$

Proof: We need to show all that states are bounded. Equations (3.51) and (3.52), Hypothesis **H14**, and Lemma 2.3.7 imply that

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \frac{2C_{p_{\delta_a}}}{bC_{p_p}} \dot{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}}. \tag{3.53}$$

However, since k_5 is the only parameter with time-varying components, (3.53) reduces to

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \frac{2C_{p_{\delta_a}}}{bC_{p_p}} \dot{k_5}^T \lambda_5^{-1} \tilde{k_5}.$$

Applying Hypotheses **H11** and **H12** give

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \left| \frac{2C_{p_{\delta_a}}}{\lambda_5 bC_{p_p}} \right| mn,$$

where m is the upper bound for $|\tilde{k_5}|$. Applying Theorem 2.3.6, the bound on $\tilde{\phi}$ is

$$\lambda_\phi \tilde{\phi}^2 \leq \left| \frac{2C_{p_{\delta_a}}}{\lambda_5 bC_{p_p}} \right| mn \text{ and} \tag{3.54}$$

$$\Rightarrow |\tilde{\phi}| \leq \sqrt{\left| \frac{2C_{p_{\delta_a}}}{\lambda_\phi \lambda_5 bC_{p_p}} \right| mn}. \tag{3.55}$$

Therefore, by increasing either the control gain, λ_ϕ, or the adaptive gain, λ_5, tracking error improves. Furthermore, Equation (3.54) implies that

$$\tilde{\phi} \in \mathcal{L}_\infty. \tag{3.56}$$

Because **H10** states that $\phi^d \in \mathcal{L}_\infty$, we have

$$\phi_m \in \mathcal{L}_\infty \tag{3.57}$$

since the model is BIBO stable. Equations (3.56), all the Hypothesis, Assumption **A3**, (3.44), and (3.57) implies that $\vartheta, \dot{\tilde{\phi}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 4 MRAC Scheme B for roll attitude hold.

```
1: Obtain $\phi$, $p$, $r$, $\alpha$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\phi}_m = k_\phi\left(\phi^d - \phi_m\right)$.
3: Compute:
   $\vartheta = \begin{pmatrix} \frac{\lambda_\phi\tilde{\phi}+q\sin\theta\tan\theta+r\cos\theta\tan\theta-\dot{\phi}_m}{V} & \frac{\dot{p}}{V^2} & -\frac{pq}{V^2} & \frac{qr}{V^2} & -1 & \psi-\chi & -\frac{r}{V} & -\delta_r \end{pmatrix}^T$.
4: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_4}\left(\Lambda V\tilde{\phi}\vartheta, \hat{\mathbf{k}}\right)$.
5: Compute the aileron command: $\delta_a = \vartheta^T\hat{\mathbf{k}}$.
```

3.1.3 MRAC Scheme C

Two MRAC controllers will be developed in this section. Like the previous section, backstepping is not used to create the two controllers. Instead, both roll and pitch equations are reduced to pseudo first-order systems where the roll and pitch rate derivatives are treated as measured states. The first controller regulates pitch, while the second controller regulates roll. The boundedness of both controllers will be proven.

Pitch Attitude Hold

In MRAC Scheme C, the aircraft model is further simplified by lumping system states into with the aircraft parameters. Although this makes the parameters vary with time, it will be shown that the system is ultimately bounded. Our desire is to pick δ_e such that Equation (3.3) is

$$\dot{\tilde{\theta}} = -\lambda_\theta\tilde{\theta}, \tag{3.58}$$

where $\lambda_\theta > 0$. The desired elevator command is $\delta_e^{des} = \vartheta^T\mathbf{k}$,where

$$\vartheta^T = \begin{pmatrix} \frac{\lambda_\theta\tilde{\theta}-k_\theta(\theta^d-\theta_m)-r\sin\phi}{V} \\ 1 \\ -\theta \end{pmatrix},$$

$$\mathbf{k} = \begin{pmatrix} \frac{\bar{c}C_{m_p}}{C_{m_{\delta_e}}} \\ \frac{J_y\dot{q}\cos\phi}{\frac{1}{2}S\bar{c}C_{m_{\delta_e}}} - \frac{C_{m_o}\cos\phi}{C_{m_{\delta_e}}} + \frac{C_{m_\alpha}\cos\phi}{C_{m_{\delta_e}}}\gamma + \frac{J_{xz}\cos\phi}{\frac{1}{2}V^2S\bar{c}C_{m_{\delta_e}}}\left(p^2 - r^2\right) - \frac{(J_z - J_x)\cos\phi}{\frac{1}{2}V^2S\bar{c}C_{m_{\delta_e}}}pr \\ \frac{C_{m_\alpha}\cos\phi}{C_{m_{\delta_e}}} \end{pmatrix}. \tag{3.59}$$

Unfortunately, the parameters, $\mathbf{k}$, are unknown, thus the estimates, $\hat{\mathbf{k}}$, are used in the elevator command such that

$$\delta_a = \vartheta^T\hat{\mathbf{k}}. \tag{3.60}$$

Using the above definition for δ_a and adding and subtracting δ_a^{des} to equation (3.32) we get

$$\begin{aligned} \dot{\tilde{\theta}} =& \frac{V\cos\phi}{C_{m_q}\bar{c}}\Bigg(\frac{2}{\rho V^2 S\bar{c}}\left[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - \left(J_z - J_x\right)pr\right] - C_{m_0} \\ & + C_{m_\alpha}\gamma - C_{m_\alpha}\theta - C_{m_{\delta_e}}\left(\delta_e^{des} + \delta_e - \delta_e^{des}\right)\Bigg) - r\sin\phi - \dot{\theta}_m \text{ and} \\ =& -\lambda_\theta\tilde{\theta} - \frac{VC_{m_{\delta_e}}}{C_{m_q}\bar{c}}\vartheta^T\tilde{\mathbf{k}}, \end{aligned} \tag{3.61}$$

where $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$. Defining the Lyapunov function candidate as

$$V \triangleq \frac{1}{2}\tilde{\theta}^2 - \frac{C_{m_{\delta_e}}}{2C_{m_q}\bar{c}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}$$

(C_{m_q} is assumed to be positive) and take its derivative we get

$$\begin{aligned} \dot{V} =& \tilde{\theta}\dot{\tilde{\theta}} - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \\ =& -\lambda_\theta\tilde{\theta}^2 - \frac{\tilde{\theta}V}{C_{m_q}\bar{c}}\vartheta^T\tilde{\mathbf{k}} - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}} + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \text{ and} \\ =& -\lambda_\theta\tilde{\theta}^2 - \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\left(\tilde{\theta}V\vartheta^T + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\right)\tilde{\mathbf{k}} + \frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}. \end{aligned} \tag{3.62}$$

The update law is chosen to be

$$\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_5}\left(-\Lambda\tilde{\theta}V\vartheta, \hat{\mathbf{k}}\right). \tag{3.63}$$

Theorem 3.1.5 If

H13: $\theta^d, p, r, \dot{q} \in \mathcal{L}_\infty$ are continuous signals,

H14: $V \in [V_{min}, V_{max}]$,

H15: $\phi \in [-\bar{\phi}, \bar{\phi}]$ where $\bar{\phi} < \pi/2$ and ϕ is continuous,

H16: $||\Lambda|| \leq o$,

H17: $\mathbf{k} \in S_5$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.1),(3.59), (3.60), and (3.63) is stable $\tilde{\theta}$ is ultimately bounded such that

$$\lim_{t \to \infty} |\tilde{\theta}| \leq \sqrt{\left| \frac{C_{m_{\delta_e}}}{\lambda_\theta o C_{m_q} \bar{c}} \right| mn}.$$

Proof: We need to show all that signals are bounded. Equations (3.62) and (3.63), Hypothesis **H17**, and Lemma 2.3.7 imply that

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \frac{C_{m_{\delta_e}}}{C_{m_q} \bar{c}} \dot{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}}. \tag{3.64}$$

Applying Hypotheses **H16** and **H17** gives

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \left| \frac{C_{m_{\delta_e}}}{C_{m_q} \bar{c}} \right| mno^{-1},$$

where m is the upper bound on the norm $\left|\left|\tilde{\mathbf{k}}\right|\right|$. Using Theorem 2.3.6, the bound on $\tilde{\theta}$ is

$$\lambda_\theta \tilde{\theta}^2 \leq \left| \frac{C_{m_{\delta_e}}}{C_{m_q} \bar{c}} \right| mno^{-1} \text{ and}$$
$$\Rightarrow |\tilde{\theta}| \leq \sqrt{\left| \frac{C_{m_{\delta_e}}}{\lambda_\theta o C_{m_q} \bar{c}} \right| mn}. \tag{3.65}$$

Therefore, increasing the control gain, λ_θ, or the matrix norm of Λ reduces the tracking error. Furthermore, Equation 3.65 implies that

$$\tilde{\theta} \in \mathcal{L}_\infty. \tag{3.66}$$

Using Hypothesis **H13** implies that

$$\theta_m \in \mathcal{L}_\infty, \tag{3.67}$$

since the model is BIBO stable. Equations (3.3), (3.66), (3.67) imply that $\theta \in \mathcal{L}_\infty$. This result coupled with all the Hypotheses and Equation (3.67) imply that $\vartheta, \dot{\tilde{\theta}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 5 MRAC Scheme C for pitch attitude hold.

```
1: Obtain $\phi$, $p$, $r$, $\alpha$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\theta}_m = k_\theta(\theta_d - \theta_m)$.
3: Compute:
   $\vartheta = \begin{pmatrix} \frac{\lambda_\theta\tilde{\theta} - k_\theta(\theta_d - \theta_m)}{V\cos\phi} & 1 & -\theta \end{pmatrix}^T$.
4: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_5}\left(-\Lambda V\cos\phi\tilde{\theta}\vartheta, \hat{\mathbf{k}}\right)$.
5: Compute the elevator command: $\delta_e = \vartheta^T\hat{\mathbf{k}}$.
```

Roll Attitude Hold

In this section, an MRAC roll controller is derived. The second order roll equations are reduced to a pseudo first-order system in order to develop the MRAC. The assumptions that system states can be lumped into the bias term for the system is used. Let the derivative of the tracking error be

$$\dot{\tilde{\phi}} = p + q\sin\phi\tan\theta + r\cos\phi\tan\theta - \dot{\phi}_m. \tag{3.68}$$

Solving for p in equation (2.15) we find that

$$\begin{aligned} p =& \frac{8}{\rho V S b^2 C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho V S b^2 C_{p_p}} pq + \frac{8\Lambda_2}{\rho V S b^2 C_{p_p}} qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta \\ &- \frac{C_{p_r}}{C_{p_p}} r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\delta_a - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r. \end{aligned} \tag{3.69}$$

Substituting this equation in equation (3.68) give us

$$\begin{aligned} \dot{\tilde{\phi}} =& \frac{8}{\rho V S b^2 C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho V S b^2 C_{p_p}} pq + \frac{8\Lambda_2}{\rho V S b^2 C_{p_p}} qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta \\ &- \frac{C_{p_r}}{C_{p_p}} r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\delta_a - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m. \end{aligned} \tag{3.70}$$

But since we want the above equation to reduce to the stable system, $\dot{\tilde{\phi}} = -\lambda_\phi \tilde{\phi}$, we define the desired aileron command to be

$$\begin{aligned} \delta_a^{des} =& \frac{bC_{p_p}}{2VC_{p_{\delta_a}}}\left(\lambda_\phi \dot{\tilde{\phi}} + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m\right) - \frac{C_{p_0}}{C_{p_{\delta_a}}} - \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\beta \\ &- \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\delta_r - \frac{C_{p_r}}{2VC_{p_{\delta_a}}} r + \frac{4}{\rho V^2 S b}\dot{p} - \frac{4\Lambda_1}{\rho V^2 S b} pq + \frac{4\Lambda_2}{\rho V^2 S b} qr. \end{aligned} \tag{3.71}$$

Using the assumption that $\beta = \chi - \psi$ we have

$$\begin{aligned} \delta_a^{des} =& \frac{bC_{p_p}}{2VC_{p_{\delta_a}}}\left(\lambda_\phi \dot{\tilde{\phi}} + q\sin\phi\tan\phi + r\cos\phi\tan\phi - \dot{\phi}_m\right) - \frac{C_{p_0}}{C_{p_{\delta_a}}} - \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\chi \\ &+ \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\psi - \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\delta_r - \frac{C_{p_r}}{2VC_{p_{\delta_a}}} r + \frac{4}{\rho V^2 S b}\dot{p} - \frac{4\Lambda_1}{\rho V^2 S b} pq + \frac{4\Lambda_2}{\rho V^2 S b} qr. \end{aligned} \tag{3.72}$$

Grouping the knowns and unknowns together to get the form $\delta_a^{des} = \vartheta^T \mathbf{k}$ gives

$$\vartheta = \begin{pmatrix} \frac{\lambda_\phi \tilde{\phi} - k_\phi\left(\phi^d - \phi_m\right)}{V} \\ -1 \\ -\chi \\ -\delta_r \end{pmatrix} \text{ and } \mathbf{k} = \begin{pmatrix} \left(k_1 = \frac{bC_{p_p}}{2C_{p_{\delta_a}}}\right) \\ \left(k_2 = C_{p_{bias}}\right) \\ \left(k_3 = \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\right) \\ \left(k_4 = \frac{C_{p_{\delta_r}}}{C_{p_{\delta_a}}}\right) \end{pmatrix}, \tag{3.73}$$

where

$$\begin{aligned}C_{p_{bias}} =&\frac{4}{V^2\rho Sb}\dot{p} + \frac{C_{p_0}}{C_{p_{\delta_a}}} + \frac{bC_{p_p}}{2C_{p_{\delta_a}}}\left(q\sin\theta\tan\theta + r\cos\theta\tan\theta\right)\\ &- \frac{4\Lambda_1}{\rho Sb}\frac{pq}{V^2} + \frac{4\Lambda_2}{\rho Sb}\frac{qr}{V^2} - \frac{C_{p_\beta}}{C_{p_{\delta_a}}}\psi - \frac{C_{p_r}}{2C_{p_{\delta_a}}}\frac{r}{V}.\end{aligned}$$

Since $\mathbf{k}$ is unknown, the aileron command is defined as

$$\delta_a = \vartheta^T\hat{\mathbf{k}}. \tag{3.74}$$

Using this definition for δ_a and adding and subtracting δ_a^{des} from Equation (3.70) gives us

$$\begin{aligned}\dot{\tilde{\phi}} =&\frac{8}{\rho VSb^2C_{p_p}}\dot{p} - \frac{8\Lambda_1}{\rho VSb^2C_{p_p}}pq + \frac{8\Lambda_2}{\rho VSb^2C_{p_p}}qr - \frac{2VC_{p_0}}{bC_{p_p}} - \frac{2VC_{p_\beta}}{bC_{p_p}}\beta\\ &- \frac{C_{p_r}}{C_{p_p}}r - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\delta_a + \delta_a^{des} - \delta_a^{des}\right) - \frac{2VC_{p_{\delta_r}}}{bC_{p_p}}\delta_r + q\sin\phi\tan\phi\\ &+ r\cos\phi\tan\phi - \dot{\phi}_m,\\ =&-\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\delta_a - \delta_a^{des}\right),\\ =&-\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\left(\vartheta^T\hat{\mathbf{k}} - \vartheta^T\mathbf{k}\right), \text{ and}\\ =&-\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\vartheta^T\tilde{\mathbf{k}},\end{aligned} \tag{3.75}$$

where $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$. Setting the Lyapunov candidate function as

$$V = \frac{1}{2}\tilde{\phi}^2 - \frac{C_{p_{\delta_a}}}{bC_{p_p}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}} \tag{3.76}$$

(C_{p_p} is assumed to be negative) and take its derivative to get

$$\begin{aligned}\dot{V} =&\tilde{\phi}\dot{\tilde{\phi}} - \frac{VC_{p_{\delta_a}}}{bC_{p_p}}\dot{\tilde{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}},\\ =&\tilde{\phi}\left(-\lambda_\phi\tilde{\phi} - \frac{2VC_{p_{\delta_a}}}{bC_{p_p}}\vartheta^T\tilde{\mathbf{k}}\right) - \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}} + \frac{2C_{p_{\delta_a}}}{bC_{p_p}}\dot{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}},\end{aligned}$$

$$
\begin{aligned}
&= -\lambda_\phi \tilde{\phi}^2 - \frac{VC_{p_{\delta_a}}}{bC_{p_p}} \dot{\hat{\mathbf{k}}}^T \Lambda^{-1} \tilde{\mathbf{k}} + \frac{VC_{p_{\delta_a}}}{bC_{p_p}} \dot{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}}, \text{ and} \\
&= -\lambda_\phi \tilde{\phi}^2 - \frac{2C_{p_{\delta_a}}}{bC_{p_p}} \left(V\tilde{\phi}\vartheta^T + \dot{\hat{\mathbf{k}}}^T \Lambda^{-1} \right) \tilde{\mathbf{k}} + \frac{VC_{p_{\delta_a}}}{bC_{p_p}} \dot{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}}. \qquad (3.77)
\end{aligned}
$$

Update law is selected as

$$
\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_6} \left(-\Lambda V \tilde{\phi} \vartheta, \hat{\mathbf{k}} \right). \qquad (3.78)
$$

Theorem 3.1.6 If

H16: $\phi^d, q, r, \dot{p} \in \mathcal{L}_\infty$ are continuous signals,

H17: $V \in [V_{min}, V_{max}]$,

H18: $\theta \in [-\bar{\theta}, \bar{\theta}]$ where $\bar{\theta} < \pi/2$ and θ is continuous,

H19: $|k_2| \leq n$,

H20: $\mathbf{k} \in S_6$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.2),(3.73), (3.74), and (3.78) is stable and $\tilde{\phi}$ has the ultimate bound

$$
\lim_{t \to \infty} |\tilde{\phi}| \leq \sqrt{\frac{C_{p_{\delta_a}}}{\lambda_\phi \lambda_2 b C_{p_p}} mn}.
$$

Proof: We need to show all that states are bounded. Equations (3.77) and (3.78), **H18**, and Lemma 2.3.7 imply that

$$
\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \frac{C_{p_{\delta_a}}}{bC_{p_p}} \dot{\mathbf{k}}^T \Lambda^{-1} \tilde{\mathbf{k}}. \qquad (3.79)
$$

However, since k_2 is the only parameter in $\mathbf{k}$ that is time-varying the derivative reduces to

$$
\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \frac{C_{p_{\delta_a}}}{bC_{p_p}} \dot{k_2}^T \lambda_2^{-1} \tilde{k}_2.
$$

Applying Hypotheses **H19** and **H20**

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \frac{C_{p_{\delta_a}}}{bC_{p_p}}\lambda_2^{-1}mn,$$

where m is the upper bound on $|\dot{\tilde{k}}_5|$. Using Theorem 2.3.6, the bound on $\tilde{\phi}$ is

$$\begin{aligned} \lambda_\phi \tilde{\phi}^2 &\leq \frac{C_{p_{\delta_a}}}{bC_{p_p}}\lambda_2^{-1}mn \text{ and} \\ \Rightarrow |\tilde{\phi}| &\leq \sqrt{\frac{C_{p_{\delta_a}}}{\lambda_\phi\lambda_2 bC_{p_p}}mn}. \end{aligned} \tag{3.80}$$

Therefore, roll tracking improves as the control gain, λ_ϕ, or adaptive gain, λ_2, increase.

Examining Equation 3.80 reveals that

$$\tilde{\phi} \in \mathcal{L}_\infty. \tag{3.81}$$

Because **H16** states that $\phi^d \in \mathcal{L}_\infty$ and the model is BIBO stable, we have

$$\phi_m \in \mathcal{L}_\infty \tag{3.82}$$

All the Hypotheses, (3.70), (3.82), and (3.81) imply that $\vartheta, \dot{\tilde{\phi}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 6 MRAC Scheme C for roll attitude hold.

```
1: Obtain $\phi$, $p$, $r$, $\alpha$, $q$, $\theta$ from sensors.
2: Update the reference model according to $\dot{\phi}_m = k_\phi\left(\phi^d - \phi_m\right)$.
3: Compute:
   $\vartheta = \left(\frac{\lambda_\phi\tilde{\phi} - k_\phi(\phi_d - \phi)}{2V} \quad 1 \quad -\delta_r - \chi\right)^T$.
4: Update the gain estimate according to: $\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_6}\left(-\Lambda V\tilde{\phi}\vartheta, \hat{\mathbf{k}}\right)$.
5: Compute the aileron command: $\delta_a = \vartheta^T\hat{\mathbf{k}}$.
```

3.1.4 MRAC Scheme D

Two MRAC controllers will be developed in this section. The pitch and roll equations are in the same forms as in Scheme C. The difference between Scheme C and Scheme D is that Scheme D lumps more terms in to the “bias“ term of the system parameters. The first controller regulates pitch, while the second controller regulates roll. The boundedness of both controllers will be proven.

Pitch Attitude Hold

An MRAC pitch controller is derived in this section. Like Schemes B and C, Scheme D reduces the pitch equations to a pseudo first-order system such that

$$\begin{aligned} q =& \frac{V}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - \left(J_z - J_x\right)pr] \right. \\ & \left. - C_{m_0} - C_{m_\alpha}\alpha - C_{m_{\delta_e}}\delta_e\right), \end{aligned}$$

where $\dot{q}$ is treated like a measured state.

Substituting into Equation (2.13) gives

$$\begin{aligned} \dot{\theta} &= \frac{V\cos\phi}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - \left(J_z - J_x\right)pr] - C_{m_0} - C_{m_\alpha}\alpha \right. \\ & \qquad \left. - C_{m_{\delta_e}}\delta_e\right) - r\sin\phi \text{ and} \\ &= -V\bar{C}_{m_{\delta_e}}\delta_e + V\bar{C}_{m_{\text{bias}}}. \end{aligned} \tag{3.83}$$

where

$$\begin{aligned} \bar{C}_{m_{\delta_e}} &= \frac{\cos\phi}{C_{m_q}\bar{c}}C_{m_{\delta_e}} \text{ and} \\ \bar{C}_{m_{\text{bias}}} &= \frac{\cos\phi}{C_{m_q}\bar{c}}\left(\frac{2}{\rho V^2 S\bar{c}}\left[J_y\dot{q} - J_{xz}\left(r^2 - p^2\right) - \left(J_z - J_x\right)pr\right] - C_{m_0} - C_{m_\alpha}\alpha\right) \\ & \quad - \frac{r\sin\phi}{V}. \end{aligned}$$

The sign of $\bar{C}_{m_{\delta_e}}$ is significant when choosing a Lyapunov function candidate. Invoking Assumption **A2** allows the $\cos\phi$ term to be positive, while the $C_{m_{\delta_e}}$ and $\bar{c}$ terms are assumed positive, and the C_{m_q} term is assumed negative, leading to $\bar{C}_{m_{\delta_e}}$ being negative. It is significant that $\bar{C}_{m_{\delta_e}}$ and $\bar{C}_{m_{\text{bias}}}$ are time varying, and thus, only boundedness can be shown.

Using Equation (3.3) we obtain

$$\begin{aligned}
\dot{\tilde{\theta}} &= \dot{\theta} - \dot{\theta}_m, \\
&= -V\bar{C}_{m_{\delta_e}}\delta_e + V\bar{C}_{m_{\text{bias}}} - k_\theta(\theta^d - \theta_m), \\
&= -\gamma\tilde{\theta} + V\bar{C}_{m_{\delta_e}}\left[\frac{\gamma\tilde{\theta} - k_\theta(\theta^d - \theta_m)}{V\bar{C}_{m_{\delta_e}}}\right] + V\bar{C}_{m_{\delta_e}}\left(\frac{\bar{C}_{m_{\text{bias}}}}{\bar{C}_{m_{\delta_e}}} - \delta_e\right), \text{ and} \\
&= -\gamma\tilde{\theta} + V\bar{C}_{m_{\delta_e}}\left(\vartheta^T\mathbf{k} - \delta_e\right), \qquad (3.84)
\end{aligned}$$

where γ is a positive control gain, and

$$\vartheta = \begin{pmatrix} \frac{\gamma\tilde{\theta} - k_\theta(\theta^d - \theta_m)}{V} \\ 1 \end{pmatrix} \text{ and } K = \begin{pmatrix} \frac{1}{\bar{C}_{m_{\delta_e}}} \\ \frac{\bar{C}_{m_{\text{bias}}}}{\bar{C}_{m_{\delta_e}}} \end{pmatrix}. \qquad (3.85)$$

By selecting $\delta_e = \vartheta^T\mathbf{k}$ we have an exponentially stable system. However, since $\mathbf{k}$ is not known, the estimate, $\hat{\mathbf{k}}$, is used such that

$$\delta_e = \vartheta^T\hat{\mathbf{k}}. \qquad (3.86)$$

Furthermore, defining $\tilde{\mathbf{k}} \overset{\triangle}{=} \hat{\mathbf{k}} - \mathbf{k}$ results in

$$\dot{\tilde{\theta}} = -\gamma\tilde{\theta} - V\bar{C}_{m_{\delta_e}}\vartheta^T\tilde{\mathbf{k}}.$$

We define the Lyapunov function candidate

$$\mathcal{V} = \frac{1}{2}\tilde{\theta}^2 - \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}},$$

noting that $\bar{C}_{m_{\delta_e}}$ being negative ensures that $\mathcal{V}$ is positive definite. Differentiating, we obtain

$$\begin{aligned}
\dot{\mathcal{V}} &= \tilde{\theta}\dot{\tilde{\theta}} - \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}}, \\
&= -\gamma\tilde{\theta}^2 - V\bar{C}_{m_{\delta_e}}\tilde{\theta}\tilde{\mathbf{k}}^T\vartheta - \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\hat{\mathbf{k}}} + \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\mathbf{k}}, \text{ and} \\
&= -\gamma\tilde{\theta}^2 - \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\left(V\tilde{\theta}\vartheta + \Lambda^{-1}\dot{\hat{\mathbf{k}}} - \Lambda^{-1}\dot{\mathbf{k}}\right). \qquad (3.87)
\end{aligned}$$

The update law is therefore selected as

$$\dot{\hat{\mathbf{k}}} = \text{Proj}_{S_7}\left(-V\tilde{\theta}\Lambda\vartheta, \hat{\mathbf{k}}\right). \qquad (3.88)$$

Theorem 3.1.7 If

H19: $\theta^d, p, r \in \mathcal{L}_\infty$ are continuous signals,

H20: $V \in [V_{min}, V_{max}]$,

H21: $\phi \in [-\bar{\phi}, \bar{\phi}]$ where $\bar{\phi} < \pi/2$ and ϕ is continuous,

H22: $||\Lambda|| \leq o$,

H23: $\mathbf{k} \in S_7$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.1),(3.85), (3.86), and (3.88) is stable and $\tilde{\theta}$ is ultimately bounded such that

$$\lim_{t\to\infty}|\tilde{\theta}| \leq \sqrt{\left|\frac{\bar{C}_{m_{\delta_e}}}{\lambda_\theta o}\right| mn}.$$

Proof: We need to show all that states are bounded. Equations (3.87) and (3.88), Hypothesis **H23**, and Lemma 2.3.7 imply that

$$\dot{\mathcal{V}} \leq -\lambda_\theta\tilde{\theta}^2 + \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\mathbf{k}}. \qquad (3.89)$$

Applying Hypotheses **H22** and **H23** gives

$$\dot{\mathcal{V}} \leq -\lambda_\theta \tilde{\theta}^2 + \left|\bar{C}_{m_{\delta_e}}\right| mno^{-1},$$

where m is the upper bound on the vector norm, $\left\|\tilde{\mathbf{k}}\right\|$. Using Theorem 2.3.6, the bound on $\tilde{\theta}$ is

$$\begin{aligned} \lambda_\theta \tilde{\theta}^2 &\leq \left|\bar{C}_{m_{\delta_e}}\right| mno^{-1} \text{ and} \\ \Rightarrow |\tilde{\theta}| &\leq \sqrt{\left|\frac{\bar{C}_{m_{\delta_e}}}{\lambda_\theta o}\right| mn}. \end{aligned} \tag{3.90}$$

Therefore, increasing the adaptive gains or the control gain, λ_θ, improves pitch tracking. Equation 3.90 implies

$$\tilde{\theta} \in \mathcal{L}_2 \cap \mathcal{L}_\infty. \tag{3.91}$$

Hypothesis **H19** states that $\theta^d \in \mathcal{L}_\infty$, therefore

$$\theta_m \in \mathcal{L}_\infty, \tag{3.92}$$

since the model is BIBO stable. Equations (3.3), (3.91), (3.92) imply that $\theta \in \mathcal{L}_\infty$. Applying this result, all the Hypotheses, (3.84), and (3.92) to (3.84) implies that $\vartheta, \dot{\tilde{\theta}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 7 MRAC Scheme D for pitch attitude hold

```
1: Obtain $\theta$ and $V$ from sensors.
2: Update the reference model according to
       $\dot{\theta}_m = k_\theta\left(\theta^d - \theta_m\right).$
3: Compute $\vartheta = \begin{pmatrix} \frac{\gamma(\theta-\theta_m)-k_\theta\left(\theta^d-\theta_m\right)}{V} & 1 \end{pmatrix}^T.$
4: Update the parameter estimate according to
       $\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_7}\left(-V(\theta-\theta_m)\Lambda\vartheta, \hat{\mathbf{k}}\right).$
5: Compute the elevator command $\delta_e = \vartheta^T\hat{\mathbf{k}}$.
```

Roll Attitude Hold

In this section, an MRAC roll controller is derived. Following the precedence of Schemes B and C, Scheme D solves for p from Equation (2.15) to get

$$p = V\left(\frac{-\frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_{\delta_a}}}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}}\right)\delta_a + \frac{V\left(\dot{p} + \hat{J}_2 qr\right)}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}}$$
$$- \frac{1}{2}\rho V^3 S\frac{b}{2}\frac{\left(C_{p_0} + C_{p_\beta}\beta + C_{p_r}\frac{br}{2V} + C_{p_{\delta_r}}\delta_r\right)}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}}.$$

Substituting into Equation (2.12) and defining

$$\bar{C}_{p_{\delta_a}} = \frac{-\frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_{\delta_a}}}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}} \text{ and}$$
$$\bar{C}_{p_{\text{bias}}} = \frac{\left(\dot{p} + \hat{J}_2 qr\right)}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}} - \frac{1}{2}\rho V^2 S\frac{b}{2}\frac{\left(C_{p_0} + C_{p_\beta}\beta + C_{p_r}\frac{br}{2V} + C_{p_{\delta_r}}\delta_r\right)}{\hat{J}_1 qV + \frac{1}{2}\rho V^2 S\frac{b}{2}C_{p_p}\frac{b}{2}}$$
$$+ \frac{q\sin\phi\tan\theta + r\cos\phi\tan\theta}{V}$$

gives

$$\dot{\phi} = V\bar{C}_{p_{\delta_a}}\delta_a + V\bar{C}_{p_{\text{bias}}}. \tag{3.93}$$

Using Equation (3.5) we obtain

$$\begin{aligned}
\dot{\tilde{\phi}} &= \dot{\phi} - \dot{\phi}_m, \\
&= V\bar{C}_{p_{\delta_a}} + V\bar{C}_{p_{\text{bias}}} - k_\phi(\phi^d - \phi_m), \\
&= -\gamma\tilde{\phi} + V\bar{C}_{p_{\delta_a}}\left[\frac{\gamma\tilde{\phi} - k_\phi(\phi^d - \phi_m)}{V\bar{C}_{p_{\delta_a}}}\right] + V\bar{C}_{p_{\delta_a}}\left(\frac{V\bar{C}_{p_{\text{bias}}}}{\bar{C}_{p_{\delta_a}}} - \delta_a\right), \text{ and} \\
&= -\gamma\tilde{\phi} + V\bar{C}_{p_{\delta_a}}\left(\vartheta^T\mathbf{k} - \delta_a\right),
\end{aligned} \tag{3.94}$$

where γ is a positive control gain, and

$$\vartheta = \begin{pmatrix} \frac{\gamma\tilde{\phi} - k_\phi(\phi^d - \phi_m)}{V} \\ 1 \end{pmatrix} \text{ and } K = \begin{pmatrix} \frac{1}{\bar{C}_{p_{\delta_a}}} \\ \frac{\bar{C}_{p_{\text{bias}}}}{\bar{C}_{p_{\delta_a}}} \end{pmatrix}. \tag{3.95}$$

By selecting $\delta_a = \vartheta^T \mathbf{k}$ we have an exponentially stable system. However, as $\mathbf{k}$ is not known, the estimate, $\hat{\mathbf{k}}$, is used such that

$$\delta_a = \vartheta^T \hat{\mathbf{k}}. \tag{3.96}$$

Furthermore, defining $\tilde{\mathbf{k}} \overset{\triangle}{=} \hat{\mathbf{k}} - \mathbf{k}$ results in

$$\dot{\tilde{\phi}} = -\gamma\tilde{\phi} - V\bar{C}_{p_{\delta_a}}\vartheta^T\tilde{\mathbf{k}}.$$

We define the Lyapunov function candidate

$$\mathcal{V} = \frac{1}{2}\tilde{\phi}^2 - \bar{C}_{m_{\delta_e}}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}},$$

noting that $\bar{C}_{m_{\delta_a}}$ being negative ensures that $\mathcal{V}$ is positive definite. Differentiating, we obtain

$$\begin{aligned}
\dot{\mathcal{V}} &= \tilde{\phi}\dot{\tilde{\phi}} - \bar{C}_{m_{\delta_a}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}}, \\
&= -\gamma\tilde{\phi}^2 - V\bar{C}_{m_{\delta_a}}\tilde{\phi}\tilde{\mathbf{k}}^T\vartheta - \bar{C}_{m_{\delta_a}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\hat{\mathbf{k}}} + \bar{C}_{m_{\delta_a}}\tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\mathbf{k}}, \text{ and} \\
&= -\gamma\tilde{\phi}^2 - \bar{C}_{m_{\delta_a}}\tilde{\mathbf{k}}^T\left(V\tilde{\phi}\vartheta + \Lambda^{-1}\dot{\hat{\mathbf{k}}} - \Lambda^{-1}\dot{\mathbf{k}}\right).
\end{aligned} \tag{3.97}$$

The update law is selected as

$$\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_8}\left(-V\tilde{\phi}\Lambda\vartheta, \hat{\mathbf{k}}\right). \tag{3.98}$$

Theorem 3.1.8 If

H22: $\phi^d, q, r \in \mathcal{L}_\infty$ are continuous signals,

H23: $V \in [V_{min}, V_{max}]$,

H24: $\theta \in [-\bar{\theta}, \bar{\theta}]$ where $\bar{\theta} < \pi/2$ and θ is continuous,

H25: $||\Lambda|| \leq o$,

H26: $\mathbf{k} \in S_8$,

where V_{min} and V_{max} are positive constants, then the MRAC described by (3.2),(3.95), (3.96), and (3.98) is stable and $\tilde{\phi}$ is ultimately bounded such that

$$\lim_{t\to\infty} |\tilde{\phi}(t)| \leq \sqrt{\left|\frac{\bar{C}_{m_{\delta_a}}}{\lambda_\phi o}\right| mn}.$$

Proof: We need to show all that states are bounded. Equations (3.97) and (3.98), Hypothesis **H26**, and Lemma 2.3.7 imply that

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \bar{C}_{m_{\delta_a}} \tilde{\mathbf{k}}^T \Lambda^{-1} \dot{\mathbf{k}}. \tag{3.99}$$

Applying Hypotheses **H25** and **H26** gives us

$$\dot{\mathcal{V}} \leq -\lambda_\phi \tilde{\phi}^2 + \left|\bar{C}_{m_{\delta_a}}\right| mno^{-1},$$

where m is the upper bound on the vector norm, $\left|\left|\dot{\tilde{\mathbf{k}}}\right|\right|$. Using Theorem 2.3.6, the bound on $\tilde{\phi}$ is

$$\begin{aligned} \lambda_\phi \tilde{\phi}^2 \leq \left|\bar{C}_{m_{\delta_a}}\right| mno^{-1} & \text{ and} \\ \Rightarrow |\tilde{\phi}| \leq \sqrt{\left|\frac{\bar{C}_{m_{\delta_a}}}{\lambda_\phi o}\right| mn}. & \end{aligned} \tag{3.100}$$

Therefore, increasing the adaptive gain or control gain, λ_ϕ, improves the error tracking. Furthermore, Equation (3.100) implies that

$$\tilde{\phi} \in \mathcal{L}_2 \cap \mathcal{L}_\infty. \tag{3.101}$$

Because **H22** states that $\phi^d \in \mathcal{L}_\infty$, we have

$$\phi_m \in \mathcal{L}_\infty \tag{3.102}$$

since the model is BIBO stable. Equations (3.101), (3.93), and (3.102) and all the Hypotheses imply that $\vartheta, \dot{\tilde{\phi}} \in \mathcal{L}_\infty$. The algorithm can be summarized as follows.

Algorithm 8 MRAC Scheme D for roll attitude hold

```
1: Obtain $\phi$ and $V$ from sensors.
2: Update the reference model according to
       $\dot{\phi}_m = k_\phi \left(\phi^d - \phi_m\right).$
3: Compute $\vartheta = \begin{pmatrix} \frac{\gamma(\phi-\phi_m)-k_\phi\left(\phi^d-\phi_m\right)}{V} & 1 \end{pmatrix}^T$.
4: Update the gain estimate according to
       $\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_8}\left(-V(\phi-\phi_m)\Lambda\vartheta, \hat{\mathbf{k}}\right).$
5: Compute the aileron command $\delta_a = \vartheta^T\hat{\mathbf{k}}$.
```

3.2 $\mathcal{L}_1$ Controllers

The $\mathcal{L}_1$ controllers discussed in this thesis are extensions of the Lyapunov MRAC controllers like the ones presented in the previous section. In this section, four different aircraft model approximations are used to derive $\mathcal{L}_1$ controllers. The first approximation will be a first-order model with one unknown parameter. The second approximation is another first-order model but with two unknown parameters. The third will be a second order approximation. The final $\mathcal{L}_1$ controller will use a physically justified model to derive the $\mathcal{L}_1$ controller. Because $\mathcal{L}_1$ are the subject of ongoing research, stability has only been shown for linear systems (see [2,3]). Work is being done to generalize the stability proofs to non-linear systems. Therefore, proofs are not provided for the $\mathcal{L}_1$ controllers in this section. Furthermore, the stability proofs for intermediate Lyapunov MRAC are not provided. All four $\mathcal{L}_1$ controllers use following tracking error definitions for pitch and roll respectively,

$$\begin{aligned} \tilde{\theta} &= \theta - \theta_m \text{ and} \\ \tilde{q} &= q - q_m. \end{aligned} \tag{3.103}$$

3.2.1 1st Order Model with One Lumped, Unknown Parameter

The objective of this section is to develop a first-order $\mathcal{L}_1$ adaptive controller. We will begin by deriving a first-order approximation. Next, we will derive an MRAC controller. This MRAC will be converted to a CMAC and then finally to an $\mathcal{L}_1$ controller.

To derive a first-order approximation, solve for q in Equation (2.16),

$$q = \frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V \bar{c}^2 S}{2J_y}} - V\left(\frac{C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e}{C_{m_q}\bar{c}}\right). \tag{3.104}$$

If the result is substituted into (2.13), then

$$\begin{aligned} \dot{\theta} &= \frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V \bar{c}^2 S}{2J_y}}\cos\phi \\ &\quad -V\left(\frac{C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e}{C_{m_q}\bar{c}}\right)\cos\phi - r\sin\phi \text{ and} \\ &= V\cos\phi k_1 + V\cos\phi k_2\delta_e, \end{aligned} \tag{3.105}$$

where

$$\begin{aligned} k_1 &= \left(\frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V^2 \bar{c}^2 S}{2J_y}} - \frac{C_{m_0} + C_{m_\alpha}\alpha}{C_{m_q}\bar{c}}\right) - \frac{r}{V}\tan\phi \text{ and} \\ k_2 &= -\frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}. \end{aligned}$$

We make the assumption that k_1 is an unknown constant and that k_2 is known. In reality, k_1 is time-varying dependent heavily upon the states of the system. The adaptive mechanism that is derived here should be able to handle the time-varying nature of k_1.

We develop an MRAC to track the first-order model

$$\dot{\theta}_m = -a\theta_m + a\theta_c, \ a > 0. \tag{3.106}$$

This MRAC will be modified to become a $\mathcal{L}_1$. Using the above model the derivative for the tracking error $\tilde{\theta}$ is

$$\dot{\tilde{\theta}} = V\cos\phi k_1 + V\cos\phi k_2\delta_e + a\left(\theta_m - \theta_c\right). \tag{3.107}$$

We define

$$\delta_e^{des} = -\frac{k_1}{k_2} - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_2} \tag{3.108}$$

to be the desired input. If k_1 was known then the tracking error becomes

$$\dot{\tilde{\theta}} = -\lambda\tilde{\theta}. \tag{3.109}$$

However, since we do not know k_1, we use an estimate of k_1, $\hat{k}_1$ to get

$$\delta_e = -\frac{\hat{k}_1}{k_2} - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_2} \tag{3.110}$$

for the control signal. Using δ_e the tracking error becomes

$$\dot{\tilde{\theta}} = -\lambda\tilde{\theta} + V\cos\phi\left(k_1 - \hat{k}_1\right). \tag{3.111}$$

The derivative of the Lyapunov function,

$$\mathcal{V}_1 = \frac{1}{2}\tilde{\theta}^2 + \frac{1}{2\gamma}\left(k_1 - \hat{k}_1\right)^2, \tag{3.112}$$

is

$$\begin{aligned}
\dot{\mathcal{V}}_1 &= \tilde{\theta}\dot{\tilde{\theta}} + \frac{1}{\gamma}\left(k_1 - \hat{k}_1\right)\dot{\hat{k}}_1, \\
&= \tilde{\theta}\left(-\lambda\tilde{\theta} + V\cos\phi\left(k_1 - \hat{k}_1\right)\right) + \frac{1}{\gamma}\left(k_1 - \hat{k}_1\right)\dot{\hat{k}}_1, \\
&= -\lambda\tilde{\theta}^2 + V\cos\phi\tilde{\theta}\left(k_1 - \hat{k}_1\right) + \frac{1}{\gamma}\left(k_1 - \hat{k}_1\right)\dot{\hat{k}}_1, \text{ and} \\
&= -\lambda\tilde{\theta}^2 + \left(\tilde{\theta}V\cos\phi + \frac{1}{\gamma}\dot{\hat{k}}_1\right)\left(k_1 - \hat{k}_1\right).
\end{aligned} \tag{3.113}$$

Thus the update equation for the parameter $\hat{k}_1$ is

$$\dot{\hat{k}}_1 = -\gamma \mathrm{Proj}_{S_9}\left(\tilde{\theta} V \cos\phi, \hat{k}_1\right). \tag{3.114}$$

Note that the derived MRAC is similar to the MRACs in Section 3.1. By modifying the model slightly, we create the companion model

$$\dot{\theta}_m = -a\theta_m + Vk_2\left(\delta_e + \frac{\hat{k}_1}{k_2} + \frac{a\theta_m + \lambda\tilde{\theta}}{V\cos\phi k_2}\right),\ a > 0. \tag{3.115}$$

Using this model we have a CMAC instead of an MRAC. Substituting in δ_e, the original model is recovered.

Modifying the CMAC above such that $\hat{k}_1$ is low-pass filtered in δ_e, yields $\mathcal{L}_1$ controller

$$\begin{aligned}
\dot{\theta}_m &= -a\theta_m + Vk_2\left(\delta_e + \frac{\hat{k}_1}{k_2} + \frac{a\theta_m + \lambda\tilde{\theta}}{V\cos\phi k_2}\right),\ a > 0, \\
\delta_e &= -\frac{C(s)\{\hat{k}_1\}}{k_2} - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_2},\ \text{and} \\
\dot{\hat{k}}_1 &= -\gamma \mathrm{Proj}_{S_9}\left(\tilde{\theta} V\cos\phi, \hat{k}_1\right).
\end{aligned} \tag{3.116}$$

This new controller differs from the MRAC model in the following way. When δ_e is plugged into Equation (3.115) the model does not reduce to the MRAC model. Instead, the model becomes

$$\dot{\theta}_m = -a\theta_m + a\theta^d + (1 - C(s))\{\hat{k}_1\}. \tag{3.117}$$

Because we are filtering the estimate $\hat{k}_1$, the companion model takes on the high frequency content of the $\hat{k}_1$ estimate.

3.2.2 1st Order Model with Two Lumped, Unknown Parameters

Like the previous section, in this section a first-order approximation is derived. Using this new approximation we will design an MRAC that has two unknown parameters. This MRAC will be converted into a CMAC and then made into a $\mathcal{L}_1$ controller.

Equation (3.104) with Assumption **A4** gives the first-order approximation,

$$q = \frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V \bar{c}^2 S}{2J_y}} - V\left(\frac{C_{m_0} + C_{m_\alpha}(\theta - \gamma) + C_{m_{\delta_e}}\delta_e}{C_{m_q}\bar{c}}\right). \tag{3.118}$$

Substituting q into Equation (3.105) gives us the approximation,

$$\begin{aligned}\dot{\theta} &= \frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V \bar{c}^2 S}{2J_y}}\cos\phi, \\ &\quad -V\left(\frac{C_{m_0} + C_{m_\alpha}(\theta - \gamma) + C_{m_{\delta_e}}\delta_e}{C_{m_q}\bar{c}}\right)\cos\phi - r\sin\phi \text{ and} \\ &= V\cos\phi k_1 + V\cos\phi k_2\theta + V\cos\phi k_3\delta_e, \end{aligned} \tag{3.119}$$

where

$$\begin{aligned} k_1 &= \left(\frac{(\dot{q} - \Lambda_4(r^2 - p^2) - \Lambda_5 pr)}{\frac{C_{m_q}\rho V^2 \bar{c}^2 S}{2J_y}} - \frac{C_{m_0} + C_{m_\alpha}\frac{-\gamma - \beta\sin\phi}{\cos\phi}}{C_{m_q}\bar{c}}\right) - \frac{r}{V}\tan\phi, \\ k_2 &= -\frac{C_{m_\alpha}}{C_{m_q}\bar{c}}, \text{ and} \\ k_3 &= -\frac{C_{m_{\delta_e}}}{C_{m_q}\bar{c}}. \end{aligned}$$

We will design the MRAC to track the first-order model

$$\dot{\theta}_m = -a\theta_m + a\theta_c, \; a > 0. \tag{3.120}$$

The derivative for the tracking error $\tilde{\theta}$ is

$$\dot{\tilde{\theta}} = V\cos\phi k_1 + V\cos\phi k_2\theta + V\cos\phi k_3\delta_e + a\left(\theta_m - \theta_c\right). \tag{3.121}$$

If we knew k_1 and k_2 then

$$\delta_e^{des} = -\frac{k_1}{k_3} - \frac{k_2}{k_3} - \frac{a(\theta_m - \theta_c) + \lambda\tilde{\theta}}{V\cos\phi k_3}$$

would make $\dot{\tilde{\theta}} = -\lambda\tilde{\theta}$. But since we do not know k_1 or k_2 we define

$$\delta_e = -\frac{\hat{k}_1}{k_3} - \frac{\hat{k}_2}{k_3}\theta - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_3}, \tag{3.122}$$

where $\hat{k}_1$ and $\hat{k}_2$ are the parameter estimates. Substituting δ_e into (3.121) yields

$$\dot{\tilde{\theta}} = -\lambda\tilde{\theta} + V\cos\phi \begin{pmatrix} k_1 - \hat{k}_1 \\ k_2 - \hat{k}_2 \end{pmatrix}^T \begin{pmatrix} 1 \\ \theta \end{pmatrix}. \tag{3.123}$$

We define $\mathbf{k} = (k_1, k_2)^T$, $\hat{\mathbf{k}} = (\hat{k}_1, \hat{k}_2)^T$, $\tilde{\mathbf{k}} = \mathbf{k} - \hat{\mathbf{k}}$, and $\vartheta = (1, \theta)^T$ which yields

$$\begin{aligned} \dot{\tilde{\theta}} &= -\lambda\tilde{\theta} + V\cos\phi\tilde{\mathbf{k}}^T\vartheta \text{ and} \\ \delta_e &= -\frac{1}{k_3}\hat{\mathbf{k}}^T\vartheta - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_3}. \end{aligned} \tag{3.124}$$

Let

$$\mathcal{V}_1 = \frac{1}{2}\tilde{\theta}^2 + \frac{1}{2}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}} \tag{3.125}$$

be the Lyapunov function candidate for the system. Taking its derivative gives us

$$\begin{aligned} \dot{\mathcal{V}}_1 &= \tilde{\theta}\dot{\tilde{\theta}} + \tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}}, \\ &= \tilde{\theta}\left(-\lambda\tilde{\theta} + V\cos\phi\tilde{\mathbf{k}}^T\vartheta\right) + \tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}}, \\ &= -\lambda\tilde{\theta}^2 + V\cos\phi\tilde{\mathbf{k}}^T\vartheta + \tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}}, \text{ and} \\ &= -\lambda\tilde{\theta}^2 + \tilde{\mathbf{k}}^T\left(\tilde{\theta}V\cos\phi + \dot{\tilde{\mathbf{k}}}\right). \end{aligned} \tag{3.126}$$

Let the update equation be

$$\dot{\hat{\mathbf{k}}} = -\Lambda\text{Proj}_{S_{10}}\left(\tilde{\theta}V\cos\phi, \hat{\mathbf{k}}\right), \tag{3.127}$$

such that we have defined a stable MRAC. Like the previous section, modifying the model as

$$\dot{\theta}_m = -a\theta_m + Vk_3\left(\delta_e + \frac{1}{k_3}\hat{\mathbf{k}}^T\vartheta + \frac{a\theta_m + \lambda\tilde{\theta}}{V\cos\phi k_3}\right), \; a > 0, \tag{3.128}$$

results in a companion model. Substituting δ_e into the companion model returns the original MRAC model.

Filtering the parameter estimate for $\mathbf{k}$ in δ_e changes the CMAC to the $\mathcal{L}_1$ controller,

$$\begin{aligned}
\dot{\theta}_m &= -a\theta_m + Vk_3\left(\delta_e + \frac{\hat{k}_1}{k_2} + \frac{\hat{k}_2}{k_3}\theta + \frac{a\theta_m + \lambda\tilde{\theta}}{V\cos\phi k_3}\right),\ a > 0, \\
\delta_e &= -\frac{1}{k_3}C(s)\{\hat{\mathbf{k}}\}^T\vartheta - \frac{a\left(\theta_m - \theta_c\right) + \lambda\tilde{\theta}}{V\cos\phi k_3},\ \text{and} \\
\dot{\hat{\mathbf{k}}} &= -\Lambda\mathrm{Proj}_{S_{10}}\left(\tilde{\theta}V\cos\phi, \hat{\mathbf{k}}\right). \qquad (3.129)
\end{aligned}$$

Taking δ_e from (3.129) and plugging it into the companion model gives

$$\dot{\theta}_m = -a(\theta_m - \theta_c) + \vartheta^T(1 - C(s))\{\hat{\mathbf{k}}\}.$$

Again, the high frequency content of the parameter estimates is preserved in the companion model.

3.2.3 2nd Order Model

Unlike the previous two section, a second order approximation is used to derive the $\mathcal{L}_1$ controller. This approximation will be used to develop a second order MRAC. The MRAC will be modified to become a CMAC. Furthermore, the CMAC will be converted to an $\mathcal{L}_1$ controller by filtering the parameter estimates that appear in the control signal.

Assuming that $\dot{\theta} \approx q$, (2.16) becomes our second-order approximation

$$\ddot{\theta} = \frac{J_{xz}}{J_y}(r^2 - p^2) + \frac{J_z - J_x}{J_y}pr + \frac{1}{2J_y}\rho V^2\bar{c}S\left[C_{m_0} + C_{m_\alpha}\alpha + \quad C_{m_{\dot{\theta}}}\frac{\bar{c}q}{V} + C_{m_{\delta_e}}\delta_e\right].$$

The above equation can be rewritten as

$$\ddot{\theta} = k_1V\dot{\theta} + k_2V^2 + k_3V^2\delta_e, \qquad (3.130)$$

where

$$\begin{aligned}
k_1 &= \frac{1}{2J_y}\rho\bar{c}SC_{m_q}\frac{\bar{c}}{V}, \\
k_2 &= \frac{J_{xz}}{V^2J_y}(r^2-p^2)+\frac{J_z-J_x}{V^2J_y}pr+\frac{1}{2J_y}\rho\bar{c}S\left[C_{m_0}+C_{m_\alpha}\alpha\right], \text{ and} \\
k_3 &= \frac{1}{2J_y}\rho\bar{c}SC_{m_{\delta_e}}.
\end{aligned}$$

The pitch angle of the model, θ_m, is described by the second order model

$$\ddot{\theta}_m = -2\zeta\omega_n\dot{\theta}_m - \omega_n^2\left(\theta_m - \theta^c\right). \tag{3.131}$$

Differentiating the tracking error, $\tilde{\theta} = \theta - \theta_m$, twice gives us

$$\ddot{\tilde{\theta}} = k_1V\dot{\theta} + k_2V^2 + k_3V^2\delta_e + 2\zeta\omega_n\dot{\theta}_m + \omega_n^2\left(\theta_m - \theta^c\right). \tag{3.132}$$

Let $r = \dot{\tilde{\theta}} + \lambda\tilde{\theta}$ and use the following Lyapunov function candidate,

$$\mathcal{V}_1 = \frac{1}{2}r^2 + k\lambda\tilde{\theta}^2.$$

The derivative of the Lyapunov candidate function is

$$\begin{aligned}
\dot{\mathcal{V}}_1 &= \dot{r}r + 2k\lambda\dot{\tilde{\theta}}\tilde{\theta} \text{ and} \\
&= \Big(k_1V\dot{\theta} + k_2V^2 + k_3V^2\delta_e + 2\zeta\omega_n\dot{\theta}_m + \omega_n^2\left(\theta_m - \theta^c\right)\Big)\, r + 2k\lambda\dot{\tilde{\theta}}\tilde{\theta}. \quad (3.133)
\end{aligned}$$

Let

$$\delta_e = \frac{1}{k_3V^2}\Big[-\hat{k}_1V\dot{\theta} - \hat{k}_2V^2 - 2\zeta\omega_n\dot{\theta}_m - \omega_n^2\left(\theta_m - \theta^c\right) - kr\Big]$$

be the elevator command, and substitute the command signal into Equation (3.133) to get

$$\dot{\mathcal{V}}_1 = -kr^2 + rV\tilde{\mathbf{k}}^T\vartheta + 2k\lambda\dot{\tilde{\theta}}\tilde{\theta}, \tag{3.134}$$

where

$$\vartheta = \begin{bmatrix} \dot{\theta} \\ V \end{bmatrix} \text{ and } \tilde{\mathbf{k}} = \begin{bmatrix} k_1 - \hat{k_1} \\ k_2 - \hat{k_2} \end{bmatrix}.$$

Defining a new Lyapunov function $\mathcal{V} = \mathcal{V}_1 + \frac{1}{2}\tilde{\mathbf{k}}^T \Lambda^{-1}\tilde{\mathbf{k}}$, its derivative is

$$\begin{aligned} \dot{\mathcal{V}} &= -kr^2 + rV\tilde{\mathbf{k}}^T\vartheta + 2k\lambda\dot{\tilde{\theta}}\tilde{\theta} + \tilde{\mathbf{k}}^T\Lambda^{-1}\dot{\tilde{\mathbf{k}}} \text{ and} \\ &= -kr^2 + 2k\lambda\dot{\tilde{\theta}}\tilde{\theta} + \tilde{\mathbf{k}}^T\left(rV\vartheta + \Lambda^{-1}\dot{\tilde{\mathbf{k}}}\right). \end{aligned} \tag{3.135}$$

If we use the update equation

$$\dot{\hat{\mathbf{k}}} = -\Lambda \mathrm{Proj}_{S_{11}}\left(rV\vartheta, \hat{\mathbf{k}}\right), \tag{3.136}$$

and modifying the model to become

$$\ddot{\theta}_m = -2\zeta\omega_n\dot{\theta}_m - \omega_n^2\theta_m + V^2k_3\left(\delta_e - \frac{1}{Vk_3}\hat{\mathbf{k}}^T\vartheta - 2\zeta\omega_n\dot{\theta}_m - \omega_n^2\theta_m - kr\right), \tag{3.137}$$

gives us a CMAC. As previously stated, CMACs are equivalent to MRACs. By low-pass filtering the parameter estimates on the control output such that

$$\delta_e = \frac{1}{k_3V^2}\left[-VC(s)\{\hat{\mathbf{k}}\}^T\vartheta - 2\zeta\omega_n\dot{\theta}_m - \omega_n^2\left(\theta_m - \theta^c\right) - kr\right],$$

we get have defined the following $\mathcal{L}_1$ controller

$$\begin{aligned} \ddot{\theta}_m &= -2\zeta\omega_n\dot{\theta}_m - \omega_n^2\theta_m + V^2k_3\left(\delta_e - \frac{1}{Vk_3}\hat{\mathbf{k}}^T\vartheta - 2\zeta\omega_n\dot{\theta}_m - \omega_n^2\theta_m - kr\right), \\ \dot{\hat{\mathbf{k}}} &= -\Lambda \mathrm{Proj}_{S_{11}}\left(rV\vartheta, \hat{\mathbf{k}}\right), \text{ and} \\ \delta_e &= \frac{1}{k_3V^2}\left[-VC(s)\{\hat{\mathbf{k}}\}^T\vartheta - 2\zeta\omega_n\dot{\theta}_m - \omega_n^2\left(\theta_m - \theta^c\right) - kr\right]. \end{aligned} \tag{3.138}$$

Replacing δ_e in the companion model gives us

$$\ddot{\theta}_m = -2\zeta\omega_n\dot{\theta}_m - \omega_n^2\theta_m + (1 - C(s))\{\hat{\mathbf{k}}\}^T\vartheta. \tag{3.139}$$

Thus, the high frequency content of the estimates remains in the companion model.

3.2.4 Physically Motivated Model

In this section, a reference model for an MRAC controller physically motivated by airplane dynamics is introduced. This model is shown to be stable. An MRAC is developed using the model, and then the model is modified to be a companion model for use in developing the $\mathcal{L}_1$ controller.

The physically motived, stable model is defined as

$$\begin{aligned} q_d &= -k_\theta\left(\theta_m - \theta_c\right), && (3.140) \\ \dot{\theta}_m &= q_m \cos\phi + \dot{\theta}_c, \text{ and} \\ \dot{q}_m &= -k_q\left(q_m - q_d\right) + \dot{q}_d + \frac{1}{k_\theta} q_d \cos\phi. && (3.141) \end{aligned}$$

Stability is shown using the following Lyapunov equation and take its derivative

$$\begin{aligned} \mathcal{V} &= \frac{1}{2}(\theta_m - \theta_c) + \frac{1}{2}(q_m - q_d) \text{ and} \\ \dot{\mathcal{V}} &= (\theta_m - \theta_c)\left(q_m \cos\phi + \dot{\theta}_c - \dot{\theta}_c\right) + \\ &\quad (q_m - q_d)\left(-k_q\left(q_m - q_d\right) + \dot{q}_d + \frac{1}{k_\theta} q_d \cos\phi - \dot{q}_d\right). \end{aligned}$$

Noting that $(\theta_m - \theta_c) = -\frac{1}{k_\theta} q_d$ from Equation (3.140), the derivative becomes

$$\begin{aligned} \dot{\mathcal{V}} &= -\frac{1}{k_\theta} q_d q_m \cos\phi - k_q\left(q_m - q_d\right)^2 + \frac{1}{k_\theta} q_d q_m \cos\phi - \frac{1}{k_\theta} q_d^2 \cos\phi, \\ &= -k_q\left(q_m - q_d\right)^2 - \frac{1}{k_\theta} q_d^2 \cos\phi, \\ &= -k_q\left(q_m - q_d\right)^2 - \frac{1}{k_\theta}\left(-k_\theta\left(\theta_m - \theta_c\right)\right)^2 \cos\phi, \text{ and} \\ &= -k_q\left(q_m - q_d\right)^2 - k_\theta\left(\theta_m - \theta_c\right)^2 \cos\phi. && (3.142) \end{aligned}$$

Note that ϕ must be restricted such that $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$ forcing $\cos\phi$ to be positive. Thus, $q_m \to q_d$ and $\theta_m \to \theta_c$ asymptotically. Furthermore, because $q_d = -k_\theta\left(\theta_m - \theta_c\right)$ as $\theta_m \to \theta_c$, $q_d \to 0$ which implies $q_m \to 0$; therefore, the model is stable.

Recalling airplane dynamics we have

$$\begin{aligned}
\dot{\theta} &= q\cos\phi - r\sin\phi \text{ and} \\
\dot{q} &= \Gamma_4 pr - \Gamma_5\left(p_2 - r_2\right) + \qquad (3.143) \\
&\quad \frac{1}{2J_y}\rho V^2 \bar{c} S\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{\bar{c}q}{V} + C_{\delta_e}\delta_e\right]. \qquad (3.144)
\end{aligned}$$

Defining $\tilde{\theta} = \theta - \theta_m$ and $\tilde{q} = q - q_m$ yields

$$\begin{aligned}
\dot{\tilde{\theta}} &= \tilde{q}\cos\phi - r\sin\phi - \dot{\theta}_c \text{ and} \qquad (3.145) \\
\dot{\tilde{q}} &= \Gamma_4 pr - \Gamma_5\left(p_2 - r_2\right) + \qquad (3.146) \\
&\quad \frac{1}{2J_y}\rho V^2 \bar{c} S\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{\bar{c}q}{V} + C_{\delta_e}\delta_e\right] - q_m. \qquad (3.147)
\end{aligned}$$

Because the input to the error system comes in on the $\tilde{q}$ channel, it is unclear how $\tilde{\theta}$ will be affected by the input. We will use backstepping to overcome this problem. Treating $\tilde{q}$ like the input to Equation (3.145) gives us the backstepping variable as

$$\tilde{q}_{des} = \frac{1}{\cos\phi}\left(r\sin\phi + \dot{\theta}_c - \lambda_1\tilde{\theta}\right). \qquad (3.148)$$

By inspection, upon substituting $\tilde{q}_{des}$ in for $\tilde{q}$ makes $\tilde{\theta}$ stable. Using

$$e = \tilde{q} - \tilde{q}_{des}$$

as the error between the pitch rate and the backstepping variable, and through a change of variables the error system becomes

$$\begin{aligned}
\dot{\tilde{\theta}} &= e\cos\phi - \lambda_1\tilde{\theta} \text{ and} \qquad (3.149) \\
\dot{e} &= \Gamma_4 pr - \Gamma_5\left(p_2 - r_2\right) + \qquad (3.150) \\
&\quad \frac{1}{2J_y}\rho V^2 \bar{c} S\left[C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{\bar{c}q}{V} + C_{\delta_e}\delta_e\right] - q_m - \dot{\tilde{q}}_{des}. \qquad (3.151)
\end{aligned}$$

To further simplify, let

$$\begin{aligned}\dot{e} &= k_1 pr - k_2\left(p_2 - r_2\right) + k_3 V^2 + k_4 V^2 \alpha + \\ & \quad k_5 V q + k_6 V^2 \delta_e - q_m - \dot{\tilde{q}}_{des} \text{ and} \\ \dot{e} &= V^2 \vartheta^T \mathbf{k} + k_6 V^2 \delta_e - q_m - \dot{\tilde{q}}_{des}, \qquad (3.152)\end{aligned}$$

where

$$\vartheta = \begin{bmatrix} \frac{pr}{V^2} \\ \frac{-(p^2-r^2)}{V^2} \\ 1 \\ \alpha \\ \frac{q}{V} \end{bmatrix} \text{ and } \mathbf{k} = \begin{bmatrix} (k_1 = \Gamma_4) \\ (k_2 = \Gamma_5) \\ \left(k_3 = \frac{1}{2J_y}\rho\bar{c}SC_{m_0}\right) \\ \left(k_4 = \frac{1}{2J_y}\rho\bar{c}SC_{m_\alpha}\right) \\ \left(k_5 = \frac{1}{2J_y}\rho\bar{c}^2SC_{m_q}\right) \\ \left(k_6 = \frac{1}{2J_y}\rho\bar{c}SC_{m_{\delta_e}}\right) \end{bmatrix}.$$

In order to design the MRAC controller, define the positive definite Lyapunov function candidate

$$\mathcal{V}_1 = \frac{1}{2}\tilde{\theta}^2 + \frac{1}{2}e^2,$$

whose derivative is

$$\begin{aligned}\dot{\mathcal{V}}_1 &= \tilde{\theta}\left(e\cos\phi - \lambda_1\tilde{\theta}\right) + e\left(V^2\vartheta^T\mathbf{k} + k_6 V^2\delta_e - q_m - \dot{\tilde{q}}_{des}\right) \text{ and} \\ &= -\lambda_1\tilde{\theta}^2 + e\left(\tilde{\theta}\cos\phi + V^2\vartheta^T\mathbf{k} + k_6V^2\delta_e - q_m - \dot{\tilde{q}}_{des}\right). \qquad (3.153)\end{aligned}$$

Selecting δ_e to be

$$\delta_e^{des} = -\frac{1}{k_6}\vartheta^T\mathbf{k} - \frac{1}{k_6V^2}\left(\tilde{\theta}\cos\phi - q_m - \dot{\tilde{q}}_{des} + \lambda_2 e\right)$$

would stabilize the system. However, since the parameters are being estimating, we really have

$$\delta_e = -\frac{1}{k_6}\vartheta^T\hat{\mathbf{k}} - \frac{1}{k_6V^2}\left(\tilde{\theta}\cos\phi - q_m - \dot{\tilde{q}}_{des} + \lambda_2 e\right). \qquad (3.154)$$

Adding and subtracting δ_e^{des} in Equation (3.153) gives

$$\begin{aligned}\dot{\mathcal{V}}_1 &= -\lambda_1\tilde{\theta}^2 + e\left(\tilde{\theta}\cos\phi + V^2\vartheta^T\mathbf{k} + k_6V^2(\delta_e - \delta_e^{des} + \delta_e^{des}) - q_m - \dot{\tilde{q}}_{des}\right), \\ &= -\lambda_1\tilde{\theta}^2 + e\left(\tilde{\theta}\cos\phi + V^2\vartheta^T\mathbf{k} + k_6V^2(-\frac{1}{k_6}\vartheta^T\tilde{\mathbf{k}} + \delta_e^{des}) - q_m - \dot{\tilde{q}}_{des}\right), \text{ and} \\ &= -\lambda_1\tilde{\theta}^2 - \lambda_2 e^2 + \left(eV^2\vartheta^T\tilde{\mathbf{k}}\right), \end{aligned} \tag{3.155}$$

where $\tilde{\mathbf{k}} = \hat{\mathbf{k}} - \mathbf{k}$. Noting that $\dot{\tilde{\mathbf{k}}} = \dot{\hat{\mathbf{k}}} - \dot{\mathbf{k}} = \dot{\hat{\mathbf{k}}}$, the new Lyapunov function candidate is

$$\begin{aligned}\mathcal{V} &= \mathcal{V}_1 + \frac{1}{2}\tilde{\mathbf{k}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \\ \dot{\mathcal{V}} &= -\lambda_1\tilde{\theta}^2 - \lambda_2 e + \left(-eV^2\vartheta^T\right)\tilde{\mathbf{k}} + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\tilde{\mathbf{k}}, \text{ and} \\ \dot{\mathcal{V}} &= -\lambda_1\tilde{\theta}^2 - \lambda_2 e^2 + \left(-eV^2\vartheta^T + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1}\right)\tilde{\mathbf{k}}.\end{aligned}$$

By setting $-eV^2\vartheta^T + \dot{\hat{\mathbf{k}}}^T\Lambda^{-1} = 0$ we get the update equation

$$\dot{\hat{\mathbf{k}}} = (\tilde{q} - \tilde{q}_{des})\, V^2\Lambda\vartheta \tag{3.156}$$

where $e = \tilde{q} - \tilde{q}_{des}$. Using the projection operator the update equation becomes

$$\dot{\hat{\mathbf{k}}} = \mathrm{Proj}_{S_{12}}\left((\tilde{q} - \tilde{q}_{des})\, V^2\Lambda\vartheta, \hat{\mathbf{k}}\right). \tag{3.157}$$

Slightly modifying the previous model gives the companion model

$$\begin{aligned} q_d &= -k_\theta\left(\theta_m - \theta_c\right), \\ \dot{\theta}_m &= q_m\cos\phi + \dot{\theta}_c, \text{ and} \\ \dot{q}_m &= -k_q q_m + V^2k_6\left(\delta_e + \frac{1}{k_6}\vartheta^T\hat{\mathbf{k}} - \frac{-k_q q_m - \tilde{\theta}\cos\phi + \dot{\tilde{q}}_{des} - \lambda_2 e}{V^2k_6}\right). \end{aligned} \tag{3.158}$$

It can be noted that the companion model is equivalent to the previous model when δ_e is substituted to give us the companion model. However, if we low-pass filter the control signal the companion model now takes on the high frequency content of the parameter estimates.

The $\mathcal{L}_1$ controller is defined as

$$
\begin{aligned}
q_d &= -k_\theta \left(\theta_m - \theta_c\right) \\
\tilde{q}_{des} &= \frac{1}{\cos\phi}\left(r\sin\phi + \dot{\theta}_c - \lambda_1\tilde{\theta}\right), \\
\dot{\theta}_m &= q_m\cos\phi + \dot{\theta}_c, \\
\dot{q}_m &= -k_q q_m + V^2 k_6\left(\delta_e + \frac{1}{k_6}\vartheta^T\hat{\mathbf{k}} - \frac{-k_q q_m - \tilde{\theta}\cos\phi + \dot{\tilde{q}}_{des} - \lambda_2 e}{V^2 k_6}\right), \\
\delta_e &= -\frac{1}{k_6}\vartheta^T C(s)\left\{\hat{\mathbf{k}}\right\} - \frac{1}{k_6 V^2}\left(\tilde{\theta}\cos\phi - q_m - \dot{\tilde{q}}_{des} + \lambda_2 e\right), \text{ and} \\
\dot{\hat{\mathbf{k}}} &= \text{Proj}_{S_{12}}\left(\left(\tilde{q} - \tilde{q}_{des}\right)V^2\Lambda\vartheta, \hat{\mathbf{k}}\right).
\end{aligned} \tag{3.159}
$$

Chapter 4

Experimental Platform

Having developed the MRAC and $\mathcal{L}_1$ algorithms in the previous chapter, their effectiveness needs to be tested. This is accomplished by running simulations, hardware-in-the-loop tests, and flight tests. This chapter describes the platforms upon which these tests are run.

4.1 Simulation

After deriving the adaptive control schemes in Chapter 3, these schemes are simulated in Simulink/Matlab 7. Matlab 7 is executed on a Pentium M 1.6 GHz processor with 1 GB memory running Windows XP. Figure 4.1 shows the Simulink setup. Both the MRAC and $\mathcal{L}_1$ simulations have the same block overview. Desired pitch and roll angles (called "th_c" and "phi_c", respectively in the figure) are sent to the controller. The controller issues elevator, aileron, rudder, and throttle commands (labeled "delta") to the UAV equations of motion (EOM) block. The UAV EOM s-function calculates the attitude and position of the UAV using the 6-DOF model from Section 2.2 with the parameter values in Table 4.1. These parameters are similar to the parameters on the test MAVs. The UAV states are fed back to the controller from the UAV EOM.

Figure 4.2 is the inside of the control block used for MRAC testing. All four Lyapunov MRAC controllers have this block outline. However they differ by the code used inside the pitch and roll s-function blocks. The commanded pitch and roll angles are fed into their respective MRAC controllers. The controllers are s-function blocks that accept commanded angles and aircraft states (labeled "x"–the algorithms

Table 4.1: Aerodynamic coefficients used in the simulations.

Longitudinal Coefficients	Lateral Coefficients
$C_{L_0} = 0.28$	$C_{Y7_0} = 0$
$C_{D_0} = 0.03$	$C_{\ell_0} = 0$
$C_{m_0} = 0$	$C_{n_0} = 0$
$C_{L_\alpha} = 3.45$	$C_{Y_\beta} = -0.98$
$C_{D_\alpha} = 0.30$	$C_{\ell_\beta} = -0.12$
$C_{m_\alpha} = -0.38$	$C_{n_\beta} = 0.25$
$C_{L_q} = 0$	$C_{Y_p} = 0$
$C_{D_q} = 0$	$C_{l_p} = -0.26$
$C_{m_q} = -3.6$	$C_{n_p} = 0.022$
$C_{L_{\delta_e}} = -0.36$	$C_{Y_r} = 0$
$C_{D_{\delta_e}} = 0$	$C_{\ell_r} = 0.14$
$C_{m_{\delta_e}} = 0.5$	$C_{n_r} = -0.35$
	$C_{Y_{\delta_a}} = 0$
	$C_{\ell_{\delta_a}} = 0.08$
	$C_{n_{\delta_a}} = 0.06$
	$C_{Y_{\delta_r}} = -0.17$
	$C_{\ell_{\delta_r}} = 0.105$
	$C_{n_{\delta_r}} = -0.032$

Table 4.2: Physical parameters used in the simulations.

Physical Parameters
$m = 1.56$ kg
$S = 0.2589$ m^2
$b = 1.4224$ m^2
$\bar{c} = 0.3302$ m^2
$S_{\text{prop}} = .0314$ m^2
P $= 101.3 \times 10^3$ N/m^2
$T = 278$ K
$g = 9.80665$ m/s^2
$\lambda = -0.0065$ K/m
$R_a = 8314.32$ J/K/kmol
$p_0 = 101325$ N/m
$R = 287.05$ J/K/kg
$M_0 = 28.9644$ kg/kmole
$R_{\text{earth}} = 6371020$ m
$\rho = 1.2682$ kg/m^3
$J = \begin{pmatrix} 0.1147 & 0 & -0.0015 \\ 0 & 0.0576 & 0 \\ -0.0015 & 0 & 0.1712 \end{pmatrix}$ kg m^2

are allowed full state feedback except for α and β) to calculate elevator or aileron deflections. Notice that the throttle and rudder commands are constants.

Figure 4.3 shows the inside of the control block for the $\mathcal{L}_1$ controller. It is similar to the MRAC environment except the parameter estimates are low pass filtered and added to the command signal. Also, the command signals are sent to the UAV EOM and fed back to the controller.

In this environment, gains can be tuned over a variety of test conditions. The testing involves commanding step changes in commanded roll and pitch angles by ± 15 degrees at 0.05 Hz, and plotting the results. By running the pitch and roll controllers simultaneously, coupling effects between roll and pitch affect are observed. The throttle is held at the constant value, 13.2494 m/s (a value to keep the aircraft in wings-level trim), while the rudder is held at zero. While testing, the controller parameters, aircraft and model attitude (pitch and roll), and control surface deflection are monitored. The results are presented in Chapter 5.

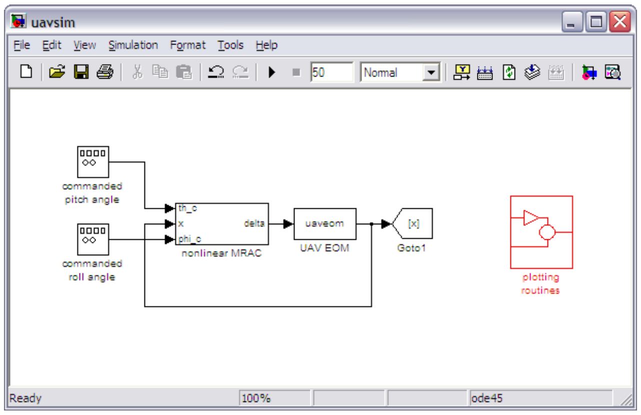


Figure 4.1: All algorithms were run in this basic Simulink block diagram.

Once tested in Matlab, the adaptive algorithms are coded in Dynamic C, a C based programming language for Rabbit microcontrollers. Next, this code is emulated on the freeware flight simulator, Aviones (see Figure 4.4). Aviones is a three dimensional flight simulator developed for creating/testing autopilots, state estimators,

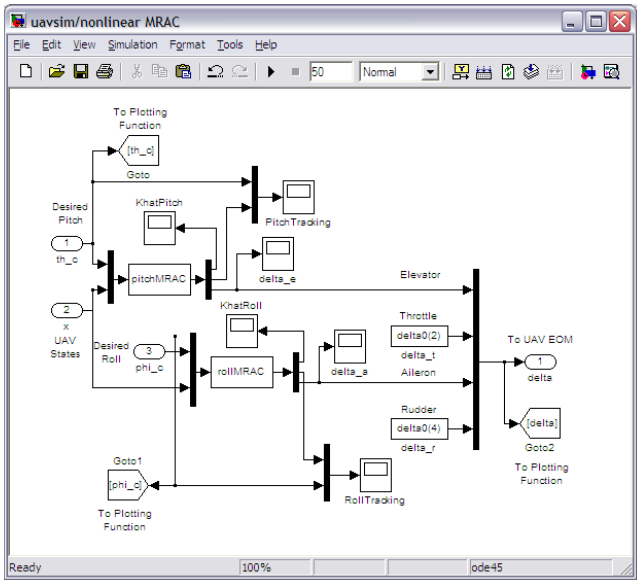


Figure 4.2: MRAC block diagram.

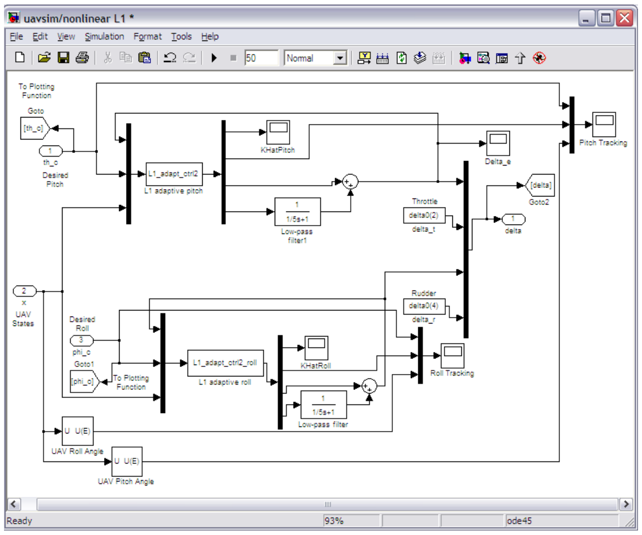
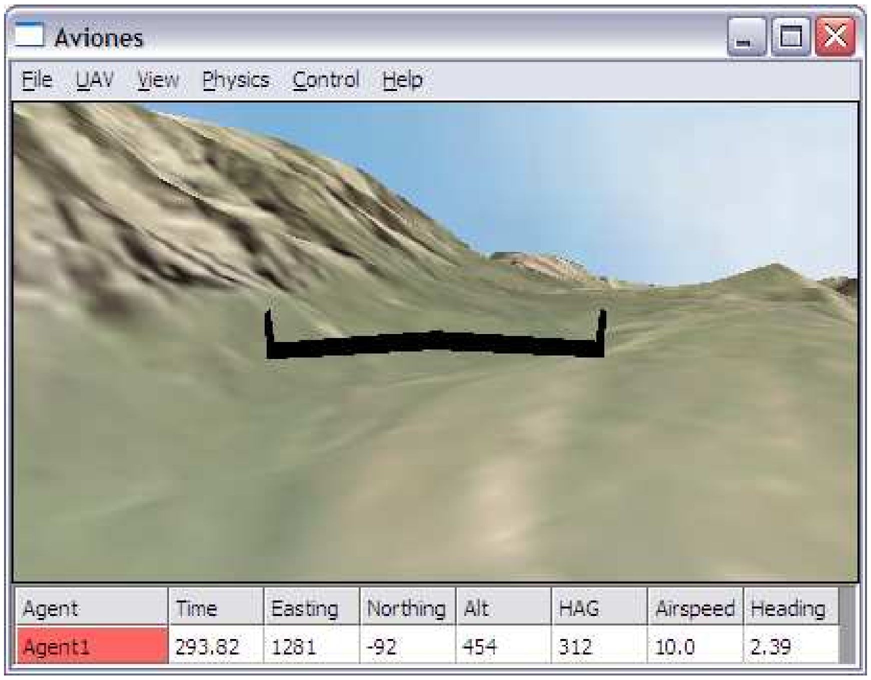


Figure 4.3: $\mathcal{L}_1$ controller block diagram.

and communication techniques [24]. Aviones is implemented on the Windows operating system and can be downloaded from SourceForge at http://sourceforge.net/projects/aviones. Aviones simulates every aspect of flight by emulating the autopilot, implementing the physics of the MAVs, and connecting to the ground station software Virtual Cockpit (VC) via the network loopback. The VC is the software that communicates with the autopilot to give waypoint information and online parame-

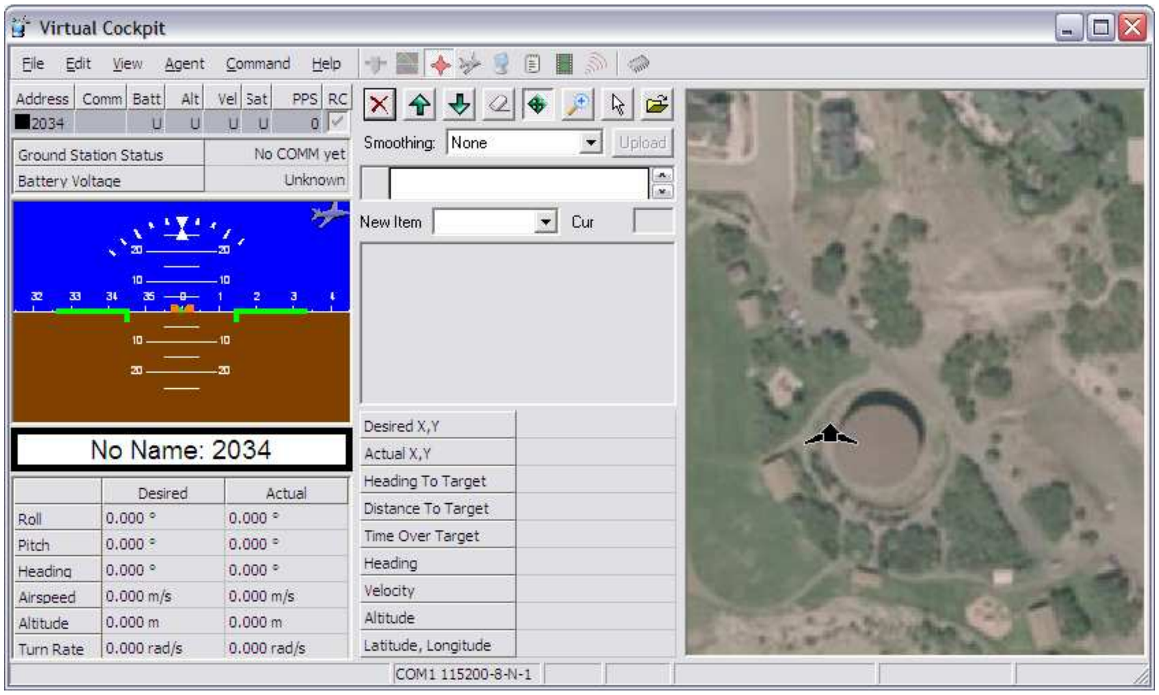


Agent	Time	Easting	Northing	Alt	HAG	Airspeed	Heading
Agent1	293.82	1281	-92	454	312	10.0	2.39

Figure 4.4: A screenshot from Aviones. Aviones is a software package that can run autopilot code and interface with the Virtual Cockpit directly, or it can feed sensor information and receive elevator/aileron/rudder commands from a Kestrel Autopilot.

ter changes (i.e. PID gains, adaptive gains, etc.), and receives telemetry, video, and aircraft position and attitude. A screenshot of VC is found in Figure 4.5.

By running the autopilot code in Aviones testing for coding errors, observe the effect of code changes and gain adjustments, and observe trends without risking damage to our test MAVs is accomplished. Aviones was run on a Pentium 4 3GHz machine running Windows XP with 512 MB memory. Aviones uses the same 6-DOF model that used in Simulink. While testing with a simulated aircraft in Aviones, the aircraft was commanded to fly an hourglass pattern with 200 meter edges and 300 meter diagonals (see Figure 4.6). The hourglass pattern ensures that right and left turns have equal representation. Flying this pattern also provides consistency between simulations and real flight tests, since same hourglass path is used.



Address	Comm	Batt	Alt	Vel	Sat	PPS	RC
2034		U	U	U	U	0	✓

Ground Station Status	No COMM yet
Battery Voltage	Unknown

Smoothing: None

No Name: 2034

	Desired	Actual
Roll	0.000 °	0.000 °
Pitch	0.000 °	0.000 °
Heading	0.000 °	0.000 °
Airspeed	0.000 m/s	0.000 m/s
Altitude	0.000 m	0.000 m
Turn Rate	0.000 rad/s	0.000 rad/s

Desired X,Y	
Actual X,Y	
Heading To Target	
Distance To Target	
Time Over Target	
Heading	
Velocity	
Altitude	
Latitude, Longitude	

Figure 4.5: A screenshot of the Virtual Cockpit ground station. Virtual Cockpit sends waypoint data and receives telemetry either for Aviones or the Kestrel Autopilot.

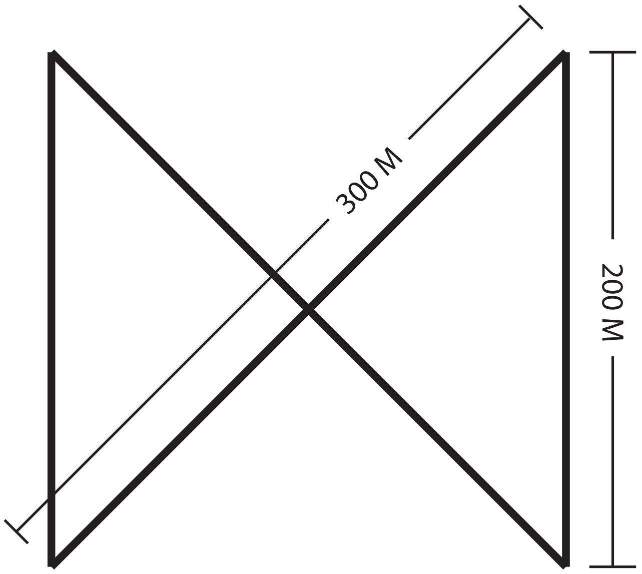


Figure 4.6: Diagram of the hourglass pattern used in simulation and flight testing of the algorithms.

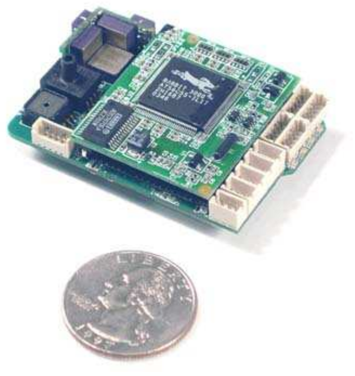


Figure 4.7: The Kestrel Autopilot.

4.2 Hardware

After simulations are complete, hardware-in-the-loop and flight tests are done. The Kestrel Autopilot [25] is used for the hardware-in-the-loop (HIL) and flight tests. The two versions of autopilot used in testing are v1.45 and v2.0. The Kestrel Autopilot, seen in Figure 4.7, has a 29 MHz Rabbit microcontroller (512K Flash and 512K RAM) and the following on board sensors: rate gyros, accelerometers, an absolute pressure sensor for measuring altitude, a differential pressure sensor for measuring airspeed, and a GPS receiver. The autopilot is programmed using Dynamic C and cycles through all subsystems at about 80-90 Hz.

The Kestrel Autopilot is connected through two Programming Cable with two Pigtails (Part# R101-0513+ADAP [25]) attached to two serial ports on a PC running both Aviones and the Virtual Cockpit (VC) during HIL testing. Figure 4.8 shows this relationship. Aviones simulates the world sending sensor information to the autopilot and accepts elevator, aileron, rudder, and throttle commands. The VC provides waypoint information and parameter changes to the autopilot while receiving telemetry. HIL tests help determine if the limited processing speed of the Kestrel Autopilot will cause problems and allow for the adaptive algorithms to be tuned to the Kestrel.

If HIL tests are successful, flight tests can be run using the following planes: a flying wing named Deliverance, a larger flying wing named Ray, and a fixed-wing

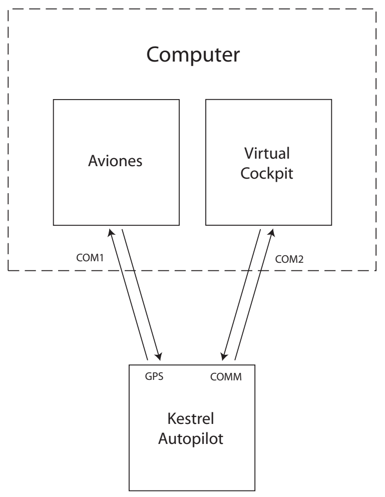


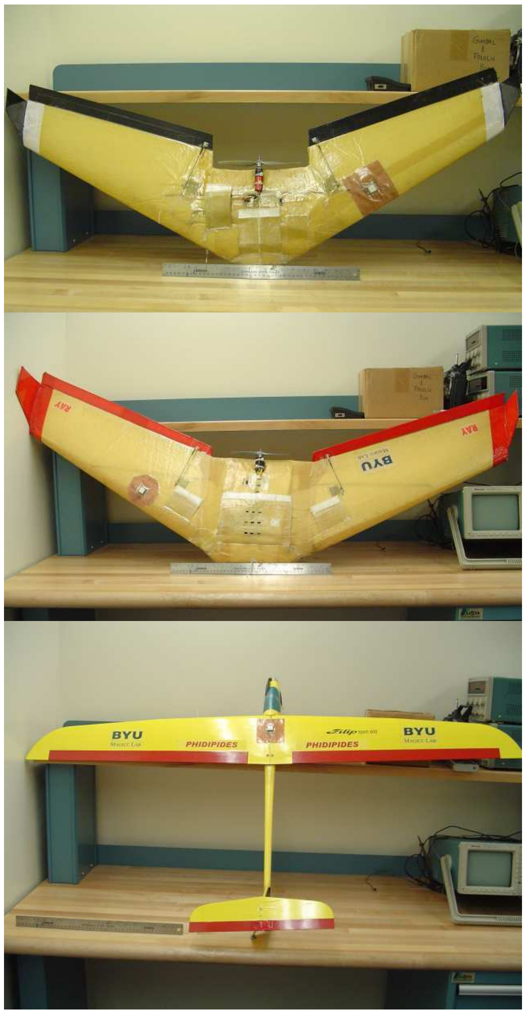
Figure 4.8: Diagram of the HIL communications.

high-efficiency aircraft named Phidipides. Figure 4.9 shows the three test aircraft. These aircraft use Kestrel Autopilots that communicate through an RF modem to a ground station running Virtual Cockpit. The ground station is a PC running the VC attached via a RS232 serial cable to an RF Comm Box that communicates with the autopilot. The RF Comm box also attaches to a Futaba T9CAP RC controller via a training cable to allow an RC pilot to take command of the aircraft. They receive waypoint commands and parameter changes from the VC while sending telemetry to the VC. Using Virtual Cockpit, adaptive control parameters are set and adaptive algorithms are turned on or off mid-flight. For a illustration of the ground station setup see Figure 4.10.

Below is a parts list for the ground station available from [25]:

- RS232 serial cable (Part# RS232)
- RF Comm Box (Part# PRTGS1.1)
- Futaba T9CAP RC Controller
- training cable (Part# RCTRAIN3)
- Virtual Cockpit (MAGICC Lab version)

Below is the list of components for Deliverance and Ray:



Deliverance
Weight: 0.734 kg
Wingspan: 1.207 m
Mean geometric chord: 0.254 m

Ray
Weight: 1.321 kg
Wingspan: 1.511 m
Mean geometric chord: 0.305 m

Phidipides
Weight: 0.969 kg
Wingspan: 1.524 m
Mean geometric chord: 0.168 m
Nose to tail: 0.914 m

Figure 4.9: RC-style Test Aircraft. The ruler is added to give scale.

- Furuno GH80 GPS Receiver
- Aerocomm 4490-1000M 900 MHz RF Modem
- Mega 16/15/5 or 16/15/6 Electric Motor
- Castle Creations Phoenix-35 Motor Controller
- Kokam 11.1 V 1500 mAh Lithium Polymer Batteries
- Hitec HS 81 MG Servos

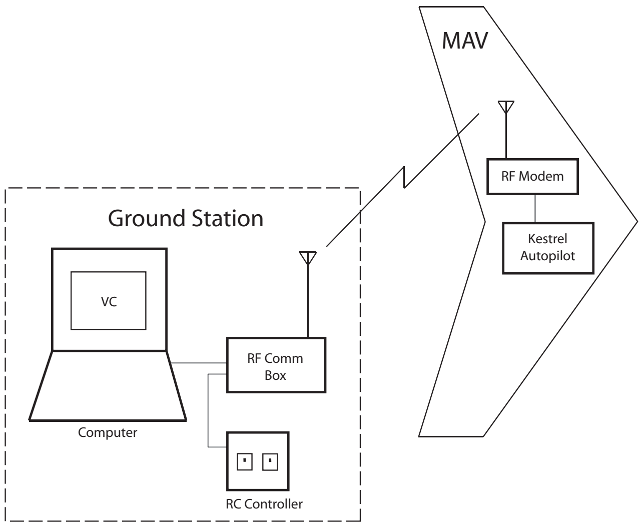


Figure 4.10: Diagram of the communication components. The ground station consists of the VC running on a computer attached to an RF Comm Box which transmits to the aircraft. The RF Comm Box is also attached to an RC Controller to allow for a pilot to take control of the aircraft. The aircraft receives and sends data to the ground station via an RF modem attached to the Kestrel Autopilot.

- Procerus Kestrel Autopilot v1.45 and v2.0
- Dipole Antenna (900MHz)

Phidipides uses the same components with the exception of Hitec HS 55 servos and an Axi brushless motor.

To test robustness and speed of adaption, a flap is attached to Deliverance. The flap is shown in Figure 4.11. Sending commands using the VC, the flap can be deployed and retracted. Deployment significantly changes the aerodynamic and lift coefficients for deliverance.

While flying, the aircraft and internal model attitude (pitch and roll), controller parameters, and control deflections are monitored. The results are presented in the next chapter.

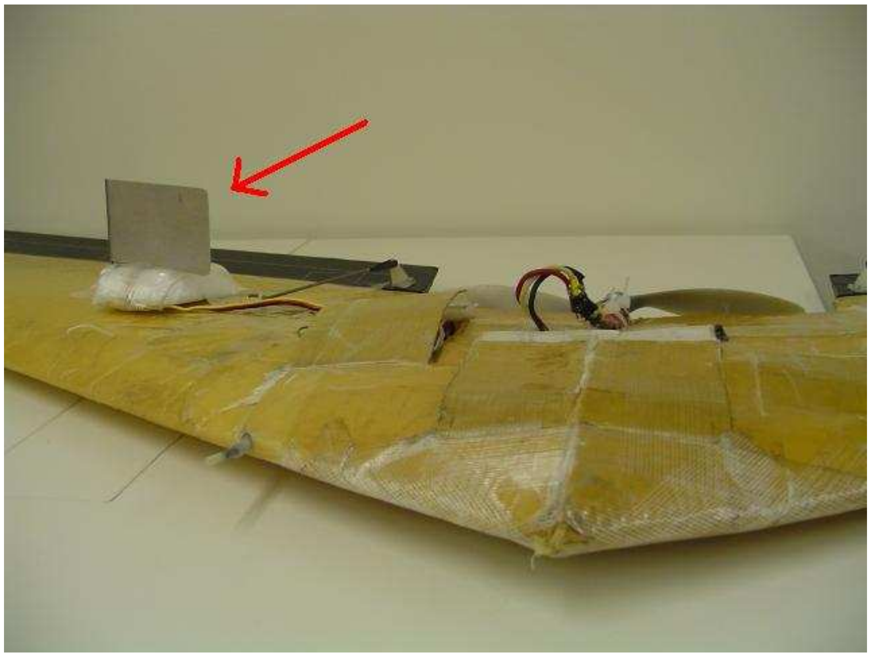


Figure 4.11: Flap deployed on Deliverance.

Chapter 5

Results

This chapter discusses the results from both simulation and flight testing of the Lyapunov MRACs and $\mathcal{L}_1$ controllers derived in Chapter 3. It is divided into two sections. The first section contains the simulations results. The second section contains the hardware results. In both sections, plots are analyzed and compared.

5.1 Simulation Results

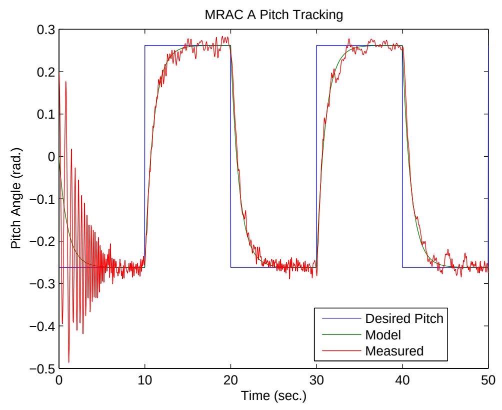
The purpose of this section is to present the results from simulations performed using Matlab described in Section 4.1. This section is split into four parts. The first part describes the Lyapunov based MRAC results. The second part discusses the results of the different $\mathcal{L}_1$ controllers. The third section analyzes and compares the $\mathcal{L}_1$ controllers and Lyapunov MRACs. Finally, the last section is the summary.

5.1.1 Lyapunov Based MRAC

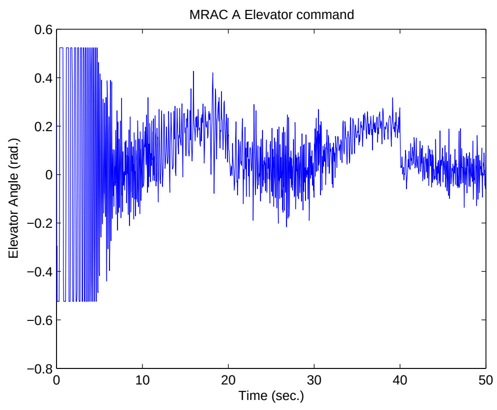
Below are the results of the four Lyapunov MRAC Schemes derived in Section 3.1. All Matlab code is located in Appendix A.1.

MRAC Scheme A

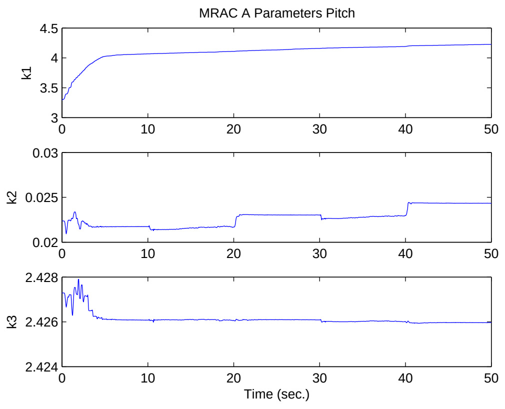
Figure 5.1 and 5.2 show pitch and roll results, respectively. Both the pitch and roll controllers performances improved over time. Both had an initial period before the first step change where the aircraft oscillated about the model. Also, neither the pitch nor the roll controller's estimates converged, which is probably due to a number of factors such as the adaptive gain size, the nature of the reference signal (persistent excitation), and duration of the simulation (see Figures 5.1 and 5.3).



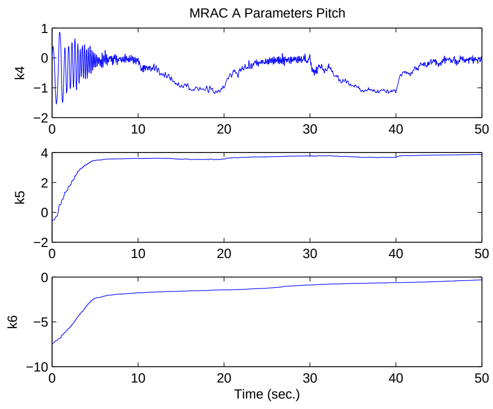
(a) Pitch. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch after the convergence period.



(b) Elevator. High frequency oscillations are typical of this type of MRAC.



(c) Parameters k_1-k_3. Without enough excitation the parameters have not settled on their true values.

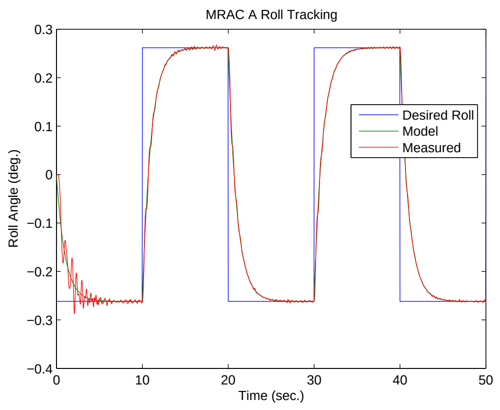


(d) Parameters k_4-k_6. Notice the correlation between k_4 and the oscillations in pitch.

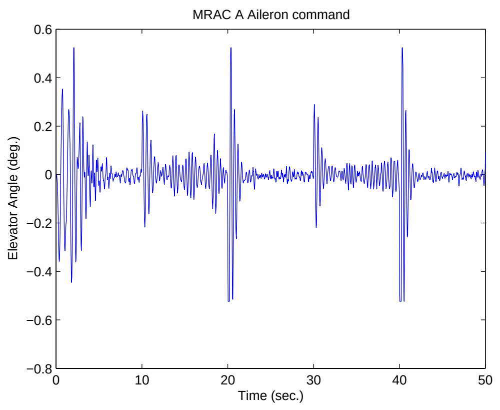
Figure 5.1: MRAC Scheme A Pitch Matlab Results.

MRAC Scheme B

Simulation results for MRAC Scheme B are found in Figures 5.4 and 5.5. Notice the oscillation in the roll angle before the reference command changes–a further example of the need for persistent excitation. Unlike the pitch and roll in the last section, the elevator and aileron commands have a discernible, consistent pattern, compare Figures 5.1(b) and 5.2(b) with 5.4(b) and 5.5(b). Like the parameter estimates in MRAC Scheme A, the parameters have not converged in either pitch or roll as evidenced in 5.4(c), 5.4(d), 5.5(c), and 5.5(d).



(a) Roll. Tracking gets better with each step input.



(b) Aileron deflections. Notice the high frequency content on the output.

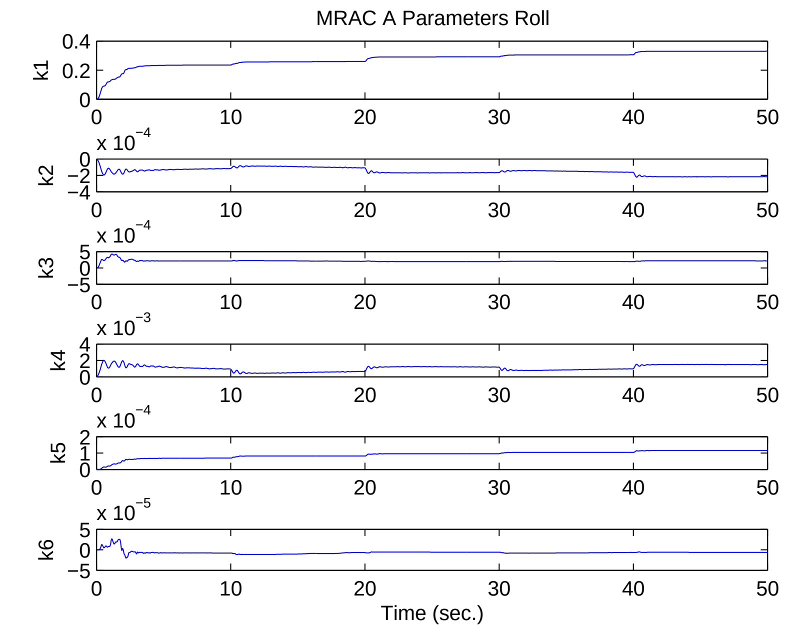
Figure 5.2: MRAC Scheme A Roll Matlab Results.

MRAC Scheme C

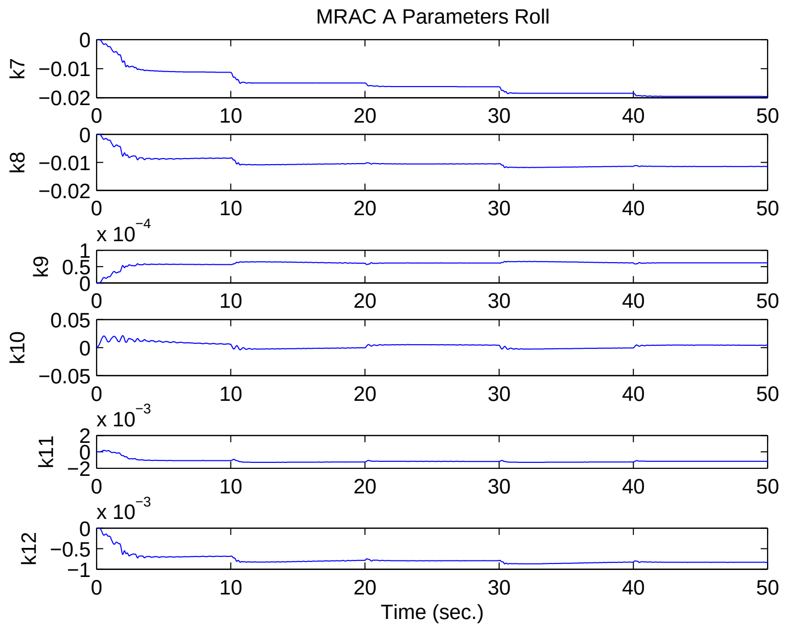
Figures 5.6 and 5.7 show the pitch and roll simulation results for the MRAC Scheme C. The literature seems to indicate that the input signal is important to guarantee parameter convergence. As such the input signal appears to be "persistently exciting" enough as the roll parameters appear to be converging. It would appear that the fewer parameters to be estimated, the less often a signal change is needed, i.e. the less persistent excitation is needed.

MRAC Scheme D

Figures 5.8-5.10 show the simulation results of the MRAC Scheme D. Roll again shows the importance of persistent excitation. Before the reference signal changes at 30 seconds in Figure 5.9(a), there is rapid oscillation that dampens after the step. The role that the reference signal plays is shown by comparing Figure 5.9(a) with Figure 5.10, where the reference signal has been sped up to 0.105 Hz. Parameters appear to be converging in both pitch and roll. Also, notice the oscillations on the elevator and aileron (Figures 5.8(b) and 5.9(b)) that are typical of this type of MRAC. Furthermore, the aileron command saturates at each step change in Figure 5.9(b).

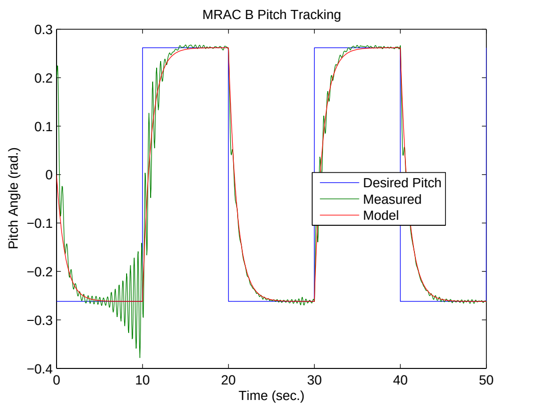


(a) Parameters k_1-k_6.

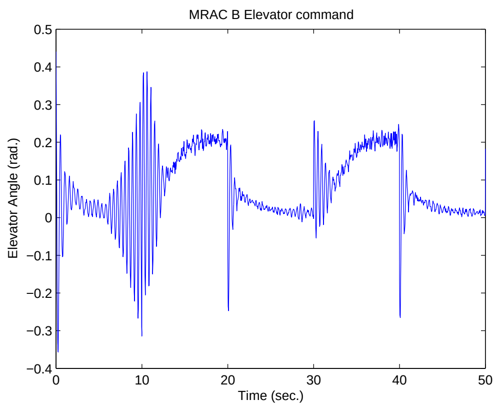


(b) Parameters k_7-k_{12}.

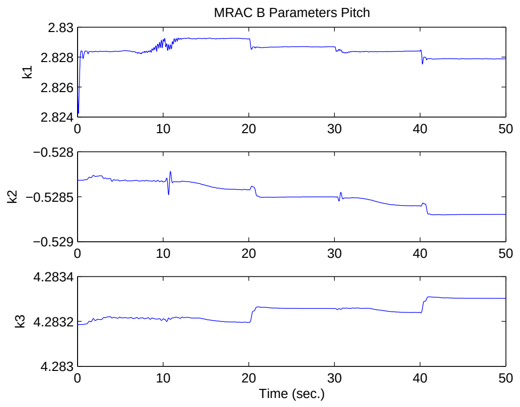
Figure 5.3: MRAC Scheme A Roll Parameter Matlab Results.



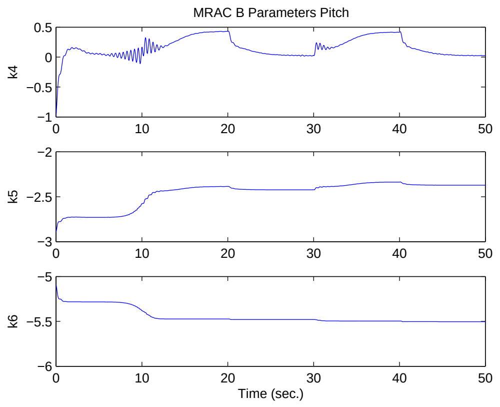
(a) Pitch. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch.



(b) Elevator. Notice the high frequency content typical of this type of MRAC.



(c) Parameters k_1-k_3.

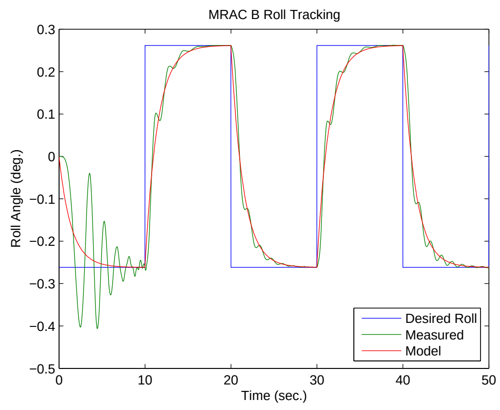


(d) Parameters k_4-k_6.

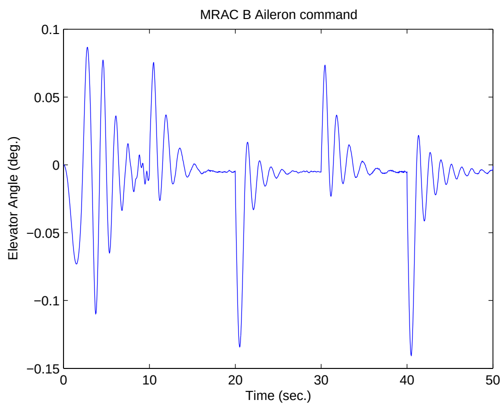
Figure 5.4: MRAC Scheme B Pitch Matlab Results.

5.1.2 $\mathcal{L}_1$ Controllers

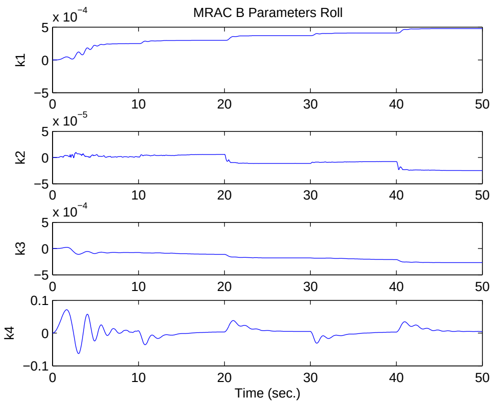
Presented below are the simulation results for the four $\mathcal{L}_1$ controllers derived in Chapter 3. The literature discusses the effect of increasing gain on improved reference signal tracking. Moreover, the $\mathcal{L}_1$ controller does not have oscillations on the control signal like Lyapunov based MRACs as the adaptive gain is increased. To show this effect, each algorithm was run twice with two different adaptive gains of 100 and .1 (adaptive gains higher than 100 slowed some simulations). This section describes the results of these simulations. Appendix B.1 has the complete Matlab code for the simulations.



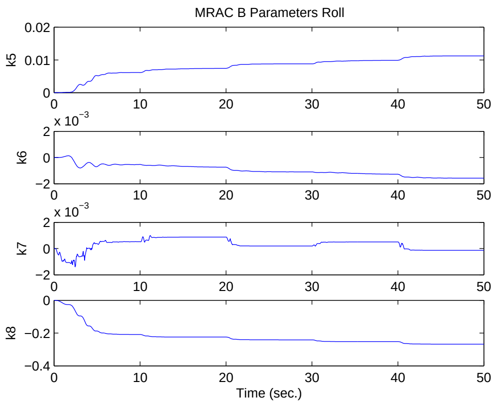
(a) Roll. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch.



(b) Aileron. Notice the high frequency content typical of this type of MRAC.



(c) Parameters k_1-k_4

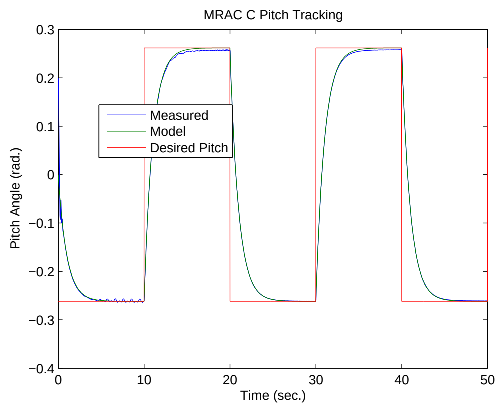


(d) Parameters k_4-k_8

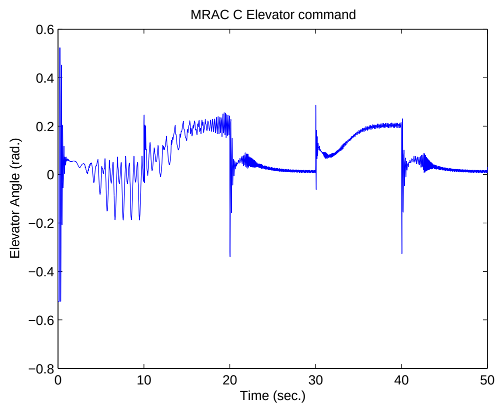
Figure 5.5: MRAC Scheme B Roll Matlab Results.

First-Order Model with One Unknown, Lumped Parameter

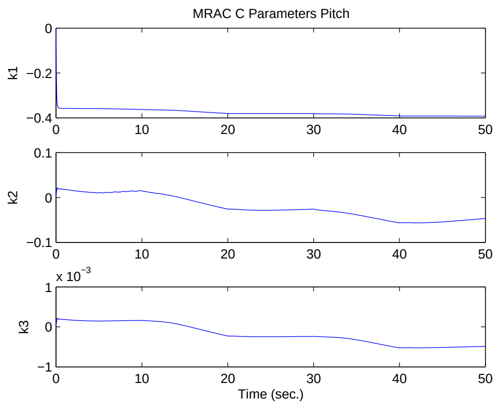
Figure 5.11 shows the Simulink results for the first order model with one unknown parameter. As can be seen in Figures 5.11(a) and 5.11(b), the angle tracking and settling time for pitch is much better with high adaptive gain. The elevator commands between the high gain and low gain simulations are similar. However, the elevator command with high gain in Figure 5.11(c) has more oscillations at edge events than Figure 5.11(d), yet it is smoother for the rest of the commanded signal. The parameters also exhibit the same behavior, though much more pronounced, as the



(a) Pitch. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch.



(b) Elevator. Notice the high frequency content typical of this type of MRAC.



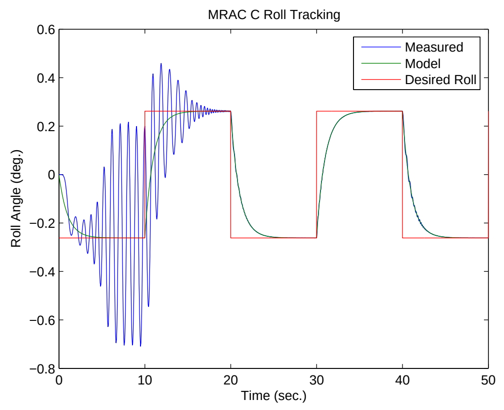
(c)

Figure 5.6: MRAC Scheme C Pitch Matlab Results.

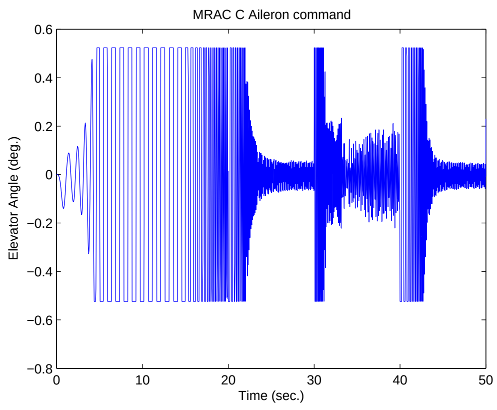
high adaptive gain parameters are a whole order of magnitude greater at their peaks (Figure 5.11(e)) than the low adaptive gain parameters (Figure 5.11(f)).

First-Order Model with Two Lumped, Unknown Parameters

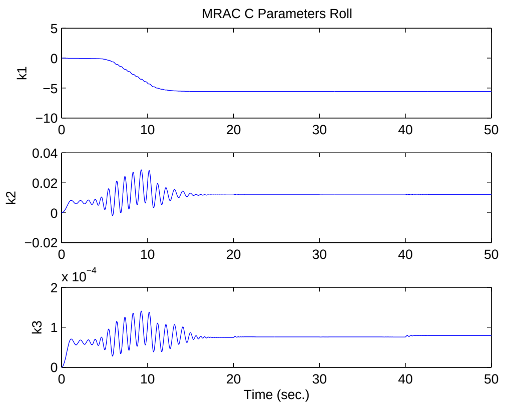
We again see in Figure 5.12 that high adaptive gain has better pitch tracking than low adaptive gain. However, comparing 5.11(a) with 5.12(a) we see that the first-order system with two lumped parameters better approximates a first-order system than the first-order model with only one parameter. The settling time is similar between the two schemes. Examining Figures 5.12(c) and 5.12(d), we see that there is more



(a) Roll. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch.



(b) Aileron. Notice the high frequency content typical of this type of MRAC.



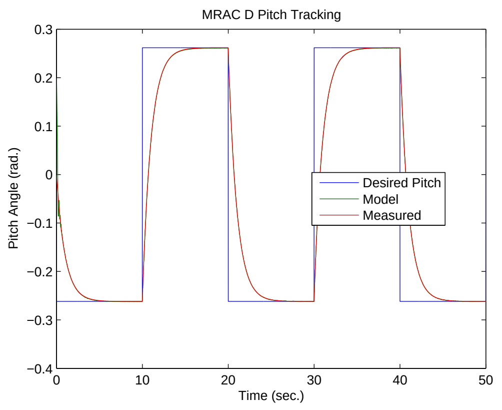
(c) Parameters. The parameters have not yet converged.

Figure 5.7: MRAC Scheme C Roll Matlab Results.

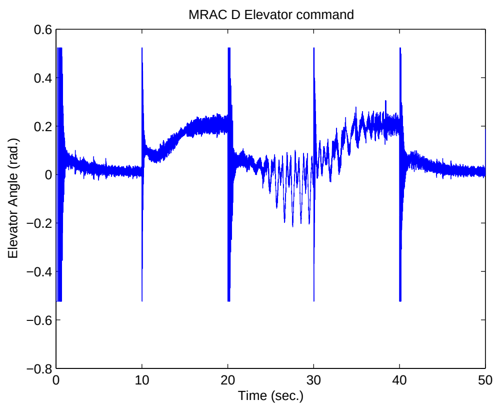
oscillation in the elevator command and parameters when there is high adaptive gain, but this is still reasonable. Figures 5.12(e) and 5.12(f) show that the high adaptive gain simulation seems to settle on parameters better than the low adaptive gain simulation.

Second-Order Model

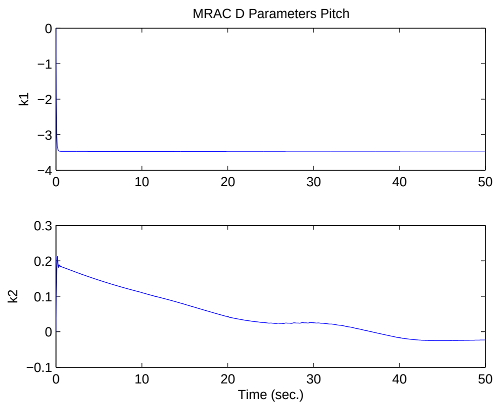
Figure 5.13 shows the simulation results for the second-order model. Comparing Figure 5.13 with Figures 5.11 and 5.12, the second order model performs similarly to the first-order models. However, there are more oscillations with the elevator



(a) Pitch. There is a convergence period after which the airplane tracks the model. Notice the slight oscillation of the actual pitch.



(b) Elevator. Notice the high frequency content typical of this type of MRAC.



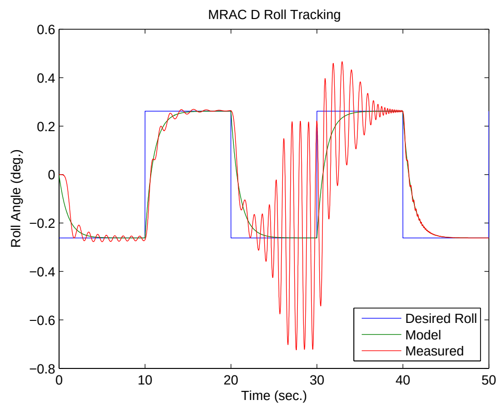
(c) Parameters. The pitch parameters appear to have converged.

Figure 5.8: MRAC Scheme D Pitch Matlab Results.

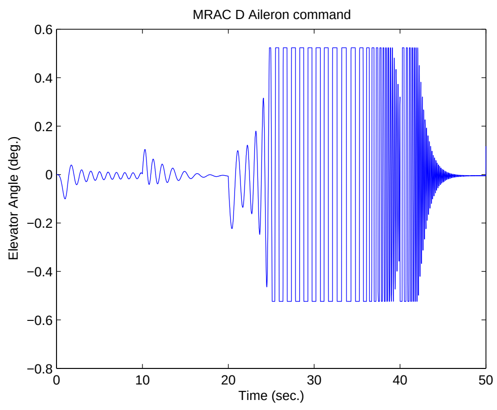
command than in the first order models. Comparing the high adaptive gain plot with the low adaptive gain plot in Figure 5.13 shows an unexpected consistency. It appears that the gains chosen do not effect the performance of the $\mathcal{L}_1$ controller.

Physically Motivated Model

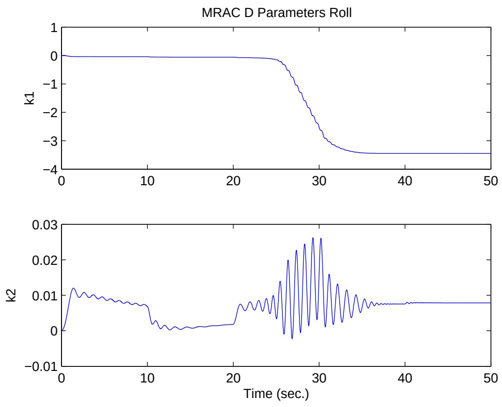
The physically motivated second-order model results are shown in Figure 5.14. This controller has similar elevator command patterns as the previous second-order model (see Figure 5.13). However, the pitch oscillates slightly like the first-order model with one unknown parameter (see Figure 5.11). The parameters oscillate more with



(a) There is a convergence period after which the aircraft tracks the model roll angle. Notice the slight oscillation of the actual roll.



(b) Notice the high frequency content of the aileron command which typical of this type of MRAC.



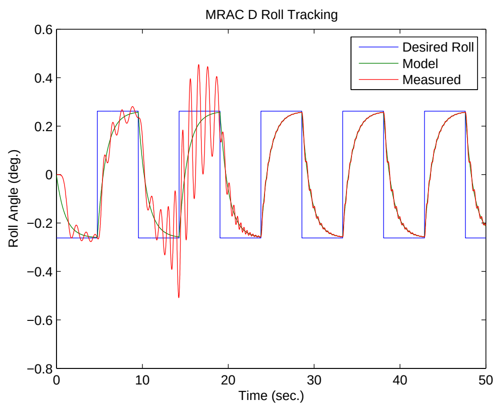
(c) The parameters appear to be converging being sufficiently excited.

Figure 5.9: MRAC Scheme D Roll Matlab Results.

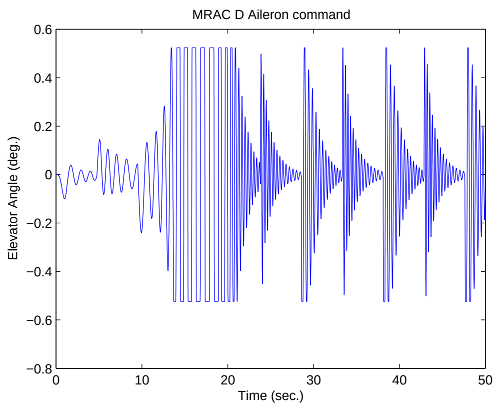
increased adaptive gain (compare Figures 5.15-5.16). However, tracking improves (compare Figure 5.14(a) to 5.14(b)). Oscillations of the elevator command are also reduced by increasing the adaptive gain present, shown in Figure 5.14(c) and 5.14(d).

5.1.3 Analysis

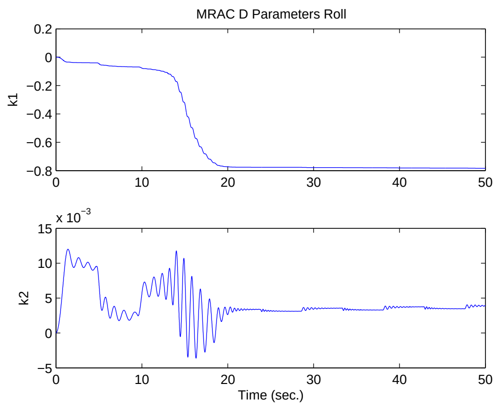
Tables 5.1-5.6 show the metrics used to evaluate the adaptive schemes. The two types of metrics were the average error between actual and model angles and the $\mathcal{L}_2$ norm of that error. Average error between the model and actual angle is defined as:



(a) Notice the roll angle oscillations start and end sooner than Figure 5.9(a).



(b) The aileron has a similar pattern to Figure5.9(b).



(c) The parameters have mostly converged, the slight oscillation remaining might be caused by saturation on the aileron.

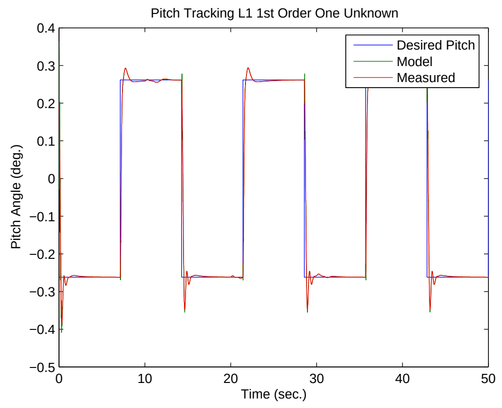
Figure 5.10: MRAC Scheme D Roll Matlab Results with 0.105 Hz Reference signal.

$$e_{avg} = \frac{1}{n}\sum_{i=1}^{n}|\alpha^d(i) - \alpha(i)|,$$

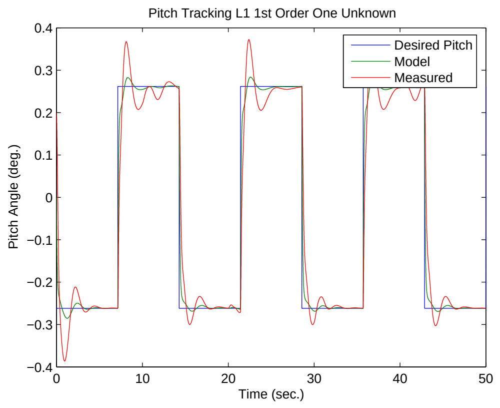
and the $\mathcal{L}_2$ norm of the error is defined as

$$e_{\mathcal{L}_2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\alpha^d(i) - \alpha(i))^2},$$

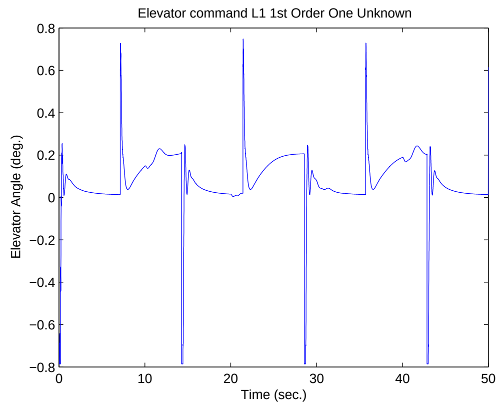
where α_d is the desired pitch or roll angle and α is the actual pitch or roll angle.



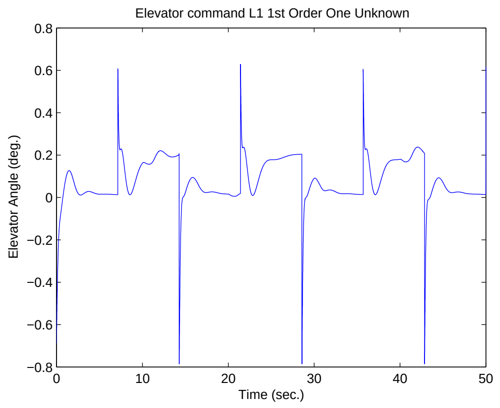
(a) Pitch with high adaptive gains that are 100.



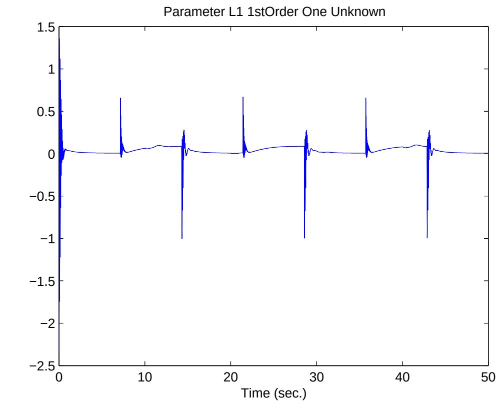
(b) Pitch with low adaptive gains that are 0.1.



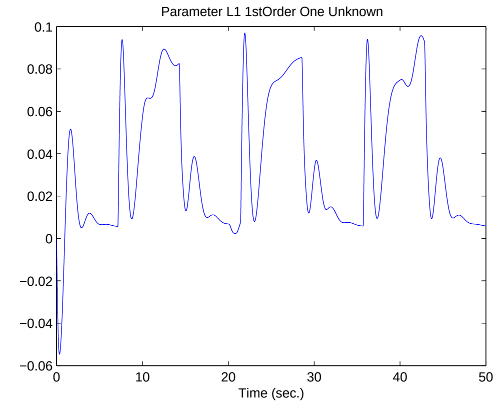
(c) Elevator command with high adaptive gains.



(d) Elevator command with low adaptive gains.

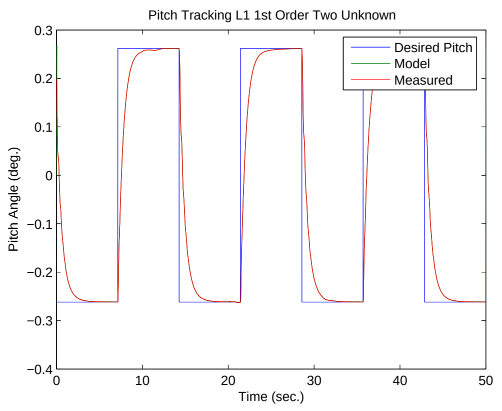


(e) Parameter estimate with high adaptive gains.

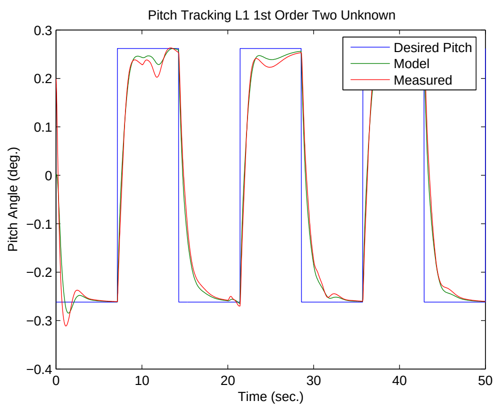


(f) Parameter estimate with low adaptive gains.

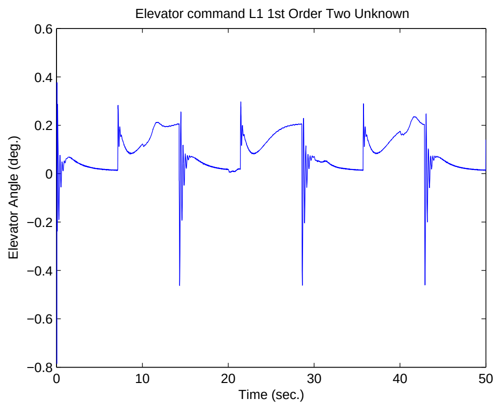
Figure 5.11: Simulink results for first-order model with one unknown pitch controller.



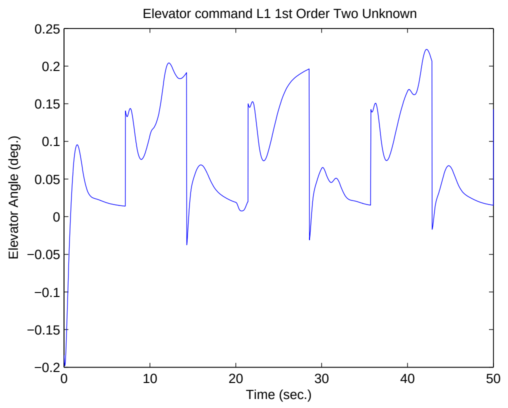
(a) Pitch angle with high adaptive gains.



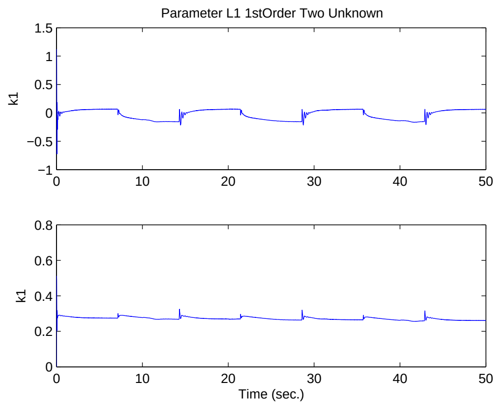
(b) Pitch angle with low adaptive gains.



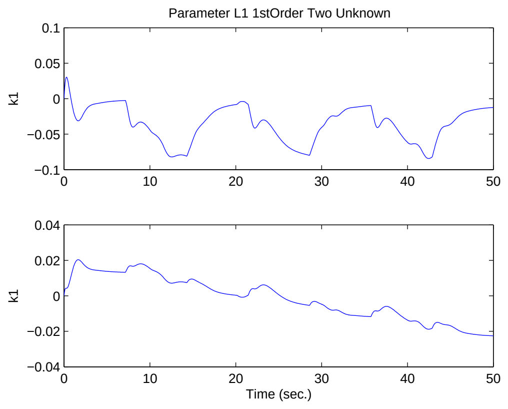
(c) Elevator command with high adaptive gains.



(d) Elevator command with low adaptive gains.

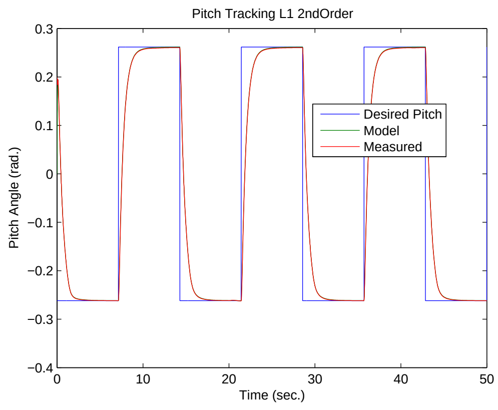


(e) Parameters with high adaptive gains.

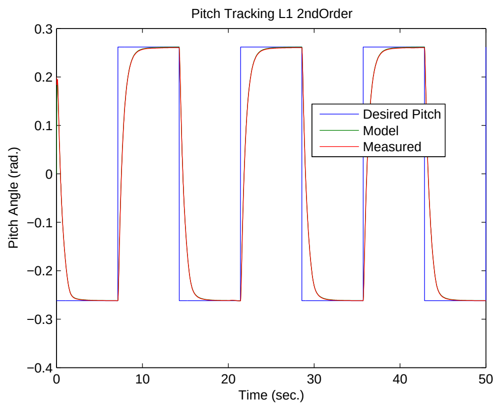


(f) Parameters with low adaptive gains.

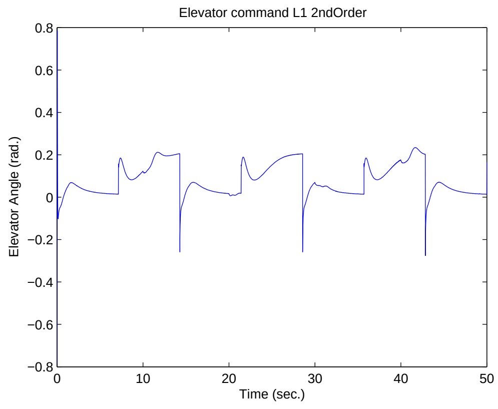
Figure 5.12: Simulink results for the first-order mode with two unknowns pitch controller.



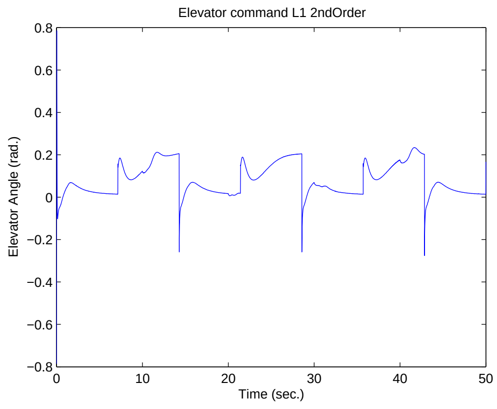
(a) Pitch with high adaptive gains that are 100.



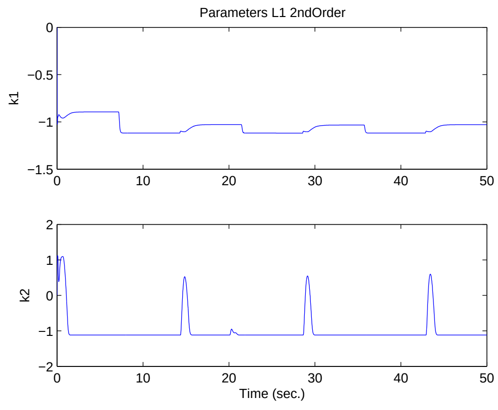
(b) Pitch with low adaptive gains that are 0.1.



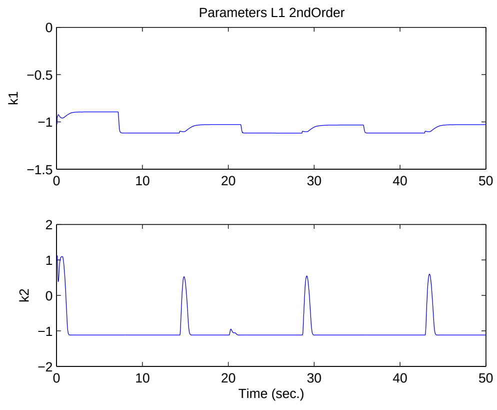
(c) Elevator command with high adaptive gains.



(d) Elevator command with low adaptive gains.

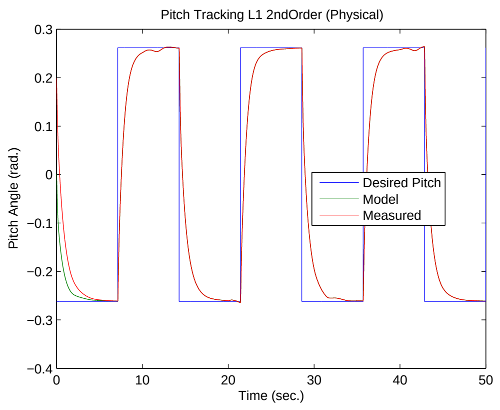


(e) Parameters with high adaptive gains.

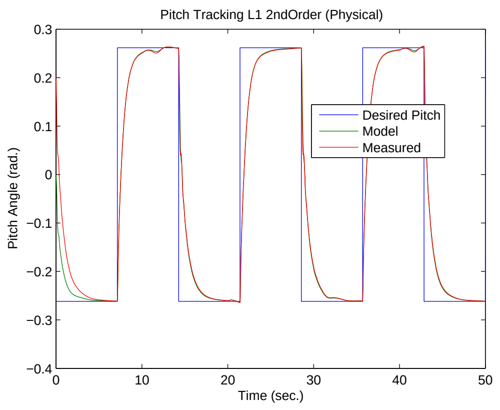


(f) Parameters with low adaptive gains.

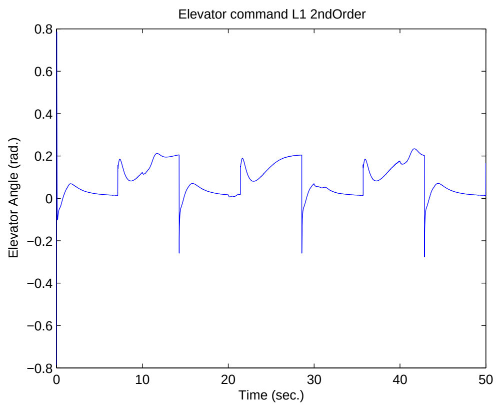
Figure 5.13: Simulink results for the second-order model pitch controller.



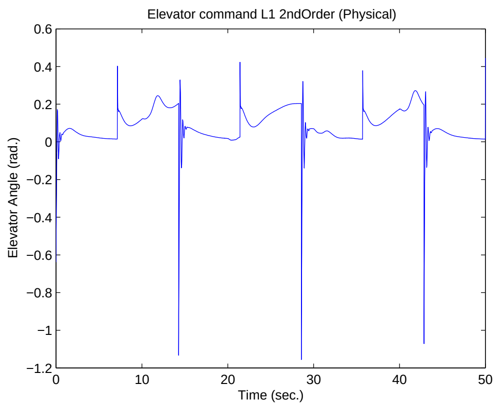
(a) Pitch with high adaptive gains that are 100.



(b) Pitch with low adaptive gains that are 0.1.



(c) Elevator command with high adaptive gains.

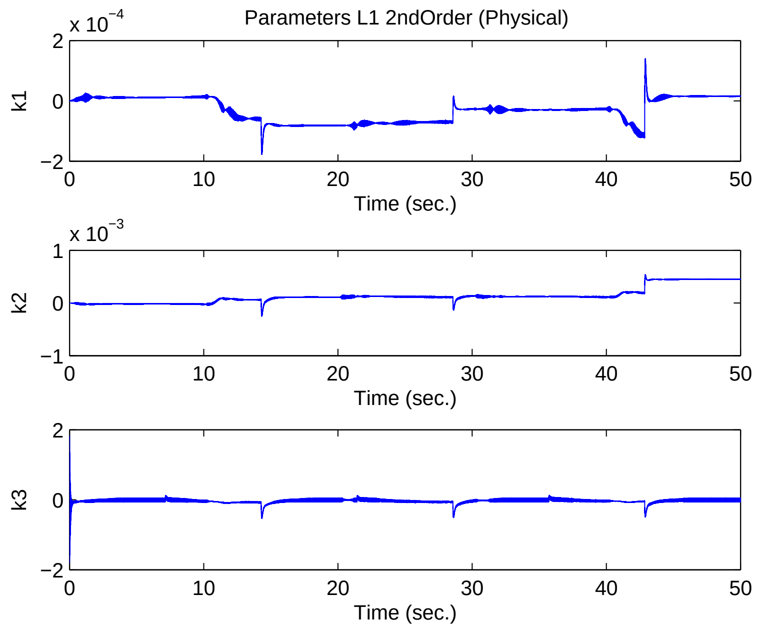


(d) Elevator command with low adaptive gains.

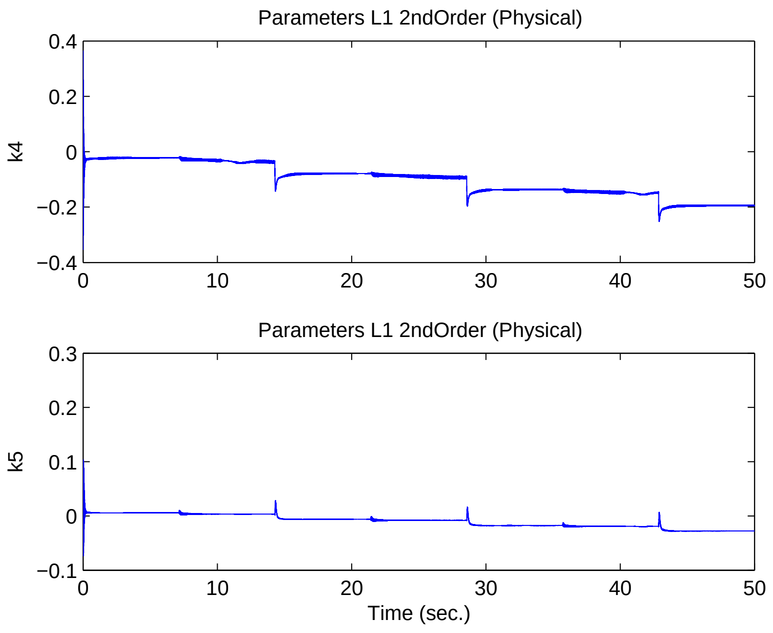
Figure 5.14: Simulink results for the Physically Motivated model pitch controller.

Table 5.1 shows the average error between the actual pitch and the desired pitch. This metric is helpful when comparing the MRAC controllers with $\mathcal{L}_1$ controllers. The reason this is a good metric is that the behavior of the $\mathcal{L}_1$ controller models change as the estimates change, whereas the Lyapunov MRAC models have fixed behavior. Using this metric the order of best performance to worst is Scheme D, first-order (1 Unknown), physically motivated, first-order (2 Unknowns), Scheme B, second-order, Scheme C, and Scheme A.

The difference in model tracking of the adaptive schemes is seen in Table 5.2. This metric shows that the Lyapunov MRACs, except Scheme D, do not track their models as well as $\mathcal{L}_1$ controllers. This is due to the ability of $\mathcal{L}_1$ models' behavior to

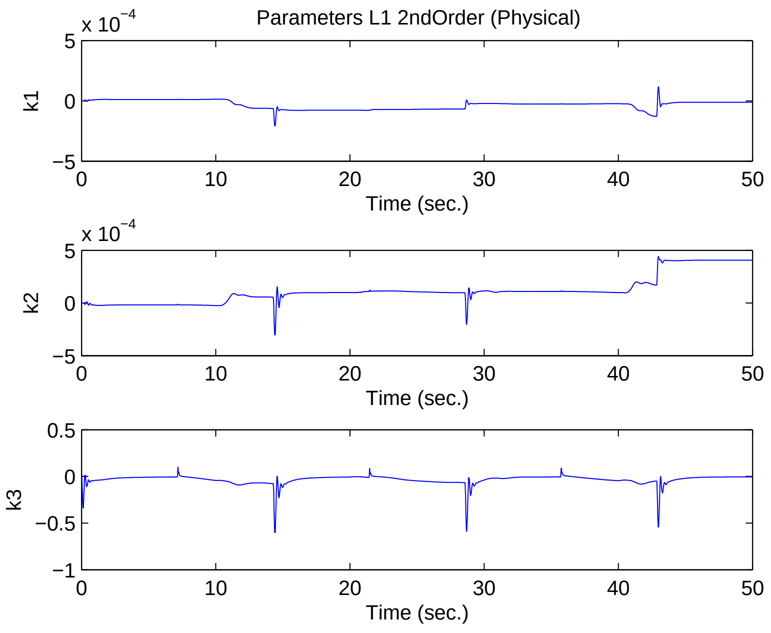


(a) Parameters $k_1 - k_3$ with high adaptive gains.

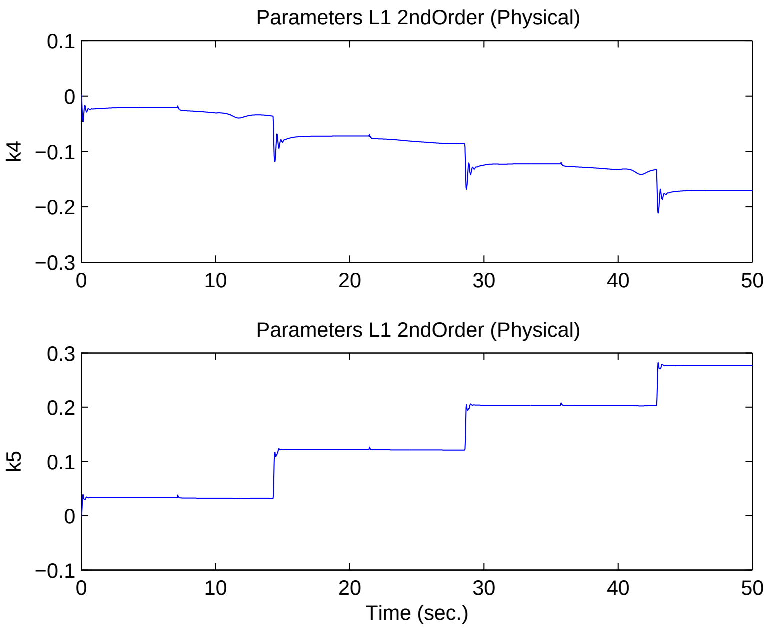


(b) Parameters $k_4 - k_5$ with high adaptive gains.

Figure 5.15: Simulink parameter results for the Physically Motivated model pitch controller with high adaptive gains.



(a) Parameters $k_1 - k_3$ with low adaptive gains.



(b) Parameters $k_4 - k_5$ with low adaptive gains.

Figure 5.16: Simulink parameter results for the Physically Motivated model pitch controller with low adaptive gain.

Table 5.1: The average error (degrees) between actual and desired pitch for simulations.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	0.2562	0.3136	0.2378	0.2061
MRAC B	0.2303	0.2335	0.2598	0.1927
MRAC C	0.2409	0.2437	0.2824	0.1871
MRAC D	0.0780	0.0887	0.0862	0.0547
$\mathcal{L}_1$ first-order (1 Unknown)	0.1196	0.1358	0.1101	0.1110
$\mathcal{L}_1$ first-order (2 Unknown)	0.2242	0.2585	0.2051	0.2049
$\mathcal{L}_1$ second-order	0.2400	0.2870	0.2261	0.1983
$\mathcal{L}_1$ Physical	0.2180	0.2394	0.2090	0.2041

Table 5.2: The average error (degrees) between actual and model pitch angle for the $\mathcal{L}_1$ simulations.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	0.1593	0.2343	0.1015	0.1047
MRAC B	0.1229	0.1927	0.0687	0.0591
MRAC C	0.2362	0.2344	0.2811	0.1829
MRAC D	0.0318	0.0416	0.0274	0.0234
$\mathcal{L}_1$ first-order (1 Unknown)	0.0351	0.0493	0.0252	0.0253
$\mathcal{L}_1$ first-order (2 Unknown)	0.0351	0.0576	0.0138	0.0137
$\mathcal{L}_1$ second-order	0.0617	0.0982	0.0311	0.0282
$\mathcal{L}_1$ Physical	0.0548	0.0949	0.0013	0.0013

change as parameter estimates change. As parameters change, the model is attracted to the aircraft's actual pitch angle.

Using the $\mathcal{L}_2$ norm, Table 5.3 shows the energy in the error between actual and desired pitch. Unfortunately it appears that the $\mathcal{L}_1$ controllers have a slower rise time than the Lyapunov MRAC controllers. This is probably because of the low-pass filtering that occurs inside $\mathcal{L}_1$ controllers. The Lyapunov MRACs have less energy in their tracking errors as a result. However, comparing only Lyapunov MRACs, they can order them from best to worst performers as Schemes D, B, A, C. This ordering is supported by Table 5.1. The ordering for $\mathcal{L}_1$ controllers is first-order (2 Unknowns), second-order, first-order (1 Unknown), and physically motivated. This is different from the order established in Table 5.1 where the first-order (1 Unknown) controller

Table 5.3: The $\mathcal{L}_2$ norm of the error between the actual and desired pitch angles for the simulations.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	4.9993	3.5613	2.8133	2.1051
MRAC B	3.8796	2.1314	2.6120	1.9199
MRAC C	5.2478	3.0765	3.6006	2.2608
MRAC D	3.0049	1.8564	1.9415	1.3466
$\mathcal{L}_1$ first-order (1 Unknown)	7.7771	5.1056	4.1224	4.1740
$\mathcal{L}_1$ first-order (2 Unknown)	6.2048	4.2068	3.1898	3.2600
$\mathcal{L}_1$ second-order	6.6718	4.7827	3.4820	3.0847
$\mathcal{L}_1$ Physical	24.5625	15.3547	13.7540	13.3559

Table 5.4: The $\mathcal{L}_2$ norm on error between actual and model pitch angle for the $\mathcal{L}_1$ simulations.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	2.1933	2.1500	0.2851	0.3273
MRAC B	1.1933	1.1651	0.2057	0.1562
MRAC C	5.1160	2.8643	3.5888	2.2561
MRAC D	0.7327	0.6685	0.2351	0.1862
$\mathcal{L}_1$ first-order (1 Unknown)	1.1521	1.0790	0.2854	0.2859
$\mathcal{L}_1$ first-order (2 Unknown)	0.6465	0.6460	0.0173	0.0160
$\mathcal{L}_1$ second-order	1.0759	1.0747	0.0389	0.0312
$\mathcal{L}_1$ Physical	3.8138	3.8138	4.0619e-004	3.9775e-004

has moved into 3rd place from 1st place. The reason for the order change is probably due to the oscillations of the actual pitch angle, which implies a slower settling time.

Table 5.4 displays the $\mathcal{L}_2$ norm of the error between the actual and model pitches. The amount of divergence from the model by either oscillations, leading, or lagging can be surmised from this table. The item of note in the table is that the first-order (2 Unknowns) controller has the least divergence from its model. Using this result with the result of Table 5.2, it appears that the first-order (2 Unknowns) controller is the best performer in terms of settling time and divergence form its model.

Tables 5.2 and 5.4 establish that the order of pitch controllers as Schemes D, B, A, and C. However, Tables 5.7 and 5.8 establish the order for best roll controller

Table 5.5: The average error between actual and desired roll angle for the MRAC simulations.

	Flight Leg			
MRAC Scheme	Total	1	2	3
MRAC A	0.2185	0.2447	0.2215	0.1863
MRAC B	0.2694	0.2716	0.3010	0.2314
MRAC C	0.3109	0.4209	0.2815	0.1836
MRAC D	0.2036	0.2098	0.2317	0.1633

Table 5.6: The $\mathcal{L}_2$ norm on error between actual and desired roll angle for the MRAC simulations.

	Flight Leg			
MRAC Scheme	Total	1	2	3
MRAC A	4.3204	2.6861	2.7341	2.0040
MRAC B	4.4756	2.4517	3.0207	2.2132
MRAC C	7.1859	5.7861	3.5986	2.2822
MRAC D	7.6598	4.5929	5.1286	3.3579

as Schemes A, B, C, then D. With the exception of Scheme A, all Lyapunov based MRAC schemes performed better at pitch control than roll control. Tables 5.7 and 5.8 show the average error and $\mathcal{L}_2$ for the Lyapunov MRACs between the actual and model pitches, respectively.

Finally, if the figures are compared, there is less elevator oscillation for $\mathcal{L}_1$ controllers than Lyapunov MRAC controllers. This is due to the filtering of the parameter estimates that the $\mathcal{L}_1$ controller uses.

Table 5.7: The average error (degrees) between actual and model roll angle for the MRAC simulations.

	Flight Leg			
MRAC Scheme	Total	1	2	3
MRAC A	0.0667	0.0960	0.0513	0.0389
MRAC B	0.1457	0.2085	0.1089	0.0915
MRAC C	0.2362	0.2344	0.2811	0.1829
MRAC D	0.3137	0.2388	0.4581	0.1684

Table 5.8: The $\mathcal{L}_2$ norm on error between actual and model roll angle for the MRAC simulations.

	Flight Leg			
MRAC Scheme	Total	1	2	3
MRAC A	0.3567	0.3270	0.1174	0.0810
MRAC B	1.3348	1.2234	0.4260	0.3218
MRAC C	5.1160	2.8643	3.5888	2.2561
MRAC D	14.0801	5.4174	12.5477	3.3844

5.1.4 Summary

All MRAC Schemes performed well in simulation. Scheme C and Scheme D appear to have parameters converging in both roll and pitch. Both Scheme C and Scheme D had periods of oscillation that illustrate the importance of persistent excitation. Scheme D was the best at tracking pitch, but the worst at roll tracking. On the contrary, Scheme A had the best roll tracking, but nearly the worst pitch tracking. Although control signal saturation was present in many of the algorithms, it did not destabilize any of the algorithms. All four MRAC schemes produced high frequency control signals. Also, all four algorithms generally showed better tracking as time progressed.

As observed in all four $\mathcal{L}_1$ controllers, high adaptive gain translates to better pitch tracking. Because the parameters are low-pass filtered when passed to the elevator command, there is significantly less commanded chatter than is typical of MRAC controllers. Both first-order models had about the same settling time, though the first-order (1 Unknown) model tracked desired pitch angle the best. Furthermore, the first-order models generally performed better than the second-order models. This seems counterintuitive as one would expect the higher order models to perform better since they should more accurately approximate the true dynamics of the system. A possible explanation is that since the second order systems have more parameters, there is a larger solution space for the parameters. In other words, more parameters mean more permutations of those parameters that can solve for the correct output. Thus the models with fewer parameters would have a smaller solution space and thus adapt faster. Allowing the adaptive gain to go to infinity might mitigate this issue–unfortunately real world computing is insufficient. It should also be noted that in all cases, persistently exciting signal are needed to have the parameters converge to their true values.

5.2 Flight Testing Results

Flight tests were performed on Deliverance, Ray, and Phidipidies described in Chapter 4. All flight testing followed a hourglass pattern to allow both right and left turns equal weight. Below are the results of the flight testing for Lyapunov based MRAC and $\mathcal{L}_1$ controllers.

5.2.1 Lyapunov Based MRAC

Initially, Deliverance was used, see Figure 4.9, to tune the adaptive control gains when flying each algorithm. This allowed the four algorithms to be compared with each other and against the usual PD controller. To obtain this comparison, hourglass-shaped pattern with legs between 200 and 300 meters long were flown on the same airframe, Deliverance, for the four algorithms and the hand-tuned PD controller. This allowed for consistent measurements for comparison.

Along with the pitch and plots, values of the $\hat{\mathbf{k}}$ parameters for each adaptive scheme are displayed. Note that the parameter estimates, though representing certain combinations of aerodynamic coefficients and other estimated quantities, are not expected to actually converge to their true values. The Lyapunov stability proofs from Chapter 3, used to derive the algorithms, do not guarantee that these parameters will converge to the true values. Rather, the proofs show that the parameters assume values that cause the tracking error to go to zero.

Before each algorithm could be flown, several adaptive control gains (namely k_m (model gain), γ_1 (gain on the error), and Γ (adaptive gain)) had to be tuned. Algorithm D was easiest to tune as it had the fewest number of gains. Algorithm D also is the simplest algorithm to implement and takes up the smallest amount of autopilot code space. In contrast, algorithms B and C were not robust when tuned for performance, while Scheme A is more complicated.

As will be discussed, MRAC Scheme D has the best MRAC in terms of robustness and simplicity. Therefore, it was chosen to fly on the other two test airframes shown in Figure 4.9 to test its scalability. Robustness of MRAC Scheme D was further tested by attaching a flap near the center of Deliverance's right wing and deploying it mid-flight (see Figure 4.11). The flap's purpose is to change the aerodynamic coefficients of the MAV quickly. The rate of adaptation and the autopilots ability to adapt can then be observed. Results for these flight tests are detailed below.

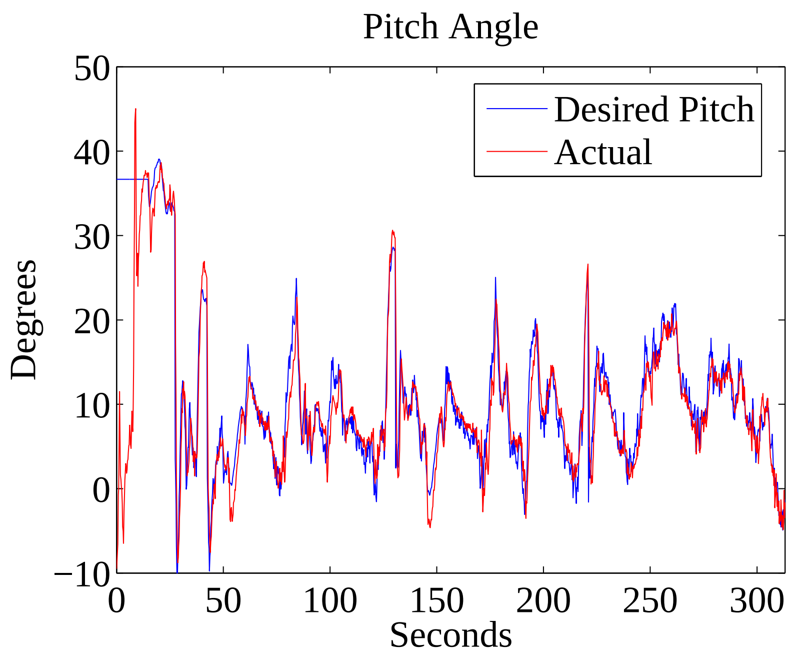
PD Control

In order to set a baseline to compare the different MRAC schemes, a conventional hand-tuned PD controller for both roll and pitch was flown along the hourglass path. Figure 5.17 shows the results of this flight.

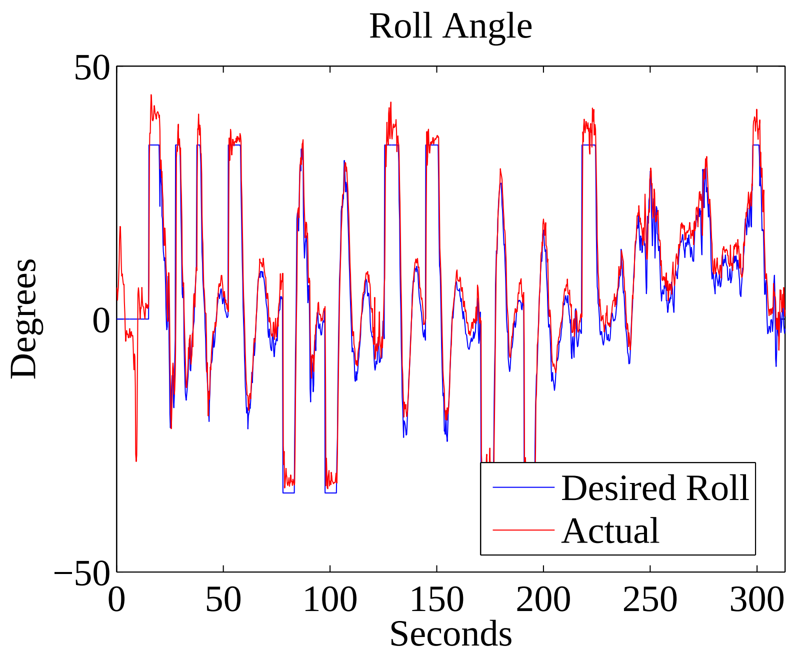
Pitch tracking has a distinct sharpness to it. Also, the desired pitch does not appear to saturate. In contrast, desired roll is limited to 30 degrees and does saturate. However, like pitch tracking, roll tracking is also sharp. Comparing the pitch and roll tracking of MRACs with the PD controller will provide further insight into the performance of these algorithms.

MRAC Scheme A

The pitch results for MRAC Scheme A are found in Figure 5.18. By matching peaks in pitch angle found in Figure 5.17(a) with peaks in Figure 5.18(a) it can be seen that the MRAC's peaks are 2-5 degrees less than PD control. This implies that altitude was held better with MRAC than PD control. The data for the elevator command is unavailable, however, it can be inferred from the oscillations in actual pitch that the elevator has high frequency oscillations–a typical result of MRACs. Furthermore, the parameters in Figure 5.18(c) may have converged to constants. Thus, further improvement is not expected.



(a) Typical PD pitch tracking on Deliverance. The average error remains around two degrees.



(b) Typical PD Roll tracking on Deliverance. The average error stays at about 3.5 degrees.

Figure 5.17: PD pitch and roll on Deliverance.

Figure 5.18 shows the flight results for MRAC A roll. Oscillations of the aircraft roll angle indicate high frequency oscillations of the aileron. The parameters in Figure 5.18(d) have not converged. This implies that roll tracking might continue to improve.

MRAC Scheme B

The results for MRAC Scheme B pitch are found in Figure 5.19. Furthermore, there is a lot of oscillation in the aircraft's actual pitch. This implies increased high frequency content on the elevator. The parameters have not converged yet in Figure 5.19(c), so there might be improvement, but it is doubtful.

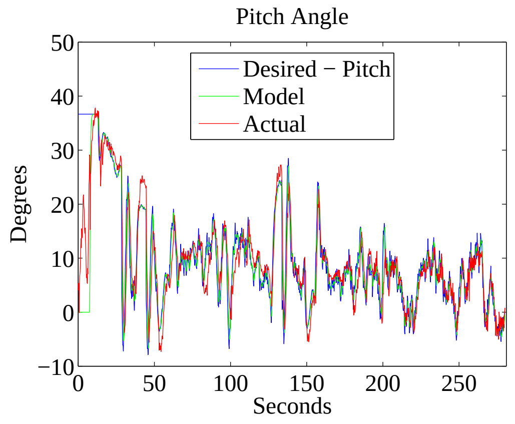
Figure 5.19 shows the results for MRAC Scheme B roll. There appears to be high frequency oscillations on the aileron since actual roll oscillates for MRAC B. The parameters seem to have converged in Figure 5.19(d); therefore, improved performance is not expected.

Flight testing MRAC Scheme B found that it is not a robust control scheme. Scheme B was sensitive to tuning and could easily destabilize the plane when tuned for performance. Therefore, it was not considered for continued testing.

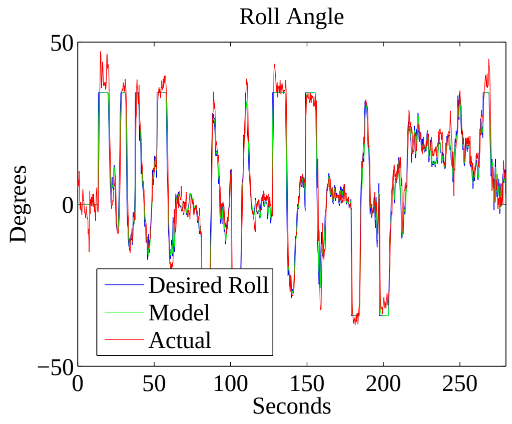
MRAC Scheme C

The left side of Figure 5.20 shows the pitch results for MRAC Scheme C. The actual pitch of the aircraft oscillates more than PD control. This implies that the elevators are oscillating. Looking at Figure 5.20 the parameters appear to have converged. Notice that when the parameters converge, oscillations of the actual pitch angle increases.

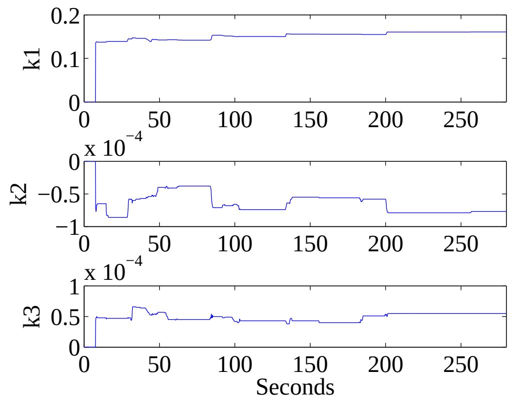
The results from MRAC Scheme C roll are shown on the right side of Figure 5.20. If we compare Figures 5.17(b) and 5.20(b), PD control has less overshoot than MRAC C. Also, counter to expectations, the actual roll appears to oscillate less for MRAC C then PD control. However, there are times of punctuated oscillations– between 120-145 seconds and 30-45 seconds (see Figure 5.20(d)). This appears related to the changes in parameter estimates.



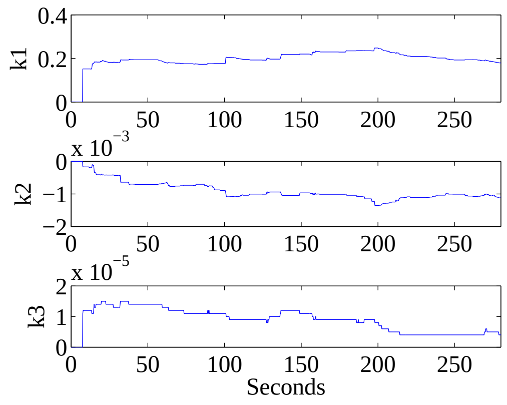
(a) Pitch. Actual pitch oscillates which implies the elevator is oscillating.



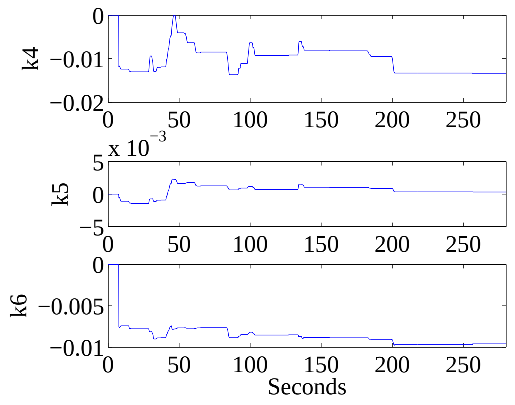
(b) Roll. Actual roll oscillates.



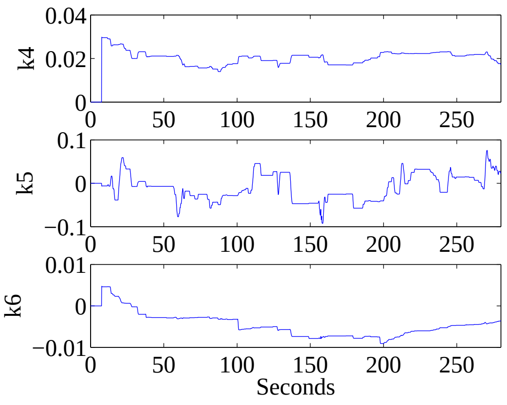
(c) Pitch parameters appear to be converging.



(d) Roll parameters have not converged.

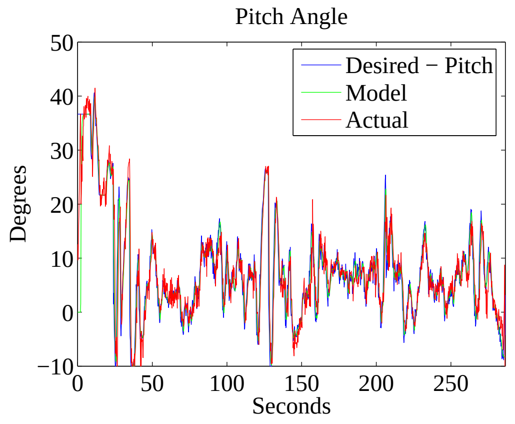


(e) Pitch parameters appear to be converging.

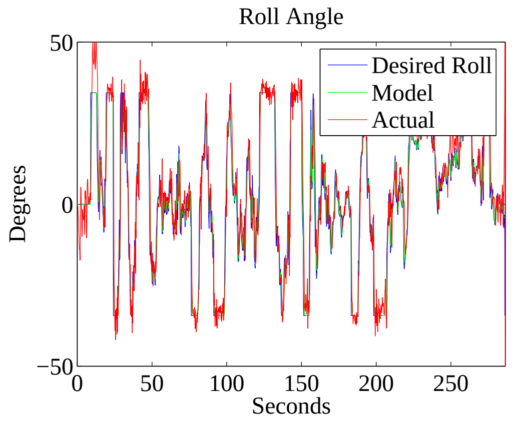


(f) Roll parameters have not converged.

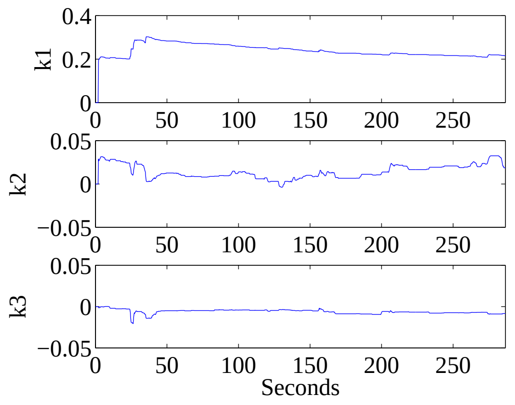
Figure 5.18: MRAC Scheme A pitch and roll on Deliverance flying the hourglass pattern.



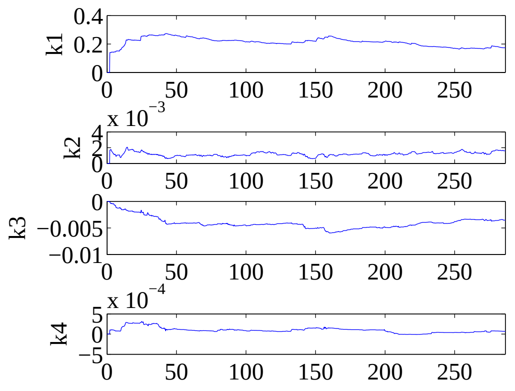
(a) MRAC Scheme B pitch shows improvement over time.



(b) MRAC Scheme B roll shows improvement over time.



(c) MRAC Scheme B pitch parameters have not converged.



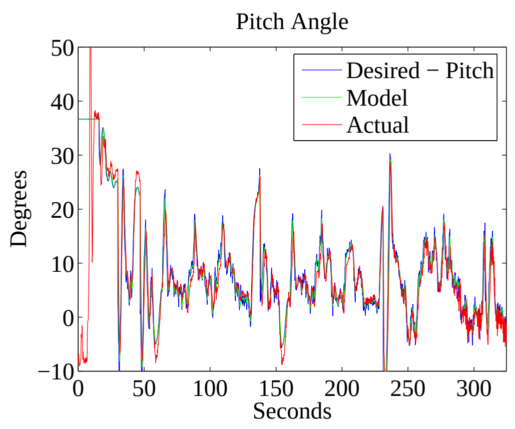
(d) MRAC Scheme B roll parameters may have converged.

Figure 5.19: MRAC Scheme B pitch and roll on Deliverance.

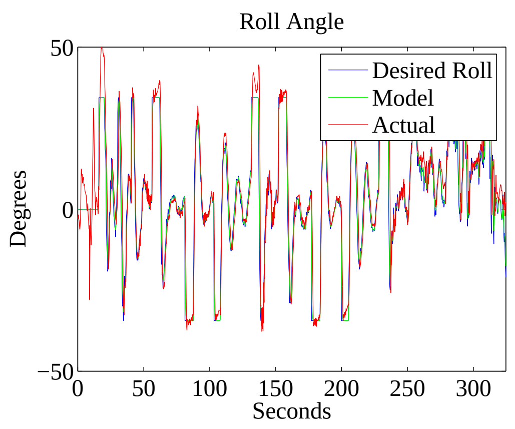
Like Scheme B, MRAC Scheme C was not found to be a robust control scheme during flight testing. Scheme C was also sensitive to tuning and could easily destabilize the plane when tuned for performance. Therefore, it was not considered for continued testing.

MRAC Scheme D

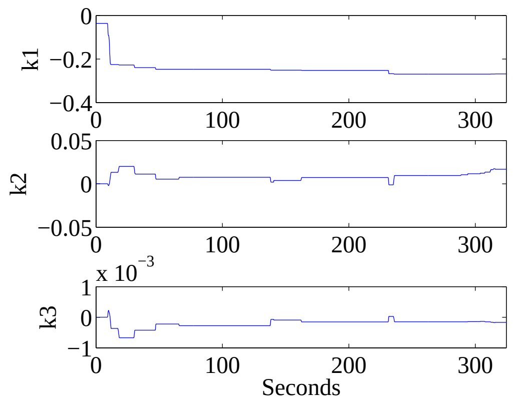
The right side of Figure 5.21 shows MRAC Scheme D flight results for pitch. Figure 5.21(a) shows that as time increases, MRAC D pitch altitude hold appears to improve. Comparing Figures 5.17(a) and 5.21(a), MRAC is seems better at small changes in pitch than PD control, though the opposite is true for large changes. The



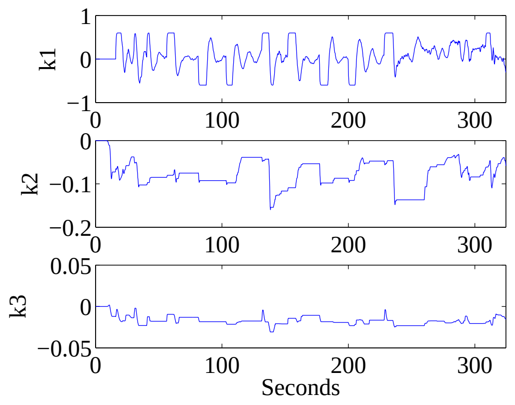
(a) MRAC Scheme C pitch does not improve overtime.



(b) MRAC Scheme C roll performance does not improve. Instead it hovers around five degrees of error.



(c) MRAC Scheme C pitch parameters on Deliverance flying an hourglass pattern. k_1 and k_3 appear to have converged to a value while k_2 has not.

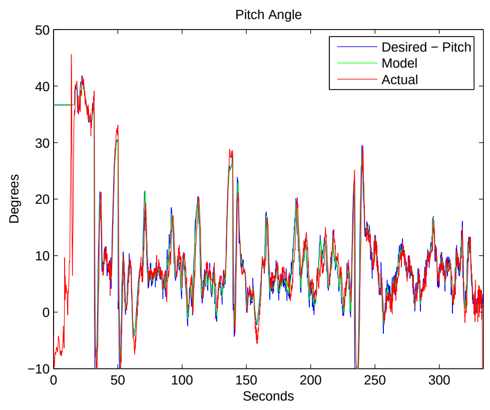


(d) MRAC Scheme C roll parameters have not converged.

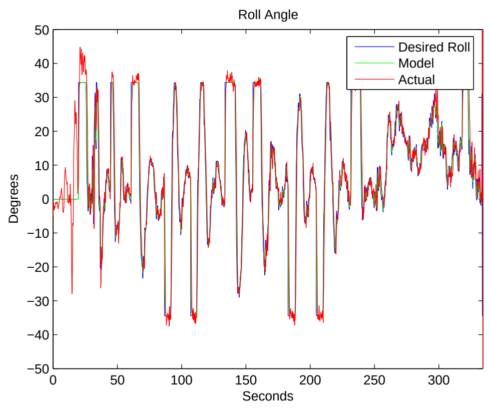
Figure 5.20: MRAC Scheme C pitch roll flight results on Deliverance.

actual pitch angle has oscillations implying that the elevator is oscillating. Figure 5.21 shows that the parameters have not converged yet, so performance may improve.

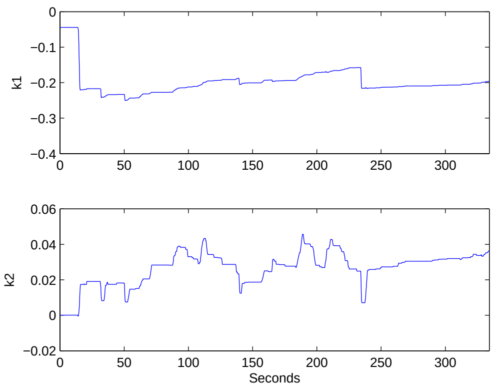
MRAC Scheme D roll results are found on the left side of Figure 5.21. MRAC D appears to be better at small roll changes than PD control, but worse at large changes in roll (see Figures 5.17(b) and 5.21(b)). However, MRAC D has less overshoot and little oscillation. From the figures, Scheme D appears to oscillate more than Scheme C, but overshoots less. Figure 5.21(d) shows that the parameters have not converged for MRAC D, so performance may yet improve.



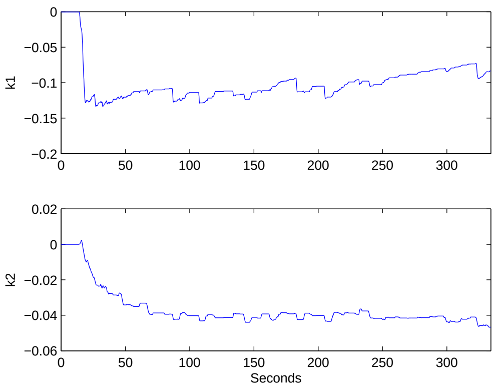
(a) MRAC Scheme D pitch improves over time.



(b) MRAC Scheme D roll improves over time.



(c) MRAC Scheme D pitch parameters have not yet converged.



(d) MRAC Scheme D roll parameters have not converged.

Figure 5.21: MRAC Scheme D pitch and roll flight results on Deliverance.

Table 5.9: Adaptive control gains used in Scheme D on all three test platforms.

Pitch k_m	3.0
Pitch γ_1	45.0
Pitch Γ_1	0.06
Pitch Γ_1	0.01
Pitch Leakage gain 1	0.001
Pitch Leakage gain 2	0.0005
Roll k_m	4.0
Roll γ_1	140.0
Roll Γ_1	0.005
Roll Γ_1	0.001
Roll Leakage gain 1	0.001
Roll Leakage gain 2	0.0001

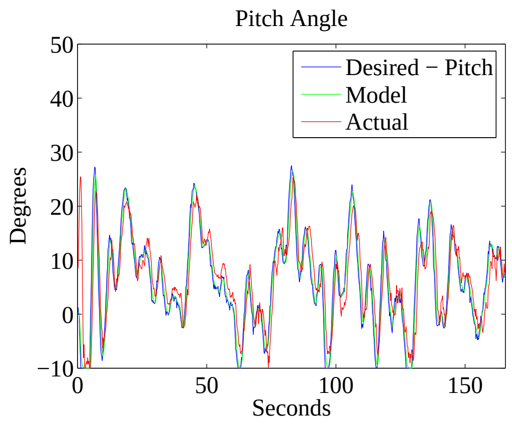
Since MRAC Scheme D is simpler than Scheme A and more robust than Schemes B and C, testing was continued for Scheme D. Figures 5.22 through 5.23 show the results of flight tests of MRAC Scheme D on Ray and Phidipides. Both Ray and Phidipides flew well with these same gains. Notice that MRAC D is converging to different values for Ray and Phidipidies than for Deliverance. Ray's desired pitch and roll oscillates more than Deliverance which is probably due to Ray's slower dynamics. Phidipides has faster dynamics which explains the higher peaks in pitch, but notice that average tracking error for both pitch and roll are comparable. Also, although PD gains for Phidipides were hard to tune, MRAC Scheme D flew without problems. This shows the main strength of adaptive control: the ability to fly the same code on different airframes without having to hand-tune gains.

In order to test MRAC Scheme D's ability to adapt to changes to an airframe in flight (i.e. damage to the airframe), a flap is attached to Deliverance. Figure 4.11 shows the flap deployed on Deliverance. When commanded from the ground station, the autopilot will deploy the flap. Deploying the flap will cause significant changes to the aerodynamic coefficients, which MRAC D should adapt to.

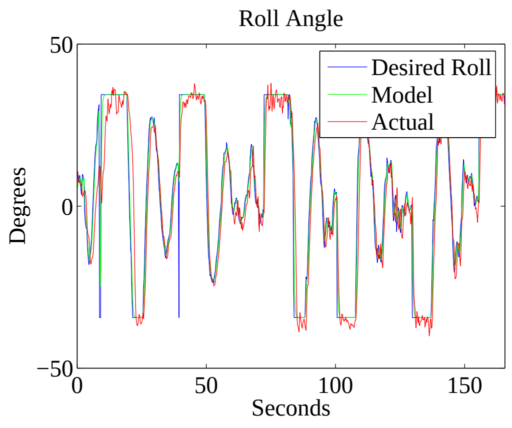
Figures 5.25 and 4.11 show the effect of deploying the flap on the PD controller and MRAC D. With the flap deployed, PD control can still fly the MAV, but with substantial steady-state error on roll. However, MRAC scheme D adapts to the change in aerodynamics within a few seconds. When the flap goes back down, the parameters once again adapt back to their previous values.

5.2.2 $\mathcal{L}_1$ Controllers

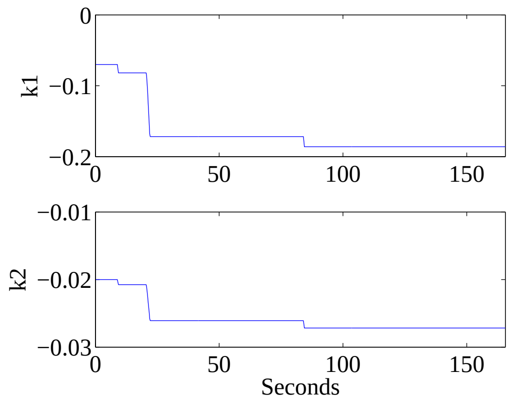
Because the first order models perform better in simulation than the second order models, the second order models were not tested. Furthermore, although the first order model, with two unknown parameters, performed slightly better than the model with only one parameter, both models have about the same settling time. The simpler version has been coded for flight testing. The following figures are from flight tests conducted on Ray (see Figure 4.9). Ray flew the typical hourglass pattern and the data for pitch controlled by the $\mathcal{L}_1$ controller and PD controller were recorded at



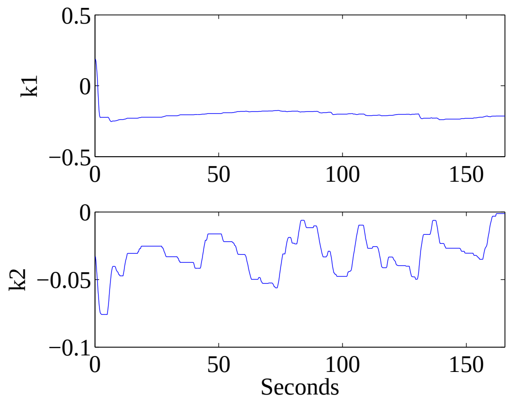
(a) Pitch tracking on Ray while flying the hourglass pattern



(b) Roll tracking on Ray while flying the hourglass pattern.



(c) Pitch parameters on Ray while flying the hourglass pattern. Parameter k_1 appears to have converged to a value while k_2 has not. It appears that the aircraft is being driven by the bias term or "trim".

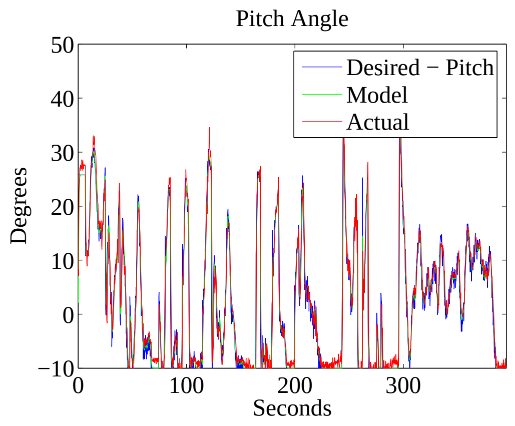


(d) Roll parameters on Ray while flying the hourglass pattern. k_1 and k_2 appear to have converged on values.

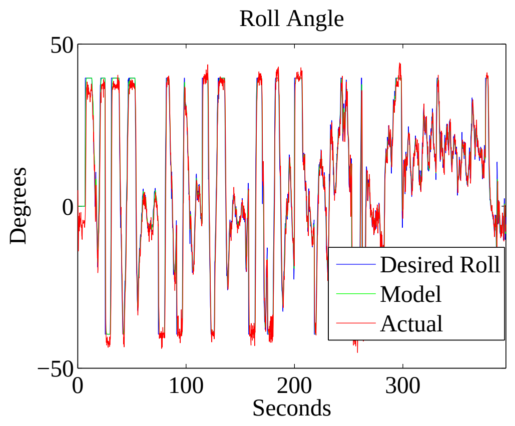
Figure 5.22: MRAC Scheme D pitch and roll on Ray flying the hourglass pattern.

separate times. The data for the PD controller and the data for the $\mathcal{L}_1$ controller were recorded starting and finishing at the same place in the hourglass pattern. Finally, the $\mathcal{L}_1$ controller was allowed to auto-land Ray. Roll was controlled with a PD controller during testing.

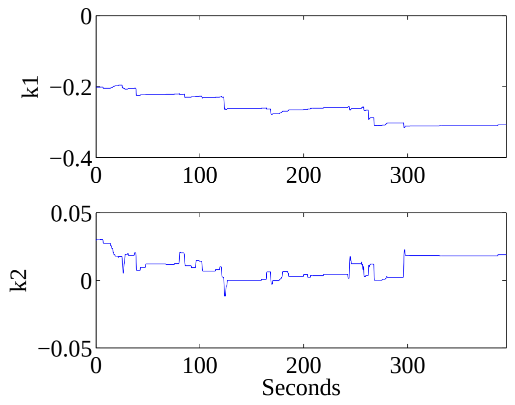
Before in-flight data recording, the $\mathcal{L}_1$ controller needed to be tuned. Although the $\mathcal{L}_1$ controller was tuned in simulation before flight testing, the gains were too high when flown in the field. One of the advantages of $\mathcal{L}_1$ controllers is that we can have high adaptive gains without causing oscillations. However, as this code



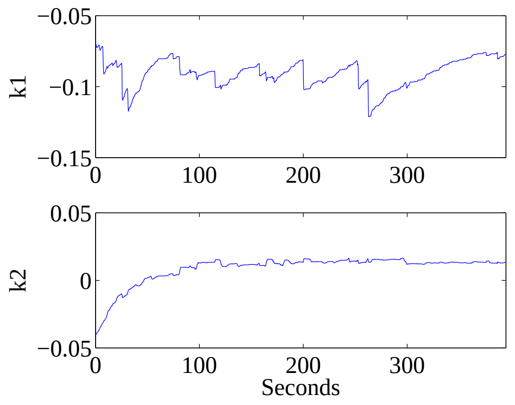
(a) Pitch tracking on Phidipides flying an hourglass pattern. As time goes on, the tracking becomes better.



(b) Roll tracking on Phidipides flying an hourglass pattern. As the flight continued roll tracking improved.



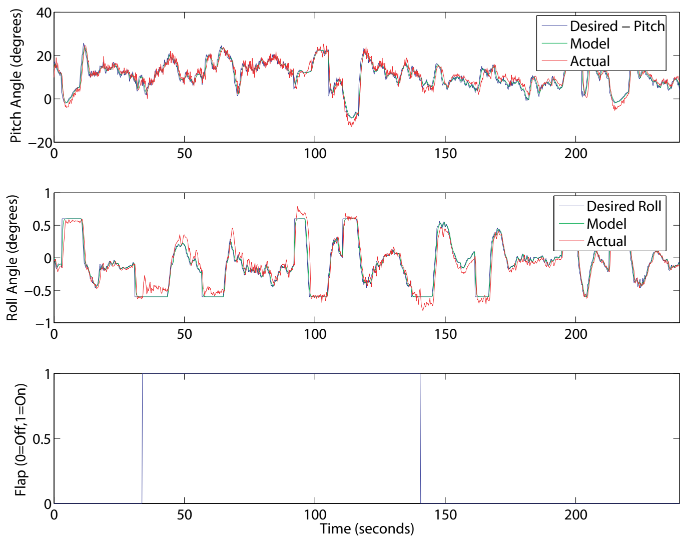
(c) Pitch Parameters on Phidipides flying an hourglass pattern. The parameters appear to have converged to values.



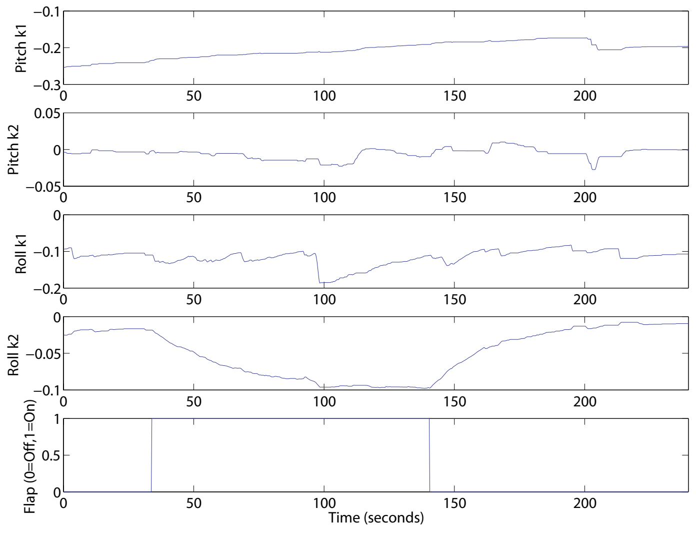
(d) Roll Parameters on Phidipides flying an hourglass pattern. k_2 seems to have converged, while k_1 may yet converged.

Figure 5.23: MRAC Scheme D pitch and roll on Phidipides flying an hourglass.

is running on a Rabbit microprocessor that loops through the entire autopilot at about 80 Hz, there is an upper limit on the adaptive gains. To counter this problem, subsampling was attempted to allow for higher adaptive gains. Initial flight testing revealed that subsampling actually slowed the autopilot code to about 10 Hz. This caused problems with the roll angle estimation. Without subsampling, the pre-filtered parameter estimates were saturating on their boundaries since the adaptive gain was now too high. None of these issues prevented Ray from flying, but Ray's performance was poor. Therefore, the $\mathcal{L}_1$ controller was tuned mid-flight.



(a) MRAC Scheme D pitch and roll as the flap is deployed. Pitch seems unaffected while roll re-adapts to the change after about 60 seconds. Roll again adapts to the flap being down, which appears to take half the time.



(b) MRAC Scheme D Pitch and Roll parameters. Notice k_2 for roll adapting to a new value when the flap is deployed and adapting back to the previous value when the flap is stowed. Roll was more sensitive to the flap than pitch. This is expected because of placement of the flap. Notice that the parameters re-adapt to the flap being put down again.

Figure 5.24: MRAC Scheme D behavior with flap deployed on Deliverance flying an hourglass path.

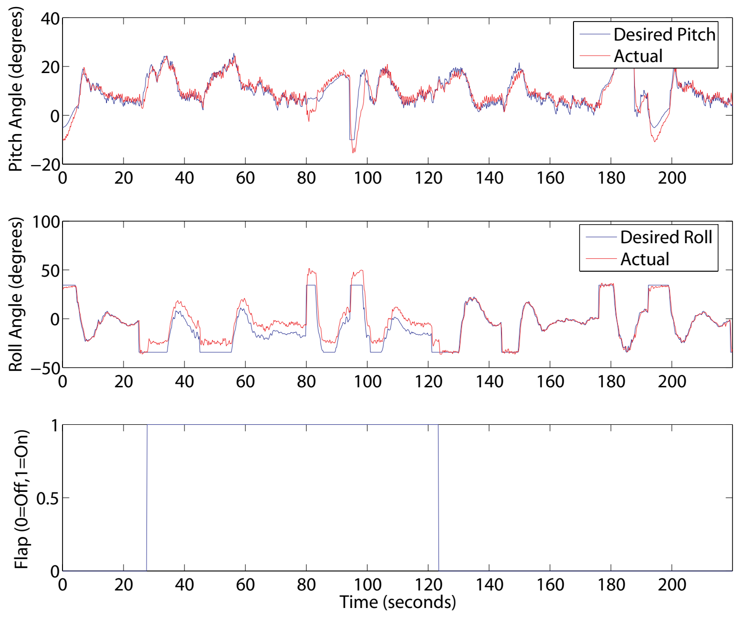
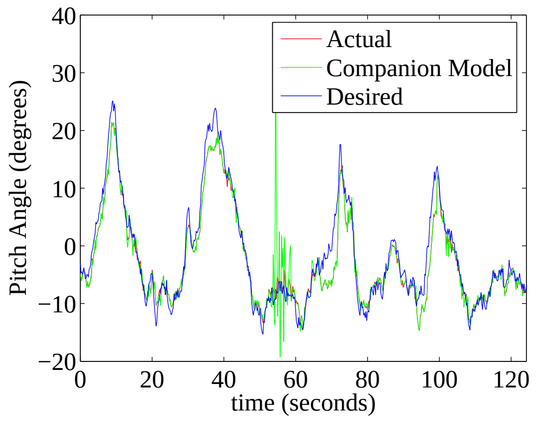


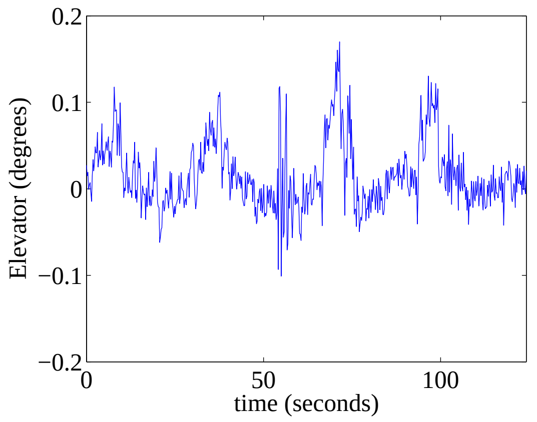
Figure 5.25: PD Control on Deliverance flying an hourglass path.

To tune the $\mathcal{L}_1$ controller on Ray, Ray was flown with the $\mathcal{L}_1$ controller running. The RC pilot would watch the plane to make sure that adjustments would not cause instability; if the the aircraft destabilized, the RC pilot would have taken control of the aircraft. As adjustments were made, the telemetry was downloaded and examined while Ray continued flying the hourglass pattern. Corrections were then uploaded to the autopilot, Ray's behavior was monitored, and the telemetry was again download and examined. This process was repeated until the $\mathcal{L}_1$ controller was tuned. The changes made on the autopilot were as follows. The first change on the autopilot was to eliminate subsampling. Next, the adaptive gains were reduced until the pre-filtered parameter estimates stopped constantly saturating. Finally, the model gain and low-pass filter were adjusted until performance was acceptable and oscillations were eliminated.

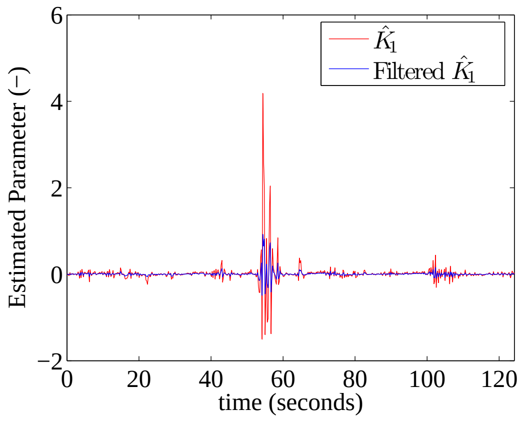
The flight data from the $\mathcal{L}_1$ and PD flights are found in Figures 5.26 and 5.28. The $\mathcal{L}_1$ controller typically had less error than the PD controller, which suffered from steady state error. The steady state error was introduced when the aircraft pitched



(a) Pitch



(b) Elevator



(c) Parameter

Figure 5.26: $\mathcal{L}_1$ pitch controller flying an hourglass pattern on Ray. Notice that around 50 seconds the companion model diverges for about 10 seconds. A 30 degree roll causes anomalies at 67 seconds and 92 seconds.

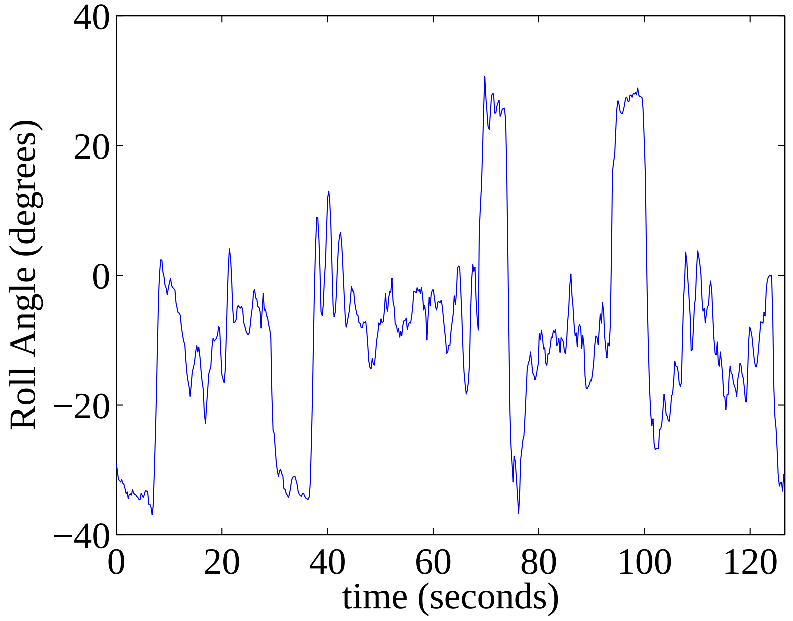


Figure 5.27: PD roll when $\mathcal{L}_1$ controller is in use. Notice how steep rolls of about 30 degrees effects pitch in Figure 5.26.

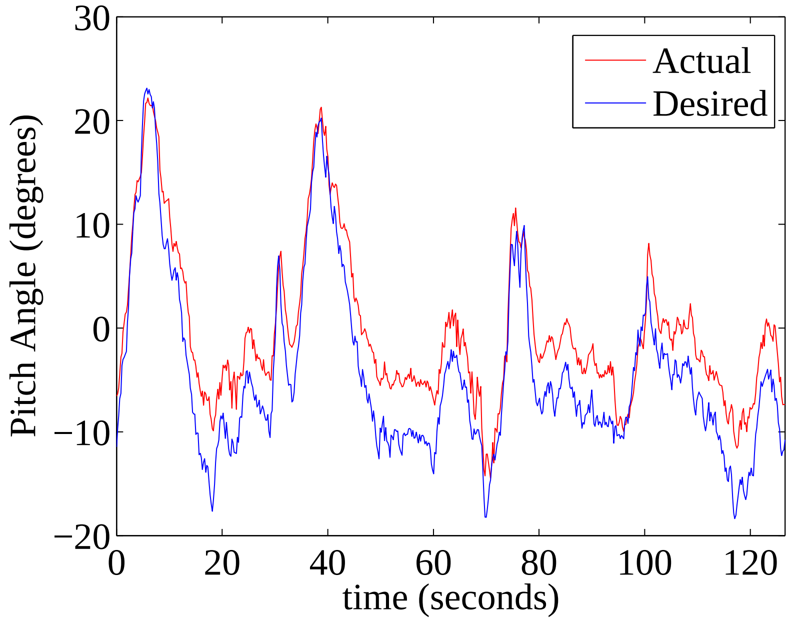


Figure 5.28: PD pitch control on Ray flying an hourglass pattern. Notice that the PD controller tracks positive pitch values well, but it does not track negative pitch values very well. This is due to the aircraft picking up speed when negatively pitched. This generates more lift and introduces steady state error.

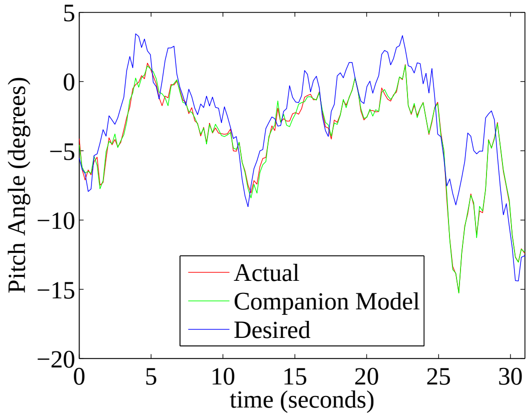
down. Pitching down caused the aircraft to increase speed which in turn increased lift. Being tuned at less lift, the PD controller settled at a steady state angle above the desired angle. Notice that when climbing, the PD controller did not have this problem as lift was decreased.

Looking at the data from the $\mathcal{L}_1$ controller, there are a couple of notable events. The first one is that at 50 seconds, when the companion model begins oscillating. This continued for about 10 seconds. To shed light on this issue, notice the update equation in Dynamic C:

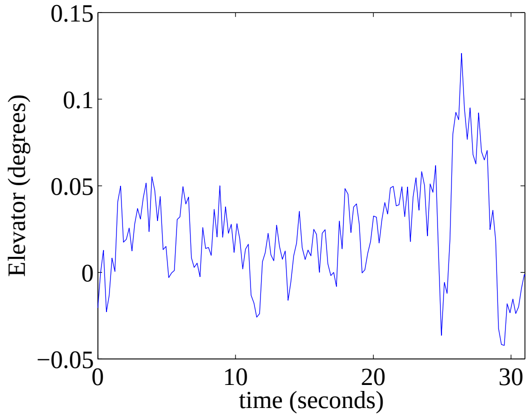
```
K1_hat = K1_hat + dt*(-gamma*(theta - theta_cm)*V*cos(phi));,
```

Noise on the airspeed, V, should not affect the parameter update much because airspeed only changes by one or two m/s. The problem is probably not θ_{cm}, because θ_{cm} began to oscillate after the parameter started to oscillate. And ϕ seems normal for level flight at 52 seconds (see Figure 5.27). Therefore, the problem most likely lies in the time term "dt". Sometimes real-time telemetry slows in the attitude window on the VC. Such a slow-down in telemetry would imply that the loops have temporarily slowed. If the loops were to slow to half their normal frequency, this would double the value normally associated with "dt". Since stability can be affected by how high the adaptive gain, γ, is relative to "dt", this may be the culprit. This probably would not even be seen by the datalogger function since it samples at specific times. For the Dynamic C code for the $\mathcal{L}_1$ controllers, see Section B.2.

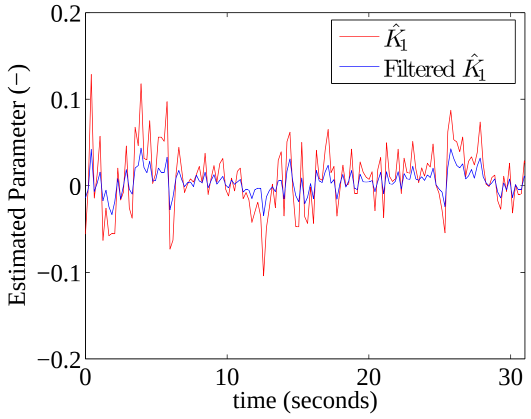
The second issue with the $\mathcal{L}_1$ controller is that when the aircraft rolled quickly, the pitch angle of the aircraft and companion model would fall away from the desired pitch, see Figures 5.26 and 5.27. Also, Figure 5.26(b) shows some oscillation in the control signal, but this is fairly well constrained well below saturation. High frequency signals have definitely been filtered on the control signal. Figure 5.29 shows the $\mathcal{L}_1$ pitch controller while landing. The plot seems fairly consistent with the plot taken while flying the hourglass pattern shown in Figure 5.26.



(a) Pitch



(b) Elevator



(c) Parameter

Figure 5.29: $\mathcal{L}_1$ pitch controller landing.

5.2.3 Analysis

Tables 5.10-5.11 show the average error and the $\mathcal{L}_2$ norm of the error for the $\mathcal{L}_1$ and PD controllers. Average error was calculated by

$$\frac{1}{n}\sum_{i=1}^{n}|\theta_c(i) - \theta(i)|,$$

and the $\mathcal{L}_1$ norm was calculated by

$$e_{\mathcal{L}_2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\theta^d(i) - \theta(i))^2},$$

where θ_c is the desired pitch and θ is the actual pitch angle.

Tables 5.10-5.13 display the average error and $\mathcal{L}_2$ norm of error for pitch and roll. Using these tables, the $\mathcal{L}_1$ controller is the best for pitch, while PD control preformed that best at roll.

Average error is displayed in Table 5.10. This is a better metric than $\mathcal{L}_2$ because when comparing the results of the PD controller from the the MRAC tests with the PD controller from the $\mathcal{L}_1$ test, there is a significant difference that suggests the norms are not comparable (see Table 5.11). However, Table 5.10 shows the $\mathcal{L}_1$ controller to be the best pitch controller followed by (ignoring the PD controllers) Schemes B, A, C, D. This result is echoed in Table 5.11, even though the metric is suspect.

Table 5.12 is the average error between the actual and desired roll angles. None of the Lyapunov MRACs outperformed the PD controller, though they appear to be comparable. The order of the MRAC controllers is Schemes D, A, C, B. Table 5.13, which is the $\mathcal{L}_2$ norm of the error between actual and desired roll angle, shows the order to be different. It shows the order to be Schemes A, B, C, D. This implies that Scheme D oscillated or lagged more than the other algorithms.

Table 5.10: The average pitch angle error (degrees) between the desired and actual pitch angles during flight testing.

	Flight Leg			
Control Scheme	Total	1	2	3
MRAC A	1.8408	2.2742	1.6916	1.4685
MRAC B	1.6422	1.7610	1.6125	1.5483
MRAC C	1.8333	2.4822	1.3327	1.4699
MRAC D (Deliverance)	1.9902	2.6843	1.5250	1.5368
PD (Deliverance)	1.6743	2.2274	1.3709	1.2546
MRAC D (Ray)	1.9358	1.9772	1.9754	1.8582
MRAC D (Phidipidies)	1.4926	1.6556	1.4580	1.3491
$\mathcal{L}_1$	1.2985	1.5783	1.1600	1.1204
PD (Ray)	2.0231	1.9177	2.0745	2.0841

Table 5.11: The $\mathcal{L}_2$ norm of the error between desired and actual pitch for the $\mathcal{L}_1$ and PD controllers.

	Flight Leg			
Control Scheme	Total	1	2	3
MRAC A	205.7361	180.9149	79.7084	57.4211
MRAC B	163.9397	116.9947	77.6737	84.6263
MRAC C	310.5198	295.5876	57.3809	76.0708
MRAC D	362.9292	333.0175	71.9569	125.1219
PD (Deliverance)	246.2636	234.2647	57.6486	49.4401
MRAC D (Ray)	143.6649	101.2380	74.4026	69.7945
MRAC D (Phidipidies)	166.5062	107.6006	95.8230	83.4651
$\mathcal{L}_1$	26.7440	21.0481	12.6149	10.7083
PD (Ray)	111.2848	59.7378	66.2767	66.8197

Table 5.12: The average error between actual and desired roll angle (degrees) during flight tests.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	2.0533	2.1187	2.0205	2.0263
MRAC B	2.3734	2.4823	2.3294	2.3081
MRAC C	2.2482	2.3301	2.1410	2.2729
MRAC D	2.0751	2.2532	1.9773	1.9845
MRAC D (Ray)	2.4159	2.7606	2.3444	2.1032
MRAC D (Phidipidies)	2.0430	2.2307	2.1318	1.7339
PD (Deliverance)	1.9063	2.0621	1.8112	1.8391

Table 5.13: The $\mathcal{L}_2$ norm on the error between actual and desired roll angle for the four MRAC schemes flown on Deliverance.

	Flight Leg			
Scheme	Total	1	2	3
MRAC A	233.6026	137.6504	147.8079	117.9087
MRAC B	354.5922	196.7593	171.1976	240.2888
MRAC C	331.8202	202.1109	180.6441	191.4892
MRAC D	405.9792	195.1959	154.1913	320.8594
MRAC D (Ray)	276.7027	205.5110	148.3329	111.2601
MRAC D (Phidipidies)	315.2272	211.7758	207.1773	107.7029
PD (Deliverance)	207.9683	149.0031	107.0236	98.0799

5.2.4 Summary

MRAC D and PD control have been shown to be comparable in flight testing. MRAC D oscillates less and is more robust than MRAC A and MRAC B. Also, MRAC D performs better overall than MRAC C. Because MRAC D appears to be the best overall MRAC, its robustness will be tested on the other two aircraft shown in Figure 4.9. As the goal is to have the algorithm adapt to other aircraft, MRAC D's gains are not adjusted. Table 5.9 shows the gains used when testing MRAC D on all three aircraft. For all of the adaptive algorithms, transients die out within about 3 seconds for pitch and about 10 seconds for roll. The parameters appear be converging to steady values after adaptation has been running for awhile. This is probably because there is sufficient excitation of the adaptive controllers. The results show that each adaptive control algorithm's performance is comparable to PD control. MRAC Scheme D was shown to be the overall best performer based on its ability to hold altitude, oscillation, and robustness. It was shown to adapt to different platforms and airframe changes effectively.

Although there was an anomaly in Figure 5.26(a) at about 50 seconds, the performance of the $\mathcal{L}_1$ pitch controller was better than the PD controller for this aircraft. There appears to be high frequency content in the control signal, shown in Figure 5.26(b), but significantly less than Lyapunov MRAC controllers (see Figures 5.18-5.21). The $\mathcal{L}_1$ controller should adapt quickly to changes in the plant, such as actuator failure or mechanical damage, because of its high adaptive gain.

Based on performance of the simulation and hardware testing, it appears that $\mathcal{L}_1$ controllers perform better, and have less oscillations on the control effort and output. The $\mathcal{L}_1$ controller is the preferred solution to controlling a MAV with unknown parameters.

Chapter 6

Conclusion

Lyapunov stability is based on the idea that solutions around an equilibrium point stay around that point if it is stable. Lyapunov stability based MRACs look similar to other types of MRAC controllers except they use this idea of stability to define a parameter update law. Parameter estimates of the plant are updated in a manner that guarantees asymptotic convergence of the error between the model and plant. This type of MRAC suffers from high frequency oscillations on the control effort. This can cause unmodeled dynamics to be excited, leading to instability. Also, like gradient based MRACs, the performance of Lyapunov MRACs is dependent upon the magnitude of the reference signal. In contrast, $\mathcal{L}_1$ controllers do not suffer from these maladies.

$\mathcal{L}_1$ controllers are a modification of Lyapunov based MRAC controllers. But, there are two key differences between Lyapunov based MRACs and $\mathcal{L}_1$ controllers. First, Lyapunov MRACs use a traditional model, while $\mathcal{L}_1$ controllers use companion models. Second, $\mathcal{L}_1$ controllers low-pass filter the parameter estimates used in the control effort. The lost frequency content from low-pass filtering the estimates is regained in the companion model. This means that no information is lost. The byproduct of low-pass filtering the parameter estimates is that the adaptive gain can be made large and not cause instability. Furthermore, the magnitude of the reference signal does not effect the response of the system.

The accomplishment of this thesis has been development and application of Lyapunov based MRACs and $\mathcal{L}_1$ controllers on MAVs. Four Lyapunov based MRAC pitch controllers, four Lyapunov based MRAC roll controllers, and four $\mathcal{L}_1$ pitch controllers were developed in Chapter 3. The algorithms were first tested in Matlab.

Next, the algorithms were tested using the Aviones flight simulator to test the code robustness. Next, they were tested using HIL tests to see the performance of the algorithms on the Kestrel Autopilot. Finally, they were flight tested on MAVs. The algorithm testing results are found in Chapter 5, with simulation results found in Section 5.1 and flight testing results found in Section 5.2 on platforms described in Chapter 4. Below are the findings of this thesis.

6.1 Finding

The best performing Lyapunov based MRAC was MRAC Scheme D. Its angle tracking is not as good as MRAC Scheme A, but had less oscillation and is simpler to implement–this makes it a good candidate for a universal autopilot. Flight testing MRAC D was found to effectively adapt to three different airframes without tuning. It was also able to adapt quickly to changes to an airframe caused by deploying a flap.

The first order $\mathcal{L}_1$ adaptive controller with one unknown parameter is the simplest to implement and had better results than the second order model $\mathcal{L}_1$ controllers. In simulation and in flight testing, it was found to have reduced high frequency oscillations on control signal. It appears to have the benefits of traditional MRAC controllers, but is more robust and has faster adaptation.

6.2 Future Work

The work in this thesis can be furthered by continued investigation of the $\mathcal{L}_1$ controller. In particular, study could be done on applicability to roll, scalability, integration with the MAGICC autopilot, expandability to aircraft with nonlinear coefficients, and saturation effect on $\mathcal{L}_1$ controllers.

This thesis only explores $\mathcal{L}_1$ pitch controllers. Work should be done to develop $\mathcal{L}_1$ roll controllers. A possible starting point might be converting the MRAC Scheme D roll controller.

Scalability should be investigated because the $\mathcal{L}_1$ controller should be able to scale to a variety of different aircraft better than Lyapunov MRACs. This is due to its ability to have high adaptive gain, while maintaining acceptable frequency content in the control effort. Also, $\mathcal{L}_1$ controllers' performance does not appear to depend on the magnitude of the reference signal.

Integration with the MAGICC autopilot is another possible branch from this work. $\mathcal{L}_1$ controllers have fast adaptability, and should be able to replace the traditional PID controllers in the MAGICC autopilot code. Work would need to be done to assure acceptable performance during auto-takeoff and to make sure that the $\mathcal{L}_1$ controller stops adapting after landing.

Because $\mathcal{L}_1$ controllers are derived for linear systems, research could also be done to see how $\mathcal{L}_1$ controllers handle aircraft where the coefficients are nonlinear. Of course, $\mathcal{L}_1$ controllers tracking ability increases as adaptive gains increase, but this has only been proven on linear systems.

All proofs in this thesis did not take into account the affect of saturating actuators. In fact, saturation can cause the Lyapunov function derivative to be indefinite, or increasing. To counter this effect, Lyapunov MRAC schemes such as μ-mod or e-mod have been introduced. Since $\mathcal{L}_1$ controllers share a common root, these solutions to saturation should be explored for use on $\mathcal{L}_1$ controllers.

Bibliography

[1] *Unmanned Aircraft Systems Roadmap 2005-2030*, Office of the Secretary of Defense, Aug. 2005.

[2] C. Cao and N. Hovakimyan, “Design and analysis of a novel $\mathcal{L}_1$ adaptive control control architecture with guaranteed transient performance, part i: Control signal and asymptotic stability,” in *Proceedings of American Control Conference*, 2006.

[3] ——, “Design and analysis of a novel $\mathcal{L}_1$ adaptive control control architecture with guaranteed transient performance, ii: Guaranteed transient response,” in *Proceedings of American Control Conference*, 2006.

[4] R. W. Beard, C. Cao, and N. Hovakimyan, “An $\mathcal{L}_1$ adaptive pitch controller for miniature air vehicles,” in *AIAA Conference on Guidance, Navigation, and Control*. Keystone, CO, 2006, (To appear).

[5] Ioannou and Sun, *Robust Adaptive Control*. Prentice Hall, 1996, pp. 149–150,162–165,788–794.

[6] H. Khalil, *Nonlinear Systems*. Prentice Hall, 2002, pp. 111,144–146,151–153,169,172,323–327.

[7] K. Åström and B. Wittenmark, *Adaptive Control*. Addison-Wesley Publishing Company, 1989, pp. 59–66,110,118,264–267.

[8] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley and Sons, Inc., 1996, pp. 514–552.

[9] S. Shatkar and M. Bodson, *Adaptive Control: Stability, Convergence, and Robustness*. Prentice Hall, 1989.

[10] D. Shore and M. Bodson, “Flight testing of a reconfigurable control system on an unmanned aircraft,” in *Proceedings of the 2004 American Control Conference*, June-July 2004, pp. 3747–3752.

[11] M. Steinberg and A. Page, “High-fidelity simulation testing of intelligent and adaptive aircraft control laws,” in *Proceedings of the American Control Conference*, 8-10 May 2002, pp. 3264–3268.

[12] B. Porter and C. Boddy, “Design of adaptive digital controllers incorporating dynamic pole-assignment compensators for high-performance aircraft,” in *Proceedings of the IEEE 1989 Aerospace and Electronics Conference*, 22-26 May 1989, pp. 372–379.

[13] B. Stevens and F. Lewis, *Aircraft Control and Simulation.* John Wiley and Sons, Inc., 2003, pp. 484–500,495.

[14] B. Kim and A. Calise, “Nonlinear flight control using neural networks,” *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 1, pp. 26–33, Jan.-Feb. 1997.

[15] C. Schumacher and R. Kumar, “Adaptive control of uavs in close-coupled formation flight,” in *Proceedings of the Ameerican Control Conference*, Jun. 2000, pp. 849–853.

[16] D. Shin and Y. Kim, “Reconfigurable flight control system design using adaptive neural networks,” *IEEE Transactions on Control Systems Technology*, vol. 12, no. 1, pp. 408–422, Jan. 2004.

[17] R. L. Broderick, “Statistical and adaptive approach for verification of a neural-based flight control system,” in *Digital Avionics Systems Conference*, vol. 2, 24-28 Oct. 2004, pp. 6.E.1–1 – 6.E.1–10.

[18] C. Schumacher and J. D. Johnson, “PI control of a tailless fighter with dynamic inversion and neural networks,” vol. 6, Jun. 1999, pp. 4173–4177.

[19] I. Mareels and B. Ydstie, “Global stability for an MIT rule based adaptive control,” in *Decision and Control, 1989., Proceedings of the 28th IEEE Conference on*, Dec. 1989, pp. 1585–1590.

[20] G. Tao, S. Chen, J. Fei, and S. Joshi, “An adaptive actuator failure compensation scheme for controlling a morphing aircraft model,” in *Proceedings of the 42nd IEEE Conference on Decision and Control*, Dec. 2003, pp. 4926–4931.

[21] S. Chen, G. Tao, and S. Joshi, “An adaptive actuator failure compensation controller for mimo systems,” in *Proceedings of the 41st IEEE Conference on Decision and Control*, December 2002, pp. 4778–4783.

[22] E. Lavretsky, N. Hovakimyan, A. Calise, and V. Stepanyan, “Adaptive vortex seeking formation flight neurocontrol,” in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Aug. 11-14 2003.

[23] J. Farrell, M. Polycarpou, and M. Sharma, “Adaptive backstepping with magnitude, rate, and bandwidth constraints: Aircraft longitude control,” in *Proceedings of the American Control Conference*, June 2003, pp. 3898–3904.

[24] http://sourceforge.net/projects/aviones.

[25] http://procerusuav.com/.

Appendix A

MRAC Code

A.1 Matlab Code

A.1.1 MRAC A Pitch

```
function [sys,x0,str,ts] = pitchMRAC(t,x,u,flag)

km = 1;
gam   = 10;
gam2  = 1;
gam3  = 50;
tau   = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,gam3,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,gam3,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 9;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 1+1+1+6;
sizes.NumInputs      = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0;0; 2.1; 0.06; 2.1; 0; -0.76; -7.2] + [0; 0; 0;randn(3,1); ...
        0; randn(2,1)];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,gam3,tau)

  x_qdot = x(1);
```

```
th_m     = x(2);
x_qdesdot = x(3);
ghat     = x(4:9);

th_c     = uu(1);
z        = [uu(2); uu(3)];
h        = uu(4);
u        = uu(5);
v        = uu(6);
w        = uu(7);
phi      = uu(8);
theta    = uu(9);
psi      = uu(10);
p        = uu(11);
q        = uu(12);
r        = uu(13);

% reference model
th_m_dot = km*(th_c-th_m);

% compute aerodynamic variables
V        = sqrt(u^2+v^2+w^2);
alpha    = atan2(w,u);
beta     = atan2(v,sqrt(u^2+w^2));
q_des = (th_m_dot - gam*(theta-th_m)+r*sin(phi))/cos(phi);

% differentiate q
x_qdot_dot = 1/tau*(q - x_qdot);
qdot       = 1/tau*(q - x_qdot);

q_desdot = 1/tau*(q_des-x_qdesdot);

% regressor
mu = [...
        (-gam2*(q-q_des)-(theta-th_m)*cos(phi))/V^2+q_desdot/V^2;...
        -(p^2-r^2)/V^2;...
        -p*r/V^2;...
        -1;...
        -alpha;...
        -q/V;...
    ];

% parameter adaptation
ghat_dot = -1/(gam3)*(q-q_des)*V^2*mu;
```

```
  Kup = [5,5,5,5,5,5];
  Klow = [−5,−5,−5,−5,−5,−5];


   for i=1:length(ghat),
     if ghat(i) > Kup(i),
        if ghat_dot(i) > 0,
            ghat_dot(i) = 0;
        end
     elseif ghat(i) < Klow(i),
        if ghat_dot(i) < 0,
            ghat_dot(i) = 0;
        end
     end
   end

sys = [x_qdot_dot; th_m_dot; q_desdot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,gam3,tau)

  x_qdot   = x(1);
  th_m     = x(2);
  x_qdesdot = x(3);
  ghat     = x(4:9);

  th_d     = uu(1);
  z        = [uu(2); uu(3)];
  h        = uu(4);
  u        = uu(5);
  v        = uu(6);
  w        = uu(7);
  phi      = uu(8);
  theta    = uu(9);
  psi      = uu(10);
  p        = uu(11);
  q        = uu(12);
  r         = uu(13);

  % compute aerodynamic variables
  V         = sqrt(u^2+v^2+w^2);
  alpha     = atan2(w,u);
  beta      = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  th_m_dot = km*(th_d-th_m);

  % derivative of q
  qdot      = 1/tau*(q-x_qdot);

  q_des = (th_m_dot - gam*(theta-th_m)+r*sin(phi))/cos(phi);
  q_desdot = 1/tau*(q_des-x_qdesdot);

  % regressor
  mu = [...
          (-gam2*(q-q_des)-(theta-th_m)*cos(phi))/V^2+q_desdot/V^2;...
          -(p^2-r^2)/V^2;...
          -p*r/V^2;...
          -1;...
          -alpha;...
          -q/V;...
      ];

  % elevator command
sys = [sat(mu'*ghat,30*pi/180); th_m; theta;ghat];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate

function y=sat(u,lim)
```

```
  if     u <=-lim, y = -lim;
  elseif u >= lim, y = lim;
  else             y = u;
  end
% end sat
```

A.1.2 MRAC A Roll

```
function [sys,x0,str,ts] = rollMRAC(t,x,u,flag)

km = 1;
gam   = 100;
gam2  = 100;
gam3  = 10;
tau   = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,gam3,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,gam3,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 16;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 1+1+1+12;
sizes.NumInputs      = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0;0; 0; 0; 0; 0; 0; 0;0;0;0;0;0;0;0] + ...
       [0; 0; 0;0; 0; 0;0;0;0;0;0;0;0;0;0;0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,gam3,tau)

  x_pdot = x(1);
  phi_m  = x(2);
  x_rdot = x(3);
  x_p_des = x(4);
  ghat    = x(5:16);
```

```
phi_c   = uu(1);
z       = [uu(2); uu(3)];
h       = uu(4);
u       = uu(5);
v       = uu(6);
w       = uu(7);
phi     = uu(8);
theta   = uu(9);
psi     = uu(10);
p       = uu(11);
q       = uu(12);
r       = uu(13);

% reference model
phi_m_dot = km*(phi_c-phi_m);

% compute aerodynamic variables
V       = sqrt(u^2+v^2+w^2);
alpha   = atan2(w,u);
beta    = atan2(v,sqrt(u^2+w^2));

% differentiate q
x_pdot_dot = 1/tau*(p - x_pdot);
pdot       = 1/tau*(p - x_pdot);
rdot       = 1/tau*(r - x_rdot);

p_des = -gam*(phi-phi_m)+phi_m_dot-q*sin(phi)*tan(theta)...
        -r*cos(phi)*tan(theta);

p_des_dot = 1/tau*(p_des - x_p_des);

% regressor
mu = [...
        (-gam2*(p-p_des)-(phi-phi_m)+p_des_dot)/V^2;...
        1/V^2;...
        q/V^2;...
        -1/V;...
        rdot/V^2;...
        -q*r/V^2;...
        -rdot;...
        -r;...
        r/V^2;...
        -1;...
        -beta;...
        -r/V;...
        ];

  % parameter adaptation
  ghat_dot = -1/gam3*(phi-phi_m)*V*mu;

sys = [x_pdot_dot; phi_m_dot; rdot; p_des_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,gam3,tau)

  x_pdot  = x(1);
  phi_m   = x(2);
  x_rdot  = x(3);
  x_p_des = x(4);
  ghat    = x(5:16);

  phi_c   = uu(1);
  z       = [uu(2); uu(3)];
  h       = uu(4);
  u       = uu(5);
  v       = uu(6);
  w       = uu(7);
  phi     = uu(8);
  theta   = uu(9);
  psi     = uu(10);
  p       = uu(11);
  q       = uu(12);
  r       = uu(13);

  % compute aerodynamic variables
  V       = sqrt(u^2+v^2+w^2);
  alpha   = atan2(w,u);
  beta    = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  phi_m_dot = km*(phi_c-phi_m);

  % derivative of q
  pdot     = 1/tau*(p−x_pdot);
  rdot     = 1/tau*(r−x_rdot);

  p_des = −gam*(phi−phi_m)+phi_m_dot−q*sin(phi)*tan(theta)−...
          r*cos(phi)*tan(theta);

  p_des_dot = 1/tau*(p_des − x_p_des);

  % regressor
  mu = [...
          (−gam2*(p−p_des)−(phi−phi_m)+p_des_dot)/V^2;...
          1/V^2;...
          q/V^2;...
          −1/V;...
          rdot/V^2;...
          −q*r/V^2;...
          −rdot;...
          −r;...
          r/V^2;...
          −1;...
          −beta;...
          −r/V;...
      ];

  % aileron command

sys = [sat(mu'*ghat,30*pi/180);phi_m;phi;ghat];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate

function y=sat(u,lim)
  if      u <=−lim, y = −lim;
  elseif u >= lim, y = lim;
  else             y = u;
  end
% end sat
```

A.1.3 MRAC B Pitch

```
function [sys,x0,str,ts] = pitchMRAC(t,x,u,flag)

km = 1;
lambda = 1;
gam    = 10;
tau    = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,lambda,gam,tau);


  % Update
  case 2,
    sys=mdlUpdate(t,x,u);


  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,lambda,gam,tau);


  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);


  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 8;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 6+1+1+1;
sizes.NumInputs     = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0; 2.1; 0.06; 2.1; −1; −2.76; −5.2] + [0; 0; randn(3,1); ...
       0; randn(2,1)];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,lambda,gam,tau)

  x_qdot = x(1);
  th_m   = x(2);
  ghat   = x(3:8);
  th_c     = uu(1);
  z        = [uu(2); uu(3)];
  h        = uu(4);
  u        = uu(5);
  v        = uu(6);
  w        = uu(7);
  phi      = uu(8);
  theta    = uu(9);
  psi      = uu(10);
  p        = uu(11);
  q        = uu(12);
  r        = uu(13);

  % compute aerodynamic variables
  V        = sqrt(u^2+v^2+w^2);
  alpha    = atan2(w,u);
  beta     = atan2(v,sqrt(u^2+w^2));

  % differentiate q
  x_qdot_dot = 1/tau*(q − x_qdot);
  qdot       = 1/tau*(q − x_qdot);

  % reference model
  th_m_dot = km*(th_c−th_m);

  % regressor
  mu = [...
          qdot/V^2;...
          −(p^2−r^2)/V^2;...
          −p*r/V^2;...
          −1;...
          −alpha;...
          −1/V*(th_m_dot−gam*(theta−th_m)+r*sin(phi))/cos(phi);...
      ];

  % parameter adaptation
  ghat_dot = −lambda*(theta−th_m)*V*cos(phi)*mu;

sys = [x_qdot_dot; th_m_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];
```

```
% end mdlUpdate


function sys=mdlOutputs(t,x,uu,km,lambda,gam,tau)

  x_qdot  = x(1);
  th_m    = x(2);
  ghat    = x(3:8);

  th_c    = uu(1);
  z       = [uu(2); uu(3)];
  h       = uu(4);
  u       = uu(5);
  v       = uu(6);
  w       = uu(7);
  phi     = uu(8);
  theta   = uu(9);
  psi     = uu(10);
  p       = uu(11);
  q       = uu(12);
  r       = uu(13);

  % compute aerodynamic variables
  V       = sqrt(u^2+v^2+w^2);
  alpha   = atan2(w,u);
  beta    = atan2(v,sqrt(u^2+w^2));

  % reference model
  th_m_dot = km*(th_c-th_m);

  % derivative of q
  qdot     = 1/tau*(q-x_qdot);

  % regressor
  mu = [...
          qdot/V^2;...
          -(p^2-r^2)/V^2;...
          -p*r/V^2;...
          -1;...
          -alpha;...
          -1/V*(th_m_dot-gam*(theta-th_m)+r*sin(phi))/cos(phi);...
      ];

  % elevator command
```

```
sys = [ghat;theta;th_m;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs


function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit


function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate


function y=sat(u,lim)
  if      u <=-lim, y = -lim;
  elseif u >= lim, y = lim;
  else              y = u;
  end
% end sat
```

A.1.4 MRAC B Roll

```
function [sys,x0,str,ts] = rollMRAC(t,x,u,flag)

km = .7;
gam   = 100;
gam2  = .06;
tau   = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,tau);
  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 12;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 9+1+1+1;
sizes.NumInputs    = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0 = [0; 0;0; 0; 0; 0.0000001; .0000003; -0.00015641; 0.0000033;...
        -0.0000016;0;-0.0000006] + [0; 0; 0;0; 0; 0;0;0;0;0;0;0];
```

```
%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,tau)

  x_pdot = x(1);
  phi_m  = x(2);
  x_rdot = x(3);
  ghat   = x(4:12);

  phi_c  = uu(1);
  z      = [uu(2); uu(3)];
  h      = uu(4);
  u      = uu(5);
  v      = uu(6);
  w      = uu(7);
  phi    = uu(8);
  theta  = uu(9);
  psi    = uu(10);
  p      = uu(11);
  q      = uu(12);
  r      = uu(13);

  % reference model
  phi_m_dot = km*(phi_c-phi_m);

  % compute aerodynamic variables
  V      = sqrt(u^2+v^2+w^2);
  alpha  = atan2(w,u);
  beta   = atan2(v,sqrt(u^2+w^2));

  % differentiate p r and p_dot
  x_pdot_dot = 1/tau*(p - x_pdot);
  pdot       = 1/tau*(p - x_pdot);
  rdot       = 1/tau*(r - x_rdot);
  M = -gam*(phi-phi_m)+phi_m_dot-q*sin(phi)*tan(theta)-...
        r*cos(phi)*tan(theta);
  % regressor
  mu = [...
          pdot/V^2;...
          0*rdot/V^2;...
          -q*r/V^2;...
          r/V^2;...
          -1;...
          -beta;...
          r/(2*V);...
          q*M/V^2;...
          -M/(2*V);...
      ];

  % parameter adaptation
  ghat_dot = -gam2*(phi-phi_m)*V*mu;

sys = [x_pdot_dot; phi_m_dot; rdot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,tau)

  x_pdot = x(1);
  phi_m  = x(2);
  x_rdot = x(3);
  ghat   = x(4:12);

  phi_c  = uu(1);
  z      = [uu(2); uu(3)];
  h      = uu(4);
  u      = uu(5);
  v      = uu(6);
  w      = uu(7);
  phi    = uu(8);
  theta  = uu(9);
  psi    = uu(10);
  p      = uu(11);
  q        = uu(12);
  r        = uu(13);

  % compute aerodynamic variables
  V        = sqrt(u^2+v^2+w^2);
  alpha    = atan2(w,u);
  beta     = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  phi_m_dot = km*(phi_c−phi_m);

  % derivative of p and r
  pdot      = 1/tau*(p−x_pdot);
  rdot      = 1/tau*(r−x_rdot);

  M = −gam*(phi−phi_m)+phi_m_dot−q*sin(phi)*tan(theta)...
        −r*cos(phi)*tan(theta);

  % regressor
  mu = [...
          pdot/V^2;...
          0*rdot/V^2;...
          −q*r/V^2;...
          r/V^2;...
          −1;...
          −beta;...
          r/(2*V);...
          q*M/V^2;...
          −M/(2*V);...
      ];

  % aileron command

sys = [ghat;phi;phi_m;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate

function y=sat(u,lim)
  if      u <=−lim, y = −lim;
  elseif u >= lim, y = lim;
  else                    y = u;
  end
% end sat
```

A.1.5 MRAC C Pitch

```
function [sys,x0,str,ts] = pitchMRAC(t,x,u,flag)

km = 1;
gam   = 100;
gam2  = .1;
tau    = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs  = 3+1+1+1;
sizes.NumInputs     = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0 = [0; 0;0; 0] + [0; 0; 0;0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,tau)

  theta_m  = x(1);
  ghat     = x(2:4);
  theta_c   = uu(1);
  z         = [uu(2); uu(3)];
  h         = uu(4);
  u         = uu(5);
  v         = uu(6);
  w         = uu(7);
  phi       = uu(8);
  theta     = uu(9);
  psi       = uu(10);
  p         = uu(11);
  q         = uu(12);
  r         = uu(13);

  % reference model
  theta_m_dot = km*(theta_c-theta_m);

  % compute aerodynamic variables
  V         = sqrt(u^2+v^2+w^2);
  alpha     = atan2(w,u);
  beta      = atan2(v,sqrt(u^2+w^2));

  % regressor
  mu = [...
          (theta_m_dot)/V + -gam*(theta_m-theta);...
          -1;...
          -1/V^2;...
      ];


  % parameter adaptation
  ghat_dot = -gam2*(theta-theta_m)*V*mu;

sys = [theta_m_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,tau)
```

```
  theta_m  = x(1);
  ghat     = x(2:4);

  theta_c   = uu(1);
  z        = [uu(2); uu(3)];
  h        = uu(4);
  u        = uu(5);
  v        = uu(6);
  w        = uu(7);
  phi      = uu(8);
  theta    = uu(9);
  psi      = uu(10);
  p        = uu(11);
  q        = uu(12);
  r        = uu(13);

  % compute aerodynamic variables
  V        = sqrt(u^2+v^2+w^2);
  alpha    = atan2(w,u);
  beta     = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  theta_m_dot = km*(theta_c-theta_m);

  % regressor
  mu = [...
        -(theta_m_dot)/V + -gam*(theta_m-theta);...
        -1;...
        -1/V^2;...
     ];

  % elevator command

sys = [ghat;theta;theta_m;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate

function y=sat(u,lim)
  if      u <=-lim, y = -lim;
  elseif u >= lim, y = lim;
  else                y = u;
  end
% end sat
```

A.1.6 MRAC C Roll

```
function [sys,x0,str,ts] = rollMRAC(t,x,u,flag)

km = 1;
gam   = 50;
gam2  = .01;
tau   = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 3+1+1+1;
sizes.NumInputs     = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0;0; 0] + [0; 0; 0;0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,tau)

  phi_m  = x(1);
  ghat   = x(2:4);
  phi_c   = uu(1);
  z       = [uu(2); uu(3)];
  h       = uu(4);
  u       = uu(5);
  v       = uu(6);
  w       = uu(7);
  phi     = uu(8);
  theta   = uu(9);
  psi     = uu(10);
  p       = uu(11);
  q       = uu(12);
  r       = uu(13);

  % reference model
  phi_m_dot = km*(phi_c-phi_m);

  % compute aerodynamic variables
  V       = sqrt(u^2+v^2+w^2);
  alpha   = atan2(w,u);
  beta    = atan2(v,sqrt(u^2+w^2));

  % regressor
  mu = [...
         (phi_m_dot)/V + -gam*(phi_m-phi);...
         -1;...
         -1/V^2;...
     ];


  % parameter adaptation
  ghat_dot = -gam2*(phi-phi_m)*V*mu;

sys = [phi_m_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,tau)
```

```
  phi_m  = x(1);
  ghat   = x(2:4);

  phi_c  = uu(1);
  z      = [uu(2); uu(3)];
  h      = uu(4);
  u      = uu(5);
  v      = uu(6);
  w      = uu(7);
  phi    = uu(8);
  theta  = uu(9);
  psi    = uu(10);
  p      = uu(11);
  q      = uu(12);
  r      = uu(13);

  % compute aerodynamic variables
  V      = sqrt(u^2+v^2+w^2);
  alpha  = atan2(w,u);
  beta   = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  phi_m_dot = km*(phi_c-phi_m);

  % regressor
  mu = [...
         -(phi_m_dot)/V + -gam*(phi_m-phi);...
         -1;...
         -1/V^2;...
     ];

  % aileron command

sys = [ghat;phi;phi_m;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate

function y=sat(u,lim)
  if     u <=-lim, y = -lim;
  elseif u >= lim, y = lim;
  else             y = u;
  end
% end sat
```

A.1.7 MRAC D Pitch

```
function [sys,x0,str,ts] = pitchMRAC(t,x,u,flag)

km = 5;
gam   = 100;
gam2  = 1;
tau   = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 3+1+1+1;
sizes.NumInputs     = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0;0; 0] + [0; 0; 0;0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,tau)

  theta_m  = x(1);
  ghat     = x(2:4);
  theta_c   = uu(1);
  z         = [uu(2); uu(3)];
  h         = uu(4);
  u         = uu(5);
  v         = uu(6);
  w         = uu(7);
  phi       = uu(8);
  theta     = uu(9);
  psi       = uu(10);
  p         = uu(11);
  q         = uu(12);
  r         = uu(13);

  % reference model
  theta_m_dot = km*(theta_c-theta_m);

  % compute aerodynamic variables
  V         = sqrt(u^2+v^2+w^2);
  alpha     = atan2(w,u);
  beta      = atan2(v,sqrt(u^2+w^2));

  % regressor
  mu = [...
        (theta_m_dot)/V + -gam*(theta_m-theta);...
        -1;...
        0*-1/V^2;...
    ];


  % parameter adaptation
  ghat_dot = -gam2*(theta-theta_m)*V*mu;

sys = [theta_m_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,tau)
```

```
  theta_m  = x(1);
  ghat     = x(2:4);

  theta_c   = uu(1);
  z         = [uu(2); uu(3)];
  h         = uu(4);
  u         = uu(5);
  v         = uu(6);
  w         = uu(7);
  phi       = uu(8);
  theta     = uu(9);
  psi       = uu(10);
  p         = uu(11);
  q         = uu(12);
  r         = uu(13);

  % compute aerodynamic variables
  V         = sqrt(u^2+v^2+w^2);
  alpha     = atan2(w,u);
  beta      = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  theta_m_dot = km*(theta_c-theta_m);

  % regressor
  mu = [...
          -(theta_m_dot)/V + -gam*(theta_m-theta);...
          -1;...
          0*-1/V^2;...
      ];

  % elevator command

sys = [ghat;theta;theta_m;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate

function y=sat(u,lim)
  if      u <=−lim, y = −lim;
  elseif u >= lim, y = lim;
  else                  y = u;
  end
% end sat
```

A.1.8 MRAC D Roll

```
function [sys,x0,str,ts] = rollMRAC(t,x,u,flag)

km = 1;
gam   = 30;
gam2  = .01;
tau    = 1;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,u,km,gam,gam2,tau);

  % Update
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,u,km,gam,gam2,tau);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes(C)

sizes = simsizes;

sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 3+1+1+1;
sizes.NumInputs     = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
% approximate coefficients
x0  = [0; 0;0; 0] + [0; 0; 0;0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,km,gam,gam2,tau)

  phi_m  = x(1);
  ghat    = x(2:4);
  phi_c   = uu(1);
  z       = [uu(2); uu(3)];
  h       = uu(4);
  u       = uu(5);
  v       = uu(6);
  w       = uu(7);
  phi     = uu(8);
  theta   = uu(9);
  psi     = uu(10);
  p       = uu(11);
  q       = uu(12);
  r       = uu(13);

  % reference model
  phi_m_dot = km*(phi_c−phi_m);

  % compute aerodynamic variables
  V       = sqrt(u^2+v^2+w^2);
  alpha   = atan2(w,u);
  beta    = atan2(v,sqrt(u^2+w^2));

  % regressor
  mu = [...
          (phi_m_dot)/V + −gam*(phi_m−phi);...
          −1;...
          0*−1/V^2;...
      ];


  % parameter adaptation
  ghat_dot = −gam2*(phi−phi_m)*V*mu;

sys = [phi_m_dot; ghat_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,km,gam,gam2,tau)
```

```
  phi_m   = x(1);
  ghat    = x(2:4);

  phi_c   = uu(1);
  z       = [uu(2); uu(3)];
  h       = uu(4);
  u       = uu(5);
  v       = uu(6);
  w       = uu(7);
  phi     = uu(8);
  theta   = uu(9);
  psi     = uu(10);
  p       = uu(11);
  q       = uu(12);
  r       = uu(13);

  % compute aerodynamic variables
  V       = sqrt(u^2+v^2+w^2);
  alpha   = atan2(w,u);
  beta    = atan2(v,sqrt(u^2+w^2));

  % reference model pitch
  phi_m_dot = km*(phi_c-phi_m);

  % regressor
  mu = [...
        -(phi_m_dot)/V + -gam*(phi_m-phi);...
        -1;...
        0*-1/V^2;...
     ];

  % aileron command

sys = [ghat;phi_m;phi;sat(mu'*ghat,30*pi/180)];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate


function y=sat(u,lim)
  if     u <=-lim, y = -lim;
  elseif u >= lim, y = lim;
  else             y = u;
  end
% end sat
```

A.2 Dynamic C Code

A.2.1 MRAC A Pitch

```
/*** BeginHeader AdaptiveControl_PitchA */
void AdaptiveControl_PitchA(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_PitchA(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
{
#ifdef ADAPTIVE_PITCH_A
        float theta_tilde, psi1, psi2, psi3, psi4, psi5, psi6;
        //for redefinitions so code is easier to read
        float V, theta, phi, p, q, r, dt;
        //used in algorithm A
        float q_d, q_tilde, tau, q_d_dot;
        float alpha, lambda1, lambda2, gamma1, gamma2, gamma3, gamma4;
        float gamma5, gamma6;
        float leakage_gain1, leakage_gain2, leakage_gain3, leakage_gain4;
        float leakage_gain5, leakage_gain6, dz_threshold;

        char outside_threshold;
        float psi_norm_div;
    float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_PITCH_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_PITCH_LAMBDA1];
        lambda2 = m_floats[MF_ADAPTIVE_PITCH_LAMBDA2];
    gamma1 = m_floats[MF_ADAPTIVE_PITCH_GAMMA1];
    gamma2 = m_floats[MF_ADAPTIVE_PITCH_GAMMA2];
    gamma3 = m_floats[MF_ADAPTIVE_PITCH_GAMMA3];
    gamma4 = m_floats[MF_ADAPTIVE_PITCH_GAMMA4];
    gamma5 = m_floats[MF_ADAPTIVE_PITCH_GAMMA5];
    gamma6 = m_floats[MF_ADAPTIVE_PITCH_GAMMA6];
    leakage_gain1 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE1];
    leakage_gain2 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE2];
    leakage_gain3 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE3];
    leakage_gain4 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE4];
    leakage_gain5 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE5];
    leakage_gain6 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE6];
    //deadzone threshold; radians
    dz_threshold = m_floats[MF_ADAPTIVE_PITCH_DZ_THRESHOLD];

    //states
    V = *(a->airspeed);
    theta = *(a->theta);
    phi = *(a->phi);
    p = *(a->p);
    q = *(a->q);
    r = *(a->r);
    dt = m_floats[MF_DT];

    //reference model
    theta_tilde = theta - a->theta_m;
    a->theta_m += dt * alpha* (theta_d - a->theta_m);

    //q_desired
if (cos(phi)==0){
    q_d = 0;}
else{
    q_d = (-lambda1*theta_tilde + alpha*(theta_d - a->theta_m) +
            r*sin(phi)) / cos(phi);}
    q_tilde = q - q_d;

    //dirty derivative for q_d_dot.
    tau = 0.1;
    q_d_dot = (q_d - a->q_d_dot_int) / tau;
    a->q_d_dot_int += q_d_dot * dt;

    //Psi
if (V*V == 0){
    psi1 = 0;
    psi2 = 0;
    psi3 = 0;
    psi4 = −1;
    psi5 = −theta;
    psi6 = 0;
}
else{
    psi1 = (−lambda2*q_tilde − theta_tilde*cos(phi) +
    q_d_dot)/(V*V);
    psi2 = (p*p − r*r) / (V*V);
    psi3 = (−p*r) / (V*V);
    psi4 = −1;
    psi5 = −theta;
    psi6 = −q / V;
}

//Normalize
if (normalize){
    psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3 +
                        psi4*psi4 + psi5*psi5 + psi6*psi6);
    if (psi_norm_div != 0){
                psi1 = psi1/psi_norm_div;
                psi2 = psi2/psi_norm_div;
                psi3 = psi3/psi_norm_div;
                psi4 = psi4/psi_norm_div;
                psi5 = psi5/psi_norm_div;
                psi6 = psi6/psi_norm_div;
    }
}

    //Deadzone
    outside_threshold = fabs(theta_tilde) > dz_threshold;
    if ((deadzone&&outside_threshold) || !deadzone){
        a−>k_hat1 += −gamma1 * q_tilde * psi1 * dt * V ;
        a−>k_hat2 += −gamma2 * q_tilde * psi2 * dt * V ;
        a−>k_hat3 += −gamma3 * q_tilde * psi3 * dt * V ;
        a−>k_hat4 += −gamma4 * q_tilde * psi4 * dt * V ;
        a−>k_hat5 += −gamma5 * q_tilde * psi5 * dt * V ;
        a−>k_hat6 += −gamma6 * q_tilde * psi6 * dt * V ;

        //Leakage
        if (leakage){
            a−>k_hat1 −= leakage_gain1*a−>k_hat1;
            a−>k_hat2 −= leakage_gain2*a−>k_hat2;
            a−>k_hat3 −= leakage_gain3*a−>k_hat3;
            a−>k_hat4 −= leakage_gain4*a−>k_hat4;
```

```
                a->k_hat5 -= leakage_gain5*a->k_hat5;
                a->k_hat6 -= leakage_gain5*a->k_hat6;
            }
        }

        //delta_e_desired
        effort = psi1*a->k_hat1 + psi2*a->k_hat2 + psi3*a->k_hat3 +
            psi4*a->k_hat4 + psi5*a->k_hat5 + psi6*a->k_hat6;

        if(invert_result)
            effort *= -1;

    if(sum_result)
            *(a->elevator) += effort;
        else
            *(a->elevator) = effort;

        m_floats[MF_ADAPTIVE_PITCH_K1] = a->k_hat1;
        m_floats[MF_ADAPTIVE_PITCH_K2] = a->k_hat2;
        m_floats[MF_ADAPTIVE_PITCH_K3] = a->k_hat3;
        m_floats[MF_ADAPTIVE_PITCH_K4] = a->k_hat4;
        m_floats[MF_ADAPTIVE_PITCH_K5] = a->k_hat5;
        m_floats[MF_ADAPTIVE_PITCH_K6] = a->k_hat6;
        m_floats[MF_ADAPTIVE_THETA_REFERENCE_MODEL] = a->theta_m;

#endif
}
```

A.2.2 MRAC A Roll

```
/*** BeginHeader AdaptiveControl_RollA */
void AdaptiveControl_RollA(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_RollA(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
{
#ifdef ADAPTIVE_ROLL_A
        float phi_tilde, psi1, psi2, psi3, psi4, psi5, psi6, psi7;
```

```
    //for redefinitions so code is easier to read
    float V, theta, phi, p, q, r, groundtrack, dt;
    float p_d, p_tilde, tau, p_d_dot, r_dot;//used in algorithm A
    float alpha, lambda1, lambda2, gamma1, gamma2, gamma3, gamma4;
    float gamma5, gamma6, gamma7;
    float leakage_gain1, leakage_gain2, leakage_gain3, leakage_gain4;
    float leakage_gain5, leakage_gain6, leakage_gain7, dz_threshold;

    char outside_threshold;
    float psi_norm_div;
float32 effort;

    //gains
    alpha = m_floats[MF_ADAPTIVE_ROLL_ALPHA];
    lambda1 = m_floats[MF_ADAPTIVE_ROLL_LAMBDA1];
    lambda2 = m_floats[MF_ADAPTIVE_ROLL_LAMBDA2];
    gamma1 = m_floats[MF_ADAPTIVE_ROLL_GAMMA1];
    gamma2 = m_floats[MF_ADAPTIVE_ROLL_GAMMA2];
    gamma3 = m_floats[MF_ADAPTIVE_ROLL_GAMMA3];
    gamma4 = m_floats[MF_ADAPTIVE_ROLL_GAMMA4];
    gamma5 = m_floats[MF_ADAPTIVE_ROLL_GAMMA5];
    gamma6 = m_floats[MF_ADAPTIVE_ROLL_GAMMA6];
 gamma7 = m_floats[MF_ADAPTIVE_ROLL_GAMMA7];
    leakage_gain1 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE1];
    leakage_gain2 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE2];
    leakage_gain3 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE3];
    leakage_gain4 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE4];
    leakage_gain5 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE5];
    leakage_gain6 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE6];
 leakage_gain7 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE7];
//deadzone threshold; radians
    dz_threshold = m_floats[MF_ADAPTIVE_ROLL_DZ_THRESHOLD];

    //states
    V = *(a->airspeed);
    theta = *(a->theta);
    phi = *(a->phi);
    groundtrack = *(a->groundtrack);
    p = *(a->p);
    q = *(a->q);
    r = *(a->r);
    dt = m_floats[MF_DT];

    //reference model
    phi_tilde = phi - a->phi_m;
    a->phi_m += dt * alpha * (phi_d - a->phi_m);

    p_d = -lambda1*phi_tilde + alpha*(phi_d - a->phi_m) -
            q*sin(phi)*tan(theta) - r*cos(phi)*tan(theta);
    p_tilde = p - p_d;

    //dirty derivative of p_d
    tau = 0.1;
    p_d_dot = (p_d - a->p_d_dot_int) / tau;
    a->p_d_dot_int += p_d_dot * dt;

//r dot
r_dot = (r - a->r_dot_int) / tau;
a->r_dot_int += r_dot * dt;

if (V*V == 0){
      psi1 = 0;
      psi2 = 0;
      psi3 = 0;
      psi4 = -1.0;
      psi5 = groundtrack;
      psi6 = 0;
      psi7 = 0;

}
else{
      psi1 = (-lambda2*p_tilde + p_d_dot - phi_tilde)/(V*V);
      psi2 = -(p*q + r_dot)/(V*V);
      psi3 = -q*r/(V*V);
      psi4 = -1.0;
      psi5 = groundtrack;
      psi6 = -p/V;
      psi7 = -r/V;
}

//Normalize
if (normalize){
    psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3 +
          psi4*psi4 + psi5*psi5 + psi6*psi6 + psi7*psi7);
    if (psi_norm_div != 0){
        psi1 = psi1/psi_norm_div;
        psi2 = psi2/psi_norm_div;
        psi3 = psi3/psi_norm_div;
        psi4 = psi4/psi_norm_div;
        psi5 = psi5/psi_norm_div;
```

```
            psi6 = psi6/psi_norm_div;
            psi7 = psi7/psi_norm_div;
        }
    }

        //Deadzone
        outside_threshold = fabs(phi_tilde) > dz_threshold;
        if ((deadzone&&outside_threshold) || !deadzone){
                a->k_hat1 += dt * -gamma1 * p_tilde * psi1 * V;
                a->k_hat2 += dt * -gamma2 * p_tilde * psi2 * V;
                a->k_hat3 += dt * -gamma3 * p_tilde * psi3 * V;
                a->k_hat4 += dt * -gamma4 * p_tilde * psi4 * V;
                a->k_hat5 += dt * -gamma5 * p_tilde * psi5 * V;
                a->k_hat6 += dt * -gamma6 * p_tilde * psi6 * V;
        a->k_hat7 += dt * -gamma7 * p_tilde * psi7 * V;
                //Leakage
                if (leakage){
                        a->k_hat1 -= leakage_gain1*a->k_hat1;
                        a->k_hat2 -= leakage_gain2*a->k_hat2;
                        a->k_hat3 -= leakage_gain3*a->k_hat3;
                        a->k_hat4 -= leakage_gain4*a->k_hat4;
                        a->k_hat5 -= leakage_gain5*a->k_hat5;
                        a->k_hat6 -= leakage_gain6*a->k_hat6;
                        a->k_hat7 -= leakage_gain7*a->k_hat7;
                }
        }

        effort = psi1*a->k_hat1 + psi2*a->k_hat2 + psi3*a->k_hat3 +
                         psi4*a->k_hat4 + psi5*a->k_hat5 + psi6*a->k_hat6 +
                         psi7*a->k_hat7;

        if(invert_result)
                effort *= -1;

    if(sum_result)
                *(a->aileron) += effort;
        else
                *(a->aileron) = effort;

        m_floats[MF_ADAPTIVE_ROLL_K1] = a->k_hat1;
        m_floats[MF_ADAPTIVE_ROLL_K2] = a->k_hat2;
        m_floats[MF_ADAPTIVE_ROLL_K3] = a->k_hat3;
        m_floats[MF_ADAPTIVE_ROLL_K4] = a->k_hat4;
        m_floats[MF_ADAPTIVE_ROLL_K5] = a->k_hat5;
        m_floats[MF_ADAPTIVE_ROLL_K6] = a->k_hat6;
        m_floats[MF_ADAPTIVE_ROLL_K7] = a->k_hat7;
        m_floats[MF_ADAPTIVE_PHI_REFERENCE_MODEL] = a->phi_m;

#endif
}
```

A.2.3 MRAC B Pitch

```
/*** BeginHeader AdaptiveControl_PitchB */
void AdaptiveControl_PitchB(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_PitchB(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
{
#ifdef ADAPTIVE_PITCH_B
        float theta_tilde, psi1, psi2, psi3, psi4, psi5;
        //for redefinitions so code is easier to read
        float V, theta, phi, p, q, r, dt;
        float alpha, lambda1, lambda2, gamma1, gamma2, gamma3, gamma4;
        float gamma5, leakage_gain1, leakage_gain2, leakage_gain3, leakage_gain4,
        float leakage_gain5, dz_threshold;

        char outside_threshold;
        float psi_norm_div;
    float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_PITCH_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_PITCH_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_PITCH_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_PITCH_GAMMA2];
        gamma3 = m_floats[MF_ADAPTIVE_PITCH_GAMMA3];
        gamma4 = m_floats[MF_ADAPTIVE_PITCH_GAMMA4];
        gamma5 = m_floats[MF_ADAPTIVE_PITCH_GAMMA5];
        leakage_gain1 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE2];
        leakage_gain3 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE3];
        leakage_gain4 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE4];
        leakage_gain5 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE5];
        //deadzone threshold; radians
    dz_threshold = m_floats[MF_ADAPTIVE_PITCH_DZ_THRESHOLD];

    //states
    V = *(a->airspeed);
    theta = *(a->theta);
    phi = *(a->phi);
    p = *(a->p);
    q = *(a->q);
    r = *(a->r);
    dt = m_floats[MF_DT];

    //reference model
    theta_tilde = theta - a->theta_m;
    a->theta_m += dt * alpha* (theta_d - a->theta_m);

if ( V * cos(phi) == 0){
    psi1 = 0;}
else{
    psi1 = (-lambda1 * theta_tilde + alpha * (theta_d - a->theta_m)
            + r * sin(phi)) / (V * cos(phi));}
    psi2 = 1;
    psi3 = p*p - r*r;
    psi4 = -p * r;
    psi5 = -theta;

    //Normalize
    if (normalize){
            psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3 +
                                    psi4*psi4 + psi5*psi5);
    if (psi_norm_div != 0){
          psi1 = psi1/psi_norm_div;
          psi2 = psi2/psi_norm_div;
          psi3 = psi3/psi_norm_div;
          psi4 = psi4/psi_norm_div;
          psi5 = psi5/psi_norm_div;
  }
    }

    //Deadzone
    outside_threshold = fabs(theta_tilde) > dz_threshold;
    if ((deadzone&&outside_threshold) || !deadzone){
            a->k_hat1 += -dt * gamma1 * theta_tilde * V * psi1;
            a->k_hat2 += -dt * gamma2 * theta_tilde * V * psi2;
            a->k_hat3 += -dt * gamma3 * theta_tilde * V * psi3;
            a->k_hat4 += -dt * gamma4 * theta_tilde * V * psi4;
```

```
                a->k_hat5 += -dt * gamma5 * theta_tilde * V * psi5;

                //Leakage
                if (leakage){
                        a->k_hat1 -= leakage_gain1*a->k_hat1;
                        a->k_hat2 -= leakage_gain2*a->k_hat2;
                        a->k_hat3 -= leakage_gain3*a->k_hat3;
                        a->k_hat4 -= leakage_gain4*a->k_hat4;
                        a->k_hat5 -= leakage_gain5*a->k_hat5;
                }
        }

        effort = psi1*a->k_hat1 + psi2*a->k_hat2 + psi3*a->k_hat3 +
                        psi4*a->k_hat4 + psi5*a->k_hat5;

        if(invert_result)
                effort *= -1;

   if(sum_result)
                *(a->elevator) += effort;
        else
                *(a->elevator) = effort;

        m_floats[MF_ADAPTIVE_PITCH_K1] = a->k_hat1;
        m_floats[MF_ADAPTIVE_PITCH_K2] = a->k_hat2;
        m_floats[MF_ADAPTIVE_PITCH_K3] = a->k_hat3;
        m_floats[MF_ADAPTIVE_PITCH_K4] = a->k_hat4;
        m_floats[MF_ADAPTIVE_PITCH_K5] = a->k_hat5;
        m_floats[MF_ADAPTIVE_PITCH_K6] = a->k_hat6;
        m_floats[MF_ADAPTIVE_THETA_REFERENCE_MODEL] = a->theta_m;

#endif
}
```

A.2.4 MRAC B Roll

```
/*** BeginHeader AdaptiveControl_RollB */
void AdaptiveControl_RollB(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_RollB(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
```

```
{
#ifdef ADAPTIVE_ROLL_B
        float phi_tilde, psi1, psi2, psi3, psi4, psi5, psi6, psi7;
    float V, theta, phi, groundtrack, p, q, r, dt;
    float r_dot, tau;
    float alpha, lambda1, lambda2, gamma1, gamma2, gamma3, gamma4;
    float gamma5, gamma6, gamma7;
    float leakage_gain1, leakage_gain2, leakage_gain3, leakage_gain4
    float leakage_gain5, leakage_gain6, leakage_gain7, dz_threshold;

    char outside_threshold;
    float psi_norm_div;
    float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_ROLL_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_ROLL_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_ROLL_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_ROLL_GAMMA2];
        gamma3 = m_floats[MF_ADAPTIVE_ROLL_GAMMA3];
        gamma4 = m_floats[MF_ADAPTIVE_ROLL_GAMMA4];
        gamma5 = m_floats[MF_ADAPTIVE_ROLL_GAMMA5];
        gamma6 = m_floats[MF_ADAPTIVE_ROLL_GAMMA6];
        gamma7 = m_floats[MF_ADAPTIVE_ROLL_GAMMA7];
        leakage_gain1 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE2];
        leakage_gain3 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE3];
        leakage_gain4 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE4];
        leakage_gain5 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE5];
        leakage_gain6 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE6];
        leakage_gain7 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE7];
        //deadzone threshold; radians
        dz_threshold = m_floats[MF_ADAPTIVE_ROLL_DZ_THRESHOLD];

        //states
        V = *(a->airspeed);
        theta = *(a->theta);
        phi = *(a->phi);
        groundtrack = *(a->groundtrack);
        p = *(a->p);
        q = *(a->q);
        r = *(a->r);
        dt = m_floats[MF_DT];

        //reference model
    phi_tilde = phi - a->phi_m;
    a->phi_m += dt * alpha * (phi_d - a->phi_m);

//r dot dirty derivative
tau = 0.1;
r_dot = (r - a->r_dot_int) / tau;
a->r_dot_int += r_dot * dt;

if (V == 0){
      psi1 = 0;
      psi2 = 0;
      psi3 = 0;
      psi4 = 0;
      psi5 = -1.0;
      psi6 = groundtrack;
      psi7 = 0;
}
else{
    psi1 = (-lambda1*phi_tilde - q*sin(phi)*tan(theta) -
          r*cos(phi)*tan(theta) + alpha*(phi_d-a->phi_m))/(2.0*V);
    psi2 = -4.0/(V*V);
    psi3 = -4.0*(p*q + r_dot)/(V*V);
    psi4 = -4.0*q*r/(V*V);
    psi5 = -1.0;
    psi6 = groundtrack;
    psi7 = -r/(2.0*V);
}

//Normalize
if (normalize){
    psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3 +
          psi4*psi4 + psi5*psi5 + psi6*psi6 + psi7*psi7);
    if (psi_norm_div != 0){
        psi1 = psi1/psi_norm_div;
        psi2 = psi2/psi_norm_div;
        psi3 = psi3/psi_norm_div;
        psi4 = psi4/psi_norm_div;
        psi5 = psi5/psi_norm_div;
        psi6 = psi6/psi_norm_div;
        psi7 = psi7/psi_norm_div;
    }
}

    //Deadzone
    outside_threshold = fabs(phi_tilde) > dz_threshold;
```

```
        if ((deadzone&&outside_threshold) || !deadzone){
                a->k_hat1 += dt * -2.0 * gamma1 * V * phi_tilde * psi1;
                a->k_hat2 += dt * -2.0 * gamma2 * V * phi_tilde * psi2;
                a->k_hat3 += dt * -2.0 * gamma3 * V * phi_tilde * psi3;
                a->k_hat4 += dt * -2.0 * gamma4 * V * phi_tilde * psi4;
                a->k_hat5 += dt * -2.0 * gamma5 * V * phi_tilde * psi5;
                a->k_hat6 += dt * -2.0 * gamma6 * V * phi_tilde * psi6;
                a->k_hat7 += dt * -2.0 * gamma7 * V * phi_tilde * psi7;
                //Leakage
                if (leakage){
                        a->k_hat1 -= leakage_gain1*a->k_hat1;
                        a->k_hat2 -= leakage_gain2*a->k_hat2;
                        a->k_hat3 -= leakage_gain3*a->k_hat3;
                        a->k_hat4 -= leakage_gain4*a->k_hat4;
                        a->k_hat5 -= leakage_gain5*a->k_hat5;
                        a->k_hat6 -= leakage_gain6*a->k_hat6;
                        a->k_hat7 -= leakage_gain7*a->k_hat7;
                }
        }

        effort = psi1*a->k_hat1 + psi2*a->k_hat2 + psi3*a->k_hat3 +
                psi4*a->k_hat4 + psi5*a->k_hat5 + psi6*a->k_hat6 +
                psi7*a->k_hat7;

        if(invert_result)
                effort *= -1;

    if(sum_result)
                *(a->aileron) += effort;
        else
                *(a->aileron) = effort;

        m_floats[MF_ADAPTIVE_ROLL_K1] = a->k_hat1;
        m_floats[MF_ADAPTIVE_ROLL_K2] = a->k_hat2;
        m_floats[MF_ADAPTIVE_ROLL_K3] = a->k_hat3;
        m_floats[MF_ADAPTIVE_ROLL_K4] = a->k_hat4;
        m_floats[MF_ADAPTIVE_ROLL_K5] = a->k_hat5;
        m_floats[MF_ADAPTIVE_ROLL_K6] = a->k_hat6;
        m_floats[MF_ADAPTIVE_ROLL_K7] = a->k_hat7;
        m_floats[MF_ADAPTIVE_PHI_REFERENCE_MODEL] = a->phi_m;

#endif//ADAPTIVE_ROLL_B
}
```

A.2.5 MRAC C Pitch

```
/*** BeginHeader AdaptiveControl_PitchC */
void AdaptiveControl_PitchC(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_PitchC(AdaptivePitch* a, float theta_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
{
#ifdef ADAPTIVE_PITCH_C
        float theta_tilde, psi1, psi2, psi3;
        //for redefinitions so code is easier to read
        float V, theta, phi, p, q, r, dt;
        float alpha, lambda1, gamma1, gamma2, gamma3;
        float leakage_gain1, leakage_gain2, leakage_gain3, dz_threshold;

        char outside_threshold;
        float psi_norm_div;
   float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_PITCH_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_PITCH_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_PITCH_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_PITCH_GAMMA2];
        gamma3 = m_floats[MF_ADAPTIVE_PITCH_GAMMA3];
        leakage_gain1 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE2];
        leakage_gain3 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE3];
        //deadzone threshold; radians
        dz_threshold = m_floats[MF_ADAPTIVE_PITCH_DZ_THRESHOLD];

        //states
        V = *(a->airspeed);
        theta = *(a->theta);
        phi = *(a->phi);
        p = *(a->p);
        q = *(a->q);
        r = *(a->r);
        dt = m_floats[MF_DT];

        //reference model
```

```
      theta_tilde = theta − a−>theta_m;
      a−>theta_m += dt * alpha* (theta_d − a−>theta_m);

if (V == 0){
   psi1 = 0;}
else{
   psi1 = (lambda1*theta_tilde − alpha*(theta_d − a−>theta_m))/V;}
      psi2 = 1;
      psi3 = −theta;

      //Normalize
      if (normalize){
              psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3);
   if (psi_norm_div != 0){
            psi1 = psi1/psi_norm_div;
            psi2 = psi2/psi_norm_div;
            psi3 = psi3/psi_norm_div;
   }
      }

      //Deadzone
      outside_threshold = fabs(theta_tilde) > dz_threshold;
      if ((deadzone&&outside_threshold) || !deadzone){
              a−>k_hat1 += −dt * gamma1 * theta_tilde * V * psi1;
              a−>k_hat2 += −dt * gamma2 * theta_tilde * V * psi2;
              a−>k_hat3 += −dt * gamma3 * theta_tilde * V * psi3;

              //Leakage
              if (leakage){
                      a−>k_hat1 −= leakage_gain1*a−>k_hat1;
                      a−>k_hat2 −= leakage_gain2*a−>k_hat2;
                      a−>k_hat3 −= leakage_gain3*a−>k_hat3;
              }
      }

      effort = psi1*a−>k_hat1 + psi2*a−>k_hat2 + psi3*a−>k_hat3;

      if(invert_result)
              effort *= −1;

      if(sum_result)
              *(a−>elevator) += effort;
      else
              *(a−>elevator) = effort;
```

```
     m_floats[MF_ADAPTIVE_PITCH_K1] = a->k_hat1;
     m_floats[MF_ADAPTIVE_PITCH_K2] = a->k_hat2;
     m_floats[MF_ADAPTIVE_PITCH_K3] = a->k_hat3;
     m_floats[MF_ADAPTIVE_PITCH_K4] = a->k_hat4;
     m_floats[MF_ADAPTIVE_PITCH_K5] = a->k_hat5;
     m_floats[MF_ADAPTIVE_PITCH_K6] = a->k_hat6;
     m_floats[MF_ADAPTIVE_THETA_REFERENCE_MODEL] = a->theta_m;

#endif//ADAPTIVE_PITCH_C
}
```

A.2.6 MRAC C Roll

```
/*** BeginHeader AdaptiveControl_RollC */
void AdaptiveControl_RollC(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result);
/*** EndHeader */

void AdaptiveControl_RollC(AdaptiveRoll* a, float phi_d, char deadzone,
        char leakage, char normalize, unsigned char invert_result,
        unsigned char sum_result)
{
#ifdef ADAPTIVE_ROLL_C
        float phi_tilde, psi1, psi2, psi3;
        float V, theta, phi, groundtrack, p, q, r, dt;
        float alpha, lambda1, gamma1, gamma2, gamma3;
        float leakage_gain1, leakage_gain2, leakage_gain3, dz_threshold;

        char outside_threshold;
        float psi_norm_div;
   float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_ROLL_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_ROLL_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_ROLL_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_ROLL_GAMMA2];
        gamma3 = m_floats[MF_ADAPTIVE_ROLL_GAMMA3];
        leakage_gain1 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE2];
        leakage_gain3 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE3];
        //deadzone threshold; radians
```

```
      dz_threshold = m_floats[MF_ADAPTIVE_ROLL_DZ_THRESHOLD];

      //states
      V = *(a->airspeed);
      theta = *(a->theta);
      phi = *(a->phi);
      groundtrack = *(a->groundtrack);
      p = *(a->p);
      q = *(a->q);
      r = *(a->r);
      dt = m_floats[MF_DT];

      //reference model
      phi_tilde = phi - a->phi_m;
      a->phi_m += dt * alpha * (phi_d - a->phi_m);

if (V != 0){
      psi1 = (lambda1*phi_tilde - alpha*(phi_d - a->phi_m))/(2.0*V);}
else{
      psi1 = 0;}
      psi2 = 1.0;
      psi3 = groundtrack;

      //Normalize
      if (normalize){
              psi_norm_div = sqrt(psi1*psi1 + psi2*psi2 + psi3*psi3);
   if (psi_norm_div != 0){
                      psi1 = psi1/psi_norm_div;
                      psi2 = psi2/psi_norm_div;
                      psi3 = psi3/psi_norm_div;
   }
      }

//Deadzone
outside_threshold = fabs(phi_tilde) > dz_threshold;
if ((deadzone&&outside_threshold) || !deadzone){
      a->k_hat1 += dt * -2.0 * gamma1 * V * phi_tilde * psi1;
      a->k_hat2 += dt * -2.0 * gamma2 * V * phi_tilde * psi2;
      a->k_hat3 += dt * -2.0 * gamma3 * V * phi_tilde * psi3;
      //Leakage
      if (leakage){
 a->k_hat1 -= leakage_gain1*a->k_hat1;
           a->k_hat2 -= leakage_gain2*a->k_hat2;
           a->k_hat3 -= leakage_gain3*a->k_hat3;
      }
}

        effort = psi1*a->k_hat1 + psi2*a->k_hat2 + psi3*a->k_hat3;

        if(invert_result)
                effort *= -1;

    if(sum_result)
                *(a->aileron) += effort;
        else
                *(a->aileron) = effort;

        m_floats[MF_ADAPTIVE_ROLL_K1] = a->k_hat1;
        m_floats[MF_ADAPTIVE_ROLL_K2] = a->k_hat2;
        m_floats[MF_ADAPTIVE_ROLL_K3] = a->k_hat3;
        m_floats[MF_ADAPTIVE_ROLL_K4] = a->k_hat4;
        m_floats[MF_ADAPTIVE_ROLL_K5] = a->k_hat5;
        m_floats[MF_ADAPTIVE_ROLL_K6] = a->k_hat6;
        m_floats[MF_ADAPTIVE_ROLL_K7] = a->k_hat7;
        m_floats[MF_ADAPTIVE_PHI_REFERENCE_MODEL] = a->phi_m;

        #endif//ADAPTIVE_ROLL_C
        }
```

A.2.7 MRAC D Pitch

```
/*** BeginHeader AdaptivePitch_init */
void AdaptivePitch_init(AdaptivePitch* a, float32* command_elevator,
float32* actual_airspeed,float32* actual_theta, float32* actual_phi,
float32* actual_p, float32* actual_q,float32* actual_r);
/*** EndHeader */


void AdaptivePitch_init(AdaptivePitch* a, float32* command_elevator,
float32* actual_airspeed,float32* actual_theta, float32* actual_phi,
float32* actual_p, float32* actual_q,float32* actual_r)
{
#ifdef USE_ADAPTIVE_CONTROL
        a->theta_m = 0;
        a->k_hat1 = 0;
        a->k_hat2 = 0;

        a->airspeed = actual_airspeed;
        a->phi = actual_phi;
        a->theta = actual_theta;
        a->p = actual_p;
        a->q = actual_q;
        a->r = actual_r;

        a->elevator = command_elevator;
#endif
}

/*** BeginHeader AdaptiveControl_PitchD */
void AdaptiveControl_PitchD(AdaptivePitch* a, float32 theta_d,
uint8 deadzone, uint8 leakage, uint8 normalize, uint8 invert_result,
uint8 sum_result);
/*** EndHeader */


void AdaptiveControl_PitchD(AdaptivePitch* a, float32 theta_d,
uint8 deadzone, uint8 leakage, uint8 normalize, uint8 invert_result,
uint8 sum_result);
{
#ifdef USE_ADAPTIVE_CONTROL
        //for redefinitions so code is easier to read
        float32 theta_tilde, psi1, psi2;
        float32 V, theta, phi, p, q, r, dt;
        float32 alpha, lambda1, gamma1, gamma2;
        float32 leakage_gain1, leakage_gain2, dz_threshold;

        uint8 outside_threshold;
        float32 psi_norm_div;
    float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_PITCH_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_PITCH_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_PITCH_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_PITCH_GAMMA2];
        leakage_gain1 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_PITCH_LEAKAGE2];
        //deadzone threshold; radians
        dz_threshold = m_floats[MF_ADAPTIVE_PITCH_DZ_THRESHOLD];

        //states
        V = *(a->airspeed);
        theta = *(a->theta);
        phi = *(a->phi);
        p = *(a->p);
    q = *(a->q);
    r = *(a->r);
    dt = m_floats[MF_DT];

    //reference model
    theta_tilde = theta - a->theta_m;
    a->theta_m += dt * alpha* (theta_d - a->theta_m);

if (V == 0)
    psi1 = 0;
else
    psi1 = (lambda1*theta_tilde - alpha*(theta_d - a->theta_m))/V;

    psi2 = 1;

    //Normalize
    if (normalize){
          psi_norm_div = sqrt(psi1*psi1 + psi2*psi2);
  if (psi_norm_div != 0){
          psi1 = psi1/psi_norm_div;
          psi2 = psi2/psi_norm_div;
  }
    }

    //Deadzone
    outside_threshold = fabs(theta_tilde) > dz_threshold;
    if ((deadzone&&outside_threshold) || !deadzone){
          a->k_hat1 += -dt * gamma1 * theta_tilde * V * psi1;
          a->k_hat2 += -dt * gamma2 * theta_tilde * V * psi2;

          //Leakage
          if (leakage){
                a->k_hat1 -= leakage_gain1*a->k_hat1;
                a->k_hat2 -= leakage_gain2*a->k_hat2;
          }
    }

    effort = psi1*a->k_hat1 + psi2*a->k_hat2;

    if(invert_result)
          effort *= -1;

if(sum_result)
          *(a->elevator) += effort;
    else
```

```
                *(a–>elevator) = effort;

#endif
}
```

A.2.8 MRAC D Roll

```
/*** BeginHeader AdaptiveRoll_init */
void AdaptiveRoll_init(AdaptiveRoll* a, float32* command_aileron,
float32* actual_airspeed, float32* actual_theta, float32* actual_phi,
float32* actual_groundtrack, float32* actual_p, float32* actual_q,
float32* actual_r);
/*** EndHeader */

void AdaptiveRoll_init(AdaptiveRoll* a, float32* command_aileron,
float32* actual_airspeed, float32* actual_theta, float32* actual_phi,
float32* actual_groundtrack, float32* actual_p, float32* actual_q,
float32* actual_r)
{
#ifdef USE_ADAPTIVE_CONTROL
        a–>phi_m = 0;
        a–>k_hat1 = 0;
        a–>k_hat2 = 0;

        a–>airspeed = actual_airspeed;
        a–>phi = actual_phi;
        a–>theta = actual_theta;
        a–>groundtrack = actual_groundtrack;
        a–>p = actual_p;
        a–>q = actual_q;
        a–>r = actual_r;

        a–>aileron = command_aileron;
#endif
}

/*** BeginHeader AdaptiveControl_RollD */
void AdaptiveControl_RollD(AdaptiveRoll* a, float32 phi_d,
uint8 deadzone, uint8 leakage, uint8 normalize, uint8 invert_result,
uint8 sum_result);
/*** EndHeader */

void AdaptiveControl_RollD(AdaptiveRoll* a, float32 phi_d,
uint8 deadzone, uint8 leakage, uint8 normalize, uint8 invert_result,
uint8 sum_result);
```

```
{
#ifdef USE_ADAPTIVE_CONTROL
        float32 phi_tilde, psi1, psi2;
        float32 V, theta, phi, groundtrack, p, q, r, dt;
        float32 alpha, lambda1, gamma1, gamma2;
        float32 leakage_gain1, leakage_gain2, dz_threshold;

        uint8 outside_threshold;
        float32 psi_norm_div;
    float32 effort;

        //gains
        alpha = m_floats[MF_ADAPTIVE_ROLL_ALPHA];
        lambda1 = m_floats[MF_ADAPTIVE_ROLL_LAMBDA1];
        gamma1 = m_floats[MF_ADAPTIVE_ROLL_GAMMA1];
        gamma2 = m_floats[MF_ADAPTIVE_ROLL_GAMMA2];
        leakage_gain1 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE1];
        leakage_gain2 = m_floats[MF_ADAPTIVE_ROLL_LEAKAGE2];
        //deadzone threshold; radians
        dz_threshold = m_floats[MF_ADAPTIVE_ROLL_DZ_THRESHOLD];

        //states
        V = *(a->airspeed);
        theta = *(a->theta);
        phi = *(a->phi);
        groundtrack = *(a->groundtrack);
        p = *(a->p);
        q = *(a->q);
        r = *(a->r);
        dt = m_floats[MF_DT];

        //reference model
        phi_tilde = phi - a->phi_m;
        a->phi_m += dt * alpha * (phi_d - a->phi_m);

   if (V != 0)
        psi1 = (lambda1*phi_tilde - alpha*(phi_d - a->phi_m))/(2*V);
   else
        psi1 = 0;

        psi2 = 1;

        //Normalize
        if (normalize){
                psi_norm_div = sqrt(psi1*psi1 + psi2*psi2);
    if (psi_norm_div != 0){
                 psi1 = psi1/psi_norm_div;
                 psi2 = psi2/psi_norm_div;
    }
      }

      //Deadzone
      outside_threshold = fabs(phi_tilde) > dz_threshold;
      if ((deadzone&&outside_threshold) || !deadzone){
            a->k_hat1 += dt * -2.0 * gamma1 * V * phi_tilde * psi1;
            a->k_hat2 += dt * -2.0 * gamma2 * V * phi_tilde * psi2;
            //Leakage
            if (leakage){
                      a->k_hat1 -= leakage_gain1*a->k_hat1;
                      a->k_hat2 -= leakage_gain2*a->k_hat2;
              }
      }

      effort = psi1*a->k_hat1 + psi2*a->k_hat2;

      if(invert_result)
              effort *= -1;

  if(sum_result)
              *(a->aileron) += effort;
      else
              *(a->aileron) = effort;

#endif//ADAPTIVE_ROLL_D
}
```

Appendix B

$\mathcal{L}_1$ Code

B.1 Matlab Code

B.1.1 1st Order Model with One Unknown Parameter

```
function [sys,x0,str,ts] = L1_adapt_ctrl1(t,x,uu,flag)

% control constants
   a    = 4;
   Gam = 100;
   Kup = 15;
   Klow = -15;
   w0 = 1000;

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,uu,a,Gam,Kup,Klow,w0);

  % Update
  case 2,
    sys=mdlUpdate(t,x,uu);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,uu,a,Gam,Kup,Klow);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,uu);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,uu);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl
function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs   = 4;
sizes.NumInputs      = 1+1+12;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [0; -.0001; 0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts   = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,a,Gam,Kup,Klow,w0)

% interpret states and input
    theta_m     = x(1);
    K1hat       = x(2);
    x_lpf        = x(3);

    new_control = uu(1);
    theta_c      = uu(2);
    z            = [uu(3); uu(4)];
    h            = uu(5);
    u            = uu(6);
    v            = uu(7);
    w            = uu(8);
    phi          = uu(9);
    theta        = uu(10);
    psi          = uu(11);
    p            = uu(12);
    q            = uu(13);
    r            = uu(14);
    gam      = 0;

    V        = sqrt(u^2+v^2+w^2);

% define constants
    K2 = 0.4206*1.1;

% companion model
    theta_m_dot = a*(theta_c - theta_m) + a*(theta-theta_m) - ...
                  (V*cos(phi)*K1hat - V*cos(phi)*new_control*K2) ;


% parameter update
    K1hatdot = -Gam*(theta-theta_m)*V*cos(phi);

% projection
delta = 1.5;
f = 2/delta*(((K1hat-(Kup+Klow)/2)/((Kup-Klow)/2))^2 + 1 - delta);
f_dot = 4/delta*(K1hat-(Kup+Klow)/2)/((Kup-Klow)/2);

if f >= 0,
    if f_dot*K1hatdot >= 0,
        K1hatdot = K1hatdot - f*K1hatdot;
    end
end

% update low pass filter
  %x_lpf_dot = w0*(V*cos(phi)*K1hat - x_lpf);
    x_lpf_dot = w0*(K1hat - x_lpf);
```

```
sys = [theta_m_dot; K1hatdot; x_lpf_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,uu)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,a,Gam,Kup,Klow)

% interpret states and input
    theta_m     = x(1);
    K1hat       = x(2);
    x_lpf       = x(3);
    new_control = uu(1);
    theta_c     = uu(2);
    z           = [uu(3); uu(4)];
    h           = uu(5);
    u           = uu(6);
    v           = uu(7);
    w           = uu(8);
    phi         = uu(9);
    theta       = uu(10);
    psi         = uu(11);
    p           = uu(12);
    q           = uu(13);
    r           = uu(14);
    gam     = 0;

    V       = sqrt(u^2+v^2+w^2);

% define constants
    K2 = 0.4206*1.1;

% control output

    delta_e = a*(theta_c-theta)/V/cos(phi)/K2...
              + a*(theta_c-theta_m)/V/cos(phi)/K2;


sys = [K1hat; theta_m; delta_e; K1hat/K2];
```

```
% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,uu)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,uu)

sys = [];

% end mdlTerminate
```

B.1.2 1st Order Model with Two Unknown Parameter

```
function [sys,x0,str,ts] = L1_adapt_ctrl1(t,x,uu,flag)

% control constants
   a    = 4; %.8
   Gam = .100;%.031; %100 %.51
   Kup = [1 1];
   Klow = [−1 −1];

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,uu,a,Gam,Kup,Klow);

  % Update
  case 2,
    sys=mdlUpdate(t,x,uu);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,uu,a,Gam,Kup,Klow);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,uu);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,uu);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs    = 5;
sizes.NumInputs     = 1+1+12;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [0; -.0001; 0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,a,Gam,Kup,Klow)
```

```
% interpret states and input
    theta_m     = x(1);
    Khat        = [x(2); x(3)];

    new_control = uu(1);
    theta_c     = uu(2);
    z           = [uu(3); uu(4)];
    h           = uu(5);
    u           = uu(6);
    v           = uu(7);
    w           = uu(8);
    phi         = uu(9);
    theta       = uu(10);
    psi         = uu(11);
    p           = uu(12);
    q           = uu(13);
    r           = uu(14);
    gam     = 0;

    V       = sqrt(u^2+v^2+w^2);

% define constants
    K3 = 0.4206;

% Regressor
    Psi = [1; theta];

% companion model
    theta_m_dot = -a*theta_m + V*cos(phi)*K3*(new_control + Khat'*Psi/K3...
                  - (-a*theta_m - a*(theta-theta_m))/V/cos(phi)/K3) ;


% parameter update
    Khatdot = Gam*(theta-theta_m)*V*cos(phi)*Psi;

delta = 1.5;
f(1) = 2/delta*(Khat(1)^2 + 1 - delta);
f(2) = 2/delta*(Khat(2)^2 + 1 - delta);
f_dot = 4/delta*Khat;

if f(1) >= 0,
    if f_dot(1)*Khatdot(1) >= 0,
        Khatdot(1) = Khatdot(1) - f(1)*Khatdot(1);
    end
```

```
end
if f(2) >= 0,
    if f_dot(2)*Khatdot(2) >= 0,
        Khatdot(2) = Khatdot(2) - f(2)*Khatdot(2);
    end
end

% update low pass filter
  %x_lpf_dot = w0*(V*cos(phi)*K1hat - x_lpf);
  % x_lpf_dot = w0*(K1hat - x_lpf);


sys = [theta_m_dot; Khatdot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,uu)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,a,Gam,Kup,Klow)

% interpret states and input
    theta_m     = x(1);
    Khat       = [x(2); x(3)];

    new_control = uu(1);
    theta_c     = uu(2);
    z           = [uu(3); uu(4)];
    h           = uu(5);
    u           = uu(6);
    v           = uu(7);
    w           = uu(8);
    phi         = uu(9);
    theta       = uu(10);
    psi         = uu(11);
    p           = uu(12);
    q           = uu(13);
    r           = uu(14);
    gam     = 0;

    V       = sqrt(u^2+v^2+w^2);
```

```
% define constants
    K3 = 0.4206;

% Regressor
    Psi = [1; theta];

% control output
    delta_e = (-a*(theta_m-theta_c) - a*(theta-theta_m))/V/cos(phi);


sys = [Khat; theta_m; delta_e; -Khat'*Psi/K3];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,uu)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,uu)

sys = [];

% end mdlTerminate
```

B.1.3 2nd Order Model

```
function [sys,x0,str,ts] = L1_adapt_ctrl(t,x,uu,flag)

% control constants
   zeta = 1.707;
   wn  = 6;
   tau  = .01;
   lambda   = [100; 50];
   Gam = .100;%1000*[.1; .1];
   Kup = [15; 15];
   Klow = [-15; -15];

switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,uu,zeta,wn,tau,lambda,Gam,Kup,Klow);

  % Update
  case 2,
    sys=mdlUpdate(t,x,uu);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,uu,zeta,wn,tau,lambda,Gam,Kup,Klow);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,uu);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,uu);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 2+2;
sizes.NumDiscStates = 0;
sizes.NumOutputs    = 2+1+1+1;
sizes.NumInputs     = 14;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
```

```
x0  = [0; 0; -.0001; 0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,zeta,wn,tau,lambda,Gam,Kup,Klow)

% interpret states and input
    theta_m     = x(1);
    theta_m_dot = x(2);
    Khat        = [x(3); x(4)];

    delta_e_lpf = uu(1);
    theta_c     = uu(2);
    z           = [uu(3); uu(4)];
    h           = uu(5);
    u           = uu(6);
    v           = uu(7);
    w           = uu(8);
    phi         = uu(9);
    theta       = uu(10);
    psi         = uu(11);
    p           = uu(12);
    q           = uu(13);
    r           = uu(14);
    gam         = 0;

% define constants
    K2 = 0.4206;

 % compute aerodynamic variables
    V      = sqrt(u^2+v^2+w^2);
    alpha  = atan2(w,u);
    beta   = atan2(v,sqrt(u^2+w^2));

    theta_dot = q*cos(phi)-r*sin(phi);
```

```
% Error model
    theta_tilde     = theta − theta_m;
    theta_tilde_dot = theta_dot − theta_m_dot;

%Regressor
    Psi = [theta_dot/V; 1];

    R = theta_tilde_dot + lambda(1)*theta_tilde;

% companion model
    theta_m_ddot = −2*zeta*wn*theta_m_dot − wn^2*theta_m...
                 + V^2*K2*(...
                     delta_e_lpf...
                     + 1/K2/V^2*Khat'*Psi...
                     − 1/V^2/K2*(...
                        − lambda(1)*(theta_tilde_dot)...
                        − 2*zeta*wn*theta_m_dot − wn^2*theta_m...
                        − lambda(2)*R...
                        )...
                     );

% parameter update
  Khatdot = R*V^2*Psi;

  delta = 1.5;
  f(1) = 2/delta*(Khat(1)^2 + 1 − delta);
  f(2) = 2/delta*(Khat(2)^2 + 1 − delta);
  f_dot = 4/delta*Khat;

if f(1) >= 0,
    if f_dot(1)*Khatdot(1) >= 0,
        Khatdot(1) = Khatdot(1) − f(1)*Khatdot(1);
    end
end
if f(2) >= 0,
    if f_dot(2)*Khatdot(2) >= 0,
        Khatdot(2) = Khatdot(2) − f(2)*Khatdot(2);
    end
end

sys = [theta_m_dot; theta_m_ddot; Khatdot];

% end mdlDerivatives
```

```
function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,zeta,wn,tau,lambda,Gam,Kup,Klow)

% interpret states and input
    theta_m     = x(1);
    theta_m_dot = x(2);
    Khat        = [x(3); x(4)];

    delta_e_lpf = uu(1);
    theta_c     = uu(2);
    z           = [uu(3); uu(4)];
    h           = uu(5);
    u           = uu(6);
    v           = uu(7);
    w           = uu(8);
    phi         = uu(9);
    theta       = uu(10);
    psi         = uu(11);
    p           = uu(12);
    q           = uu(13);
    r           = uu(14);
    gam         = 0;

% Define constants
    K2 = 0.4206;

% Compute aerodynamic variables
    V     = sqrt(u^2+v^2+w^2);
    alpha = atan2(w,u);
    beta  = atan2(v,sqrt(u^2+w^2));

% Aircraft theta_dot
    theta_dot = q*cos(phi)-r*sin(phi);

% Error model
    theta_tilde     = theta - theta_m;
    theta_tilde_dot = theta_dot - theta_m_dot;

% Regressor
    Psi = [theta_dot/V; 1];
    R = theta_tilde_dot + lambda(1)*theta_tilde;

% Control input
    delta_e = (...
                 - lambda(1)*(theta_tilde_dot)...
                 - 2*zeta*wn*theta_m_dot - wn^2*(theta_m-theta_c)...
                 - lambda(2)*R...
                )/V^2/K2;

sys = [Khat; theta_m; delta_e;-1/K2/V^2*Khat'*Psi];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

B.1.4 Physically Motivated Model

```
 function [sys,x0,str,ts] = L1_adapt_ctrl1(t,x,uu,flag)

% control constants
   Gam      = 1*[.10,   0,    0,    0,    0;
                   0, .10,    0,    0,    0;
                   0,   0,  .10,    0,    0;
                   0,   0,    0,  .10,    0;
                   0,   0,    0,    0,  .10];
   Kup      = [ 15, 15, 15, 15, 15];
   Klow     = [-15,-15,-15,-15,-15];
   epsilon1 = 10000;
   epsilon2 = epsilon1;
   k_theta  = 5;
   k_q      = 20;
   gamma1   = 1;
   gamma2   = 1;
```

```
switch flag,

  % Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Derivatives
  case 1,
    sys=mdlDerivatives(t,x,uu,epsilon1,epsilon2,k_theta,k_q,gamma1,...
                          gamma2,Gam,Kup,Klow);

  % Update
  case 2,
    sys=mdlUpdate(t,x,uu);

  % Outputs
  case 3,
    sys=mdlOutputs(t,x,uu,epsilon1,epsilon2,k_theta,k_q,gamma1,...
                    gamma2,Gam,Kup,Klow);

  % GetTimeOfNextVarHit
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,uu);

  % Terminate
  case 9,
    sys=mdlTerminate(t,x,uu);

  % Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 10;
sizes.NumDiscStates = 0;
sizes.NumOutputs    = 5+1+1+1;
sizes.NumInputs     = 1+1+12;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% end mdlInitializeSizes

function sys=mdlDerivatives(t,x,uu,epsilon1,epsilon2,k_theta,k_q,gamma1,...
                   gamma2,Gam,Kup,Klow)

% interpret states and input
   theta_m     = x(1);
   q_m         = x(2);
   Khat        = [ x(3);  x(4);   x(5);   x(6);   x(7)];
   q_des_dot   = x(8);
   q_m_d_dot    = x(9);
   theta_c_dot = x(10);

   new_control = uu(1);
   theta_c     = uu(2);
   z           = [uu(3); uu(4)];
   h           = uu(5);
   u           = uu(6);
   v           = uu(7);
   w           = uu(8);
   phi         = uu(9);
   theta       = uu(10);
   psi         = uu(11);
   p           = uu(12);
   q           = uu(13);
```

```
    r           = uu(14);
    gam     = 0;

    V       = sqrt(u^2+v^2+w^2);
    alpha   = atan2(w,u);
    beta    = atan2(v,sqrt(u^2+w^2));


   % projection

   for i = 1:length(Khat);
   if Khat(i) > Kup(i),
         Khat(i) = Kup(i);
   elseif Khat(i) < Klow(i),
         Khat(i) = Klow(i);
   end
end

% define constants
    K6 = 0.4206;

% Dirty derivative of theta_c
    theta_c_dot = 1/epsilon1*(−theta_c_dot + theta_c);

% Define regressor vector
   Psi = [p*r/V^2;−(p^2−r^2)/V^2;1;alpha;q/V];
   % Psi = [0/V^2;1/V^2;1;0;1/V];

% Calculate q desired for model
    q_m_d = −k_theta*(theta_m−theta_c);
    q_m_d_dot = 1/epsilon1*(−q_m_d_dot + q_m_d);

% Error from model
    theta_e = theta − theta_m;
    q_e     = q − q_m;

% Backstepping variable
    q_des     = (r*sin(phi) + theta_c_dot − gamma1*theta_e)/cos(phi);
    q_des_dot = 1/epsilon1*(−q_des_dot + q_des);

% Error from backstepping variable
    e = q_e − q_des;

% companion model
    theta_m_dot = q_m*cos(phi) + theta_c_dot;
    q_m_dot    = -k_q*q_m ...
                 + V^2*K6*(...
                       new_control...
                       + 1/K6*Psi'*Khat...
                       - 1/(V^2*K6)*(...
                             + k_q*q_m...
                             - theta_e*cos(phi)...
                             - q_des_dot...
                             + gamma2*e...
                         )...
                   );

% parameter update
    Khatdot = (q_e-q_des)*V^2*Gam*Psi;

% projection
for i = 1:length(Khat);
   if Khat(i) >= Kup(i),
      if Khatdot(i) > 0,
        Khatdot(i) = 0;
        Khat(i) = Kup(i);
      end
   elseif Khat(i) <= Klow(i),
      if Khatdot(i) < 0,
        Khatdot(i) = 0;
        Khat(i) = Klow(i);
      end
   end
end

sys = [theta_m_dot; q_m_dot; Khatdot; q_des_dot; q_m_d_dot; theta_c_dot];

% end mdlDerivatives

function sys=mdlUpdate(t,x,uu)

sys = [];

% end mdlUpdate

function sys=mdlOutputs(t,x,uu,epsilon1,epsilon2,k_theta,k_q,gamma1,...
                        gamma2,Gam,Kup,Klow)

% interpret states and input
    theta_m    = x(1);
    q_m          = x(2);
    Khat         = [ x(3);  x(4);   x(5);   x(6);   x(7)];
    q_des_dot   = x(8);
    q_m_d_dot    = x(9);
    theta_c_dot = x(10);

    new_control = uu(1);
    theta_c      = uu(2);
    z             = [uu(3); uu(4)];
    h             = uu(5);
    u             = uu(6);
    v             = uu(7);
    w             = uu(8);
    phi           = uu(9);
    theta         = uu(10);
    psi           = uu(11);
    p             = uu(12);
    q             = uu(13);
    r             = uu(14);

    V        = sqrt(u^2+v^2+w^2);
    alpha   = atan2(w,u);
    beta    = atan2(v,sqrt(u^2+w^2));

% define constants
    K6 = 0.4206;

% Dirty derivative of theta_c
    theta_c_dot = 1/epsilon1*(−theta_c_dot + theta_c);

% Define regressor vector
   Psi = [q*r/V^2;−(p^2−r^2)/V^2;1;alpha;q/V];
    %Psi = [p*r/V^2;-(p^2-r^2)/V^2;1;alpha;q/V];

% Calculate q desired for model
    q_m_d = −k_theta*(theta_m−theta_c);
    q_m_d_dot = 1/epsilon1*(−q_m_d_dot + q_m_d);

% Error from model
    theta_e = theta − theta_m;
    q_e      = q − q_m;

% Backstepping variable
    q_des      = (r*sin(phi) + theta_c_dot − gamma1*theta_e)/cos(phi);
    q_des_dot = 1/epsilon1*(−q_des_dot + q_des);
```

```
% Error from backstepping variable
    e = q_e − q_des;

% control output
    delta_e = −1/K6*Psi'*Khat*0 . . .
              −1/(K6*V^2)*(. . .
                       theta_e*cos(phi). . .
                       + k_q*(q_m−q_m_d). . .
                       − q_m_d_dot. . .
                       − 1/k_theta*q_m_d*cos(phi). . .
                       − q_des_dot. . .
                       + gamma2*e. . .
               );


sys = [Khat; theta_m; delta_e; −1/K6*Psi'*Khat];

% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,uu)

sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,uu)

sys = [];

% end mdlTerminate
```

B.2 Dynamic C Code

```
/*** BeginHeader L1_Pitch */
void L1_Pitch(float32 theta_d, float32 V, float32* effort_e,
uint8 invert_result);
/*** EndHeader */

void L1_Pitch(float32 theta_d, float32 V, float32* effort_e,
uint8 invert_result);
{
#ifdef USE_L1_ADAPTIVE_CONTROL
```

```
    uint8 i;
    float32 alpha, alpha_filt, gamma, Wo, K1_u, K1_l, N, dt,
                theta, phi, K2, K1_hat, K1_hat_lowpass, theta_cm;
static float32 delta_e;

    #GLOBAL_INIT
    {
            delta_e = 0;
    }

    // Gains
    alpha = m_floats[MF_L1_ALPHA];
    gamma = m_floats[MF_L1_GAMMA];
    Wo = m_floats[MF_L1_CUTOFF];
    K1_u = m_floats[MF_L1_K1_UPPER_LIMIT];
    K1_l = m_floats[MF_L1_K1_LOWER_LIMIT];
    N = m_floats[MF_L1_NUM_OF_ITERATIONS];

    // Variables
    dt = m_floats[MF_DT];
    theta = m_floats[MF_THETA];
    phi = m_floats[MF_PHI];
    K2 = m_floats[MF_L1_K2];
    K1_hat = m_floats[MF_L1_K1_HAT];
    K1_hat_lowpass = m_floats[MF_L1_K1_HAT_LOWPASS];
    theta_cm = m_floats[MF_L1_THETA_CM];

    // Flag
    if(m_ints[MI_L1_INIT] == 1)
    {
            theta_cm = theta;
            m_ints[MI_L1_INIT] = 0;
    }

    // Companion Model
    theta_cm = theta_cm + dt*(-alpha*theta_cm + alpha*theta -
            V*cos(phi)*K1_hat + V*cos(phi)*K2*delta_e);

    // Parameter Update
    K1_hat = K1_hat + dt*(-gamma*(theta - theta_cm)*V*cos(phi));

    // Projection of K1
    if(K1_hat >= K1_u)  {  K1_hat = K1_u; }
    if(K1_hat <= K1_l)  {  K1_hat = K1_l; }
```

```
        // Filter
        alpha_filt = dt*Wo/(1 + dt*Wo);
        K1_hat_lowpass = (1 − alpha_filt)*K1_hat_lowpass +
                alpha_filt*(K1_hat − K1_hat_lowpass);


        // Control
        if(V*cos(phi)*K2 > 0.0001 || V*cos(phi)*K2 < −0.0001)
                delta_e = K1_hat_lowpass/(V*cos(phi)*K2) +
                        alpha*(theta_d − theta)/(V*cos(phi)*K2);

        *effort_e = delta_e;

        //invert if needed
        if(invert_result)
                *effort_e *= −1;

        // Update Misc floats
        m_floats[MF_L1_THETA_CM] = theta_cm;
        m_floats[MF_L1_K1_HAT] = K1_hat;
        m_floats[MF_L1_K1_HAT_LOWPASS] = K1_hat_lowpass;
    m_floats[MF_L1_DELTA_E] = delta_e;

#endif
}
```

Glossary

C_*	The aircraft's aerodynamic coefficients.	6
J_{ij}	The inertial moments of the aircraft about i and j (e.g. x and y) axes.	6
S	Surface area of the aircraft's wings	6
V	Commanded airspeed in m/s.	6
$\hat{J}_*$	Constants that depend on J.	6
α	Known as the angle of attack, it is angle between the wing cord and the direction of motion in the air mass.	6
$\bar{c}$	Length of a wing's cord length.	6
β	Known as the sideslip angle, it is the angle between the direction of motion in the air mass and heading.	6
χ	The direction the aircraft travels in reference to the ground.	6
δ_a	Commanded aileron position in radians.	6
δ_e	Commanded elevator position in radians.	6
δ_r	Commanded rudder position in radians.	6
γ	Angle from level the aircraft is climbing or descending.	6
$\mathcal{L}_1$ Controller	A modified Lyapunov based MRAC controller where the parameter estimates are low-pass filtered.	30
ϕ	Roll angle.	6
ϕ^d	Desired roll angle.	6
ψ	Heading.	6
θ	Pitch angle.	6
θ^d	Desired pitch angle.	6
b	Length between wing tips.	6
h	The aircraft's georeferenced altitude.	7
p	Rotation about the body frames x axis.	6
p_e	The aircraft's georeferenced east position.	7
p_n	The aircraft's georeferenced north position.	7
q	Rotation about the body frames y axis.	6
r	Rotation about the body frames z axis.	6
u	The aircraft's velocity out the nose.	7
v	The aircraft's velocity out the right wing.	7
w	The aircraft's velocity out the belly.	8

Adaptive control	A type of control theory where the controller adapts to unknown or changing system parameters.	2
Aviones	An aircraft simulator.	83
body frame	A coordinate system fixed to the aircraft such that the x axis is out the nose, the y axis is out the right wing, and the z axis is out the belly of the aircraft.	6
Dynamic C	A C based language used to program Rabbit microcontrollers.	83
Dynamic inversion	A technique that inverts the dynamics of the plant to allow for the design of a simple controller. DI can be used to make an adaptive controller when used in conjunction with neural networks or least squares estimation.	22
Ground station	The equipment and software bundle needed to communicate with a MAV in flight.	84
Least squares	Uses to perform an online plant identification. In adaptive control, the system identification is used to adjust the controller.	19
Lyapunov MRACs	MRACs that use Lyapunov stability theory to update the parameter estimates.	27
Micro air vehicles	Light-weight UAs with small wing spans.	1
MIT Rule	The adaptive portion of the controller changes the parameter estimates based on the gradient of a cost function.	26
MRAC	Model reference adaptive control (MRAC) is a type of adaptive controller that uses a model of the plant to aid in adaption.	25
PID	A control scheme where the control effort is calculated summing the weighted error, integral of the error, and derivative of the error.	2
Projection operator	A function whose inputs are projected into the subspace S.	5
Unmanned aircraft	Autonomous aircraft that are unmanned.	1
Virtual Cockpit	Ground station software used to control MAVs.	84