

# **Smart Control of 2-Degree of Freedom Helicopters**

by

Glenn Janiak and Ken Vonckx

Advisor: Dr. Suruz Miah

Electrical and Computer Engineering Department  
Caterpillar College of Engineering and Technology  
Bradley University

© Glenn Janiak and Ken Vonckx, Peoria, Illinois, 2019

## **Abstract**

This project proposes a modular and cost-effective smart real-time motion control framework for a group of two degrees of freedom (2-DOF) helicopters. The helicopter are controlled by a mobile device which sends position signals over a wireless network to a microcontroller. The microcontroller uses control algorithms to determine the amount of voltage to apply to DC motors. This project tests and compares three control algorithms, LQR, LQG, and ADP as well as theorizes possible methods to improve the control in future projects.

## **Acknowledgements**

Special Thanks to Andrew Fandel, Anthony Birge, and Dr. Suruz Miah for their work with Machine Learning on a 2-DOF Helicopter.

Thanks to Mr. Christopher Mattus for his assistance in setting up our lab environment.

Thank you to everyone else who helped make this project possible and sucessful.

## **Dedication**

This is dedicated to our loved ones and friends who supported us through our time working on this project.

# Table of Contents

<b>List of Tables</b>	viii
<b>List of Figures</b>	ix
<b>Nomenclature</b>	1
<b>1 Introduction</b>	2
1.1 Problem Statement . . . . .	2
1.2 Literature Review . . . . .	2
1.3 Report Organization . . . . .	3
<b>2 Modeling 2-DOF Helicopters</b>	4
<b>3 Control Algorithms</b>	6
3.1 LQR . . . . .	6
3.2 LQG . . . . .	6
3.3 ADP . . . . .	7
3.4 Improvements . . . . .	8
<b>4 Numerical Simulations</b>	9
4.1 LQR (P controller) . . . . .	9
4.2 LQR (PI controller) . . . . .	9
4.3 LQG (PI Controller) . . . . .	9
<b>5 Implementation</b>	15
5.1 Experimental Setup . . . . .	15
5.1.1 USB . . . . .	15

5.1.2	Raspberry Pi . . . . .	15
5.1.3	Mobile Device . . . . .	15
5.2	USB . . . . .	19
5.2.1	LQR . . . . .	19
5.2.2	ADP . . . . .	20
5.2.3	Root Mean Square Error . . . . .	25
5.3	Raspberry Pi . . . . .	28
5.3.1	LQR (P Type Controller) . . . . .	28
5.3.2	ADP . . . . .	28
5.4	Mobile Device . . . . .	29
5.4.1	LQR (P Type Controller) . . . . .	29
5.4.2	ADP . . . . .	30
<b>6</b>	<b>Conclusion and Future Work</b>	<b>34</b>
6.1	Conclusion . . . . .	34
6.2	Future Work . . . . .	35
6.2.1	Enhanced Smart Framework . . . . .	35
6.2.2	Digital Compass . . . . .	36
6.2.3	Expanded PI Control . . . . .	36
<b>APPENDICES</b>		<b>36</b>
<b>A</b>	<b>Parameters and State-Space</b>	<b>37</b>
A.1	Parameters . . . . .	37
A.2	State-Space Model . . . . .	38
<b>B</b>	<b>MATLAB Simulation Code</b>	<b>40</b>
B.1	LQR (P controller) . . . . .	40
<b>C</b>	<b>Raspberry Pi MATLAB Code</b>	<b>43</b>
C.1	Initialization Code . . . . .	43
C.2	SPI Protocol . . . . .	43
<b>D</b>	<b>ADP MATLAB Code</b>	<b>63</b>
D.1	Initialization Code . . . . .	63
D.2	Update Weights . . . . .	66

<b>E Tutorials</b>	<b>71</b>
E.1 USB Connection . . . . .	71
E.2 Raspberry Pi Implementation . . . . .	71
E.3 Android Application . . . . .	73
<b>References</b>	<b>74</b>

# List of Tables

5.1	Root mean squared error and improvement for LQR P and PI controller .	26
5.2	Root mean squared error and improvement for LQR P and ADP P controller	26

# List of Figures

2.1	The Quanser Aero 2-DOF used for experiments in the project. . . . .	4
2.2	Freebody diagram of the forces exerted on a 2-DOF helicopter. . . . .	5
3.1	Input nodes are connected to the hidden layer and then to the output layer. . . . .	7
3.2	Data is collected for every $\tau$ seconds and then the weights are adjusted every $T$ seconds. . . . .	7
4.1	Simulations for proportional gain calculated by LQR. . . . .	10
4.2	Block diagram for LQR simulation using simulink. . . . .	11
4.3	Simulations for proportional and integral gain calculated by LQR. . . . .	12
4.4	Block diagram for LQG simulation using simulink. . . . .	13
4.5	Simulations for proportional and integral gain calculated by LQG. . . . .	14
5.1	Block diagram of SPI communication protocol used for communication between the Raspberry Pi and Quanser Aero. . . . .	16
5.2	Experiment setup for controlling the Quanser Aeros via a mobile device. . . . .	16
5.3	Illustration of the TCP model which describes how packets are sent and received from mobile devices to the Quanser Aero. . . . .	17
5.4	Simulink model of the android interface using two Quanser Aeros. . . . .	17
5.5	Screenshot of the mobile user interface. . . . .	18
5.6	Block diagram for LQR P type controller with a USB connection. . . . .	19
5.7	Block diagram for LQR PI type controller with a USB connection. . . . .	20
5.8	USB implementation for proportional controller and proportional-integral controller calculated by LQR with a step input. . . . .	21
5.9	USB implementation for proportional controller and proportional-integral controller calculated by LQR with a square wave input. . . . .	22
5.10	USB implementation for proportional controller and proportional-integral controller calculated by LQR with a sinusoidal input. . . . .	23

5.11	Block diagram for ADP P type controller with a USB connection. . . . .	24
5.12	Block diagram for ADP P type controller algorithm. . . . .	24
5.13	USB implementation for ADP proportional controller with a step input. . . . .	25
5.14	USB implementation for ADP proportional controller with a square wave input. . . . .	26
5.15	USB implementation for ADP proportional controller with a sinusoidal input. . . . .	27
5.16	Simulink model of LQR P type using raspberry pi communication. . . . .	28
5.17	Simulink model of ADP using raspberry pi communication. . . . .	29
5.18	Simulink model of LQR using mobile device communication. . . . .	30
5.19	Implementation results for LQR P type controller using mobile device as user input. . . . .	31
5.20	Simulink model of ADP using mobile device communication. . . . .	32
5.21	Implementation results for ADP controller using mobile device as user input. . . . .	33
6.1	Smart algorithm server used for connectivity. . . . .	35
A.1	System parameters used for the Quanser Aero. . . . .	37
E.1	Quanser Aero with QFLEX 2 USB panel installed . . . . .	72
E.2	Quanser Aero with QFLEX 2 Embedded panel installed . . . . .	73

---

## Abbreviations

**2-DOF** - 2 Degrees-Of-Freedom

**ADP** - Approximate Dynamic Programming

**LQG** - Linear Quadratic Gaussian

**LQR** - Linear Quadratic Regulator

**RMSE** - Root Mean Square Error

**SPI** - Serial Peripheral Interface

**TCP** - Transmission Control Protocol

**UDP** - User Datagram Protocol

**USB** - Universal Serial Bus

## Mathematical Symbols

$D_p$  - viscous damping of the pitch axis

$D_y$  - viscous damping of the yaw axis

$J_p$  - moment of inertia about pitch axis

$J_y$  - moment of inertia about yaw axis

$K_{sp}$  - stiffness of the axes

$K_{pp}$  - pitch motor thrust constant

$K_{py}$  - thrust constant acting on the pitch angle from the yaw motor

$K_{yp}$  - thrust constant acting on yaw angle from pitch motor

$K_{yy}$  - yaw motor thrust constant

# Chapter 1

## Introduction

### 1.1 Problem Statement

Helicopters are of a paramount importance as they are used in many civilian and military applications due to their ability for vertical take-off and landing. To enable their use in such applications, intensive research has been conducted to date since helicopters involve complex nonlinear dynamics. Most of the work on helicopter based research requires dedicated computers for controlling their motion to specific configurations and resistant to turbulent conditions. Such methods are expensive and time-consuming to develop. Implementation of motion control techniques using cost-effective hardware is still a challenge.

In this project, we are proposing an algorithm for smart control of a team of two degree-of freedom (two-DOF) helicopters using conventional motion control in cooperation with machine learning techniques where a user will be able to configure helicopters from any initial position. Even though conventional techniques have been tested with simple platforms in the literature, the current project employs conventional motion control strategies in cooperation with machine learning technique (reinforcement learning, for instance) for a team of helicopters as well as introducing user control via mobile devices. This project is expected to encourage research in this area as well as serve as an educational tool in teaching environments.

### 1.2 Literature Review

Our project requires a great deal of research as some of our tasks have not been attempted before. As a result, we have examined research papers, work complete by other projects at Bradley University, and documentation/teaching materials from Quanser Inc.

Among the major challenges in developing unmanned systems is to implement a modular, cost-effective, and robust teleoperation system, where the motion of a group of helicopters is controlled by mobile devices. In some cases, computer simulations are conducted to reveal the performance control structures of a 2-DOF helicopter [7]. A large body of

research has been conducted in the literature to focus on developing different control structures, such as linear-quadratic regulators (LQR)/Gaussian (LQG), sliding-mode controls (SMC), and advance nonlinear controls, that are specifically applied to 2-DOF helicopters. See [8, 1], for example, and some references therein. Furthermore, soft-computing tools, such as fuzzy-logic, neural networks, and a few combinations of them are employed for controlling the motion of a 2-DOF helicopter [2, 6, 4, 5].

The documentation of Quanser AERO<sup>1</sup> employs LQR and LQG motion control techniques for teaching purposes. This involves creating a linearized system model to calculate the LQR state-feedback gain. A model reference adaptive control (MARC) scheme using Lyapunov functions has been used by [8] for adaptive motion control of a 2-DOF helicopter. SMC is a nonlinear control technique to drive the system states onto a surface in the state space. This method has been used by [1]. Fuzzy-logic controllers use an inference engine to produce an output as used in [2, 6]. The ADP technique does not rely on knowledge of the system model. Instead, it uses data to reconstruct the states as preformed by [4]. Authors in [5] used high order neural networks (HONN) to approximate non-linearities in the system model.

As can be noticed, most of the motion control techniques are either tested using computer simulations or use dedicated computational platforms, that may not be modular and/or cost-effective, for developing motion control algorithms. This is due to the fact that the aforementioned techniques are mainly to propose novel motion control techniques and not focused on hardware implementation platforms. Therefore, the current work considers implementing a conventional motion control technique using modular and cost-effective hardware platforms in the context of a teleoperation system, where a human operator has the ability to control the motion of a team of helicopters using a smart mobile device.

## 1.3 Report Organization

This report is organised into 6 chapters. Chapter 1 discusses what this project hopes to accomplish and what similar projects have completed. Chapter 2 explains how to create a mathematical model for the system. Chapter 3 provides a brief explanation of the control algorithms and architectures used. Chapter 4 contains the results from our simulations. Chapter 5 discusses results from USB, Raspberry Pi, and Mobile Device implementation. Chapter 6 concludes the paper and provides insight into future directions.

---

<sup>1</sup>See <https://www.quanser.com/> for details

# Chapter 2

## Modeling 2-DOF Helicopters

The 2-DOF helicopter, Quanser Aero [manufactured by Quanser Inc. (<https://www.quanser.com/>)], used in the current work can be configured as a dual-rotor helicopter that has a fixed base. The front rotor (horizontal to the ground) is configured to rotate about pitch axis and the

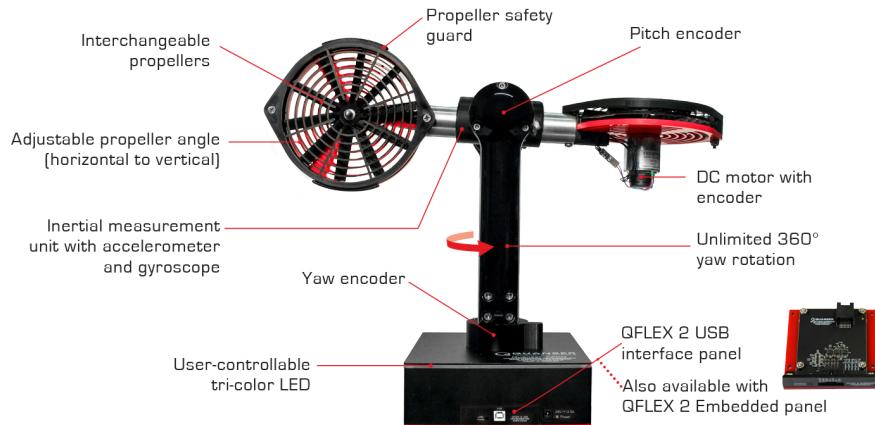


Figure 2.1: The Quanser Aero 2-DOF used for experiments in the project.

tail rotor (parallel to vertical plane) is mounted to rotate about yaw axis. To measure the pitch ( $\theta$ ) and yaw ( $\psi$ ) angles, two position sensors (encoders) are mounted as shown in figure 2.1. For sake of simplicity in modeling 2-DOF helicopter, the dynamic coupling of the Quanser Aero is omitted due to high-efficiency rotors mounted on it<sup>1</sup>. The main (tail) rotor is attached to the horizontal (vertical) propeller, which is rotated by applying input voltages to corresponding DC motors. Let  $v_p(v_y)$  denote the input voltage to pitch (yaw) DC motor and the state of the 2-DOF helicopter at time  $t \geq 0$  is denoted by  $\mathbf{x}^T(t) \equiv [\theta(t), \psi(t), \dot{\theta}(t), \dot{\psi}(t)]$ , where  $\dot{\theta}(\dot{\psi})$  denote the rotational speed of the pitch (yaw). The free-body diagram of the 2-DOF helicopter is shown in figure 2.2,

where  $F_p(F_y)$  is force from the pitch (yaw) motor. If  $\mathbf{u}^T(t) \equiv [v_p(t), v_y(t)]$  denote the vector of the input voltages for DC motors at time  $t \geq 0$ , then the state-space model of

<sup>1</sup>See Quanser Aero user manual for details.

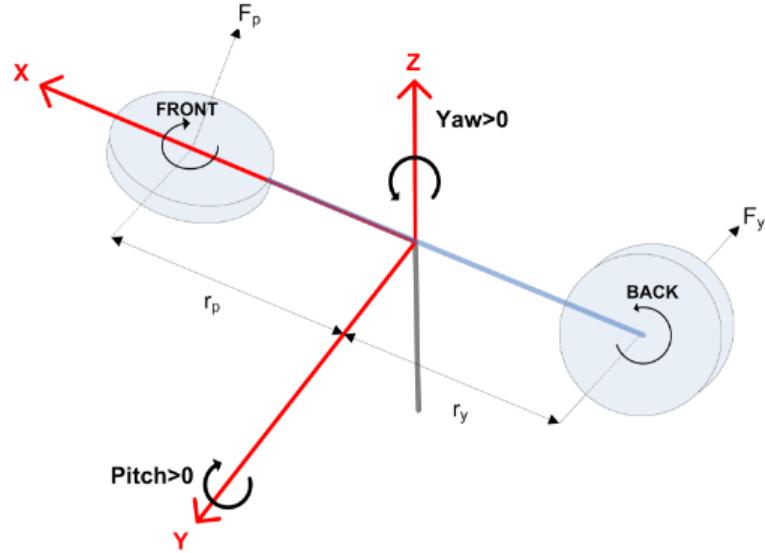


Figure 2.2: Freebody diagram of the forces exerted on a 2-DOF helicopter.

the 2-DOF helicopter (Quanser Aero) is described by [3]:

$$\dot{\mathbf{x}}(t) = \mathbf{Ax}(t) + \mathbf{Bu}(t), \text{ where} \quad (2.1)$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_{sp}}{J_p} & 0 & -\frac{D_p}{J_p} & 0 \\ 0 & 0 & 0 & -\frac{D_y}{J_y} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{K_{pp}}{J_p} & \frac{K_{py}}{J_p} \\ \frac{K_{yp}}{J_y} & \frac{K_{yy}}{J_y} \end{bmatrix}, \quad (2.2)$$

with  $K_{sp}$ ,  $K_{pp}$ ,  $K_{py}$ ,  $K_{yp}$ ,  $K_{yy}$ ,  $J_p$ ,  $J_y$ ,  $D_p$ , and  $D_y$  being the stiffness of the axes, pitch motor thrust constant, thrust constant acting on the pitch angle from the yaw motor, thrust constant acting on yaw angle from pitch motor, yaw motor thrust constant, moment of inertia about pitch axis, moment of inertia about yaw axis, viscous damping of the pitch axis, and viscous damping of the yaw axis, respectively.

In the context of a teleoperation system, a standard problem in controlling motion of states of a 2-DOF helicopter is to find optimal actuator commands  $\mathbf{u}^*(t) \equiv [v_p^*(t), v_y^*(t)]^T$ , such that it follows the command signal, which is the desired (reference) state  $\mathbf{x}^d = [\theta^d, \psi^d, 0, 0]^T$ , sent by the human operator through a mobile device.

# Chapter 3

## Control Algorithms

This chapter discusses the three control algorithms used in the experiment. These are linear quadratic regulator (LQR), linear quadratic gaussian (LQG), and approximate dynamic programming (ADP).

### 3.1 LQR

LQR is a optimal control algorithm that is used to calculate the control gain.  
After creating the system model

$$\dot{x} = Ax + Bu \quad (3.1)$$

use the state feedback law

$$u = -Kx \quad (3.2)$$

to minimize the quadratic cost function:

$$J(u) = \int_0^{\infty} (x^T Q x + u^T R u + 2x^T N u) dt \quad (3.3)$$

Find the solution  $S$  to the Riccati equation

$$A^T S + S A - (S B + N) R^{-1} (B^T S + N^T) + Q = 0 \quad (3.4)$$

Calculate gain,  $K$

$$K = R^{-1} (B^T S + N^T) \quad (3.5)$$

### 3.2 LQG

LQG utilizes gain calculated in LQR but it adds a Kalman filter to reduce external disturbances to the system.

### 3.3 ADP

ADP is a reinforcement learning approach. It utilizes the error of the state variables and feeds them into a hidden layer of neurons as shown in figure 3.1. Each of these neuron uses a quadratic activation function which is a function of error. The result of these nodes are multiplied by a weight which is then sent to an output node where they are summed. This value is then used to calculate the control gain. While the system is running, the controller

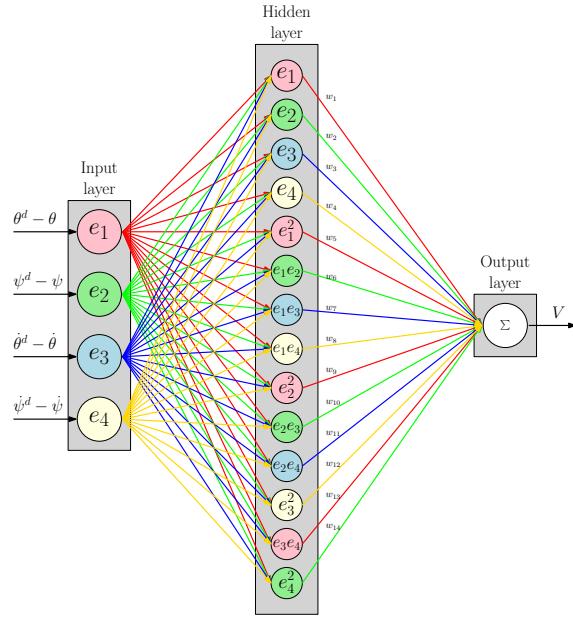


Figure 3.1: Input nodes are connected to the hidden layer and then to the output layer.

collects data every  $\tau$  seconds as shown in figure 3.2. After  $T$  seconds, the data is then fed back into the neural network where the weights are updated.

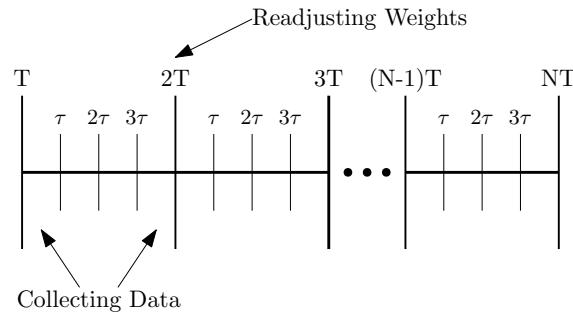


Figure 3.2: Data is collected for every  $\tau$  seconds and then the weights are adjusted every  $T$  seconds.

## 3.4 Improvements

Most of these algorithms are typically implemented only using proportional gain which causes steady-state error to certain input signals in some systems. In our case, we experience steady-state error for a step input. To reduce this, the type of the controller needs to be increased by adding an integrator. We solve this problem by implementing a PI controller where LQR and ADP are used to find the optimal proportional and integral gain.

This is done by creating a new state variable to the system which will represent the integrated state. As a result, the A and B will need to be augmented matrix with different dimensions. This will also result in a different dimension of the K matrix. The K matrix will need to be separated into proportional and integral gain.

# Chapter 4

## Numerical Simulations

This chapter discusses the simulation results for the three control algorithms. These simulations are for the LQR P type LQR PI type, and LQG PI type controllers.

### 4.1 LQR (P controller)

Using MATLAB, we created a program, as seen in section B.1, that calculates LQR for our helicopter and then uses differential equations to simulate the trajectory. Figure 4.1(a) represents the simulated position of the helicopter, figure 4.1(b), illustrates the difference between the actual and desired configurations, and figure 4.1(c) show the simulated voltage to the DC motors. Surprisingly, the error is very small yet not zero which is attributed to a high gain.

### 4.2 LQR (PI controller)

Unlike the P controller, we created a Simulink model instead of using differential equations. To do this, we had to create a block that represented our plant and create feedback for the system as seen in figure 4.2. As expected, as time goes to infinity, the error becomes zero. A dip in figure 4.5(a) can be seen as a result of the voltage decreasing in figure 4.5(d) so that the yaw configuration in figure 4.5(b) can converge on the desired orientation.

### 4.3 LQG (PI Controller)

The LQG block diagram required some minor changes including the use a kalman filter as seen in figure 4.4. The kalman had a larger effect on the system than we were expecting. The position in figure 4.5(a) and figure 4.5(b) had a much slower rise time. The simulated voltage in figure 4.5(c) and figure 4.5(d) also rose much more slowly than LQR. This means that the system is using less energy to reach the desired configuration.

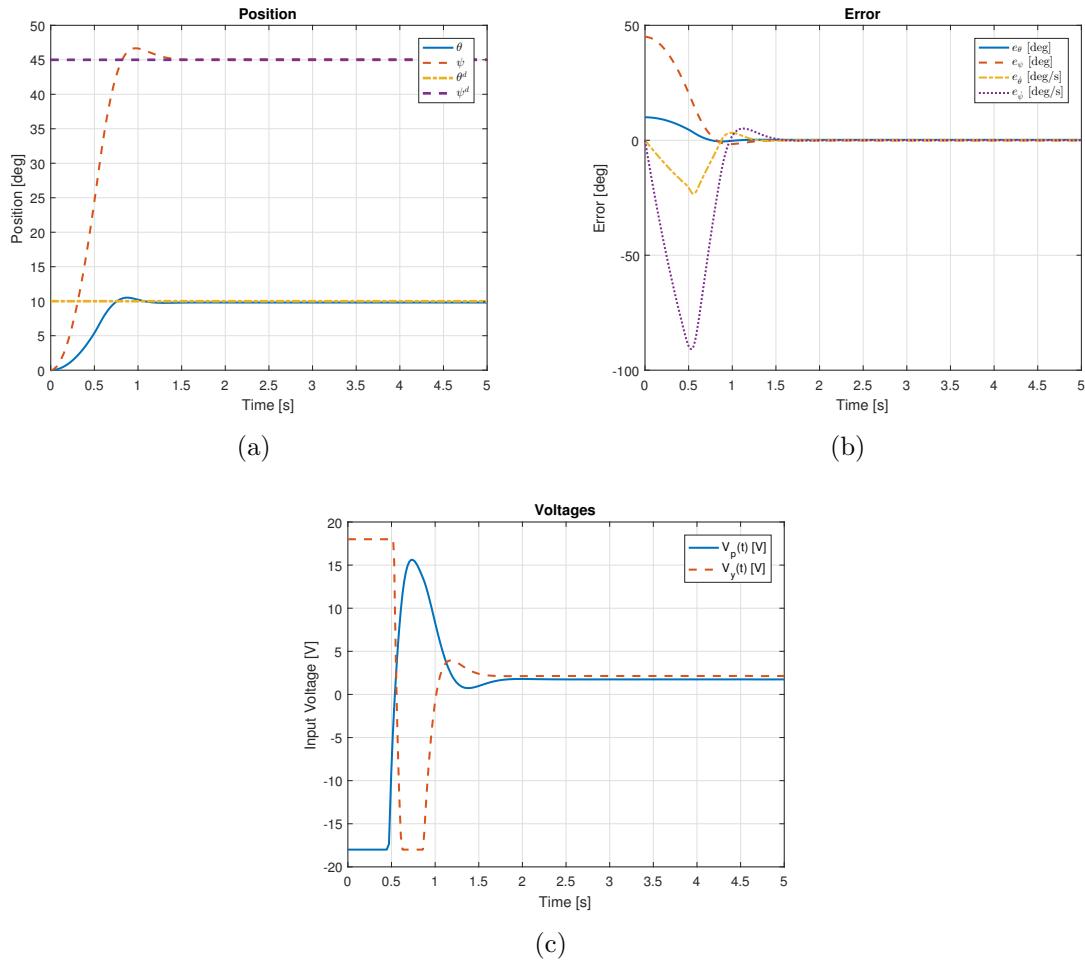


Figure 4.1: Simulations for proportional gain calculated by LQR.

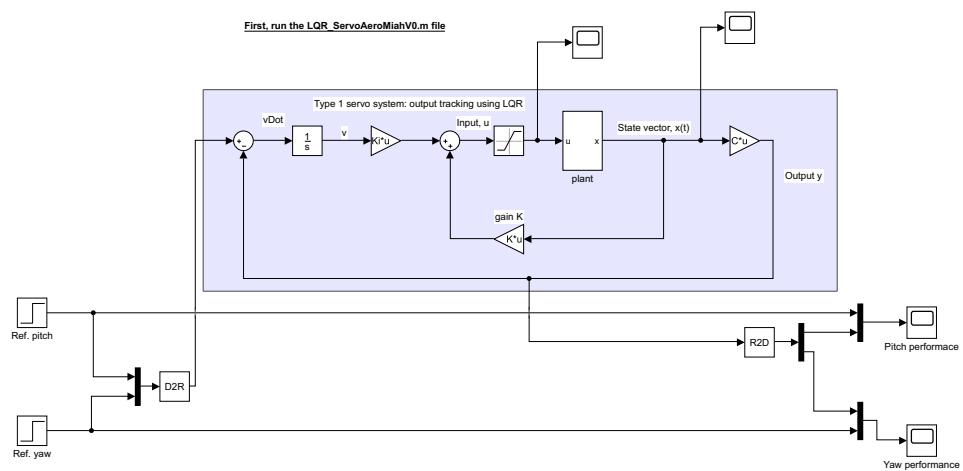


Figure 4.2: Block diagram for LQR simulation using simulink.

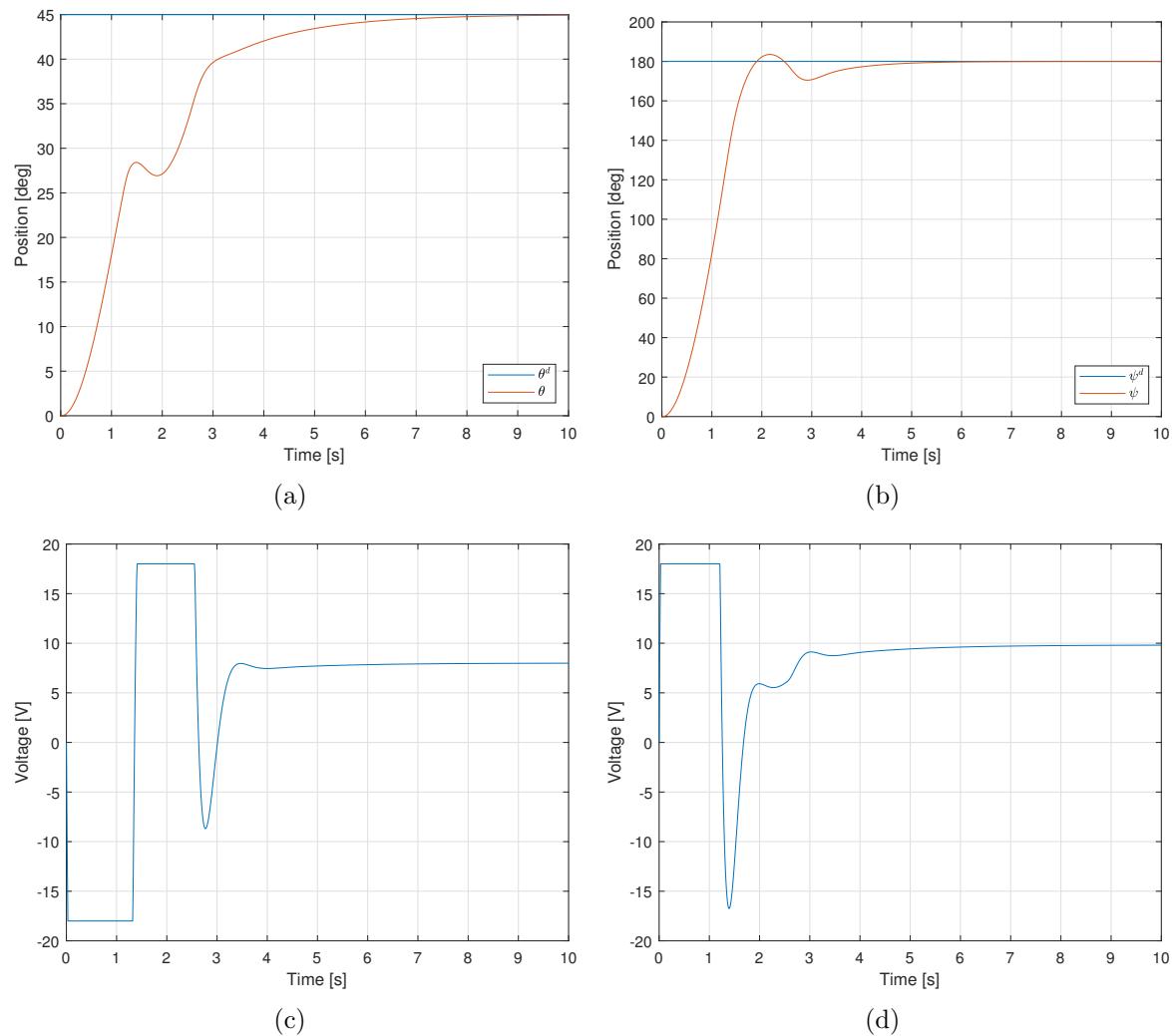


Figure 4.3: Simulations for proportional and integral gain calculated by LQR.

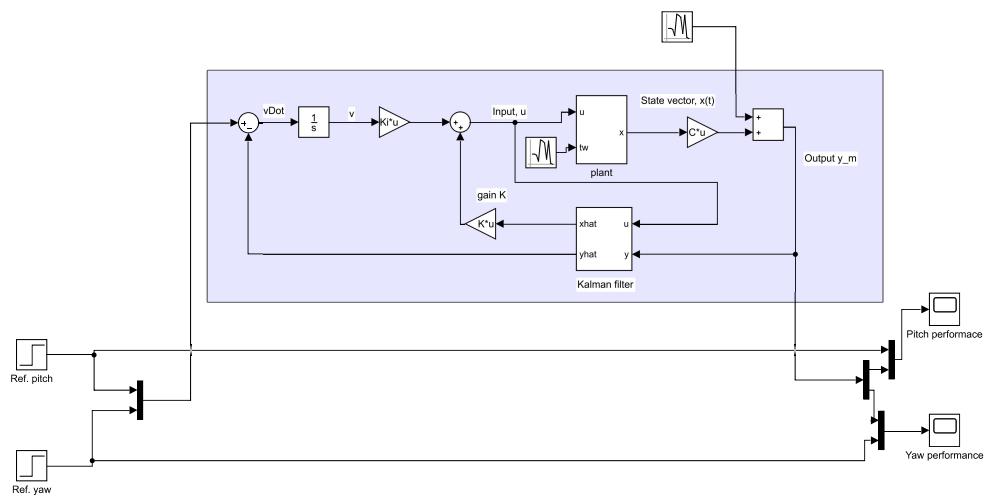


Figure 4.4: Block diagram for LQG simulation using simulink.

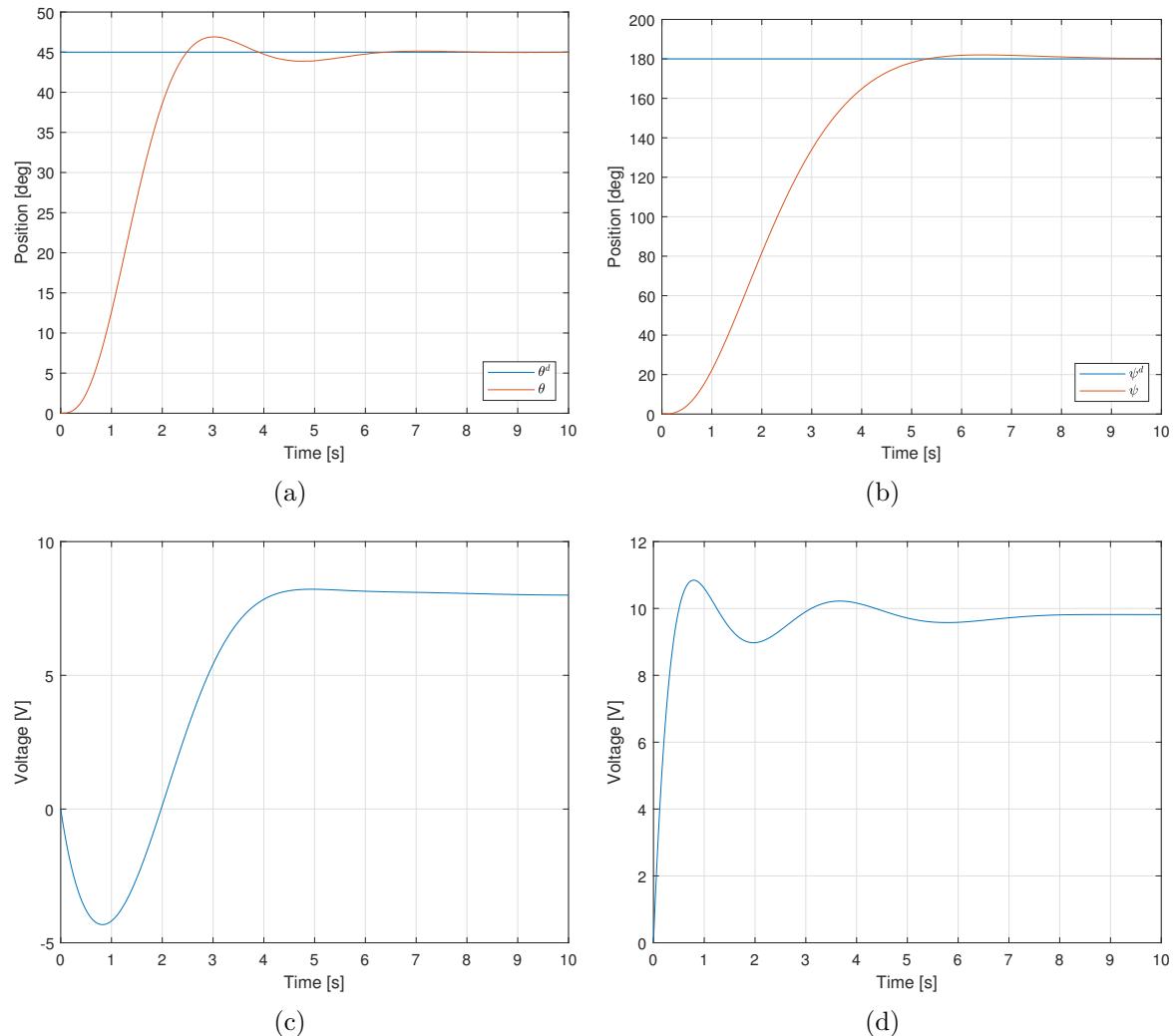


Figure 4.5: Simulations for proportional and integral gain calculated by LQG.

# Chapter 5

## Implementation

This chapter discusses the implementation method setups and results for the three control algorithms. The implementation methods are USB, raspberry pi, and mobile device.

### 5.1 Experimental Setup

Before implementation was started setup needed to be done to get the system ready for testing.

#### 5.1.1 USB

To setup for USB implementation the QFLEX 2 USB panel had to be installed onto the Quanser Aero. Figure E.1 is the USB panel. Section E.1 is a tutorial for the USB communication.

#### 5.1.2 Raspberry Pi

Section E.2 is the tutorial to setup and implement wireless communication using the raspberry pi. The part that allows this implementation to be used is the QFLEX 2 EMBEDDED which is shown in figure E.2. The communication protocol utilized by the Q-Flex2 embedded panel is SPI. To enable this communication figure 5.1 is used in simulink.

#### 5.1.3 Mobile Device

Section E.3 is the tutorial to setup and implement a simulink model using a mobile device as a user interface. Figure 5.2 is the physical lab setup for the mobile device experiment using two helicopters and two raspberry pi's. Figure 5.3 is the TCP communication model between the mobile device and the raspberry pi. When a user wants to send a configuration

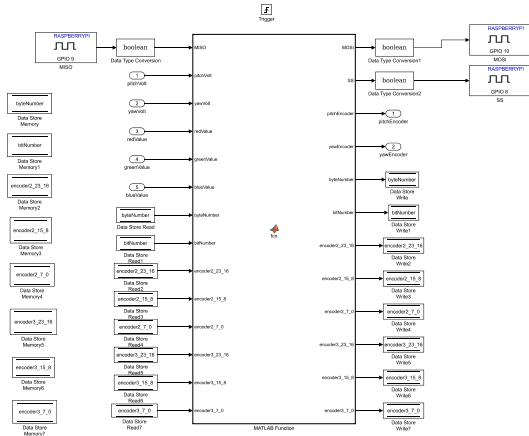


Figure 5.1: Block diagram of SPI communication protocol used for communication between the Raspberry Pi and Quanser Aero.

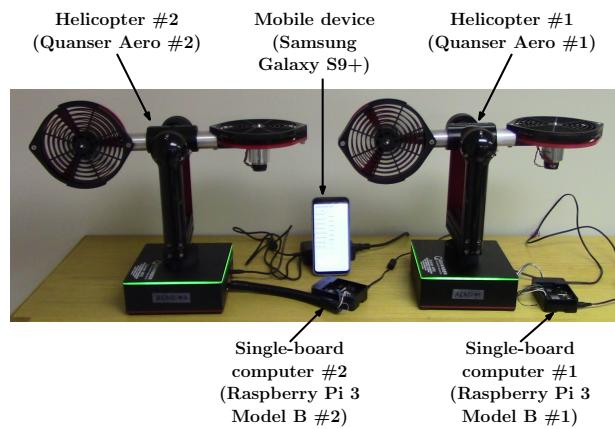


Figure 5.2: Experiment setup for controlling the Quanser Aeros via a mobile device.

to the helicopter, the mobile device's network interface card recognizes the protocol that is being used. In this case, UDP is used to maximize the transmission speed. Metadata is appended to the orginal data for routing purposes. After the packet is prepared, it is sent to the wirless router and across the network. Once it reaches the raspberry pi, it discards the metadata and uses the remaining data to calculate the voltage to apply to the helicopter. Figure 5.4 is the simulink model to generate the mobile device application.

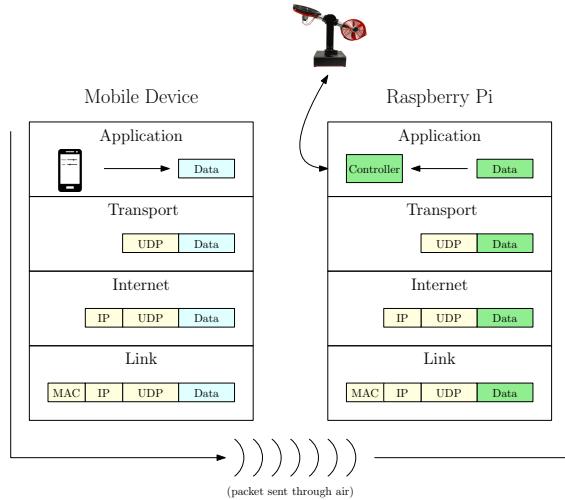


Figure 5.3: Illustration of the TCP model which describes how packets are sent and received from mobile devices to the Quanser Aero.

A screenshot of the application is shown in figure 5.5. As it can be seen, the application

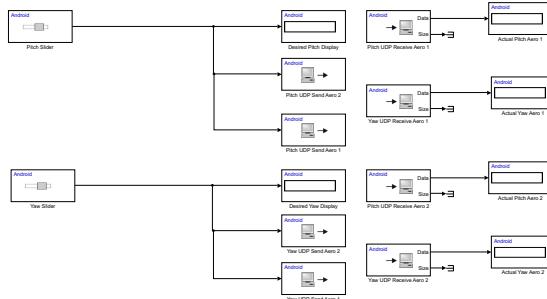


Figure 5.4: Simulink model of the android interface using two Quanser Aeros.

displays the actual pitch and yaw configurations for both helicopters. It also displays the desired configuration that is being set by the two slide bars.

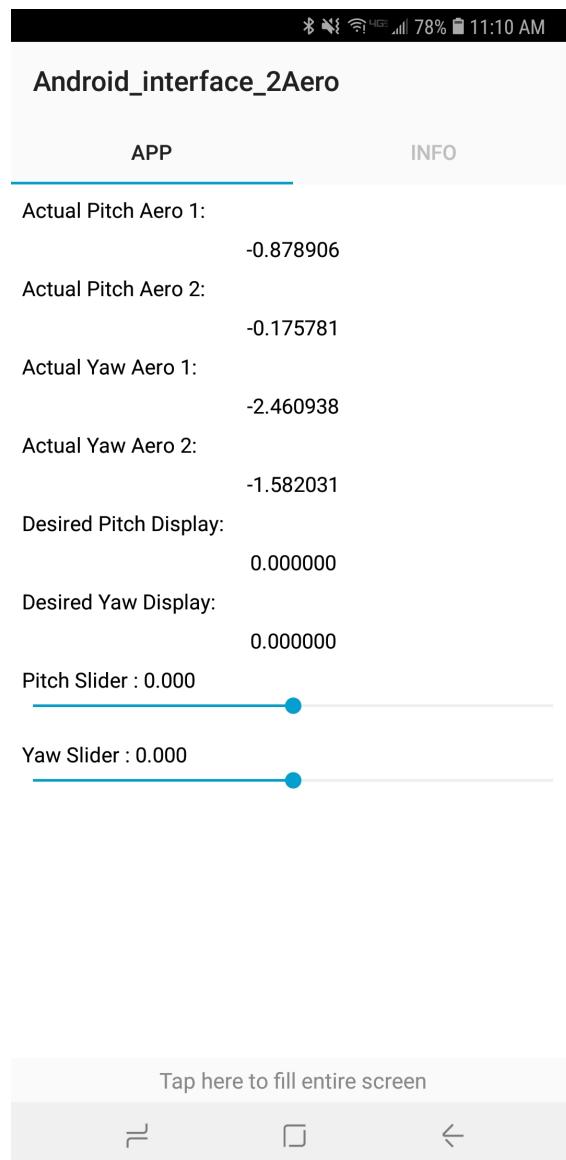


Figure 5.5: Screenshot of the mobile user interface.

## 5.2 USB

Before we tested using wireless communication the algorithms had to be tested using a wired connection to make sure they worked properly. This connection was done using a USB cable and the QFLEX 2 USB panel. Over the next couple of sections the experiment on the Quanser Aero using the USB connection is discussed.

### 5.2.1 LQR

For the LQR motion control algorithm, two types of controllers were implemented. These are P type and PI type controllers. Figure 5.6 is the simulink model for the LQR P type controller using the USB connection to the PC. In the model the desired input configurations are taken in and applied to the LQR P type controller, which calculates the voltages to be applied to the motors of the helicopter. These voltages are applied directly to the helicopter using the USB connection.

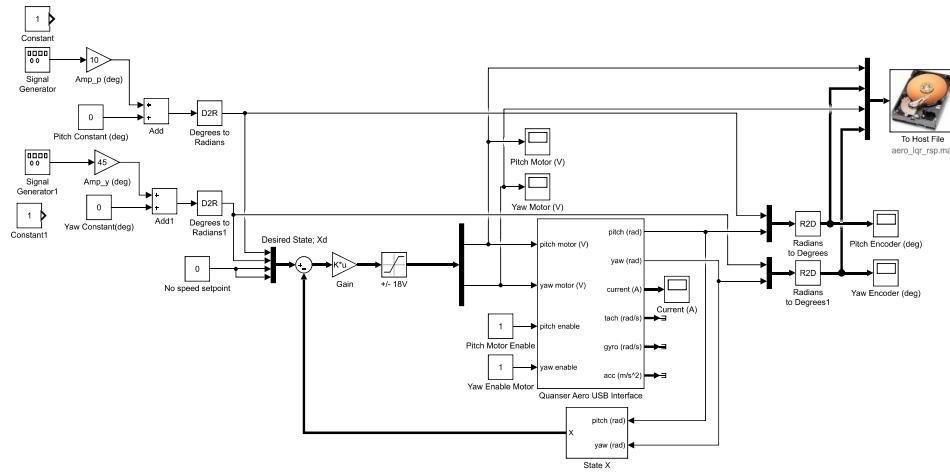


Figure 5.6: Block diagram for LQR P type controller with a USB connection.

Figure 5.7 is the simulink model for the LQR PI type controller using the USB connection. In the model the desired input configurations are taken in and applied to the LQR PI type controller, which calculates the voltages to be applied to the motors of the helicopter. These voltages are applied directly to the helicopter using the USB connection.

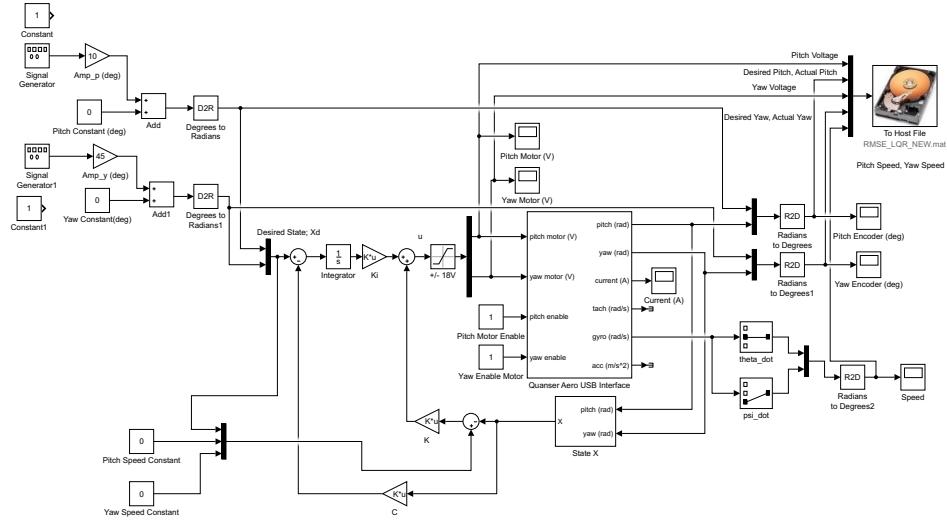


Figure 5.7: Block diagram for LQR PI type controller with a USB connection.

Figure 5.8 shows the wave forms for a step input while comparing LQR P type to LQR PI type controllers. Figure 5.8(a) shows the pitch values of the controllers and the desired. Figure 5.8(b) shows the yaw values of the controllers and the desired. Figure 5.8(c) shows the voltages being applied to the main motor for both controllers. Figure 5.8(d) shows the voltages being applied to the tail motor for both controllers. Figure 5.8(e) displays the speeds of the motion in the pitch direction for both controllers. Figure 5.8(f) displays the speeds of the motion in the yaw direction for both controllers. Figure 5.9 and figure 5.10 have similar figures to figure 5.8, except figure 5.9 has a square wave input and figure 5.10 has a sine wave input.

### 5.2.2 ADP

Figure 5.11 is the simulink model for the ADP motion control algorithm using the USB connection between the Quanser Aero and the PC. It takes the desired configurations as the inputs and applies them and the actual states to the ADP block to calculate the voltages for the main and tail motors. These are then applied directly to the helicopter through the USB connection. Figure 5.12 is what is inside the ADP block in Figure 5.11.

For the results of this experiment figure 5.13 shows the results for ADP using a step input and USB connection. Figure 5.13(a) shows the pitch configurations of both the actual and desired angles versus time. Figure 5.13(b) shows the yaw configurations of both the actual and desired angles versus time. Figure 5.13(c) displays the voltage that is applied to

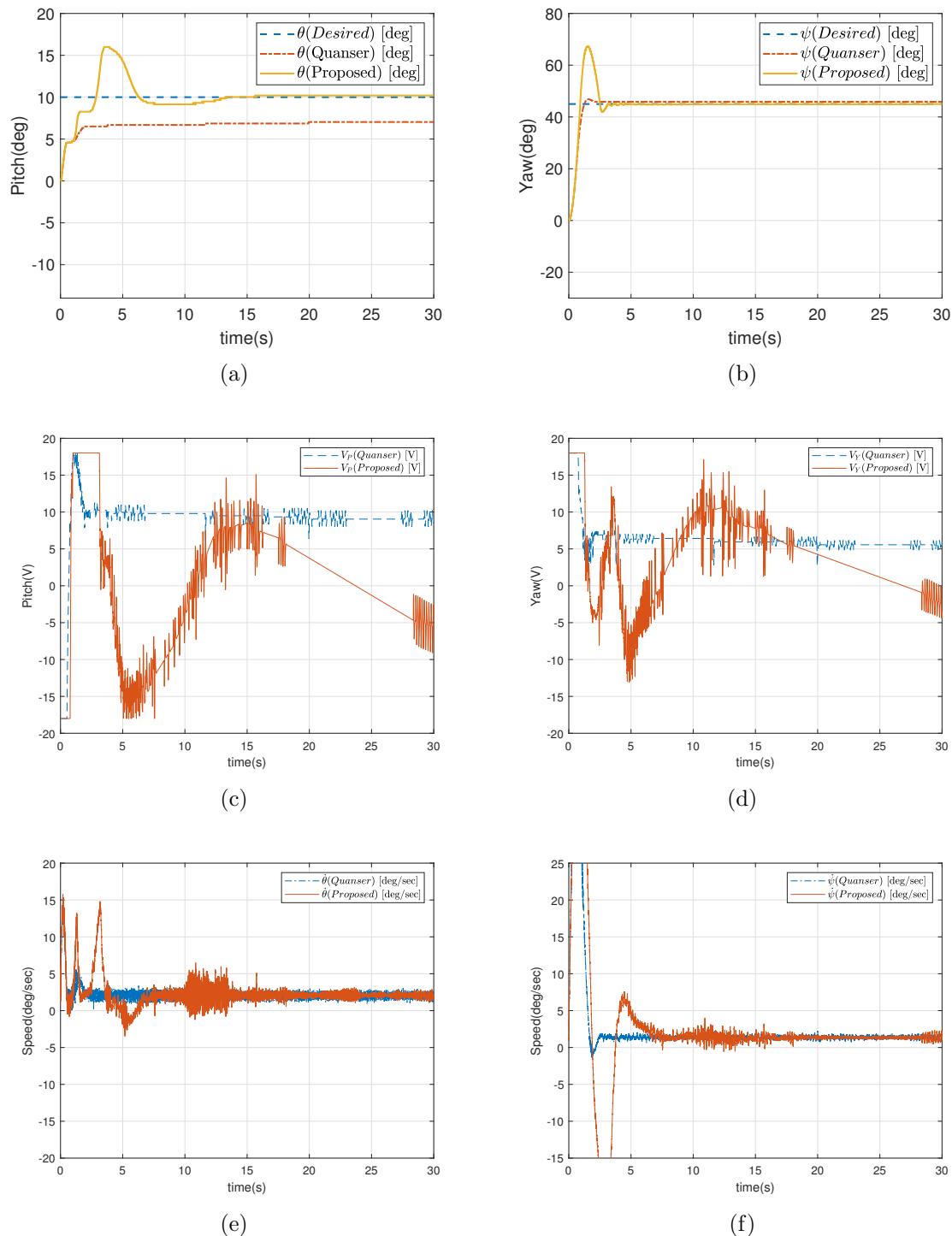


Figure 5.8: USB implementation for proportional controller and proportional-integral controller calculated by LQR with a step input.

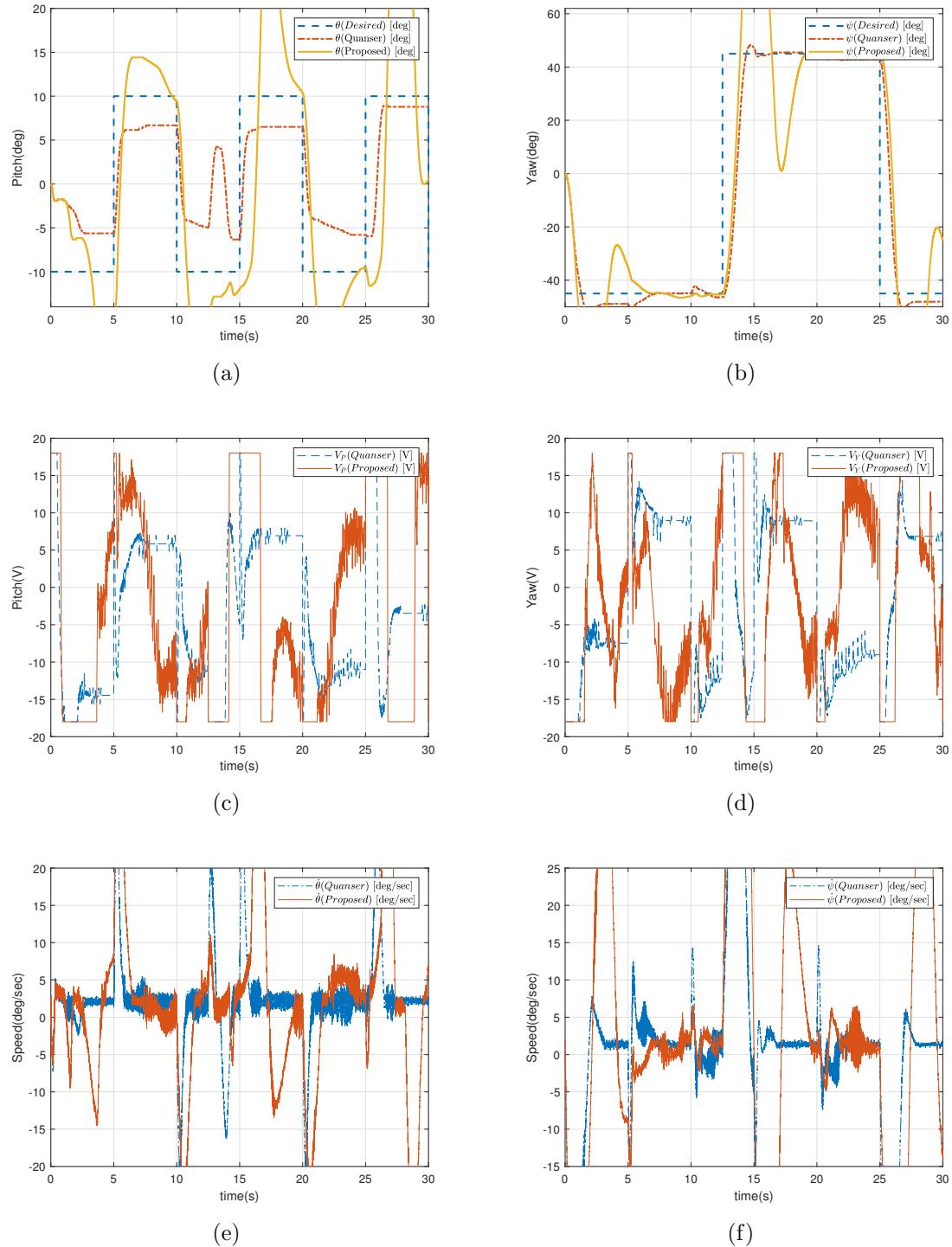


Figure 5.9: USB implementation for proportional controller and proportional-integral controller calculated by LQR with a square wave input.

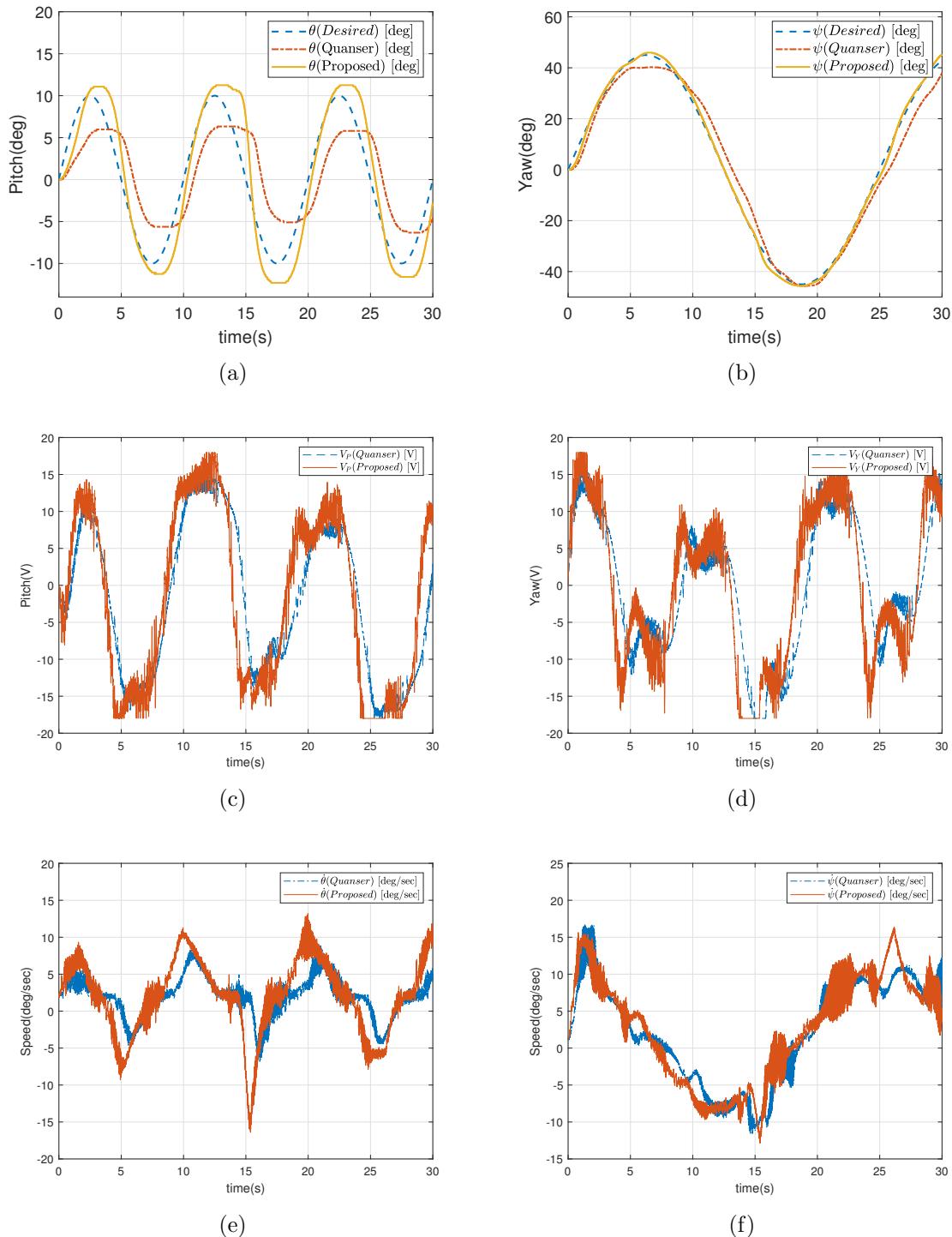


Figure 5.10: USB implementation for proportional controller and proportional-integral controller calculated by LQR with a sinusoidal input.

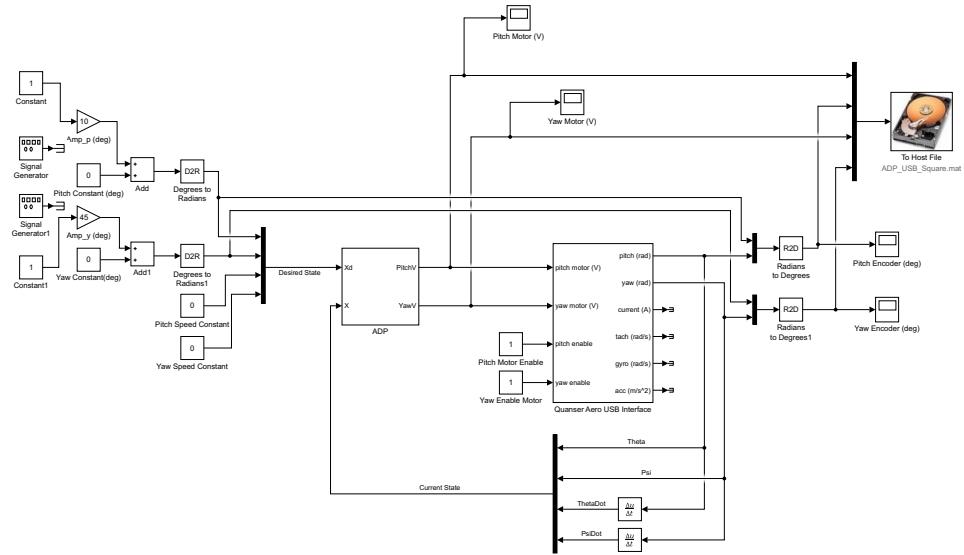


Figure 5.11: Block diagram for ADP P type controller with a USB connection.

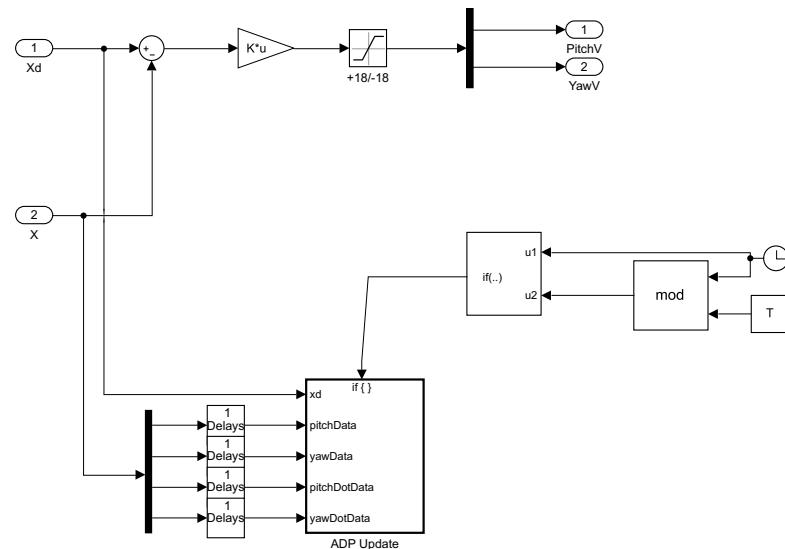


Figure 5.12: Block diagram for ADP P type controller algorithm.

the main motor verses time. Figure 5.13(d) displays the voltage that is applied to the tail motor verses time. Figure 5.14 and figure 5.15 have similar figures to figure 5.13, except figure 5.14 has a square wave input and figure 5.15 has a sine wave input.

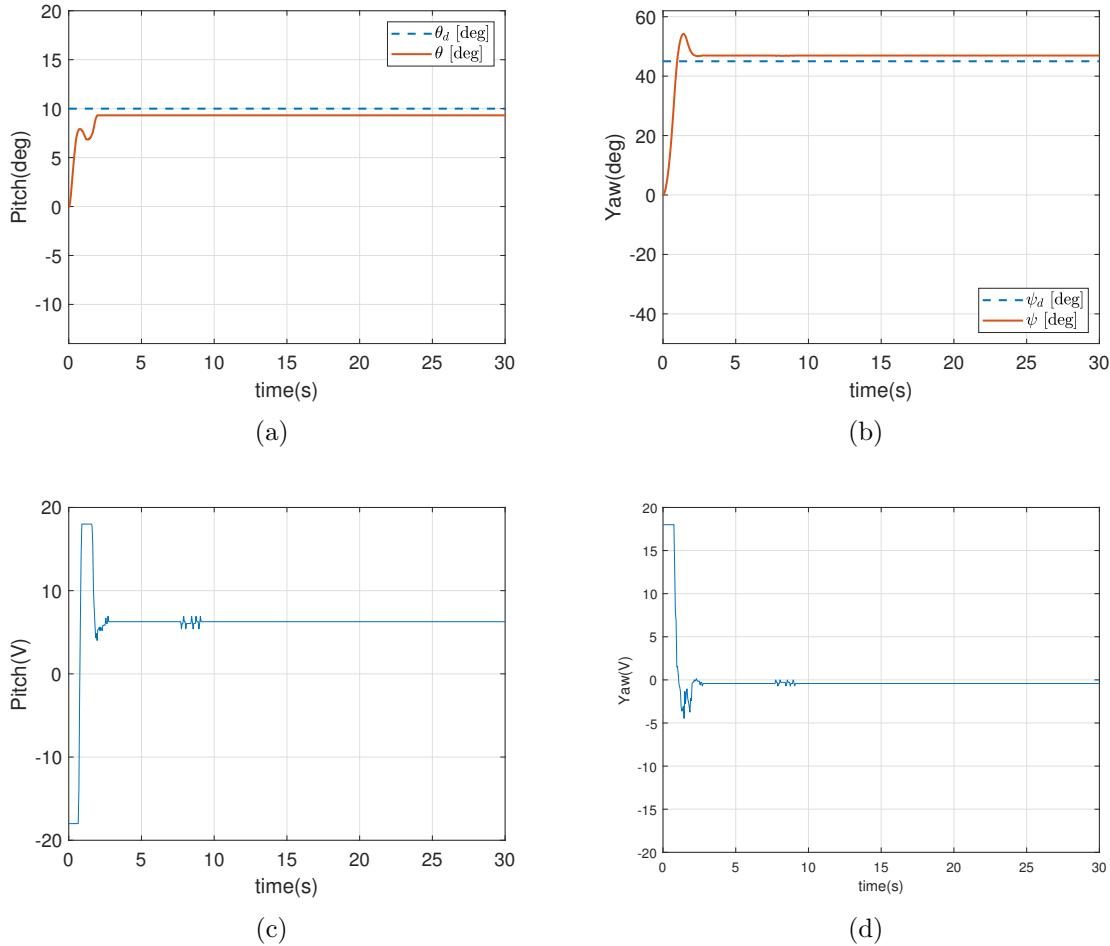


Figure 5.13: USB implementation for ADP proportional controller with a step input.

### 5.2.3 Root Mean Square Error

Table 5.1 give the root mean square error (RMSE) values for LQR P type verses LQR PI type controllers. What the RMSE values show are the error distances to the desired configurations. This table displays these values to show the improvement in the PI type controller compared to the P type controller. Table 5.2 give the RMSE values for LQR P type verses ADP motion controllers. This table displays these values to show the improvement in the ADP algorithm compared to the LQR P type controller algorithm.

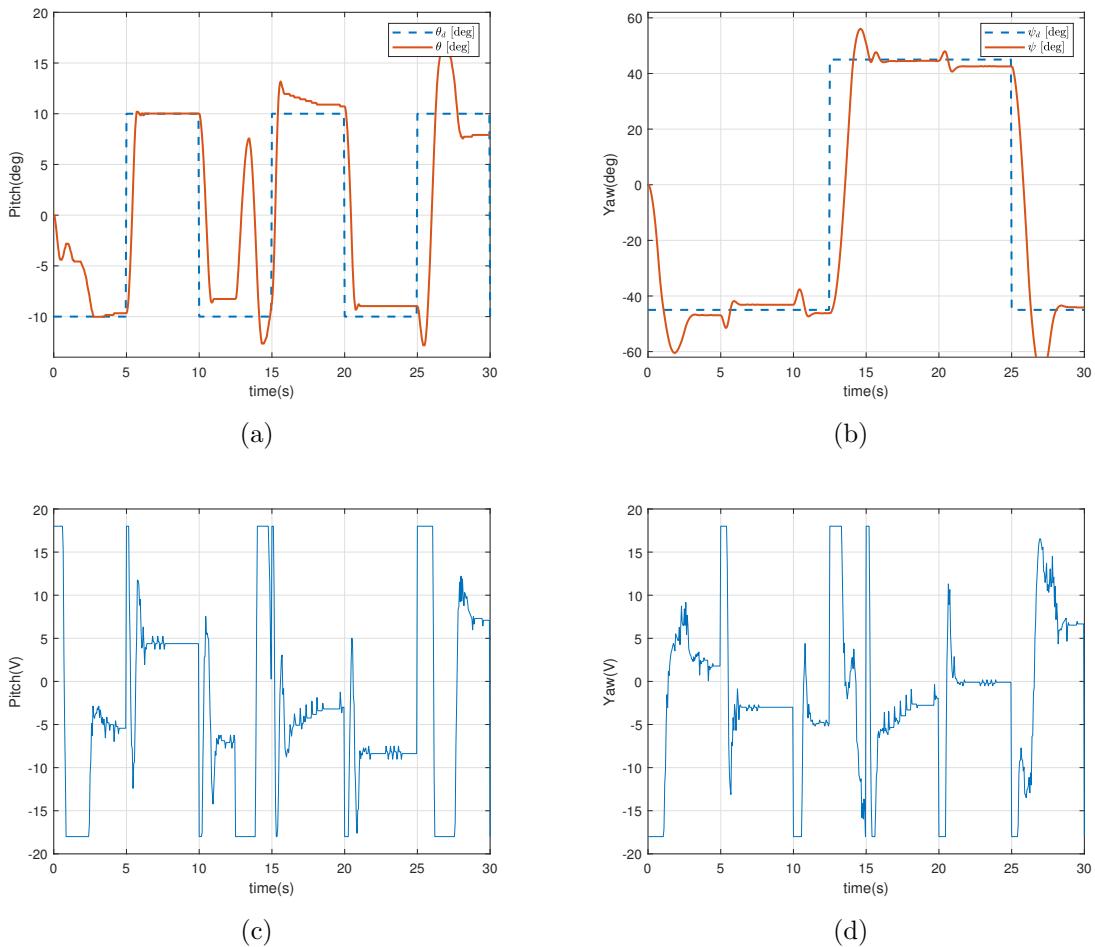


Figure 5.14: USB implementation for ADP proportional controller with a square wave input.

	<b>Pitch Step</b>	<b>Yaw Step</b>	<b>Pitch Squ.</b>	<b>Yaw Squ.</b>	<b>Pitch Sine</b>	<b>Yaw Sine</b>
LQR P	3.5025	5.8502	6.2819	20.4623	4.2469	2.8644
LQR PI	1.2349	5.5058	6.9206	21.0709	1.3383	1.7852
Improvement	64.7437%	0.5408%	-10.1675%	-2.9740%	68.4872%	63.2998%

Table 5.1: Root mean squared error and improvement for LQR P and PI controller

	<b>Pitch Step</b>	<b>Yaw Step</b>	<b>Pitch Squ.</b>	<b>Yaw Squ.</b>	<b>Pitch Sine</b>	<b>Yaw Sine</b>
LQR P	3.5025	5.8502	6.2819	20.4623	4.2469	2.8644
ADP P	1.3067	6.1991	6.5790	21.1923	2.1877	3.6307
Improvement	62.6923%	-5.9638%	-4.7294%	-0.3567%	48.4871%	-26.7525%

Table 5.2: Root mean squared error and improvement for LQR P and ADP P controller

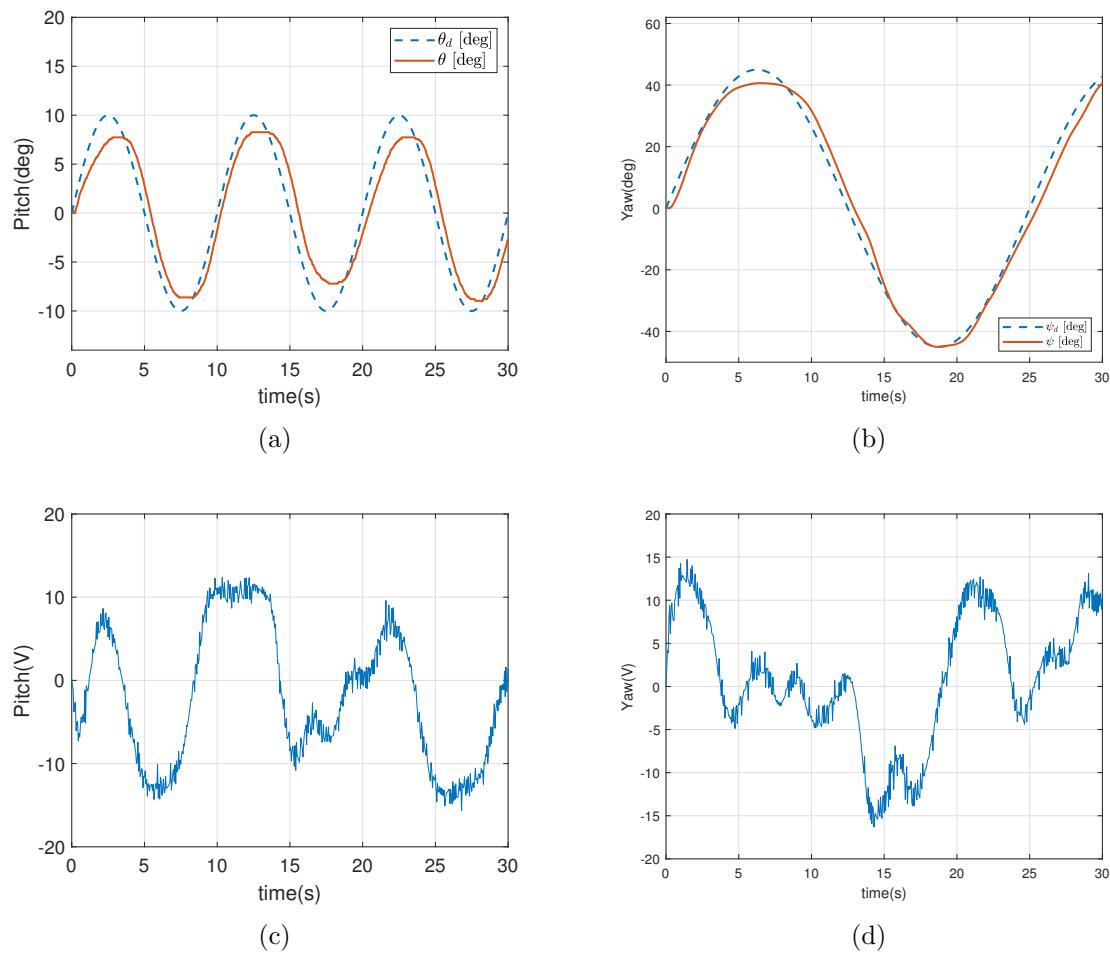


Figure 5.15: USB implementation for ADP proportional controller with a sinusoidal input.

## 5.3 Raspberry Pi

Before the algorithms were implemented using the mobile device they were tested using only the raspberry pi to make sure that the wireless communication worked.

### 5.3.1 LQR (P Type Controller)

Figure 5.16 shows the simulink model that was built to the raspberry pi to test the LQR P type algorithm. It can be seen that it takes predetermined constant inputs to set the desired configuration. Those desired values are subtracted by the actual state which is then multiplied by the control gain and sent to the Quanser Aero via the SPI connection with the raspberry pi.

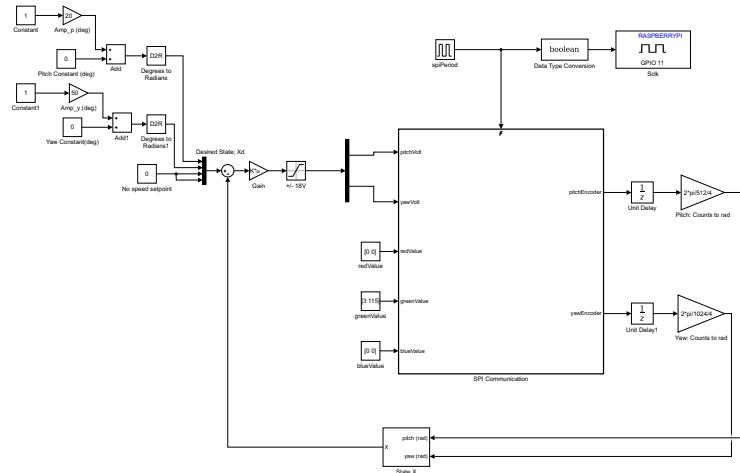


Figure 5.16: Simulink model of LQR P type using raspberry pi communication.

### 5.3.2 ADP

Figure 5.17 shows the simulink model that was built to the raspberry pi to test the ADP algorithm. It can be seen that it takes predetermined constant inputs to set the desired configuration. Those desired values as well as the actual configuration values are sent into the ADP block which calculates the voltages to be applied to the system. These voltages are then sent to the Quanser Aero via the SPI connection with the raspberry pi.

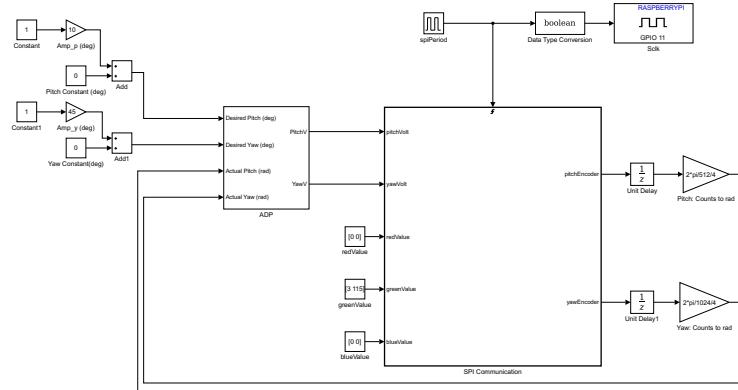


Figure 5.17: Simulink model of ADP using raspberry pi communication.

## 5.4 Mobile Device

Testing and implementation using the mobile device user interface was then done for the LQR p type and ADP motion control algorithms.

### 5.4.1 LQR (P Type Controller)

Figure 5.18 is the simulink model for mobile device communication using LQR p type motion control algorithm. As it can be seen the desired configurations are taken in from the mobile device using UDP receive blocks. These values are then sent into the LQR block with the actual configurations to calculate the voltages for the two motors. These voltage values are then sent to the Quanser Aero from the raspberry pi using the SPI communication. The actual configuration values are then read from the Quanser Aero using the SPI communication, then the raspberry pi sends those actual configuration values to the mobile device using UDP send blocks.

The figure 5.19 shows the results of the implementation using LQR p type for the mobile device user interface. Figure 5.19(a) shows the actual pitch configuration and the desired that was set using the mobile device verses time. Figure 5.19(b) shows the actual yaw configuration and the desired that was set using the mobile device verses time. As can be seen, the actual configuration tries to follow the wave form that is being set by the user. Figure 5.19(c) and figure 5.19(c) show the voltages that are applied to the main and tail motors to get these motions in the pitch and yaw directions.

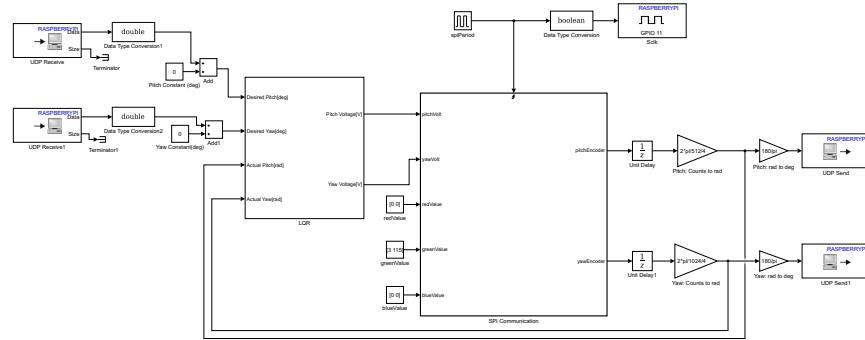


Figure 5.18: Simulink model of LQR using mobile device communication.

#### 5.4.2 ADP

Figure 5.20 is the simulink model for mobile device communication using the ADP motion control algorithm. As it can be seen the desired configurations are taken in from the mobile device using UDP receive blocks. These values are then sent into the ADP block with the actual configurations to calculate the voltages for the two motors. These voltage values are then sent to the Quanser Aero from the raspberry pi using the SPI communication. The actual configuration values are then read from the Quanser Aero using the SPI communication, then the raspberry pi sends those actual configuration values to the mobile device using UDP send blocks.

The figure 5.21 shows the results of the implementation using ADP for the mobile device user interface. Figure 5.21(a) shows the actual pitch configuration and the desired that was set using the mobile device verses time. Figure 5.21(b) shows the actual yaw configuration and the desired that was set using the mobile device verses time. As can be seen, the actual configuration tries to follow the wave form that is being set by the user. Figure 5.21(c) and figure 5.21(c) show the voltages that are applied to the main and tail motors to get these motions in the pitch and yaw directions.

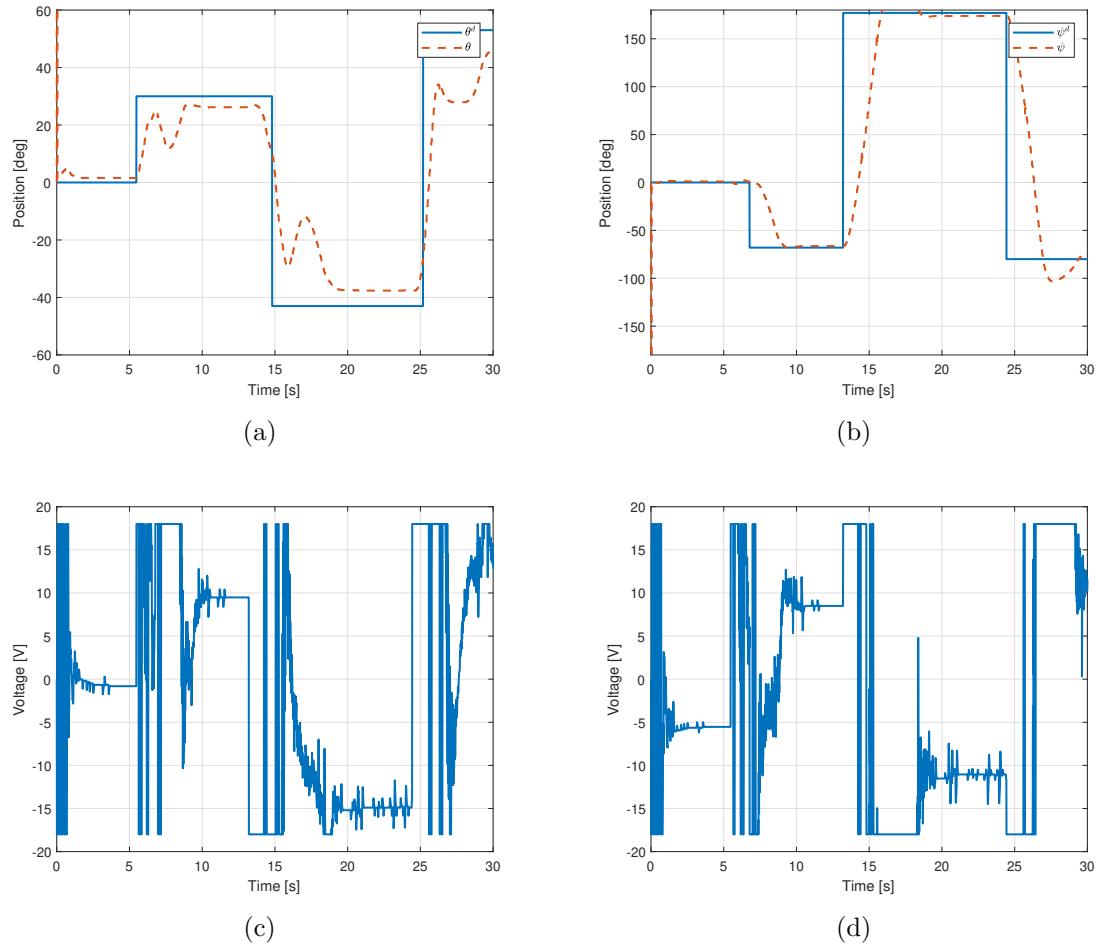


Figure 5.19: Implementation results for LQR P type controller using mobile device as user input.

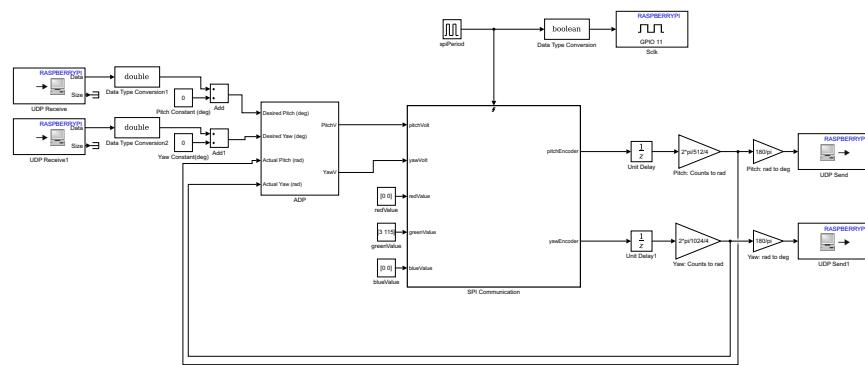


Figure 5.20: Simulink model of ADP using mobile device communication.

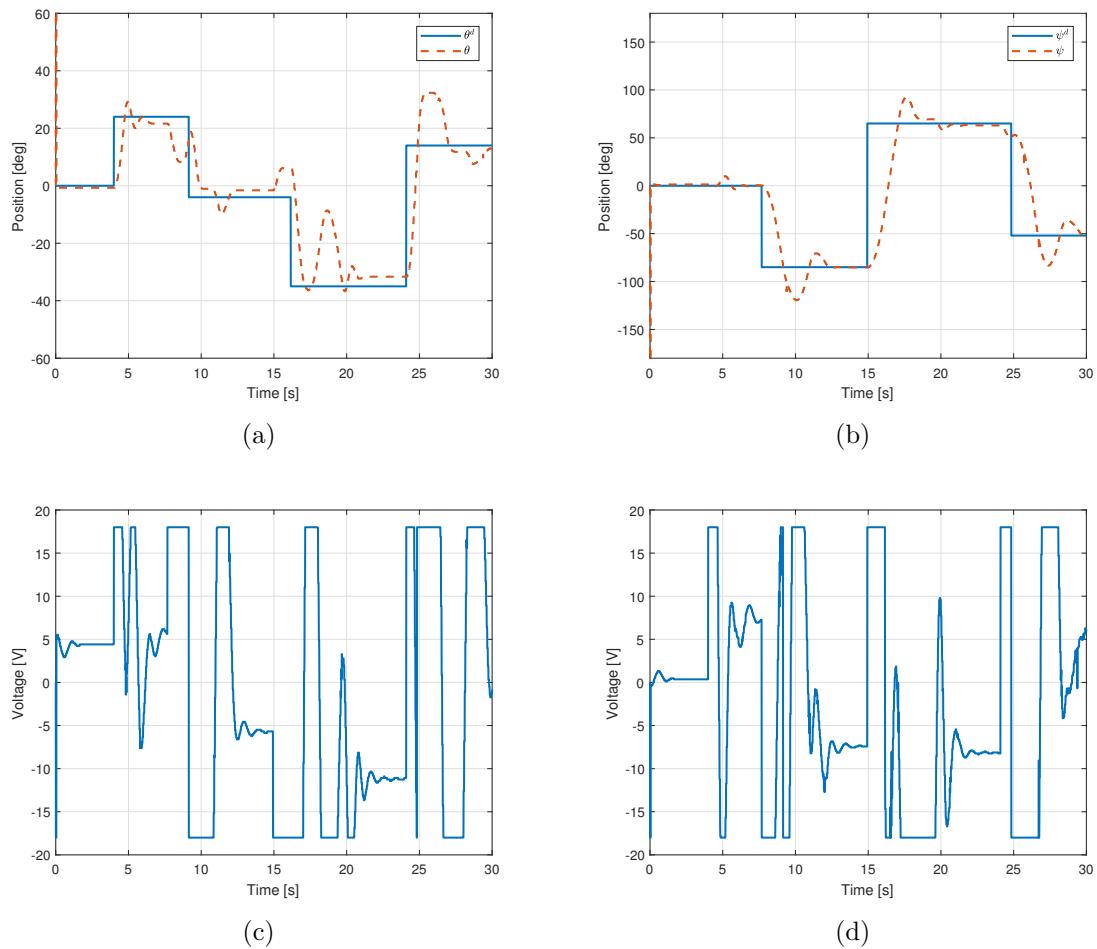


Figure 5.21: Implementation results for ADP controller using mobile device as user input.

# Chapter 6

## Conclusion and Future Work

In this work, we presented a project that successfully utilized, control theory, embedded development, and mechatronics. The following sections will overview the accomplishments and provide suggestions for future work.

### 6.1 Conclusion

Before we started this project, we did research on different kinds of control algorithms done in previous works and addressed by literature. We chose to study and implement LQR, LQG, and ADP on the Quanser Aero because we believed that would be the best for an embedded device.

Simulations were conducted for two of the three different control algorithms explained in this work: LQR and LQG. This was done using Simulink and creating a model for our plant. ADP was already extensively simulated and modeled in last years work.

After we were confident our controller was stable and had met our expectations, we used a laboratory computer to implement our design. This utilized a USB connection to send voltage commands to the helicopter and receive position data. The results were similar to our simulations, but some changes were needed to be made in order to error.

We then began to make adjustments to our design so that it could be implemented on a single board computer. This required us to change the interface for the helicopter to the Q-flex2 panel. This method uses SPI communication and is slower than USB. As a result the performance slightly dropped. We were also unable to implement LQG and the PI version of LQR.

Once we were ready to move to the mobile device, we had to create a new model to be implemented on the mobile device. Also, all models now had to receive UDP send packets containing configuration data.

## 6.2 Future Work

This project has great potential as a learning tool for future students. This section has prepared a list of improvements that could be incorporated in future work.

### 6.2.1 Enhanced Smart Framework

For a device to be considered Smart device, it must meet a certain set of criteria. It must have the ability to interact remotely over a network, respond to user input, and have some degree of autonomy. Although our framework exhibits these characteristics, there is room to expand upon network connectivity. Currently, IP address must be manually set before your programs are built and compiled. As more helicopters and controllers are added to the network, the complexity of the mobile device's program will increase. If too many devices are communicating with the mobile device, it could overload the device's network ability causing effects similar to a denial of service (DoS) attack.

To address this issue, future work could be implemented to create a server which would handle the flow of information from the mobile device to the controllers and vice versa as shown in figure 6.1. In this case, the mobile device should send a configuration signal to the server and specify which helicopters or group of helicopters the configuration should be applied to. This would allow the server to more efficiently manage packets sent and received between devices.

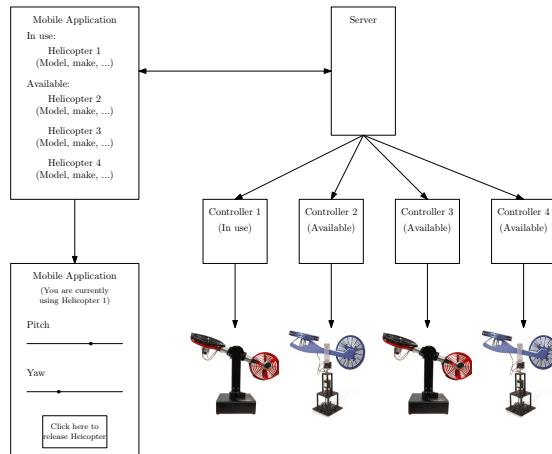


Figure 6.1: Smart algorithm server used for connectivity.

In order to incorporate this kind of connectivity into this framework, it may be necessary to consider a convergence project between electrical engineer and computer science or management information system students.

### 6.2.2 Digital Compass

When the helicopter is initially turned on, the current configuration is assumed to be the point of origin. This is because there is no infrastructure for the helicopter to determine an equilibrium point. This can cause problems if the user is in a remote location and cannot visually confirm the helicopter's heading. We propose that a digital compass be incorporated into the design of the controller both to assist in determining initial orientation, but also to account for encoder error. Encoders commonly report error when left running for long periods of time.

### 6.2.3 Expanded PI Control

This project encountered trouble implementing the PI control architecture as well as LQG on the Raspberry Pi. This may be because the integrator and the kalman filter require a fixed sampling time. Simulink has the option to use a variable sampling time when set to continuous time. Since the model is being loaded onto an embedded system which relies upon a fixed sampling time, this may be affecting the values being outputted by these blocks.

To correct this issue, we recommend that future work done on this project starts with converting the helicopter model into discrete time. Consider equation 6.1 and equation 6.2 to convert the continuous model in equation 2.1:

$$x(k+1) = \Phi x(k) + \Gamma u(k) + \omega u(k) \quad (6.1)$$

$$y(k) = Hx(k) + Ju(k) \quad (6.2)$$

where  $T$  is the sampling time,  $\Phi = e^{AT}$ , and  $\Gamma = \int_0^T e^{A\eta} d\eta B$ .

This project also did not create a PI version for ADP. Future work should also explore this architecture as based on the results discussed in this paper, it is likely that it will prove to be the best control algorithm for this platform.

# Appendix A

## Parameters and State-Space

### A.1 Parameters

Symbol	Description	Value
<b>DC Motor</b>		
$V_{\text{nom}}$	Nominal input voltage	18.0 V
$\tau_{\text{nom}}$	Nominal torque	22.0 mN-m
$\omega_{\text{nom}}$	Nominal speed	3050 RPM
$I_{\text{nom}}$	Nominal current	0.540 A
$R_m$	Terminal resistance	8.4 $\Omega$
$k_t$	Torque constant	0.042 N-m/A
$k_m$	Motor back-emf constant	0.042 V/(rad/s)
$J_m$	Rotor inertia	$4.0 \times 10^{-6}$ kg-m <sup>2</sup>
$L_m$	Rotor inductance	1.16 mH
<b>Aero Body</b>		
$M_b$	Mass of body	1.075 kg
$D_m$	Center of mass	-7.59 mm
$J_p$	Pitch inertia	$2.15 \times 10^{-2}$ kg-m <sup>2</sup>
$J_y$	Yaw inertia	$2.37 \times 10^{-2}$ kg-m <sup>2</sup>
$D_t$	Thrust displacement	15.8 cm
<b>Motor and Pitch Encoders</b>		
	Encoder line count	512 lines/rev
	Encoder line count in quadrature	2048 lines/rev
	Encoder resolution (in quadrature)	0.176 deg/count
<b>Yaw Encoder</b>		
	Encoder line count	1024 lines/rev
	Encoder line count in quadrature	4096 lines/rev
	Encoder resolution (in quadrature)	0.088 deg/count
<b>Amplifier</b>		
	Amplifier type	PWM
	Peak Current	2 A
	Continuous Current	0.5 A
	Output voltage range (recommended)	$\pm 18$ V
	Output voltage range (maximum)	$\pm 24$ V

Figure A.1: System parameters used for the Quanser Aero.

```

1 %
2 %% Quanser Aero Parameters
3 % Moment of Inertia of helicopter body (kg-m^2)
4 L_body = 6.5*0.0254; % length of horizontal body (metal tube)
5 m_body = 0.094; % mass of horizontal body (metal tube)
6 J_body = m_body * L_body^2 / 12; % horizontal cylinder rotating about CM
7 %
8 % Moment of Inertia of yoke fork that rotates about yaw axis (kg-m^2)
9 m_yoke = 0.526; % mass of entire yoke assembly (kg)
10 % h_yoke = 9*0.0254; % height of yoke assembly (m)
11 r_fork = 0.04/2; % radius of each fork (approximated as cylinder)
12 J_yoke = 0.5*m_yoke*r_fork^2;
13 %
14 % Moment of Inertia from motor + guard assembly about pivot (kg-m^2)
15 m_prop = 0.43; % mass of dc motor + shield + propeller shield
16 % m_motor = 0.203; % mass of dc motor
17 r_prop = 6.25*0.0254; % distance from CM to center of pitch axis
18 J_prop = m_prop * r_prop^2; % using parallel axis theorem
19 %
20 % Equivalent Moment of Inertia about Pitch and Yaw Axis (kg-m^2)
21 Jp = J_body + 2*J_prop; % pitch: body and 2 props
22 Jy = J_body + 2*J_prop + J_yoke; % yaw: body, 2 props, and yoke
23 %
24 % Thrust-torque constant (N-m/V) [found experimentally]
25 Kpp = 0.0011; % (pre-production unit: 0.0015)
26 Kyy = 0.0022; % (pre-production unit: 0.0040)
27 Kpy = 0.0021; % thrust acting on pitch from yaw (pre-production unit: ...
    0.0020)
28 Kyp = -0.0027; % thrust acting on yaw from pitch (pre-production ...
    unit: -0.0017)
29 %
30 % Stiffness (N-m/rad) [found experimentally]
31 Ksp = 0.037463;
32 %
33 % Viscous damping (N-m-s/rad) [found experimentally]
34 Dp = 0.0071116; % pitch axis (pre-production unit: Dp = 0.0226)
35 Dy = 0.0220; % yaw axis (pre-production unit: Dy = 0.0211)
36 %

```

## A.2 State-Space Model

```

1 %% State-Space Representation
2 A = [0 0 1 0;
3     0 0 0 1;
4     -Ksp/Jp 0 -Dp/Jp 0;
5     0 0 0 -Dy/Jy];
6
7 B = [0 0;
8     0 0;

```

```
9      Kpp/Jp  Kpy/Jp;  
10     Kyp/Jy  KyY/Jy];  
11  
12 C = eye(2,4);  
13 D = zeros(2,2);  
14 %
```

# Appendix B

## MATLAB Simulation Code

### B.1 LQR (P controller)

```
1 function KV_LQRContinuousTime
2 %% Setup Differential Equation
3 close all
4 clear
5 clc
6
7 quanser_aero_parameters; % contains parameters Quanser Aero
8 quanser_aero_state_space; % puts parameters into state-space model
9
10 Q = diag([3500 500 0 0 ]); % Quadratic term
11 R = 0.005*eye(2,2); % Regulator term
12 K = lqr(A,B,Q,R);
13
14 xInit = zeros(1,4); % set the initial conditions to 0 degrees pitch
15 % and 0 degrees yaw
16 xD = [deg2rad(10) deg2rad(45) 0 0]'; % desidied position is 45 degrees
17 % pitch and 90 degrees yaw
18 eInit = xD'-xInit; % error matrix is desired transposed minus actual
19
20 maxInputVoltage = 18; % cannot exceed this output in [V]
21
22 options = odeset('RelTol', 1e-4, 'AbsTol', 1e-4*ones(1,4));
23
24 t_sim = 10; % [sec]
25 [Te,E] = ode45(@(t,e) stateEqn(t,e,A,B,K,xD,maxInputVoltage), [0 ...
26 t_sim/2], ...
27 eInit, options);
28 xD=-xD;
29 [Te,E] = ode45(@(t,e) stateEqn(t,e,A,B,K,xD,maxInputVoltage), [t_sim/2 ...
30 t_sim], ...
31 eInit, options);
32 %% Position Error Graph
33 figure;
```

```

32 e_1 = plot(Te, rad2deg(E(:,1)), '-','LineWidth', 1.5);
33 hold on
34 e_2 = plot(Te, rad2deg(E(:,2)), '--','LineWidth', 1.5);
35 hold on
36 e_3 = plot(Te, rad2deg(E(:,3)), '-. ','LineWidth', 1.5);
37 hold on
38 e_4 = plot(Te, rad2deg(E(:,4)), ':','LineWidth', 1.5);
39
40 xlabel('Time [s]');
41 ylabel('Error [deg]');
42 title('Error');
43 legend([e_1 e_2 e_3 e_4],{'$e_{\theta}^{\sim}[deg]$', '$e_{\psi}^{\sim}[deg]$', ...
44     '$e_{\dot{\theta}}^{\sim}[deg/s]$', '$e_{\dot{\psi}}^{\sim}[deg/s]$'}, ...
45     'Interpreter', 'latex');
46
47 grid on
48
49 %% Position Graph
50 for i=1:size(E,1)
51     u(i,:) = (K*(E(i,:)))';
52     uLimit(i,:) = sign(u(i,:)).*min(abs(u(i,:)),[maxInputVoltage, ...
53         maxInputVoltage]);
54 end
55
56 xDt = repmat(xD',size(E,1),1); % repeat xDt for all time instants
57 QuanserAeroStates = xDt - E;
58
59 figure;
60 theta = plot(Te, rad2deg(QuanserAeroStates(:,1)), '-','LineWidth', 1.5);
61 hold on
62 psi = plot(Te, rad2deg(QuanserAeroStates(:,2)), '--','LineWidth', 1.5);
63 hold on
64 thetaD = plot(Te, rad2deg(xDt(:,1)), '-. ','LineWidth', 2);
65 hold on
66 psiD = plot(Te, rad2deg(xDt(:,2)), '--','LineWidth', 2);
67
68 xlabel('Time [s]');
69 ylabel('Position [deg]');
70 title('Position');
71 legend([theta psi thetaD psiD],{'$\theta$', '$\psi$', '$\theta^d$', ...
72     '$\psi^d$'}, 'Interpreter', 'latex');
73 grid on
74
75 %% Voltage Graph
76 figure;
77 vP = plot(Te,uLimit(:,1), '-','LineWidth', 1.5);
78 hold on
79 vY = plot(Te,uLimit(:,2), '--','LineWidth', 1.5);
80
81 xlabel('Time [s]');
82 ylabel('Input Voltage [V]');
83 title('Voltages');
84 legend([vP vY], 'V_p(t) [V]', 'V_y(t) [V]');
85 grid on

```

```
86
87 function eDot = stateEqn(t,e,A,B,K,xD,uMax)
88     u = K*e;
89     uLimit = sign(u).*min(abs(u),[uMax;uMax]);
90     eDot = A*e-B*uLimit - A*xD;
```

# Appendix C

## Raspberry Pi MATLAB Code

### C.1 Initialization Code

```
1 %rpi = raspi('192.168.1.79','pi','raspberry'); %Raspberry Pi 1
2 rpi = raspi('192.168.1.20','pi','raspberry'); %Raspberry Pi 2
3
4
5 enableSPI(rpi);
6 disableSPI(rpi);
7
8 % Configure pins on Raspberry Pi for the SPI communication
9 % MOSI Output
10 configurePin(rpi, 10,'DigitalOutput');
11 % SPI Clock Output
12 configurePin(rpi, 11,'DigitalOutput');
13 % SS Output
14 configurePin(rpi, 8,'DigitalOutput');
15 % MISO Input
16 configurePin(rpi, 9,'DigitalInput');
17
18 % SPI output clock period 2us (500kHz)
19 spiPeriod = 0.0001;
20
21 enableSPI(rpi);
22
23 clearvars -except rpi spiPeriod K
```

### C.2 SPI Protocol

```
1 % THIS FUNCTION CONTROLS THE SPI INTERFACING
2 % THIS FUNCTION DETERMINES WHAT DATA IS SENT TO THE QUANSER AERO AND WHAT
```

```

3 % DATA IS RETRIEVED FROM THE QUANSER AERO
4 function ...
    [MOSI,SS,pitchEncoder,yawEncoder,byteNumber,bitNumber,encoder2_23_16,encoder2_15_8,
     = fcn(MISO, ...
        pitchVolt,yawVolt,redValue,greenValue,blueValue,byteNumber,bitNumber,encoder2_23_16
5 % Variables used in the function
6 complement = '0000000000000000';
7 pitchEncoderBin = '00000000000000000000000000000000';
8 yawEncoderBin = '00000000000000000000000000000000';
9 tempBin = '00000000000000000000000000000000';
10 tempVoltBin = '00000000';
11
12 % Output the value of the pitch encoder using the latest byte values
13 % Combine all of the encoder bytes
14 pitchEncoderIn = [encoder2_23_16 encoder2_15_8 encoder2_7_0];
15 % Convert the encoder binary vector into a character array
16 for i = 1:24
17     if (pitchEncoderIn(i) == 1)
18         pitchEncoderBin(i) = '1';
19     else
20         pitchEncoderBin(i) = '0';
21     end
22 end
23 % Calculate the 2's complement value of the encoder
24 if(pitchEncoderBin(1) == '1')
25     for i = 1:32
26         if (i < 9)
27             tempBin(i) = '1';
28         else
29             tempBin(i) = pitchEncoderBin(i - 8);
30         end
31     end
32 else
33     for i = 1:32
34         if (i < 9)
35             tempBin(i) = '0';
36         else
37             tempBin(i) = pitchEncoderBin(i - 8);
38         end
39     end
40 end
41 pitchEncoderTemp = typecast(uint32(bin2dec(tempBin)), 'int32');
42 pitchEncoderOut = cast(pitchEncoderTemp, 'double');
43 pitchEncoder = pitchEncoderOut(1);
44
45 % Output the value of the yaw encoder using the latest byte values
46 % Combine all of the encoder bytes
47 yawEncoderIn = [encoder3_23_16 encoder3_15_8 encoder3_7_0];
48 % Convert the encoder binary vector into a character array
49 for i = 1:24
50     if (yawEncoderIn(i) == 1)
51         yawEncoderBin(i) = '1';
52     else
53         yawEncoderBin(i) = '0';

```

```

54     end
55 end
56 % Calculate the 2's complement value of the encoder
57 if(yawEncoderBin(1) == '1')
58     for i = 1:32
59         if (i < 9)
60             tempBin(i) = '1';
61         else
62             tempBin(i) = yawEncoderBin(i - 8);
63         end
64     end
65 else
66     for i = 1:32
67         if (i < 9)
68             tempBin(i) = '0';
69         else
70             tempBin(i) = yawEncoderBin(i - 8);
71         end
72     end
73 end
74 yawEncoderTemp = typecast(uint32(bin2dec(tempBin)), 'int32');
75 yawEncoderOut = cast(yawEncoderTemp, 'double');
76 yawEncoder = yawEncoderOut(1);
77
78 % Determine which byte we are currently sending
79 % Specify the slave select value
80 % - 0 to activate the SPI on the Quanser AERO
81 % - 1 to de-activate the SPI on the Quanser AERO
82 % Determine the what the byte to be transmitted should be
83 switch byteNumber
84     case 0
85         % ** START OF BASE PACKET ***
86         % BYTE 0
87         % MOSI DATA - BASE MODE (0X01)
88         % MISO DATA - BASE ID MSB
89         % Beginning of the transmission, so SS goes low
90         SS = 0;
91         readyMOSI = dec2bin(1,8);
92
93     case 1
94         % BYTE 1
95         % MOSI DATA - PADDING BYTE (0X00)
96         % MISO DATA - BASE ID LSB
97         SS = 0;
98         readyMOSI = dec2bin(0,8);
99
100    case 2
101        % BYTE 2
102        % MOSI DATA - BASE WRITE MASK
103        % MISO DATA - ENCODER 2 (23-16)
104        SS = 0;
105        % Enable the overwriting of encoders 2 and 3 and the LED colors
106        % Bit 4 - Set encoder 3 enable
107        % Bit 3 - Set encoder 2 enable

```

```

108     % Bit 2 - Write blue LED
109     % Bit 1 - Write green LED
110     % Bit 0 - Write red LED
111     % Don't want to overwrite the encoder values
112     readyMOSI = dec2bin(7,8);
113     % Receive the MISO byte bit by bit and update the vector holding the
114     % byte
115     switch bitNumber
116         case 1
117             if (MISO == true)
118                 encoder2_23_16(1) = 1;
119             else
120                 encoder2_23_16(1) = 0;
121             end
122         case 2
123             if (MISO == true)
124                 encoder2_23_16(2) = 1;
125             else
126                 encoder2_23_16(2) = 0;
127             end
128         case 3
129             if (MISO == true)
130                 encoder2_23_16(3) = 1;
131             else
132                 encoder2_23_16(3) = 0;
133             end
134         case 4
135             if (MISO == true)
136                 encoder2_23_16(4) = 1;
137             else
138                 encoder2_23_16(4) = 0;
139             end
140         case 5
141             if (MISO == true)
142                 encoder2_23_16(5) = 1;
143             else
144                 encoder2_23_16(5) = 0;
145             end
146         case 6
147             if (MISO == true)
148                 encoder2_23_16(6) = 1;
149             else
150                 encoder2_23_16(6) = 0;
151             end
152         case 7
153             if (MISO == true)
154                 encoder2_23_16(7) = 1;
155             else
156                 encoder2_23_16(7) = 0;
157             end
158         case 8
159             if (MISO == true)
160                 encoder2_23_16(8) = 1;
161             else

```

```

162             encoder2_23_16(8) = 0;
163         end
164     end
165
166     case 3
167         % BYTE 3
168         % MOSI DATA - RED LED MSB
169         % MISO DATA - ENCODER 2 (15-8)
170         SS = 0;
171         readyMOSI = dec2bin(redValue(1),8);
172         % Receive the MISO byte bit by bit and update the vector holding the
173         % byte
174         switch bitNumber
175             case 1
176                 if (MISO == true)
177                     encoder2_15_8(1) = 1;
178                 else
179                     encoder2_15_8(1) = 0;
180                 end
181             case 2
182                 if (MISO == true)
183                     encoder2_15_8(2) = 1;
184                 else
185                     encoder2_15_8(2) = 0;
186                 end
187             case 3
188                 if (MISO == true)
189                     encoder2_15_8(3) = 1;
190                 else
191                     encoder2_15_8(3) = 0;
192                 end
193             case 4
194                 if (MISO == true)
195                     encoder2_15_8(4) = 1;
196                 else
197                     encoder2_15_8(4) = 0;
198                 end
199             case 5
200                 if (MISO == true)
201                     encoder2_15_8(5) = 1;
202                 else
203                     encoder2_15_8(5) = 0;
204                 end
205             case 6
206                 if (MISO == true)
207                     encoder2_15_8(6) = 1;
208                 else
209                     encoder2_15_8(6) = 0;
210                 end
211             case 7
212                 if (MISO == true)
213                     encoder2_15_8(7) = 1;
214                 else
215                     encoder2_15_8(7) = 0;

```

```

216         end
217     case 8
218         if (MISO == true)
219             encoder2_15_8(8) = 1;
220         else
221             encoder2_15_8(8) = 0;
222         end
223     end
224
225 case 4
226 % BYTE 4
227 % MOSI DATA - RED LED LSB
228 % MISO DATA - ENCODER 2 (7-0)
229 SS = 0;
230 readyMOSI = dec2bin(redValue(2),8);
231 % Receive the MISO byte bit by bit and update the vector holding the
232 % byte
233 switch bitNumber
234     case 1
235         if (MISO == true)
236             encoder2_7_0(1) = 1;
237         else
238             encoder2_7_0(1) = 0;
239         end
240     case 2
241         if (MISO == true)
242             encoder2_7_0(2) = 1;
243         else
244             encoder2_7_0(2) = 0;
245         end
246     case 3
247         if (MISO == true)
248             encoder2_7_0(3) = 1;
249         else
250             encoder2_7_0(3) = 0;
251         end
252     case 4
253         if (MISO == true)
254             encoder2_7_0(4) = 1;
255         else
256             encoder2_7_0(4) = 0;
257         end
258     case 5
259         if (MISO == true)
260             encoder2_7_0(5) = 1;
261         else
262             encoder2_7_0(5) = 0;
263         end
264     case 6
265         if (MISO == true)
266             encoder2_7_0(6) = 1;
267         else
268             encoder2_7_0(6) = 0;
269         end

```

```

270         case 7
271             if (MISO == true)
272                 encoder2_7_0(7) = 1;
273             else
274                 encoder2_7_0(7) = 0;
275             end
276         case 8
277             if (MISO == true)
278                 encoder2_7_0(8) = 1;
279             else
280                 encoder2_7_0(8) = 0;
281             end
282         end
283
284     case 5
285     % BYTE 5
286     % MOSI DATA — GREEN LED MSB
287     % MISO DATA — ENCODER 3 (23–16)
288     SS = 0;
289     readyMOSI = dec2bin(greenValue(1),8);
290     % Receive the MISO byte bit by bit and update the vector holding the
291     % byte
292     switch bitNumber
293         case 1
294             if (MISO == true)
295                 encoder3_23_16(1) = 1;
296             else
297                 encoder3_23_16(1) = 0;
298             end
299         case 2
300             if (MISO == true)
301                 encoder3_23_16(2) = 1;
302             else
303                 encoder3_23_16(2) = 0;
304             end
305         case 3
306             if (MISO == true)
307                 encoder3_23_16(3) = 1;
308             else
309                 encoder3_23_16(3) = 0;
310             end
311         case 4
312             if (MISO == true)
313                 encoder3_23_16(4) = 1;
314             else
315                 encoder3_23_16(4) = 0;
316             end
317         case 5
318             if (MISO == true)
319                 encoder3_23_16(5) = 1;
320             else
321                 encoder3_23_16(5) = 0;
322             end
323         case 6

```

```

324         if (MISO == true)
325             encoder3_23_16(6) = 1;
326         else
327             encoder3_23_16(6) = 0;
328         end
329     case 7
330         if (MISO == true)
331             encoder3_23_16(7) = 1;
332         else
333             encoder3_23_16(7) = 0;
334         end
335     case 8
336         if (MISO == true)
337             encoder3_23_16(8) = 1;
338         else
339             encoder3_23_16(8) = 0;
340         end
341     end
342
343 case 6
344 % BYTE 6
345 % MOSI DATA — GREEN LED LSB
346 % MISO DATA — ENCODER 3 (15–8)
347 SS = 0;
348 readyMOSI = dec2bin(greenValue(2),8);
349 % Receive the MISO byte bit by bit and update the vector holding the
350 % byte
351 switch bitNumber
352     case 1
353         if (MISO == true)
354             encoder3_15_8(1) = 1;
355         else
356             encoder3_15_8(1) = 0;
357         end
358     case 2
359         if (MISO == true)
360             encoder3_15_8(2) = 1;
361         else
362             encoder3_15_8(2) = 0;
363         end
364     case 3
365         if (MISO == true)
366             encoder3_15_8(3) = 1;
367         else
368             encoder3_15_8(3) = 0;
369         end
370     case 4
371         if (MISO == true)
372             encoder3_15_8(4) = 1;
373         else
374             encoder3_15_8(4) = 0;
375         end
376     case 5
377         if (MISO == true)

```

```

378         encoder3_15_8(5) = 1;
379     else
380         encoder3_15_8(5) = 0;
381     end
382     case 6
383         if (MISO == true)
384             encoder3_15_8(6) = 1;
385         else
386             encoder3_15_8(6) = 0;
387         end
388     case 7
389         if (MISO == true)
390             encoder3_15_8(7) = 1;
391         else
392             encoder3_15_8(7) = 0;
393         end
394     case 8
395         if (MISO == true)
396             encoder3_15_8(8) = 1;
397         else
398             encoder3_15_8(8) = 0;
399         end
400     end
401
402     case 7
403     % BYTE 7
404     % MOSI DATA - BLUE LED MSB
405     % MISO DATA - ENCODER 3 (7-0)
406     SS = 0;
407     readyMOSI = dec2bin(blueValue(1),8);
408     % Receive the MISO byte bit by bit and update the vector holding the
409     % byte
410     switch bitNumber
411         case 1
412             if (MISO == true)
413                 encoder3_7_0(1) = 1;
414             else
415                 encoder3_7_0(1) = 0;
416             end
417         case 2
418             if (MISO == true)
419                 encoder3_7_0(2) = 1;
420             else
421                 encoder3_7_0(2) = 0;
422             end
423         case 3
424             if (MISO == true)
425                 encoder3_7_0(3) = 1;
426             else
427                 encoder3_7_0(3) = 0;
428             end
429         case 4
430             if (MISO == true)
431                 encoder3_7_0(4) = 1;

```

```

432         else
433             encoder3_7_0(4) = 0;
434         end
435     case 5
436         if (MISO == true)
437             encoder3_7_0(5) = 1;
438         else
439             encoder3_7_0(5) = 0;
440         end
441     case 6
442         if (MISO == true)
443             encoder3_7_0(6) = 1;
444         else
445             encoder3_7_0(6) = 0;
446         end
447     case 7
448         if (MISO == true)
449             encoder3_7_0(7) = 1;
450         else
451             encoder3_7_0(7) = 0;
452         end
453     case 8
454         if (MISO == true)
455             encoder3_7_0(8) = 1;
456         else
457             encoder3_7_0(8) = 0;
458         end
459     end
460
461     case 8
462     % BYTE 8
463     % MOSI DATA — BLUE LED LSB
464     % MISO DATA — TACHOMETER 2 (23–16)
465     SS = 0;
466     readyMOSI = dec2bin(blueValue(2),8);
467
468     case 9
469     % BYTE 9
470     % MOSI DATA — SET ENCODER 2 (23–16)
471     % MISO DATA — TACHOMETER 2 (15–8)
472     SS = 0;
473     readyMOSI = dec2bin(0,8);
474
475     case 10
476     % BYTE 10
477     % MOSI DATA — SET ENCODER 2 (15–8)
478     % MISO DATA — TACHOMETER 2 (7–0)
479     SS = 0;
480     readyMOSI = dec2bin(0,8);
481
482     case 11
483     % BYTE 11
484     % MOSI DATA — SET ENCODER 2 (7–0)
485     % MISO DATA — TACHOMETER 3 (23–16)

```

```

486     SS = 0;
487     readyMOSI = dec2bin(0,8);
488
489     case 12
490     % BYTE 12
491     % MOSI DATA — SET ENCODER 3 (23–16)
492     % MISO DATA — TACHOMETER 3 (15–8)
493     SS = 0;
494     readyMOSI = dec2bin(0,8);
495
496     case 13
497     % BYTE 13
498     % MOSI DATA — SET ENCODER 3 (15–8)
499     % MISO DATA — TACHOMETER 3 (7–0)
500     SS = 0;
501     readyMOSI = dec2bin(0,8);
502
503     case 14
504     % BYTE 14
505     % MOSI DATA — SET ENCODER 3 (7–0)
506     % MISO DATA — RESERVED (0X00)
507     SS = 0;
508     readyMOSI = dec2bin(0,8);
509
510     case 15
511     % ** START OF CORE PACKET **
512     % BYTE 15
513     % MOSI DATA — CORE MODE (0X01)
514     % MISO DATA — CORE ID MSB
515     SS = 0;
516     readyMOSI = dec2bin(1,8);
517
518     case 16
519     % BYTE 16
520     % MOSI DATA — PADDING BYTE (0X00)
521     % MISO DATA — CORE ID LSB
522     SS = 0;
523     readyMOSI = dec2bin(0,8);
524
525     case 17
526     % BYTE 17
527     % MOSI DATA — CORE WRITE MASK
528     % MISO DATA — CURRENT SENSE 0 (15–8)
529     SS = 0;
530     % Do not enable the overwriting of encoders 0 and 1, but do ...
      enable the
      % overwriting of the motor voltages
531     % Bit 5 — Set encoder 1 enable
532     % Bit 4 — Set encoder 0 enable
533     % Bit 3 — Write motor 1 voltage
534     % Bit 2 — Write motor 1 enable
535     % Bit 1 — Write motor 0 voltage
536     % Bit 0 — Write motor 0 enable
537
538     readyMOSI = dec2bin(15,8);

```

```

539
540     case 18
541         % BYTE 18
542         % MOSI DATA - MOTOR 0 COMMAND (15-8)
543         % MISO DATA - CURRENT SENSE 0 (7-0)
544         SS = 0;
545         % Convert the desired voltage to a value between -999 and 999
546         % 24 is the saturation level in the model
547         pitchVoltTemp = ceil((999*pitchVolt)/24);
548         % If the desired voltage is positive, concatenate a '1' to the ...
549             front of
550             % the voltage value
551             % Pass the MSB of the new value
552             if (sign(pitchVoltTemp) == 1)
553                 temp = dec2bin(pitchVoltTemp,15);
554                 for i = 1:8
555                     if (i == 1)
556                         tempVoltBin(i) = '1';
557                     else
558                         tempVoltBin(i) = temp(i - 1);
559                     end
560                 end
561                 readyMOSI = tempVoltBin;
562                 % If the desired voltage is negative, find the 2's complement ...
563                 % value of
564                 % the voltage
565                 elseif (sign(pitchVoltTemp) == -1)
566                     temp = dec2bin(-1*pitchVoltTemp,15);
567                     % Find the complement
568                     for i = 1:15
569                         if (temp(i) == '1')
570                             complement(i) = '0';
571                         else
572                             complement(i) = '1';
573                         end
574                     end
575                     temp1 = bin2dec(complement) + 1;
576                     temp2 = dec2bin(temp1,15);
577                     for i = 1:8
578                         if (i == 1)
579                             tempVoltBin(i) = '1';
580                         else
581                             tempVoltBin(i) = temp2(i - 1);
582                         end
583                     end
584                     readyMOSI = tempVoltBin;
585                     % If the desired voltage is zero, don't activate the motor
586                     else
587                         readyMOSI = dec2bin(0,8);
588                     end
589
590             case 19
591             % BYTE 19
592             % MOSI DATA - MOTOR 0 COMMAND (7-0)

```

```

591 % MISO DATA - CURRENT SENSE 1 (15-8)
592 SS = 0;
593 % Convert the desired voltage to a value between -999 and 999
594 % 24 is the saturation level in the model
595 pitchVoltTemp = ceil((999*pitchVolt)/24);
596 % If the voltage is positive, pass the LSB
597 if (sign(pitchVoltTemp) == 1)
598     temp = dec2bin(pitchVoltTemp,15);
599     readyMOSI = temp(8:15);
600 % If the voltage is negative, find the 2's complement value and then
601 % pass the LSB
602 elseif (sign(pitchVoltTemp) == -1)
603     temp = dec2bin(-1*pitchVoltTemp,15);
604     % Find the complement
605     for i = 1:15
606         if (temp(i) == '1')
607             complement(i) = '0';
608         else
609             complement(i) = '1';
610         end
611     end
612     temp1 = bin2dec(complement) + 1;
613     temp2 = dec2bin(temp1,15);
614     readyMOSI = temp2(8:15);
615 % If the desired voltage is zero, do not activate the motor
616 else
617     readyMOSI = dec2bin(0,8);
618 end
619
620 case 20
621 % BYTE 20
622 % MOSI DATA - MOTOR 1 COMMAND (15-8)
623 % MISO DATA - CURRENT SENSE 1 (7-0)
624 SS = 0;
625 % Convert the desired voltage to a value between -999 and 999
626 % 24 is the saturation level in the model
627 yawVoltTemp = ceil((999*yawVolt)/24);
628 % If the desired voltage is positive, concatenate a '1' to the ...
629 % front of
630 % the voltage value
631 % Pass the MSB of the new value
632 if (sign(yawVoltTemp) == 1)
633     temp = dec2bin(yawVoltTemp,15);
634     for i = 1:8
635         if (i == 1)
636             tempVoltBin(i) = '1';
637         else
638             tempVoltBin(i) = temp(i - 1);
639         end
640     end
641     readyMOSI = tempVoltBin;
642 % If the desired voltage is negative, find the 2's complement ...
643 % value of
644 % the voltage

```

```

643     elseif (sign(yawVoltTemp) == -1)
644         temp = dec2bin(-1*yawVoltTemp,15);
645         % Find the complement
646         for i = 1:15
647             if (temp(i) == '1')
648                 complement(i) = '0';
649             else
650                 complement(i) = '1';
651             end
652         end
653         temp1 = bin2dec(complement) + 1;
654         temp2 = dec2bin(temp1,15);
655         for i = 1:8
656             if (i == 1)
657                 tempVoltBin(i) = '1';
658             else
659                 tempVoltBin(i) = temp2(i - 1);
660             end
661         end
662         readyMOSI = tempVoltBin;
663         % If the desired voltage is zero, don't activate the motor
664     else
665         readyMOSI = dec2bin(0,8);
666     end
667
668 case 21
669 % BYTE 21
670 % MOSI DATA — MOTOR 1 COMMAND (7–0)
671 % MISO DATA — TACHOMETER 0 (23–16)
672 SS = 0;
673 % Convert the desired voltage to a value between -999 and 999
674 % 24 is the saturation level in the model
675 yawVoltTemp = ceil((999*yawVolt)/24);
676 % If the voltage is positive, pass the LSB
677 if (sign(yawVoltTemp) == 1)
678     temp = dec2bin(yawVoltTemp,15);
679     readyMOSI = temp(8:15);
680 % If the voltage is negative, find the 2's complement value and then
681 % pass the LSB
682 elseif (sign(yawVoltTemp) == -1)
683     temp = dec2bin(-1*yawVoltTemp,15);
684     for i = 1:15
685         if (temp(i) == '1')
686             complement(i) = '0';
687         else
688             complement(i) = '1';
689         end
690     end
691     temp1 = bin2dec(complement) + 1;
692     temp2 = dec2bin(temp1,15);
693     readyMOSI = temp2(8:15);
694 % If the desired voltage is zero, do not activate the motor
695 else
696     readyMOSI = dec2bin(0,8);

```

```
697     end
698
699 case 22
700 % BYTE 22
701 % MOSI DATA — SET ENCODER 0 (23–16)
702 % MISO DATA — TACHOMETER 0 (15–8)
703 SS = 0;
704 readyMOSI = dec2bin(0,8);
705
706 case 23
707 % BYTE 23
708 % MOSI DATA — SET ENCODER (7–0)
709 % MISO DATA — TACHOMETER 0 (7–0)
710 SS = 0;
711 readyMOSI = dec2bin(0,8);
712
713 case 24
714 % BYTE 24
715 % MOSI DATA — SET ENCODER 0 (7–0)
716 % MISO DATA — TACHOMETER 1 (23–16)
717 SS = 0;
718 readyMOSI = dec2bin(0,8);
719
720 case 25
721 % BYTE 25
722 % MOSI DATA — SET ENCODER 1 (23–16)
723 % MISO DATA — TACHOMETER 1 (15–8)
724 SS = 0;
725 readyMOSI = dec2bin(0,8);
726
727 case 26
728 % BYTE 26
729 % MOSI DATA — SET ENCODER 1 (15–8)
730 % MISO DATA — TACHOMETER 1 (7–0)
731 SS = 0;
732 readyMOSI = dec2bin(0,8);
733
734 case 27
735 % BYTE 27
736 % MOSI DATA — SET ENCODER 1 (7–0)
737 % MISO DATA — STATUS
738 SS = 0;
739 readyMOSI = dec2bin(0,8);
740
741 case 28
742 % BYTE 28
743 % MOSI DATA — PADDING BYTE (0X00)
744 % MISO DATA — ENCODER 0 (23–16)
745 SS = 0;
746 readyMOSI = dec2bin(0,8);
747
748 case 29
749 % BYTE 29
750 % MOSI DATA — PADDING BYTE (0X00)
```

```
751 % MISO DATA - ENCODER 0 (15-8)
752 SS = 0;
753 readyMOSI = dec2bin(0,8);
754
755 case 30
756 % BYTE 30
757 % MOSI DATA - PADDING BYTE (0X00)
758 % MISO DATA - ENCODER 0 (7-0)
759 SS = 0;
760 readyMOSI = dec2bin(0,8);
761
762 case 31
763 % BYTE 31
764 % MOSI DATA - PADDING BYTE (0X00)
765 % MISO DATA - ENCODER 1 (23-16)
766 SS = 0;
767 readyMOSI = dec2bin(0,8);
768
769 case 32
770 % BYTE 32
771 % MOSI DATA - PADDING BYTE (0X00)
772 % MISO DATA - ENCODER 1 (15-8)
773 SS = 0;
774 readyMOSI = dec2bin(0,8);
775
776 case 33
777 % BYTE 33
778 % MOSI DATA - PADDING BYTE (0X00)
779 % MISO DATA - ENCODER 1 (7-0)
780 SS = 0;
781 readyMOSI = dec2bin(0,8);
782
783 case 34
784 % BYTE 34
785 % MOSI DATA - PADDING BYTE (0X00)
786 % MISO DATA - ACCELEROMETER X (15-8)
787 SS = 0;
788 readyMOSI = dec2bin(0,8);
789
790 case 35
791 % BYTE 35
792 % MOSI DATA - PADDING BYTE (0X00)
793 % MISO DATA - ACCELEROMETER X (7-0)
794 SS = 0;
795 readyMOSI = dec2bin(0,8);
796
797 case 36
798 % BYTE 36
799 % MOSI DATA - PADDING BYTE (0X00)
800 % MISO DATA - ACCELEROMETER Y (15-8)
801 SS = 0;
802 readyMOSI = dec2bin(0,8);
803
804 case 37
```

```

805    % BYTE 37
806    % MOSI DATA — PADDING BYTE (0X00)
807    % MISO DATA — ACCELEROMETER Y (7–0)
808    SS = 0;
809    readyMOSI = dec2bin(0,8);

810
811    case 38
812    % BYTE 38
813    % MOSI DATA — PADDING BYTE (0X00)
814    % MISO DATA — ACCELEROMETER Z (15–8)
815    SS = 0;
816    readyMOSI = dec2bin(0,8);

817
818    case 39
819    % BYTE 39
820    % MOSI DATA — PADDING BYTE (0X00)
821    % MISO DATA — ACCELEROMETER Z (7–0)
822    SS = 0;
823    readyMOSI = dec2bin(0,8);

824
825    case 40
826    % BYTE 40
827    % MOSI DATA — PADDING BYTE (0X00)
828    % MISO DATA — GYROSCOPE X (15–8)
829    SS = 0;
830    readyMOSI = dec2bin(0,8);

831
832    case 41
833    % BYTE 41
834    % MOSI DATA — PADDING BYTE (0X00)
835    % MISO DATA — GYROSCOPE X (7–0)
836    SS = 0;
837    readyMOSI = dec2bin(0,8);

838
839    case 42
840    % BYTE 42
841    % MOSI DATA — PADDING BYTE (0X00)
842    % MISO DATA — GYROSCOPE Y (15–8)
843    SS = 0;
844    readyMOSI = dec2bin(0,8);

845
846    case 43
847    % BYTE 43
848    % MOSI DATA — PADDING BYTE (0X00)
849    % MISO DATA — GYROSCOPE Y (7–0)
850    SS = 0;
851    readyMOSI = dec2bin(0,8);

852
853    case 44
854    % BYTE 44
855    % MOSI DATA — PADDING BYTE (0X00)
856    % MISO DATA — GYROSCOPE Z (15–8)
857    SS = 0;
858    readyMOSI = dec2bin(0,8);

```

```

859
860     case 45
861         % BYTE 45
862         % MOSI DATA — PADDING BYTE (0X00)
863         % MISO DATA — GYROSCOPE Z (7–0)
864         SS = 0;
865         readyMOSI = dec2bin(0,8);
866
867     case 46
868         % BYTE 46
869         % MOSI DATA — PADDING BYTE (0X00)
870         % MISO DATA — RESERVED (0X00)
871         SS = 0;
872         readyMOSI = dec2bin(0,8);
873
874     case 47
875         % BYTE 47
876         % MOSI DATA — PADDING BYTE (0X00)
877         % MISO DATA — RESERVED (0X00)
878         SS = 0;
879         readyMOSI = dec2bin(0,8);
880
881     case 48
882         % BYTE 48
883         % MOSI DATA — PADDING BYTE (0X00)
884         % MISO DATA — RESERVED (0X00)
885         SS = 0;
886         readyMOSI = dec2bin(0,8);
887
888     case 49
889         % BYTE 49
890         % MOSI DATA — PADDING BYTE (0X00)
891         % MISO DATA — RESERVED (0X00)
892         SS = 0;
893         readyMOSI = dec2bin(0,8);
894
895     case 50
896         % BYTE 50
897         % MOSI DATA — PADDING BYTE (0X00)
898         % MISO DATA — RESERVED (0X00)
899         readyMOSI = dec2bin(0,8);
900         % Reset the slave select to begin the process again
901         % Maybe not needed
902         SS = 1;
903
904     otherwise
905         % Just in case
906         readyMOSI = dec2bin(0,8);
907         SS = 0;
908 end
909 % Pass the byte we have derived bit by bit
910 % Use the character array to determine the bit value
911 switch bitNumber
912     case 1

```

```

913         if (readyMOSI(1) == '1')
914             MOSI = 1;
915         else
916             MOSI = 0;
917         end
918     case 2
919         if (readyMOSI(2) == '1')
920             MOSI = 1;
921         else
922             MOSI = 0;
923         end
924     case 3
925         if (readyMOSI(3) == '1')
926             MOSI = 1;
927         else
928             MOSI = 0;
929         end
930     case 4
931         if (readyMOSI(4) == '1')
932             MOSI = 1;
933         else
934             MOSI = 0;
935         end
936     case 5
937         if (readyMOSI(5) == '1')
938             MOSI = 1;
939         else
940             MOSI = 0;
941         end
942     case 6
943         if (readyMOSI(6) == '1')
944             MOSI = 1;
945         else
946             MOSI = 0;
947         end
948     case 7
949         if (readyMOSI(7) == '1')
950             MOSI = 1;
951         else
952             MOSI = 0;
953         end
954     case 8
955         if (readyMOSI(8) == '1')
956             MOSI = 1;
957         else
958             MOSI = 0;
959         end
960         % If we have just sent the last bit, update the byte number
961         byteNumber = byteNumber + 1;
962         % Reset back to zero if we have finished one transfer
963         if (byteNumber == 51)
964             byteNumber = 0;
965         end
966     otherwise

```

```
967         MOSI = 0;
968 end
969 % Update the bit number that we are sending
970 bitNumber = bitNumber + 1;
971 % Reset back to 1 if we have finished one byte
972 if (bitNumber == 9)
973     bitNumber = 1;
974 end
975 end
```

# Appendix D

## ADP MATLAB Code

### D.1 Initialization Code

```
1 %Glenn Janiak & Ken Vonckx
2 close all;
3 clear;
4 clc;
5
6 quanser_aero_parameters;
7 quanser_aero_state_space;
8
9 % Sampling time [s]
10 tau = 0.01;
11 % Update time [s]
12 T = 1;
13
14 % COST FUNCTION
15 % Q and R matrices used in the cost function
16 Q_Mat_ADP = diag([270 100 1 1]);
17 R_Mat_ADP = 0.005*diag([1 1]);
18
19 [n,~] = size(B);
20
21 nbar = T/tau;      % T/tau = 20
22
23 e_vec = (2*(pi)).*rand(n,nbar) - (pi);
24
25 fbar = @(e) A*e;
26 gbar = -B;
27
28 f = @(e) fbar(e)*tau + e;
29 g = gbar*tau;
30
31 Qbar = @(e) e'*Q_Mat_ADP*e;
32 Rbar = R_Mat_ADP;
33
```

```

34 Q = @(e) Qbar(e)*tau;
35 R = Rbar*tau;
36
37
38 rho = @(e) [e(1); e(2); e(3); e(4); ...
39             e(1)^2; e(1)*e(2); e(1)*e(3); e(1)*e(4); ...
40             e(2)^2; e(2)*e(3); e(2)*e(4); e(3)^2; e(3)*e(4); e(4)^2];
41
42 drhode =@(e) [1, 0, 0, 0;
43                 0, 1, 0, 0;
44                 0, 0, 1, 0;
45                 0, 0, 0, 1;
46                 2*e(1), 0, 0, 0;
47                 e(2), e(1), 0, 0;
48                 e(3), 0, e(1), 0;
49                 e(4), 0, 0, e(1);
50                 0, 2*e(2), 0, 0;
51                 0, e(3), e(2), 0;
52                 0, e(4), 0, e(2);
53                 0, 0, 2*e(3), 0;
54                 0, 0, e(4), e(3);
55                 0, 0, 0, 2*e(4)];
56
57 % Learning Rate
58 EpsilonPolicy = 0.1;
59 EpsilonWcritic = 0.1;
60
61 % Number of outer loop iterations
62 outerLoopMax = 700;
63 % Number of inner loop iterations
64 innerLoopMax = 100;
65
66 % Number of training samples
67 [~,nbar] = size(e_vec);
68
69 % WEIGHT INITIALIZATION
70 % Initialize the weights of the critic neural network to zero
71 WcLast = zeros(length(rho(e_vec(:,1))),1);
72
73 % LEAST-SQUARES COMPUTATION INITIALIZATION
74 % Matrices required for computing least squares weights of the critic
75 % neural networks -- EQ 20
76 V = zeros(nbar,1);
77 Lambda = zeros(nbar,length(rho(e_vec(:,1)))); 
78
79 % Matrix to hold the derivative of the error model during policy
80 % updating
81 e_k_plus_1 = zeros(n,nbar);
82
83 % Product of the least squares matrices must be invertible
84 % Logic flag indicating if the critic weights are unsolvable
85 % The weights are unsolvable because the least squares matrices have no
86 % solution -- not invertible
87 diverged = 0;

```

```

88
89 % OUTER LOOP
90 for i = 1:(outerLoopMax-1)
91 % Determine if the least squares matrices are invertible
92 if diverged == 0
93     % For each of the data collection (discrete time index)
94     for k = 1:nbar
95         % Initialize the optimal inputs to zero
96         uNew = [0; 0];
97         % INNER LOOP
98         for j = 1:(innerLoopMax-1)
99             % Get the updated input value
100            uLast = uNew;
101            % Update the error model
102            e_k_plus_1(:,k) = f(e_vec(:,k)) + g*uLast;
103            % Compute the new optimal inputs
104            uNew = -0.5*(R^(-1))*g'*drhode(e_k_plus_1(:,k))'*WcLast;
105
106            % Check convergence of the optimal inputs
107            if norm(uNew - uLast) < EpsilonPolicy
108                break;
109            end
110        end
111
112        % Update the values for the least-squares computation
113        V(k,:) = Q(e_vec(:,k)) + uNew'*R*uNew + ...
114            WcLast'*rho(e_k_plus_1(:,k));
115        Lambda(k,:) = rho(e_vec(:,k))';
116    end
117
118    % Verify the least square solution exists for the critic's weights
119    % If the error data is consistent or there is no error, this will not
120    % hold, so set the weights to zero
121    if det(Lambda'*Lambda) == 0
122        weights = zeros(length(rho(e_vec(:,1))),1);
123        break;
124    end;
125
126    % Calculate least squares solution of critic's weights -- EQ 20
127    WcNew = (Lambda'*Lambda)^(-1)*Lambda'*V;
128    % Make sure the weights did not diverge
129    % If the weights are diverging, just set them to a large number
130    if isnan(WcNew)
131        weights = 1000*ones(length(rho(e_vec(:,1))),1);
132        break;
133    end;
134
135    % Check for convergence of the critic weights
136    if norm(WcNew - WcLast) < EpsilonWcritic
137        weights = WcNew;
138        break;
139    end
140    % If the weights did not converge, repeat the loop

```

```

141 WcLast = WcNew;
142
143 % If we reached the last iteration of the loop, just use the last
144 % weights found
145 if (i == (outerLoopMax-1))
146     weights = WcNew;
147 end
148 end
149
150 % Use the weights to determine the P matrix
151 P_Mat = [weights(5) weights(6) weights(7) weights(8);
152           weights(6) weights(9) weights(10) weights(11);
153           weights(7) weights(10) weights(12) weights(13);
154           weights(8) weights(11) weights(13) weights(14)];
155
156 % Find the state-feedback gain EQ
157 K = 0.5*(R_Mat_ADP^-1)*B'*P_Mat;
158
159 % Save the initial weights
160 wcInit = weights;
161
162 % Keep only the ADP gain
163 clearvars -except K tau T wcInit;

```

## D.2 Update Weights

```

1 % CRITIC WEIGHT TUNING NEURAL NETWORK
2 % This function is specific to the helicopter because of the number of
3 % weights and error model
4 % THIS FUNCTION CANNOT HAVE ANY ANONYMOUS FUNCTIONS DUE TO THE C CODE
5 % GENERATION UNSUPPORTING IT
6 function K = ...
    quanserAEROCriticTuning(xd,pitchData,yawData,pitchDotData,yawDotData,wcInit)
7     % CREATE THE ERROR VECTOR MATRIX
8     % Use the data from the tapped delay blocks to create the error state
9     % vector matrix
10    e_vec = [pitchData'; yawData'; pitchDotData'; yawDotData'];
11
12    % REFERENCE DR. MIAH'S PAPER FOR EQUATION NUMBERS
13    % Sampling time [s]
14    tau = 0.05;
15
16    % COST FUNCTION
17    % Q and R matrices used in the cost function
18    Q_Mat = diag([270 100 1 1]);
19    R_Mat = 0.005*diag([1 1]);
20
21    % SYSTEM PARAMETERS SPECIFIC TO THE 2-DOF QUANSER AERO
22    A = [0 0 1 0; 0 0 0 1; -1.7442 0 -0.3307 0; 0 0 0 -0.9283];
23    B = [0 0; 0 0; -0.0149 0.0414; -0.0751 -0.1295];

```

```

1 % UNCOMMENT TO USE A STATE-SPACE MODEL THAT IS NOT THE CORRECT ...
2 % ONE DERIVED
3 % A = [1 1 1 1; 1 1 1 1; 1 1 1 1; 1 1 1 1];
4 % B = [0 0; 0 0; 0.1 0.1; -0.3 -0.3];
5 % System dimensions specific for our model
6 [n,~] = size(B);
7
8 % ERROR MODEL OF THE HELICOPTER
9 % gbar and hbar — EQ 8
10 % An anonymous function cannot be used for fbar
11 gbar = -B;
12 hbar = -A*xd;
13
14 % DISCRETE-TIME ERROR MODEL FOR TIME TAU
15 % g and h — EQ 10
16 % An anonymous function cannot be used for f
17 g = gbar*tau;
18 h = hbar*tau;
19
20 % COST FUNCTION PARAMETERS
21 % The Q matrix cannot be treated as an anonymous function as in the
22 % MATLAB simulations
23 % Control penalizing matrix in the continuous cost function
24 Rbar = R_Mat;
25 % The discrete-time cost function will have terms:
26 % Right after EQ 11 in paper
27 % Control penalizing matrix in the discretized cost function
28 R = Rbar*tau;
29
30 % NEURAL NETWORK FUNCTIONS
31 % These functions cannot be written as anonymous functions as in the
32 % MATLAB simulations
33
34 % TOLERANCES
35 % Convergence tolerance for control policy
36 EpsilonPolicy = 0.1;
37 % Convergence tolerance for critic neural network
38 EpsilonWcritic = 0.1;
39
40 % TRAINING PARAMETERS
41 % Number of outer loop iterations
42 outerLoopMax = 700;
43 % Number of inner loop iterations
44 innerLoopMax = 100;
45 % Number of equations needed for training, number of sub-intervals
46 % Number of training samples
47 [~,nbar] = size(e_vec);
48
49 % WEIGHT INITIALIZATION
50 % Initialize the weights of the critic neural network to zero
51 % Number of rows hard-coded for the helicopter
52 WcLast = zeros(14,1);
53
54 % LEAST-SQUARES COMPUTATION INITIALIZATION

```

```

77 % Matrices required for computing least squares weights of the critic
78 % neural networks -- EQ 20
79 V = zeros(nbar,1);
80 Lambda = zeros(nbar,14);
81
82 % Matrix to hold the derivative of the error model during policy
83 % updating
84 e_k_plus_1 = zeros(n,nbar);
85
86 % Product of the least squares matrices must be invertible
87 % Logic flag indicating if the critic weights are unsolvable
88 % The weights are unsolvable because the least squares matrices ...
89 % have no
90 % solution -- not invertible
91 diverged = 0;
92
93 % OUTER LOOP
94 for i = 1:(outerLoopMax-1)
95 % Determine if the least squares matrices are invertible
96 if diverged == 0
97     % For each of the data collection (discrete time index)
98     for k = 1:nbar
99         % Initialize the optimal inputs to zero
100        uNew = [0; 0];
101        % INNER LOOP
102        for j = 1:(innerLoopMax-1)
103            % Get the updated input value
104            uLast = uNew;
105            % Update the error model
106            % Because of no anonymous functions we have modified f
107            % fbar = @(e) A*e;
108            % f = @(e) fbar(e)*tau + e;
109            e_k_plus_1(:,k) = (A*e_vec(:,k)*tau) + e_vec(:,k) + ...
110                g*uLast + h;
111            % Compute the new optimal inputs
112            % Partial derivative of rho with respect to e
113            drhode = [1, 0, 0, 0;
114                      0, 1, 0, 0;
115                      0, 0, 1, 0;
116                      0, 0, 0, 1;
117                      2*e_vec(1,k), 0, 0, 0;
118                      e_vec(2,k), e_vec(1,k), 0, 0;
119                      e_vec(3,k), 0, e_vec(1,k), 0;
120                      e_vec(4,k), 0, 0, e_vec(1,k);
121                      0, 2*e_vec(2,k), 0, 0;
122                      0, e_vec(3,k), e_vec(2,k), 0;
123                      0, e_vec(4,k), 0, e_vec(2,k);
124                      0, 0, 2*e_vec(3,k), 0;
125                      0, 0, e_vec(4,k), e_vec(3,k);
126                      0, 0, 0, 2*e_vec(4,k)];
127            uNew = -0.5*(R^(-1))*g'*drhode'*WcLast;
128
129            % Check convergence of the optimal inputs
130            if norm(uNew - uLast) < EpsilonPolicy

```

```

129             break;
130         end
131     end
132
133     % Update the values for the least-squares computation
134     % Critic neural network activation functions
135     rhoE = [e_vec(1,k); e_vec(2,k); e_vec(3,k); e_vec(4,k); ...
136             e_vec(1,k)^2; e_vec(1,k)*e_vec(2,k); ...
137             e_vec(1,k)*e_vec(3,k); e_vec(1,k)*e_vec(4,k); ...
138             e_vec(2,k)^2; e_vec(2,k)*e_vec(3,k); ...
139             e_vec(2,k)*e_vec(4,k); e_vec(3,k)^2; ...
140             e_vec(3,k)*e_vec(4,k); e_vec(4,k)^2];
141     rhoEK = [e_k_plus_1(1,k); e_k_plus_1(2,k); e_k_plus_1(3,k); ...
142             e_k_plus_1(4,k); e_k_plus_1(1,k)^2; ...
143             e_k_plus_1(1,k)*e_k_plus_1(2,k); ...
144             e_k_plus_1(1,k)*e_k_plus_1(3,k); ...
145             e_k_plus_1(1,k)*e_k_plus_1(4,k); ...
146             e_k_plus_1(2,k)^2; ...
147             e_k_plus_1(2,k)*e_k_plus_1(3,k); ...
148             e_k_plus_1(2,k)*e_k_plus_1(4,k); ...
149             e_k_plus_1(3,k)^2; ...
150             e_k_plus_1(3,k)*e_k_plus_1(4,k); e_k_plus_1(4,k)^2];
151
152     % State penalizing function in the continuous cost function
153     % Qbar = @(e) e'*Q_Mat*e;
154     % State penalizing function in the discretized cost function
155     % Q = @(e) Qbar(e)*tau;
156     V(k,:) = (e_vec(:,k)'*Q_Mat*e_vec(:,k)*tau) + ...
157             uNew'*R*uNew + WcLast'*rhoEK;
158     Lambda(k,:) = rhoE';
159
160     end
161 end
162
163 % Verify the least square solution exists for the critic's weights
164 % If the error data is consistent or there is no error, this will not
165 % hold, so set the weights to what they were initially before the
166 % simulation
167 if det(Lambda'*Lambda) == 0
168     weights = wcInit;
169     break;
170 end;
171
172 % Calulcate least squares solution of critic's weights — EQ 20
173 WcNew = (Lambda'*Lambda)^(-1)*Lambda'*V;
174 % Make sure the weights did not diverge
175 % If the weights diverged, set the weights to what they were ...
176     % initially
177 % before the simulation
178 if isnan(WcNew)
179     weights = wcInit;
180     break;
181 % Check for convergence of the critic weights
182 elseif norm(WcNew - WcLast) < EpsilonWcritic
183     weights = WcNew;
184     break;
185

```

```
179 % If we reach the last iteration of the loop, just use the last ...
180 % weights
181 elseif (i == (outerLoopMax-1))
182     weights = WcNew;
183 % If all else fails, just set the weights to the initial weights
184 else
185     weights = wcInit;
186 end
187 % If the weights did not converge, do another iteration of the loop
188 WcLast = WcNew;
189 end
190 % Use the weights to determine the P matrix
191 P_Mat = [weights(5) weights(6) weights(7) weights(8);
192           weights(6) weights(9) weights(10) weights(11);
193           weights(7) weights(10) weights(12) weights(13);
194           weights(8) weights(11) weights(13) weights(14)];
195
196 % Find the state-feedback gain EQ
197 K = 0.5*(R_Mat^-1)*B'*P_Mat;
198 end
```

# Appendix E

## Tutorials

### E.1 USB Connection

1. Verify QuaRC is installed on computer
2. Install QFLEX 2 USB into Quanser Aero as seen in figure [E.1](#)
3. Open Simulink model
4. Connect Quanser Aero to computer via the included USB adapter cable
5. Set simulation mode to external
6. Run MATLAB initialization code for motion controller
7. Click "Build Model" button
8. Click "Connect To Target" button
9. Click "Run" button

### E.2 Raspberry Pi Implementation

1. Make sure software add ons are installed on MATLAB
  - (a) MATLAB Support Package for Raspberry Pi Hardware
  - (b) Simulink Support Package for Raspberry Pi Hardware
2. Install QFLEX 2 EMBEDDED into Quanser Aero as seen in figure [E.2](#)
3. Assuming Raspberry PI is not connected to network, establish a serial connection to the Raspberry PI



Figure E.1: Quanser Aero with QFLEX 2 USB panel installed

- (a) Plug SD card into Linux computer and enable serial connectivity
  - (b) Insert SD card back into Raspberry Pi
  - (c) Connect serial cable to the Raspberry Pi, Pin 3 (black wire), Pin 4 (white wire), Pin 5 (green wire)
  - (d) Plug Raspberry Pi into computer
  - (e) Locate COM port for the connection
  - (f) Open Putty, select Serial connection, input COM number and 115200 as the speed
  - (g) Log into Raspberry Pi
  - (h) Configure network, type "sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf" and add network settings
4. Connect SPI wires to Raspberry PI. Pin 1 (white wire) on the embedded panel connects to +5V on the Pi. Pin 2 (yellow wire) connects to GPIO 10 on the Pi. Pin 3 (blue wire) connects to GPIO 9 on the Pi. Pin 4 (green wire) connects to GPIO 11 on the Pi. Pin 6 (purple wire) connects to GPIO 8 on the Pi. Pin 7 (Red wire) connects to ground on the Pi.
  5. Connect Raspberry PI to QFLEX 2 Embedded
  6. Right click on Simulink model
  7. Open "Model Configuration Parameters"
  8. Select "Hardware Implementation"
  9. Set "Hardware Board" to "Raspberry Pi"
  10. Under "Board Parameters" under "Groups" type in the IP address of the Raspberry PI, the username, and password

11. Under "Build options", set build action to "Build and run" and type in the file path that you want the model to be stored on the Raspberry PI
12. Set simulation mode to external
13. Run MATLAB initialization code for motion controller
14. Run MATLAB Raspberry PI/SPI initialization code in section [D.2](#)
15. Click "Deploy to Hardware" button
16. To stop program you must "kill" it
17. Open Putty, connect to your Raspberry PI, and log in
18. Type "ps -A" to view running programs and find the process number of your Simulink model
19. Type "sudo kill -9 #####" where ##### is your process number. This will kill the program
20. To run a file already on the Raspberry Pi use the command "sudo FILEPATH/FILE-NAME.elf", if file is in current directory use "sudo ./FILENAME.elf"
21. If the model runs to time infinity when started from the Raspberry Pi you can use Control-C to stop the model



Figure E.2: Quanser Aero with QFLEX 2 Embedded panel installed

## E.3 Android Application

1. Make sure software add ons are installed on MATLAB, "Simulink Support Package for Android Devices"
2. Install Android Studio

3. Repeat steps from appendix [E.2](#) to set up Raspberry Pi and push model to Raspberry Pi
4. From MATLAB open add on manager and click on the "Setup" gear next to the Android package
5. Verify Android Studio installed, Next
6. Verify Android SDK Tools installed
7. NOTE: The recent version of Android Studio removed one of their files that MATLAB needs to build android applications. Fixed by downloading and installing an older verison of Android Studio NDK bundle (December 2017) and copying over the missing folder into the file path that MATLAB was calling. This issue might be fixed in a newer version of Android Studio.
8. Click next
9. Follow intrustions on prompts to turn android on in developer mode
10. Turn on USB debugging
11. Install driver
12. Connect android to computer with USB cable
13. Allow USB debugging
14. Make sure both computer and android are on same network with internet access
15. Select device
16. Build and run Test App, App should close and delete itself after test is complete
17. Open simulink model's configuration parameters
18. Under "Hardware Implementation" set "Hardware Board" to "Android Device"
19. Build and deploy simulink model to android
20. Once APP is running, swtich network to same network as the Raspberry Pi
21. Execute code on Raspberry Pi
22. Control helicopter

# Bibliography

- [1] Q. Ahmed, A. I. Bhatti, S. Iqbal, and I. H. Kazmi. 2-sliding mode based robust control for 2-dof helicopter. In *2010 11th International Workshop on Variable Structure Systems (VSS)*, pages 481–486, June 2010.
- [2] W. Chang, J.H. Moon, and H.J. Lee. Fuzzy model-based output-tracking control for 2 degree-of-freedom helicopter. *Journal of Electrical Engineering Technology*, 12.00(1):1921–1928, 2017. Quanser product(s): 2 DOF Helicopter.
- [3] Andrew Fandel, Anthony Birge, and Suruz Miah. Development of reinforcement learning algorithm for 2-dof helicopter model. In *IEEE International Symposium on Industrial Electronics*, Cairns, Australia, June 2018.
- [4] W. Gao and Z. P. Jiang. Data-driven adaptive optimal output-feedback control of a 2-dof helicopter. In *2016 American Control Conference (ACC)*, pages 2512–2517, July 2016.
- [5] M. Hernandez-Gonzalez, A.Y. Alanis, and E.A. Hernandez-Vargas. Decentralized discrete-time neural control for a quanser 2-dof helicopter. In *Applied Soft Computing*, pages 2462–2469, February 2012.
- [6] E. Kayacan and M.A. Khanesar. Recurrent interval type-2 fuzzy control of 2-dof helicopter with finite time training algorithm. In *IFAC-PapersOnLine*, pages 293–299, July 2016.
- [7] C. Prez-D'Arpino, W. Medina-Melndez, L. Fermn-Len, J. M. Bogado, R. R. Torrealba, and G. Fernndez-Lpez. Generalized bilateral mimo control by states convergence with time delay and application for the teleoperation of a 2-dof helicopter. In *2010 IEEE International Conference on Robotics and Automation*, pages 5328–5333, May 2010.
- [8] R.G. Subramanian and V.K. Elumalai. Robust mrac augmented baseline lqr for tracking control of 2-dof helicopter. In *Robotics and Autonomous Systems*, pages 70–77, August 2016.