Spring 5-2016

Design and Implementation of a Controller for a BeagleBone Quadcopter

Peter Olejnik
Rose-Hulman Institute of Technology, olejnipr@rose-hulman.edu

Design and Implementation of a Controller for a BeagleBone Quadcopter

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

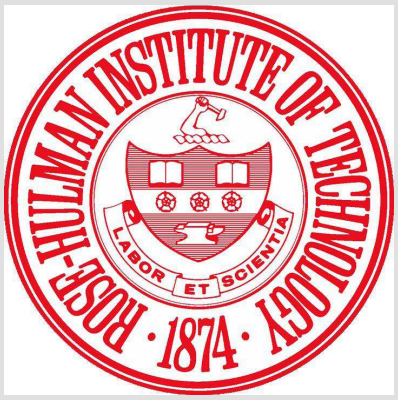
Peter Olejnik

In Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Mechanical Engineering

May 2016



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Examination Report

Peter R. Olejnik — Name

Mechanical Engineering — Graduate Major

Thesis Title **Design and Implementation of a Controller for a BeagleBone Quadcopter**

DATE OF EXAM: **May 27, 2016**

EXAMINATION COMMITTEE:

	Thesis Advisory Committee	Department
Thesis Advisor:	**Bradley Burchett**	**ME**
	David Fisher	**ME**
	Mark Yoder	**ECE**

PASSED X **FAILED** _____

Abstract

Unmanned aerial vehicles are quickly becoming a significant and permanent feature in today's world of aviation. Amongst the various types of UAVs, a popular type is the quadcopter. Also referred to as a quadrotor, this rotor craft's defining feature is that it has four propellers. While its use is common in the hobbyist community, this aircraft's use within industry is blooming.

Presented are the efforts to design and implement a controller for a BeagleBone based quadcopter. As part of this effort, characteristics of the quadcopter were experimentally determined. These characteristics consist of physical properties of the quadcopter, such as the moments of inertia, the motor performance characteristics, and variance within its sensors. A model was then created and implemented within a MATLAB environment to simulate the flight of the quadcopter. With a simulated environment created, a controller was designed to control the flight of the quadcopter and a Kalman Filter was implemented to filter a sensor input. These designs were then verified in the simulated environment.

Acknowledgements

Over the course of my studies and work, I was fortunate and privileged to have been surrounded by people who made sure that my efforts did not fail in the most spectacular of manners. Without hesitation, they were willing to render assistance and support me when I needed it most. Of note:

- Dr. Bradley Burchett, my ever supportive adviser who took me under his wings and who's ideas helped pave the road to my success.
- Mike Fulk, Ron Hofmann, and Jerry Leturgez, for always supporting me and whose consistent assistance in running my ever extravagant experiments made sure that I could run them and that they would results in successes
- Mark Crosby, Gary Meyer, and Jack Shrader, for imparting your collective knowledge and assisting me in matters dealing with electrical engineering, all which has allowed me to perform feats which would make a real electrical engineer proud
- The Learning Center and its staff, for proof reading this work and suggesting improvements
- The whole of the Rose-Hulman faculty, for imparting your vast collective knowledge and wisdom to me over the years, as well as providing the constant tidbits of advice which has allowed this work to take flight
- My friends, my family, and all others who have supported me throughout this entire endeavor, both intellectually and, more importantly, morally

To you all, you have and will always have my sincerest appreciation and deepest gratitude. Without you all, the work I present here would have not been possible.

Thank you

Our duty is to further the course of humanity.

Table of Contents

Abstract **v**

Acknowledgements **vii**

Table of Contents **xi**

List Of Figures **xvii**

List Of Tables **xxi**

Abbreviations **xxiii**

Symbols and Notation **xxv**

1 An Introduction to Unmanned Aerial Vehicles **1**

1.1 Introduction 1

1.2 A Brief Overview of UAVs 1

1.3 A More Detailed Look Into Quadcopters 3

2 The Moments of Inertia of the Quadcopter **5**

2.1 Introduction 5

2.2 The Mass 5

2.3 Theory Collecting the Moments of Inertia 6

2.4 Identifying the Moments of Inertia 8

3 The MPU6050 **11**

3.1 Introduction 11

3.2 Testing the Variance in the MPU6050 11

3.3 Results of Testing the Variance in the MPU6050 12

4 Motor Experimental Setup **17**

4.1 Introduction 17

4.2 Thrust Collection 17

4.3 Torque Collection 20

4.4 Test Procedure 22

5 Experimental Motor Results **25**

5.1 Introduction 25

5.2 Trimming the Data 25

5.3 Thrust 25

5.4 Torque 31

5.5 Thrust Curve Fitting 36

5.6 Torque Curve Fitting 39

5.7 Results 41

6 Quadcopter Flight Mechanics **43**

6.1 Introduction 43

6.2 Rotational Matrices 44

6.3 Translational Flight Mechanics 45

6.4 Angular Flight Mechanics 47

7 Design of the Kalman Filter **49**

7.1 Introduction 49

7.2 The Kalman Filter 49

7.3 Implementing the Extended Kalman Filter 50

8 Design of the Linear Quadratic Regulator **55**

8.1 Introduction 55

8.2 The Linear Quadratic Regulator 55

8.3 Derivation of the Discreate Linear Quadratic Regulator 56

8.4 Linearizing the Model 59

8.5 The Linearized State Space Model 62

9 Implementing the System Within a MATLAB Environment **67**

9.1 Introduction . . . 67

9.2 Implementing the the Simulation . . . 67

9.3 Simulation Results . . . 71

10 Future Works **79**

10.1 Implementing the Designs on the Quadcopter . . . 79

10.2 More Detailed Theoretical Model . . . 79

10.3 Use Different Control Techniques . . . 80

10.4 Implement a Continuous Model via an Analog Computer . . . 80

10.5 Alternative and Additional Sensors . . . 80

10.6 Refinements to the Quadcopter . . . 81

A Prepping the BeagleBone System **83**

A.1 Introduction . . . 83

A.2 Updating the OS . . . 83

A.3 Updating Programs and Libraries . . . 84

A.4 WiFi Connectivity . . . 84

A.5 Configuring Systemd . . . 84

A.6 Final Steps . . . 85

B Theory Behind Implementing the System on Hardware **87**

B.1 Introduction . . . 87

B.2 Implementation on the quadcopter . . . 87

C Back EMF Feedback **91**

C.1 Introduction . . . 91

C.2 Collection and Processing . . . 91

C.3 EMF Curve Fitting . . . 96

D Listed Quadcopter Hardware Specs **101**

D.1 BeagleBone Black . . . 101

D.2 Turnigy AX-2204C . . . 102

D.3 Exceed RC Proton 10A Electronic Speed Control System . . . 102
D.4 InvenSense MPU 6050 Accelerometer + Gyro . . . 103
D.5 GY-521 Schematic . . . 103

E BeagleBone Black Cape PCB Design 105

E.1 Pin Usage . . . 105
E.2 Cape PCB Schematic . . . 106
E.3 Cape PCB Layout . . . 107

F Experimental Hardware Specs 109

F.1 3132 Micro Load Cell (0-780g) . . . 109
F.2 LT 1167 Instrumentation Amplifier . . . 110
F.3 TL 7660 CMOS Voltage Converter . . . 110

G Experimental Setup 111

G.1 Experimental Electrical Schematic . . . 111
G.2 Apparatus CAD Models . . . 112
G.3 Apparatus CAD Drawings . . . 113

H Experimentation Code 117

H.1 Sensor Collection Code . . . 117
H.2 Sensor Analysis Code . . . 119
H.3 Load Cell Calibration . . . 121
H.4 Quanser Configuration . . . 121
H.5 Motor Analysis Code . . . 125

I Simulation Code 149

I.1 Main Simulation Code . . . 149
I.2 Data Reader . . . 163
I.3 Force to Throttle Converter . . . 165
I.4 Theoretical State Computer . . . 166
I.5 Noise Addition . . . 167
I.6 Force to Torque Converter . . . 167

I.7 Torque to Force Converter 168

J Quadcopter Code **171**

J.1 Set Up Part 1 171

J.2 Set Up Part 2 172

J.3 Quadcopter Start Service 173

J.4 DHCP Configuration 173

J.5 Hostapd Path Configuration 176

J.6 Hostapd Configuration 176

J.7 Interfaces Configuration 177

J.8 ISC-DHCP-Server Device Configuration 178

J.9 Quadcopter Functions 179

J.10 Flight Program 186

Bibliography **188**

List of Figures

1.1 A picture of the quadcopter which this thesis used as a basis. 4

2.1 A picture of the setup used to identify the period of a swing. 6
2.2 FBD and KD of the experimental set up to obtain the moments of inertia. 7
2.3 Simulink diagram which controlled the Quanser data collection. 9

3.1 Variance present in the X-axis in the MPU6050's accelerometer. 12
3.2 Variance present in the Y-axis in the MPU6050's accelerometer. 13
3.3 Variance present in the Z-axis in the MPU6050's accelerometer. 13
3.4 Variance present along the X-axis in the MPU6050's gyro. 14
3.5 Variance present along the Y-axis in the MPU6050's gyro. 14
3.6 Variance present along the Z-axis in the MPU6050's gyro. 15

4.1 Collected results from the thrust load cells. 18
4.2 Processed thrust load cell calibration. 19
4.3 Free body diagram of the mount that was used to collect the torque. 21
4.4 Simulink diagram which controlled the Quanser data collection. 23
4.5 View of a single throttle point. Note how the data reached steady state conditions only after about two seconds. 23
4.6 Picture of the setup used to collect the data. 24

5.1 Collected thrust data from motor 1. 26
5.2 Collected thrust data from motor 2. 27
5.3 Collected thrust data from motor 3. 27
5.4 Collected thrust data from motor 4. 28
5.5 Trimmed thrust data from motor 1. 29
5.6 Trimmed thrust data from motor 2. 29
5.7 Trimmed thrust data from motor 3. 30
5.8 Trimmed thrust data from motor 4. 30

5.9 Collected torque data from motor 1. 31
5.10 Collected torque data from motor 2. 32
5.11 Collected torque data from motor 3. 32
5.12 Collected torque data from motor 4. 33
5.13 Trimmed torque data from motor 1. 34
5.14 Trimmed torque data from motor 2. 34
5.15 Trimmed torque data from motor 3. 35
5.16 Trimmed torque data from motor 4. 35
5.17 Motor 1 thrust curve. 36
5.18 Motor 2 thrust curve. 37
5.19 Motor 3 thrust curve. 37
5.20 Motor 4 thrust curve. 38
5.21 Motor 1 torque curve. 39
5.22 Motor 2 torque curve. 40
5.23 Motor 3 torque curve. 40
5.24 Motor 4 torque curve. 41
5.25 All the thrust and torque curves next to each other, with throttle standardized. 42

6.1 Diagram of the notation of the quadcopter. 44

7.1 Block diagram of the Discrete Extended Kalman Filter used. 54

8.1 Block diagram of the Tracking Discrete LQR used. 59

9.1 Diagram of the flow of the simulation. 68
9.2 Block diagram of the combined Kalman Filter and LQR controller that was used 70
9.3 Flow diagram of the steps to compute the output throttles. 71
9.4 The throttles fed into the plant over the simulated duration. 72
9.5 The inputs which were converted into throttles and which were fed into the state calculations. 72
9.6 The simulated translational acceleration results. 73
9.7 The simulated angular acceleration results. 74
9.8 The simulated translational velocity results. 74
9.9 The simulated angular velocity results. 75

9.10 The simulated translational position results. 75
9.11 The simulated angular position results. 76
9.12 The residual between the Kalman estimates and the simulated sensor readouts of the vertical accelerations. 77
9.13 The residual between the Kalman estimates and the simulated sensor readouts of the angular velocities. 78

B.1 Diagram of the flow of the software implemented on the quadcopter. 88

C.1 Collected back EMF data from motor 1. 92
C.2 Collected back EMF data from motor 2. 92
C.3 Collected back EMF data from motor 3. 93
C.4 Collected back EMF data from motor 4. 93
C.5 Trimmed back EMF data from motor 1. 94
C.6 Trimmed back EMF data from motor 2. 95
C.7 Trimmed back EMF data from motor 3. 95
C.8 Trimmed back EMF data from motor 4. 96
C.9 Motor 1 EMF curve. 97
C.10 Motor 2 EMF curve. 97
C.11 Motor 3 EMF curve. 98
C.12 Motor 4 EMF curve. 98

D.1 Schematic of the GY-521 breakout board [1]. 103

E.1 Schematic of the cape. 106
E.2 Layout of the cape. 107

G.1 Electrical schematic of the experimental setup. 111
G.2 CAD model of the component where the motor mounts to. 112
G.3 CAD model of the component which rotates on the encoder. 112
G.4 CAD mock up of the entire apparatus. 113
G.5 CAD drawing of element where the motor mounts to. Dimensions are in mm. 113
G.6 CAD drawing of element where the motor mounts to. Dimensions are in mm. 114

G.7 CAD drawing of the entire apparatus. The dimensions shown refer to distances which are significant to the calculation of reactionary torque. Dimensions are in mm. 115

H.1 Simulink used for collecting thrust cell calibration readings. 121

H.2 Configuration of the unit. 122

H.3 Configuration of the analog ports. 123

H.4 Configuration of the encoder ports. 124

H.5 Configuration of the PWM ports. 125

List of Tables

2.1 Found mass of the quadcopter. 5

2.2 Calculated moments of inertia of the QC. 9

3.1 Calculated variances in the MPU 6050. The accelerometer's units are $(\frac{m}{s^2})^2$ whilst the gyro's units are $(\frac{rad}{s})$. 15

4.1 Table of the masses and the distances point of rotation 22

5.1 Table of the found performance characteristics. For the curves, the values are components of a polynomial, as described in equation (5.1). For the saturation values, A is the saturation point at the low end while B is at the high end. 42

C.1 Table of the found performance characteristics. For the curves, the values are components of a polynomial, as described in equation (C.1). For the saturation values, A is the saturation point at the low end while B is at the high end. 99

D.1 BeagleBone Black specifications [2]. 101

D.2 Turnigy AX-2204C specifications [3]. 102

D.3 Turnigy AX-2204C with a 7038 propeller specifications [3]. 102

D.4 Exceed RC Proton 10A ESC specifications [4]. 102

D.5 InvenSense MPU 6050 accelerometer + gyro specifications [1]. 103

E.1 Pin usage on the BeagleBone Black quadcopter. 105

F.1 3132 micro load cell (0-780g) specifications [5]. 109

F.2 LT 1167 instrumentation amplifier specifications [6]. 110

F.3 TL 7660 CMOS voltage converter specifications [7]. 110

Abbreviations

UAV	Unmanned Ariel Vehicle
DOD	Department of Defense
FAA	Federal Aviation Administration
CAD	Computer Aided Design
QC	Quadcopter
FBD	Free Body Diagram
KD	Kinetic Diagram
BBB	BeagleBone Black
I2C	Inter-Integrated Circuit
EMF	Electromotive Force
DAQ	Data Acquisition
ADC	Analog to Digital Converter
IID	Independent and Identically Distributed
CG	Center of Gravity
PWM	Pulse Width Modulation
KF	Kalman Filter
EKF	Extended Kalman Filter
PID	Proportional Integral Derivative
LQR	Linear Quadratic Regulator
DARE	Discrete Algebraic Riccati Equation
RK45	RungeKuttaFehlberg Method
GPS	Global Positioning System

USB	Universal Serial Bus
OS	Operating System
SSH	Secure Shell
DHCP	Dynamic Host Configuration Protocol
IP	Internet Protocol
AP	Access Point
HDMI	High-Definition Multimedia Interface

Symbols and Notation

Symbols

χ	States
V	$\begin{bmatrix} X & Y & Z \end{bmatrix}^\top$
X	X-axis Position (m)
Y	Y-axis Position (m)
Z	Z-axis Position (m)
$\dot{V}$	$\begin{bmatrix} \dot{X} & \dot{Y} & \dot{Z} \end{bmatrix}^\top$
$\dot{X}$	X-axis Velocity (m/s)
$\dot{Y}$	Y-axis Velocity (m/s)
$\dot{Z}$	Z-axis Velocity (m/s)
$\ddot{V}$	$\begin{bmatrix} \ddot{X} & \ddot{Y} & \ddot{Z} \end{bmatrix}^\top$
$\ddot{X}$	X-axis Acceleration (m/s^2)
$\ddot{Y}$	Y-axis Acceleration (m/s^2)
$\ddot{Z}$	Z-axis Acceleration (m/s^2)
Ω	$\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top$
ϕ	Pitch Position (rad)
θ	Roll Position (rad)
ψ	Yaw Position (rad)
$\dot{\Omega}$	$\begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^\top$
$\dot{\phi}$	Pitch Angular Velocity (rad/s)
$\dot{\theta}$	Roll Angular Velocity (rad/s)
$\dot{\psi}$	Yaw Angular Velocity (rad/s)
$\ddot{\Omega}$	$\begin{bmatrix} \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \\ cc & & \end{bmatrix}^\top$

$\ddot{\phi}$	Pitch Angular Acceleration (rad/s^2)
$\ddot{\theta}$	Roll Angular Acceleration (rad/s^2)
$\ddot{\psi}$	Yaw Angular Acceleration (rad/s^2)
I	The Moment of Inertia $(kg \cdot m^2)$
Θ	Angle of Deflection (rad)
m	Mass (kg)
g	Gravity (N)
l	Length of String (m)
d	Distance from CG to String Attachment (m)
ω	Angular Velocity (rad/s)
t	Time (s)
F	Motor Thrust (N)
τ	Reaction Torque $(N \cdot m)$
$\mathbf{R}$	Rotation Matrix
a	Theoretical Model
u	Inputs
$\mathbf{C}$	Output Matrix
w	Model Noise
Z	Sensor Reading
h	Sensor Noisefree
v	Sensor Noise
P	Covariance of the Error
$\mathbf{Q}_{kf}$	Covariance of the Model
$\mathbf{R}_{kf}$	Covariance of the Sensor
$\mathbf{L}$	Kalman Gain
$\mathbf{I}$	Identity Matrix
dt	Time Increment Between Calculations (s)

$\mathbf{A}_{lqr}$ Linear System Matrix

$\mathbf{B}_{lqr}$ Linear Input Matrix

$\mathbf{J}$ Cost to be Minimized

$\mathbf{Q}_{lqr}$ Cost Weighting of the Target Error

$\mathbf{R}_{lqr}$ Cost Weighting of the Inputs

$\mathbf{K}$ Gain Matrix

Subscripts

m Motor

mm Motor Mount

lc Load Cell

r Reaction Lever

w Weight

X X-Axis

Y Y-Axis

Z Z-Axis

1 Motor 1

2 Motor 2

3 Motor 3

4 Motor 4

0 Starting Conditions

k Iteration

Chapter 1

An Introduction to Unmanned Aerial Vehicles

1.1 Introduction

In the past, the sky was the domain of the birds, the bees, and the infrequent meteor. Humanity, in our intrinsic need to go "There", looked to join this club since the days when the wheel was the new and hip thing. Many centuries later, with many technological advances and even more attempts, Homo Sapiens added themselves to the list of things that traverse the sky. It was on December 17, 1903, however, when humanity took flight with the first heavier-then-air vehicle. Following this milestone, humanity started to go higher, faster, and further in aircraft that range from those only large enough to hold a single person to multi ton monstrosities that dwarf buildings. Most recently, humans began to take the pilot out of the cockpit, leading to the birth of the unmanned aerial vehicle.

1.2 A Brief Overview of UAVs

The name, unmanned aerial vehicle, or UAV for short, is a term that came to use in the late 1990s. Common and broad names such as drone and remotely operated aircraft, as well as more exotic and specific terms such Sperry's "Aerial Torpedo", have also been used as names for the same sort of devices [8]. As the name suggests, these are vehicles that are in the air that lack a person in them.

While some devices are quickly associated with UAVs, others inclusion in the UAV spectrum can be debated. A guided rocket or even a humble messenger pigeon could be considered to be a UAV, depending

on how inclusive a definition is. According to the Department of Defense in their Dictionary of Military and Associated Terms, commonly referred to as JP1-02 [9], a unmanned aerial vehicle is

> *A powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload. Ballistic or semiballistic vehicles, cruise missiles, and artillery projectiles are not considered unmanned aerial vehicles.*

The Federal Aviation Administration also has its own definition [10]. Defined in the FAA Modernization and Reform Act of 2012 (Public Law 112-95), a unmanned aircraft is

> *...an aircraft that is operated without the possibility of direct human intervention from within or on the aircraft*

while a unmanned aircraft system is

> *...an unmanned aircraft and associated elements (including communication links and the components that control the unmanned aircraft) that are required for the pilot in command to operate safely and efficiently in the national airspace system.*

While many other respectable and reputable organizations put forth their own definitions, they all paint a very similar picture of what a UAV is, with minor deviations that are often specific to the organization which puts out the definition. These organizations also put forth more specific terms to classify UAVs within subclasses. Because of how quickly this field is expanding, these definitions are also being adjusted at a very frequent rate to keep pace with new breakthroughs.

As with a good chunk of humanity's drive for technological advancement, the development of UAVs has military origins [11]. Ironically, one of the first uses for drones was for target practice for ground crews. It was during the Cold War when the use was expanded for surveillance purposes and even use in combat roles, though very limited in scope.

While UAVs have existed for some time now, it is only now at the start of the twenty-first century where these devices have truly become ubiquitous. While technological advancements as a whole can be attributed to this increase in the prominence of UAVs, it is the bounding leaps within semiconductor devices which has breathed life into the field of unmanned vehicles as a whole. With computers becoming more powerful, smaller, and cheaper at an exponential rate, it has allowed UAVs to become intelligent

systems which can "think". More powerful computers have also allowed the development of many of the algorithms which allow an unmanned system to be intelligent.

UAVs are still used within the military, though their role has vastly expanded. While still used for target practice and surveillance, their use and capabilities have vastly increased from their Cold War predecessors. In addition, unmanned systems are beginning to have significant combat roles, notably in the second Iraqi War and the War on Terror.

The use of unmanned systems hasn't been limited only to the military. With lowering costs, the use of drones has blossomed within the public sector. From increasing crop yields within agriculture [12], to delivering impulse online purchases [13], to uses in academia as a learning tool, new uses are appearing daily.

1.3 A More Detailed Look Into Quadcopters

One popular type of UAV is the quadcopter. Also commonly referred to as a quadrotor, this type of rotor craft's defining feature is that it has four propellers, as seen in Figure 1.1. The idea of an aerial vehicle with four propellers has been around for almost as long as heavier-than-air vehicles. However, another one of this craft's defining features is that the body is inherently unstable. Since controlling four separate rotors manually proved to be too difficult, this led to to the design being scrapped in favor of single or dual rotor crafts.

Figure 1.1: A picture of the quadcopter which this thesis used as a basis.

In recent years, quadcopters have had a resurgence in popularity. With the advancements of modern day electronics, an on board computer controls the four rotors, allowing the inherently unstable system to have stable flight. While this resurgence has had an effect in nearly all fields, its presence has been most felt in localized commercial applications, academia, and among hobbyists. This is due to a quadcopters capability to be small, be able to move in any direction or hover, and its low cost compared to other aerial vehicles.

Chapter 2

The Moments of Inertia of the Quadcopter

2.1 Introduction

An object in motion tends to stay in motion. These famous words serve as part of the core of Newtonian mechanics. A quadcopter will want to continue moving unless an external force acts upon it. To properly model this phenomena, a few characteristics need to be identified. For the linear flight mechanics, the mass needs to be identified. For the angular flight mechanics, the various moments of inertia need to be identified.

2.2 The Mass

There are two ways that the mass can be identified. The first way is to model the quad copter in a CAD program and then have the computer compute what the mass would be. While the payoffs of such a method are fairly high, this would be a very laborious and long process. The other method is the more straightforward experimental approach, which consists of placing the quadcopter on a scale and measuring the weight. The later option was chosen as the potential benefits of the first are not worth the additional cost in time and effort.

Table 2.1: Found mass of the quadcopter.

Mass of the QC
479.95 g

2.3 Theory Collecting the Moments of Inertia

Identifying the various moments of inertia present on a quadcopter is not as straight forward of a task as was the mass. Once again, the theoretical or the experimental approach can be used to compute the moments. Once again, the experimental was chosen, as it is the easier of the two.

To experimentally determine the moments, a procedure suggested in [14] was used. The quadcopter was wired up as a pendulum and then swung. The period on the swing could be then use to compute what the moments are. Figure 2.1 is a picture of the setup used.

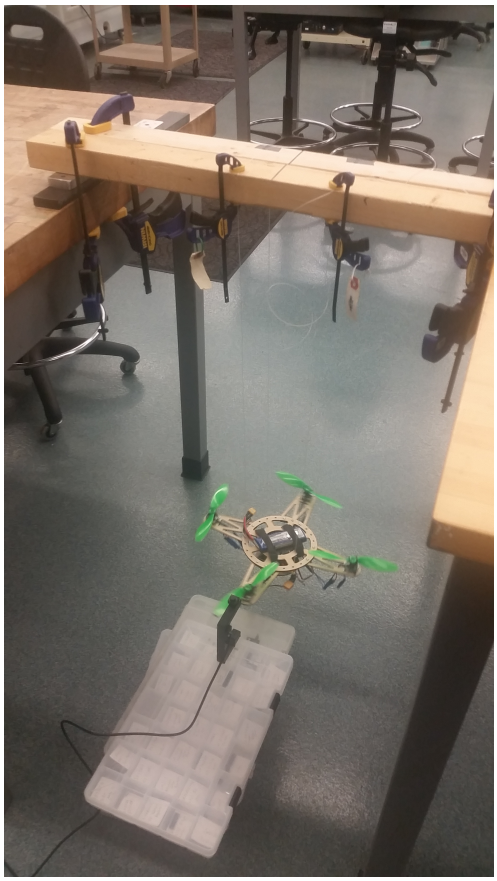


Figure 2.1: A picture of the setup used to identify the period of a swing.

The starting point in transforming an oscillation period into a moment of inertia is an FBD KD diagram, Figure 2.2, and the conservation of angular momentum.

$$\frac{\partial L_{sys}}{\partial t} = \sum Moments + \sum_{in} (r \times V)\dot{m} - \sum_{out} (r \times V)\dot{m} \tag{2.1}$$

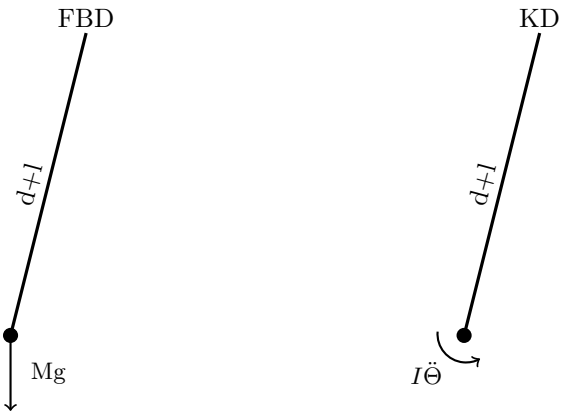


Figure 2.2: FBD and KD of the experimental set up to obtain the moments of inertia.

These two elements are combined to form

$$I\ddot{\Theta} = -Mg(l+d)\sin\Theta. \tag{2.2}$$

The inertia, as listed right now is incorrect, as the point of rotation is not at the location of mass but is instead at the very top of the pendulum. Thus, the parallel axis theorem must be applied to correctly find the moment of the quadcopter. This tweaks (2.2) into

$$(I + m(l+d)^2)\ddot{\Theta} = -mg(l+d)\sin\Theta. \tag{2.3}$$

A small amount of rearranging transforms (2.3) into

$$\ddot{\Theta} + \frac{mg(l+d)}{(I+m(l+d)^2)}\sin\Theta = 0. \tag{2.4}$$

The next step is to linearize $\sin\Theta$. This is permissible as the angle of deflection, Θ, will always be very small. To linearize $\sin\Theta$ around 0, a Taylor expansion is used. The expansion will only be to the first order. This yields

$$\sin\Theta \approx \sin 0 + \Theta\cos 0 = \Theta \tag{2.5}$$

which, when substituted into (2.4) results in

$$\ddot{\Theta} + \frac{mg(l+d)}{(I+m(l+d)^2)}\Theta = 0. \tag{2.6}$$

This leads (2.6) to take on the shape of the simple harmonic oscillator equation.

$$\ddot{\Theta} + \omega^2 \Theta = 0 \tag{2.7}$$

where

$$\omega = \frac{2\pi}{t}. \tag{2.8}$$

This allows the constants from (2.6) and (2.7) to be pulled out and set equal to each other. This then yields

$$\omega^2 = \frac{mg(l+d)}{(I + m(l+d)^2)}. \tag{2.9}$$

Substituting (2.8) into (2.9) yields

$$\left(\frac{2\pi}{t}\right)^2 = \frac{mg(l+d)}{(I + m(l+d)^2)} \tag{2.10}$$

and with a little bit of rearranging, (2.10) can be solved for the moment of inertia.

$$I = \frac{mg(l+d)t^2}{(2\pi)^2} - m(l+d)^2 \tag{2.11}$$

2.4 Identifying the Moments of Inertia

The one element that needs to be identified is the period of oscillation; everything else is a known constant on the quadcopter. To capture the period of an oscillation, the quadcopter was hung. A photogate was used in conjunction with the Quanser unit to capture the duration of the swing. The Quanser unit was configured through Simulink as shown in Figure 2.3.

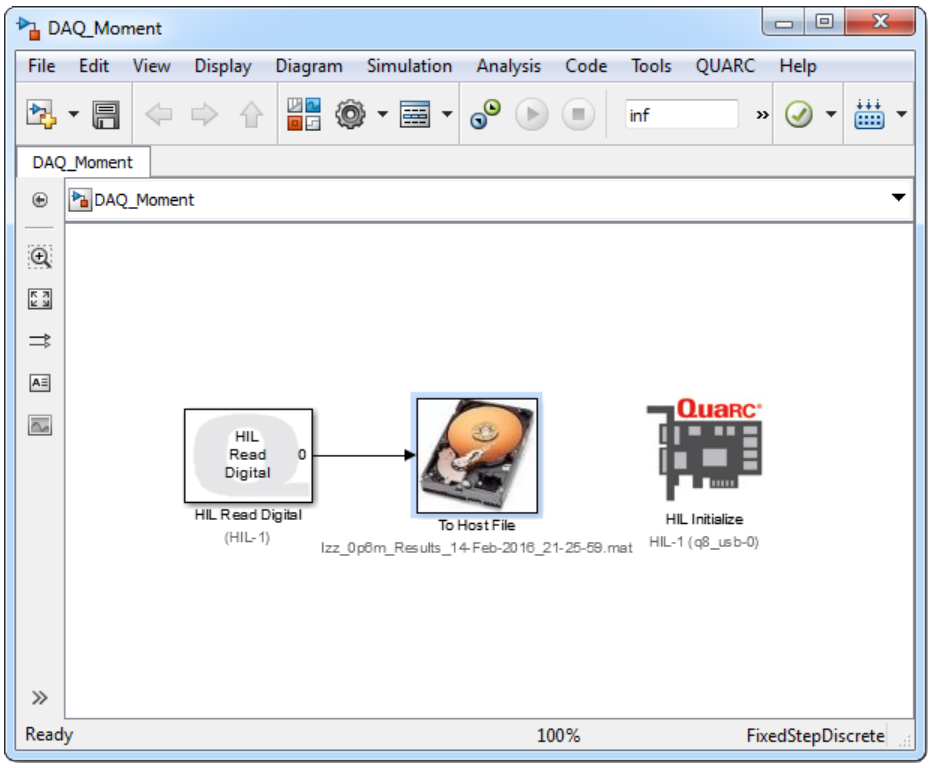


Figure 2.3: Simulink diagram which controlled the Quanser data collection.

Two separate tests were conducted: One for the momentum around the Z-axis and one for the momentum about both the X and Y-axis. What allows the moments for the X and Y-axis to be the same is the assumption that the quadcopter is symmetric in nature.

The time periods were collected and processed. The length of string from which the quadcopter was hung was also varied. As long as the lengths are accounted for in the equation, the moments should remain the same.

The results were then averaged and the following moments were obtained:

Table 2.2: Calculated moments of inertia of the QC.

Axis	Calculated Moment (kgm^2)
Ixx & Iyy	6.9351e-03
Izz	1.1474e-01

Chapter 3

The MPU6050

3.1 Introduction

It is common to use a sensor to give a device feedback. This is no exception for the quadcopter. Given the inherently unstable nature of a quadcopter, having a feedback mechanism is critical. The BBB quadcopter makes use of the MPU6050, a six direction accelerometer and gyro to provide that feedback. The sensor is able to read values with 16-bits of precision. It then is able to communicate those results back to its master device through the use of the I2C communication protocol.

As with all sensors, noise is present. Before the sensor is blindly used, finding out the characteristics of this noise is advantageous, most notable in the development of an accurate simulation.

3.2 Testing the Variance in the MPU6050

To gather the data points on the MPU6050, each sensor's outputs were read at 10Hz. One-hundred data points per axis were gathered. In practice, only thirty data points are needed per axis, as the central limit theory kicks in at that point. However, the one-hundred data points per axis were collected because the cost of collecting data is negligibly small. Additional data also helps in detecting any anomalies present in the sensor's normal operation.

During the test, the quadcopter was set on a concrete floor to prevent any unintended interference.

3.3 Results of Testing the Variance in the MPU6050

When working through the data, two things were observed. First, the variance in the sensor's various axes. The second was a check to see if the variance was Gaussian in nature. This is more important than the actual variance since most noise eliminating techniques depend on the idea that the noise is normally distributed.

To check the normality of the data, the data was plotted as a histogram in Figures 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6. This was done over a test like the Anderson-Darling because of the discrete nature of the data. The gyro and accelerometer only provide discretized results. This, combined with a significant amount of points, results in traditional tests for testing normality in breaking down.

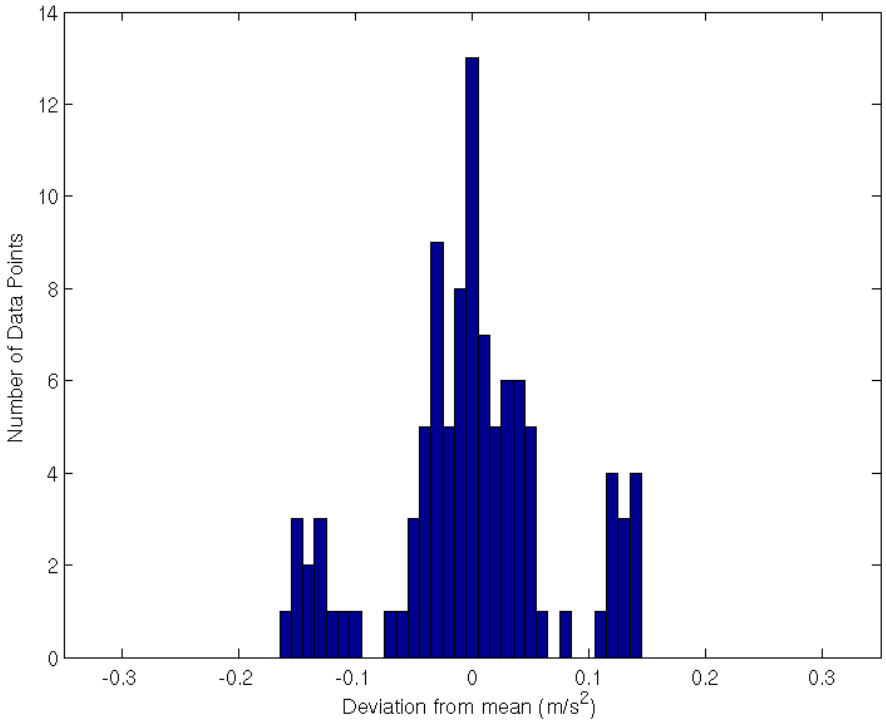


Figure 3.1: Variance present in the X-axis in the MPU6050's accelerometer.

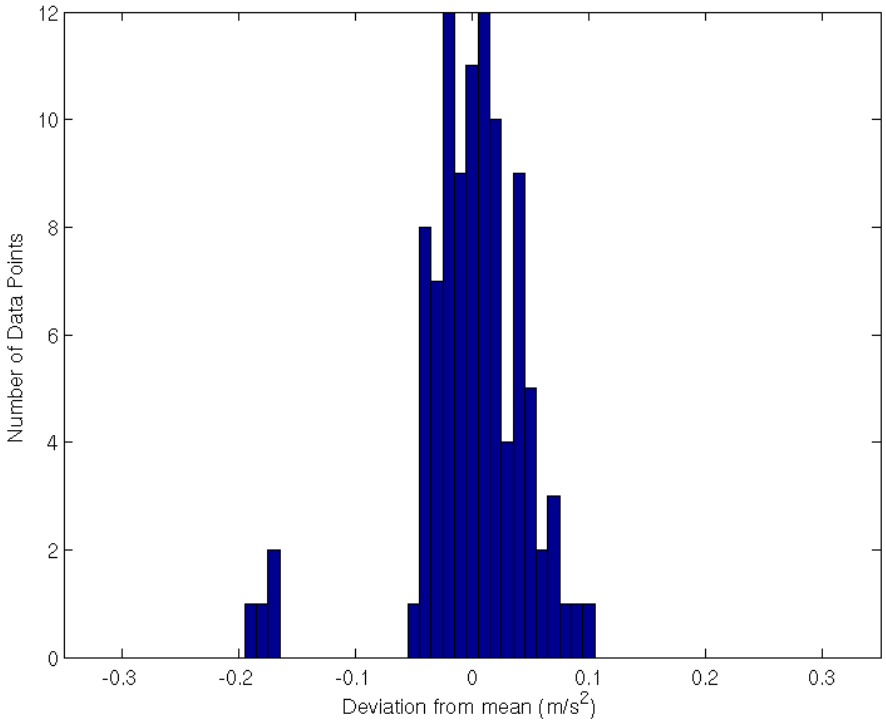


Figure 3.2: Variance present in the Y-axis in the MPU6050's accelerometer.

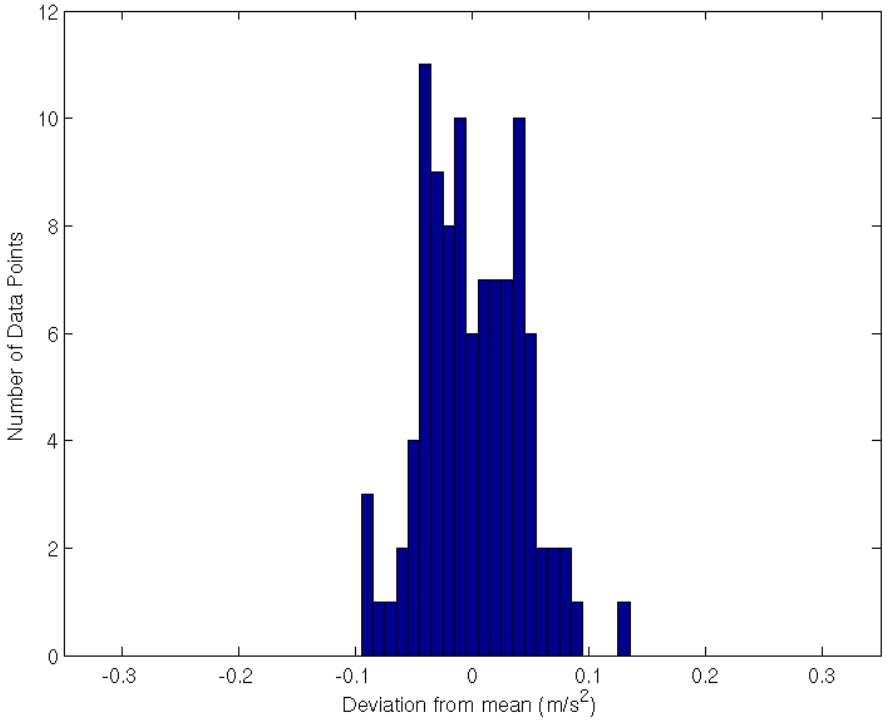


Figure 3.3: Variance present in the Z-axis in the MPU6050's accelerometer.

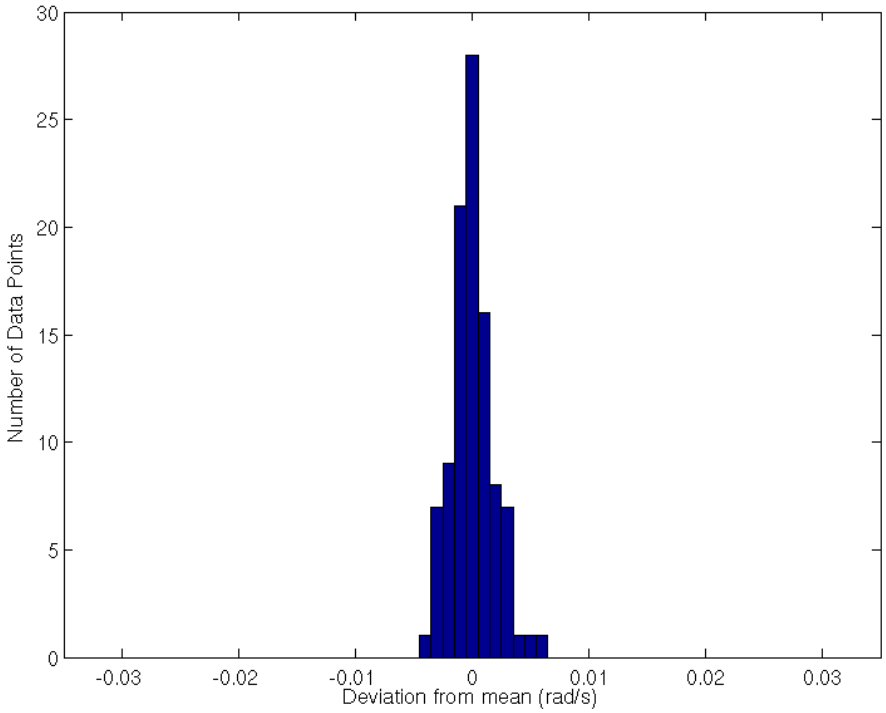


Figure 3.4: Variance present along the X-axis in the MPU6050's gyro.

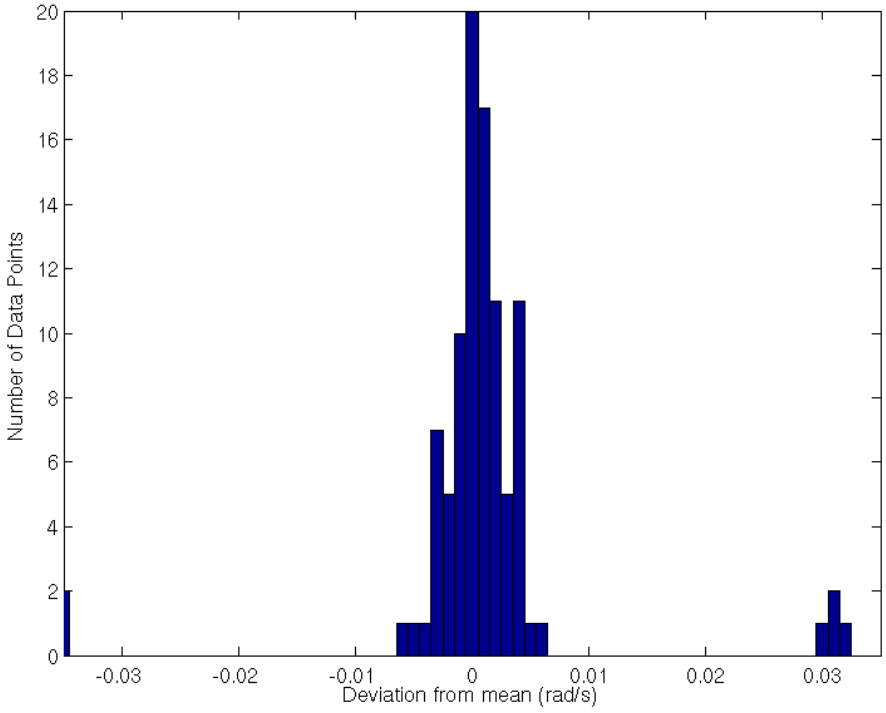


Figure 3.5: Variance present along the Y-axis in the MPU6050's gyro.

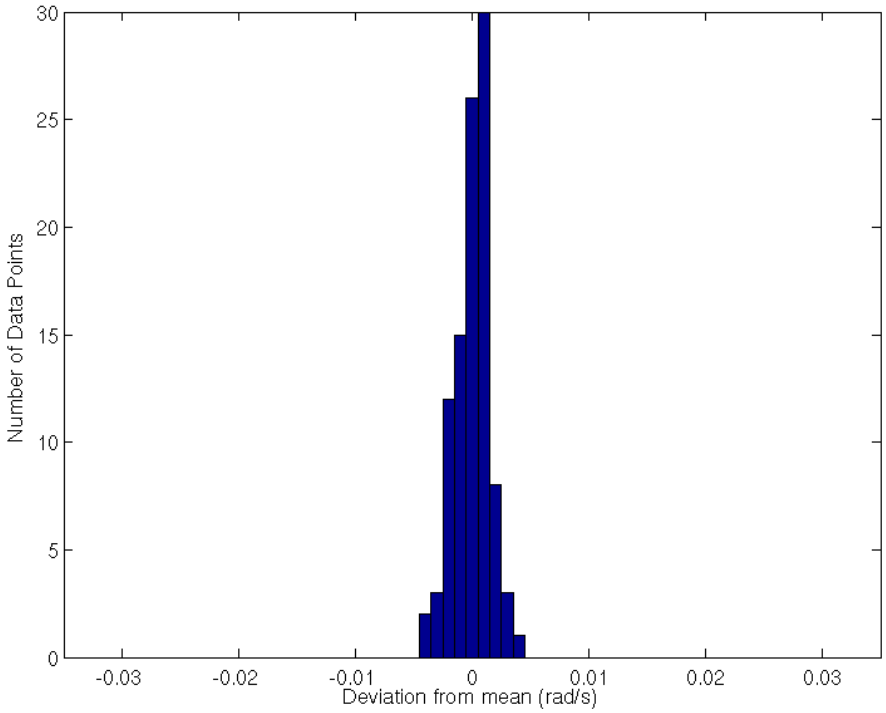


Figure 3.6: Variance present along the Z-axis in the MPU6050's gyro.

Visually inspecting the distributions, all of the directions sensed have a distribution that is Gaussian in nature. While expected, this is good to see and have verified. The sample variance was then computed.

Table 3.1: Calculated variances in the MPU 6050. The accelerometer's units are $(\frac{m}{s^2})^2$ whilst the gyro's units are $(\frac{rad}{s})$.

Sensor	Computed Variance
X Acceleration	4.9609e-03
Y Acceleration	2.3615e-03
Z Acceleration	1.7702e-03
X Rotational Velocity	3.3864e-06
Y Rotational Velocity	1.0749e-04
Z Rotational Velocity	2.1762e-06

Chapter 4

Motor Experimental Setup

4.1 Introduction

One of the most important elements in a quadcopter is its motors. Consequently, one of the most important elements in a quadcopter's model is the motors' performance characteristics. Four motor characteristics are desired. The two obviously desired traits are the thrust and torque curves, as they directly affect the flight mechanics of a quadcopter. The two others are the motor saturation points and the back EMF curve. The motor saturation points are desired to identify the true operation range of the motors. The back EMF was explored but not used in the final design. Any findings made in exploring its use can be found in Appendix C

Data about the three curves is explicitly collected while information on the saturation points is extracted from the three types of data. A Quanser unit was used to collect the data. As most DAQs are, the Quanser unit is limited to digital and analog electrical signals. This chapter discusses how the desired motor characteristics are captured and stored for analysis. Data reduction is presented in Chapter 5.

4.2 Thrust Collection

The thrust is read by a load cell. When a force is applied to a load cell, the load cell deflects. A resistor present on the load cell also experiences deflection, which results in a changed resistance. This resistor is in a wheatstone bridge configuration, which allows the change in resistance to be observed as a change in voltage.

The change in voltage, however, is small. Even though the Quanser's ADC has 16 bit precision, ampli-

fying the collected voltage is desired. The best way to do so is to amplify the signal with an instrumentation amplifier. While one could be built from multiple op-amps, a pre-built one, the LT1167, was used.

Because the voltage reading came from a load cell and was then amplified, a calibration curve was fit to the load cell. The load cell was loaded at 50 gram intervals, up to its maximum specified load of 780 grams. One hundred samples per load were collected. One hundred was chosen because a minimum of thirty samples should be collected to ensure that the central limit theory kicks in for the data collected. The cost of obtaining more samples was negligible and additional samples do help. All these sample points were collected and plotted in Figure 4.1, where the black points represent the individual data points while the red points represent their averages.

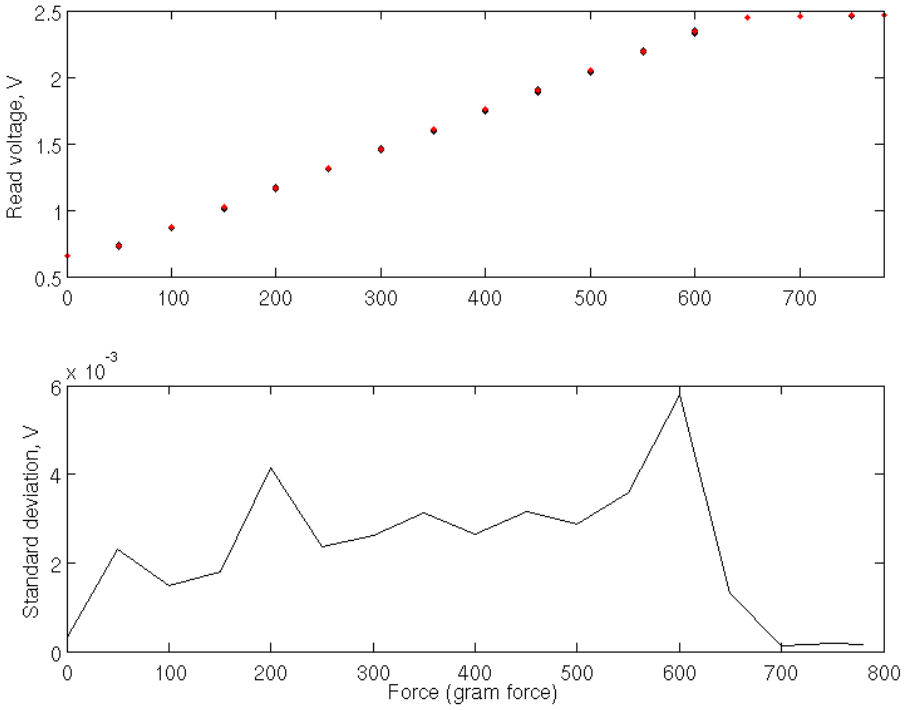


Figure 4.1: Collected results from the thrust load cells.

At this point comes the job of fitting the data. When the data is inspected, some of the higher load points capped out, most likely due to saturation in the instrumentation amp. Those points are not needed and will not be used to create a fit. Some strange nonlinearities were also observed in the low ranges, so the data from the unloaded cell was also not used to generate the fit. The rest of the data was used to create a fit as seen in Figure 4.2, where the blue line is the fit.

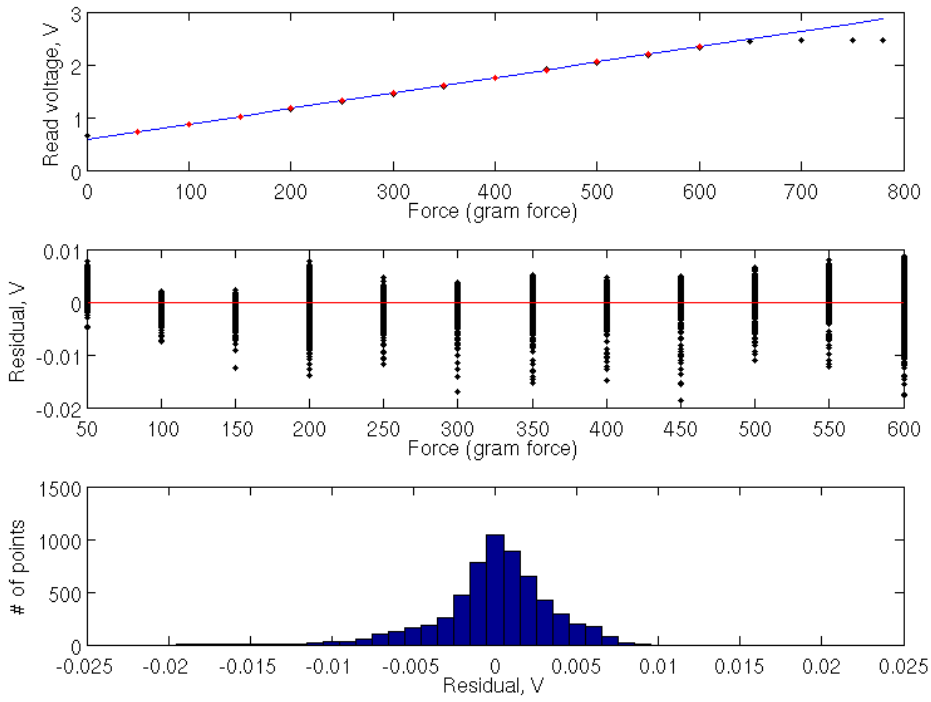


Figure 4.2: Processed thrust load cell calibration.

As tempting as it may be to fit a line and call it done, a proper fit requires more. A fit is truly a good fit if it meets five assumptions:

1. The mean value of the errors is zero.
2. The variance of the errors is homoscedastic through the data.
3. The data is IID.
4. The errors are normally distributed.
5. There is negligible variance in the predictor values.

All five of these assumptions must be satisfied to consider a regression fit to be a proper representation of the desired data [15] [16].

Fortunately, the fifth assumption, that there is negligible variance in the predictor values, can be assumed to be true unless there is reason to believe otherwise, which is not present in this case. The third assumption, where the data is IID, is also very reasonable to assume in this case, as all the points should be identically distributed and independent of each other. This leaves only three assumptions to check.

Plotting the residuals can tell a lot of information and is the best way to check the first two assumptions. The residuals are centered around zero. That satisfies the first assumption. The second assumption can be verified by looking at the spread, as the spread should be rather even around the zero axis. This is also the case, thus satisfying the second assumption.

This leaves the fourth assumption to be verified. Normally, an Anderson-Darling test would be the best way to go about checking this. However, the data is discrete due to the limitations of an ADC. This, coupled with a large number of points, will cause the algorithm to break down. Fortunately, an easy workaround is present, especially when the data is discrete. Looking at a histogram of the residuals, if the data is normally distributed, then the data should take the shape of a Gaussian curve. This is the case, thus confirming the fourth assumption.

Now that the quality of the fit was verified, the fit can be used to convert voltage to thrust load and back.

4.3 Torque Collection

Collecting reliable and believable data which represented the reactionary torque of the motors proved to be a challenging task. This was because the torque is very small, making it very difficult to measure. The best results came from using a configuration of weights, levers, and an encoder.

The setup, as represented in Figure 4.3, relies on the reaction torque to deflect the lever and weights which the motor is mounted on. The angle of deflection is then read through the use of an encoder. The deflection can then be used to calculate the torque that created the deflection.

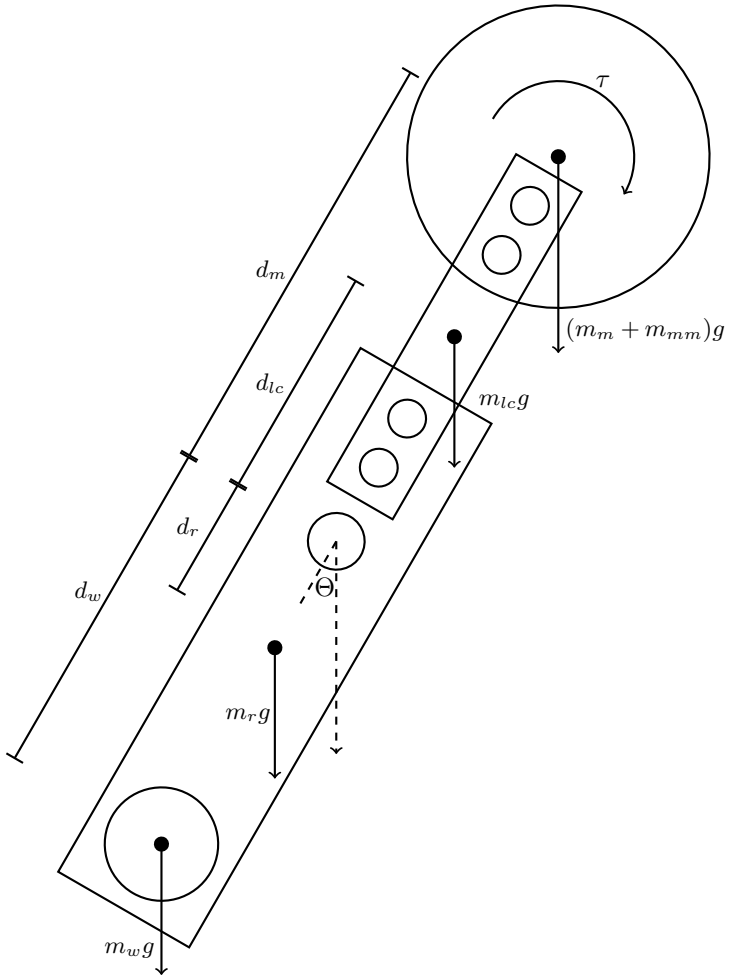


Figure 4.3: Free body diagram of the mount that was used to collect the torque.

Converting the encoder readout to reactionary torque begins by converting the encoder counts to an angle.

$$\Theta = \frac{Counts}{2608 \times 4} \tag{4.1}$$

The constant 2608 is the amount of encoder counts that is equivalent to one radian [17]. The 4 comes from the way which the Quarc unit was configured. The counter torque that prevented the mount from wildly spinning was computed. Because the mount was not moving when deflected, the sum of torques must equal zero. With that, the reaction torque can then be deduced, as written mathematically below.

$$\tau = (m_r d_r + m_w d_w - m_{lc} d_{lc} - (m_m + m_{mm}) d_m) g \sin \Theta \tag{4.2}$$

The properties of the mount were constant and are documented in Table 4.1.

Table 4.1: Table of the masses and the distances point of rotation

Part	Mass (g)	Distance (mm)
Weight (w)	100.00	69.77
Rotating Mount (r)	15.48	25.91
Loadcell (lc)	6.44	32.83
Mount (mm)	3.83	69.88
Motor (m)	28.74	69.88

In practice, the mount for the motor would have its center of gravity shifted slightly closer to the point of rotation. However, for ease of calculation and because the shift would be very subtle, the CG was assumed to be the same as that of the motor.

4.4 Test Procedure

To collect the desired data, the unit was set up to control the motor output by sending out a PWM throttle value between 0% and 100% at 1% increments, through the Simulink model in Figure 4.4. Data was collected for 4 seconds at 1kHz for each throttle increment. Four second sets of data were taken to allow the data to reach steady state conditions. Observing the data collection process in Figure 4.5, it took about 2 seconds for the data to stabilize in the most extreme case when the motors were off and then turned on. While in practice this delay is not present in the motors' operation, it is present due to the apparatus. Thus, even though four seconds of data was taken, only the last half-second will be used to draw conclusions.

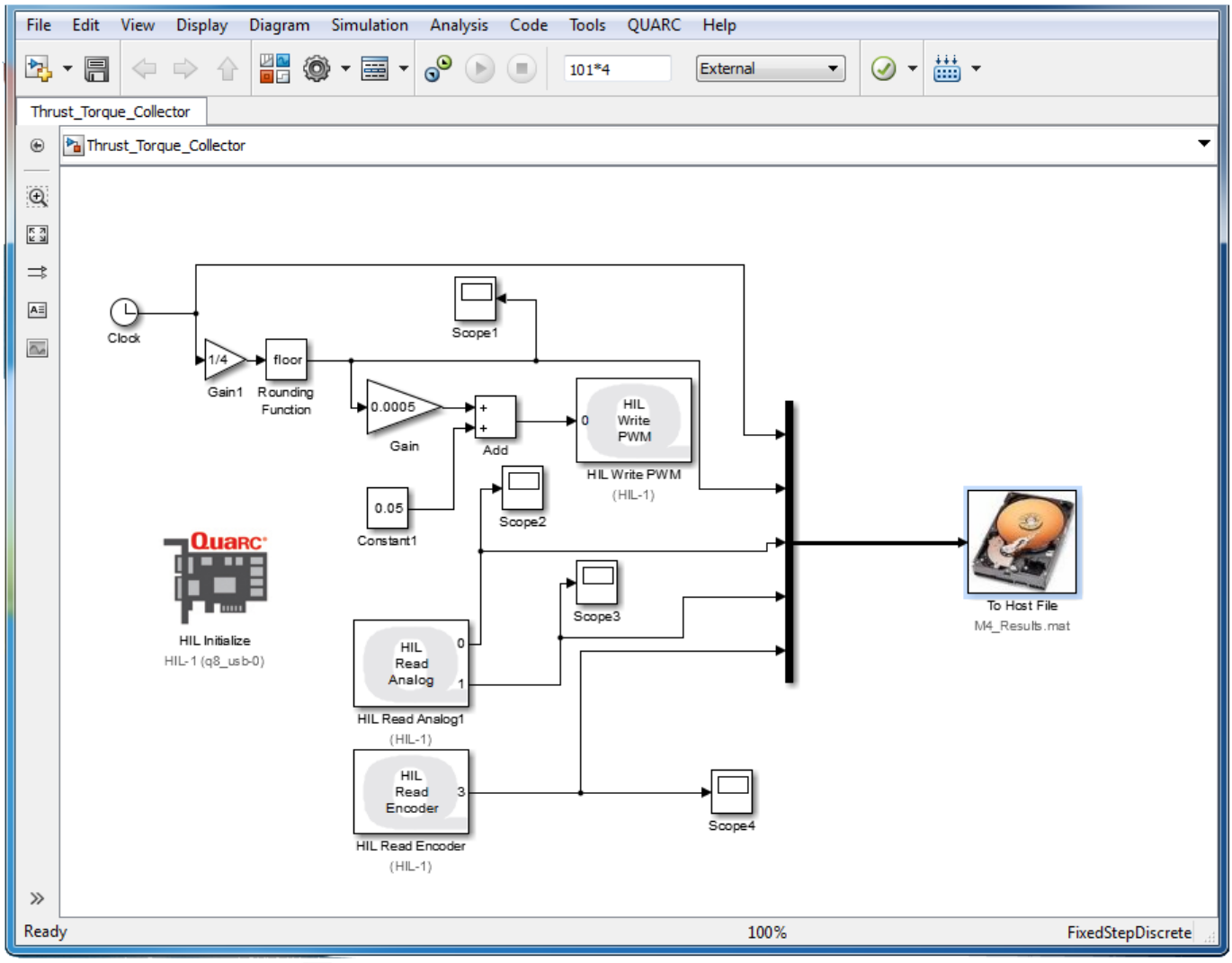


Figure 4.4: Simulink diagram which controlled the Quanser data collection.

Figure 4.5: View of a single throttle point. Note how the data reached steady state conditions only after about two seconds.

The thrust, torque, back EMF, and range were collected at the same time. The test set up can be seen in Figure 4.6. The motors were powered by a power supply at 7.4 volts, the theoretical output of the batteries. The motors were not limited in their current draw. The procedure to collect data was done for each motor to collect all the desired information. Once all the data was collected, it was processed to obtain useful information. This processing will be detailed in the next chapter.

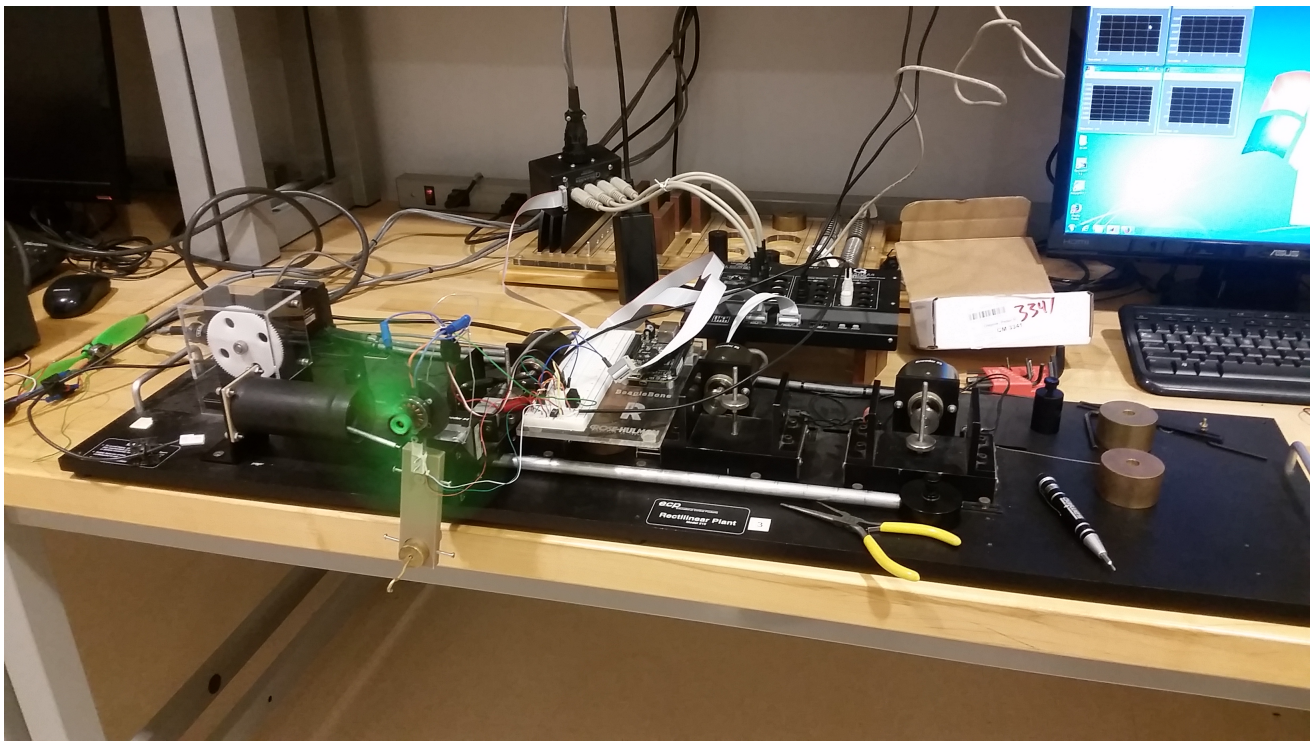


Figure 4.6: Picture of the setup used to collect the data.

Chapter 5

Experimental Motor Results

5.1 Introduction

Data was collected on the performance characteristics for each motor. This section covers the process of converting that data into useful information. From the data, there are four components that are desired: saturation points, thrust curves, torque curves, and back EMF curves. Once again, because the back EMF results were not used in the final design, they are located in Appendix C.

5.2 Trimming the Data

As mentioned previously in Chapter 4, four seconds of data was collected but only about the last two are at steady state conditions. Thus, the first three and a half seconds were trimmed off. The additional second and a half was chopped off to give an extra margin of assurance that the used data was at steady state conditions. Even though only the last half-second is observed, the collection rate of 1kHz still allows a large amount of data to be collected.

5.3 Thrust

The thrust data was used to find both the thrust curve and the range of operation. The first saturation point, where the motors start spinning, can be seen where the thrust voltage begins to increase. The second saturation point, where the motor maxes out in speed, is where the voltage from the load cell stops climbing as the throttle is increased.

The data was also inspected for reasonability. Red points are the means of a throttle set, while black points are individual points. The computed standard deviation was also looked at as an informative factor. This took place in Figures 5.1, 5.2, 5.3, and 5.4.

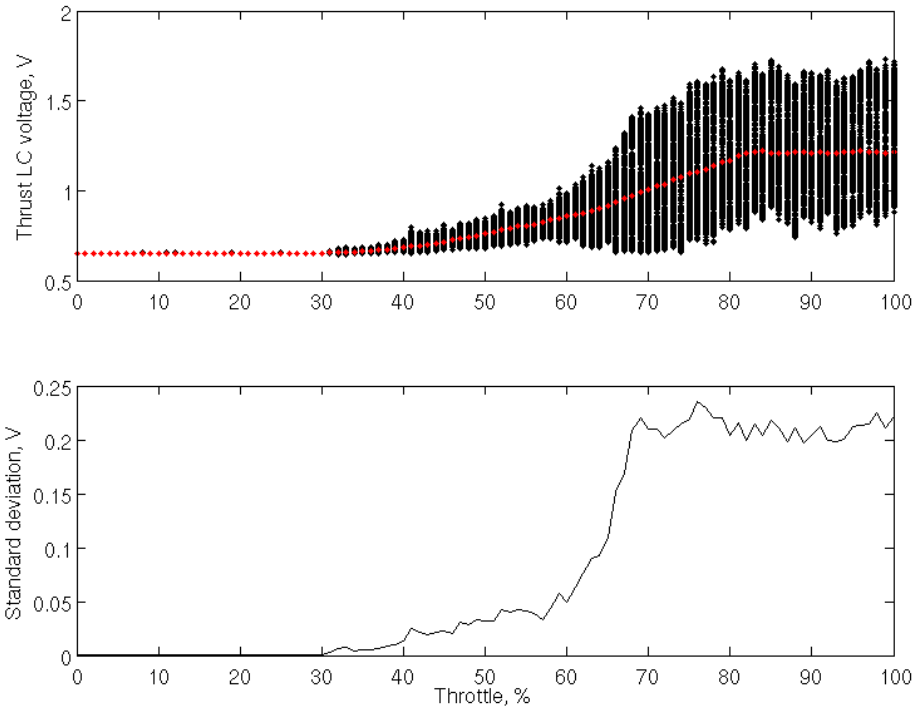


Figure 5.1: Collected thrust data from motor 1.

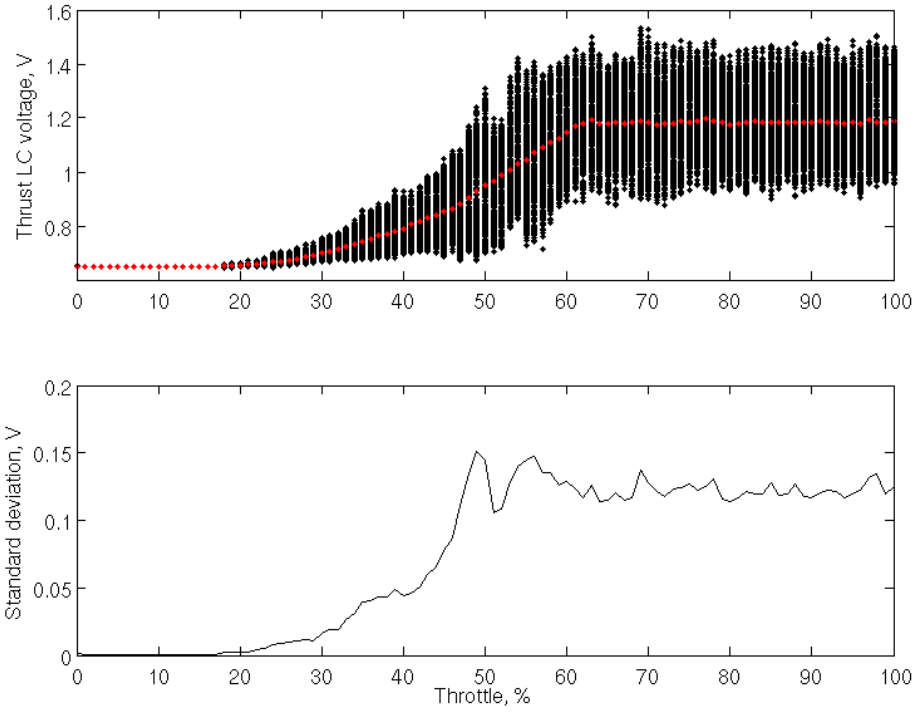


Figure 5.2: Collected thrust data from motor 2.

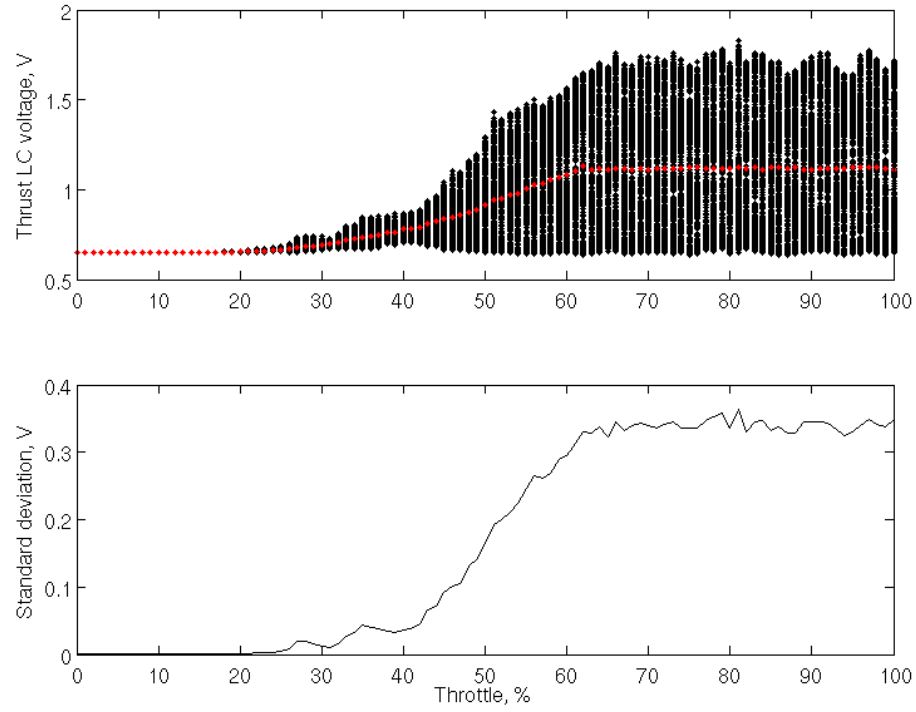


Figure 5.3: Collected thrust data from motor 3.

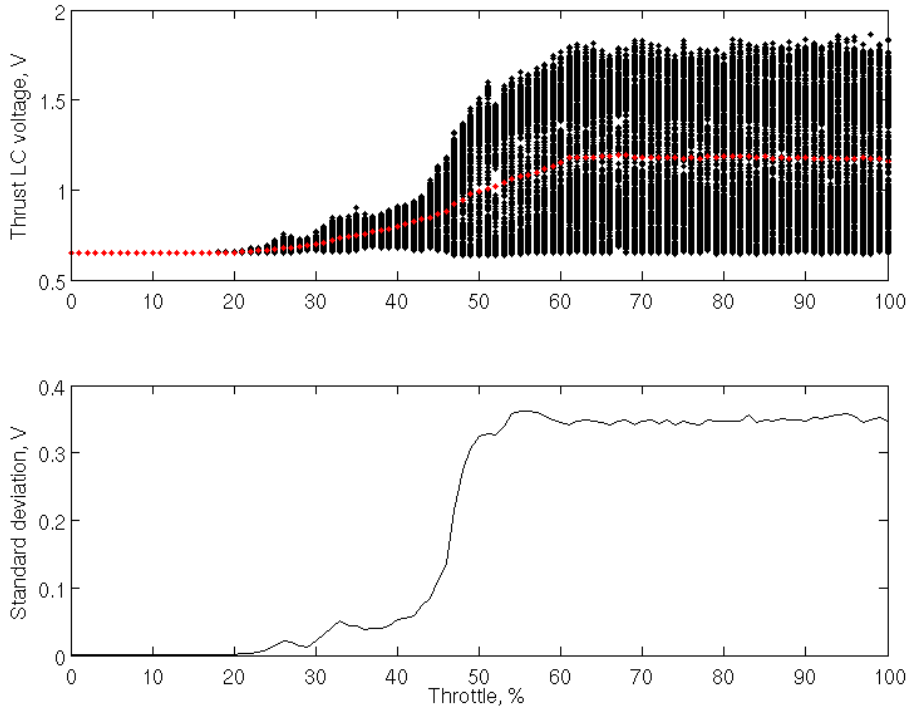


Figure 5.4: Collected thrust data from motor 4.

Looking at the data, it becomes very quickly apparent that there is an increasing spread amongst the data points as throttle goes up. This can be easily explained as a result of vibration. As the motors spin up, vibration increases, adding additional noise. Because of the large amount of data points taken and that the noise is assumed (and later verified) to be Gaussian in nature, this is not a significant issue. Aside from that, the collected data appears as expected. The data is then trimmed outside the motor ranges, yielding Figures 5.5, 5.6, 5.7, and 5.8.

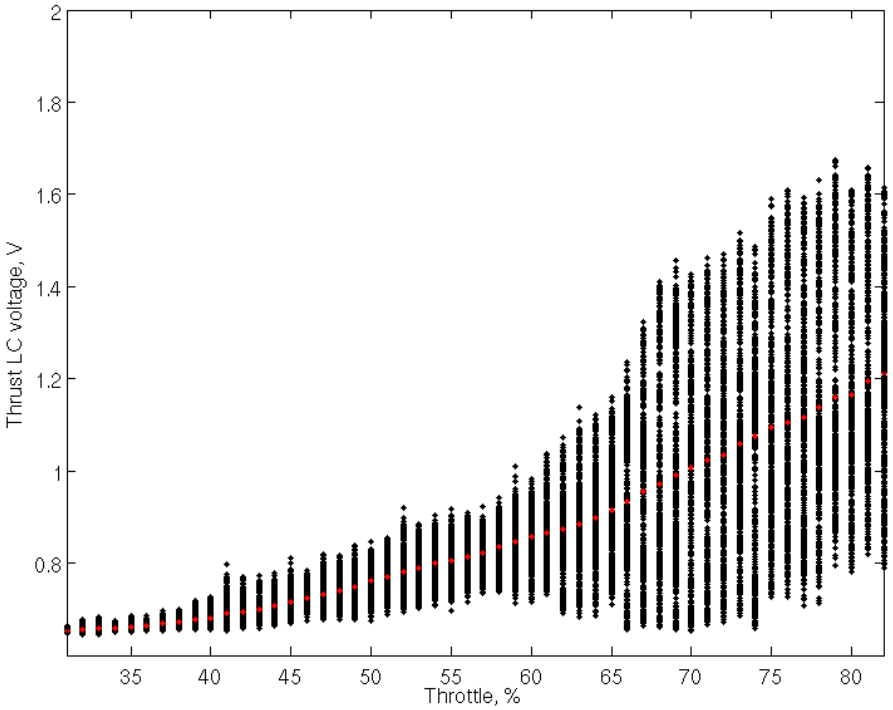


Figure 5.5: Trimmed thrust data from motor 1.

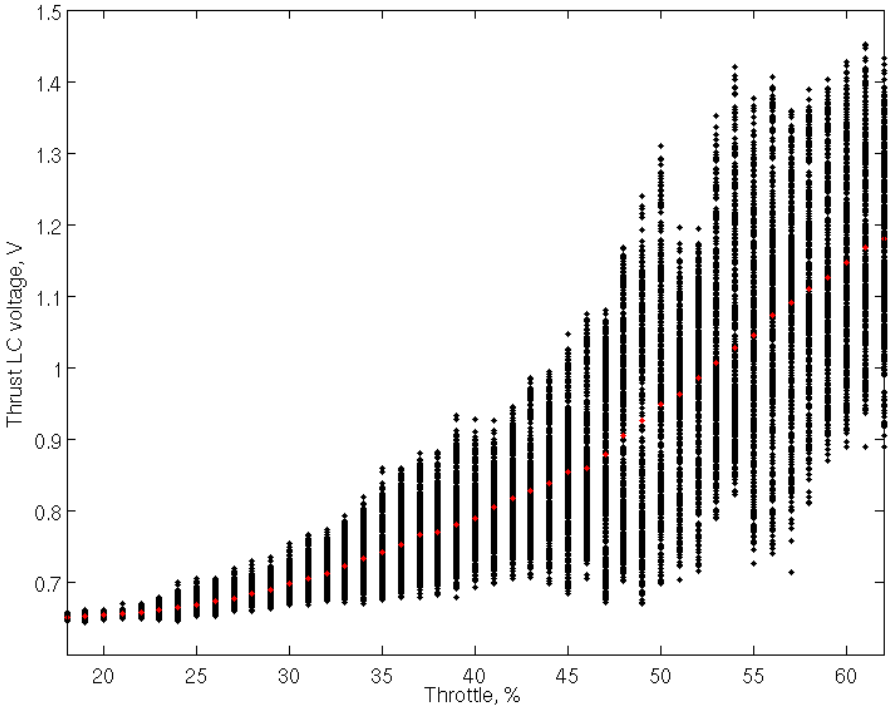


Figure 5.6: Trimmed thrust data from motor 2.

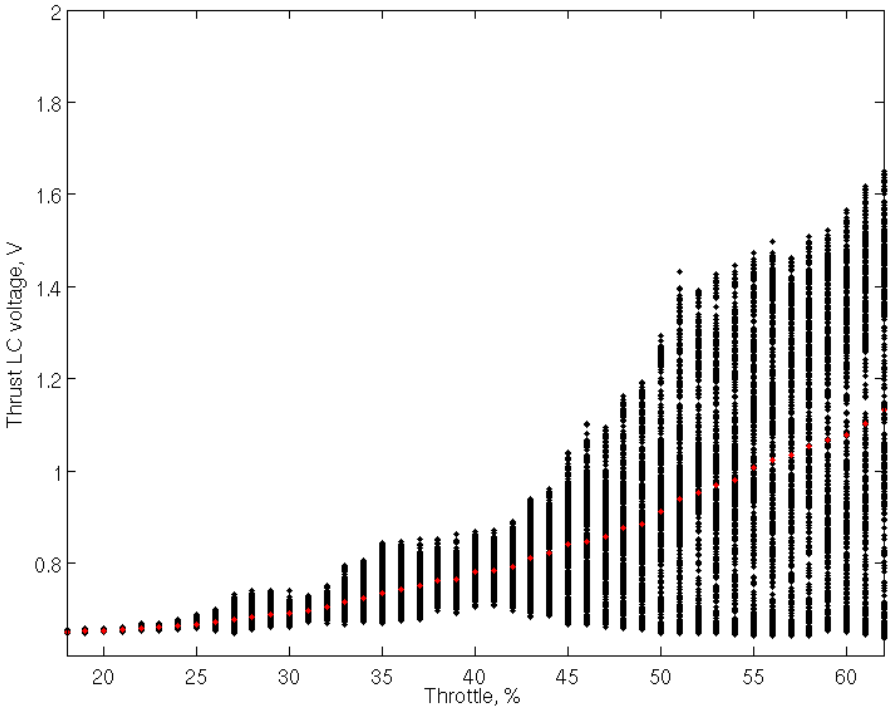


Figure 5.7: Trimmed thrust data from motor 3.

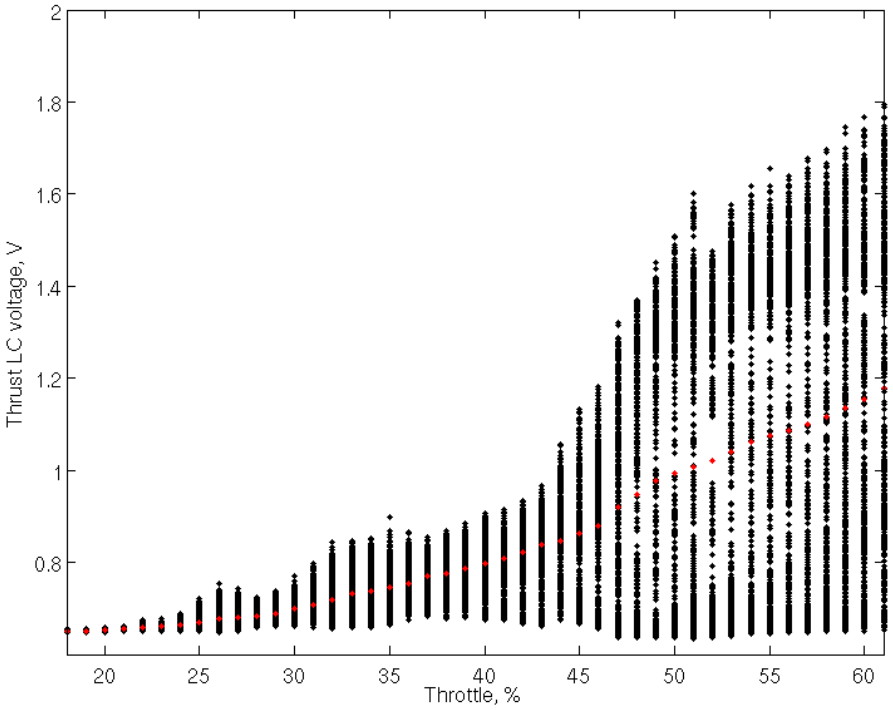


Figure 5.8: Trimmed thrust data from motor 4.

5.4 Torque

As with the thrust, the encoder readouts for torque were inspected for reasonability in Figures 5.9, 5.10, 5.11, and 5.12. The saturation points are known so the points are only visually verified.

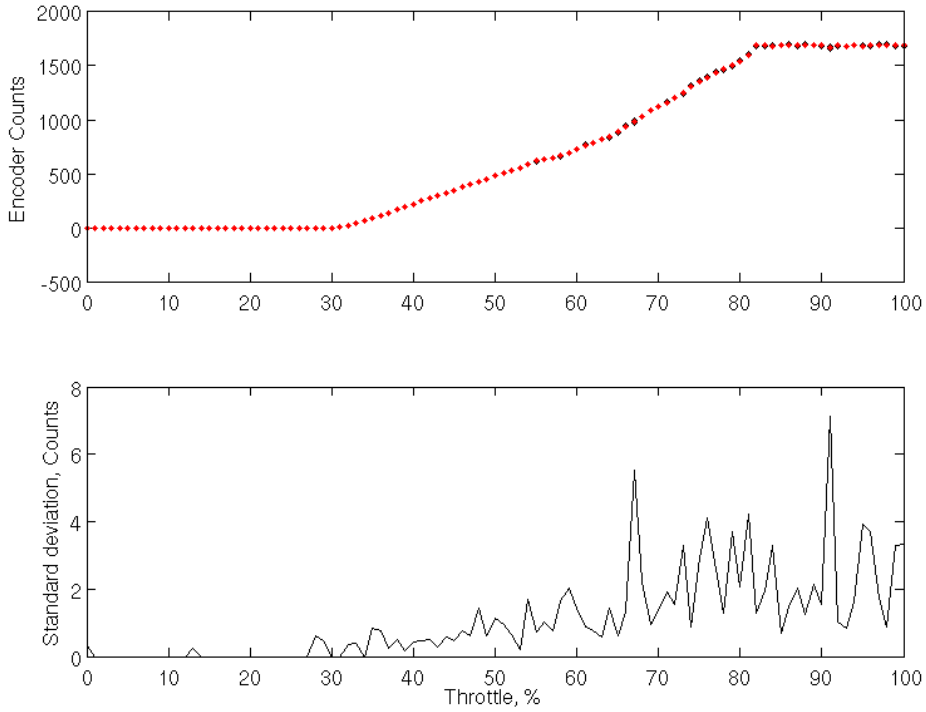


Figure 5.9: Collected torque data from motor 1.

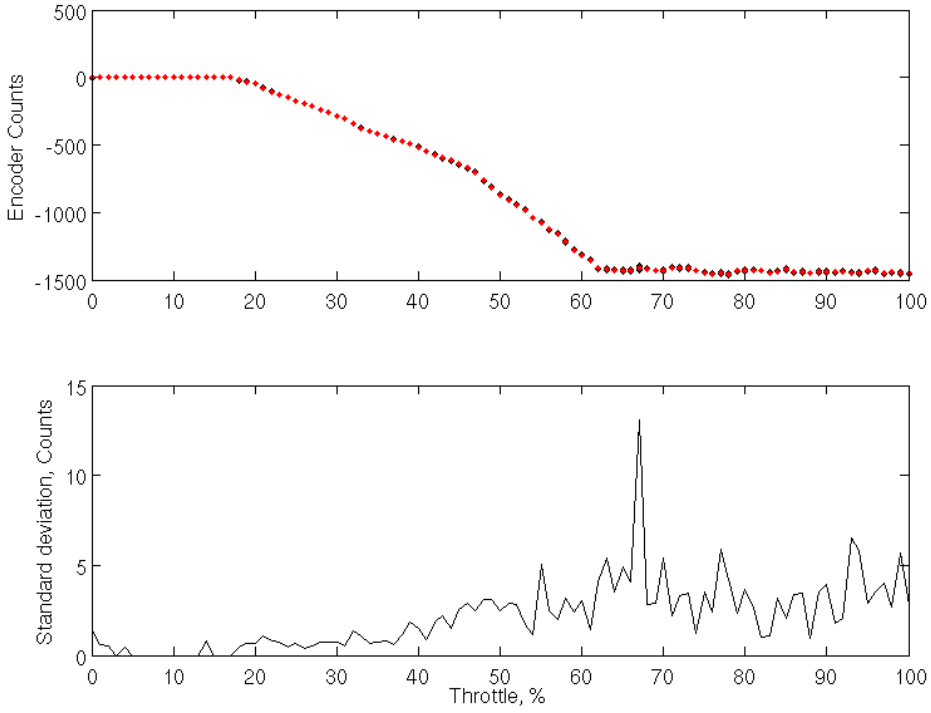


Figure 5.10: Collected torque data from motor 2.

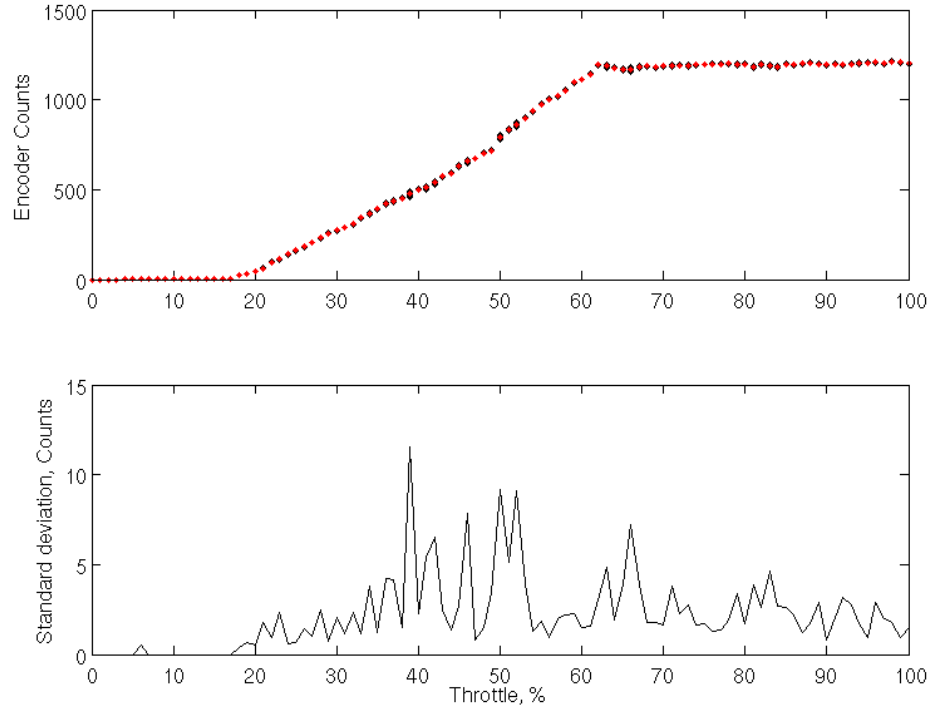


Figure 5.11: Collected torque data from motor 3.

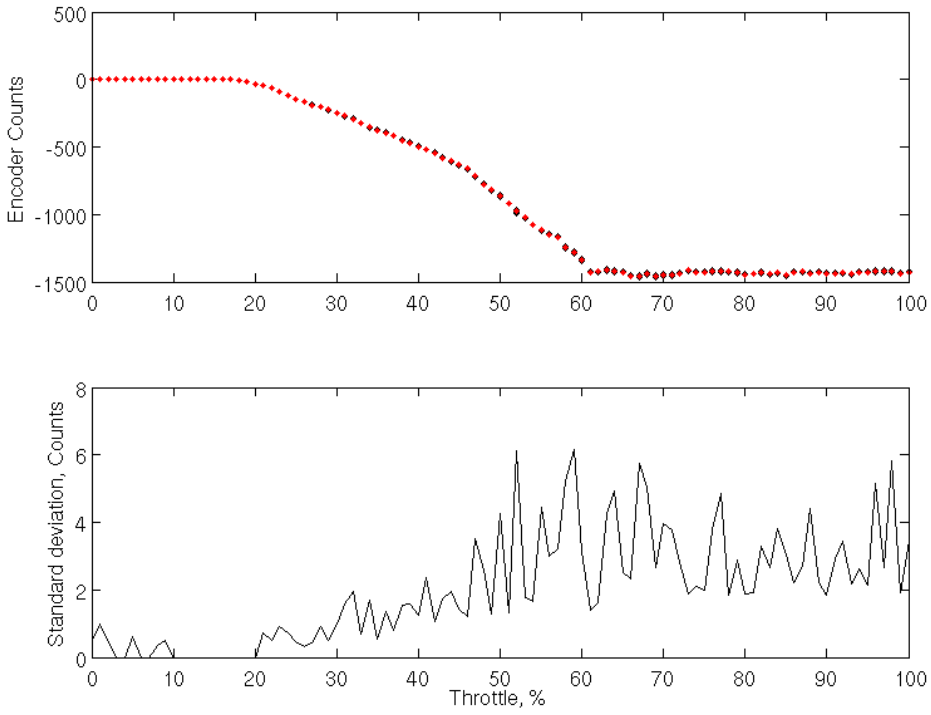


Figure 5.12: Collected torque data from motor 4.

Unlike with the thrust, the torque response may be either positive or negative. This is expected, since motors 1 and 3 spin opposite of motors 2 and 4; the sign denotes direction. As expected, as the motor spins faster, deflection increases until it hits the upper saturation point. As with the thrust data, the torque data outside the motor ranges is trimmed, yielding Figures 5.13, 5.14, 5.15, and 5.16.

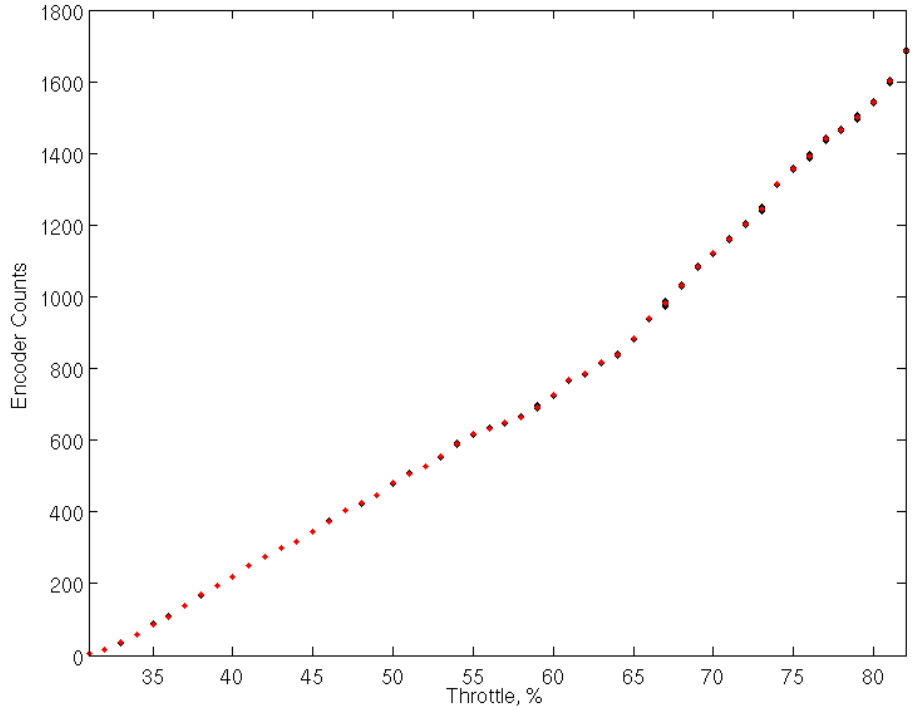


Figure 5.13: Trimmed torque data from motor 1.

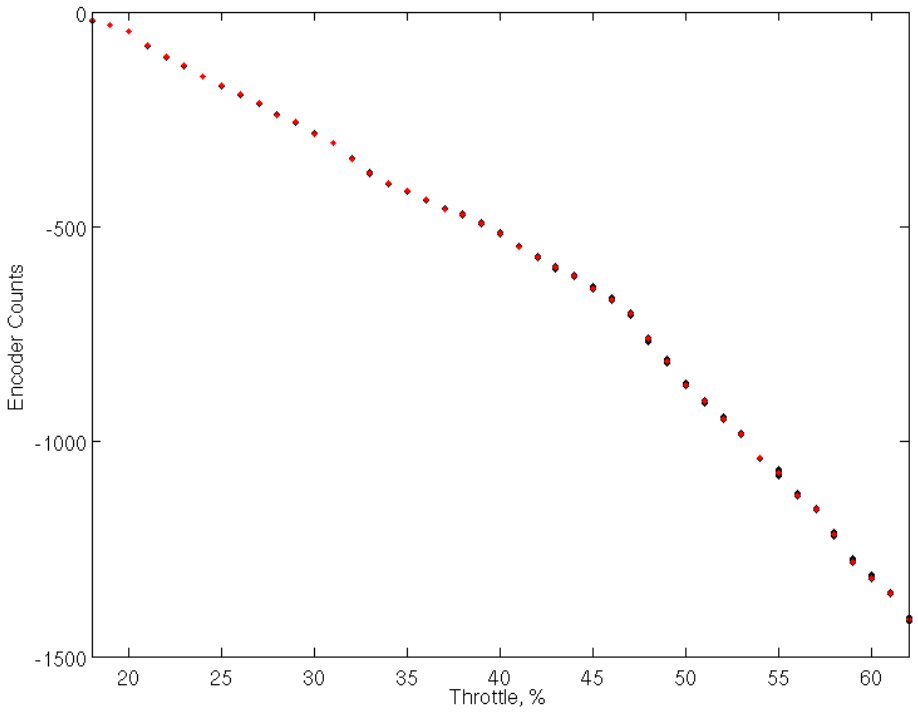


Figure 5.14: Trimmed torque data from motor 2.

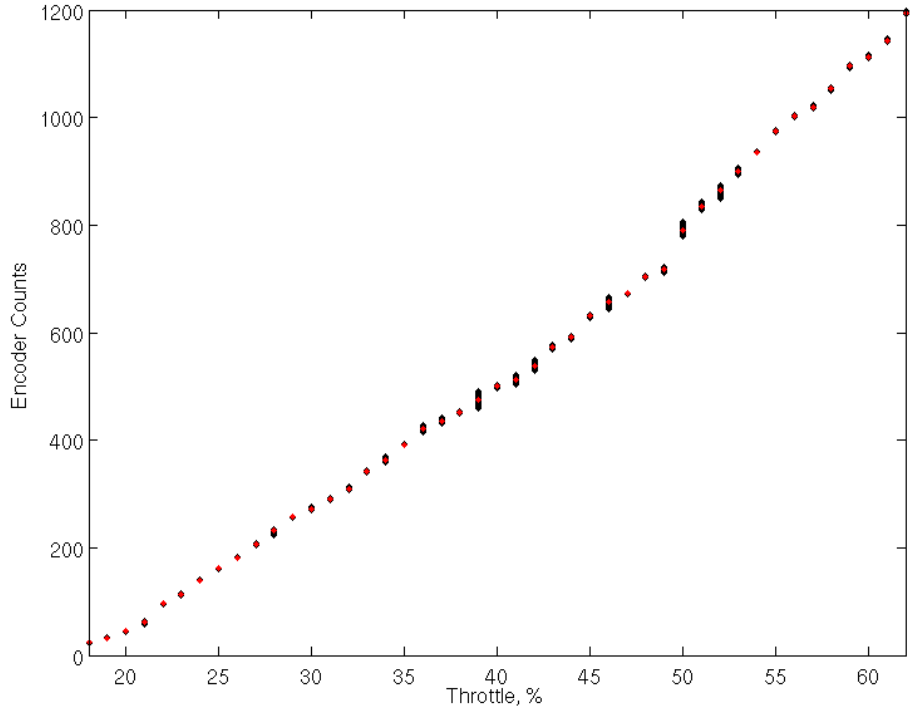


Figure 5.15: Trimmed torque data from motor 3.

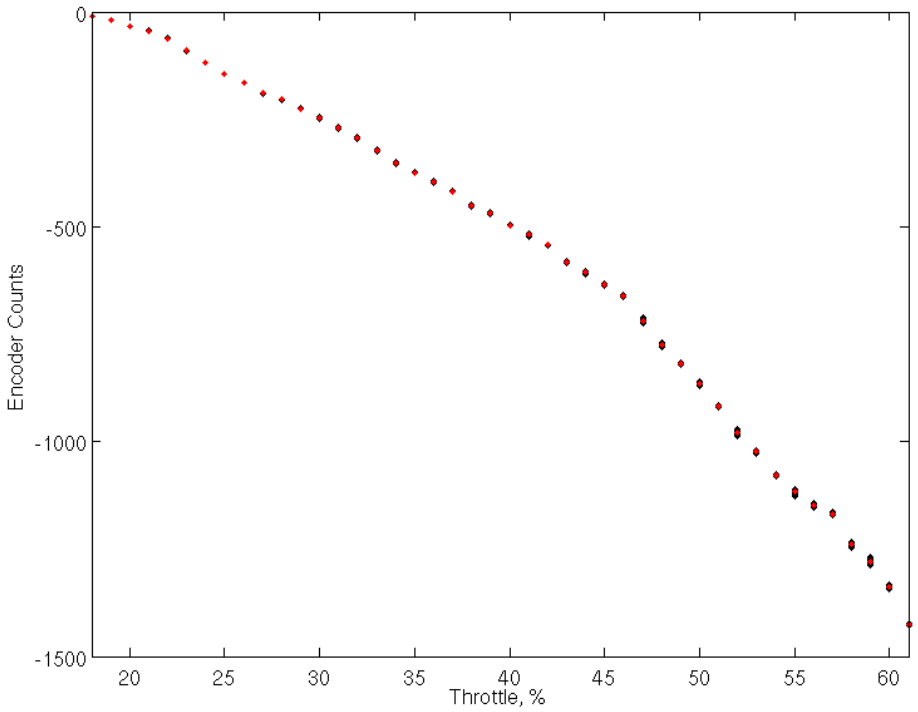


Figure 5.16: Trimmed torque data from motor 4.

5.5 Thrust Curve Fitting

Now that all the data is trimmed between the saturation points, curves have to be fit. As previously explained in Chapter 4, a curve should not be just fitted. Once again, the third and fifth assumption can be assumed to be true. This leaves the first, second, and fourth assumptions to be verified.

The first set of curves that will be fit are the thrust curves. After a bit of experimentation, a second order polynomial fit was chosen, as done in Figures 5.17, 5.18, 5.19, and 5.20.

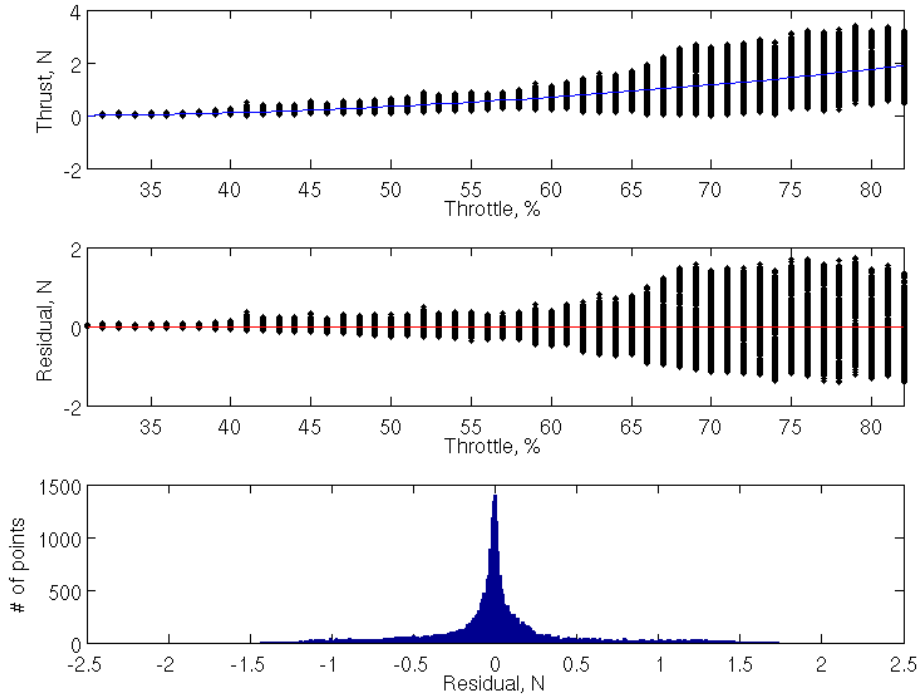


Figure 5.17: Motor 1 thrust curve.

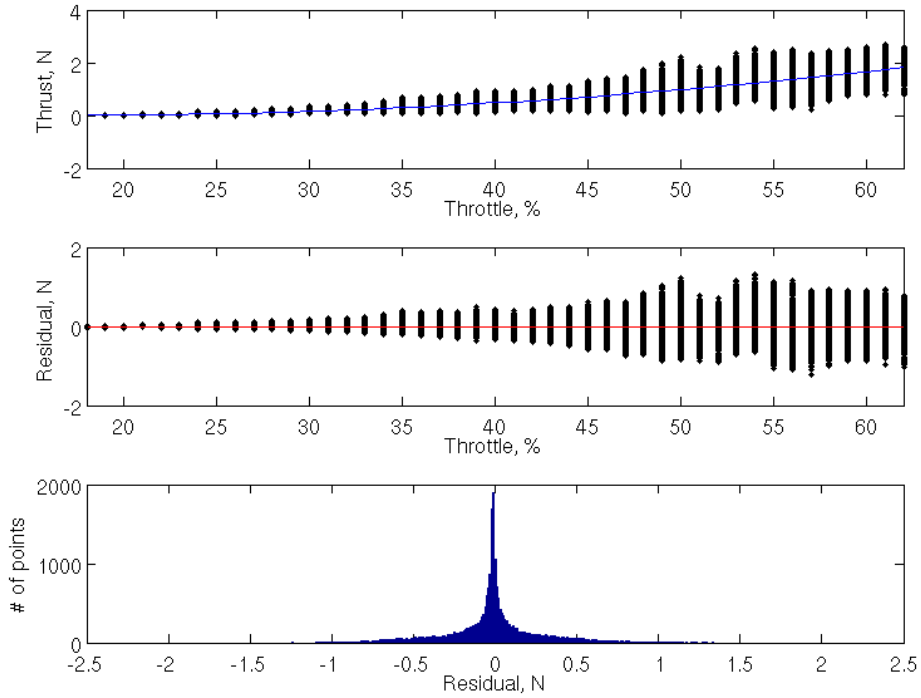


Figure 5.18: Motor 2 thrust curve.

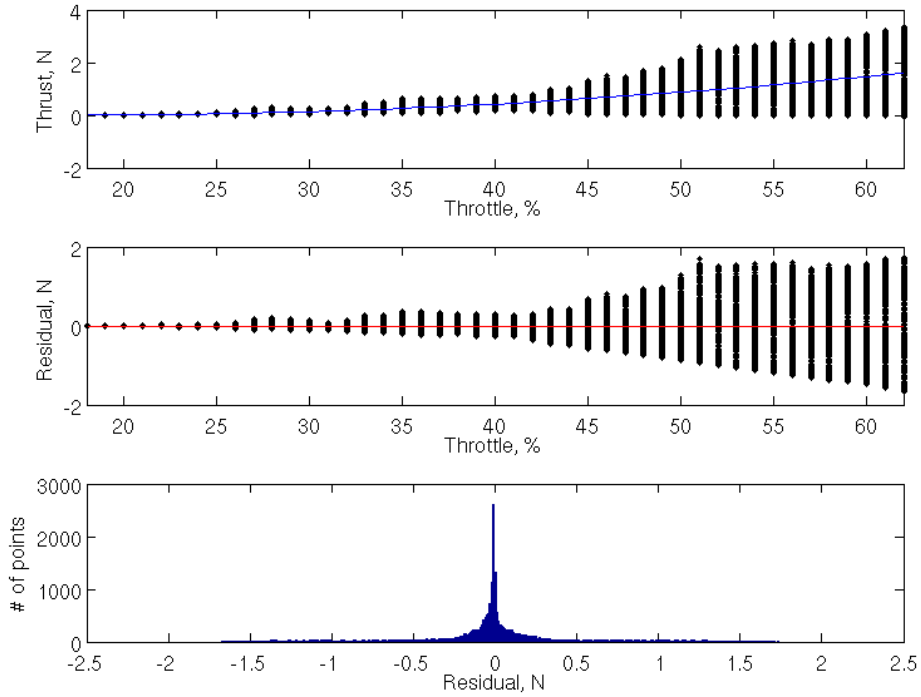


Figure 5.19: Motor 3 thrust curve.

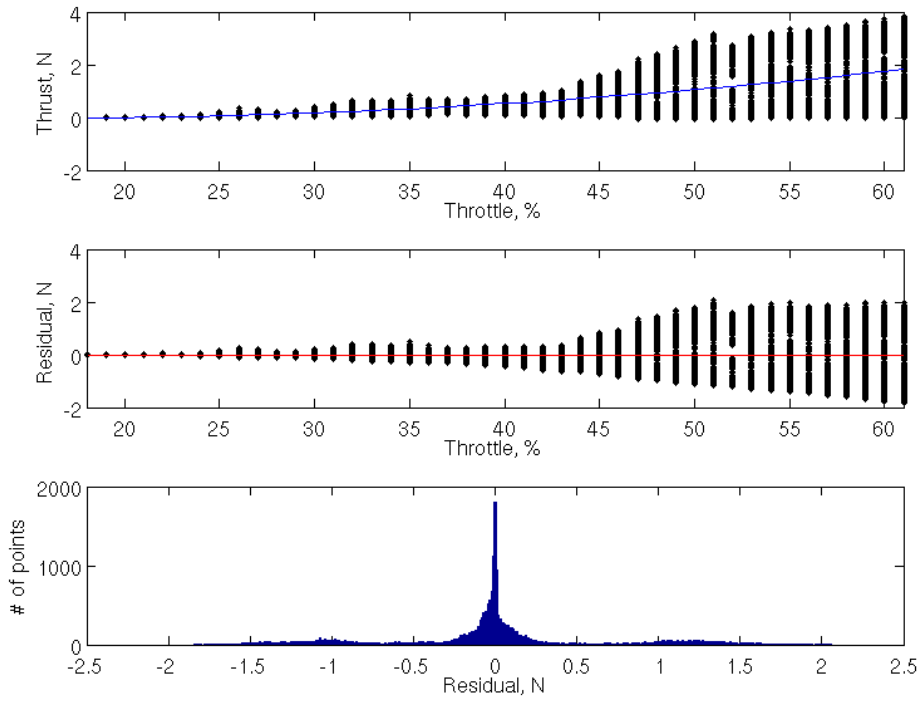


Figure 5.20: Motor 4 thrust curve.

Based on the residuals, it is apparent that the first assumption that the mean of the errors is centered around zero is valid. The fourth assumption, that the errors are normally distributed also holds, as the histograms have a nice Gaussian profile. However, the second assumption, homoscedastic data, does not hold as the variance increases with the throttle.

A phenomena of this sort is usually indicative that the data is logarithmic, exponential, or non-algebraic non-linear in nature. There are two ways to deal with such a phenomenon. The proper way is to have the data undergo a transformation. The transformation would account for the data's non-linearities and would thus correct the data's homoscedasticity problem. This is the formal and correct way of fixing the data. This will not be done.

The second way to deal with such a phenomena is to continue ahead with what was obtained, even if the model inherently has flaws. This is what was done here. There are a few factors that led to this choice. First, the time cost that would need to be spent to deduce non-algebraic non-linearities is significantly higher and could be better spent. The second reason is that the model will have inherent inaccuracies to it. As will be covered in Chapter 6, aerodynamic effects are omitted from the model. Therefore, the model will already have inherent inaccuracies. That is ok, as these inaccuracies are expected to be small.

This leads to the third point, that upon visual inspection, the model still accurately depicts the observed trend. Because the model will be used in conjunction with a sensor, the inaccuracy should be corrected for. Finally, even at the cost of accuracy, a second order polynomial model is significantly preferred over non-linear models due to a significant reduction of computational cost and ease of algebraic manipulation.

Thus, while it is important to realize that the model is not the perfect representation of what is going on with the motors, it very well represents the trend of what is occurring and that it is the best model for the end application.

5.6 Torque Curve Fitting

The second set of curves that will be fit are the torque curves. A second order polynomial fit was chosen and fit in Figures C.9, C.10, C.11, and C.12.

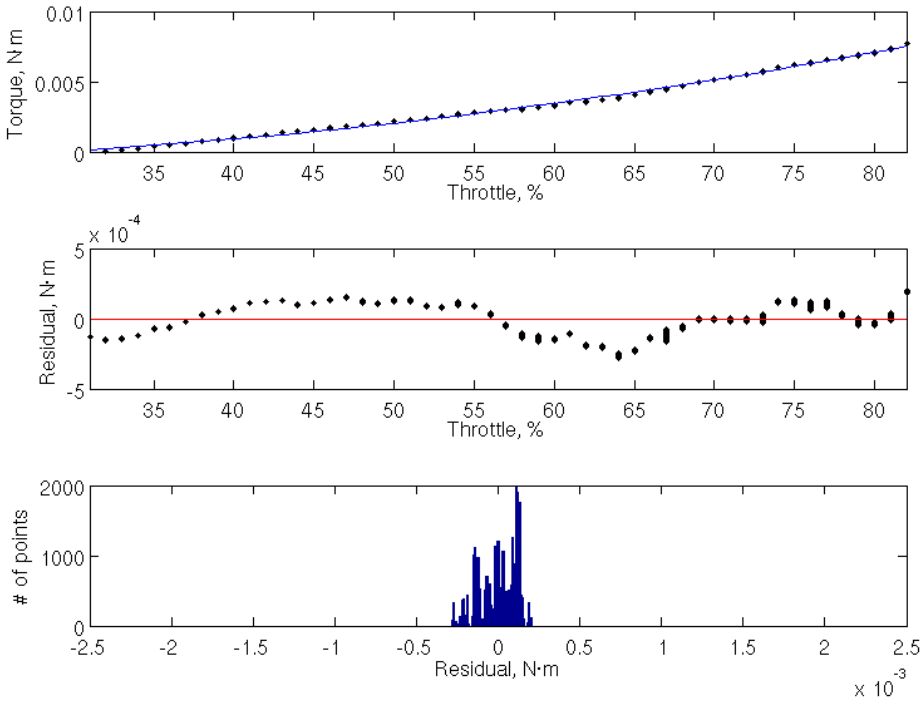


Figure 5.21: Motor 1 torque curve.

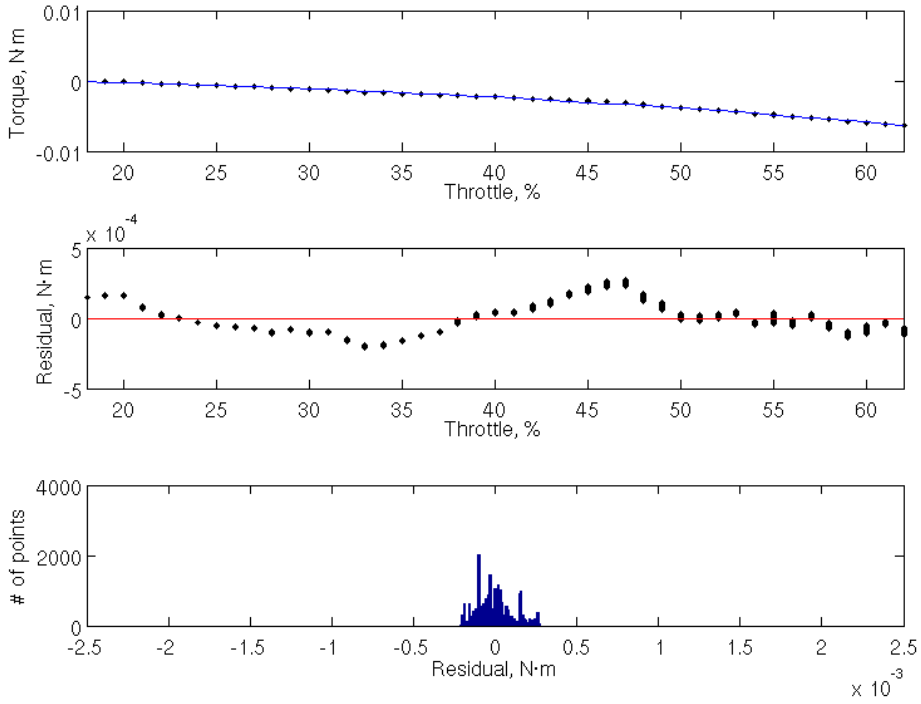


Figure 5.22: Motor 2 torque curve.

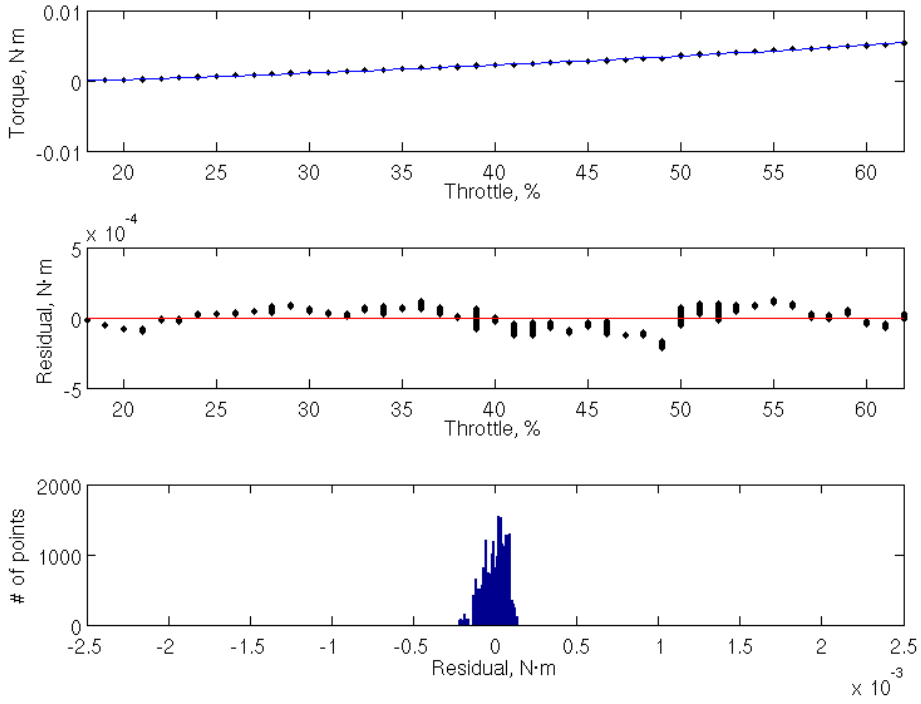


Figure 5.23: Motor 3 torque curve.

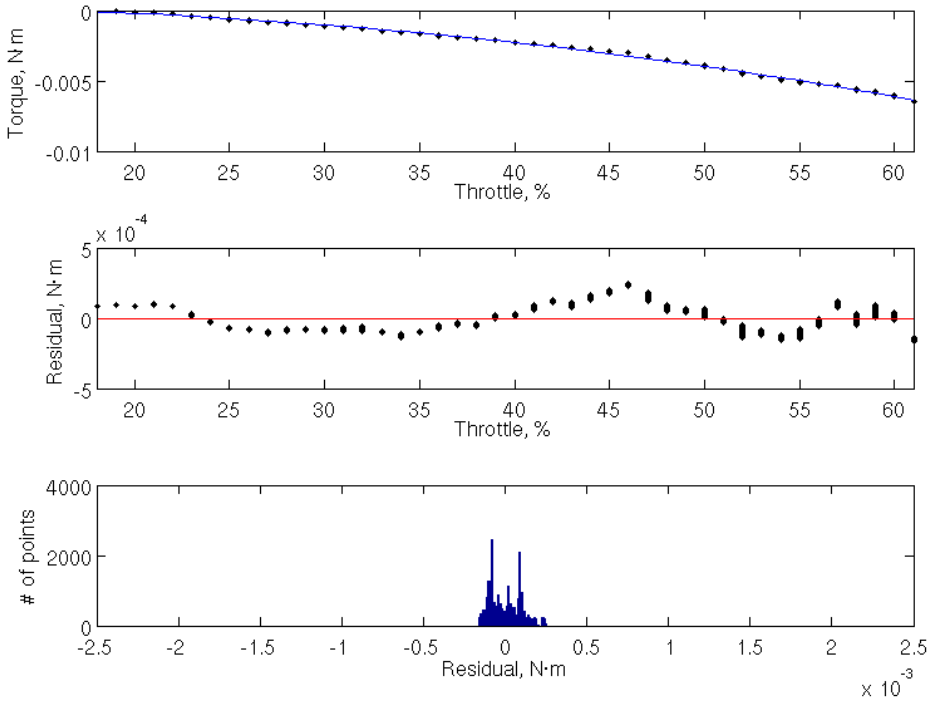


Figure 5.24: Motor 4 torque curve.

Once again, some of the underlying assumptions are not met. While the assumption that the residuals being homoscedastic appears to be in order, the mean of the errors is not zero. A comment can not formally be made in favor or against the data's normality, as the way it is being checked is visual.

Again, the "good enough" argument will be used, as visually inspecting the data the overall trend is relatively well matched. While it does not perfectly model the viewed performance, it will be ample for a model that is used in conjunction with an on-board sensor. The benefits from spending additional time and using a more computationally expensive model are not justified.

5.7 Results

Overall, the general trend of the motor performance was captured. These curves were then plotted against each other in Figure 5.25. The curves will help improve the model's accuracy.

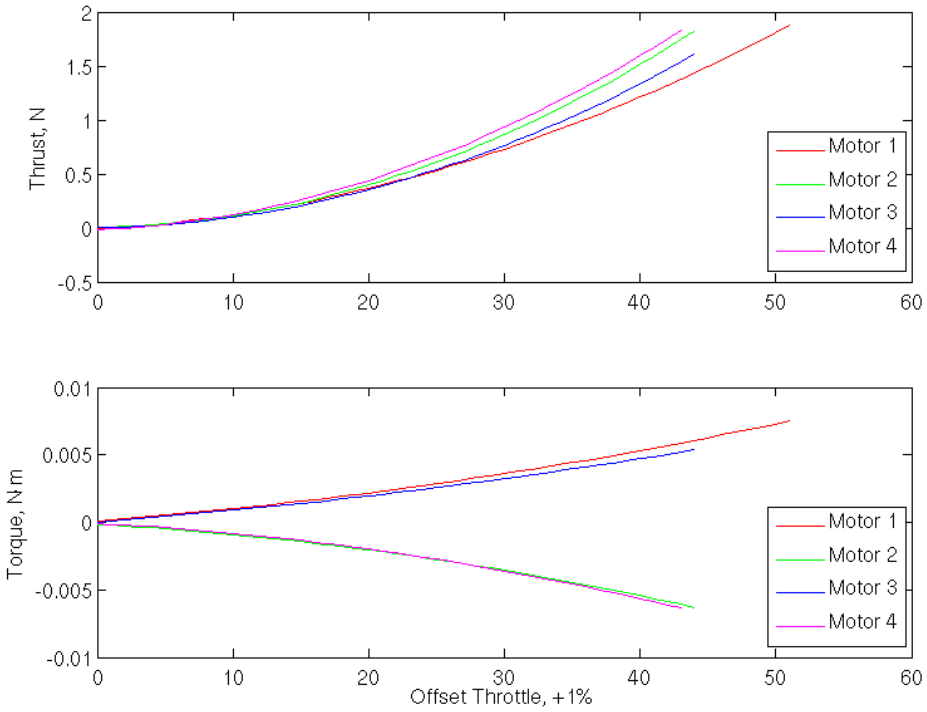


Figure 5.25: All the thrust and torque curves next to each other, with throttle standardized.

Table 5.1: Table of the found performance characteristics. For the curves, the values are components of a polynomial, as described in equation (5.1). For the saturation values, A is the saturation point at the low end while B is at the high end.

Motor	Characteristic	A	B	C
Motor 1	Saturation	31	82	-
	Thrust Curve (N)	5.886725e-04	-2.956450e-02	3.441971e-01
	Torque Curve (N·m)	1.360575e-06	-9.062366e-06	-8.977076e-04
Motor 2	Saturation	18	62	-
	Thrust Curve (N)	9.065253e-04	-3.120432e-02	2.748603e-01
	Torque Curve (N·m)	-1.867634e-06	9.761993e-06	2.801106e-04
Motor 3	Saturation	18	62	-
	Thrust Curve (N)	7.807968e-04	-2.598113e-02	2.155322e-01
	Torque Curve (N·m)	1.005201e-06	4.116699e-05	-1.050385e-03
Motor 4	Saturation	18	61	-
	Thrust Curve (N)	8.723219e-04	-2.593143e-02	1.671722e-01
	Torque Curve (N·m)	-2.213927e-06	2.939343e-05	9.939724e-05

$$(Output) = A(Throttle)^2 + B(Throttle) + C \tag{5.1}$$

Chapter 6

Quadcopter Flight Mechanics

6.1 Introduction

The flight mechanics of a quadcopter is a topic which has been already thoroughly explored by many people. As each author had different objectives in their research and different sets of hardware, the model presented at the end varies. In [14], [18], and [19], the thrust and torque generated by the motors are modeled through the use of coefficients and the speeds of the motors. In [20] and [21], the aerodynamic characteristics of the quadcopter are added to the model. The models presented in [22] and [23] has axis orientation along the arms of the quadcopter, not in the direction of movement.

The derivation presented here is based on all the works listed above. For the purposes of this paper and this research, no effort was made to attempt to model effects of air on the quadcopter's body. The thrust and torque characteristics of the motors were identified experimentally.

The overall objective here is take the well established theory and tailor it to be consistent with the notation and directions shown in Figure 6.1.

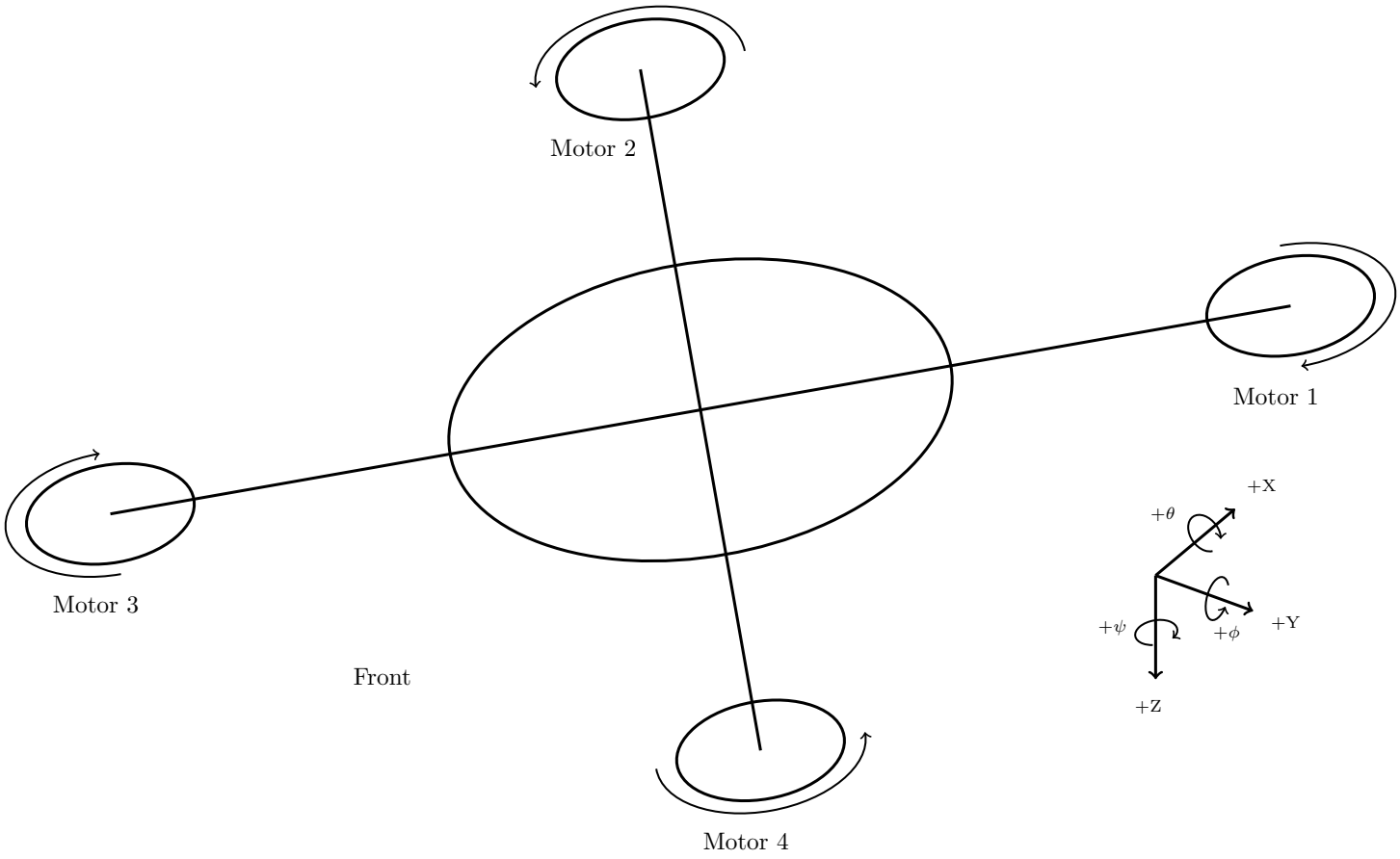


Figure 6.1: Diagram of the notation of the quadcopter.

6.2 Rotational Matrices

To accurately model the quadcopter as it moves through space, the vehicle pose is one of the more fundamental elements that need to be modeled. Fortunately, this can be done through well established rotational matrices. The rotation of the quadcopter about the X-axis is modeled as

$$\mathbf{R}_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}. \tag{6.1}$$

About the Y-axis, this is mathematically written as

$$\mathbf{R}_Y = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}. \tag{6.2}$$

Finally, about the Z-axis, the rotation of quadcopter is written as

$$\mathbf{R}_Z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.3}$$

By multiplying (6.1), (6.2), and (6.3) together, the effects of rotation on the quadcopter can be represented as

$$\mathbf{R} = \mathbf{R}_X\mathbf{R}_Y\mathbf{R}_Z = \begin{bmatrix} \cos\phi\cos\psi & -\cos\phi\sin\psi & \sin\phi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\theta \\ \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi & \cos\phi\cos\theta \end{bmatrix}. \tag{6.4}$$

6.3 Translational Flight Mechanics

When creating a model, the thing that often comes to mind is modeling the translational effects, or the movement along the axes.

A couple of forces dominate the translational flight mechanics of a quadcopter. Most notable is the thrust generated by the four motors. Their effects can be mathematically written as

$$\mathbf{F} = \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -F_1 - F_2 - F_3 - F_4 \end{bmatrix}. \tag{6.5}$$

Another important force that acts on the quadcopter is gravity. This is represented as

$$m\mathbf{g} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}. \tag{6.6}$$

To derive the translational flight mechanics, conservation of linear momentum is used.

$$\frac{\partial P_{sys}}{\partial t} = \sum Forces + \sum_{in} \dot{m}V - \sum_{out} \dot{m}V \tag{6.7}$$

Since there should be no mass lost or added as the quadcopter flies, this simplifies (6.7). Dividing (6.7) with no change in mass yields

$$\ddot{V} = \frac{\sum Forces}{m} \tag{6.8}$$

At this point (6.5) and (6.6) are added to (6.8). However, it is important to realize that the direction of thrust changes as quadcopter rotates around in space. To account for the change in thrust vector, (6.4) needs to be used. All these equations combined produce

$$\ddot{V} = \begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \mathbf{R}\frac{\mathbf{F}}{m} + \mathbf{g}. \tag{6.9}$$

While not included in this derivation, the inclusion of aerodynamic effects would be added onto (6.9). These effects would most likely act as a damper on the system.

This model can then be expanded out to

$$\ddot{V} = \begin{bmatrix} \sin\phi(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} \\ -\cos\phi\sin\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} \\ \cos\phi\cos\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} + g \end{bmatrix}. \tag{6.10}$$

This puts the equations in a form which will make it easier to work with later on.

To get the velocity, the acceleration identified in (6.9) needs to be integrated over time.

$$\dot{V} = \int \ddot{V}\delta t + \dot{V}_0 \tag{6.11}$$

The most practical way to go about integrating is to use a Euler approximation. With a small enough timestep, dt, any errors present should be negligible.

$$\dot{V}_{n+1} = \ddot{V}_k dt + \dot{V}_k \tag{6.12}$$

The same idea holds for integrating velocity into a position.

$$V_{n+1} = \dot{V}_k dt + V_k \tag{6.13}$$

This results in equations (6.10), (6.12), and (6.13) serving as the model for the translational movements of the quadcopter.

6.4 Angular Flight Mechanics

The second set of movements that need to be modeled are the angular dynamics; the rotations around the axis.

A critical parameter in the modeling of the angular flight mechanics are the moments of inertia.

$$\overline{I} = \begin{bmatrix} I_{XX} & I_{XY} & I_{XZ} \\ I_{YX} & I_{YY} & I_{YZ} \\ I_{ZX} & I_{ZY} & I_{ZZ} \end{bmatrix} \tag{6.14}$$

A standard assumption is that the quadcopter is symmetric about all of its axes. This simplifies (6.14) down to

$$\overline{I} = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix}. \tag{6.15}$$

Knowledge of the torques on the system is also required. Along the X and Y-axis, the torques are a result of the thrust from the motors. Along the Z-axis, the torques are a result of the torque reaction from the propellers spinning. This, mathematically described, is

$$\overline{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} d(F_1 + F_2 - F_3 - F_4) \\ \frac{\sqrt{2}}{2} d(F_2 + F_3 - F_1 - F_4) \\ \tau_1 + \tau_2 + \tau_3 + \tau_4 \end{bmatrix}. \tag{6.16}$$

With these terms identified, derivation can begin. The starting point will be at the conservation of angular momentum.

$$\frac{\partial L_{sys}}{\partial t} = \sum Moments + \sum_{in} (r \times V)\dot{m} - \sum_{out} (r \times V)\dot{m} \tag{6.17}$$

As in (6.7), mass should not be added or lost from the quadcopter during flight. The moments of the quadcopter consist of the torques, as well as the effects of inertia. Thus, combining (6.16) with (6.15) into (6.17) results in

$$\overline{I}\ddot{\Omega} = \overline{\tau} + \dot{\Omega} \times \overline{I}\dot{\Omega}. \tag{6.18}$$

To obtain the angular acceleration rates, a little more rearranging is required.

$$\ddot{\Omega} = \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \overline{I}^{-1} \left[\overline{\tau} + \dot{\Omega} \times \overline{I}\dot{\Omega} \right] \tag{6.19}$$

Once again, the effects of aerodynamics would be added at this point, most likely acting as dampers on the system.

When (6.19) is expanded out, it yields

$$\ddot{\Omega} = \begin{bmatrix} \frac{\sqrt{2}}{2} d(F_1 + F_2 - F_3 - F_4) \frac{1}{I_{xx}} + \dot{\theta}\dot{\psi} \left(\frac{I_{zz} - I_{yy}}{I_{xx}} \right) \\ \frac{\sqrt{2}}{2} d(F_2 + F_3 - F_1 - F_4) \frac{1}{I_{yy}} + \dot{\phi}\dot{\psi} \left(\frac{I_{xx} - I_{zz}}{I_{yy}} \right) \\ (\tau_1 + \tau_2 + \tau_3 + \tau_4) \frac{1}{I_{zz}} + \dot{\phi}\dot{\theta} \left(\frac{I_{yy} - I_{xx}}{I_{zz}} \right) \end{bmatrix}. \tag{6.20}$$

From here, like with (6.9), it is a matter of integration. As done in (6.12) and (6.13), it is most practical to do this integration though the use of Euler approximations.

$$\dot{\Omega}_{k+1} = \ddot{\Omega}_k dt + \dot{\Omega}_k \tag{6.21}$$

$$\Omega_{k+1} = \dot{\Omega}_k dt + \Omega_k \tag{6.22}$$

This results in equations (6.20), (6.21), and (6.22) serving as the model for the angular movements of the quadcopter.

Chapter 7

Design of the Kalman Filter

7.1 Introduction

As the quadcopter flies in the air, knowledge of its position relative to the rest of the world is very important. A model can be used to compute the state of the quadcopter. However, if the model is slightly off or doesn't account for external factors, the results will be inaccurate. The position could also be computed with a sensor. However, sensors are prone to noise, diminishing their accuracy. By combining both a model and sensors together, the accuracy of both gets improved. This process of combining the two is called a Kalman filter.

7.2 The Kalman Filter

The Kalman filter has become one of the most popular algorithms in signal processing. First introduced in 1960 [24] for linear systems, it was soon expanded for non linear systems. Of these extensions, the two most popular are the Extended Kalman Filter [25] and the Unscented Kalman Filter [26]. What makes the Kalman filter so popular is its ability to clean up sensor data with a large degree of success, while retaining a relatively low computational cost. Today, the use of Kalman filters has become a staple in navigation and information processing [27].

The filter works by using a prediction from a fairly accurate model and coupling it with sensor readout. By combining the results, they are much more accurate.

Kalman filters have been used from applications as varied as satellites, airplanes, and cellphones. Given the range of applications the algoritm is used in, it does not take a wild imagination to think that a Kalman

filter could be implemented on a quadcopter. In fact, the idea of using a Kalman filter in a quad copter was explored in [28].

7.3 Implementing the Extended Kalman Filter

The system which the filter is supposed to work for is non-linear. This means that either the system must be linearized to use the standard Kalman filter or that a non-linear Kalman filter must be used. While a linearization was attempted, it yielded poor estimates. Thus, the Extended Kalman filter was chosen to be used. To implement the Kalman filter as described, some set up is required. The steps shown here are adapted from what is shown by Lewis [29] and has been streamlined for implementation on this system.

The filter starts with system and measurement models

$$\chi = a(\chi, u) + \mathbf{C}w \tag{7.1}$$

and

$$Z_k = h[\chi, k] + v_k, \tag{7.2}$$

where

$$\chi(0) \sim (\chi_0, P_0), w \sim (0, \mathbf{Q}_{kf}), v_k \sim (0, \mathbf{R}_{kf}) \tag{7.3}$$

The model, based on equations (6.10), (6.12), (6.13), (6.20), (6.21), and (6.22), estimate the states

$$\hat{\chi} = \begin{bmatrix} X \\ \dot{X} \\ \ddot{X} \\ Y \\ \dot{Y} \\ \ddot{Y} \\ Z \\ \dot{Z} \\ \ddot{Z} \\ \phi \\ \dot{\phi} \\ \ddot{\phi} \\ \theta \\ \dot{\theta} \\ \ddot{\theta} \\ \psi \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = a(\chi, u) = \begin{bmatrix} X + \dot{X}dt \\ \dot{X} + \ddot{X}dt \\ \sin\phi(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} \\ Y + \dot{Y}dt \\ \dot{Y} + \ddot{Y}dt \\ -\cos\phi\sin\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} \\ Z + \dot{Z}dt \\ \dot{Z} + \ddot{Z}dt \\ \cos\phi\cos\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m} + g \\ \phi + \dot{\phi}dt \\ \dot{\phi} + \ddot{\phi}dt \\ \frac{\sqrt{2}}{2}d(F_1 + F_2 - F_3 - F_4)\frac{1}{I_{xx}} + \dot{\theta}\dot{\psi}\left(\frac{I_{zz} - I_{yy}}{I_{xx}}\right) \\ \theta + \dot{\theta}dt \\ \dot{\theta} + \ddot{\theta}dt \\ \frac{\sqrt{2}}{2}d(F_2 + F_3 - F_1 - F_4)\frac{1}{I_{yy}} + \dot{\phi}\dot{\psi}\left(\frac{I_{xx} - I_{zz}}{I_{yy}}\right) \\ \psi + \dot{\psi}dt \\ \dot{\psi} + \ddot{\psi}dt \\ (\tau_1 + \tau_2 + \tau_3 + \tau_4)\frac{1}{I_{zz}} + \dot{\phi}\dot{\theta}\left(\frac{I_{yy} - I_{xx}}{I_{zz}}\right) \end{bmatrix} \tag{7.4}$$

To add the sensor readings to the estimate, the Kalman gain needs to be computed. To compute the Kalman gain, L, the covariance of the error, P, must be computed. The Kalman gain and covariance are computed for every time iteration as

$$P_{k^-} = \left[\frac{\partial a}{\partial \chi}\right] P_{k-1} + P_{k-1}\left[\frac{\partial a}{\partial \chi}\right]^T + \mathbf{Q}_{kf} \tag{7.5}$$

$$\mathbf{L}_k = P_{k^-}\mathbf{C}^T\left[\mathbf{C}\mathbf{P}_{k^-}\mathbf{C}^T + \mathbf{R}_{kf}\right]^{-1} \tag{7.6}$$

$$P_k = \left[\mathbf{I}_{(18,18)} - \mathbf{L}_k\mathbf{C}P_{k^-}\right] \tag{7.7}$$

The P matrix needs to have a starting condition for the computations to go through. What was found

to work well was to set the initial matrix to zeros.

Looking at (7.5), (7.6), and (7.7), a few more terms need to be defined. The Jacobian of the estimate, a, needs to be computed and updated for every iteration, as it is affected by the states of the quadcopter. When computed out, the Jacobian for the system is found to be

$$\frac{\partial a}{\partial \chi} = \begin{bmatrix}
1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_3}{\partial \chi_{10}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_6}{\partial \chi_{10}} & 0 & 0 & \frac{\partial a_6}{\partial \chi_{13}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_9}{\partial \chi_{10}} & 0 & 0 & \frac{\partial a_9}{\partial \chi_{13}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_{12}}{\partial \chi_{14}} & 0 & 0 & \frac{\partial a_{12}}{\partial \chi_{17}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_{15}}{\partial \chi_{11}} & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_{15}}{\partial \chi_{17}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial a_{18}}{\partial \chi_{11}} & 0 & 0 & \frac{\partial a_{18}}{\partial \chi_{14}} & 0 & 0 & 0 & 0
\end{bmatrix} \tag{7.8}$$

where

$$\frac{\partial a_3}{\partial \chi_{10}} = \cos\phi(-F_1 - F_2 - F_3 - F_4)\frac{1}{m}, \tag{7.9}$$

$$\frac{\partial a_6}{\partial \chi_{10}} = \sin\phi\sin\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m}, \tag{7.10}$$

$$\frac{\partial a_6}{\partial \chi_{13}} = -\cos\phi\cos\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m}, \tag{7.11}$$

$$\frac{\partial a_9}{\partial \chi_{10}} = -\sin\phi\cos\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m}, \tag{7.12}$$

$$\frac{\partial a_9}{\partial \chi_{13}} = -\cos\phi\sin\theta(-F_1 - F_2 - F_3 - F_4)\frac{1}{m}, \tag{7.13}$$

$$\frac{\partial a_{12}}{\partial \chi_{14}} = \dot{\psi}\left(\frac{I_{zz} - I_{yy}}{I_{xx}}\right), \tag{7.14}$$

$$\frac{\partial a_{12}}{\partial \chi_{17}} = \dot{\theta}\left(\frac{I_{zz} - I_{yy}}{I_{xx}}\right), \tag{7.15}$$

$$\frac{\partial a_{15}}{\partial \chi_{11}} = \dot{\psi}\left(\frac{I_{xx} - I_{zz}}{I_{yy}}\right), \tag{7.16}$$

$$\frac{\partial a_{15}}{\partial \chi_{17}} = \dot{\phi}\left(\frac{I_{xx} - I_{zz}}{I_{yy}}\right), \tag{7.17}$$

$$\frac{\partial a_{18}}{\partial \chi_{11}} = \dot{\theta}\left(\frac{I_{yy} - I_{xx}}{I_{zz}}\right), \tag{7.18}$$

and

$$\frac{\partial a_{18}}{\partial \chi_{14}} = \dot{\phi}\left(\frac{I_{yy} - I_{xx}}{I_{zz}}\right). \tag{7.19}$$

C also needs to be defined. While this is also a Jacobian, when computed out it is found that it isn't affected by time or by the states of the quadcopter.

$$\mathbf{C} = \frac{\partial c}{\partial \chi} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{7.20}$$

$\mathbf{Q}_{kf}$ and $\mathbf{R}_{kf}$, need to be also set. The values chosen were identified experimentally, through the use of the simulation. What was found was that setting them to large values worked. Equations (7.21) and (7.22) show the values used in the simulations.

$$\mathbf{Q}_{kf} = \mathbf{I}_{(18,18)} \times 1000 \tag{7.21}$$

$$\mathbf{R}_{kf} = \mathbf{I}_{(6,6)} \times 1000 \tag{7.22}$$

With all the components found to compute the Kalman gain, the Kalman estimate can be computed. The estimate is defined as

$$\hat{\chi}_{k+1} = a(\chi, u)_k + \mathbf{L}_k(Sens_k - \mathbf{C}a(\chi, u)_k) \tag{7.23}$$

Equation (7.23), when shown in a block diagram, is represented by Figure 7.1.

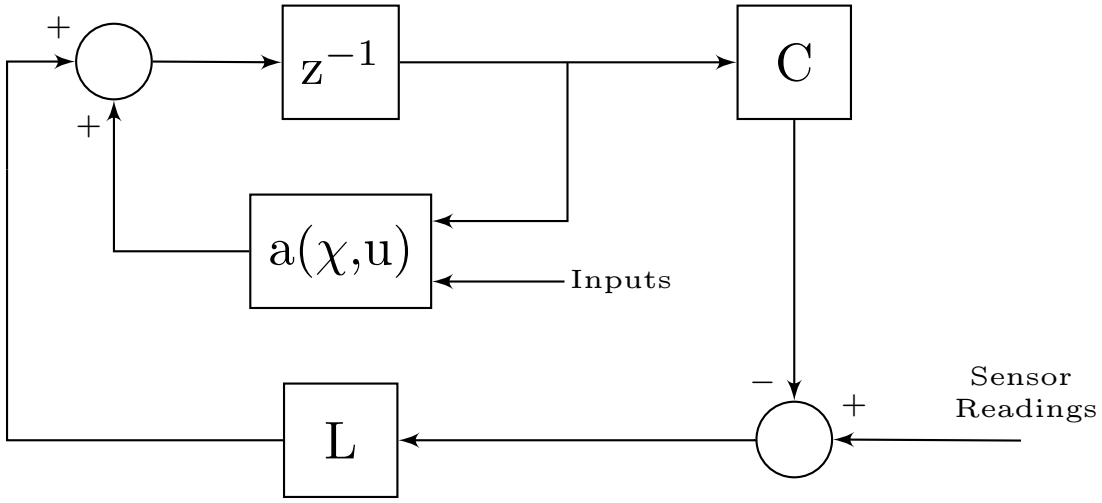


Figure 7.1: Block diagram of the Discrete Extended Kalman Filter used.

Chapter 8

Design of the Linear Quadratic Regulator

8.1 Introduction

To direct the flight of the quadcopter, a controller needs to be implemented. As the name suggests, a controller controls the movements of a dynamic system to desired states. Many different types of controls algorithms have been developed, where PID, Fuzzy, and optimal are some of the big groups. In this work, an optimal controller is implemented.

8.2 The Linear Quadratic Regulator

One of the most well known family of optimal controllers that exists is the linear quadratic regulator. As the name suggests, an optimal controller is a controller which creates a response that minimizes a cost function. This cost function takes into consideration how far the desired states are from the target states and how aggressive the inputs into the system are. Depending on how it is set up, an end state penalty can be applied as well. The calculations that take place also vary depending whether the controller is in discrete time or if it is continuous.

For this work, an infinite-horizon time-invariant discrete linear quadratic regulator is implemented. In plain terms, this means that the controller does not have a end state penalty associated with it, that the terms do not change over time, and that it operates in discrete time.

8.3 Derivation of the Discreate Linear Quadratic Regulator

The steps shown here have been adapted from Datta's work [30], which in turn was adapted from Sage and White's work [31].

The derivation starts with a general discrete system equation

$$\chi_{k+1} = \mathbf{A}\chi_k + \mathbf{B}u_k, \tag{8.1}$$

and a cost function, defined as

$$J = \frac{1}{2}\sum_{k=0}^{\infty}(\chi_k^T \mathbf{Q}_{lqr}\chi_k + u_k^T \mathbf{R}_{lqr}u_k). \tag{8.2}$$

From these, a Hamiltonian is formed, defined as

$$H_k = \frac{1}{2}\chi_k^T \mathbf{Q}_{lqr}\chi_k + \frac{1}{2}u_k^T \mathbf{R}_{lqr}u_k + \lambda_{k+1}^T\left[\mathbf{A}\chi_k + \mathbf{B}u_k\right] \tag{8.3}$$

Due to standard assumptions, a non-trivial function exists that satisfies

$$\lambda_k = \frac{\partial H}{\partial \chi_k}. \tag{8.4}$$

Thus, this is used to define an adjoint vector equation

$$\lambda_k = \mathbf{Q}_{lqr}\chi_k + \mathbf{A}^T\lambda_{k+1} \tag{8.5}$$

An implied condition is that the partial of the Hamiltonian along the inputs is equal to zero. Thus,

$$\frac{\partial H}{\partial u_k} = 0 = \mathbf{R}_{lqr}u_k + \mathbf{B}^T\lambda_{k+1} \tag{8.6}$$

Substituting (8.6) into (8.3), the linear difference equations are identified to solve for an optimal solution.

$$\begin{bmatrix}\chi_{k+1}\\ \lambda_k\end{bmatrix} = \begin{bmatrix}\mathbf{A} & -\mathbf{B}\mathbf{R}_{lqr}^{-1}\mathbf{B}^T\\ \mathbf{Q}_{lqr} & \mathbf{A}^T\end{bmatrix}\begin{bmatrix}\chi_k\\ \lambda_{k+1}\end{bmatrix} \tag{8.7}$$

A solution is then guessed as

$$\lambda_k = P_k \chi_k, \tag{8.8}$$

which then (8.8) is substituted into (8.7).

$$\begin{bmatrix} \chi_{k+1} \\ \lambda_k \end{bmatrix} = \begin{bmatrix} \mathbf{A} & -\mathbf{B}\mathbf{R}_{lqr}^{-1}\mathbf{B}^T \\ \mathbf{Q}_{lqr} & \mathbf{A}^T \end{bmatrix} \begin{bmatrix} \chi_k \\ P_{k+1}\chi_{k+1} \end{bmatrix} \tag{8.9}$$

These equations are then solved for χ_{k+1}, eliminating it in the process, which results in

$$P_k \chi_k = \mathbf{Q}_{lqr}\chi_k + \mathbf{A}^T \left[P_{k+1}^{-1} + \mathbf{B}R^{-1}\mathbf{B}^T\right]^{-1} \mathbf{A}\chi_k \tag{8.10}$$

Then, through the use of the matrix inversion lemma, (8.10) can be rearranged as

$$\mathbf{A}^T P_{k+1}\mathbf{A} + \mathbf{Q}_{lqr} - \mathbf{A}^T P_{k+1}\mathbf{B}\left[\mathbf{R}_{lqr} + \mathbf{B}^T P_{k+1}\mathbf{B}\right]^{-1}\mathbf{B}^T P_{k+1}\mathbf{A} = P_k \tag{8.11}$$

Equation (8.11) is better known as the discrete algebraic Riccati equation. While the DARE could be used in its current state in the final solution, one step further can be taken in finding the steady state solution. The one big advantage of the steady state solution is that the control gain needs to be only computed once, versus every iteration. The steady state is defined by

$$P_{k+1} = P_k = P. \tag{8.12}$$

Implementing (8.12) changes (8.11) into

$$\mathbf{A}^T P\mathbf{A} - P + \mathbf{Q}_{lqr} - \mathbf{A}^T P\mathbf{B}\left[\mathbf{R}_{lqr} + \mathbf{B}^T P\mathbf{B}\right]^{-1}\mathbf{B}^T P\mathbf{A} = 0. \tag{8.13}$$

Solving (8.13) we can get P. The next step is to compute the feedback gain. The model, with the gain as illustrated in Figure 8.1, is

$$\chi_{k+1} = \mathbf{A}\chi_k + \mathbf{B}\mathbf{K}\chi_k. \tag{8.14}$$

To compute K, equation (8.13) is taken and rearranged. χ_k is also multiplied thru. This yields

$$\mathbf{A}^T P\mathbf{A}\chi_k - \mathbf{A}^T P\mathbf{B}\left[\mathbf{R}_{lqr} + \mathbf{B}^T P\mathbf{B}\right]^{-1}\mathbf{B}^T P\mathbf{A}\chi_k = P\chi_k - \mathbf{Q}_{lqr}\chi_k. \tag{8.15}$$

Now, the second equation in (8.9) and (8.8) can be both taken and rearranged, resulting in

$$P\chi_k = \mathbf{Q}_{lqr}\chi_k + \mathbf{A}^T P\chi_{k+1} \tag{8.16}$$

(8.16) can be rearranged some more.

$$P\chi_k - \mathbf{Q}_{lqr}\chi_k = \mathbf{A}^T P\chi_{k+1} \tag{8.17}$$

(8.17) can then be plugged into (8.15), yielding

$$\mathbf{A}^T P\mathbf{A}\chi_k - \mathbf{A}^T P\mathbf{B}\left[\mathbf{R}_{lqr} + \mathbf{B}^T P\mathbf{B}\right]^{-1}\mathbf{B}^T P\mathbf{A}\chi_k = \mathbf{A}^T P\chi_{k+1} \tag{8.18}$$

which, if conditioned a bit more, results in

$$\chi_{k+1} = \mathbf{A}\chi_k - \mathbf{B}\left[\mathbf{R}_{lqr} + \mathbf{B}^T P\mathbf{B}\right]^{-1}\mathbf{B}^T P\mathbf{A}\chi_k \tag{8.19}$$

(8.19) looks awfully similar to (8.14). Because of this, the feedback gain, $\mathbf{K}$, can be pulled out. This results in

$$\mathbf{K} = -(\mathbf{R}_{lqr} + \mathbf{B}^T P\mathbf{B})^{-1}\mathbf{B}^T P\mathbf{A}. \tag{8.20}$$

Now, by default, equation (8.14) will attempt to drive all of its states to zero. In order to have it drive to a set of desired states, equation (8.14) needs to have target states added.

$$\chi_{k+1} = \mathbf{A}\chi_k + \mathbf{B}\mathbf{K}(\chi_k - Targets). \tag{8.21}$$

Equation (8.21), when shown in a block diagram, is represented by Figure 8.1.

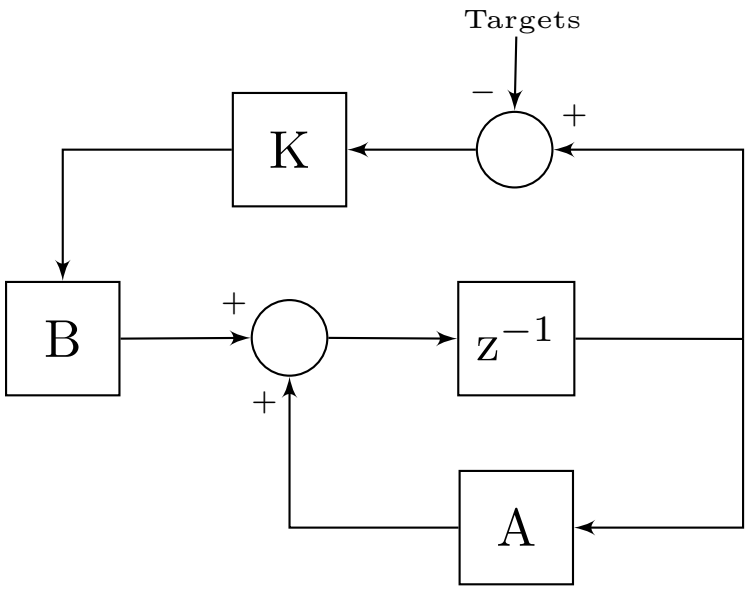


Figure 8.1: Block diagram of the Tracking Discrete LQR used.

8.4 Linearizing the Model

A linear quadratic regulator relies on a linear model. The model developed in Chapter 6, is non-linear. Techniques do exist to control non-linear systems. A simpler approach, however, is to linearize the model. The process of linearizing a model can be interpreted as creating a close estimate of the dynamics, often by looking at a specific case of the model. If it is known that a system will operate within certain parameters, that knowledge can be exploited to create a specific case of the model, which often times means that the model can be simplified for the desired purpose. This often leads to the model becoming linear in nature.

The process of linearizing the model begins with the non-linear model, equations (6.10) and (6.20). One of the first observations that can be made is that the quadcopter will be operating near or parallel to the ground. Mathematically, this means that the pitch and roll angles will be near or at zero. While the first impulse is to just plug zero in, this is highly undesirable in this case, as the horizontal movements in the X and Y plane would drop out. The model should be still able to account for the horizontal movements of the quadcopter, no matter how linearized the final model becomes. Another commonly used method is to estimate the non-linearities with a Taylor series expansion. This is more likely a better option, since a Taylor series expansion is an estimate, not an outright substitution. The expansion is performed on the model's trigometric functions:

$$\sin\phi_{(0,0)} \approx \sin 0 + \phi\cos 0 = \phi \tag{8.22}$$

$$-\cos\phi\sin\theta_{(0,0)} \approx -\cos 0\sin 0 + \phi\sin 0\sin 0 - \theta\cos 0\cos 0 = -\theta \tag{8.23}$$

$$\cos\phi\cos\theta_{(0,0)} \approx \cos 0\cos 0 - \phi\sin 0\cos 0 - \theta\cos 0\sin 0 = 1 \tag{8.24}$$

When the linearizations made in (8.22), (8.23), and (8.24) are substituted into (6.10), this yields

$$\ddot{\overline{V}} = \begin{bmatrix} -\phi(F_1+F_2+F_3+F_4)\frac{1}{m} \\ \theta(F_1+F_2+F_3+F_4)\frac{1}{m} \\ -(F_1+F_2+F_3+F_4)\frac{1}{M}+g \end{bmatrix}. \tag{8.25}$$

The next components of the model that can be simplified down are the angular velocity components present in (6.20). The angular velocities, or the speed which the quadcopter will be rotating around its axis, are intended to be very small (barrel rolls are outside the scope of this thesis work). A very small number multiplied then by another very small number will result in an even smaller number, a number that in practice will be near zero. This results the gyroscopic terms being dropped. This yields

$$\ddot{\overline{\Omega}} = \begin{bmatrix} \frac{\sqrt{2}}{2}d(F_1+F_2-F_3-F_4)\frac{1}{I_{xx}} \\ \frac{\sqrt{2}}{2}d(F_2+F_3-F_1-F_4)\frac{1}{I_{yy}} \\ (\tau_1+\tau_2+\tau_3+\tau_4)\frac{1}{I_{zz}} \end{bmatrix}. \tag{8.26}$$

Returning back to (8.25), further simplifications can be made. The quadcopter's controller will be designed so that its movements will be smooth and gradual, not sharp and violent, in nature. To achieve such movements, the change in motor performance would be small from the no movement hover conditions; most of the force would come from the quadcopter's requirement to just stay afloat. Because the force would remain the same, more or less, for the purposes of estimation in the horizontal direction, it can be assumed that the forces from the motor will be constant. That constant force would be assumed to be the steady state force. Mathematically, this is described as

$$F_{ss} = Mg. \tag{8.27}$$

When (8.27) is plugged into (8.25), it simplifies the model further to

$$\ddot{\overline{V}} = \begin{bmatrix} -\phi g \\ \theta g \\ -(F_1 + F_2 + F_3 + F_4)\frac{1}{m} + g \end{bmatrix}. \tag{8.28}$$

The final simplification to linearize the model to a desired level is to remove the gravity in the vertical term. Gravity, at the heights the quadcopter is intended to fly, is constant.

$$\ddot{\overline{V}} = \begin{bmatrix} -\phi g \\ \theta g \\ -(F_1 + F_2 + F_3 + F_4)\frac{1}{m} \end{bmatrix}. \tag{8.29}$$

By removing the gravity term, the model still works but the force the model has input must be viewed as a change from the no movement force. This viewpoint, while not absolutely necessary for the torques, is also the preferred way to look at the model. Because of this, the effects of gravity must be included elsewhere.

This results in the final, linearized model being (8.29) for the linear accelerations and (8.26) for the angular accelerations. This model will serve as the basis for the design of a filter and controller. However, it is important to reiterate that this is an estimate which is only valid in the following conditions:

- The pitch and roll angles will be near zero
- The angular velocities are near zero
- The motor outputs during operation will remain close to the outputs if the motor was to operate at no movement conditions
- The inputs must be considered as a delta, not as an absolute
- The effects of gravity must be compensated for at some point in the design

These conditions must be considered when designing a controller. If the controller pushes the quadcopter out of these conditions, then the designed controller will not be suited for its job.

8.5 The Linearized State Space Model

With the model linearized, the next step is to compute the gain. The calculation for the gain was laid out earlier, resulting in (8.20). To proceeded with the calculations, the system and the input matrices need to be defined. The system matrix comes from the linearlized equations (8.29) and (8.26) and their first order Euler integrations.

This results in a linearized system of equations

$$\chi_{k+1} = \mathbf{A}\chi_k + \mathbf{B}u_k = \begin{bmatrix} X + \dot{X}dt \\ \dot{X} + \ddot{X}dt \\ -\phi g \\ Y + \dot{Y}dt \\ \dot{Y} + \ddot{Y}dt \\ \theta g \\ Z + \dot{Z}dt \\ \dot{Z} + \ddot{Z}dt \\ -(F_1 + F_2 + F_3 + F_4)\frac{1}{m} \\ \phi + \dot{\phi}dt \\ \dot{\phi} + \ddot{\phi}dt \\ \frac{\sqrt{2}}{2}d(F_1 + F_2 - F_3 - F_4)\frac{1}{I_{xx}} \\ \theta + \dot{\theta}dt \\ \dot{\theta} + \ddot{\theta}dt \\ \frac{\sqrt{2}}{2}d(F_2 + F_3 - F_1 - F_4)\frac{1}{I_{yy}} \\ \psi + \dot{\psi}dt \\ \dot{\psi} + \ddot{\psi}dt \\ (\tau_1 + \tau_2 + \tau_3 + \tau_4)\frac{1}{I_{zz}} \end{bmatrix} \tag{8.30}$$

with a set of inputs defined as

$$u = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{bmatrix}^\top, \tag{8.31}$$

From (8.30), the A and B matrices can be extracted. This results in state matrix of

$$\mathbf{A} = \begin{bmatrix}
1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{8.32}$$

and an input matrix of

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\sqrt{2}d}{2I_{xx}} & \frac{\sqrt{2}d}{2I_{xx}} & -\frac{\sqrt{2}d}{2I_{xx}} & -\frac{\sqrt{2}d}{2I_{xx}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{\sqrt{2}d}{2I_{yy}} & \frac{\sqrt{2}d}{2I_{yy}} & \frac{\sqrt{2}d}{2I_{yy}} & -\frac{\sqrt{2}d}{2I_{yy}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_{zz}} & \frac{1}{I_{zz}} & \frac{1}{I_{zz}} & \frac{1}{I_{zz}} \end{bmatrix} \tag{8.33}$$

Note how there are eight inputs, yet there are only four motors. This is because the motors torques and forces are kept separate for computation purposes. It is only when the forces and torques are converted into individual throttles that the separate inputs are combined.

The $\mathbf{Q}_{lqr}$ and $\mathbf{R}_{lqr}$ are also needed for the cost function. What these will do is dictate how aggressive the responses will be; a higher $\mathbf{Q}_{lqr}$ values will penalize being off the target states more heavily, while higher $\mathbf{R}_{lqr}$ values will penalize more aggressive actuator actions. Finding the best setup was done iteratively, once the simulation was running. To get the system to react as desired, it was seen that penalizing the actuators was beneficial to prevent violent movements. It was also found that the cost of being not on the target positions needed to be much high then that of the accelerations and velocities. Equations (8.34) and (8.35) show the values used in the simulations.

$$\mathbf{Q}_{lqr} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1000 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1000
\end{bmatrix} \tag{8.34}$$

$$\mathbf{R}_{lqr} = \mathbf{I}_{(8,8)} \times 3000 \tag{8.35}$$

With these values determined, (8.13) can be solved, which in turn allows the gain to be found.

Chapter 9

Implementing the System Within a MATLAB Environment

9.1 Introduction

With the properties of the quadcopter identified, a detailed model derived, and a filter and controller created, the next step is to verify these elements. While these pieces could be directly implemented onto a piece of hardware, it is more prudent to implement these pieces within a simulated environment. By doing so, verification and adjustments can be made quickly and without risking physical hardware and time delays.

The environment for creating a simulated world was chosen to be MATLAB. This language was built with the intended use for computation, especially linear algebra. While Python with the Numpy package would work just as well and would have made transitioning the simulated code to the quadcopter more straightforward, MATLAB was chosen because of its simpler syntax for matrices, built in computational packages, and the author's strong familiarity with the language compared to Python.

9.2 Implementing the the Simulation

While the raw code used to create the simulation can be viewed in Appendix I, it is of benefit to go through the computational flow of the simulation.

First is a general flow diagram, showing the simulations computations overall.

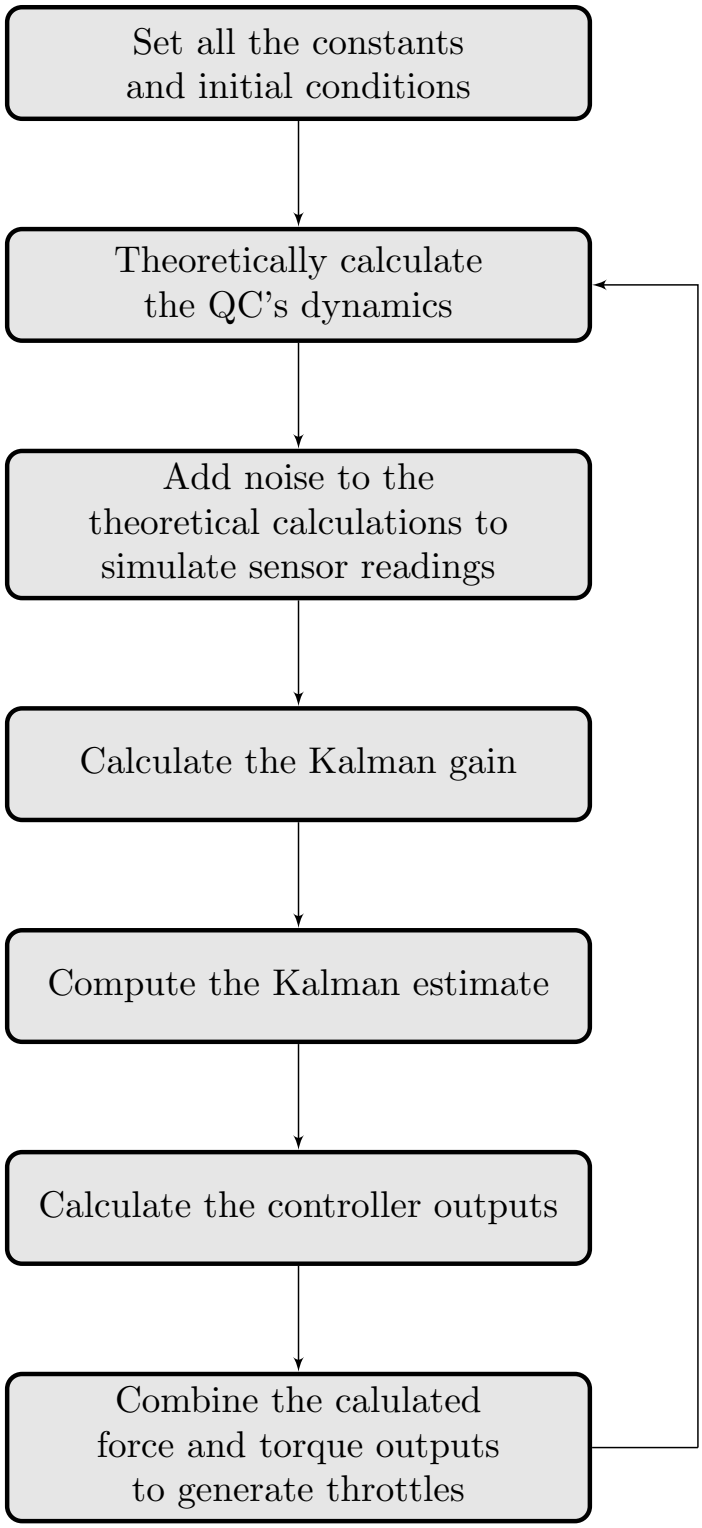


Figure 9.1: Diagram of the flow of the simulation.

The first step in the simulation is to import, initialize, and compute constants that are present in the model. These include

- The properties of the motors
- The properties of the quadcopter
- The variance of the sensors
- The state space matrices
- The Kalman gain computations
- Other simulation related variables, such as duration and timestep

With these constants initialized, the iterative process can begin. Every iteration starts by computing what the theoretical position of the quadcopter is. This is done by using the six degree of freedom non-linear model identified in Chapter 6 to get all the accelerations. The accelerations then are integrated through the use of a first order Euler approximation. A first order approximation was chosen over other methods, such as RK45, because the difference in error should be small given the small time step and that the Euler approximation will be more representative of the computations when implemented on a quadcopter. The only factor that impacts the theoretical flight of the quadcopter is each motor's throttle. This is intentional, as the motor throttles will be the only input the on-board computer will have to the rest of the quadcopter.

The next section to compute is to simulate the sensor readings. This can be simulated with a rather large degree of accuracy because the noise characteristics of the sensors are well known, being identified in Chapter 3. To simulate the sensor, the theoretical model's states are taken and Gaussian noise, equivalent to the variance present in each corresponding sensor, is added.

These two steps essentially simulate the quadcopter, which leads to the remaining steps being the designed controller and Kalman filter. First, the filtered states need to be computed. This begins by updating the Kalman gain estimate, as highlighted in Chapter 7. Following this computation, the Kalman state estimates can be computed. With these computed, they can be fed into the controller gains calculated at the very start, with the states being adjusted to correspond with the desired target states. This flow can be better illustrated in Figure 9.2.

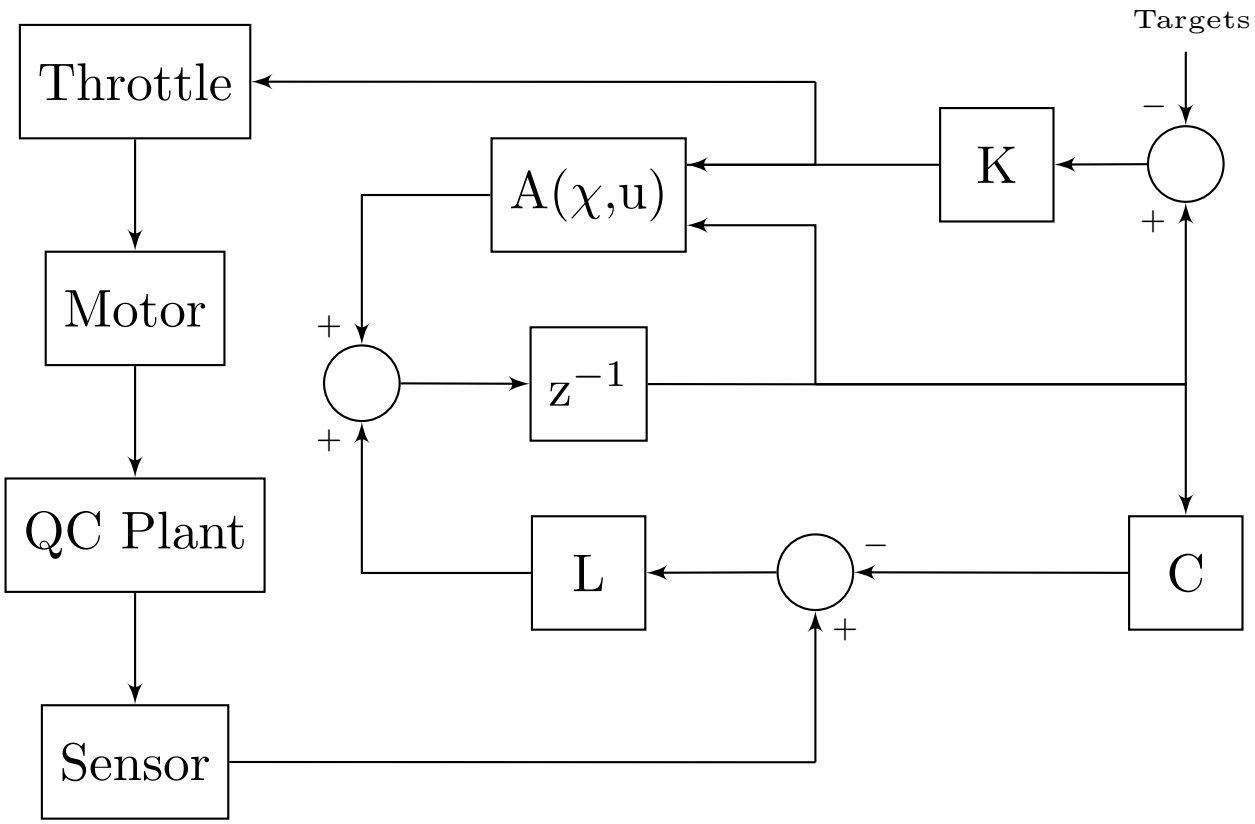


Figure 9.2: Block diagram of the combined Kalman Filter and LQR controller that was used

A point to highlight is that the block diagram is a combination of Figures 7.1 and 8.1. While the the Kalman gain, L, and its associated components do change every iteration, the rest remains constant. The controller gain was also computed as detailed in Chapter 8, using the linearized version of the system matrix.

Undergoing these steps yields a set of outputs. These outputs are the changes in force and torques to incite movement. However, to input into the motor, a throttle must be computed. This can be done rather easily, because the characteristics of the motors are well known.

The translation starts with adding the computed force changes to the force needed to counter the effects of gravity. Because gravity should never change significantly for this very low altitude craft, this value is a constant. These forces are then translated into the equivalent torques which the motor would produce if it were to produce the dictated force output. At this point, the torque adjustments are added on. These adjusted torques are then translated finally into the output throttles. This flow is shown in Figure 9.3.

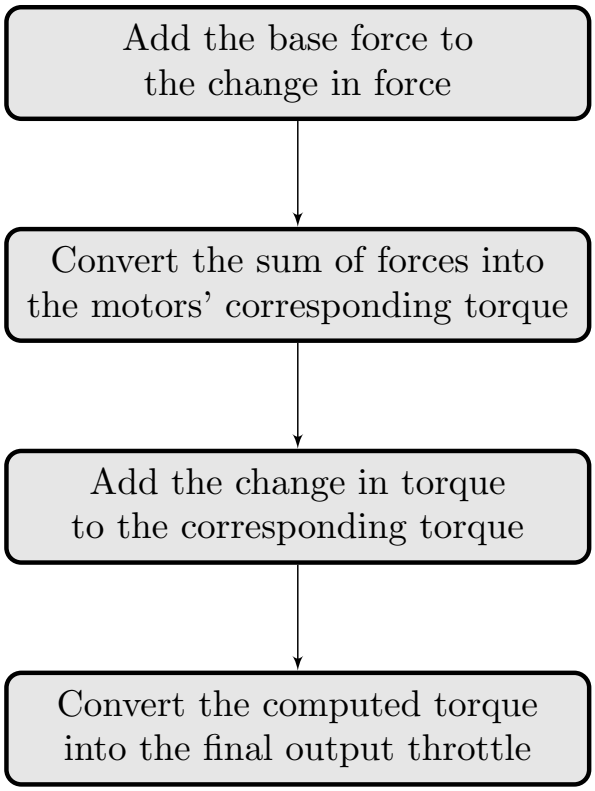


Figure 9.3: Flow diagram of the steps to compute the output throttles.

At this point, the simulation has undergone a single iteration and the time is stepped one time step. These iterative computations continue until the simulation has run its intended course.

9.3 Simulation Results

The simulation was implemented and run, testing the designed controller, filter, and other control allocation designed components. Refinements were then made to improve performance, resulting in the final results as presented below.

First looked at were the target throttles in Figures 9.4 and 9.5. A specification for the controller was that the movements of the quadcopter would not be aggressive. Looking at both the throttles and dictated inputs, its seen that only a subtle change is needed to achieve desired results. This is as designed.

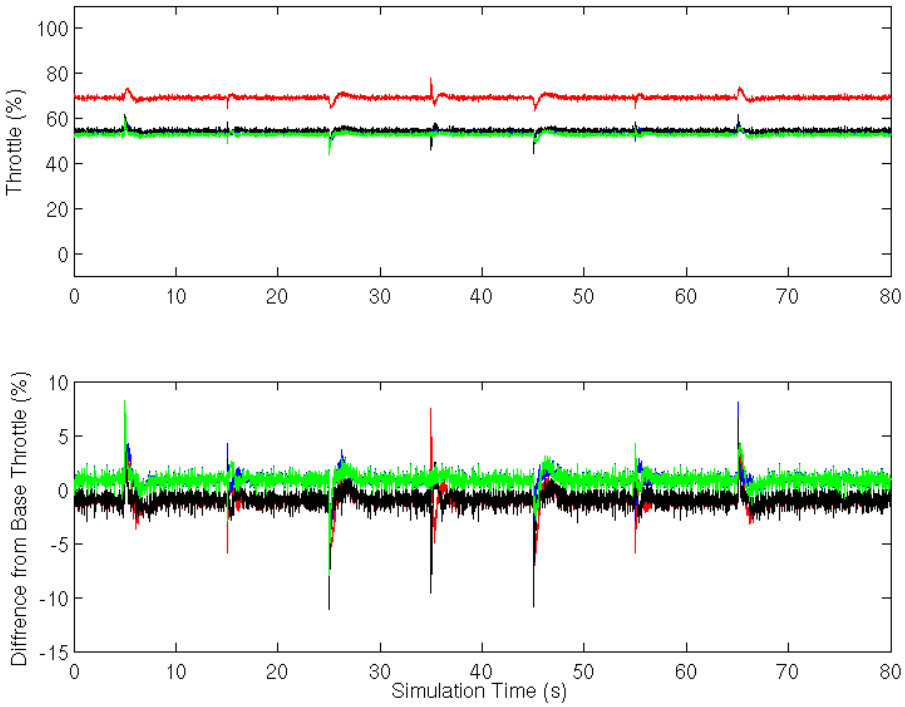


Figure 9.4: The throttles fed into the plant over the simulated duration.

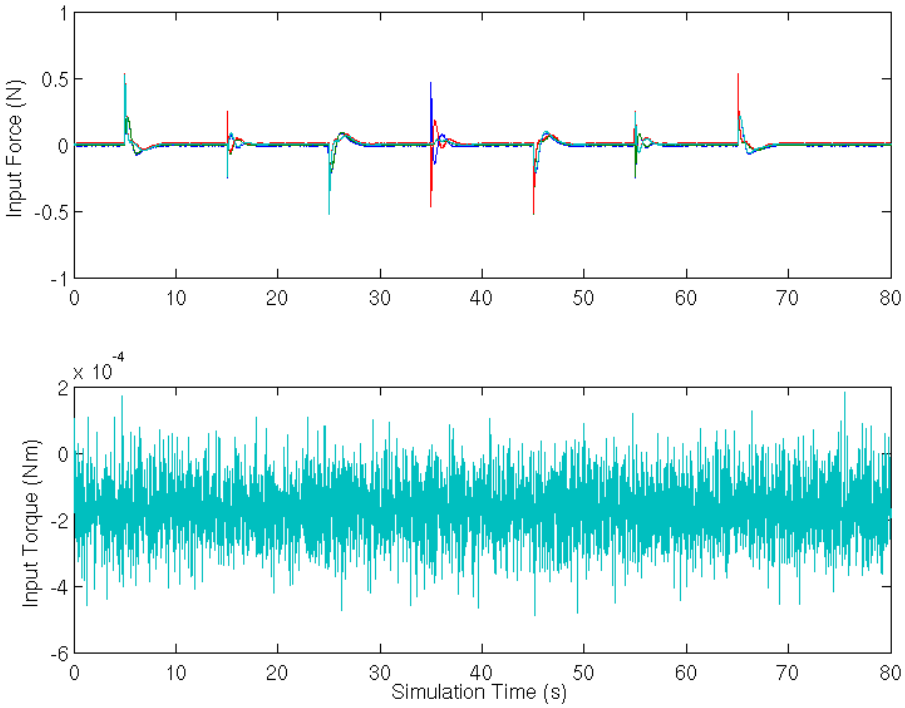


Figure 9.5: The inputs which were converted into throttles and which were fed into the state calculations.

The next six plots, Figures 9.6, 9.7, 9.8, 9.9, 9.10, and 9.11, show the target states and how well the controller directed the quadcopters states toward them. While at times the controller does deviate from its target states, this is actually a very good thing. If the quadcopter could not accelerate or turn, then it would not be able to reach its final position states. That it is "intelligent" enough to break from a designed target state so that it can reach eventually all the target states verifies that the controller is working as intended.

In these plots, blue points represent the theoretical flight of the quadcopter. Red points represent the simulated sensor readings. Green points represent the Kalman estimate. Magenta points represent the target values for the states.

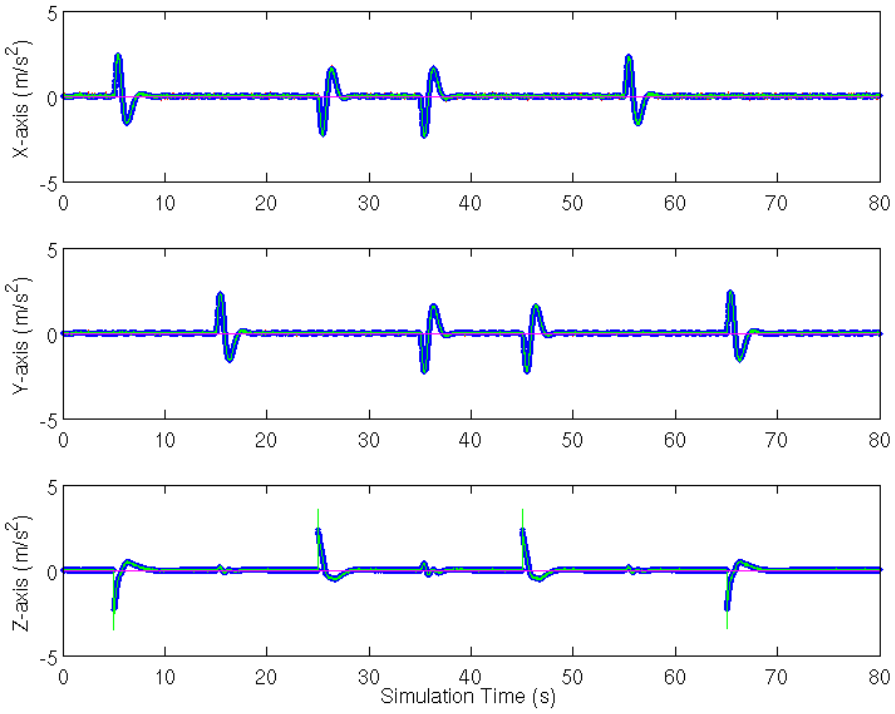


Figure 9.6: The simulated translational acceleration results.

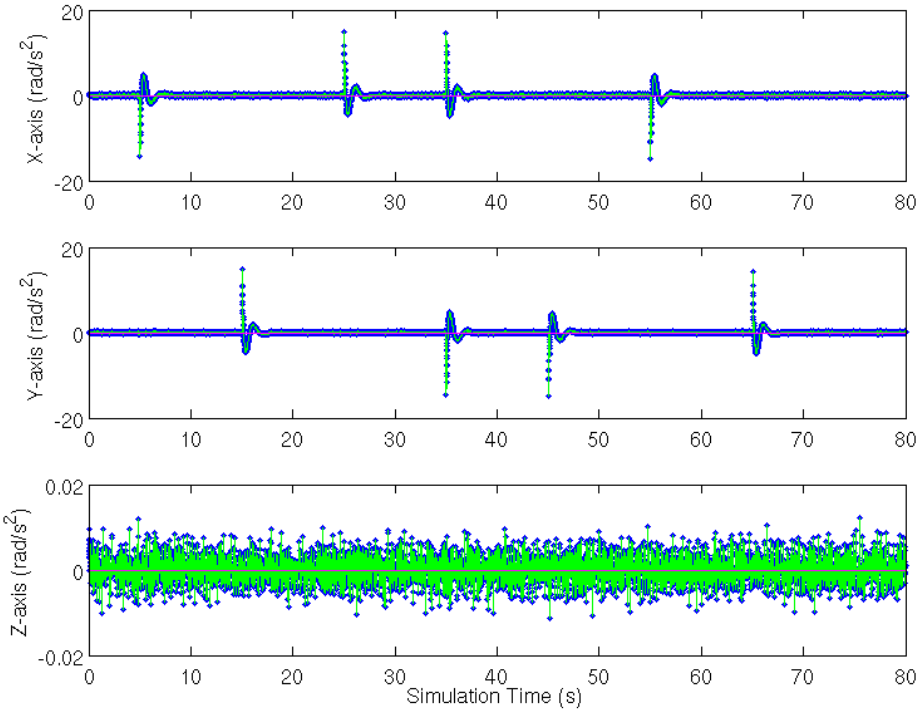


Figure 9.7: The simulated angular acceleration results.

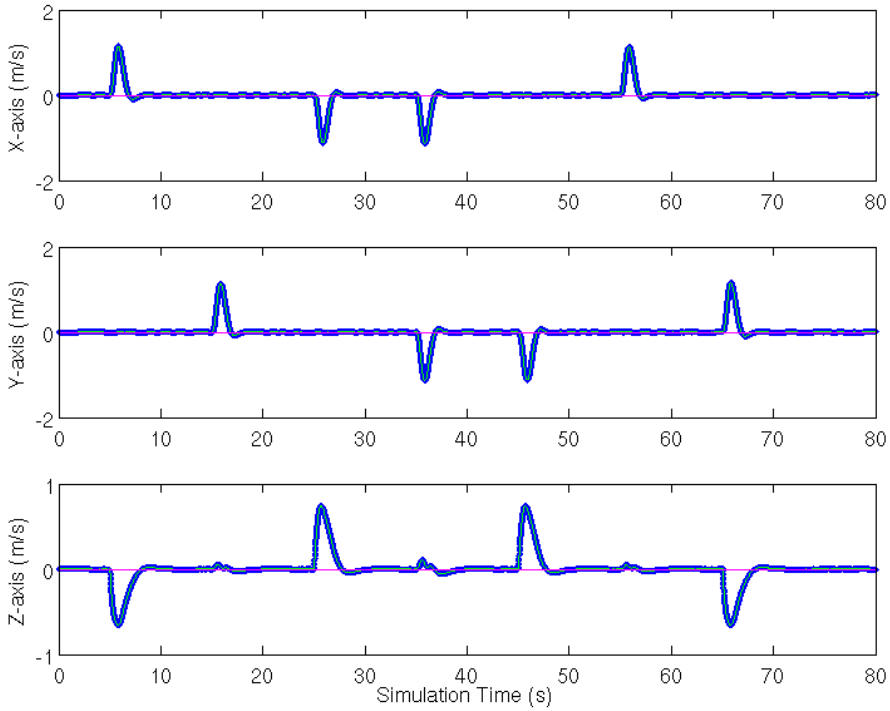


Figure 9.8: The simulated translational velocity results.

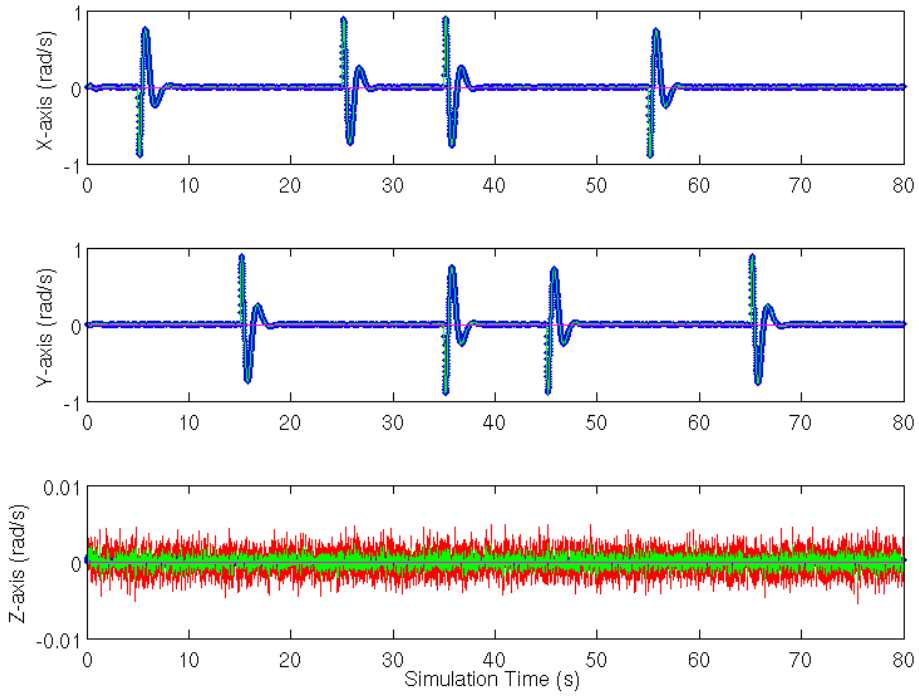


Figure 9.9: The simulated angular velocity results.

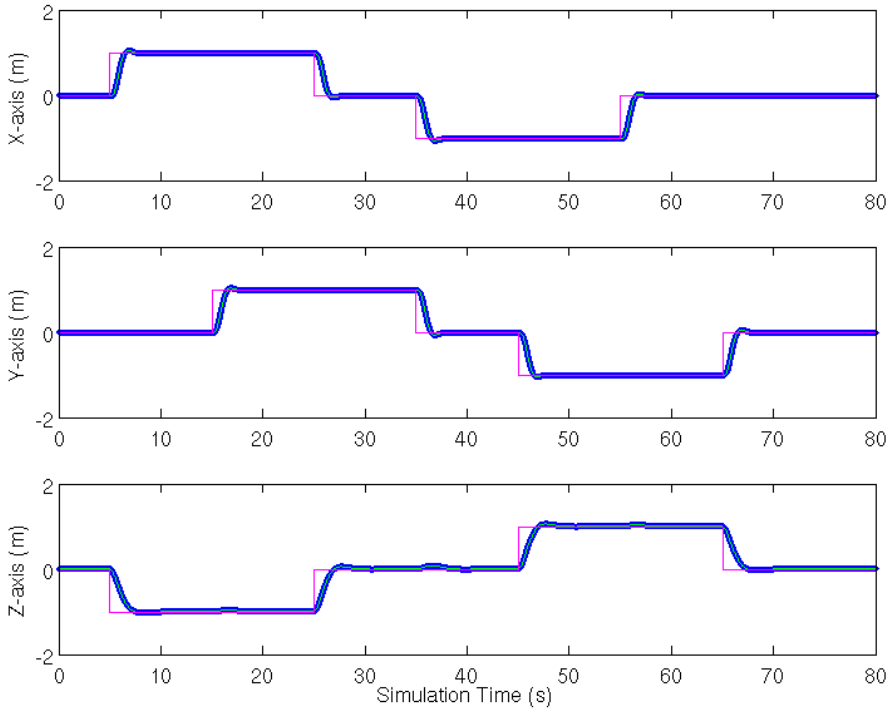


Figure 9.10: The simulated translational position results.

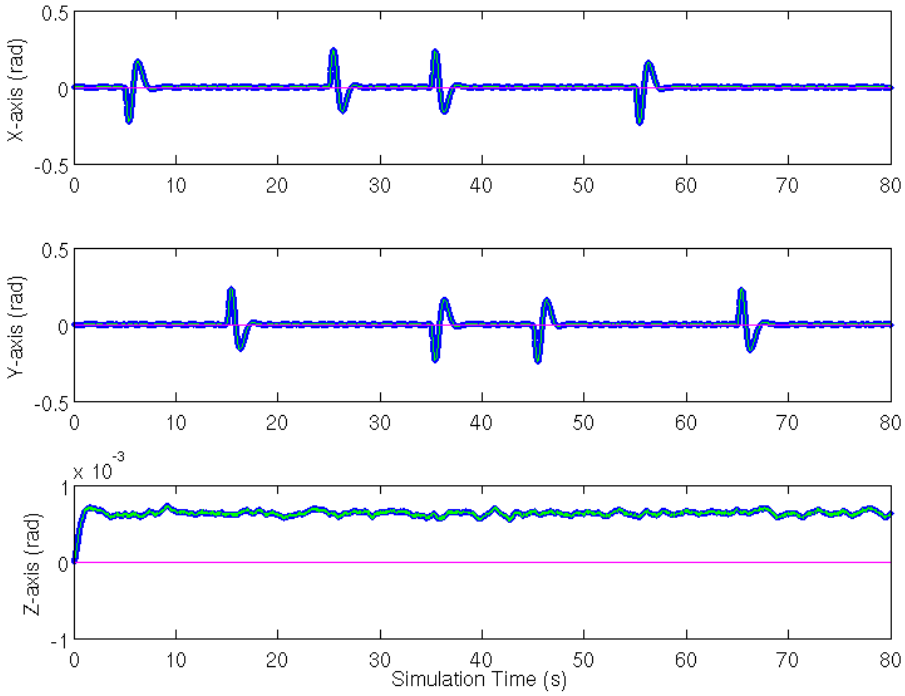


Figure 9.11: The simulated angular position results.

Figures 9.12 and 9.13 are used to verify that the Kalman estimate is in fact tracking. A filter that tracked correctly would have its residuals be centered around zero. This is the case for all six sensor readings, which verifies that the Kalman filter is working as intended.

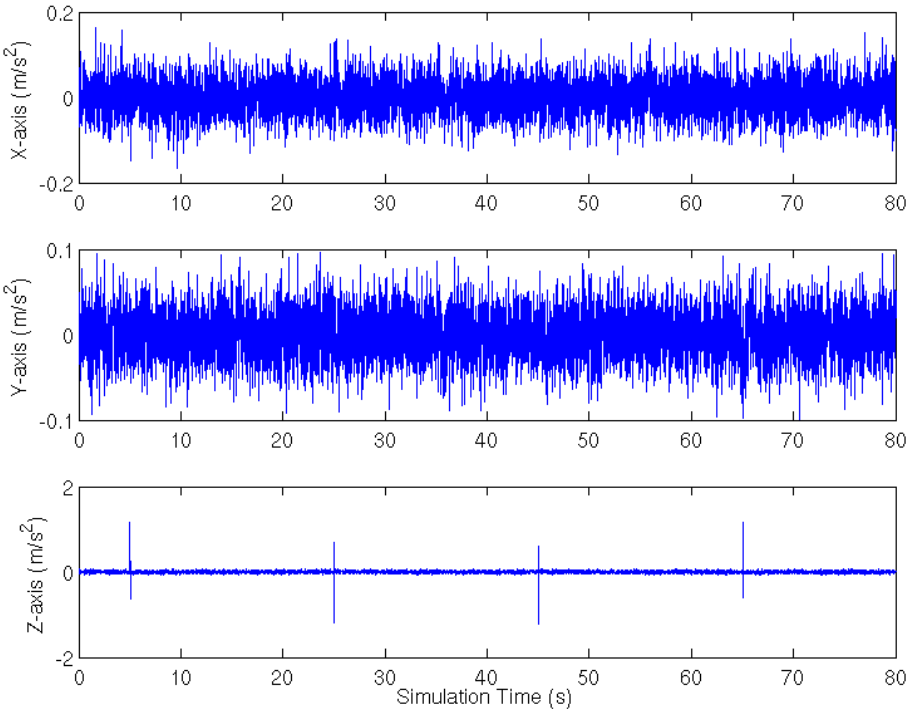


Figure 9.12: The residual between the Kalman estimates and the simulated sensor readouts of the vertical accelerations.

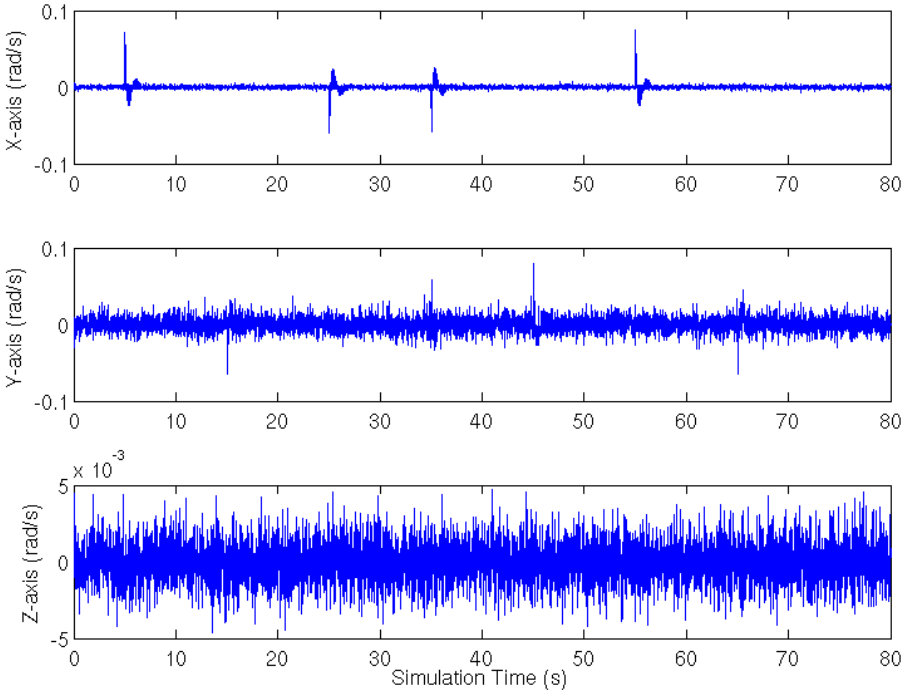


Figure 9.13: The residual between the Kalman estimates and the simulated sensor readouts of the angular velocities.

All together, he plots tell the story that the designed components work as intended and as designed.

Chapter 10

Future Works

10.1 Implementing the Designs on the Quadcopter

The initial end goal of this work was to have the designed elements implemented onto a real BeagleBone Black quadcopter. Unfortunately, due to time limitations, implementation was never achieved. However, a significant amount of time was devoted to the task and considerable progress was made. Appendix A discusses how the BeagleBone on the quadcopter needs to be prepared. Appendix B covers the theory on implementing the designs onto the quadcopter. Appendix J presents the incomplete code which was written to implement the designed algorithms onto the quadcopter. All the information presented should serve as an excellent springboard to take the designs and to implement them.

10.2 More Detailed Theoretical Model

The theoretical model developed as of now serves as a decent representation of the quadcopter's movements in the world. However, the model created can be refined and expanded upon, notably in two directions: The motor performance and the quadcopter model.

The current motor models developed don't account for various input voltages. When a fresh battery is put into the quadcopter, it has a voltage that is higher then the 7.4V the battery is rated to output. On the other side, a battery that is drained will output a voltage that is lower then the 7.4V it is rated. The linear motor models, as explained earlier, are also, strictly speaking, not correct. Creating a statistically correct model and including the voltage of the battery into the model are both ways in which to improve the motor models.

As for the quadcopter model, aerodynamic effects are omitted from the model. A more accurate model would have these effects included. These effects have been already explored in a few papers so any work in this area would have solid prior art from which to work.

10.3 Use Different Control Techniques

In this work, an optimal controller is used to control the movements of the quadcopter. Many other controllers exist however. In many works that deal with the control of a quadcopter, a PID controller is used. A fuzzy logic controller could also be implemented.

In addition, a linear optimal controller is used. While, as shown, the controller does work well, it would be interesting to see a non-linear optimal controller successfully implemented.

10.4 Implement a Continuous Model via an Analog Computer

The current controller is designed in discrete time and the computations take place in a digital environment. It would be interesting to see the discrete system implemented as a continuous system and within an analog environment. While the changes in computations would be fairly straight forward, it is in implementation where the challenges would be present, as circuit consisting of resistors, op-amps, and other components would be used to create an analog computer to do the calculations and generate the outputs.

10.5 Alternative and Additional Sensors

The only feedback mechanism the quadcopter uses at this point in time is an accelerometer and a gyro. However, there are a significant number of other sensors that could be used. GPS is easily implementable, low in cost to implement, and commonly seen implemented on many devices today. Back EMF was originally investigated in this paper to use as a feedback mechanism. These, as well as many other sensors could be used for feedback. It would be interesting to see one of these other sensors, or even a combination of them, implemented in the hardware setup, used in the Kalman filter component of the controller, and all used to increase the intelligence of the quadcopter.

10.6 Refinements to the Quadcopter

The software implementation on the quadcopter could be improved upon. The overall structure of the software was implemented in a manner just to make everything work. However, it is far from an optimal implementation and significant improvement can be made to make the BeagleBone better use its resources.

From a hardware standpoint, the way that the user communicates with the quadcopter is by accessing the quadcopter's access point, an access point which is generated by a small USB WiFi device. While this works, the range the quadcopter is able to travel from the user is rather limited, due to the small range of the device. In addition, creating a WiFi network between the quadcopter and the user's computer may be overkill, depending on what is intended to be transmitted between the two devices. Thus, a different communication protocol, such as radio, may be better.

Appendix A

Prepping the BeagleBone System

A.1 Introduction

In an ideal world, a computer would do everything it needs to do right out of the box. However, ideal worlds only exist on the whiteboards in classrooms. Thus, to prepare the BeagleBone for flight, the system needs to be configured in a few ways. This chapter will provide a higher level overview of the steps taken to prepare the BeagleBone. For specifics, code, and files used, see Appendix J.

A.2 Updating the OS

The BBB came preloaded with an Angstrom image that is not up-to-date. The OS needs to be updated, as bugs emerge on the older image, especially with newer libraries and programs.

The OS was replaced with a newer Debian image. Debian was chosen over Angstrom because development is much easier, as any instructions for a Debian based system (which importantly includes Ubuntu) are more likely to exist and work. Debian's long held reputation of stability also helps affirm the choice. Angstrom development has also ceased.

Changing the OS on the BBB is a straightforward process. First, the latest eMMC flasher needs to be downloaded from the eLinux repository [32]. Once downloaded, the image is then unzipped and flashed onto an SD card. Once the image is on the SD card, it can then be flashed onto the BBB built-in eMMC flash.

A.3 Updating Programs and Libraries

Even though the image loaded is fairly new, it doesn't necessarily mean its software is completely up to date. Fortunately, updating the software on a linux system, even if it is an embedded system, is a straightforward process, as this is done through the software repository. A special emphasis needs to be given to the Adafruit Python libraries [33], as these libraries are key to interfacing with the hardware.

A.4 WiFi Connectivity

It is difficult to achieve free flight with a quadcopter if a long wire tethers it to the ground. Fortunately, wireless communication exists. The easiest way to do this is to enable WiFi communication on the BBB, where the quadcopter acts as a wireless access point. With an access point, the quadcopter can connect to any device with WiFi capabilities. Once connected, a user can then SSH into the BBB. Creating a WiFi access point requires three components: Configuring the network settings on the BBB, running a DHCP server on the BBB, and finally configuring the WiFi access point software itself.

The network settings have to be configured to let the BeagleBone know that a wireless connection can exist. On top of that, an IP address range is given.

A DHCP server needs to be running on the BeagleBone. Without one present, the BBB is unable to assign an IP address between it and the device on the wireless network and is unable to connect to that device. For this project, isc-dhcp-server was used [34].

Finally, the access point itself is set up. This is done with hostapd [35]. Through it, the AP name, password, and protocols are set up.

With all three of these elements in place, the quadcopter can now connect wirelessly.

A.5 Configuring Systemd

When a computer of any type starts up, multiple programs and scripts are run to enable the user space and prepare the computer to be used. A popular service manager used by many Linux distributions, especially those geared toward embedded systems, is systemd [36]. While the system boots up, it starts the services that run the computer, such as network daemons and dekstops. Since it is desired to have a few more custom scripts run on start up, the desired scripts can be added to the list that is run by systemd.

A.6 Final Steps

The bulk of the work has been completed at this point and now some final tweaks are made. An aesthetic change, the computer name of the BeagleBone was changed to reflect the nature of it being a quadcopter.

The BBB also has a HDMI port on it. This port is disabled by commenting out the appropriate line from the uENV.txt file. This frees the pins which were bound to the HDMI port for use.

Appendix B

Theory Behind Implementing the System on Hardware

B.1 Introduction

While a considerable amount of time was spent in implementing the final designs, time ran out. However, a fairly good idea of how to implement the designs on a real quadcopter was developed. Presented here is the author's vision of how the designs would be implemented onto a real quadcopter, presented with the hope that they will serve as a solid base for anyone who wishes to continue to work made here or, at the very least, present some ideas on how to go about implementing the final designs.

B.2 Implementation on the quadcopter

The set out goal was to create a proof of concept. Thus, quite a lot of the implementation would be static in nature, such as what the target points are for the quadcopter to reach. While from a practical standpoint this implementation does not have much practical use, it is a starting point. In the author's opinion, taking a static implementation and expanding it to make it dynamic would be a simply a question of time.

The flow of the software for a static implementation would be very similar to the simulation. This is due to the fact that the code on the quadcopter utilizes Python with the Numpy package. From a syntax and implementation standpoint, they are very similar. This allows the preservation of the structure that was created for the simulation.

The only real changes from the two sets of code is that a real quadcopter is used, and not a simulated one. This means that the code actually becomes lighter, since no theoretical calculations or calculations to assist in analysis needs to be present. However, a mechanism needs to exist to make sure that the computations take place at specified intervals. Another place where code is "added" is at the start, simply to import values that were computed earlier. Figure B.1 runs down the flow of the simulation but it is nearly identical

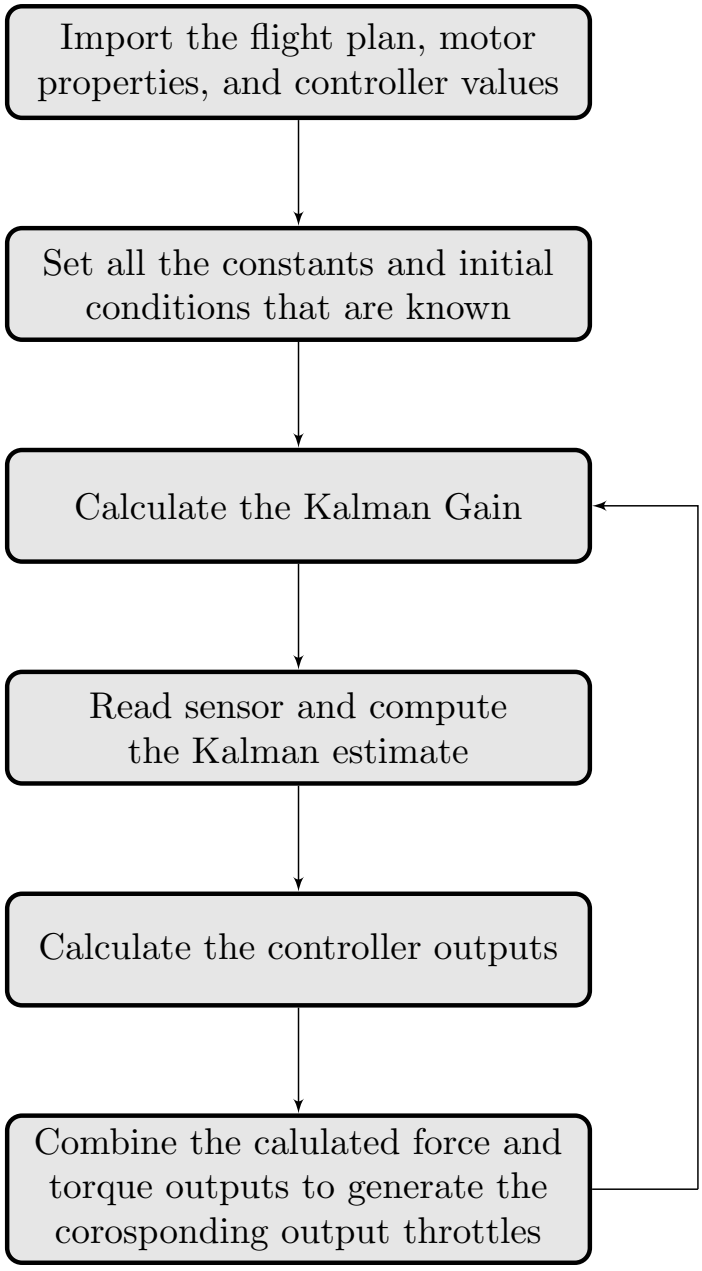


Figure B.1: Diagram of the flow of the software implemented on the quadcopter.

An important factor which really must be stressed is to take into consideration the computational cost

per cycle. If computational cost is too high, then this is a cause of computational errors, which ultimately will yield in a lack of flight. This is precisely what happened in the author's efforts to implement his designs and which ultimately stifled his ability to implement the designs presented within the time frame of his work.

Appendix C

Back EMF Feedback

C.1 Introduction

Back EMF was considered being used as a feedback mechanism because it is sensor less; no additional hardware, such as encoders, would be needed. Using back EMF is a well established and commonly used technique to get feedback of the motor's speed and position [37] [38]. While it was not used in the end, the effort in identifying how the back EMF changes with the change in the motors' throttles was made.

C.2 Collection and Processing

As with the thrust and torque curves, data was collected and curves were fit to the data. Of the types of data collected, the back EMF is the most straightforward. The back EMF was fed through a resistor-capacitor set up identical to that of the quadcopter. At the point where the voltage would normally feed into the BeagleBones ADC, the Quanser ADC was connected instead.

Once collected, the data was inspected for reasonability in Figures C.1, C.2, C.3, and C.4.

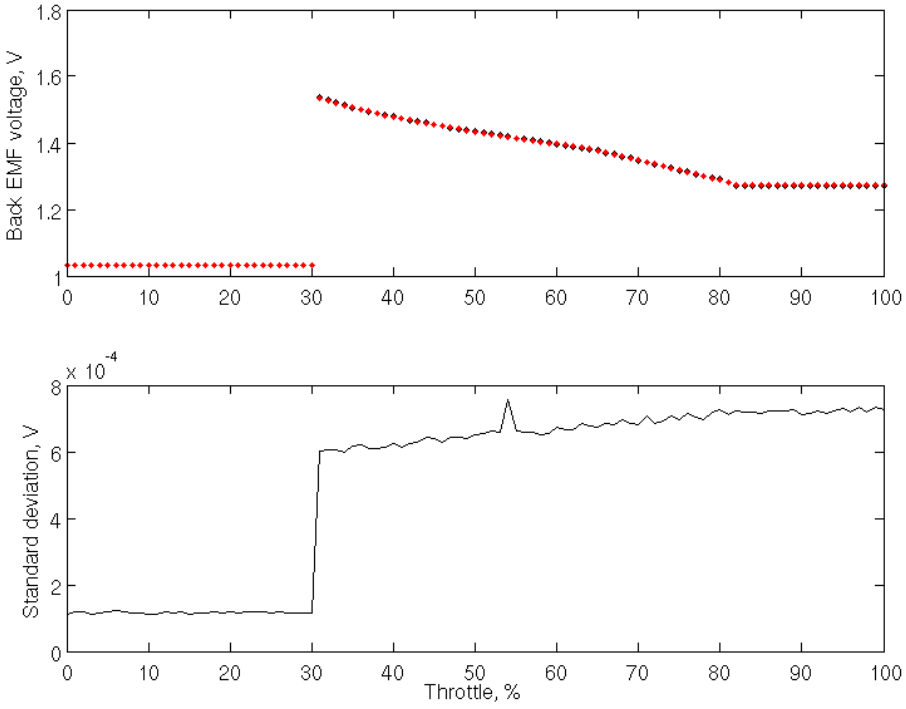


Figure C.1: Collected back EMF data from motor 1.

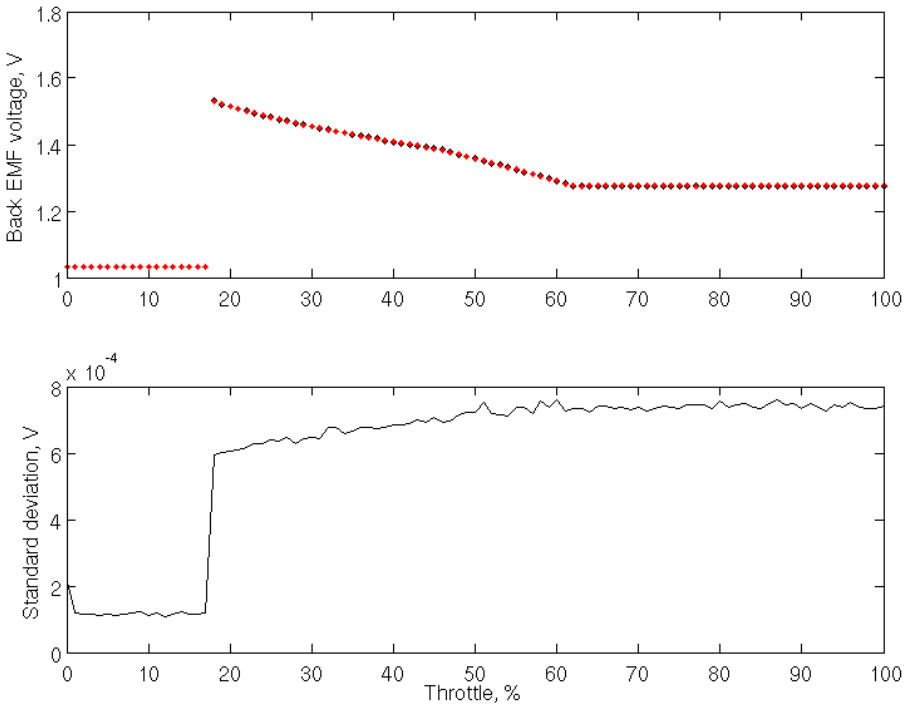


Figure C.2: Collected back EMF data from motor 2.

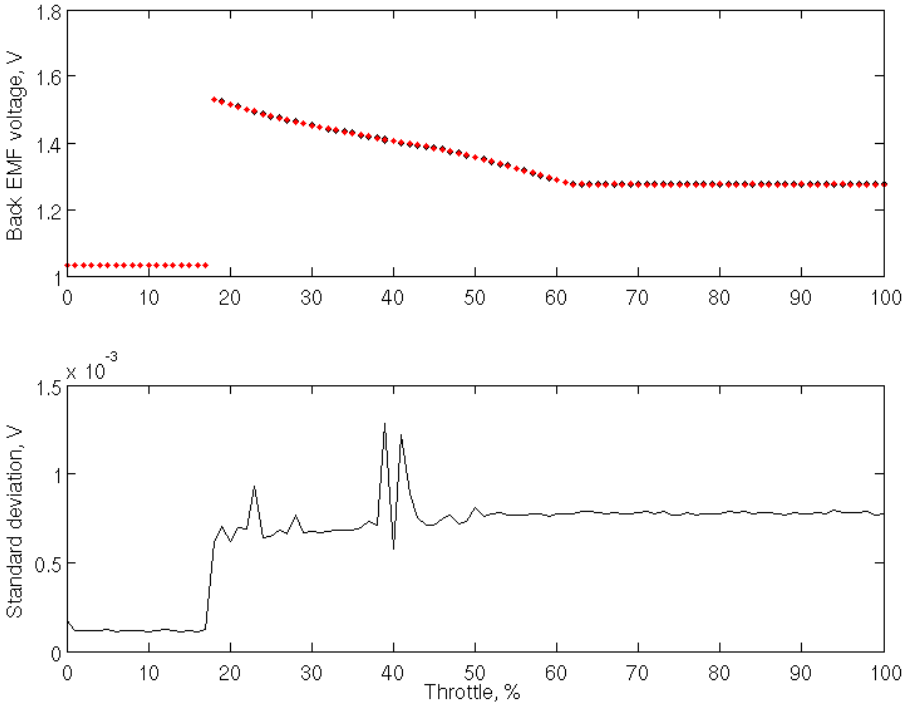


Figure C.3: Collected back EMF data from motor 3.

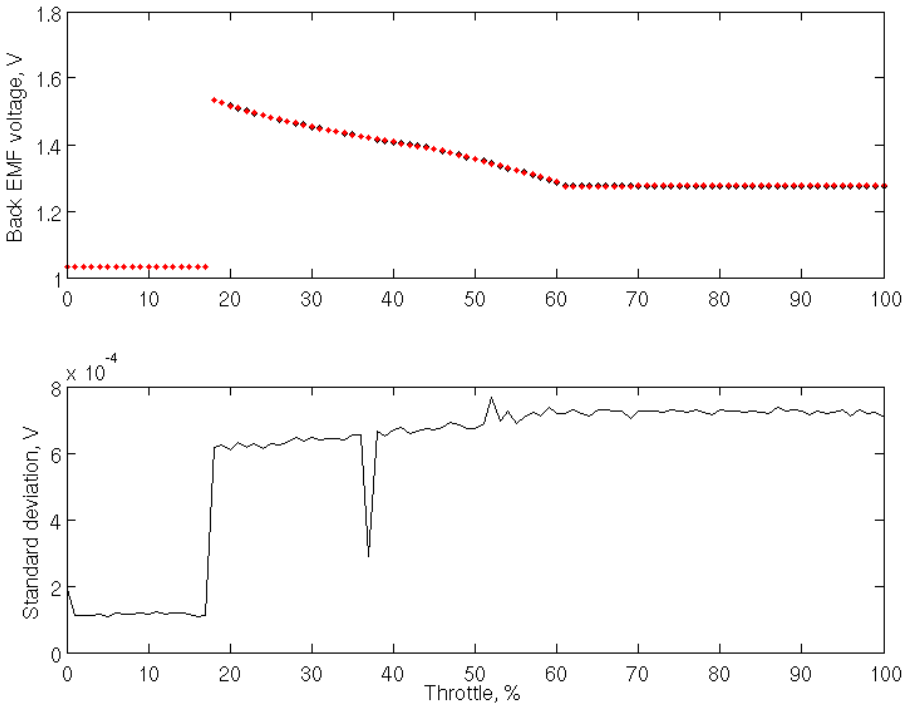


Figure C.4: Collected back EMF data from motor 4.

Identifying the saturation points is a straightforward process with the back EMF. The lower saturation point is present where the spike in back EMF voltage occurs. The upper saturation point is present where the voltage stops dropping. Data below and above these points was trimmed away. Inspecting the data in Figures C.5, C.6, C.7, and C.8, the data appears believable, since the standard deviation is consistent throughout the data set, where the only real change occurs when the motors turn on.

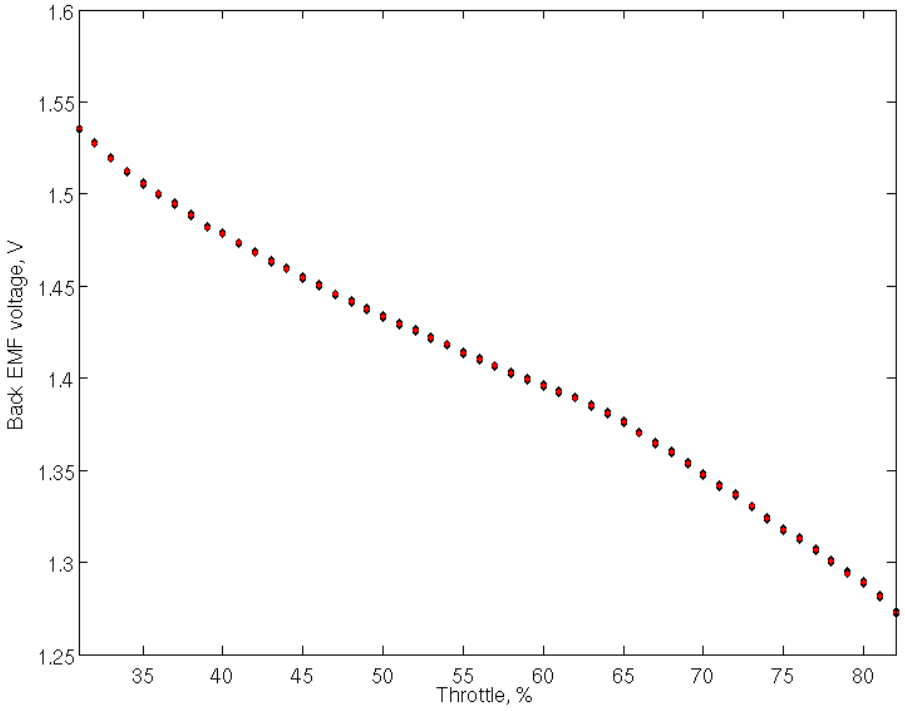


Figure C.5: Trimmed back EMF data from motor 1.

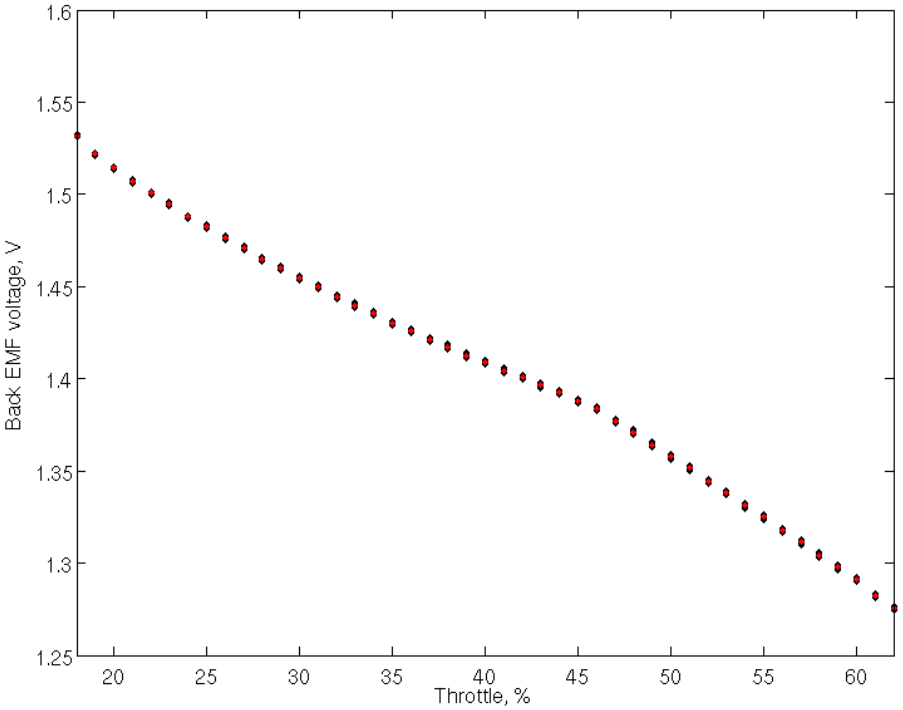


Figure C.6: Trimmed back EMF data from motor 2.

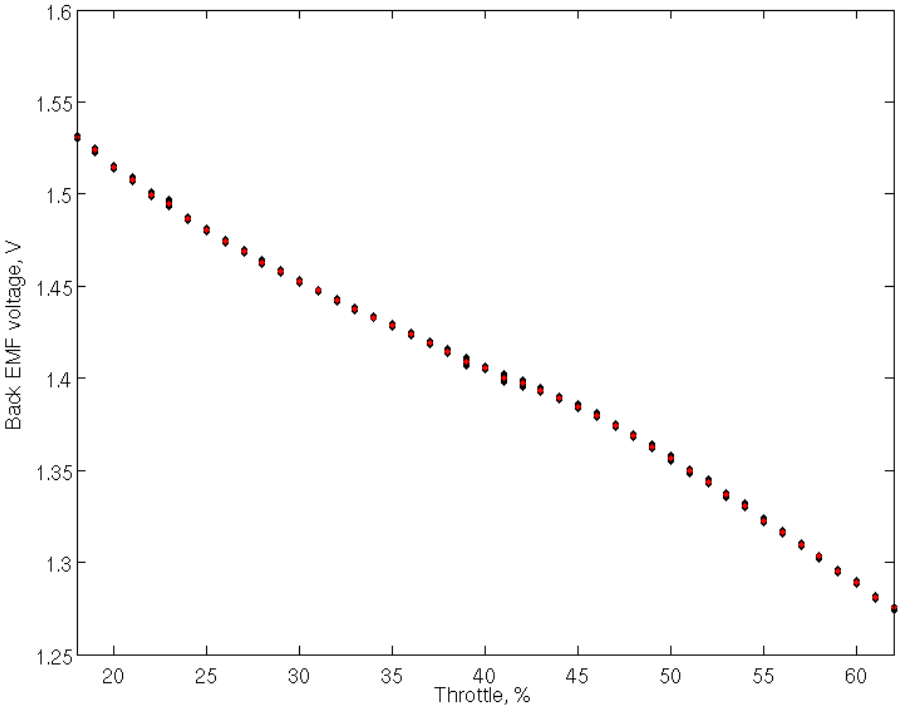


Figure C.7: Trimmed back EMF data from motor 3.

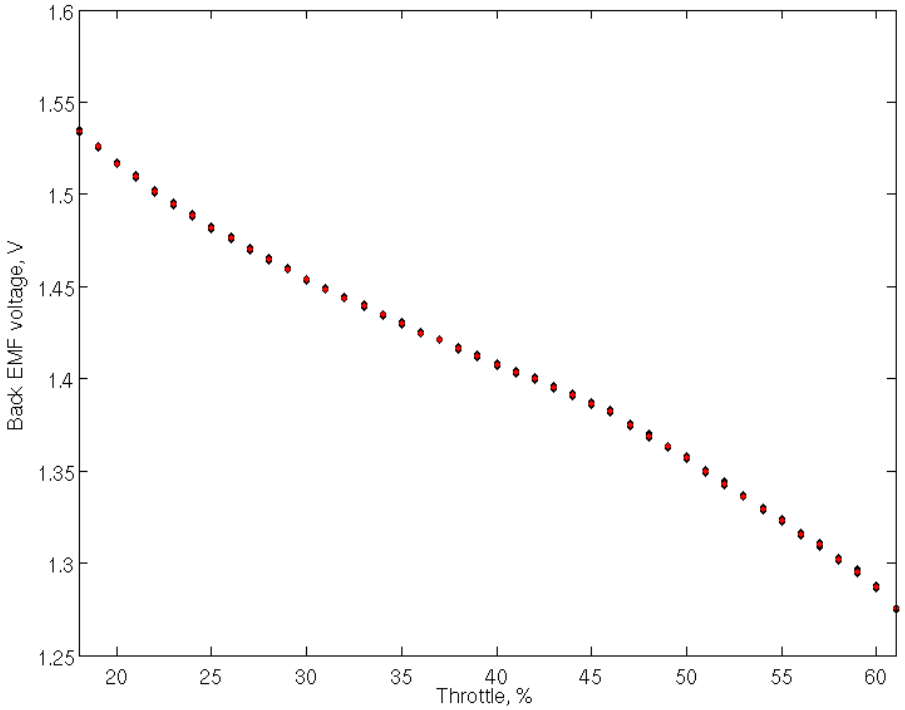


Figure C.8: Trimmed back EMF data from motor 4.

C.3 EMF Curve Fitting

Once the data was cleaned up, the third set of curves was fit. A third order polynomial fit was found to be the best choice, as seen in Figures C.9, C.10, C.11, and C.12.

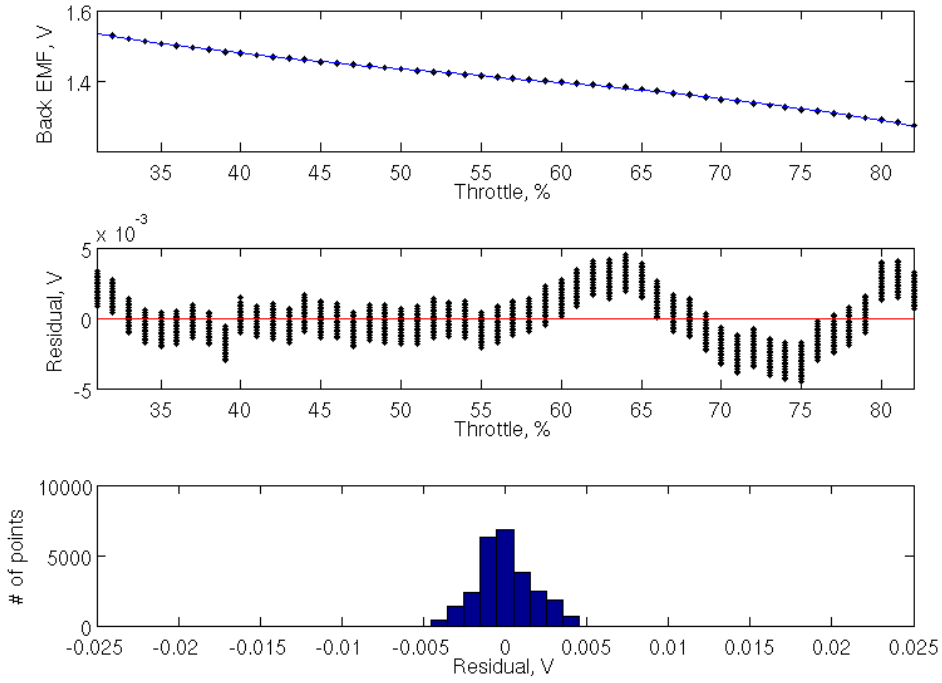


Figure C.9: Motor 1 EMF curve.

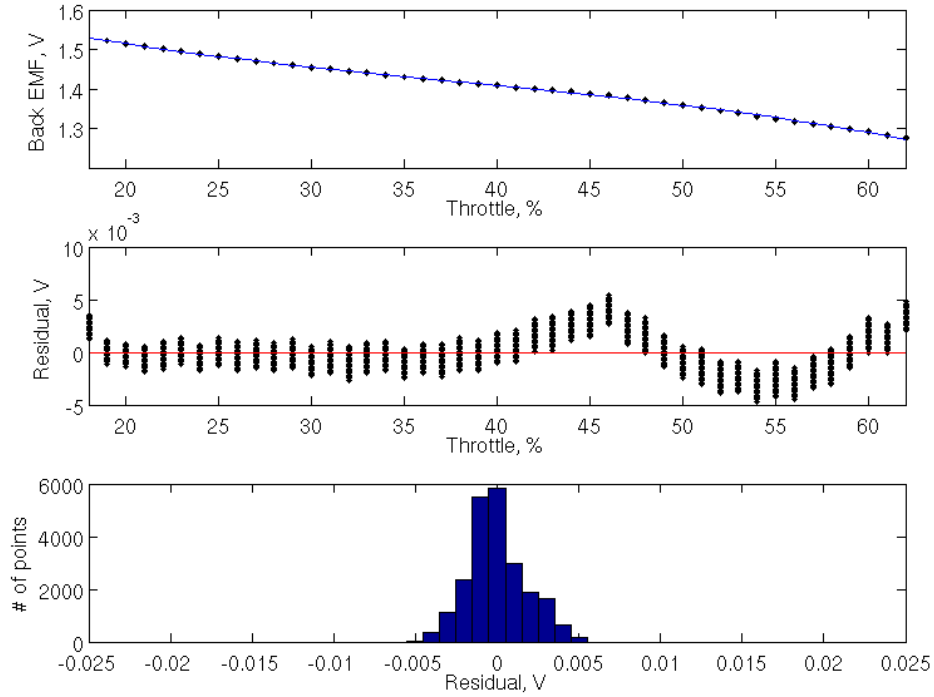


Figure C.10: Motor 2 EMF curve.

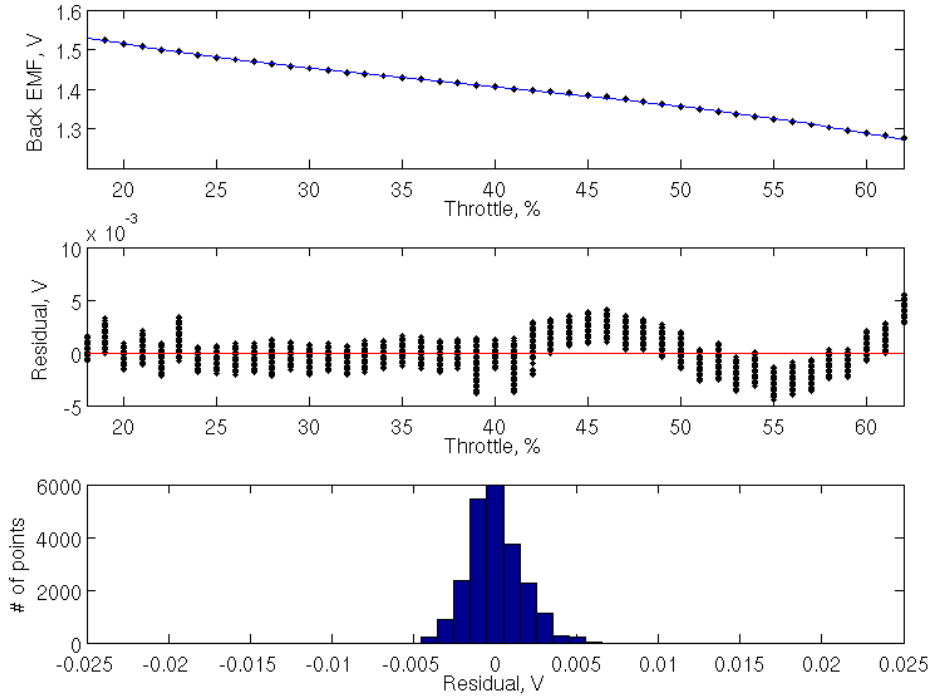


Figure C.11: Motor 3 EMF curve.

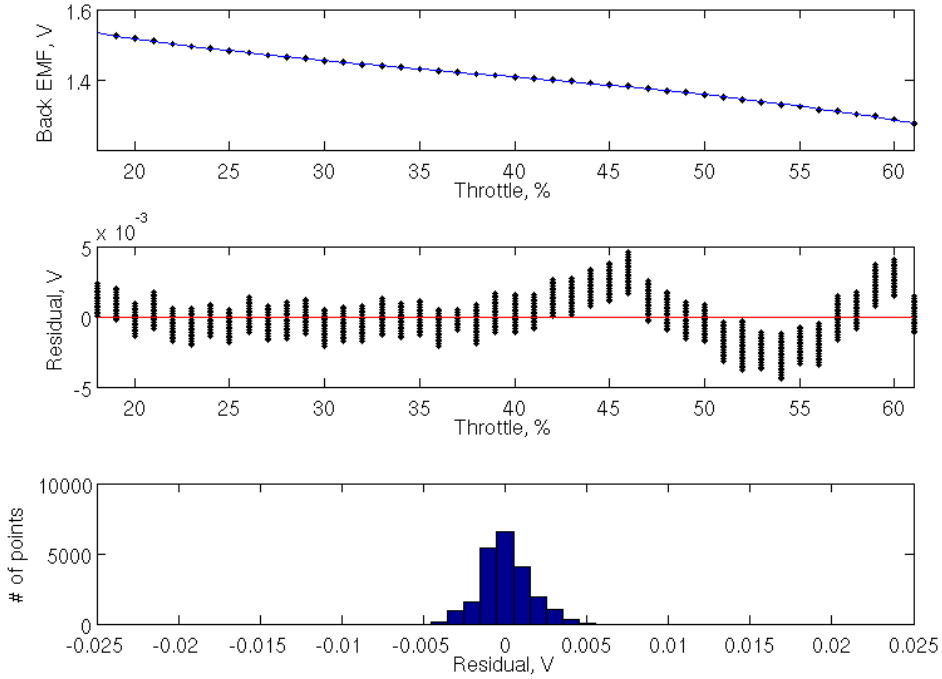


Figure C.12: Motor 4 EMF curve.

The results from the back EMF curve appear to be similar to the torque. Homoscedasticity seems to be in place, but the mean error value is not zero. While normality seems to be in place, a statement can not properly be made in favor or against it. Comparing the fit to the data, it appears to be a "good enough" fit. The issues are also only present in the later half of the data, so, for early values, the fit actually appears to be valid.

It is very interesting to note that the present non-linearities seem to be similar in both the torque and back EMF, as seen in each data type's residuals. This would make sense, as both traits are based on the same motors. This non-linearity may be also present in the thrust, but is hidden underneath the noise.

Table C.1: Table of the found performance characteristics. For the curves, the values are components of a polynomial, as described in equation (C.1). For the saturation values, A is the saturation point at the low end while B is at the high end.

Motor	Characteristic	A	B	C	D
Motor 1	Saturation	31	82	-	-
	Back EMF Curve (V)	-1.937652e-06	3.182961e-04	-2.129580e-02	1.945238e+00
Motor 2	Saturation	18	62	-	-
	Back EMF Curve (V)	-2.649846e-06	3.018689e-04	-1.599489e-02	1.734894e+00
Motor 3	Saturation	18	62	-	-
	Back EMF Curve (V)	-2.751654e-06	3.196993e-04	-1.692950e-02	1.747535e+00
Motor 4	Saturation	18	61	-	-
	Back EMF Curve (V)	-3.235114e-06	3.707291e-04	-1.865102e-02	1.767636e+00

$$(Output) = A(Throttle)^3 + B(Throttle)^2 + C(Throttle) + D \tag{C.1}$$

Appendix D

Listed Quadcopter Hardware Specs

D.1 BeagleBone Black

Table D.1: BeagleBone Black specifications [2].

Feature	
Processor	Sitara AM3359AZCZ100 1GHz, 2000 MIPS
Graphics Engine	SGX530 3D, 20M PolygonsS
SDRAM Memory	512MB DDR3L 606MHZ
Onboard Flash	2GB, 8bit Embedded MMC PMIC TPS65217C PMIC regulator and one additional LDO.
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header
Power Source	miniUSB USB or DC Jack
PCB	3.4" x 2.1"
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated
Ethernet	10/100, RJ45
SDMMC Connector	microSD , 3.3V
User Input	Reset Button, Boot Button, Power Button
Video Out	16b HDMI, 1280x1024 (MAX), 1024x768,1280x720,1440x900 w/EDID Support
Audio	Via HDMI Interface, Stereo
Expansion Connectors	Power 5V, 3.3V, VDD_ADC (1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 3 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)
Weight	1.4 oz (39.68 grams)
Power	Refer to Section 6.1.7

D.2 Turnigy AX-2204C

Table D.2: Turnigy AX-2204C specifications [3].

	Specs
KV(RPM/V)	1450
Lipo Cells	2S
Max Current	8A
Max Power	70W
No Load Current	0.3A
Internal Resistance	0.3 ohm
Number of Poles	12
Shaft Size	3.0mm
Dimensions (Dia x L)	27 x 25mm
Bolt Hole Spacing	30 x 30mm
Bolt Thread	2.5mm
Suggested ESC	10A
Prop Fitting	Prop saver type (O ring required)
Weight	20g

In addition, the following specs were provided when used in conjuction with a 7038 propeller:

Table D.3: Turnigy AX-2204C with a 7038 propeller specifications [3].

7 x 3.8 2 Blade	
Voltage	7.3V
Current	4.7A
Watts	35W
Thrust	211g

D.3 Exceed RC Proton 10A Electronic Speed Control System

Table D.4: Exceed RC Proton 10A ESC specifications [4].

	Specification
Output	Continuous 10A, Burst 12A up to 10 Secs.
Input Voltage	2-4 cells lithium battery or 5-12 cells NiCd/NIMh battery.
BEC	1A / 5V (Linear mode).
Max Speed	210,000rpm for 2 Poles BLM, 70,000rpm for 6 poles BLM, 35,000rpm for 12 poles BLM. (BLM: BrushLess Motor)
1.5 Size	27mm (L) * 17mm (W) * 6mm (H).
Weight	9g.

D.4 InvenSense MPU 6050 Accelerometer + Gyro

The specifications listed here are a brief overview of the MPU 6050. Detailed specifications can be found in the specification sheet, provided by InvenSense.

Table D.5: InvenSense MPU 6050 accelerometer + gyro specifications [1].

Specification	
Part Status	STD
Operating Voltage Supply	2.375V - 3.46V V
Gyro Full Scale Range	±250 ±500 ±1000 ±2000 °/sec
Gyro Sensitivity	131 65.5 32.8 16.4 LSB/°/sec
Accel Full Scale Range	±2 ±4 ±8 ±16 g
Accel Sensitivity	16834 8192 4096 2048
Digital Output	I2C
Logic Supply Voltage	1.71 to VDD V
Package Size	4x4x0.9 mm

In addition, the sensor was used on a breakout board, the GY-521. This breakout board allows a +5V input voltage to be used, due to the boards low drop regulator.

D.5 GY-521 Schematic

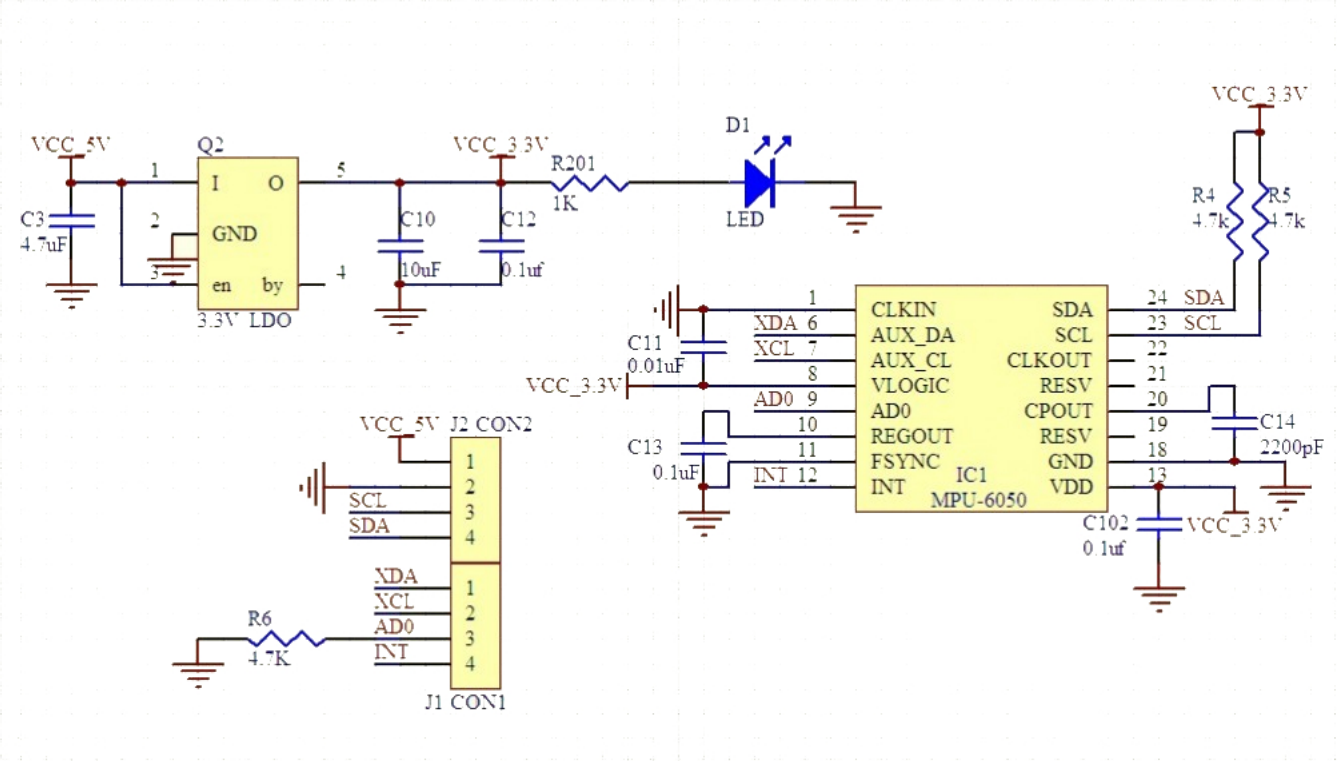


Figure D.1: Schematic of the GY-521 breakout board [1].

Appendix E

BeagleBone Black Cape PCB Design

E.1 Pin Usage

Table E.1: Pin usage on the BeagleBone Black quadcopter.

Pin Usage	Pin
Ground	P9_1
	P9_2
	P9_43
	P9_44
	P9_45
	P9_46
+5V In	P9_7
	P9_8
Motor 1 PWM Signal	P8_13
Motor 2 PWM Signal	P9_14
Motor 3 PWM Signal	P9_22
Motor 4 PWM Signal	P8_19
	P8_2
Motor 1 ADC Input	P9_38
Motor 2 ADC Input	P9_36
Motor 3 ADC Input	P9_39
Motor 4 ADC Input	P9_40
ADC Ground	P9_34
I2C Clock	P9_19
I2C Data	P9_20
Not Connected	P8_1
	P8_2
	P8_45
	P8_46

E.2 Cape PCB Schematic

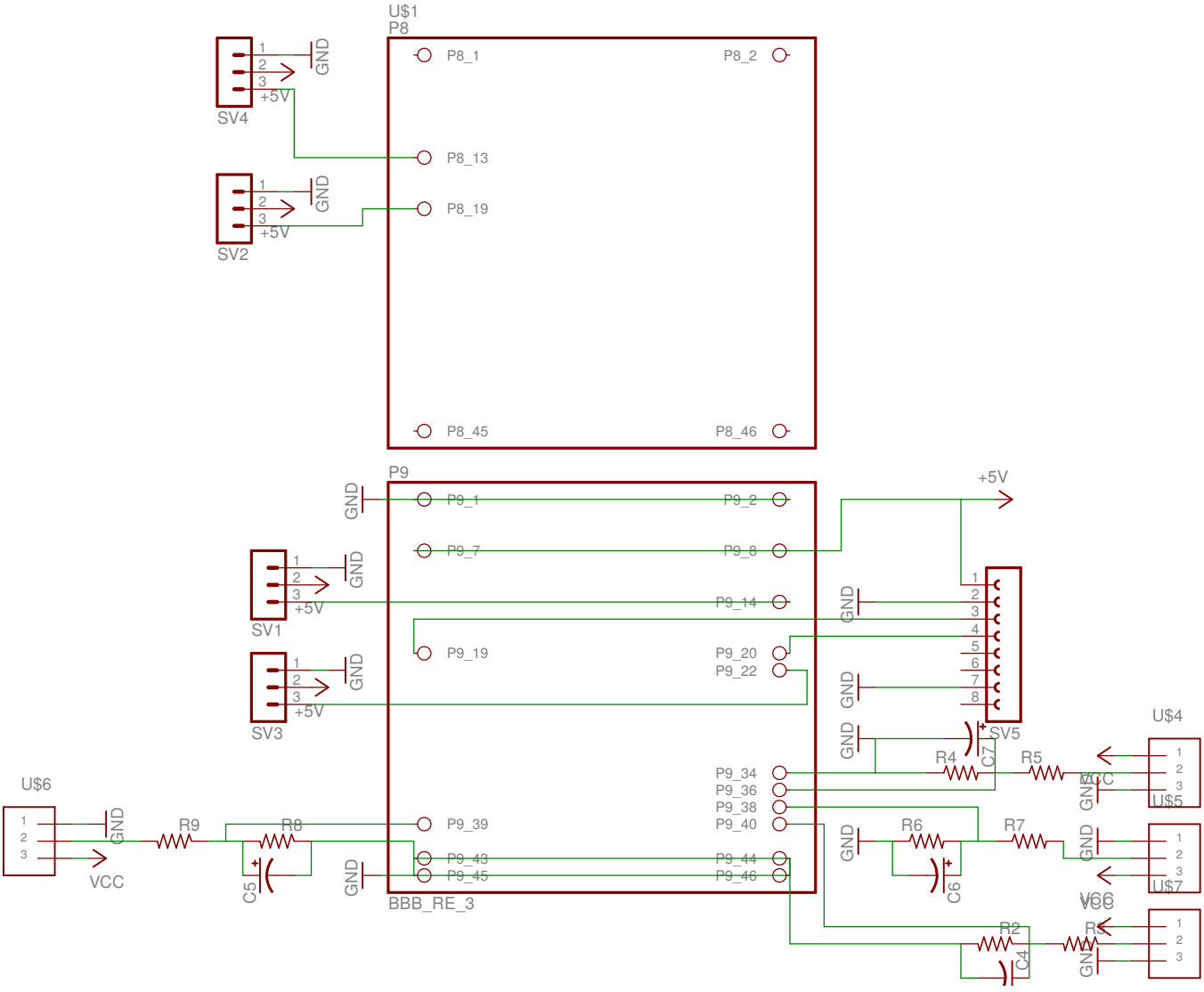


Figure E.1: Schematic of the cape.

E.3 Cape PCB Layout

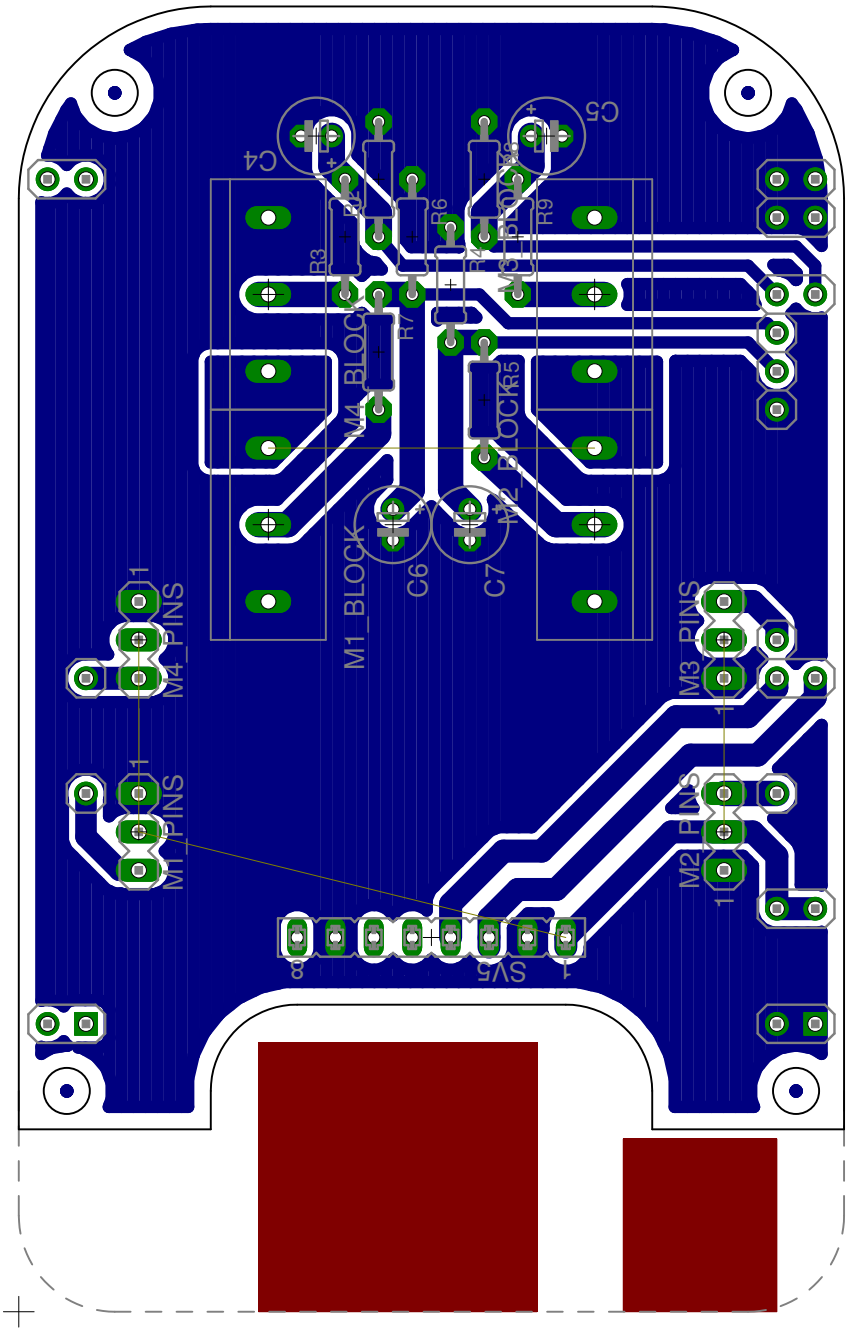


Figure E.2: Layout of the cape.

Appendix F

Experimental Hardware Specs

F.1 3132 Micro Load Cell (0-780g)

Table F.1: 3132 micro load cell (0-780g) specifications [5].

Sensor Properties	
Sensor Type	Shear Load Cell
Weight Capacity Max	780 g
Maximum Overload	936 g
Creep	1.6 g/hr
Zero Balance	± 11.7 g
Cell Repeatability Error Max	± 390 mg
Cell Non-Linearity Max	390 mg
Cell Hysteresis Max	390 mg
Temperature Effect on Span	39 mg/°C
Temperature Effect on Zero	39 mg/°C
Electrical Properties	
Rated Output	800 V/V
Rated Output Error Max	± 100 V/V
Output Impedance	1 kΩ
Supply Voltage Max	5 V DC
Physical Properties	
Compensated Temperature Min	-10 °C
Compensated Temperature Max	40 °C
Operating Temperature Min	-20 °C
Operating Temperature Max	55 °C
Cable Length	200 mm
Cable Gauge	30 AWG
Material	Aluminium Alloy (LY12CZ)
Screw Thread Size	M3x0.5

F.2 LT 1167 Instrumentation Amplifier

Table F.2: LT 1167 instrumentation amplifier specifications [6].

	Features
Single Gain Set Resistor	G = 1 to 10,000
Gain Error	G = 10, 0.08% Max
Input Offset Voltage Drift	0.3V/°C Max
	Meets IEC 1000-4-2 Level 4
	ESD Tests with Two External 5k Resistors
Gain Nonlinearity	G = 10, 10ppm Max
Input Offset Voltage	G = 10, 60μV Max
Input Bias Current	350pA Max
PSRR at G = 1	105dB Min
CMRR at G = 1	90dB Min
Supply Current	1.3mA Max
Wide Supply Range	±2.3V to ±18V
1kHz Voltage Noise	7.5nV/$\sqrt{Hz}$
0.1Hz to 10Hz Noise	0.28VP-P
	Available in 8-Pin PDIP and SO Packages

F.3 TL 7660 CMOS Voltage Converter

Table F.3: TL 7660 CMOS voltage converter specifications [7].

Features

- Simple Voltage Conversion, Including
  - Negative Converter
  - Voltage Doubler
- Wide Operating Range 1.5 V to 10 V
- Requires Only Two External (Noncritical) Capacitors
- No External Diode Over Full Temperature and Voltage Range
- Typical Open-Circuit Voltage Conversion
- Efficiency 99.9%
- Typical Power Efficiency 98%
- Full Testing at 3 V

Appendix G

Experimental Setup

G.1 Experimental Electrical Schematic

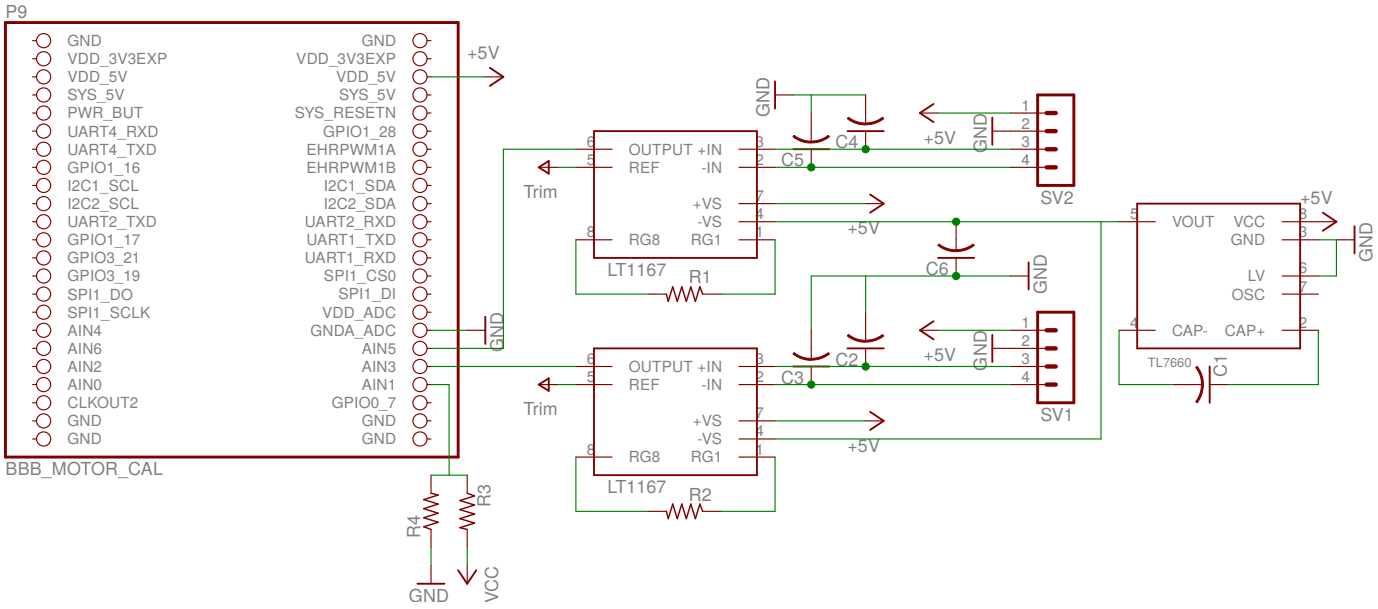


Figure G.1: Electrical schematic of the experimental setup.

G.2 Apparatus CAD Models

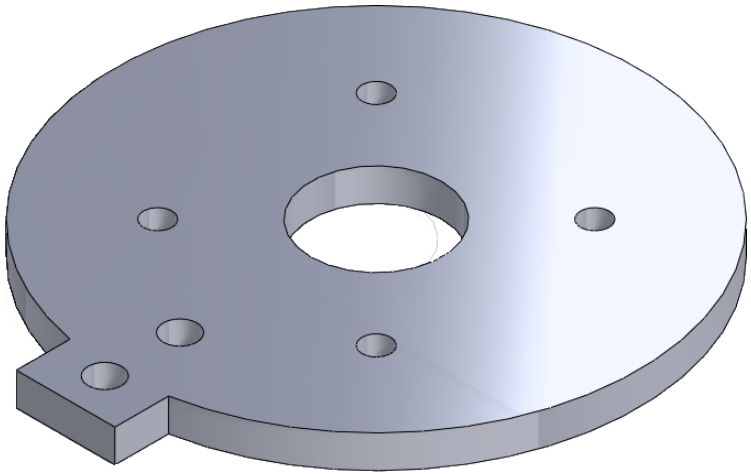


Figure G.2: CAD model of the component where the motor mounts to.

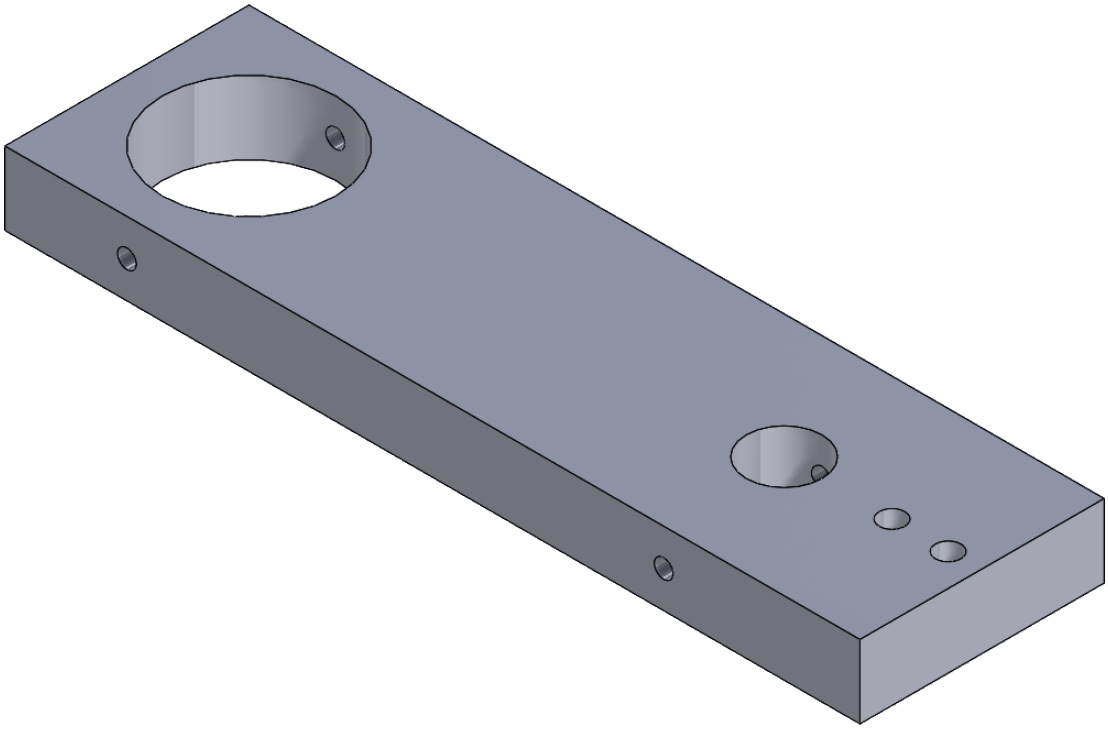


Figure G.3: CAD model of the component which rotates on the encoder.

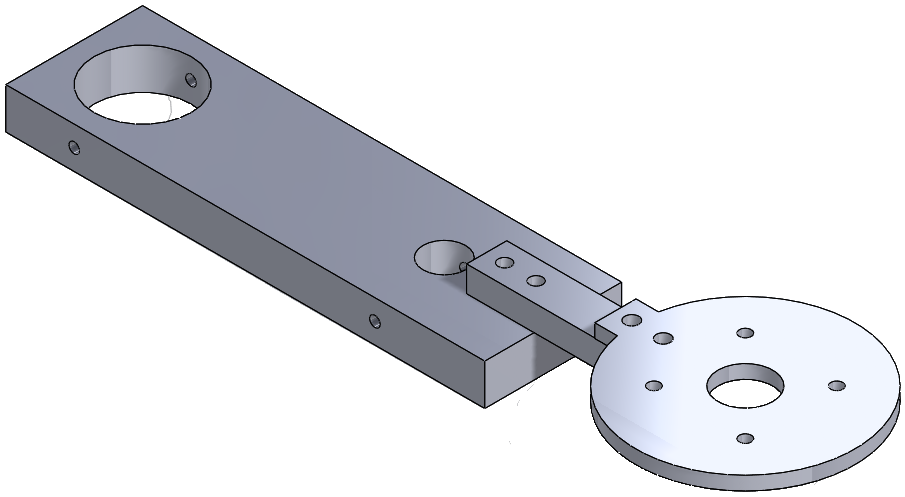


Figure G.4: CAD mock up of the entire apparatus.

G.3 Apparatus CAD Drawings

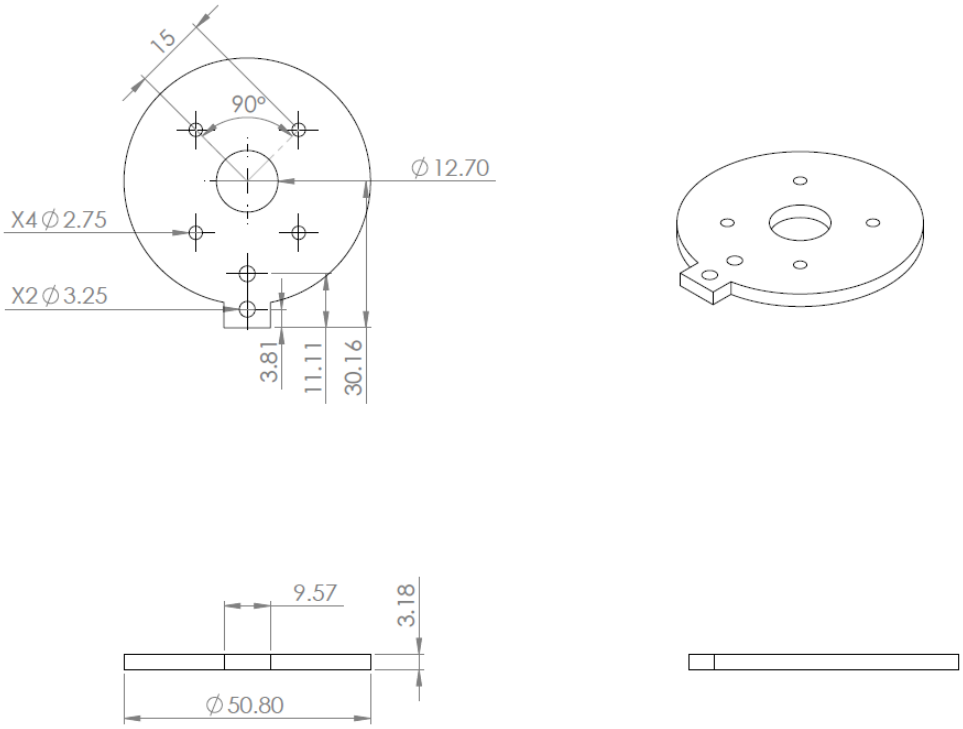


Figure G.5: CAD drawing of element where the motor mounts to. Dimensions are in mm.

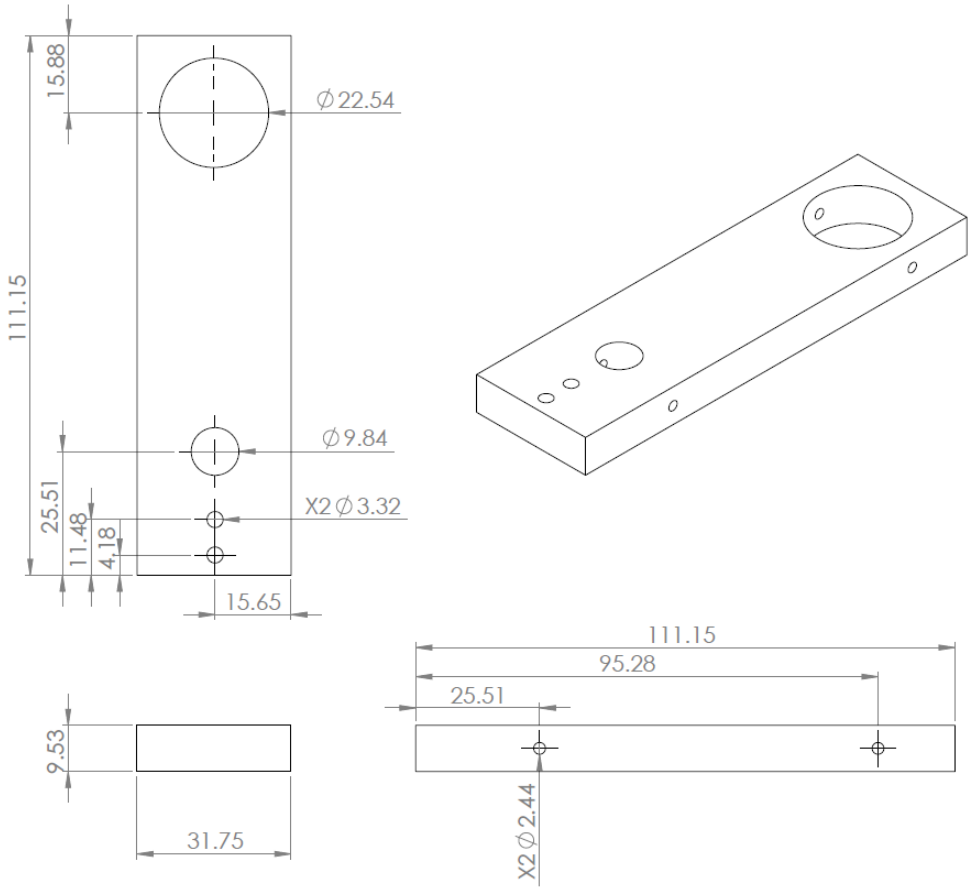


Figure G.6: CAD drawing of element where the motor mounts to. Dimensions are in mm.

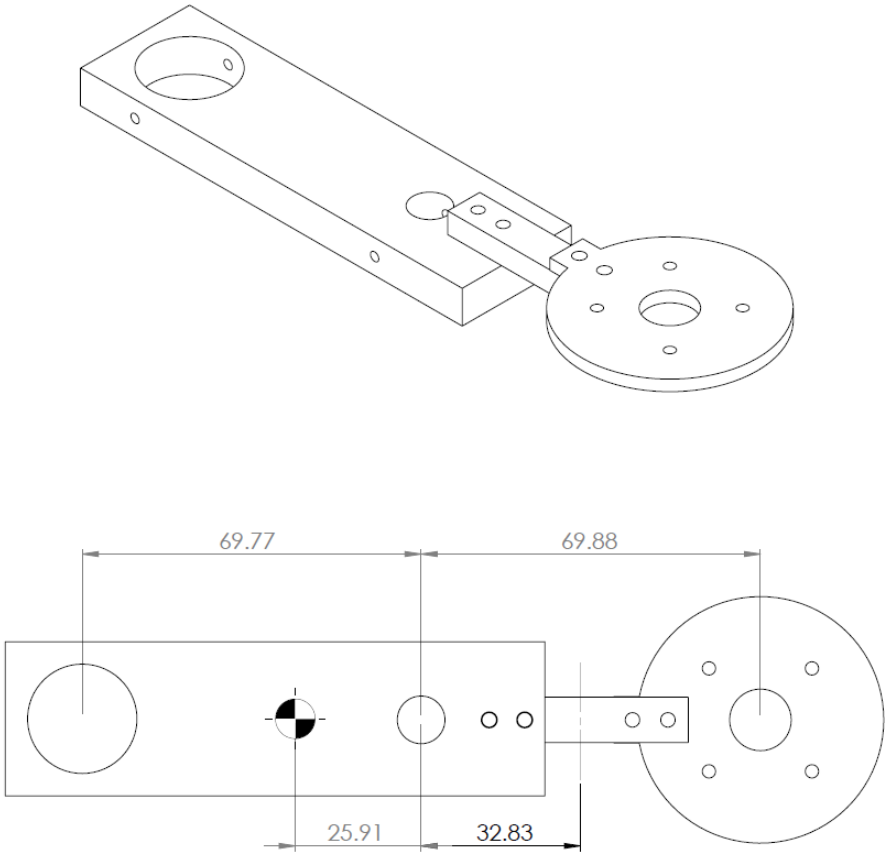


Figure G.7: CAD drawing of the entire apparatus. The dimensions shown refer to distances which are significant to the calculation of reactionary torque. Dimensions are in mm.

Appendix H

Experimentation Code

H.1 Sensor Collection Code

The code used to read the sensor on the quadcopter and then save the readings into a CSV.

```
1  #!/usr/bin/python
2
3  # Sensor_Var_Collection.py
4  # Created by Peter Olejnik
5  #
6  # Purpose - This codes purpose is to read from the MPU6050 and then save the
7  #                        readings into a CSV. 100 samples are taken at 10hz.
8  #
9  ################ Imports ################
10
11 import BBB_QC # Custom Libraries for the QC
12 import os     # identifies what the opperating system is
13 import sys    # related to the system.
14 import signal # detects inputs
15 import time   # time delay
16 import csv    # adds csv functionality
17
18 ################ Initializing Variables ################
19
20 XA_Read = []
21 YA_Read = []
22 ZA_Read = []
23 XG_Read = []
24 YG_Read = []
25 ZG_Read = []
26
27 ################ Ctrl-C Interupt ################
28
```

```
29 def signal_handler(signal, frame):
30
31         print('Program halted! Exiting program!')
32         sys.exit(0)
33
34 ############### Main Program ###############
35
36 os.system('cls' if os.name == 'nt' else 'clear')
37
38 signal.signal(signal.SIGINT, signal_handler)
39
40 BBB_QC.Sensor_Initialize()
41
42 Data = open('SensorData.csv','w')
43
44 # Read Data
45
46 for num in range(0,100,1):
47
48         XA_Raw = BBB_QC.Read_X_A()
49         YA_Raw = BBB_QC.Read_Y_A()
50         ZA_Raw = BBB_QC.Read_Z_A()
51         XG_Raw = BBB_QC.Read_X_G()
52         YG_Raw = BBB_QC.Read_Y_G()
53         ZG_Raw = BBB_QC.Read_Z_G()
54
55         XA_Read.append(XA_Raw)
56         YA_Read.append(YA_Raw)
57         ZA_Read.append(ZA_Raw)
58         XG_Read.append(XG_Raw)
59         YG_Read.append(YG_Raw)
60         ZG_Read.append(ZG_Raw)
61
62         time.sleep(0.1)
63
64 # Save Data
65 csv.writer(Data).writerow(XA_Read)
66 csv.writer(Data).writerow(YA_Read)
67 csv.writer(Data).writerow(ZA_Read)
68 csv.writer(Data).writerow(XG_Read)
69 csv.writer(Data).writerow(YG_Read)
70 csv.writer(Data).writerow(ZG_Read)
71
72
73 Data.close()
```

H.2 Sensor Analysis Code

The code used to analyze the collected sensor readings.

```
% Data Analyser
% Created by Peter Olejnik
%
% Purpose:  This code analyses collected data from the BBB motor test setup
%           and generates fits to them. It generates multiple plots in
%           addition for continuing analysis.
%

clc
clear all
close all

%%%%%%%%%%%%%%% Import Sensor Data %%%%%%%%%%%%%%%

ReadData = csvread('SensorData.csv');
MRD = mean(ReadData')'; % Get mean of sensors

ReadData = ReadData - repmat(MRD,1,size(ReadData,2)); % Zero out data

%%%%%%%%%%%%%%% Create Plots %%%%%%%%%%%%%%%

% Set ranges for plots

Arange = -0.5:0.01:0.5;
Grange = -0.05:0.001:0.05;
Avis = [-0.35,0.35];
Gvis = [-0.035,0.035];

figure % X Accelerometer
hist(ReadData(1,:),Arange)
xlim([min(Avis) max(Avis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (m/s^2)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

figure % Y Accelerometer
hist(ReadData(2,:),Arange)
xlim([min(Avis) max(Avis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (m/s^2)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

figure % Z Accelerometer
hist(ReadData(3,:),Arange)
```

```
xlim([min(Avis) max(Avis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (m/s^2)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

figure % X Gyro
hist(ReadData(4,:),Grange)
xlim([min(Gvis) max(Gvis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (rad/s)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

figure % Y Gyro
hist(ReadData(5,:),Grange)
xlim([min(Gvis) max(Gvis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (rad/s)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

figure % Z Gyro
hist(ReadData(6,:),Grange)
xlim([min(Gvis) max(Gvis)])
set(gca, 'FontSize', 14)
xlabel('Deviation from mean (rad/s)','fontsize',14)
ylabel('Number of Data Points','fontsize',14)

format shortE

SensorVariances = var(ReadData')'

format short

% Save all the plots

h = get(0,'children');

for i=1:length(h)

    savename = strcat('../../Thesis/Figures/Sensor_Var_',num2str(length(h)+1-i));
    set(h(i),'color','w', 'Position', [100, 100, 840, 630]);
    img = getframe(h(i));
    imwrite(img.cdata, [savename, '.png']);
    set(h(i),'color',[0.8 0.8 0.8]);

%     saveas(h(i),savename,'epsc')

end
```

H.3 Load Cell Calibration

This Simulink was the software end of the voltage collection from the load cell. Each weight point was independently collected.

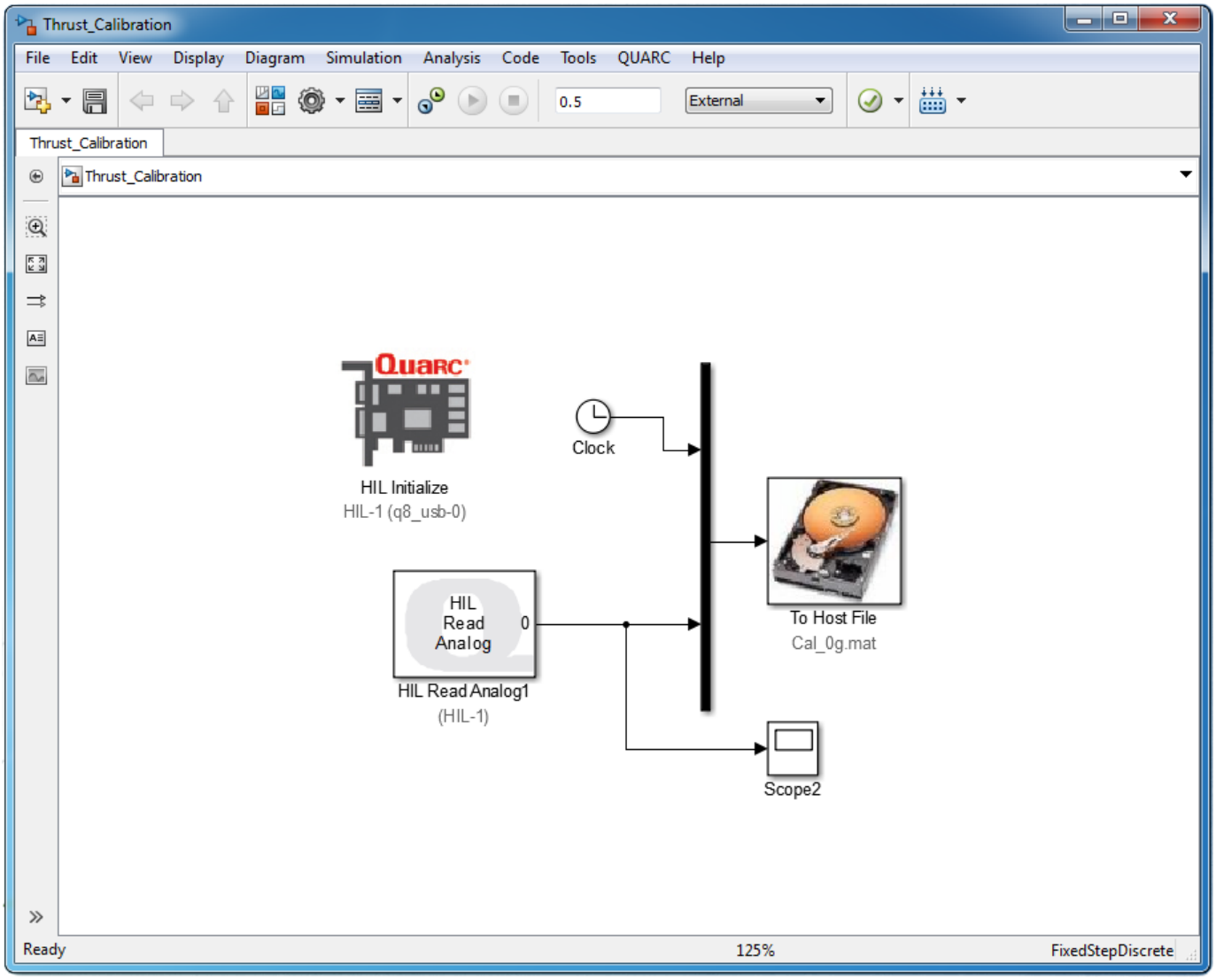


Figure H.1: Simulink used for collecting thrust cell calibration readings.

H.4 Quanser Configuration

These are screen shots of how the Quarc unit was configured for all the data collection.

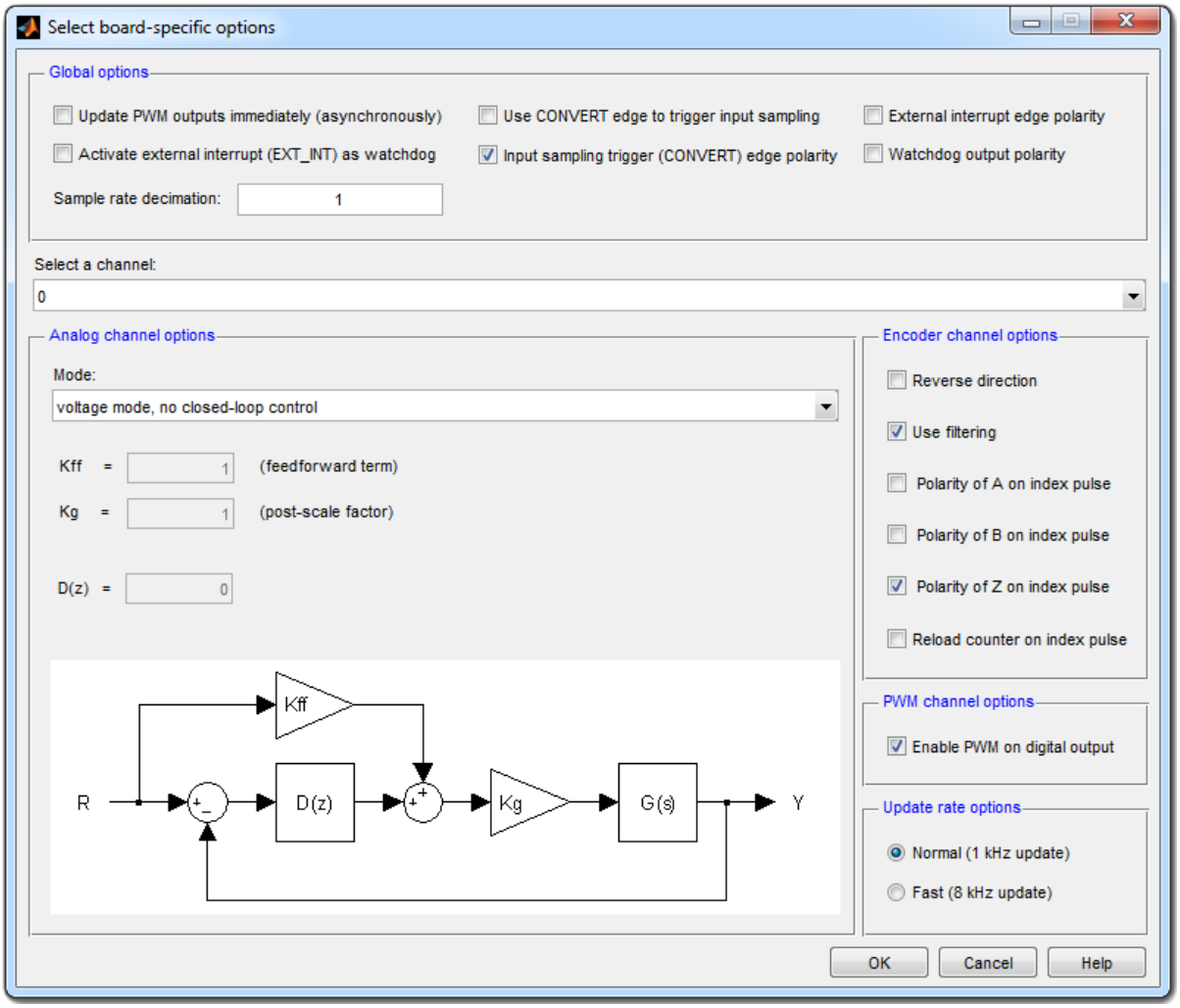


Figure H.2: Configuration of the unit.

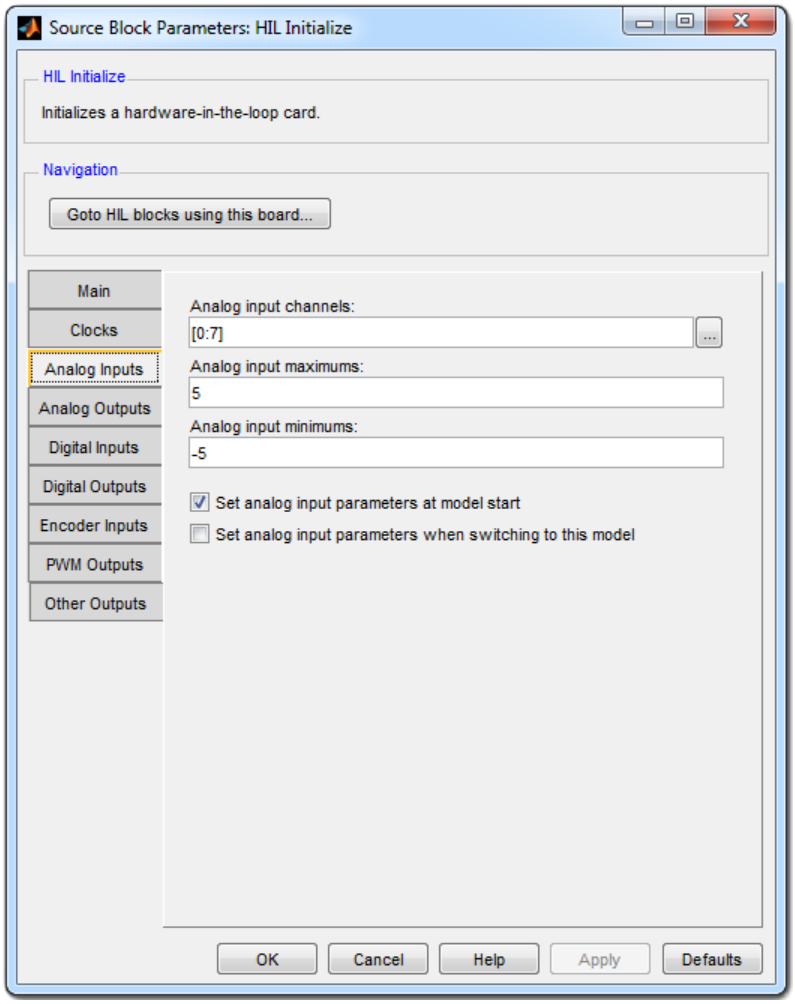


Figure H.3: Configuration of the analog ports.

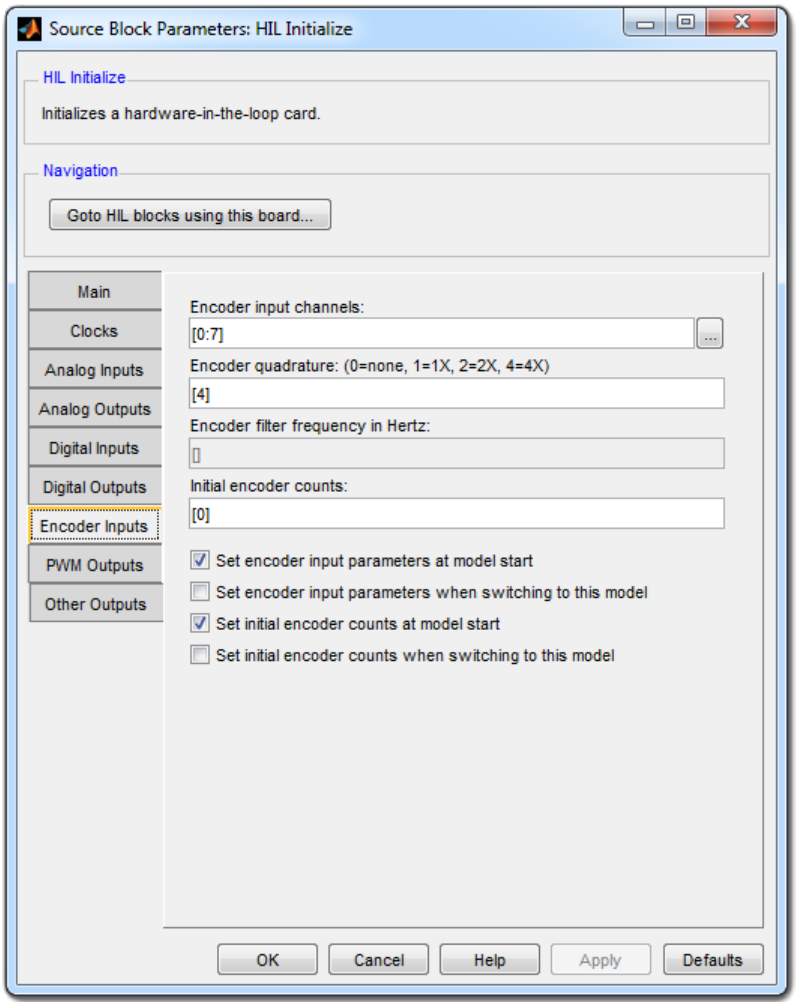


Figure H.4: Configuration of the encoder ports.

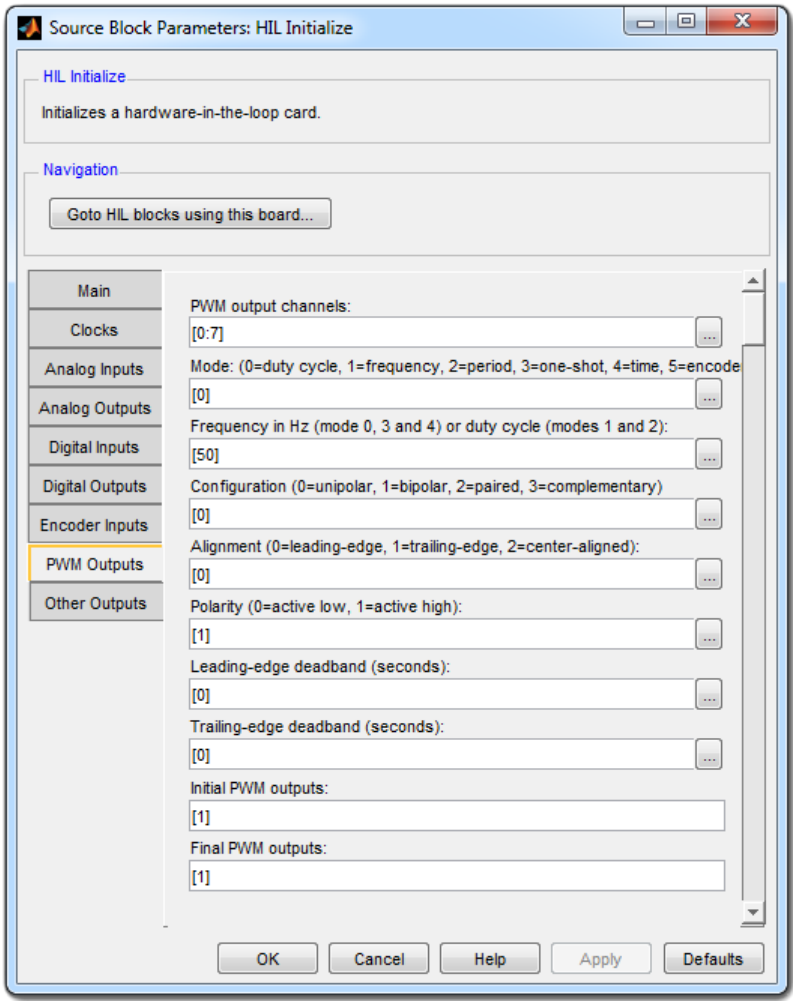


Figure H.5: Configuration of the PWM ports.

H.5 Motor Analysis Code

This is the code which analysis the collected motor data and extracts the thrust, torque, and EMF curves. It also identifies all the ranges. At the end, it exports the values into a .dat file.

```
% Data Analyser
% Created by Peter Olejnik
%
% Purpose:  This code imports all the motor and calibration data. It then
%           cleans up the data to the desired range and creates a thrust,
%           torque, and EMF curve for each motor. It exportes these curves
%           in a .dat file.
%

clc
clear all
```

```
12 close all
13
14 %%%%%% First look at the load cell calibrations and generate them
15
16 %%% Thrust Calabration Data
17
18 load('Cal_0g.mat')
19 Thr_Data_Read = Raw_Results(3,:);
20 load('Cal_50g.mat')
21 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
22 load('Cal_100g.mat')
23 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
24 load('Cal_150g.mat')
25 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
26 load('Cal_200g.mat')
27 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
28 load('Cal_250g.mat')
29 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
30 load('Cal_300g.mat')
31 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
32 load('Cal_350g.mat')
33 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
34 load('Cal_400g.mat')
35 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
36 load('Cal_450g.mat')
37 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
38 load('Cal_500g.mat')
39 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
40 load('Cal_550g.mat')
41 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
42 load('Cal_600g.mat')
43 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
44 load('Cal_650g.mat')
45 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
46 load('Cal_700g.mat')
47 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
48 load('Cal_750g.mat')
49 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
50 load('Cal_780g.mat')
51 Thr_Data_Read = [Thr_Data_Read;Raw_Results(3,:)];
52
53 % Generate means and standard deviations
54 Thr_Cal_Mean = mean(Thr_Data_Read');
55 Thr_Cal_Std = std(Thr_Data_Read');
56
57 Load_Test_Weights = [0:50:750,780];
58
59 % Plot of the read data
60 set(0,'defaultAxesFontSize',16)
```

```
61
62  figure
63
64  subplot(2,1,1)
65  plot(Load_Test_Weights,Thr_Data_Read,'.k',Load_Test_Weights,Thr_Cal_Mean,'.r')
66  set(gca, 'FontSize', 14)
67  xlim([0 780])
68  ylabel('Read voltage, V')
69
70  subplot(2,1,2)
71  plot(Load_Test_Weights,Thr_Cal_Std,'k')
72  set(gca, 'FontSize', 14)
73  xlabel('Force (gram force)')
74  ylabel('Standard deviation, V')
75
76  % Trim data
77
78  Trimmed_Test_Weights = Load_Test_Weights(2:13);
79  Thr_Cal_Mean = Thr_Cal_Mean(2:13);
80  Thr_Cal_Std = Thr_Cal_Std(2:13);
81
82  % Fit a 1st order fit to the data
83  Thr_Fit = polyfit(Trimmed_Test_Weights,Thr_Cal_Mean,1);
84  Thr_Fit_Vals = polyval(Thr_Fit,Load_Test_Weights);
85
86  % Residuals and assumption checking
87  Thrust_Cal_Residuals = Thr_Data_Read(2:13,:) -
    ↪  repmat(Thr_Fit_Vals(2:13)',1,size(Thr_Data_Read,2));
88
89  % Plot of the trimmed data with the fit
90  figure
91  subplot(3,1,1)
92  plot(Load_Test_Weights,Thr_Data_Read,'.k',Load_Test_Weights,Thr_Fit_Vals,Trimmed_Test_Weights,Thr_
93  set(gca, 'FontSize', 14)
94  xlabel('Force (gram force)')
95  ylabel('Read voltage, V')
96  subplot(3,1,2)
97  plot(Trimmed_Test_Weights,Thrust_Cal_Residuals,'.k',[0,650],[0,0],'r')
98  set(gca, 'FontSize', 14)
99  xlim([50,600])
100 xlabel('Force (gram force)')
101 ylabel('Residual, V')
102 subplot(3,1,3)
103 hist(reshape(Thrust_Cal_Residuals,1,[]),-0.02:0.001:0.02)
104 set(gca, 'FontSize', 14)
105 xlabel('Residual, V')
106 ylabel('# of points')
107
108 % Voltage to grams force
```

```
109 Thr_V_Fit = [Thr_Fit(1)^-1,-Thr_Fit(2)/Thr_Fit(1)];
110
111 %%%%% Now Convert motor Data
112
113 Motor_Throttle = 0:100;
114
115 %%% Motor 1
116
117 load('M1_Results.mat');
118 reshaped_res = reshape(Raw_Results(4:end,1:end-1)',4000,[])';
119 Working_Data = reshaped_res(:,end-499:end);
120
121 % Separate the collected data into thrust, torque, and back emf
122 M1_Thr_Read = Working_Data(1:101,:);
123 M1_Tor_Read = Working_Data(203:303,:);
124 M1_EMF_Read = Working_Data(102:202,:);
125
126 M1_Thr_Org = M1_Thr_Read;
127 M1_Tor_Org = M1_Tor_Read;
128 M1_EMF_Org = M1_EMF_Read;
129
130 % generate the thrust, torque, and emf means and standard deviations
131 M1_Thr_Mean = mean(M1_Thr_Read');
132 M1_Thr_Std = std(M1_Thr_Read');
133
134 M1_Tor_Mean = mean(M1_Tor_Read');
135 M1_Tor_Std = std(M1_Tor_Read');
136
137 M1_EMF_Mean = mean(M1_EMF_Read');
138 M1_EMF_Std = std(M1_EMF_Read');
139
140
141 % plot the found results
142 % thrust
143 figure
144 subplot(2,1,1)
145 plot(Motor_Throttle,M1_Thr_Read,'.k',Motor_Throttle,M1_Thr_Mean,'.r')
146 set(gca, 'FontSize', 14)
147 ylabel('Thrust LC voltage, V')
148
149 subplot(2,1,2)
150 plot(Motor_Throttle,M1_Thr_Std','k')
151 set(gca, 'FontSize', 14)
152 xlabel('Throttle, %')
153 ylabel('Standard deviation, V')
154
155 % torque
156 figure
157 subplot(2,1,1)
```

```
158 plot(Motor_Throttle,M1_Tor_Read,'.k',Motor_Throttle,M1_Tor_Mean,'.r')
159 set(gca, 'FontSize', 14)
160 ylabel('Encoder Counts')
161
162 subplot(2,1,2)
163 plot(Motor_Throttle,M1_Tor_Std','k')
164 set(gca, 'FontSize', 14)
165 xlabel('Throttle, %')
166 ylabel('Standard deviation, Counts')
167
168 % back emf
169 figure
170 subplot(2,1,1)
171 plot(Motor_Throttle,M1_EMF_Read,'.k',Motor_Throttle,M1_EMF_Mean,'.r')
172 set(gca, 'FontSize', 14)
173 ylabel('Back EMF voltage, V')
174
175 subplot(2,1,2)
176 plot(Motor_Throttle,M1_EMF_Std','k')
177 set(gca, 'FontSize', 14)
178 xlabel('Throttle, %')
179 ylabel('Standard deviation, V')
180
181
182 % Identify at what throttle the motors start working and when they max out
183 % looking at the derivative, they start at the large spike and they max out
184 % where the abs(derivative) falls below 5*10^-3
185 M1_EMF_Dif = abs(M1_EMF_Mean(2:end)-M1_EMF_Mean(1:end-1));
186
187 M1_Min_Throt = find(M1_EMF_Dif == max(M1_EMF_Dif));
188 M1_Max_Throt = find(M1_EMF_Dif>5*10^-3,1,'last');
189
190 % Trim the data form min to max
191 M1_Throt = M1_Min_Throt:M1_Max_Throt;
192
193 M1_Thr_Read = M1_Thr_Read(M1_Min_Throt+1:M1_Max_Throt+1,:);
194 M1_Tor_Read = M1_Tor_Read(M1_Min_Throt+1:M1_Max_Throt+1,:);
195 M1_EMF_Read = M1_EMF_Read(M1_Min_Throt+1:M1_Max_Throt+1,:);
196
197 M1_Thr_Mean = M1_Thr_Mean(M1_Min_Throt+1:M1_Max_Throt+1);
198 M1_Thr_Std = M1_Thr_Std(M1_Min_Throt+1:M1_Max_Throt+1);
199
200 M1_Tor_Mean = M1_Tor_Mean(M1_Min_Throt+1:M1_Max_Throt+1);
201 M1_Tor_Std = M1_Tor_Std(M1_Min_Throt+1:M1_Max_Throt+1);
202
203 M1_EMF_Mean = M1_EMF_Mean(M1_Min_Throt+1:M1_Max_Throt+1);
204 M1_EMF_Std = M1_EMF_Std(M1_Min_Throt+1:M1_Max_Throt+1);
205
206 % Plot the trimmed data
```

```
% thrust
figure
plot(M1_Throt,M1_Thr_Read,'.k',M1_Throt,M1_Thr_Mean,'.r')
set(gca, 'FontSize', 14)
xlim([min(M1_Throt) max(M1_Throt)])
xlabel('Throttle, %')
ylabel('Thrust LC voltage, V')

% torque
figure
plot(M1_Throt,M1_Tor_Read,'.k',M1_Throt,M1_Tor_Mean,'.r')
set(gca, 'FontSize', 14)
xlim([min(M1_Throt) max(M1_Throt)])
xlabel('Throttle, %')
ylabel('Encoder Counts')

% back emf
figure
plot(M1_Throt,M1_EMF_Read,'.k',M1_Throt,M1_EMF_Mean,'.r')
set(gca, 'FontSize', 14)
xlim([min(M1_Throt) max(M1_Throt)])
xlabel('Throttle, %')
ylabel('Back EMF voltage, V')

%%% Motor 2

load('M2_Results.mat');
reshaped_res = reshape(Raw_Results(4:end,1:end-1)',4000,[])';
Working_Data = reshaped_res(:,end-499:end);

% Separate the collected data into thrust, torque, and back emf
M2_Thr_Read = Working_Data(1:101,:);
M2_Tor_Read = Working_Data(203:303,:);
M2_EMF_Read = Working_Data(102:202,:);


M2_Thr_Org = M2_Thr_Read;
M2_Tor_Org = M2_Tor_Read;
M2_EMF_Org = M2_EMF_Read;

% generate the thrust, torque, and emf means and standard deviations
M2_Thr_Mean = mean(M2_Thr_Read');
M2_Thr_Std = std(M2_Thr_Read');

M2_Tor_Mean = mean(M2_Tor_Read');
M2_Tor_Std = std(M2_Tor_Read');

M2_EMF_Mean = mean(M2_EMF_Read');
M2_EMF_Std = std(M2_EMF_Read');
```

```
256
257 % plot the found results
258 % thrust
259 figure
260 subplot(2,1,1)
261 plot(Motor_Throttle,M2_Thr_Read,'.k',Motor_Throttle,M2_Thr_Mean,'.r')
262 set(gca, 'FontSize', 14)
263 ylabel('Thrust LC voltage, V')
264
265 subplot(2,1,2)
266 plot(Motor_Throttle,M2_Thr_Std','k')
267 set(gca, 'FontSize', 14)
268 xlabel('Throttle, %')
269 ylabel('Standard deviation, V')
270
271 % torque
272 figure
273 subplot(2,1,1)
274 plot(Motor_Throttle,M2_Tor_Read,'.k',Motor_Throttle,M2_Tor_Mean,'.r')
275 set(gca, 'FontSize', 14)
276 ylabel('Encoder Counts')
277
278 subplot(2,1,2)
279 plot(Motor_Throttle,M2_Tor_Std','k')
280 set(gca, 'FontSize', 14)
281 xlabel('Throttle, %')
282 ylabel('Standard deviation, Counts')
283
284 % back emf
285 figure
286 subplot(2,1,1)
287 plot(Motor_Throttle,M2_EMF_Read,'.k',Motor_Throttle,M2_EMF_Mean,'.r')
288 set(gca, 'FontSize', 14)
289 ylabel('Back EMF voltage, V')
290
291 subplot(2,1,2)
292 plot(Motor_Throttle,M2_EMF_Std','k')
293 set(gca, 'FontSize', 14)
294 xlabel('Throttle, %')
295 ylabel('Standard deviation, V')
296
297
298 % Identify at what throttle the motors start working and when they max out
299 % looking at the derivative, they start at the large spike and they max out
300 % where the abs(derivative) falls below 5*10^-3
301 M2_EMF_Dif = abs(M2_EMF_Mean(2:end)-M2_EMF_Mean(1:end-1));
302
303 M2_Min_Throt = find(M2_EMF_Dif == max(M2_EMF_Dif));
304 M2_Max_Throt = find(M2_EMF_Dif>5*10^-3,1,'last');
```

```
305
306 % Trim the data form min to max
307 M2_Throt = M2_Min_Throt:M2_Max_Throt;
308
309 M2_Thr_Read = M2_Thr_Read(M2_Min_Throt+1:M2_Max_Throt+1,:);
310 M2_Tor_Read = M2_Tor_Read(M2_Min_Throt+1:M2_Max_Throt+1,:);
311 M2_EMF_Read = M2_EMF_Read(M2_Min_Throt+1:M2_Max_Throt+1,:);
312
313 M2_Thr_Mean = M2_Thr_Mean(M2_Min_Throt+1:M2_Max_Throt+1);
314 M2_Thr_Std = M2_Thr_Std(M2_Min_Throt+1:M2_Max_Throt+1);
315
316 M2_Tor_Mean = M2_Tor_Mean(M2_Min_Throt+1:M2_Max_Throt+1);
317 M2_Tor_Std = M2_Tor_Std(M2_Min_Throt+1:M2_Max_Throt+1);
318
319 M2_EMF_Mean = M2_EMF_Mean(M2_Min_Throt+1:M2_Max_Throt+1);
320 M2_EMF_Std = M2_EMF_Std(M2_Min_Throt+1:M2_Max_Throt+1);
321
322 % Plot the trimmed data
323 % thrust
324 figure
325 plot(M2_Throt,M2_Thr_Read,'.k',M2_Throt,M2_Thr_Mean,'.r')
326 set(gca, 'FontSize', 14)
327 xlim([min(M2_Throt) max(M2_Throt)])
328 xlabel('Throttle, %')
329 ylabel('Thrust LC voltage, V')
330
331 % torque
332 figure
333 plot(M2_Throt,M2_Tor_Read,'.k',M2_Throt,M2_Tor_Mean,'.r')
334 set(gca, 'FontSize', 14)
335 xlim([min(M2_Throt) max(M2_Throt)])
336 xlabel('Throttle, %')
337 ylabel('Encoder Counts')
338
339 % back emf
340 figure
341 plot(M2_Throt,M2_EMF_Read,'.k',M2_Throt,M2_EMF_Mean,'.r')
342 set(gca, 'FontSize', 14)
343 xlim([min(M2_Throt) max(M2_Throt)])
344 xlabel('Throttle, %')
345 ylabel('Back EMF voltage, V')
346
347 %%% Motor 3
348
349 load('M3_Results.mat');
350 reshaped_res = reshape(Raw_Results(4:end,1:end-1)',4000,[])';
351 Working_Data = reshaped_res(:,end-499:end);
352
353 % Separate the collected data into thrust, torque, and back emf
```

```
354 M3_Thr_Read = Working_Data(1:101,:);
355 M3_Tor_Read = Working_Data(203:303,:);
356 M3_EMF_Read = Working_Data(102:202,:);
357
358
359 M3_Thr_Org = M3_Thr_Read;
360 M3_Tor_Org = M3_Tor_Read;
361 M3_EMF_Org = M3_EMF_Read;
362
363 % generate the thrust, torque, and emf means and standard deviations
364 M3_Thr_Mean = mean(M3_Thr_Read');
365 M3_Thr_Std = std(M3_Thr_Read');
366
367 M3_Tor_Mean = mean(M3_Tor_Read');
368 M3_Tor_Std = std(M3_Tor_Read');
369
370 M3_EMF_Mean = mean(M3_EMF_Read');
371 M3_EMF_Std = std(M3_EMF_Read');
372
373 % plot the found results
374 % thrust
375 figure
376 subplot(2,1,1)
377 plot(Motor_Throttle,M3_Thr_Read,'.k',Motor_Throttle,M3_Thr_Mean,'.r')
378 set(gca, 'FontSize', 14)
379 ylabel('Thrust LC voltage, V')
380
381 subplot(2,1,2)
382 plot(Motor_Throttle,M3_Thr_Std','k')
383 set(gca, 'FontSize', 14)
384 xlabel('Throttle, %')
385 ylabel('Standard deviation, V')
386
387 % torque
388 figure
389 subplot(2,1,1)
390 plot(Motor_Throttle,M3_Tor_Read,'.k',Motor_Throttle,M3_Tor_Mean,'.r')
391 set(gca, 'FontSize', 14)
392 ylabel('Encoder Counts')
393
394 subplot(2,1,2)
395 plot(Motor_Throttle,M3_Tor_Std','k')
396 set(gca, 'FontSize', 14)
397 xlabel('Throttle, %')
398 ylabel('Standard deviation, Counts')
399
400 % back emf
401 figure
402 subplot(2,1,1)
```

```
403 plot(Motor_Throttle,M3_EMF_Read,'.k',Motor_Throttle,M3_EMF_Mean,'.r')
404 set(gca, 'FontSize', 14)
405 ylabel('Back EMF voltage, V')
406
407 subplot(2,1,2)
408 plot(Motor_Throttle,M3_EMF_Std','k')
409 set(gca, 'FontSize', 14)
410 xlabel('Throttle, %')
411 ylabel('Standard deviation, V')
412
413
414 % Identify at what throttle the motors start working and when they max out
415 % looking at the derivative, they start at the large spike and they max out
416 % where the abs(derivative) falls below 5*10^-3
417 M3_EMF_Dif = abs(M3_EMF_Mean(2:end)-M3_EMF_Mean(1:end-1));
418
419 M3_Min_Throt = find(M3_EMF_Dif == max(M3_EMF_Dif));
420 M3_Max_Throt = find(M3_EMF_Dif>5*10^-3,1,'last');
421
422 % Trim the data form min to max
423 M3_Throt = M3_Min_Throt:M3_Max_Throt;
424
425 M3_Thr_Read = M3_Thr_Read(M3_Min_Throt+1:M3_Max_Throt+1,:);
426 M3_Tor_Read = M3_Tor_Read(M3_Min_Throt+1:M3_Max_Throt+1,:);
427 M3_EMF_Read = M3_EMF_Read(M3_Min_Throt+1:M3_Max_Throt+1,:);
428
429 M3_Thr_Mean = M3_Thr_Mean(M3_Min_Throt+1:M3_Max_Throt+1);
430 M3_Thr_Std = M3_Thr_Std(M3_Min_Throt+1:M3_Max_Throt+1);
431
432 M3_Tor_Mean = M3_Tor_Mean(M3_Min_Throt+1:M3_Max_Throt+1);
433 M3_Tor_Std = M3_Tor_Std(M3_Min_Throt+1:M3_Max_Throt+1);
434
435 M3_EMF_Mean = M3_EMF_Mean(M3_Min_Throt+1:M3_Max_Throt+1);
436 M3_EMF_Std = M3_EMF_Std(M3_Min_Throt+1:M3_Max_Throt+1);
437
438 % Plot the trimmed data
439 % thrust
440 figure
441 plot(M3_Throt,M3_Thr_Read,'.k',M3_Throt,M3_Thr_Mean,'.r')
442 set(gca, 'FontSize', 14)
443 xlim([min(M3_Throt) max(M3_Throt)])
444 xlabel('Throttle, %')
445 ylabel('Thrust LC voltage, V')
446
447 % torque
448 figure
449 plot(M3_Throt,M3_Tor_Read,'.k',M3_Throt,M3_Tor_Mean,'.r')
450 set(gca, 'FontSize', 14)
451 xlim([min(M3_Throt) max(M3_Throt)])
```

```
452 xlabel('Throttle, %')
453 ylabel('Encoder Counts')
454
455 % back emf
456 figure
457 plot(M3_Throt,M3_EMF_Read,'.k',M3_Throt,M3_EMF_Mean,'.r')
458 set(gca, 'FontSize', 14)
459 xlim([min(M3_Throt) max(M3_Throt)])
460 xlabel('Throttle, %')
461 ylabel('Back EMF voltage, V')
462
463
464 %%% Motor 4
465
466 load('M4_Results.mat');
467 reshaped_res = reshape(Raw_Results(4:end,1:end-1)',4000,[])';
468 Working_Data = reshaped_res(:,end-499:end);
469
470 % Separate the collected data into thrust, torque, and back emf
471 M4_Thr_Read = Working_Data(1:101,:);
472 M4_Tor_Read = Working_Data(203:303,:);
473 M4_EMF_Read = Working_Data(102:202,:);
474
475
476 M4_Thr_Org = M4_Thr_Read;
477 M4_Tor_Org = M4_Tor_Read;
478 M4_EMF_Org = M4_EMF_Read;
479
480 % generate the thrust, torque, and emf means and standard deviations
481 M4_Thr_Mean = mean(M4_Thr_Read');
482 M4_Thr_Std = std(M4_Thr_Read');
483
484 M4_Tor_Mean = mean(M4_Tor_Read');
485 M4_Tor_Std = std(M4_Tor_Read');
486
487 M4_EMF_Mean = mean(M4_EMF_Read');
488 M4_EMF_Std = std(M4_EMF_Read');
489
490 % plot the found results
491 % thrust
492 figure
493 subplot(2,1,1)
494 plot(Motor_Throttle,M4_Thr_Read,'.k',Motor_Throttle,M4_Thr_Mean,'.r')
495 set(gca, 'FontSize', 14)
496 ylabel('Thrust LC voltage, V')
497
498 subplot(2,1,2)
499 plot(Motor_Throttle,M4_Thr_Std','k')
500 set(gca, 'FontSize', 14)
```

```
xlabel('Throttle, %')
ylabel('Standard deviation, V')

% torque
figure
subplot(2,1,1)
plot(Motor_Throttle,M4_Tor_Read,'.k',Motor_Throttle,M4_Tor_Mean,'.r')
set(gca, 'FontSize', 14)
ylabel('Encoder Counts')

subplot(2,1,2)
plot(Motor_Throttle,M4_Tor_Std','k')
set(gca, 'FontSize', 14)
xlabel('Throttle, %')
ylabel('Standard deviation, Counts')

% back emf
figure
subplot(2,1,1)
plot(Motor_Throttle,M4_EMF_Read,'.k',Motor_Throttle,M4_EMF_Mean,'.r')
set(gca, 'FontSize', 14)
ylabel('Back EMF voltage, V')

subplot(2,1,2)
plot(Motor_Throttle,M4_EMF_Std','k')
set(gca, 'FontSize', 14)
xlabel('Throttle, %')
ylabel('Standard deviation, V')


% Identify at what throttle the motors start working and when they max out
% looking at the derivative, they start at the large spike and they max out
% where the abs(derivative) falls below 5*10^-3
M4_EMF_Dif = abs(M4_EMF_Mean(2:end)-M4_EMF_Mean(1:end-1));

M4_Min_Throt = find(M4_EMF_Dif == max(M4_EMF_Dif));
M4_Max_Throt = find(M4_EMF_Dif>5*10^-3,1,'last');

% Trim the data form min to max
M4_Throt = M4_Min_Throt:M4_Max_Throt;

% Trim the results
M4_Thr_Read = M4_Thr_Read(M4_Min_Throt+1:M4_Max_Throt+1,:);
M4_Tor_Read = M4_Tor_Read(M4_Min_Throt+1:M4_Max_Throt+1,:);
M4_EMF_Read = M4_EMF_Read(M4_Min_Throt+1:M4_Max_Throt+1,:);

M4_Thr_Mean = M4_Thr_Mean(M4_Min_Throt+1:M4_Max_Throt+1);
M4_Thr_Std = M4_Thr_Std(M4_Min_Throt+1:M4_Max_Throt+1);

```

```
550 M4_Tor_Mean = M4_Tor_Mean(M4_Min_Throt+1:M4_Max_Throt+1);
551 M4_Tor_Std = M4_Tor_Std(M4_Min_Throt+1:M4_Max_Throt+1);
552
553 M4_EMF_Mean = M4_EMF_Mean(M4_Min_Throt+1:M4_Max_Throt+1);
554 M4_EMF_Std = M4_EMF_Std(M4_Min_Throt+1:M4_Max_Throt+1);
555
556 % Plot the trimmed data
557 % thrust
558 figure
559 plot(M4_Throt,M4_Thr_Read,'.k',M4_Throt,M4_Thr_Mean,'.r')
560 set(gca, 'FontSize', 14)
561 xlim([min(M4_Throt) max(M4_Throt)])
562 xlabel('Throttle, %')
563 ylabel('Thrust LC voltage, V')
564
565 % torque
566 figure
567 plot(M4_Throt,M4_Tor_Read,'.k',M4_Throt,M4_Tor_Mean,'.r')
568 set(gca, 'FontSize', 14)
569 xlim([min(M4_Throt) max(M4_Throt)])
570 xlabel('Throttle, %')
571 ylabel('Encoder Counts')
572
573 % back emf
574 figure
575 plot(M4_Throt,M4_EMF_Read,'.k',M4_Throt,M4_EMF_Mean,'.r')
576 set(gca, 'FontSize', 14)
577 xlim([min(M4_Throt) max(M4_Throt)])
578 xlabel('Throttle, %')
579 ylabel('Back EMF voltage, V')
580
581 %%%% Conversion from V to Thrust/Torque
582
583 ThrustConvFact = 9.81/1000; % Grams to Newtons
584
585 M1_Thr_Vals = polyval(Thr_V_Fit,M1_Thr_Mean)*ThrustConvFact;
586 M2_Thr_Vals = polyval(Thr_V_Fit,M2_Thr_Mean)*ThrustConvFact;
587 M3_Thr_Vals = polyval(Thr_V_Fit,M3_Thr_Mean)*ThrustConvFact;
588 M4_Thr_Vals = polyval(Thr_V_Fit,M4_Thr_Mean)*ThrustConvFact;
589
590 M1_Thr_Org_Vals = polyval(Thr_V_Fit,M1_Thr_Org)*ThrustConvFact;
591 M2_Thr_Org_Vals = polyval(Thr_V_Fit,M2_Thr_Org)*ThrustConvFact;
592 M3_Thr_Org_Vals = polyval(Thr_V_Fit,M3_Thr_Org)*ThrustConvFact;
593 M4_Thr_Org_Vals = polyval(Thr_V_Fit,M4_Thr_Org)*ThrustConvFact;
594
595 EncoderConvFact = 1/(4*2608); % Encoder ticks to radians
596 TorqueConvFact = ...
597     (15.48*25.91 + 100*69.77 - 6.44*32.83 - (3.83+28.74)*69.88)...
598     *9.81/(1000*1000); % sin(angle) to
```

```
599
600 M1_Tor_Vals = sin(M1_Tor_Mean*EncoderConvFact)*TorqueConvFact;
601 M2_Tor_Vals = sin(M2_Tor_Mean*EncoderConvFact)*TorqueConvFact;
602 M3_Tor_Vals = sin(M3_Tor_Mean*EncoderConvFact)*TorqueConvFact;
603 M4_Tor_Vals = sin(M4_Tor_Mean*EncoderConvFact)*TorqueConvFact;
604
605 M1_Tor_Org_Vals = sin(M1_Tor_Org*EncoderConvFact)*TorqueConvFact;
606 M2_Tor_Org_Vals = sin(M2_Tor_Org*EncoderConvFact)*TorqueConvFact;
607 M3_Tor_Org_Vals = sin(M3_Tor_Org*EncoderConvFact)*TorqueConvFact;
608 M4_Tor_Org_Vals = sin(M4_Tor_Org*EncoderConvFact)*TorqueConvFact;
609
610 %%% Offset so first value is zero
611
612 M1_Thr_Org_Vals = M1_Thr_Org_Vals - M1_Thr_Vals(1);
613 M2_Thr_Org_Vals = M2_Thr_Org_Vals - M1_Thr_Vals(1);
614 M3_Thr_Org_Vals = M3_Thr_Org_Vals - M1_Thr_Vals(1);
615 M4_Thr_Org_Vals = M4_Thr_Org_Vals - M1_Thr_Vals(1);
616
617 M1_Thr_Vals = M1_Thr_Vals - M1_Thr_Vals(1);
618 M2_Thr_Vals = M2_Thr_Vals - M2_Thr_Vals(1);
619 M3_Thr_Vals = M3_Thr_Vals - M3_Thr_Vals(1);
620 M4_Thr_Vals = M4_Thr_Vals - M4_Thr_Vals(1);
621
622 M1_Tor_Org_Vals = M1_Tor_Org_Vals - M1_Tor_Vals(1);
623 M2_Tor_Org_Vals = M2_Tor_Org_Vals - M2_Tor_Vals(1);
624 M3_Tor_Org_Vals = M3_Tor_Org_Vals - M3_Tor_Vals(1);
625 M4_Tor_Org_Vals = M4_Tor_Org_Vals - M4_Tor_Vals(1);
626
627 M1_Tor_Vals = M1_Tor_Vals - M1_Tor_Vals(1);
628 M2_Tor_Vals = M2_Tor_Vals - M2_Tor_Vals(1);
629 M3_Tor_Vals = M3_Tor_Vals - M3_Tor_Vals(1);
630 M4_Tor_Vals = M4_Tor_Vals - M4_Tor_Vals(1);
631
632
633 %%% And the fitting begins
634 Thrust_Order = 2;
635 Torque_Order = 2;
636 EMF_Order = 3;
637
638 % Motor 1 thrust fit
639 % y = a*x^2 + b*x + c
640
641 M1_Thr_Fit = polyfit(M1_Throt,M1_Thr_Vals,Thrust_Order);
642 M1_Thr_Fit_Vals = polyval(M1_Thr_Fit,M1_Throt);
643
644 % Residuals and assumption checking
645 M1_Thr_Res = M1_Thr_Org_Vals(min(M1_Throt)+1:1:max(M1_Throt)+1,:) - ...
646     repmat(M1_Thr_Fit_Vals',1,size(M1_Thr_Org_Vals,2));
647
```

```
% Motor 1 torque fit
% y = a*x^2 + b*x + c
M1_Tor_Fit = polyfit(M1_Throt,M1_Tor_Vals,Torque_Order);
M1_Tor_Fit_Vals = polyval(M1_Tor_Fit,M1_Throt);

% Residuals and assumption checking
M1_Tor_Res = M1_Tor_Org_Vals(min(M1_Throt)+1:1:max(M1_Throt)+1,:) - ...
    repmat(M1_Tor_Fit_Vals',1,size(M1_Tor_Org_Vals,2));

% Motor 1 EMF fit
% y=a*x+b
M1_EMF_Fit = polyfit(M1_Throt,M1_EMF_Mean,EMF_Order);
M1_EMF_Fit_Vals = polyval(M1_EMF_Fit,M1_Throt);

% Residuals and assumption checking
M1_EMF_Res = M1_EMF_Org(min(M1_Throt)+1:1:max(M1_Throt)+1,:) - ...
    repmat(M1_EMF_Fit_Vals',1,size(M1_EMF_Org,2));

% Motor 2 thrust fit
% y = a*x^2 + b*x + c

M2_Thr_Fit = polyfit(M2_Throt,M2_Thr_Vals,Thrust_Order);
M2_Thr_Fit_Vals = polyval(M2_Thr_Fit,M2_Throt);

% Residuals and assumption checking
M2_Thr_Res = M2_Thr_Org_Vals(min(M2_Throt)+1:1:max(M2_Throt)+1,:) - ...
    repmat(M2_Thr_Fit_Vals',1,size(M2_Thr_Org_Vals,2));

% Motor 2 torque fit
% y = a*x^2 + b*x + c
M2_Tor_Fit = polyfit(M2_Throt,M2_Tor_Vals,Torque_Order);
M2_Tor_Fit_Vals = polyval(M2_Tor_Fit,M2_Throt);

% Residuals and assumption checking
M2_Tor_Res = M2_Tor_Org_Vals(min(M2_Throt)+1:1:max(M2_Throt)+1,:) - ...
    repmat(M2_Tor_Fit_Vals',1,size(M2_Tor_Org_Vals,2));

% Motor 2 EMF fit
% y=a*x+b
M2_EMF_Fit = polyfit(M2_Throt,M2_EMF_Mean,EMF_Order);
M2_EMF_Fit_Vals = polyval(M2_EMF_Fit,M2_Throt);

% Residuals and assumption checking
M2_EMF_Res = M2_EMF_Org(min(M2_Throt)+1:1:max(M2_Throt)+1,:) - ...
    repmat(M2_EMF_Fit_Vals',1,size(M2_EMF_Org,2));

% Motor 3 thrust fit
% y = a*x^2 + b*x + c

```

```
697 M3_Thr_Fit = polyfit(M3_Throt,M3_Thr_Vals,Thrust_Order);
698 M3_Thr_Fit_Vals = polyval(M3_Thr_Fit,M3_Throt);
699
700 % Residuals and assumption checking
701 M3_Thr_Res = M3_Thr_Org_Vals(min(M3_Throt)+1:1:max(M3_Throt)+1,:) - ...
702     repmat(M3_Thr_Fit_Vals',1,size(M3_Thr_Org_Vals,2));
703
704 % Motor 3 torque fit
705 % y = a*x^2 + b*x + c
706 M3_Tor_Fit = polyfit(M3_Throt,M3_Tor_Vals,Torque_Order);
707 M3_Tor_Fit_Vals = polyval(M3_Tor_Fit,M3_Throt);
708
709 % Residuals and assumption checking
710 M3_Tor_Res = M3_Tor_Org_Vals(min(M3_Throt)+1:1:max(M3_Throt)+1,:) - ...
711     repmat(M3_Tor_Fit_Vals',1,size(M3_Tor_Org_Vals,2));
712
713 % Motor 3 EMF fit
714 % y=a*x+b
715 M3_EMF_Fit = polyfit(M3_Throt,M3_EMF_Mean,EMF_Order);
716 M3_EMF_Fit_Vals = polyval(M3_EMF_Fit,M3_Throt);
717
718 % Residuals and assumption checking
719 M3_EMF_Res = M3_EMF_Org(min(M3_Throt)+1:1:max(M3_Throt)+1,:) - ...
720     repmat(M3_EMF_Fit_Vals',1,size(M3_EMF_Org,2));
721
722 % Motor 4 thrust fit
723 % y = a*x^2 + b*x + c
724
725 M4_Thr_Fit = polyfit(M4_Throt,M4_Thr_Vals,Thrust_Order);
726 M4_Thr_Fit_Vals = polyval(M4_Thr_Fit,M4_Throt);
727
728 % Residuals and assumption checking
729 M4_Thr_Res = M4_Thr_Org_Vals(min(M4_Throt)+1:1:max(M4_Throt)+1,:) - ...
730     repmat(M4_Thr_Fit_Vals',1,size(M4_Thr_Org_Vals,2));
731
732 % Motor 4 torque fit
733 % y = a*x^2 + b*x + c
734 M4_Tor_Fit = polyfit(M4_Throt,M4_Tor_Vals,Torque_Order);
735 M4_Tor_Fit_Vals = polyval(M4_Tor_Fit,M4_Throt);
736
737 % Residuals and assumption checking
738 M4_Tor_Res = M4_Tor_Org_Vals(min(M4_Throt)+1:1:max(M4_Throt)+1,:) - ...
739     repmat(M4_Tor_Fit_Vals',1,size(M4_Tor_Org_Vals,2));
740
741 % Motor 4 EMF fit
742 % y=a*x+b
743 M4_EMF_Fit = polyfit(M4_Throt,M4_EMF_Mean,EMF_Order);
744 M4_EMF_Fit_Vals = polyval(M4_EMF_Fit,M4_Throt);
745
```

```
746 % Residuals and assumption checking
747 M4_EMF_Res = M4_EMF_Org(min(M4_Throt)+1:1:max(M4_Throt)+1,:) - ...
748     repmat(M4_EMF_Fit_Vals',1,size(M4_EMF_Org,2));
749
750 % Plot all the fits against the mean values
751 % M1
752 figure
753
754 subplot(3,1,1)
755 plot(Motor_Throttle,M1_Thr_Org_Vals,'.k',M1_Throt,M1_Thr_Fit_Vals);
756 set(gca, 'FontSize', 14)
757 xlim([min(M1_Throt) max(M1_Throt)])
758 xlabel('Throttle, %')
759 ylabel('Thrust, N')
760 subplot(3,1,2)
761 plot(M1_Throt,M1_Thr_Res,'.k',[0,650],[0,0],'r')
762 set(gca, 'FontSize', 14)
763 xlim([min(M1_Throt) max(M1_Throt)])
764 xlabel('Throttle, %')
765 ylabel('Residual, N')
766 subplot(3,1,3)
767 hist(reshape(M1_Thr_Res,1,[]),-2:0.01:2)
768 set(gca, 'FontSize', 14)
769 xlabel('Residual, N')
770 ylabel('# of points')
771
772 figure
773
774 subplot(3,1,1)
775 plot(Motor_Throttle,M1_Tor_Org_Vals,'.k',M1_Throt,M1_Tor_Fit_Vals);
776 set(gca, 'FontSize', 14)
777 xlim([min(M1_Throt) max(M1_Throt)])
778 xlabel('Throttle, %')
779 ylabel('Torque, N\cdotm')
780 subplot(3,1,2)
781 plot(M1_Throt,M1_Tor_Res,'.k',[0,650],[0,0],'r')
782 set(gca, 'FontSize', 14)
783 xlim([min(M1_Throt) max(M1_Throt)])
784 xlabel('Throttle, %')
785 ylabel('Residual, N\cdotm')
786 subplot(3,1,3)
787 hist(reshape(M1_Tor_Res,1,[]),-0.002:0.00001:0.002)
788 set(gca, 'FontSize', 14)
789 xlabel('Residual, N\cdotm')
790 ylabel('# of points')
791
792 figure
793
794 subplot(3,1,1)
```

```
795 plot(Motor_Throttle,M1_EMF_Org,'.k',M1_Throt,M1_EMF_Fit_Vals);
796 set(gca, 'FontSize', 14)
797 xlim([min(M1_Throt) max(M1_Throt)])
798 xlabel('Throttle, %')
799 ylabel('Back EMF, V')
800 subplot(3,1,2)
801 plot(M1_Throt,M1_EMF_Res,'.k',[0,650],[0,0],'r')
802 set(gca, 'FontSize', 14)
803 xlim([min(M1_Throt) max(M1_Throt)])
804 xlabel('Throttle, %')
805 ylabel('Residual, V')
806 subplot(3,1,3)
807 hist(reshape(M1_EMF_Res,1,[]),-0.02:0.001:0.02)
808 set(gca, 'FontSize', 14)
809 xlabel('Residual, V')
810 ylabel('# of points')
811
812
813 % M2
814 figure
815
816 subplot(3,1,1)
817 plot(Motor_Throttle,M2_Thr_Org_Vals,'.k',M2_Throt,M2_Thr_Fit_Vals);
818 set(gca, 'FontSize', 14)
819 xlim([min(M2_Throt) max(M2_Throt)])
820 xlabel('Throttle, %')
821 ylabel('Thrust, N')
822 subplot(3,1,2)
823 plot(M2_Throt,M2_Thr_Res,'.k',[0,650],[0,0],'r')
824 set(gca, 'FontSize', 14)
825 xlim([min(M2_Throt) max(M2_Throt)])
826 xlabel('Throttle, %')
827 ylabel('Residual, N')
828 subplot(3,1,3)
829 hist(reshape(M2_Thr_Res,1,[]),-2:0.01:2)
830 set(gca, 'FontSize', 14)
831 xlabel('Residual, N')
832 ylabel('# of points')
833
834 figure
835
836 subplot(3,1,1)
837 plot(Motor_Throttle,M2_Tor_Org_Vals,'.k',M2_Throt,M2_Tor_Fit_Vals);
838 set(gca, 'FontSize', 14)
839 xlim([min(M2_Throt) max(M2_Throt)])
840 xlabel('Throttle, %')
841 ylabel('Torque, N\cdotm')
842 subplot(3,1,2)
843 plot(M2_Throt,M2_Tor_Res,'.k',[0,650],[0,0],'r')
```

```
844 set(gca, 'FontSize', 14)
845 xlim([min(M2_Throt) max(M2_Throt)])
846 xlabel('Throttle, %')
847 ylabel('Residual, N\cdotm')
848 subplot(3,1,3)
849 hist(reshape(M2_Tor_Res,1,[]),-0.002:0.00001:0.002)
850 set(gca, 'FontSize', 14)
851 xlabel('Residual, N\cdotm')
852 ylabel('# of points')
853
854 figure
855
856 subplot(3,1,1)
857 plot(Motor_Throttle,M2_EMF_Org,'.k',M2_Throt,M2_EMF_Fit_Vals);
858 set(gca, 'FontSize', 14)
859 xlim([min(M2_Throt) max(M2_Throt)])
860 xlabel('Throttle, %')
861 ylabel('Back EMF, V')
862 subplot(3,1,2)
863 plot(M2_Throt,M2_EMF_Res,'.k',[0,650],[0,0],'r')
864 set(gca, 'FontSize', 14)
865 xlim([min(M2_Throt) max(M2_Throt)])
866 xlabel('Throttle, %')
867 ylabel('Residual, V')
868 subplot(3,1,3)
869 hist(reshape(M2_EMF_Res,1,[]),-0.02:0.001:0.02)
870 set(gca, 'FontSize', 14)
871 xlabel('Residual, V')
872 ylabel('# of points')
873
874 % M3
875 figure
876
877 subplot(3,1,1)
878 plot(Motor_Throttle,M3_Thr_Org_Vals,'.k',M3_Throt,M3_Thr_Fit_Vals);
879 set(gca, 'FontSize', 14)
880 xlim([min(M3_Throt) max(M3_Throt)])
881 xlabel('Throttle, %')
882 ylabel('Thrust, N')
883 subplot(3,1,2)
884 plot(M3_Throt,M3_Thr_Res,'.k',[0,650],[0,0],'r')
885 set(gca, 'FontSize', 14)
886 xlim([min(M3_Throt) max(M3_Throt)])
887 xlabel('Throttle, %')
888 ylabel('Residual, N')
889 subplot(3,1,3)
890 hist(reshape(M3_Thr_Res,1,[]),-2:0.01:2)
891 set(gca, 'FontSize', 14)
892 xlabel('Residual, N')
```

```
893 ylabel('# of points')
894
895 figure
896
897 subplot(3,1,1)
898 plot(Motor_Throttle,M3_Tor_Org_Vals,'.k',M3_Throt,M3_Tor_Fit_Vals);
899 set(gca, 'FontSize', 14)
900 xlim([min(M3_Throt) max(M3_Throt)])
901 xlabel('Throttle, %')
902 ylabel('Torque, N\cdotm')
903 subplot(3,1,2)
904 plot(M3_Throt,M3_Tor_Res,'.k',[0,650],[0,0],'r')
905 set(gca, 'FontSize', 14)
906 xlim([min(M3_Throt) max(M3_Throt)])
907 xlabel('Throttle, %')
908 ylabel('Residual, N\cdotm')
909 subplot(3,1,3)
910 hist(reshape(M3_Tor_Res,1,[]),-0.002:0.00001:0.002)
911 set(gca, 'FontSize', 14)
912 xlabel('Residual, N\cdotm')
913 ylabel('# of points')
914
915 figure
916
917 subplot(3,1,1)
918 plot(Motor_Throttle,M3_EMF_Org,'.k',M3_Throt,M3_EMF_Fit_Vals);
919 set(gca, 'FontSize', 14)
920 xlim([min(M3_Throt) max(M3_Throt)])
921 xlabel('Throttle, %')
922 ylabel('Back EMF, V')
923 subplot(3,1,2)
924 plot(M3_Throt,M3_EMF_Res,'.k',[0,650],[0,0],'r')
925 set(gca, 'FontSize', 14)
926 xlim([min(M3_Throt) max(M3_Throt)])
927 xlabel('Throttle, %')
928 ylabel('Residual, V')
929 subplot(3,1,3)
930 hist(reshape(M3_EMF_Res,1,[]),-0.02:0.001:0.02)
931 set(gca, 'FontSize', 14)
932 xlabel('Residual, V')
933 ylabel('# of points')
934
935 % M4
936 figure
937
938 subplot(3,1,1)
939 plot(Motor_Throttle,M4_Thr_Org_Vals,'.k',M4_Throt,M4_Thr_Fit_Vals);
940 set(gca, 'FontSize', 14)
941 xlim([min(M4_Throt) max(M4_Throt)])
```

```
xlabel('Throttle, %')
ylabel('Thrust, N')
subplot(3,1,2)
plot(M4_Throt,M4_Thr_Res,'.k',[0,650],[0,0],'r')
set(gca, 'FontSize', 14)
xlim([min(M4_Throt) max(M4_Throt)])
xlabel('Throttle, %')
ylabel('Residual, N')
subplot(3,1,3)
hist(reshape(M4_Thr_Res,1,[]),-2:0.01:2)
set(gca, 'FontSize', 14)
xlabel('Residual, N')
ylabel('# of points')

figure

subplot(3,1,1)
plot(Motor_Throttle,M4_Tor_Org_Vals,'.k',M4_Throt,M4_Tor_Fit_Vals);
set(gca, 'FontSize', 14)
xlim([min(M4_Throt) max(M4_Throt)])
xlabel('Throttle, %')
ylabel('Torque, N\cdotm')
subplot(3,1,2)
plot(M4_Throt,M4_Tor_Res,'.k',[0,650],[0,0],'r')
set(gca, 'FontSize', 14)
xlim([min(M4_Throt) max(M4_Throt)])
xlabel('Throttle, %')
ylabel('Residual, N\cdotm')
subplot(3,1,3)
hist(reshape(M4_Tor_Res,1,[]),-0.002:0.00001:0.002)
set(gca, 'FontSize', 14)
xlabel('Residual, N\cdotm')
ylabel('# of points')

figure

subplot(3,1,1)
plot(Motor_Throttle,M4_EMF_Org,'.k',M4_Throt,M4_EMF_Fit_Vals);
set(gca, 'FontSize', 14)
xlim([min(M4_Throt) max(M4_Throt)])
xlabel('Throttle, %')
ylabel('Back EMF, V')
subplot(3,1,2)
plot(M4_Throt,M4_EMF_Res,'.k',[0,650],[0,0],'r')
set(gca, 'FontSize', 14)
xlim([min(M4_Throt) max(M4_Throt)])
xlabel('Throttle, %')
ylabel('Residual, V')
subplot(3,1,3)
```

```
hist(reshape(M4_EMF_Res,1,[]),-0.02:0.001:0.02)
set(gca, 'FontSize', 14)
xlabel('Residual, V')
ylabel('# of points')

% compare the fits of each motor agains each other
figure
subplot(2,1,1)
plot(M1_Throt-M1_Throt(1),M1_Thr_Fit_Vals,'r',M2_Throt-M2_Throt(1),...
    M2_Thr_Fit_Vals,'g',M3_Throt-M3_Throt(1),M3_Thr_Fit_Vals,'b',...
    M4_Throt-M4_Throt(1),M4_Thr_Fit_Vals,'m');
set(gca, 'FontSize', 14)
legend('location','SouthEast','Motor 1','Motor 2','Motor 3','Motor 4')
ylabel('Thrust, N')

subplot(2,1,2)

plot(M1_Throt-M1_Throt(1),M1_Tor_Fit_Vals,'r',M2_Throt-M2_Throt(1),...
    M2_Tor_Fit_Vals,'g',M3_Throt-M3_Throt(1),M3_Tor_Fit_Vals,'b',...
    M4_Throt-M4_Throt(1),M4_Tor_Fit_Vals,'m');
set(gca, 'FontSize', 14)
legend('location','SouthEast','Motor 1','Motor 2','Motor 3','Motor 4')
xlabel('Offset Throttle, +1%')
ylabel('Torque, N\cdotm')

%%%%% Generate .dat file

dat_file = fopen('Motor_Cal.dat','w');

fprintf(dat_file,'M1 Range\n');
fprintf(dat_file,'%d\n',min(M1_Throt));
fprintf(dat_file,'%d\n',max(M1_Throt));
fprintf(dat_file,'\n');
fprintf(dat_file,'M1 Thrust\n');
fprintf(dat_file,'%d\n',M1_Thr_Fit(1));
fprintf(dat_file,'%d\n',M1_Thr_Fit(2));
fprintf(dat_file,'%d\n',M1_Thr_Fit(3));
fprintf(dat_file,'\n');
fprintf(dat_file,'M1 Torque\n');
fprintf(dat_file,'%d\n',M1_Tor_Fit(1));
fprintf(dat_file,'%d\n',M1_Tor_Fit(2));
fprintf(dat_file,'%d\n',M1_Tor_Fit(3));
fprintf(dat_file,'\n');
fprintf(dat_file,'M1 EMF\n');
fprintf(dat_file,'%d\n',M1_EMF_Fit(1));
fprintf(dat_file,'%d\n',M1_EMF_Fit(2));
fprintf(dat_file,'%d\n',M1_EMF_Fit(3));
fprintf(dat_file,'%d\n',M1_EMF_Fit(4));
fprintf(dat_file,'\n');
```

```
1040
1041 fprintf(dat_file,'M2 Range\n');
1042 fprintf(dat_file,'%d\n',min(M2_Throt));
1043 fprintf(dat_file,'%d\n',max(M2_Throt));
1044 fprintf(dat_file,'\n');
1045 fprintf(dat_file,'M2 Thrust\n');
1046 fprintf(dat_file,'%d\n',M2_Thr_Fit(1));
1047 fprintf(dat_file,'%d\n',M2_Thr_Fit(2));
1048 fprintf(dat_file,'%d\n',M2_Thr_Fit(3));
1049 fprintf(dat_file,'\n');
1050 fprintf(dat_file,'M2 Torque\n');
1051 fprintf(dat_file,'%d\n',M2_Tor_Fit(1));
1052 fprintf(dat_file,'%d\n',M2_Tor_Fit(2));
1053 fprintf(dat_file,'%d\n',M2_Tor_Fit(3));
1054 fprintf(dat_file,'\n');
1055 fprintf(dat_file,'M2 EMF\n');
1056 fprintf(dat_file,'%d\n',M2_EMF_Fit(1));
1057 fprintf(dat_file,'%d\n',M2_EMF_Fit(2));
1058 fprintf(dat_file,'%d\n',M2_EMF_Fit(3));
1059 fprintf(dat_file,'%d\n',M2_EMF_Fit(4));
1060 fprintf(dat_file,'\n');
1061
1062 fprintf(dat_file,'M3 Range\n');
1063 fprintf(dat_file,'%d\n',min(M3_Throt));
1064 fprintf(dat_file,'%d\n',max(M3_Throt));
1065 fprintf(dat_file,'\n');
1066 fprintf(dat_file,'M3 Thrust\n');
1067 fprintf(dat_file,'%d\n',M3_Thr_Fit(1));
1068 fprintf(dat_file,'%d\n',M3_Thr_Fit(2));
1069 fprintf(dat_file,'%d\n',M3_Thr_Fit(3));
1070 fprintf(dat_file,'\n');
1071 fprintf(dat_file,'M3 Torque\n');
1072 fprintf(dat_file,'%d\n',M3_Tor_Fit(1));
1073 fprintf(dat_file,'%d\n',M3_Tor_Fit(2));
1074 fprintf(dat_file,'%d\n',M3_Tor_Fit(3));
1075 fprintf(dat_file,'\n');
1076 fprintf(dat_file,'M3 EMF\n');
1077 fprintf(dat_file,'%d\n',M3_EMF_Fit(1));
1078 fprintf(dat_file,'%d\n',M3_EMF_Fit(2));
1079 fprintf(dat_file,'%d\n',M3_EMF_Fit(3));
1080 fprintf(dat_file,'%d\n',M3_EMF_Fit(4));
1081 fprintf(dat_file,'\n');
1082
1083 fprintf(dat_file,'M4 Range\n');
1084 fprintf(dat_file,'%d\n',min(M4_Throt));
1085 fprintf(dat_file,'%d\n',max(M4_Throt));
1086 fprintf(dat_file,'\n');
1087 fprintf(dat_file,'M4 Thrust\n');
1088 fprintf(dat_file,'%d\n',M4_Thr_Fit(1));
```

```
1089 fprintf(dat_file,'%d\n',M4_Thr_Fit(2));
1090 fprintf(dat_file,'%d\n',M4_Thr_Fit(3));
1091 fprintf(dat_file,'\n');
1092 fprintf(dat_file,'M4 Torque\n');
1093 fprintf(dat_file,'%d\n',M4_Tor_Fit(1));
1094 fprintf(dat_file,'%d\n',M4_Tor_Fit(2));
1095 fprintf(dat_file,'%d\n',M4_Tor_Fit(3));
1096 fprintf(dat_file,'\n');
1097 fprintf(dat_file,'M4 EMF\n');
1098 fprintf(dat_file,'%d\n',M4_EMF_Fit(1));
1099 fprintf(dat_file,'%d\n',M4_EMF_Fit(2));
1100 fprintf(dat_file,'%d\n',M4_EMF_Fit(3));
1101 fprintf(dat_file,'%d\n',M4_EMF_Fit(4));
1102 fprintf(dat_file,'\n');
1103
1104
1105 fclose(dat_file);
1106
1107 %%% Save all the plots
1108
1109 h = get(0,'children');
1110
1111 for i=1:length(h)
1112
1113     savename =
     ↪ strcat('../../Thesis/Figures/Motor_Curve_Figure_',num2str(length(h)+1-i));
1114     set(h(i),'color','w', 'Position', [100, 100, 840, 630]);
1115     img = getframe(h(i));
1116     imwrite(img.cdata, [savename, '.png']);
1117     set(h(i),'color',[0.8 0.8 0.8]);
1118
1119 %     saveas(h(i),savename,'epsc')
1120
1121 end
```

Appendix I

Simulation Code

I.1 Main Simulation Code

This is the main code which simulates the flight of the quadcopter.

```
% Simulation
% Created by Peter Olejnik
%
% Purpose:  This is the code which simulates the quadcopter flight.
%

clc
clear all
close all

% Define variables

[M1_Traits,M2_Traits,M3_Traits,M4_Traits] = Dat_File_Reader();

% QC Traits

M1_Range = M1_Traits(1,1:2);
M1_Thrust = M1_Traits(2,1:3);
M1_Torque = M1_Traits(3,1:3);
M1_EMF = M1_Traits(4,:);

M2_Range = M2_Traits(1,1:2);
M2_Thrust = M2_Traits(2,1:3);
M2_Torque = M2_Traits(3,1:3);
M2_EMF = M2_Traits(4,:);

M3_Range = M3_Traits(1,1:2);
M3_Thrust = M3_Traits(2,1:3);
```

```
29 M3_Torque = M3_Traits(3,1:3);
30 M3_EMF = M3_Traits(4,:);
31
32 M4_Range = M4_Traits(1,1:2);
33 M4_Thrust = M4_Traits(2,1:3);
34 M4_Torque = M4_Traits(3,1:3);
35 M4_EMF = M4_Traits(4,:);
36
37 Ixx = 6.9351e-03;
38 Iyy = 6.9351e-03;
39 Izz = 1.1474e-01;
40
41 mQc = 479.95/1000;
42 g = 9.81;
43 d = 0.1525;
44
45 s2d2 = sqrt(2)*d/2;
46
47 % Timing
48
49 time = 0;
50 dt = 0.01;
51 tf = 80;
52
53 % Sensor Varianceu
54
55 AX_Var = 4.9609e-03;
56 AY_Var = 2.3615e-03;
57 AZ_Var = 1.7702e-03;
58
59 GX_Var = 3.3864e-06;
60 GY_Var = 1.0749e-04;
61 GZ_Var = 2.1762e-06;
62
63 % Theoretical Traits
64
65 Theoretical_A = [0;0;0;0;0;0];
66 Theoretical_V = [0;0;0;0;0;0];
67 Theoretical_P = [0;0;0;0;0;0];
68
69 Sens_A = Theoretical_A;
70 Sens_V = Theoretical_V;
71 Sens_P = Theoretical_P;
72
73 Kal_A = [0;0;0;0;0;0];
74 Kal_V = [0;0;0;0;0;0];
75 Kal_P = [0;0;0;0;0;0];
76
77 Kal_A_States = [0;0;0;0;0;0];
```

```
78  Kal_V_States = [0;0;0;0;0;0];
79  Kal_P_States = [0;0;0;0;0;0];
80
81  % LQR Controller and Kalman Filter Matricies
82
83      % A matricies
84
85  A_lqr = zeros(18);
86  A_lqr(1:3,1:3) = [1,dt,0;
87                    0,1,dt;
88                    0,0,0];
89  A_lqr(3,10) = -g;
90  A_lqr(4:6,4:6) = [1,dt,0;
91                    0,1,dt;
92                    0,0,0];
93  A_lqr(6,13) = g;
94  A_lqr(7:9,7:9) = [1,dt,0;
95                    0,1,dt;
96                    0,0,0];
97  A_lqr(10:12,10:12) = [1,dt,0;
98                        0,1,dt;
99                        0,0,0];
100 A_lqr(13:15,13:15) = [1,dt,0;
101                       0,1,dt;
102                       0,0,0];
103 A_lqr(16:18,16:18) = [1,dt,0;
104                       0,1,dt;
105                       0,0,0];
106
107 A_Jac = zeros(18);
108 A_Jac(1:3,1:3) = [1,dt,0;
109                   0,1,dt;
110                   0,0,0];
111 A_Jac(4:6,4:6) = [1,dt,0;
112                   0,1,dt;
113                   0,0,0];
114 A_Jac(7:9,7:9) = [1,dt,0;
115                   0,1,dt;
116                   0,0,0];
117 A_Jac(10:12,10:12) = [1,dt,0;
118                       0,1,dt;
119                       0,0,0];
120 A_Jac(13:15,13:15) = [1,dt,0;
121                       0,1,dt;
122                       0,0,0];
123 A_Jac(16:18,16:18) = [1,dt,0;
124                       0,1,dt;
125                       0,0,0];
126
```

```
127     % B matricies
128 
129 B_lqr = zeros(18,8);
130 B_lqr(9,1:4) = -1/mQc;
131 B_lqr(12,1:2) = s2d2/Ixx;
132 B_lqr(12,3:4) = -s2d2/Ixx;
133 B_lqr(15,2:3) = s2d2/Iyy;
134 B_lqr(15,1:3:4) = -s2d2/Iyy;
135 B_lqr(18,5:8) = 1/Izz;
136 
137     % C matricies
138 
139 C_kf = zeros(6,18);
140 C_kf(1,3) = 1;
141 C_kf(2,6) = 1;
142 C_kf(3,9) = 1;
143 C_kf(4,11) = 1;
144 C_kf(5,14) = 1;
145 C_kf(6,17) = 1;
146 
147     % Q matricies
148 
149 Q_lqr = eye(18);
150 Q_lqr(1,1) = 1000;
151 Q_lqr(4,4) = 1000;
152 Q_lqr(7,7) = 1000;
153 Q_lqr(10,10) = 1000;
154 Q_lqr(13,13) = 1000;
155 Q_lqr(16,16) = 1000;
156 
157 Q_kf = eye(18)*1e3;
158 
159     % R matricies
160 
161 R_lqr = eye(8)*3e3;
162 
163 R_kf = eye(6)*1e3;
164 
165     % P matricies
166 
167 P_lqr = dare(A_lqr,B_lqr,Q_lqr,R_lqr);
168 
169 P_kf = zeros(18);
170 
171 
172     % K matricies
173 
174 K_lqr = -inv(R_lqr + B_lqr'*P_lqr*B_lqr)*B_lqr'*P_lqr*A_lqr;
175 
```

```
176 % Other constants
177
178 Base_Force = [mQc*g/4;mQc*g/4;mQc*g/4;mQc*g/4];
179
180 Net_F = Base_Force;
181
182 Throttle = Force2Throttle(Net_F,M1_Traits,M2_Traits,M3_Traits,M4_Traits)';
183
184 ThrottleUsed = Throttle;
185
186 BaseThrottle = Throttle;
187
188 Inputs = zeros(8,1);
189
190 States = zeros(18,1);
191
192 time_sim = 0;
193
194 Tar_Error = zeros(18,1);
195
196 Target_Points = [0;0;0;    % X
197                  0;0;0;    % Y
198                  0;0;0;    % Z
199                  0;0;0;    % Phi
200                  0;0;0;    % Theta
201                  0;0;0];   % Phi
202
203 KF_Filter_Corrections = States;
204
205 while (time < tf)
206
207     if any(any(isnan(States)))
208         fprintf('Nan condition hit. Terminating.')
209         error
210     end
211
212 % Targets/Error
213
214     if ((time < 5) && (time >= -1))
215         Targets = [0;0;0;    % X
216                    0;0;0;    % Y
217                    0;0;0;    % Z
218                    0;0;0;    % Phi
219                    0;0;0;    % Theta
220                    0;0;0];   % Phi
221     elseif ((time < 15) && (time >= 5))
222         Targets = [1;0;0;    % X
223                    0;0;0;    % Y
224                    -1;0;0;    % Z
```

```
225                     0;0;0;    % Phi
226                     0;0;0;    % Theta
227                     0;0;0];   % Phi
228     elseif ((time < 25) && (time >= 15))
229         Targets = [1;0;0;     % X
230                    1;0;0;     % Y
231                    -1;0;0;    % Z
232                    0;0;0;     % Phi
233                    0;0;0;     % Theta
234                    0;0;0];    % Phi
235     elseif ((time < 35) && (time >= 25))
236         Targets = [0;0;0;     % X
237                    1;0;0;     % Y
238                    0;0;0;     % Z
239                    0;0;0;     % Phi
240                    0;0;0;     % Theta
241                    0;0;0];    % Phi
242     elseif ((time < 45) && (time >= 35))
243         Targets = [-1;0;0;    % X
244                    0;0;0;     % Y
245                    0;0;0;     % Z
246                    0;0;0;     % Phi
247                    0;0;0;     % Theta
248                    0;0;0];    % Phi
249     elseif ((time < 55) && (time >= 45))
250         Targets = [-1;0;0;    % X
251                    -1;0;0;    % Y
252                    1;0;0;     % Z
253                    0;0;0;     % Phi
254                    0;0;0;     % Theta
255                    0;0;0];    % Phi
256     elseif ((time < 65) && (time >= 55))
257         Targets = [0;0;0;     % X
258                    -1;0;0;    % Y
259                    1;0;0;     % Z
260                    0;0;0;     % Phi
261                    0;0;0;     % Theta
262                    0;0;0];    % Phi
263     else
264         Targets = [0;0;0;     % X
265                    0;0;0;     % Y
266                    0;0;0;     % Z
267                    0;0;0;     % Phi
268                    0;0;0;     % Theta
269                    0;0;0];    % Phi
270     end
271
272 % Theoretical flight calculation
273
```

```
    Theoretical_A(:,end+1) = Theoretical_Model(Throttle,...
    M1_Traits,M2_Traits,M3_Traits,M4_Traits,...
    Theoretical_P(4,end),Theoretical_P(5,end),...
    Theoretical_V(4,end),Theoretical_V(5,end),Theoretical_V(6,end),...
    Ixx,Iyy,Izz,mQc,g,d);

    Theoretical_V(:,end+1) = Theoretical_V(:,end) + Theoretical_A(:,end)*dt;
    Theoretical_P(:,end+1) = Theoretical_P(:,end) + Theoretical_V(:,end)*dt;

% Simulated Sensor Reads

    Sensor_Out = Noiseafier([Theoretical_A(1:3,end);Theoretical_V(4:6,end)],...
        sqrt([AX_Var;AY_Var;AZ_Var;GX_Var;GY_Var;GZ_Var]));

    Sens_V(:,end+1) = [Sens_V(1,end)+Sens_A(1,end)*dt;
                       Sens_V(2,end)+Sens_A(2,end)*dt;
                       Sens_V(3,end)+Sens_A(3,end)*dt;
                       Sensor_Out(4);Sensor_Out(5);Sensor_Out(6)];
    Sens_A(:,end+1) = [Sensor_Out(1);Sensor_Out(2);Sensor_Out(3);
                       (Sens_V(4:6,end)-Sens_V(4:6,end-1))/dt];
    Sens_P(:,end+1) = Sens_P(:,end) + Sens_V(:,end)*dt;

    Sensor_Read = [Sens_A(1,end);
                   Sens_A(2,end);
                   Sens_A(3,end);
                   Sens_V(4,end);
                   Sens_V(5,end);
                   Sens_V(6,end)];

% Kalman Filter

    Residual = Sensor_Read - C_kf*States(:,end);

    Kal_A_States = Theoretical_Model(Throttle,...
                   M1_Traits,M2_Traits,M3_Traits,M4_Traits,...
                   Kal_P_States(4,end),Kal_P_States(5,end),...
                   Kal_V_States(4,end),Kal_V_States(5,end),Kal_V_States(6,end),...
                   Ixx,Iyy,Izz,mQc,g,d);

    Kal_V_States = Kal_V_States + Kal_A_States*dt;
    Kal_P_States = Kal_P_States + Kal_V_States*dt;

    States(:,end+1) = [Kal_P_States(1);
                       Kal_V_States(1);
                       Kal_A_States(1);
                       Kal_P_States(2);
                       Kal_V_States(2);
                       Kal_A_States(2);
                       Kal_P_States(3);
```

```
                        Kal_V_States(3);
                        Kal_A_States(3);
                        Kal_P_States(4);
                        Kal_V_States(4);
                        Kal_A_States(4);
                        Kal_P_States(5);
                        Kal_V_States(5);
                        Kal_A_States(5);
                        Kal_P_States(6);
                        Kal_V_States(6);
                        Kal_A_States(6)];

    A_Jac(3,10) = cos(Kal_P(4,end))*(-Net_F(1)-Net_F(2)-Net_F(3)-Net_F(4))/mQc;
    A_Jac(6,10) =
↪   sin(Kal_P(4,end))*sin(Kal_P(5,end))*(-Net_F(1)-Net_F(2)-Net_F(3)-Net_F(4))/mQc;
    A_Jac(6,13) =
↪   -cos(Kal_P(4,end))*cos(Kal_P(5,end))*(-Net_F(1)-Net_F(2)-Net_F(3)-Net_F(4))/mQc;
    A_Jac(9,10) =
↪   -sin(Kal_P(4,end))*cos(Kal_P(5,end))*(-Net_F(1)-Net_F(2)-Net_F(3)-Net_F(4))/mQc;
    A_Jac(9,13) =
↪   -cos(Kal_P(4,end))*sin(Kal_P(5,end))*(-Net_F(1)-Net_F(2)-Net_F(3)-Net_F(4))/mQc;
    A_Jac(12,14) = Kal_V(6,end)*(Izz-Iyy)/Ixx;
    A_Jac(12,17) = Kal_V(5,end)*(Izz-Iyy)/Ixx;
    A_Jac(15,11) = Kal_V(6,end)*(Ixx-Izz)/Iyy;
    A_Jac(15,17) = Kal_V(4,end)*(Ixx-Izz)/Iyy;
    A_Jac(18,11) = Kal_V(5,end)*(Iyy-Ixx)/Izz;
    A_Jac(18,14) = Kal_V(4,end)*(Iyy-Ixx)/Izz;

    P_kf = A_Jac*P_kf + P_kf*A_Jac' + Q_kf;
    K_kf = P_kf*C_kf'*inv(C_kf*P_kf*C_kf' + R_kf);
    P_kf = (eye(18) - K_kf*C_kf*P_kf);

    KF_Filter_Corrections(:,end+1) = K_kf*Residual;

    States(:,end) = States(:,end) + KF_Filter_Corrections(:,end);

    Kal_A(:,end+1) = [States(3,end);
                      States(6,end);
                      States(9,end);
                      States(12,end);
                      States(15,end);
                      States(18,end)];

    Kal_V(:,end+1) = [States(2,end);
                      States(5,end);
                      States(8,end);
                      States(11,end);
                      States(14,end);
                      States(17,end)];
```

```
368
369     Kal_P(:,end+1) = [States(1,end);
370                       States(4,end);
371                       States(7,end);
372                       States(10,end);
373                       States(13,end);
374                       States(16,end)];
375
376 % Optimal Controller
377
378     Tar_Error(:,end+1) = States(:,end) - Targets;
379
380     Inputs(:,end+1) = K_lqr*Tar_Error(:,end);
381
382 % Force Combiner
383
384     PT_Force = Inputs(1:4,end) + Base_Force;
385
386     Translated_Torques =
    ↪ Force2Torque(PT_Force,M1_Traits,M2_Traits,M3_Traits,M4_Traits)';
387
388     Net_Torques = Translated_Torques + Inputs(5:8,end);
389
390     Net_F = Torque2Force(Net_Torques,M1_Traits,M2_Traits,M3_Traits,M4_Traits)';
391
392     Throttle = Force2Throttle(Net_F,M1_Traits,M2_Traits,M3_Traits,M4_Traits)';
393
394     ThrottleUsed(:,end+1) = Throttle;
395
396     time = time + dt;
397
398     time_sim(end+1) = time;
399
400     Target_Points(:,end+1) = Targets;
401
402 end
403
404 %%%%% Save Data
405
406 M_Range = [M1_Range;M2_Range;M3_Range;M4_Range];
407 M_Thrust = [M1_Thrust;M2_Thrust;M3_Thrust;M4_Thrust];
408 M_Torque = [M1_Torque;M2_Torque;M3_Torque;M4_Torque];
409 M_EMF = [M1_EMF;M2_EMF;M3_EMF;M4_EMF];
410
411 save -ascii ./Properties/Motor_Range.txt M_Range
412 save -ascii ./Properties/Motor_Thrust.txt M_Thrust
413 save -ascii ./Properties/Motor_Torque.txt M_Torque
414 save -ascii ./Properties/Motor_EMF.txt M_EMF
415
```

```
save -ascii ./Properties/Q_kf.txt Q_kf
save -ascii ./Properties/R_kf.txt R_kf
save -ascii ./Properties/K_Lqr.txt K_lqr

%%% The figures

% Plots the throttles
figure
subplot(2,1,1)
plot(time_sim,ThrottleUsed(1,:),'r',time_sim,ThrottleUsed(2,:),'b',...
     time_sim,ThrottleUsed(3,:),'k',time_sim,ThrottleUsed(4,:),'g')
set(gca, 'FontSize', 14)
ylabel('Throttle (%)','fontsize',14)
axis([0,tf,-10,110])
subplot(2,1,2)
plot(time_sim,ThrottleUsed(1,:)-repmat(BaseThrottle(1,:),1,length(ThrottleUsed)),'r',...
 ↪  time_sim,ThrottleUsed(2,:)-repmat(BaseThrottle(2,:),1,length(ThrottleUsed)),'b',...
 ↪  time_sim,ThrottleUsed(3,:)-repmat(BaseThrottle(3,:),1,length(ThrottleUsed)),'k',...
     time_sim,ThrottleUsed(4,:)-repmat(BaseThrottle(4,:),1,length(ThrottleUsed)),'g')
set(gca, 'FontSize', 14)
xlabel('Simulation Time (s)','fontsize',14)
ylabel('Diffrence from Base Throttle (%)','fontsize',14)
xlim([0,tf])

% Plots the outputs
figure
subplot(2,1,1)
plot(time_sim,Inputs(1:4,:))
set(gca, 'FontSize', 14)
ylabel('Input Force (N)','fontsize',14)
xlim([0,tf])
subplot(2,1,2)
plot(time_sim,Inputs(5:8,:))
set(gca, 'FontSize', 14)
xlabel('Simulation Time (s)','fontsize',14)
ylabel('Input Torque (Nm)','fontsize',14)
xlim([0,tf])

% Plots the linear accelerations
figure
subplot(3,1,1)
plot(time_sim,Sens_A(1,:),'r',...
     time_sim,Theoretical_A(1,:),'.b',...
     time_sim,Kal_A(1,:),'g',...
     time_sim,Target_Points(3,:),'m');
set(gca, 'FontSize', 14)
ylabel('X-axis (m/s^2)','fontsize',14)
```

```
463 xlim([0,tf])
464 subplot(3,1,2)
465 plot(time_sim,Sens_A(2,:),'r',...
466      time_sim,Theoretical_A(2,:),'.b',...
467      time_sim,Kal_A(2,:),'g',...
468      time_sim,Target_Points(6,:),'m');
469 set(gca, 'FontSize', 14)
470 ylabel('Y-axis (m/s^2)','fontsize',14)
471 xlim([0,tf])
472 subplot(3,1,3)
473 plot(time_sim,Sens_A(3,:),'r',...
474      time_sim,Theoretical_A(3,:),'.b',...
475      time_sim,Kal_A(3,:),'g',...
476      time_sim,Target_Points(9,:),'m');
477 set(gca, 'FontSize', 14)
478 xlabel('Simulation Time (s)','fontsize',14)
479 ylabel('Z-axis (m/s^2)','fontsize',14)
480 xlim([0,tf])
481
482 % Plots the angular accelerations
483 figure
484 subplot(3,1,1)
485 plot(time_sim,Theoretical_A(4,:),'.b',...
486      time_sim,Kal_A(4,:),'g',...
487      time_sim,Target_Points(12,:),'m');
488 set(gca, 'FontSize', 14)
489 ylabel('X-axis (rad/s^2)','fontsize',14)
490 xlim([0,tf])
491 subplot(3,1,2)
492 plot(time_sim,Theoretical_A(5,:),'.b',...
493      time_sim,Kal_A(5,:),'g',...
494      time_sim,Target_Points(15,:),'m');
495 set(gca, 'FontSize', 14)
496 ylabel('Y-axis (rad/s^2)','fontsize',14)
497 xlim([0,tf])
498 subplot(3,1,3)
499 plot(time_sim,Theoretical_A(6,:),'.b',...
500      time_sim,Kal_A(6,:),'g',...
501      time_sim,Target_Points(18,:),'m');
502 set(gca, 'FontSize', 14)
503 xlabel('Simulation Time (s)','fontsize',14)
504 ylabel('Z-axis (rad/s^2)','fontsize',14)
505 xlim([0,tf])
506
507 % Plots the linear velocities
508 figure
509 subplot(3,1,1)
510 plot(time_sim,Theoretical_V(1,:),'.b',...
511      time_sim,Kal_V(1,:),'g',...
```

```
512       time_sim,Target_Points(2,:),'m');
513  set(gca, 'FontSize', 14)
514  ylabel('X-axis (m/s)','fontsize',14)
515  xlim([0,tf])
516  subplot(3,1,2)
517  plot(time_sim,Theoretical_V(2,:),'.b',...
518       time_sim,Kal_V(2,:),'g',...
519       time_sim,Target_Points(5,:),'m');
520  set(gca, 'FontSize', 14)
521  ylabel('Y-axis (m/s)','fontsize',14)
522  xlim([0,tf])
523  subplot(3,1,3)
524  plot(time_sim,Theoretical_V(3,:),'.b',...
525       time_sim,Kal_V(3,:),'g',...
526       time_sim,Target_Points(8,:),'m');
527  set(gca, 'FontSize', 14)
528  xlabel('Simulation Time (s)','fontsize',14)
529  ylabel('Z-axis (m/s)','fontsize',14)
530  xlim([0,tf])
531
532  % Plots the angular velocities
533  figure
534  subplot(3,1,1)
535  plot(time_sim,Sens_V(4,:),'r',...
536       time_sim,Theoretical_V(4,:),'.b',...
537       time_sim,Kal_V(4,:),'g',...
538       time_sim,Target_Points(11,:),'m');
539  set(gca, 'FontSize', 14)
540  ylabel('X-axis (rad/s)','fontsize',14)
541  xlim([0,tf])
542  subplot(3,1,2)
543  plot(time_sim,Sens_V(5,:),'r',...
544       time_sim,Theoretical_V(5,:),'.b',...
545       time_sim,Kal_V(5,:),'g',...
546       time_sim,Target_Points(14,:),'m');
547  set(gca, 'FontSize', 14)
548  ylabel('Y-axis (rad/s)','fontsize',14)
549  xlim([0,tf])
550  subplot(3,1,3)
551  plot(time_sim,Sens_V(6,:),'r',...
552       time_sim,Theoretical_V(6,:),'.b',...
553       time_sim,Kal_V(6,:),'g',...
554       time_sim,Target_Points(17,:),'m');
555  set(gca, 'FontSize', 14)
556  xlabel('Simulation Time (s)','fontsize',14)
557  ylabel('Z-axis (rad/s)','fontsize',14)
558  xlim([0,tf])
559
560  % Plots the linear positions
```

```
figure
subplot(3,1,1)
plot(time_sim,Theoretical_P(1,:),'.b',...
     time_sim,Kal_P(1,:),'g',...
     time_sim,Target_Points(1,:),'m');
set(gca, 'FontSize', 14)
ylabel('X-axis (m)','fontsize',14)
xlim([0,tf])
subplot(3,1,2)
plot(time_sim,Theoretical_P(2,:),'.b',...
     time_sim,Kal_P(2,:),'g',...
     time_sim,Target_Points(4,:),'m');
set(gca, 'FontSize', 14)
ylabel('Y-axis (m)','fontsize',14)
xlim([0,tf])
subplot(3,1,3)
plot(time_sim,Theoretical_P(3,:),'.b',...
     time_sim,Kal_P(3,:),'g',...
     time_sim,Target_Points(7,:),'m');
set(gca, 'FontSize', 14)
xlabel('Simulation Time (s)','fontsize',14)
ylabel('Z-axis (m)','fontsize',14)
xlim([0,tf])

% Plots the angular positions
figure
subplot(3,1,1)
plot(time_sim,Theoretical_P(4,:),'.b',...
     time_sim,Kal_P(4,:),'g',...
     time_sim,Target_Points(10,:),'m');
set(gca, 'FontSize', 14)
ylabel('X-axis (rad)','fontsize',14)
xlim([0,tf])
subplot(3,1,2)
plot(time_sim,Theoretical_P(5,:),'.b',...
     time_sim,Kal_P(5,:),'g',...
     time_sim,Target_Points(13,:),'m');
set(gca, 'FontSize', 14)
ylabel('Y-axis (rad)','fontsize',14)
xlim([0,tf])
subplot(3,1,3)
plot(time_sim,Theoretical_P(6,:),'.b',...
     time_sim,Kal_P(6,:),'g',...
     time_sim,Target_Points(16,:),'m');
set(gca, 'FontSize', 14)
xlabel('Simulation Time (s)','fontsize',14)
ylabel('Z-axis (rad)','fontsize',14)
xlim([0,tf])

```

```
610 % Plots the linear acceleration residuals
611 figure
612 subplot(3,1,1)
613 plot(time_sim,Sens_A(1,:)-Kal_A(1,:));
614 set(gca, 'FontSize', 14)
615 ylabel('X-axis (m/s^2)','fontsize',14)
616 xlim([0,tf])
617 subplot(3,1,2)
618 plot(time_sim,Sens_A(2,:)-Kal_A(2,:));
619 set(gca, 'FontSize', 14)
620 ylabel('Y-axis (m/s^2)','fontsize',14)
621 xlim([0,tf])
622 subplot(3,1,3)
623 plot(time_sim,Sens_A(3,:)-Kal_A(3,:));
624 set(gca, 'FontSize', 14)
625 xlabel('Simulation Time (s)','fontsize',14)
626 ylabel('Z-axis (m/s^2)','fontsize',14)
627 xlim([0,tf])
628
629 % Plots the angular velocity residuals
630 figure
631 subplot(3,1,1)
632 plot(time_sim,Sens_V(4,:)-Kal_V(4,:));
633 set(gca, 'FontSize', 14)
634 ylabel('X-axis (rad/s)','fontsize',14)
635 xlim([0,tf])
636 subplot(3,1,2)
637 plot(time_sim,Sens_V(5,:)-Kal_V(5,:));
638 set(gca, 'FontSize', 14)
639 ylabel('Y-axis (rad/s)','fontsize',14)
640 xlim([0,tf])
641 subplot(3,1,3)
642 plot(time_sim,Sens_V(6,:)-Kal_V(6,:));
643 set(gca, 'FontSize', 14)
644 xlabel('Simulation Time (s)','fontsize',14)
645 ylabel('Z-axis (rad/s)','fontsize',14)
646 xlim([0,tf])
647
648 %%% Save the plots
649
650 h = get(0,'children');
651
652 for i=1:length(h)
653
654     savename =
    ↪ strcat('/mnt/Common_Drive/College/Rose-Hulman/Current_Classes/GRAD_590/Thesis/Figures/Simulate
655     set(h(i),'color','w', 'Position', [100, 100, 840, 630]);
656     img = getframe(h(i));
657     imwrite(img.cdata, [savename, '.png']);
```

```
658     set(h(i),'color',[0.8 0.8 0.8]);
659
660 %     saveas(h(i),savename,'epsc')
661
662 end
```

I.2 Data Reader

This code imports the motor performance curves into a simulation.

```
1  % Dat_File_Reader
2  % Created by Peter Olejnik
3  %
4  % Purpose: Imports the motor properties from the .dat file which were
5  %          identified earlier
6  %
7
8  function [M1,M2,M3,M4] = Dat_File_Reader()
9
10 fn = fopen('Motor_Cal.dat');
11 Read_Data = textscan(fn,'%s','Delimiter','\n');
12 fclose(fn);
13
14 % Lines where the data is
15
16 M1_Range = find(ismember(Read_Data{1},'M1 Range'));
17 M1_Thrust = find(ismember(Read_Data{1},'M1 Thrust'));
18 M1_Torque = find(ismember(Read_Data{1},'M1 Torque'));
19 M1_EMF = find(ismember(Read_Data{1},'M1 EMF'));
20
21 M2_Range = find(ismember(Read_Data{1},'M2 Range'));
22 M2_Thrust = find(ismember(Read_Data{1},'M2 Thrust'));
23 M2_Torque = find(ismember(Read_Data{1},'M2 Torque'));
24 M2_EMF = find(ismember(Read_Data{1},'M2 EMF'));
25
26 M3_Range = find(ismember(Read_Data{1},'M3 Range'));
27 M3_Thrust = find(ismember(Read_Data{1},'M3 Thrust'));
28 M3_Torque = find(ismember(Read_Data{1},'M3 Torque'));
29 M3_EMF = find(ismember(Read_Data{1},'M3 EMF'));
30
31 M4_Range = find(ismember(Read_Data{1},'M4 Range'));
32 M4_Thrust = find(ismember(Read_Data{1},'M4 Thrust'));
33 M4_Torque = find(ismember(Read_Data{1},'M4 Torque'));
34 M4_EMF = find(ismember(Read_Data{1},'M4 EMF'));
35
36 % Motor 1 imports
37
```

```
M1_Range = ...
    [str2num(Read_Data{1}{M1_Range+1}),...
    str2num(Read_Data{1}{M1_Range+2}),...
    0,0];
M1_Thrust = ...
    [str2num(Read_Data{1}{M1_Thrust+1}),...
    str2num(Read_Data{1}{M1_Thrust+2}),...
    str2num(Read_Data{1}{M1_Thrust+3}),...
    0];
M1_Torque = ...
    [str2num(Read_Data{1}{M1_Torque+1}),...
    str2num(Read_Data{1}{M1_Torque+2}),...
    str2num(Read_Data{1}{M1_Torque+3}),...
    0];
M1_EMF = ...
    [str2num(Read_Data{1}{M1_EMF+1}),...
    str2num(Read_Data{1}{M1_EMF+2}),...
    str2num(Read_Data{1}{M1_EMF+3}),...
    str2num(Read_Data{1}{M1_EMF+4})];
M1 = [M1_Range;M1_Thrust;M1_Torque;M1_EMF];

% Motor 2 imports

M2_Range = ...
    [str2num(Read_Data{1}{M2_Range+1}),...
    str2num(Read_Data{1}{M2_Range+2}),...
    0,0];
M2_Thrust = ...
    [str2num(Read_Data{1}{M2_Thrust+1}),...
    str2num(Read_Data{1}{M2_Thrust+2}),...
    str2num(Read_Data{1}{M2_Thrust+3}),...
    0];
M2_Torque = ...
    [str2num(Read_Data{1}{M2_Torque+1}),...
    str2num(Read_Data{1}{M2_Torque+2}),...
    str2num(Read_Data{1}{M2_Torque+3}),...
    0];
M2_EMF = ...
    [str2num(Read_Data{1}{M2_EMF+1}),...
    str2num(Read_Data{1}{M2_EMF+2}),...
    str2num(Read_Data{1}{M2_EMF+3}),...
    str2num(Read_Data{1}{M2_EMF+4})];
M2 = [M2_Range;M2_Thrust;M2_Torque;M2_EMF];

% Motor 3 imports

M3_Range = ...
    [str2num(Read_Data{1}{M3_Range+1}),...
    str2num(Read_Data{1}{M3_Range+2}),...
```

```
    0,0];
M3_Thrust = ...
    [str2num(Read_Data{1}{M3_Thrust+1}),...
    str2num(Read_Data{1}{M3_Thrust+2}),...
    str2num(Read_Data{1}{M3_Thrust+3}),...
    0];
M3_Torque = ...
    [str2num(Read_Data{1}{M3_Torque+1}),...
    str2num(Read_Data{1}{M3_Torque+2}),...
    str2num(Read_Data{1}{M3_Torque+3}),...
    0];
M3_EMF = ...
    [str2num(Read_Data{1}{M3_EMF+1}),...
    str2num(Read_Data{1}{M3_EMF+2}),...
    str2num(Read_Data{1}{M3_EMF+3}),...
    str2num(Read_Data{1}{M3_EMF+4})];
M3 = [M3_Range;M3_Thrust;M3_Torque;M3_EMF];

% Motor 4 imports

M4_Range = ...
    [str2num(Read_Data{1}{M4_Range+1}),...
    str2num(Read_Data{1}{M4_Range+2}),...
    0,0];
M4_Thrust = ...
    [str2num(Read_Data{1}{M4_Thrust+1}),...
    str2num(Read_Data{1}{M4_Thrust+2}),...
    str2num(Read_Data{1}{M4_Thrust+3}),...
    0];
M4_Torque = ...
    [str2num(Read_Data{1}{M4_Torque+1}),...
    str2num(Read_Data{1}{M4_Torque+2}),...
    str2num(Read_Data{1}{M4_Torque+3}),...
    0];
M4_EMF = ...
    [str2num(Read_Data{1}{M4_EMF+1}),...
    str2num(Read_Data{1}{M4_EMF+2}),...
    str2num(Read_Data{1}{M4_EMF+3}),...
    str2num(Read_Data{1}{M4_EMF+4})];
M4 = [M4_Range;M4_Thrust;M4_Torque;M4_EMF];

end
```

I.3 Force to Throttle Converter

This code coverts a motors force into its equivalent throttle

```
% Force2Throttle
% Created by Peter Olejnik
%
% Purpose:  Convert a motors force to its equivelent throttle
%

function [Throttle] = Force2Throttle(Forces,M1_Traits,M2_Traits,M3_Traits,M4_Traits)

Throttle = [[],[],[],[]];

M1_Range = M1_Traits(1,1:2);
M1_Thrust = M1_Traits(2,1:3);
M1_Torque = M1_Traits(3,1:3);

M2_Range = M2_Traits(1,1:2);
M2_Thrust = M2_Traits(2,1:3);
M2_Torque = M2_Traits(3,1:3);

M3_Range = M3_Traits(1,1:2);
M3_Thrust = M3_Traits(2,1:3);
M3_Torque = M3_Traits(3,1:3);

M4_Range = M4_Traits(1,1:2);
M4_Thrust = M4_Traits(2,1:3);
M4_Torque = M4_Traits(3,1:3);

M1_Throttle = (-M1_Thrust(2) + sqrt(M1_Thrust(2)^2 - ...
            4*M1_Thrust(1)*(M1_Thrust(3)-Forces(1))))/...
            (2*M1_Thrust(1));
M2_Throttle = (-M2_Thrust(2) + sqrt(M2_Thrust(2)^2 - ...
            4*M2_Thrust(1)*(M2_Thrust(3)-Forces(2))))/...
            (2*M2_Thrust(1));
M3_Throttle = (-M3_Thrust(2) + sqrt(M3_Thrust(2)^2 - ...
            4*M3_Thrust(1)*(M3_Thrust(3)-Forces(3))))/...
            (2*M3_Thrust(1));
M4_Throttle = (-M4_Thrust(2) + sqrt(M4_Thrust(2)^2 - ...
            4*M4_Thrust(1)*(M4_Thrust(3)-Forces(4))))/...
            (2*M4_Thrust(1));

Throttle(1) = min(max(M1_Throttle,M1_Range(1)),M1_Range(2));
Throttle(2) = min(max(M2_Throttle,M2_Range(1)),M2_Range(2));
Throttle(3) = min(max(M3_Throttle,M3_Range(1)),M3_Range(2));
Throttle(4) = min(max(M4_Throttle,M4_Range(1)),M4_Range(2));
```

I.4 Theoretical State Computer

This bit of code computes the theoretical states, based on the current states of the system.

```
% created on 15/02/2016

function [Sim_Out] = Theoretical_Model(Throttle,...
    M1_Traits,M2_Traits,M3_Traits,M4_Traits,...
    Phi,Theta,Phi_d,Theta_d,Psi_d,Ixx,Iyy,Izz,mQc,g,d)

Inputs = Motor_Performance(Throttle,M1_Traits,M2_Traits,M3_Traits,M4_Traits);
F = Inputs(1:4);
T = Inputs(5:8);

X_d_d = -sin(Phi)*(F(1)+F(2)+F(3)+F(4))/mQc;
Y_d_d = cos(Phi)*sin(Theta)*(F(1)+F(2)+F(3)+F(4))/mQc;
Z_d_d = -cos(Phi)*cos(Theta)*(F(1)+F(2)+F(3)+F(4))/mQc + g;

Phi_d_d = sqrt(2)*d*(F(1)+F(2)-F(3)-F(4))/(Ixx*2) + ...
    Theta_d*Psi_d*(Izz-Iyy)/(Ixx);
Theta_d_d = sqrt(2)*d*(F(2)+F(3)-F(1)-F(4))/(Iyy*2) + ...
    Phi_d*Psi_d*(Ixx-Izz)/(Iyy);
Psi_d_d = (T(1)+T(2)+T(3)+T(4))/(Izz) + ...
    Phi_d*Theta_d*(Iyy-Ixx)/(Izz);

Sim_Out = [X_d_d;Y_d_d;Z_d_d;Phi_d_d;Theta_d_d;Psi_d_d];

end
```

I.5 Noise Addition

This code is responsible for simulating the sensor output.

```
% Noiseafier
% Created by Peter Olejnik
%
% Purpose:  Adds noise to a theoretical calculation to simulate a sensor
%           reading
%

function [NoisyOutput] = Noiseafier(Input,Deviatiion)
   NoisyOutput = Input + random('norm',0,Deviatiion);
end
```

I.6 Force to Torque Converter

This code coverts a motors force into its equivalent torque

```
% Force2Torque
% Created by Peter Olejnik
%
% Purpose:  Convert a motors force to its equivelent torque
%

function [Torques] = Force2Torque(Forces,M1_Traits,M2_Traits,M3_Traits,M4_Traits)

M1_Thrust = M1_Traits(2,1:3);
M1_Torque = M1_Traits(3,1:3);

M2_Thrust = M2_Traits(2,1:3);
M2_Torque = M2_Traits(3,1:3);

M3_Thrust = M3_Traits(2,1:3);
M3_Torque = M3_Traits(3,1:3);

M4_Thrust = M4_Traits(2,1:3);
M4_Torque = M4_Traits(3,1:3);

M1_Throttle = (-M1_Thrust(2) + sqrt(M1_Thrust(2)^2 - ...
            4*M1_Thrust(1)*(M1_Thrust(3)-Forces(1))))/...
            (2*M1_Thrust(1));
M2_Throttle = (-M2_Thrust(2) + sqrt(M2_Thrust(2)^2 - ...
            4*M2_Thrust(1)*(M2_Thrust(3)-Forces(2))))/...
            (2*M2_Thrust(1));
M3_Throttle = (-M3_Thrust(2) + sqrt(M3_Thrust(2)^2 - ...
            4*M3_Thrust(1)*(M3_Thrust(3)-Forces(3))))/...
            (2*M3_Thrust(1));
M4_Throttle = (-M4_Thrust(2) + sqrt(M4_Thrust(2)^2 - ...
            4*M4_Thrust(1)*(M4_Thrust(3)-Forces(4))))/...
            (2*M4_Thrust(1));

Torques(1) = polyval(M1_Torque,M1_Throttle);
Torques(2) = polyval(M2_Torque,M2_Throttle);
Torques(3) = polyval(M3_Torque,M3_Throttle);
Torques(4) = polyval(M4_Torque,M4_Throttle);

end
```

I.7 Torque to Force Converter

This code coverts a motors torque into its equivalent force

```
% Torque2Force
% Created by Peter Olejnik
%
```

```
% Purpose:  Convert a torque force to its equivelent force
%


function [Forces] = Torque2Force(Torques,M1_Traits,M2_Traits,M3_Traits,M4_Traits)

M1_Thrust = M1_Traits(2,1:3);
M1_Torque = M1_Traits(3,1:3);

M2_Thrust = M2_Traits(2,1:3);
M2_Torque = M2_Traits(3,1:3);

M3_Thrust = M3_Traits(2,1:3);
M3_Torque = M3_Traits(3,1:3);

M4_Thrust = M4_Traits(2,1:3);
M4_Torque = M4_Traits(3,1:3);

M1_Throttle = (-M1_Torque(2) + sqrt(M1_Torque(2)^2 - ...
            4*M1_Torque(1)*(M1_Torque(3)-Torques(1))))/...
            (2*M1_Torque(1));
M2_Throttle = (-M2_Torque(2) - sqrt(M2_Torque(2)^2 - ...
            4*M2_Torque(1)*(M2_Torque(3)-Torques(2))))/...
            (2*M2_Torque(1));
M3_Throttle = (-M3_Torque(2) + sqrt(M3_Torque(2)^2 - ...
            4*M3_Torque(1)*(M3_Torque(3)-Torques(3))))/...
            (2*M3_Torque(1));
M4_Throttle = (-M4_Torque(2) - sqrt(M4_Torque(2)^2 - ...
            4*M4_Torque(1)*(M4_Torque(3)-Torques(4))))/...
            (2*M4_Torque(1));

Forces(1) = polyval(M1_Thrust,M1_Throttle);
Forces(2) = polyval(M2_Thrust,M2_Throttle);
Forces(3) = polyval(M3_Thrust,M3_Throttle);
Forces(4) = polyval(M4_Thrust,M4_Throttle);

end
```

Appendix J

Quadcopter Code

J.1 Set Up Part 1

The script which configures the quadcopter into a quadcopter comes in two parts. The first part sets up Github on the quadcopter.

```
#!/bin/sh

# Setup_P1.sh
# Created by Peter Olejnik
#
# Purpose - This is part one of the set up script, which sets up a blank BBB
#           into a QC.
#

# Update time and ensure it is in sync
sudo ntpdate pool.ntp.org

#Update the system
sudo apt-get update

# BBB Python libraries
sudo apt-get install -y build-essential python-dev python-setuptools python-pip
 ↪  python-smbus
sudo pip install Adafruit_BBIO

# Git
sudo apt-get install -y git xclip

ssh-keygen -t rsa -b 4096 -C "olejnikpr@gmail.com"
eval `ssh-agent -s`
ssh-add ~/.ssh/id_rsa
```

```
xclip -sel clip < ~/.ssh/id_rsa.pub
```

J.2 Set Up Part 2

The second part of the set up script is what really does most of the configuration. Preconfigured files are pulled from the Github repository. Programs are also installed from the linux repositories.The preconfigured files are then placed in the correct places.

```
#!/bin/sh

# Setup_P2.sh
# Created by Peter Olejnik
#
# Purpose - This is part two of the set up script, which sets up a blank BBB
#           into a QC.
#

cd /
git clone git@github.com:ValRose/BBB_Quadcopter.git
git clone git@github.com:ValRose/Rose_Bone.git

# Host name

cp /BBB_Quadcopter/QC_Setup/hostname /etc/
cp /BBB_Quadcopter/QC_Setup/hosts /etc/

# network files

rm /etc/network/interfaces
ln -s /BBB_Quadcopter/QC_System/interfaces /etc/network/interfaces

# hostapd

sudo apt-get install -y hostapd

cd ./RTL8188-hostapd-2.0/hostapd
make
make install

rm /etc/default/hostapd
rm /etc/hostapd/hostapd.conf
ln -s /BBB_Quadcopter/QC_System/hostapd /etc/default/hostapd
ln -s /BBB_Quadcopter/QC_System/hostapd.conf /etc/hostapd/hostapd.conf

#rm /usr/sbin/hostapd
#cp /BBB_Quadcopter/QC_Setup/hostapd /usr/sbin/
```

```

# DHCP

sudo apt-get install isc-dhcp-server

rm /etc/dhcp/dhcpd.conf
ln -s /BBB_Quadcopter/QC_System/dhcpd.conf /etc/dhcp/dhcpd.conf

# Service Creation for start up script

cp /BBB_Quadcopter/QC_System/BBB_QC_Start.service
 ↪  /lib/systemd/system/BBB_QC_Start.service
ln -s /BBB_Quadcopter/QC_System/isc-dhcp-server /etc/default/isc-dhcp-server

systemctl daemon-reload
systemctl enable BBB_QC_Start.service

rm /lib/systemd/system/BBB_QC_Start.service
ln -s /BBB_Quadcopter/QC_System/BBB_QC_Start.service
 ↪  /lib/systemd/system/BBB_QC_Start.service

systemctl start BBB_QC_Start.service
```

J.3 Quadcopter Start Service

This is the file which governs the start up service for Systemd.

```
[Unit]
Description=BeagleBone Black Quadcopter Start Script

[Service]
Type=simple
ExecStart=/BBB_Quadcopter/QC_System/BBB_QC_Start_Script.sh

[Install]
WantedBy=multi-user.target
```

J.4 DHCP Configuration

This is the file which configures the DHCP server. The only bit that was changed from the default file were the end lines 109-111.

```
#
# Sample configuration file for ISC dhcpd for Debian
#
```

```
4  #
5
6  # The ddns-updates-style parameter controls whether or not the server will
7  # attempt to do a DNS update when a lease is confirmed. We default to the
8  # behavior of the version 2 packages ('none', since DHCP v2 didn't
9  # have support for DDNS.)
10 ddns-update-style none;
11
12 # option definitions common to all supported networks...
13 option domain-name "example.org";
14 option domain-name-servers ns1.example.org, ns2.example.org;
15
16 default-lease-time 600;
17 max-lease-time 7200;
18
19 # If this DHCP server is the official DHCP server for the local
20 # network, the authoritative directive should be uncommented.
21 # authoritative;
22
23 # Use this to send dhcp log messages to a different log file (you also
24 # have to hack syslog.conf to complete the redirection).
25 log-facility local7;
26
27 # No service will be given on this subnet, but declaring it helps the
28 # DHCP server to understand the network topology.
29
30 #subnet 10.152.187.0 netmask 255.255.255.0 {
31 #}
32
33 # This is a very basic subnet declaration.
34
35 #subnet 10.254.239.0 netmask 255.255.255.224 {
36 #  range 10.254.239.10 10.254.239.20;
37 #  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
38 #}
39
40 # This declaration allows BOOTP clients to get dynamic addresses,
41 # which we don't really recommend.
42
43 #subnet 10.254.239.32 netmask 255.255.255.224 {
44 #  range dynamic-bootp 10.254.239.40 10.254.239.60;
45 #  option broadcast-address 10.254.239.31;
46 #  option routers rtr-239-32-1.example.org;
47 #}
48
49 # A slightly different configuration for an internal subnet.
50 #subnet 10.5.5.0 netmask 255.255.255.224 {
51 #  range 10.5.5.26 10.5.5.30;
52 #  option domain-name-servers ns1.internal.example.org;
```

```
53 #  option domain-name "internal.example.org";
54 #  option routers 10.5.5.1;
55 #  option broadcast-address 10.5.5.31;
56 #  default-lease-time 600;
57 #  max-lease-time 7200;
58 #}
59
60 # Hosts which require special configuration options can be listed in
61 # host statements.   If no address is specified, the address will be
62 # allocated dynamically (if possible), but the host-specific information
63 # will still come from the host declaration.
64
65 #host passacaglia {
66 #  hardware ethernet 0:0:c0:5d:bd:95;
67 #  filename "vmunix.passacaglia";
68 #  server-name "toccata.fugue.com";
69 #}
70
71 # Fixed IP addresses can also be specified for hosts.   These addresses
72 # should not also be listed as being available for dynamic assignment.
73 # Hosts for which fixed IP addresses have been specified can boot using
74 # BOOTP or DHCP.   Hosts for which no fixed address is specified can only
75 # be booted with DHCP, unless there is an address range on the subnet
76 # to which a BOOTP client is connected which has the dynamic-bootp flag
77 # set.
78 #host fantasia {
79 #  hardware ethernet 08:00:07:26:c0:a5;
80 #  fixed-address fantasia.fugue.com;
81 #}
82
83 # You can declare a class of clients and then do address allocation
84 # based on that.   The example below shows a case where all clients
85 # in a certain class get addresses on the 10.17.224/24 subnet, and all
86 # other clients get addresses on the 10.0.29/24 subnet.
87
88 #class "foo" {
89 #  match if substring (option vendor-class-identifier, 0, 4) = "SUNW";
90 #}
91
92 #shared-network 224-29 {
93 #  subnet 10.17.224.0 netmask 255.255.255.0 {
94 #    option routers rtr-224.example.org;
95 #  }
96 #  subnet 10.0.29.0 netmask 255.255.255.0 {
97 #    option routers rtr-29.example.org;
98 #  }
99 #  pool {
100 #    allow members of "foo";
101 #    range 10.17.224.10 10.17.224.250;
```

```
#  }
#  pool {
#    deny members of "foo";
#    range 10.0.29.10 10.0.29.230;
#  }
#}

subnet 192.168.4.0 netmask 255.255.255.0 {
  range 192.168.4.2 192.168.4.10;
}
```

J.5 Hostapd Path Configuration

This is the file which configures the hostapd path for the configuration file. The only bit that was changed from the default file was line 10.

```
# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
#         -d   show more debug messages (-dd for even more)
#         -K   include key data in debug messages
#         -t   include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
#
#DAEMON_OPTS=""
```

J.6 Hostapd Configuration

This is the file which configures the hostapd.

```
# Basic configuration

interface=wlan0
ssid=BBB_QC_AP
```

```
channel=1
#bridge=br0

# WPA and WPA2 configuration

macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=3
wpa_passphrase=Val_Rose
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP

# Hardware configuration

driver=rtl871xdrv
ieee80211n=1
hw_mode=g
device_name=RTL8192CU
manufacturer=Realtek
```

J.7 Interfaces Configuration

This is the file which configures the various devices which the BeagleBone uses to talk with other devices.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
#auto eth0
#iface eth0 inet dhcp
# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE

# The secondary network interface
#auto eth1
#iface eth1 inet dhcp

# WiFi Example
auto wlan0
allow-hotplug wlan0
```

```
iface wlan0 inet static
    address 192.168.4.1
    netmask 255.255.255.0
    network 192.168.4.0
    broadcast 192.168.4.255

# Ethernet/RNDIS gadget (g_ether)
# Used by: /opt/scripts/boot/autoconfigure_usb0.sh
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.252
    network 192.168.7.0
    gateway 192.168.7.1
```

J.8 ISC-DHCP-Server Device Configuration

This is the file which configures the ISC-DHCP-Server and lets it know what piece of hardware it interfaces with.

```
# Defaults for isc-dhcp-server initscript
# sourced by /etc/init.d/isc-dhcp-server
# installed at /etc/default/isc-dhcp-server by the maintainer scripts

#
# This is a POSIX shell fragment
#

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPD_CONF=/etc/dhcp/dhcpd.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPD_PID=/var/run/dhcpd.pid

# Additional options to start dhcpd with.
#       Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="wlan0"
```

J.9 Quadcopter Functions

The script below contains the functions which interact with the quadcopter's hardware. The script also does have a few functions related to the filter and controller.

```
#!/usr/bin/python

# BBB_QC.py
# Created by Peter Olejnik
#
# Purpose - This is the general library that will provide the functions that
#                       will control the higherlevel Beaglebone Black QuadCopter.
 ↪  Specificly
#                       written with the thesis QC in mind.
#
################ Imports ###############

import Adafruit_BBIO.GPIO as GPIO # Adafruit GPIO libraries
import Adafruit_BBIO.PWM as PWM # Adafruit PWM libraries
import Adafruit_BBIO.ADC as ADC        # Adafruit ADC libraries
import smbus # Libraries used for I2C communication
                        # Adafruit libraries also exist
import os # Libraries for system nature jobs
import sys # Libraries for system nature jobs
import time # For delays
import signal # detects inputs/interupts
import math # A math function library

import numpy as np # Numpy, the Matlab alternative!

################ Variables ###############

I2C_Bus = []

XA_OS = 0
YA_OS = 0
ZA_OS = 0

XG_OS = 0
YG_OS = 0
ZG_OS = 0

# Define all the pins
# These are up here because multiple functions use them

M_Range = []
M_Thrust = []
M_Torque = []
```

```
43 M_EMF = []
44
45 Q_kf = []
46 R_kf = []
47 K_lqr = []
48
49 Motor_PWM_1 = 'P9_14'
50 Motor_PWM_2 = 'P8_13'
51 Motor_PWM_3 = 'P8_19'
52 Motor_PWM_4 = 'P9_22'
53
54 Motor_ADC_1 = 'P9_36'
55 Motor_ADC_2 = 'P9_38'
56 Motor_ADC_3 = 'P9_40'
57 Motor_ADC_4 = 'P9_39'
58
59 ############### Functions ###############
60
61 # These functions govern the throttle of the BBB QC motors.
62
63 def M1_Throttle(Throttle):
64         PWM.set_duty_cycle(Motor_PWM_1,5.0 + Throttle*5.0/100)
65
66 def M2_Throttle(Throttle):
67         PWM.set_duty_cycle(Motor_PWM_2,5.0 + Throttle*5.0/100)
68
69 def M3_Throttle(Throttle):
70         PWM.set_duty_cycle(Motor_PWM_3,5.0 + Throttle*5.0/100)
71
72 def M4_Throttle(Throttle):
73         PWM.set_duty_cycle(Motor_PWM_4,5.0 + Throttle*5.0/100)
74
75 def Motor_All_Throttle(Throttle):
76         M1_Throttle(Throttle)
77         M2_Throttle(Throttle)
78         M3_Throttle(Throttle)
79         M4_Throttle(Throttle)
80
81 # These functions read the adc ports
82
83 def M1_ADC_Read():
84         return ADC.read(Motor_ADC_1)*1.8
85
86 def M2_ADC_Read():
87         return ADC.read(Motor_ADC_2)*1.8
88
89 def M3_ADC_Read():
90         return ADC.read(Motor_ADC_3)*1.8
91
```

```
92  def M4_ADC_Read():
93          return ADC.read(Motor_ADC_4)*1.8
94
95  # This function initializes the quadcopter motors.
96
97  def Motor_Initialize():
98          ADC.setup()
99          PWM.start(Motor_PWM_1, 0, 50)
100         PWM.start(Motor_PWM_2, 0, 50)
101         PWM.start(Motor_PWM_3, 0, 50)
102         PWM.start(Motor_PWM_4, 0, 50)
103
104 # These functions read the sensors and return the read values.
105
106 def Read_X_A():
107         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x3b)
108         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x3c)
109         Full_Bits = (Upper_Bits << 8) + Lower_Bits
110         if (Full_Bits >= 0x8000):
111                 Twos_Comp = -((65535 - Full_Bits) + 1)
112         else:
113                 Twos_Comp = Full_Bits
114         return (Twos_Comp/16384.0 - XA_OS)*9.81 # Value is in "m/s2"
115
116 def Read_Y_A():
117         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x3d)
118         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x3e)
119         Full_Bits = (Upper_Bits << 8) + Lower_Bits
120         if (Full_Bits >= 0x8000):
121                 Twos_Comp = -((65535 - Full_Bits) + 1)
122         else:
123                 Twos_Comp = Full_Bits
124         return (Twos_Comp/16384.0 - YA_OS)*9.81 # Value is in "m/s2"
125
126 def Read_Z_A():
127         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x3f)
128         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x40)
129         Full_Bits = (Upper_Bits << 8) + Lower_Bits
130         if (Full_Bits >= 0x8000):
131                 Twos_Comp = -((65535 - Full_Bits) + 1)
132         else:
133                 Twos_Comp = Full_Bits
134         return (Twos_Comp/16384.0 - ZA_OS)*9.81 # Value is in "m/s2"
135
136 def Read_X_G():
137         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x43)
138         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x44)
139         Full_Bits = (Upper_Bits << 8) + Lower_Bits
140         if (Full_Bits >= 0x8000):
```

```
141                 Twos_Comp = -((65535 - Full_Bits) + 1)}
142         else:
143                 Twos_Comp = Full_Bits
144         return math.radians(Twos_Comp/131.0 - XG_OS) # Value is in "rad/s"
145
146 def Read_Y_G():
147         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x45)
148         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x46)
149         Full_Bits = (Upper_Bits << 8) + Lower_Bits
150         if (Full_Bits >= 0x8000):
151                 Twos_Comp = -((65535 - Full_Bits) + 1)
152         else:
153                 Twos_Comp = Full_Bits
154         return math.radians(Twos_Comp/131.0 - YG_OS) # Value is in "rad/s"
155
156 def Read_Z_G():
157         Upper_Bits = I2C_Bus.read_byte_data(0x68, 0x47)
158         Lower_Bits = I2C_Bus.read_byte_data(0x68, 0x48)
159         Full_Bits = (Upper_Bits << 8) + Lower_Bits
160         if (Full_Bits >= 0x8000):
161                 Twos_Comp = -((65535 - Full_Bits) + 1)
162         else:
163                 Twos_Comp = Full_Bits
164         return math.radians(Twos_Comp/131.0 - ZG_OS) # Value is in "rad/s"
165
166 # This function initializes the sensors and zeros them out.
167
168 def Sensor_Initialize():
169
170         print "Initializing MPU 6050..."
171
172         global I2C_Bus
173         I2C_Bus = smbus.SMBus(1) # Set I2C bus
174
175         print "Waking Device..."
176         I2C_Bus.write_byte_data(0x68, 0x6b, 0) # Address, registar, value
177
178         print "MPU 6050 fully initialized!"
179
180         print "Generating offsets"
181
182         global XA_OS
183         global YA_OS
184         global ZA_OS
185
186         global XG_OS
187         global YG_OS
188         global ZG_OS
189
```

```
190         XA_OS = Read_X_A()
191         YA_OS = Read_Y_A()
192         ZA_OS = Read_Z_A()
193 
194         XG_OS = Read_X_G()
195         YG_OS = Read_Y_G()
196         ZG_OS = Read_Z_G()
197 
198         print "\n OFFSETS"
199         print " X Acceleration     Y Acceleration     Z Acceleration"
200         print "----------------   ----------------   ----------------"
201 
202         print "{0:16.12f}   {1:16.12f}   {2:16.12f}".format(XA_OS,YA_OS,ZA_OS)
203 
204         print "\n X Angular Vel       Y Angular Vel      Z Angular Vel"
205         print "----------------   ----------------   ----------------"
206 
207         print "{0:16.12f}   {1:16.12f}   {2:16.12f}\n".format(XG_OS,YG_OS,ZG_OS)
208 
209 # This function imports the quadcopter property values
210 
211 def Load_Parameters():
212 
213         global M_Range
214         global M_Thrust
215         global M_Torque
216         global M_EMF
217 
218         global Q_kf
219         global R_kf
220         global K_lqr
221 
222         M_Range = np.genfromtxt('./Properties/Motor_Range.txt');
223         M_Thrust = np.genfromtxt('./Properties/Motor_Thrust.txt');
224         M_Torque = np.genfromtxt('./Properties/Motor_Torque.txt');
225         M_EMF = np.genfromtxt('./Properties/Motor_EMF.txt');
226 
227         Q_kf = np.genfromtxt('./Properties/Q_kf.txt');
228         R_kf = np.genfromtxt('./Properties/R_kf.txt');
229         K_lqr = np.genfromtxt('./Properties/K_Lqr.txt');
230 
231         Ixx = 6.9351e-03
232         Iyy = 6.9351e-03
233         Izz = 1.1474e-01
234 
235         mQc = 0.47995
236         g = 9.81
237         d = 0.1525
238 
```

```
239         s2d2 = math.sqrt(2)*d/2
240
241 # This function imports a flight plan for the quadcopter to follow
242
243 def Load_Flightplan():
244
245         global Flight_Plan
246
247         Flight_Plan = np.genfromtxt('./Properties/Flight_Plan.txt');
248
249 # This function calculates the theoretical states of the quadcopter, based on
250 # the current states, the current outputs, and time step.
251
252 def Theoretical_States(Outputs,States,dt):
253
254         return np.array([\
255         [States[0]+dt*States[1]],\
256         [States[1]+dt*States[2]],\
257         [-sin(States[9])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc],\
258         [States[3]+dt*States[4]],\
259         [States[4]+dt*States[5]],\
260         [cos(States[9])*sin(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc],\
261         [States[6]+dt*States[7]],\
262         [States[7]+dt*States[8]],\
263         [-cos(States[9])*cos(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
    ↪  + g],\
264         [States[9]+dt*States[10]],\
265         [States[10]+dt*States[11]],\
266         [s2d2*(Outputs[0]+Outputs[1]-Outputs[2]-Outputs[3])/(Ixx) +
    ↪  States[13]*States[16]*(Izz-Iyy)/(Ixx)],\
267         [States[12]+dt*States[13]],\
268         [States[13]+dt*States[14]],\
269         [s2d2*(-Outputs[0]+Outputs[1]+Outputs[2]-Outputs[3])/(Iyy) +
    ↪  States[10]*States[16]*(Ixx-Izz)/(Iyy)],\
270         [States[15]+dt*States[16]],\
271         [States[16]+dt*States[17]],\
272         [(Outputs[4]+Outputs[5]+Outputs[6]+Outputs[7])/(Izz) +
    ↪  States[10]*States[13]*(Iyy-Ixx)/(Izz)]\
273         ])
274
275 # This function augments the current Jacobina matrix for the Kalman estimate
276
277 def A_Jacobian_Modder(Outputs,States,A_Jac):
278
279         A_Jac[2,9] = -cos(States[9])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
280         A_Jac(5,9) =
    ↪  -sin(States[9])*sin(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
281         A_Jac(5,12) =
    ↪  cos(States[9])*cos(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
```

```
282         A_Jac(8,9) =
    ↪ sin(States[9])*cos(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
283         A_Jac(8,12) =
    ↪ cos(States[9])*sin(States[12])*(Outputs[0]+Outputs[1]+Outputs[2]+Outputs[3])/mQc
284         A_Jac(11,13) = States[16]*(Izz-Iyy)/Ixx
285         A_Jac(11,16) = States[13]*(Izz-Iyy)/Ixx
286         A_Jac(14,10) = States[16]*(Ixx-Izz)/Iyy
287         A_Jac(14,16) = States[10]*(Ixx-Izz)/Iyy
288         A_Jac(17,10) = States[13]*(Iyy-Ixx)/Izz
289         A_Jac(17,13) = States[10]*(Iyy-Ixx)/Izz
290         return A_Jac
291
292 # This function takes the 8 outputs and converts them into 4 throttles.
293
294 def Throttle_Generator(Outputs):
295
296         F_Net = Outputs[:4] + (mQc*g/4)
297
298         Throttle_1 = (-M_Thrust[0,1] + math.sqrt(M_Thrust[0,1]*M_Thrust[0,1] -
    ↪ 4*M_Thrust[0,0]*(M_Thrust[0,2]-F_Net[0])))/(2*M_Thrust[0,0])
299         Throttle_2 = (-M_Thrust[1,1] + math.sqrt(M_Thrust[1,1]*M_Thrust[1,1] -
    ↪ 4*M_Thrust[1,0]*(M_Thrust[1,2]-F_Net[1])))/(2*M_Thrust[1,0])
300         Throttle_3 = (-M_Thrust[2,1] + math.sqrt(M_Thrust[2,1]*M_Thrust[2,1] -
    ↪ 4*M_Thrust[2,0]*(M_Thrust[2,2]-F_Net[2])))/(2*M_Thrust[2,0])
301         Throttle_4 = (-M_Thrust[3,1] + math.sqrt(M_Thrust[3,1]*M_Thrust[3,1] -
    ↪ 4*M_Thrust[3,0]*(M_Thrust[3,2]-F_Net[3])))/(2*M_Thrust[3,0])
302
303         T1 = M_Torque[0,0]*Throttle_1*Throttle_1 + M_Torque[0,1]*Throttle_1 +
    ↪ M_Torque[0,2] + Outputs[4]
304         T2 = M_Torque[1,0]*Throttle_2*Throttle_2 + M_Torque[1,1]*Throttle_2 +
    ↪ M_Torque[1,2] + Outputs[5]
305         T3 = M_Torque[2,0]*Throttle_3*Throttle_3 + M_Torque[2,1]*Throttle_3 +
    ↪ M_Torque[2,2] + Outputs[6]
306         T4 = M_Torque[3,0]*Throttle_4*Throttle_4 + M_Torque[3,1]*Throttle_4 +
    ↪ M_Torque[3,2] + Outputs[7]
307
308         Throttle_1 = (-M_Torque[0,1] + math.sqrt(M_Torque[0,1]*M_Torque[0,1] -
    ↪ 4*M_Torque[0,0]*(M_Torque[0,2]-T1)))/(2*M_Torque[0,0])
309         Throttle_2 = (-M_Torque[1,1] - math.sqrt(M_Torque[1,1]*M_Torque[1,1] -
    ↪ 4*M_Torque[1,0]*(M_Torque[1,2]-T2)))/(2*M_Torque[1,0])
310         Throttle_3 = (-M_Torque[2,1] + math.sqrt(M_Torque[2,1]*M_Torque[2,1] -
    ↪ 4*M_Torque[2,0]*(M_Torque[2,2]-T3)))/(2*M_Torque[2,0])
311         Throttle_4 = (-M_Torque[3,1] - math.sqrt(M_Torque[3,1]*M_Torque[3,1] -
    ↪ 4*M_Torque[3,0]*(M_Torque[3,2]-T4)))/(2*M_Torque[3,0])
312
313         M1_Throttle(Throttle_1)
314         M1_Throttle(Throttle_2)
315         M1_Throttle(Throttle_3)
316         M1_Throttle(Throttle_4)
```

J.10 Flight Program

This is the script which operates the flight of the quadcopter. It is here where the Kalman filter and the LQR controller are present.

```
#!/usr/bin/python

# BBB_QC_Run_Program.py
# Created by Peter Olejnik
#
# Purpose - The program which flys the quadcopter.
#

################ Imports ################

import BBB_QC # Written libraries for the Quadcopter
import Adafruit_BBIO.GPIO as GPIO # Adafruit GPIO libraries
import Adafruit_BBIO.PWM as PWM # Adafruit PWM libraries
import sys # Libraries for system nature jobs
import time        # For times and delays
import signal # detects inputs/interupts
import numpy as np # Numpy, the Matlab alternative!

################ Ctrl-C Interupt ################

def Control_C_Exit(signal, frame):

        GPIO.cleanup()
        PWM.cleanup()
        print("\nProgram halted! Exiting program!")
        sys.exit()

################ Main Program ################

signal.signal(signal.SIGINT, Control_C_Exit) # For cleaning up mid run

# Initialize components and variables

BBB_QC.Motor_Initialize()
BBB_QC.Sensor_Initialize()
BBB_QC.Load_Parameters()
BBB_QC.Load_Flightplan()

dt = 0.01 # Time Increment

target_set = 0
Tar_X = Flight_Plan[target_set,0]
Tar_Y = Flight_Plan[target_set,1]
```

```
44 Tar_Z = Flight_Plan[target_set,2]
45
46 Flight_Plan[2:4,2:4] = np.array([[1,2],[3,4]])
47
48 A_Jac = np.zeros((18,18))
49 A_Jac[0:3,0:3] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
50 A_Jac[3:6,3:6] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
51 A_Jac[6:9,6:9] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
52 A_Jac[9:12,9:12] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
53 A_Jac[12:15,12:15] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
54 A_Jac[15:18,15:18] = np.array([[1,dt,0],[0,1,dt],[0,0,0]])
55
56 C_kf = np.zeros((6,18))
57 C_kf[0,2] = 1
58 C_kf[1,5] = 1
59 C_kf[2,8] = 1
60 C_kf[3,10] = 1
61 C_kf[4,13] = 1
62 C_kf[5,16] = 1
63
64 States = np.zeros((18,1))
65 Outputs = np.zeros((8,1))
66
67 R_kf = BBB_QC.R_kf
68 Q_kf = BBB_QC.Q_kf
69 K_lqr = BBB_QC.K_lqr
70
71 P_kf = np.zeros((18,18))
72
73 t_c = 0 # Current time
74 t_f = 85 # Final Time
75 t_s = dt # Time to calculate new conditions
76
77 t_0 = datetime.datetime.now() # time Zero
78
79 # Begin flight loop
80
81 while(t_c < t_f):
82
83 # Find out what the time diffrence is from the start
84
85         t_c = (datetime.datetime.now() - t_0).total_seconds()
86
87 # Adjust target position
88
89         if (t_c > BBB_QC.Flight_Plan[target_set+1,3]):
90                 target_set+=1
91                 Tar_X = BBB_QC.Flight_Plan[target_set,0]
92                 Tar_Y = BBB_QC.Flight_Plan[target_set,1]
```

```
93                  Tar_Z = BBB_QC.Flight_Plan[target_set,2]
94
95          if (t_c > t_s):
96
97                  States = BBB_QC.Theoretical_States(Outputs,States,dt)
98
99  # Compute the residual
100
101                 Residual = np.array(\
102                 [Read_X_A()],\
103                 [Read_Y_A()],\
104                 [Read_Z_A()],\
105                 [Read_X_G()],\
106                 [Read_Y_G()],\
107                 [Read_Z_G()]) - C_kf.dot(States)
108
109 # Compute the Kalman gain
110
111                 A_Jac = BBB_QC.A_Jacobian_Modder(Outputs,States,A_Jac)
112
113                 P_kf = A_Jac.dot(P_kf) + P_kf.dot(A_Jac.T) + Q_kf
114                 K_kf = P_kf.dot(C_kf.T).dot(linalg.inv(C_kf.dot(P_kf).dot(C_kf.T) +
    ↪  R_kf))
115                 P_kf = (np.eye(18) - K_kf.dot(C_kf).dot(P_kf))
116
117 # Compute the Kalman estimate
118
119                 States+= K_kf.dot(Residual)
120
121                 Tar_Error = States
122                 Tar_Error[0]-= Tar_X
123                 Tar_Error[3]-= Tar_Y
124                 Tar_Error[6]-= Tar_Z
125
126 # Compute the output based on the states
127
128                 Outputs = K_lqr.dot(Tar_Error)
129
130 # Compute the output throttles
131
132                 BBB_QC.Throttle_Generator(Outputs)
133
134                 t_s += dt
```

Bibliography

[1] vicatcu, "Wiring a GY-521 to an Arduino Uno R3?" Mar. 2013. [Online]. Available: http://electronics.stackexchange.com/questions/61286/wiring-a-gy-521-to-an-arduino-uno-r3

[2] G. Coley, "BeagleBone Black System Reference Manual," Jan. 2014. [Online]. Available: https://github.com/CircuitCo/BeagleBone-Black/blob/rev_c/BBB_SRM.pdf

[3] "Turnigy AX-2204c 1450kv/70w Brushless Outrunner Motor." [Online]. Available: http://www.hobbyking.com/hobbyking/store/__36280__Turnigy_AX_2204C_1450KV_70W_Brushless_Outrunner_Motor.html

[4] "Exceed RC Proton 10a Brushless Speed Controller ESC." [Online]. Available: http://www.hobbypartz.com/07e22-proton-10a.html

[5] "3132_0 - Micro Load Cell (0-780g) - CZL616c." [Online]. Available: http://www.phidgets.com/products.php?product_id=3132

[6] "LT1167 Single Resistor Gain Programmable, Precision Instrumentation Amplifier," Aug. 2011. [Online]. Available: http://cds.linear.com/docs/en/datasheet/1167fc.pdf

[7] "TL7660 CMOS VOLTAGE CONVERTER," Jun. 2006. [Online]. Available: http://www.ti.com/lit/ds/symlink/tl7660.pdf

[8] L. R. Newcome, *Unmanned aviation: a brief history of unmanned aerial vehicles.* Reston, Va: American Institute of Aeronautics and Astronautics, Inc, 2004.

[9] "Department of Defense Dictionary of Military and Associated Terms, Joint Publication 1-02," Oct. 2007.

[10] J. Mica, "FAA Modernization and Reform Act of 2012," Feb. 2012.

[11] K. Dalamagkidis, K. P. Valavanis, and L. A. Piegl, *On Integrating Unmanned Aircraft Systems into the National Airspace System.* Dordrecht: Springer Netherlands, 2012. [Online]. Available: http://link.springer.com/10.1007/978-94-007-2479-2

[12] C. Anderson, "Agricultural Drones Relatively cheap drones with advanced sensors and imaging capabilities are giving farmers new ways to increase yields and reduce crop damage." *MIT Technology Review.* [Online]. Available: https://www.technologyreview.com/s/526491/agricultural-drones/

[13] "Amazon Prime Air." [Online]. Available: http://www.amazon.com/b?node=8037720011

[14] J. Kunzmann, "The Design, Modeling, and Control of a 4-Rotor Aerial Robot," Masters Thesis, Rose-Hulman Institute of Technology, Aug. 2011.

[15] P. R. Olejnik, "Using Regression to Evaluate Project Results, part 1," Feb. 2015. [Online]. Available: http://blog.minitab.com/blog/statistics-in-the-field/using-regression-to-evaluate-project-results%2C-part-1

[16] ——, "Using Regression to Evaluate Project Results, part 2," Feb. 2015. [Online]. Available: http://blog.minitab.com/blog/statistics-in-the-field/using-regression-to-evaluate-project-results%2C-part-2

[17] "Manual for A-51 Inverted Pendulum Accessory," 2002.

[18] C. Predoi, L. Coereanu, C. Axente, and D. I. Suteu, "Aspects Regarding Software Simulation of a Quadcopter Dynamics," *Applied Mechanics and Materials*, vol. 332, pp. 62–67, Jul. 2013. [Online]. Available: http://www.scientific.net/AMM.332.62

[19] K. Turkoglu and A. Ji, "Development of a Low-Cost Experimental Quadcopter Testbed using an Arduino controller for Video Surveillance." American Institute of Aeronautics and Astronautics, Jan. 2015. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2015-0716

[20] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, 2011.

[21] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a quad-rotor robot," in *Proceedings Australasian Conference on Robotics and Automation 2006.* Auckland, New Zealand: Australian Robotics and Automation Association Inc., 2006. [Online]. Available: http://www.araa.asn.au/acra/acra2006/papers/paper_5_26.pdf

[22] R. W. Beard, "Quadrotor dynamics and control," Tech. Rep., 2008.

[23] A. Gibiansky, "Quadcopter Dynamics, Simulation, and Control." [Online]. Available: http://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf

[24] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, p. 35, 1960. [Online]. Available: http://FluidsEngineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402

[25] H. Kushner, "Dynamical equations for optimal nonlinear filtering," *Journal of Differential Equations*, vol. 3, no. 2, pp. 179–190, Apr. 1967. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/002203966790023X

[26] S. Julier and J. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, Mar. 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1271397

[27] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, Sep. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6279585

[28] M. Leccadito, T. M. Bakker, R. Niu, and R. H. Klenke, "A Kalman Filter Based Attitude Heading Reference System Using a Low Cost Inertial Measurement Unit." American Institute of Aeronautics and Astronautics, Jan. 2015. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2015-0604

[29] F. L. Lewis, L. Xie, and D. Popa, *Optimal and Robust Estimation With an Introduction to Stochastic Control Theory, Second Edition.* Hoboken: CRC Press, 2007. [Online]. Available: http://public.eblib.com/choice/PublicFullRecord.aspx?p=1446876

[30] B. N. Datta, *Numerical methods for linear control systems: design and analysis.* Amsterdam ; Boston: Elsevier Academic Press, 2004.

[31] A. P. Sage and C. C. White, *Optimum systems control*, 2nd ed. Englewood Cliffs, N.J: Prentice-Hall, 1977.

[32] R. C. Nelson, "Beagleboard:BeagleBoneBlack Debian." [Online]. Available: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

[33] "Adafruit-beaglebone-io-python." [Online]. Available: https://github.com/adafruit/adafruit-beaglebone-io-python

[34] "ISC DHCP." [Online]. Available: https://www.isc.org/downloads/dhcp/

[35] J. Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1x/WPA/WPA2/EAP/RADIUS Authenticator." [Online]. Available: https://w1.fi/hostapd/

[36] "systemd System and Service Manager." [Online]. Available: http://www.freedesktop.org/wiki/Software/systemd/

[37] J. C. Gamazo-Real, E. Vzquez-Snchez, and J. Gmez-Gil, "Position and Speed Control of Brushless DC Motors Using Sensorless Techniques and Application Trends," *Sensors*, vol. 10, no. 7, pp. 6901–6947, Jul. 2010. [Online]. Available: http://www.mdpi.com/1424-8220/10/7/6901/

[38] J. Shao, "Direct back EMF detection method for sensorless brushless DC (BLDC) motor drives," Masters Thesis, Virginia Polytechnic Institute, 2003.