Quadcopter Plant Model and Control System Development With MATLAB/Simulink Implementation

By

Nicholas Ferry

A Research Paper Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

PROF_____
(Dr. Daniel Kaputa, Research Advisor)

PROF_____
(Dr. Sohail A. Dianat, Department Head)

DEPARMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

December 2017

Contents

Table of Figures

List of Tables

Nomenclature

UAV:	Unmanned Aerial Vehicle
UAS:	Unmanned Aerial System
ESC:	Electronic Speed Controller
PWM:	Pulse Width Modulation
RPM:	Rotations Per Minute
Hz:	Hertz
MHz:	Megahertz
GHz:	Gigahertz
PID:	Proportional, Integral, Derivative
LQR:	Linear Quadratic Regulator
DC:	Direct Current
FRF:	Frequency Response Function
ID:	Identification
IMU:	Inertial Measurement Unit
SOC:	State of Charge
ADC:	Analog to Digital Converter
DAC:	Digital to Analog Converter
ZOH:	Zero Order Hold
LiPo:	Lithium Polymer
NED:	North-East-Down
LLF:	Local-Level Frame
GPS:	Global Positioning System
BLDC:	Brushless DC

Chapter 1 - Introduction

Quadcopters are an emerging technology gaining in popularity. Most users purchase quadcopters fully assembled with everything that's needed to fly. One of the more interesting aspects of quadcopters that may make flying them even more amazing is understanding how they work. Believe it or not, users have the ability to install their own control system in order to have the quadcopter fly exactly how they want, depending on the hardware of course. Many users are limited by the controller that comes preinstalled on quadcopters. A custom control system can provide more aggressive flight, or more relaxed flight. As well as having control over desired flight, a custom control system, depending on the complexity of the system, can allow the quadcopter to have automatic functions. These functions could include flips, or inverted flight. However, to do this, a plant model of the quadcopter, which is a system of equations that represent the dynamics of the quadcopter, is needed to simulate flight to prove the control system works prior to installing on the quadcopter. Creating a plant model and simulating it provides a prediction of what the real-world quadcopter will do. Quadcopter plant models can increase in accuracy depending on how involved the modeling process is. For example, a simple plant model could be comprised of basic equations of motion, but wouldn't be awfully accurate. A complex plant model could encompass the equations of motion along with air resistance, wind forces, motor dynamics, battery dynamics, etc. The addition of these within the plant model will increase the accuracy, however, some are difficult to model.

This idea of controlling and creating a plant model of a quadcopter is rather complex but can be broken down significantly so it can be easily understood. Control systems and plant models can be seen almost everywhere without knowing it. The 3 basic pieces required to create a model are the control system, a plant model, and sensors. An everyday example of this idea can be seen from driving a car, as seen in Figure 1.1 below.

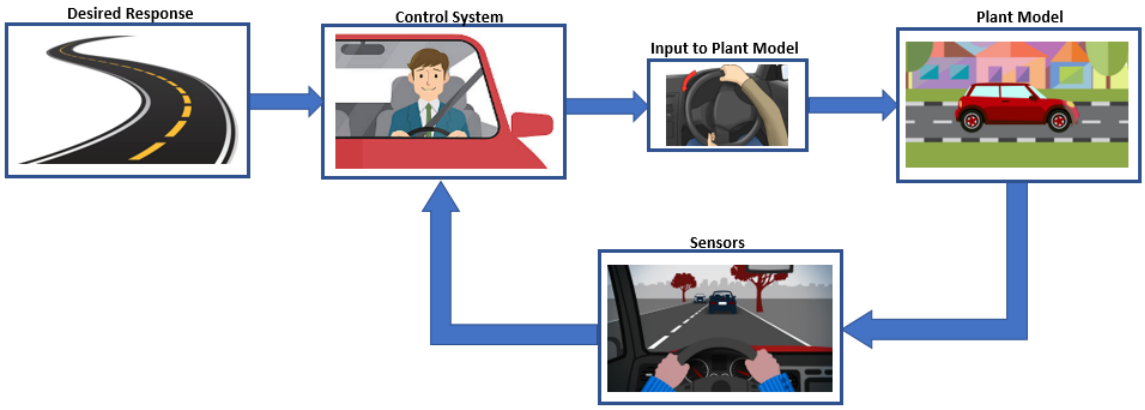


Figure 1.1-Basic driving car control scheme example

The car can be thought of as the plant model. The car has many parts that contribute to it to make it a plant model, such as the tires, engine, transmission etc. The input to the plant model is rotation of the steering wheel, and the output is the trajectory of the car. The desired response of the system is to stay within the lines. The sensors of the system are the eyes of the person driving, which view the road. The control system is the brain of the person driving, who uses the sensors to see the output of the plant. The person driving, or the control system reacts to changes in the cars trajectory along with the desired trajectory by rotating the steering wheel, which is also the input to the plant. This example is rather relatable to quadcopter. The plant model is the actual quadcopter which is comprised of motors, propellers etc. The control system is the flight controller. The sensors are accelerometers and gyroscopes, and the desired response are roll, pitch, yaw angles and throttle.

Since the control system and plant model of a quadcopter can be complex, the paper will demonstrate how to model the basic physical quadcopter, then increase the plant model's complexity and then demonstrate how the model can be implemented in MATLAB/Simulink. It also demonstrates how to build and tune controllers that are able to control a quadcopter's roll, pitch, yaw, altitude and motor dynamics. The benefits of using MATLAB/Simulink to create and implement these models come from the computing power of the software tool, along with the ease of implementation of Simulink models to hardware.

1.1 Components

Quadcopters are, in layman's terms, multirotor helicopters. Quadcopters are sometimes referred to as drones, or unmanned aerial vehicles. They are aerial vehicles that possess six degrees of freedom (X, Y, Z roll, pitch, yaw) through producing thrust in each of the rotors. The way in which quadcopters control the degrees of freedom is a large component that distinguish them from helicopters. Helicopters use complex variable pitched blades and rotor axis forces to control their path, where quadcopters simply apply thrust to specific motors. Quadcopters, are units that range in size and that can be controlled through a transmitter or can be programmed to follow a flight trajectory/path.

There are multiple components to any quadcopter that are important to understand. These components include: propellers, motors, battery, electronic speed controllers (ESC), flight controller, transmitter/receiver, and air frame.

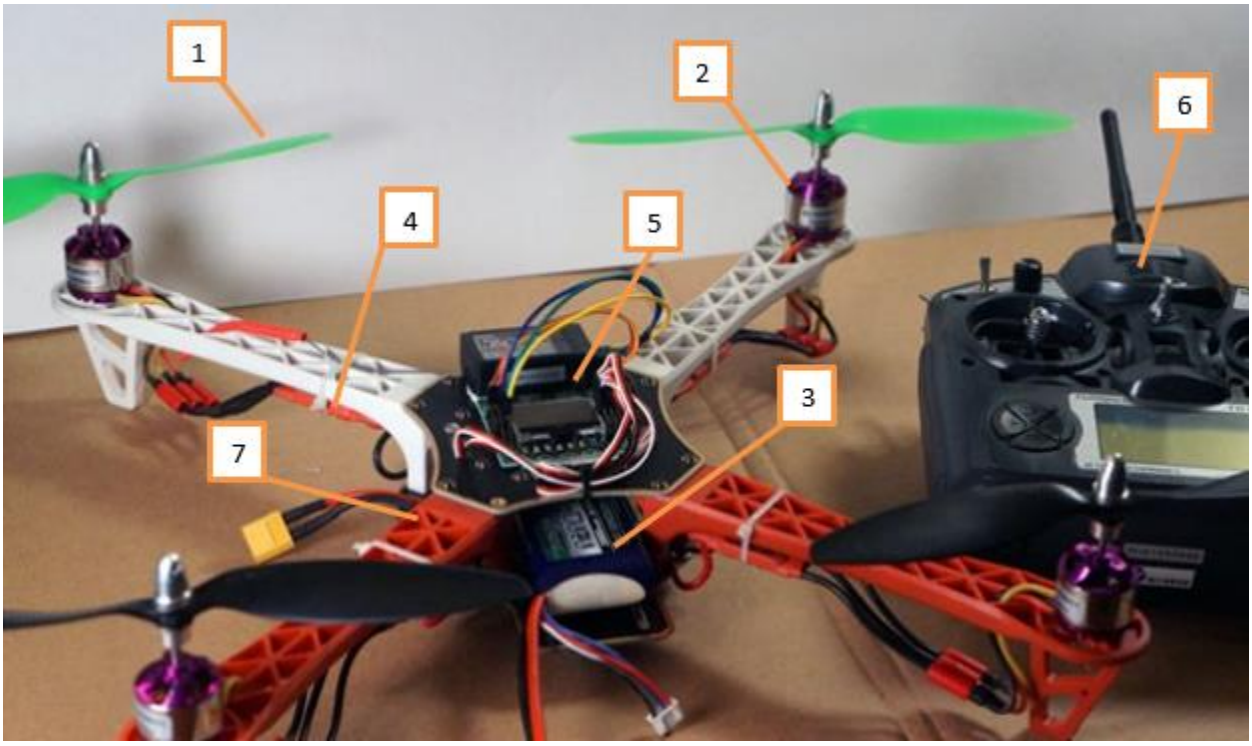


Figure 1.2-Quadcopter components [1]

1.1.1 Propellers (1)

The propellers of a quadcopter are integral components that play a large role in the quadcopter dynamics and plant model. While spinning, the propellers produce downward thrust to overcome the force of gravity allowing the quadcopter to fly. Quadcopters can use multiple types of propellers, these include push or pull propellers, along with propellers that have more than 2 blades. There are multiple variables associated with the propellers which include: propeller pitch, diameter, chord, and material. Using the pitch and diameter, important parameters of the propellers can be found such as power and thrust. A method of finding an approximate amount of thrust from a propeller can be found in Section A.2 of the appendix.

1.1.2 Motor (2)

There are two main types of motors used in quadcopters, brushed and brushless. The motors are an important part of the quadcopters plant model as they are the component that spin the propellers which produce thrust. The architecture of the two types of motors can be seen in Figure 1.3. Both types of motors have the same general components, a permanent magnet and an electromagnet. In brushed motors, the electromagnet is surrounded by curved permanent magnets and in brushless motors, electromagnets surround a permanent magnet. In both cases the physics driving the rotation in these motors comes from opposite poles and magnets attracting, and like poles and magnets repelling which induces rotation. The use of either of these motor types comes with pros and cons. Brushed DC motors do not require an electronic speed controller, but breakdown faster. The brushless DC motors require the electronic speed controller but don't break down as fast.

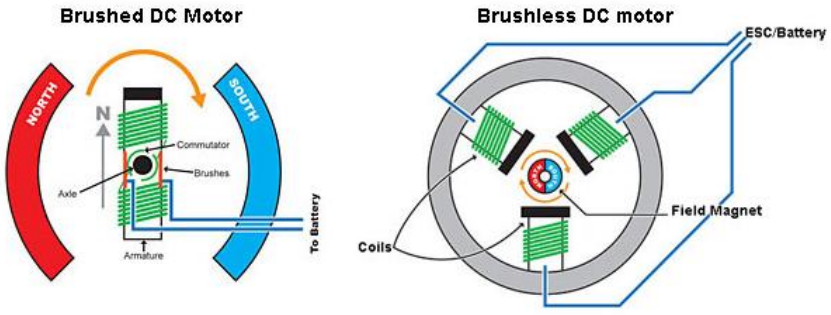


Figure 1.3-Brushed and brushless motor [2]

1.1.3 Battery (3)

Batteries are what give life to any quadcopter, as they power the flight controller, ESC, receiver, and motors. The battery can be modeled in the plant model to increase the quadcopter plant model's accuracy. There are a few typical types of batteries, nowadays: Alkaline, Lithium-ion and Lithium polymer. The battery is usually the majority of the weight of a quadcopter.

1.1.4 Electronic Speed Controller (ESC) (4)

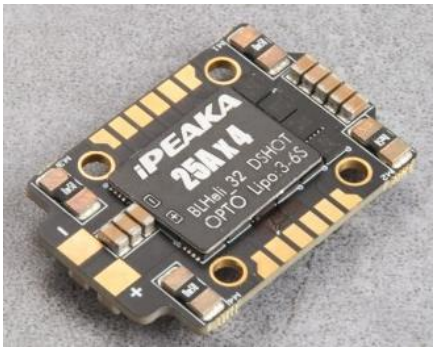


Figure 1.4-iPeaka ESC

When using a brushless motor, an ESC is required to control the motor. An ESC is an electronic circuit board that provides control and three phase AC power to the motor. These boards usually contain an 8-bit microcontroller that runs embedded software for the control. Many quadcopters use 4 ESC's (one for each motor) which are 1 channel ESC's. However, ESCs can come in 4 channel packages, which means one ESC can control power to each of the 4 motors.

The ESC reads input signals from the motor and flight controller. They use the back-EMF force from the motor to determine rotor position. The ESC outputs the voltage that will control the polarity of the electromagnetic coils in the motor. By controlling the timing of the voltage that is applied to the motor, speed is controlled [3]. Ultimately, the ESC converts input PWM signals from the flight controller and drives the brushless DC motors with the correct power. The ESC can, in ways, can be encompassed in the quadcopter plant model because it controls the motor's speed. Since it controls the motors speed it can also be considered a controller, and because it senses the motor's back EMF it can be considered a sensor as well.

1.1.5 Flight Controller (5)

Figure 1.5-EMAX Skyline32 flight controller

The flight controller is essentially the brain of the quadcopter. They are configurable and programmable allowing for adjustments based on varying quadcopter configurations. The hardware of a flight controller consists of a complex circuit board with built in sensors such as accelerometers and gyroscopes that can detect accelerations and orientation changes. With these sensors, the flight controller can stabilize the quadcopter by controlling the RPM of the motors. The flight controller also receives signals from the user indicating the desired trajectory/function. The flight controller takes these commands, uses sensor data, and adjusts the quadcopters motors speed such that the quadcopter achieves the user's desired trajectory and stability [4]. The flight controller encompasses all 3 portions that create a system, controller, plant and sensors. Since the flight controller is on the quadcopter it contributes to the quadcopters plant model's parameters. It controls the quadcopters throttle, roll, pitch, and yaw, and it senses the quadcopters acceleration and angles.

1.1.6 Transmitter/Receiver (6)

Figure 1.6-Spektrum receiver and transmitter

The transmitter and receiver are what allow the user to communicate with the quadcopter. The transmitter is a device that allows the user to control the quadcopter wirelessly. The receiver, which contributes to the plant model's parameters, receives the commands from the transmitter and relays the commands to the flight controller. The frequency associated with the transmitter & receiver is generally in the MHz or GHz band. A popular quadcopter frequency is 2.4GHz, but can be lower if longer range communication is required [5].

There are multiple channels associated with the transmitter & receiver. Each channel sends different commands from the transmitter to the receiver. For example, one channel can control the quadcopter throttle, another can control roll, another can control pitch, etc. The minimum number of channels required to control a quadcopter is 4 (throttle, roll, pitch, yaw).

1.1.7 Air Frame (7)

Figure 1.7-Basic quadcopter airframe

The air frame of a quadcopter generally has an "x" or "+" shape. One arm for each motor and the central area for flight controller, receiver, and battery mounting. It is important to have a light weight, sturdy air frame to have desirable flight characteristics and be resilient to crashes. One of the more popular materials for quadcopter air frames is carbon fiber, which is light weight and durable. Carbon fiber, however, blocks radio signals which is the downfall of using that material for an air frame. The airframe is an integral portion of the quadcopters plant model as the plant model relies heavily on physical characteristics of the quadcopter.

1.2 Modeling Overview

The medium in which the quadcopter plant model and controller will be developed is MATLAB/Simulink. MATLAB is a numerical computing software which interfaces with Simulink. Simulink is a software tool that allows a user to build systems using block models. When modeling a quadcopter, a MATLAB script is used to store constant variables and simulation data that can then be plotted. Simulink will be used to build the physical plant model and controller using multiple block models such as integral, product, summing, and step blocks. MATLAB/Simulink has the ability to convert Simulink and Stateflow models to C and C++ code. This is convenient because many flight controllers use C or C++ code for their control systems.

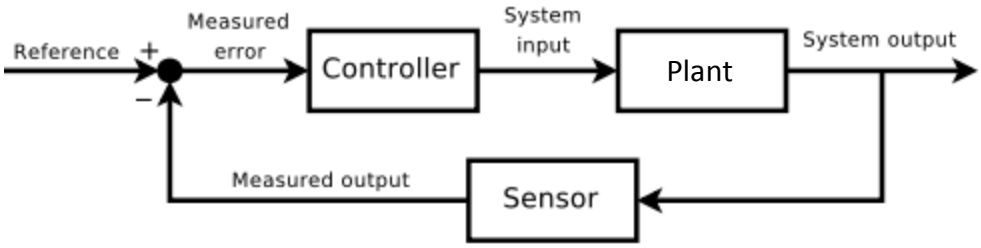


Figure 1.8-General controller, plant, sensor scheme architecture [6]

A plant model is either an experimentally generated transfer function of a physical system or a set of equations that describe a physical system. In order to simulate, and create a control a system a plant model is generally needed. With the plant model, control systems can quickly be tuned and tested. The plant model of a quadcopter takes propeller angular velocity for each rotor as *system inputs* and the *system outputs* are the states that represent what the quadcopter physically does. For the model in this paper, a set of physics equations are used to generate the quadcopter plant model. The components of the quadcopter that contribute to the plant model can be seen in Table 1.1 below. It is worthy to note that although the physics do not change, there are multiple ways of mathematically modeling a quadcopter.

Table 1.1-Component to model classification

Component	Plant Model	Controller	Sensor
Propeller	X		
Motors	X		
Battery	X		
ESC	X	X	X
Flight Controller	X	X	X
Transmitter			
Receiver	X		
Air Frame	X		

A controller is a system that takes a *measured error* signal, applies gains, and outputs a *system input* to the plant model such that the plant model will output the desired response. There are many different types of control methods, but for control in this paper, Proportional, Integral and Derivative (PID) controllers are used. These are very popular controllers and are relatively simple compared to other control methods. In essence, the goal of these controllers is to drive an input error signal (desired setpoint subtracted from the current state) to zero. In the PID controller case this is done by multiplying the error signal by a general

gain (Proportional portion), generating a gain through summing the error signal (Integral portion), and creating a gain by tracking the slope of the error signal (Derivative portion). Ultimately driving the error to zero and the output to the desired input. The PID controller will be covered in depth in Section 4.2.

The general control scheme has the controller feeding to the plant model and feedback from the plant to generate a *measured error* signal. The feedback in simulation is usually from the plant model, however, in the actual physical system, feedback, or *measured output* is generated from sensors on the quadcopter. The main signals the quadcopter needs to control are from accelerometers and gyroscope sensors. Accelerometers measure the acceleration in each direction, and gyroscopes measure the angular velocity, or how fast an object is rotating. The general controls scheme can be seen in Figure 1.8 above.

1.3 Building a Model

As seen in Figure 1.8 the plant model receives inputs and produces an output. The plant model is made from equations or experimental data which use the inputs, states and parameters to create the outputs. Within the plant are states which represent dynamics of the plant model. Associated with the states are the state's reference frame. This is the states starting point, for example on an X, Y plane, the 0,0 coordinate is the reference point. It's important to discern the positive and negative directions of the reference frame also. Depending on the location, and positive directions of a reference frame, the same point location can be represented different ways. An example of this can be seen in Figure 1.9 below.

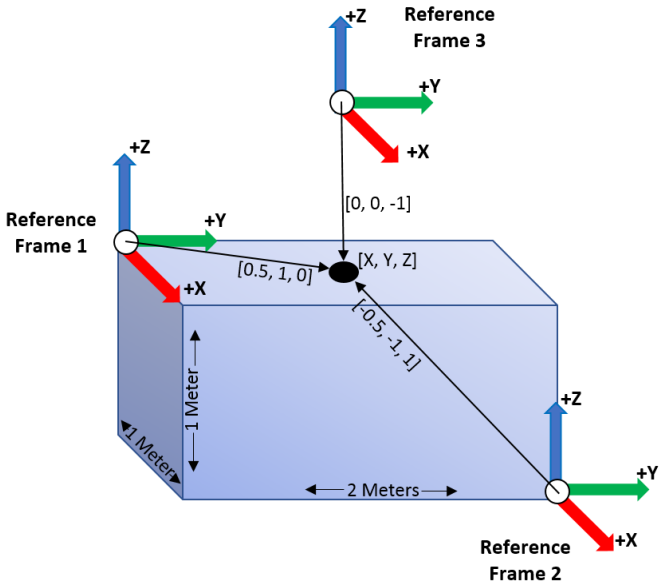


Figure 1.9-Reference frame depiction

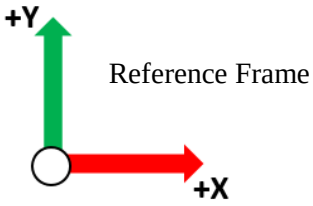
Parameters are constants that are used in the plant model equations. To build a basic plant model a system needs to be defined. Inputs and outputs then need to be defined along with the plant model's states, parameters and equations. An example of this can be seen below.

System: A block at rest on a frictionless table in a vacuum, then a force is applied

Input: Force applied to the block, 1N applied for 1 second in the positive X direction

Output: Velocity$\left(\frac{m}{s}\right)$

States: Acceleration$(\frac{m}{s^2})$, Velocity$\left(\frac{m}{s}\right)$



Reference Frame

Reference: (0,0) on a X, Y plane

Parameter: Mass(kg)

Equations: $Force(N) = Mass(kg) * Acceleration\ (\frac{m}{s^2}), Velocity\left(\frac{m}{s}\right) = \int Acceleration\ (\frac{m}{s^2})$

In this example a force is applied to a block with no other forces acting on it. The goal of the plant model is to represent the output, which in this case is velocity. This plant model can be created in Simulink where the inputs and outputs can be plotted. This example can be seen in Figures 1.10-1.12 below.

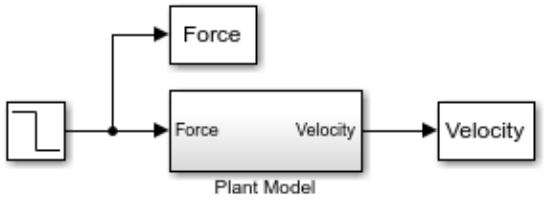


Figure 1.10-Basic plant model

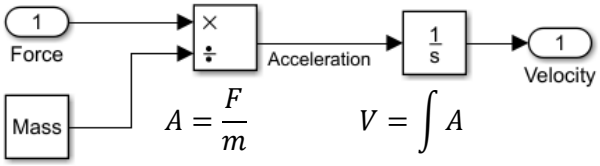


Figure 1.12-Basic plant model equations

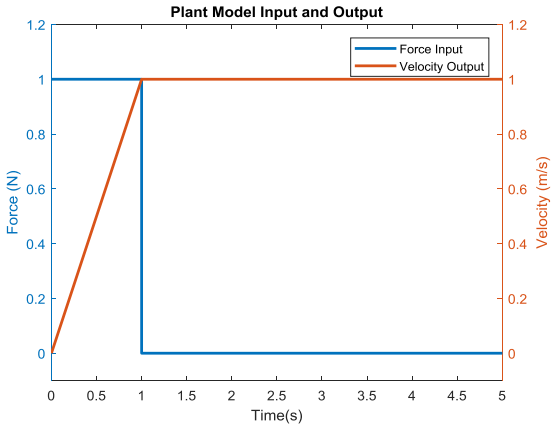


Figure 1.11-Basic plant model input and output

Chapter 2 - Building the Plant Model

To realize the quadcopter plant model the block model seen in Figure 2.1 below can be used. The plant model is a system of equations that generate outputs based upon inputs, parameters, and system states. Within the broad plant model of the quadcopter there are separate plant models such as the body dynamics, motor, and battery models. There is somewhat of a flow of variables within the plant model. The system starts with the inputs, which in this case are motor angular velocity commands. With these commands, states within the plant model change such as altitude, plant model angular velocity, and axial velocity. Parameters are constants that effect the states, such as the friction coefficient of the motors. The outputs of the plant are based on the previous states, parameters, and inputs. Lastly, it is important to note that the states dependent on the frame of reference, or where the zero point is.

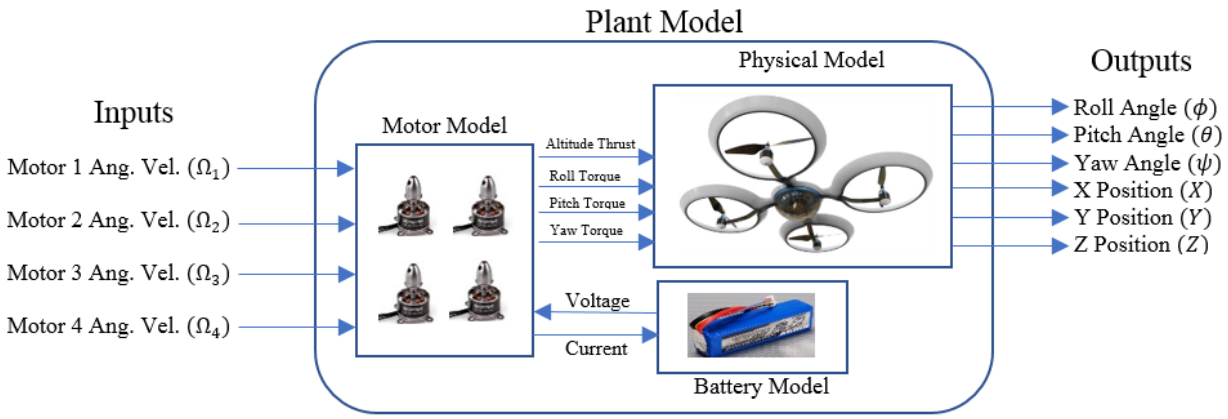


Figure 2.1-Plant model realization

The ideal plant model would simply be the physical model, assuming instant thrust, no disturbances, and unlimited state values. A real-world model would include motor and battery dynamics along with state limitations.

2.1 Quadcopter Plant Model

2.1.1 Plant Model States

Quadcopters inherently have 6 degrees of freedom. This means the device can travel in the X, Y and Z directions, as well as rotate in roll, pitch and yaw directions. These terms are generally related to most aircraft, subs, and robotics as not many other platforms are able to control all 6 degrees. The roll and pitch are what allow the quadcopter to move in the X and Y directions. Roll can be defined as the rotation around the X axis, pitch can be defined as the rotation around the Y axis and yaw can be defined as the rotation around the Z axis. With this, the axis of rotation for the quadcopter and state variables can be defined. State variables are physical dynamics of the quadcopter that include angles and positions. Many of the states are dependent on other states and user input. The reference frame of the quadcopter will be local NED, which is why the Z axis is pointed down. This will be explained in later sections.

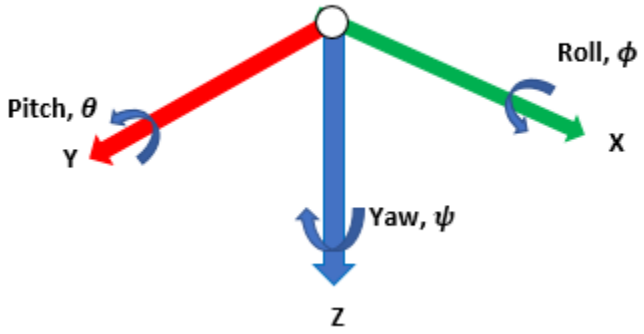


Figure 2.2-Model states

A fully observable state is one that can directly be measured using sensors. For example, imagine a police officer measuring a car's speed with a radar. The velocity of the car would be the fully observable state and the radar would be the sensor. Quadcopters have many sensors that include accelerometers, gyroscopes, optical, etc. The quadcopters states and observability can be seen in Table 2.1 below.

Table 2.1-Quadcopter states

Symbol	Description	Unit	Observability
θ	Pitch Euler Angle	rad	Stereo Vision
$\dot{\theta}$	Pitch Euler Angular Velocity	rad/s	Gyroscope
$\ddot{\theta}$	Pitch Euler Angular Accel.	rad/s^2	-
ϕ	Roll Euler Angle	rad	Stereo Vision
$\dot{\phi}$	Roll Euler Angular Velocity	rad/s	Gyroscope
$\ddot{\phi}$	Roll Euler Angular Accel.	rad/s^2	-
ψ	Yaw Euler Angle	rad	Stereo Vision
$\dot{\psi}$	Yaw Euler Angular Velocity	rad/s	Gyroscope
$\ddot{\psi}$	Yaw Euler Angular Accel.	rad/s^2	-
X	Position in X	m	Stereo Vision
$\dot{X}$	Velocity in X	m/s	-
$\ddot{X}$	Accel. in X	m/s^2	Accelerometer
Y	Position in Y	m	Stereo Vision
$\dot{Y}$	Velocity in Y	m/s	-
$\ddot{Y}$	Accel. in Y	m/s^2	Accelerometer
Z	Position in Z	m	Stereo Vision
$\dot{Z}$	Velocity in Z	m/s	-
$\ddot{Z}$	Accel. in Z	m/s^2	Accelerometer
Ω_1	Motor 1 Angular Velocity	rad/s	Voltage/Current/Optical
Ω_2	Motor 2 Angular Velocity	rad/s	Voltage/Current/Optical
Ω_3	Motor 3 Angular Velocity	rad/s	Voltage/Current/Optical
Ω_4	Motor 4 Angular Velocity	rad/s	Voltage/Current/Optical
θ-Theta (/they-tuh/), ϕ-Phi (/fi/), ψ-Psi (/sai/), Ω-Omega (/oh-mega/)			

2.1.2 Plant Model Reference Frame

The frame of reference and euler angles now need to be defined. Euler angles are 3 angles that are used to describe the orientation of a rigid body with respect to a fixed body coordinate system [10]. Figure 2.3 below can help depict the euler angles.

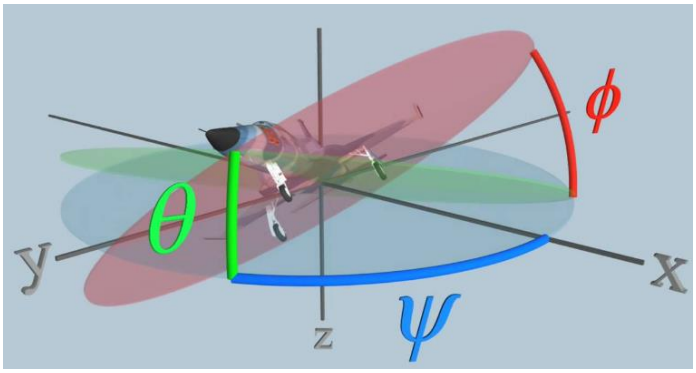


Figure 2.3-Euler angles explanation [10]

The body frame of reference for the quadcopter is similar to Figure 2.3 above. All of the angles will begin at zero and have a range from -pi to +pi. This concept can be seen in Figure 2.4 below.

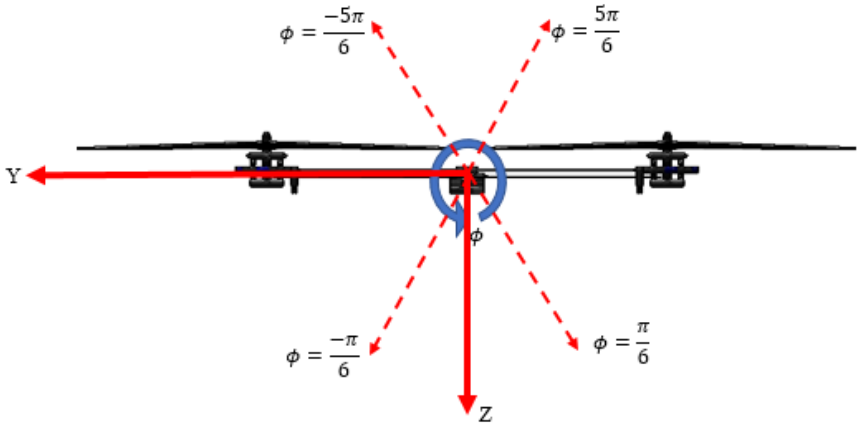


Figure 2.4-Phi angle limits

The position reference for the quadcopter will be the local NED reference frame. This means the axial accelerations, velocities, and position are relative to a fixed point. The local NED frame is used to represent a vehicles attitude and velocity when on or near the surface of the earth. In this frame, the X axis is pointed north, Y axis is pointed east, and the Z axis is pointed downwards normal to the X and Y plane. This frame can be seen in Figure 2.5 below. Although the axial acceleration, velocity and position are relative to the

local NED frame, the euler angles are relative to the body frame of the quadcopter. In the local NED coordinate system, with positive roll angles the quadcopter travels east, with positive pitching angles the quadcopter travels south (both assuming a zero yaw angle), and with positive yaw angles the quadcopter rotates clockwise.

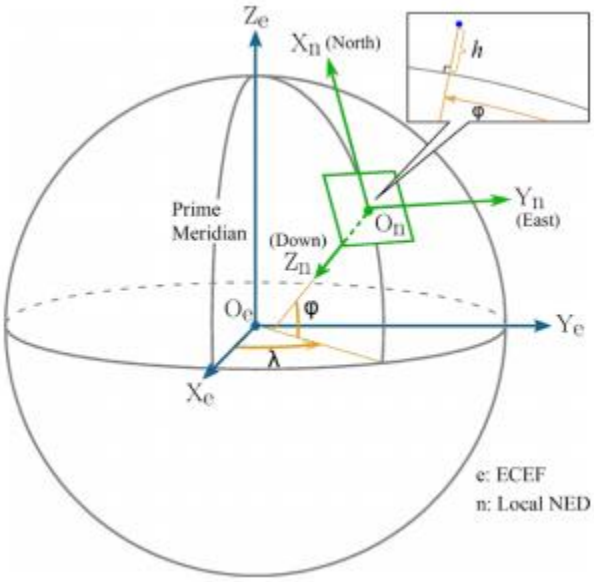


Figure 2.5-Local NED frame depiction

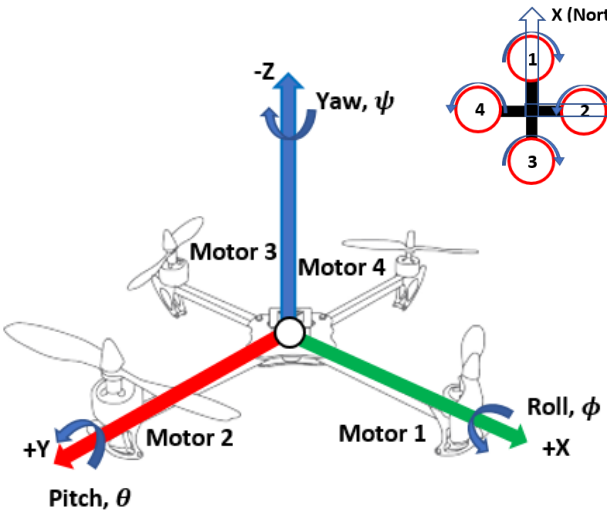


Figure 2.6-Quadcopter '+' orientation configuration

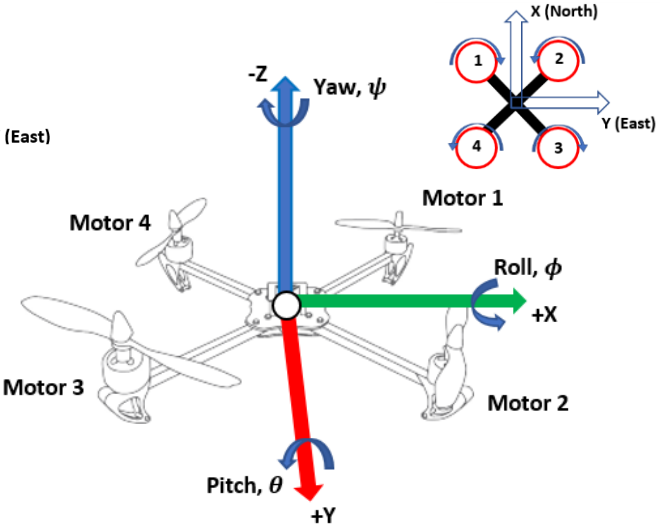


Figure 2.7-Quadcopter 'x' orientation configuration

The body frame axis of rotation can be defined in either of two forms, one is the 'x' formation and the other is the '+' formation. The difference between the two comes from which way the X and Y direction is, as well as what motors are used to change direction. The axial orientation, states, and units can be seen in Figures 2.6 and 2.7 above.

2.1.3 Plant Model Parameters

Before generating the plant model equations of motion, parameters need to be defined. It is important to know the difference between states and parameters. States are dynamics of the quadcopter, where parameters are constants associated with the quadcopter. The first parameters to be defined are the moments of inertia in each axis. The moment of inertia is the amount of torque needed for a desired angular acceleration around a rotation axis. These relies heavily on the physical design and components of the quadcopter. These are generated through various methods which mostly include estimates from lengths/diameters and masses of the quadcopter. A detailed explanation of finding moments of inertia can be found in Section A.1 of the appendix. The rotor inertia, which is the total rotational moment of inertia around the propeller axis is another parameter used in the equations of motion. The rotor inertia equation can also be found in Section A.1 of the appendix. The distance from the rotor axis to the center of the quadcopter also can be defined, along with the thrust and drag coefficients. These can be seen in Table 2.2 below.

Table 2.2-Quadcopter parameters

Symbol	Description	Unit
I_{xx}, I_{yy}, I_{zz}	Inertia moments	$Kg\ m^2$
J_r	Rotor inertia	$Kg\ m^2$
l	Rotor axis to copter center distance	m
b	Thrust coefficient	N/s^2
d	Drag coefficient	Nm/s^2

To experimentally find the thrust and drag coefficients used in the input equations, thrust, torque, and RPM data needs to be acquired. To do this, a thrust stand and dynamometer is needed, which measures thrust, torque, voltage, current, etc. Once test data is acquired using the test equipment and a data logging system, thrust, torque, and RPM data needs to be examined. With this, Equations (2.1) and (2.2) below can be used to find the thrust and torque coefficients. The thrust and torque coefficients data vs. thrust of the dynamometer test can be plotted. An example of this can be seen below in Figure 2.8 and 2.9. Here, since its known that for the most part quadcopters are hovering, the coefficients should be found around the hovering thrust. This can be found by knowing the weight of the quadcopter, dividing the weight by how many motors are used, then map that amount to a thrust in the plot. Assuming the mass is 0.284 kg, the thrust to hover for each motor is 0.071 kg. Using the plot the thrust coefficient is roughly 1.144E-8, and the torque coefficient is roughly 9.94e-10.

$$Thrust = b\Omega^2 \tag{2.1}$$

$$Torque = d\Omega^2 \tag{2.2}$$

$$\Omega = 2\pi(\text{RPM})/60 \tag{2.3}$$

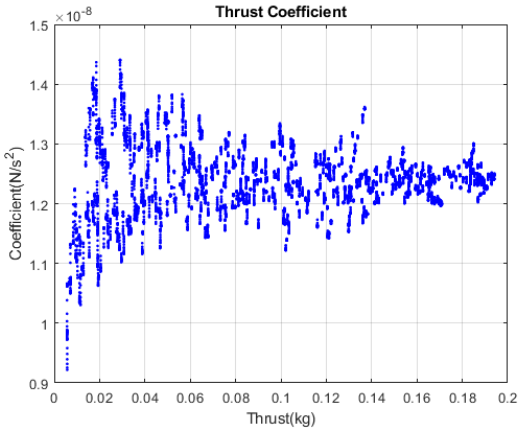


Figure 2.8-Thrust coefficient vs. thrust

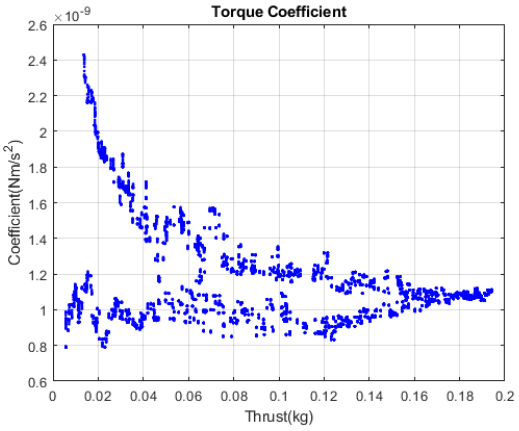


Figure 2.9-Torque coefficient vs. thrust

It is also useful to know what the thrust curve will resemble at higher RPM's than are tested, or how accurate Equation (2.1) is. Using Equation (2.1) and (2.3) above, as well as the experimentally generated thrust

coefficient, the thrust data and thrust curve can be plotted together. Figure 2.10 shows how accurately Equation (2.1) represents the thrust.

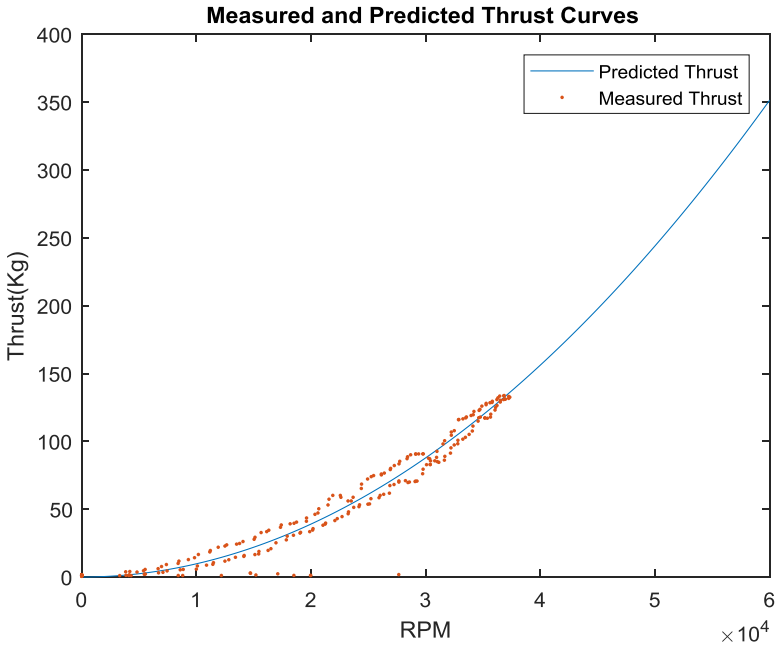


Figure 2.10-Measured and predicted thrust curves

The next set of parameters have to do with the quadcopter and environment. In the real world, quadcopters experience aerodynamical effects, namely from air resistance and wind. Air resistance can be thought of as a drag on the quadcopter, opposing the motion of the quadcopter. This is what makes quadcopters slow down when no actuator force is acting on them. To envision air resistance, one can simply drop a piece of paper, and pencil from the same height. Air resistance is what makes the piece of paper take longer to hit the ground than the pencil. Since air resistance effects the quadcopter in all 3 axial directions, a coefficient is needed for each. Measuring the air resistance is extremely complex, so in most cases estimates suffice. The values used for the plant model were 0.1 kg/s for the resistance in each axis. The parameters can be seen in Table 2.3 below.

Table 2.3-Enviornmental parameters

Symbol	Description	Unit
A_x, A_y, A_z	Air resistance in each axis	kg/s

2.1.4 Plant Model Equations

Before the quadcopter is modeled in MATLAB/Simulink the mathematical equations need to be generated. To do this a few assumptions are made [7].

- The drone is a rigid structure
- The air frame and components are symmetric
- The center of gravity and air frame origin coincide
- Thrust and drag are proportional to the square of propeller speed
- There are no external disturbances to the quadcopter such as wind, temperature, etc.

Next, the physical effects on the quadcopter need to be considered. These include aerodynamic effects from the propellers, inertial counter-torques from changes in propeller speed, gravity, and gyroscopic effects from changes in body orientation. These all play a role in the equations of motion of the quadcopter.

The internal plant model equations of the system can now be described. There are 4 main thrust equations, vertical thrust, roll thrust, pitch thrust, and yaw thrust. With the '+' coordinate system seen in Figure 2.6 above, these can be defined by applying thrust to each of the motors. The equations are defined in Table 2.4 below.

Table 2.4- Input Equations in '+' orientation

Input	Thrust Equation	Description	
$U1_+$	$b(\Omega_1{}^2 + \Omega_2{}^2 + \Omega_3{}^2 + \Omega_4{}^2)$	General Thrust	(2.4)
$U2_+$	$b(-\Omega_2{}^2 + \Omega_4{}^2)$	Roll Thrust	(2.5)
$U3_+$	$b(\Omega_1{}^2 - \Omega_3{}^2)$	Pitch Thrust	(2.6)
$U4_+$	$d(\Omega_1{}^2 - \Omega_2{}^2 + \Omega_3{}^2 - \Omega_4{}^2)$	Yaw Thrust	(2.7)

$U1_+$ applies a general thrust to each of the motors to increase the altitude. $U2_+$ changes thrust between motor 2 and motor 4 to roll the quadcopter. $U3_+$ changes thrust between motor 3 and motor 1 to pitch the

quadcopter. $U4_+$changes the thrust to motors 2 and 4 and equally changes the thrust to motors 1 and 3 to yaw the quadcopter. Figure 2.11 below illustrates these inputs.

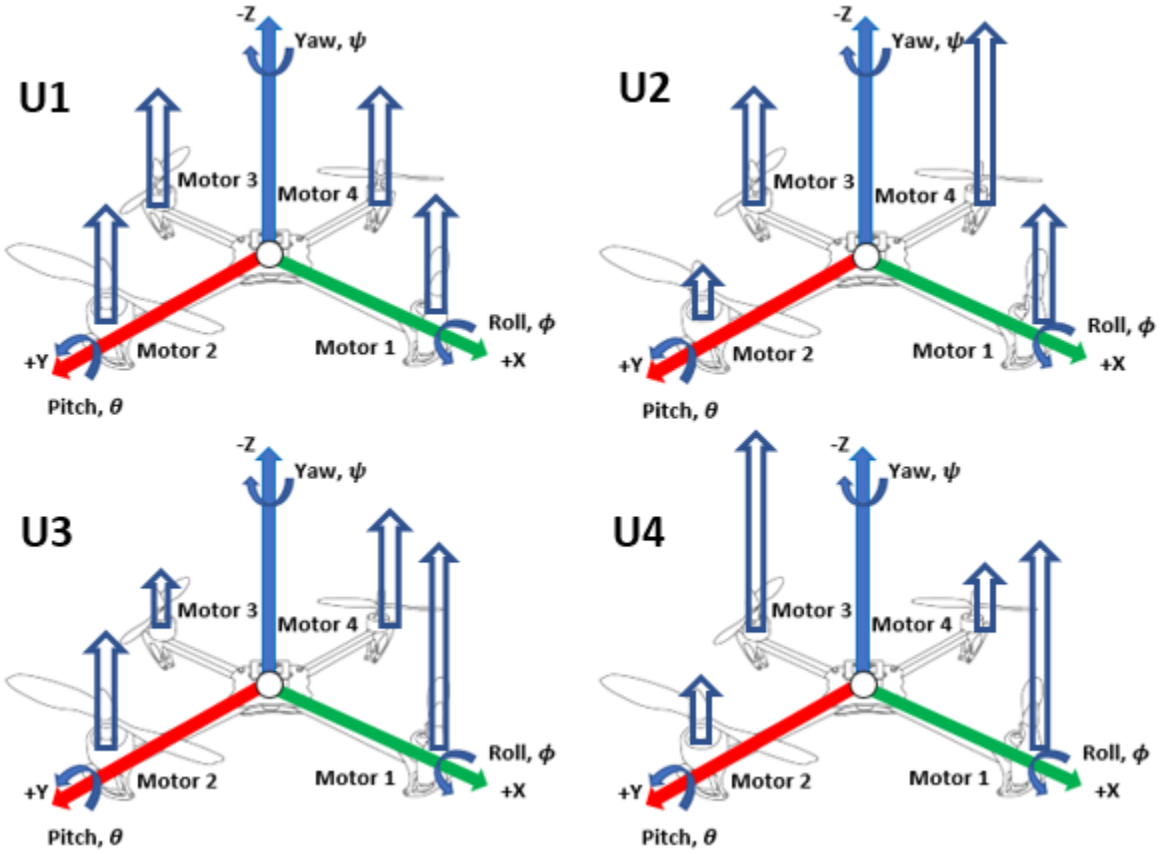


Figure 2.11-'+' orientation input commands

The quadcopter inputs can also be defined in the 'x' orientation [9]. These input equations can be seen in Table 2.5 below.

Table 2.5-Input equations in the 'x' orientation

Input	Thrust Equation	Description	
$U1_x$	$b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2)$	General Thrust	(2.8)
$U2_x$	$b \sin\left(\frac{pi}{4}\right)({\Omega_1}^2 - {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2)$	Roll Thrust	(2.9)
$U3_x$	$b \sin\left(\frac{pi}{4}\right)({\Omega_1}^2 + {\Omega_2}^2 - {\Omega_3}^2 - {\Omega_4}^2)$	Pitch Thrust	(2.10)
$U4_x$	$d({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2)$	Yaw Thrust	(2.11)

$U1_x$ applies a general thrust between each of the motors to increase the altitude. $U2_x$ changes thrust between all the motors to roll the quadcopter. $U3_x$ changes thrust between all the motors to pitch the quadcopter. $U4_x$ changes thrust to motors 2 and 4 and motors 1 and 3 to yaw the quadcopter. Figure 2.12 below illustrates these inputs.

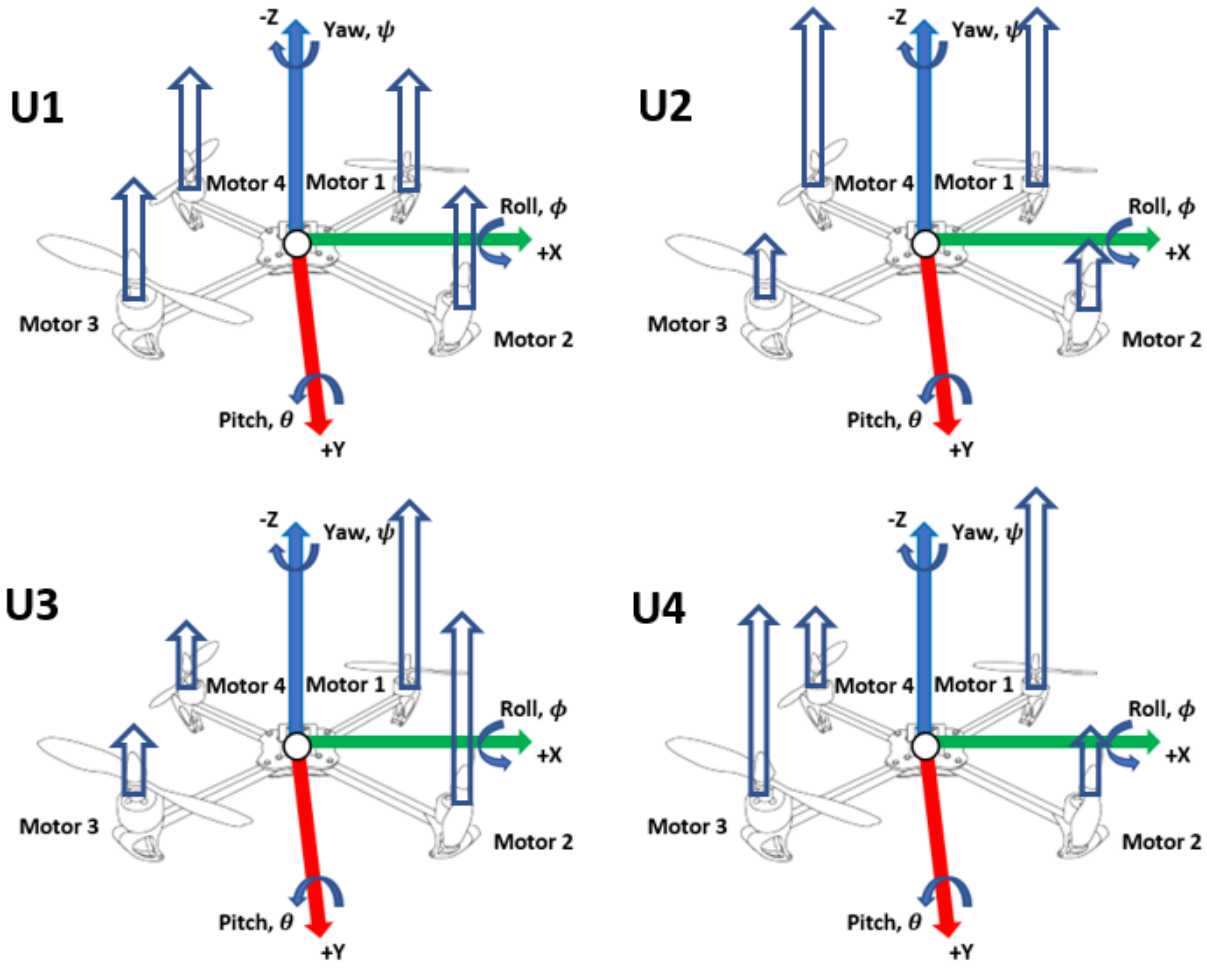


Figure 2.12-'x' orientation input commands

When using a transmitter, the joysticks represent $U1, U2, U3\ and\ U4$. The left joystick is used for throttle and yaw. The right joystick is used for pitch and roll. These can be seen in Figure 2.13 below.

Figure 2.13-Transmitter inputs

The overall angular velocity of the all the propellers can also be defined. This will be used in angular acceleration equations. The positives and negatives in Equation (2.12) are used because of the direction the propeller is spinning. In most cases the opposite side motors are spinning in the same direction (motor 1 and 3 in the same direction, and 2 and 4 in the same direction).

$$\Omega_r = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4 \tag{2.12}$$

Now equations of motion can now be defined. The origin of the equations of motion can be seen in Equations (2.13) - (2.15).

$$Force = b(\Omega_n)^2 = (Thrust\ Coeff) * (Angular\ Velocity)^2 \tag{2.13}$$

$$Force = ma = (Mass) * (Acceleration) \tag{2.14}$$

$$Torque = I\alpha = (Moment\ of\ Inertia) * (Angular\ Acceleration) \tag{2.15}$$

Starting with the moments of a quadcopter, which include, rolling moments, pitching moments, and yawing moments, each of these moments consist of body gyro effects, propeller gyro effects, and actuator action.

- Body gyroscopic effects - changes in the quadcopters orientation
- Propeller gyroscopic effects - propeller rotation coupled with quadcopter frame orientation
- Actuation action - forces produced by the rotors

These moments respectively sum to form the angular acceleration equations which can be seen below. To describe these equations in a physical sense a use case can be made for Equation (2.17). Beginning with Figure 2.14 below depicting each part of the equation.

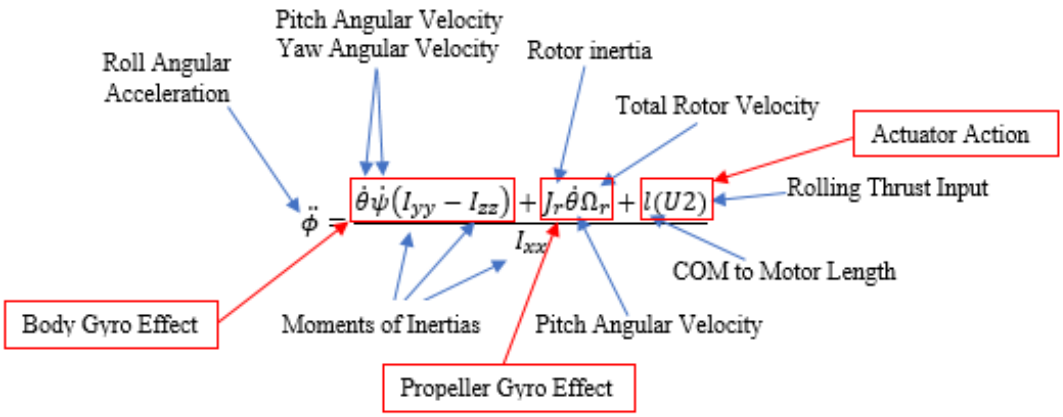


Figure 2.14-Rolling acceleration equation explained

Imagine the quadcopter is at a perfectly still hover. This means that the pitch and yaw angular velocities are zero, so the body gyro effects and the propeller gyro effects are zero. This leaves the roll actuator action $(U2)$ input, which would be zero also if the quadcopter is still, thus the roll angular acceleration is zero. Now imagine the user applies an input signal to $U2$. Since this input only applies thrust in the rolling axis the pitch and yaw angular velocities will be zero, thus the body gyro and propeller gyro effects again are zero. The actuator action is not zero due to the user input, thus the quadcopter will have a rolling angular acceleration equal to $\frac{l(U2)}{I_{xx}}$ and the quadcopter will begin to roll. It is worthy to note within the equations of motion, it doesn't matter if the system is using a '+' or 'x' orientation, thus the subscript will not be used.

Rolling Torque $$I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}\left(I_{yy} - I_{zz}\right) + J_r\dot{\theta}\Omega_r + l(U2) \quad (2.16)$$

Roll Ang. Accel. $$\ddot{\phi} = \frac{\dot{\theta}\dot{\psi}\left(I_{yy} - I_{zz}\right) + J_r\dot{\theta}\Omega_r + l(U2)}{I_{xx}} \quad (2.17)$$

Pitching Torque $$I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\Omega_r + l(U3) \quad (2.18)$$

Pitch Ang. Accel. $$\ddot{\theta} = \frac{\dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\Omega_r + l(U3)}{I_{yy}} \quad (2.19)$$

Yawing Torque $$I_{zz}\ddot{\psi} = \dot{\theta}\dot{\phi}\left(I_{xx} - I_{yy}\right) + (U4) \quad (2.20)$$

Yaw Ang. Accel. $$\ddot{\psi} = \frac{\dot{\theta}\dot{\phi}\left(I_{xx} - I_{yy}\right) + (U4)}{I_{zz}} \quad (2.21)$$

Next, the force along each fixed frame axis can be defined, which include actuators actions and aerodynamical effects. These forces can be seen in Equations (2.22) - (2.27) below. To describe these equations in a physical sense a use case can be made for Equation (2.23). Beginning with Figure 2.15 below depicting each part of the equation.

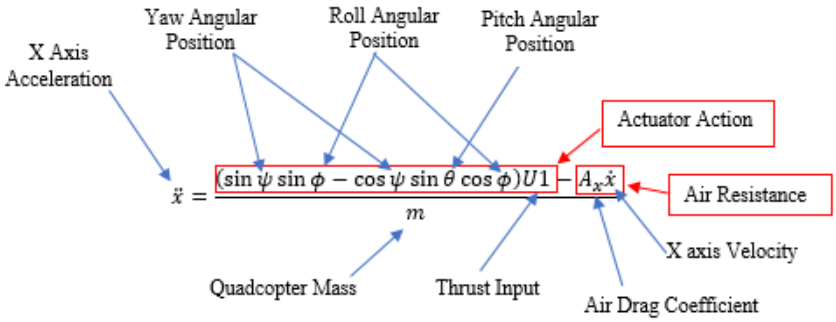


Figure 2.15-X axis acceleration equation explained

Now, imagine the quadcopter is at a perfectly still hover, neglecting air resistance, and all euler angles are at zero. Looking at the X direction acceleration equation, the actuator action portion of the equation is zero due to the sin terms, thus the X direction acceleration is also zero. Now, imagine if the roll and yaw angles are zero, but the pitch angle is $30^{\circ}\ or\ \frac{\pi}{6}rad$. This means that the acceleration in the X direction would be equal to $\frac{-\pi(U1)}{6(m)}$.

X force $$m\ddot{X} = (\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi)U1 - A_x\dot{X} \quad (2.22)$$

X Accel. $$\ddot{X} = \frac{(\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi)U1 - A_x\dot{X}}{m} \quad (2.23)$$

Y force $$m\ddot{Y} = (\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi)U1 - A_y\dot{Y} \quad (2.24)$$

Y Accel. $$\ddot{Y} = \frac{(\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi)U1 - A_y\dot{Y}}{m} \quad (2.25)$$

Z force $$m\ddot{Z} = mg - (\cos\theta\cos\phi)U1 - A_z\dot{Z} \quad (2.26)$$

Z Accel. $$\ddot{Z} = \frac{mg - (\cos\theta\cos\phi)U1 - A_z\dot{Z}}{m} \quad (2.27)$$

It is important to know the difference between the local NED reference frame and body reference frame. Equations (2.22) - (2.27) are in the local NED reference frame. To get from the body reference frame to a local NED reference frame a rotational transformation must be done. Thus, the accelerations seen in the equations above are relative to the local NED axis. If the acceleration were not multiplied by a rotational transformation matrix the X or Y acceleration would strictly be dependent on thrust and roll or pitch. An example of this can be seen below using Equations (2.22) - (2.27).

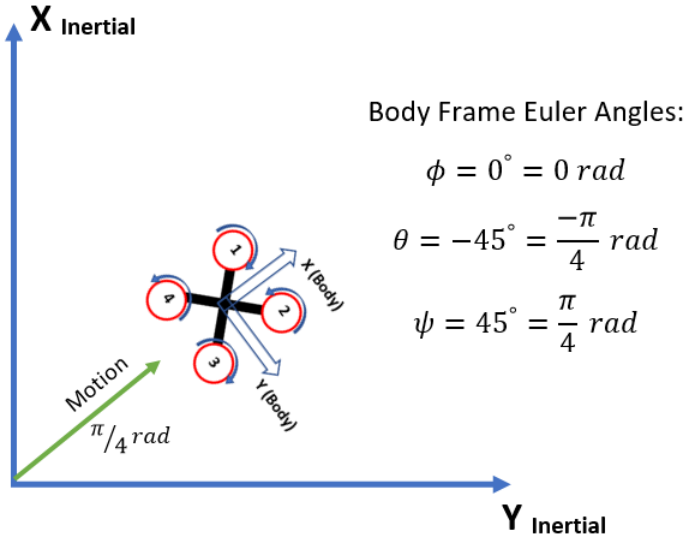


Figure 2.16-Local NED frame vs. body reference frame

To realize the local NED and body frame points of reference, imagine a fixed coordinate system with a quadcopter at some point. If the quadcopter is oriented such that it is only pitching forward to travel at a

$45^{\circ} or\ {}^{\pi}/_{4}\, rad$ angle from the fixed reference point it should have the same local NED frame X and Y accelerations. A depiction of this can be seen in Figure 2.16 above. Applying the body euler angles to Equations (2.23) and (2.25) which can be seen below, the acceleration, neglecting air resistance, is equal in the X and Y local NED frame coordinates.

$$\ddot{X} = \frac{\left(\sin\frac{\pi}{4}\sin 0 - \cos\frac{\pi}{4}\sin\frac{-\pi}{4}\cos 0\right)U1}{m}$$

$$\ddot{X} = \frac{U1}{2m}$$

$$\ddot{Y} = \frac{\left(\cos\frac{\pi}{4}\sin 0 + \sin\frac{\pi}{4}\sin\frac{\pi}{4}\cos 0\right)U1}{m}$$

$$\ddot{Y} = \frac{U1}{2m}$$

To summarize the plant model equations, the forces in each of the 6 degrees of freedom have been defined. From these, the directional acceleration can be found, as well as the angular acceleration. It is clear from the equations of motion that the directional accelerations depend on roll, pitch and yaw. Because acceleration is the 2nd derivative of position, integrals can be performed on the acceleration to get position. This is what will be done later to build the Simulink model. This concept can be illustrated in Table 2.6 and 2.7 below.

Table 2.6-Motion of equation dependencies

	Roll	Pitch	Yaw	X	Y	Z
Position	$\phi = \int \dot{\phi}$	$\theta = \int \dot{\theta}$	$\psi = \int \dot{\psi}$	$X = \int \dot{X}$	$Y = \int \dot{Y}$	$Z = \int \dot{Z}$
Velocity	$\dot{\phi} = \int \ddot{\phi}$	$\dot{\theta} = \int \ddot{\theta}$	$\dot{\psi} = \int \ddot{\psi}$	$\dot{X} = \int \ddot{X}$	$\dot{Y} = \int \ddot{Y}$	$\dot{Z} = \int \ddot{Z}$
Acceleration	$\ddot{\phi}$	$\ddot{\theta}$	$\ddot{\psi}$	$\ddot{X}$	$\ddot{Y}$	$\ddot{Z}$

Required for

Table 2.7-State and equation summary

State/Input	Dependencies	Equation
θ	$\dot{\theta}$	$\theta = \int \dot{\theta}$
$\dot{\theta}$	$\ddot{\theta}$	$\dot{\theta} = \int \ddot{\theta}$
$\ddot{\theta}$	$\dot{\phi}, \dot{\psi}, \Omega_r, U3$	$\ddot{\theta} = \frac{\dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\Omega_r + l(U3)}{I_{yy}}$
ϕ	$\dot{\phi}$	$\phi = \int \dot{\phi}$
$\dot{\phi}$	$\ddot{\phi}$	$\dot{\phi} = \int \ddot{\phi}$
$\ddot{\phi}$	$\dot{\theta}, \dot{\psi}, \Omega_r, U2$	$\ddot{\phi} = \frac{\dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) + J_r\dot{\theta}\Omega_r + l(U2)}{I_{xx}}$
ψ	$\dot{\psi}$	$\psi = \int \dot{\psi}$
$\dot{\psi}$	$\ddot{\psi}$	$\dot{\psi} = \int \ddot{\psi}$
$\ddot{\psi}$	$\dot{\theta}, \dot{\phi}, U4$	$\ddot{\psi} = \frac{\dot{\theta}\dot{\phi}(I_{xx} - I_{yy}) + (U4)}{I_{zz}}$
X	$\dot{X}$	$x = \int \dot{x}$
$\dot{X}$	$\ddot{X}$	$\dot{x} = \int \ddot{x}$
$\ddot{X}$	$\psi, \phi, \theta, U1$	$\ddot{X} = \frac{(\sin\psi \sin\phi - \cos\psi \sin\theta \cos\phi)U1 - A_x\dot{X}}{m}$
Y	$\dot{Y}$	$Y = \int \dot{Y}$
$\dot{Y}$	$\ddot{Y}$	$\dot{Y} = \int \ddot{Y}$
$\ddot{Y}$	$\psi, \phi, \theta, U1$	$\ddot{Y} = \frac{(\cos\psi \sin\phi + \sin\psi \sin\theta \cos\phi)U1 - A_y\dot{Y}}{m}$
Z	$\dot{Z}$	$Z = \int \dot{Z}$
$\dot{Z}$	$\ddot{Z}$	$\dot{Z} = \int \ddot{Z}$
$\ddot{Z}$	$\psi, \phi, U1$	$\ddot{Z} = \frac{mg - (\cos\theta \cos\phi)U1 - A_z\dot{Z}}{m}$
Ω_r	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$\Omega_r = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$
$\mathrm{U1}_+$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U1_+ = b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2)$
$\mathrm{U2}_+$	Ω_2, Ω_4	$U2_+ = b(-{\Omega_2}^2 + {\Omega_4}^2)$
$\mathrm{U3}_+$	Ω_1, Ω_3	$U3_+ = b({\Omega_1}^2 - {\Omega_3}^2)$
$\mathrm{U4}_+$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U4_+ = d({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2)$
$\mathrm{U1}_x$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U1_x = b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2)$
$\mathrm{U2}_x$	Ω_2, Ω_4	$U2_x = b\sin\left(\frac{pi}{4}\right)({\Omega_1}^2 - {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2)$

$U3_x$	Ω_1, Ω_3	$U3_x = b \sin\left(\frac{pi}{4}\right)({\Omega_1}^2 + {\Omega_2}^2 - {\Omega_3}^2 - {\Omega_4}^2)$
$U4_x$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U4_x = d({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2)$

2.1.5 Quaternions

Although quaternions will not be used in this paper, it would be an incomplete depiction of quadcopters without mentioning them. Quaternions are simply another way to represent a quadcopters angular position, similar to roll, pitch and yaw angles. Quaternions are represented with a 4-element vector. The reason to use quaternions is to eliminate gimbal lock which is a loss of one degree of freedom and occurs when two axes of rotation are driven into a parallel configuration.

To start realizing quaternions, one can begin with a unit vector around a normal 3D axis system. This unit vector and the unit vector rotation, along with some equations are all that's needed to realize quaternions. This can be seen in Figure 2.17 below.

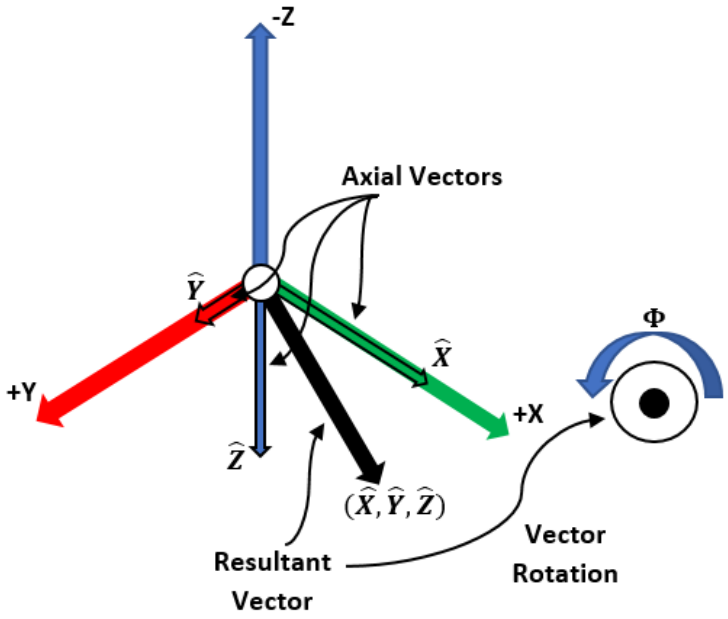


Figure 2.17-Quaternion realization

At this point the information representing the resultant vector and its rotation are $(\Phi, \hat{X}, \hat{Y}, \hat{Z})$. The next thing to do is define quaternion equations. These can be seen in Equations (2.28) - (2.31) below.

$$q_0 = \cos\left(\frac{\Phi}{2}\right) \tag{2.28}$$

$$q_1 = \hat{X}\sin\left(\frac{\Phi}{2}\right) \tag{2.29}$$

$$q_2 = \hat{Y}\sin\left(\frac{\Phi}{2}\right) \tag{2.30}$$

$$q_3 = \hat{Z}\sin\left(\frac{\Phi}{2}\right) \tag{2.31}$$

These components are what make up quaternions. They are 4 unit vectors which contain one scalar and 3 components that represent vectors. Thus, the final quaternion 4 unit vector can be seen in Equation 2.32 below.

$$\boldsymbol{q} = [q_0\ q_1\ q_2\ q_3] \tag{2.32}$$

The final mention with respect to quaternions is how to convert from Euler angles to quaternions, and then back again. First, converting from Euler angles to quaternions can be seen in Equations (2.33) - (2.36) where C represents cosine, and S represents sine.

$$q_0 = C\left(\frac{\phi}{2}\right)C\left(\frac{\theta}{2}\right)C\left(\frac{\psi}{2}\right) + S\left(\frac{\phi}{2}\right)S\left(\frac{\theta}{2}\right)S\left(\frac{\psi}{2}\right) \tag{2.33}$$

$$q_1 = S\left(\frac{\phi}{2}\right)C\left(\frac{\theta}{2}\right)C\left(\frac{\psi}{2}\right) - C\left(\frac{\phi}{2}\right)S\left(\frac{\theta}{2}\right)S\left(\frac{\psi}{2}\right) \tag{2.34}$$

$$q_2 = C\left(\frac{\phi}{2}\right)S\left(\frac{\theta}{2}\right)C\left(\frac{\psi}{2}\right) + S\left(\frac{\phi}{2}\right)C\left(\frac{\theta}{2}\right)S\left(\frac{\psi}{2}\right) \tag{2.35}$$

$$q_3 = C\left(\frac{\phi}{2}\right)C\left(\frac{\theta}{2}\right)S\left(\frac{\psi}{2}\right) - S\left(\frac{\phi}{2}\right)S\left(\frac{\theta}{2}\right)C\left(\frac{\psi}{2}\right) \tag{2.36}$$

Secondly, converting from quaternions back to Euler angles. This can be seen in Equations (2.37) - (2.39) below.

$$\phi = \tan^{-1}\left(\frac{2(q_0 q_1 + q_2 q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}\right) \tag{2.37}$$

$$\theta = \sin^{-1}\big(2(q_0 q_2 - q_1 q_3)\big) \tag{2.38}$$

$$\psi = \tan^{-1}\left(\frac{2(q_0 q_3 + q_1 q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \tag{2.39}$$

2.2 Basic Simulink Plant Model

Now that the basic equations of motion have been generated, the Simulink plant model can now be formed. The MATLAB script and Simulink models for this section can be found in the folder titled 'Basic_Simulink_Model'. First, a basic script that defines the inertia moments, rotor inertia, length, mass, thrust, and drag coefficients needs to be created. Then the Simulink model is created. Starting with a fresh Simulink block diagram, first define the solver options. On the top bar of Simulink hit Simulation, then Model Configuration Parameters. Under solver option, make sure the type is variable step, and the solver is on auto. A new subsystem can now be defined and named "Quadcopter plant". The subsystem has 4 inputs ($\Omega_1, \Omega_2, \Omega_3, \Omega_4$) and 6 outputs (X, Y, Z, roll, pitch, yaw). At this point the model should be looking like Figure 2.18 and 2.19 below.

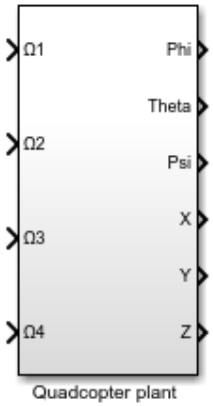


Figure 2.18-Quadcopter plant subsystem block

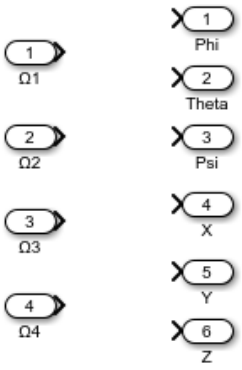


Figure 2.19-Quadcopter plant subsystem block inputs and outputs

Next, the inputs need to be set up. Here another subsystem block is created, and dependent on the orientation style ('+' or 'x'), Equations (2.4) - (2.7) or (2.8) - (2.11) are used. It's know that 4 inputs (Ω_1, Ω_2, Ω_3, Ω_4) and 5 outputs ($U1, U2, U3, U4$, Ω_r) are needed. One of the main reasons this block is needed is to generate Ω_r and the thrust inputs to the plant. This block is also later used to store motor dynamics. The first thing that is done in this block is form Equation 2.12 to generate Ω_r. This can be seen in Figure 2.20 below.

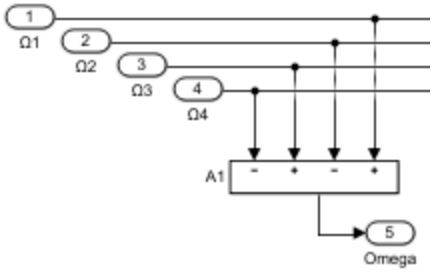


Figure 2.20-Angular velocity equations

Now that the angular velocities have been summed, the thrust inputs need to be formed. This is done by following Equations (2.8) - (2.11). This can be seen in Figure 2.21 below.

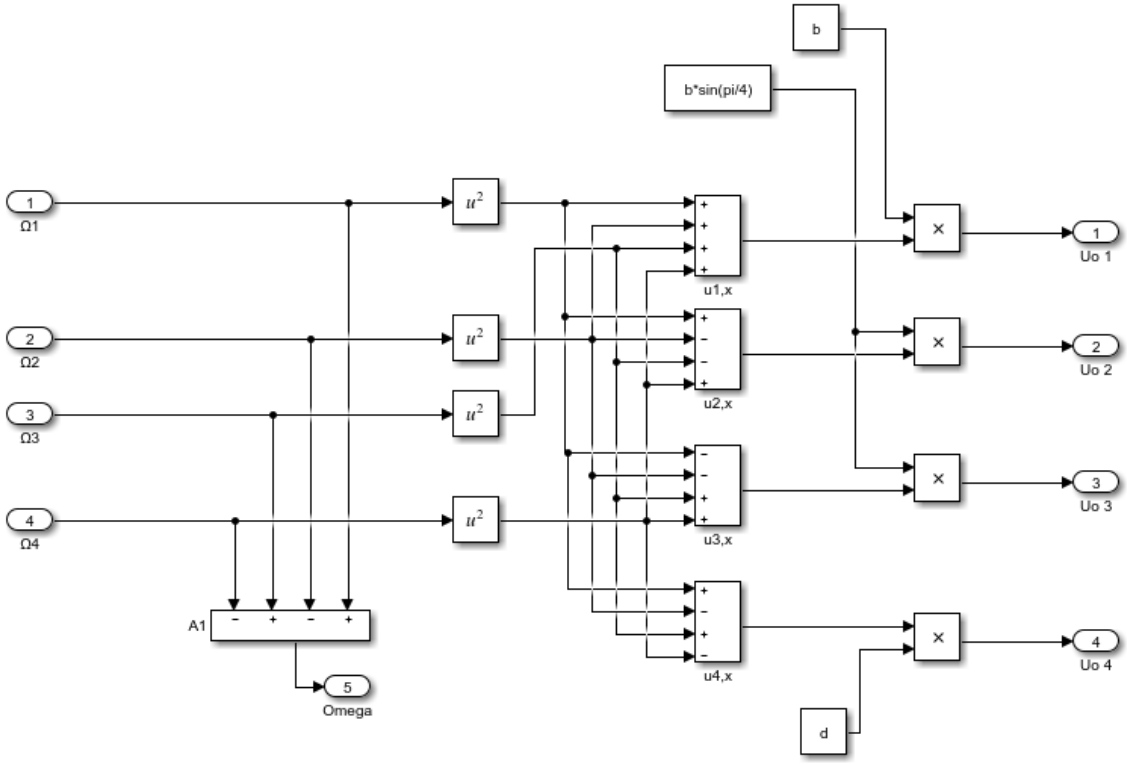


Figure 2.21-Angular velocity and thrust equations

The ‘+’ configuration can also be formed. This can be seen in Figure 2.22 below.

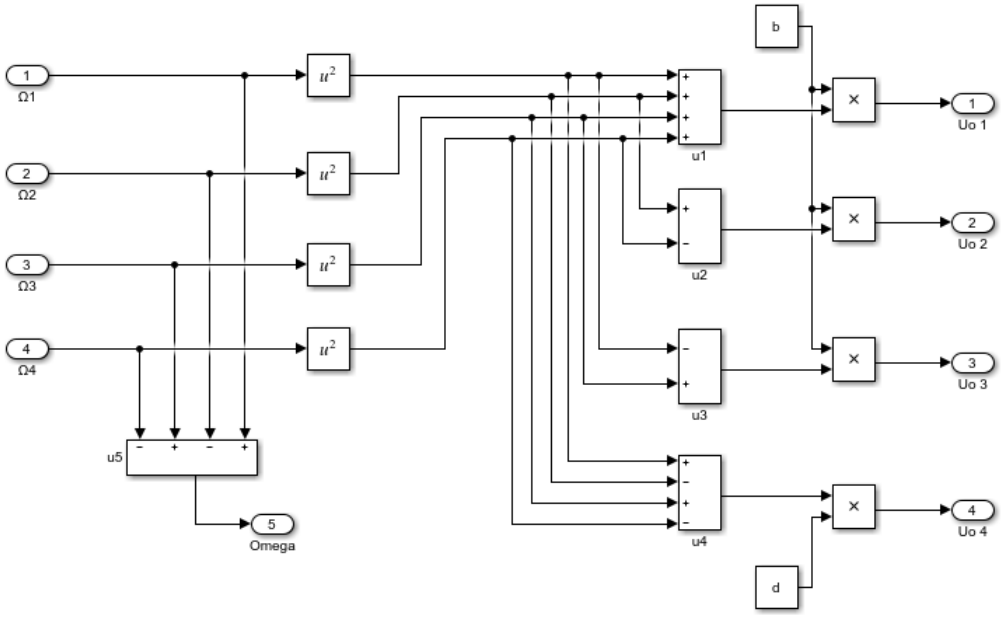


Figure 2.22-'+' orientation input commands

The next step in the process is to use Equations (2.17), (2.19), and (2.21) to create the angular accelerations. First create a subsystem block, there will be 3 outputs ($\ddot{\phi}$, $\ddot{\theta}$, and $\ddot{\psi}$) and 7 inputs ($\dot{\phi}, \dot{\theta}, \dot{\psi}, U2, U3, U4$ and Ω_r). Within this block, other blocks can be created for each of the equations. This can be seen in Figure 2.23 below.

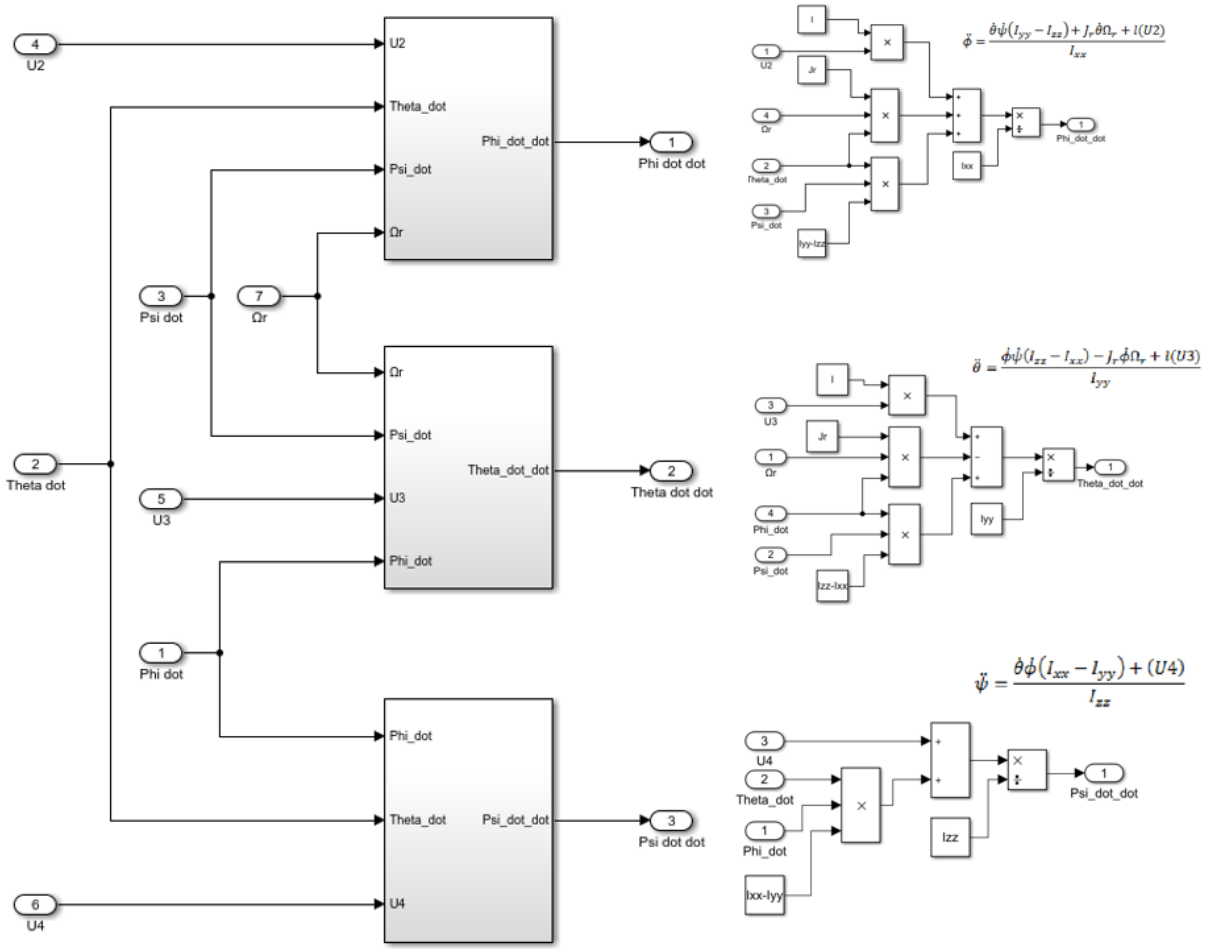


Figure 2.23-Quadcopter plant angular acceleration equations

Since the model now has the angular accelerations, the angular velocities and positions are needed. This can be done by taking integrals of the acceleration. This can be seen in Figure 2.24 below.

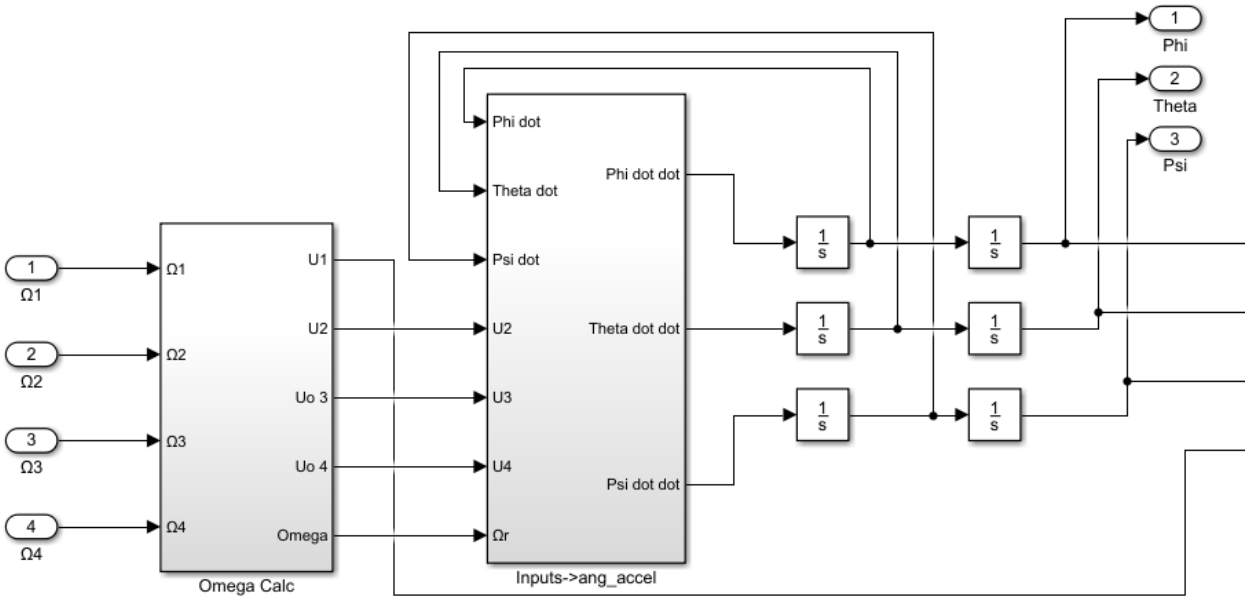


Figure 2.24-Angular accelerations to angular velocities and positions

Now a subsystem block for Equations (2.23), (2.25), & (2.27) can be created. It's known from the equations 3 outputs ($\ddot{X}, \ddot{Y}$ and $\ddot{Z}$) and 7 inputs ($\dot{X}, \dot{Y}, \dot{Z}, \phi, \theta, \psi$ and $U1$) are needed. Similar to the angular accelerations, subsystem blocks can be created for each of the axial accelerations. This can be seen in Figure 2.25 below.

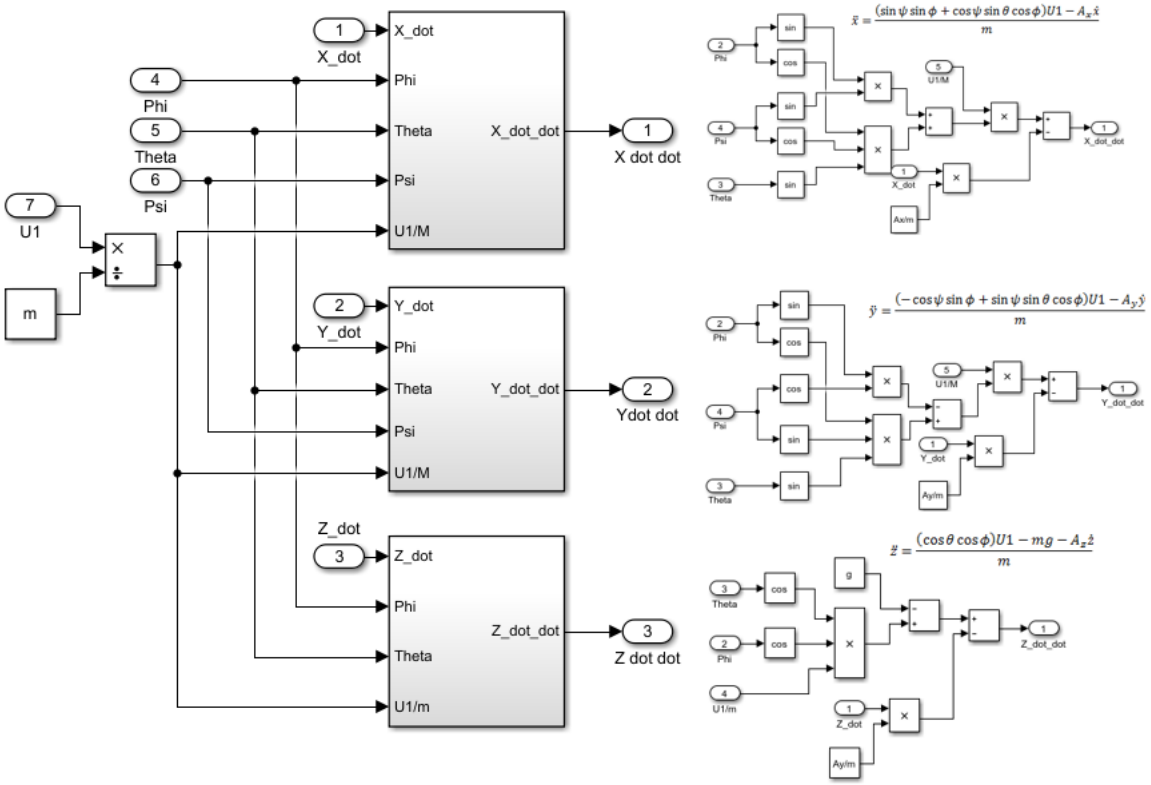


Figure 2.25-Quadcopter plant angular acceleration equation blocks

From this point the axial accelerations and positions need to be made. This can be done using integral blocks. This can be seen in Figure 2.26 below.

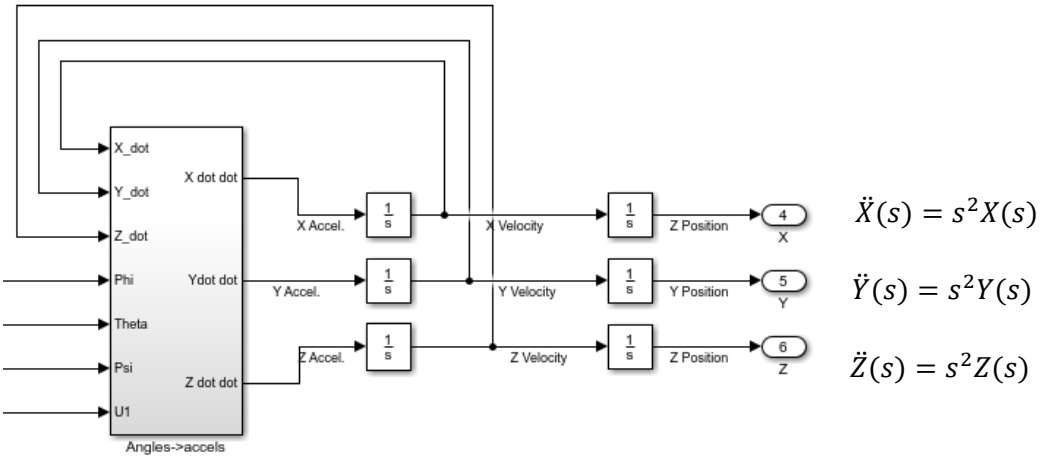


Figure 2.26-Quadcopter plant integrals

The final plant model can be seen in Figure 2.27 below.

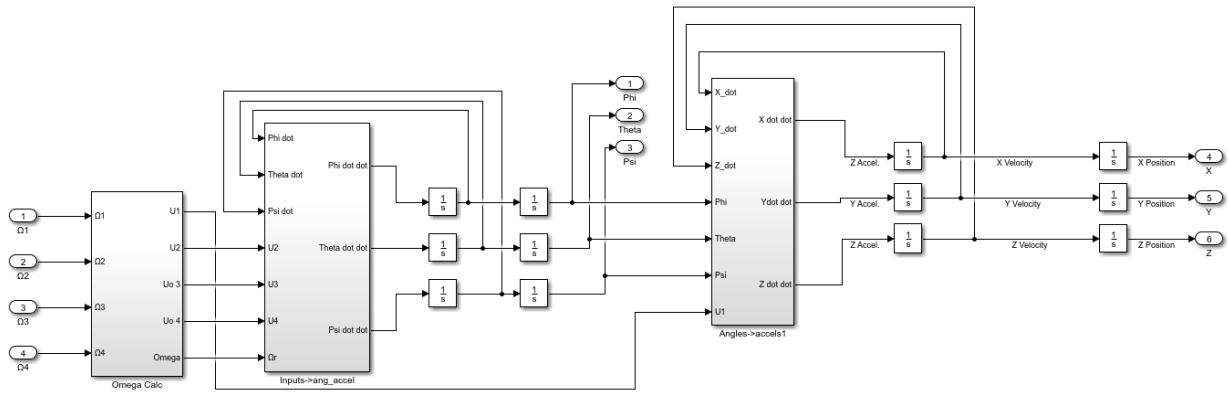


Figure 2.27-Quadcopter plant feedback

2.3 Real World Limits

At this point a general plant model has been created by simply using the equations of motion, along with input thrust equations. An important aspect to realize about the model is that the equations of motion do not perfectly represent a quadcopter. For instance, in many quadcopters the motors only spin in one direction, or that the quadcopter cannot physically be in the positive Z coordinate (depending on frame of reference of course). Because of this, limits need to be applied to the model.

First, the frame of reference needs to be considered. For this model the X and Y directions will be free, and the Z direction will be limited from zero to negative infinity. This is because the negative Z direction is up. There are multiple methods that can be used to implement these limits to the model. The saturation block can be used, or integral limits can be applied. Applying the integral limit can be seen in Figure 2.28 below.

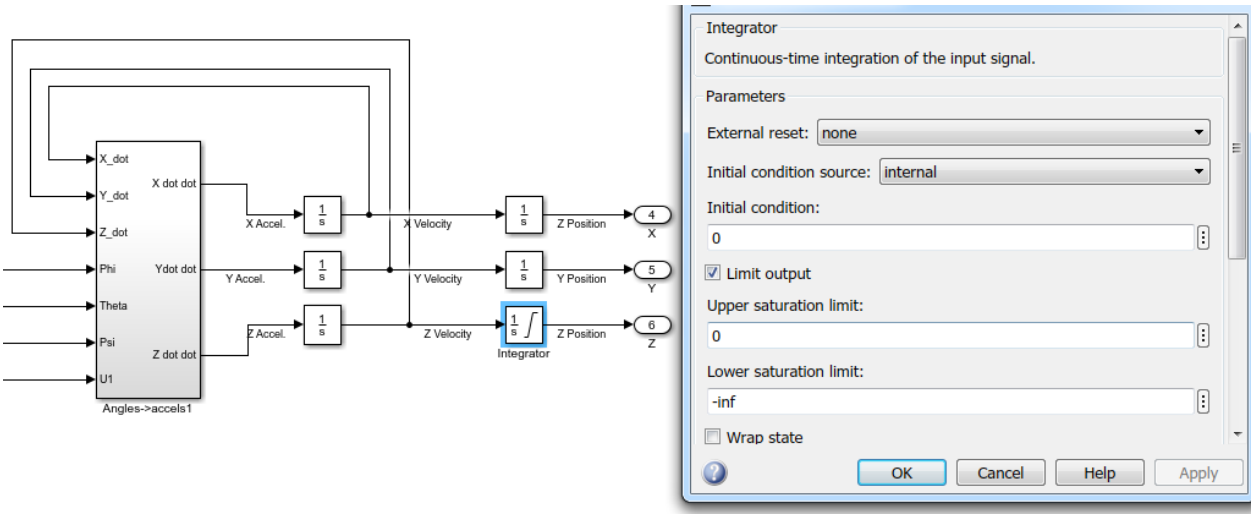


Figure 2.28-Applying integral limits

Next, limits should be applied to the angular velocity of the rotors. Some quadcopters possess the ability to reverse the velocity of their rotors midflight, however, the model being created this shall only possess the ability to rotate in one direction. These limits should be applied in the subsystem block where the angular velocity was found. Here, saturation blocks will be used, limited from zero to infinity. In later sections the angular velocity will no longer have an upper limit of infinity due to motor dynamic limitations. The application of these limits can be seen in Figure 2.29 below.

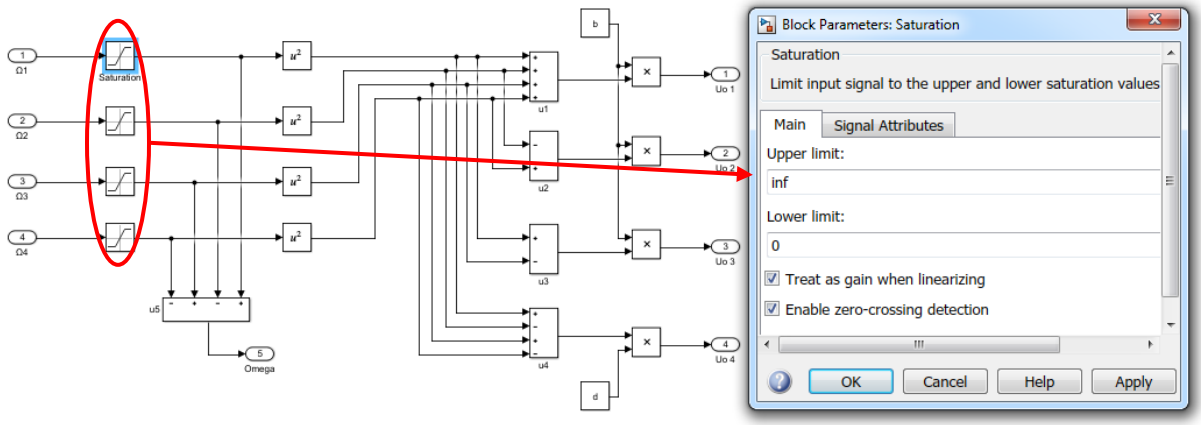


Figure 2.29-Applying angular velocity limits

Lastly, the euler angles can be wrapped. This limits the angles to be between -pi and pi. This limit can be placed in the model in the integrals from angular velocity to angular position. This step can be seen in Figure 2.30 below.

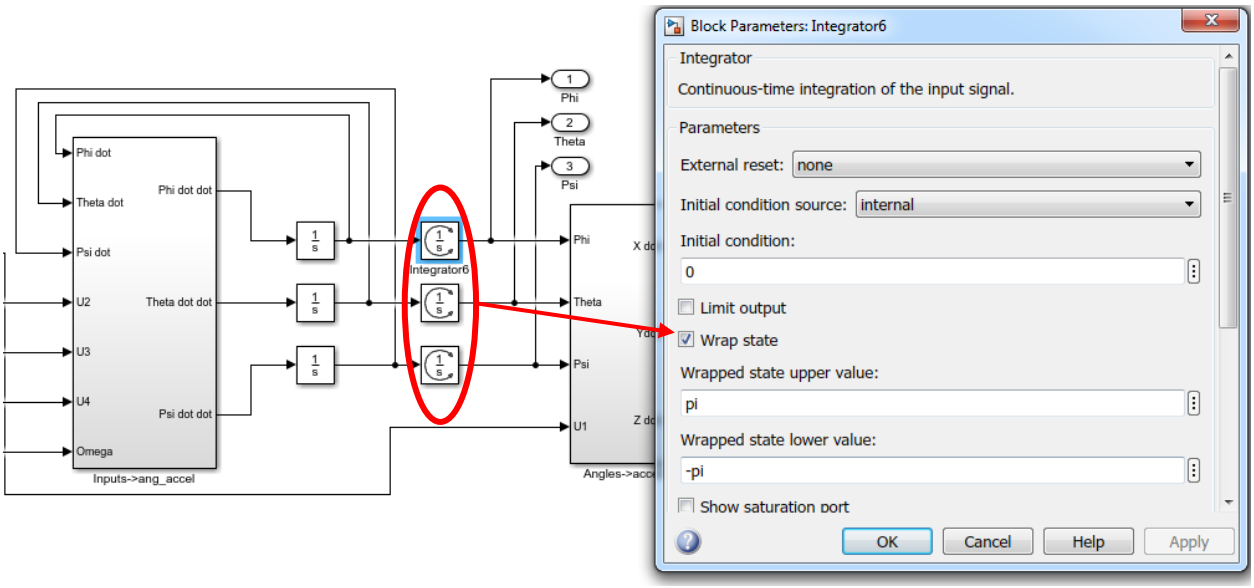


Figure 2.30-Limiting euler angular positions

At this point some inputs can be applied to the plant model. Since the input to the plant is rotor angular velocity either a constant block or step block can be sued to represent an angular velocity input. Thrust, roll, pitch and yaw can be simulated by applying velocities that follow Equations (2.4-2.7) or (2.8-2.11). These simulations in the 3D plane can be seen in Figures 2.31-2.34 below.

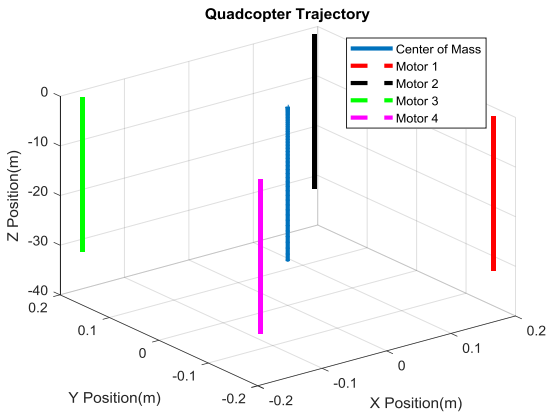


Figure 2.31-Thrust plant simulation

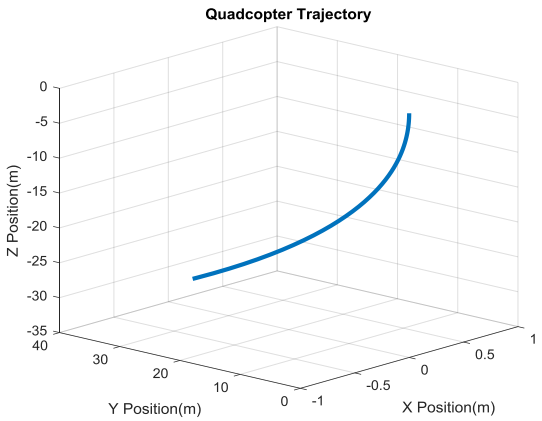


Figure 2.32-Roll plant simulation

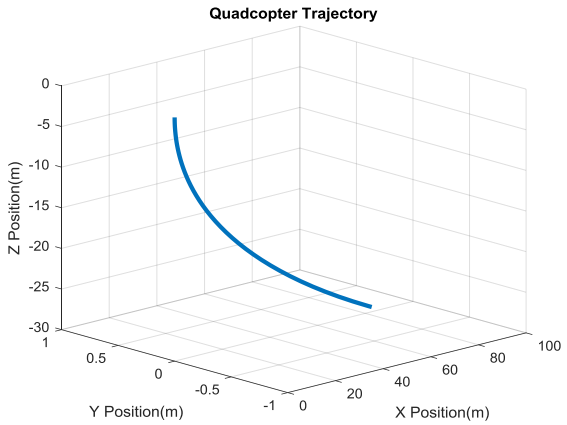


Figure 2.33-Pitch plant simulation

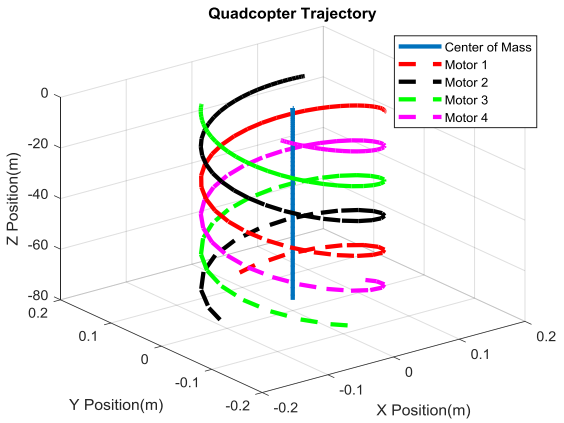


Figure 2.34-Yaw plant simulation

2.4 Motor Dynamics

Now that a basic quadcopter plant model has been generated, more detail can be added to it. As the plant model is presently, the motors can continue building angular velocity and generate instant thrust. This would not happen in an actual quadcopter due to power and motor limitations. In order to model the limitations of the quadcopter, the brushless DC motor must be modeled. The motor model input is voltage, the states are current, and angular acceleration, the parameters include a damping coefficient, and torque coeffect, and the output is motor angular velocity.

2.4.1 Motor Dynamics Equations

Starting with the most basic motor model, a series circuit, composed of a voltage source, resistor, inductor and voltage drop across the motor, the modeling can begin. The motor block diagram can be seen in Figure 2.35 below along with the parameters in Table 2.8 below.

Table 2.8-Motor model parameters

J	Rotor moment of inertia	$kg \cdot m^2$
b	Damping coefficient	Nms
K_{emf}	Back electromotive force coef.	Vs/rad
K_t	Torque coefficient	Nm/A
R	Electrical resistance	Ω
L	Electrical Inductance	H

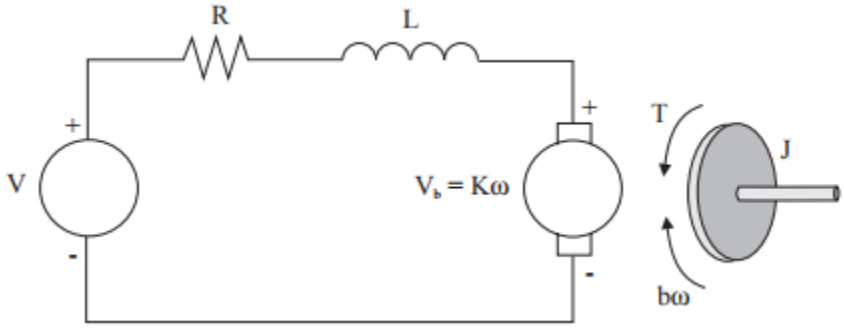


Figure 2.35-Basic DC motor model

This model has several important DC motor first principal equations associated with it which leads to forming transfer functions for the system [11].

$$L\frac{di}{dt} = V - RI - K_{emf}\Omega \tag{2.40}$$

$$J\frac{dw}{dt} = K_t I - b\omega \tag{2.41}$$

Using data generated with a dynamometer the parameters can be found. Beginning with torque, current and time data, torque over current can be plotted to find the motor torque constant. Plotting this data, a plot that looks similar to the plot seen in Figure 2.36 below can be generated. The torque coefficient is taken where the thrust will hover the quadcopter, which in this case is $K_t = 0.00255\, Nm/A$. Since K_t and K_{emf} are equal $K_{emf} = 0.00255\, Vs/rad$. The damping coefficient can be found in a similar way to the torque coefficient. It can be found by plotting torque vs. angular velocity. This plot can also be seen below in Figure 2.37, where it can be seen that the damping coefficient is roughly $= 2.415 - 6\, Nms$. J can be determined from finding the moment of inertia of the propeller which was found while creating the plant model. The inductance and resistance can in most cases be found in the motors datasheet.

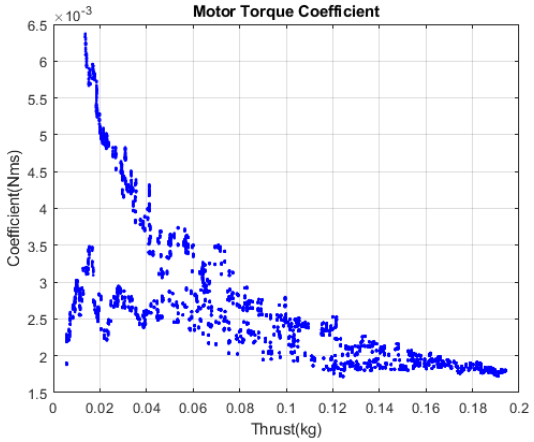


Figure 2.36-Motor torque coefficient plot

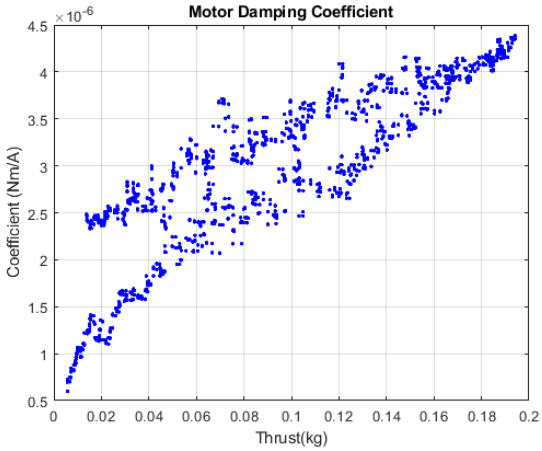


Figure 2.37-Motor damping coefficient plot

2.4.2 Motor Simulink Model

With the equations seen in section 2.4.1, the Simulink model can be built, which can be seen in Figure 2.38 below. The MATLAB script and Simulink model for this section can be found in the folder titled 'Motor_Model'. It should be noted that the model comes with limits on top of the differential equations shown above. Integration limits should be put on both integrals limiting the lower limit to zero.

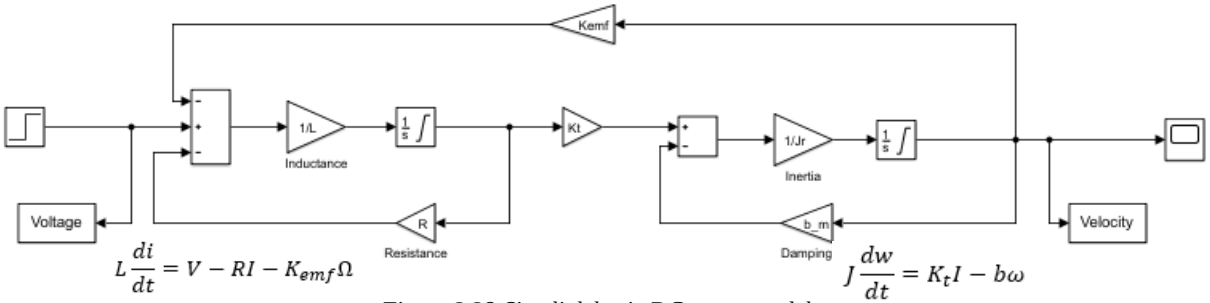


Figure 2.38-Simulink basic DC motor model

Applying a step to the system using the parameters seen below. The output velocity can be seen in Figure 2.39 below.

Table 2.9-Motor parameters

$J = 6.5e-7\ kg \cdot m^2$
$b = 2.415-6\ Nms$
$K_{emf} = 0.00255\ Vs/rad$
$K_t = 0.00255\ Nm/A$
$R = 0.117\ \Omega$
$L = 1.17e-4\ H$

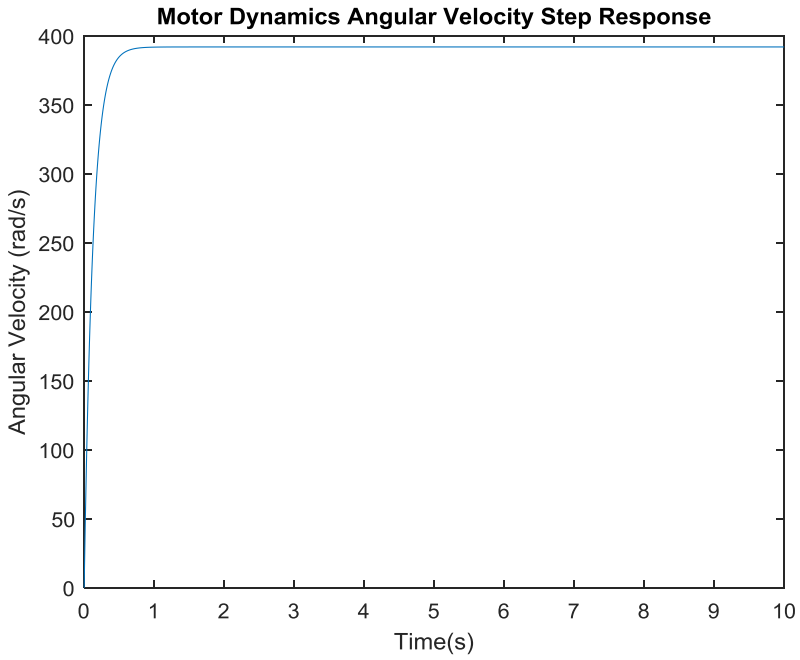


Figure 2.39-Motor dynamics plant model step response

An important aspect of the motor model to understand is the frequency response (FRF), or bode plot. Prior to obtaining the FRF, the system's transfer function should be found. The transfer function can be found using MATLAB's system ID toolbox. To use the toolbox, input and output step response data needs to be acquired using To Workspace blocks in Simulink. Next, open the system identification toolbox by using the '*systemIdentification'* command, and import the input (voltage) and output (angular velocity) time domain data. This step can be seen in Figure 2.40 below.

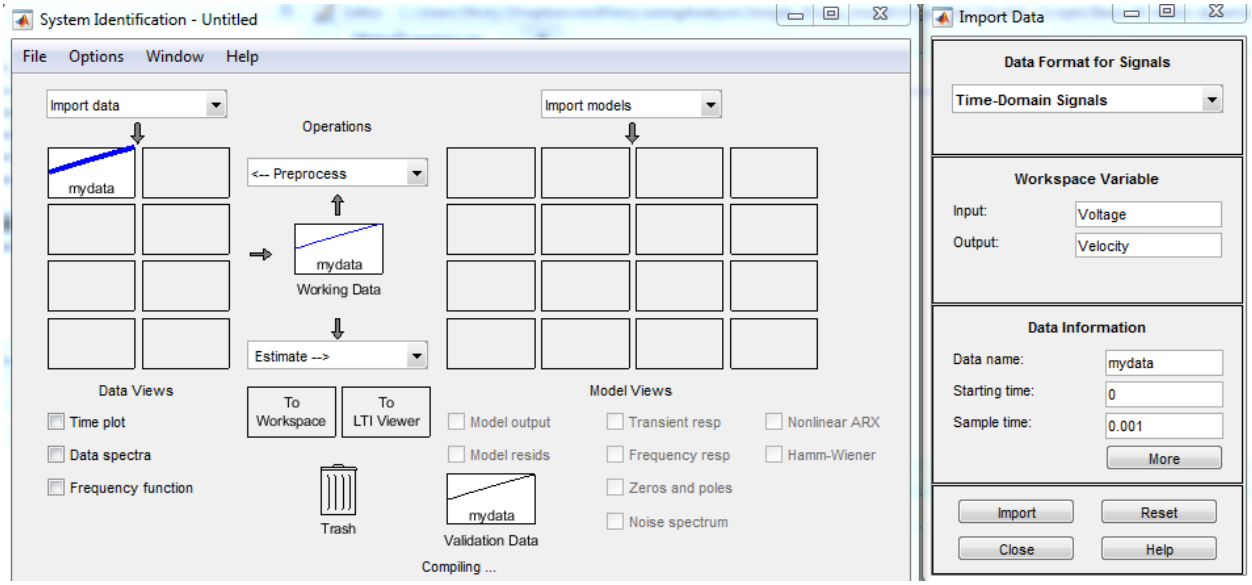


Figure 2.40-System identification toolbox

Next, use the system identification toolbox to estimate the transfer function by choosing estimate transfer function. The estimate should automatically be populated in the workspace. Using the step function, the estimated transfer function's step response can be plotted over the actual systems step response. This can be seen in Figure 2.41 below.

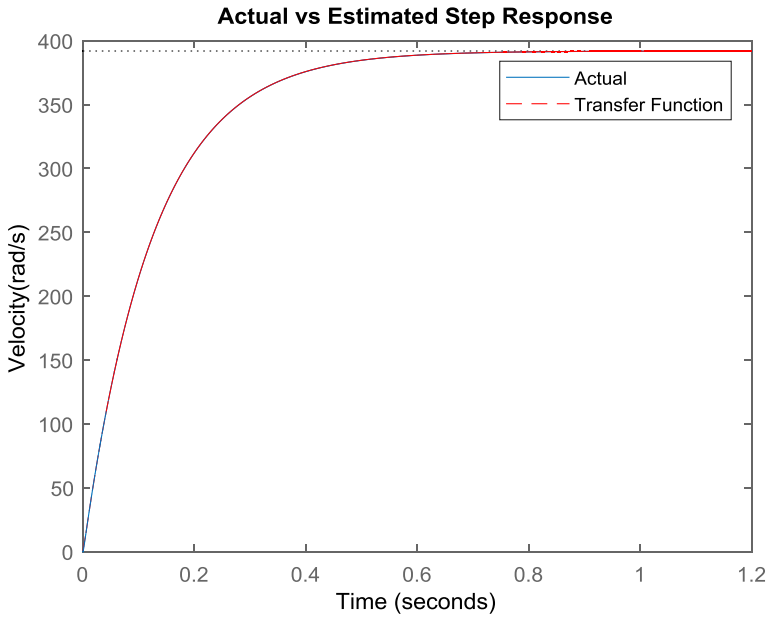


Figure 2.41-Actual vs transfer function step responses

From here the *margin* function can be used to view the FRF of the transfer function. Along with the FRF, useful information to know is where the poles and zeros of the system are. These can be found using the *pzmap* function within MATLAB. Both the FRF and pole-zero map of the motor model can be seen below.

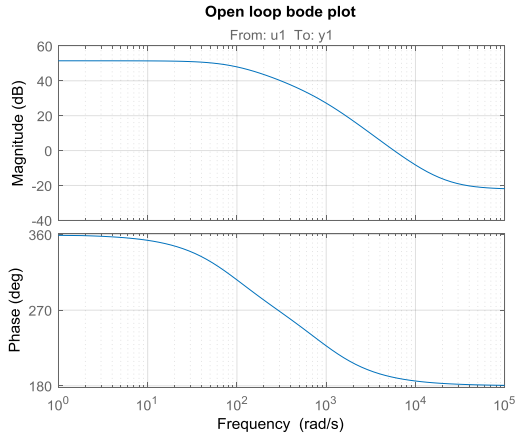


Figure 2.42-Motor plant model frequency response

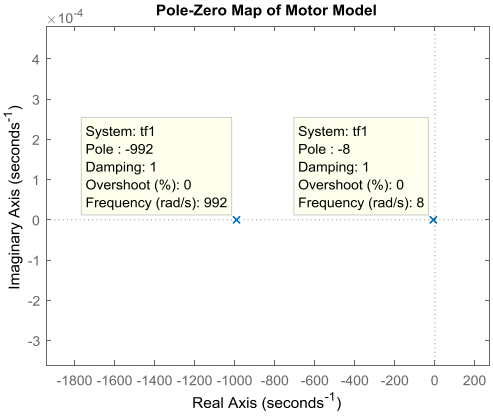


Figure 2.43-Motor plant model pole-zero map

A significant amount of information can be generated from the bode plot. What this magnitude plot shows is that for all frequencies up to 1100 rad/s the output is greater than the input. However, the output is greater at lower frequencies than at higher frequencies. What frequency the poles and zeros are can also be deciphered directly from the FRF. The poles are around 4 rad/s, however these can be confirmed with the

pole-zero plot. Inputting a biased sinusoid to the open loop motor model can demonstrate the concept seen in the bode plot. Figure 2.44 has a sinusoidal input to the motor model with a frequency of 1 rad/s, the output of the motor model has a sinusoidal signal that is roughly 50 dB greater than the input, which is what the bode plot depicts. Figure 2.45 has a sinusoidal input to the motor model with a frequency of 100 rad/s, the output of the motor model has s sinusoidal signal that is roughly 27 dB greater than the input signal, which is also represented in the bode plot.

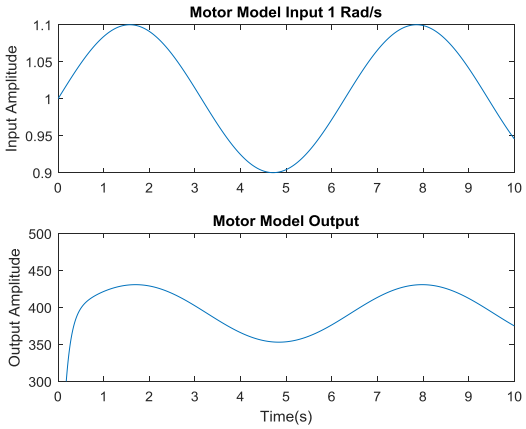


Figure 2.44-Motor model input & output at 1 rad/s

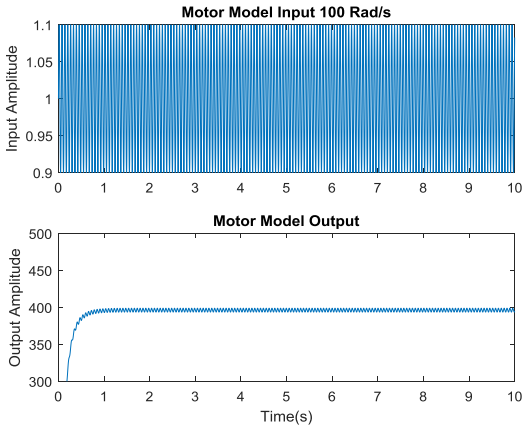


Figure 2.45-Motor model input & output at 100 rad/s

2.4.3 Brushless DC Motor Control

Now that the motor model has been created a controller needs to be implemented. A motor controller is needed to ensure the that motor is operating at the desired angular velocity. Multiple methods can be used to control the motor. For now, simply closing the loop, or applying unity proportional feedback will be used. Closing the loop is one of the simplest feedback control methods that exist. The closed loop motor model can be seen in Figure 2.46 below. Closing the loop means that feedback is taken from the output, assuming the output is observable, and it is subtracted from the desired state. The error term is then fed back into the plant.

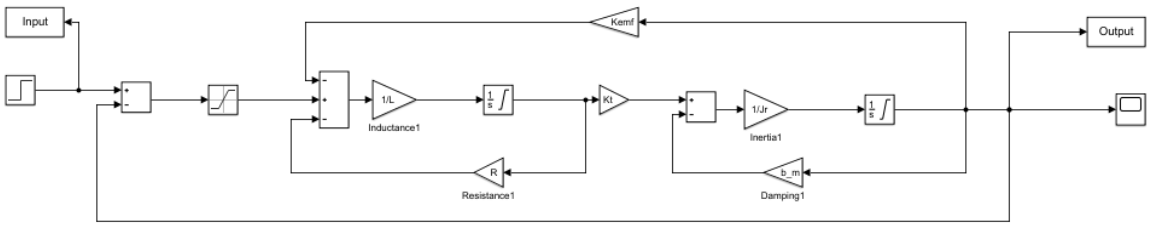


Figure 2.46-Closed loop motor control

The FRF of the closed loop system can be seen in Figure 2.47 below. Here it can be seen that the controller does a relatively good job of keeping the gain of the system constant at lower frequencies.

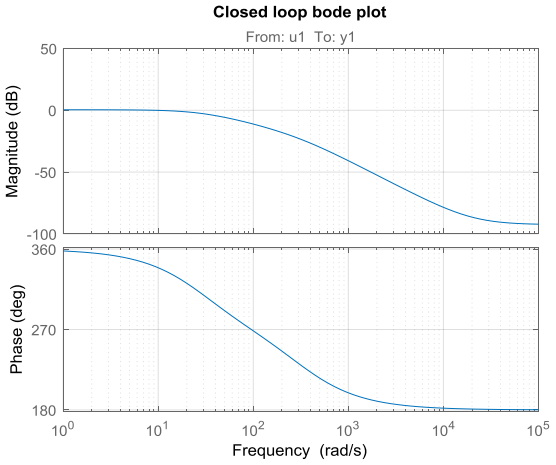


Figure 2.47-Closed loop motor model bode plots

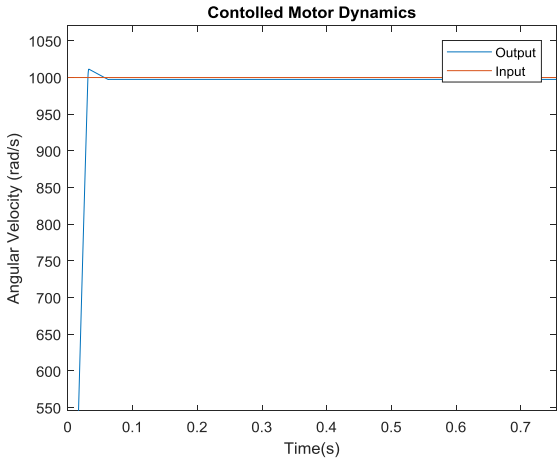


Figure 2.48-Controlled motor model step response

Because there is a close to a zero dB gain at low frequencies of the closed loop system a constant input should provide close to zero steady state error. The step response can be seen in Figure 2.48 above. It is worthy to note that the step would have a much faster settling time if a voltage saturation block was not used. The system is limited to a max voltage of 12 volts.

Now that the motor dynamics are controlled, they can be incorporated them into the Simulink plant model. The motor dynamics can be put into a subsystem block and placed into the ‘Omega Calc’ subsystem block within the plant model. The system outputs and RPMs can be plotted once the system becomes controlled in later sections.

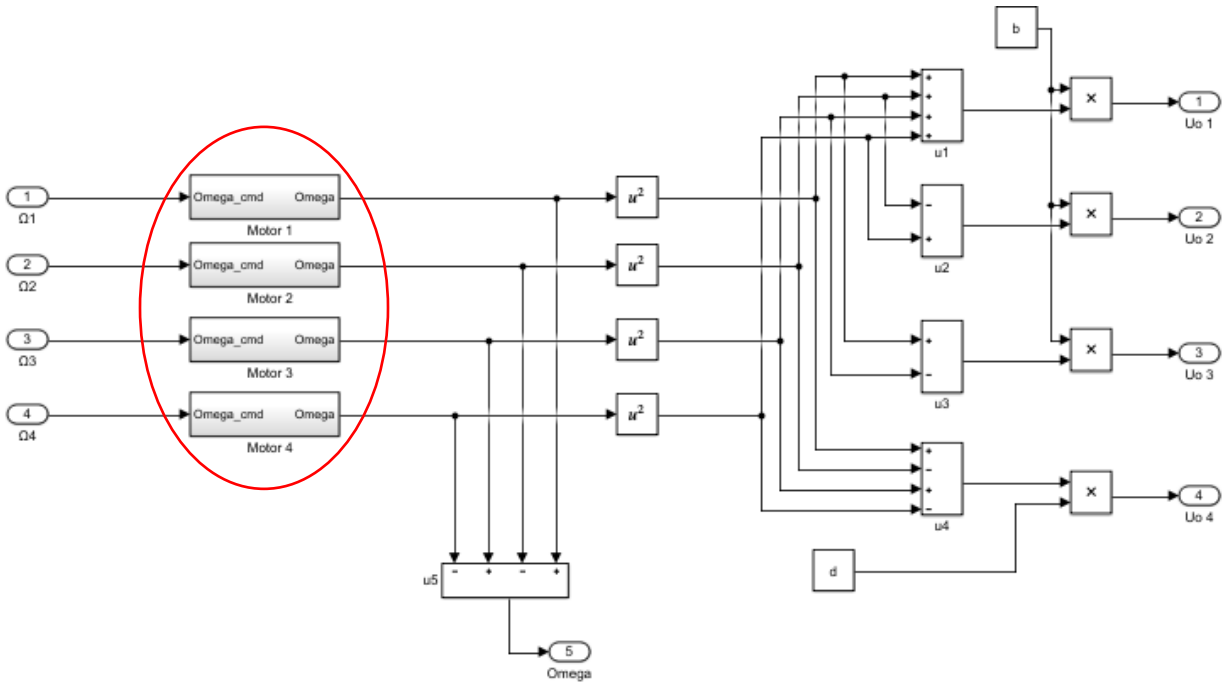


Figure 2.49-Plant model with motor dynamics

This is a good model of a DC motor but, brushless motors that are used in quadcopters are controlled a slightly different way. They require ESC's to work properly and are driven using a PWM signal, which have update delays along with step times associated with them. The PWM that is used is defined by the duty cycle. The duty cycle defines how much of a PWM signal's period is high or low. For example, a duty cycle of 100% would be a signal that is entirely high, a duty cycle of 50% would be a signal that is half high and half low. This is then related to voltage which commands the motors.

2.5 Battery Modeling

The battery of a quadcopter supplies power to components such as the motors, flight controller and receiver. Understanding battery dynamics is important to know. The battery voltage, current, and amp hour capacity are all important parameters to know about the battery. In the simulation domain the input to the battery is the current draw, and the output is voltage. States include state of charge, voltage, current, and temperature. Some of the parameters include battery capacity, and discharge rate.

2.5.1 Battery Model Equations

Looking at a quadcopters LiPo battery data sheet, which can be seen below, there are multiple terms that can be defined.

Figure 2.50-LiPo battery [12]

Starting with the *Capacity*, this is a measure of how much power the battery can hold. This rating is in the unit of milliamp hours, which means how much current the battery can supply constantly per hour. For the battery pictured above, it would be able to supply a constant 700mA, or 0.7A for 1 hour.

Next, the *Configuration*, this has to do with the number of cells within the battery. Each cell, or the "S" count supplies around 3.7 volts. The battery above has a 3S configuration, which means 3 cells, which together provide 11.1 volts [13]. This voltage is important with respect to the motors, as each motor has a KV rating which tells how fast the motor will run per volt supplied. For example, a 5000 KV motor will run at 50,000 RPM if 10 volts are supplied. With higher cell count, motors can rotate faster.

$$RPM = Volts * KV \tag{2.42}$$

Constant Discharge and *Peak Discharge* have to do with how much current can safely be supplied by the battery. For these numbers to be useful, the battery capacity also needs to be known. The equation to find the continuous current is capacity (Amps) times the constant discharge, or peak discharge C rating. The

result will give a constant and burst current amount. For example, the safe current discharge rate is 60 times 0.7A which equals 42A, which means the battery can safely supply 42 amps.

$$Discharge\ Current = Discharge\ Rate * Capcity \tag{2.43}$$

The other parameters seen on the spec sheet have to do with physical properties of the battery. The mass, and dimensions of the batter will affect the moment of inertia calculations. The charging ports will affect what types of chargers that can be used to charge and discharge the battery.

2.5.2 Battery Simulink Model

At this point there are enough equations to model the battery/flight time of the quadcopter. The MATLAB script and Simulink model for this section can be found in the folder titled 'Battery_Model'. The main equations that are needed are the capacity and discharge equations. To begin modeling the battery, open the Simulink plant model where the motor dynamics are stored. The battery will be modeled here because the motors are what need the most amount of power from the battery. Start the model by initializing the parameters of the battery in a MATLAB script which include the battery capacity, and discharge rates, then calculate the safe operating current. Next, create a new subsystem battery block. This block will have one input, current draw, and two outputs, available current and battery voltage. The motor dynamics block will also need to have input and outputs added to it, one that is used to calculate current draw, a second to limit the amount of current supplied and a third to limit the voltage. Once that is done the new Simulink plant model should look like Figure 2.51 below.

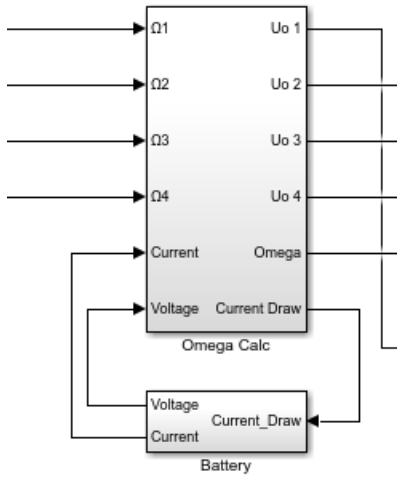


Figure 2.51-Simulink battery plant model

It's know from the battery spec sheet what the capacity is, for example, 700 mAh. This means a constant 700 mA for an hour. From there it can be multiplied by 60 to get how many amps can be supplied per minute, then multiplied by 60 again to get how many amps can be supplied per second. After doing this, multiplication, a constant amount of available current per 3600 seconds has been created. The goal of modeling the battery is to capture how much current the motors need, and to continually subtract that from this total amount. To envision this, imagine a full gas tank. The number of gallons is similar to the constant amount of current. Driving causes the gas level to go down which is the same idea as the current total decreasing when the motors need current. At this point the current draw from the motors needs to be captured. The current can be seen in the motor model after the first integral term, then summing the current of each motor. The process should done within each motor dynamic subsystem then summed in the omega calculation subsystem. The motor model and omega subsystems Simulink model that describe this process can be seen in Figures 2.52 and 2.53 below.

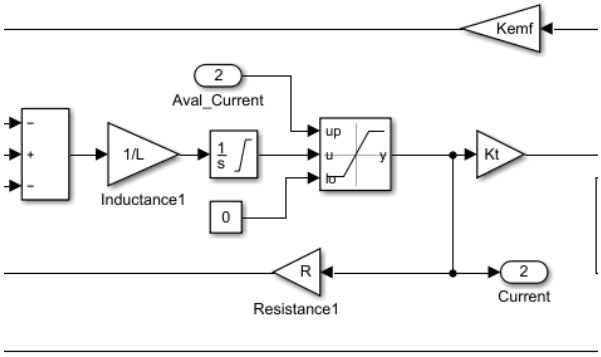


Figure 2.52-Motor model current output

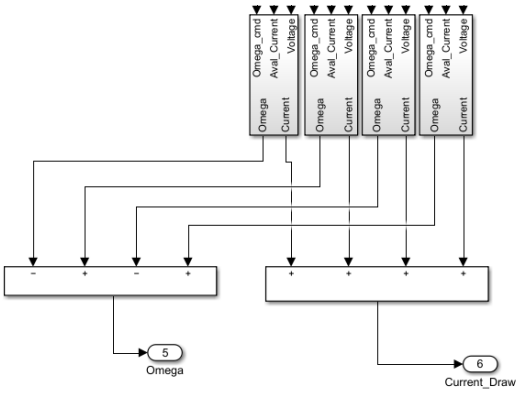


Figure 2.53-Motor's current summation

At this point there should be a total current draw vector coming from the Omega calculation subsystem. This can now be summed and subtracted from the total current. However, before this is done it should be noted how Simulink's summing blocks work. As in the summing block above, each time sample iteration provides a new sum. This means that if a to workspace block was used, the sum at each time sample would be able to be captured. This block does not continually sum new samples with previous samples, which is what the goal is. To sum the total current drawn, a time delay block will be used as a feedback loop to a summing block. This delay block saves the previous sum and adds it to the new sum. A rate transition block can be used to increase or decrease the sample time, it is also used to ensure that the total current available is correct. This Simulink model can be seen in Figure 2.54 below.

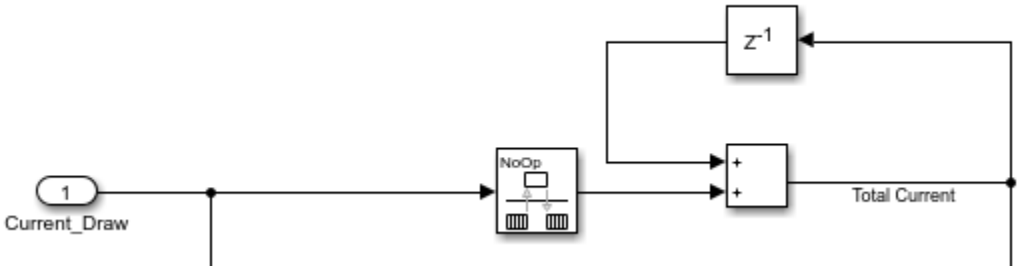


Figure 2.54-Total current calculation

At this point it's known that the total amount of current available per 3600 seconds, however our current addition scheme seen above sums the total current at 1000Hz sample rate. To account for this the total current available needs to be multiplied by 1000. Next, the total current draw can be subtracted from the

total current. Limits need to be placed on the available current such that the motors do not draw more than the battery is rated for. The maximum current that can be drawn, or max burst current is the burst C rating times the battery capacity. This calculation can be seen in Figure 2.55 which also contains the current limit block.

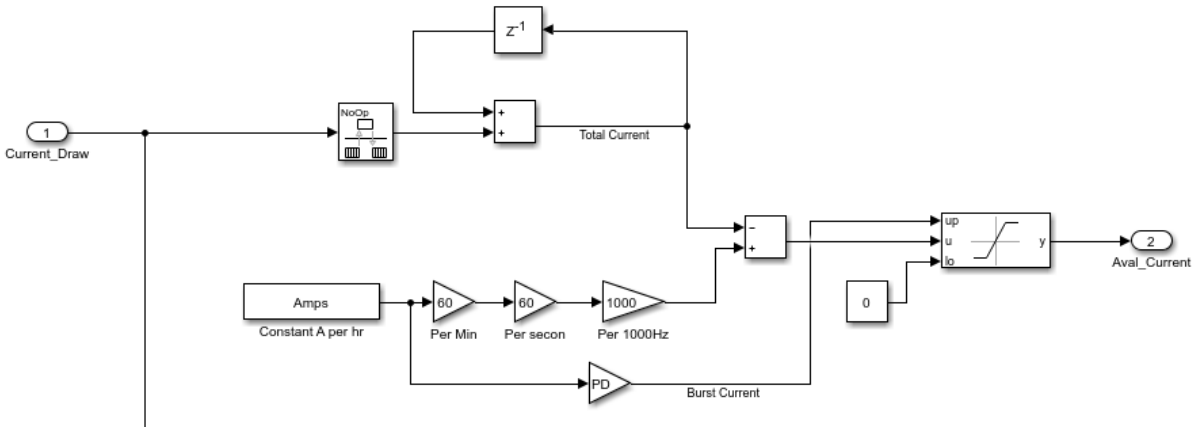


Figure 2.55-Available current limited

Now that the current has been limited, the current within the motor dynamics subsystems can be limited. This can be done by adding inputs to the motor dynamics blocks then using a limiting block. This can be seen in Figure 2.56 below.

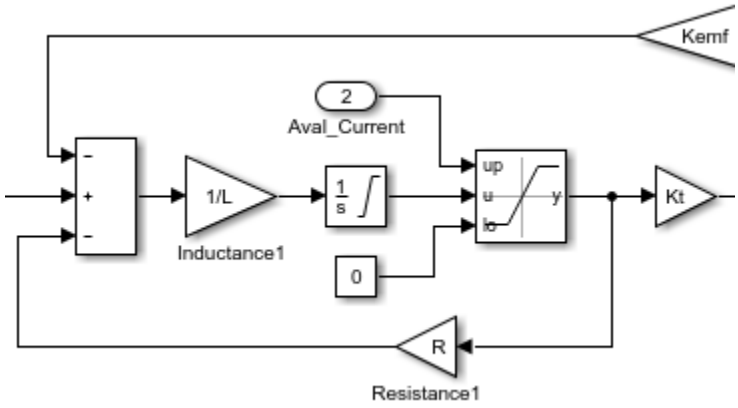


Figure 2.56-Current limiting motor dynamics

At this point if the model runs long enough the current supply will eventually dry-up and the motors will no longer spin. Another aspect that is useful to know is the battery voltage curve. The battery voltage discharge curve is dependent on type of battery, initial charge, and discharge rate. The curve is important

because if a battery is discharged too far it can damage the battery. This can be simulated in Simulink using the SimScape library.

In the battery subsystem, place the SimScape battery component and change the type of battery to the one that is being simulated, and populate the parameters using the battery of interest. Next, run the current draw through the battery using a controlled current source block. Lastly, place a powergui block to have the SimScape blocks run. The battery block outputs the battery voltage along with the battery's state of charge, which represents the amount of power left in the battery. The SimScape portion of the Simulink diagram can be seen in Figure 2.57 below.

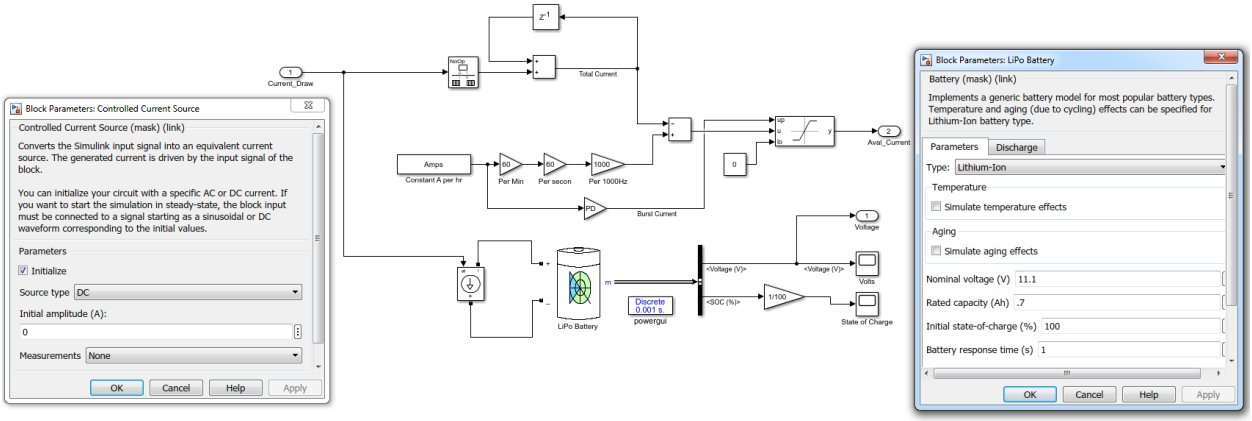


Figure 2.57-SimScape battery model

Now that the voltage of the battery is modeled, it can be compared to experimental results. Figure 2.58 below shows a comparison of experimental to simulated battery results.

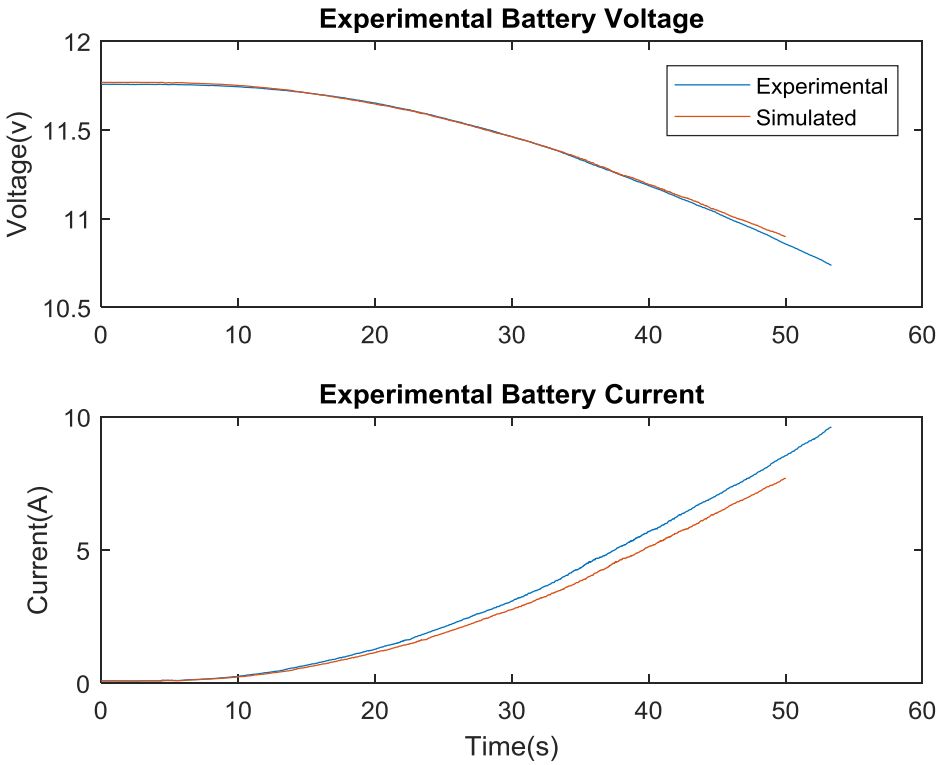


Figure 2.58-Battery experimental & simulated results

Now the battery simulation can be used to limit the supply voltage of the motor models. This can be done in the same way that the current was limited. The Simulink diagram of the voltage and current limited motor dynamics can be seen in Figure 2.59 below.

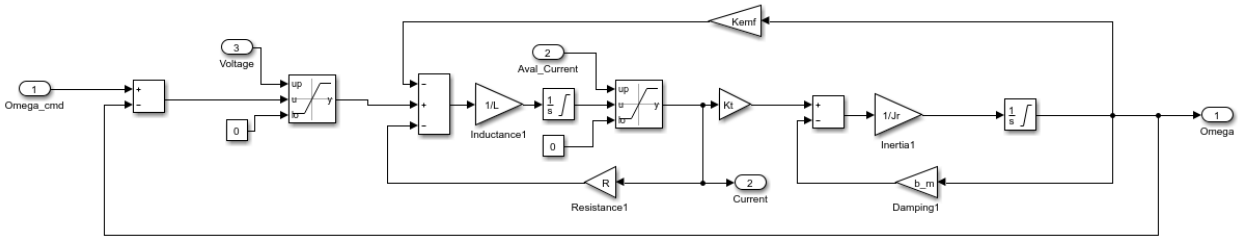


Figure 2.59-Voltage and current limited motor dynamics

Running a hover simulation, the battery voltage, state of charge, and quadcopter altitude can be plotted. The results of the simulation can be seen in Figure 2.60 below. Figure 2.60 shows how the quadcopter drops when the state of charge becomes zero.

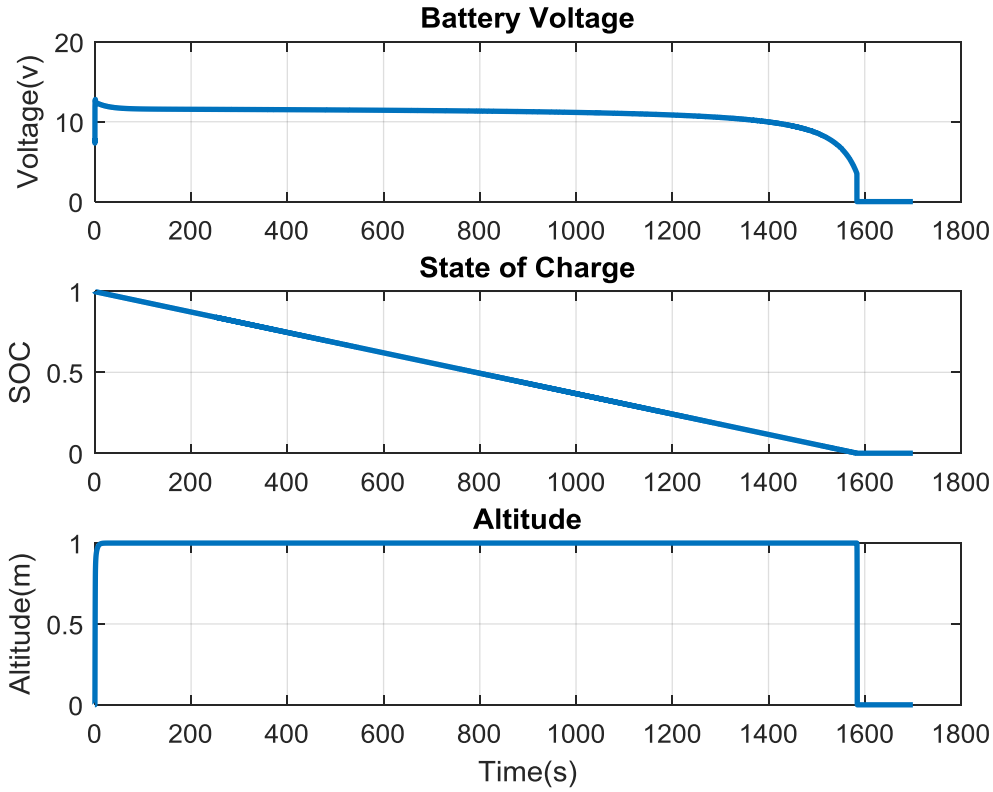


Figure 2.60-Battery and model dynamics

Chapter 3 - Quadcopter Sensors

The plant model represents the ideal response of the system given inputs, but in the real world that's not exactly what the quadcopter will do. There are many factors playing a role in the motion of the quadcopter such as wind, temperature, and approximations when creating the model. The feedback in quadcopters comes from several types of sensors. These include accelerometers and gyroscopes which are generally housed within what is called an Inertial Measurement Unit (IMU). The gyroscope provides angular velocity data and the accelerometer provides axial acceleration data. When modeling a quadcopter, the ideal form of sensors can be described as feed through, which means that the output of the plant goes directly to the controller. Real world modeling of the sensors includes sensor bias, noise, and delay additions.

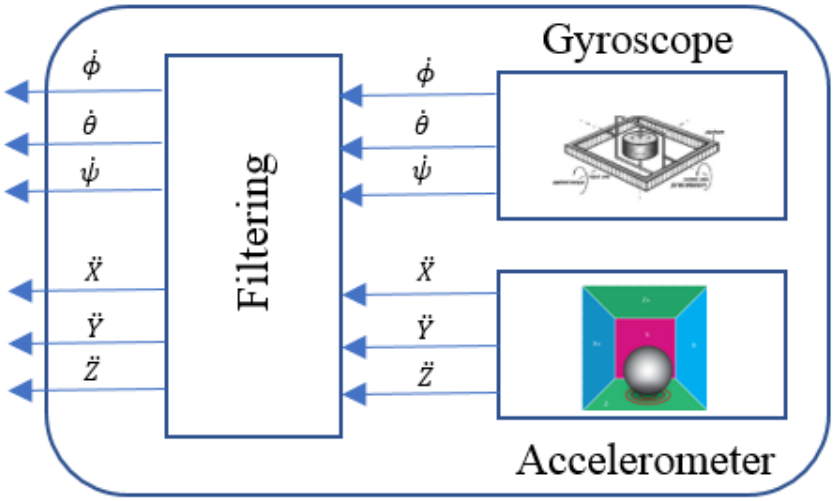


Figure 3.1-Quadcopter sensors

3.1 Gyroscope/Accelerometer Equations

The gyroscope/accelerometer are sensors that measures the three rotational velocities and three axial accelerations. This data is usually filtered, then fed into the controller. In essence these sensors provide $\ddot{X}, \ddot{Y}, \ddot{Z}, \dot{\theta}, \dot{\phi}$ and $\dot{\psi}$. The sensors are housed in the flight controller which is ideally placed in at the center of the quadcopter in order to provide the most accurate data. A very important aspect of using these sensors is initialization. Generally, these sensors will have a biased output, which means the reading isn't perfect. However, this bias can be removed by initializing the sensors. Imagine a quadcopter is sitting perfectly still on a table. Its known that the accelerometer should be reading zero acceleration in the X and Y direction, and 9.81 m/s^2 in the Z direction, and the gyroscope should read zero rotational velocities in all three axis. The sensors most likely will not be reading those values. Initialization is recording the bias and subtracting it from the measured accelerations.

These sensors also have a delay associated with them. This delay can be modeled as a low pass filter. Along with the low-pass filter, they have a bias gain associated with them and lastly noise. These aspects of the sensors can be added as seen in Equation (3.1) below.

$$Sensor\ Output = K_{Bias} + LowPass + Noise \tag{3.1}$$

3.2 Sensor Simulink Model

Because the model isn't taking in any raw data, the initialization process can't be demonstrated perfectly, however, the concept can be demonstrated. It's assumed that the input to the gyroscope and accelerometer come from the plant model. The MATLAB script and Simulink model for this section can be found in the folder titled 'Sensors'. This is a fair assumption for the gyroscope data, however the accelerometer data will be inaccurate. This is because the plant model uses a rotational matrix to transformation the acceleration to the local NED reference frame from the body frame. To achieve a more accurate model of the sensors the same rotational matrix must be used to transform back to the body frame. This equation can be seen below, where c represents cosine and s represents sine. This matrix multiplication will be covered further in Section 4.4.

$$\begin{bmatrix} \ddot{X}_{LLF} \\ \ddot{Y}_{LLF} \\ \ddot{Z}_{LLF} \end{bmatrix} = \begin{bmatrix} C\psi C\theta & S\psi C\phi + C\psi S\theta S\phi & S\psi S\phi - C\psi S\theta C\phi \\ -S\psi C\theta & C\psi C\phi - S\theta S\psi S\phi & C\psi S\phi + S\theta S\psi C\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} \ddot{X}_{Body} \\ \ddot{Y}_{Body} \\ \ddot{Z}_{Body} \end{bmatrix} \tag{3.2}$$

The Simulink model of this can be seen in Figure 3.2 below.

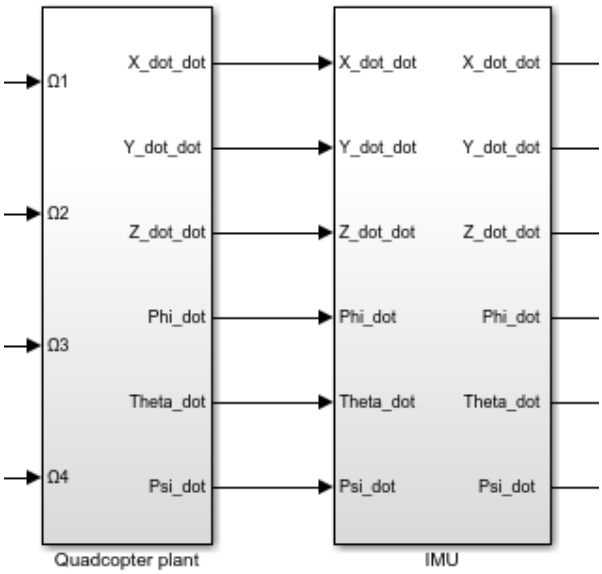


Figure 3.2-Accelerometer and gyroscope in model

First simulate just the plant model with zero inputs to generate the average of all the sensor outputs. Because this is an ideal model, the zero input sensor outputs will be ideal ($\ddot{X} = 0, \ddot{Y} = 0, \ddot{Z} = 9.81, \dot{\theta} = 0, \dot{\phi} = 0$ and $\dot{\psi} = 0$), however a bias can be added to the signals with a simple addition block. The next step is to record the outputs, then take the average and subtract the bias from the sensor output. This step can be seen in the gyroscope block below. Once this is done the sensors are initialized!

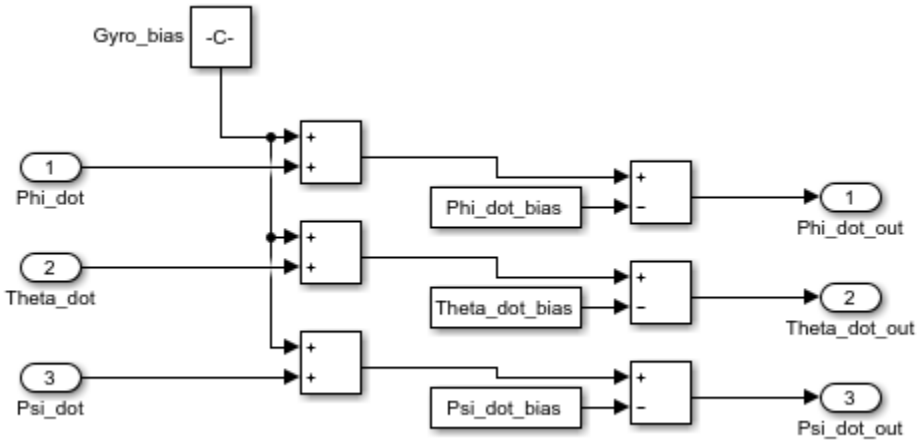


Figure 3.3-Bias subtraction

Actual IMU's have noise associated with their signals as well a bandwidth. This noise and filtering can also be simulated within the Simulink model. Normally distributed noise can be added to the accelerometer and gyroscope similarly to how the bias was added to them. This step can be seen in Figure 3.4 below.

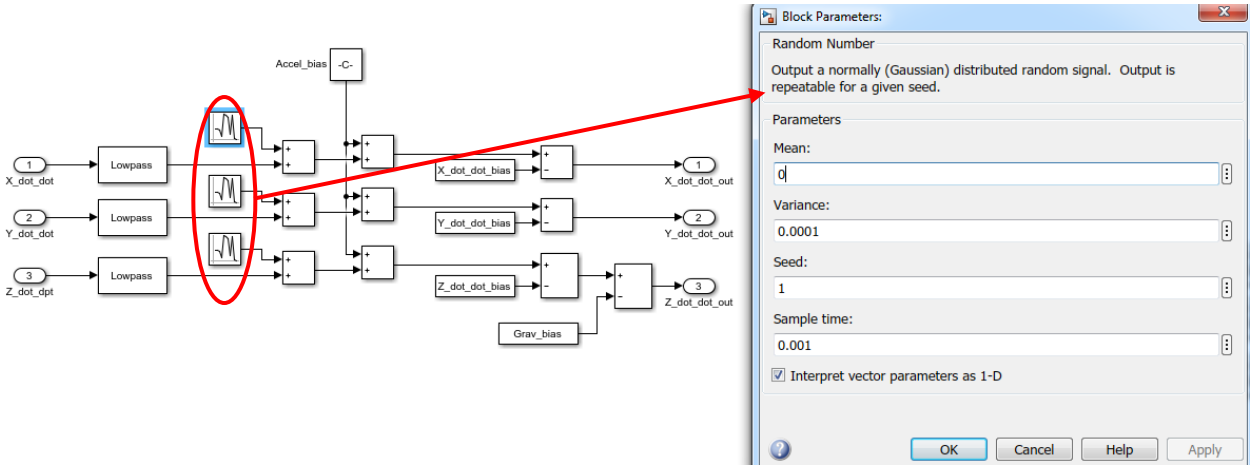


Figure 3.4-Addition of noise to IMU

Once the noise is added, the same steps as above can be done to unbias the sensors. Figure 3.5 & 3.6 below depicts the un-biasing of the IMU's.

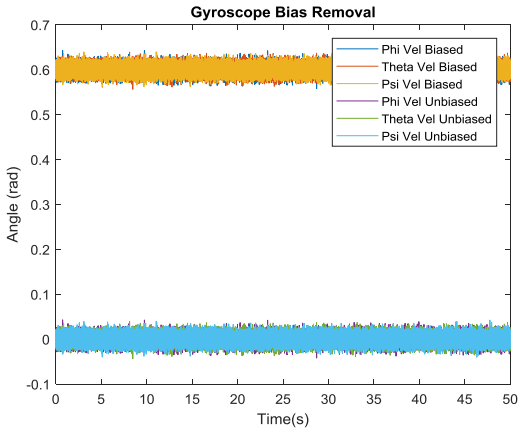


Figure 3.5-Gyroscope bias removal

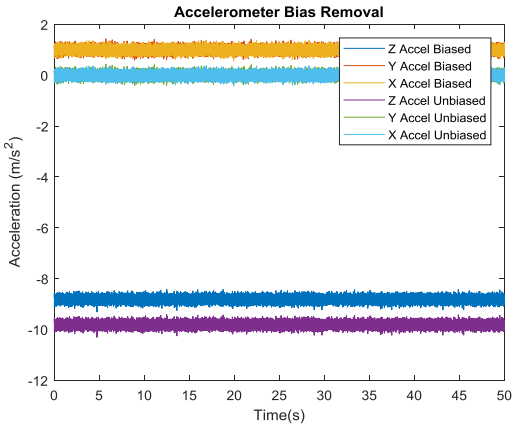


Figure 3.6-Accelerometer bias removal

What may come as a surprise, because the noise seems somewhat significant in Figures 3.5 & 3.6 above, is that the noise doesn't affect the model's outputs as much. The plant model outputs the 2nd order integral of the accelerometer and first derivative of the gyroscope. This means that the accelerometer data is put through two forms of low pass filters, and the gyroscope goes through one. The bode plot of a 2nd order integral filter can be seen in Figure 3.7 below.

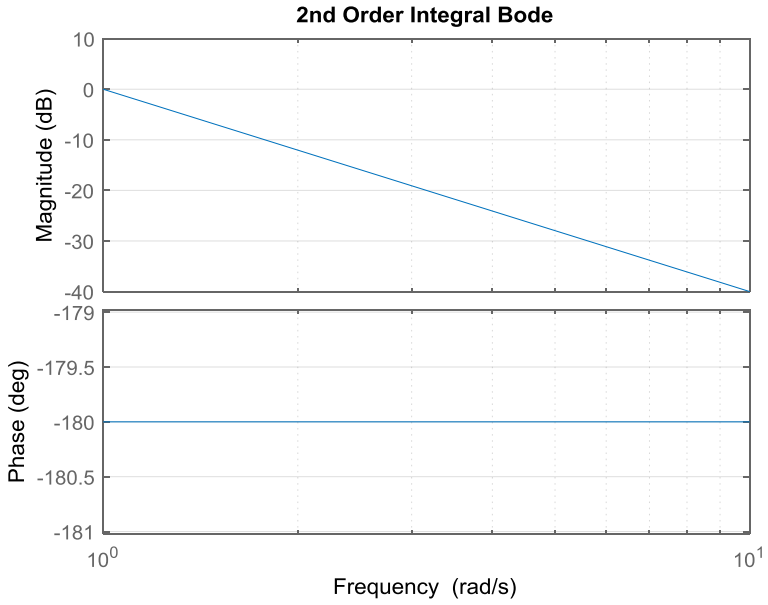


Figure 3.7-Second order pole FRF

What this means is that higher frequency noise seen from the gyroscope and accelerometer will have less power once it goes through the integrators and out of the plant model. This concept can be seen in Figures 3.8 & 3.9 below. There is less noise in the position, however, there is what is called drift noise. This means that the quadcopter wanders. The use of GPS can aid in the reduction of wander.

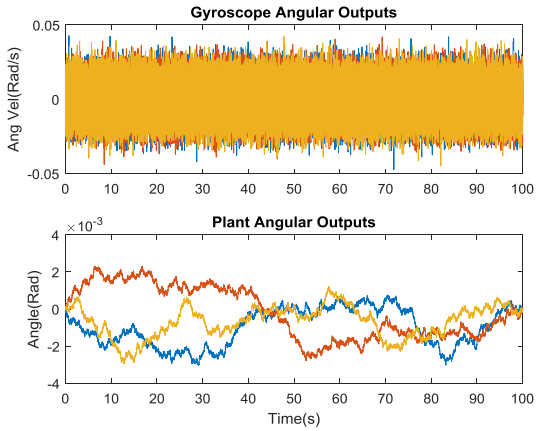


Figure 3.8-Gyrosocope output compared to plant output

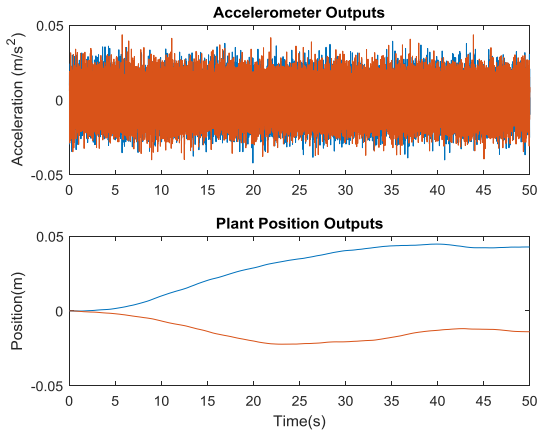


Figure 3.9-Accelerometer output compared to plant output

3.3 Complimentary filter

An interesting aspect of the accelerometer and gyroscope sensors is that the accelerometer can provide the same angle measurement as the gyroscope through basic trigonometry. Not only can the accelerometer provide the same angular position, it is more accurate at low frequencies than the gyroscope! Since both sensors can provide angular data, a filter is required to decipher what data to use for feedback. This is where the complimentary filter comes in. The filter uses two first order filters, one high pass and one low pass. The accelerometer data is ran through the low pass filter and the gyroscope data is ran through the high pass filter then the filters are summed such that high frequency data comes from the gyroscopes and low frequency data comes from the accelerometers. An illustration of this can be seen in Figure 3.10 below.

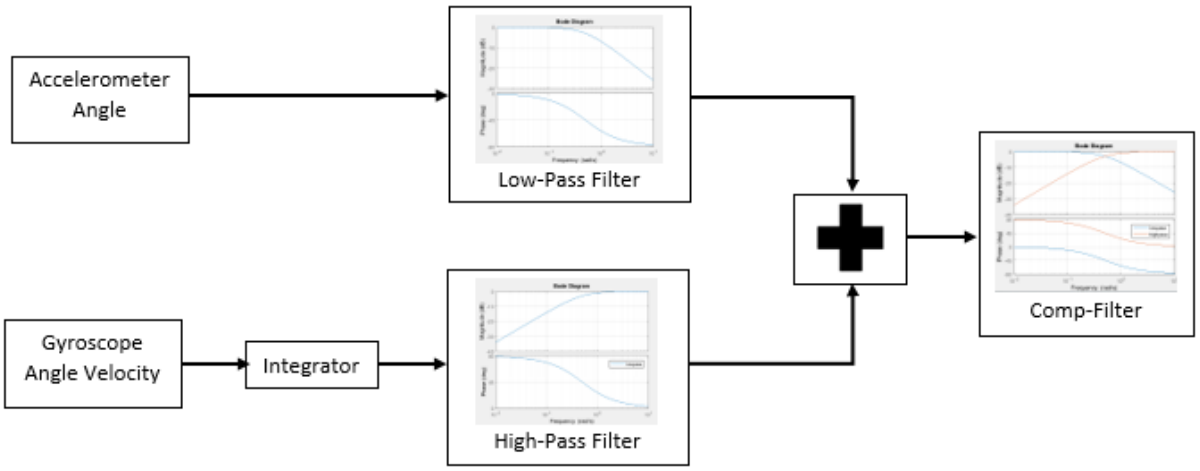


Figure 3.10-Complimentary filter illustration

Before this can be done the accelerometer equations need to be formed such that the rotational data can be generated. These can be seen in Equations (3.3) - (3.4) below. The accelerations are body reference accelerations.

$$\phi_A = \tan^{-1}\left(\frac{\ddot{Y}}{\ddot{Z}}\right) \tag{3.3}$$

$$\theta_A = \tan^{-1}\left(\frac{\ddot{X}}{\sqrt{\ddot{Y}^2 + \ddot{Z}^2}}\right) \tag{3.4}$$

With the high level model seen in Figure 3.10, along with Equations (3.3) and (3.4) the complimentary filter can be formed in Simulink. The first task that should be done is creating a new sub system that will contain the filter. The inputs for the subsystem should be the body reference accelerations, along with the gyroscope angular velocities. The outputs of the subsystem should also be body reference accelerations along with the three euler angles.

At this point, the euler angles need to be formed from the body reference frame accelerations. This can be done using a MATLAB function block. These blocks allow the use of code to execute functions. Using Equations (3.3) and (3.4) the function block contents can be seen in Figure 3.11 below.

```
function [Phi,Theta] = Accels2Angles(Accels)
X_dot_dot=Accels(1);
Y_dot_dot=Accels(2);
Z_dot_dot=Accels(3);

Phi=atan(-Y_dot_dot/Z_dot_dot);
Theta=atan(-X_dot_dot/sqrt(Y_dot_dot^2+Z_dot_dot^2));
```

Figure 3.11-Simulink MATLAB function block code snip converting acceleration to euler angle

Next comes the implementation of the complimentary filter. This is formed quite similar to the model seen in Figure 3.10. All that is needed is an integrator block and transfer function blocks. The continuous time equations for first order low and high pass filters can be seen in Equations (3.5) and (3.6) below where T is inversely proportional to the filters cut off frequency.

$$LowPass = \frac{1}{Ts + 1} \tag{3.5}$$

$$HighPass = \frac{Ts}{Ts + 1} \tag{3.6}$$

These high and low pass filters can be implemented into Simulink relatively easily. The complimentary filter Simulink model can be seen in Figure 3.12 below.

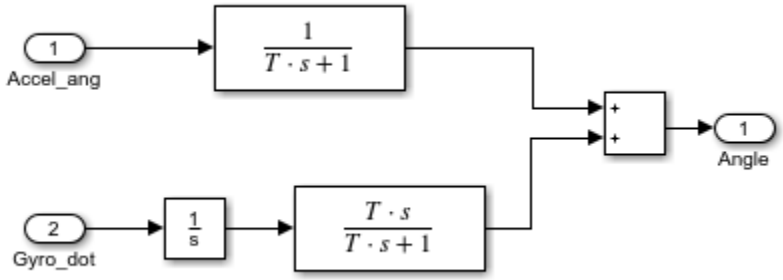


Figure 3.12-Complimentary filter Simulink implementation

With a filter being used for both the roll and pitch of the quadcopter, a simulation can be formed. Using a chirp signal, which varies frequency, a sinusoidal roll angle can be simulated. With the increase in

frequency the filter can be seen slowly change the weight of the accelerometer data vs the gyroscope data. This simulation can be seen in Figure 3.13 below.

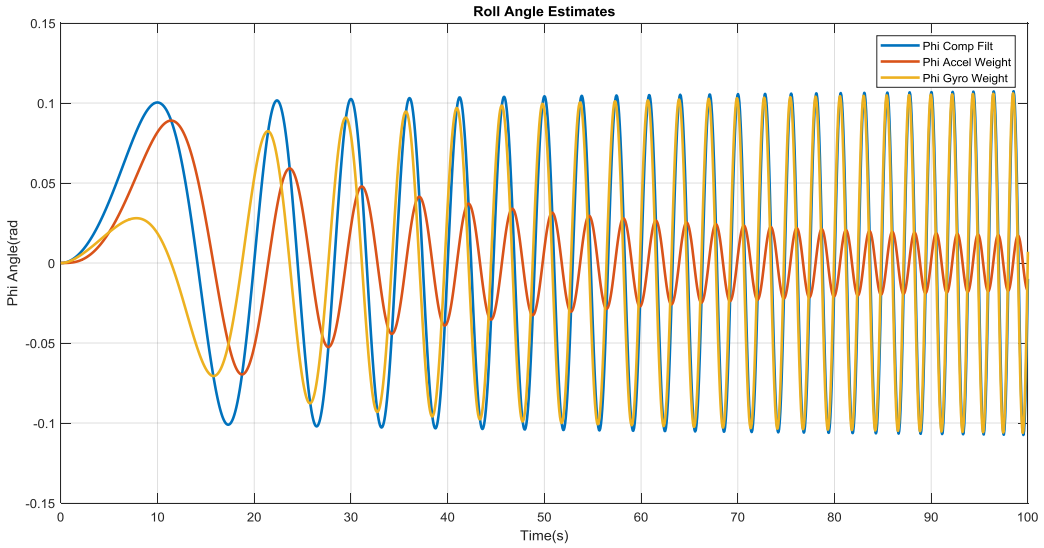


Figure 3.13-Complimentary filter simulation with 0.5 rad/s cutoff frequency

Real quadcopter data can also be used to see the effects of the complimentary filter. Raw quadcopter accelerometer and gyroscope data can be seen in Figures 3.14-3.15 below.

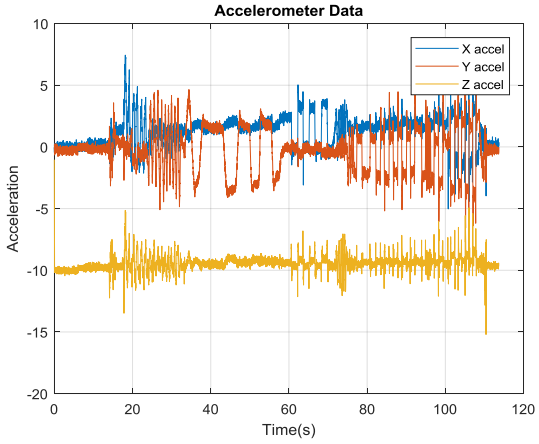


Figure 3.14-Raw accelerometer data

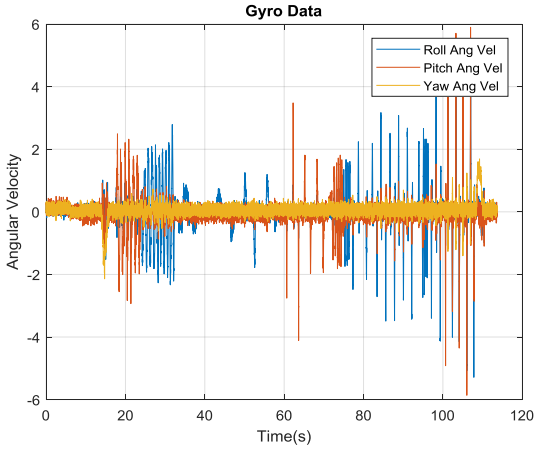


Figure 3.15-Raw gyroscope data

With this data, the complimentary filter can be implemented. The roll angle output of the complimentary filter an be seen in Figure 3.16 below with accelerometer and gyroscope weighting. It can be seen that the gyroscope takes over at high frequencies and the accelerometer takes over at lower frequencies.

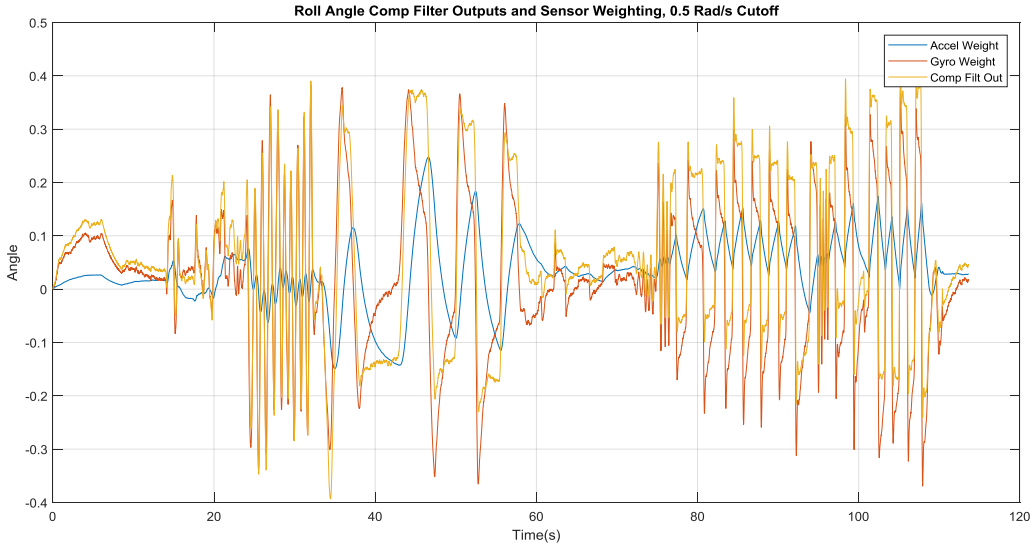


Figure 3.16-Roll angle complimentary filter output with 0.5 rad/s cutoff

Chapter 4 - Building the Controller

4.1 Controller

The controller is a very importation portion to any system as without it the system would inherently be unstable. There are many different control methods but for this section PID control is used. A high level model of a control system can be seen in Figure 4.1 below.

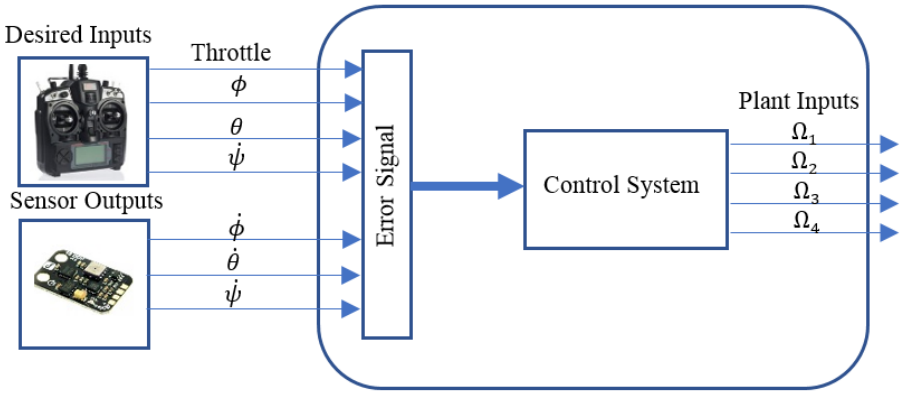


Figure 4.1-High level control architecture

The aspects that need to be controlled are altitude, roll angle, pitch angle and yaw velocity. Feedback from the plant model that was created is used to close the feedback loop. In many cases the user will be controlling throttle to the quadcopter opposed to altitude. However, it is rather difficult to maintain a constant altitude in simulation without a controller, thus one will be used. To use a PID controller a desired state, or setpoint is needed. For example, the desired altitude is 1. Next, the error signal needs to be created, which is done by subtracting the feedback from the desired state. The objective of the controller is to drive the error to 0. A good representation of the PID control scheme can be seen in Figure 4.2 below.

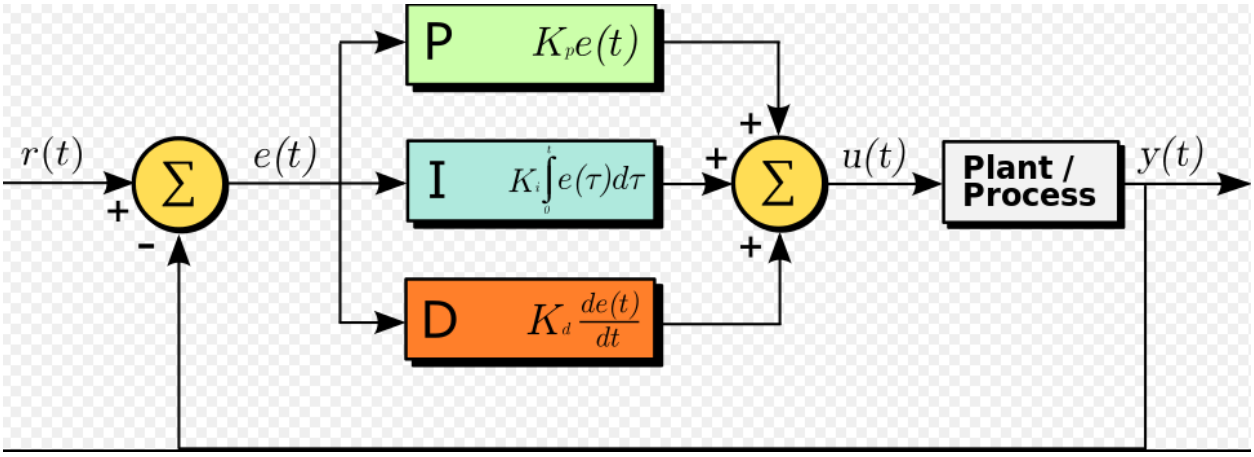


Figure 4.2-PID controller [14]

The Proportional part of the controller produces a general gain for the system. The Integral portion adjusts the gain by accumulating the error signal. The Derivate portion adjusts the gain by looking at the slope of the error signal, greater slope, greater contribution. Each portion of the PID controller sums to create the input for the plant model.

4.2 PID Controller

PID controllers can be represented as gains in Simulink, or as a continuous time transfer function. A transfer function is a mathematical way of describing the output of a system given an input. The continuous time transfer function can be formed using time domain data with the Laplace transform. A PID controller can be represented with the following transfer function.

$$G_c(s) = \frac{K_d s^2 + K_p s + K_i}{s} \tag{4.1}$$

The transfer function of a controller/plant model changes when feedback is introduced. Without feedback the system transfer function can be represented by multiplying the controller and plant transfer function. The closed loop transfer function is found by solving for the output of the system over input of the system. The open and closed loop transfer functions can be seen in Table 4.1 below.

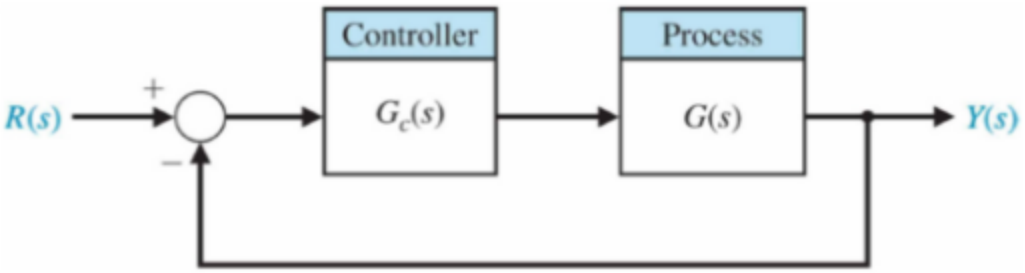


Figure 4.3-Feedback control system

Table 4.1-Open and closed loop transfer function

Open Loop	$H(s) = G_c(s)G_p(s)$	(4.2)
Closed Loop	$H(s) = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)}$	(4.3)

The transfer functions can be represented in Simulink using the transfer function block. To use the block, the coefficients of the numerator and denominator need to be known. Then the coefficients are simply put into vector form and input to the transfer function block.

The controller can be built, starting with another subsystem. It's known 10 inputs (X Acceleration, Y Acceleration, Z Acceleration, Desired Z, Phi Velocity, Desired Phi, Theta Velocity, Desired theta, Psi Velocity, Desired Psi Velocity), and 4 outputs ($\Omega_1, \Omega_2, \Omega_3, \Omega_4$) are needed. The feedback are the outputs of the sensors. The MATLAB scrip and Simulink model for this section can be found in the folder titled 'Controller_Plant'. At this point, on the highest level there should be 3 subsystems, a controller, a plant, sensors, and feedback going from the sensors to the controller. It should look like Figure 4.4 below.

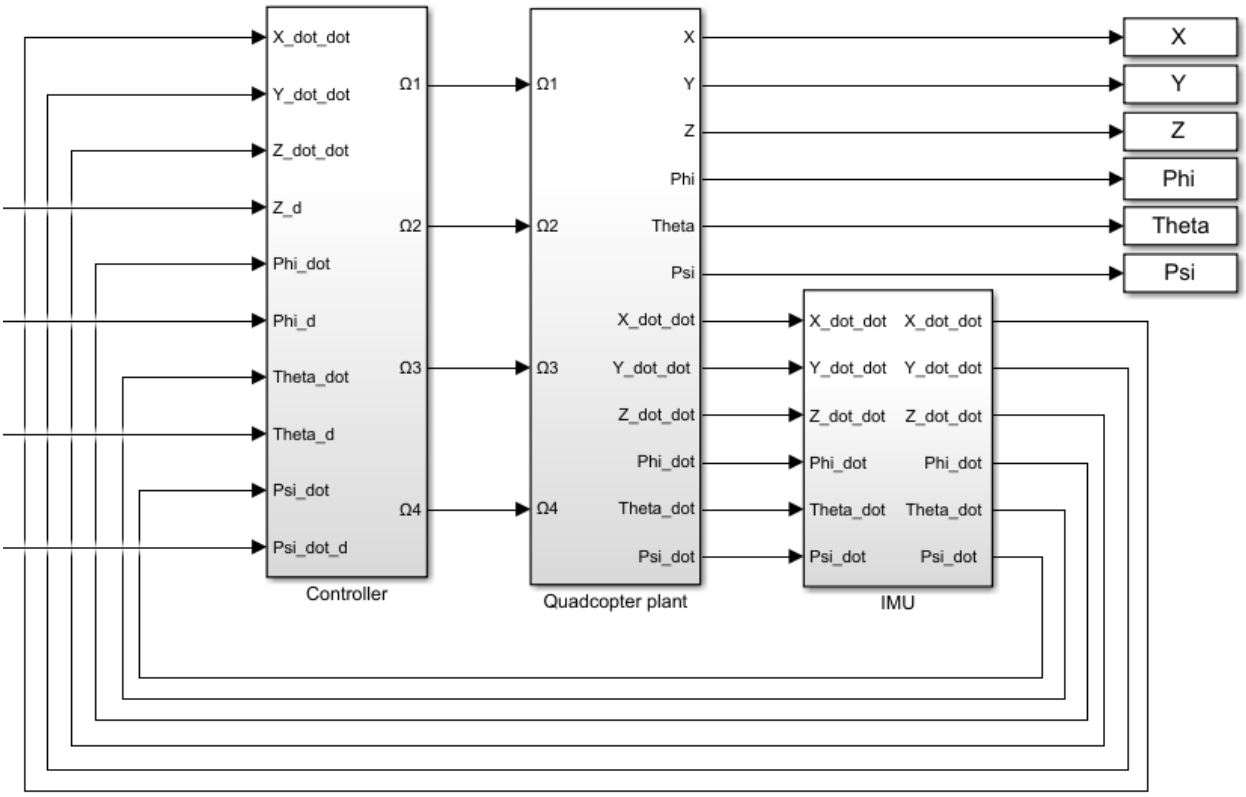


Figure 4.4- Controller- plant feedback

Inside the controller, a PID controller for each set of error signals is created. There are a few ways of creating PID controllers using Simulink. There is a PID block, but for demonstration reasons they will be created with the raw derivative, integral and gain blocks. They are formed very similarly to Figure 4.2 depicting PID controllers above. Figure 4.5 below shows the PID controller controlling the yaw of the system.

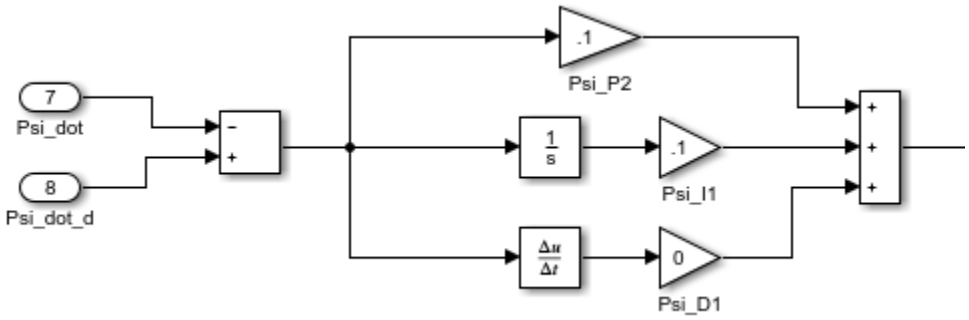


Figure 4.5-Simple PID controller

Now, create a PID controller for each set of inputs to the controller. An important portion of the PID to realize is what each portion of the controller does and knowing what input signals will be simulated. An input signal that is normally used as a desired signal is the step signal. One of the problems with the PID controller and step responses is that the derivative term can cause impulses in the system. Imagine at time zero, a step is applied to the altitude control. The error will be 1, the slope of the error will be vertical. Since the derivative calculates the slope of the line, this will cause an impulse in the PID control effort, which is undesirable. Because of this, a PI-D structure can be used, as this structure simply uses the slope of the state, and not the error [8]. To accomplish this with the sensor feedback, integrals need to be taken such that the desired state have the same units as the feedback states. Once all the controllers are created in the controller subsystem block, the next step is to separate out the angular velocities. This can be done by backing out of Equations (2.4) - (2.7) for the '+' configuration or Equations (2.8) - (2.11) for the 'x' configuration. Once this is done the controller subsystem should be like Figure 4.6 below.

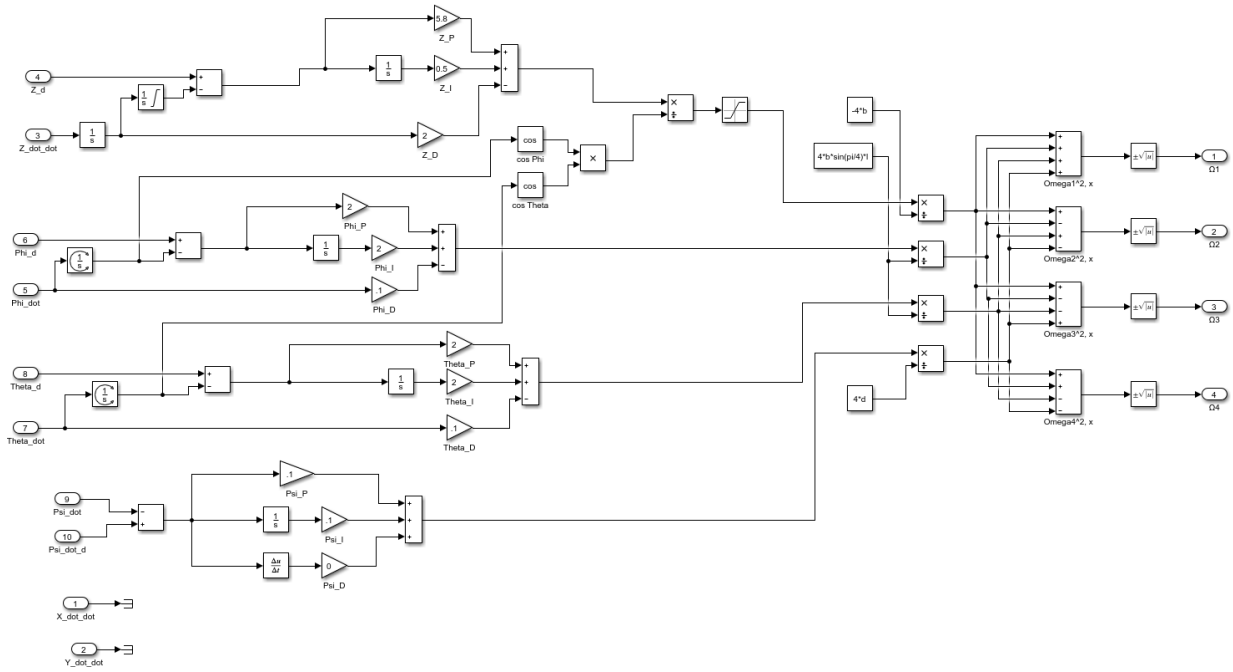
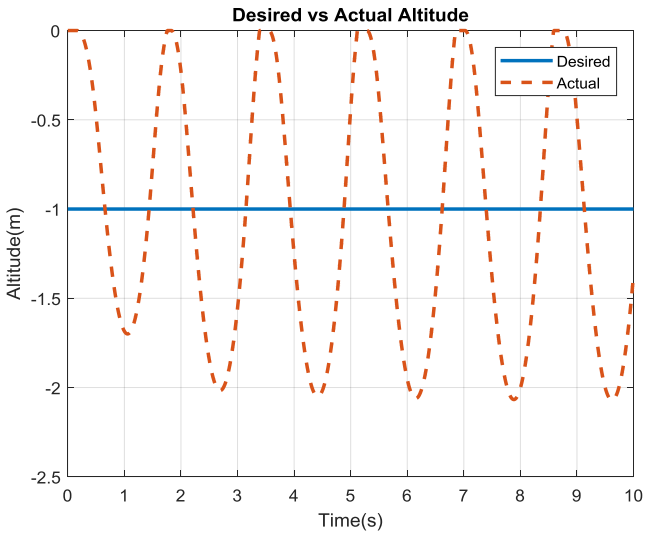
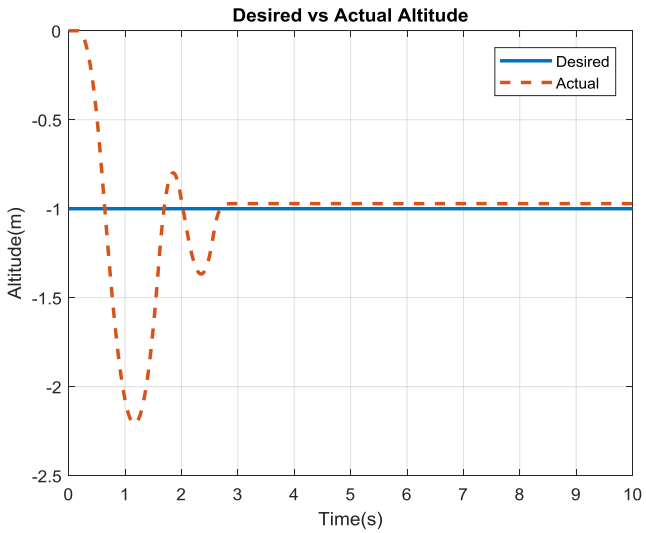


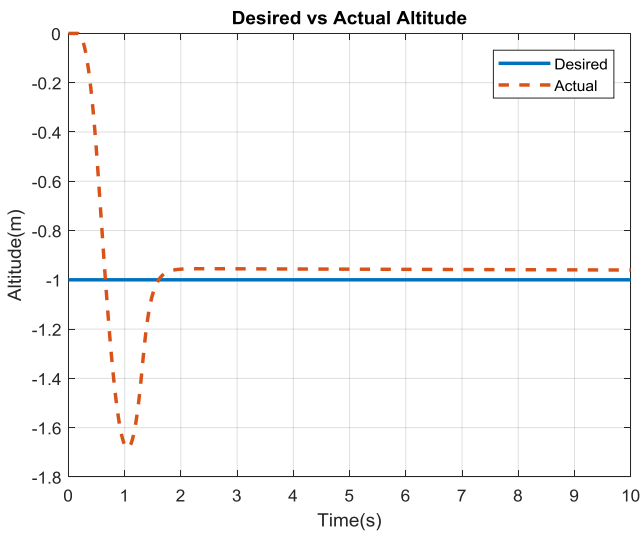
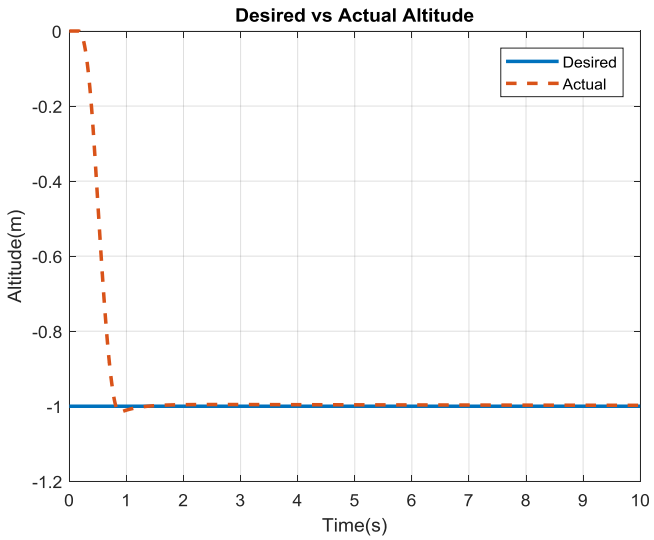
Figure 4.6-Phi, theta, psi, Z PID controllers

At this point inputs are ready to be applied and gains can be calculated for each controller. There are many methods that can be used to tune PID controllers. For this section, the controllers will be tuned manually.

It's known that the Proportional portion applies gain to the whole system, thus tuning will start with the proportional portion of the altitude controller. Increase the gain until the system rises fast enough. Then tune the Integral and Derivative gain such that the system converges. Lastly, fine tune the Integral and Derivative gains so desired characteristics are achieved. This can observe the tuning process in Table 4.2 below using a step input.

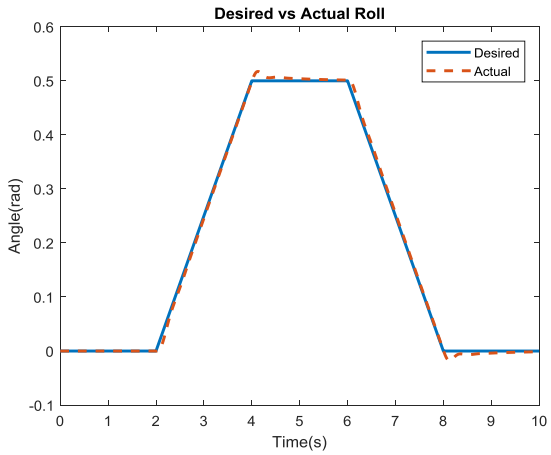
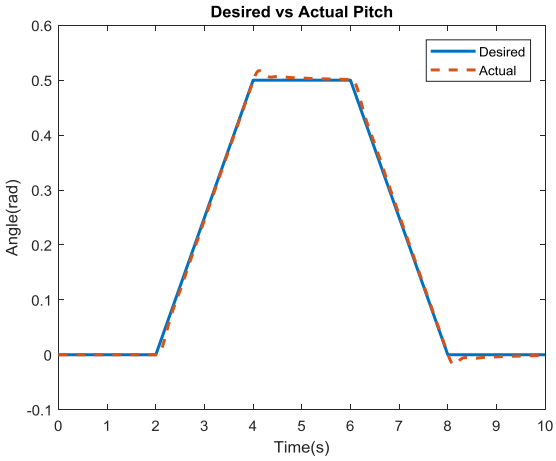
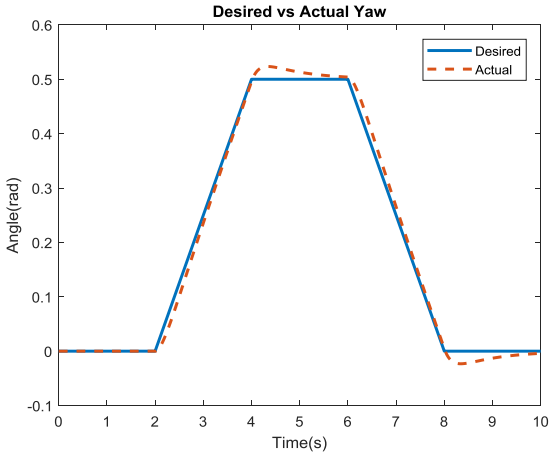
Table 4.2-PID turning process

P gain=1 I gain=0 D gain=0 $u(t) = 1e(t)$	 *Figure 4.7-Altitude control iteration 1*
P gain=10 I gain=0 D gain=1 $u(t) = 10e(t) - \frac{dz}{dt}$	 *Figure 4.8-Altitude control iteration 2*

P gain=6 I gain=0.1 D gain=1 $u(t) = 6e(t) + 0.1\int e(t)dt - \frac{dz}{dt}$	 *Figure 4.9-Altitude control iteration 3*
P gain=5.8 I gain=0.5 D gain=2 $u(t) = 5.8e(t) + 0.5\int e(t)dt - 2\frac{dz}{dt}$	 *Figure 4.10-Altitude control iteration 4*

Since a reasonable altitude response as been generated, tuning for roll angle, pitch angle and yaw velocity can now be done. These are tested using a limited ramp input. This is because in a real-world controller the user would gradually apply the input, opposed to a step. Table 4.3 below shows the tuned controller gains and their response.

Table 4.3-Manually tuned PID controllers

Phi	P gain=2 I gain=2 D gain=0.1 $u(t) = 2e(t) + 2\int e(t)dt - 0.1\frac{d\phi}{dt}$	 *Figure 4.11-Phi control response*
Theta	P gain=2 I gain=2 D gain=0.1 $u(t) = 2e(t) + 2\int e(t)dt - 0.1\frac{d\theta}{dt}$	 *Figure 4.12-Theta control response*
Yaw	P gain=0.1 I gain=0.1 D gain=0 $u(t) = 0.1e(t) + 0.1\int e(t)dt$	 *Figure 4.13-Yaw control response*

It is now possible to capture the state responses and motor RPMs. To prove the simulation is controlled and working properly, a trajectory can be simulated. The desired trajectory can be given as a starting altitude of 1 meter, then have the quadcopter go in a square formation, increase altitude by a meter, rotate, then go in the square formation again. The trajectory inputs and states can be seen below in Figure 4.14 below. The 3D plot of the trajectory can be seen below in Figure 4.15 and 4.16, proving the quadcopter is following commands correctly.

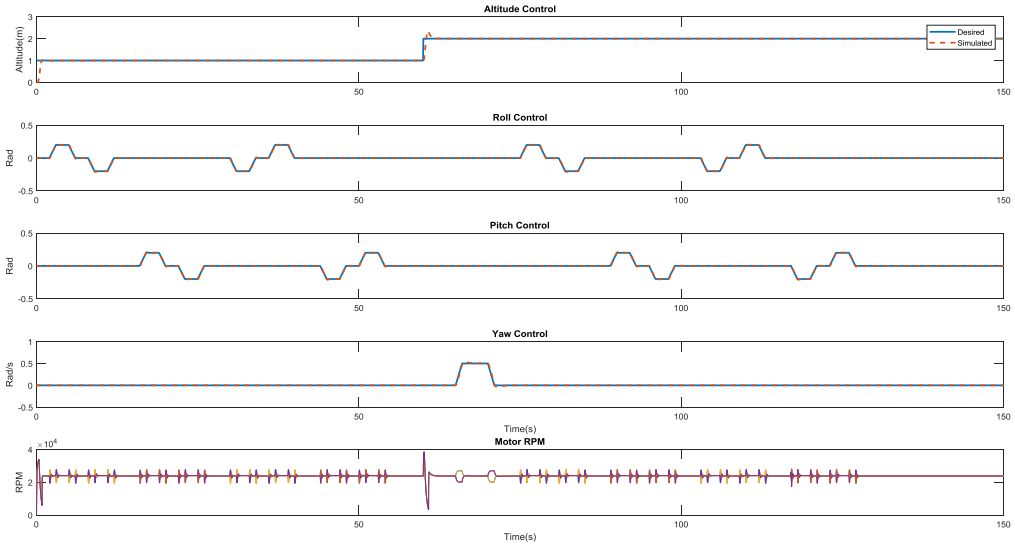


Figure 4.14-Model inputs and outputs

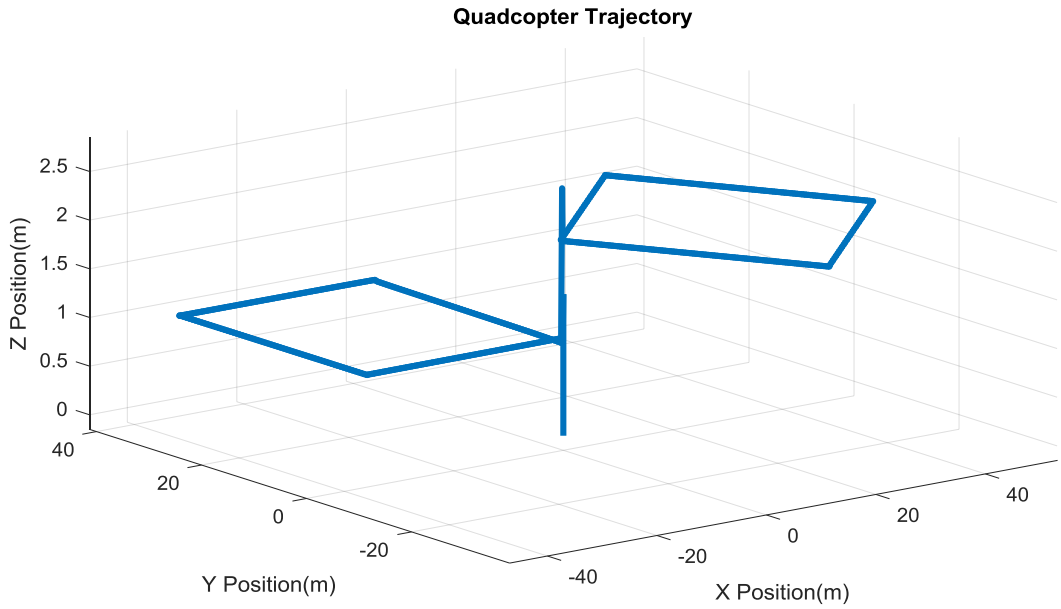


Figure 4.15-3D plot of quadcopter trajectory

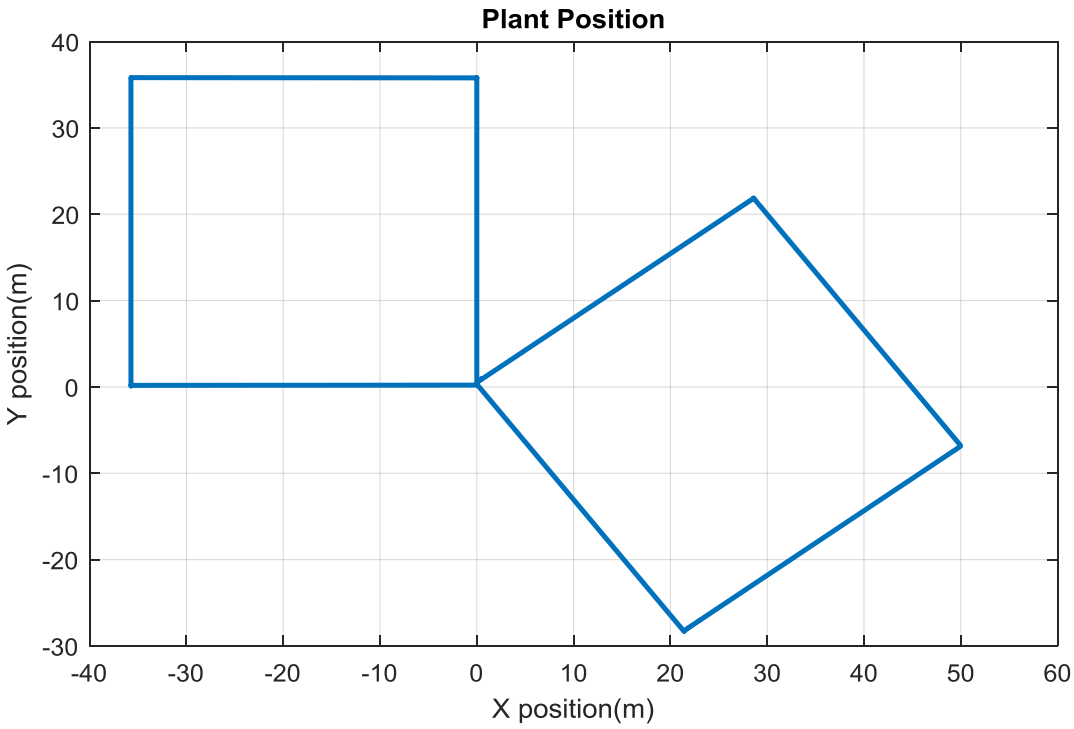


Figure 4.16-Top down view of quadcopter trajectory

An interesting portion that can be noted in the simulation is what the sensors of the system read versus what the plant model outputs. Since the plant model outputs accelerations, velocities, and positions relative to the local NED frame, and the sensors read body frame accelerations, velocities, and positions, the outputs can be significantly different. The sensor position output of the same simulation as seen in Figures 4.15 and 4.16 can be seen in Figure 4.17 below. It can be seen that the sensors only account for body frame accelerations.

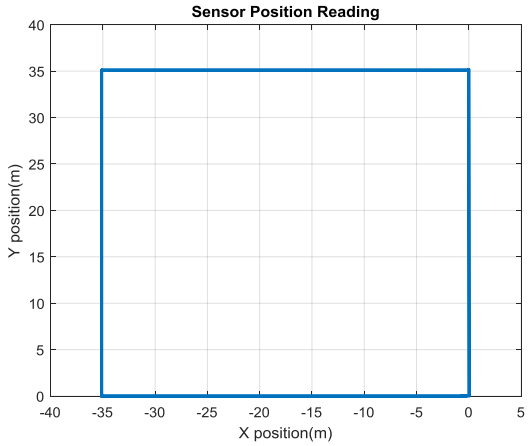


Figure 4.17-Sensor position reading

Simulation can also show the effects of air resistance. Applying a pitch angle, then removing it will show how air resistance slows and stops the quadcopter. This can be seen in the simulations in Figures 4.18 and 4.19 below.

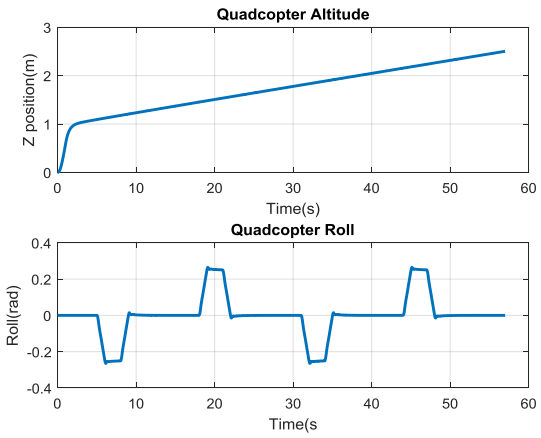


Figure 4.18-Quadcopter altitude and roll

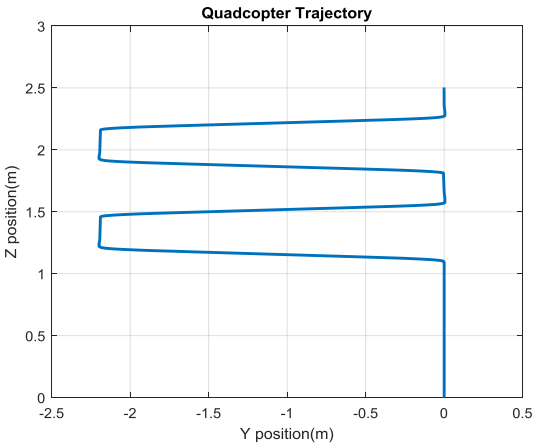


Figure 4.19-Quadcopter trajectory

4.3 Optimal PID Control

Although manual tuning in the case presented in section 4.2 seams relatively easy, it can become difficult with more complex systems. A method of finding PID gains that is more robust and automated is though numeric optimization. Through this numeric optimization the best results are obtainable under the specified conditions. When using PID control, the goal is to use the cost function, in order to minimize the error between the output and desired stated. To start, define a cost function, which can be seen below.

$$J = \int_0^{tf} [Qe^2(t) + Ru^2(t)]dt \tag{4.4}$$

Where $e(t)$ is the error between the desired state and the actual state, $u(t)$ is the input to the plant, and Q & R are designed weighting functions. If too much controller efforted is observed, increase R, which penalizes the input. Choosing Q and R is an iterative process to obtain the desired transient response of the system. It is worthy to note that the actual quantity given to Q and R doesn't matter as much as the ratio between the two. In order to implement the cost function and obtain optimal PID controller gains an optimal function in MATLAB must first be created.

The goal of the function is to return the product of the cost function. The MATLAB scrip and Simulink model for this section can be found in the folder titled 'Optimal_PID'. To this set up a MATLAB function and define global gain variables $K_p, K_i,$ and $K_d,$ which will all be part of the input vector to the function. Next, simulate the Simulink controller/plant model and record several variables. Record the desired input, the input to the controller, and the output of the plant. To do this, set up some output blocks in the Simulink model. This can be seen in Figure 4.20 below.

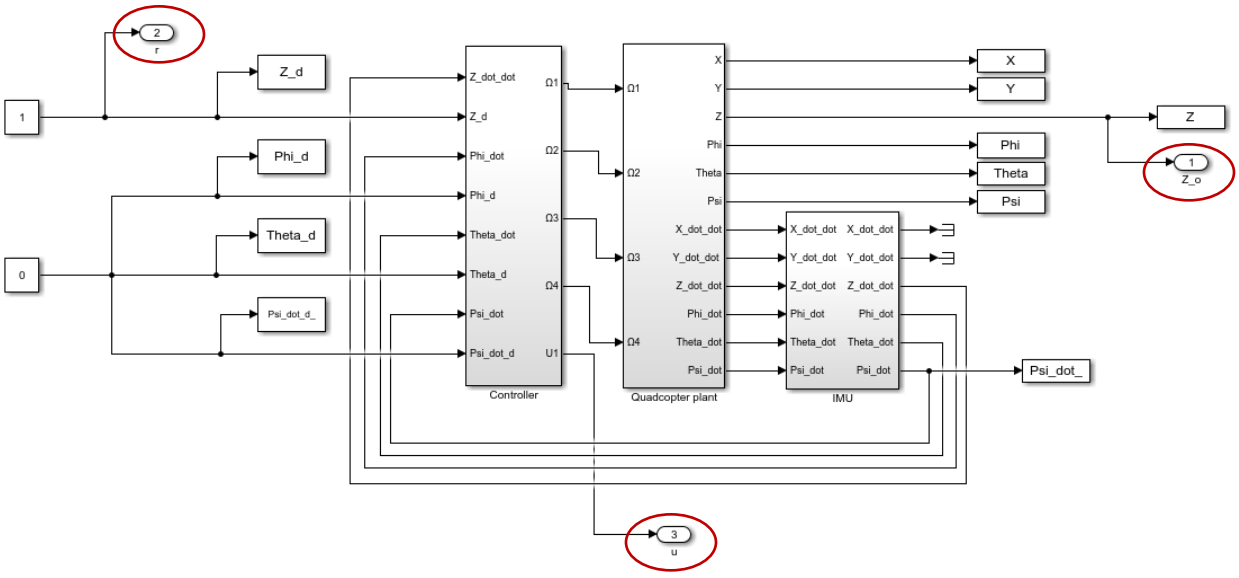


Figure 4.20-Simulink model used for optimal controller

Next, the error signal needs to be generated by subtracting the state from the desired input. The input to the plant within the function also need to be defined. Lastly, set up the cost function and define initial Q and R values. Here, the $trapz()$ MATLAB function is used. The total function can be seen in Figure 4.21 below.

```
function [ J ] = OptPID( x )
global Kp Ki Kd

Kp=x(1);
Ki=x(2);
Kd=x(3);

[Tsim,Xsim,Ysim]=sim('Q_M');

Q=10000;
R=1;
error=(Ysim(:,2)-Ysim(:,1));
U=Ysim(:,3);
J=trapz(Q*error.*error+R*U.*U)*0.01;
end
```

Figure 4.21-Optimal PID function

Lastly, in the main MATLAB script that holds the constants that are used in the Simulink model, define initial gain values. Then use the $fminsearch()$ MATLAB function to find the minimal cost PID gains. This portion of the MATLAB script can be seen in Figure 4.22 below. Once the script is run, the

$fminsearch$ function will return gain values. Using the gain values simulate the model and see if a desired transient output occurs. If not, iterate through other Q to R ratios or increase or decrease initial gain values.

```
global Kp Ki Kd
format short e
Kp=100;
Ki=100;
Kd=100;
x0=[Kp; Ki;Kd];
x=fminsearch('OptPID',x0)
Kp=x(1);
Ki=x(2);
Kd=x(3);
```

Figure 4.22-Optimal PID function code snip

To demonstrate this a few examples controlling the altitude can be seen below.

Table 4.4-Optimal PID control example

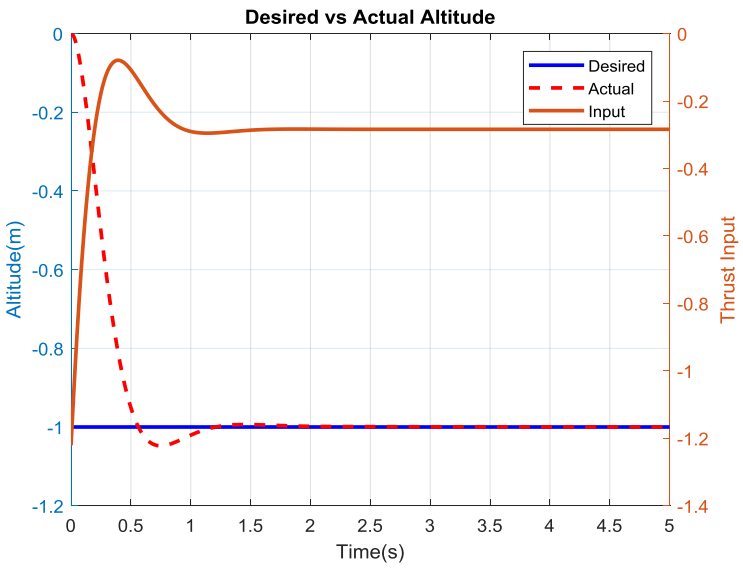
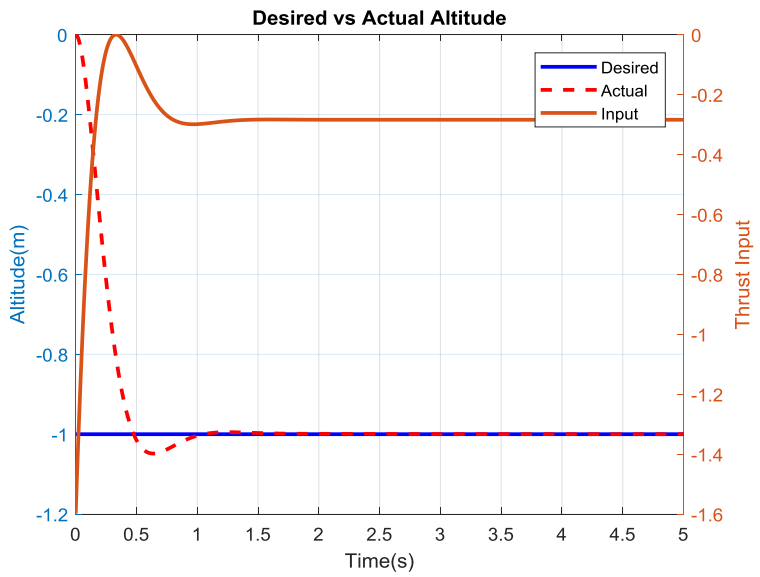
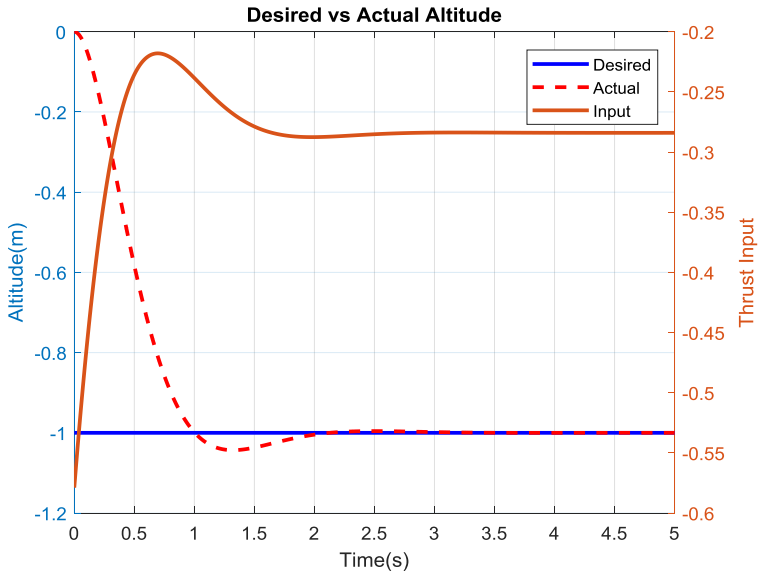
Initial K_p=1 Initial K_i=1 Initial K_d=1 Q=1 R=1	Final K_p=1.22 Final K_i=1.14 Final K_d=0.2589	

Figure 4.23-Altitude optimal PI-D response and thrust input Q=1, R=1

Initial K_p=1 Initial K_i=1 Initial K_d=1 Q=2 R=1	Final K_p=1.5949 Final K_i=1.3662 Final K_d=0.2919	 *Figure 4.24-Altitude optimal PI-D response and thrust input Q=2, R=1*
Initial K_p=1 Initial K_i=1 Initial K_d=1 Q=1 R=10	Final K_p=0.5789 Final K_i=0.6486 Final K_d=0.1859	 *Figure 4.25-Altitude optimal PI-D response and thrust input Q=1, R=10*

This optimal PID gain process can be used for the other controllable states of the quadcopter. This is a good method to find gains with nonlinear system. This is a good method to find gains that will produce a stable system such that the control effort doesn't reach system limits.

4.4 Controller, Plant Vectorization

Along with the method of modeling the controller presented, a model can be created using vectors and matrices. This method is less intuitive than the previous method, however it is more concise.

4.4.1 Plant Model Matrix Equations

The first matrix to introduce is the torque matrix. This is the torque on each of the axis' caused by the rotor thrust. The torque equations are similar to the $U2 - U4$ thrust equations seen in Chapter 2, however these simply have a length-from-axis multiplication, which infers torque. This equation can be seen below.

$$\tau_{Bx} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} sin\left(\frac{\pi}{4}\right) lb\left({\Omega_1}^2 - {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2\right) \\ sin\left(\frac{\pi}{4}\right) lb\left({\Omega_1}^2 + {\Omega_2}^2 - {\Omega_3}^2 - {\Omega_4}^2\right) \\ d\left({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2\right) \end{bmatrix} \tag{4.5}$$

The next matrix to introduce is the thrust matrix. A portion of this matrix was also introduced in Chapter 2, however this is in vector form. The only thrust applied to the quadcopter is in the Z axis.

$$T_B = \begin{bmatrix} 0 \\ 0 \\ b\left({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2\right) \end{bmatrix} \tag{4.6}$$

Section 3.2 briefly introduced a transformation matrix. A transformation matrix is required to use different frames of reference. This transformation matrix is comprised of rotation matrixes for each axis. The rotation matrix for each axis can be seen in Equations (4.7)-(4.9) below.

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{4.7}$$

$$R_\theta = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & cos\theta \end{bmatrix} \tag{4.8}$$

$$R_\psi = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

These rotational matrixes can be multiplied in the Z-Y-X direction to create a transformation matrix that is able to transform the X, Y, Z coordinate system from a body frame to a local NED frame. This transformation matrix can be seen in the equation below where C represents cosine and S represents sine.

$$R = R_{\psi}R_{\theta}R_{\phi} = \begin{bmatrix} c\psi c\theta & s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi - c\psi s\theta c\phi \\ -s\psi c\theta & c\psi c\phi - s\theta s\psi s\phi & c\psi s\phi + s\theta s\psi c\phi \\ s\theta & -c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{4.10}$$

Now the inertial acceleration equation can be created. This is comprised of the rotational matrix, thrust matrix and gravitational matrix. This equation is equivalent to Equations (2.23), (2.25), and (2.27).

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \frac{1}{m}\begin{bmatrix} C\psi C\theta & S\psi C\phi + C\psi S\theta S\phi & S\psi S\phi - C\psi S\theta C\phi \\ -S\psi C\theta & C\psi C\phi - S\theta S\psi S\phi & C\psi S\phi + S\theta S\psi C\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ b\left(\Omega_1{}^2 + \Omega_2{}^2 + \Omega_3{}^2 + \Omega_4{}^2\right) \end{bmatrix} - \frac{1}{m}\begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix}\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} \tag{4.11}$$

The final matrix required to build the plant model is the angular acceleration matrix. This matrix is comprised of the moments of inertia, previous angular velocity, and torque matrix.

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})\dot{\theta}\dot{\psi}/I_{xx} \\ (I_{zz} - I_{xx})\dot{\phi}\dot{\psi}/I_{yy} \\ (I_{xx} - I_{yy})\dot{\theta}\dot{\phi}/I_{zz} \end{bmatrix} - J_r\begin{bmatrix} \dot{\theta}/I_{xx} \\ -\dot{\phi}/I_{yy} \\ 0 \end{bmatrix}\Omega_r + \begin{bmatrix} \tau_{\phi}/I_{xx} \\ \tau_{\theta}/I_{yy} \\ \tau_{\psi}/I_{zz} \end{bmatrix} \tag{4.12}$$

4.4.2 Plant Model Simulink in Matrix Form

The MATLAB Scrip and Simulink model for this section can be found in the folder titled 'Vectorized_Controller_Plant'. Creating a new plant model subsystem, the first matrix to create is the matrix seen in Equation (4.5). One row matrices can be formed using mux blocks, and can be deformed using demux blocks. Since only angular velocities are inputs to the plant model, they'll need to be squared to form the equation. The next matrix to form is the matrix seen in Equation 4.6, which is built similarly to the torque matrix. This step can be seen in Figure 4.26 below.

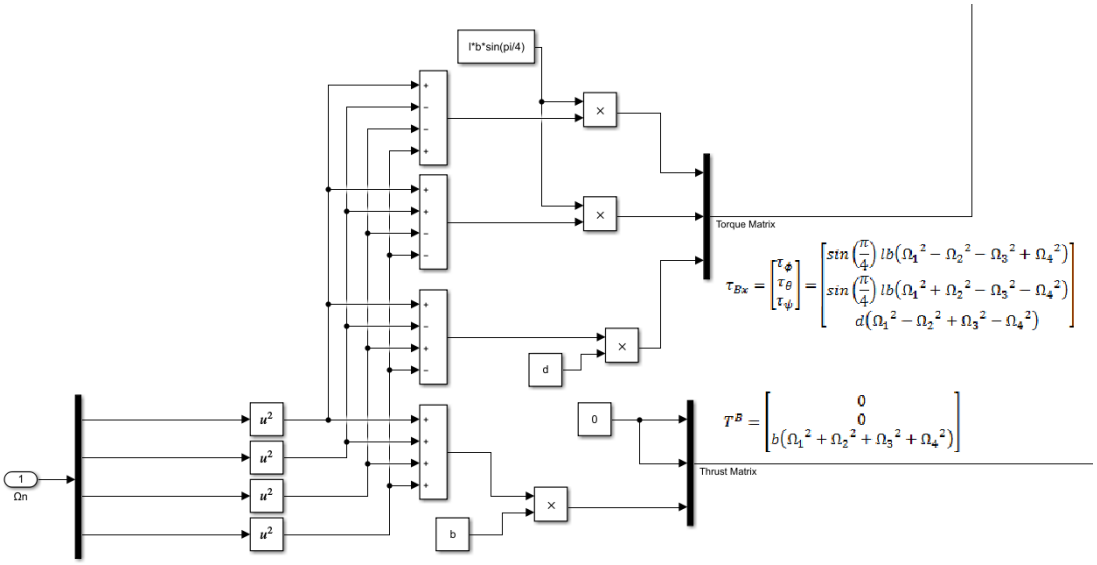


Figure 4.26-Torque and thrust matrix formation

The next step is to form the matrix seen in Equation (4.12). This can be done by using a MATLAB function block, which allows the user to write code to create the output. This step can be seen in Figure 4.27 below.

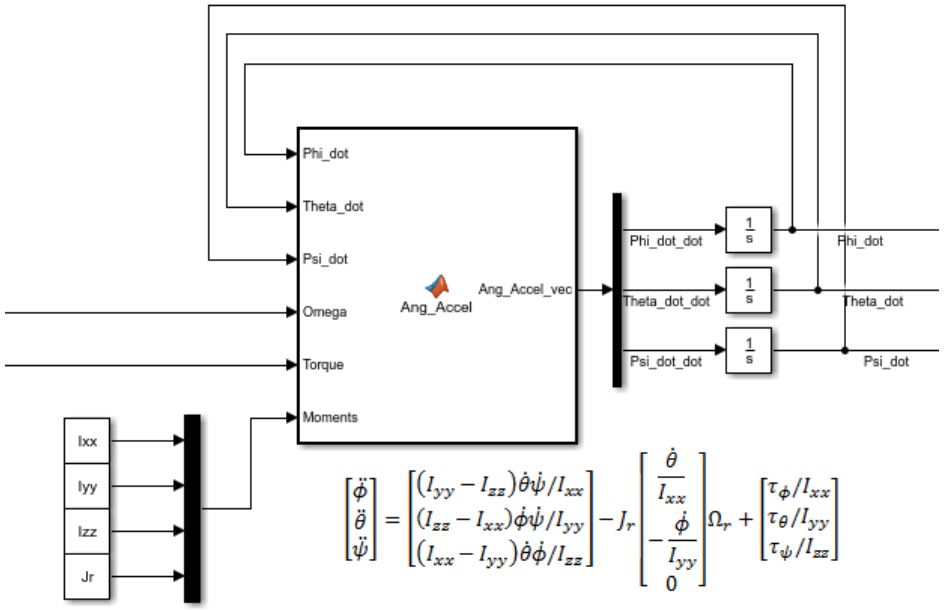


Figure 4.27-Angular acceleration matrix formation

```
function Ang_Accel_vec = Ang_Accel(Phi_dot,Theta_dot,Psi_dot,Omega,Torque,Moments)
Ixx=Moments(1);
Iyy=Moments(2);
Izz=Moments(3);
Jr=Moments(4);
Body_gyro_effect=[((Iyy-Izz)*Theta_dot*Psi_dot/Ixx);...
       ((Izz-Ixx)*Phi_dot*Psi_dot/Iyy);...
       ((Ixx-Iyy)*Phi_dot*Theta_dot/Izz)];
Rotor_inertia=[Theta_dot/Ixx;...
       -Phi_dot/Iyy;...
       0];
Actuator_action=[Torque(1)/Ixx;...
       Torque(2)/Iyy;...
       Torque(3)/Izz];

Ang_Accel_vec=Body_gyro_effect-(Jr.*Rotor_inertia.*Omega)+Actuator_action;
```

Figure 4.28-Angular acceleration matrix MATLAB function block

The next step is to create the rotation matrix in order to form the matrix equation seen in Equation 4.11. This can be done using the same method seen in Figure 4.27 and 4.28 above. Once this has been done Equation 4.11 can be created, which can be done by using the matrix multiplication block. Once this has been done integrals can be taken to get the local NED frame accelerations. The final plant model can be seen in Figure 4.23 below.

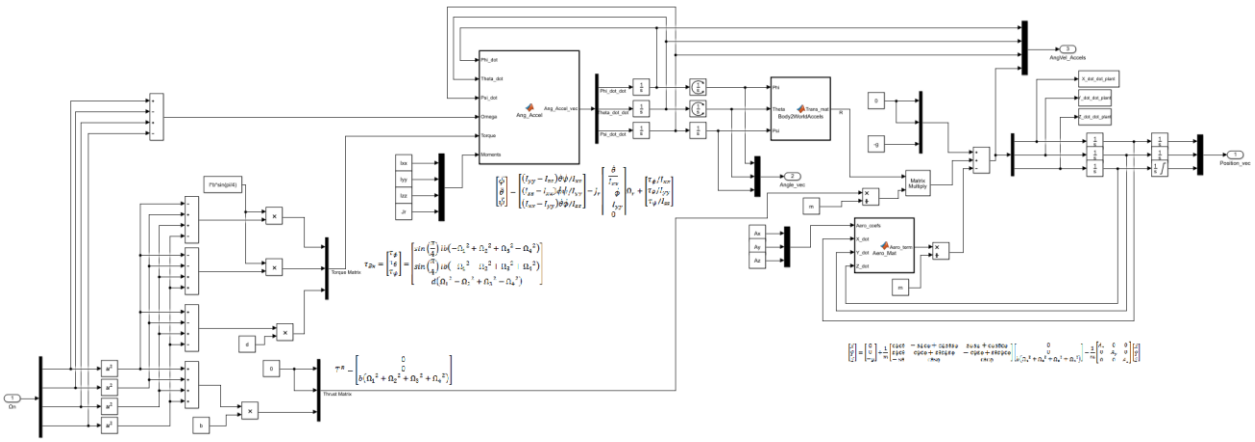


Figure 4.29-Vectorized Simulink plant model

A similar vectorization method can be applied to the controller and sensor blocks. Once this is done the system can run simulations. Figure 4.30 below shows the quadcopter simulating flying in circular pattern.

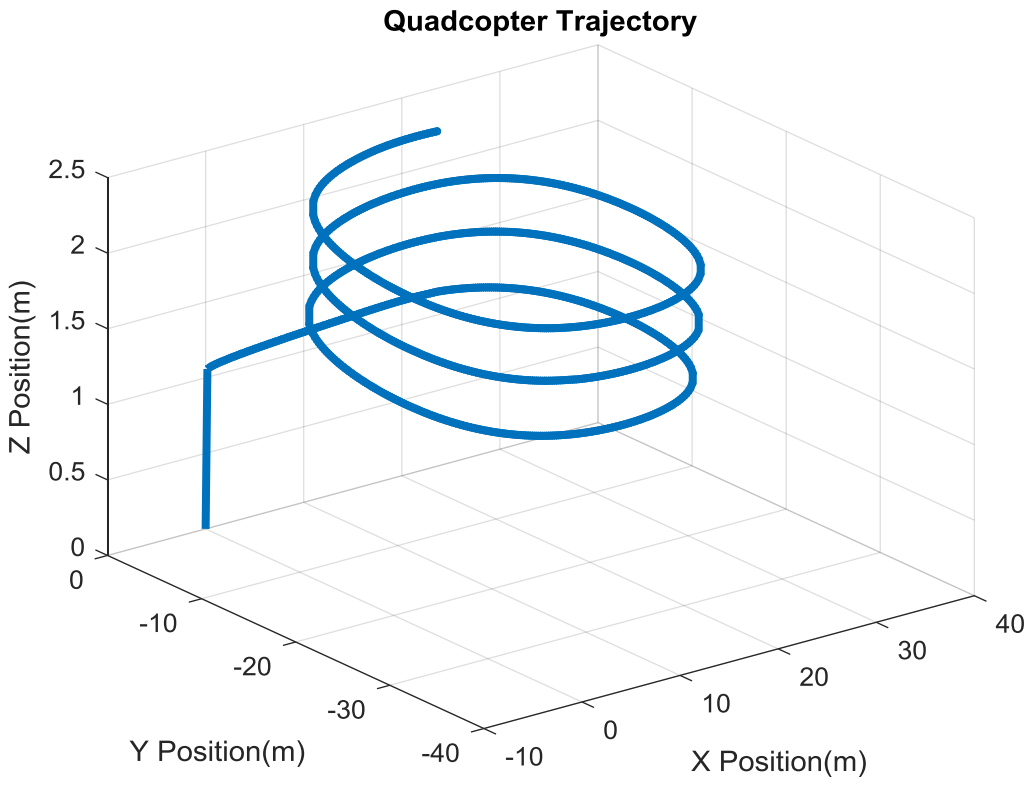


Figure 4.30-Quadcopter simulation, circular trajectory

4.5 Real World Controller

Up to this point a controller has been generated for the given plant model. The differences between the controller for the simulation differs only slightly from a controller that should be loaded onto a quadcopter. The main difference comes from user inputs along with the general thrust control. The controller made for the simulation used a PI-D controller for altitude. This would not be used in a real-world controller, instead a throttle command would be used. The user inputs in a real-world controller are also limited. The throttle is limited from 0 to 1, and roll, pitch and yaw are limited from -1 to 1. To encompass these restrictions simple saturation and gain blocks will need to use used.

4.5.1 Real World Controller Simulink Model

To create a real-world controller, the controller subsystem from section 4.2 can be modified. The first thing that needs to be done is to create a throttle input. Start by removing the altitude PI-D controller, and inputs, and replacing them with a throttle input, a saturation block and a gain block. The saturation block should go from 0 to 1, and the gain block should be dependent on the max thrust the motors can produce. From the motor modeling section its known that the maximum thrust is roughly 0.2 kg per motor. Thus, the maximum thrust produced from the entire quadcopter (4 motors * 0.2 kg) is 0.8 kg. The gain block for the throttle should be set to 0.7. This is so the motor doesn't operate at its full potential.

Next are the roll and pitch commands. The only blocks that need to be added to these sections in the controller are saturation and gain blocks. The saturation block should be limited from -1 to 1 so the quadcopter can move in the positive and negative angular direction. The gain block should be set such that the limit is the maximum desired angle. If the greatest roll or pitch angle that should be allowed is $22.5^{\circ}\ or\ \frac{\pi}{8} rad$, the gain should also equal $\frac{\pi}{8}$.

Lastly the yaw velocity controller also simply needs a saturation and gain block. The saturation should be limited from -1 to 1 so the quadcopter can rotate in both the clockwise and counter clockwise direction. The gain should be set such that the limit of the input is the maximum rotational velocity that is desired. If the maximum angular velocity for the quadcopter should be $45^{\circ}\ or\ \frac{\pi}{4} rad/s$, the gain should also be set to $\frac{\pi}{4}$. The entire real-world controller subsystem can be seen in Figure 4.31 below.

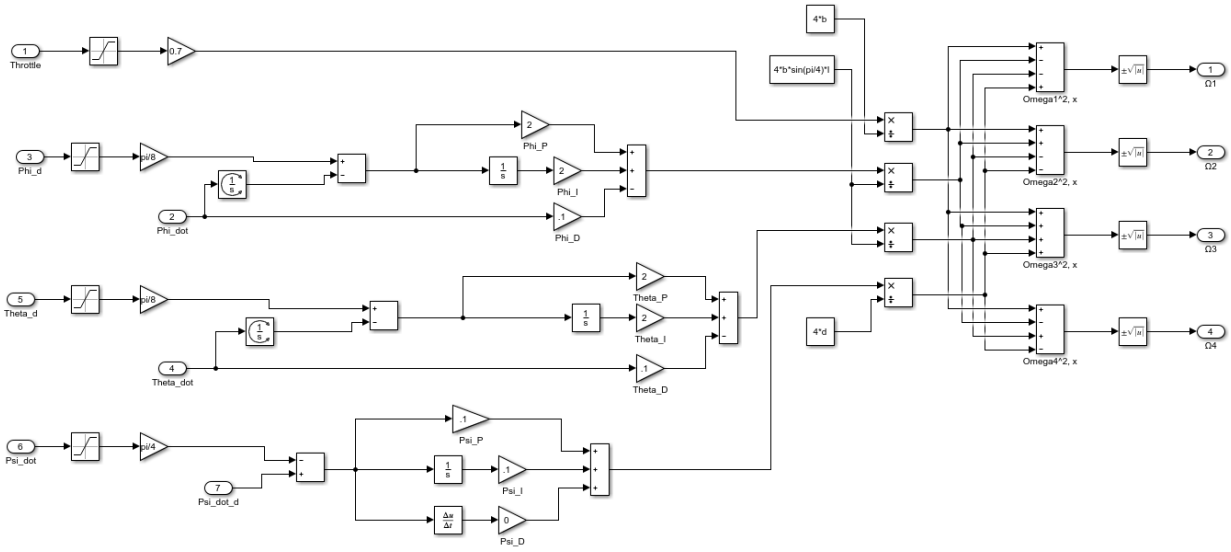


Figure 4.31-Real-world controller

Chapter 5 Digital Domain

Up to this point most of what has been created has been in the continuous time domain. In reality the only portion of the controller/plant/sensor combination that is actually in the continuous time domain is the plant model. This is because that represents simple physics, where the rest of the model represents executions within processors, which is inherently in the digital domain. In order to implement the system in the digital domain a few concepts need to be understood.

5.1 Converters

There are two types of converters that need to be defined, the first being an analog-to-digital converter and the second being digital-to-analog converter. Starting with the analog-to-digital converter, it takes an analog signal as an input, such as a continuous signal from an accelerometer, samples the signal, and holds the value until the next time sample. This process outputs a discrete time and discrete amplitude signal which is limited to the amount of bits the converter has. This process can be seen in Figure 5.1 below.

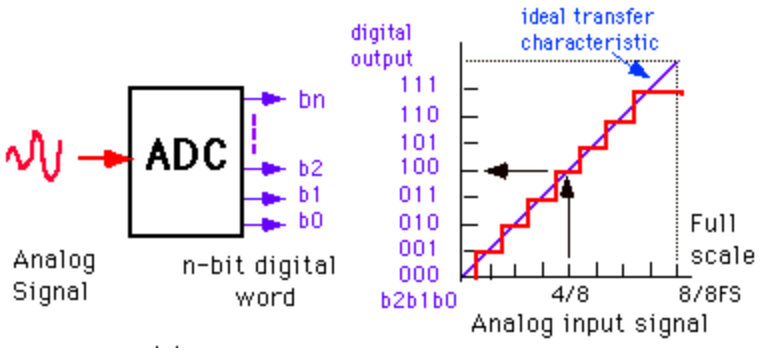


Figure 5.1-Analog-to-digital converter concept [15]

The digital-to-analog converter is a system that converts the digital signal to an analog signal, by taking binary numbers and outputting an analog voltage or current signal. These signals are usually filtered such that they have an smooth the signal. This concept can be seen in Figure 5.2 below.

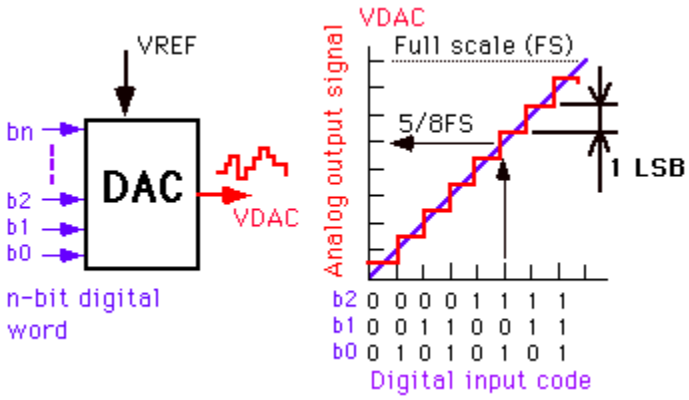


Figure 5.2-Digital-to-analog converter [15]

These signals conversions are needed in order for computers to use input signals and control output signals. This concept can be seen in Figure 5.3 below.

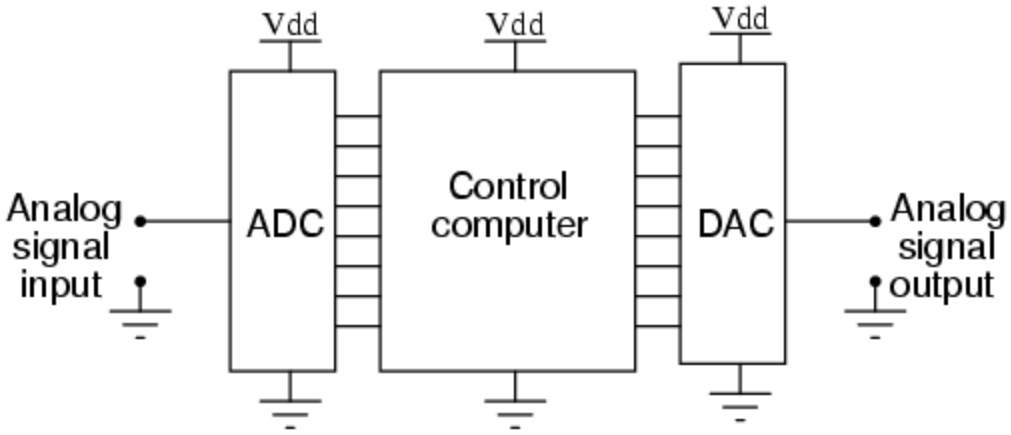


Figure 5.3-ADC and DAC architecture [16]

The analog to digital converter, in the quadcopter simulation, needs to be used to convert continuous time sensor signals to digital signals so the digital controller can use them. The digital to analog converter takes the output of the controllers and creates analog inputs to the plant. In the real world these come in the form of voltages to the motors.

An important Simulink block when using digital signals is the zero-order-hold block. In essence this is a sample and hold block. It takes an input signal, samples it, and outputs that value until the next sample time.

5.2 Discrete PID

As stated earlier, continuous time PID controllers take an error signal, applies a general gain, and applies a gain from an integral of the error and the derivative of the error. The digital domain PID controller isn't much different. The main idea behind the discrete PID controller is that it only looks at the signal at the current time, and the previous time signal. The discrete PID equation can be seen below [17, 18], where k is the index, $u[k]$ is the controller input at index k, T_s is the sample rate, K_p, K_i and K_d are controller gains, and e is the error.

$$u[k] = u[k-1] + \frac{K_d}{T_s}(e[k] - 2e[k-1] + e[k-2]) + K_p(e[k] - e[k-1]) + K_i T_s e[k] \tag{5.1}$$

In the Z domain, or discrete domain, this can be represented as the following.

$$U(z) = \left[K_p + \frac{K_i}{1 - z^{-1}} + K_d(1 - z^{-1})\right] E(z) \tag{5.2}$$

5.3 Discrete Controller Simulink Implementation

Due to the paper being more for the control and modeling of a quadcopter, the signals in the model will be simply digitized using the zero-order hold block, no binary numbers will be used. The MATLAB scrip and Simulink model for this section can be found in the folder titled 'Digital_Controller_Plant'. To build the discrete control system in Simulink, first take the full controller, plant and sensor model that was earlier designed. Place zero order hold blocks at the outputs of the IMU and at the desired states. set the zero order hold to the desired sample rate, 0.001 for 1000Hz. This step creates the digital signal for the controller. This step can be seen in Figure 5.4 below.

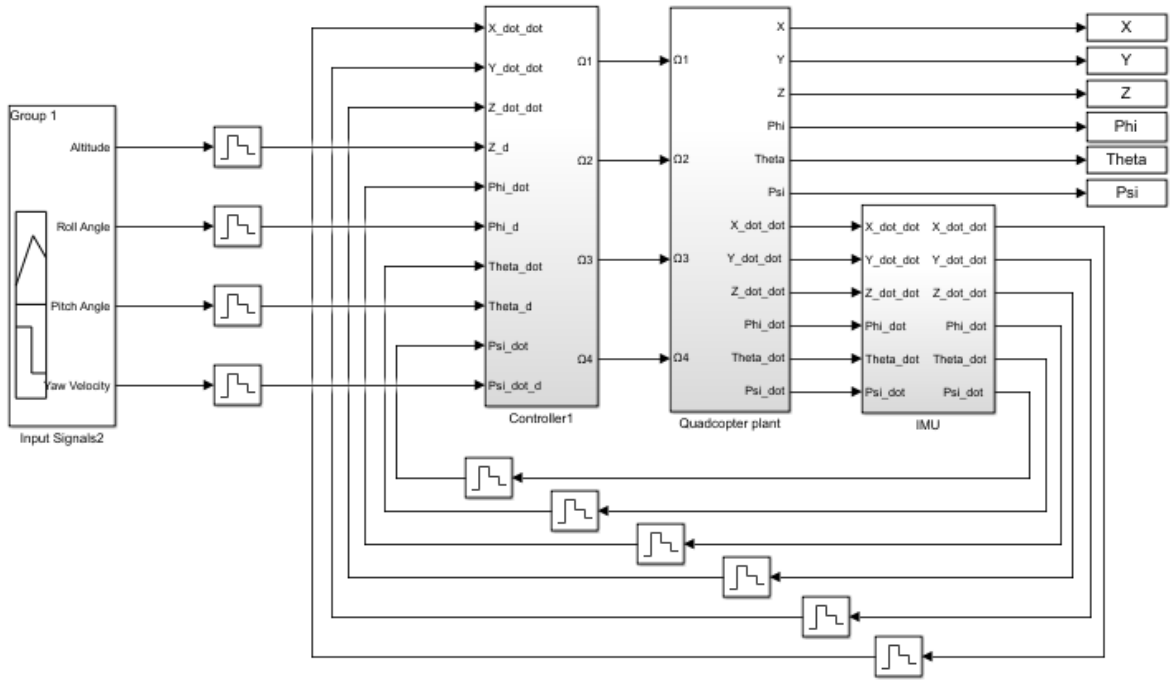


Figure 5.4-Zero order hold implementation

Next, implement the discrete PID controller previously defined. This can be done using summation, product and Z delay blocks. One Z delay block can represent the $e[k-1]$ portion of the controller. An example of this for the yaw control can be seen in Figure 5.5 below.

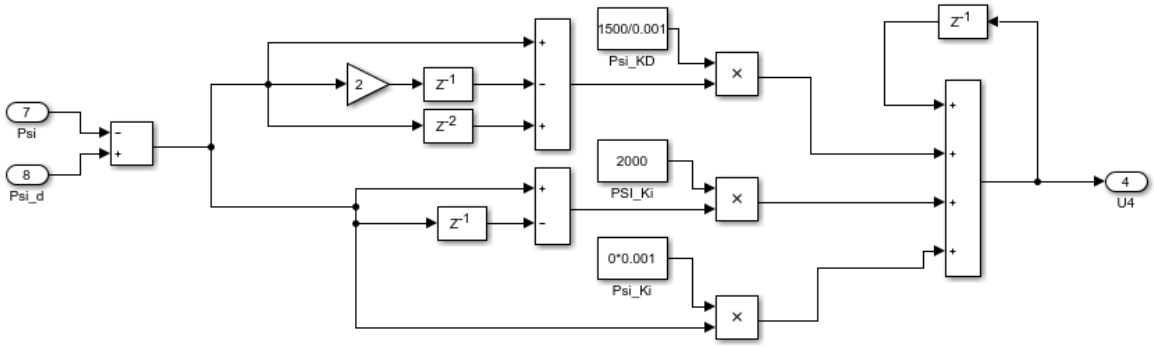


Figure 5.5-Discrete yaw PID controller

Chapter 6 Conclusion

Creating a quadcopter plant model and controller is a complex task, however, if broken down correctly it can be relatively simple. There are multiple ways of creating the plat model for a quadcopter, ranging from simple to extremely articulate. The basic plant model contained simple thrust and torque coefficients along with the motions of equation. The basic plant model also had the choice of being in the 'x' or '+' coordinate system which slightly changed some thrust equations within the model. From there, a more accurate plant model was created, the motor dynamics were added along with battery dynamics. These add limited state values within the plant model which was more realistic to real-world use. Adding air resistance also increased the accuracy of the quadcopter plant model. These allowed the plant model to slowdown and stop, if the plant model was traveling in a direction and not accelerating fast enough.

The controller for a quadcopter is a tough idea to envision, however, knowing how controllers work make them a tad easier to envision. Control systems for quadcopters can be many types, PID, LQR, Sliding Mode Control, etc. PI-D can also be used to dampen states enough such that they will not become unstable. The tuning method for PID controllers can also be relatively simple. The two methods presented were manual and optimal PID control. Both can achieve a stable, controllable system. However, the optimal PID control tuning method was found to be more automated and less intuitive. After the PID controllers were tuned it was found that the system reacted better to ramp inputs opposed to step responses (i.e. there was less error between the reference signal and state response).

The sensors of the quadcopter were also able to be modeled. The accelerometer and gyroscope was difficult to model due to only having the plant model outputs available. However, they were modeled by using the inverse transformation matrix such that the sensor readings would act as though they were on the body frame of the quadcopter opposed to the local NED frame. Once this was accomplished the sensor model was able to demonstrate the effects of noise, bias and the initialization process, which removed the sensor bias.

A very important aspect when creating a plant model, controller, sensor model, etc., is how to implement them into a simulation tool. MALAB/Simulink was a very good software tool to implement these. The block models within Simulink make the equation-to-model conversion very intuitive. Along with being intuitive, Simulink allows the user to trouble shoot models. If a response within the system is not as expected, the user can step back through the model to find the error in the model, or where the problem is coming from. MATLAB/Simulink is also useful due to the ease of converting a system from a block model to C or C++ code. This allows the user to implement a control system on a quadcopter easily, as well as speedy tuning control systems, as the conversion process is quick.

The process of creating a plant model, developing a controller and implementing them into MATLAB/Simulink was daunting. However, broken down enough the task was simple. Knowing how quadcopters fly and operate is a very interesting field to explore. Understanding the basic quadcopters dynamics and modeling process can help create even more complex models!

Chapter 7 Future Work

Although the modeling in this paper could be viewed as complex, there are more characteristics that can be added to the model such as hub forces, and ground effects. The addition of these in a model will increase the accuracy of the model to a real-world quadcopter. Along with increased accuracy in the model, a range of control techniques may improve the control of the quadcopter. The control technique used, PI-D, may not offer the control desired by the user. In which case control techniques such as sliding mode control, LQR, etc. can be explored.

An aspect about the quadcopter modeling process that was not mentioned was the use of complimentary filters to achieve the true euler angles. The current model assumes the integrals within the controller achieve a perfect angle reading. In reality this complimentary filter is used such that the accelerometer is used in conjunction with the gyroscopes to provide more accurate angle readings. This would be a beneficial portion to add to a plant model to achieve a more real-world model.

Along with the additions in complexity to the model and varying control methods, it would be beneficial to run the real-world controller on a quadcopter. Logging the data from the quadcopter could prove the ability of the controller to stabilize and control the quadcopter. It would also allow for the comparison between the real-world quadcopter and the model created. This would also allow the user to increase the accuracy of the plant model that was created. In many cases parameters used in the plant model are not perfect, comparing the output of a real-world quadcopter to the plant model could elicit a slight change in some of the parameters.

APPENDIX

A.1 Moment of Inertia Estimation

An important aspect of modeling any quadcopter is approximating its moment of inertia. This aspect is integral in forming the model's equations of motion. It is also important to understand the difference between Moments and Moment of inertia. A moment can be thought of as a force, or a torque on an object. The moment of inertia can be thought of as an objects resistance to rotation which is a constant. The process of determining this relies on breaking down the quadcopter into individual components, measuring the components moment's of inertia, and adding them together [22].

To use the method of approximating the moment of inertia shown below, the quadcopter must be as close to symmetric about the X, Y and Z axis as possible. Some relevant equations for the method shown include the parallel axis theory or Steiner's Theorem[23]. This theory is used to determine the moment of inertia of an object if it is rotated about a parallel axis a distance from the original axis. This theory is shown in Equation (A.1) below, where I_{cm} is the objects center of mass moment of inertia, m is the object's mass and d_{a2a} is the distance from the two-parallel axis'. An example of this can be seen in Figure A.1 below. Here the moment inertia about the Z axis is I_{cm} and the moment of inertia about the Z' axis is I.

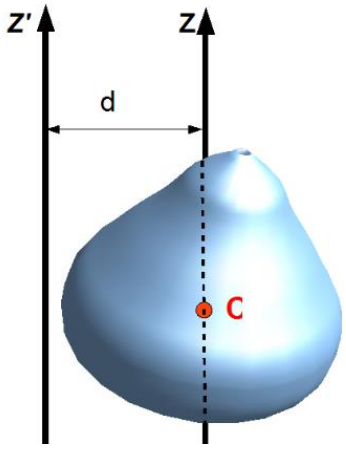


Figure A.1-Parellel axis theorem example

$$I = I_{cm} + md_{a2a}^2 \tag{A.1}$$

The moment of inertia of a solid cylinder can be represented in the following way, where r is the radius of the cylinder, h is the height of the cylinder and m is the mass.

$$I_{cylinder,x} = \frac{1}{12}m(3r^2 + h^2) \tag{A.2}$$

$$I_{cylinder,y} = \frac{1}{12}m(3r^2 + h^2) \tag{A.3}$$

$$I_{cylinder,z} = \frac{1}{2}mr^2 \tag{A.4}$$

The moment of inertia of a solid cuboid can be represented the following way, where h, d and w represent height, diameter and width respectively.

$$I_{cuboid,x} = \frac{1}{12}m(w^2 + h^2) \tag{A.5}$$

$$I_{cuboid,y} = \frac{1}{12}m(h^2 + d^2) \tag{A.6}$$

$$I_{cuboid,z} = \frac{1}{12}m(w^2 + d^2) \tag{A.7}$$

The moment of inertia of a thin rectangular plate can be represented the following way

$$I_{Thin\ Plate,x} = \frac{1}{12}m(w^2) \tag{A.8}$$

$$I_{hin\ Plate,y} = \frac{1}{12}m(h^2) \tag{A.9}$$

$$I_{hin\ Plate,z} = \frac{1}{12}m(w^2 + h^2) \tag{A.10}$$

A.1.1 Motor Moment of Inertia

The motors of a quadcopter can be represented as identical cylinders. Using Equations (A.2)-(A.4) the moment of inertia is known for each motor in each axis. The tricky part is finding the moment of inertia from the motors relative to the center of mass of the quadcopter. To do this, the parallel axis theory seen in Equation (A.1) will be used.

The best way to explain this is through example. Imagine a quadcopter with motors of mass m, height of h, radius of r and distance from the center of mass to center of motor as d_{a2a}. First, find the moment of inertia in the x axis ($I_{motor,x}$). Starting with the motors that reside on the x axis, Equation (A.11) can be used. The factor of 2 is from two motors being on the x axis.

$$I_{motor,x} = 2(\frac{1}{12}m(3r^2 + h^2)) \tag{A.11}$$

Next, the motors that reside on the perpendicular, y axis need to be accounted for. Because the X axis of these motors are d_{a2a} from the original x axis, the parallel axis theory will be used. Using this, it will extend Equation (A.11) into the following. The leading factor of 2 in the addition to the equation is because there are 2 motors that are a distance d_{a2a} from the original x axis.

$$I_{motor,x} = 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right) \tag{A.12}$$

Figure A.2 can help visualize the motor moment of inertia in the X axis.

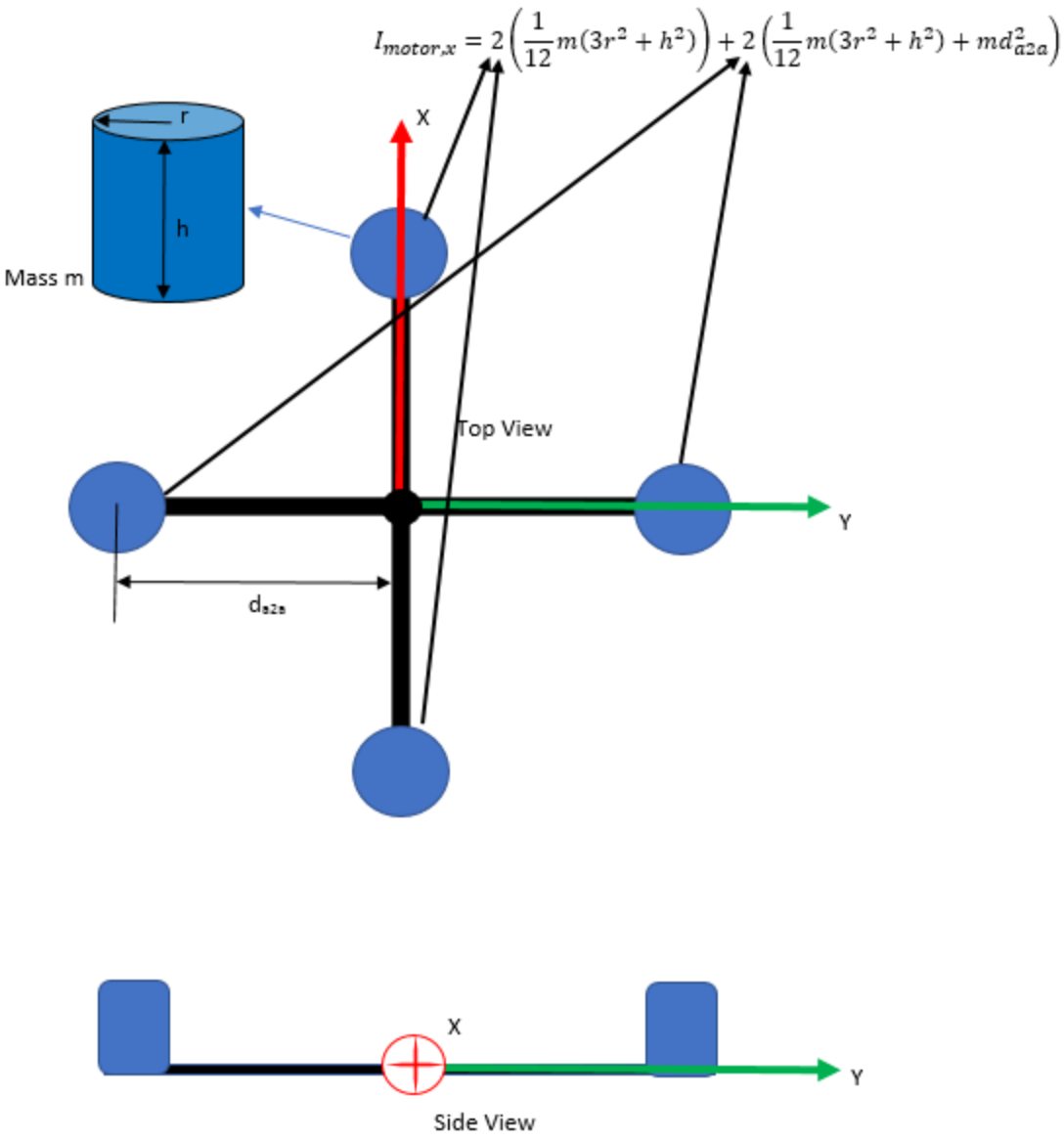


Figure A.2-Motor moment of inertia in the X axis

Next, the moment of inertia in the y axis ($I_{motor,y}$) needs to be found. Luckily because of symmetry the moment of inertia in the x axis is equal to the moment of inertia in the y axis.

$$I_{motor,x} = I_{motor,y} \tag{A.13}$$

Finally, the moment of inertia due to the motors in the Z axis ($I_{motor,z}$) should be found. Here, use Equation (A.1) and Equation (A.4). The parallel theory needs to be used because the z axis of each individual motor

is d_{a2a} away from the original z axis. Using these two equations following is formed. The leading factor of 4 is because there are 4 motors.

$$I_{motor,z} = 4(\frac{1}{2}mr^2 + md_{a2a}^2) \tag{A.14}$$

Now that the moment of inertia due to the motors in the x, y and z axis they can put them into a condensed vector seen below.

$$I_{Motor}^b = [I_{motor,x} \quad I_{motor,y} \quad I_{motor,z}] \tag{A.15}$$

$$I_{Motor}^b = \left[2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right) \quad 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right) \quad 4(\frac{1}{2}mr^2 + md_{a2a}^2)\right] \tag{A.16}$$

A.1.2 ESC Moment of Inertia

The moment of inertia due to the ESCs can be looked at in two ways. The ESCs can be thought of in two ways, as a thin rectangle (neglecting the thickness) or they can be treated as a solid cuboid (accounting for thickness). This section, will go through examples of both cases.

ESC as Thin Rectangle

Imagine 4 ESCs equipped on a quadcopter. Each ESC has a height of h a width of w, and the center point being a distance d_{a2a} away from the center of mass. Like the motor case Equations (A.2), along with (A.5)-(A.7) will be used to form this moment of inertia.

Figure A.3-Thin Rectangle ESC Dimension [19]

Starting with the moment of inertia in the X axis, ($I_{ESC,x}$), the two ESCs that lie on the X axis with Equation (A.17) can be used. The off X axis ESCs can be represented with a combination of Equation (A.1) and Equation (A.10). This can be seen in the Equation below.

$$I_{ESC,x} = 2\left(\frac{1}{12}m(w^2)\right) + 2\left(\frac{1}{12}m(h^2) + md_{a2a}^2\right) \tag{A.17}$$

Next, the moment of inertia in the y axis ($I_{ESC,y}$) needs to be found. Because of symmetry the moment of inertia in the x axis is equal to the moment of inertia in the y axis.

$$I_{ESC,x} = I_{ESC,y} \tag{A.18}$$

Lastly, the moment of inertia about the Z axis($I_{ESC,x}$) needs to be found. Using the same ideology of the z axis of the motors Equations (A.1) and (A.10) will be used to form the following moment of inertia.

$$I_{ESC,z} = 4(\frac{1}{12}m(w^2 + h^2) + md_{a2a}^2) \tag{A.19}$$

Now that the moment of inertia in each axis from the thin rectangle modeled ESCs has been found they can put into a condensed form.

$$I_{ESC}^b = [I_{ESC,x} \quad I_{ESC,y} \quad I_{ESC,z}] \tag{A.20}$$

$$I^b_{Motor} = \left[2\left(\frac{1}{12}m(w^2)\right) + 2\left(\frac{1}{12}m(h^2) + md^2_{a2a}\right) \quad 2\left(\frac{1}{12}m(w^2)\right) + 2\left(\frac{1}{12}m(h^2) + md^2_{a2a}\right) \quad 4(\frac{1}{12}m(w^2 + h^2) + md^2_{a2a})\right] \tag{A.21}$$

ESC as Solid Cuboid

Imagine 4 ESCs equipped on a quadcopter. Each ESC has a height of h a width of w, a depth of d and the center point being a distance d_{a2a} away from the center of mass. Like the motor case Equations (A.1), along with (A.5)-(A.7) will be used to form this moment of inertia.

Figure A.4-Solid cuboid ESC dimensions [19]

Starting with the moment of inertia in the X axis, ($I_{ESC,x}$), the two ESCs that lie on the X axis can be represented with Equation (A.5). The off X axis ESCs with a combination of Equation (A.1) and Equation (A.5) can be used. This can be seen in the Equation below.

$$I_{ESC,x} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md^2_{a2a}\right) \tag{A.22}$$

Next, the moment of inertia in the y axis ($I_{ESC,y}$) needs to be found. Because of symmetry the moment of inertia in the x axis is equal to the moment of inertia in the y axis.

$$I_{ESC,x} = I_{ESC,y} \tag{A.23}$$

Lastly, the moment of inertia about the Z axis ($I_{ESC,z}$) needs to be found. Using the same ideology of the z axis of the motors Equations (A.1) and (A.7) will be used to form the following moment of inertia.

$$I_{ESC,z} = 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2) \tag{A.24}$$

Now that the moment of inertia in each axis from the thin rectangle modeled ESCs they can put into a condensed form.

$$I_{ESC}^b = [I_{ESC,x} \quad I_{ESC,y} \quad I_{ESC,z}] \tag{A.25}$$

$$I_{Motor}^b = \left[2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right) \quad 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right) \quad 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2)\right] \tag{A.26}$$

A.1.3 Arms Moment of Inertia

Like the ESCs the arms of a quadcopter can be thought of in multiple ways. In this section, the moment of inertia representing the arms as solid cuboids, as well as cylinders will be found. A similar methods as prior moment of inertia calculations will be used

Quadcopter Arms as Solid Cuboids

Imagine 4 cubic arms on a quadcopter. Each arm has a height of h a width of w, a depth of d and the center point being a distance d_{a2a} away from the center of mass. Like the motor case Equations (A.1), along with (A.5) -(A.7) will be used to form this moment of inertia.

Starting with the moment of inertia in the x axis, ($I_{Arm,x}$), the two arms that lie on the X axis can be represented with Equation (A.5). The off X axis arms can be represented with a combination of Equation (A.1) and Equation (A.5). This can be seen in the Equation below.

$$I_{Arm,x} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right) \tag{A.27}$$

Next, the moment of inertia in the y axis ($I_{Arm,y}$) needs to be found. Because of symmetry the moment of inertia in the X axis is equal to the moment of inertia in the Y axis.

$$I_{Arm,x} = I_{Arm,y} \tag{A.28}$$

Lastly, the moment of inertia about the Z axis ($I_{Arm,z}$) needs to be found. Using the same ideology of the Z axis of the motors Equations (A.1) and (A.7) will be used to form the following moment of inertia.

$$I_{Arm,z} = 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2) \tag{A.29}$$

Now that the moment of inertia in each axis has been formed, the solid cuboid modeled arms can be put into a condensed form.

$$I_{Arm}^b = [I_{Arm,x} \quad I_{Arm,y} \quad I_{Arm,z}] \tag{A.30}$$

$$I_{Motor}^b = \left[2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right) \quad 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right) \quad 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2)\right] \tag{A.31}$$

Quadcopter Arms as Cylinders

Imagine 4 cylindrical arms on a quadcopter. Each arm has a radius of r a length of L, and the length center point being a distance d_{a2a} away from the center of mass. Like the motor case Equations (A.1), along with (A.2) -(A.4) will be used to form this moment of inertia.

Starting with the moment of inertia in the X axis, ($I_{Arm,x}$), the two arms that lie on the X axis can be represented with Equation (A.2). The off-X axis arms with a combination of Equation (A.1) and Equation (A.2) can be used. This can be seen in Equation (A.32) below.

$$I_{Arm,x} = 2\left(\frac{1}{2}mr^2\right) + 2\left(\frac{1}{12}m(3r^2 + L^2) + md_{a2a}^2\right) \tag{A.32}$$

Next, the moment of inertia in the y axis ($I_{Arm,y}$) needs to be found. Because of symmetry the moment of inertia in the x axis is equal to the moment of inertia in the y axis.

$$I_{Arm,x} = I_{Arm,y} \tag{A.33}$$

Lastly, the moment of inertia about the Z axis ($I_{Arm,z}$) needs to be found. Using the same ideology of the z axis of the motors Equations (A.1) and (A.4) will be used to form the following moment of inertia.

$$I_{Arm,z} = 4(\frac{1}{12}m(3r^2 + L^2) + md_{a2a}^2) \tag{A.34}$$

Now that the moment of inertia in each axis has been formed, the solid cuboid modeled arms can be put into a condensed form.

$$I_{Arm}^b = [I_{Arm,x} \quad I_{Arm,y} \quad I_{Arm,z}] \tag{A.35}$$

$$I_{Arm}^b = \left[2\left(\frac{1}{2}mr^2\right) + 2\left(\frac{1}{12}m(3r^2 + L^2) + md_{a2a}^2\right) \quad 2\left(\frac{1}{2}mr^2\right) + 2\left(\frac{1}{12}m(3r^2 + L^2) + md_{a2a}^2\right) \quad 4(\frac{1}{12}m(3r^2 + L^2) + md_{a2a}^2)\right] \tag{A.36}$$

A.1.4 Central Component Moment of Inertia

Every quadcopter has a central component that usually houses a flight controller and battery. Depending how in-depth the moment of inertia approximation needs to be, these components can be split up and a similar method as previous sections can be used to find the moment of inertia. Or this central component can be represented as either a solid cuboid, or a cylinder. In either case, the only difference between these calculations and prior ones is that the assumption is that the center of this component is the center of mass that we've been using the parallel axis theorem for. For this discussion, the battery and flight controller will be split.

Imagine a battery is lying on the center of mass, it has a height of h a width of w, and a depth of d. Using Equations (A.5)-(A.7) the battery can be represented the following way.

$$I_{Battery,x} = \frac{1}{12}m(w^2 + h^2) \tag{A.37}$$

$$I_{Battery,y} = \frac{1}{12}m(h^2 + d^2) \tag{A.38}$$

$$I_{Battery,z} = \frac{1}{12} m(w^2 + d^2) \tag{A.39}$$

Imagine a flight controller with the same dimensioning notation, the moment of inertia would follow the same equations.

$$I_{FC,x} = \frac{1}{12} m(w^2 + h^2) \tag{A.40}$$

$$I_{FC,y} = \frac{1}{12} m(h^2 + d^2) \tag{A.41}$$

$$I_{FC,z} = \frac{1}{12} m(w^2 + d^2) \tag{A.42}$$

Now that the moment of inertia in each axis for the central component has been found, they can put into a condensed form.

$$I^b_{Battery} = [I_{Battery,x} \quad I_{Battery,y} \quad I_{Battery,z}] \tag{A.43}$$

$$I^b_{Battery} = \left[\frac{1}{12} m\left(w^2 + h^2\right) \;\; \frac{1}{12} m\left(h^2 + d^2\right) \;\; \frac{1}{12} m\left(w^2 + d^2\right)\right] \tag{A.44}$$

$$I^b_{FC} = [I_{FC,x} \quad I_{FC,y} \quad I_{FC,z}] \tag{A.45}$$

$$I^b_{FC} = \left[\frac{1}{12} m\left(w^2 + h^2\right) \;\; \frac{1}{12} m\left(h^2 + d^2\right) \;\; \frac{1}{12} m\left(w^2 + d^2\right)\right] \tag{A.46}$$

A.1.5 Rotor moment of inertia

Using the propellers mass and radius the moment of inertia for the rotor can be estimated with the following equation

$$J_r = \frac{1}{2} m R^2 \tag{A.47}$$

A.2 Moment of Inertia Summary

Table A.1-Components moment of inertia

Motor MoI	$I_{motor,x} = 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right)$ $I_{motor,y} = 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right)$ $I_{motor,z} = 4(\frac{1}{2}mr^2 + md_{a2a}^2)$
ESC MoI	$I_{ESC,x} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{ESC,y} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{ESC,z} = 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2)$
Arms MoI	$I_{Arm,x} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{Arm,y} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{Arm,z} = 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2)$
Battery MoI	$I_{Battery,x} = \frac{1}{12}m(w^2 + h^2)$ $I_{Battery,y} = \frac{1}{12}m(h^2 + d^2)$ $I_{Battery,z} = \frac{1}{12}m(w^2 + d^2)$
Flight Controller MoI	$I_{FC,x} = \frac{1}{12}m(w^2 + h^2)$ $I_{FC,y} = \frac{1}{12}m(h^2 + d^2)$ $I_{FC,z} = \frac{1}{12}m(w^2 + d^2)$

A.3 Propeller Analysis

Propellers

The propellers of a quadcopter are integral components that play a large role in the quadcopter dynamics. The propellers produce downward thrust to overcome the force of gravity allowing the quadcopter to fly. There are multiple variables associated with the propellers, these include: propeller pitch, diameter, chord and material. Using the pitch and diameter, some important parameters of the propellers can be found such as power and thrust.

Pitch and Diameter

The diameter of a propeller is the distance from tip to tip usually given in inches which can be seen in Figure A.3. The pitch of a propeller is how much the blade is angled and is usually a function of radius. It is determined by measuring the distance the propeller would advance if turned one revolution through solid medium. Figure A.3 shows the effects of a low and high-pitched propeller. The low-pitched propeller moves less distance than the high-pitched propeller. One may be wondering why high-pitched propellers are not always used. It will be shown in later sections that there is a trade off with power associated with the pitch, diameter and thrust. For example, if both propellers seen in Figure A.5 were rotating at the same rotation per minute (RPM), the higher pitched propeller would require more power than the low-pitched propeller.

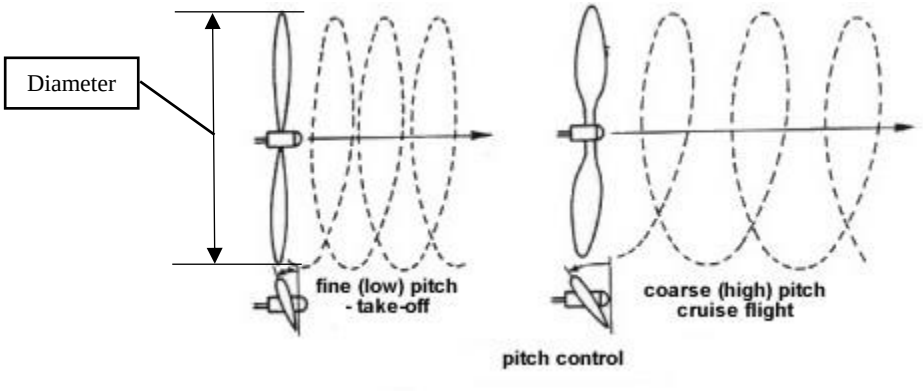


Figure A.5-Low and high pitch propeller comparison [20]

The general notation of propeller is Diameter(in) x Pitch(in), for example a propeller with a 5-inch diameter and a 3-inch pitch would be denoted at "5x3".

Power and Thrust

Power and thrust can be derived from the pitch, diameter and a few other parameters. Thrust is measured in Newtons (N), it is the upward force generated by the propellers. It is basically the "push" generated by props so the quadcopter can fly. The basic equation for thrust can be seen below in Equations (A.48)-(A.50)[21, 22]. These are just a few of the forms of thrust equations.

$$Thrust = \frac{1}{2}\rho A V_e^2 \tag{A.48}$$

$$Thrust = C_T \rho A r^2 \Omega^2 \tag{A.49}$$

$$Thrust = c_T \varpi \Omega^2 \tag{A.50}$$

The parameters of these equations are as follows: ρ is the air density which is a function of air pressure, altitude, and temperature, A is the cross-sectional area of the propeller, V_e is the velocity of the exiting air from the propeller. C_T is a propeller characteristic coefficient, and Ω is angular velocity. These parameters can be calculated using the equations below.

$$\rho = \frac{p}{R_{specific} T_k} \tag{A.51}$$

Where p is the absolute pressure (Pa), T is the absolute temperature (K), and $R_{specific}$ is the specific gas constant for dry air $(J/(kg * K))$.

$$A = \pi r^2 \tag{A.52}$$

Where r is the propellers radius.

$$V_e = 0.0254 * Pitch * (\frac{RPM}{60}) \tag{A.53}$$

The power required to generate thrust of a propeller can be described using the Momentum theory. This theory describes the mathematical model of a propeller and relates it to power. This equation can be seen

below. Equation (A.54) shows power and thrust are related in a nonlinear fashion, $P^2 \propto T^3$. This relationship can be plotted & seen below in Figure A.6.

$$P = \sqrt{\frac{T^3}{2\rho A}} \tag{A.54}$$

Or to relate power to air velocity,

$$P = \frac{1}{2}\rho A v^3 \tag{A.55}$$

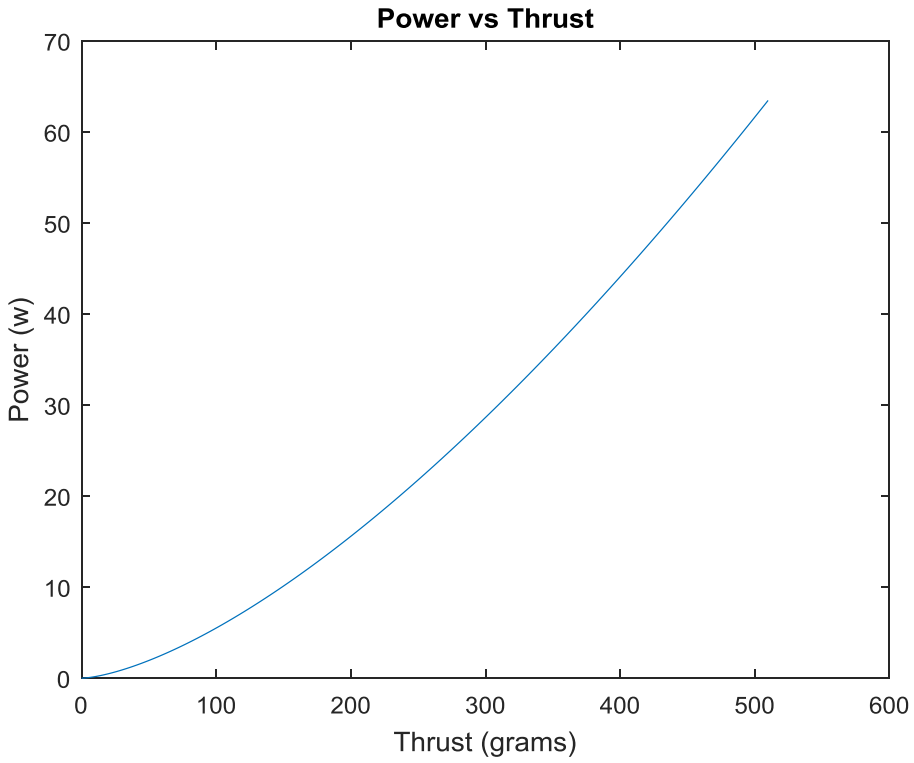


Figure A.6-Power vs. thrust curve generated using the momentum theory with a 5 inch propeller

Equations (A.48) - (A.55) are extremely basic equations. The mathematics involved with determining thrust are much more advanced and are specific to the propeller in question. A better way of finding thrust is to use online databases where the thrust has been experimentally determined.

Another useful aspect to know about propellers is how pitch, diameter and RPM change the thrust. Using a thrust equation in which the coefficients are known, MATLAB can be used to create a plot relating these

parameters. Below shows a plot using a general thrust equation that can somewhat accurately represent most propellers regardless of the shape.

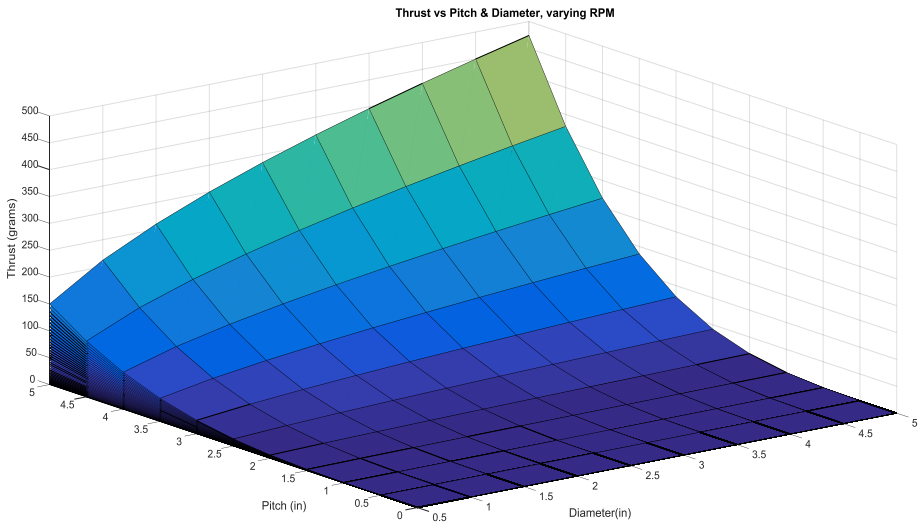


Figure A.7-Thrust vs pitch vs. diameter over a 20k RPM range for a 5-inch propeller

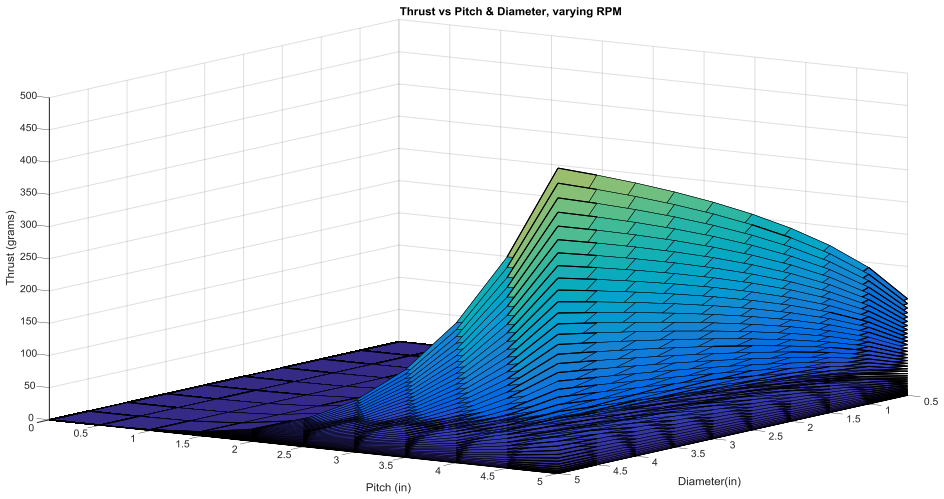


Figure A.8-Thrust vs. Pitch vs. diameter over a 20k RPM range for a 5-inch Propeller

The plots in Figure A.7 and Figure A.8 are relative to the thrust equation that is used, regardless of the equation used the general shape of the plots will be similar. As pitch and diameter increase so does the thrust. Figure A.7 is a good depiction of how the thrust increases more exponentially depending on pitch

and diameter at higher RPM than at lower RPM (I.e. – at low RPM there is less change in thrust depending on pitch and thrust, while at high RPM the pitch and diameter are weighted heavier).

Plant Equations

State/Input	Dependencies	Equation
θ	$\dot{\theta}$	$\theta = \int \dot{\theta}$
$\dot{\theta}$	$\ddot{\theta}$	$\dot{\theta} = \int \ddot{\theta}$
$\ddot{\theta}$	$\dot{\phi}, \dot{\psi}, \Omega_r, U3$	$\ddot{\theta} = \frac{\dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\Omega_r + l(U3)}{I_{yy}}$
ϕ	$\dot{\phi}$	$\phi = \int \dot{\phi}$
$\dot{\phi}$	$\ddot{\phi}$	$\dot{\phi} = \int \ddot{\phi}$
$\ddot{\phi}$	$\dot{\theta}, \dot{\psi}, \Omega_r, U2$	$\ddot{\phi} = \frac{\dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) + J_r\dot{\theta}\Omega_r + l(U2)}{I_{xx}}$
ψ	$\dot{\psi}$	$\psi = \int \dot{\psi}$
$\dot{\psi}$	$\ddot{\psi}$	$\dot{\psi} = \int \ddot{\psi}$
$\ddot{\psi}$	$\dot{\theta}, \dot{\phi}, U4$	$\ddot{\psi} = \frac{\dot{\theta}\dot{\phi}(I_{xx} - I_{yy}) + (U4)}{I_{zz}}$
X	$\dot{X}$	$x = \int \dot{x}$
$\dot{X}$	$\ddot{X}$	$\dot{x} = \int \ddot{x}$
$\ddot{X}$	$\psi, \phi, \theta, U1$	$\ddot{X} = \frac{(\sin\psi \sin\phi - \cos\psi \sin\theta \cos\phi)U1 - A_x\dot{X}}{m}$
Y	$\dot{Y}$	$Y = \int \dot{Y}$
$\dot{Y}$	$\ddot{Y}$	$\dot{Y} = \int \ddot{Y}$
$\ddot{Y}$	$\psi, \phi, \theta, U1$	$\ddot{Y} = \frac{(\cos\psi \sin\phi + \sin\psi \sin\theta \cos\phi)U1 - A_y\dot{Y}}{m}$
Z	$\dot{Z}$	$Z = \int \dot{Z}$

$\dot{Z}$	$\ddot{Z}$	$\dot{Z} = \int \ddot{Z}$
$\ddot{Z}$	$\psi, \phi, U1$	$\ddot{Z} = \frac{mg - (\cos\theta \cos\phi)U1 - A_z\dot{Z}}{m}$
Ω_r	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$\Omega_r = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$
$\mathrm{U1}_+$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U1_+ = b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2)$
$\mathrm{U2}_+$	Ω_2, Ω_4	$U2_+ = b(-{\Omega_2}^2 + {\Omega_4}^2)$
$\mathrm{U3}_+$	Ω_1, Ω_3	$U3_+ = b({\Omega_1}^2 - {\Omega_3}^2)$
$\mathrm{U4}_+$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U4_+ = d({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2)$
$\mathrm{U1}_x$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U1_x = b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2)$
$\mathrm{U2}_x$	Ω_2, Ω_4	$U2_x = b\sin\left(\frac{pi}{4}\right)({\Omega_1}^2 - {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2)$
$\mathrm{U3}_x$	Ω_1, Ω_3	$U3_x = b\sin\left(\frac{pi}{4}\right)({\Omega_1}^2 + {\Omega_2}^2 - {\Omega_3}^2 - {\Omega_4}^2)$
$\mathrm{U4}_x$	$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	$U4_x = d({\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 - {\Omega_4}^2)$

Bibliography

[1] Quadcopter Parts List | What You Need to Build a DIY Quadcopter. (n.d.). Retrieved November 06, 2017, from http://quadcoptergarage.com/quadcopter-parts-list-what-you-need-to-build-a-diy-quadcopter/

[2] Brushed vs Brushless Motors. (n.d.). Retrieved November 06, 2017, from http://www.thinkrc.com/faq/brushless-motors.php

[3] Electronic speed control. (2017, November 03). Retrieved November 07, 2017, from https://en.wikipedia.org/wiki/Electronic_speed_control

[4] Montgomery, C. (2014, June 03). An Introduction to Quadcopter Flight Controllers. Retrieved November 07, 2017, from http://www.tomshardware.com/reviews/multi-rotor-quadcopter-fpv,3828-2.html

[5] How To Choose Radio Transmitter & Receiver for Racing Drones and Quadcopter. (2017, October 04). Retrieved November 07, 2017, from https://oscarliang.com/choose-rc-transmitter-quadcopter/

[6] Control theory. (n.d.). Retrieved November 06, 2017, from http://www.wikiwand.com/en/Control_theory

[7] Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying*(Unpublished master's thesis). Thèse no 3727 sc. EPF Lausanne.

[8] Bresciano, T.: *Modelling, Identification and Control of a Quadrotor Helicopter*. M.Sc. Thesis. Lund University – Department of Automatic Control, Lund, 2008,

[9] Carrillo, L. R., López, A. E., Lozano, R., & Pégard, C. (2013). Modeling the Quad-Rotor Mini-Rotorcraft. *Advances in Industrial Control Quad Rotorcraft Control,* 23-34. doi:10.1007/978-1-4471-4399-4_2

[10] Horn, G. (2009, March 17). The Aerospace Euler Angles. Retrieved November 06, 2017, from https://www.youtube.com/watch?v=UpSMNYTVqQI

[11] *DC Motor Position: Simulink Controller Design* [Scholarly project]. (n.d.). In *Control Tutorials For MATLAB & Simulink*. Retrieved October 31, 2017, from http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition§ion=SimulinkControl

[12] Turnigy 700mAh 3S 60C Lipo Pack (XT30). (n.d.). Retrieved November 06, 2017, from https://hobbyking.com/en_us/turnigy-700mah-3s-60c-xt30.html?___store=en_u

[13] How to choose LiPo Battery Beginner Guide for Mini Quad, Drones and Quadcopters. (2017, August 14). Retrieved November 06, 2017, from https://oscarliang.com/lipo-battery-guide/

[14] Urquizo, A. (n.d.). PID controller overview. Retrieved November 06, 2017, from https://commons.wikimedia.org/wiki/File:PID_en.svg

[15] Spiegel, J. (2010, February 8). *Analog-to-Digital(ADC) and Digital-to-analog(DAC) Converter*[Scholarly project]. In *University of Pennsylvania Department of Electrical and Systems Engineering*. Retrieved October 31, 2017, from http://www.seas.upenn.edu/~ese206/labs/adc206/adc206.html

[16] Introduction to Digital-Analog Conversion. (n.d.). Retrieved October 31, 2017, from https://www.allaboutcircuits.com/textbook/digital/chpt-13/digital-analog-conversion/

[17] Kravit, G. (2014, December 13). *FPGA Implementation of a Digital Controller for a Small VTOL UAV* [Scholarly project]. In *6.111 Introductory Digital Systems Laboratory*. Retrieved October 31, 2017, from http://web.mit.edu/6.111/www/f2014/projects/gkravit_Project_Final_Report.pdf

[18] Toochinda, V., Dr. (2011, June). *Digital PID Controllers* [Scholarly project]. In *Control Systems Lab*. Retrieved October 20, 2017, from http://controlsystemslab.com/

[19] Turnigy Multistar 20A V2 ESC With BLHeli and 4A LBEC 2-6S. (n.d.). Retrieved November 06, 2017, from https://hobbyking.com/en_us/turnigy-multistar-20a-v2-esc-with-blheli-and-4a-lbec-2-6s.html

[20] Fixed Wing Flight Training. (n.d.). Retrieved November 06, 2017, from http://www.pilotfriend.com/training/flight_training/fxd_wing/props.htm

[21] Propeller Thrust. (2015, May 05). Retrieved November 07, 2017, from https://www.grc.nasa.gov/www/k-12/airplane/propth.html

[22] Hartman, D. K. (n.d.). *Quad-Sim* [Scholarly project]. In *Quadcopter Dynamic Modeling and Simulation (Quad-Sim)*. Retrieved November 6, 2017, from https://github.com/dch33/Quad-Sim

[23] Parallel axis theorem. (2017, November 05). Retrieved November 13, 2017, from https://en.wikipedia.org/wiki/Parallel_axis_theorem