AALBORG UNIVERSITY
STUDENT REPORT

ED5-3-E16

Hovering control of a quadcopter

Students:
Alexandra Dorina Török
Andrius Kulšinskas

Supervisors:
Christian Mai
Zhenyu Yang

December 19, 2016

AALBORG UNIVERSITY

STUDENT REPORT

School of Information and Communication Technology
Niels Bohrs Vej 8
DK-6700 Esbjerg
http://sict.aau.dk

Title:
Hovering control of a quadcopter

Theme:
Scientific Theme

Project Period:
Autumn Semester 2016

Project Group:
ED5-3-E16

Participant(s):
Alexandra Dorina Török
Andrius Kulšinskas

Supervisor(s):
Christian Mai
Zhenyu Yang

Copies: x

Page Numbers: 61

Date of Completion:
December 19, 2016

Abstract:

xxx

Contents

Preface

The project entitled *Hovering control of a quadcopter* was made by two students from the Electronics and Computer Engineering programme at Aalborg University Esbjerg, for the P5 project during the fifth semester.

From hereby on, every mention of 'we' refers to the two co-authors listed below.

Aalborg University, December 19, 2016.

Andrius Kulšinskas
<akulsi14@student.aau.dk>

Alexandra Dorina Török
<atarak14@student.aau.dk>

Chapter 1

Introduction

1.1 Introduction

The theme of this semester's project lies within *Automation*. Automation can be simply described as being the use of diverse control systems for fulfilling a certain task with little to no human interaction. As known from the previous semesters, a control system is an instrument which has the role of adapting the behaviour of a system according to a desired state, also knows as steady-steady or reference. Any control system has three components: measurement, control and actuation. Without one of these, automation would not be possible. Essentially, the measure reflects the current state of the system, the controller is the brain that given the measurement decides which action will be performed and the actuator is the one executing the action.

Project ideas around the topic of automation are unlimited, since it is so widely spread. Having discussed a few of them that would meet the semester's requirements, we finally decided to work on the control of a quadcopter. Our decision was, for the most part, based on the fact that the university had the required equipment available, which enabled us to start working on the project right away.

Unmanned Aerial Vehicles (UAVs) have been attracting attention for many decades now. Powered UAVs were, at first, utilized by the military to execute reconnaissance missions. Nowadays, they have found other uses, such as aerial photography, search and rescue, delivery, geographic mapping and more. While there are different types of UAVs, our focus is on the multirotors. Multirotor can be defined as a rotorcraft with more than two motors. Based on this, the four most common types are tricopter, quadcopter, hexecopter and octocopter, each having 3, 4, 6 and 8 motors respectively. Each type of multirotor has its ups and downs - more motors mean higher liftforce and more reliable stability, but they also increase both price and damage caused in case of an error. Due to their price, size and ease of setup, quadcopters are the most popular type. They are popularly referred to as drones. Our goal for the project is to design a control system that makes it possible for the quadcopter to be stable - hovering mid-air. Basically, our input will be a certain height

and the quadcopter will have to automatically adjust to that height and maintain its stability when no further inputs are given.

Chapter 2

General characteristics of a quadcopter

Before delving into the mathematical modelling of the quadcopter, it is important to understand a few basic concepts in regards to the working principles of a drone and the frames in which the final model will be defined.

2.1 Systems of coordinates

Describing the movement of the quadcopter in a three dimensional space can be done by using two frames, which can be seen in Figure 2.1. The NED frame is the one that we refer to in our daily life. It points towards North, East and Downwards normal to the surface of Earth. The origin of the NED coordinate system is fixed in O. The frame can also be referred to as inertial and the vectors expressed in it will be marked with ^{I}. The other frame is fixed on the quadcopter, therefore it is called Body-fixed frame. It has the center in O_C, which is the center of mass of the drone. Its three axis are x, y and z. The x axis points towards one of the motors, the z axis points to the ground and the y axis completes the orthogonal coordinate system. The vectors described in this frame will be marked with ^{B} [1] [2] .

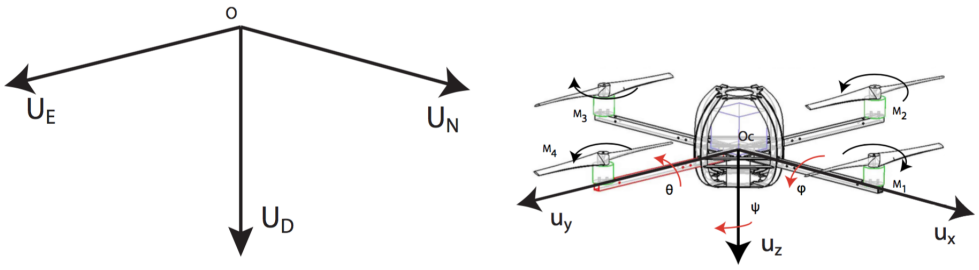

Figure 2.1: Inertial and Body-fixed Frames.

Besides the coordinate system, Figure 2.1 also shows the hexagonal placement of

the four motors, the spinning direction of each of them and the roll - ϕ, pitch - θ and yaw - ψ angles.

The quadcopter's position represents the O to O_C displacement:

$$P^I = [X, Y, Z]^T \tag{2.1}$$

The drone's velocity is express as:

$$V^B = [U, V, W]^T \tag{2.2}$$

The quadcopter's rotation can be described using the 3 Euler angles φ, θ and ψ as:

$$\Psi = [\varphi, \theta, \psi]^T \tag{2.3}$$

The drone's angular velocity can be expressed using the angular velocity about the u_x axis - P, the angular velocity about the u_y axis - Q and the angular velocity about the u_z axis - R:

$$\Omega^B = [P, Q, R]^T \tag{2.4}$$

2.2 Working principle

To have an understanding of how an aerial vehicle actually works like, it is important to know the forces affecting it. See Figure 2.2.

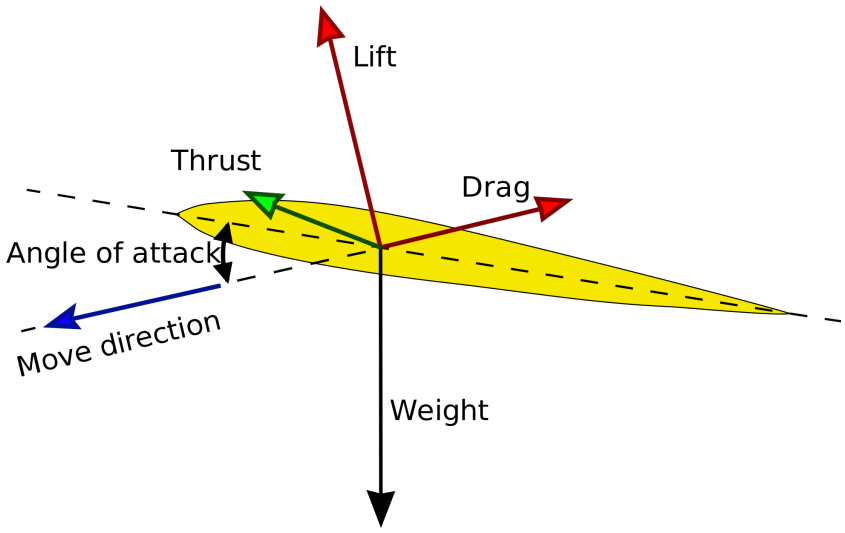

Figure 2.2: Forces Affecting an Aerial Vehicle [3].

The 4 main forces acting on the quadcopter - or any aerial vehicle, for that matter are:

- **Drop** (gravitational force) affects the vehicle at all times. As any object on Earth, its mass is driven towards the centre of the planet. This force is always expressed as: $F_g = mg$, where m is the mass of the drone and g is the gravitational constant.
- **Thrust** is the force generated by the motors that is allowing the vehicle to move towards its heading. In the case of a quadcopter, this force only exists when the force generated by the motors is uneven.
- **Lift** is the force created through a difference in the air pressure above and below the motor (according to Bernoulli's principle). Quadcopter's motors constantly generate lift force, which must be higher than the drop force in order for the vehicle to take flight.
- **Draw** is the resistance created by the air as the vehicle moves through it. It opposes the thrust force and therefore must be lower than the thrust force in order for the quadcopter to move on. In cases when there is no wind, such as indoors area, this force can be disregarded.

If the motors are powered off, the only force affecting the drone is the drop and therefore the quadcopter stays on the ground. In order to lift it up, we need to understand the relationship between quadcopter and thrust to weight ratio - or TWR for short. This ratio can be determined by equation F_t/F_g and describes the vehicle's ability to move up. With TWR expressed as a number, assuming that each motor generates equal amount of thrust, three cases can be identified:

- **TWR<1**: The gravitational force is higher than the lift force and therefore the quadcopter is drawn towards the ground.
- **TWR=1**: The forces are equal, causing quadcopter's altitude to stay constant.
- **TWR>1**: The thrust is higher than drop force, allowing vehicle to move upwards.

Therefore, in order to get a quadcopter up in the air, it is necessary to generate enough thrust for TWR ration to be higher than one. In order to land it, the TWR must be smaller than 1, allowing the quadcopter to move downwards.

2.3 Quadcopter maneuvering

Quadcopters are made with four identical motors, each of them having an attached propeller. As it can be seen in Figure 2.3, each pair of propellers rotate in different directions. The odd numbered motors have clockwise rotation, while even numbered motors have counter-clockwise rotation. The speed of the motors has to be adjusted in order to obtain the maneuvers shown in Figure 2.3.

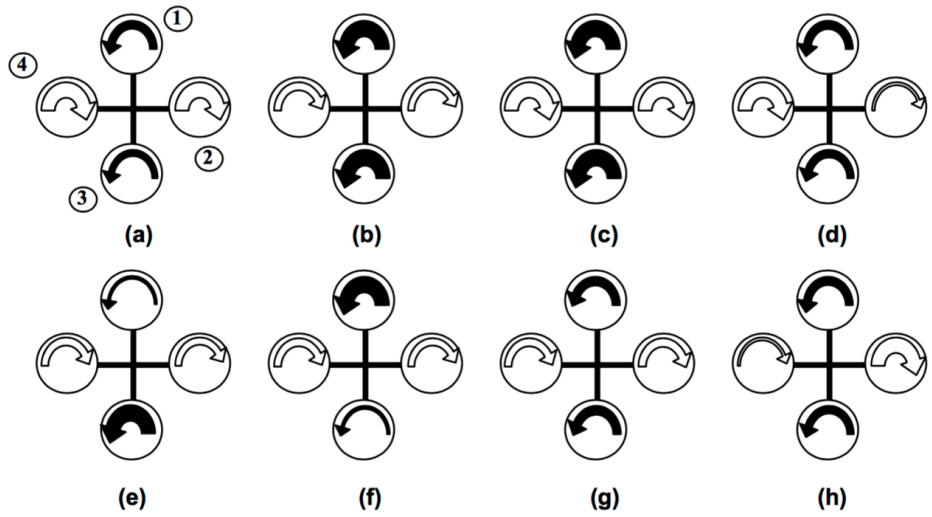

Figure 2.3: Quadcopter Rotation System [4].

The arrows in Figure 2.3 represent the angular speeds of the motors and they can be written as:

$$\omega = [\omega_1, \omega_2, \omega_3, \omega_4]^T \tag{2.5}$$

The quadcopter's possible motions can be split into four categories:

- **Rolling motion** is represented by the rotation of the vehicle about the u_x axis and it can be obtained when ω_2 and ω_4 are manipulated. In order to obtain a positive rolling, ω_4 is decreased while ω_2 is increased. The opposite will result in a negative rolling motion. Both movements can be seen in cases (a) and (b).
- **Pitch motion** is represented by the rotation of the quadcopter about the u_y axis and it can be obtained when ω_1 and ω_3 are manipulated. In order to obtain a positive pitch, ω_3 is decreased while ω_1 is increased. The opposite will result in a negative pitch motion. Both movements can be seen in cases (c) and (d).
- **Yaw motion** is represented by the rotation of the drone about the u_z axis and it can be obtained by having a difference in the torque developed by each pair of propellers. The torque can be changed by having a bigger angular speed on one of the propeller pairs over the other. Both negative and positive yaw motions can be seen in cases (e) and (f).

- **Translational motion** or vertical movement can be obtained by equally increasind or decreasing the angular speeds of all motors as seen in cases (g) and (h) [1].

2.4 Assumptions

Given the fact that stabilizing a quadcopter is quite a complex problem, we will make a few general assumptions that will enable us to make a more simple version of the model:

- The quadcopter is symmetric along u_x and u_y.
- The quadcopter is a rigid body.
- The flapping effects of the rotors are ignored.
- Nonlinearities of the battery are ignored.
- Both accelerometer and gyroscope sensors are considered to be at the center of mass of the quadcopter.
- All motors have the same time constant.
- All aerodynamic forces acting on the quadcopter are ignored.

Chapter 3

Prototype

This chapter will give a general overview on the pieces of technical equipment which we are using and show what the purpose of each one is.

3.1 Prototype Hardware

3.1.1 Motors

Controlling a quadcopter can be done efficiently by using high-quality motors with fast response, which will ensure more of a stable flight. The motors must also be powerful enough to be able to lift the quadcopter and perform the required aerial movements.

The motor that we are using is the Turnigy Multistar Brushless Motor seen in Figure 3.1.

Figure 3.1: Turnigy Multistar 2213-980 V2 Brushless Motor.

3.1.2 Propellers

The propellers don't have such strict requirements as the motors. They are needed to be light and have a size and lift potential in order for the quadcopter to hover at less than 50% of the motor capacity. For our quadcopter, we are using plasting 10x4.5" propellers with light weight - 60g. They have a length of 254 mm and a pitch inclination of 114mm. They can be seen in Figure 3.2.

Figure 3.2: Hobbyking Slowfly Propeller 10x4.5.

3.1.3 Electric Speed Controller

Electronic Speed Controller (ESC) is a widely used device in rotorcrafts. The purpose of an ESC is to vary the electric motor's speed. They also come with programmable features, such as braking or selecting appropriate type of battery. We need the ESC to have a fast response, for the same reasons mentioned for the motors in Section **??**. The ESC that we are using is the TURNIGY Plush 30A which is shown in Figure 3.3.

Figure 3.3: TURNIGY Plush 30A Speed Controller.

3.1.4 APM Flight Controller

ArduPilotMega (APM) is an open source unmanned aerial vehicle (UAV) platform which is able to control autonomous multicopters. It is illustrated in Figure 3.4. The system was improved uses Inertial Measurement Unit (IMU) - a combination of accelerometers, gyroscopes and magnetometers. The "Ardu" part of the project name shows that the programming can be done using Arduino open-source language.

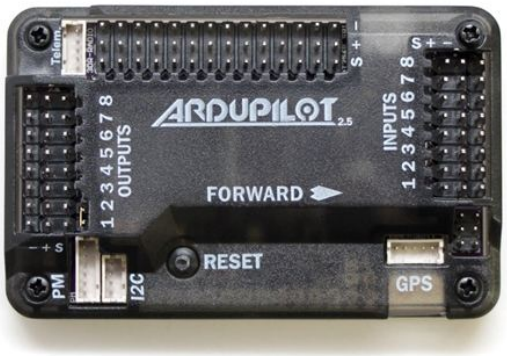

Figure 3.4: APM 2.5 Board.

3.1.5 Power Distribution Board

To reduce the number of connections straight to the battery, we used the a power distribution board made for a previous project. A board like this is an easy solution since it enables us to connect the four ESCs directly to the board and then connect the board to the battery.

3.1.6 Ultrasonic Sensors

Four HC-SR04 Ultrasonic Sensors are located on each side of the drone. Containing four pins - V_{cc}, GND, $Trig$ and $Echo$, these easy-to-use sensors detect objects at up to $400cm$ away.

Figure 3.5: HC-SR04 Ultrasonic Sensor.

3.1.7 Battery

To power up our quadcopter, we will use a TURNIGY nano-tech Lipoly battery, which can be seen in Figure 3.6. Higher voltage under load, straighter discharge curves and excellent performance are the factors that make it suitable for our project.

Figure 3.6: Turnigy Nano-tech 6000mah 3S 25 50C Lipo Pack.

3.1.8 Drone Frame

The drone frame, including arms for the motors, supporting legs and surface for the flight controller and battery, has been chosen and assembled by a previous group. The complete drone with all of its components put together can be seen in Figure 3.7.

Figure 3.7: Fully Assembled Drone.

The wiring of the electronic components can be seen in Figure 3.8.

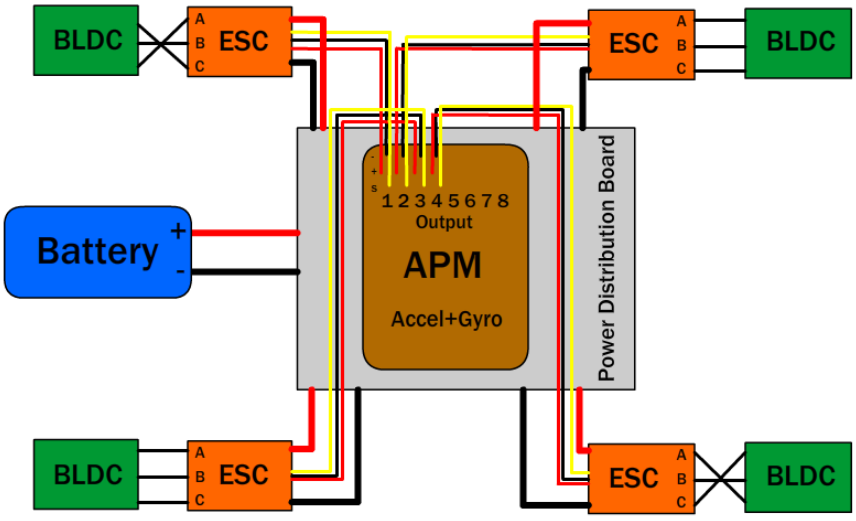

Figure 3.8: Wiring of the Electronic Components.

3.2 Prototype Measurements

The arm length has been measured to be 24.5cm and the total height of the drone has been estimated to be 13.4cm. To obtain the total weight of the drone, all parts of it, including the battery, were mounted on the frame. Then, a piece of rope was

tied around all arms of the vehicle, to lift it up in the air. The other end of the rope was put on a hook of PASCO PS-3202 newtonmeter. The device was communicating its data to the PASCO Capstone software on a computer, giving a plot of force over time. The following plot can be seen in Figure 3.9.

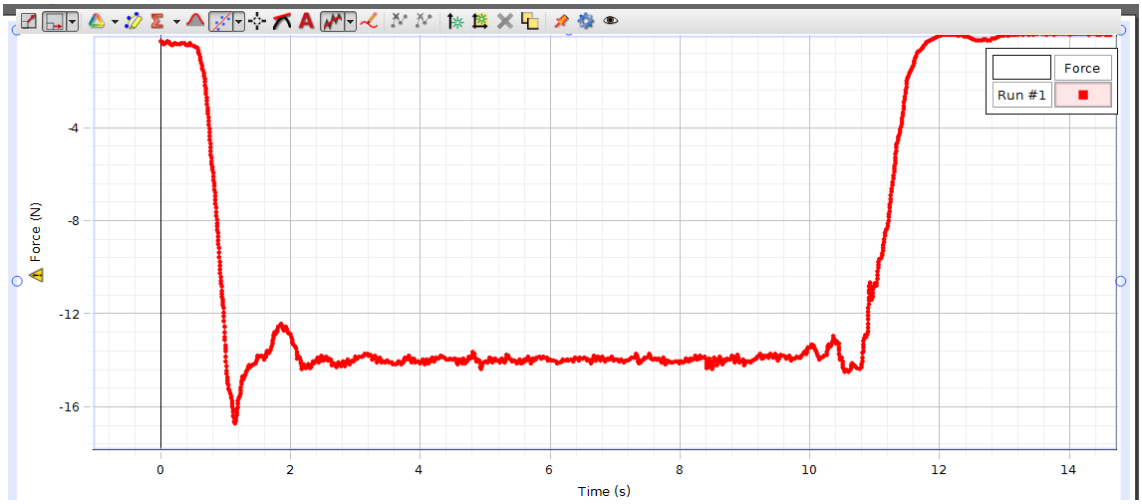

Figure 3.9: Force vs. Time Plot of the Drone.

After some time period, the prototype was lifted into the air to let the gravitational force affecting the drone be measured by the newtonmeter. It was measured that the gravitational force affecting the vehicle is approximately $14N$. From Newton's second law, we know that $F = ma$, where in this particular scenario, acceleration a is equal to the gravitational force $g = 9.81m/s^2$. Therefore, the mass of the drone is equal to $m = \frac{F}{g} = \frac{14N}{9.81m/s^2} = 1.427kg$.

Chapter 4

Quadcopter Model

Quadcopter control is a complex, yet interesting problem. One of the reasons why this control problem is challenging is the fact that a quadcopter has six degrees of freedom, but only four inputs which affects the linearity of the dynamics and makes the quadcopter under-actuated.

In this chapter we will take a look at the kinematics and dynamics equations that describe our quadcopter system for which we have drawn the block diagram shown in Figure 4.1.

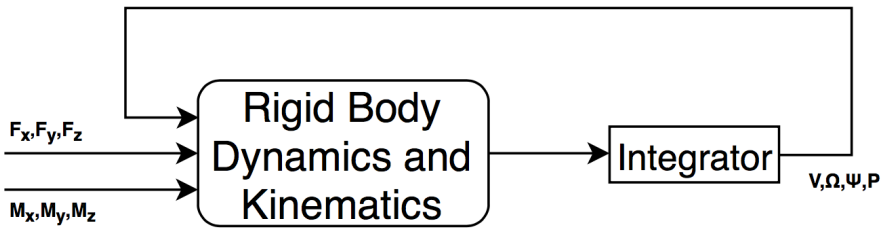

Figure 4.1: Dynamics and Kinematics Block Diagram.

4.1 Quaternions and Euler angles

In Section 2.1, we have explained how the position of the quadcopter is expressed in the inertial frame and how the velocity of the quadcopter is express in the body-fixed frame. Therefore, we need to be able to find a relationship between the two, so that we can move from one to the other.

Euler angles have to be applied as a sequence of rotations. This report will use the *roll, pitch, yaw* convention. Therefore, the roll rotation will be $(R(\phi)^T)$, the pitch rotation will be $(R(\theta)^T)$ and the yaw rotation will be $(R(\psi)^T)$. Equations 4.1, 4.2 and 4.3 describe the quadcopter's orientation relative to the inertial frame in matrix form:

$$R(\phi)^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{4.1}$$

$$R(\theta)^T = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{4.2}$$

$$R(\psi)^T = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Merging the three rotations as:

$$S = R(\phi)^T R(\theta)^T R(\psi)^T \tag{4.4}$$

gives the rotation matrix - S, which expresses a vector from the inertial frame to the body-fixed frame:

$$S = \begin{bmatrix} cos\theta cos\psi & cos\theta sin\psi & -sin\theta \\ cos\psi sin\phi sin\theta - cos\phi sin\psi & sin\psi sin\theta sin\phi + cos\phi cos\psi & cos\theta sin\phi \\ cos\psi cos\phi sin\theta & cos\phi sin\theta sin\psi - cos\psi sin\phi & cos\theta cos\phi \end{bmatrix} \tag{4.5}$$

We can notice that if we were to have $\theta = \Pi/2$, the rotation matrix brings out a singularity by turning into:

$$S = \begin{bmatrix} 0 & 0 & -1 \\ sin(\phi - \psi) & cos(\phi - \psi) & 0 \\ cos(\phi - \psi) & -sin(\phi - \psi) & 0 \end{bmatrix} \tag{4.6}$$

As a result, one degree of freedom in the three dimensional space is lost. In addition, a change in either ϕ or ψ will now have the same effect, which causes confusion. In order to be able to avoid this problem, a quaternion-based method can be applied instead.

A quaternion can be expressed as $q = [q_0, q_1, q_2, q_3]^T$, which yields:

$$q = \begin{bmatrix} cos(\phi/2)cos(\theta/2)cos(\psi/2) + sin(\phi/2)sin(\theta/2)sin(\psi/2) \\ sin(\phi/2)cos(\theta/2)cos(\psi/2) - cos(\phi/2)sin(\theta/2)sin(\psi/2) \\ cos(\phi/2)sin(\theta/2)cos(\psi/2) + sin(\phi/2)cos(\theta/2)sin(\psi/2) \\ cos(\phi/2)cos(\theta/2)sin(\psi/2) - sin(\phi/2)sin(\theta/2)cos(\psi/2) \end{bmatrix} \tag{4.7}$$

The quaternion and Euler angles are equivalent in terms of attitude, therefore we can convert from one representation to the other. The rotation matrix can be then rewritten as:

$$S_q = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_3q_0) & 2(q_1q_3-q_2q_0) \\ 2(q_1q_2-q_3q_0) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_1q_0) \\ 2(q_1q_3+q_2q_0) & 2(q_2q_3-q_1q_0) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{4.8}$$

4.2 Quaternions Representation

Knowing the position of the quadcopter relative to the inertial frame, the linear velocity in the body frame and the rotation matrix, a relationship between the three can be identified as:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = S_q^T \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{4.9}$$

The quadcopter's attitude can be written using the angular velocities vector $\Omega^B = [P, Q, R]^T$:

$$\begin{bmatrix} \dot{q_0} \\ \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -P & -Q & -R \\ P & 0 & R & -Q \\ Q & -R & 0 & P \\ R & Q & -P & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{4.10}$$

Because we chose to neglect all other forces acting on the quadcopter except of the propeller thrust and gravity, we can further express the forces produced by the propellers along the u_z axis as $[F_1, F_2, F_3, F_4]^T$. Therefore, the force in the body-fixed frame can be written as $F^B = [F_x, F_y, F_z]^T = [0, 0, -\sum_{i=1}^{4} F_i]^T$. Each force produces a moment around the axes of the quadcopter and as a result the moment in the body-fixed frame can be expressed as: $M^B = [M_x, M_y, M_z]^T$. These moments are explained in more detail in Chapter x.

The dynamics of the quadcopter in regards to the rotations are given by $I\dot{\Omega} = -\Omega \times I\Omega + M_B$, which yields:

$$\begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} \frac{M_x}{I_x} \\ \frac{M_y}{I_y} \\ \frac{M_z}{I_z} \end{bmatrix} - \begin{bmatrix} \frac{(I_z-I_y)QR}{I_x} \\ \frac{(I_x-I_z)QR}{I_y} \\ \frac{(I_y-I_x)QR}{I_z} \end{bmatrix} \tag{4.11}$$

where $I = diag(I_x, I_y, I_z)$ is the inertia matrix.

Finally, by applying Newton's second law, we obtain $ma^B = F^B + mS_q g^I - \Omega \times V^B$, where $a^B = \dot{V}^B = [dotU, dotV, dotW]^T$ is the acceleration vector and $g^I = [0, 0, g_0]^T$ is the gravity vector with $g = 9.81 m/s^2$.

Therefore:

$$\begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} + g_0 \begin{bmatrix} 2(q_1 q_3 - q_2 q_0) \\ 2(q_2 q_3 + q_1 q_0) \\ 1 - 2(q_1^2 + q_2^2) \end{bmatrix} - \begin{bmatrix} QW - RV \\ RU - PW \\ PV - QU \end{bmatrix} \tag{4.12}$$

4.3 Euler Representation

In order to make the Euler angle rates correspond to the angular velocity vector, we can write:

$$\begin{bmatrix} P \\ Q \\ R \end{bmatrix} = R(\phi)^T R(\theta)^T R(\psi)^T \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R(\phi)^T R(\theta)^T \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R(\phi)^T \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{4.13}$$

Solving for Euler angles rates finally gives:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & tan\theta sin\phi & tan\theta cos\phi \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi/cos\theta & cos\phi/cos\theta \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{4.14}$$

Chapter 5

Sensors

This chapter will, step by step, introduce the sensors, describe how they function and define the methods used to extract the data from them. The ultrasonic sensor has a pretty straightforward functionality - a signal is sent out from the *trigger* pin, while the *echo* pin catches the signal coming back and estimates the distance. The accelerometer and gyroscope that we are using are both part of the built-in IMU of the Ardupilot, fitted on 6-axis MPU6000 14-bit chip. The accelerometer works by detecting a force that is actually the opposite of the acceleration vector. This force is not always caused by acceleration, but it can be. It just happens that acceleration causes an inertial force that is captured by the force detection mechanism of the accelerometer. The gyroscope measures the rotation around one of the axes.

5.1 Ultrasonic Sensor

The HC-SR04 datasheet [5] states that the sensor can measure distances between 2 and $400cm$, with $3mm$ accuracy. The trigger pin, when commanded to, orders the module to send out an 8 cycle sonic burst at $40kHz$ frequency. Then, if any of the signals return, the distance is calculated. The signal travel time is measured based on the time the module stayed on high level. While distance travelled d is equal to velocity of sound $v(340m/s) \times t$ - the time, it should be noted that the signal travels both from and back to the sensor. Therefore, the whole equation should be divided by 2, in order to find the real distance between the sensor and the object. Utilizing the $NewPing$ library for ultrasonic sensors, a simple code can be made, found in appendix Listing 12.1. In order to determine the performance of the sensor, it was placed in front of a stationary object. The code ran for 30 seconds and produced 20262 readings - resulting in frequency of $675Hz$. The output in cm was plotted against time in Matlab, producing Figure 5.1.

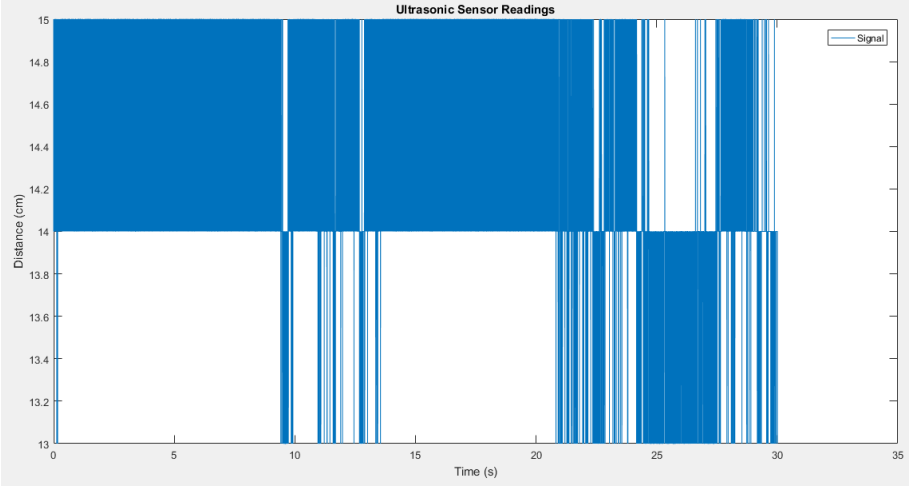

Figure 5.1: Ultrasonic Sensor Results over 30 Seconds.

It is evident, that although the readings are not completely constant, the error is very minimal and no filter is needed.

5.2 Accelerometer

According to the MPU-6000 datasheet [6], the raw accelerometer data can be converted into multiples of g by dividing with a factor of 16384. However, we have computed the rotations around the axes by using an alternative method. We have first chosen a reference position, which is the typical orientation of a device with the x and y axis in the 0 g field plane and the x axis in the 1 g field. This is shown in figure REF, where θ is the angle between the horizon and the x-axis of the accelerometer, ψ as the angle between the horizon and the y-axis of the accelerometer, and ϕ as the angle between the gravity vector and the z-axis. When in the initial position of *0* g on the x- and y-axes and 1 g on the z-axis, all calculated angles would be 0 degrees [7].

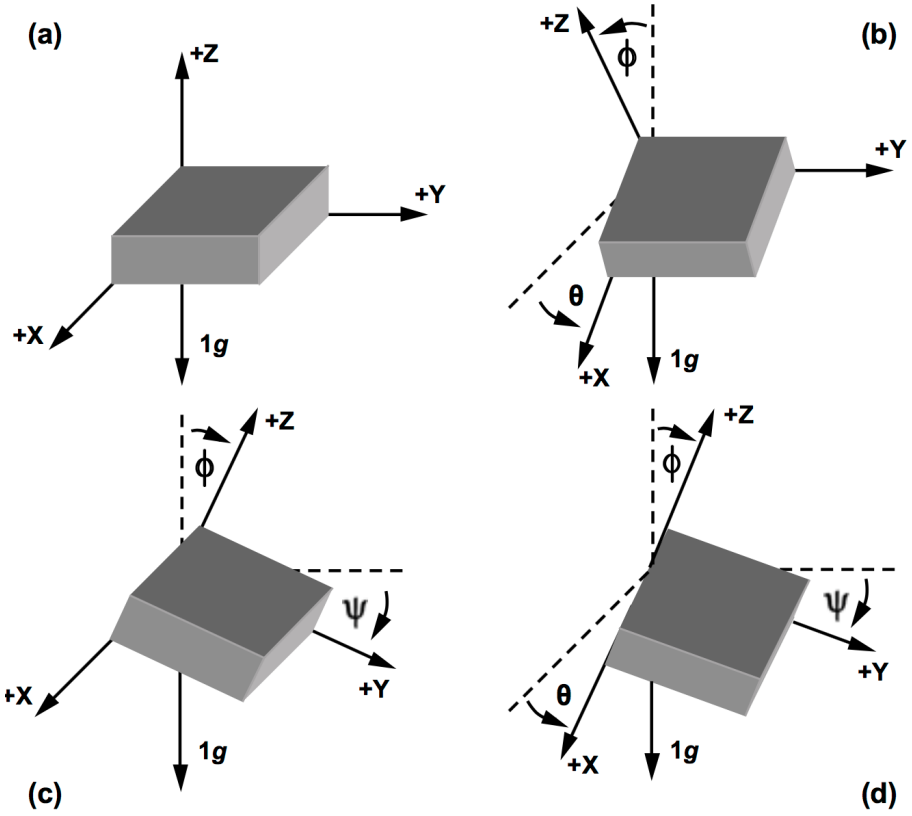

Figure 5.2: Inclination Angles [7]

Therefore, by using basic trigonometry, the formulas for the converted accelerometer data are:

$$\theta = tan^{-1}(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}) \tag{5.1}$$

$$\psi = tan^{-1}(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}) \tag{5.2}$$

$$\phi = tan^{-1}(\frac{\sqrt{A_X^2 + A_Y^2}}{A_Z}) \tag{5.3}$$

The inversion of the last equation is due to the initial position being a 1 g field. If the horizon is desired as the reference for the z-axis, the operand can be inverted. A positive angle means that the corresponding positive axis of the accelerometer is pointed above the horizon, whereas a negative angle means that the axis is pointed below the horizon. Because the inverse tangent function and a ratio of accelerations is used, the benefits mentioned in the dual-axis example apply, namely that the effective incremental sensitivity is constant and that the angles can be accurately measured for all points around the unit sphere.

An important thing to keep in mind is that the accelerometer provides accurate orientation angles when gravity is the only force acting on the sensor. When manouvering the sensor, other forces are being applied to it, which causes a fluctuation in the measurements. As a result, the accelerometer provides accurate data over the long term, but is noisy in the short term [8].

5.3 Gyroscope

Computing orientation from the gyroscope sensor is different, since the gyroscope measures angular velocity (the rate of change in orientation angle), not angular orientation itself. To compute the orientation, we must first initialize the sensor position, then measure the angular velocity ω around the X, Y and Z axes at measured intervals (Δt). Then $\omega \times \Delta t$ is the change in angle. The new orientation angle will be the original angle plus this change. The problem with this approach is that we are integrating – adding up many small computed intervals – to find orientation. Repeatedly adding up increments of $\omega \times \Delta t$ will result in small systematic errors becoming magnified over time. This is the cause of gyroscopic drift, and over long timescales the gyroscope data will become increasingly inaccurate. In short, the gyroscope provides accurate data about changing orientation in the short term, but the necessary integration causes the results to drift over longer time scales.

The MPU-6000 datasheet [6] shows that dividing the raw gyroscope values by 131 gives angular velocity in degrees per second, which multiplied by the time between sensor readings, gives the change in angular position. If we save the previous angular position, we simply add the computed change each time to find the new value.

5.4 Programming

It is important to understand how the sensors can actually deliver us the data. Usually, they fall into two categories: analogue and digital. The one we are using is digital and it can be programmed using I2C, SPI or USART communication. We have decided on SPI and the setup code for it is shown below:

```
    //As per APM standard code, stop the barometer from holding the SPI bus
    pinMode(40, OUTPUT);
    digitalWrite(40, HIGH);
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV16);
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    delay(100);
    pinMode(ChipSelPin1, OUTPUT);
    ConfigureMPU6000();  // configure chip
}
```

Code Listing 5.1: Setup

As we can see, the setup ends by calling the *ConfigureMPU6000()* function:

```
  // SMPRT_DIV @ SMPRT_DIV, sample rate at 1000Hz
  SPIwrite(0x19,0x00,ChipSelPin1);
  delay(150);

  // DLPF_CFG @ CONFIG, digital low pass filter at 42Hz
  SPIwrite(0x1A,0x03,ChipSelPin1);
  delay(150);

  // FS_SEL @ GYRO_CONFIG, gyro scale at 250dps
  SPIwrite(0x1B,0x00,ChipSelPin1);
  delay(150);

  // AFS_SEL @ ACCEL_CONFIG, accel scale at 2g (1g=8192)
  SPIwrite(0x1C,0x00,ChipSelPin1);
  delay(150);
}
```

Code Listing 5.2: Function to configure chip

Firstly, we need to get the raw values from both accelerometer and gyroscope and that is done by accessing their registers. We have made separate functions, which can be seen in Code REFERENCE, that read from the registers of the accelerometer and gyroscope and that return the raw values of each sensor. These functions are very similar, the only difference being the registers that are accessed to get the information needed.

```

int AcceZ(int ChipSelPin) {
  uint8_t AcceZ_H=SPIread(0x3F,ChipSelPin);
  uint8_t AcceZ_L=SPIread(0x40,ChipSelPin);
  int16_t AcceZ=AcceZ_H<<8|AcceZ_L;
```

Code Listing 5.3: Function that accesses registers and returns a raw value.

```

int AcceZ(int ChipSelPin) {
  uint8_t AcceZ_H=SPIread(0x3F,ChipSelPin);
  uint8_t AcceZ_L=SPIread(0x40,ChipSelPin);
  int16_t AcceZ=AcceZ_H<<8|AcceZ_L;
  return(AcceZ);
}

int GyroX(int ChipSelPin) {
  uint8_t GyroX_H=SPIread(0x43,ChipSelPin);
  uint8_t GyroX_L=SPIread(0x44,ChipSelPin);
  int16_t GyroX=GyroX_H<<8|GyroX_L;
  return(GyroX);
}

int GyroY(int ChipSelPin) {
```

```
20    uint8_t GyroY_H=SPIread(0x45,ChipSelPin);
      uint8_t GyroY_L=SPIread(0x46,ChipSelPin);
22    int16_t GyroY=GyroY_H<<8|GyroY_L;
      return(GyroY);
24  }

26
    int GyroZ(int ChipSelPin) {
28    uint8_t GyroZ_H=SPIread(0x47,ChipSelPin);
      uint8_t GyroZ_L=SPIread(0x48,ChipSelPin);
30    int16_t GyroZ=GyroZ_H<<8|GyroZ_L;
      return(GyroZ);
32  }

34
    //--- Function to obtain angles based on accelerometer readings ---//
36  float AcceDeg(int ChipSelPin,int AxisSelect) {
      float Ax=ToG(AcceX(ChipSelPin));
38    float Ay=ToG(AcceY(ChipSelPin));
      float Az=ToG(AcceZ(ChipSelPin));
40    float ADegX=((atan(Ax/(sqrt((Ay*Ay)+(Az*Az)))))/PI)*180;
      float ADegY=((atan(Ay/(sqrt((Ax*Ax)+(Az*Az)))))/PI)*180;
42    float ADegZ=((atan((sqrt((Ax*Ax)+(Ay*Ay)))/Az))/PI)*180;
      switch (AxisSelect)
44    {
        case 0:
46      return ADegX;
```

Code Listing 5.4: Functions that accesses the registers and returns raw values

If we want to calculate the inclination of a device relative to the ground for example, we can calculate the angle between the force vector and one of the z-axis. We can do that by using the formulas in SECTION X and by degree conversion. The functions that take care of converting the raw values from both accelerometer and gyroscope are the following:

```
      break;
2     case 2:
      return ADegZ;
4     break;
    }
6   }

8
    //--- Function to obtain angles based on gyroscope readings ---//
10  float GyroDeg(int ChipSelPin, int AxisSelect) {
      time_old=time;
12    time=millis();
      float dt=time-time_old;
14    if (dt>=1000)
      {
16      dt=0;
      }
18    float Gx=ToD(GyroX(ChipSelPin));
      float Gy=1-ToD(GyroY(ChipSelPin));
```

```
20    float Gz=ToD(GyroZ(ChipSelPin));
```

Code Listing 5.5: Function to obtain angles based on the raw values of the accelerometer.

```
1   switch (AxisSelect)
    {
3     case 0:
      return angleX;
5     break;
      case 1:
7     return angleY;
      break;
9     case 2:
      return angleZ;
11    break;
    }
13  }

15  void ConfigureMPU6000()
    {
17    // DEVICE_RESET @ PWR_MGMT_1, reset device
      SPIwrite(0x6B,0x80,ChipSelPin1);
19    delay(150);

21    // TEMP_DIS @ PWR_MGMT_1, wake device and select GyroZ clock
      SPIwrite(0x6B,0x03,ChipSelPin1);
23    delay(150);

25    // I2C_IF_DIS @ USER_CTRL, disable I2C interface
      SPIwrite(0x6A,0x10,ChipSelPin1);
27    delay(150);
```

Code Listing 5.6: Function to obtain angles based on the raw values of the accelerometer.

We have used the Serial Monitor within the Arduino IDE environment to read the outputs from both sensors and they seem to be accurate. The difference in the values of the accelerometer output is because accelerometers are more sensitive to noise and vibrations. The figure below shows the printed outputs at steady state, that is when the quadcopter is parallel to the ground.

The serial monitor was used to print out the the angles. At steady state, it yielded the results seen in Figure 5.3.

```
Acc X 0.48   Acc Y -0.27   Acc Z 0.54   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.71   Acc Y -0.43   Acc Z 0.88   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.60   Acc Y -0.42   Acc Z 0.70   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.65   Acc Y -0.32   Acc Z 0.71   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.66   Acc Y -0.35   Acc Z 0.79   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.62   Acc Y -0.45   Acc Z 0.79   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.76   Acc Y -0.35   Acc Z 0.71   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.60   Acc Y -0.20   Acc Z 0.72   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.66   Acc Y -0.29   Acc Z 0.73   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.68   Acc Y -0.19   Acc Z 0.69   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.75   Acc Y -0.25   Acc Z 0.79   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.65   Acc Y -0.22   Acc Z 0.63   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.69   Acc Y -0.26   Acc Z 0.59   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.69   Acc Y -0.19   Acc Z 0.62   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.68   Acc Y -0.37   Acc Z 0.76   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.64   Acc Y -0.33   Acc Z 0.74   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.53   Acc Y -0.28   Acc Z 0.59   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.67   Acc Y -0.23   Acc Z 0.75   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.69   Acc Y -0.23   Acc Z 0.70   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.52   Acc Y -0.45   Acc Z 0.75   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.67   Acc Y -0.18   Acc Z 0.62   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.55   Acc Y -0.31   Acc Z 0.60   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.64   Acc Y -0.16   Acc Z 0.64   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.74   Acc Y -0.35   Acc Z 0.79   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.75   Acc Y -0.26   Acc Z 0.73   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.71   Acc Y -0.39   Acc Z 0.82   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.55   Acc Y -0.31   Acc Z 0.61   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.60   Acc Y -0.31   Acc Z 0.67   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.52   Acc Y -0.22   Acc Z 0.63   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.69   Acc Y -0.18   Acc Z 0.78   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.61   Acc Y -0.35   Acc Z 0.64   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.73   Acc Y -0.28   Acc Z 0.68   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.62   Acc Y -0.26   Acc Z 0.74   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.64   Acc Y -0.25   Acc Z 0.74   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.64   Acc Y -0.23   Acc Z 0.68   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
Acc X 0.68   Acc Y -0.10   Acc Z 0.70   Gyro X 0.00   Gyro Y 0.00   Gyro Z 0.00
```

Figure 5.3: Angles Obtained from Accelerometer and Gyroscope.

//insert prinscreen for tilting

5.5 Filtering sensor data

The configuration for the accelerometer and gyroscope enables a low-pass filter at 42Hz. However, looking at the data that the accelerometer reads, there is still some considerable amount of noise and occasional spikes in a steady state. In order to design a filter to counter these problems, flight of the prototype had to be simulated, to record the sensor values when the vehicle goes up in the air and tires to stabilize itself. Therefore, flight controller was unmounted from the frame, connected to a computer and then raised in the air, while tilting it to the sides. The test ran

for 30 seconds and provided a 4591 × 6 matrix, containing 4591 readings for all 3 axis of accelerometer and gyroscope. The plotted data over time for x-axis of the accelerometer and gyroscope can be seen in Figure 5.4.

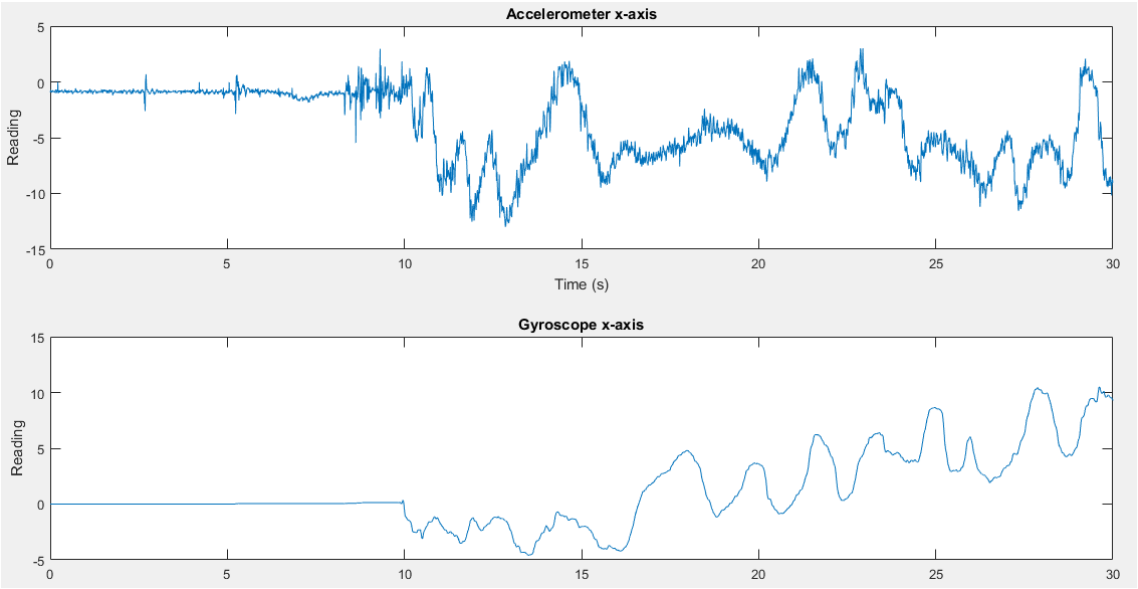

Figure 5.4: Accelerometer and gyroscope x-axis test results.

The sensor is running at approximately $\frac{4591}{30s} = 153Hz$ frequency, making any additional filter more difficult to perform. However, looking at the test data, it is evident that gyroscope provides clean results just with the help of on-board filter. The main focus then shifts towards the accelerometer - looking at the spectrogram of the accelerometer x-axis data (see Figure 5.5, a low-pass FIR filter is chosen.

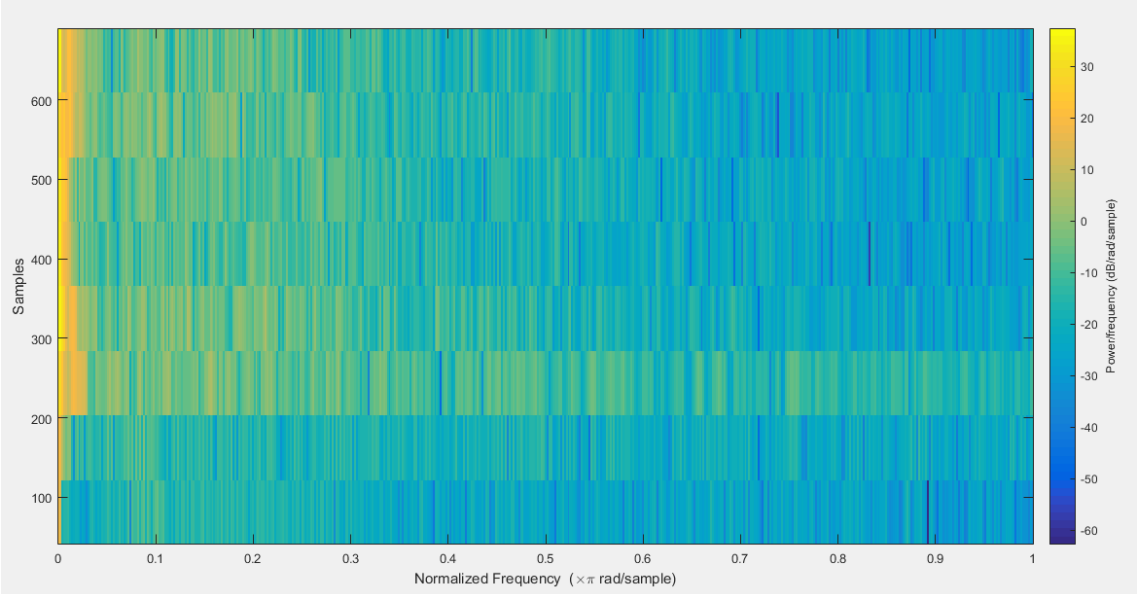

Figure 5.5: Accelerometer x-axis Spectrogram.

In order to get rid of as much noise as possible, the cut-off frequency was chosen to be the maximum possible value - half of the sampling frequency. The order for the filter was chosen to be 10, mostly through trial and error - at this point, the filter has considerable effect, while not having a very high order, which carries its own cost. The filter was applied to the signal using simple code in Matlab, seen in Listing 5.7.

```
Fs = 153;                          % sample rate in Hz
N = 4591;                           % number of signal samples
a1 = z(:,[1]);          % signal taken from the imported matrix
t = (0:N-1)/Fs;                    % time vector
df1 = designfilt('lowpassfir','FilterOrder',10,'CutoffFrequency',Fnorm)
    ;
y1 = filter(df1,a1);
```

Code Listing 5.7: Filter generation code.

The filter was proven to be quite efficient, as seen in Figure 5.6. While not completely perfect, possible improvements will be covered in the discussion chapter.

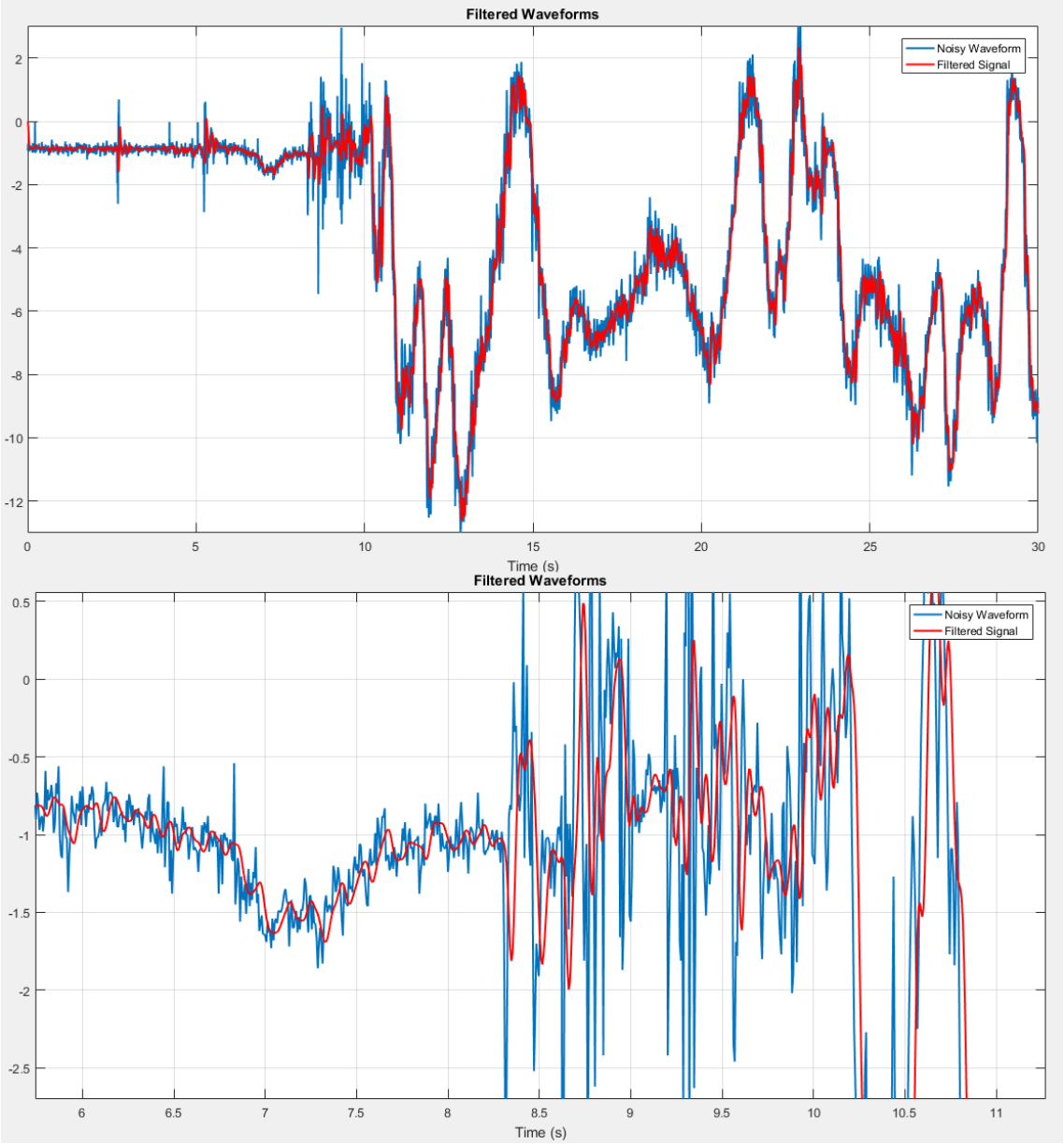

Figure 5.6: Implemented Filter and its Close-up.

Chapter 6

Actuators

Actuators are an important part of any control system because they are the ones responsible for bringing it to the desired state. They do this by applying forces on the system. In our case, the actuators are the motor and the propellers. Figure x displays the actuators' configuration. Ardupilot, which is the "brain" of our control system and has the controller implemented on it is connected to the speed controller, which is in charge of controlling the motor through a PWM signal.

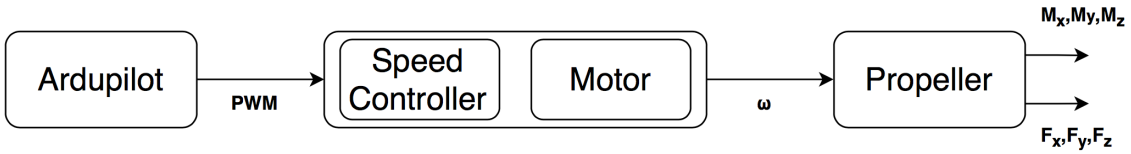

Figure 6.1: Actuators Block Diagram.

Each motor produces an angular velocity ω, therefore the propeller spins at that defined angular velocity. In this chapter, we will be introducing more comprehensible models for both motor and propeller as well as some experiments regarding the PWM signal - motor speed relationship.

6.1 Propeller model

For this part of the report, we have decided not to take into account the aerodynamic forces that act on the propeller, therefore neglecting the air friction, the flapping of the biased and the ground effect. This enables us to create a simpler model, which is easy to understand and implement.

The thrust F_i and the moment across the M_i along the u_z axis can be written as functions of angular velocities such as:

$$F_i = K_T \omega_i^2 \tag{6.1}$$

$$M_i = K_M \omega_i^2 \tag{6.2}$$

where the constant related to thrust K_T and the constant related to the moment K_M can be describes as:

$$K_T = c_T \frac{\rho D^4}{4\Pi^2} \tag{6.3}$$

$$K_M = c_P \frac{\rho D^5}{8\Pi^2} \tag{6.4}$$

with D = being the diameter of the propeller, ρ being the air density, c_T and c_P being the thrust and power coefficient. We are considering these values to have small values (<0.2) based on what we have read from other projects.

Therefore [1]:

$$K_T \approx 1.45 \times 10^{-5} \tag{6.5}$$

$$K_M \approx 3.5 \times 10^{-7} \tag{6.6}$$

We can write $K \approx K_M / K_T = 4.1 \times 10^{-7} = 0.024$. The forces and moments can now be expressed as:

$$F_i = K_T \omega_i^2 \tag{6.7}$$

$$M_x = (\omega_2^2 - \omega_4^2) K_T D = (F_2 - F_4) D \tag{6.8}$$

$$M_y = (\omega_1^2 - \omega_3^2) K_T D = (F_1 - F_3) D \tag{6.9}$$

$$M_z = (\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2) K_M = (F_1 + F_3 - F_2 - F_4) K \tag{6.10}$$

where D is expressed in centimetres.

6.2 Motor model

Providing a signal to the motor is done through Ardupilot, using PWM, which enables providing an analogue signal by digital means. In order to create a PWM signal, we have used the *write.Microseconds()* function from the*Servo.h* library within the Arduino IDE environment. The role of the speed controllers is to turn the PWM signal into a three-phase signal to feed the BLDC motors.

Each motor receives a three-phase signal that is proportional with the angular velocity ω. In general, the behaviour of a motor can be analysed by looking at the electrical and mechanical part of its structure. These parts are represented in Figure 6.2.

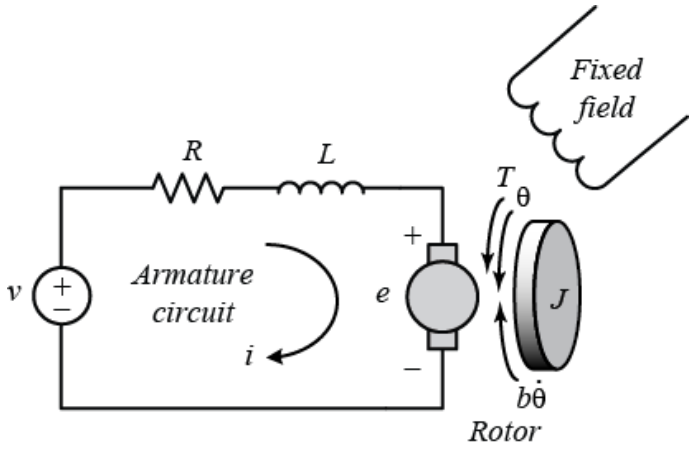

Figure 6.2: Electrical and Mechanical Part of the Motor. [9]

A second order system can be obtained by writing the equations that describe the electrical and mechanical part as:

$$V = RI + L\frac{dI}{dt} + K_e\dot{\theta} \tag{6.11}$$

$$K_eI = J\ddot{\theta} + b\dot{\theta} \tag{6.12}$$

The transfer function can be taking the Laplace transform of each equation and manipulating it into:

$$\frac{\omega}{PWM} = \frac{K_e}{(Js+b)(Ls+R)+K_e^2} \tag{6.13}$$

where ω is the angular velocity of the motor, PWM is the signal to the motor, K_e is the electromotive constant, J is the rotor's moment of inertia, b is the damping ratio of the mechanical system, L is the inductance, R is the resistance and I is the measurement of the current.

Equation 6.13 can be further simplified and written as a first order system, since it is difficult to measure all the motor parameters: [1]

$$\frac{\omega}{PWM} = \frac{k_i}{\tau s + 1} \tag{6.14}$$

where τ is the time constant of the system and k_i is the DC gain.

6.3 Electronic Speed Controllers

ESCs have 5 input pins, 3 of which come from the flight controller. These 3 wires supply the signal for the ESC to translate into the angle of the shaft for the motor. Before ESCs can be used, they need to be properly calibrated. Programming additional settings is optional, but in most cases necessary too. Normally, the calibration for UAVs is done by setting the throttle to full on the radio controller before powering the ESCs. However, since the board translates the throttle signal into a PWM signal, it is possible to calibrate and run the motors directly from the flight controller by sending a PWM signal using software.

6.3.1 Calibration and Programming

The calibration is done by first sending the maximum length of signal the user wishes to use. The ESC emits sounds that are brand-specific and indicate whether the signal was successfully recognised or not. Once the maximum signal is accepted, the user the emits the minimum signal and waits for approval. Additional sound is then emitted, indicating that the calibration was successful.

Using Arduino's Servo library, a self-explanatory built-in function *servo.writeMicroseconds(int value)* is used to send the signal. By default, the library has the minimum signal set to $544\mu s$ and the maximum - to $2400\mu s$. For this project, we shortened the range down to $700\mu s$ and $2000\mu s$ for both signals respectively. A commented code used to calibrate the ESC connected to the first pin of the board can be seen in Listing 6.1.

```
#define MAX_SIGNAL 2000 //define max and min signal length
#define MIN_SIGNAL 700
#define MOTOR_PIN1 12 //pin 1 on the board
Servo motor1;
void setup() {
  Serial.begin(9600); //begin serial communication at 9600 rate
  Serial.println("Program begin...");
  motor1.attach(MOTOR_PIN1); //attach the pin to the servo variable
  Serial.println("Now writing maximum output.");
  Serial.println("Turn on power source, then wait 2 seconds and press
    any key.");
  motor1.writeMicroseconds(MAX_SIGNAL); //this signal should only be
    sent when calibrating for the first time
```

```
12    while (!Serial.available()); //wait for an input, so you have time to
        plug in the ESC
      Serial.read();
14    Serial.println("Sending minimum output");
      motor1.writeMicroseconds(MIN_SIGNAL); //send the min signal, which is
        the only needed signal when already calibrated
16    while (!Serial.available()); //wait for the beeps to indicate
       calibration is done
      Serial.read();
18    Serial.println("Calibrated");
}
20
void loop() {
22
}
```

Code Listing 6.1: Calibrating the ESC.

Programming additional settings of the ESC is done in a similar manner - by sending minimum and maximum signals after the ESC emits particular sounds, indicating wanted selections. The programmable features and selections are brand-specific and can be found in the datasheets. For the purposes of this project, the ESCs have been programmed to include two features - brake mode off and selection of li-ion battery.

6.4 Ardupilot and Motor Identification

This section will show the experiments that we have carried out in regards to the motor as well as Ardupilot frequency analysis.

6.4.1 APM Frequency

Due to lack of proper documentation, it was necessary to measure the frequency of the signals sent out by the flight controller. To do so, Tektronix DPO 20424b oscilloscope was connected to one of the output pins of the board. Then, using the servo library, a signal was sent out. The interval between signals was found to be $20ms$, therefore, the frequency of the board is $50Hz$, as seen in Figure 6.3.

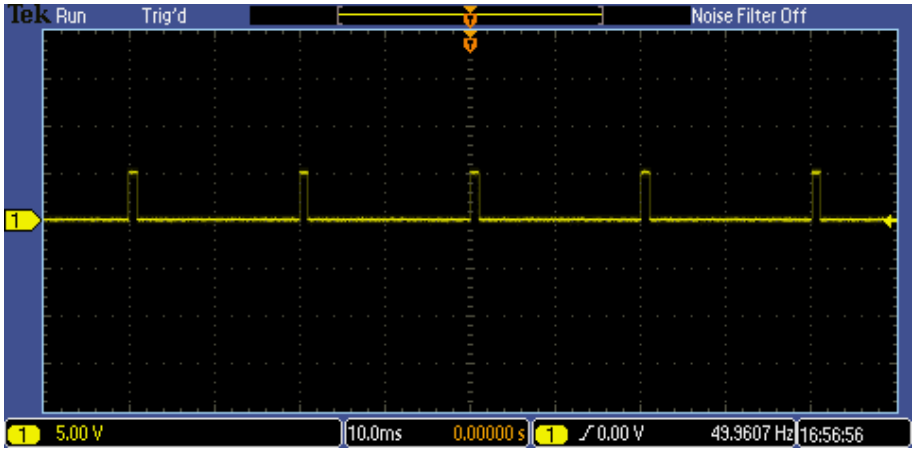

Figure 6.3: Oscilloscope Measuring APM's Frequency.

The second experiment on the board was then made to determine how the flight controller handles the output signals during those $20ms$. First two outputs of the APM were connected to the oscilloscope, both utilizing the Servo library to send signals of length of $2000\mu s$. Results can be seen in Figure 6.4.

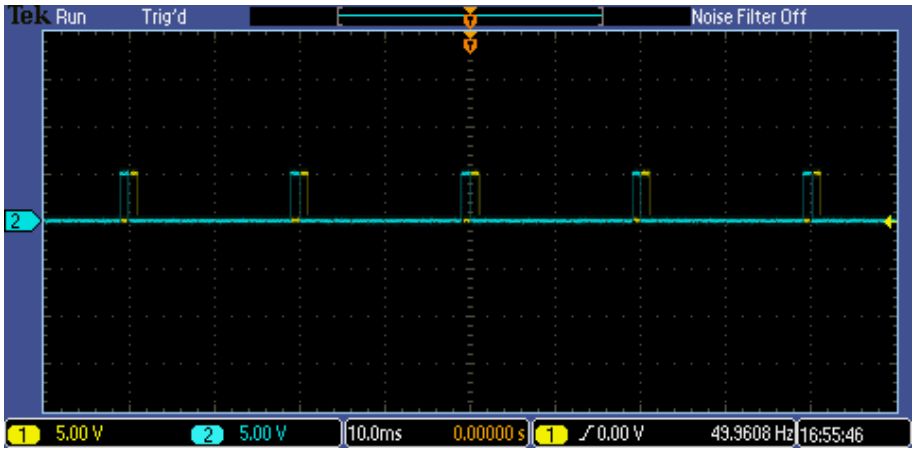

Figure 6.4: Readings of the Two Output Signals.

Then, for further testing purposes, both outputs were given different values in two scenarios, as seen in Figure 6.5.

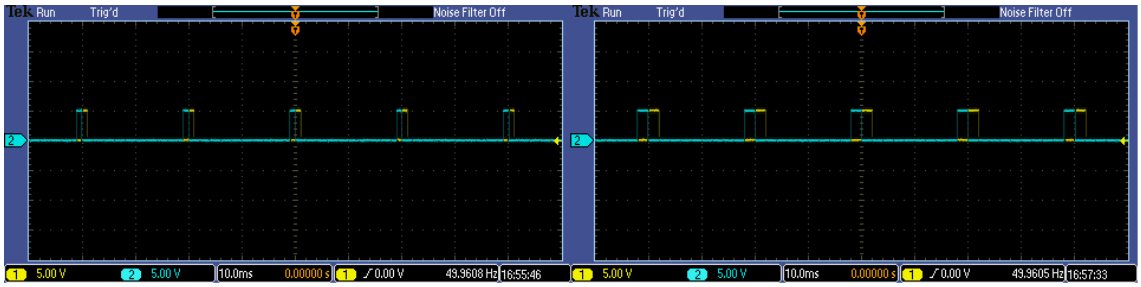

Figure 6.5: Left - Signals Running at $1000\mu s$; Right - $2000\mu s$.

From this, two conclusions can be made:

1. If the first signal is shorter than the second one, the second signal will still follow right after the first signal ends. In other words, the APM leaves no gaps between the outputs.
2. Since the board runs at the frequency of $50Hz$ and has a period of $20ms$, this leaves $\frac{20}{8} = 2.5ms$ maximum length for each output signal. The servo library is hard-capped at $2.4ms$ and thus is well within the limits of the board.

6.4.2 Expected and Real Motor Performance

The motors used in the prototype specify to be rated at K_v of 980. K_v is a constant describing the ratio between RPM and the applied voltage and is expressed as $K_v = \frac{RPM}{V}$. Derived from this, voltage's effect on the RPM can be seen in Figure 6.6.

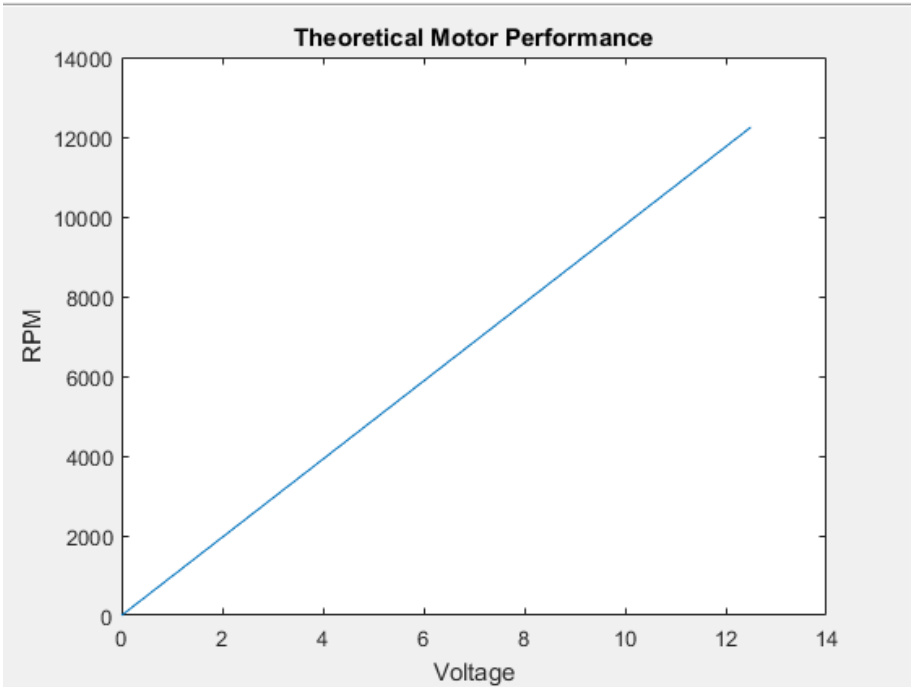

Figure 6.6: Expected Motor Performance.

With a fully charged battery, the RPM is expected to be $980 \times 12.6V = 12348RPM$. In order to confirm this, the actual RPM was measured using SHIMPO DT-205 digital tachometer. A piece of reflective paper was taped to one of the motors so that the tachometer would have something to lock onto, as seen in Figure 6.7.

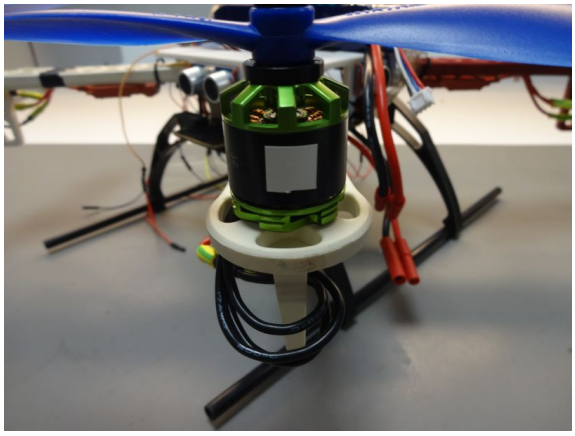

Figure 6.7: Reflective Paper on the Motor.

While it is possible that the mounted propeller could affect the speed of the motor, the measurements had to be done with no propeller on the motor, as it would not only make it difficult to detect the reflective paper, but also be dangerous, since the thrust generated by the fully operational motor is unknown. By sending the maximum signal from the board, the RPM was measured to be 11468, with a small possible error. The measured number does not match the K_v given in the datasheet and there are many possible causes. However, it the result is not too far off the expected value, so the there are no major problems.

6.4.3 Measuring Motor Speed

In order to use the board's output signal as an input in a mathematical model, it is necessary to find an equation to translate it into an angular rate. First, an equation has been found that describes the relationship between motor's RPM, ESC's output signal frequency and number of motor poles [10]:

$$RPM = \frac{120 \times f}{n} \tag{6.15}$$

Here, f is the frequency and n is the number of poles (in the case of this project - 14).

Therefore, it is possible to measure the frequency at various output values, which can then be used to define an equation that uses the board's signal length as an input value and results in an RPM.

The frequency was measured using a oscilloscope and by connecting the probe's ground clip to the ground of the battery and the probe tip to any one of the wires between motor and the ESC. Then, by providing different output signals in some arbitrary range from the flight controller, the frequency was measured on the oscilloscope screen. An on-board filter was used to filter out some noise, especially at lower output signals. The recorded frequency was later converted into RPM using Equa-

tion 6.15, for later comparison. The output signal lengths and the corresponding frequencies converted into RPM can be seen in Table 12.1.

While measuring, it was observed that the frequency would never reach stable readings and therefore, a sizeable error is possible between measurements. Because of that, RPM was manually measured using the tachometer at same output values, as seen in Table 12.2.

While measuring with tachometer, the error appeared to be less significant. The two Tables 12.2 and 12.1 were then plotted to see the difference, which is seen in Figure 6.8.

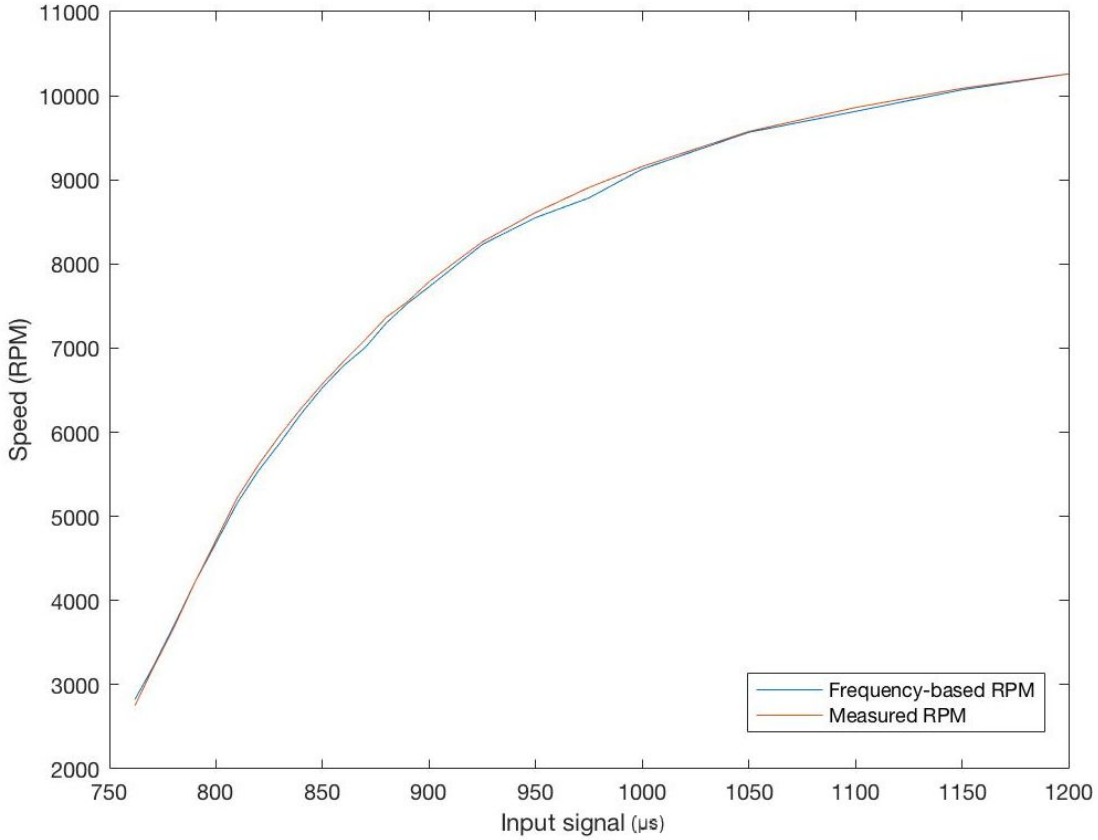

Figure 6.8: Plotted Values of Manually Measured and Frequency-based RPM.

The graphs show some difference between the two measuring methods. The frequency method gives more or less linearly downscaled values. Due to lack of accurate way of determining which readings are correct, it was chosen to go with the tachometer measurements. The process of obtaining measurements with an oscilloscope took more time and was, therefore, deemed inefficient.

In order to make use of these findings, the RPM values have to be converted into rad/s for use as angular rate. The conversion equation is:

$$\omega = \frac{2\pi \times RPM}{60} \tag{6.16}$$

6.4.4 Measuring Speed in Desired Range

Once it was decided how to measure angular velocity, an output range of 1500 to 2400 μs was chosen. The RPM was measured at certain microseconds signal, with increments of 100 across all four motors and converted into rad/s. The measured values have been put into Table 12.3. The plots of the angular velocity (see Figure 6.9) have revealed interesting discoveries - all four motors run at different speeds. Moreover, there seems to be no linear relationship between them throughout the whole range, so it is impossible to express one motor's speed as a function of another motor multiplied by some constant. After failing to find the cause of the difference in the speeds, a conclusion was reached that the motors run at different speeds due to their structure and therefore cannot be changed.

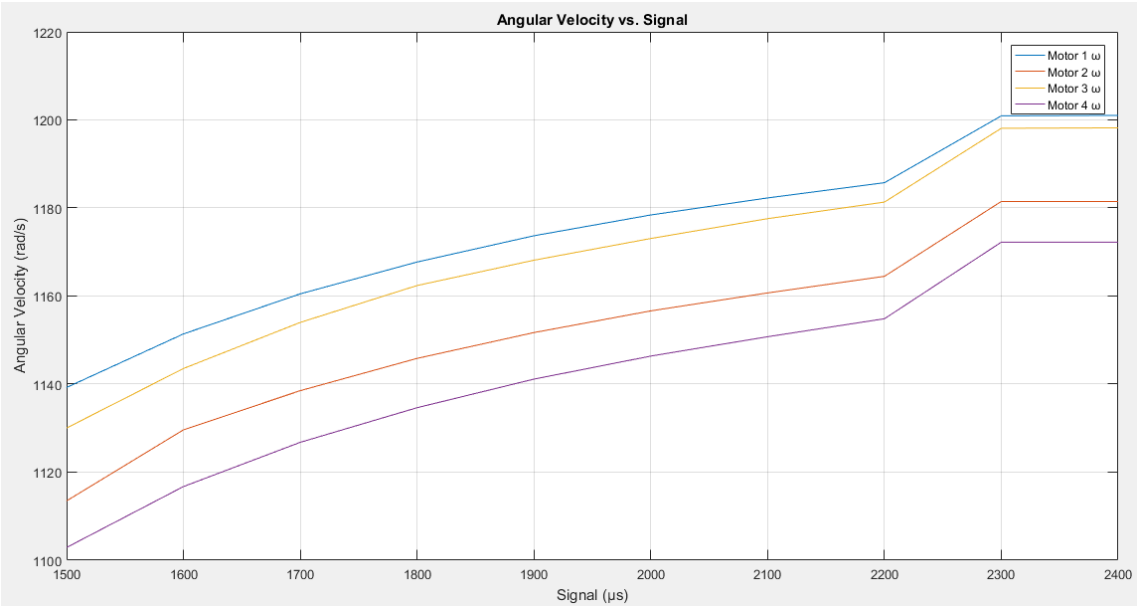

Figure 6.9: Angular Velocity vs. Output Signal for all Motors.

In order to use this data, it was decided to first reduce the range of the measurements - from 1500 to 2000 μs. Then, utilising the $polyfit$ function in Matlab, it is possible to derive linear equations with signal as input and angular velocity as output. The equations have been plotted in Figure 6.10.

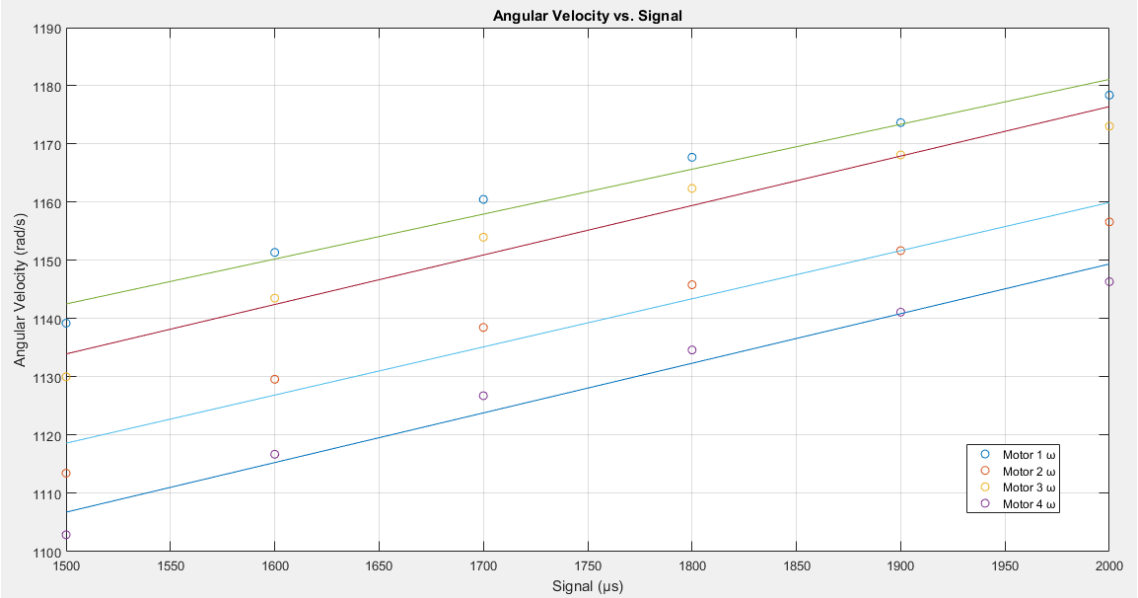

Figure 6.10: Linear Relationship between Signals and Angular Velocities.

The following equations for the linear functions then are:

$$\omega_1 = 0.0771 \times PWM1 + 1026.837 \tag{6.17}$$

$$\omega_2 = 0.0826 \times PWM2 + 994.62 \tag{6.18}$$

$$\omega_3 = 0.0849 \times PWM3 + 1006.507 \tag{6.19}$$

$$\omega_4 = 0.0852 \times PWM4 + 978.91 \tag{6.20}$$

The linear equations do not fit the original graph perfectly, but require less computational power to perform calculations. It is possible to instead acquire 2^{nd} polynomial that would be a more accurate estimation of the original graph, however, that would require more time to compute. This question will be briefly addressed in the discussion chapter.

6.4.5 Operating Range

Due to lack of linearities in the equations expressing the angular velocity, some operating range has to be defined when calculating coefficients, to find reasonable

approximates that would hold true for the most of the prototype's function. Given the drone's main objective - hovering - it is not a far-fetched idea to chose operating point as $TWR = 1$. A newtonmeter was attached to the first motor, with the propeller mounted on it. Then, a signal from the board was sent to the motor, starting at 700 and increasing by 100, up to 2400 μs. The recorded data had significant amount of noise and had to be filtered out in Matlab. Figure 6.11 shows the measured force in newtons, where every significant increase in the force shows increment in the signal.

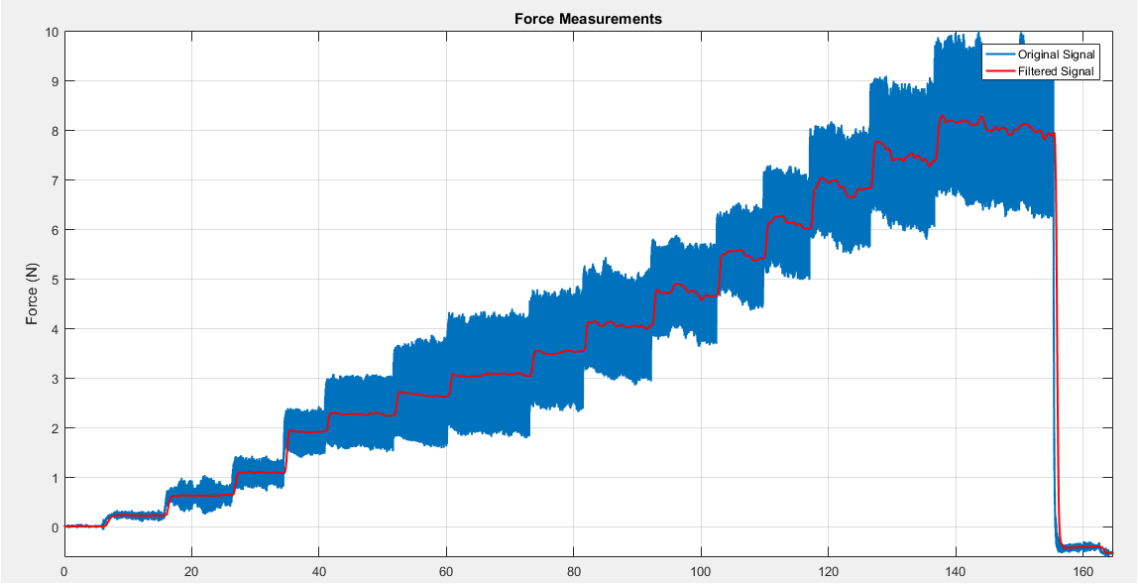

Figure 6.11: Thrust Force Created by First Motor.

The test reveals two things: at maximum speed, the motor can provide solid $8N$ thrust and at higher signals coming in from the board, some overshoot in the motors can be seen, which, in later steps, will have to be taken into consideration.

Given the fact that the drone is drawn to the ground by a $14N$ force, each motor needs to produce about $\frac{14N}{4} = 3.5N$ thrust. Looking at the plot of the test in Figure 6.11, first motor reaches this point at $1499\mu s$ signal coming from the APM, which, based on Figure 6.9, is equal to $1139rad/s$. Then, using same graphs, it is possible to find the input signals corresponding to same angular velocity for the remaining motors. These values will be given in Table 6.1. The coefficients and performance of the motors will be centred around these values.

6.4.6 Measuring Motor Coefficients

First, through trial and error, the deadzones for when the motors are receiving a signal from the flight controller but are not spinning yet were found (see Table 6.1).

The motor time constant τ is described as the time it takes the motor to reach 63% of its expected speed, since the receiving the signal. To do this, PASPORT Rotary

Motion Sensor PS-2120A has been connected to one of the motors. Then, using the SPARKvue software, the angular velocity of the motor for a specific time frame was obtained. A signal was sent from the flight controller to the motor and the angular velocity was plotted against the time, as seen in Figure 6.12.

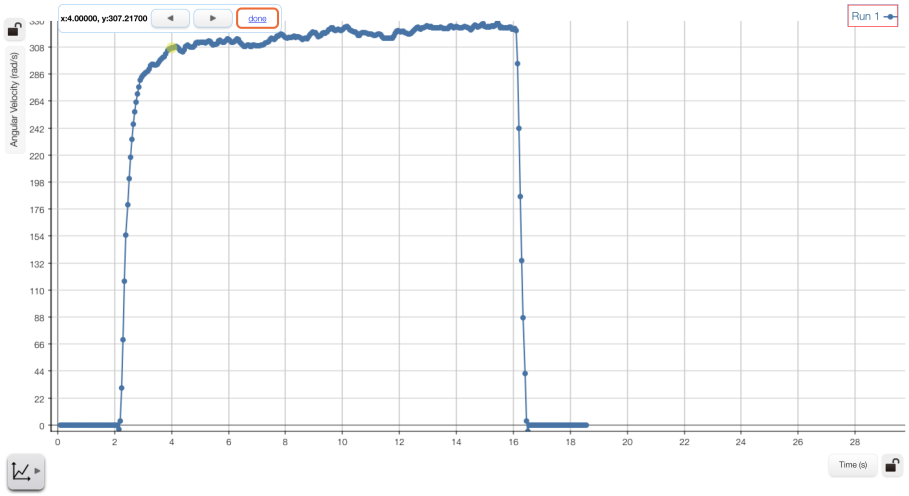

Figure 6.12: Angular Velocity vs. Time.

The readings were a bit distorted due to the placement of rotary sensor - the rubber band connecting the motor and the sensor's wheel was not in a fixed position, slowing down the motor at times. Nevertheless, the important part was the rise of the angular velocity. It was determined that 63% of the peak value $308 rad/s$ was reached in $0.361 s$.

The DC gain k_i is the constant, describing the relationship between angular velocity ω and the PWM input signal. An equation for a specific motor's angular velocity can be given as:

$$\omega = k_i * (PWM - PWM^{Dz}) \tag{6.21}$$

where PWM^{Dz} is the deadzone of the motor. Since the deadzone for all motors is known and so is the angular velocity and the PWM signal at a specific point, it is then possible to use Equation 6.21 to find k_i for the motors. The full list of coefficients is given in Table 6.1.

Motor	k_i	PWM	PWM^{Dz}
1	1.585104	1499	780
2	1.22626	1710	780
3	1.465694	1559	781
4	1.058975	1857	781

Table 6.1: Motor Deadzones, k_i Coefficient and the Lift-off PWM Requirements.

Chapter 7

State space model

7.1 Nonlinear state space representation

The state vector $X = [V, \Omega, P, \Psi]^T$ is obtained using the linear velocity vector, the angular velocity vector, the position vector and the attitude of the quadcopter. Each of these vectors has three components.

The quaternion representation described in Section X is used to simulate the dynamics and kinematics of the quadcopter, while the control is applied with Euler angles. Hence, in order to obtain control techniques based on the model of the system, the model is derived using Euler angles.

The model of the system can be described by defining some functions g and h which fulfils the nonlinear state space representation:

$$\dot{\mathbf{X}} = g(\mathbf{X},\ \mathbf{U}) \tag{7.1}$$

$$\mathbf{Y} = h(\mathbf{X},\mathbf{U}) \tag{7.2}$$

Equation 7.3 is the dynamic equation and Equation 7.4 is the output equation, where $\mathbf{X}$ is the state vector of the system, $\mathbf{U}$ is the input vector of the system and $\mathbf{Y}$ is the ouput vector of the system.

The quadcopter dynamic are described by Equations 4.9-4.14. As it can be observed, the input vector is obtained from the forces and moments applied to the quadcopter. But, we have decided to choose a more intuitive method and go with a model based on the signal sent from the board. Therefore, $U = [PWM_1, PWM_2, PWM_3, PWM_4]$. The relationship between the forces and moments and the angular velocity of the motors was found in Equations 6.7-6.10. Using them and the linearization of angular velocities as functions of PWM enables us to describe our system's input.

The ouput vector of the system has as components the measurements from the accelerometer and gyroscope.

7.2 Linear state space representation

In the previous Section, the state space was described as nonlinear. Therefore, a linearization must be obtained in order to come up with a control method for the system. This linearization is done by approximating the nonlinear state space by a linear expression:

$$\dot{\mathbf{X}} \approx \mathbf{AX}+\mathbf{BU} \tag{7.3}$$

$$\mathbf{Y} \approx \mathbf{CX}+\mathbf{DU} \tag{7.4}$$

It is important to note that this kind of linearization is only accurate in the surroundings of a certain operating point X_0. For hovering conditions, $X_0 = [0, 0, 0, 0, 0, 0, 0, 0, -z, 0, 0, 0]^T$. A linearization of a function can be computed mathematically using Taylor series.

The operating point for the input vector can be obtained by solving Equation ,

$$F_z = mg_0 \tag{7.5}$$

$$F_i = \frac{mg_0}{4} \tag{7.6}$$

$$\omega_i^2 = \frac{mg_0}{4K_T} \tag{7.7}$$

where ω_i^2 is the angular velocity for linearization.

MORE ABOUT MOTORS AND KI HERE

The linearization point in terms of angular velocities can then be determined from the curves representing the behaviour of the model of the motors. The polynomials are then linearized around that point and the static gain is obtained.

Finally, the matrices for the linearized model can be obtained:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.8}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1.585104 & 0 & 0 & 0 \\ 0 & 1.22626 & 0 & 0 \\ 0 & 0 & 1.465694 & 0 \\ 0 & 0 & 0 & 1.058975 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.9}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.10}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.11}$$

Observability and controllability are two notions common to the state space analysis of a system. The controllability translates the possibility of moving the quadrotor through its states. The observability translates the ability to measure the internal states of the system. It is confirmed that both models are fully controllable. REFERENCE THESIS

The poles of the system hold the information of the dynamic response or behaviour of the system. The eigenvalues of matrix A reveal the poles of the chosen continuous system. Using Matlab, it is shown that the twelve poles are located at the origin; the system is unstable. REPHRASE

Chapter 8

System Simulation

This chapter will present the actual implementation of our model in the *Simulink* environment together with its open loop analysis of stability.

8.1 Implementation

8.1.1 Motors

The model of the motors is displayed in Figure 8.1. Each transfer function was obtaineed by putting the coefficients obtained in Section x in place. The deadzone for each motor was also added to the simulation. The output of the system is represented by the angular velocities of the propellers.

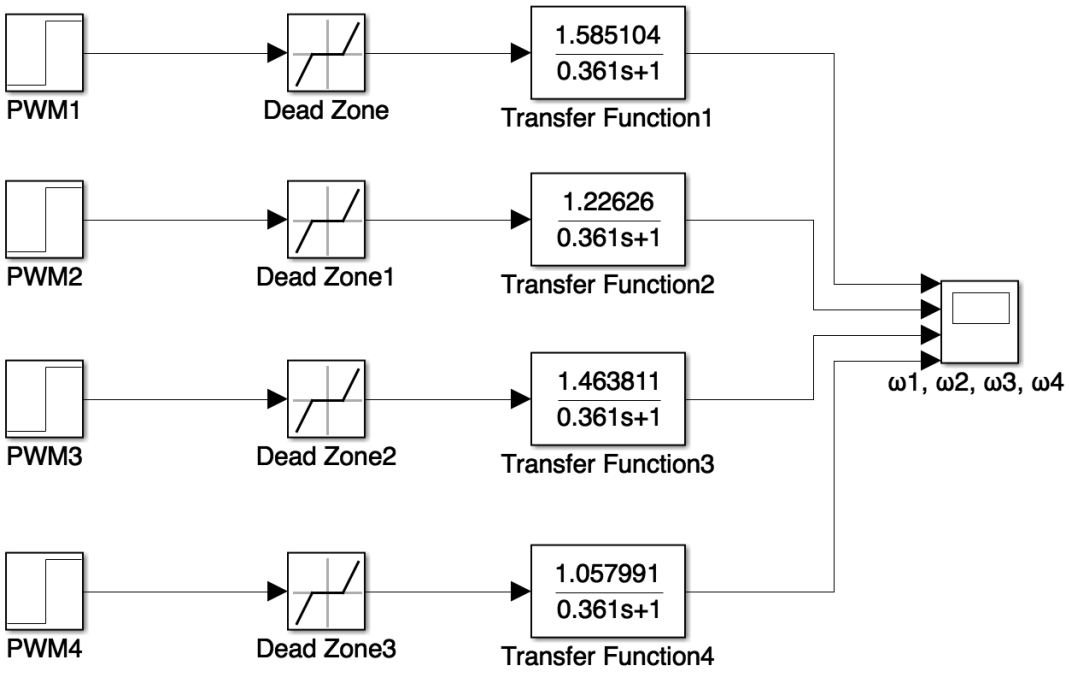

Figure 8.1: Implementation of the motors' model.

By giving the PWM values from Table 6.1, the simulation shows angular velocity of all four motors. It can be seen in Figure 8.2 that the speeds still do not match perfectly. This is because none of the functions are linear, giving only the approximations of the coefficients used in the model and not exact values. As such, some controller will need to be implemented to try and compensate for the differences.

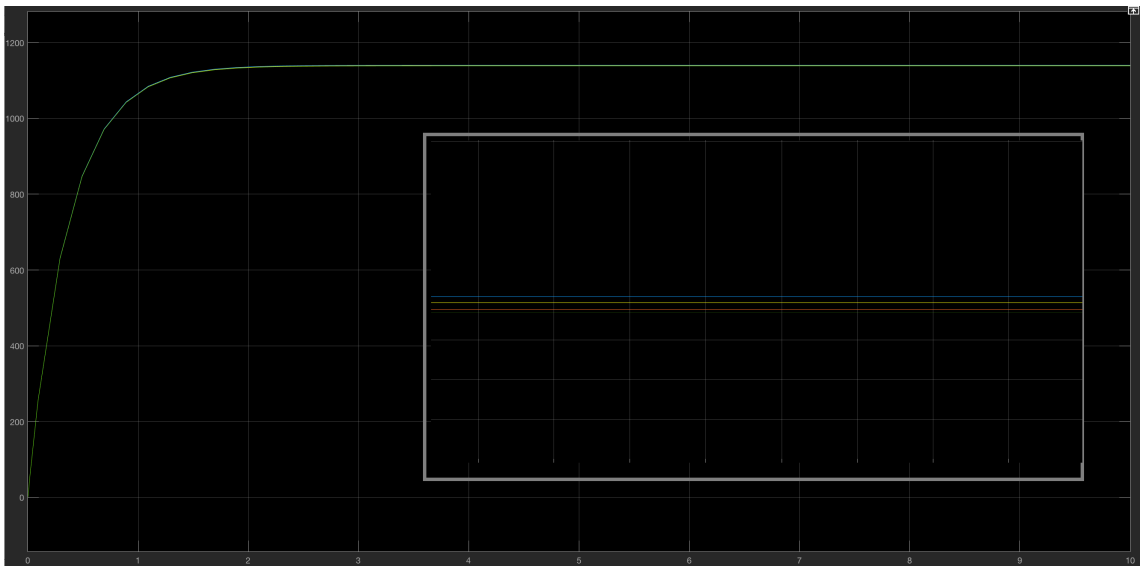

Figure 8.2: The Output of the Model and a Close Up.

8.1.2 Model Simulation

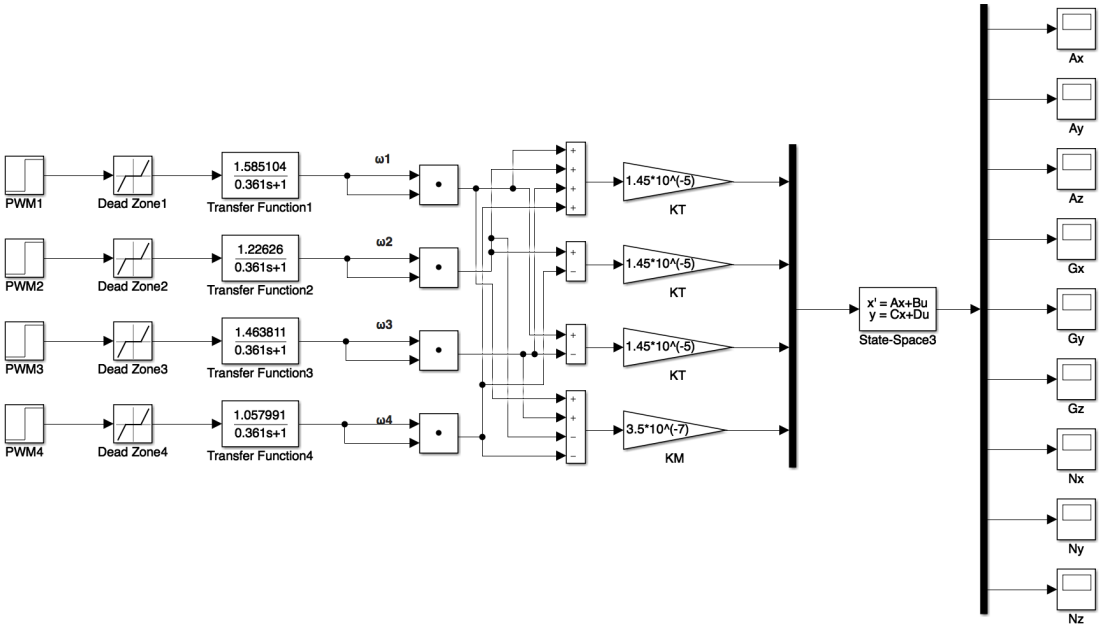

Figure 8.3: Implementation of the full model.

Open loop response:

Ax,Ay,Az,Nx,Nz:

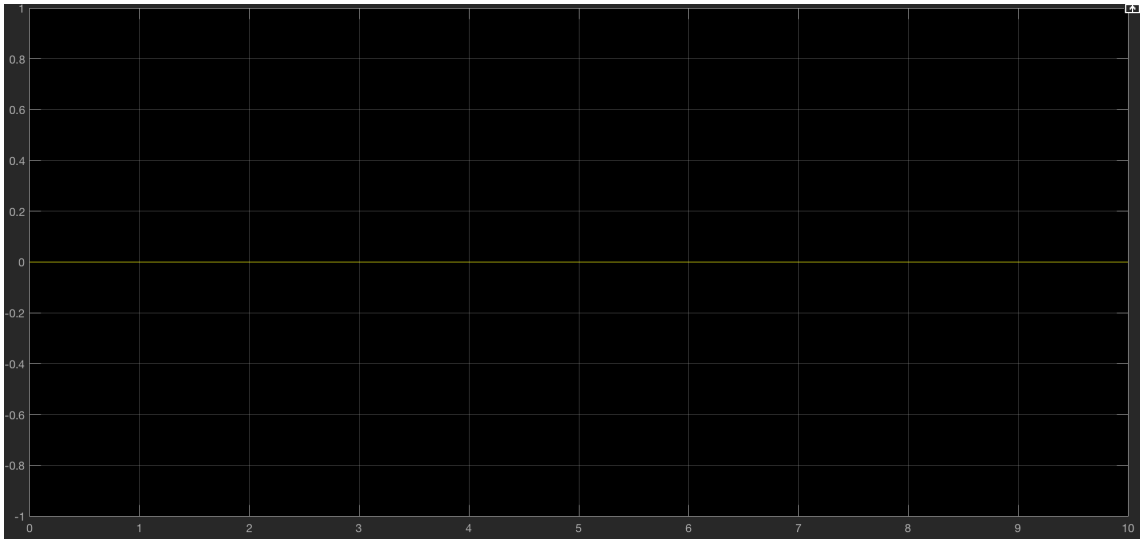

Figure 8.4: something.

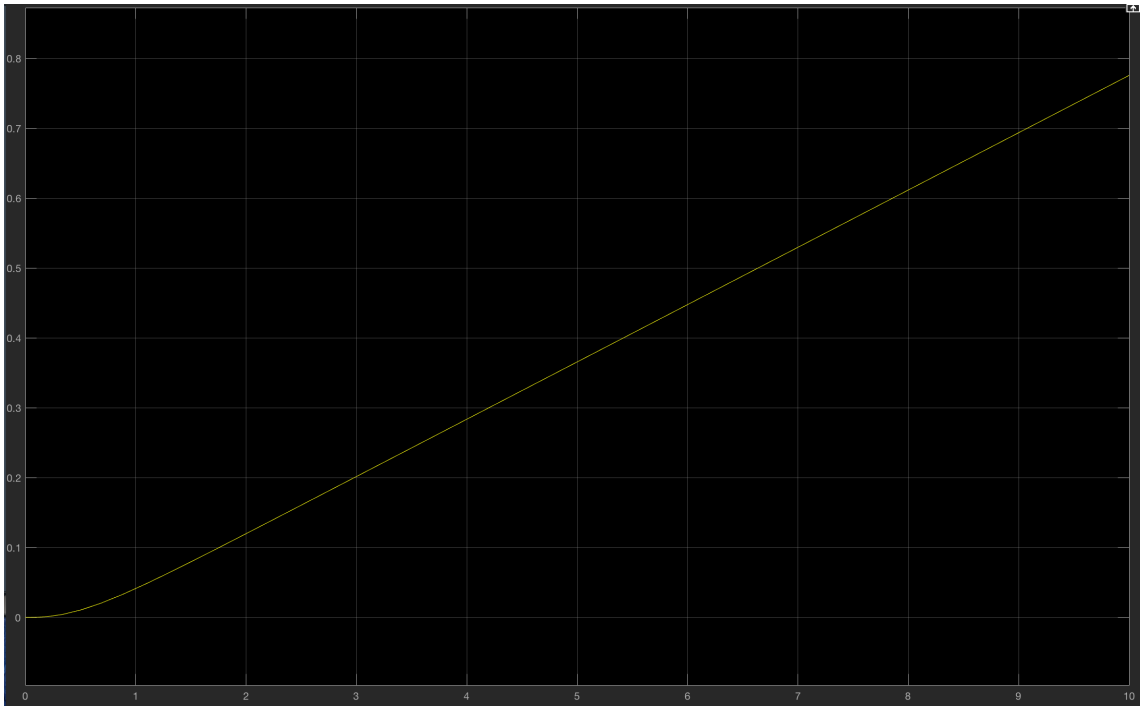

Figure 8.5: Implementation of the full model.

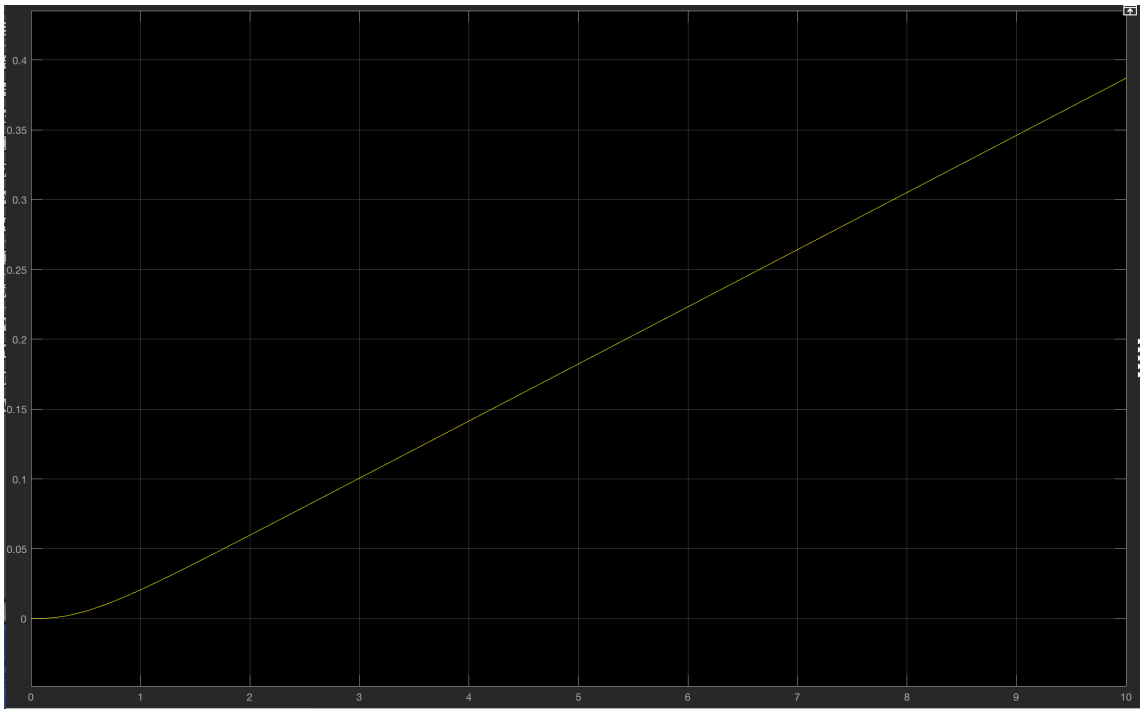

Figure 8.6: Implementation of the full model.

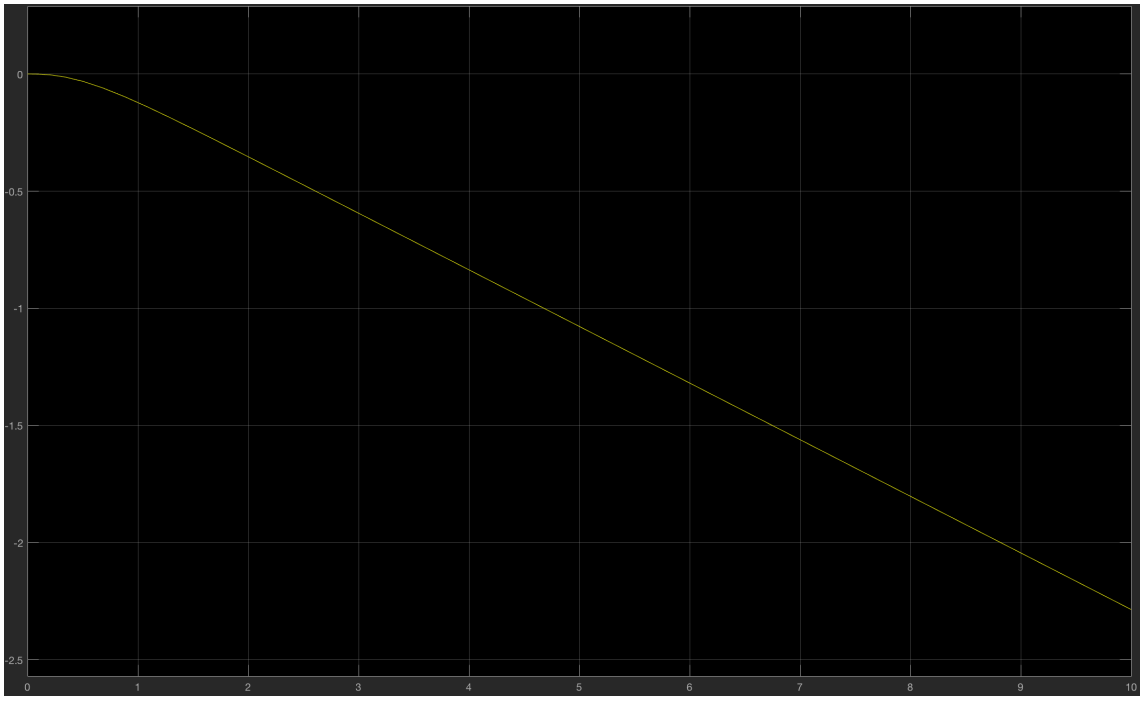

Figure 8.7: Implementation of the full model.

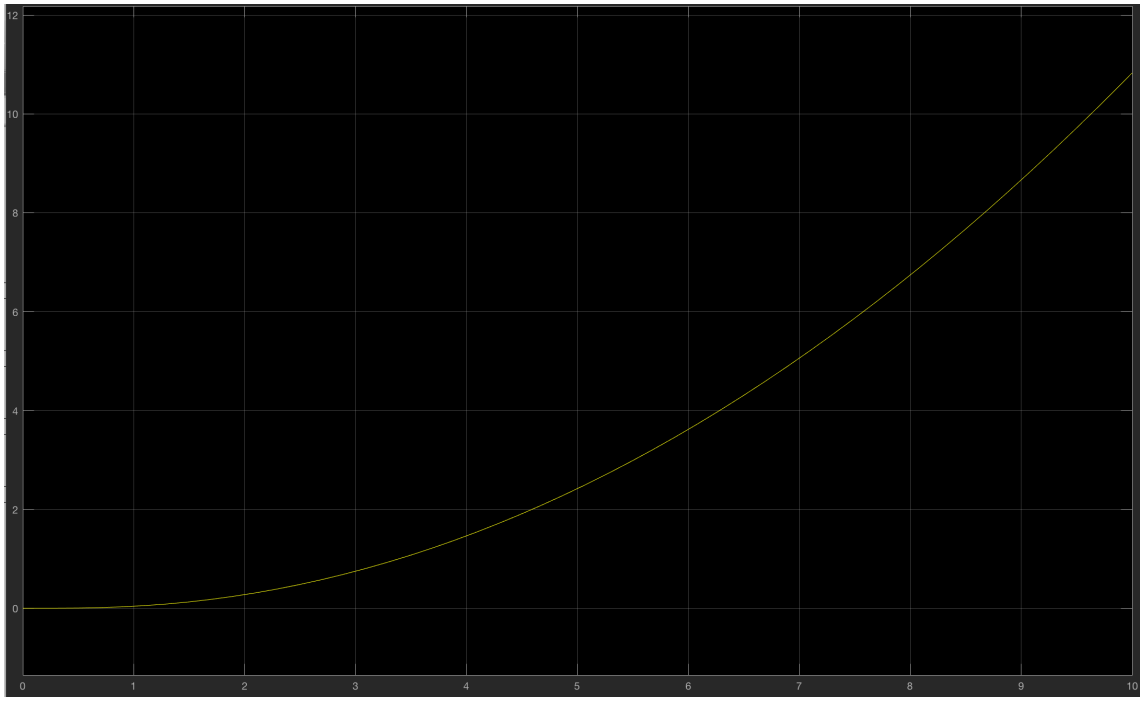

Figure 8.8: Implementation of the full model.

Chapter 9

Control Design

closed loop stuff, explain

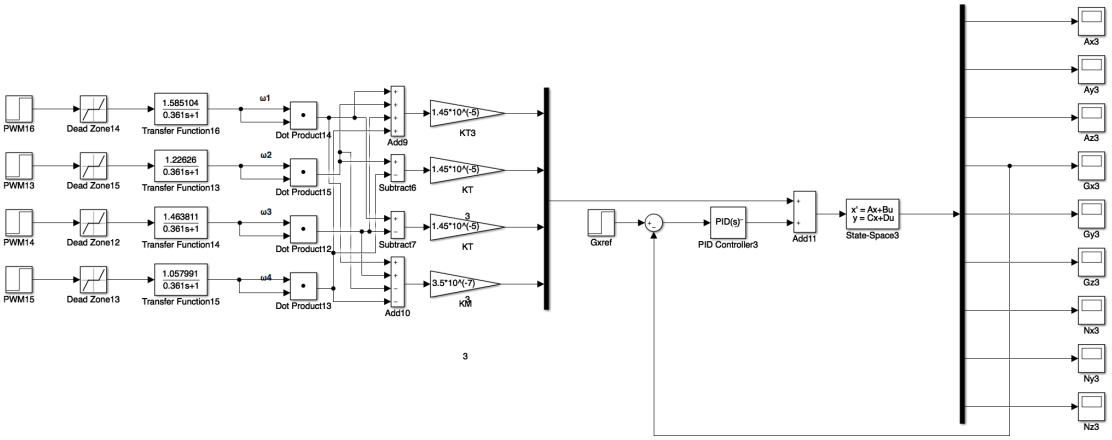

Figure 9.1: smth.

using autotune, pid whatever stuff, maybe merge figures into one.

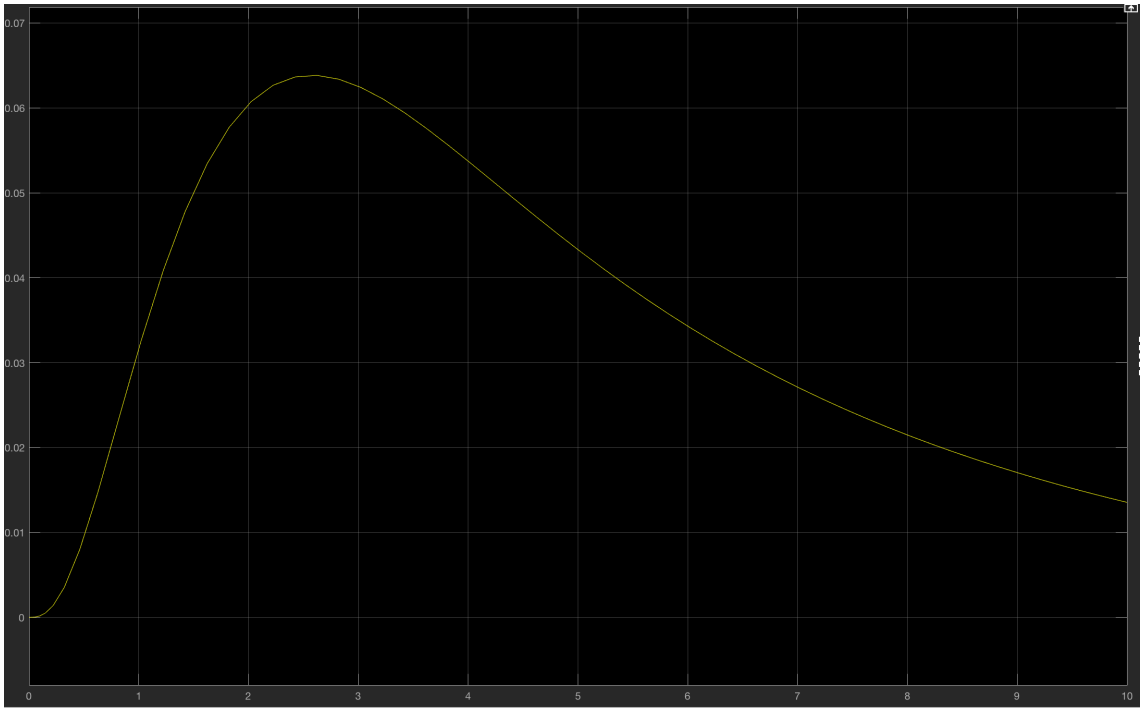

Figure 9.2: smth.

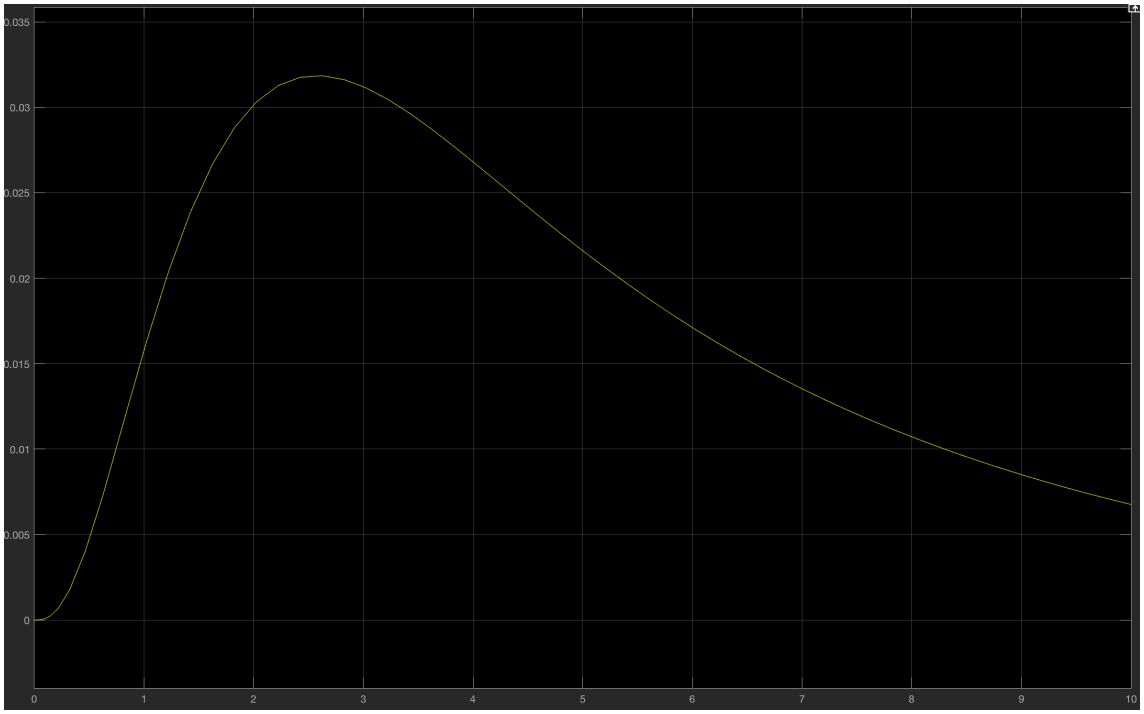

Figure 9.3: smth.

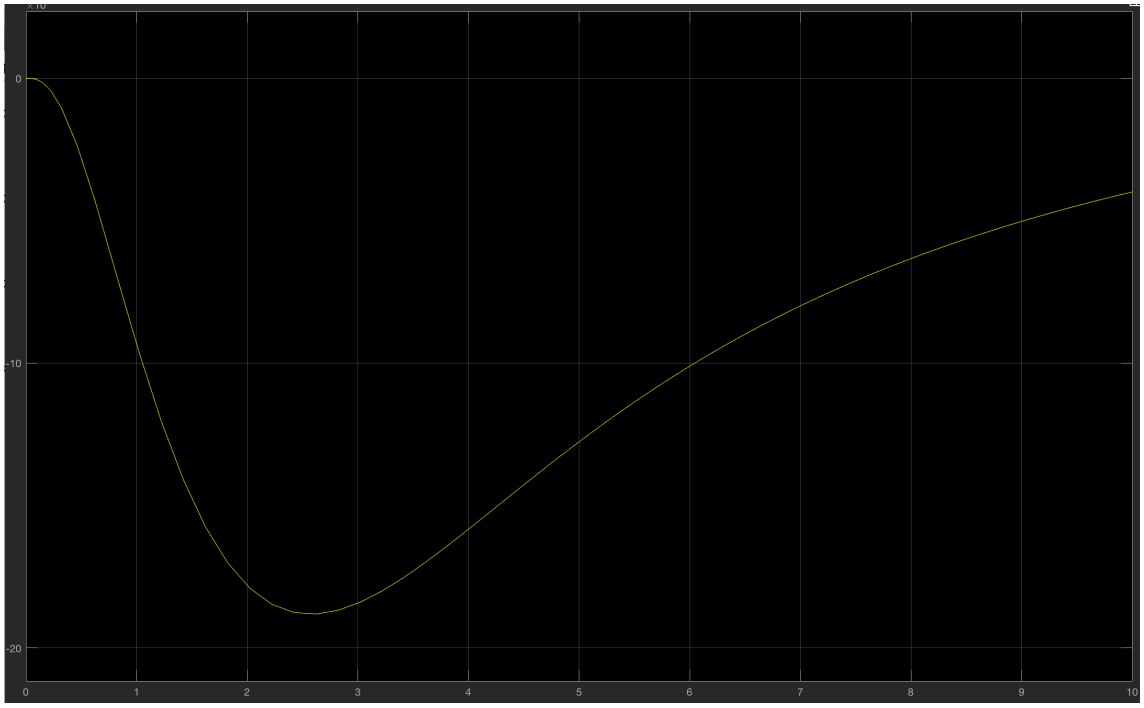

Figure 9.4: smth.

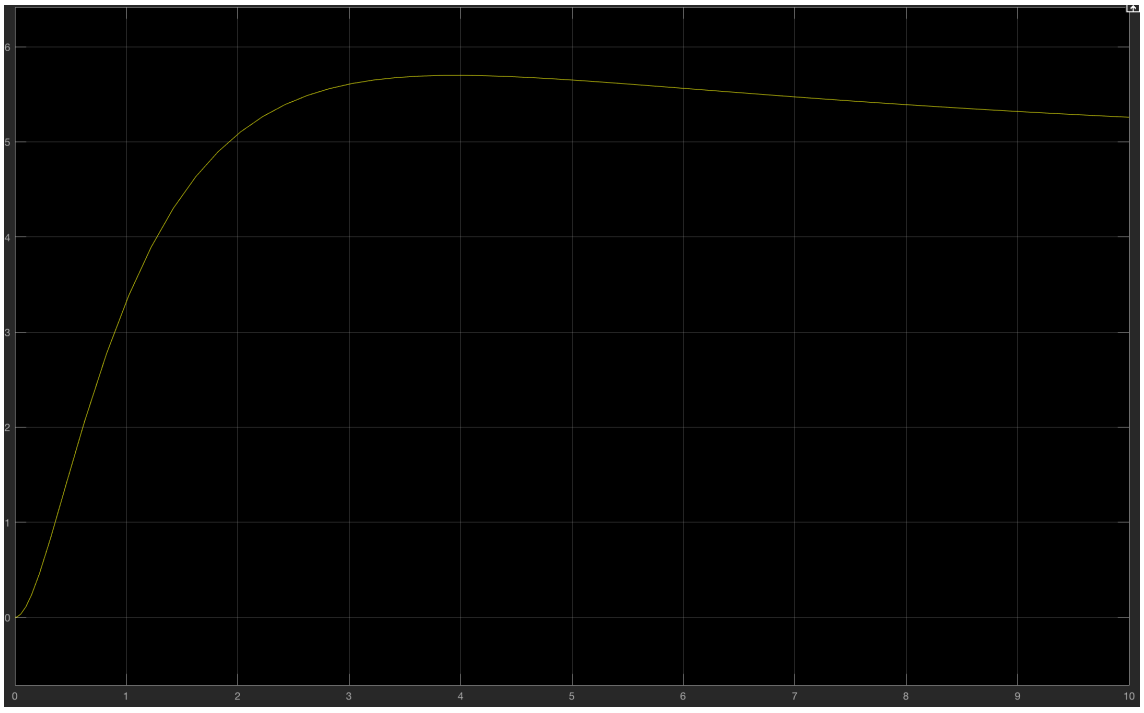

Figure 9.5: smth.

Chapter 10

Discussion

In this chapter, the ups and downs of the report will be covered and the quality of the tests and data acquired will be evaluated. Moreover, the unimplemented features will be discussed, as well as the requirements and methods for possible implementation.

10.1 Sensor Filters

It was previously mentioned, that the filters designed for the accelerometer provide reasonable results. The simulated flight conditions provided reasonable data to design a filter that would suit a fully functioning prototype. Still, some noise remained, despite using both an on-board filter and an additional filter designed in Matlab. This noise, although minimal, could take up some computational power, demanding recalculation of the whole system, despite no real change in stability being present. To help reduce the problem, a simple solution of throwing in more computational power could work - a higher order filter could be designed, provided that the microcontroller allows it. Increasing the order by another 10 produces even better results, as seen in Figure 10.1.

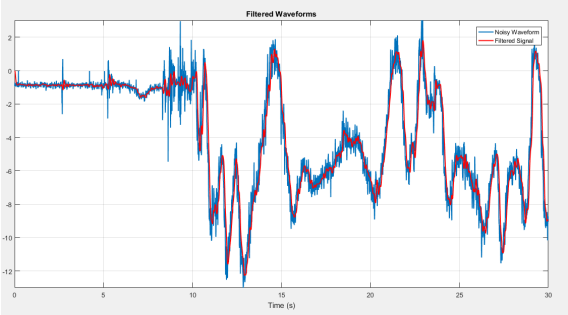

Figure 10.1: A Low-Pass 20th Order FIR Filter for x-axis of Accelerometer.

However, on top of more complex equation, the 20th order filter introduces a slight delay of about $0.1s$. For a hovering quadcopter, a tiny delay like such would carry no

consequences, but a completely autonomous drone carrying out complicated tasks could suffer from such lag. The filter part of the project could be concluded in such way - a more effective filter design is possible, but it has additional costs. The only way to know whether a stronger filter is needed is to first get the prototype completely working.

10.2 Validity of tests

It is necessary to talk about the quality of tests performed on motors and sensors. While the tools used to measure various data such as RPM and force were accurate, lack of proper documentation of the prototype's components lead to unknown expected performances. Moreover, lack of any linearities in the motor performance meant that accurate calculations could only be done around the operating range - anything beyond that point would remain an approximate, with undoubtedly increasing error. In order to avoid such problems, it would be necessary to build a new prototype from scratch, carefully picking out required components. This would, however, require both time and money, as well as additional tools and understanding of desired performance. Overall, a longer working period would be needed to produce a prototype with better performance.

TALK ABOUT THE LACK OF COMPLETE IMPLEMENTATION AND POSSIBLE APPROACHES TO IT.

Chapter 11

Conclusion

Bibliography

[1] Bernardo Sousa Machado Henriques. Estimation and control of a quadrotor attitude. Master's thesis, Universidade Tecnica de Lisboa, 2011.

[2] C BALAS. Modelling and linear control of a quadrotor. Master's thesis, CRANFIELD UNIVERSITY, 2006-2007.

[3] Tradeoffs for locomotion in air and water. https://en.wikipedia.org/wiki/Tradeoffs_for_locomotion_in_air_and_water/.

[4] Toward obstacle avoidance on quadrotors. http://e-collection.library.ethz.ch/eserv/eth:7849/eth-7849-01.pdf.

[5] Ultrasonic ranging module hc - sr04. http://www.micropik.com/PDF/HCSR04.pdf.

[6] Mpu-6000 and mpu-6050 product specification revision 3.4. https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf.

[7] Gyroscopes and accelerometers on a chip. http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/.

[8] Christopher J. Fisher. Using an accelerometer for inclination sensing. http://www.analog.com/media/en/technical-documentation/application-notes/AN-1057.pdf.

[9] Dc motor speed: System modeling. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling.

[10] Vfds how do i calculate rpm for three phase induction motors? http://www.precision-elec.com/faq-vfd-how-do-i-calculate-rpm-for-three-phase-induction-motors/.

Chapter 12

Appendix

12.1 Data Tables

12.1.1 Measuring Method Test Results

$Output, \mu s$	RPM
762	2825
770	3189
780	3689
790	4211
800	4683
810	5161
820	5546
830	5874
840	6223
850	6532
860	6795
870	7008
880	7301
890	7534
900	7730
925	8233
950	8552
975	8786
1000	9129
1050	9566
1100	9814
1150	10071
1200	10260

Table 12.1: Frequency Measured at ESC Output, Converted into RPM.

$Output, \mu s$	RPM
762	2754
770	3170
780	3664
790	4210
800	4715
810	5223
820	5617
830	5965
840	6285
850	6578
860	6846
870	7100
880	7368
890	7555
900	7790
925	8265
950	8614
975	8910
1000	9160
1050	9576
1100	9860
1150	10089
1200	10261

Table 12.2: RPM measured at different values.

12.1.2 Motor Speed Test Results

$Output, \mu s$	$Motor1$	$Motor2$	$Motor3$	$Motor4$
1500	1139.24	1113.48	1130.03	1102.90
1600	1151.39	1129.61	1143.54	1116.73
1700	1160.50	1138.51	1154.01	1126.78
1800	1167.73	1145.84	1162.39	1134.64
1900	1173.69	1151.70	1168.14	1141.13
2000	1178.41	1156.63	1173.07	1146.36
2100	1182.28	1160.71	1177.57	1150.76
2200	1185.74	1164.48	1181.34	1154.84
2300	1200.92	1181.48	1198.10	1172.23
2400	1201.03	1181.48	1198.20	1172.23

Table 12.3: Measured ω for All Four Motors.

12.2 Appendix code

12.2.1 Ultrasonic Sensor Code

```
#include <NewPing.h>

#define TRIGGER_PIN  A0
#define ECHO_PIN     A1
#define MAX_DISTANCE 200
long prevTimer = 0; //initial timer
long currTimer = 100; //initial timer
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.print("Ping: ");
  long currTimer = millis(); //get time
  if(currTimer - prevTimer < 30000){
    Serial.print(sonar.ping_cm());
    Serial.println("cm");
  }
}
```

Code Listing 12.1: Ultrasonic Sensor Code Utilizing NewPing Library.