



**AALBORG UNIVERSITY**

STUDENT REPORT

ED5-3-E16

---

# Hovering control of a quadcopter

---

*Students:*

Alexandra Dorina Török  
Andrius Kulšinskas

*Supervisors:*

Christian Mai  
Zhenyu Yang

December 18, 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

**Abstract:**

XXX

**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:** 43

**Date of Completion:**  
December 18, 2016



# Contents

<b>Preface</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Introduction . . . . .	2
<b>2 General characteristics of a quadcopter</b>	<b>4</b>
2.1 Systems of coordinates . . . . .	4
2.2 Working principle . . . . .	5
2.3 Quadcopter manouvering . . . . .	6
2.4 Assumptions . . . . .	8
<b>3 Prototype</b>	<b>9</b>
3.1 Prototype Hardware . . . . .	9
3.2 Prototype Measurements . . . . .	12
<b>4 Quadcopter Model</b>	<b>13</b>
4.1 Quaternions and Euler angles . . . . .	13
4.2 Quaternions Representation . . . . .	15
4.3 Euler Representation . . . . .	16
<b>5 Sensors</b>	<b>17</b>
5.1 Accelerometer . . . . .	17
5.2 Gyroscope . . . . .	19
5.3 Programming . . . . .	19

<b>6 Actuators</b>	<b>24</b>
6.1 Propeller model . . . . .	24
6.2 Motor model . . . . .	26
6.3 Motor and Ardupilot Identification . . . . .	27
6.3.1 APM Frequency . . . . .	27
6.3.2 Output Operating Range . . . . .	28
6.3.3 Expected and Real Motor Performance . . . . .	29
6.3.4 APM Output and Motor Speed Relationship . . . . .	30
6.4 Electronic Speed Controllers . . . . .	34
6.4.1 Calibration and Programming . . . . .	34
<b>7 State space model</b>	<b>36</b>
7.1 Nonlinear state space representation . . . . .	36
7.2 Linear state space representation . . . . .	37
<b>8 Discussion</b>	<b>40</b>
<b>9 Conclusion</b>	<b>41</b>
<b>Bibliography</b>	<b>42</b>
<b>10 Appendix</b>	<b>43</b>
10.1 Appendix code . . . . .	43

# 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 18, 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 semester, a control system is an instrument which has the role of adapting the behaviour of a system according to a desired state, also known as steady-state 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.

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 multicopters. Multicopter can be defined as a rotorcraft with more than two motors. Based on this, the four most common types are tricopter, quadcopter, hexacopter and octocopter, each having 3, 4, 6 and 8 motors respectively. Each type of multicopter 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 - and to also act according to the user's input - maneuvering. 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. A safety feature - obstacle avoidance - will also be implemented.

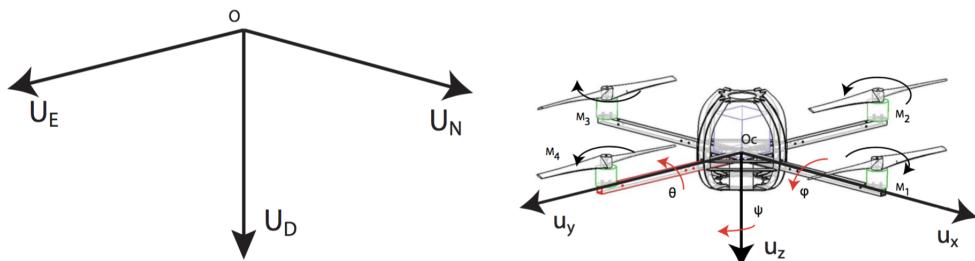
# Chapter 2

## 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 quadcopter 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 Boxy-fixed frame. It has the center in  $O_C$ , which is the center of mass of the quadcopter. 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 market 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 -  $\phi$ , pitch -  $\theta$  and yaw -  $\psi$  angles.

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

$$P^I = [X, Y, Z]^T \quad (2.1)$$

The quadcopter's velocity is express as:

$$V^B = [U, V, W]^T \quad (2.2)$$

The quadcopter's rotation can be described using the 3 Euler angles  $\varphi$ ,  $\theta$  and  $\psi$  as:

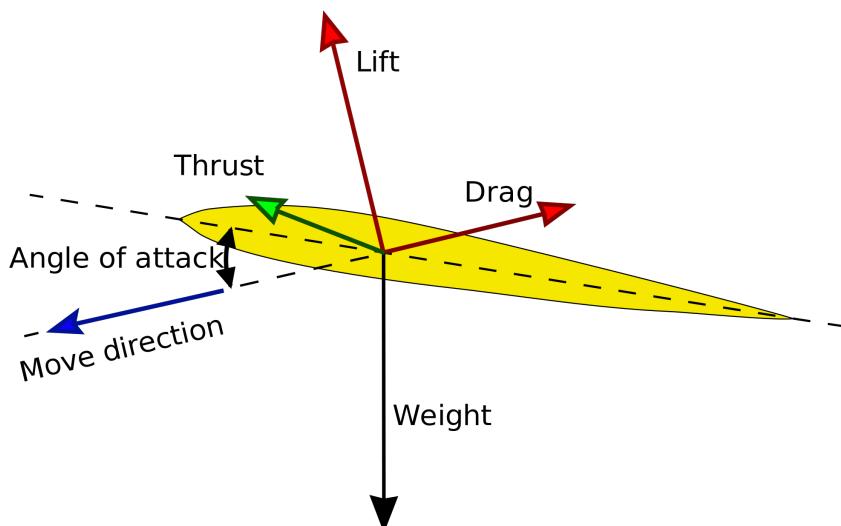
$$\Psi = [\varphi, \theta, \psi]^T \quad (2.3)$$

The quadcopter'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:

$$\Omega^B = [P, Q, R]^T \quad (2.4)$$

## 2.2 Working principle

To have an understanding of how a quadcopter 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 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 due to lack of wind.

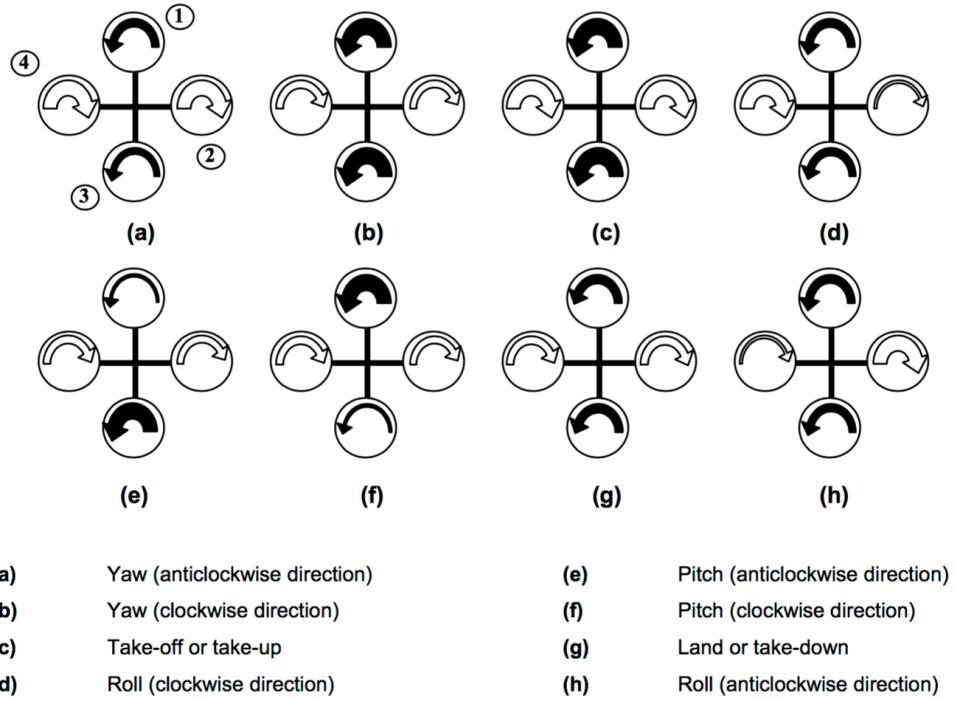
If the motors are powered off, the only force affected is the drop and therefore the quadcopter stays on the ground. In order to lift it up, we need to understand the relationship between quadcopter and thrust to weight ratio - or TWR for short. This ratio can be determined by equation  $F_t/F_g$  and 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 ratio 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 manouvering

Quadcopters are made with four identical motors, each of them having an attached propeller. As it can be seen in Figure , 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 manouvers shown in Figure .

**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:

$$\omega = [\omega_1, \omega_2, \omega_3, \omega_4]^T \quad (2.5)$$

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

- **Rolling motion** is represented by the rotation of the quadcopter about the  $u_x$  axis and it can be obtained when  $\omega_2$  and  $\omega_4$  are manipulated. In order to obtain a positive rolling,  $\omega_4$  is decreased while  $\omega_2$  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  $\omega_1$  and  $\omega_3$  are manipulated. In order to obtain a positive pitch,  $\omega_3$  is decreased while  $\omega_1$  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 quadcopter 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 increasing 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. In addition, a diagram of the wiring of the components can be found in appendix X.

### 3.1 Prototype Hardware

#### 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

## 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 plastic 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

## 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

## 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

## 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.

## Battery

To power up our quadcopter, we will use a TURNIGY nano-tech Lipoly battery, which can be seen in Figure 3.5. Higher voltage under load, straighter discharge curves and excellent performance are the factors that make it suitable for our project.



Figure 3.5: Turnigy nano-tech 6000mah 3S 25 50C Lipo Pack

## 3.2 Prototype Measurements

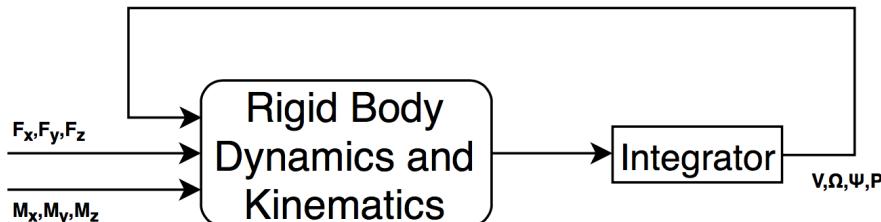
mass, length, height, moment of inertia

# 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:

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

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

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

Merging the three rotations as:

$$S = R(\phi)^T R(\theta)^T R(\psi)^T \quad (4.4)$$

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

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

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

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

As a result, one degree of freedom in the three dimensional space is lost. In addition, a change in either  $\phi$  or  $\psi$  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:

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

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:

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

## 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:

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

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

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -P & -Q & -R \\ P & 0 & R & -Q \\ Q & -R & 0 & P \\ R & Q & -P & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (4.10)$$

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:

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

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

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

Therefore:

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

### 4.3 Euler Representation

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

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

Solving for Euler angles rates finally gives:

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

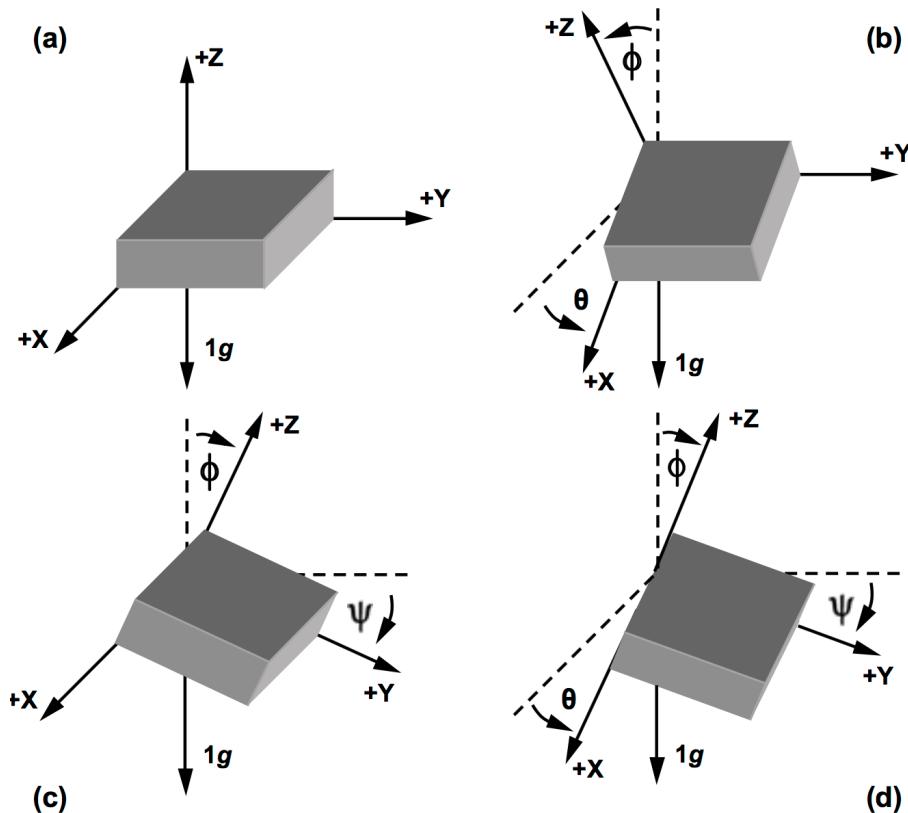
# Chapter 5

## 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 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 Accelerometer

According to the MPU-6000 datasheet (ADD REFERENCE), 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  $\theta$  is the angle between the horizon and the x-axis of the accelerometer,  $\psi$  as the angle between the horizon and the y-axis of the accelerometer, and  $\phi$  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. ADD REF SAME AS PICTURE



**Figure 5.1:** Inclination angles //<http://www.analog.com/media/en/technical-documentation/application-notes/AN-1057.pdf>

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

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

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

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

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 manoeuvring 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. ADD REFERENCE <http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>

## 5.2 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  $\omega$  around the X, Y and Z axes at measured intervals ( $\Delta t$ ). 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 (ADD REFERENCE) 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.

## 5.3 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:

```

1  Wire.begin() ;
2  //As per APM standard code, stop the barometer from holding the SPI
3  //bus
4  pinMode(40, OUTPUT);
5  digitalWrite(40, HIGH);
6  SPI.begin();
7  SPI.setClockDivider(SPI_CLOCK_DIV16);
8  SPI.setBitOrder(MSBFIRST);
9  SPI.setDataMode(SPI_MODE0);
10 delay(100);
11 pinMode(ChipSelPin1, OUTPUT);

```

```
11 ConfigureMPU6000(); // configure chip
```

**Code Listing 5.1:** Setup

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

```
1 void ConfigureMPU6000()
{
3 // DEVICE_RESET @ PWR_MGMT_1, reset device
5 SPIwrite(0x6B,0x80,ChipSelPin1);
delay(150);

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

11 // I2C_IF_DIS @ USER_CTRL, disable I2C interface
13 SPIwrite(0x6A,0x10,ChipSelPin1);
delay(150);

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

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

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

27 // AFS_SEL @ ACCEL_CONFIG, accel scale at 2g (1g=8192)
29 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.

```
1 int AcceX(int ChipSelPin) {
2     uint8_t AcceX_H=SPIread(0x3B,ChipSelPin);
3     uint8_t AcceX_L=SPIread(0x3C,ChipSelPin);
4     int16_t AcceX=AcceX_H<<8|AcceX_L;
5     return(AcceX);
6 }
8 int AcceY(int ChipSelPin) {
```

```

10  uint8_t AcceY_H=SPIread(0x3D,ChipSelPin);
11  uint8_t AcceY_L=SPIread(0x3E,ChipSelPin);
12  int16_t AcceY=AcceY_H<<8|AcceY_L;
13  return(AcceY);
14 }

16  int AcceZ(int ChipSelPin) {
17  uint8_t AcceZ_H=SPIread(0x3F,ChipSelPin);
18  uint8_t AcceZ_L=SPIread(0x40,ChipSelPin);
19  int16_t AcceZ=AcceZ_H<<8|AcceZ_L;
20  return(AcceZ);
21 }

24  int GyroX(int ChipSelPin) {
25  uint8_t GyroX_H=SPIread(0x43,ChipSelPin);
26  uint8_t GyroX_L=SPIread(0x44,ChipSelPin);
27  int16_t GyroX=GyroX_H<<8|GyroX_L;
28  return(GyroX);
29 }

32  int GyroY(int ChipSelPin) {
33  uint8_t GyroY_H=SPIread(0x45,ChipSelPin);
34  uint8_t GyroY_L=SPIread(0x46,ChipSelPin);
35  int16_t GyroY=GyroY_H<<8|GyroY_L;
36  return(GyroY);
37 }

40  int GyroZ(int ChipSelPin) {
41  uint8_t GyroZ_H=SPIread(0x47,ChipSelPin);
42  uint8_t GyroZ_L=SPIread(0x48,ChipSelPin);
43  int16_t GyroZ=GyroZ_H<<8|GyroZ_L;
44  return(GyroZ);
45 }

46 }
```

**Code Listing 5.3:** 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:

```

float AcceDeg(int ChipSelPin,int AxisSelect) {
2  float Ax=ToG(AcceX(ChipSelPin));
3  float Ay=ToG(AcceY(ChipSelPin));
4  float Az=ToG(AcceZ(ChipSelPin));
5  float ADegX=((atan(Ax/(sqrt((Ay*Ay)+(Az*Az)))))/PI)*180;
6  float ADegY=((atan(Ay/(sqrt((Ax*Ax)+(Az*Az)))))/PI)*180;
7  float ADegZ=((atan((sqrt((Ax*Ax)+(Ay*Ay)))/Az))/PI)*180;
8  switch (AxisSelect)
{
```

```

10   case 0:
11     return ADegX;
12   break;
13   case 1:
14     return ADegY;
15   break;
16   case 2:
17     return ADegZ;
18   break;
19 }
20 }
```

**Code Listing 5.4:** Function to obtain angles based on the raw values of the accelerometer

```

1 float GyroDeg(int ChipSelPin, int AxisSelect) {
2   time_old=time;
3   time=millis();
4   float dt=time-time_old;
5   if (dt>=1000)
6   {
7     dt=0;
8   }
9   float Gx=ToD(GyroX(ChipSelPin));
10  float Gy=1-ToD(GyroY(ChipSelPin));
11  float Gz=ToD(GyroZ(ChipSelPin));
12  angleX+=Gx*(dt/1000);
13  angleY+=Gy*(dt/1000);
14  angleZ+=Gz*(dt/1000);
15  switch (AxisSelect)
16  {
17    case 0:
18      return angleX;
19      break;
20    case 1:
21      return angleY;
22      break;
23    case 2:
24      return angleZ;
25      break;
26  }
27 }
```

**Code Listing 5.5:** 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.

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 5.2: Angles obtained from Accelerometer and Gyroscope

//insert prinscreen for tilting

# 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.

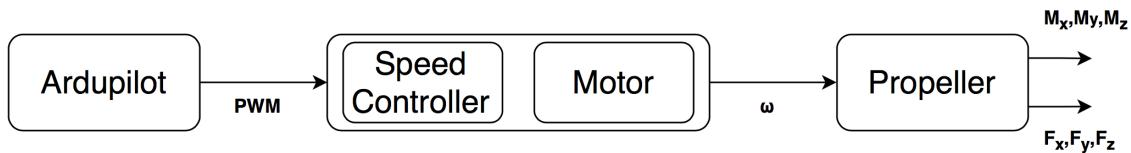


Figure 6.1: Actuators Block Diagram

Each motor produces an angular velocity  $\omega$ , 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:

$$F_i = K_T \omega_i^2 \quad (6.1)$$

$$M_i = K_M \omega_i^2 \quad (6.2)$$

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

$$K_T = c_T \frac{\rho D^4}{4\Pi^2} \quad (6.3)$$

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

with  $D$  being the diameter of the propeller,  $\rho$  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:

$$K_T \approx 1.45 \times 10^{-5} \quad (6.5)$$

$$K_M \approx 3.5 \times 10^{-7} \quad (6.6)$$

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:

$$F_i = K_T \omega_i^2 \quad (6.7)$$

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

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

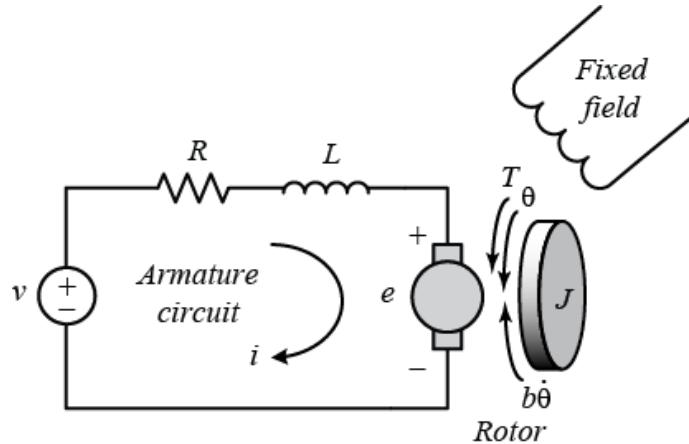
$$M_z = (\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2) K_M = (F_1 + F_3 - F_2 - F_4) K \quad (6.10)$$

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  $\omega$ . 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  
[5]

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

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

$$K_e I = J \ddot{\theta} + b \dot{\theta} \quad (6.12)$$

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

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

where  $\omega$  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]

$$\frac{\omega}{PWM} = \frac{k_i}{\tau s + 1} \quad (6.14)$$

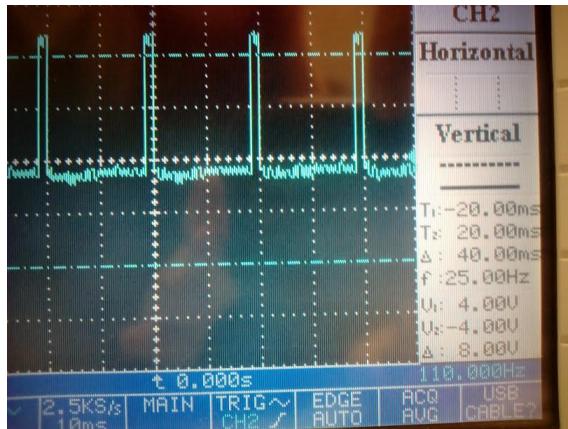
where  $\tau$  is the time constant of the system and  $k_i$  is the DC gain.

## 6.3 Motor and Ardupilot Identification

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

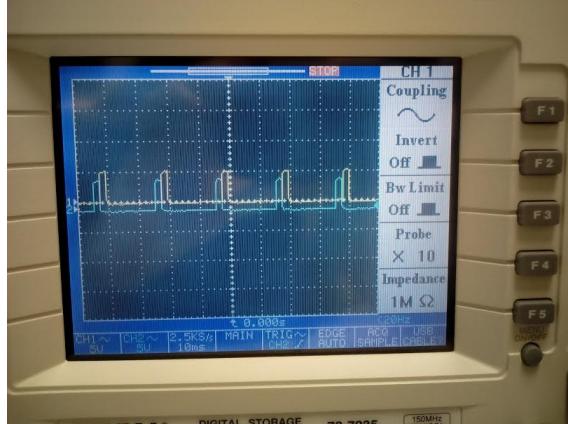
### 6.3.1 APM Frequency

Due to lack of proper documentation, it was necessary to do measure the frequency of the signals sent out by the flight controller. To do so, an 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.



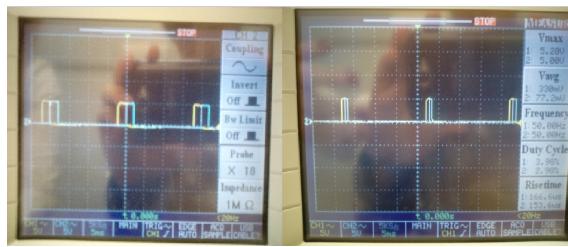
**Figure 6.3:** Oscilloscope measuring APM's frequency

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 2000μs. 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  $2000\mu s$ ; Right -  $700\mu s$ .

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.

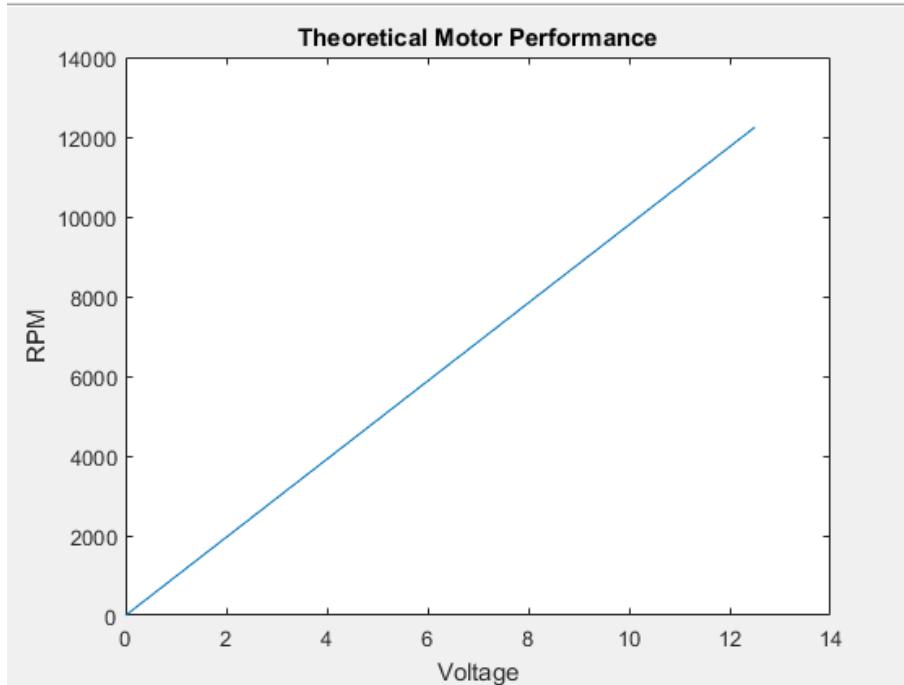
### 6.3.2 Output Operating Range

Since there is no direct translation between the output signal from the board and the speed of the motor and the ESC allows more current than the motor can recognise, it is necessary to find the operating range for the board output that has the most noticeable effect on the RPM of the motor. Through trial and error and by measuring the RPM using SHIMPO DT-205 digital tachometer. The lowest operating point was found to be  $762\mu s$ , at which the motor finally begins to spin. The highest point was defined at  $1200\mu s$ . Higher values were found to have very small effect on the RPM. Therefore, despite the ESC recognising the values between 700 and 2000 $\mu s$ ,

the most noticeable effect will be between 762 and  $1200\mu s$  and will be primarily used in the project.

### 6.3.3 Expected and Real Motor Performance

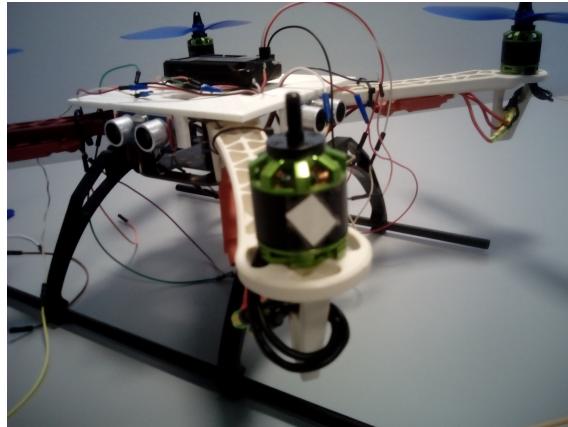
The motors used in the prototype specify to be rated at  $K_v$  of 980.  $K_v$  is a constant describing the ratio between RPM and the applied voltage and is expressed as  $K_v = \frac{RPM}{V}$ . Derived from this, voltage's effect on the RPM can be seen in figure 6.6.



**Figure 6.6:** Expected motor performance

With a fully charged battery, the RPM is expected to be  $980 \times 11.1V = 10878RPM$ .

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

By sending the maximum signal from the board, the RPM was measured to be 10812, with a small possible error. In conclusion, the motors'  $K_v$  coefficient found in the datasheet is in fact correct.

### 6.3.4 APM Output and Motor Speed Relationship

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 [6]:

$$RPM = \frac{120 \times f}{n} \quad (6.15)$$

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 an (NAME) 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 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 equation 6.15, for later comparison. The output signal lengths and the corresponding frequencies converted into RPM can be seen in table 6.1.

<i>Output, <math>\mu s</math></i>	<i>RPM</i>
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 6.1:** Frequency measured at ESC output, converted into RPM

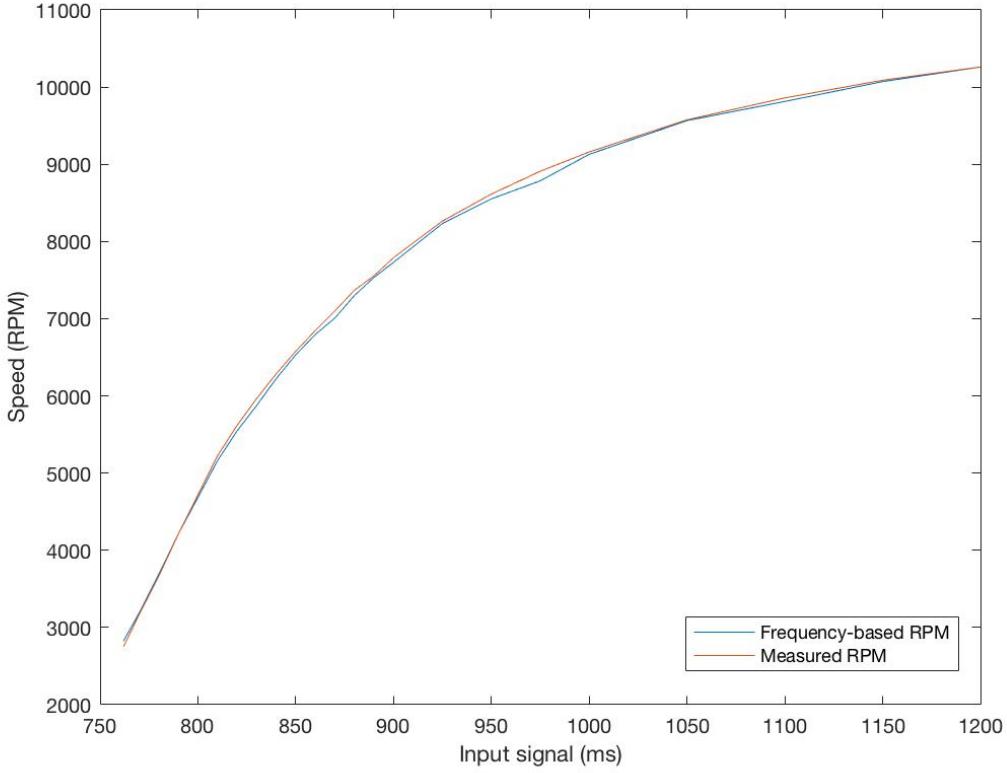
While measuring, it was observed that the frequency would never reach a steady-state and therefore, a sizeable error is possible between measurements. Because of that, RPM was manually measured using the tachometer at same output values, as seen in table 6.2.

<i>Output, <math>\mu s</math></i>	<i>RPM</i>
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 6.2:** RPM measured at different values

While measuring with tachometer, the error appeared to be less significant.

The two tables 6.2 and 6.1 were then plotted to see the difference, which is seen in figure 6.8.



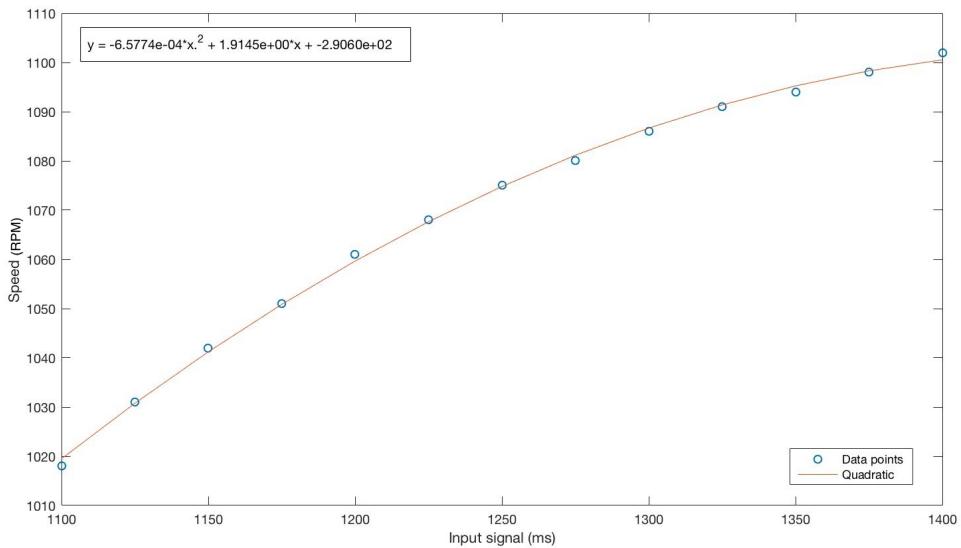
**Figure 6.8:** Plotted values of manually measured and frequency-based RPM

From the plotted graphs, it can be seen that the difference is very much noticeable, especially at certain points. Due to the fact that measurements with tachometer provided smaller errors than measurements with an oscilloscope, it was decided to use the values measured with the tachometer for later calculations.

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:

$$\omega = \frac{2\pi \times RPM}{60} \quad (6.16)$$

The values measured with tachometer were then converted into  $rad/s$  and plotted in Matlab. Then, by utilising the *polyfit* function, a 2<sup>nd</sup> order polynomial equation fitting the original graph was obtained. The equation seen in Figure can then be used as an approximation for the conversion of the sent signal in  $\mu s$  into angular rate.



**Figure 6.9:** Converted and plotted RPM values and the trendline

## 6.4 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.

### 6.4.1 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 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.

<sup>1</sup> `#define MAX_SIGNAL 2000 //define max and min signal length`

```
#define MIN_SIGNAL 700
3 #define MOTOR_PIN1 12 //pin 1 on the board
  Servo motor1;
5 void setup() {
  Serial.begin(9600); //begin serial communication at 9600 rate
7   Serial.println("Program begin...");
  motor1.attach(MOTOR_PIN1); //attach the pin to the servo variable
9   Serial.println("Now writing maximum output.");
  Serial.println("Turn on power source, then wait 2 seconds and press
    any key.");
11  motor1.writeMicroseconds(MAX_SIGNAL); //this signal should only be
    sent when calibrating for the first time
13  while (!Serial.available()); //wait for an input, so you have time to
    plug in the ESC
15  Serial.read();
  Serial.println("Sending minimum output");
17  motor1.writeMicroseconds(MIN_SIGNAL); //send the min signal, which is
    the only needed signal when already calibrated
19  while (!Serial.available()); //wait for the beeps to indicate
    calibration is done
21  Serial.read();
  Serial.println("Calibrated");
23 }
```

**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.

# 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:

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

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

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 -. 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 -. 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:

$$\dot{\mathbf{X}} \approx \mathbf{AX} + \mathbf{BU} \quad (7.3)$$

$$\mathbf{Y} \approx \mathbf{CX} + \mathbf{DU} \quad (7.4)$$

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 ,

$$F_z = mg_0 \quad (7.5)$$

$$F_i = \frac{mg_0}{4} \quad (7.6)$$

$$\omega_i^2 = \frac{mg_0}{4K_T} \quad (7.7)$$

where  $\omega_i^2$  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 behavior 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 linearizatid model can be obtained:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (7.8)$$

## REWRITE B

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (7.11)$$

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 behavior 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**

## **Discussion**

# **Chapter 9**

## **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/](https://en.wikipedia.org/wiki/Tradeoffs_for_locomotion_in_air_and_water/). (Accessed 08/12/16).
- [4] Toward obstacle avoidance on quadrotors. <http://e-collection.library.ethz.ch/eserv/eth:7849/eth-7849-01.pdf>. (Accessed 08/12/16).
- [5] Dc motor speed: System modeling. <http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling>. (Accessed 08/12/16).
- [6] 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/>. (Accessed 19/11/16).

# Chapter 10

## Appendix

### 10.1 Appendix code