Sudan University of Science and Technology

College of Engineering

School of Electrical and Nuclear Engineering

Navigation and Control of the Quad Rotor Flying Robots

التحكم والتوجيه للطائرات الآلية الرباعية

A Project Submitted In Partial Fulfillment for the Requirements of the Degree of B.Sc. (Honor) In Electrical Engineering (Control)

Prepared By:

1. Babiker Kamal Babiker Omer
2. Moaz Salah Awad Ahmed
3. Mohamed Mubarak Mohamed Awadalaleem
4. Wayel Kamal Eldeen Ali Abdelgleel

Supervised By:

Dr. Awadalla Taifour Ali

October 2016

الآية

قال تعالى :

(اللّهُ لاَ إِلَـهَ إِلاَّ هُوَ الْحَيُّ الْقَيُّومُ لاَ تَأْخُذُهُ سِنَةٌ وَلاَ نَوْمٌ لَّهُ مَا فِي السَّمَاوَاتِ وَمَا فِي الأَرْضِ مَن ذَا الَّذِي يَشْفَعُ عِنْدَهُ إِلاَّ بِإِذْنِهِ يَعْلَمُ مَا بَيْنَ أَيْدِيهِمْ وَمَا خَلْفَهُمْ وَلاَ يُحِيطُونَ بِشَيْءٍ مِّنْ عِلْمِهِ إِلاَّ بِمَا شَاء وَسِعَ كُرْسِيُّهُ السَّمَاوَاتِ وَالأَرْضَ وَلاَ يَؤُودُهُ حِفْظُهُمَا وَهُوَ الْعَلِيُّ الْعَظِيمُ).

[البقرة :255]

DEDICATIONS

Every challenging work needs self efforts as well as guidance of elders especially those who were very close to our hearts. Our humble effort is dedicated to our sweet and loving parents, our brothers, our sisters and our friends whose affection, love, encouragement and prays in days and nights make us able to get such success and honor. Without their love and support this project would have never been made possible.

ACKNOWLEDGEMENT

First and foremost we would like to express our sincere gratitude to our advisor **Dr.Awadalla Taifour Ali** for his continuous support in this project. His sharp mind, intuitive understanding, powerful observation and immense knowledge were great help to us. Without his assistance and dedicated involvement in every step throughout the process, this project would have never been accomplished

ABSTRACT

An autonomous quad-rotor is an unmanned aerial vehicle that has four-fixed rotors, where two rotors per axis and each of the axes are aligned with the other. Rotors are powered by four motors and propellers to lift the aircraft quad-rotor are a type of a helicopter aircraft that has vertical take-off and landing capabilities. The main objective of this project is to study and develop an autonomous quad-copter that is capable of flying under the control of an autopilot that's sustainable and expandable for future researches. In order to achieve this objective a comprehensive study of aerodynamics has been included. Focusing on helicopters in particular especially in the field of control surfaces and components, in addition to rapid survey to the history of helicopter development.

An electronic circuit was designed for the control of the quad-rotor. Components used in the design and the implementation of the quad-rotor such as brushless DC motors, propellers, sensors, batteries and the microcontroller are all discussed in detail. The MATLAB software was used to analysis for a particular combination of components, and the system has given good flight performances and acceptable efficiency. The Arduino C programming language has been used for programming the system. The microcomputer receives the Hover, Pitch, Roll, and Yaw commands plus the feedback from the sensors and generates control signals to the four motors. The movement of the motors is controlled by varying PWM signals that are sent to each of the motors.

المستخلص

المروحية الرباعية طائرة بدون طيار ذات أربعة دوّارات ثابتة،حيث يوجد دوارين إثنين علي كل محور ويتعامد كل من المحورين.تدور الدوّرات بواسطة أربعة محركات كهربائية لرفع جسم الطائرة في الهواء، وتعتبر من الطائرات العمودية والتي تمتلك إمكانية الاقلاع والهبوط رأسياً . إن الهدف الرئيسي من هذا البحث هو دراسة وتطوير منظومة المروحية الرباعية لتكون قادرة علي الطيران بتحكم آلـي،وتكون نواة للبحث والتطوير المستقبلي . ولتحقيق تلك الأهداف يتضمن البحث دراسة شاملة للديناميكية الهوائية وعلى وجه الخصوص التركيز علي المروحيات وبالتحديد اسطح التحكم للمروحية ومكوناتها.

تم تصميم دائرة التحكم الإلكترونية للمروحية بعد دراسة المكونات الإلكترونية المستخدمة في بناء المنظومة مثل محركات التيار المستمر والمراوح والمحساسات والبطاريات والمتحكم الدقيق.تم إستخدام برنامج (MATLAB) لتحليل المكونات الإلكترونية المكونة للنظام ، وقد اعطى النظام مدى طيران وكفاءة جيدين. تم إستخدام لغة برمجة (Arduino C) لكتابة أوامر التحكم في نظام المروحية، حيث يستقبل المتحكم الدقيق أوامر التحكم من القنوات الأربعة والتغذية العكسية للمقارنة مع المحساسات وبذا يتم توليد إشارات للتحكم في المحركات الأربعة . يتم التحكم في سرعة المحركات بواسطة التحكم بعرض النبضة للإشارة التي يتم ارسالها إلى كل محرك على حدا.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABREVIATIONS

MEMS	Micro Electro Mechanical Systems
UAVs	Unmanned Aerial Vehicles
GPS	Global Positioning System
PID	Proportional Integral Derivative
LQR	Linear Quadratic Regulator
ISR	Intelligence, Surveillance and Reconnaissance
DC	Direct Current
ESCs	Electronic speed controllers
PWM	Pulse Width Modulation
EMF	Electro-Motive Force
FETs	Field Effect Transistors
IMU	Inertial Measurement Unit
RAM	Random Access Memory
ROM	Read Only Memory
PROM	Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
USB	Universal Serial Bus
LIPO	Lithium Polymer battery
CW	Clock Wise
CCW	Counter Clock Wise
BEC	Battery Eliminator Circuit
NIMH	Nickel-Metal Hydride battery
NICD	Nickel Cadmium(battery technology)
BLDC	Brushless Direct Current(motor)
Aux	Auxiliary
DOF	Degree of Freedom
I2C	Inter Integrated Circuit
SE(3)	Set of Euclidian in the 3 Dimension space

LIST OF SYMBOLS

ϕ	Pitch Angle, rad
$\boldsymbol{\theta}$	Roll Angle, rad
$\boldsymbol{\psi}$	Yaw Angle, rad
K_p	Proportional gain
K_d	Derivative gain
K_i	Integral gain
$e(t)$	Error
C(s)	Output
R	Rotational matrix
P	Point in space
ξ	Position and orientation of an object(pose)
${}^{A}_{B}\xi$	Pose of frame {B} relative to frame {A}
$R_x(\Theta)$	Rotation around X axis
$R_y(\Theta)$	Rotation around Y axis
$R_z(\Theta)$	Rotation around Z axis
T	Thrust force, N
b	Constant
D	Constant
$\boldsymbol{\omega}$	Angular velocity, rad/sec
T_B	Thrust vector, N
τ_x	The torque about the vehicle's X-axis, N.m
τ_y	The torque about the vehicle's Y-axis, N.m
τ_z	The torque about the vehicle's Z-axis, N.m
K	Constant
m	Mass of the quadrotor, kg
$\boldsymbol{v}$	Linear velocity, m/s
G	Gravitational acceleration, m/s^2
J	Moment of inertia matrix, kg. m^2
J_{xx}	Moment of inertia about X axis, kg. m^2
J_{yy}	Moment of inertia about Y axis, kg. m^2
J_{zz}	Moment of inertia about Z axis, kg. m^2
Γ	Torque vector, N.m
A	Coefficients matrix
d	Quadrotor arm Length, cm
r	Radius of sphere, cm
M	Mass of motors, kg
λ_1	Square of angular velocity for motor 1, rad/sec
λ_2	Square of angular velocity for motor 2, rad/sec
λ_3	Square of angular velocity for motor 3, rad/sec
λ_4	Square of angular velocity for motor 4, rad/sec

CHAPTER ONE
INTRODUCTION

1.1 General Concepts

Flying robots or Unmanned Aerial Vehicles (UAV) are becoming increasingly common and span a huge range of size and shape. They have 6 degrees of freedom and they are actuated by forces that is, their motion model is expressed in terms of forces and torques rather than velocities and so,A dynamic model is used rather than a kinematic model [1].

The system under design is quad-rotor flying robot, compared to the fixed wing aircraft quad-rotor is highly maneuverable and can be flown safely indoors which makes quad-rotor well suited for laboratories, compared to conventional helicopters which have large main rotor and tail rotor. Quad-rotor is easier to fly, does not has the complex swash plate mechanism and is easier to model and control. The system takes a simple configuration of four motors spinning a total of four propellers attached to the shafts of each motor, the motors are housed in a frame takes a shape of – × –configuration.

1.2 Problem Statement

Quadrotor flying robots are highly unstable systems. The problem is to take them back to stability during flight as fast as it could, and also achieve the autonomous navigation of a pre-planned path or trajectory.

1.3 Objectives

- Implement the real time response of quad-rotor flying robot
- Design a proportional integral derivative (PID) controller to control and improve system performance.

- Design a complementary filter algorithm to get the sensor fusion using the raw data of the IMU sensor board.
- Plan a specific path for the quadrotor to take to a reach certain target using trajectory algorithms.

1.4 Methodology

- Study of all previous studies.
- Study the basics of 3D geometry, linear algebra and the frame of reference coordinate system.
- Study and analyze the dynamics and the kinematics of the system.
- Design of PID controller using manual tuning.
- Build Arduino microcontroller program to control quad-rotor altitude with PID algorithm as software included in the controller.
- Mathematically model the system and create a well descriptive simulated design.

1.5 Layout of Thesis

This study consists of five chapters; Chapter One gives an introduction to the principles of the work, in addition its reasons, motivation and objectives. Chapter Two discuses the theoretical background of control systems, navigation, nonlinear systems, flying robots, quad-rotor background, quad-rotor structure, quad-rotor manoeuvrability, quad-rotor components, PID controller and microcontroller systems. Chapter Three presents the system mathematical model includes kinematic and dynamic equations that describe the behavior of quad-rotor and control design that accomplish the stabilization of quad-rotor system. Chapter Four deal with the practical model of the system and shows the experimental results. Finally, Chapter five provides the conclusions and recommendations.

CHAPTER TWO

THEORICAL BACKGROUND AND LITERATURE REVIEW

2.1 Control System

One of the most commonly asked questions by a novice on a control system is: What is a control system? To answer the question, we can say that in our daily lives there are numerous "objectives" that need to be accomplished. For instance, in the domestic domain, we need to regulate the temperature and humidity of homes and buildings for comfortable living. For transportation, we need to control the automobile and airplane to go from one point to another accurately and safely. Industrially, manufacturing processes contain numerous objectives for products that will satisfy the precision and cost effectiveness requirements [1].

In recent years, control systems have assumed an increasingly important role in the development and advancement of modern civilization and technology. Practically every aspect of our day-to-day activities is affected by some type of control system [1].

Control systems are found in abundance in all sectors of industry, such as quality control of manufactured products, automatic assembly lines, machine-tool control, space technology and weapon systems, computer control, transportation systems, power systems, robotics, Micro-Electro-Mechanical Systems (MEMS), nanotechnology, and many others. Even the control of inventory and social and economic systems may be approached from the theory of automatic control [1].

Since advances in the theory and practice of automatic control provide the means for attaining optimal performance of dynamic systems, improving productivity, relieving the drudgery of many routine repetitive manual operations, and more, most engineers and scientists must now have a good understanding of this field[2].

2.2.1 Advantages of control system

With control systems we can move large equipment with precision that would otherwise be impossible. We can point huge antennas toward the farthest reaches of the universe to pick up faint radio signal controlling these antennas by hand would be impossible. Because of control systems, elevators carry us quickly to our destination, automatically stopping at the right floor. We alone could not provide the power required for the load and the speed; motors provide the power, and control systems regulate the position and speed [2].

2.1.2 Historical review

The first significant work in automatic control was James Watt's centrifugal governor for the speed control of a steam engine in the eighteenth century. Other significant works in the early stages of development of control theory were due to Minor sky, Hazen and Nyquist among many others. In 1922, Minor sky worked on automatic controllers for steering ships and showed how stability could be determined from the differential equations describing the system. In 1932, Nyquist developed a relatively simple procedure for determining the stability of closed-loop systems on the basis of open-loop response to steady-state sinusoidal inputs.

A significant date in the history of automatic feedback control systems is 1934, when Hazen’s paper ‘‘Theory of Servomechanisms’’ was published in the Journal of the Franklin Institute, marking the beginning of the very intense interest in this new field. It was in this paper that the word servomechanism originated, from the words servant (or slave) and mechanism. Black’s important paper on feedback amplifiers appeared in the same year. After World War II, control theory was studied intensively and applications have proliferated many books and thousands of articles and technical papers have been written, and the application of control systems in the industrial and military fields has been extensive. This rapid growth of feedback control systems was accelerated by the equally rapid development and widespread use of computers.

During the decade of the 1940s frequency response methods (especially the Bode diagram methods due to Bode) made it possible for engineers to design linear closed loop control systems that satisfied performance requirements. From the end of the 1940s to the early 1950s the root-locus method due to Evans was fully developed. The frequency-response and root-locus methods, which are the core of classical control theory, lead to systems that are stable and satisfy a set of more or less arbitrary performance requirements. Such systems are, in general, acceptable but not optimal in any meaningful sense.

Classical control theory, which deals only with single input single output systems, becomes powerless for multiple input multiple output systems. Since about 1960, because the availability of digital computers made possible time domain analysis of complex systems, modern control theory, based on time domain analysis and synthesis using state variables, has been developed to cope with the increased complexity of modern plants and the stringent requirements on accuracy, weight, cost in military, space and industrial applications [2].

2.1.3 Open loop control system

Those systems in which the output has no effect on the control action are called open-loop control systems. In other words, in an open-loop control system the output is neither measured nor fed back for comparison with the input. One practical example is a washing machine. Soaking, washing, and rinsing in the washer operate on a time basis. The machine does not measure the output signal, that is, the cleanliness of the clothes.

In any open-loop control system the output is not compared with the reference input. Thus, to each reference input there corresponds a fixed operating condition; as a result, the accuracy of the system depends on calibration. In the presence of disturbances, an open-loop control system will not perform the desired task. Open-loop control can be used, in practice, only if the relationship between the input and output is known and if there are neither internal nor external disturbances. Clearly, such systems are not feedback control systems. Note that any control system that operates on a time basis is open loop. For instance, traffic control by means of signals operated on a time basis is another [2].

2.1.4 Closed-loop control systems

A system that maintains a prescribed relationship between the output and the reference input by comparing them and using the difference as a means of control is called a closed-loop control system. An example would be a room temperature control system. By measuring the actual room temperature and comparing it with the reference temperature, the thermostat turns the heating or cooling equipment on or off in such a way as to ensure that the room temperature remains at a comfortable level regardless of outside conditions.

In a closed-loop control system the actuating error signal, which is the difference between the input signal and the feedback signal (which may be the output signal itself or a function of the output signal and its derivatives and/or integrals), is feedback to the controller so as to reduce the error and bring the output of the system to a desired value. The term closed-loop control always implies the use of feedback control action in order to reduce system error [2].

2.2 Nonlinear Systems

A system is nonlinear if the principle of superposition does not apply. Thus, for a nonlinear system the response to two inputs cannot be calculated by treating one input at a time and adding the results. Although many physical relationships are often represented by linear equations, in most cases actual relationships are not quite linear. In fact, a careful study of physical systems reveals that even so-called "linear systems" are really linear only in limited operating ranges.

In practice, many electromechanical systems, hydraulic systems, pneumatic systems, and so on, involve nonlinear relationships among the variables. For example, the output of a component may saturate for large input signals. There may be a dead space that affects small signals. (The dead space of a component is a small range of input variations to which the component is insensitive). Square-law nonlinearity may occur in some components. For instance, dampers used in physical systems may be linear for low-velocity operations but may become nonlinear at high velocities, and the damping force may become proportional to the square of the operating velocity.

In control engineering a normal operation of the system may be around an equilibrium point, and the signals may be considered small signals around the equilibrium. (It should be pointed out that there are many exceptions to such a case). However, if the system operates around an equilibrium point and if the signals involved are small signals, then it is possible to approximate the nonlinear system by a linear system. Such a linear system is equivalent to the nonlinear system considered within a limited operating range. Such a linearized model (Linear Time-Invariant model) is very important in control engineering.

The linearization procedure to be presented in the following is based on the expansion of nonlinear function into a Taylor series about the operating point and the retention of only the linear term. Because of neglecting higher-order terms of Taylor series expansion, these neglected terms must be small enough; that is, the variables deviate only slightly from the operating condition [1, 2].

2.3 Flying Robots

Flying robots or Unmanned Aerial Vehicles (UAV) are becoming increasingly common and span a huge range of size and shape. Applications include military operations, surveillance, meteorological investigations and robotics research. Fixed wing UAVs are similar in principle to passenger aircraft with wings to provide lift, a propeller or jet to provide forward thrust and control surface for maneuvering. Rotorcraft UAVs have a variety of configurations that include conventional helicopter design with a main and tail rotor, a coax with counter-rotating coaxial rotors and quadrotors. Rotorcraft UAVs are used for inspection and research and have the advantage of being able to take off vertically [3].

Flying robots differ from ground robots in some important ways:

Firstly they have 6 degrees of freedom and their configuration $q \in SE(3)$. Secondly they are actuated by forces, their motion model is expressed in term of forces and torques rather than velocities as was the case for the bicycle model – we use a dynamic rather than a kinematic model. Underwater robots have many similarities to flying robots and can be considered as vehicles that fly through water and there are underwater equivalents to fixed wing aircraft and rotorcraft. The principle differences underwater are an upward buoyancy force; drag forces that are much more significant than in air and added mass [3].

2.3.1 Robot navigation

Robot navigation is the problem of guiding a robot towards a goal. The human approach to navigation is to make maps and erect signposts, and at first glance it seems obvious that robots should operate the same way. However many robotic tasks can be achieved without any map at all, using an approach referred to as reactive navigation. For example heading towards a light, following a white line on the ground, moving through a maze by following a wall, or vacuuming a room by following a random path. The robot is reacting directly to its environment: the intensity of the light, the relative position of the white line or contact with a wall.

Today more than 5 million Roomba vacuum cleaners are cleaning floors without using any map of the rooms they work in. The robots work by making random moves and sensing only that they have made contact with an obstacle. The more familiar human-style map-based navigation is used by more sophisticated robots. This approach supports more complex tasks but is itself more complex. It imposes a number of requirements, not the least of which is a map of the environment. It also requires that the robot's position is always known [3].

- **Reactive navigation**

Surprisingly complex tasks can be performed by a robot even if it has no map and no real idea about where it is. As already mentioned robotic vacuum cleaners use only random motion and information from contact sensors to perform a complex task. Insects such as ants and bees gather food and return it to the nest based on input from their senses, they have far too few neurons to create any kind of mental map of the world and plan paths through it. Even single-celled organisms such as flagellate protozoa exhibited goal seeking behavior [3].

- **Map-Based navigation**

The key to achieving the best path between points A and B, is to use a map. Typically best means the shortest distance but it may also include some penalty term or cost related to traversability which is how easy the terrain is to drive over – it might be quicker to travel further but over better roads. Amore sophisticated planner might also consider the kinematics and dynamics of the vehicle and avoid paths that involve turns that are tighter than the vehicle can execute [3].

2.3.2 Robot localization

In the previous discussion of map-based navigation it is assumed that robot had a means of known position. Some of common techniques had been discussed to estimate the location of a robot in the world using an approach known as localization.

Today GPS makes outdoor localization so easy to be generated. Unfortunately GPS is a far from perfect sensor since it relies on very weak radio signals received from distant orbiting satellites. This means that GPS cannot work where there is no line of sight radio reception, for instance indoors, underwater, underground, in urban canyons or in deep mining pits. GPS signals are also extremely weak and can be easily jammed and this is not acceptable for some applications. GPS has only been in use since 1995 yet human-kind has been navigating the planet and localizing for many thousands of years [3].

2.4 Quadrotor Robot

A quadrotor is a helicopter which has four equally spaced rotors, usually arranged at the corners of a square body. With four independent rotors, the need for a swash plate mechanism is alleviated. The swash plate mechanism was needed to allow the helicopter to utilize more degrees of freedom, but the same level of control can be obtained by adding two more rotors [3].

The development of quadrotors has stalled until very recently, because controlling four independent rotors has proven to be incredibly difficult and impossible without electronic assistance. The decreasing cost of modern microprocessors has made electronic and even completely autonomous control of quadrotors feasible for commercial and military [3].

Quadrotor has received considerable attention from researchers as the complex phenomena of the quadrotor have generated several areas of interest. The basic dynamical model of the quadrotor is the starting point for all of the studies but more complex aerodynamic properties has been introduced. Different control methods has been researched, including PID controllers , back stepping control , nonlinear H∞ control, LQR controllers, and nonlinear controllers with nested saturations. Control methods require accurate information from the position and attitude measurements performed with a gyroscope, an accelerometer, and other measuring devices, such as GPS, and sonar and laser sensors [3].

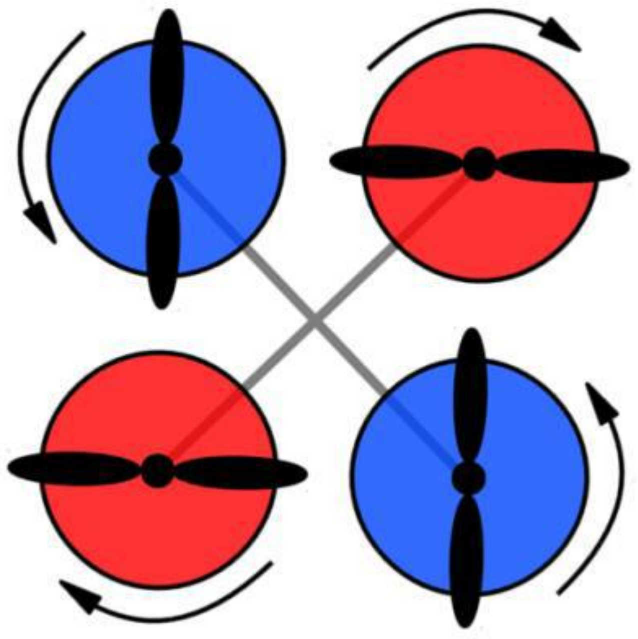

Figure 2.1: Quadrotor propellers direction and drag torques.

2.5 Quadrotor Structure

Quadrotor control is a fundamentally difficult and interesting problem. With six degrees of freedom (three translational and three rotational) and only four independent inputs (rotor speeds), quadcopters are severely under actuated. In order to achieve six degrees of freedom, rotational and translational motions are coupled.

The resulting dynamics are highly nonlinear, especially after accounting for the complicated aerodynamic effects. Finally, unlike ground vehicles, quadrotors have very little friction to prevent their motion, so they must provide their own damping in order to stop moving and remain stable. Together, these factors create a very interesting control problem. A very simplified model of quadrotor dynamics and design controllers is presented to follow a designated trajectory. Then controllers will be tested with a numerical simulation of a quadrotor in flight.

2.5.1 Quadrotor configurations

Figure 2.2 shows the two simple configurations of quadrotors, and as shown the four motors are arranged in '+' and '×' configuration. Propellers 1 and 3 rotates clockwise, while 2 and 4 rotates counter clockwise and this is required to compensate the action/reaction effect generated by the rotor. In order to have all thrusts in the same direction propellers 1 and 3 are selected to be in opposite pitch with respect to 2 and 4 [4, 5].

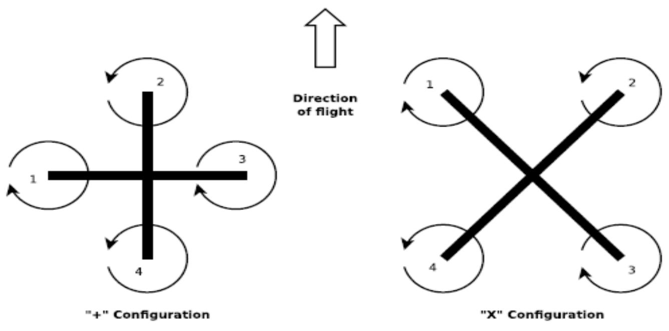

Figure 2.2: Quadrotor Configurations

2.5.2 Quadrotor basic mechanics

If speeds of independent motors are varied, the position and orientation of the robot will be controlled. If one of the rotors spins faster as in Figure 2.3 below, the robot will pitch in one direction. In order to move the vehicle from one side to another, just translating it along the horizontal direction the robot needs to be pitched forward so that the thrust factor points in the horizontal direction. That allows the vehicle to accelerate forward. But then when the robot get close to the destination the vehicle needs to be stopped by pitching it in the opposite direction, creating a reverse thrust that allows it slow down when it gets to its destination. And finally pitch back to equilibrium [5].

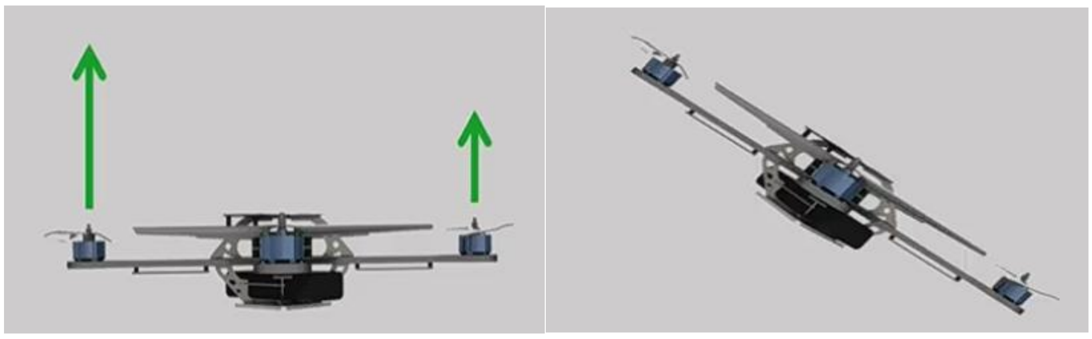
Figure 2.3: The creation of movement in quadrotors

2.6 Quadrotor Manoeuvrability

Quadrotors are three dimensional robots that need six degrees of freedom. The movement in the three dimensional space is generated by coupling the linear movement (translation) with the angular movement (rotation) resulting in a robot that can linearly move through and rotate about the three principal axis (x, y and z) [5].

The rotation around the three principal axes is shown in figure 3.3. Assuming the robot heading (nose) is aligned with the Y axis, each axis rotation is defined by an angle as follow:

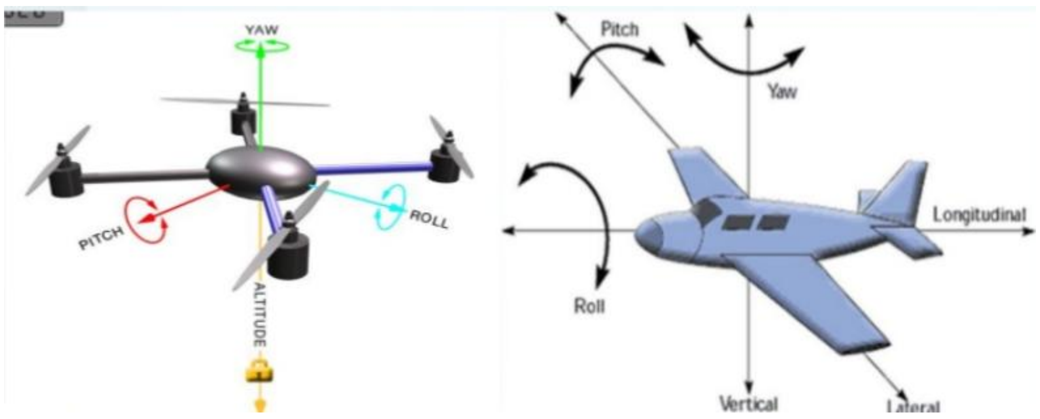

Figure 2.4: Quadrotor orientation

- The rotation around the Y axis is known as the pitch, denoted by Phi ($\boldsymbol{\Phi}$)
- The rotation around the X axis is known as the roll, denoted by Theta ($\boldsymbol{\theta}$)
- The rotation around the Z axis is known as the yaw, denoted by Psi ($\boldsymbol{\psi}$)

2.6.1 Roll and pitch

To make the quadrotor rotate about the roll or pitch axes, the flight controller makes the motors on one side of the quadrotor spin faster than the motors on the other side. This means that one side of the quadrotor will have more lift than the other side, causing the quadrotor to tilt. So, for example, to make a quad rotor roll right (or rotate about the roll axis clockwise), the flight controller will make the two motors on the left side of the quadrotor spin faster than the two motors on the right side. The left side of the craft will then have more lift than the right side, which causes the quadrotor to tilt.

Similarly, to make a quad rotor pitch down (rotate about the pitch axis clockwise) the flight controller will make the two motors on the back of the craft spin faster than the two motors on the front.

2.6.2 Yaw rotation

Controlling the quadrotor's rotation about the yaw axis is a bit more complex than controlling its rotation about the roll or pitch axes. First, rotation about the yaw axis is shown as in Figure 2.6.

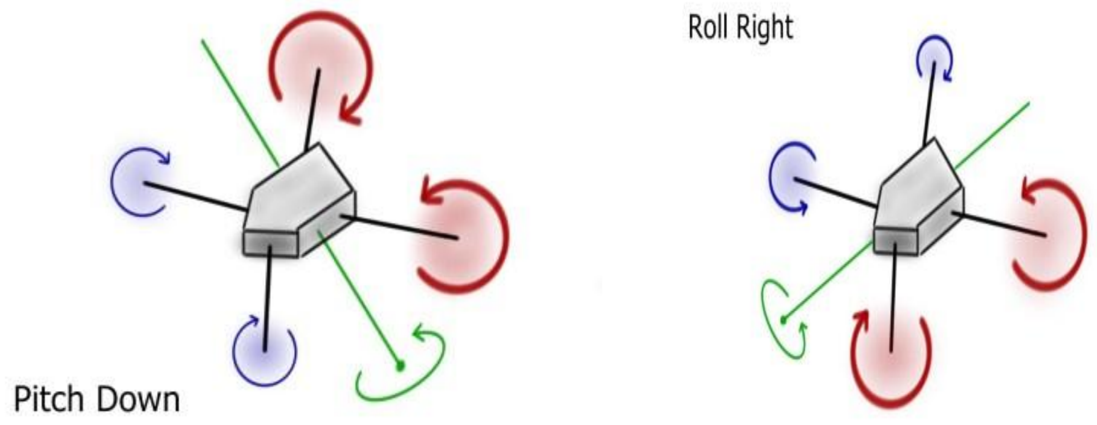

Figure 2.5: The roll and pitch orientations

When assembling and programming quadrotors, The motors were set up so that each motor spins in the opposite direction than its neighbors. In other words, starting from the front-left motor and moving around the quadrotor clockwise, the motors rotational directions alternate, CW, CCW, CW, CCW. This rotational configuration is used to neutralize, or cancel out, each motor's tendency to make the quadrotor rotate.

When a propeller spins, for example, clockwise, conservation of angular momentum means that the body of the quadrotor will have a tendency to spin counter-clockwise. This is due to Newton's third law of motion, "for every action, there is an equal and opposite reaction." The body of the quadrotor will tend to spin in the direction opposite to the rotational direction of the propellers. This is shown clearly in Figure 2.6 below:

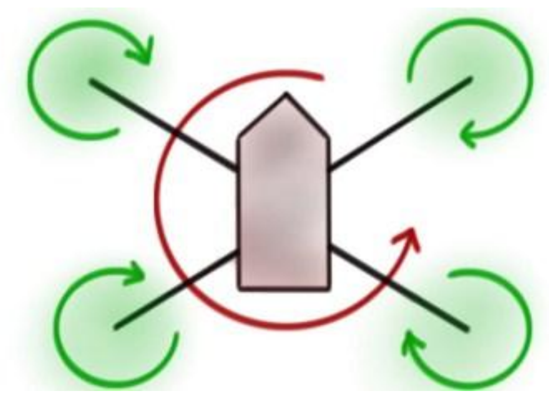

Figure 2.6: The reaction moments due to rotor spinning

Bringing it all together now, each of the quadrotor four rotors tends to make the quadrotor rotate in the opposite direction than their spin. So by using pairs of rotors spinning in opposite directions it is possible to cancel out this effect and the quadrotor does not spin about the yaw axis.

So therefore, when it is actually required to rotate the quadrotor about the yaw axis, the flight controller will slow down opposite pairs of motors relative to the other pair. This means the angular momentum of the two pairs of propellers will no longer be in balance and the craft rotates. So quadrotor can be rotated in either direction by slowing down different pairs of motors.

2.6.3 Application of quadrotors

Currently, the main quadrotor applications are defense related and the main investments are driven by future military scenarios. Most military unmanned aircraft systems are primarily used for intelligence, surveillance, reconnaissance (ISR), and strikes. The next generation of quadrotors will execute more complex missions such as air combat, target detection, recognition, destruction, strike/suppression of an enemy's air defense, electronic attack, network node/communications relay, aerial delivery/resupply, anti-surface ship warfare, anti-submarine warfare, mine warfare, ship to objective maneuvers, offensive and defensive counter air and airlift.

Today the civilian markets for Quadrotors are still emerging. However, the expectations for the market growth of civil and commercial Quadrotors are very high for the next decade [5].

Potential civil applications of Quadrotors are:

- Inspection of terrain, pipelines, utilities, buildings, etc
- Law enforcement and security applications
- Surveillance of coastal borders, road traffic, etc
- Disaster and crisis management, search and rescue

- Environmental monitoring.
- Agriculture and forestry.
- Fire fighting.
- Communications relay and remote sensing.
- Aerial mapping and meteorology.
- Research by university laboratories.
- And many other applications.

2.7 Brushless DC Motors

A motor is an electrical machine that converts electrical energy into mechanical energy. The working principal is based on the theory that when a current-carrying conductor is placed in a magnetic field, it experiences a mechanical force whose direction is giving by Fleming’s left- hand rule.

The brushless motor, unlike the DC brushed motor, has the permanent magnets glued on the rotor. It has usually four magnets around the perimeter. The stator of the motor is composed by the electromagnets, usually four of them, placed in a cross pattern with 90 degrees angle between them.

The major advantage of the brushless motors due to the fact that the rotor carries only the permanent magnets, is that it needs no power at all. No connection needs to be made with the rotor, thus, no brush-commutator pair is needed and this is how the brushless motors took their name. This feature in brushless motor increase the reliability, as the brushes wears off very fast. Moreover, brushless motors are more silent and more efficient in terms of power consumption.

2.8 Electronic Speed Controllers

An Electronic Speed Controller or ESC is an electronic circuit with the purpose of varying motor's speed, its direction and possibly also to act as a dynamic brake. The ESC generally accepts a nominal 50 hertz PWM servo input signal whose pulse width varies from 1 ms s to 2 ms, when supplied with a 1 ms width pulse at 50 Hz, the ESC responds by turning off the DC motor attached to its output. A 1.5 ms pulse-width input signal drives the motor at approximately half-speed. When presented with 2.0 ms input signal, the motor runs at full speed.

The correct phase varies with the motor rotation, which is to be taken into account by the ESC: Usually, back EMF from the motor is used to detect this rotation. Computer-programmable speed controls generally have user-specified options which allow setting low voltage cut-off limits, timing, acceleration, braking and direction of rotation. Reversing the motor's direction may also be accomplished by switching any two of the three leads from the ESC to the motor. There are three wires that go between the motor and the ESC since these motors are three phase motors, there are three coils inside. The coils are energized in sequence to make the motors spin. So the ESC's job is to energize the coils in sequence, but it needs to time each cycle correctly so the motor can actually accelerate to the right speed. The ESC has a microcontroller inside that turns on or off the coils using FETs and also determines timing by measuring the feedback in the coils caused by the movement of the magnets.

2.9 Inertial Measurement Unit Sensor

A sensor is a device that converts a physical phenomenon into an electrical signal. As such, sensors represent a main part of the interface between the physical world and the world of electrical devices. The other part of this interface is represented by actuators, which convert electrical signals into physical phenomena.

An Inertial Measurement Unit or IMU is the main component of inertial guidance systems used in air space, and watercraft, including guided missiles. An IMU works by sensing motion including the type, rate, and direction of that motion using a combination of accelerometers and gyroscopes. Accelerometers are placed such that their measuring axes are orthogonal to each other. An IMU works by detecting the current rate of acceleration, as well as its changes in rotational attributes, including pitch, roll and yaw. This data is then fed into a computer, which calculates the current speed and position, given a known initial speed and position.

IMU usually consists of two main parts they are:

- Accelerometer
- Rate Gyros

2.10 Proportional Integral Derivative Controller

A Proportional Integral Derivative PID controller is a feedback control

algorithm widely used in industrial control systems. A PID controller calculates the error value which is the difference between a measured process variable and a desired set point, the controller attempts to minimize the error by adjusting the process.

The PID controller algorithm involves three separate constant parameters as shown is Figure 2.7 and is accordingly sometimes called three-term control: the proportional, the integral and derivative values, denoted P, I, and D. These values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, a damper, or the power supplied to a heating element.

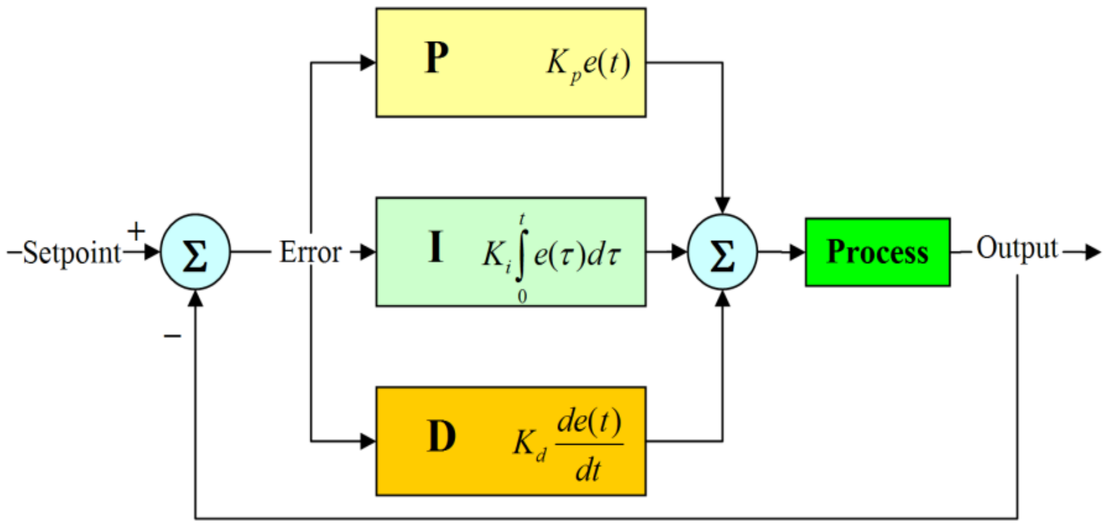

Figure 2.7: Proportional Integral Derivative PID

2.10.1 Proportional controller

The Proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant Kp, called the proportional gain constant. The proportional term is given by:

$$P=Kp\ (e\ (t)) \qquad (2.1)$$

Where P is the proportional controller, Kp is the gain, e(t) the error signal.

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. Tuning theory and industrial practice indicate that the proportional term should contribute the bulk of the output change.

2.10.2 Proportional integral controller

The main function of the integral action is to make sure that the process output agrees with the set point in steady state. With Proportional control, there is normally a control error in steady state. With integral action, small positive error will always lead to an increasing signal, and a negative error will give a decreasing control signal no matter how small the error is.

$$PI = K_P (e(t)) + K_i \int e(t) dt \tag{2.2}$$

Where PI is the Proportional integral controller, K_i is the integral gain.

2.10.3 Proportional derivative controller

The purpose of the derivative action is to improve the close-loop stability. Because of the process dynamics, it will take some time before a change in the control variable is noticeable in the progress output. Thus, the control system will be late in correction for an error. The action of a controller with proportional and derivative may be interpreted as if the control is made proportional to the predicted process output, where the prediction is made by extrapolating the error by the tangent to the error curve.

$$PD = K_p\, e(t) + K_d \frac{de(t)}{dx} \tag{2.3}$$

Where PD is the proportional derivative controller, K_d is the gain.

2.10.4 Proportional integral derivative controller

The Proportional Integral Derivative PID controller has three terms. The Proportional term P corresponds to proportional control. The integral term I give a control action that is proportional to the time integral of the zero. The derivative term D is proportional to the time derivative of the control error. This term allows prediction of the future error. There are many variations of the PID algorithm that will substantially improve its performance and operability.

$$\text{PID}=K_p\, e(t) + K_d\, \frac{de(t)}{dx} + K_i \int e(t)dt \tag{2.4}$$

By taking Laplace transform PID controller transfer function become:

$$C(s)=K_p + \frac{K_i}{s} + K_d\, s \tag{2.5}$$

Where $C(s)$ is the output signal.

2.10.5 Tuning of proportional integral derivative controller

The process of selecting the controller parameters to meet given performance specifications is known as controller tuning. There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, and then choosing P, I, and D based on the dynamic model parameters.

In particular, when the mathematical model of the plant is unknown and therefore analytical design methods cannot be used, PID controls prove to be most useful. In the field of process control systems, it is well known that the basic and modified PID control schemes have proved their usefulness in providing satisfactory control, although in many given situations they may not provide optimal control.

If a mathematical model of the plant can be derived, then it is possible to apply various design techniques for determining parameters of the controller that will meet the transient and steady-state specifications of the closed loop system. However, if the plant is so complicated that its mathematical model cannot be easily obtained, then an analytical or computational approach to the design of a PID controller is not possible then only experimental approaches are used to the tuning of PID controllers.

There are many methods of PID tuning such as:

- Manual tuning.
- Ziegler-Nichols.
- Tyreus luyben.
- Cohen-coon.
- Software tools.

2.11 Microcontroller

It is a highly integrated chip that contains all the components comprising a controller. Typically this includes a CPU, RAM, ROM and I/O ports. Unlike a general-purpose computer, which also includes all of these components, a microcontroller is designed for a very specific task to control a particular system. As a result, the parts can be simplified and reduced, which cuts down on production cost.

2.11.1 History of microcontroller

The first computer system on a chip optimized for control applications was the Intel 8048 microcontroller with both RAM and ROM on the same chip. Most microcontrollers at that time had two variants; one had an erasable EEPROM program memory, which was significantly more expensive than the PROM variant which was only programmable once.

The introduction of EEPROM memory allowed microcontrollers (beginning with the Microchip PIC16x84) to be electrically erased quickly without an expensive package as required for EPROM. The same year, Atmel introduced the first microcontroller using Flash memory. Other companies rapidly followed suit, with both memory types. Nowadays microcontrollers are low cost and readily available for hobbyists, with large online communities around certain processors.

A microcontroller is a single-chip computer .Micro suggests that the device is small, and controller suggests that it is used in control applications. Another term for microcontroller is embedded controller, since most of the microcontrollers are built into (or embedded in) the devices that controlling.

Microcontrollers have traditionally been programmed using the assembly language of the target device. Although the assembly language is fast, it has several disadvantages. An assembly program consists of mnemonics, which makes learning and maintaining a program written using the assembly language difficult. Also, microcontrollers manufactured by different firms have different assembly languages, so the user must learn a new language with every new microcontroller he or she uses.

Microcontrollers can also be programmed using a high-level language, such as BASIC, PASCAL, or C. High-level languages are much easier to learn than assembly languages and also facilitate the development of large and complex programs.

2.11.2 Microcontroller application

Microcontroller applications are found in many life filed, for example in Cell phone, watch, recorder, calculators, mouse, keyboard, modem, fax card, soundcard, battery charger, door lock, alarm clock, thermostat, air conditioner, TV Remotes, in Industrial equipment like Temperature and pressure controllers, counters and timers.

2.11.3 Arduino microcontroller

Arduino is a small microcontroller board with a USB plug to connect to the computer and a number of connection sockets that can be wired up to external electronics, such as motors, relays, light sensors, laser diodes, loudspeakers, microphones, etc. Arduino can either be powered through the USB connection from the computer or from a 9V battery. Arduino can be controlled from the computer or programmed by the computer and then disconnected and allowed to work independently.

- **The Arduino board**

It is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments in simple terms, the Arduino is a tiny computer system that can be programmed with instructions to interact with various forms of input and output. The current Arduino board model, the Uno, is quite small in size compared to the average human hand, as shown in Figure 2.8.

Although it might not look like much to the new observer, the Arduino system allows creating devices that can interact with the world. By using an almost unlimited range of input and output devices, sensors, indicators, displays, motors, and more, the exact interactions required to create a

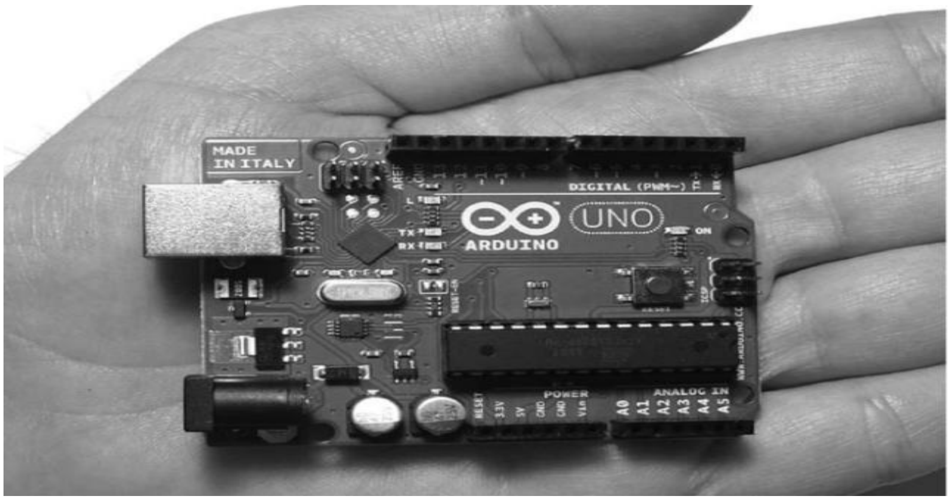

Figure 2.8: The size of Arduino.

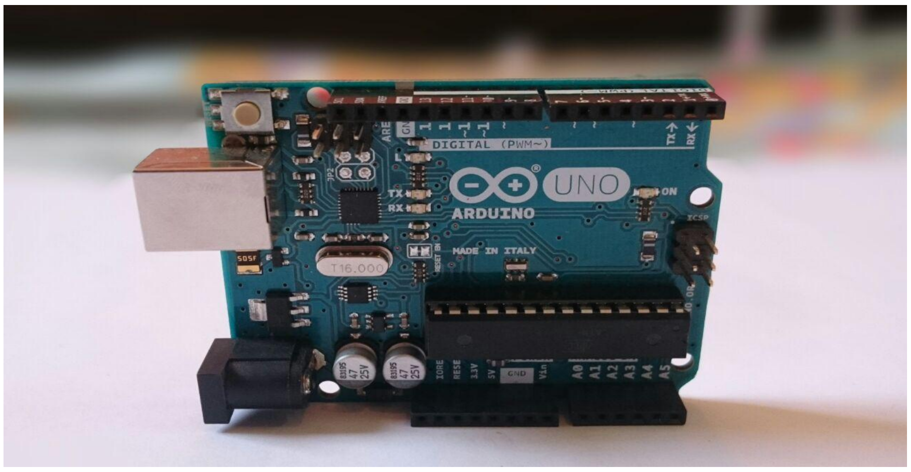
Figure 2.9: Arduino microcontroller board

functional device can be programmed. For example, artists have created installations with patterns of blinking lights that respond to the movements of passers-by, high school students have built autonomous robots that can detect an open flame and extinguish it, and geographers have designed systems that monitor temperature and humidity and transmit this data back to their offices via text message. In fact, there are infinite numbers of examples with a quick search on the Internet. By taking a quick tour of the Uno Starting at the left side of the board there are two connectors, as shown in Figure 2.10.

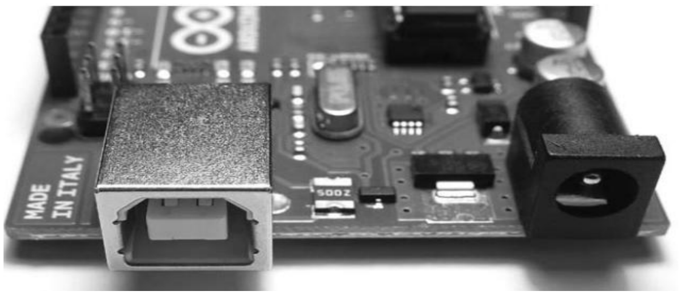
Figure 2.10: The USB and power connectors

On the far left is the Universal Serial Bus (USB) connector. This connects the board to the computer for three reasons; to supply power to the board, to upload the instructions to the Arduino, and to send and receive from a computer. On the right is the power connector, this connector can power the Arduino with a standard mains power adapter.

At the lower middle is the heart of the board: the microcontroller, as shown in Figure 2.11.

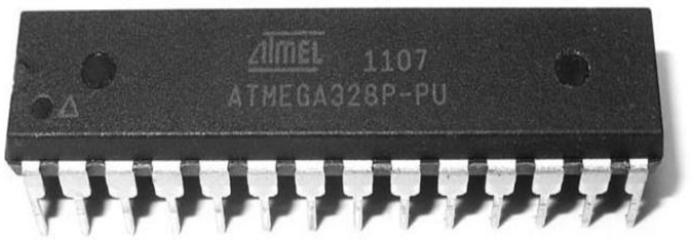

Figure 2.11: The microcontroller

The microcontrollers represent the "brains" of the Arduino. It is a tiny computer that contains a processor to execute instructions, includes various types of memory to hold data and instructions from the sketches, and provides various avenues of sending and receiving data. Just below the microcontroller are two rows of small sockets, as shown in Figure 2.12.

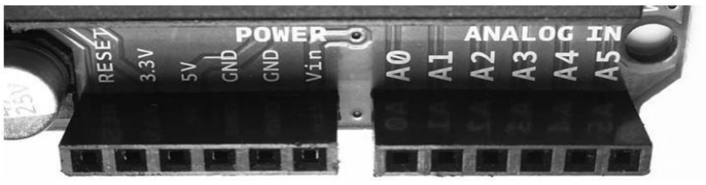

Figure 2.12: The power and analog sockets

The first row offers power connections and the ability to use an external RESET button. The second row offers six analog inputs that are used to measure electrical signals that vary in voltage. Furthermore, pins A4 and A5 can also be used for sending data to and receiving it from other devices. Along the top of the board are two more rows of sockets, as shown in Figure 2.13.

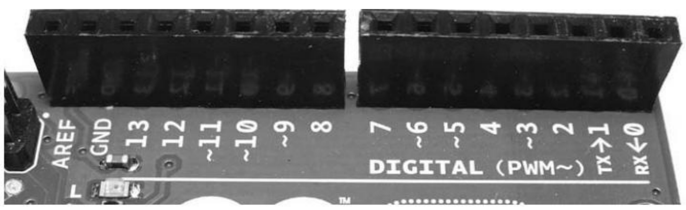

Figure 2.13: The digital input/output pins

Sockets (or pins) numbered 0 to 13 are digital input/output (I/O) pins. They can either detect whether or not an electrical signal is present or generate a signal on command. Pins 0 and 1 are also known as the serial port, which is used to send and receive data to other devices, such as a computer via the USB Connector circuit. The pins labelled with a tilde (~) can also generate a varying electrical signal, which can be useful for such things as creating lighting effects or controlling electric motors.

Next are some very useful devices called light-emitting diodes (LEDs); these very tiny devices light up when a current passes through them. The Arduino board has four LEDs: one on the far right labelled ON, which indicates when the board has power, and three in another group, as shown in Figure 2.14.

The LEDs labelled TX and RX light up when data is being transmitted or received between the Arduino and attached devices via the serial port and USB.

The L-LED connected to the digital I/O pin number 13. The little black square part to the left of the LEDs is a tiny microcontroller that controls the USB interface that allows Arduino to send data to and receive it from a computer.

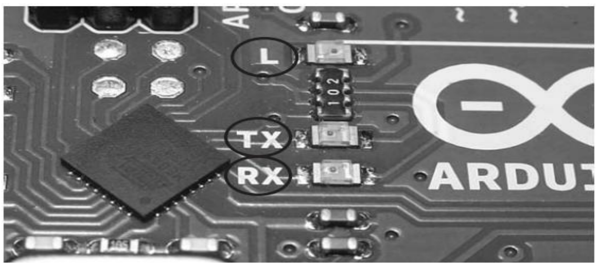

Figure 2.14: The onboard LEDs

And, finally, the RESET button is shown in Figure 2.15.

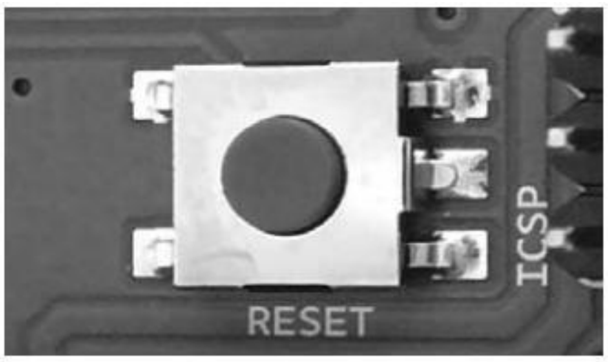

Figure 2.15: The RESET button

CHAPTER THREE
SYSTEM MODELLING AND DESIGN

3.1 Introduction

A quadrotor is a helicopter which has four equally spaced rotors, usually arranged at the corners of a square body .With four independent rotors, the development of quadcopters has stalled until very recently, because controlling four independent rotors has proven to be incredibly difficult and impossible without electronic assistance. The decreasing cost of modern microprocessors has made electronic and even completely autonomous control of quadcopters feasible for commercial, military, and even hobbyist purposes.

The design and development of quadrotor platforms have gained popularity in recent years, due to their flexibility and potential capabilities. Considerable work has been done to investigate aerodynamic factors in hope of enhancing vehicle performance and efficiency. Some of these factors include the distortion and disruption of airflow due to the flow interaction between the four rotating blades at close proximity and propeller blade flapping. Airfoil and platform designs can also be customized in order to improve aerodynamic efficiency.

The flight dynamics of a quadcopter is complex and this makes attitude and position estimation as well as controller implementation challenging. There have been several attempts to model the quadrotor helicopter in order to comprehend its dynamics.

Numerous controllers, both linear and nonlinear, have been proposed to allow the platform to achieve a high level of stability. Examples include model reference adaptive control a nonlinear controller derived using back-stepping approaches and a controller based on a nested saturation technique.

There is also a considerable body of work on estimating attitude for onboard flight control systems. Apart from attitude estimation and stabilization, work has been done to expand the capabilities of the platforms. Altitude control is seen as essential for many applications and thus, it has been investigated and some significant studies on it can be found. Autonomous navigation and collision avoidance algorithms are useful in enhancing the system autonomy.

3.2 Quadcopter Kinematics

Kinematic is the branch of classical mechanics which describes the motion of points (alternatively "particles"), bodies (objects), and systems of bodies without consideration of the masses of neither those objects nor the forces that may have caused the motion. Understanding the kinematics is the key idea of understanding the behavior of the quadrotor; because it helps defining the position and orientation of objects in an environment, which is in general a fundamental requirement in robotics.

3.2.1 The coordinate system and frame of reference theory

A point in space is a familiar concept from mathematics and can be described by a coordinate vector, also known as a bound vector, as shown in Figure 3.1a. The vector represents the displacement of the point with respect to some reference coordinate frame. A coordinate frame, or Cartesian coordinate system, is a set of orthogonal axes which intersect at a point known as the origin.

More frequently it is required to consider a set of points that comprise some object. It is assumed that the quadrotor body is rigid and that its constituent points maintain a constant relative position with respect to the object’s coordinate frame as shown in figure 3.1b. Instead of describing the individual points, the position and orientation of the quadrotors are described by the position and orientation of its coordinate frame.

The position and orientation of a coordinate frame is known as its pose and is shown graphically as a set of coordinate axes. The relative pose of a frame with respect to a reference coordinate frame is denoted by the symbol 'ξ', pronounced ksi. Figure 3.2 shows two frames {A} and {B} and the relative pose ${}^{A}_{B}\xi$ which describes {B} with respect to {A}. The leading superscript denotes the reference coordinate frame and the subscript denotes the frame being described.

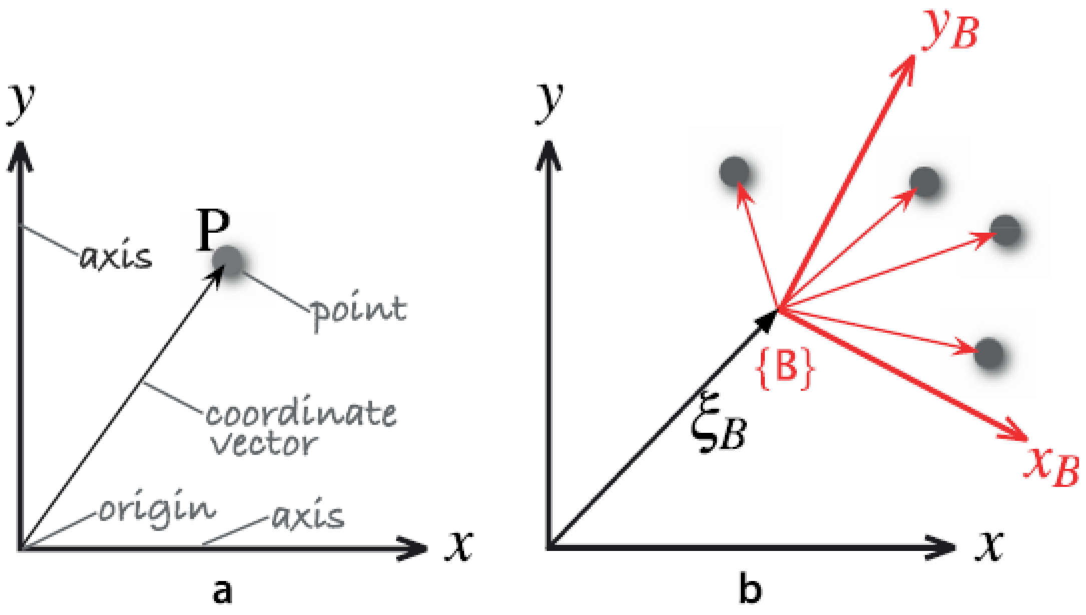

Figure 3.1: Point P is described by a coordinate vector with respect to an absolute coordinate frame.

${}^{A}_{B}\xi$ can be thought of as describing some motion –picking up {A} and applying a displacement and a rotation so that it is transformed to {B}. If the initial superscript is missing we assume that the change in pose is relative to the world coordinate frame denoted O.

The point P in Figure 3.2 can be described with respect to either coordinate frame. Formally it's expressed as:

$$p^{A} = {}^{A}_{B}\xi \cdot p^{B} \tag{3.1}$$

where the right-hand side expresses the motion from {A} to {B} and then to P. The operator · transforms the vector, resulting in a new vector that describes the same point but with respect to a different coordinate frame.

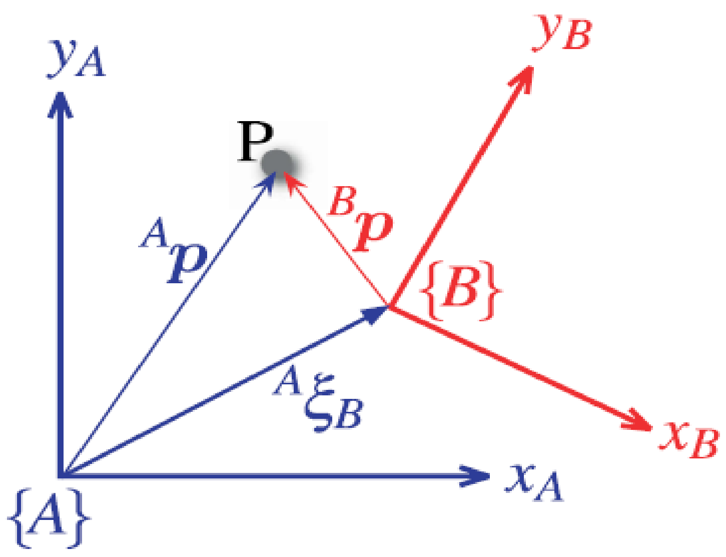

Figure 3.2: The pose of {B} relative to {A} is $^A_B\xi$

3.2.2 Representing pose in 2-dimention

Figure 3.7 shows a coordinate frame {B} that we wish to describe with respect to the reference frame {A}. It is clearly seen that the origin of {B} has been displaced by the vector t =(x, y) and then rotated counter-clockwise by an angle θ. A concrete representation of pose is therefore the 3-vector $^A_B\xi$ ~(x, y, θ), the symbol~denote that the two representations are equivalent.

The approach is to consider an arbitrary point P with respect to each of the coordinate frames and to determine the relationship between p^A and p^B. Referring again to Figure 3.3 the problem will be considered in two parts: rotation and then translation.

To consider just rotation a new frame {V} whose axes are parallel to those of {A} is created but whose origin is the same as {B} as in Figure 3.4 .Point **P** can be expressed with respect to {V} in terms of the unit-vectors that define the axes of the frame as follow:

$$\begin{pmatrix} x^V \\ y^V \end{pmatrix} = \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} x^B \\ y^B \end{pmatrix} \tag{3.1}$$

Which describes how points are transformed from frame {B} to frame {V} when the frame is rotated, this type of matrix is known as a rotation matrix and denoted by ${}^V_B R$. Eq.3.1 can be written as in Eq.3.2

$$\begin{pmatrix} x^V \\ y^V \end{pmatrix} = {}^V_B R \begin{pmatrix} x^B \\ y^B \end{pmatrix} \tag{3.2}$$

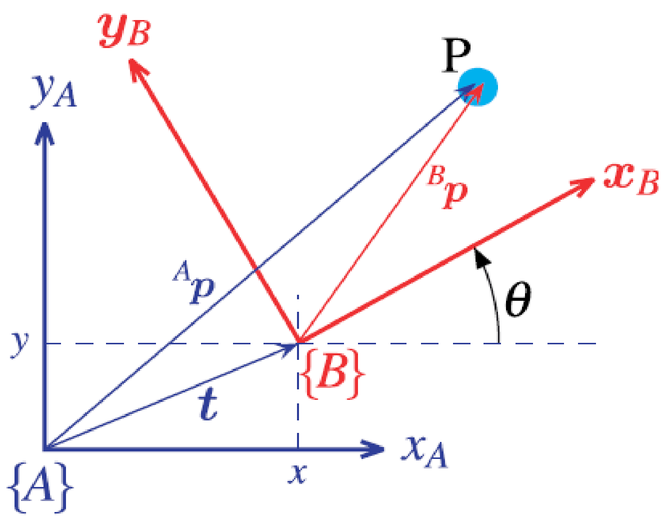

Figure 3.3: Relative pose between frame {A} and {B}

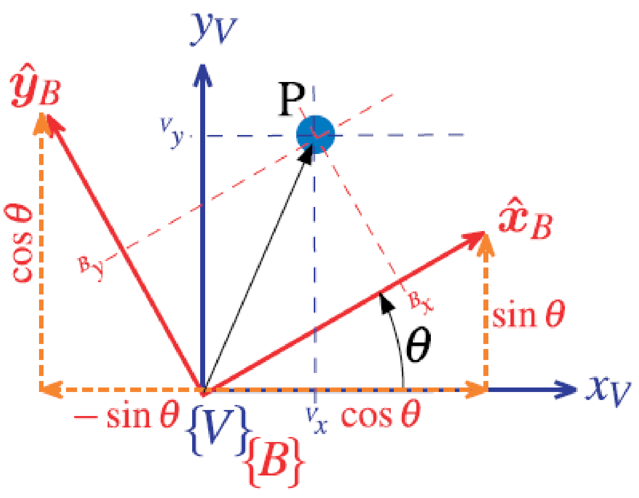

Figure 3.4: A new frame {V} parallel to {A} to account for rotation

The second part of representing pose is to account for the translation between the origins of the frames shown in Figure 3.3. Since the axes {V} and {A} are parallel this is simply vector addition

$$\begin{pmatrix} \cos\Theta & -\sin\Theta & x \\ \sin\Theta & \cos\Theta & y \end{pmatrix} \begin{pmatrix} x^B \\ y^B \\ 1 \end{pmatrix} \tag{3.3}$$

Or more compactly:

$$\begin{pmatrix} x^A \\ y^A \\ 1 \end{pmatrix} = \begin{pmatrix} {}^A_B R & t \\ 0_{1\times2} & 1 \end{pmatrix} \begin{pmatrix} x^B \\ y^B \\ 1 \end{pmatrix}$$

$$\tilde{P}^A = \begin{pmatrix} {}^A_B R & t \\ 0_{1\times2} & 1 \end{pmatrix} \tilde{P}^B \quad , \quad \tilde{P}^A = {}^A_B T \tilde{P}^B \tag{3.4}$$

where t =(x, y) is the translation of the frame, ${}^A_B\boldsymbol{R}$ is the rotation and ${}^A_B\boldsymbol{T}$ is referred to as a homogeneous transformation, which represent the combination of the translation and rotation.

It is clear now that the pose ${}^A_B\boldsymbol{\xi}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$ ~ $\begin{pmatrix} \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} & \boldsymbol{x} \\ \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} & \boldsymbol{y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{pmatrix}$

Defining the pose (position and orientation) of an object helps defining the robot position relative to different surrounding environment objects like cameras and different obstacles, as in Figure 3.5.

3.2.3 Representing pose in 3-dimention

Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis [3]. The 3-dimensional case is an extension of the 2-dimensional case discussed in the previous section. An extra coordinate axis is added, typically denoted by Z that is orthogonal to both the X and Y axes. The direction of the z-axis obeys the right-hand rule and forms a right-handed coordinate frame. Unit vectors parallel to the axes are denoted $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}$ such that $\hat{\boldsymbol{z}} = \hat{\boldsymbol{x}} \times \hat{\boldsymbol{y}}$.

A point P is represented by its X, Y and Z coordinates (X, Y,Z) or as a bound vector:

$$P = x\,\hat{\boldsymbol{x}} + y\hat{\boldsymbol{y}} + z\,\hat{\boldsymbol{z}} \qquad (3.5)$$

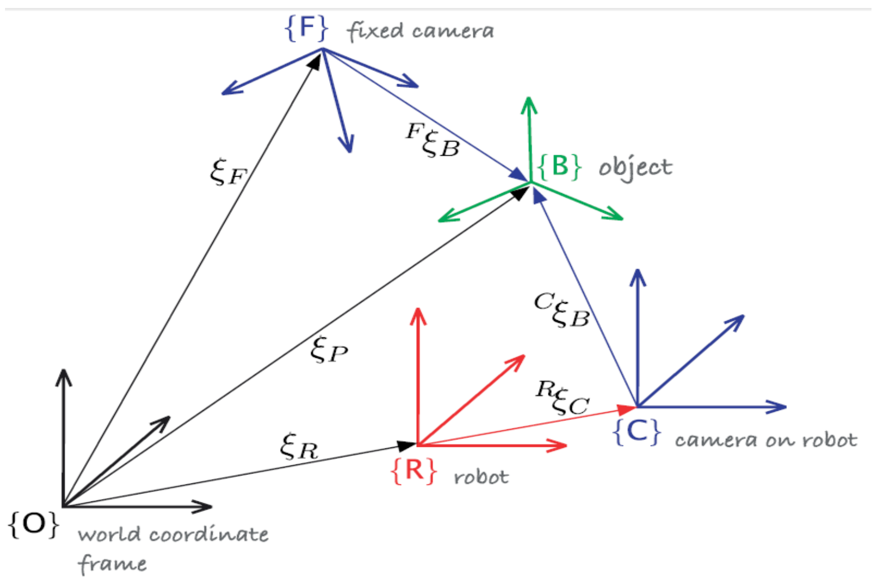

Figure 3.5: Multiple 3-dimensional coordinate frames and relative poses

The approach is to again to consider an arbitrary point P with respect to each of the coordinate frames and to determine the relationship between $\boldsymbol{p}^{\boldsymbol{A}}$ and $\boldsymbol{p}^{\boldsymbol{B}}$ .The problem will be considered in two parts: rotation and then translation.

Rotation is surprisingly complex for the 3-dimensional case, considering rotation about a single coordinate axis. Figure 3.6 shows a right-handed coordinate frame, and that same frame after it has been rotated by various angles about different coordinate axes.

The issue of rotation has some subtleties which are illustrated in Figure 3.7. This shows a sequence of two rotations applied in different orders. It's clearly seen that the final orientation depends on the order in which the rotations are applied. This is a deep and confounding characteristic of the 3-

dimensional world which has intrigued mathematicians for a long time. Mathematicians have developed many ways to represent rotation like: orthonormal rotation matrices, Euler and Cardan angles, rotation axis and angle, and unit quaternions.

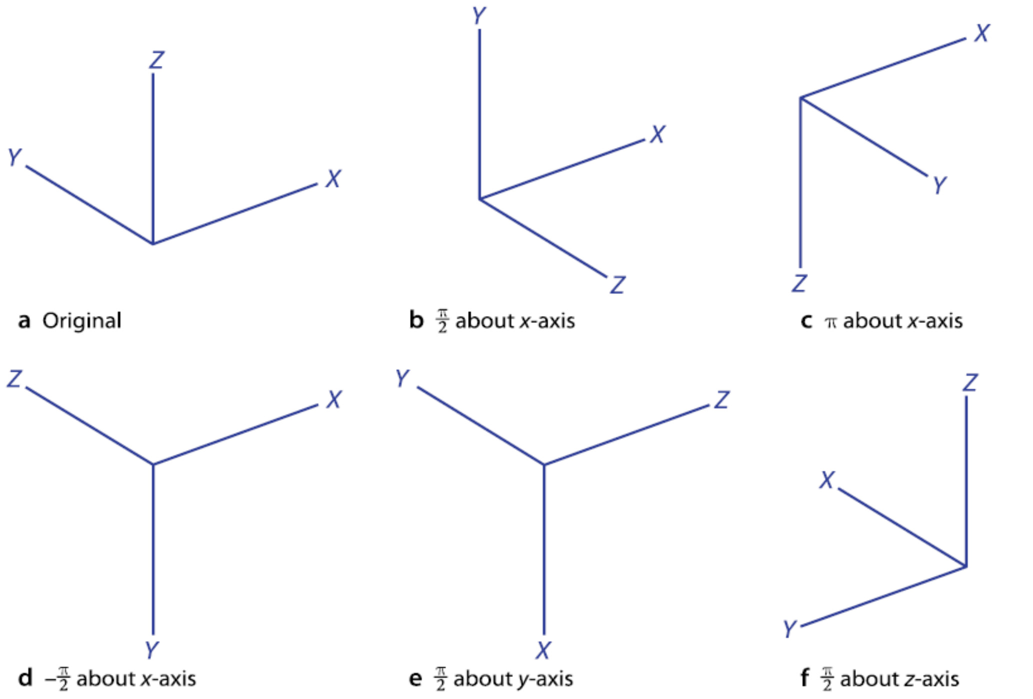
a Original
b $\frac{\pi}{2}$ about x-axis
c π about x-axis
d $-\frac{\pi}{2}$ about x-axis
e $\frac{\pi}{2}$ about y-axis
f $\frac{\pi}{2}$ about z-axis

Figure 3.6: Rotation of a 3D coordinate frame.

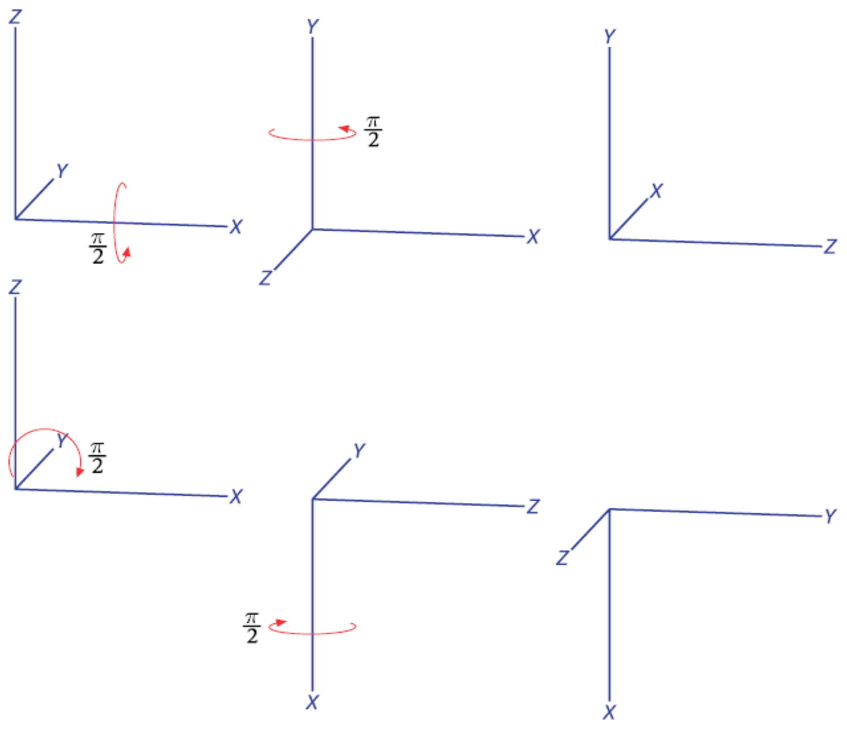

Figure 3.7: The non-commutivity of rotation

- **Orthonormal rotation matrices**

Just as for the 2-dimensional case the orientation of a coordinate frame can be represented by its unit vectors expressed in terms of the reference coordinate frame. Each unit vector has three elements and they form the columns of a 3×3 orthonormal matrics$\boldsymbol{{}^{A}_{B}R}$.

$$\begin{pmatrix} x^A \\ y^A \\ z^A \end{pmatrix} = {}^{A}_{B}R \begin{pmatrix} x^B \\ y^B \\ z^B \end{pmatrix} \tag{3.6}$$

which rotates a vector defined with respect to frame {B} to a vector with respect to {A}. The orthonormal rotation matrices for rotation of θ about the X, Y and Z axes are:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{3.7}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{3.8}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

Which describe the rotation around each of the principal axis by an angle $\boldsymbol{\theta}$.

- **Three Angle Representations**

Euler's rotation theorem states that any rotation can be represented by not more than three rotations about coordinate axes. This means that in general an arbitrary rotation between frames can be decomposed into a sequence of three rotation angles and associated rotation axes

Euler's rotation theorem requires successive rotation about three axes such that no two successive rotations are about the same axis. There are two classes

of rotation sequence: Eulerian and Cardanian, named after Euler and Cardano respectively. The Eulerian type involves repetition, but not successive, of rotations about one particular axis: XYX, XZX, YXY, YZY, ZXZ, or ZYZ. The Cardanian type is characterized by rotations about all three axes: XYZ, XZY, YZX, YXZ, ZXY, or ZYX. In common usage all these sequences are called Euler angles and there are a total of twelve to choose from.
If we take the ZYZ, the total rotation would be:

$$R = R_Z(\Phi)R_Y(\theta)R_Z(\psi) \tag{3.10}$$

Another widely used convention is the roll-pitch-yaw angle sequence angle:

$$R = R_X(\theta_r)\ R_y(\theta_p)\ R_z(\theta_y) \tag{3.11}$$

which are intuitive when describing the attitude of quadrotor . Roll, pitch and yaw (also called bank, attitude and heading) refer to rotations about the X, Y, Z axes, respectively. This XYZ angle sequence, technically Cardan angles, is also known as Tait-Bryan angles or nautical angles. Generally for aerospace and ground vehicles the x-axis is commonly defined in the forward direction, z-axis downward and the y-axis to the right-hand side.

So either the orthonormal rotation matrices or there angle representations a different rotation matrix will be obtained. The rotation matrix is then combined with the translation part to yield the transformation matrix, which contain all the information needed to describe the position and orientation of the quadrotor with respect to the world coordinate frame.

3.3 Quadcopter Dynamics

To start analyzing the dynamics, two coordinate frames are defined. One attached to the moving robot, and the other, the inertial coordinate system as in Figure 3.8.

b1, b2, b3 constitute the set of unit vectors that describe the body fixed coordinate system, and likewise a1, a2, a3 describe a coordinate system that's fixed to the inertial frame. There are four rotors, each of which is independently actuated. r is the position vector of the center of mass. The Z-X-Y convention will be used. The first rotation about the Z axis through psi (***ψ***), the second rotation about the X axis through phi (***Φ***), this is the roll angle, and finally, the pitch about the Y axis through theta (***θ***).

Looking at the external forces and moments that act on the airframe, the sum of the forces is obtained by adding up the thrust vectors and the gravity vector. The sum of the moments is obtained by adding up the reaction moments, as well as the moments of the truss forces. The notation for the quadrotor model is shown in Figure 3.9. The body-fixed coordinate frame {B} has its z-axis downward following the aerospace convention.

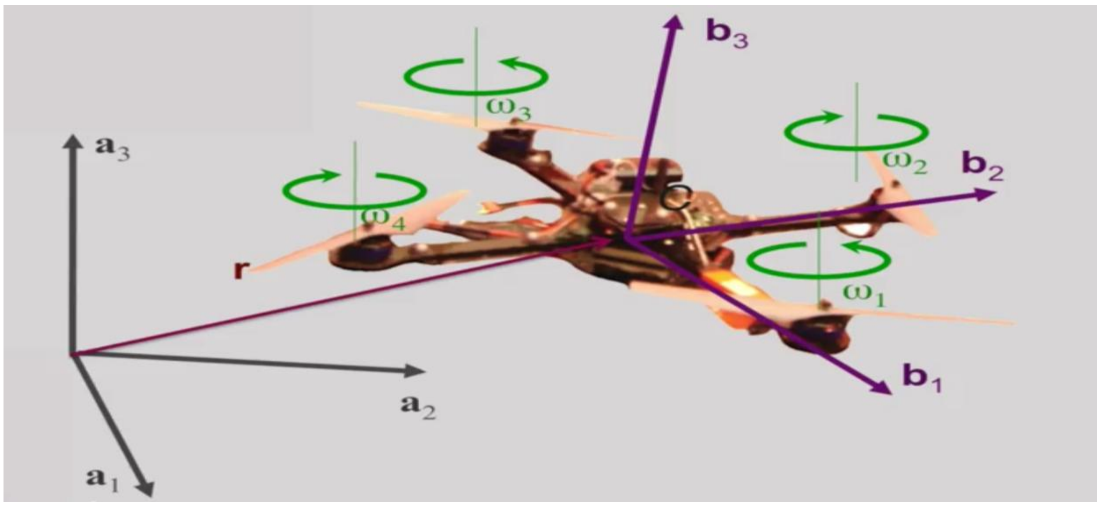

Figure 3.8: Body frame and inertial frame of the quadrotor

The quadrotor has four rotors, labeled 1 to 4, mounted at the end of each cross arm. The rotors are driven by electric motors powered by electronic speed controllers. Some low-cost quadrotors use small motors and reduction gearing to achieve sufficient torque.

3.3.1 Acting force calculations

The rotor speed is $\boldsymbol{\omega_i}$ and the thrust is an upward vector in the vehicle's negative z-direction, where b > 0 is the lift constant that depends on the air density, the cube of the rotor blade radius, the number of blades, and the chord length of the blade.

$$T_i = b\omega^2{}_i \quad , \text{i} = 1,2,3,4 \qquad (3.12)$$

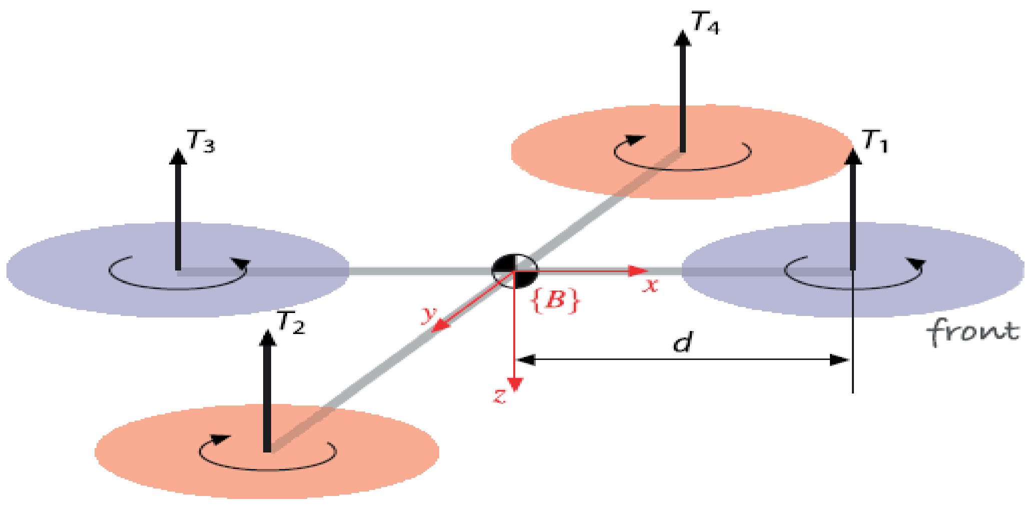

Figure 3.9: Quadrotor notation showing the four rotors, their thrust vectors and directions of rotation.

Summing over all the motors, it is found that the total thrusts on the quadcopter (in the body frame) is given by:

$$T_B = \sum_{i=1}^{4} T_i \; = \begin{bmatrix} 0 \\ 0 \\ \sum \omega^2{}_i \end{bmatrix} \qquad (3.13)$$

3.3.2 The net torques

Once the forces on the quadcopter have been computed, the acting torque should also be determined. Each rotor contributes some torque about the body z axis. This torque is the torque required to keep the propeller spinning and providing thrust; it creates the instantaneous angular acceleration and overcomes the frictional drag forces. Three different torques are generated as follow:

- The torque about the vehicle's x-axis, the rolling torque, is

$$\tau_x = \mathrm{db}({\omega^2}_4 - {\omega^2}_2) \tag{3.14}$$

- The torque about the vehicle's y-axis, the pitching torque, is

$$\tau_y = \mathrm{db}({\omega^2}_1 - {\omega^2}_3) \tag{3.15}$$

- The torque about the vehicle's z-axis, the yawing torque:

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k({\omega^2}_1 + {\omega^2}_3 - {\omega^2}_2 - {\omega^2}_4) \tag{3.16}$$

where the different signs are due to the different rotation directions of the rotors. A yaw torque can be created simply by appropriate coordinated control of all four rotor speeds. The torques in the body frame:

$$\tau_B = \begin{pmatrix} db({\omega^2}_1 - {\omega^2}_3) \\ db({\omega^2}_4 - {\omega^2}_2) \\ k({\omega^2}_1 + {\omega^2}_3 - {\omega^2}_2 - {\omega^2}_4) \end{pmatrix} \tag{3.17}$$

3.3.3 Equations of motion

In the inertial frame, the acceleration of the quadcopter is due to thrust, gravity, and linear friction. The thrust vector in the inertial frame can be obtained by using rotation matrix R to map the thrust vector from the body frame to the inertial frame. Thus, the translational dynamics of the vehicle in world coordinates is given by Newton's second law:

$$m\dot{v} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} - {}^{0}_{B}R \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} \tag{3.18}$$

where $\boldsymbol{v}$ is the velocity of the vehicle in the world frame, g is gravitational acceleration, m is the total mass of the vehicle and $\boldsymbol{T}$ is the total upward thrust. The first term is the force of gravity which acts downward in the world frame and the second term is the total thrust in the vehicle frame rotated into the world coordinate frame.

While it is convenient to have the linear equations of motion in the inertial frame, the rotational equations of motion are useful in the body frame, so that rotations about the center of the quadcopter can be expressed instead of about the inertial center. Deriving the rotational equations of motion from Euler's equations for rigid body dynamics expressed in vector form, Euler's equations can be written as:

$$J\dot{\omega} = -\omega \times J\omega + \Gamma \tag{3.19}$$

Where J is a 3 ×3 inertia matrix of the vehicle. ω is the angular velocity vector and Γ = $(\tau_x, \tau_y, \tau_z)^T$, τ is the torque applied to the airframe according to Eq. 3.14 to 3.16

The quadrotor can be modeled as two thin uniform rods crossed at the origin with a point mass (motor) at the end of each as in figure 3.9. With this in mind, it's clear that the symmetries result in a diagonal inertia matrix of the form:

$$J = \begin{pmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{pmatrix} \tag{3. 20}$$

The motion of the quadrotor is obtained by integrating the forward dynamics. Eq. 3.18 and Eq. 3.19 where the forces and moments on the airframe are functions of the rotor speeds.

$$\begin{pmatrix} T \\ \Gamma \end{pmatrix} = \begin{pmatrix} -b & -b & -b & -b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{pmatrix} \begin{pmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \end{pmatrix} = A \begin{pmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \end{pmatrix} \tag{3.21}$$

The matrix A is of full rank if b, k, d > 0 and can be inverted to give the rotor speeds required to apply a specified thrust T and moment Γ to the airframe.

$$\begin{pmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \end{pmatrix} = A^{-1} \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \tag{3.22}$$

3.3.4 Quadrotor parameters determine

Eq. 3.22 is the equation of motion to be used in the six degree-of-freedom quadrotor model. However, Quadrotor parameters b, k, d and I Matrix must be determined.

The moments of inertia for the quadrotor are calculated assuming a spherical dense center with mass M and radius R, and point masses of mass m located at a distance of l from the center as shown in Figure 3.10.

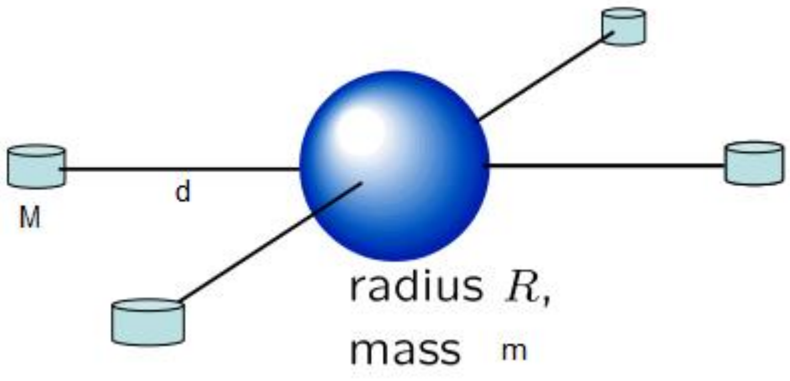

Figure 3.10: Simplified mechanical model for the inertia matrices

The inertia for a solid sphere is given by $J = 2MR^2/5$. Therefore

$$j_{xx} = \frac{2MR^2}{5} + 2d^2m \tag{3.23}$$

$$j_{yy} = \frac{2MR^2}{5} + 2d^2m \tag{3.24}$$

$$j_{zz} = \frac{2MR^2}{5} + 4d^2m \tag{3.25}$$

Where M is the mass of the sphere (body without the motors), m is the mass of the motor plus its propeller, d is distance from the center of the motor to the center of the sphere.

The parameters were experimentally obtained in table 3.1

Table 3.1: Quad rotor experimentally obtained Parameters

Parameter	Value
G	9.81 m/s^2
M	1.04 kg
R	7 cm
d	16.5 cm
j_{xx}	6*10^{-3} $kg.m^2$
j_{yy}	6*10^{-3} $kg.m^2$
j_{zz}	12*10^{-3} $kg.m^2$

The two constant b and k were not experimentally obtained but were obtained by setting random values > 0 and then observing and comparing the system performance in simulation and hardware real time, the approximated values were found to be : k = 3×10^{-7} , b = 10^{-7}.

3.4 Control Design

The purpose of deriving a mathematical model of a quadcopter is to assist in developing controllers for physical quadcopters. The inputs to the system consist of the angular velocities of each rotor, since the voltages across the motors are that can be controlled. Note that in the simplified model, only the square of the angular velocities, $\boldsymbol{\omega^2}_i$ is used and never the angular velocity itself, $\boldsymbol{\omega}$. For notational simplicity, the inputs $\boldsymbol{\lambda}_i = \boldsymbol{\omega^2}_i$ is introduced. If $\boldsymbol{\omega^2}_i$ can be set, clearly $\boldsymbol{\lambda_i}$ can be set as well. With this, the system can be written as a first order differential equation in state space.

Let $\boldsymbol{x_1}$ be the position in space of the quadcopter, $\boldsymbol{x_2}$ be the quadcopter linear velocity, $\boldsymbol{x_3}$ be the roll, pitch, and yaw angles, and $\boldsymbol{x_4}$ be the angular velocity vector. System state space equations are:

$$\dot{x_1} = x_2 \tag{3.26}$$

$$\dot{x_2} = \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} + \frac{1}{m} R\, \tau_B \tag{3.27}$$

$$\dot{x_3} = \begin{pmatrix} 1 & 0 & -s_\theta \\ 0 & c_\theta & c_\theta s_\Phi \\ 0 & -s_\Phi & c_\theta c_\Phi \end{pmatrix}^{-1} x_4 \tag{3.28}$$

C is cosine , S is sine

$$\dot{x_4} = \begin{pmatrix} \tau_\Phi\, {I_{xx}}^{-1} \\ \tau_\theta\, {I_{yy}}^{-1} \\ \tau_\psi\, {I_{zz}}^{-1} \end{pmatrix} - \begin{pmatrix} \frac{(I_{yy}-I_{zz})\omega_y\omega_z}{I_{xx}} \\ \frac{(I_{zz}-I_{xx})\omega_x\omega_z}{I_{yy}} \\ \frac{(I_{xx}-I_{yy})\omega_x\omega_y}{I_{zz}} \end{pmatrix} \tag{3.29}$$

3.5 PID Controller

In order to stabilize the quadcopter, PID controller algorithm will be used, with a component proportional to the error between the desired trajectory and the observed trajectory, a component proportional to the derivative of the error and a component proportional to the integral of the error. The quadcopter under design will only have a gyro, so only the angle derivatives $\dot{\boldsymbol{\Phi}}$, $\dot{\boldsymbol{\theta}}$ and $\dot{\boldsymbol{\psi}}$ will be used as a feedback to the controller; these measured values will give us the derivative of our error, and their integral will provide us with the actual error. It is required to stabilize the quadrotor in a horizontal position, so the desired velocities and angles will all be zero. Torques are related to the angular velocities by $\boldsymbol{\tau} = \boldsymbol{I}\ddot{\boldsymbol{\theta}}$, so it is required to set the torques proportional to the output controller, with $\boldsymbol{\tau} = \boldsymbol{I}\boldsymbol{u}(\boldsymbol{t})$. Thus,

$$\tau_B = \begin{pmatrix} db(\lambda_1 - \lambda_3) \\ db(\lambda_4 - \lambda_2) \\ k(\lambda_1 + \lambda_3 - \lambda_2 - \lambda_4) \end{pmatrix} =$$

$$\begin{pmatrix} -I_{xx}(k_d\dot{\Phi} + k_p\int_0^T \dot{\Phi} + k_i\int_0^T\int_0^T \dot{\Phi}) \\ -I_{yy}(k_d\dot{\theta} + k_p\int_0^T \dot{\theta} + k_i\int_0^T\int_0^T \dot{\theta}) \\ -I_{zz}(k_d\dot{\psi} + k_p\int_0^T \dot{\psi} + k_i\int_0^T\int_0^T \dot{\psi}) \end{pmatrix} \tag{3.30}$$

The thrust is proportional to a weighted sum of the inputs:

$$T = \frac{mg}{cos(\Theta)cos\,(\Phi)} = b\sum\lambda_i \xrightarrow{yields} \quad \sum\lambda_i = \frac{mg}{bcos(\Theta)\cos\,(\Phi)} \tag{3.31}$$

Solving for each λ_i, the following input values are obtained:

$$\lambda_1 = \frac{\mathrm{mg}}{4\mathrm{bcos}(\Theta)\cos\,(\Phi)} - \frac{2\mathrm{k}e_{\Phi}I_{xx} + e_{\psi}I_{zz}\mathrm{db}}{4\mathrm{kdb}} \tag{3.32}$$

$$\lambda_2 = \frac{\mathrm{mg}}{4\mathrm{bcos}(\Theta)\cos\,(\Phi)} + \frac{e_{\psi}I_{zz}}{4\mathrm{k}} - \frac{e_{\Theta}I_{yy}}{2\mathrm{db}} \tag{3.33}$$

$$\lambda_3 = \frac{\mathrm{mg}}{4\mathrm{bcos}(\Theta)\cos\,(\Phi)} - \frac{-2\mathrm{k}e_{\Phi}I_{xx} + e_{\psi}I_{zz}\mathrm{db}}{4\mathrm{kdb}} \tag{3.34}$$

$$\lambda_4 = \frac{mg}{4b\cos(\Theta)\cos(\Phi)} + \frac{e_{\psi} I_{zz}}{4k} + \frac{e_{\Theta} I_{yy}}{2db} \tag{3.35}$$

The block diagram in Figure 3.11 bellow shows the internal structure of the PID controller used for stability; the three angular velocities are feedback signals that are compared to a set point to determine the status that the system should apply

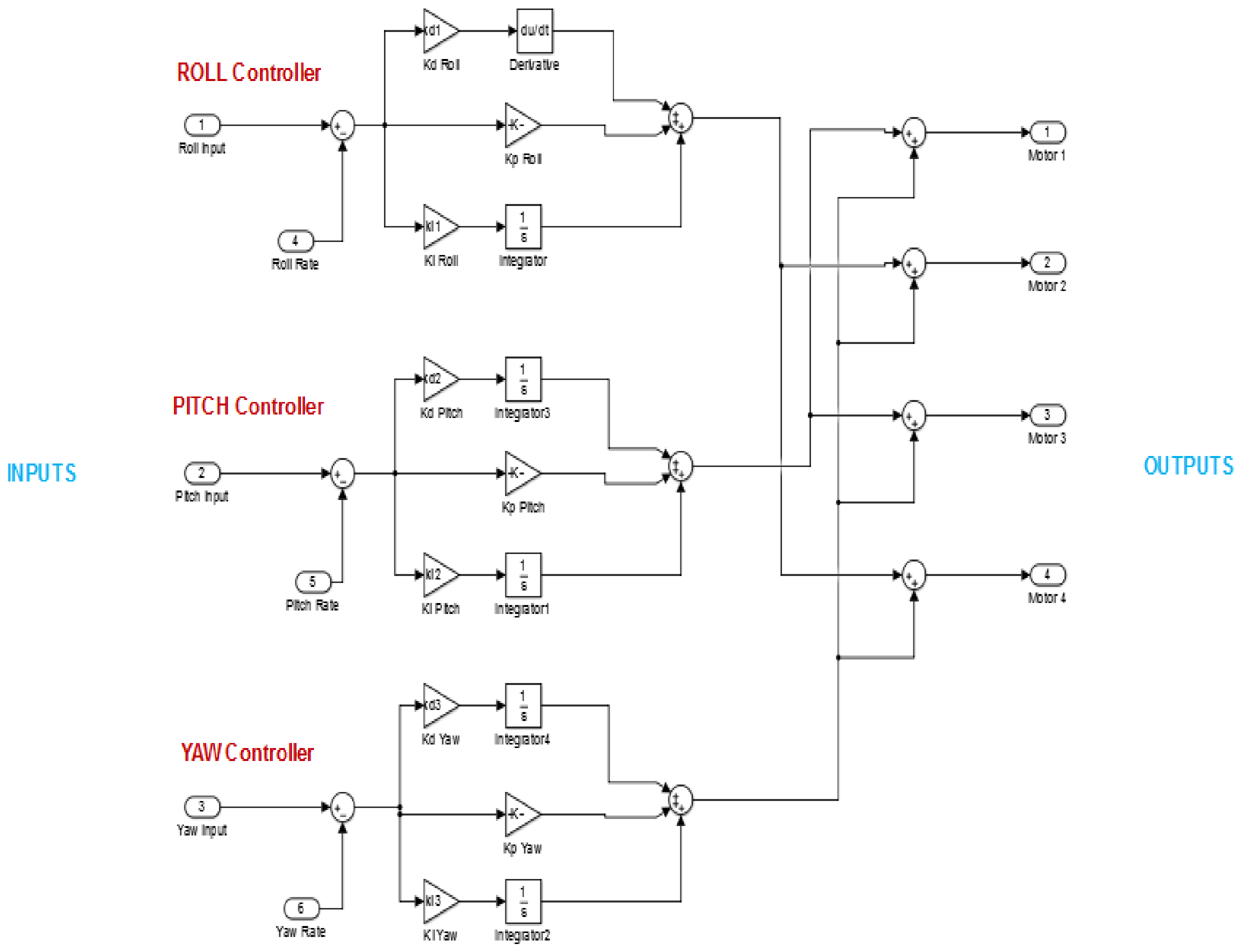

Figure 3.11: PID controller block diagram

CHAPTER FOUR

QUADROTOR IMPLEMNTAION and SIMULATION

4.1 System Simulation

The equations of motion describing the quadrotor behavior have been completed, it is now possible to create a simulation environment in which to test and view the results of various inputs and controllers. Although more advanced methods are available, a simulator which utilizes Euler's method for solving differential equations to evolve the system state will be used In MATLAB. Codes for MATLAB simulations are available in Appendix B.

4.1.1 Angle simulation

Figure 4.1 shows the Roll, Pitch and yaw angles for the status of the quadrotor and as it is clear from the graph the PID controller tends to drive the three angles to zero in order to stabilize the robot.

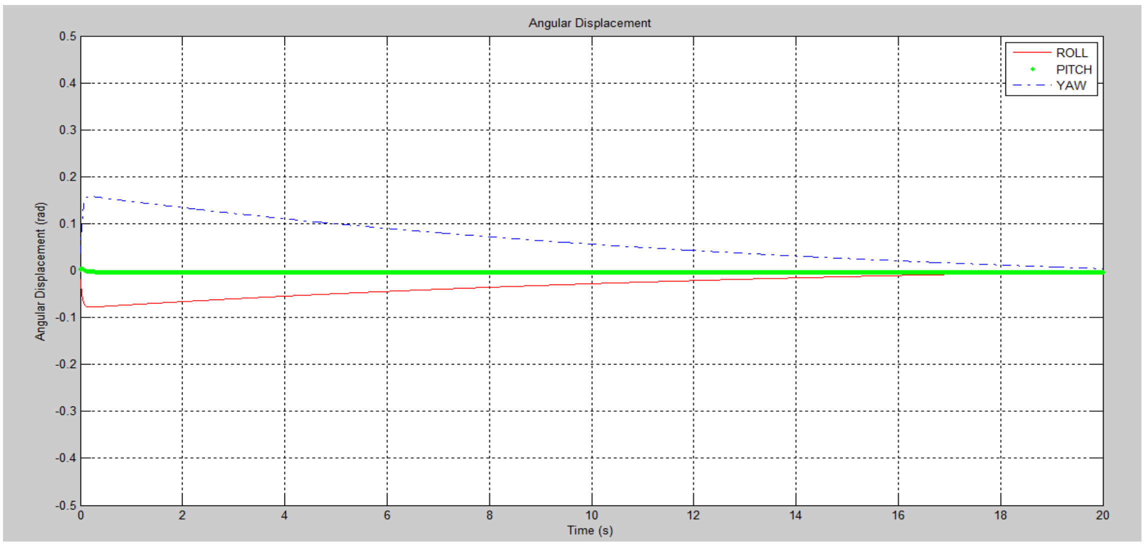

Figure 4.1: PID sets the three angles to zero

4.1.2 Angular velocity simulation

Figure 4.2 shows the angular velocity of the quadrotor (Roll, Pitch and Yaw rates). The controller tends to drive the angular velocity (in all three axes) to zero in order to stabilize the robot

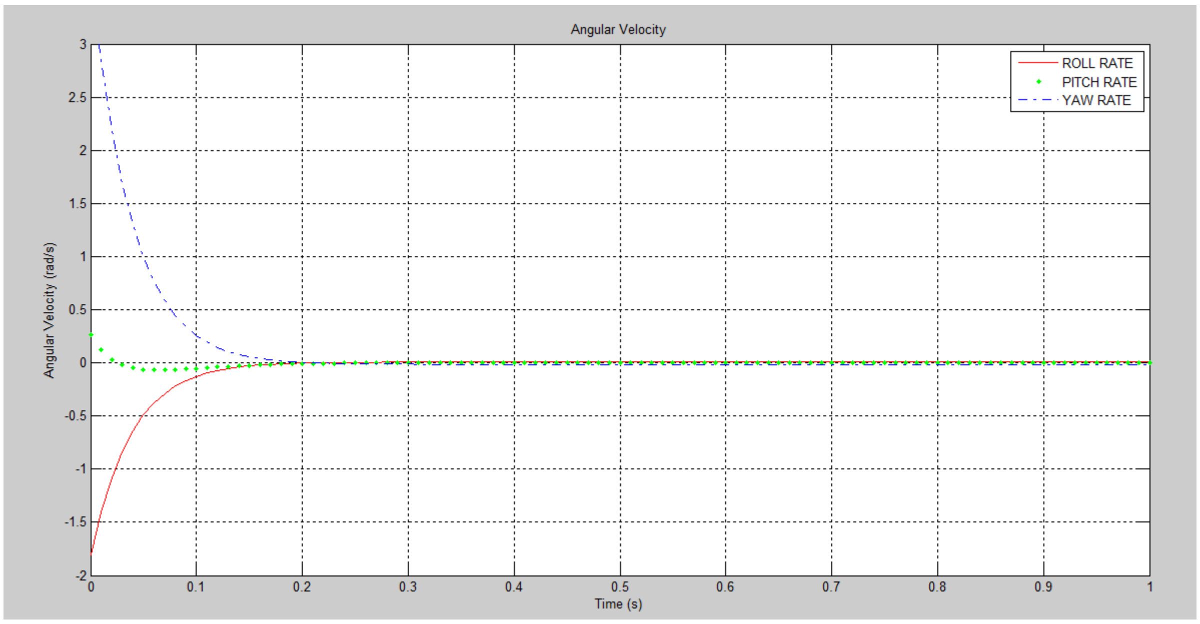

Figure.4.2: PID sets the three angular velocities to zero.

4.1.3 Flight simulator

Reading the forces (thrusts) acting on the quadrotor body it is possible to draw the quadrotor in a three-dimensional visualization which is updated as the simulation is running as in Figure 4.3 bellow:

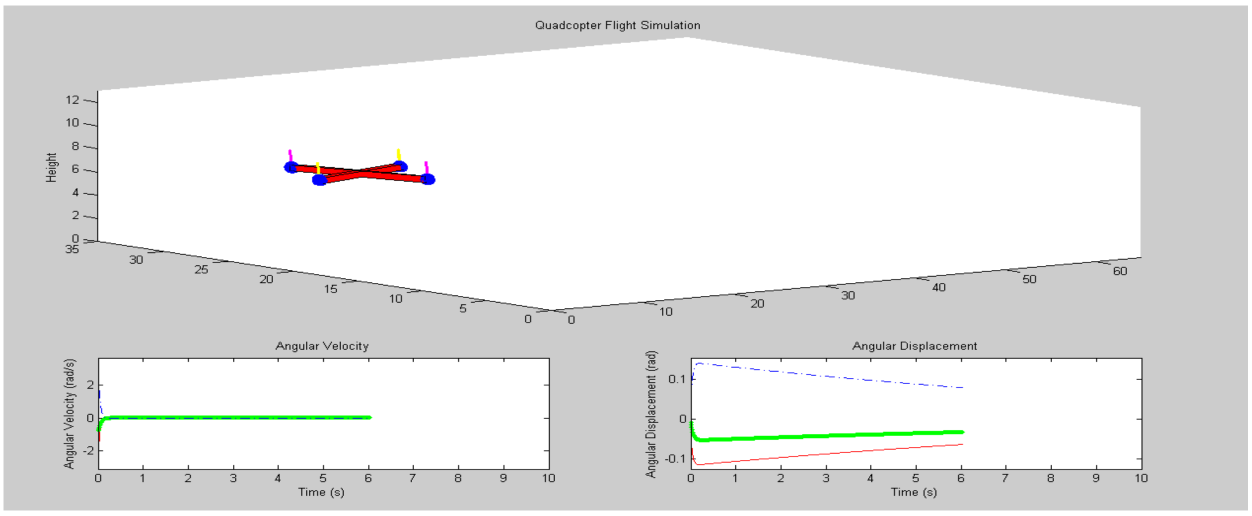

Figure 4.3: Quadrotor visual simulation.

4.2 System Practical Model

System practical model is separated into two parts: the mechanical parts which consist of frame, arms and propellers. And electrical part which consists of Arduino UNO, Electronic speed controller, Lithium Polymer battery, brushless motor, propellers, transmitter and receiver, and the inertial measurement unit.

4.1.1 System mechanical part

Selection of appropriate material for a mechanical part is an essential element of all engineering projects. The main mechanical parts of the system are the frame, arms and propellers as in Figure 4.4 bellow:

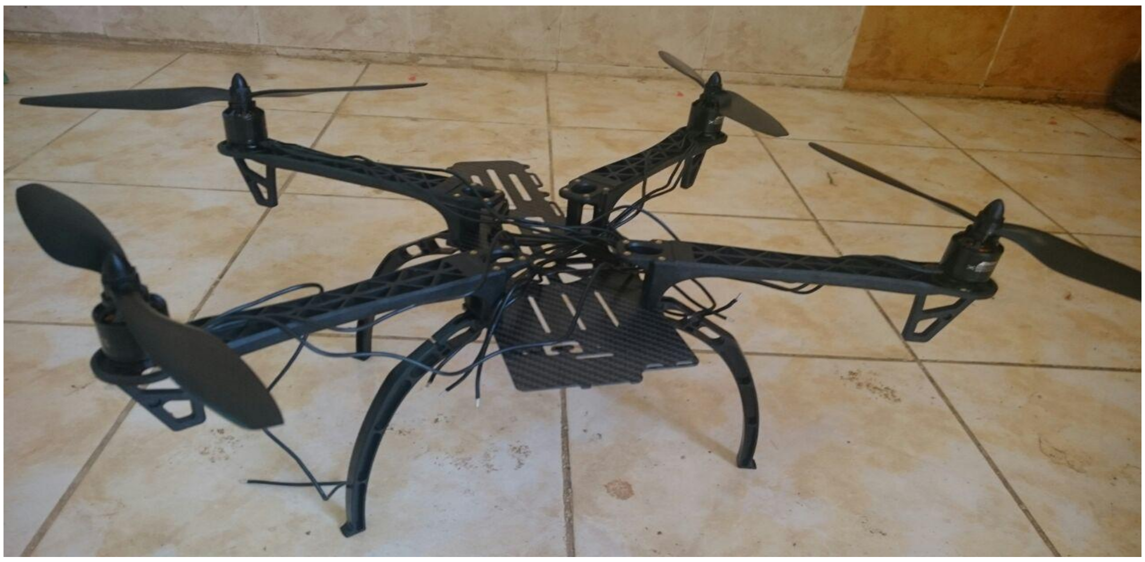

Figure 4.4: Assembled mechanical frame.

i. Frame

In considering the frame, the first consideration is the material to be used. It must be lightweight, sturdy, and affordable. The forces which act on the aircraft primarily will be gravity and air pressure. Gravity allows for construction under the guidance of a limited mass to allow for structural stability on the ground, as well as control of the quadrotor in the air. Air pressure, which is used to determine the airspeed, will affect the quadrotor’s stress on the screws at higher altitudes. The higher the altitude, the lighter the air, the smaller the forces against the frame, which implies the quadrotor's frame, is being stretched. This is what is kept in mind when considering for the base material for our aircraft. For the project, three materials are possibilities due to their popularity: wood, aluminum and plastic-carbon fiber.

Plastic carbon fiber was the best choice. With plastic, tests flight can be performed repeatedly, and modifications can be easily made on it. Furthermore, due to its increased strength to stress, plastic is less likely to bend due to take-off or stable flight; also, it carries a stronger stability to the frame. Figure 4.5 shows unassembled frame components.

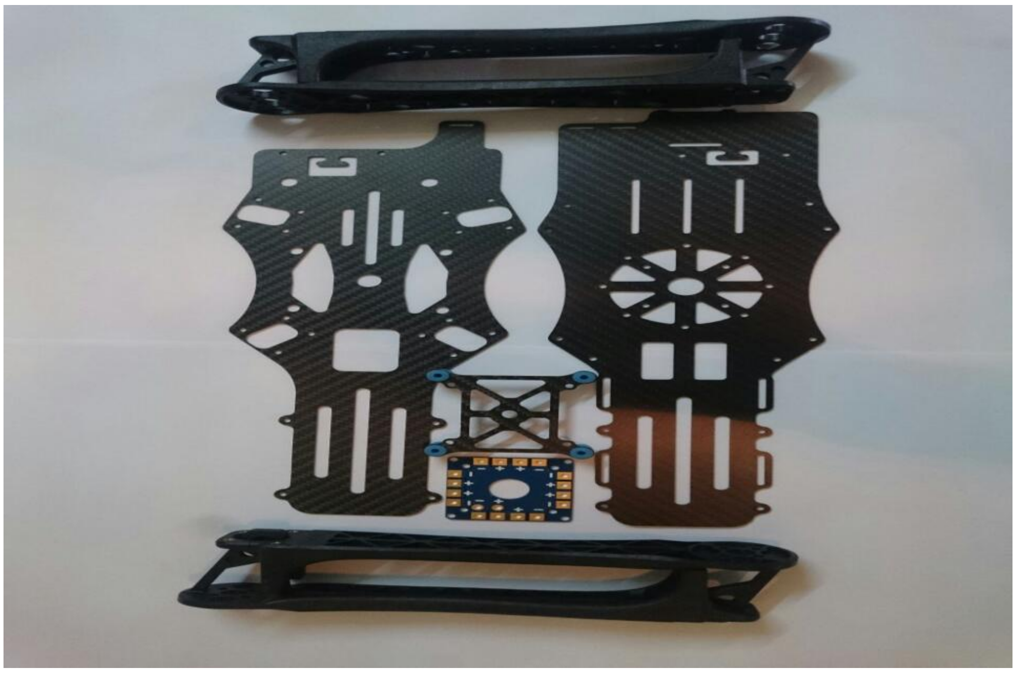

Figure 4.5: Carbon – Fiber quadrotor frame

Plastic is less weight from the other material to meet the minimum quadrotor requirements. The lightweight frame must be designed to support all the quadrotor subsystem.

ii. Propeller

Quadrotors use two clockwise (CW) and two counter clockwise (CCW) propellers. Propellers are classified by length and pitch. For example 10×4.5 propellers are 10 inch long and have a pitch of 4.5 inch.

Generally, increased propeller pitch and length will draw more current. Also the pitch can be defined as the travel distance of one single prop rotation. In a nutshell, higher pitch means slower rotation, but will increase vehicle's speed which also uses more power.

- **comparison between high pitch and low pitch props**

  Generally a propeller with low pitch numbers can generate more torque. The motors do not need to work as hard so it pulls less current with this type of propeller. Lower pitch propellers will also improve stability. A higher pitch propeller moves greater amount of air, which could create turbulence and cause the aircraft to wobble during hovering.

- **comparison between small length and large length props**

  When it comes to the length, propeller efficiency is closely related to the contact area of a propeller with air, so a small increase in propeller length will increase the propeller efficiency. A smaller propeller is easier to stop or speed up while a larger propeller takes longer to change speeds (inertia of movement). Smaller propeller also means it draws less current; that is why Hexacoptors and Octacopters tend to use smaller propellers than quadrotor of similar size.

  Increased propeller pitch and length will draw more current and the high current effect on the motors and increase their heat quickly; high KV motors need small propellers to work efficiently. Low KV motors need a large propeller to work efficiently**,** If a large propeller is used on a high KV motor, it will draw a lot of current and become so hot quickly **,** If a small propeller on a low KV motor is used **,** it will pull a small weight and decrease stability. With a well balanced motor and propeller combination, the quadrotor should achieve great efficiency, not only improve battery life time, but also allows great user control experience.

Figure 4.6: 10 × 4.5 propeller

4.1.2 System electrical part

The electrical circuit consists of Arduino UNO, ESC, LiPo battery, brushless DC motor, transmitter and receiver

i. Arduino Uno

Arduino Uno as shown in Figure 4.3 is a microcontroller board based on ATmega 328P microcontroller. It has 14 digital input/output pins (of which 6 can be used as pulse width modulation (PWM) outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, header and a reset button. Using PID library in Arduino Uno code (See Appendix C) as a PID controller has many benefits:

- There are many ways to write the PID algorithm. A lot of time was spent making the algorithm in this library as solid as any found in industry.
- When using the library all the PID code is self-contained. This makes your code easier to understand. It also lets you do more complex stuff, like having 8 PIDs in the same program.

Figure 4.7: Arduino Uno

iii. Electronic speed controller(Brushless ESC 40A)

This is a solid Electronic Speed Controller or ESC, shown in Figure 4.4, ideal for use with quadrotor. It can support a max of 30 Amp continuous output to handle both large and small motors. This ESC used with Configuration to perform a throttle calibration with all motors at once with quadrotor.

Specifications:

The Specifications of Esc is

- Cont Current:30A
- Burst Current: 55A
- BEC Mode: Linear
- BEC : 5v / 3A
- Lipo Cells: 2-6
- NiMH : 5-18
- Weight: 33g
- Size: 55x28x13mm

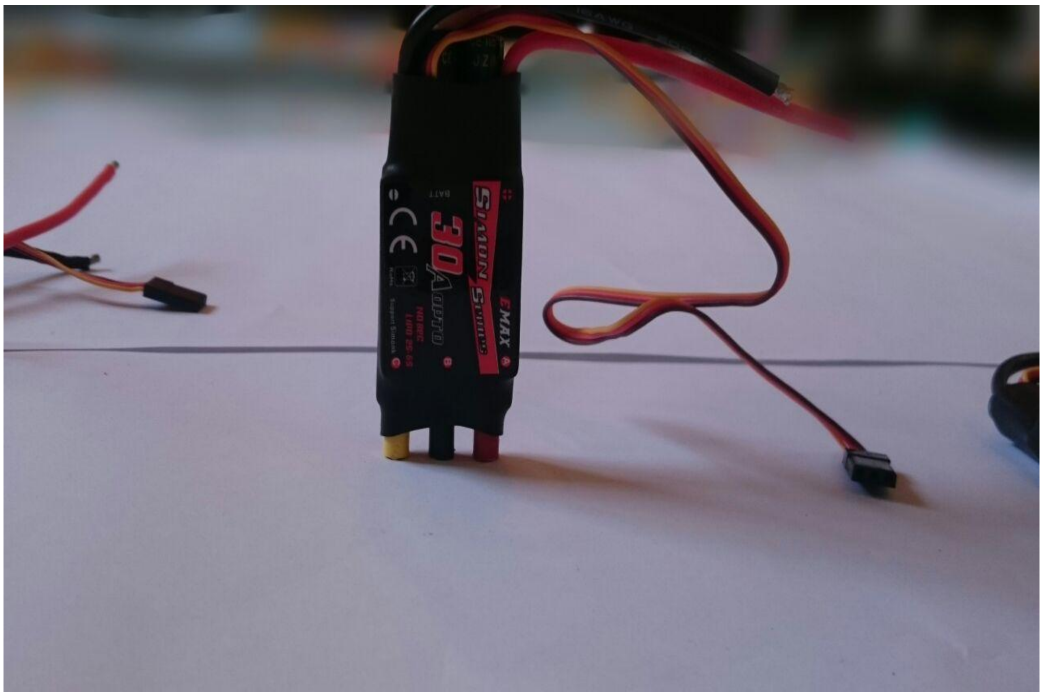
Figure 4.8: Electronic speed controller (ESC) 30A.

iv. Lithium-Polymer Battery Series

Lithium-polymer batteries offer a variety of significant advantages over NiCd, NiMH and Li-Ion batteries for use in R/C electric devices. It is very important to have a good understanding of the operating characteristics of LiPo batteries - especially how to charge and care for them safely. Always read the specifications printed on the battery's label and this instruction sheet in their entirety prior to use. Failure to follow these instructions can quickly result in severe, permanent damage to the batteries and its surroundings and even start a Fire. Before and after every use of the LiPo battery, all cells should be inspected to ensure no physical damage or swelling is evident. Such signs can often indicate a dangerous problem exists with the battery that could lead to failure.

Specifications

- Minimum Capacity: 5200mAh
- Configuration: 14.8v / 4Cell
- Constant Discharge rate: 30C

- Pack Weight: 361g
- Pack Size: 136x 42 x 30mm

- **Voltage Cells:**

While the batteries exact voltage may not be printed on the battery but it will tell how many cells the battery has. LiPo batteries are made up of cells. Each cell is 3.5 volts, for example the battery shown in Figure 4.9 is a 4s battery. This means that is has 4cells, which would give it a total voltage of: 3.7x4=14.8v

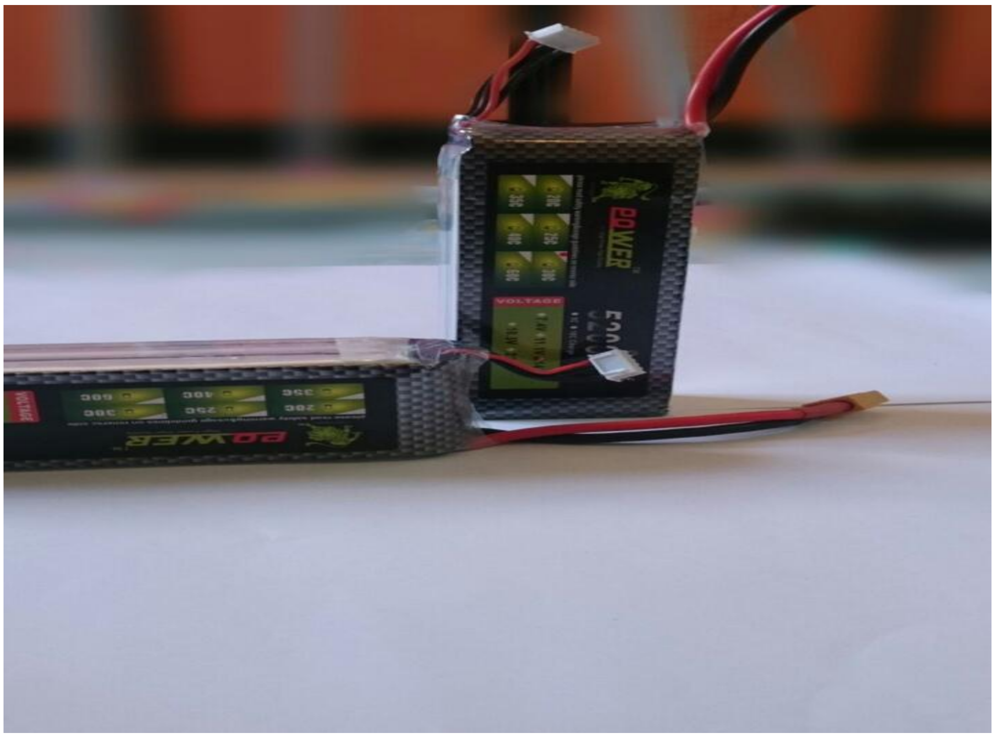

Figure 4.9: Lipo battery 5200 mah

v. Brushless DC Motor

Brushless DC Motor's stator comprises steel laminations, slotted axially to accommodate an even number of windings along the inner periphery. The rotor is constructed from permanent magnets with from two-to-eight N-S pole pairs. The BLDC motor's electronic commutator sequentially energizes the stator coils generating a rotating electric field that drags the rotor around with it. Efficient operation is achieved by ensuring that the coils are energized at precisely the right time. Brushless motors have neither commutator nor brushes.

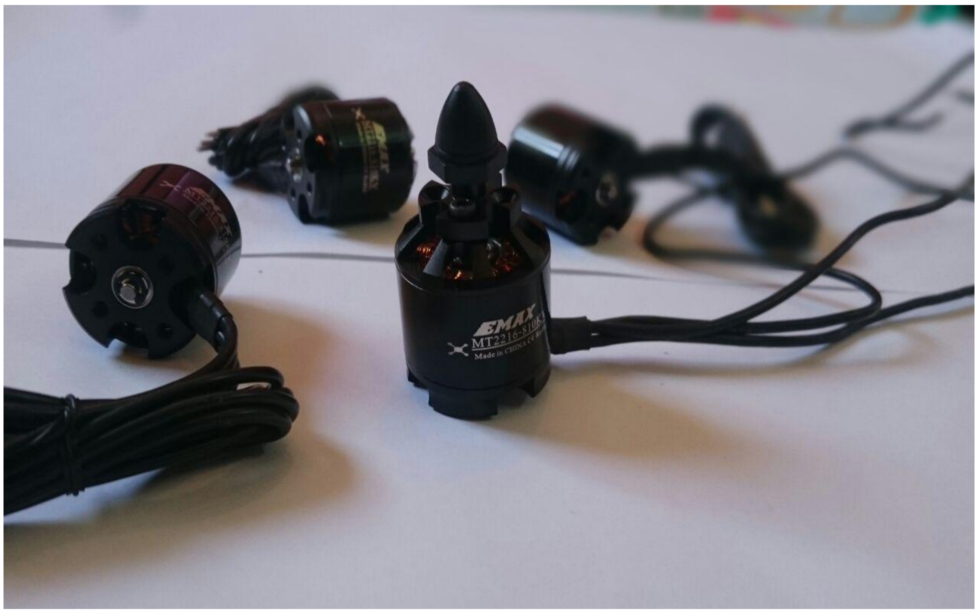

Figure 4.10: Brushless DC motor.

vi. Transmitter and Receiver

The receiver is what goes into your aircraft and controls the servos and motor(s). You can see from this receiver that it is a 5 channel receiver (Aileron, Elevator, Throttle, Rudder, and Aux). The BAT slot is not considered a channel. The receiver above connects wirelessly to the transmitter using a 2.4 GHz frequency. 2.4 GHz frequency is the standard frequency for RC planes. The receiver runs by 5v, and sends signals to the servos to turn them. It also sends a signal to the ESC to tell it how fast the run the motor. The receiver gets its 5 volts from gets the ESC's or battery elimination circuit.

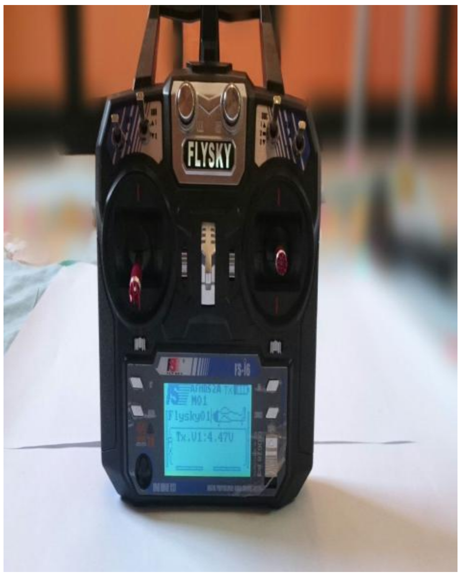
Figure 4.11: Fly sky 6 channel transmitter

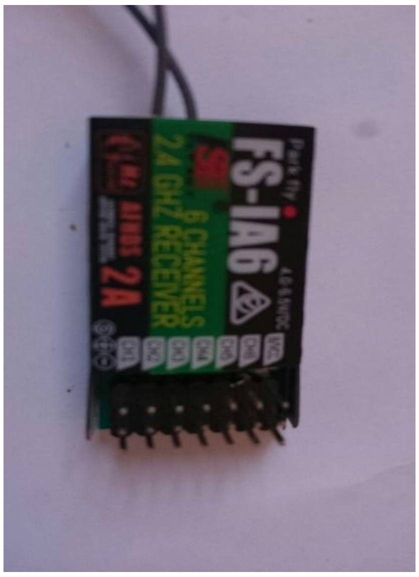
Figure 4.12: 6 channel receiver

vii. Inertial measurement unit

The inertial measurement unit (MPU-6050) shown on Figure 4.13 is a 6 degrees of freedom (6 DOF) sensor. The unit include three independent sensor:

- 3 Axis accelerometer that measure linear acceleration in all three axis
- 3 Axis gyroscope that measure angular velocity in all three axis
- Temperature sensor for compensation purposes

The IMU uses the I2C communication protocol to send readings to the main controller via the interrupt pins.

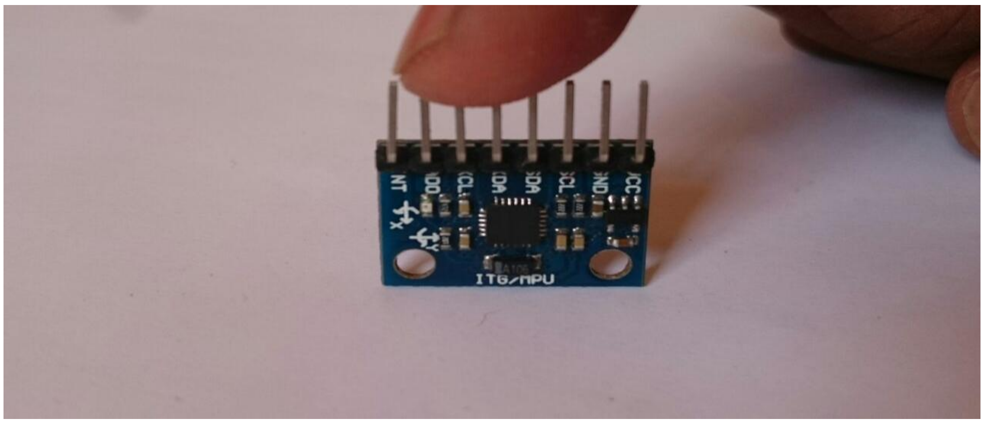
Figure 4.13: MPU-6050 6 DOF Inertial measurement unit (IMU)

Figure 4.14 shows a block diagram for the assembled electrical component and their corresponding electrical wiring. The figure shows how the component are arranged and connected to the Arduino main board.

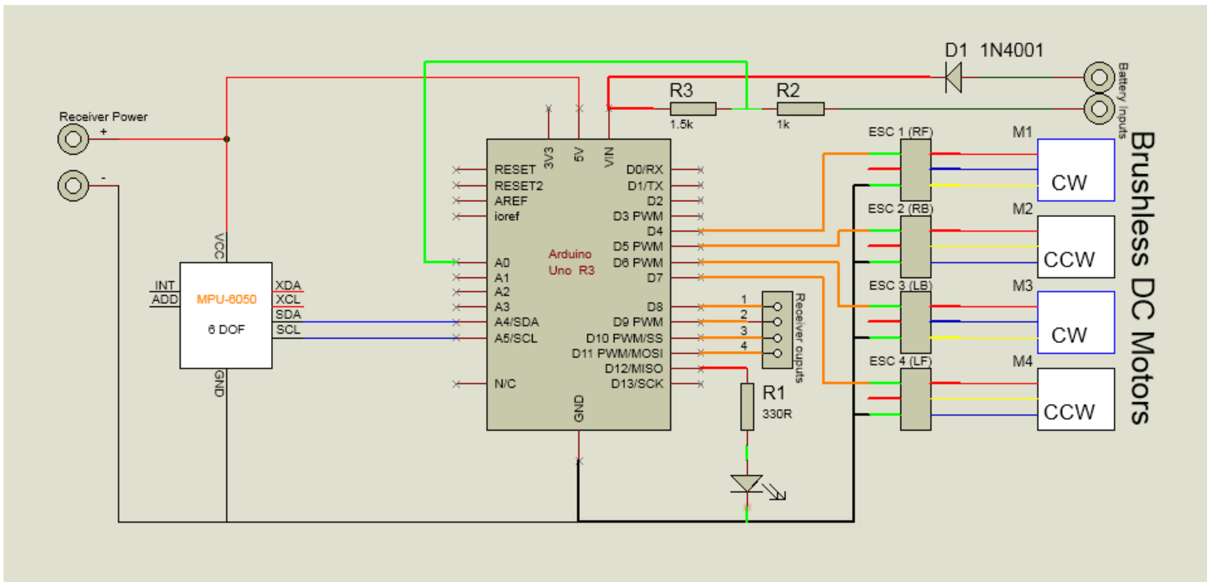

Figure 4.14: Quadrotor components electrical wiring diagram

4.3 Testing and results

Setting the quadrotor in a hover mode it was possible to acquire the angular velocities reading in real time using the inertial sensor embedded on a smart mobile phone that is attached to quadrotor body.

- **Real time data acquisition at reset mode (quadrotor on the ground)**

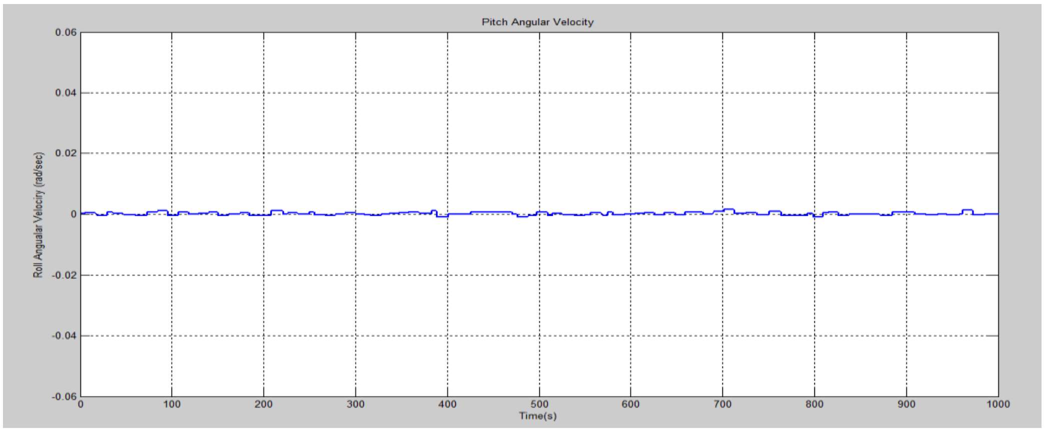

Figure 4.15: Real time Pitch angular velocity when the robot is on the ground

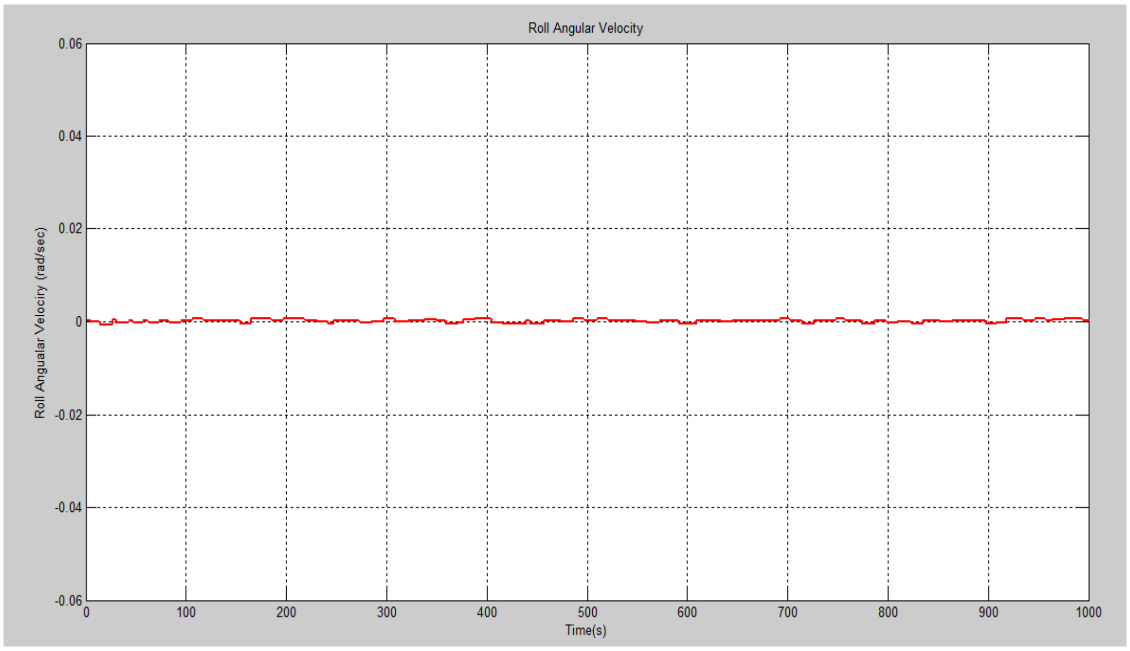

Figure 4.16: Real time Roll angular velocity when the robot is on the ground

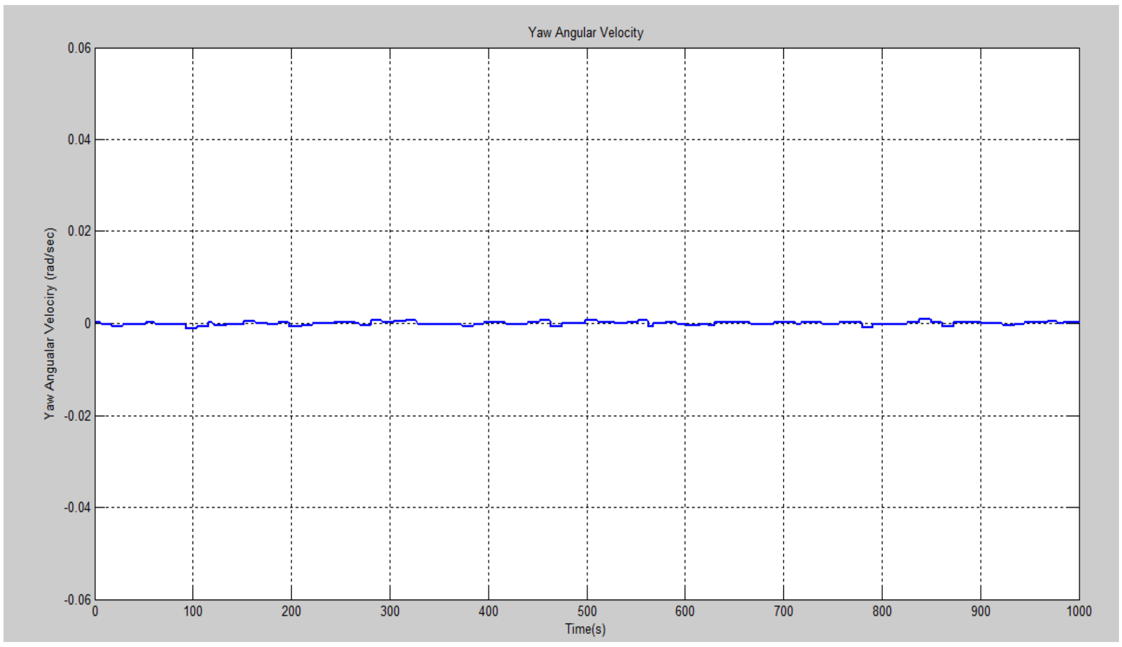

Figure 4.17: Real time Yaw angular velocity when the robot is on the ground

- **Real time data acquisition at hover mode (quadrotor is off the ground)**

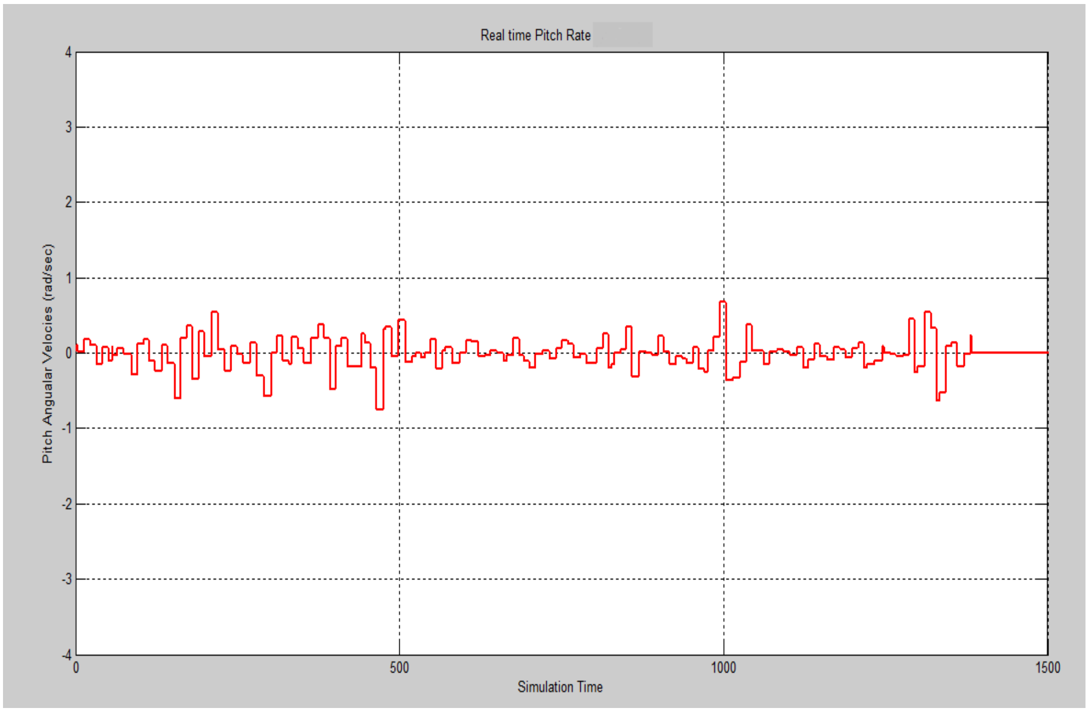

Figure 4.18: Real time Pitch angular velocity when the robot is on the move (off the ground)

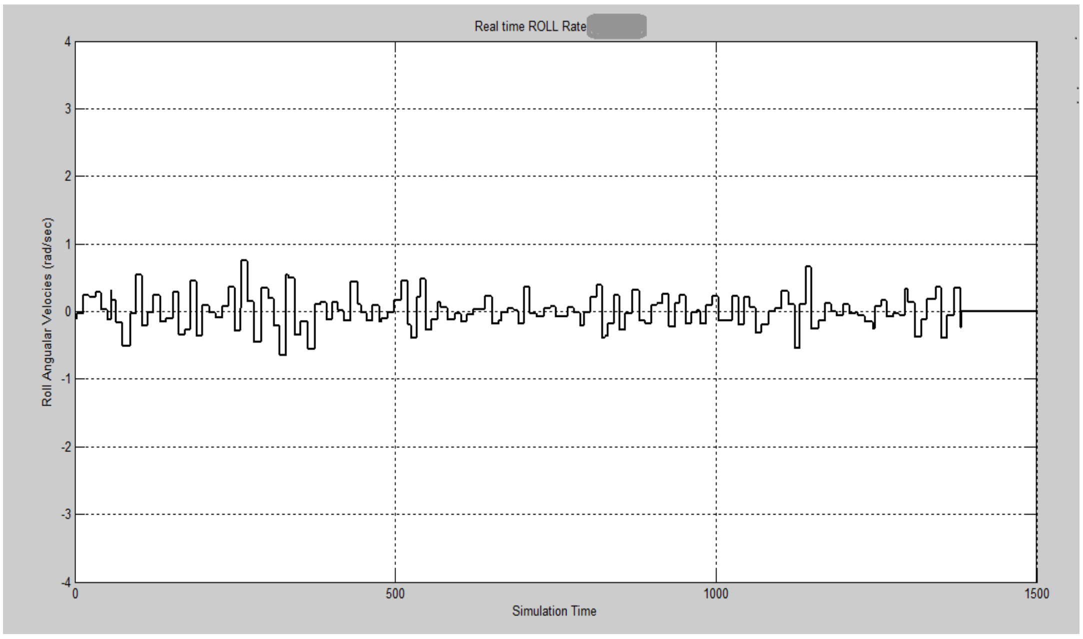

Figure 4.19: Real time Roll angular velocity when the robot is on the move (off the ground)

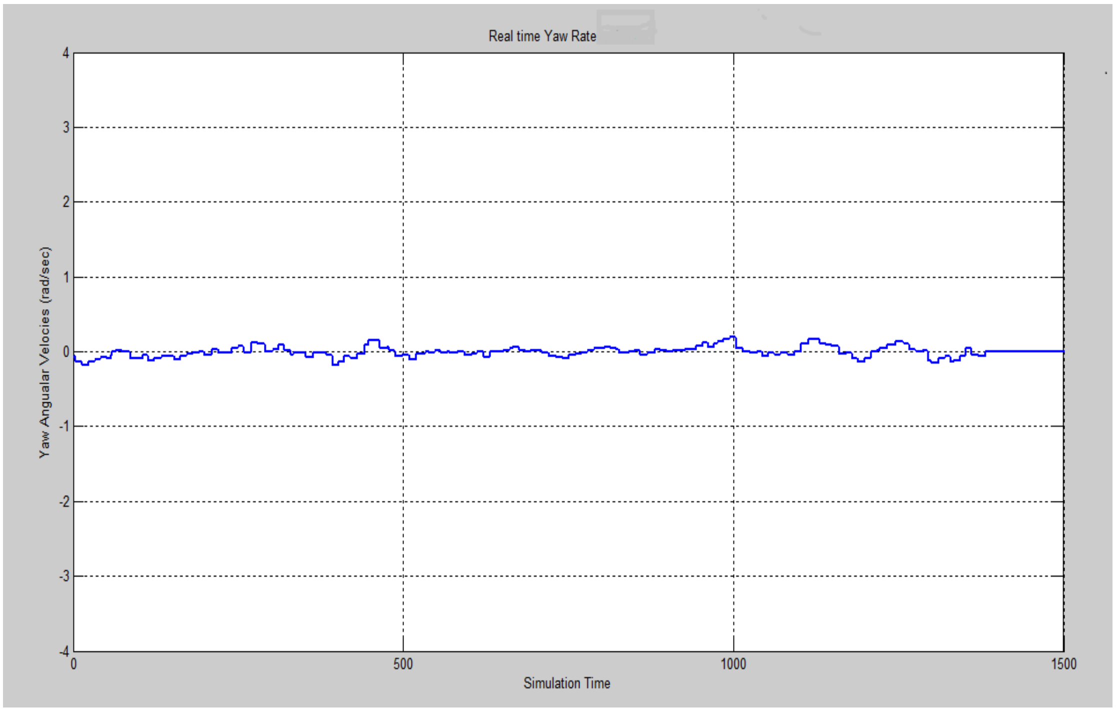

Figure 4.20: Real time Yaw angular velocity when the robot is on the move (off the ground)

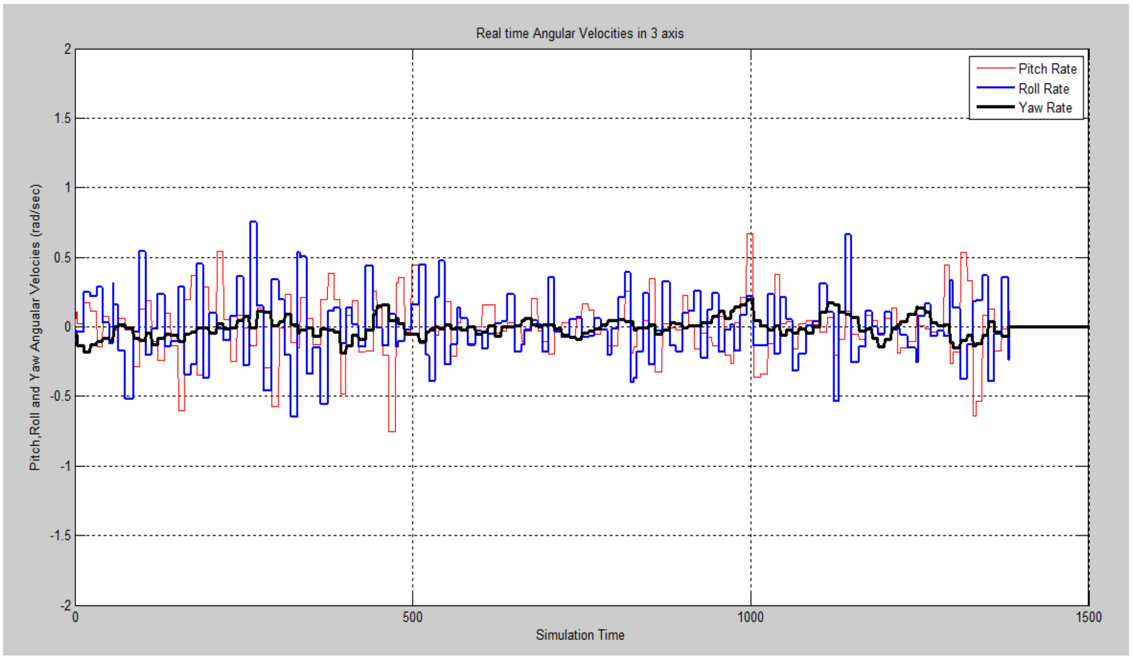

Figure 4.21: Real time Pitch ,Roll and Yaw angular velocity when the robot is on the move (off the ground)

CHAPTER FIVE
CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

A mathematical model of the **quadrotor robot system** was developed by using physical and electrical laws. A simplified mathematical model was derived by system parameters. The controller parameters values (Kp, Ki and Kd) were obtained by using manual tuning method from practical model so as to perform best system response. From experimental results, it is found that the best controller parameters which gave the best response of the system were: Kp=1.6, Ki=.07 and Kd=23.

In addition to that, the construction of quadrotor's mathematical model has been carried out and the applied forces and torques affecting the quadrotor has been calculated. MATLAB development environment has been used to analyze the equations of motion that describes the mathematical model and demonstrates the results of the close loop control system of the four movements, which are Hover, Pitch, Roll and Yaw. The results obtained from the simulation were acceptable and as had been expected.

5.2 Recommendations

1. Use of Intelligent Control theory to control quadrotor system to obtain better performance.

2. Use of Genetic Algorithm to optimize controller parameter.

3. Use of Trajectory Algorithm to Plan a specific path for the quadrotor to reach a certain target.

REFERENCES

[1] Farid Golnaraghi, Benjamin C. Kuo, “Automatic Control Systems”, JOHN WILEY & SONS, Printed in the United States of America, 2010.

[2] N.S.Nise,“Control systems engineering’’, Wiley,Hoboken N,2004.

[3] Peter Corke, “Robotics Vision and Control Fundamental Algorithms in MATLAB”, Brisbane, Queensland, 2011.

[4] F. B. HILDEBRAND, “Introduction to Numerical Analysis”, Dover Publication, INC. NEW YORK, 2006.

[5] Christopher Alexander Herda, “Implementation of a Quadrotor Unmanned Aerial Vehicle”, CALIFORNIA STATE UNIVERSITY, NORTHRIDGE, May 2012

[6] John Iovine , “Robots, Androids, and Animatrons”, McGraw-Hill, United States of America, 2002.

[7] A. L. Salih, M. Moghavveemi, H. A. F. Mohamed, and K. S. Gaeid, “Flight PID controller design for a UAV quadrotor”, Scientific Research and Essays, vol. 5, pp. 3360-3367, 2010.

APPENDIX A

Transformation metrics in 2-dimention

Referring to Figure 3.3 and Figure 3.4

$p^V = x^V \hat{x}_V + y^V \hat{y}_V$

$$= (\hat{x}_V \quad \hat{y}_V) \begin{pmatrix} x^V \\ y^V \end{pmatrix} \qquad (A.1)$$

Where $\hat{x}$, $\hat{y}$ unit-vectors parallel to the principal axes . The coordinate frame $\{B\}$ is completely described by its two orthogonal axes which we represent by two unit vectors

$$\hat{x}_B = \hat{x}_V \cos\theta + \hat{y}_V \sin\theta \qquad (A.2)$$

$$\hat{y}_B = -\hat{x}_V \sin\theta + \hat{y}_V \cos\theta \qquad (A.3)$$

which can be factorized into matrix form as

$$(\hat{x}_B \quad \hat{y}_B) = (\hat{x}_V \quad \hat{y}_V) \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad (A.4)$$

Point **P** can be expressed with respect to $\{B\}$ as

$p^B = x^B \hat{x}_B + y^B \hat{y}_B$

$$= (\hat{x}_B \quad \hat{y}_B) \begin{pmatrix} x^B \\ y^B \end{pmatrix} \qquad (A.5)$$

And substituting Eq.A.4:

$$p^B = (\hat{x}_V \quad \hat{y}_V) \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x^B \\ y^B \end{pmatrix} \qquad (A.6)$$

Now by equating the coefficients of the right-hand sides of Eq.A.6 and Eq.A.1:

$$\begin{pmatrix} x^V \\ y^V \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x^B \\ y^B \end{pmatrix} \qquad (A.7)$$

which describes how points are transformed from frame $\{B\}$ to frame $\{V\}$ when the frame is rotated. This type of matrix is known as **a rotation matrix** and denoted by${}^{V}_{B}\boldsymbol{R}$.

$$\begin{pmatrix} x^V \\ y^V \end{pmatrix} = {}^{V}_{B}\boldsymbol{R} \begin{pmatrix} x^B \\ y^B \end{pmatrix} \qquad (A.8)$$

APPENDIX B

MATLAB code for simulation and real time results

B.1 MATLAB code for three-dimensional visualization

```
% Create a controller based on it's name, using a look-up table.
function c = controller(name, Kd, Kp, Ki)
% Use manually tuned parameters, unless arguments provide the parameters.
if nargin == 1
Kd = 23.0; % 4;
Kp = 1.6; % 3;
Ki =0.07; % 6;
elseif nargin == 2 || nargin > 4
error('Incorrect number of parameters.');
end
if strcmpi(name, 'pd')
c = @(state, thetadot) pd_controller(state, thetadot, Kd, Kp);
elseif strcmpi(name, 'pid')
c = @(state, thetadot) pid_controller(state, thetadot, Kd, Kp, Ki);
else
error(sprintf('Unknown controller type "%s"', name));
end
end
% Implement a PD controller. See simulate(controller).
function [input, state] = pd_controller(state, thetadot, Kd, Kp)
% Initialize integral to zero when it doesn't exist.
if ~isfield(state, 'integral')
state.integral = zeros(3, 1);
end
% Compute total thrust.
total = state.m * state.g / state.k / ...
(cos(state.integral(1)) * cos(state.integral(2)));
% Compute PD error and inputs.
err = Kd * thetadot + Kp * state.integral;
input = err2inputs(state, err, total);
% Update controller state.
state.integral = state.integral + state.dt .* thetadot;
end
% Implement a PID controller. See simulate(controller).
function [input, state] = pid_controller(state, thetadot, Kd, Kp, Ki)
% Initialize integrals to zero when it doesn't exist.
if ~isfield(state, 'integral')
state.integral = zeros(3, 1);
state.integral2 = zeros(3, 1);
end
% Prevent wind-up
if max(abs(state.integral2)) > 0.01
state.integral2(:) = 0;
end
% Compute total thrust.
total = state.m * state.g / state.k / ...
(cos(state.integral(1)) * cos(state.integral(2)));
% Compute error and inputs.
err = Kd * thetadot + Kp * state.integral - Ki * state.integral2;
input = err2inputs(state, err, total);
% Update controller state.
state.integral = state.integral + state.dt .* thetadot;
state.integral2 = state.integral2 + state.dt .* state.integral;
end
% Given desired torques, desired total thrust, and physical parameters,
% solve for required system inputs.
function inputs = err2inputs(state, err, total)
e1 = err(1);
e2 = err(2);
e3 = err(3);
Ix = state.I(1, 1);
Iy = state.I(2, 2);
Iz = state.I(3, 3);
k = state.k;
L = state.L;
b = state.b;
inputs = zeros(4, 1);
inputs(1) = total/4 -(2 * b * e1 * Ix + e3 * Iz * k * L)/(4 * b * k * L);
inputs(2) = total/4 + e3 * Iz/(4 * b) - (e2 * Iy)/(2 * k * L);
inputs(3) = total/4 -(-2 * b * e1 * Ix + e3 * Iz * k * L)/(4 * b * k * L);
inputs(4) = total/4 + e3 * Iz/(4 * b) + (e2 * Iy)/(2 * k * L);
end


%%


function result = simulate(controller, tstart, tend, dt)
% Physical constants.

g = 9.81;
m = 1.340;
L = 0.165;
```

```
k = 3e-6;
b = 1e-7;
I = diag([6e-3, 6e-3, 12e-3]);
kd = 0.25;
% Simulation times, in seconds.
if nargin < 4
tstart = 0;
tend = 10;
dt = 0.005;
end
ts = tstart:dt:tend;
% Number of points in the simulation.
N = numel(ts);
% Output values, recorded as the simulation runs.
xout = zeros(3, N);
xdotout = zeros(3, N);
thetaout = zeros(3, N);
thetadotout = zeros(3, N);
inputout = zeros(4, N);
% Struct given to the controller. Controller may store its persistent state in it.
controller_params = struct('dt', dt, 'I', I, 'k', k, 'L', L, 'b', b, 'm', m, 'g', g);
% Initial system state.
x = [0; 0; 10];
xdot = zeros(3, 1);
theta = zeros(3, 1);
% If we are running without a controller, do not disturb the system.
if nargin == 0
thetadot = zeros(3, 1);
else
% With a control, give a random deviation in the angular velocity.
% Deviation is in degrees/sec.
deviation = 300;
thetadot = deg2rad(2 * deviation * rand(3, 1) - deviation);
end
ind = 0;
for t = ts
ind = ind + 1;
% Get input from built-in input or controller.
if nargin == 0
i =input(t);
else
[i, controller_params] = controller(controller_params, thetadot);
end
% Compute forces, torques, and accelerations.
```

```
omega = thetadot2omega(thetadot, theta);
a = acceleration(i, theta, xdot, m, g, k, kd);
omegadot = angular_acceleration(i, omega, I, L, b, k);
% Advance system state.
omega = omega + dt * omegadot;
thetadot = omega2thetadot(omega, theta);
theta = theta + dt * thetadot;
xdot = xdot + dt * a;
x = x + dt * xdot;
% Store simulation state for output.
xout(:, ind) = x;
xdotout(:, ind) = xdot;
thetaout(:, ind) = theta;
thetadotout(:, ind) = thetadot;
inputout(:, ind) = i;
end
% Put all simulation variables into an output struct.
result = struct('x', xout, 'theta', thetaout, 'vel', xdotout, ...
'angvel', thetadotout, 't', ts, 'dt', dt, 'input', inputout);
end
% Arbitrary test input.
function in = input(t)
in = zeros(4, 1);
in(:) = 700;
in(1) = in(1) + 350;
in(3) = in(3) - 350;
in = in .^ 2;
end
% Compute thrust given current inputs and thrust coefficient.
function T = thrust(inputs, k)
T = [0; 0; k * sum(inputs)];
end
% Compute torques, given current inputs, length, drag coefficient, and thrust
coefficient.
function tau = torques(inputs, L, b, k)
tau = [
L * k * (inputs(1) - inputs(3))
L * k * (inputs(2) - inputs(4))
b * (inputs(1) - inputs(2) + inputs(3) - inputs(4))
];
end
% Compute acceleration in inertial reference frame
% Parameters:
% g: gravity acceleration
% m: mass of quadcopter
% k: thrust coefficient
% kd: global drag coefficient
function a = acceleration(inputs, angles, vels, m, g, k, kd)
gravity = [0; 0; -g];
R = rotation(angles);
T = R * thrust(inputs, k);
Fd = -kd * vels;
a = gravity + 1 / m * T + Fd;
end
% Compute angular acceleration in body frame
% Parameters:
% I: inertia matrix
function omegad = angular_acceleration(inputs, omega, I, L, b, k)
tau = torques(inputs, L, b, k);
omegad = inv(I) * (tau - cross(omega, I * omega));
end
% Convert derivatives of roll, pitch, yaw to omega.
function omega = thetadot2omega(thetadot, angles)
phi = angles(1);
theta = angles(2);
psi = angles(3);
W = [
1, 0, -sin(theta)
0, cos(phi), cos(theta)*sin(phi)
0, -sin(phi), cos(theta)*cos(phi)
];
omega = W * thetadot;
end
% Convert omega to roll, pitch, yaw derivatives
function thetadot = omega2thetadot(omega, angles)
phi = angles(1);
theta = angles(2);
psi = angles(3);
W = [
1, 0, -sin(theta)
0, cos(phi), cos(theta)*sin(phi)
0, -sin(phi), cos(theta)*cos(phi)
];
thetadot = inv(W) * omega;
end


%%
```

```
function h = visualize_test(data)
% Create a figure with three parts. One part is for a 3D visualization,
% and the other two are for running graphs of angular velocity and
displacement.
figure; plots = [subplot(3, 2, 1:4), subplot(3, 2, 5), subplot(3, 2, 6)];
subplot(plots(1));
pause;
% Create the quadcopter object. Returns a handle to
% the quadcopter itself as well as the thrust-display cylinders.
[t thrusts] = quadcopter;
% Set axis scale and labels.
axis([-10 30 -20 20 5 15]);
zlabel('Height');
title('Quadcopter Flight Simulation');
% Animate the quadcopter with data from the simulation.
animate(data, t, thrusts, plots);
end
% Animate a quadcopter in flight, using data from the simulation.
function animate(data, model, thrusts, plots)
% Show frames from the animation. However, in the interest of speed,
% skip some frames to make the animation more visually appealing.
for t = 1:2:length(data.t)
% The first, main part, is for the 3D visualization.
subplot(plots(1));
% Compute translation to correct linear position coordinates.
dx = data.x(:, t);
move = makehgtform('translate', dx);
% Compute rotation to correct angles. Then, turn this rotation
% into a 4x4 matrix repesting this affine transformation.
angles = data.theta(:, t);
rotate = rotation(angles);
rotate = [rotate zeros(3, 1); zeros(1, 3) 1];
% Move the quadcopter to the right place, after putting it in the correct
orientation.
set(model,'Matrix', move * rotate);
% Compute scaling for the thrust cylinders. The lengths should represent
relative
% strength of the thrust at each propeller, and this is just a heuristic that seems
% to give a good visual indication of thrusts.
scales = exp(data.input(:, t) / min(abs(data.input(:, t))) + 5) - exp(6) + 1.5;
for i = 1:4
% Scale each cylinder. For negative scales, we need to flip the cylinder
% using a rotation, because makehgtform does not understand negative
scaling.
```

```
s = scales(i);
if s < 0
scalez = makehgtform('yrotate', pi) * makehgtform('scale', [1, 1, abs(s)]);
elseif s > 0
scalez = makehgtform('scale', [1, 1, s]);
end
% Scale the cylinder as appropriate, then move it to
% be at the same place as the quadcopter propeller.
set(thrusts(i), 'Matrix', move * rotate * scalez);
end
% Update the drawing.
%xmin = data.x(1,t)-20;
%xmax = data.x(1,t)+20;
%ymin = data.x(2,t)-20;
%ymax = data.x(2,t)+20;
%zmin = data.x(3,t)-5;
%zmax = data.x(3,t)+5;
%axis([xmin xmax ymin ymax zmin zmax]);
axis([0 65 0 35 0 13]);
drawnow;
% Use the bottom two parts for angular velocity and displacement.
subplot(plots(2));
multiplot(data, data.angvel, t);
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
title('Angular Velocity');
subplot(plots(3));
multiplot(data, data.theta, t);
xlabel('Time (s)');
ylabel('Angular Displacement (rad)');
title('Angular Displacement');
end
end
% Plot three components of a vector in RGB.
function multiplot(data, values, ind)
% Select the parts of the data to plot.
times = data.t(:, 1:ind);
values = values(:, 1:ind);
% Plot in RGB, with different markers for different components.
plot(times, values(1, :), 'r-', times, values(2, :), 'g.', times, values(3, :), 'b-.');
% Set axes to remain constant throughout plotting.
xmin = min(data.t);
xmax = max(data.t);
ymin = 1.1 * min(min(values));
```

```
ymax = 1.1 * max(max(values));
axis([xmin xmax ymin ymax]);
end
% Draw a quadcopter. Return a handle to the quadcopter object
% and an array of handles to the thrust display cylinders.
% These will be transformed during the animation to display
% relative thrust forces.
function [h thrusts] = quadcopter()
% Draw arms.
h(1) = prism(-5, -0.25, -0.25, 10, 0.5, 0.5);
h(2) = prism(-0.25, -5, -0.25, 0.5, 10, 0.5);
% Draw bulbs representing propellers at the end of each arm.
[x y z] = sphere;
x = 0.5 * x;
y = 0.5 * y;
z = 0.5 * z;
h(3) = surf(x - 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'b');
h(4) = surf(x + 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'b');
h(5) = surf(x, y - 5, z, 'EdgeColor', 'none', 'FaceColor', 'b');
h(6) = surf(x, y + 5, z, 'EdgeColor', 'none', 'FaceColor', 'b');
% Draw thrust cylinders.
[x y z] = cylinder(0.1, 7);
thrusts(1) = surf(x, y + 5, z, 'EdgeColor', 'none', 'FaceColor', 'm');
thrusts(2) = surf(x + 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'y');
thrusts(3) = surf(x, y - 5, z, 'EdgeColor', 'none', 'FaceColor', 'm');
thrusts(4) = surf(x - 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'y');
% Create handles for each of the thrust cylinders.
for i = 1:4
x = hgtransform;
set(thrusts(i), 'Parent', x);
thrusts(i) = x;
end
% Conjoin all quadcopter parts into one object.
t = hgtransform;
set(h, 'Parent', t);
h = t;
end
% Draw a 3D prism at (x, y, z) with width w,
% length l, and height h. Return a handle to
% the prism object.
function h = prism(x, y, z, w, l, h)
[X Y Z] = prism_faces(x, y, z, w, l, h);
faces(1, :) = [4 2 1 3];
faces(2, :) = [4 2 1 3] + 4;
faces(3, :) = [4 2 6 8];
faces(4, :) = [4 2 6 8] - 1;
faces(5, :) = [1 2 6 5];
faces(6, :) = [1 2 6 5] + 2;
for i = 1:size(faces, 1)
h(i) = fill3(X(faces(i, :)), Y(faces(i, :)), Z(faces(i, :)), 'r'); hold on;
end
% Conjoin all prism faces into one object.
t = hgtransform;
set(h, 'Parent', t);
h = t;
end
% Compute the points on the edge of a prism at
% location (x, y, z) with width w, length l, and height h.
function [X Y Z] = prism_faces(x, y, z, w, l, h)
X = [x x x x x+w x+w x+w x+w];
Y = [y y y+l y+l y y y+l y+l];
Z = [z z+h z z+h z z+h z z+h];
end
```

B.2 MATLAB code for real time data acquisition

```
clear m
  m = mobiledev;
  m.AngularVelocitySensorEnabled = 1;
  m.Logging = 1;

  %// initialize data for rolling plot
  data = zeros(1,200);
i=1;
  tic
  while (toc < 30)%run for 30 secs

      %// read from accel
      a =  m.AngularVelocity
      %//  conditional  prevents  it  from
indexing an empty array the first couple
      %// of times
      if(exist('a','var')&&~isempty(a))
```

```
            %get new z coordinate
            newPoint = a(1);
            l(i) = newPoint;
            newPoint1 = a(2);
            o(i) = newPoint1;
            newPoint2 = a(3);
            q(i) = newPoint2;
            %// concatenate and pop oldest
point off
            data     =     [data(2:length(data))
newPoint];

            %draw plot with set axes
            plot(data);
            axis([0 200 -90 90]);
            %t1=trotz(data(i),'deg');
            %t2=trotz(data(i),'deg');
            %wktranimate(t1,t2);
            drawnow
            i=i+1;
        end
    end
```

APPENDIX C

Arduino microcontroller code for quadrotor operation

```
#include <Wire.h>

#include <EEPROM.h>

float pid_p_gain_roll = 1.6;

float pid_i_gain_roll = 0.07;

float pid_d_gain_roll = 23;

int pid_max_roll = 400;

float pid_p_gain_pitch = pid_p_gain_roll;

float pid_i_gain_pitch = pid_i_gain_roll;

float pid_d_gain_pitch = pid_d_gain_roll;

int pid_max_pitch = pid_max_roll;

float pid_p_gain_yaw = 3;

float pid_i_gain_yaw = 0.02;

float pid_d_gain_yaw = 0;              /

int pid_max_yaw = 400;

byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;

byte eeprom_data[36];

byte highByte, lowByte;

int receiver_input_channel_1, receiver_input_channel_2,
receiver_input_channel_3, receiver_input_channel_4;
```

```
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;

int esc_1, esc_2, esc_3, esc_4;

int throttle, battery_voltage;

int cal_int, start, gyro_address;

int receiver_input[5];

unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;

unsigned long timer_1, timer_2, timer_3, timer_4, current_time;

unsigned long loop_timer;

double gyro_pitch, gyro_roll, gyro_yaw;

double gyro_axis[4], gyro_axis_cal[4];

float pid_error_temp;

float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;

float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch, pid_last_pitch_d_error;

float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;

//Setup routine

void setup(){

 for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);

 gyro_address = eeprom_data[32];

 Wire.begin();

 DDRD |= B11110000;

 DDRB |= B00110000;
 digitalWrite(12,HIGH);
 while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' ||
eeprom_data[35] != 'B')delay(10);
 set_gyro_registers();
 for (cal_int = 0; cal_int < 1250 ; cal_int ++){
   PORTD |= B11110000;
   delayMicroseconds(1000);
   PORTD &= B00001111;
   delayMicroseconds(3000;
 }
 for (cal_int = 0; cal_int < 2000 ; cal_int ++){
   if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12));
   gyro_signalen();
   gyro_axis_cal[1] += gyro_axis[1];
   gyro_axis_cal[2] += gyro_axis[2. ];
   gyro_axis_cal[3] += gyro_axis[3];
       PORTD |= B11110000;
   delayMicroseconds(1000);
   PORTD &= B00001111;
   delay(3);
 }
   gyro_axis_cal[1] /= 2000;
 gyro_axis_cal[2] /= 2000;
 gyro_axis_cal[3] /= 2000;
 PCICR |= (1 << PCIE0);
 PCMSK0 |= (1 << PCINT0);
```

```
  PCMSK0 |= (1 << PCINT1);
  PCMSK0 |= (1 << PCINT2);
  PCMSK0 |= (1 << PCINT3);
  while(receiver_input_channel_3 < 1000 || receiver_input_channel_3 > 1020 || receiver_input_channel_4 < 1400){
    receiver_input_channel_3 = convert_receiver_channel(3);
    receiver_input_channel_4 = convert_receiver_channel(4);
    start ++;
    PORTD |= B11110000;
    delayMicroseconds(1000);
    PORTD &= B00001111;
    delay(3);
    if(start == 125){
     digitalWrite(12, !digitalRead(12));
      start = 0;
    }
  }
  start = 0;
  battery_voltage = (analogRead(0) + 65) * 1.2317;
  digitalWrite(12,LOW);
}
void loop(){
  receiver_input_channel_1 = convert_receiver_channel(1);       //Convert the actual receiver signals for pitch to the standard 1000 - 2000us.
  receiver_input_channel_2 = convert_receiver_channel(2);       //Convert the actual receiver signals for roll to the standard 1000 - 2000us.
  receiver_input_channel_3 = convert_receiver_channel(3);        //Convert
the actual receiver signals for throttle to the standard 1000 - 2000us.

  receiver_input_channel_4 = convert_receiver_channel(4);        //Convert
the actual receiver signals for yaw to the standard 1000 - 2000us.

  gyro_signalen();

  gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll / 57.14286) *
0.2);

  gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch / 57.14286) *
0.2);

  gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw / 57.14286) *
0.2);

  if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 <
1050)start = 1;

  if(start == 1 && receiver_input_channel_3 < 1050 &&
receiver_input_channel_4 > 1450){

    start = 2;

    pid_i_mem_roll = 0;

    pid_last_roll_d_error = 0;

    pid_i_mem_pitch = 0;

    pid_last_pitch_d_error = 0;

    pid_i_mem_yaw = 0;

    pid_last_yaw_d_error = 0;

  }

  if(start == 2 && receiver_input_channel_3 < 1050 &&
receiver_input_channel_4 > 1950)start = 0;

  pid_roll_setpoint = 0;

  if(receiver_input_channel_1 > 1508)pid_roll_setpoint =
(receiver_input_channel_1 - 1508)/3.0;
```

```
  else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = (receiver_input_channel_1 - 1492)/3.0;

  pid_pitch_setpoint = 0;

  if(receiver_input_channel_2 > 1508)pid_pitch_setpoint = (receiver_input_channel_2 - 1508)/3.0;

  else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = (receiver_input_channel_2 - 1492)/3.0;

  pid_yaw_setpoint = 0;

  if(receiver_input_channel_3 > 1050){ //Do not yaw when turning off the motors.

    if(receiver_input_channel_4 > 1508)pid_yaw_setpoint = (receiver_input_channel_4 - 1508)/3.0;

    else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint = (receiver_input_channel_4 - 1492)/3.0;

  }

  calculate_pid();

  battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;

  if(battery_voltage < 1030 && battery_voltage > 600)digitalWrite(12, HIGH);

  throttle = receiver_input_channel_3;

 the throttle signal as a base signal.

  if (start == 2){

    if (throttle > 1800) throttle = 1800;

    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw;    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw;

    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw;
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw
  if (esc_1 < 1200) esc_1 = 1200;
    if (esc_2 < 1200) esc_2 = 1200;
    if (esc_3 < 1200) esc_3 = 1200;
    if (esc_4 < 1200) esc_4 = 1200;
    if(esc_1 > 2000)esc_1 = 2000;
    if(esc_2 > 2000)esc_2 = 2000;
    if(esc_3 > 2000)esc_3 = 2000;
    if(esc_4 > 2000)esc_4 = 2000;
   }
   else{
    esc_1 = 1000;
    esc_2 = 1000;
    esc_3 = 1000;
    esc_4 = 1000;
   }
   while(micros() - loop_timer < 4000);
   loop_timer = micros();
   PORTD |= B11110000;
   timer_channel_1 = esc_1 + loop_timer;
   timer_channel_2 = esc_2 + loop_timer;
   timer_channel_3 = esc_3 + loop_timer;
   timer_channel_4 = esc_4 + loop_timer;
   while(PORTD >= 16){
    esc_loop_timer = micros();
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;
if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;
  }
}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                              //Is input 8 high?
    if(last_channel_1 == 0){
      last_channel_1 = 1;
      timer_1 = current_time;
    }
  }
  else if(last_channel_1 == 1){
    last_channel_1 = 0;                            state
    receiver_input[1] = current_time - timer_1;
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){
    if(last_channel_2 == 0){
      last_channel_2 = 1;
      timer_2 = current_time;
    }
  }
  else if(last_channel_2 == 1){
    last_channel_2 = 0;                                   /
    receiver_input[2] = current_time - timer_2;
  }
  //Channel 3==========================================
  if(PINB & B00000100 ){
    if(last_channel_3 == 0){
      last_channel_3 = 1;
      timer_3 = current_time;
    }
  }
  else if(last_channel_3 == 1){
    last_channel_3 = 0;
    receiver_input[3] = current_time - timer_3;
  }
  //Channel 4==========================================
  if(PINB & B00001000 ){
    if(last_channel_4 == 0){
      last_channel_4 = 1;
      timer_4 = current_time;
    }
  }
  else if(last_channel_4 == 1){
    last_channel_4 = 0;
    receiver_input[4] = current_time - timer_4;
  }
}

//Subroutine for reading the gyro
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);
    Wire.write(0x43);
    Wire.endTransmission();                              //End the transmission
    Wire.requestFrom(gyro_address,6);
    while(Wire.available() < 6);
    gyro_axis[1] = Wire.read()<<8|Wire.read();
    gyro_axis[2] = Wire.read()<<8|Wire.read();
    gyro_axis[3] = Wire.read()<<8|Wire.read();
  }
  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];
    gyro_axis[2] -= gyro_axis_cal[2];
    gyro_axis[3] -= gyro_axis_cal[3];
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;
}
//Subroutine for calculating pid outputs


void calculate_pid(){
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll =
pid_max_roll * -1;


  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll
+ pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll =
pid_max_roll * -1;
  pid_last_roll_d_error = pid_error_temp;
  //Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch =
pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch =
pid_max_pitch * -1;
```

```
  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;
  pid_last_pitch_d_error = pid_error_temp;


  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;
  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_last_yaw_d_error);
  if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;
  pid_last_yaw_d_error = pid_error_temp;
}
int convert_receiver_channel(byte function){
  byte channel, reverse;
  int low, center, high, actual;
  int difference;
  channel = eeprom_data[function + 23] & 0b00000111;
if(eeprom_data[function + 23] & 0b10000000)reverse = 1;

  else reverse = 0;

  actual = receiver_input[channel];

  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 +
14];  //Store the low value for the specific receiver input channel

  center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2
- 2]; //Store the center value for the specific receiver input channel

  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 +
6];   //Store the high value for the specific receiver input channel

  if(actual < center){

    if(actual < low)actual = low;

  difference = ((long)(center - actual) * (long)500) / (center - low);
if(reverse == 1)return 1500 + difference;

    else return 1500 - difference;

  }

  else if(actual > center){

    if(actual > high)actual = high;

    difference = ((long)(actual - center) * (long)500) / (high - center);

    if(reverse == 1)return 1500 – difference;

    else return 1500 + difference;

  }

  else return 1500;

}

void set_gyro_registers(){

  if(eeprom_data[31] == 1){

    Wire.beginTransmission(gyro_address);
    Wire.write(0x6B);
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1B);
    Wire.write(0x08);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1B);
    Wire.endTransmission();                        //End the transmission
    Wire.requestFrom(gyro_address, 1);
    while(Wire.available() < 1);
    if(Wire.read() != 0x08){
      digitalWrite(12,HIGH);
      while(1)delay(10);
    }
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1A);
    Wire.write(0x03);
    Wire.endTransmission();
  }
    Wire.beginTransmission(gyro_address);
    Wire.write(0x23);
    Wire.endTransmission();                        //End the transmission
    Wire.requestFrom(gyro_address, 1);
```

```
    while(Wire.available() < 1);
    if(Wire.read() != 0x90){
      digitalWrite(12,HIGH);
      while(1)delay(10);
    }
  }
    Wire.beginTransmission(gyro_address);
    Wire.write(0x23);
    Wire.endTransmission();                                    //End the transmission
    Wire.requestFrom(gyro_address, 1);
    while(Wire.available() < 1);
    if(Wire.read() != 0x90){
      digitalWrite(12,HIGH);
      while(1)delay(10);
    }
  }
}
```