

# MULTI-PURPOSE QUADCOPTER



**KITHINJI A. MURIUNGI**

**EC/09/14**

**ENGINEERING PROJECT II**

**ECE 590**

*Supervisor: Mr. Severinus Kifalu*

Academic Year: 2018/19

**BACHELOR OF ENGINEERING**

**(ELECTRICAL & ELECTRONICS)**

**A Project report submitted to Moi University in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering (Electrical & Electronics) in the Department of Electrical & Communication Engineering, School of Engineering.**

May 7, 2019

## DEDICATION

I dedicate this project to all supporters of Drone Technologies intending to build custom drones for applications with a focus in cutting-edge technologies like the Internet of Things (IoT) and Cloud Computing.

I also dedicate the project to the students with interests in exploring Drones to make improvement and modifications where possible as well as build unusual solutions based on this project idea.

Lastly, I dedicate this project to Drone Regulators and Companies dealing with related technologies so that they can see the more opportunities available in this area and support this kind of research and development projects.

### DECLARATION

I do declare that this report represents my personal work to the best of my know-how both in substance and form. The report has not been presented for any academic purpose whatsoever in any other institution of higher learning. Where any other scholarly work has been used, references have been provided.

Signature .....

Date.....

Kithinji A. Muriungi

EC/09/14

Moi University – Department of Electrical & Communication Engineering

### CERTIFICATION

This is to certify that Kithinji Muriungi (EC/09/14) has so far worked on this project, detailed in this report, under my supervision.

Signature .....

Date.....

Mr. Severinus Baraka Kifalu

Project Supervisor

Moi University – Department of Electrical & Communication Engineering

## ACKNOWLEDGMENT

First, I would like to acknowledge the Almighty God for the physical, mental and spiritual strength he has given me throughout the course as well as through this project.

Secondly, special thanks and appreciation for the encouragement, assistance, and guidance given by Moi University Team: Team Tolv, my supervisor: Mr. S. B. Kifalu, and Ben Salcie throughout the designing, building, implementation, and testing of the project.

Lastly, my life in school and working on this project would not be complete without my family and friends for their constant motivation and encouragement.

## Table of Contents

DEDICATION .....	ii
DECLARATION .....	iii
CERTIFICATION.....	iii
ACKNOWLEDGMENT .....	iv
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
LIST OF SYMBOLS.....	xiii
LIST OF ABBREVIATIONS.....	xiv
ABSTRACT .....	xv
CHAPTER ONE .....	1
1. INTRODUCTION .....	1
1.1 Quadcopter at a Glance .....	1
1.2 Motivation .....	2
1.3 Statement of the Problem .....	2
1.4 Justification.....	2
1.5 Aims and Objectives .....	3
1.6 Scope and Limitations .....	4
CHAPTER TWO .....	5
2. REVIEW OF LITERATURES .....	5
2.1 overview .....	5
2.2 Quadcopter Hardware Components.....	6
2.2.1 Flight Controller .....	6
2.2.2 Frame.....	7
2.2.3 Motor.....	7
2.2.4 Electronic Speed Controller (ESC) .....	7
2.2.5 Propellers .....	8
2.2.6 Transmitter .....	8
2.2.7 Receiver .....	8
2.2.8 Lithium-Polymer (Li-Po) Battery.....	9
2.2.9 Battery Monitor .....	9

2.2.11 MPU6050 Sensor .....	9
2.3 Other Hardware Components .....	10
2.3.1 ESP8266 (Node MCU) .....	10
2.3.2 Gas Sensor .....	11
2.4 Miscellaneous Components.....	11
2.5 Software and other Platforms .....	13
2.5.1 Android Studio .....	13
2.5.2 Firebase: Google Cloud Platform .....	14
2.5.2 Fritzing .....	14
2.5.3 Other Platforms .....	15
CHAPTER THREE .....	16
3. DESIGN METHODOLOGY .....	16
3.1 Overview.....	16
3.2 MECHANICS AND SYSTEM DESIGN.....	16
3.2.1 Basic Mechanics.....	16
3.2.1.1 Motor Speed.....	17
3.2.1.2 Motor Torque .....	17
3.2.1.3 Motor Force.....	17
3.2.1.4 Motor Moments.....	17
3.2.1.5 Resultant Forces (F) .....	19
3.2.1.6 Resultant Moments (M) .....	19
3.2.1.7 At Equilibrium.....	19
3.2.1.8 Vertical Acceleration (Thrust).....	19
3.2.2 Controls of a Quadcopter .....	21
3.2.2.1 Control of Height.....	21
3.2.2.1.1 Control of a Linear 2 <sup>nd</sup> Order System .....	22
3.2.2.1.2 General Approach.....	22
3.2.2.1.3 Strategy .....	22
3.2.2.1.4 PID Control .....	23
3.2.3 Design Considerations .....	29
3.2.3.1 Thrust versus Weight.....	29

3.2.3.1.1 Control with Thrust Limitations.....	29
3.2.3.1.2 PD Control .....	29
3.2.3.1.3 PID Control.....	29
3.2.3.1.4 Effects of Thrust/Weight Ratio .....	30
3.2.3.2 Power, Thrust, and Energy.....	30
3.2.3.3 Mass Distribution .....	32
3.2.3.4 Quadcopter Motion Ratios .....	33
3.2.4 Agility .....	34
3.2.5 Components Selection .....	36
3.2.6 Effects of Size .....	37
3.2.6.1 Mass, Inertia .....	37
3.2.6.2 Thrust .....	37
3.2.6.3 Moment .....	38
3.2.6.4 Maximum Acceleration .....	38
3.2.6.5 Scaling Principles.....	38
3.2.6.5.1 Froude Scaling .....	38
3.2.6.5.2 Mach Scaling .....	38
3.3 ROTATION.....	39
3.3.1 Quadcopter Coordinate System.....	39
3.3.2 Euler Angles .....	40
3.3.3 States of the Quadcopter .....	41
3.3.4 How Quadcopter Works .....	41
CHAPTER FOUR .....	43
4. DESIGN AND CONSTRUCTION.....	43
4.1 hardware configurations .....	43
4.1.1 General Quadcopter Block Diagram.....	43
4.1.2 Breadboard Wiring Diagram.....	44
4.1.2.1 Quadcopter wiring .....	44
4.1.2.2 Gas Detection wiring .....	44
4.1.3 Quadcopter Schematic Diagram .....	46
4.1.3.1 Motors and ESCs .....	46

4.1.3.2 Receiver .....	47
Transmitter .....	48
4.1.3.3 MPU6050 .....	49
4.1.3.4 Other Components .....	50
4.2 SOFTWARE CONFIGURATIONS .....	50
4.2.1 Quadcopter PID Controller .....	51
4.2.1.1 Overview .....	51
4.2.1.2 Implementation of Quadcopter PID Controller .....	52
4.2.1.2.1 Proportional.....	52
4.2.1.2.2 Integral.....	53
4.2.1.2.3 Derivative .....	53
4.2.1.2.4 PID in Code.....	53
4.2.1.3 Flight Controller Algorithm .....	54
4.2.1.3.1 Initial Set-Up.....	54
4.2.1.3.2 Main Loop.....	55
4.2.1.3.3 Interrupt Function .....	58
4.2.1.4 Flight Code Model.....	58
4.2.2 Air Quality Monitor .....	59
4.2.2.1 Overview .....	59
4.2.2.2 Communication and Data processing .....	59
4.2.2.3 Node MCU Code Structure.....	60
4.2.2.4 Firebase: Google Cloud Access .....	60
4.2.2.5 Android Mobile Application .....	61
CHAPTER FIVE .....	62
5. RESULTS AND DISCUSSIONS.....	62
5.1 Calibration of the sensors and actuators.....	62
5.1.1 Gyro Calibration.....	63
5.1.2 ESCs Calibration .....	64
5.1.3 Transmitter Initial Set-up.....	65
5.1.4 Motors and Propellers Calibration .....	65
5.2 PID Tuning.....	66



5.2.1 Manual PID Tuning Procedure .....	67
5.3 SYSTEM TESTING .....	68
5.3.1 Quadcopter PID Controller .....	68
5.3.2 Air Monitor .....	70
5.4 Performance Analysis and Evaluation .....	71
5.4.1 PID Gains Analysis and Evaluation .....	71
5.4.2 Common Mistakes when Operating a Quadcopter .....	73
CHAPTER SIX .....	74
6. Project schedule and budget .....	74
6.1 Project Schedule .....	74
6.2 budget .....	75
CHAPTER SEVEN .....	76
7. CHALLENGEs, conclusion, and recommendations .....	76
7.1 Challenges .....	76
7.2 Conclusion .....	76
7.3 Recommendations .....	77
7.4 Future Works .....	77
REFERENCES .....	78
APPENDICES .....	81
KCAA Drone License .....	81
Air Detection and Monitoring Arduino Code .....	82
QUADCOPTER ARDUINO CODE .....	83

## LIST OF TABLES

Table 1: Li-Po Battery Specifications.....	32
Table 2: Quadcopter Mass Distribution.....	33
Table 3: Quadcopter Size Specifications.....	36
Table 4: Motor Mixing Algorithm (MMA).....	51

## LIST OF FIGURES

Figure 1: Arduino Uno Microcontroller .....	6
Figure 2: Arducopter Flight Controller .....	6
Figure 3: Quadcopter X-Frame .....	7
Figure 4: Brushless DC Motors .....	7
Figure 5: ESCs .....	7
Figure 6: Propellers .....	8
Figure 7: Transmitter .....	8
Figure 8: Receiver .....	8
Figure 9: Li-Po Battery .....	9
Figure 10: Battery Monitor .....	9
Figure 11: MPU6050 Sensor .....	9
Figure 12: ESP8266 .....	10
Figure 13: MQ-2 Gas Sensor .....	11
Figure 14: Strip Board Circuit .....	12
Figure 15: Quadcopter Components .....	12
Figure 16: Android Studio .....	13
Figure 17: Firebase .....	14
Figure 18: Fritzing .....	15
Figure 19: Force, Moment Vs Speed .....	17
Figure 20: Force, Moment, Torque, & Speed .....	18
Figure 21: Forces & Torque on the rotors .....	19
Figure 22: Thrust torque and force .....	20
Figure 23: Quadcopter Propeller Thrust .....	20
Figure 24: General Control of a Quadcopter .....	21
Figure 25: PD Control .....	23
Figure 26: PD Controller Response .....	23
Figure 27: PID Control Representation .....	24
Figure 28: PID Control Response .....	24
Figure 29: PD Stable Response .....	25
Figure 30: PD Marginally Stable Response .....	25
Figure 31: PD Unstable Response .....	26
Figure 32: PD Controller General Response .....	26
Figure 33: High Kp Gain Response .....	27
Figure 34: Low Kp Gain Response .....	27
Figure 35: High Kd Gain Response .....	28
Figure 36: Ki Gain Response .....	28
Figure 37: Thrust Vs Weight Ratio .....	30
Figure 38: Li-Po Specific Power Vs Specific Energy .....	31
Figure 39: Agility Explained .....	35
Figure 40: Quadcopter Size Configurations .....	37

Figure 41: Thrust .....	41
Figure 42: Pitch.....	41
Figure 43: Roll .....	42
Figure 44: Yaw .....	42
Figure 45: Yaw & Roll Motion .....	42
Figure 46: General Quadcopter Block Diagram .....	43
Figure 47: Fritzing Quadcopter Wiring .....	44
Figure 48: Fritzing Gas Detection Wiring.....	44
Figure 49: Gas Detection and Monitoring.....	45
Figure 50: Quadcopter Schematic Diagram .....	46
Figure 51: Transmitter MODE 2 Configurations .....	48
Figure 52: Transmitter: Starting, Stopped, & Started .....	48
Figure 53: MPU6050 Arduino Connection .....	49
Figure 54: Mounting of MPU6050 on a Strip Board .....	50
Figure 55: Implementation of MMA .....	52
Figure 56: Flight Code Model .....	58
Figure 57: Communication and Data Processing .....	59
Figure 58: Firebase Cloud Access.....	60
Figure 59: Android Mobile App Interface .....	61
Figure 60: Gyro Configuration.....	64
Figure 61: Transmitter MODE Configurations .....	65
Figure 62: PID Response Testing .....	67
Figure 63: PID Controller Manual Tuning .....	68
Figure 64: PID Controller Testing .....	69
Figure 65: Quadcopter Flying .....	69
Figure 66: Final Quadcopter Prototype .....	70

## LIST OF SYMBOLS

$x$	forward position in body coordinate frame	
$y$	lateral position in body coordinate frame	
$z$	vertical position in body coordinate frame	
$X, Y, Z$	position in world coordinate frame	
$u_1$	vertical thrust generated by the four rotors	
$u_2$	pitching moment	
$u_3$	yawing moment	
$u_4$	rolling moment	
$\Theta$	pitch angle	} Euler angles
$\Psi$	yaw angle	
$\Phi$	roll angle	
$m$	mass of the Quadcopter	
$K_i$	drag coefficients for the system	
$f_i$	thrusts generated by four rotors	
$I_i$	moments of inertia with respect to the axes	
$C$	force to moment scaling factor	
$\omega = [\omega_x \ \omega_y \ \omega_z]^T$	angular velocity vector of the Quadcopter resolved into body frame	
$K_P$	PID proportional gain	
$K_I$	PID integral gain	
$K_D$	PID differential gain	
$L$	half the length of the Quadcopter	
$G$	acceleration due to gravity	
$m_p$	linear momentum	

## LIST OF ABBREVIATIONS

MEMS - Micro-Electro-Mechanical Systems

UAV - Unmanned Aerial Vehicle

PID – Proportional, Integral, and Derivative

DoFs – Degrees of Freedom

IMU – Inertial Measurement Unit

GUI - Graphical User Interface

ESC - Electronic Speed Controller

Rpm – Revolutions per Minute

EoMs - Equations of Motion

CW – Clockwise

CCW – Anti-Clockwise

LPG – Liquified Petroleum Gas

SDA – Serial Data

SCL – Serial Clock

## ABSTRACT

"Multi-Purpose Quadcopter" is the undergraduate final year engineering project done by Kithinji A. Muriungi (EC/09/14) under the supervision of Mr. Severinus Baraka Kifalu. This project took one academic year from the start to its completion. The project requires the student to have a solid understanding of drone technologies, be intelligent design, construct, test, and finally implement a fully working drone with the custom PID flight controller. After finishing the basic quadcopter implementation, the student is also required to perform a real-life working application on the drone.

This report is comprised of seven chapters that explain and describes different aspects of the project from start to end. Chapter one introduces the project ranging from what a drone is to motivations and aims of the project. Chapter two describes the reviews of the literature to be used in the entire period that one will be handling the project. Chapter three brings a different approach from the previous two sections since it is geared in ensuring the drone mechanics and system designs are fully understood. This spans from basic mechanics of a quadcopter through design considerations, agility, components selection, controls, effects of different aspects and rotation understanding.

Chapter four highlights in details the actual design and construction did. This includes hardware and software configurations among other elements of the project that made the project fully implemented. The following section discusses the working of the system ranging from calibration, tuning, testing, and performance analysis. The project schedule and budget allocation of the project are also highlighted.

Finally, the report concluded by going through the challenges encountered during the duration of undertaking the project while recommending the best approaches and ways of making future projects better. The last subsection gives plans and works related to the projects. The entire report sums up with text and visual in making the project understood while solving the two tasks assigned to the project.

## CHAPTER ONE

### 1. INTRODUCTION

#### 1.1 QUADCOPTER AT A GLANCE

Advancements in sensors, microcontroller technology, control, and aerodynamics theory have made multi-purpose drones a reality. A quadcopter uses four fixed propellers to control lift and a combination of propeller torques to control yaw, thrust, roll, and pitch. The changeable size, cheap, and maneuverability of these systems have made them potential solutions in a large class of applications from amateur to complex scientific exploration applications.

The compact structure of drones makes them more vulnerable to environmental and weight distribution effects which affect its stability. The fundamental problem with the safe operation of quadcopters with wingspan smaller than one meter is reliable stabilization, robustness to unpredictable changes in the environment, and resilience to noisy data from small sensor systems. Autonomous operation of quadcopters depends on PID control capabilities which ensures the achievement of a stable flight.

The decreasing cost of modern microcontrollers and high-level programming language like Arduino has made electronics and software of completely autonomous control of quadcopters feasible for commercial, military, and even hobbyist purposes. Quadcopter control is a fundamentally new and challenging problem as this forms the ultimate challenge of this project. With 6 DoFs (3 translational and three rotational) and only four independent inputs (rotor speeds), quadcopters are much under-actuated. To achieve 6 DoFs, rotational and translational motion are coupled. To ensure you attain a smooth correlation of all these degrees, a PID controller is implemented. These 6 DoFs creates a fascinating control problem.



## 1.2 MOTIVATION

The primary motivation of this project lies in the design of my solutions to the many challenges faced during the development of a quadcopter. This project has therefore enabled learning the various aspects regarding drones and its applications.

## 1.3 STATEMENT OF THE PROBLEM

Quadcopter control is fundamentally tricky but at the same time an exciting problem. With 6 DoFs and only four independent inputs, the quadcopter is, and the resulting dynamics are highly nonlinear. Unlike ground vehicles, aerial vehicles have very little friction to prevent their motion; therefore, they must provide their damping to stop moving and remain stable.

This leads to the design and implementation of the Arduino-Based PID controller. This is a challenging task as many of the controllers used are preprogrammed with almost perfect tuning. These make the stabilization of the quadcopter a fascinating control problem. The span and complexity make this project a great learning platform.

## 1.4 JUSTIFICATION

Designing a quadcopter is simple since most of the specific designs have been done according to universally acceptable specifications. Building requires precision. This comes hand-in-hand in ensuring everything is balanced. Without a correct balance of almost everything, while building, it becomes complicated in attaining a stable control. Apart from designing and building, the ultimate challenge comes from developing electronics and software to reach firm control. This is achieved by the implementation of a PID Control which depends upon designing and building of the quadcopter.

By attaining the stable PID control of the quadcopter, the following applications can be realized:

- ✓ Remote sensing
- ✓ Commercial aerial surveillance
- ✓ Domestic policing

- ✓ Oil, gas and mineral exploration and production
- ✓ Transport of materials
- ✓ Scientific research
- ✓ Armed attacks
- ✓ Aerial target practice in the training of human pilots
- ✓ Search and rescue
- ✓ Maritime patrol
- ✓ Forest fire detection

Among the above applications, Drone Air Quality Monitoring has been implemented to achieve the application task of this project.

## 1.5 AIMS AND OBJECTIVES

This project aims in:

- i. Demonstrating understanding of Drone (Quadcopter) basic mechanics and other related factors in designing a quadcopter.
- ii. Designing based on the pre-existing models and also building based on the prescribed specifications.
- iii. It is implementing a fully functional Arduino-based PID Controller which works well while the quadcopter is in thrust, pitch, yaw, or roll mode. This should enable us to attain a stable flight.
- iv. Implementation of the real application of IoT-Based solution on the quadcopter: Air Quality Monitoring.

## 1.6 SCOPE AND LIMITATIONS

The project involves theoretical understanding, hardware interfacing, software programming and implementation of other cloud-based platforms. Extensive know-how is needed in handling a project of this nature. This will ensure fast designing and also building without many hindrances in implementing. The choice of hardware is the critical task in hardware interfacing since all equipment in a quadcopter depends upon the measurements of each other. The wrong choice of one component will affect the entire design leading to the faulty building which is far much hectic when it comes to software programming.

Several flight controller is in existence: some having complex algorithms and software programming but also simple to implement especially Ardupilot. To make the scope of this project match the capacity of an engineering project, the PID flight controller has been designed from scratch based on Arduino Microcontroller and MPU6050 as the IMU. This is the ultimate challenge of this project since it forms the basis of achieving stable flight which also facilitates other applications to be realized.

On the application, Android Studio has been used to develop the Android app that helps in real-time air quality monitoring. Incorporation of Google Cloud Platform: Firebase – the real-time cloud database has made this achievable.

## CHAPTER TWO

### 2. REVIEW OF LITERATURES

#### 2.1 OVERVIEW

This describes the model of a quadcopter with PID controller which is Arduino-based mounted ESP8266 for IoT based solution. This quadcopter uses carbon fiber frame to reduce the weight of the quadcopter. The flight was able to lift relatively more payloads due to the use of 920KVA brushless DC motor. However, the quadcopter has the challenge of less flight time due to more weight and considerate low power rating of the LiPO battery as per the specifications. The hardware, electronics, and software were carefully integrated to ensure a stable flight is achieved through a PID controller.

The proposed and designed system and algorithm is necessary to allow a quadcopter to take a stable flight, hover when appropriate, and auto-level when there is any disturbance such as wind. The quadcopter was able to fly in place but could not carry much payload and lift comfortably due to more weight and less power rating from the battery. It was also meant for air quality monitoring which led to adding more component that affected the weight distribution causing imbalance and short battery life. The air quality monitor has an android application installed in a smartphone to monitor the parameters through GUI.

The project has been developed in a manner that the quadcopter is capable of taking a stable flight from a different environment and be able to monitor gas leakages and various air quality. The project was a success since the parameters from the sensors could be, and the thrust, pitch, yaw, and roll of the controller could be adequately felt. The quadcopter, however, could not resist much vibration and lacked stability due to: unbalanced masses, more weight, and low battery power rating as compared to load, and inconsistency data from the MPU6050.

Thus, amongst all the challenges above, this project is envisioned to design and implement a fully working Arduino-based PID quadcopter controller for multi-purpose applications but limited to gas leakage and air quality monitoring for this project application. The quadcopter could be capable of autonomous flight and can be able to achieve a nominal flight altitude of

fewer than 50 meters for safe operations and also following the state law: KCAA Drones Regulations. The total mass of the quadcopter is above 2kg, and the battery life can endure for at least 15 minutes. In all, the quadcopter is be capable of carrying moderate payloads such as extra controller and gas sensors among other custom devices and components like improvised propeller guides.

## 2.2 QUADCOPTER HARDWARE COMPONENTS

Understanding different components that are used in quadcopter making is essential. This helps on get a deeper understanding when examining quadcopter dynamics and states involved. Apart from examining dynamic states, many factors become a key player when designing the controller of the quadcopter. These components include:

### 2.2.1 Flight Controller

This is the brain of the quadcopter. Its main function is to provide control for all the interfaced inputs and output devices. Arducopter is a flight controller that is already preprogrammed and enabled plug-and-play with no coding and with only software (GUI – based configurations). It is one of the popular



Figure 2: Arducopter Flight Controller



Figure 1: Arduino Uno Microcontroller

controllers used although for this project Arduino Uno has been used as a microcontroller in implementing the flight controller. Code is written according to the flight specifications then uploaded to the Arduino where all the computations and processing during the control will be taking place.

### 2.2.2 Frame

This provides the physical structure for the entire quadcopter. It joins the motors, ESCs and also houses other components. The frame must be broad enough to allow all propellers to spin properly without collision, but must also not be too large hence being too heavy for the motors. For this



Figure 3: Quadcopter X-Frame

project, a 450mm frame has been used as seen in Figure, which measures at 450mm across opposite motors. Therefore, this makes up the body of the quadcopter with arms, landing gears, mounts, power distribution boards among other components. The X-frame is chosen because of its better symmetry, simpler design, and more flexibility and suitable for abrupt tilting.

### 2.2.3 Motor

Selection of motors depends on the size of the payload the drone is expected to carry among other factors. Rotors of lower rpm should power a larger propeller because of the higher air resistance. In a quadcopter, two motors will be rotated clockwise and two motors anti-clockwise, depending on how they will be connected to the ESC output.



Figure 4: Brushless DC Motors

### 2.2.4 Electronic Speed Controller (ESC)

ESC draws current to drive the motor based on output from the controller's commands. The maximum allowable current that can flow through the ESC determines its rating. Therefore, the selection of ESC is supposed to be such that the ESC rating is 1.2 to 1.5 times the maximum rating of the rotor; according to the equation:



Figure 5: ESCs

$$\frac{\text{ESC rating}}{\text{Maximum rotor rating}}$$

### 2.2.5 Propellers

The props have to be selected based on the intended reliability and the sized according to the motor speed. Carbon fiber propellers are of higher quality but may not be cost-worthy. After all, carbon fiber propellers also will break under similar circumstances of impact. Softer propellers, on the other hand, may even be more flexible

and less fragile. They are also less stiff in resisting air, thus may be better at reducing possibilities of motor heating in large drones. Balancing propellers minimize vibrations that could arise from unbalanced propellers.



Figure 6: Propellers

### 2.2.6 Transmitter

A high-grade transmitter is what required to enable good control of a quadcopter. This is vital because otherwise, fatal injuries could be the result following impaired communication. The transmitter comes as a pair with the receiver. The transmitter operates at 2.4GHz. It can transmit for more than two kilometers provided that all factors are kept constant. It also operates on different modes and also several other capabilities like onboard battery monitor.



Figure 7: Transmitter

### 2.2.7 Receiver

The radio receiver (Rx) receives radio signals from an RC transmitter and converts them into control signals for each control channel (throttle, yaw, roll & pitch). Modern RC receivers operate on a 2.4 GHz radio frequency. Rx units may have as few as four channels, but many have more channels for additional control options. FS-iA6 Rx is a 6 Channel Receiver selected for this project. It is highly recommended to use a transceiver pair whose receiver can communicate with the controller through the SBUS communication protocol. SBUS is a new single-wire communication protocol that is supported by several controllers.



Figure 8: Receiver

### 2.2.8 Lithium-Polymer (Li-Po) Battery

Lithium battery has one of the highest power-to-weight ratios, making it one of the best choices for powering UAVs. However, Li-Po battery has to be taken care of thoroughly and have its voltage monitored so that each cell stays at least 3V (3.4V to be safe) before it is being recharged. The Li-Po battery used is a 2200 mAh battery. To determine if the discharge limit, C is sufficient for the motors used, the maximum discharge current is calculated from.

$$I_{max} = C * Ah$$

### 2.2.9 Battery Monitor

To protect the Li-Po battery from possibilities of over-discharging, a BB alarm – a loud warning alarm when battery voltage of any cells of the Li-Po dropped below-set values – is in place to monitor the battery voltage from time to time. If the battery is allowed to discharge under 3 V per cell, it can be permanently damaged and never be able to recharge back again.



Figure 9: Li-Po Battery

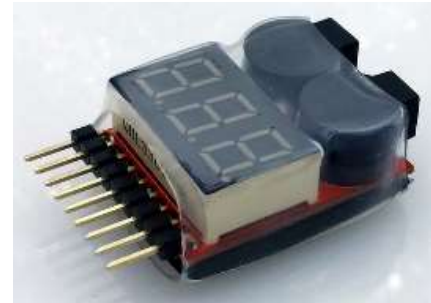


Figure 10: Battery Monitor

### 2.2.11 MPU6050 Sensor

MPU6050 sensor is a combination of both gyroscope and accelerometer. This sensor provides the flight controller with readings of the quadcopter's orientation. At the minimum, this requires a gyroscope, but most quadcopters also incorporate an accelerometer. The two components are all available on the same board. For a self-leveling and self-stabilizing drone, an accelerometer is required. For a quadcopter application, a 6-axis inertial measurement unit (IMU) is desired,

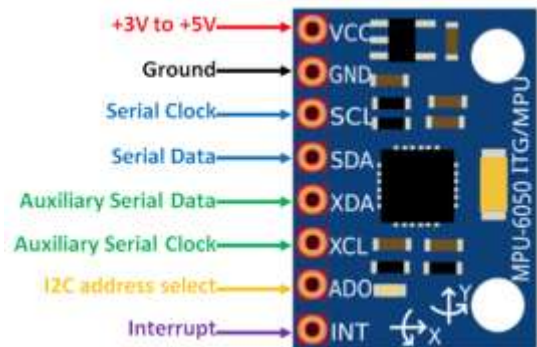


Figure 11: MPU6050 Sensor



consisting of a gyroscope (3-axis) and an accelerometer (3-axis) on the same board. This board includes an ADXL345 accelerometer, an ITG-3200 gyroscope all coupled in the same board. All the parameters help in making the quadcopter achieve the desired flight based on its rotational angles and acceleration giving the quadcopter a defined position and altitude.

## 2.3 OTHER HARDWARE COMPONENTS

Apart from making a quadcopter, this project also requires the implementation of the drone-based air quality/gas leakage monitoring systems. This system requires a different controller and sensors since based on the interrupts created on the quadcopter microcontroller (Arduino Uno), other commands cannot be executed in the same microcontroller. Otherwise, if the same microcontroller is used, there would result in a delayed signal between the transmitter and receiver resulting in not having a real-time response of the quadcopter. This system will work independently but with enhanced capabilities offered by the quadcopter. All these systems are mounted on the quadcopter taking into consideration the weight balance.

### 2.3.1 ESP8266 (Node MCU)

This is the microcontroller with a Wi-Fi enhanced capabilities. It used the Arduino IDE to program it although it has different setting for it be fully compatible. With the Wi-Fi capabilities, the systems is able to have a stable implementation of cloud-based IoT system. ESP8266 is also known as Node MCU. The limitation of this module is that it only has one analog pin / port. Although this can be extended by application of other multiplexing components.



Figure 12: ESP8266

### 2.3.2 Gas Sensor

MQ-2 gas sensor is one among the many gas sensors that can be used in gas detection and air quality monitoring. This sensor detects smoke and combustible gases. This sensor is useful for home and industry applications. It is appropriate for sensing smoke, methane, LPG, alcohol, carbon monoxide, hydrogen, and propane concentrations or leakages in the air. This gas sensor may need to have sensitivity increased when using it on outdoor applications. The sensor utilizes 5V DC and draws 800mW. This sensor detects a concentration between 200 to 10000ppm.



Figure 13: MQ-2 Gas Sensor

### 2.4 MISCELLANEOUS COMPONENTS

Apart from the main components of the project, there are other essential and critical components to make the project a success. The following parts are also needed:

*Zip-ties*: - To hold cables and other components tightly.

*Soldering iron*: - To make ideal connections

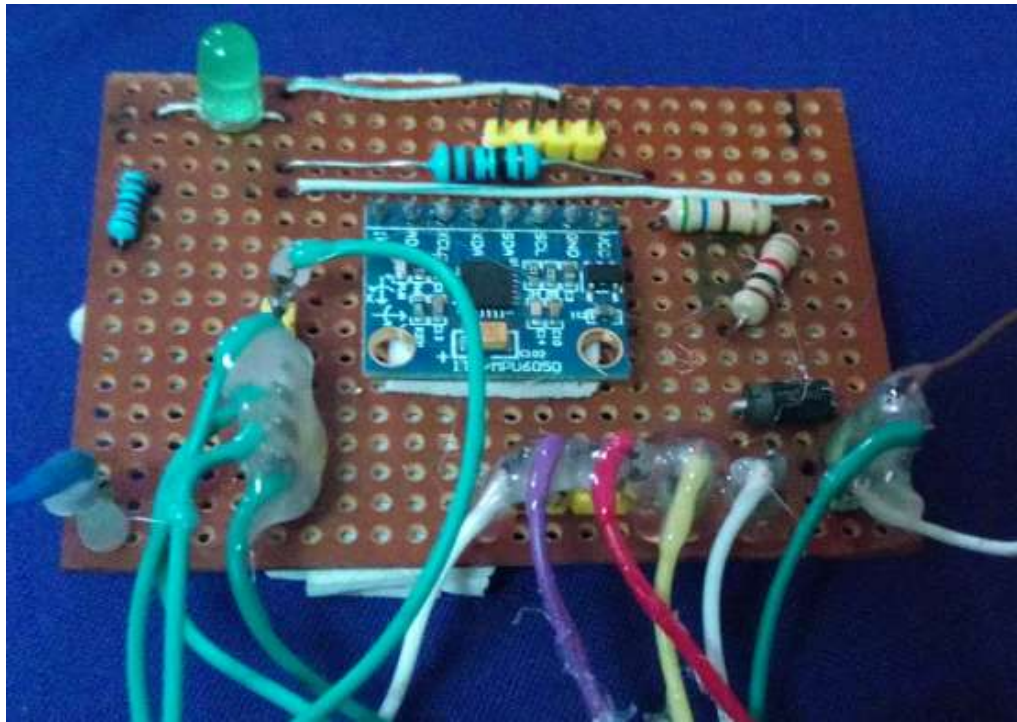
*Connector cables and jumper wires*: - For connecting components

*Hot glue*: - To stick some pieces together that cannot be tightened by the zip-ties.

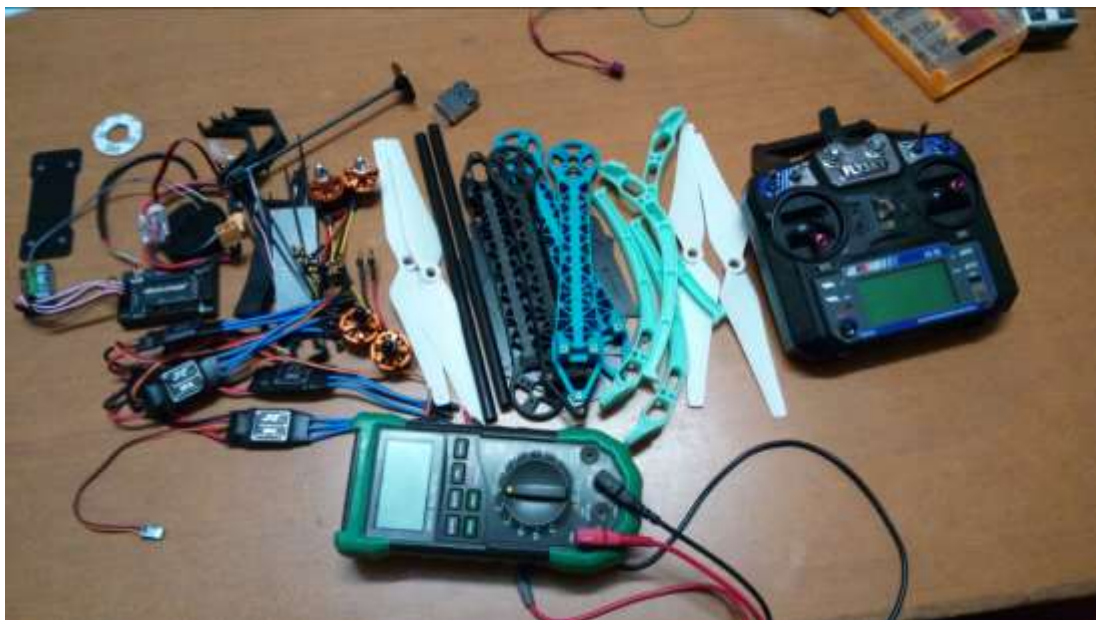
*Double-sided tape*: - To hold the MPU6050 sensor on the frame. This enables vibration damping which affects the sensitivity and the parameter input of the detector.

*Resistors*: - This helps during the conversion of voltage to be used in both motors and controller electronics which consumes 12V DC and 5V DC respectively.

*LED*: - This is used as an indicator for the status of the mode of operations. It indicates status especially when having initial configurations and also indicating status such as fail-safe when setup has failed.



*Figure 14: Strip Board Circuit*



*Figure 15: Quadcopter Components*

## 2.5 SOFTWARE AND OTHER PLATFORMS

To enhance project development and implementation, several software and platforms has been used. This include:

- i. Android Studio
- ii. Firebase
- iii. Fritzing

### 2.5.1 Android Studio

This is the software that has been used to create the Android Application that is used to monitor the real-time change of the air quality and also the availability of the gas leakages. The custom application developer has to be installed to a smartphone or any gadget that can support the Android Operating System. The application requires an internet connection to operate since it relies on a cloud connection for data availability.

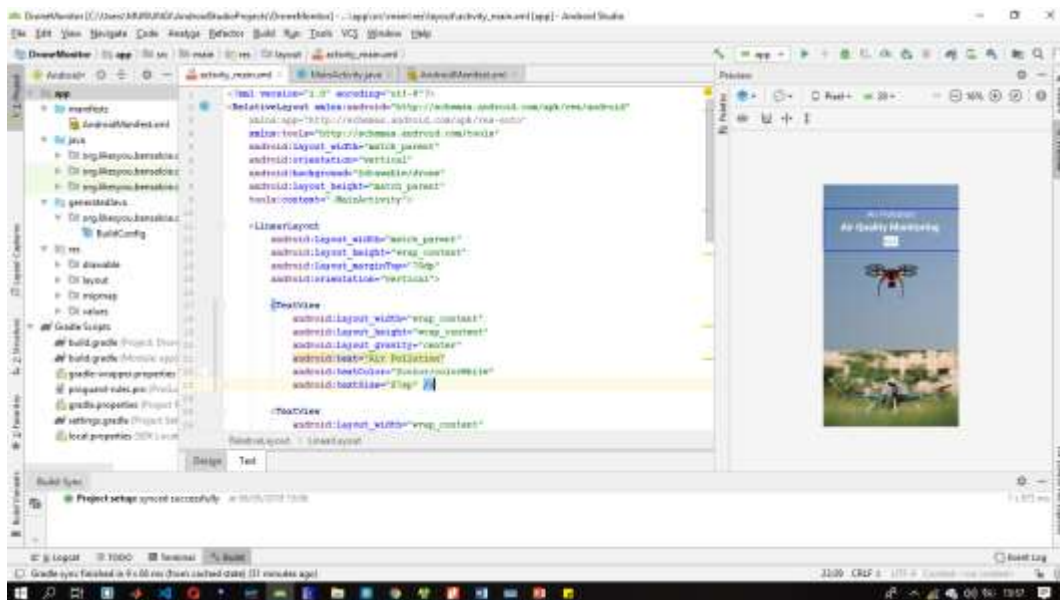
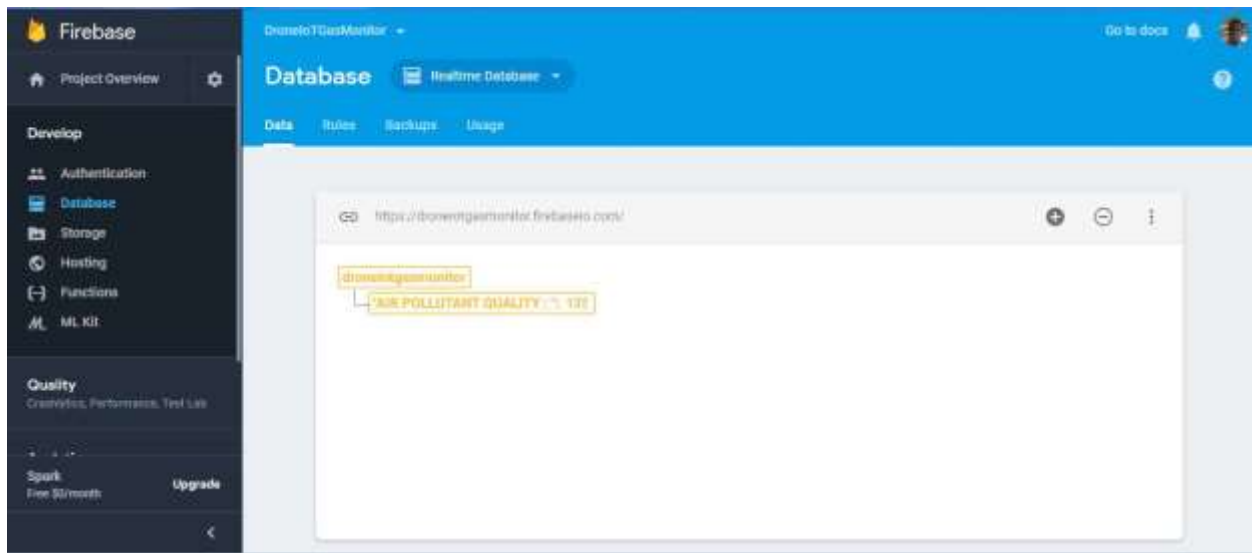


Figure 16: Android Studio

### 2.5.2 Firebase: Google Cloud Platform

This is the cloud platform that offers access to a real-time database that allows access to real-time updates from the sensor to the mobile application. This platform enables extended applications like analytics and predictions based on the gathered data. Firebase can also be incorporated with other web-based services or IoT operating system found.



*Figure 17: Firebase*

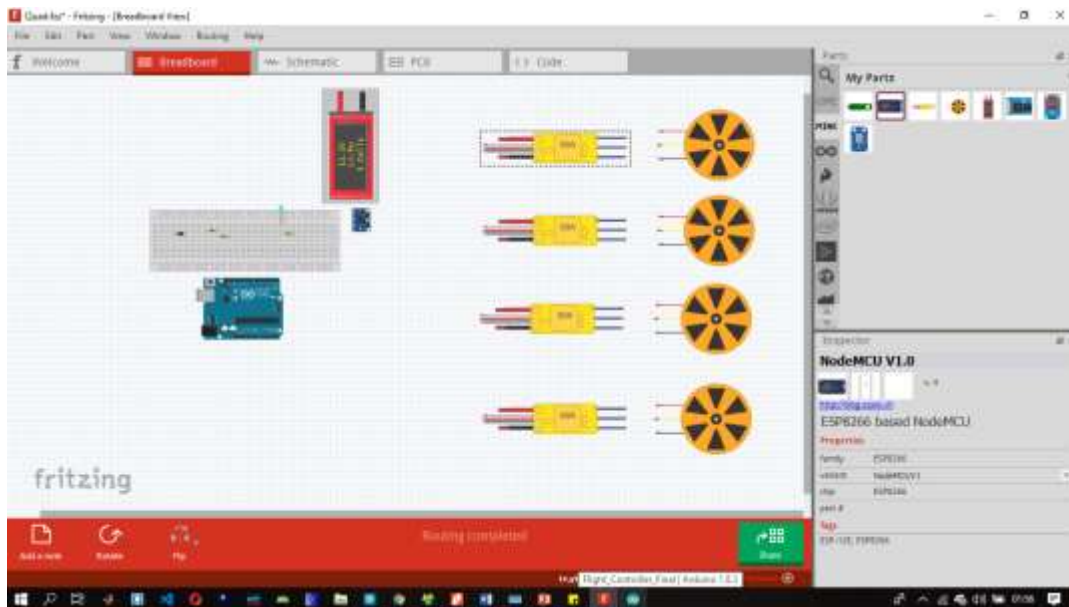
### 2.5.2 Fritzing

This is a platform that is used during hardware interfacing. It provides more visual-friendly diagrams that are easy to follow and which are easily understood. This platform offers the user the ability to wire and interface components similar way that one does on a breadboard. After the hardware interfacing is done, schematic and PCB representation of the following interface can be generated from the initial design. The software is easy to use since changes are updated on a real-time basis on all the three different models.

This software also provides the ability to the continuity of the connections ensuring that all links are closed and complete. Apart from hardware design and circuit generation (schematic and PCB), the software also provides an avenue for coding and having a real simulation in the same platform. This allows the designer to all the four (4) services provided in one platform. This



ensures ease in designing, testing and prototyping components and devices with more visual aids.



*Figure 18: Fritzing*

### 2.5.3 Other Platforms

Other online resources have also been used for the provision of extra services to enhance project delivery. These platforms include “draw.io” among others that are very resourceful in flow diagram designs.

## CHAPTER THREE

### 3. DESIGN METHODOLOGY

#### 3.1 OVERVIEW

The theory of how quadcopters are designed, built, and operated is generally straightforward. The challenge lies in implementing each subsystem required. The implementation requires a high level of understanding and attention to details for the quadcopter to function correctly. This chapter highlights the design methodology that was followed for the implementation of the quadcopter and the other component of the project and details of how each subsystem works. The first phase of the project highlights the mechanics and system design of the quadcopter while the second phase involved the implementation of different components and finally testing and analysis of the results obtained.

#### 3.2 MECHANICS AND SYSTEM DESIGN

This means several other steps or parts that are comprised of the detailed methodology in quadcopter designing. These include:

- i. Basic Mechanics
- ii. Controls
- iii. Design Considerations
- iv. Agility
- v. Components Selection
- vi. Effects of Size

##### **3.2.1 Basic Mechanics**

This deals with the basic rotor mechanics of a quadcopter that relates the: Thrust force (N), Speed (rpm), and Drag moment (NM). These three components are being represented using 3-axis since they are related either directly or indirectly with one another.

All the three elements can be represented as in the below graph

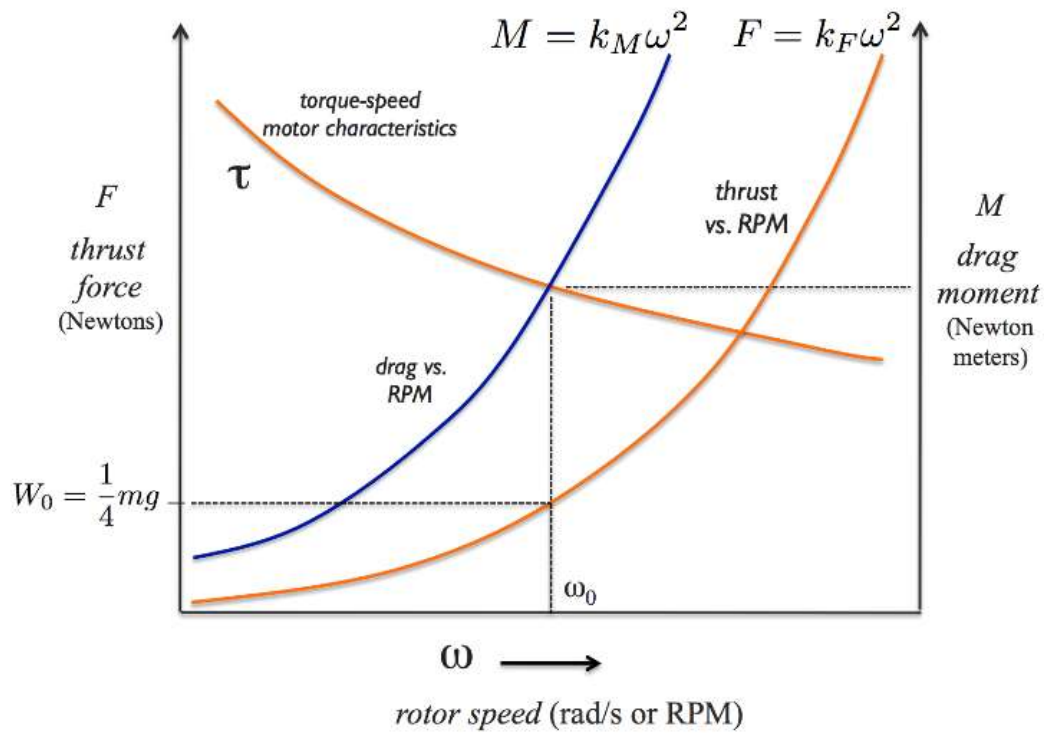


Figure 19: Force, Moment Vs Speed

### 3.2.1.1 Motor Speed

$$k_f w_i = \frac{1}{4} mg$$

### 3.2.1.2 Motor Torque

$$T_i = k_m w_i^2$$

### 3.2.1.3 Motor Force

$$F_i = k_f w_i^2$$

### 3.2.1.4 Motor Moments

$$M_i = k_m w_i^2$$

The below diagram represents all the four components put together in a free fall body representation of a quadcopter. Each motor has a Speed, Torque, Force, and Moment. Each of



these components plays a major role when it comes to controls of the quadcopter to attain a stable flight.

$k_m$  and  $k_f$  = constants of proportionality

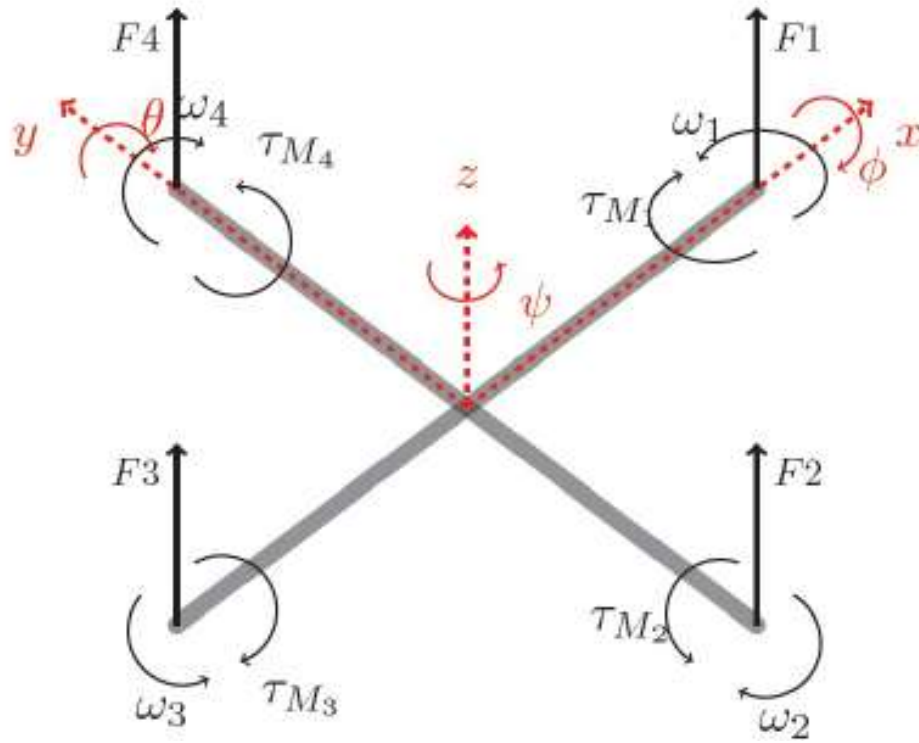


Figure 20: Force, Moment, Torque, & Speed

Combination of the four elements creates the inertia due to forces and torque acting on the rotors of the quadcopter as represented below.

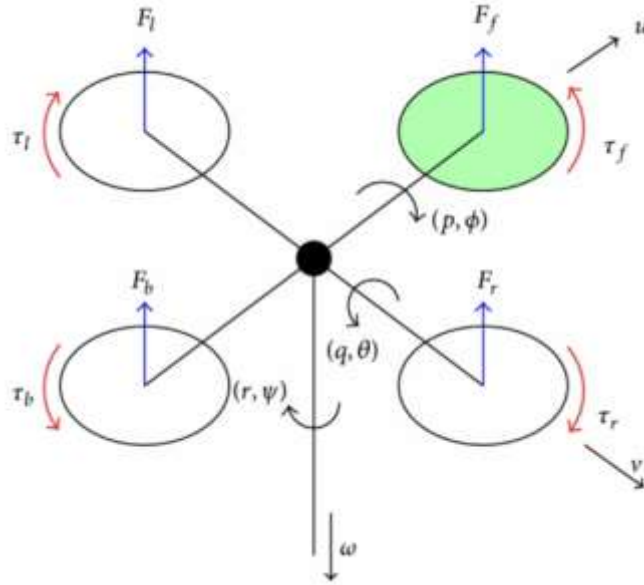


Figure 21: Forces & Torque on the rotors

### 3.2.1.5 Resultant Forces ( $F$ )

$$F = F_1 + F_2 + F_3 + F_4 - m g a_3$$

### 3.2.1.6 Resultant Moments ( $M$ )

$$M = r_1 * F_1 + r_2 * F_2 + r_3 * F_3 + r_4 * F_4 + M_1 + M_2 + M_3 + M_4$$

### 3.2.1.7 At Equilibrium

$$F = 0 \quad ; \quad M = 0$$

When  $F \neq 0$  and  $M \neq 0$  ; you get Acceleration

### 3.2.1.8 Vertical Acceleration (Thrust)

Each motor gets equal thrust during vertical acceleration. Increase in motor speed results in upward thrust while the decrease in motor speed results in downward thrust due to deceleration.

At Standstill:  $\sum_{i=1}^4 k_f w_i^2 + m g = 0$

Acceleration / Deceleration:  $\sum_{i=1}^4 k_f w_i^2 + m g = m a$

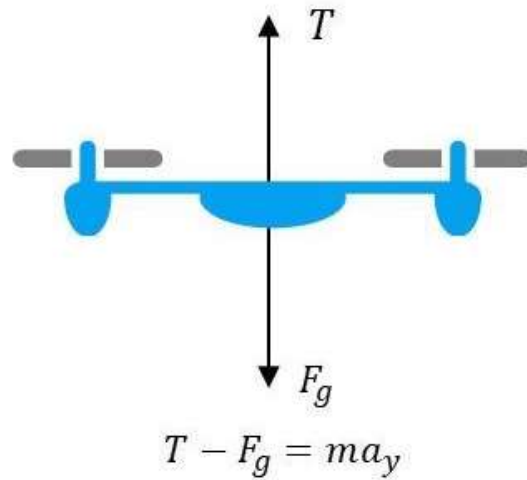


Figure 22: Thrust torque and force

Thrust generated at the propeller level are as follows:

### Quadcopter Propeller Thrust

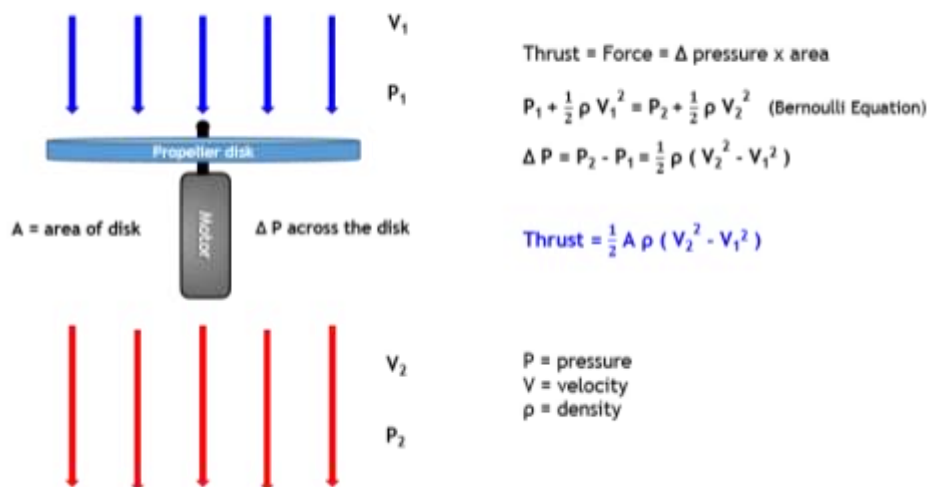


Figure 23: Quadcopter Propeller Thrust

### 3.2.2 Controls of a Quadcopter

The below diagram represents the general flow of control of a quadcopter. This represents all the 6 degrees of freedom that are controlled with the four motors in providing the desired output.

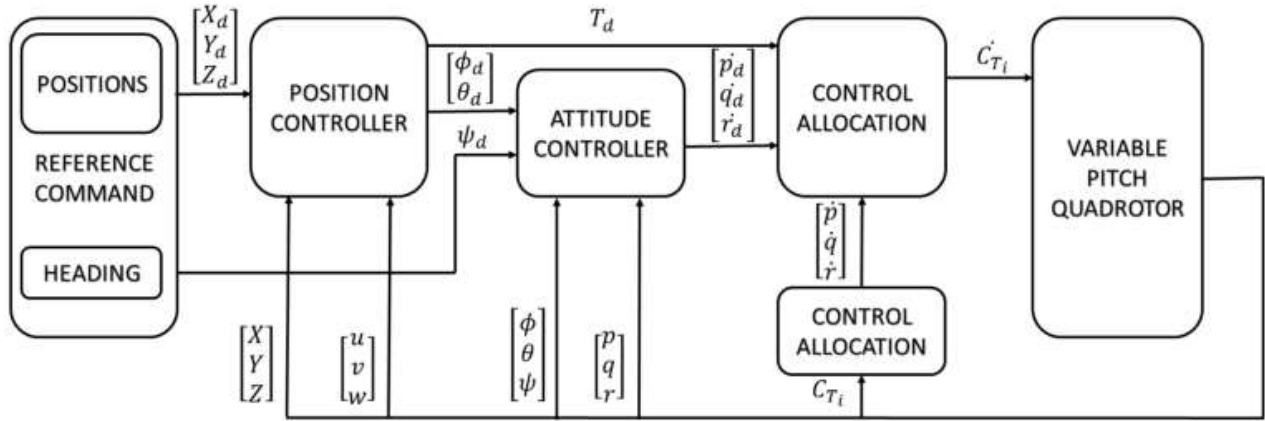


Figure 24: General Control of a Quadcopter

#### 3.2.2.1 Control of Height

Control of height can be achieved through the acceleration or deceleration to achieve the desired height. This is done through:

$$\sum_{i=1}^4 k_f w_{i^2} + mg = ma$$

$$a = \frac{d^2x}{dt^2} = \ddot{x}$$

$$\text{Input } u = \frac{1}{m} [\sum_{i=1}^4 k_f w_{i^2} + mg = ma]$$

Acceleration is represented by a 2<sup>nd</sup> Order Dynamic System

$$u = \ddot{x}$$

### 3.2.2.1.1 Control of a Linear 2<sup>nd</sup> Order System

State Input  $x, u \in \mathbb{R}$

Plant Model  $\ddot{x} = u$

To attain the control input of the quadcopter,  $u(t)$  is solved so that to achieve a desired trajectory  $x^{des}(t)$ .

### 3.2.2.1.2 General Approach

Desired Error,

$$e(t) = x^{des}(t) - x(t)$$

In order to work effectively;

$e(t)$  Has to converge exponentially to zero.

### 3.2.2.1.3 Strategy

$e(t)$  Has to meet the below conditions:

$$\ddot{e} + K_d \dot{e} + K_p e = 0$$

$$K_p, K_d > 0$$

$$u(t) = \ddot{x}^{des}(t) + K_d \dot{e}(t) + K_p e(t)$$

Where:

$\ddot{x}^{des}$  - Feed Forward

$K_d$  - Derivative Gain

$K_p$  - Proportional Gain

General representation of a PD Control:

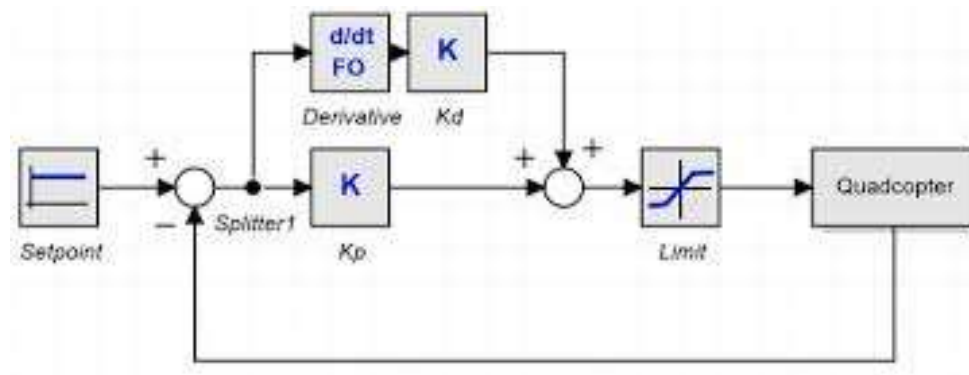


Figure 25: PD Control

Therefore, this results into a PD Controller as shown below.

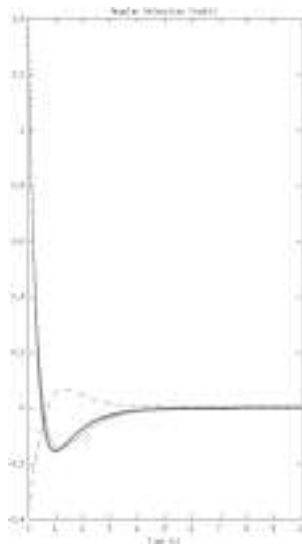


Figure 26: PD Controller Response

Proportional control acts like a spring (capacitance) response while Derivative control has a viscous dashpot (resistance) response. Large derivative gain ( $K_d$ ) makes the system overdamped, and the system converges slowly.

#### 3.2.2.1.4 PID Control

In the presence of any disturbance like wind or when modeling errors like unknown mass, it is often advantageous to use PID Control.

$$u(t) = \ddot{x}^{des}(t) + K_d \dot{e}(t) + K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

Where:

$K_i \int_0^t e(\tau) d\tau$  - Compensate Unknown Components

PID Control generates a 3<sup>rd</sup> – order closed-loop system. The integral control component makes the steady-state error go to zero.

General representation of the PID Control:

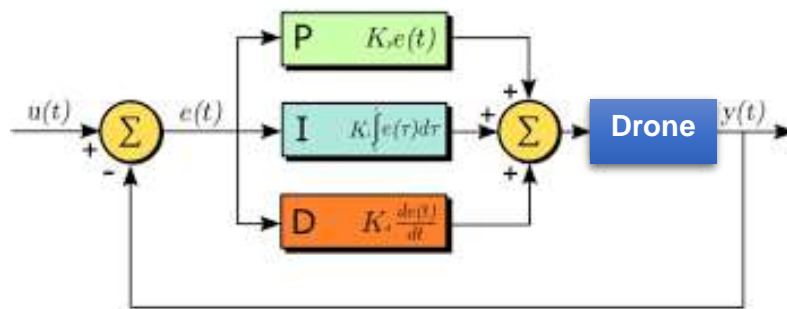


Figure 27: PID Control Representation

General response of a PID Control response:

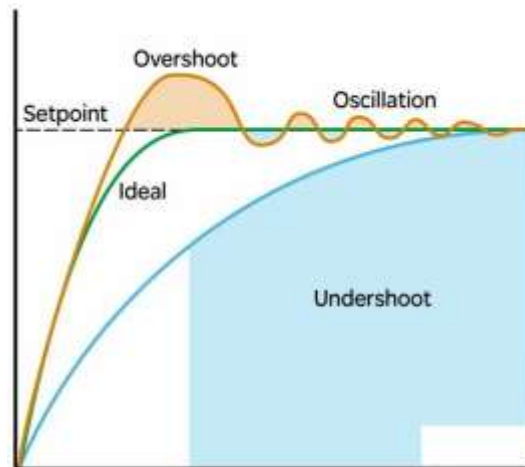
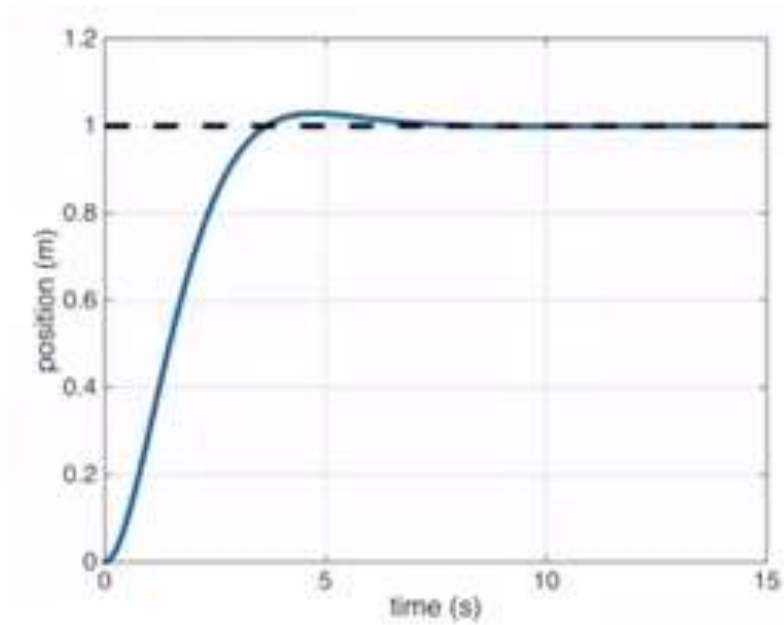


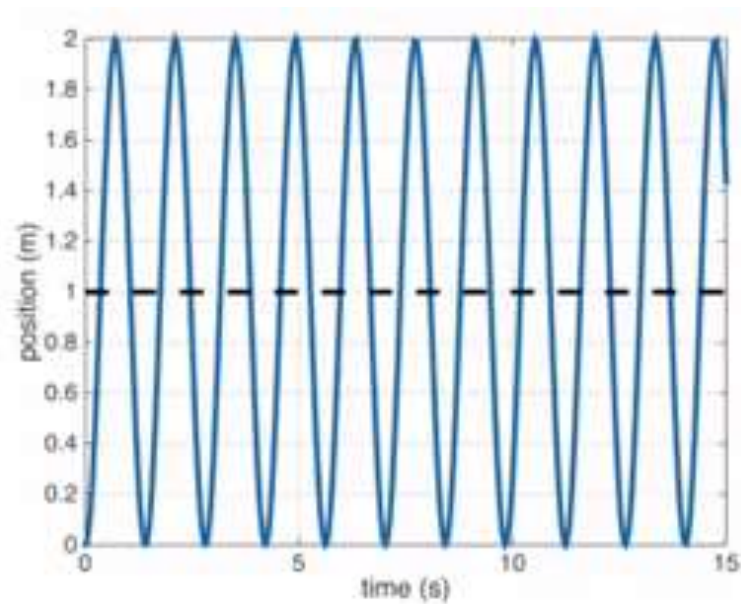
Figure 28: PID Control Response

Stable ( $K_p, K_d > 0$ )



*Figure 29: PD Stable Response*

Marginally Stable ( $K_p > 0; K_d = 0$ )



*Figure 30: PD Marginally Stable Response*



Unstable ( $K_p$  or  $K_d < 0$ )

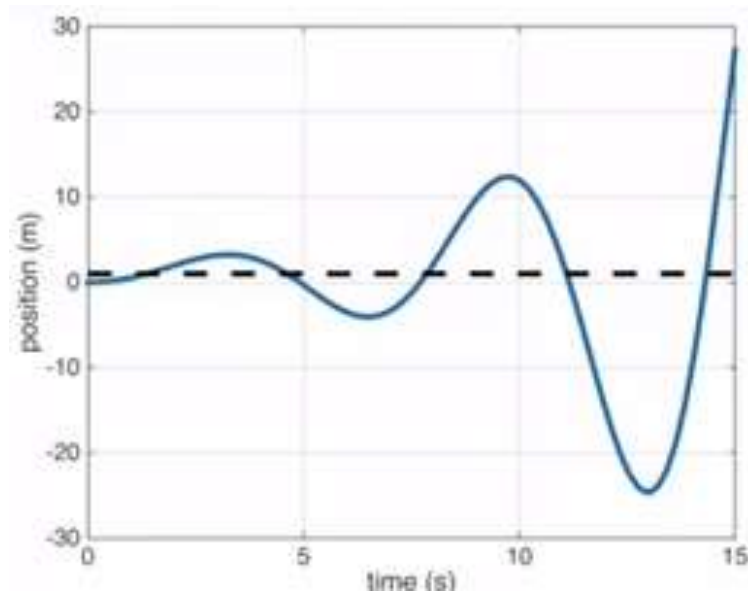


Figure 31: PD Unstable Response

PD Controller

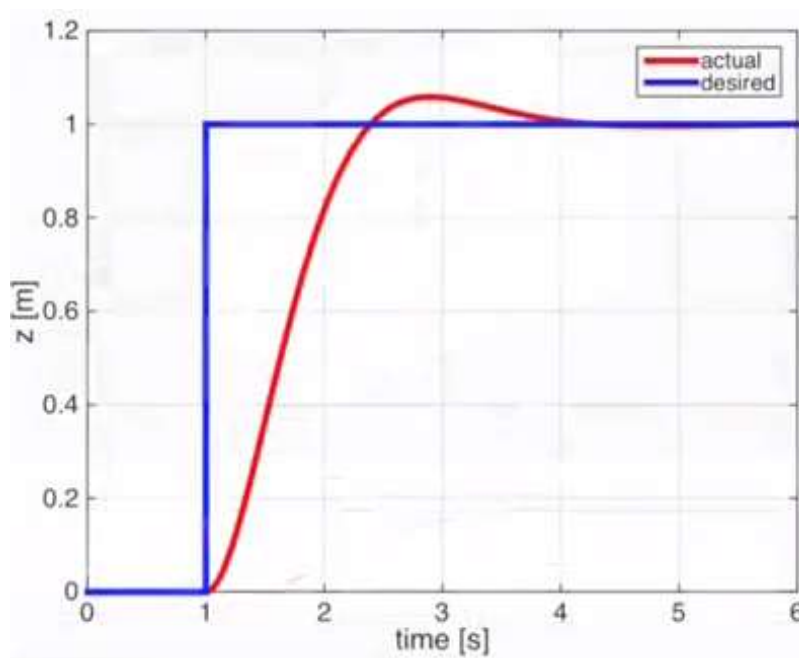
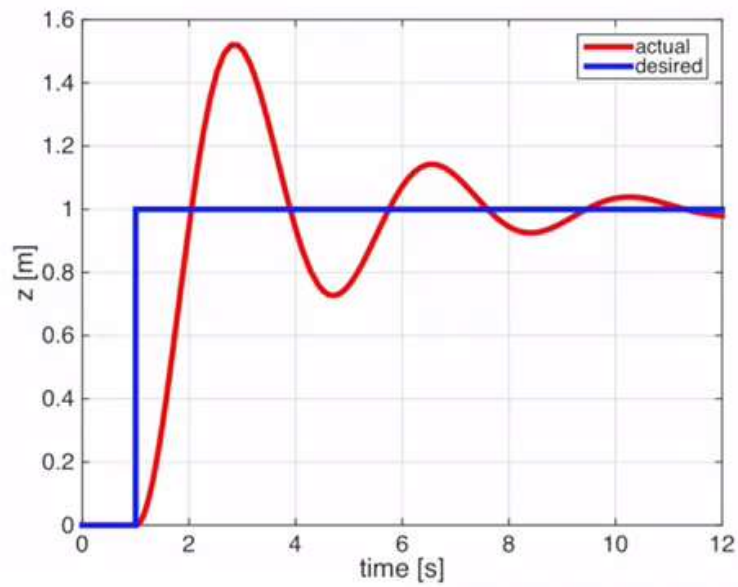
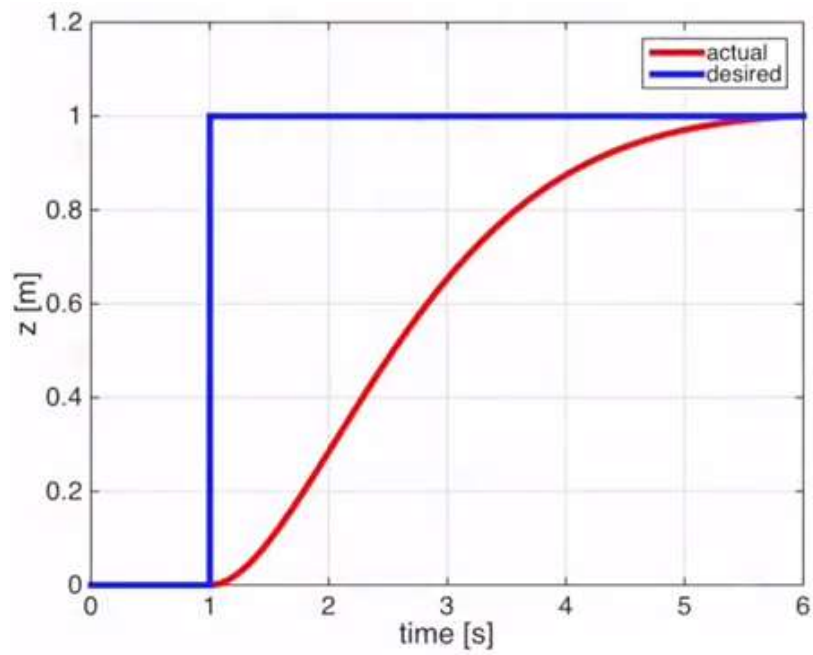
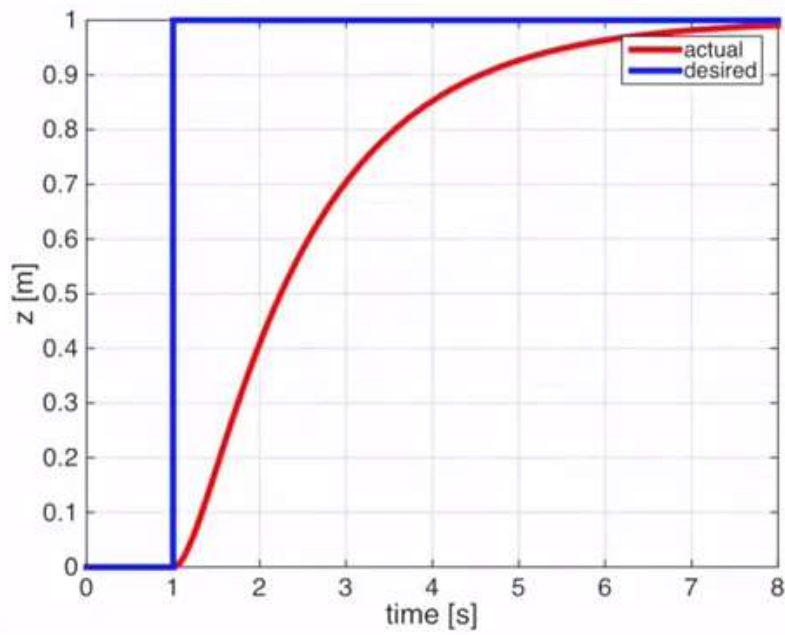


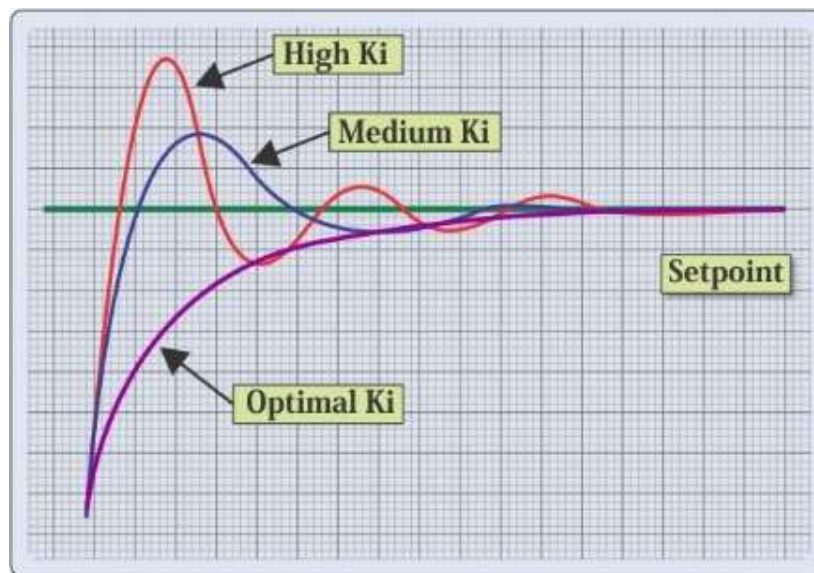
Figure 32: PD Controller General Response

High  $K_p$ *Figure 33: High  $K_p$  Gain Response*Low  $K_p$  (Soft Response)*Figure 34: Low  $K_p$  Gain Response*

## High Kd (Overdamped)

*Figure 35: High Kd Gain Response*

## Ki Response

*Figure 36: Ki Gain Response*

### 3.2.3 Design Considerations

#### 3.2.3.1 Thrust versus Weight

##### 3.2.3.1.1 Control with Thrust Limitations

Effects of Maximum thrust on input can be determined by:

$$u = \frac{1}{m} \left[ \sum_{i=1}^4 k_f w_{i^2} + mg \right]$$

$$= \frac{1}{m} [T \uparrow + mg \downarrow]$$

Where:

T - Thrust

Mg - Weight

Maximum input is also determined by the Maximum Thrust provided by the motor torque.

$$U_{max} = \frac{1}{m} [T_{max} \uparrow + mg \downarrow]$$

$T_{max}$  - Maximum Thrust; as determined by the peak motor torque.

These limits results into the following control limitations:

##### 3.2.3.1.2 PD Control

$$U(t) = \min(\ddot{x}^{des}(t) + K_d \dot{e}(t) + K_p e(t), U_{max})$$

##### 3.2.3.1.3 PID Control

$$U(t) = \min(\ddot{x}^{des}(t) + K_d \dot{e}(t) + K_p e(t) + K_i \int_0^t e(\tau) d\tau, U_{max})$$

### 3.2.3.1.4 Effects of Thrust/Weight Ratio

Increasing the weight while using the same motors decreases the thrust and vice versa. This takes effects of the size of the propeller as well. The effect of weight on motors is the same as a weight to propellers.

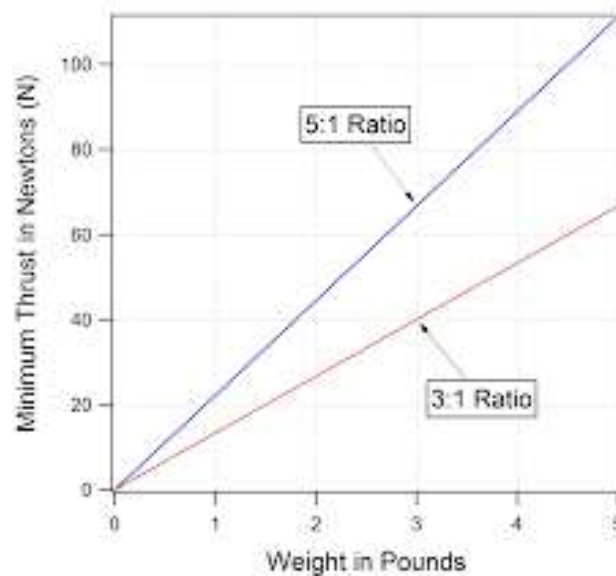


Figure 37: Thrust Vs Weight Ratio

$$\frac{\text{Thrust}}{\text{Weight}} = 5:1$$

$$\frac{\text{Thrust}}{\text{Weight}} = 3:1$$

### 3.2.3.2 Power, Thrust, and Energy

These three components determines:

- ✓ Battery selection,
- ✓ Power Consumption, and
- ✓ Total Energy carried by the battery.

Lithium Polymer (LiPO) battery are the most preferred since they provide high energy and power although they are costly.

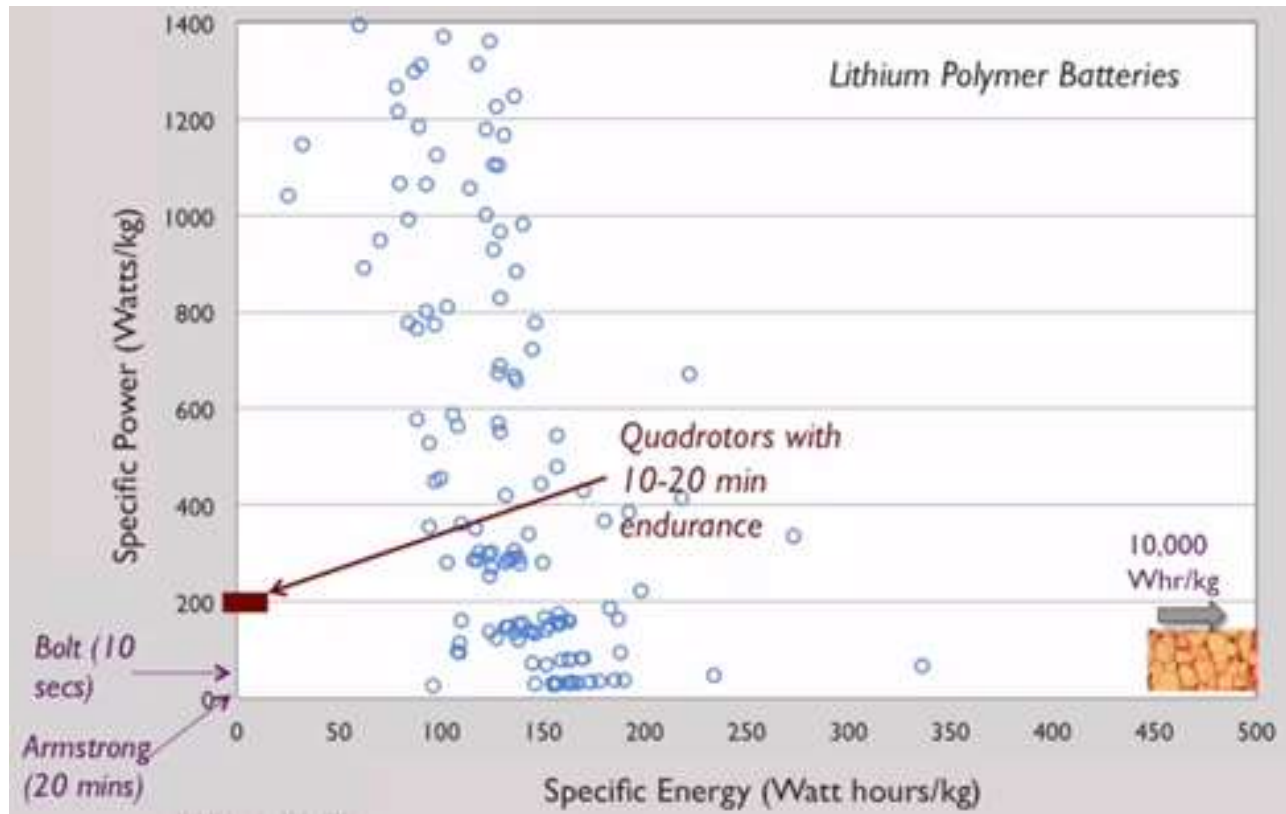


Figure 38: Li-Po Specific Power Vs Specific Energy

On average, a well-balanced quadcopter hovers at 200Watts/Kg which typically provides 10-20 minutes endurance. This is an inefficient mechanism while it comes to testing the energy supplied by the battery.

Li-Po battery size based on quadcopter size:

Quadcopter Size/mm	Li-Po Battery Size/mAh	Number of Cells/S
Mini Quad	80-800	1 or 2
180	1000-1300	3 or 4
210	1000-1300	3 or 4
250	1300-1800	3
280-290	1500-3300	4
330-360	2200-3200	4
400	3200-3300	4
450	3300	4
500	3300-5000	4
540	5000-5200	4
550-750	5000-8000	4 or 5 or 6
800 or bigger	8000-30000	6

*Table 1: Li-Po Battery Specifications*

Courtesy of dji.com

### *3.2.3.3 Mass Distribution*

In a well-balanced quadcopter, battery, motors, and propellers contribute more than 50% of the total weight of the quadcopter. For example:

$$\text{Battery} = 33\% \text{ Mass}$$

$$\text{Motors} + \text{Propellers} = 25\% \text{ Mass}$$

These factors have effects on energy either during normal operations or when the quadcopter is in motion. For example, an 80grams camera has 1.5 Watts of normal operation and 15 Watts during mobility.

### 3.2.3.4 Quadcopter Motion Ratios

The calculation of motion ratios of a quadcopter is essential to make sure the quadcopter can take-off, pitch, yaw, roll, and fly stably with desired payloads.

Calculating the weight of the quadcopter with assumed values:

Components	No. of Quantity	Mass per Quantity (g)	Total Mass (g)
Frame	1	460	460
Battery	1	420	420
Motors	4	60	240
ESCs	4	20	80
Propellers	4	50	200
BB Alarm	1	25	25
Power Distribution Board	1	60	60
Custom Propeller Protector	4	80	320
Cables and Connectors	-	100	100
ESP8266	1	10	10
MPU-6050	1	3	3
Circuit Board	1	50	50
MQ-2 Gas Sensor	1	5	5
Miscellaneous	-	100	100
<b>TOTAL</b>			<b>2073</b>

Table 2: Quadcopter Mass Distribution



The total assumed mass of the quadcopter without any payloads is around 2 kg. For the drone to be able to take off, the assumption is that the thrust that the rotors produce in total has to exceed that value, or twice of that.

$$\text{Total thrust} = 2 * \text{Total Weight of Quadcopter}$$

The generated thrust of a quadcopter can be calculated from the equation:

$$m = \frac{T}{g} = \frac{\sqrt[3]{\frac{\pi}{2}} D^2 \rho P^2}{g}$$

Where,

T = Thrust (N)

D = Propeller diameter (m)

$\rho$  = Density of air (1.225 kg/m<sup>3</sup>)

P = Power of rotor (W)

m = Equivalent mass of thrust

g = Gravitational acceleration (9.81 m/s<sup>2</sup>)

In most cases, these calculations involved can be very complex and therefore it is advisable to rely more on data sheets from the suppliers.

Thus the thrust-to-weight ratio of the quadcopter without any payloads:

$$\text{ratio} = \frac{\text{equivalent mass of thrust}}{\text{flight mass of quadcopter}}$$

### 3.2.4 Agility

This determines the rate at which the quadcopter starts at rest and accelerate to rah maximum speed very quickly.

Maximum velocity to rest:

To maximize agility, stopping distance has to be minimized.

Turn quickly without slowing down:

To maximize agility, minimize turning radius.

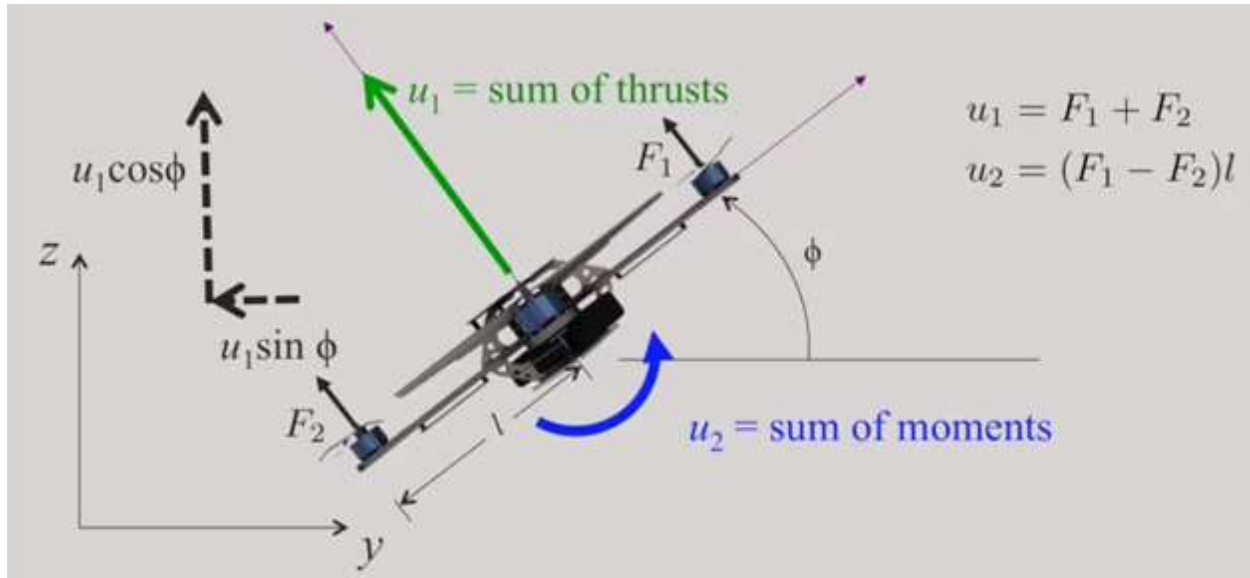


Figure 39: Agility Explained

$$\begin{aligned} \text{Linear Acceleration, } a \left\{ \begin{bmatrix} \ddot{y} \\ \ddot{z} \end{bmatrix} \right. &= \begin{bmatrix} 0 \\ g \end{bmatrix} + \begin{bmatrix} \frac{1}{m} \sin \phi & 0 \\ \frac{1}{m} \cos \phi & 0 \\ 0 & \frac{1}{I_{xx}} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \\ \text{Angular Acceleration, } \alpha \left\{ \begin{bmatrix} \ddot{\phi} \end{bmatrix} \right. & \end{aligned}$$

Where:

$I_{xx}$  – Inertia

Key factors to be considered in agility include:

- i. Ability to accelerate quickly (*Linear*:  $a_{max}$ )
- ii. Ability to roll/pitch quickly (*Angular*:  $\alpha_{max}$ )

$$a_{max} = \text{Maximize } \frac{U_1, \max}{W}$$

$$\alpha_{max} = \text{Maximize } \frac{U_2, \max}{I_{xx}}$$

### 3.2.5 Components Selection

Some of the basic hardware of the quadcopter include:

- i. Frame, motors, propellers,
- ii. Controllers (Arducopter and Pixhawk)
- iii. Communication and Processing (Remote Control – Transmitter & Receiver, Telemetry)

Several standards exist in a component selection based on pre-determined platforms.

These predefined standard component match with limitations of changing the provided parameters. These parameters are supplied with several constraints put in place. They are altering any parameter results in changing several other functions to suit the required specifications.

The arrangement of the airframe size has to match with those of the batteries, propellers, as well as motors, as shown in the table.

Frame Size/mm	Prop Size/in	Motor Size	Motor Speed/KV	Li-Po Size/mAh
<b>120/Smaller</b>	3	1104-1105	4000+	80-800 1S/2S
<b>150-160</b>	3-4	1306-1407	3000+	600-900 2S/3S
<b>180</b>	4	1806-2204	2600+	1000-1300 3S/4S
<b>210</b>	5	2204-2206	2300-2700	1000-1300 3S/4S
<b>250</b>	6	2204-2208	2000-2300	1300-1800 3S/4S
<b>330-350</b>	7-8	2208-2212	1500-1600	2200-3200 3S/4S
<b>450-500</b>	9-12	2212-2216	800-1000	3300+ 3S/5S

*Table 3: Quadcopter Size Specifications*

Courtesy of dji.com

### 3.2.6 Effects of Size

The agility of a quadcopter is largely affected by the scaling of the size of different components of a quadcopter.

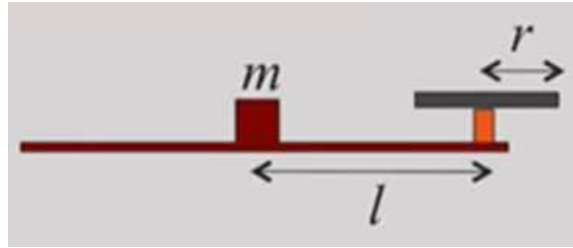


Figure 40: Quadcopter Size Configurations

Size of a quadcopter affects:

- i. Mass, Inertia
- ii. Thrust
- iii. Moment

#### 3.2.6.1 Mass, Inertia

$$m \sim l^3$$

$$I \sim l^5$$

$$r \sim l$$

#### 3.2.6.2 Thrust

$$F \sim \pi r^2 * (\omega r)^2$$

$$F \sim r^2 v^2$$

$$F \sim l^2 v^2$$

$$a \sim \frac{F}{m} \sim l^3$$

Where:

$(\omega r)^2$  - Motor Angular Speed

$v^2$  - Propeller/blade tip speed

### 3.2.6.3 Moment

$$M \sim l^3 v^2$$

$$\alpha \sim \frac{M}{I} \sim l^5$$

### 3.2.6.4 Maximum Acceleration

To attain the maximum acceleration without much disturbance and avoiding to overheat the motor, the following conditions have to be met.

$$a \sim \frac{v^2}{l}$$

$$\alpha \sim \frac{v^2}{l^2}$$

### 3.2.6.5 Scaling Principles

Size scaling of a quadcopter can be done based on two well-known principles:

1. Froude Scaling
2. Mach Scaling

#### 3.2.6.5.1 Froude Scaling

$$v \sim \sqrt{l}$$

$$a \sim 1 \quad ; \quad \alpha \sim \frac{1}{l}$$

#### 3.2.6.5.2 Mach Scaling

$$v \sim 1 = \text{Constant}$$

$$F \sim l^2$$

$$a \sim \frac{1}{l} \quad ; \quad \alpha \sim \frac{1}{l^2}$$

Therefore, the smaller the quadcopter, the larger the acceleration hence higher agility. This is the key principle used to differentiate racing drones and other types of drones.

### 3.3 ROTATION

The group of rotations is called Special Orthogonal Matrices. Coordinates for Special Orthogonal includes:

- i. Rotation Matrices
- ii. Euler's Angles
- iii. Axis Angle Parameterization
- iv. Exponential Coordinates
- v. Quaternions

All these coordinates contribute to ensuring quadcopter mechanics related to rotations are understood properly.

Therefore, Rotation about the X-axis through( $\theta$ ):

$$Rot(x, \theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

#### 3.3.1 Quadcopter Coordinate System

A quadcopter employs different control mechanism such as roll, pitch, and, yaw which in most cases are represented by the angle of rotation around the center of the quadcopter. These angles make up for the control of the altitude of the quadcopter and to track the altitude of the quadcopter, a two-coordinate system is required. There is the body frame system, which is attached to the quad at its center of gravity, and the earth frame system, which is fixed to the earth, and it is sometimes referred to as an inertial coordinate system.

The angular difference between the two coordinate helps define the behavior of the quad altitude in space. The attitude system can be derived by rotating the body frame around the z-axis

of the earth frame by the yaw angle  $\theta$ , which is then followed by turning around the y-axis by the pitch angle  $\phi$  and finally by rotating around the x-axis by the roll angle  $\varphi$ . This is shown in the rotation matrix that has the body and earth frame parallel to each other, and their sequence of rotation is known as the Z-Y-X rotation and its rotation matrix.

### 3.3.2 Euler Angles

Euler angles comprise of 3 Angles of 3 rotations. I.e., x, y, and z:  $(\varphi), (\phi), (\theta)$

Any rotation can be described by three successive rotations about linearly independent axes.

$$A_{R_D} = A_{R_B} * B_{R_C} * C_{R_D}$$

$$A_{R_D} = Rot(x, \varphi) * Rot(y, \phi) * Rot(z, \theta)$$

Where:

$Rot(x, \varphi)$  - Roll

$Rot(y, \phi)$  - Pitch

$Rot(z, \theta)$  - Yaw

Determinant of Euler Angles

$$R = Rot(x, \varphi) * Rot(y, \phi) * Rot(z, \theta)$$

$$\text{Known Rotation Matrix} \Rightarrow \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

Therefore, Rotation matrix (R) is represented as:

$$R = \begin{bmatrix} \cos \theta \sin \varphi & \sin \phi \sin \theta \cos \varphi & \cos \phi \sin \theta \cos \varphi + \sin \phi \sin \varphi \\ \cos \theta \sin \varphi & \sin \phi \sin \theta \sin \varphi + \cos \theta \cos \varphi & \cos \phi \sin \theta \sin \varphi - \sin \theta \cos \varphi \\ -\sin \theta & \sin \phi \cos \theta & \cos \phi \cos \theta \end{bmatrix}$$

Euler's Theorem of Rotation states that any displacement of a rigid body such that a point on the rigid body, say O, remains fixed, is equivalent to a rotation about a fixed axis through the point O.

### 3.3.3 States of the Quadcopter

From the section of the coordinate system, the **angle** of the roll, pitch, and yaw are represented as  $\phi$ ,  $\theta$ ,  $\varphi$  in addition to their angle, **angular velocity** is also required and can be described as  $\dot{\theta}$ ,  $\dot{\phi}$ ,  $\dot{\varphi}$ . These are the first six state of the quadcopter that shows a relationship between the quadcopter and the earth coordinate system. The next six states show a physical connection of the physical location within the earth fixed network, and it is denoted as  $X$ ,  $Y$ ,  $Z$ . In addition to their physical position is their quad velocity along these axes and it is denoted as  $\dot{X}$ ,  $\dot{Y}$ ,  $\dot{Z}$ . Together they make up the 12 states of the quadcopter and as shown below.

$$x = [ \theta \ \varphi \ \phi \ \dot{\theta} \ \dot{\varphi} \ \dot{\phi} \ X \ Y \ Z \ \dot{X} \ \dot{Y} \ \dot{Z} ]$$

### 3.3.4 How Quadcopter Works

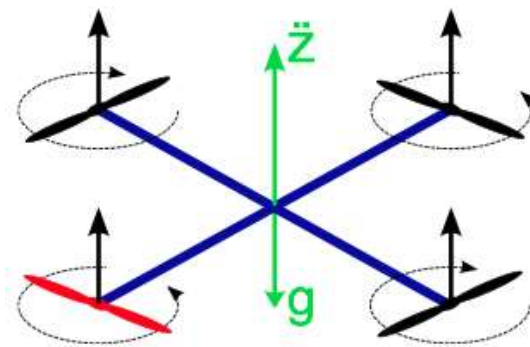


Figure 41: Thrust

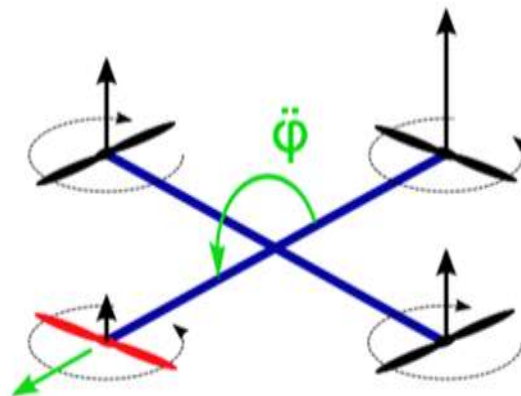


Figure 42: Pitch



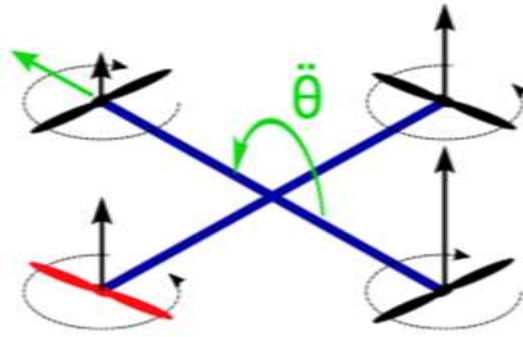


Figure 43: Roll

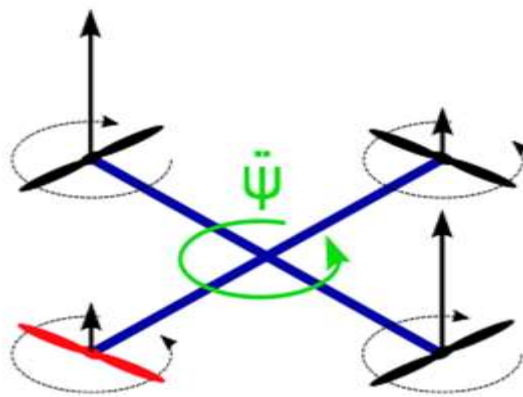


Figure 44: Yaw

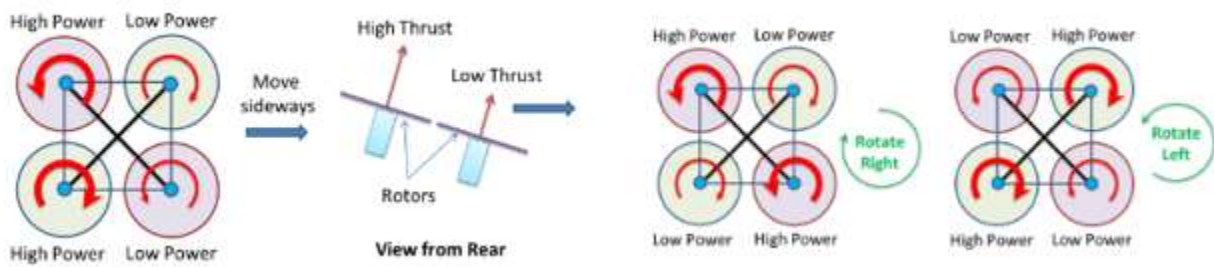


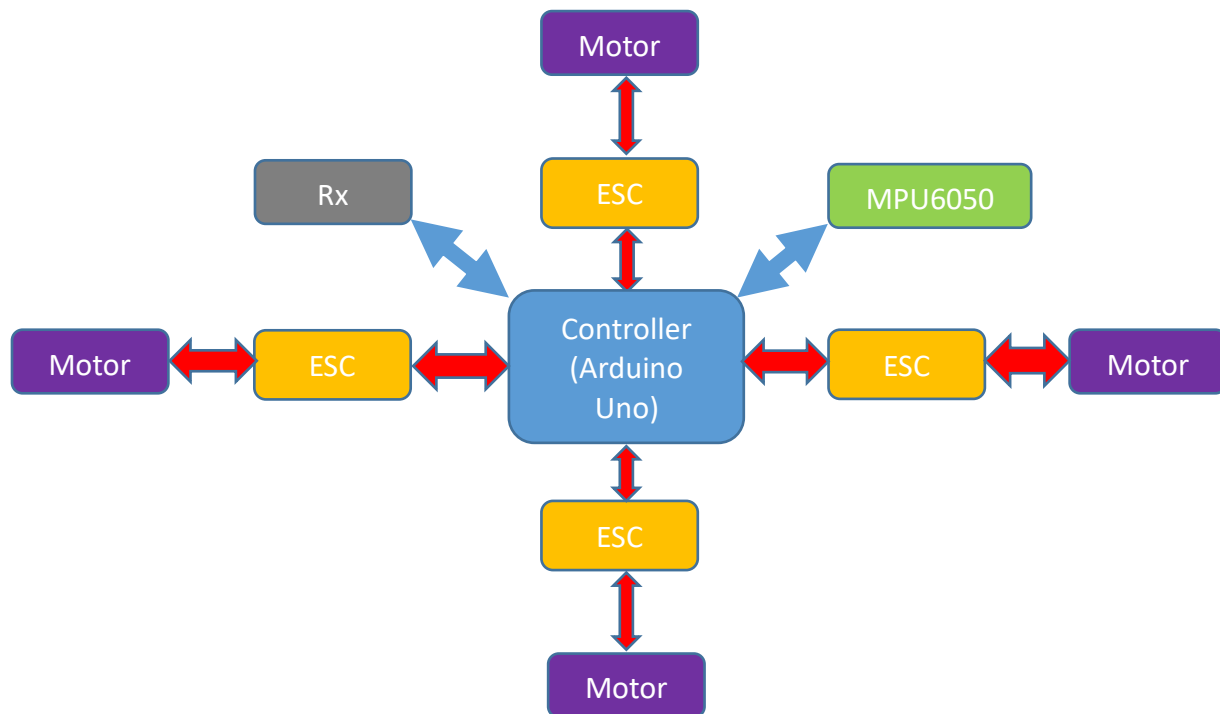
Figure 45: Yaw &amp; Roll Motion

## CHAPTER FOUR

## 4. DESIGN AND CONSTRUCTION

## 4.1 HARDWARE CONFIGURATIONS

## 4.1.1 General Quadcopter Block Diagram



*Figure 46: General Quadcopter Block Diagram*

The above diagram represents the general system block diagram of the quadcopter. A typical quadcopter has this kind of a block which is farther refined by the quadcopter specifications to provide a more specific description.

The above general diagram can be represented in several other different circuit diagrams for simulation and broad understanding of the working of the system.

## 4.1.2 Breadboard Wiring Diagram

### 4.1.2.1 Quadcopter wiring

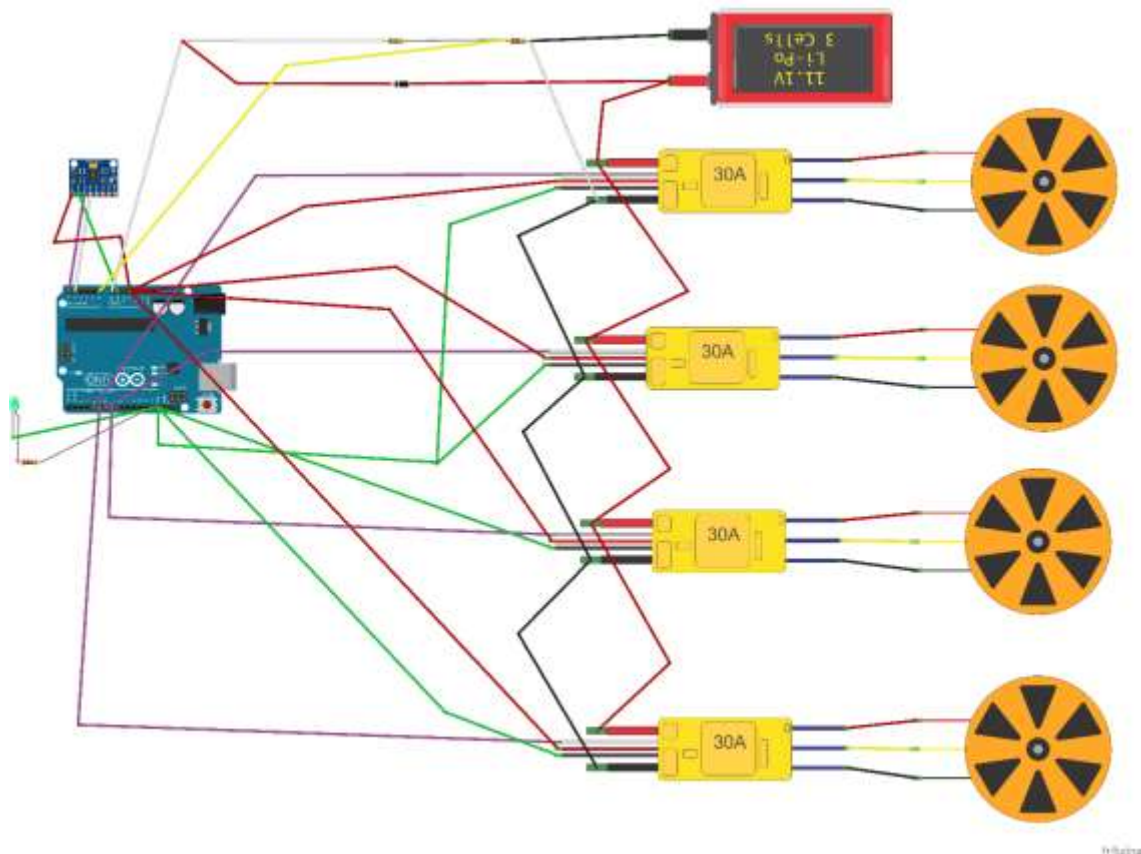


Figure 47: Fritzing Quadcopter Wiring

### 4.1.2.2 Gas Detection wiring

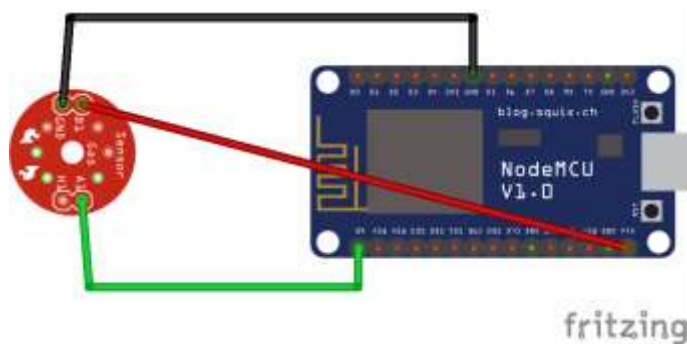
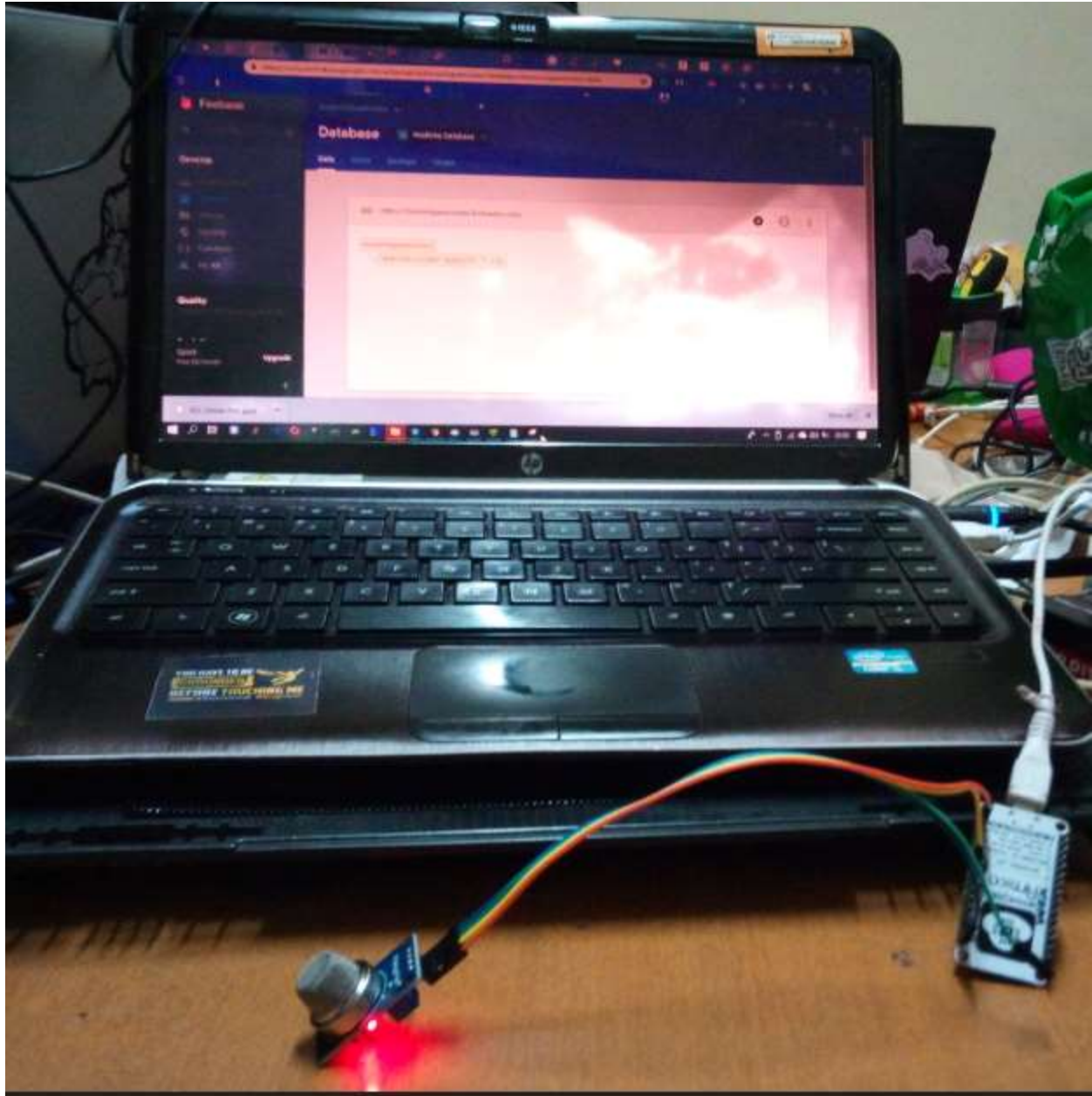


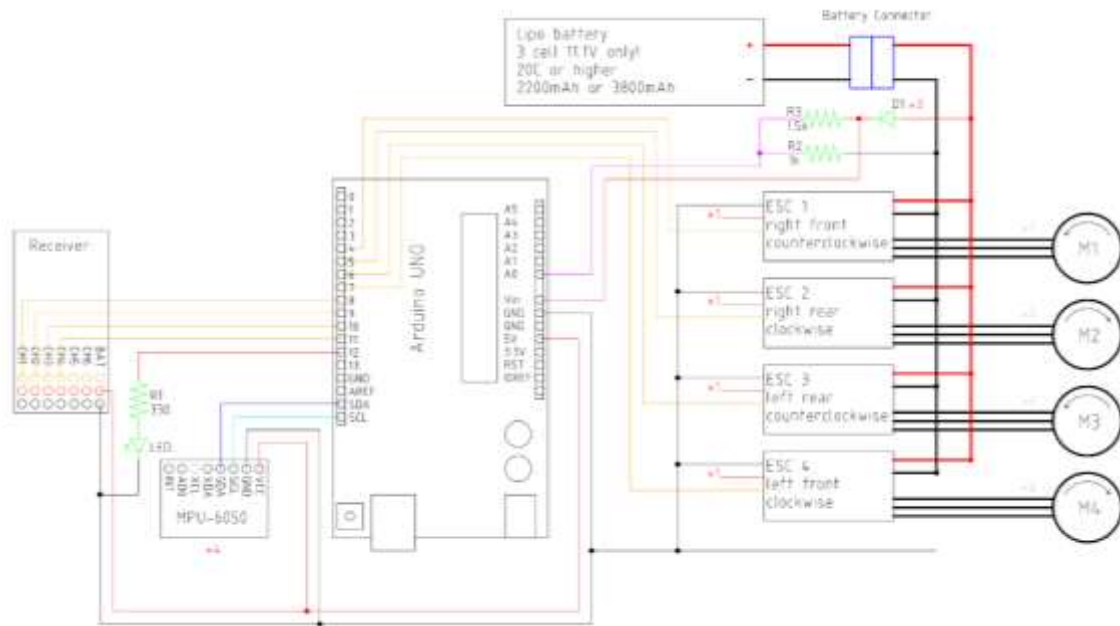
Figure 48: Fritzing Gas Detection Wiring



*Figure 49: Gas Detection and Monitoring*

MQ-2 gas sensor is connected to the analog pin of the ESP8266. This ensures the input is in continuous form enhancing real-time monitoring of the constant change of the air quality or the gas leakage.

### 4.1.3 Quadcopter Schematic Diagram



*Figure 50: Quadcopter Schematic Diagram*

The necessary wiring of the different components with Arduino Uno microcontroller include:

#### 4.1.3.1 Motors and ESCs

Motors are wired directly to the ESCs. Their connection depends on the direction in which they rotate. To change the course of the rotation, two random terminals are interchanged. This is done to ensure two motor run on the clockwise direction and the other two on the anti-clockwise path. This configuration also provides thrust, yaw, roll, and pitch can be commanded independently. The Arduino pin configuration is vital and needs to be considered well to avoid miscommunication when it comes to issuing commands from the transmitter based on the software configurations. The connection of the ECSs to the Arduino is as follow:

ESC/Motor A – Pin 4

ESC/Motor B – Pin 6

ESC/Motor C – Pin 7

## ESC/Motor D – Pin 5

*4.1.3.2 Receiver*

The receiver is configured to allow stable and reliable communication between the transmitter and the microcontroller to the ESC which finally provides the necessary command to the motors. The wiring of the receiver to the Arduino is considered as follows:

Channel 1 – Pin 8

Channel 2 – Pin 9

Channel 3 – Pin 10

Channel 4 – Pin 11

These channels are also configured in a way to command specific motions based on the present MODE of the transmitter. For this project, Transmitter MODE 2 has been used and has the following features:

Channel 1 - Roll

Channel 2 – Pitch

Channel 3 – Throttle

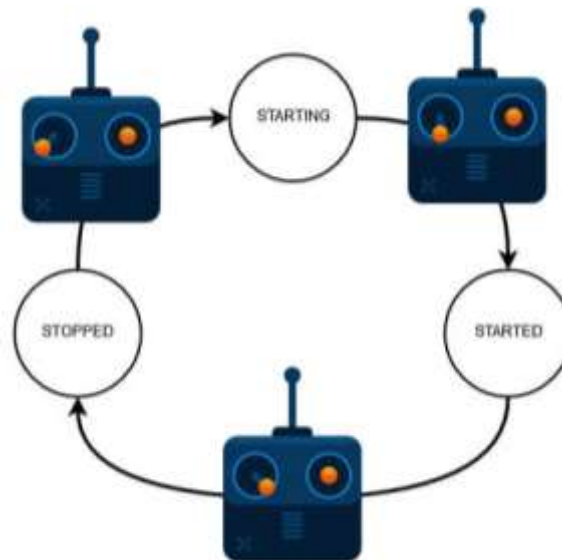
Channel 4 – Yaw



*Figure 51: Transmitter MODE 2 Configurations*

### *Transmitter*

This is configured based on the MODE of the operation and the defined controller functions. The configuration of the transmitter follows a specific sequence of operation during "START" and "STOP" based on the executed flight controller program.



*Figure 52: Transmitter: Starting, Stopped, & Started*



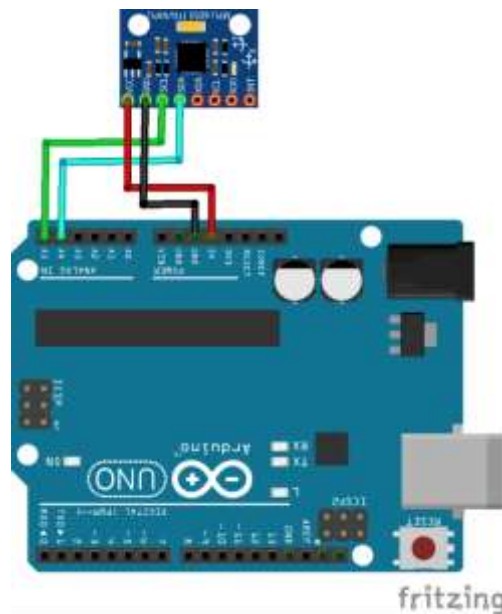
#### 4.1.3.3 MPU6050

The IMU (sensor) operates on Serial mode, and hence only two pins are wired to the Arduino for communication and data processing purposes. The power pins are wired to the appropriate pins based on the nature of the circuit. The MPU6050 can operate with voltage ranging from 3.3V – 5V DC. So long as the energy is stable, the sensor is likely to give a stable reading. Wiring of the sensor to the Arduino is as follows:

SDA – A4

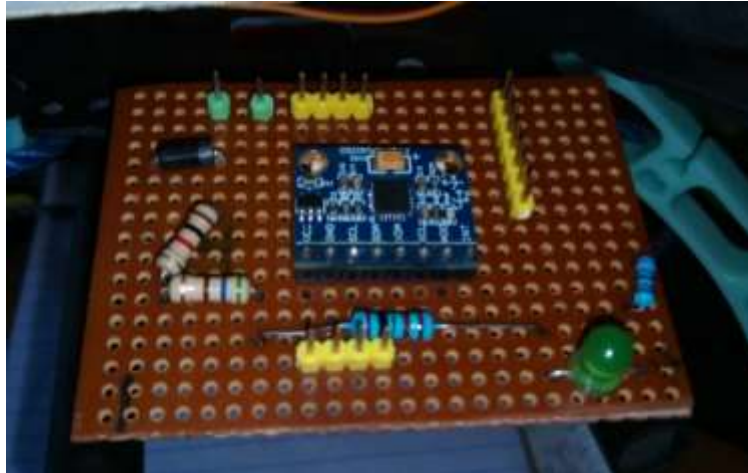
SCL – A5

This configuration varies from different Arduino microcontrollers since these unique ports are assigned to different specific Arduino pins.



*Figure 53: MPU6050 Arduino Connection*





*Figure 54: Mounting of MPU6050 on a Strip Board*

#### 4.1.3.4 Other Components

*Status Indicator (LED):* This is connected to the Arduino Pin 12.

*Battery Level Monitor:* This is wired to the Analog Pin A0 of Arduino. The analog pin ensures that the voltage is monitored continuously and the microcontroller is capable of communicating with the transmitter for real-time updates of the battery level.

*Input Voltage:* Arduino Pin Vin enable the provision of external power to the Arduino. Since Arduino operates within a range of less than 12V DC and 1 Ampere, these parameters should be maintained at any time.

## 4.2 SOFTWARE CONFIGURATIONS

This is the section that binds the hardware to the specific function based on the program that is executed by the microcontroller. The program is coded as the defined algorithm that gives the quadcopter, and the entire system operates most appropriately.

This project requires programming of two different controllers:

- i. Quadcopter PID Controller
- ii. Air quality monitor

### 4.2.1 Quadcopter PID Controller

#### 4.2.1.1 Overview

This is based on the Motor Mixing Algorithm (MMA) which is defined based on the below combinations:

	THRUST	YAW	PITCH	ROLL
$M_{FR}$	=T	+Y	+P	+R
$M_{FL}$	=T	-Y	+P	-R
$M_{BR}$	=T	-Y	-P	+R
$M_{BL}$	=T	+Y	-P	-R

*Table 4: Motor Mixing Algorithm (MMA)*

Where:

M – Motor	T - Thrust
R – Right	Y - Yaw
L – Left	P - Pitch
F – Front	R - Roll
B – Back	

MMA is implemented in a PID Controller that allows commanding of thrust, yaw, pitch, and roll independently. This is a robust algorithm since all quadcopters are controlled using it.

Implementation of MMA in a PID is shown below:

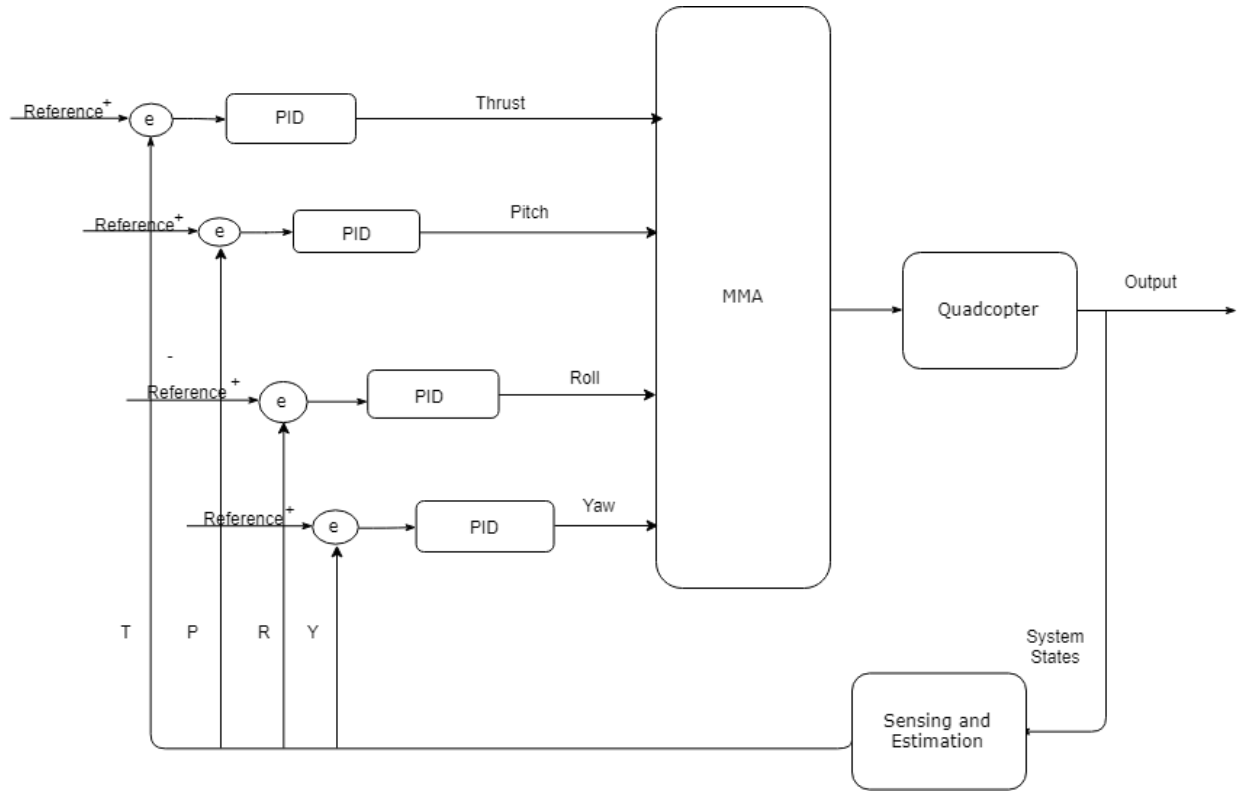


Figure 55: Implementation of MMA

To dampen the vibrations and other disturbances for the sensors and the parameters provided a complimentary filter is implemented in the code form. For an MPU6050, Gyro facilitates integral factor which is a representation of the High Pass Filter (HPF) while accelerometer facilitates angle conversion factor which represents Low Pass Filter (LPF). Summing these two filters results into a complementary filter which is a whole signal.

#### 4.2.1.2 Implementation of Quadcopter PID Controller

$$u(t) = \ddot{x}^{des}(t) + K_d \dot{e}(t) + K_p e(t) + K_i \int_0^t e(T) dT$$

##### 4.2.1.2.1 Proportional

$$P_{output} = (Gyro - Receiver) * P_{gain}$$

#### 4.2.1.2.2 Integral

$$I_{output} = I_{output} + ((Gyro - Receiver) * I_{gain})$$

#### 4.2.1.2.3 Derivative

$$I_{output} = (Gyro - Receiver - Gyro_{prev} - Receiver_{prev}) * D_{gain})$$

#### 4.2.1.2.4 PID in Code

An implementation of the PID in roll motion:

$$pid_{error_{temp}} = gyro_{roll_{input}} - pid_{roll_{setpoint}};$$

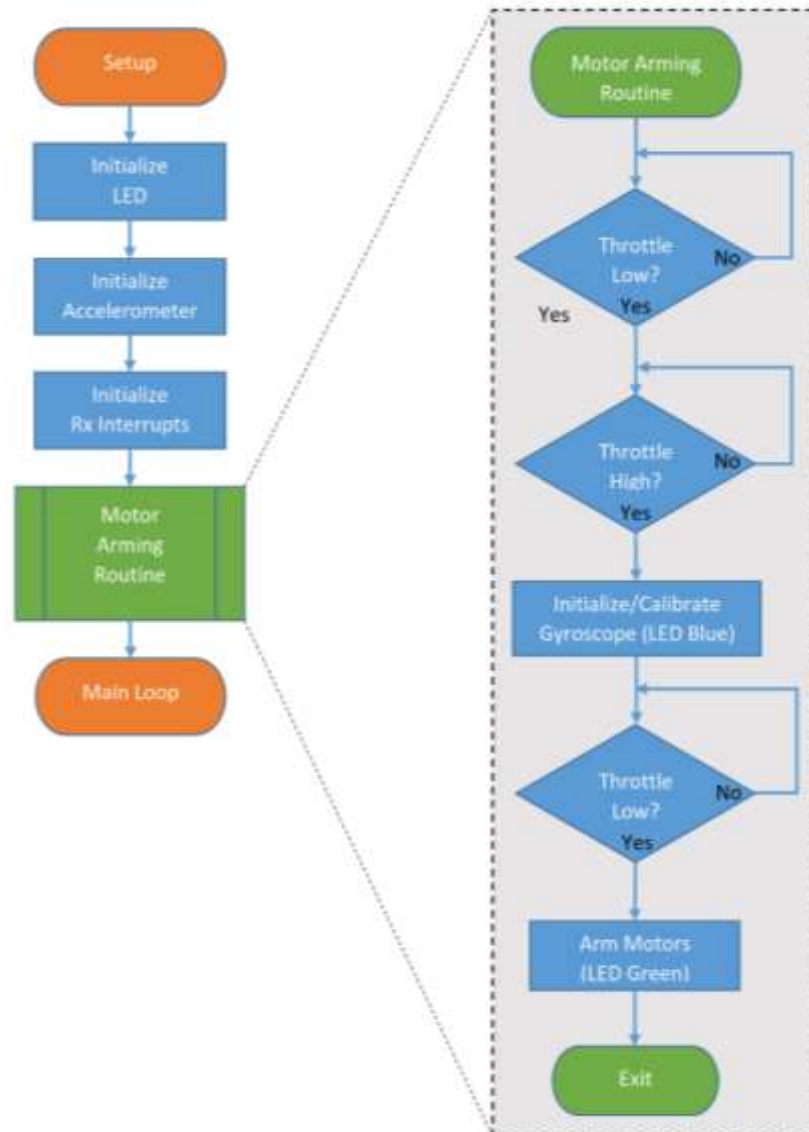
$$pid_{i_{new_{roll}}} = pid_{i_{gain_{roll}}} * pid_{error_{temp}};$$

$$pid_{output_{roll}} = pid_{p_{gain_{roll}}} * pid_{error_{temp}} + pid_{i_{new_{roll}}} + pid_{d_{gain_{roll}}} * (pid_{error_{temp}} - pid_{last_{d_{error}}});$$

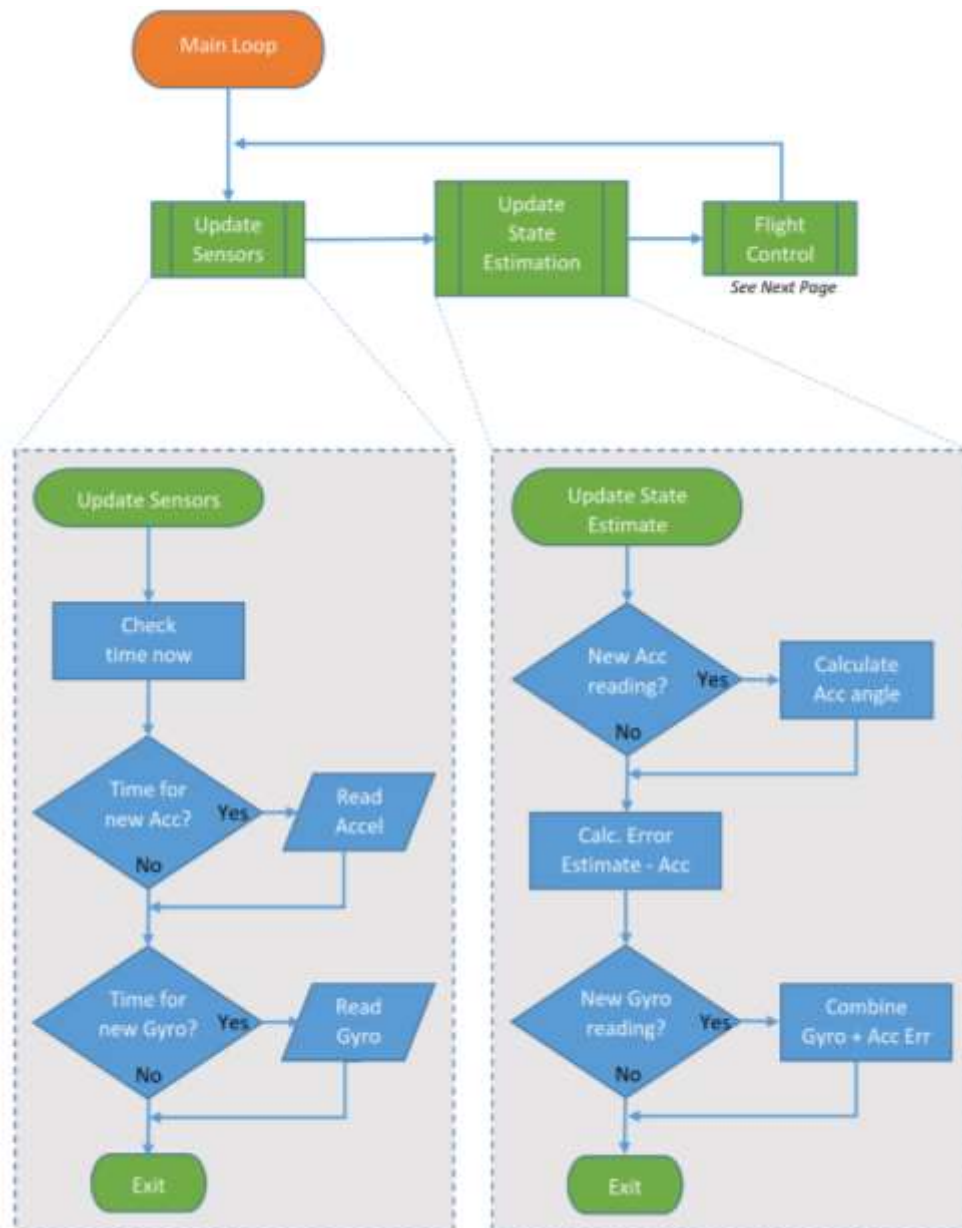
$$pid_{last_{d_{error}}} = pid_{error_{temp}}$$

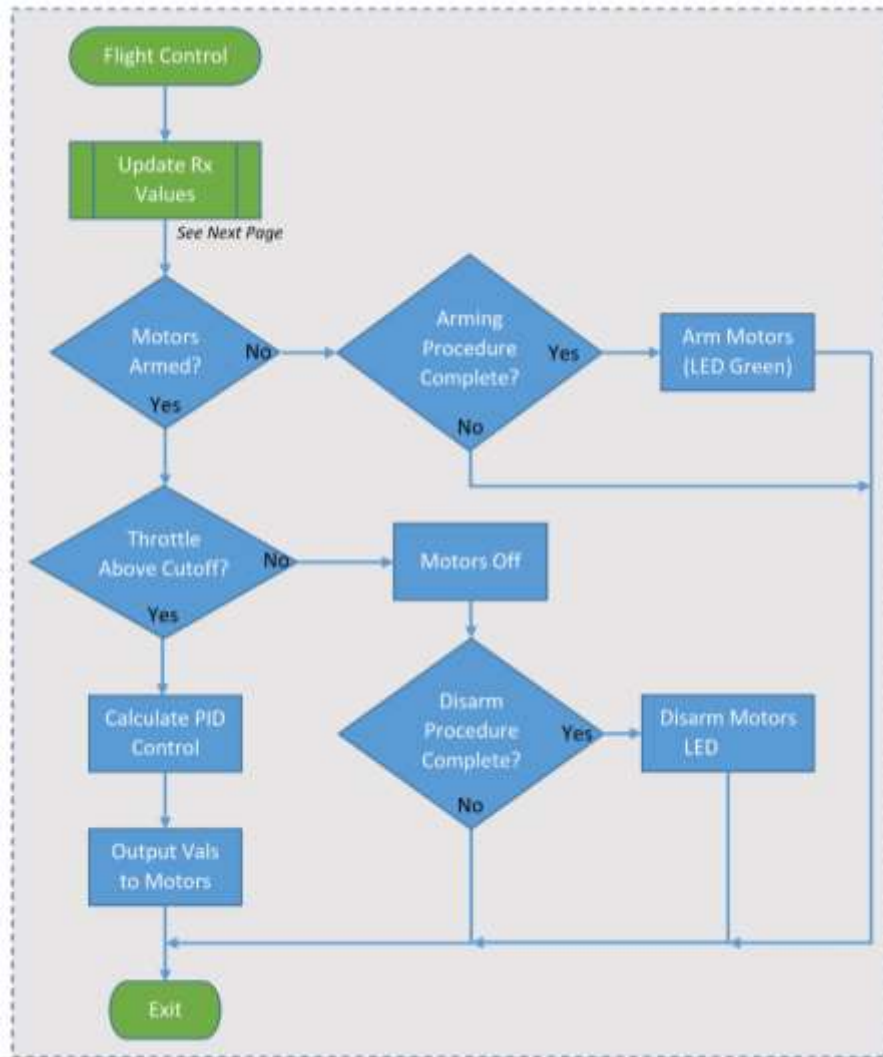
### 4.2.1.3 Flight Controller Algorithm

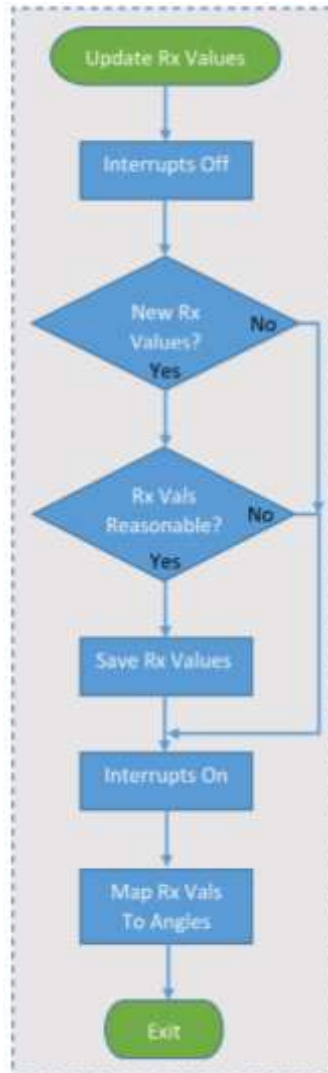
#### 4.2.1.3.1 Initial Set-Up



#### 4.2.1.3.2 Main Loop

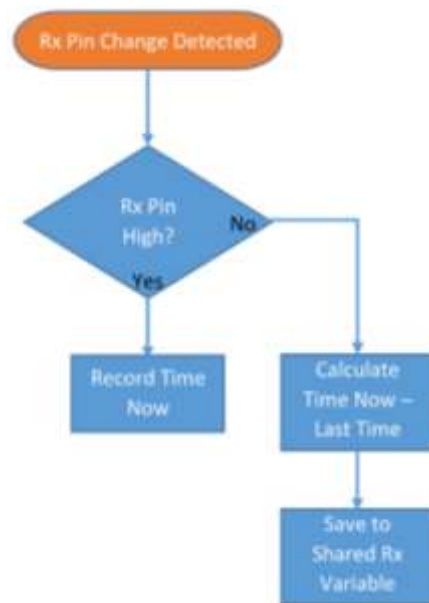








#### 4.2.1.3.3 Interrupt Function



#### 4.2.1.4 Flight Code Model

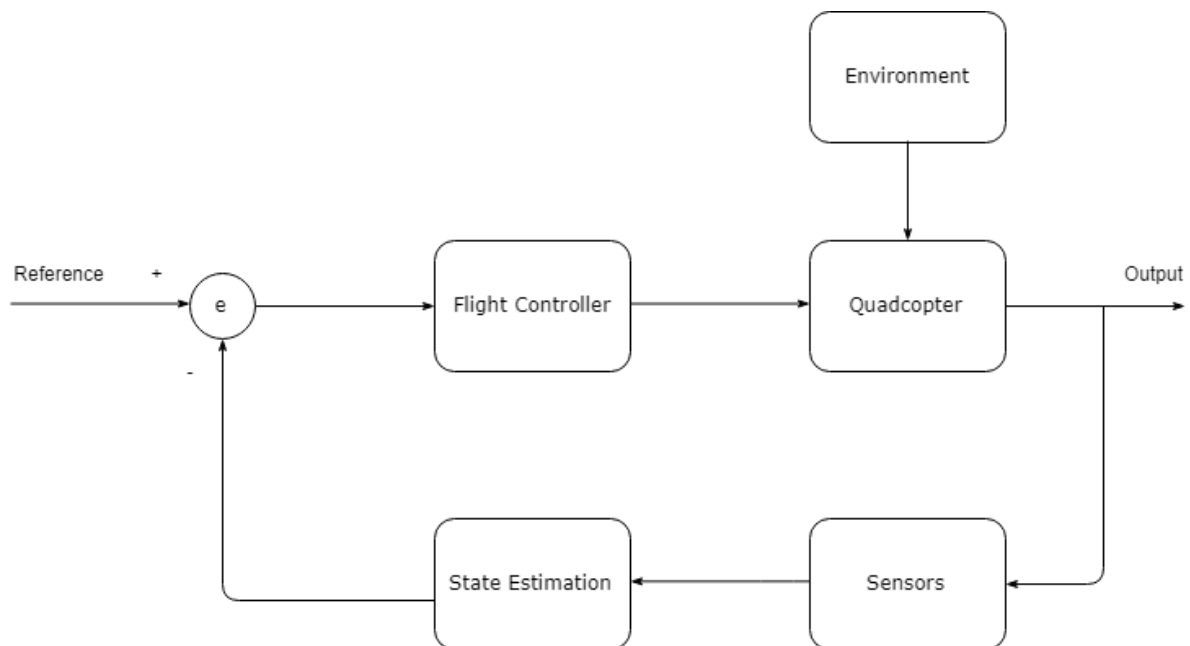


Figure 56: Flight Code Model

## 4.2.2 Air Quality Monitor

### 4.2.2.1 Overview

This system is the application among the many that can be implemented by the use of the drone. This system uses ESP8266 as the microcontroller and MQ-2 as the sensor. These components are enhanced by the use of Firebase: Google Cloud Platform which provides cloud-based real-time database access. For better monitoring of the change of parameters, an Android application has been built to facilitate that. The use and the entire system requires an internet connection to work.

### 4.2.2.2 Communication and Data processing

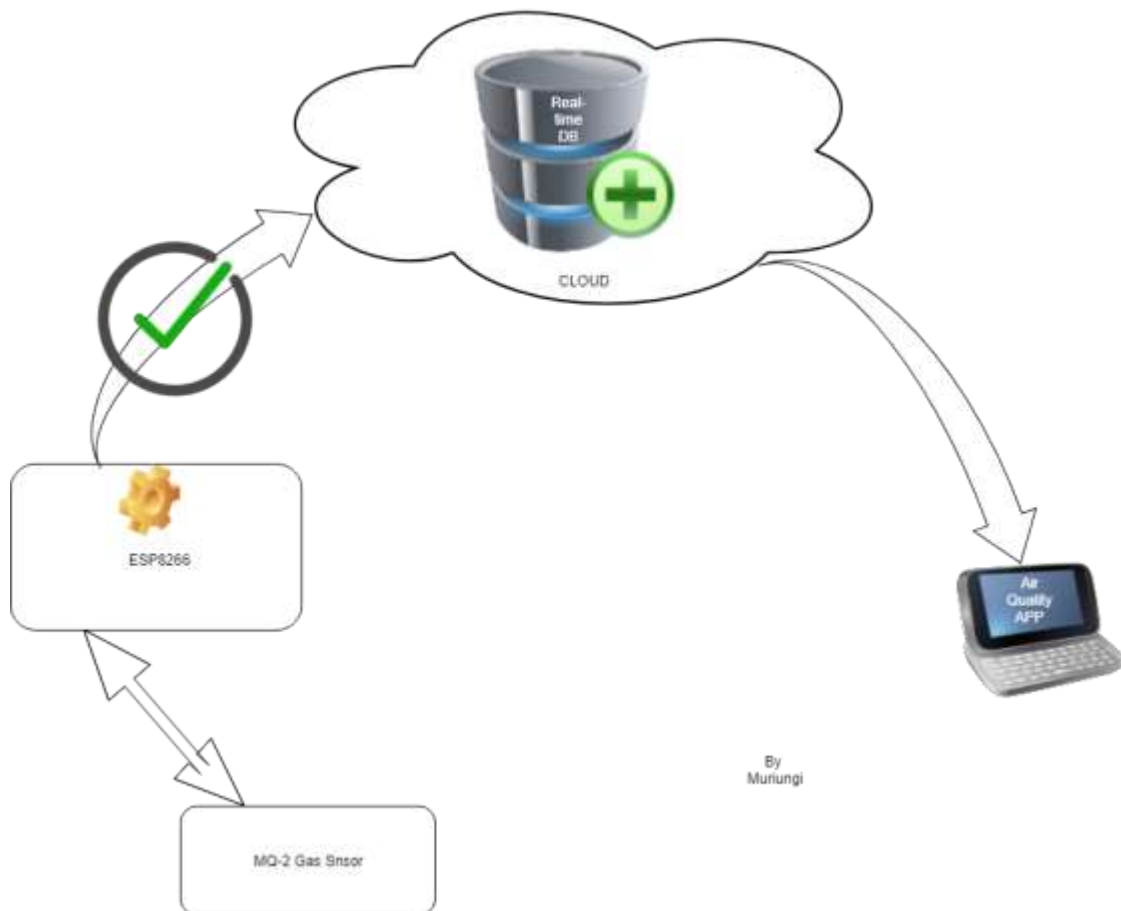


Figure 57: Communication and Data Processing

#### 4.2.2.3 Node MCU Code Structure

ESP8266 module requires special libraries to have the initial set up done correctly. Apart from these libraries, cloud configuration is done to ensure the hardware set up communicates well with the cloud resources. The module also should be provided with the Wi-Fi in which it gets connected to. The Wi-Fi connection should be strong enough to allow a stable connection to avoid delayed data being relayed. The initial set up is explained below.

```
12 #include <ESP8266WiFi.h> //  
13 #include <FirebaseArduino.h>  
14  
15 // Set these to run example.  
16 #define FIREBASE_HOST "droneiotgasmonitor.firebaseio.com"  
17 #define FIREBASE_AUTH "J45Fu0kzUjCUwaKZ8Dg83GFIKF6NBpD4es7No123"  
18  
19 #define WIFI_SSID "Drones" //Your Wi-Fi SSID  
20 #define WIFI_PASSWORD "Muriungi254" //Your Wi-Fi Password  
21  
22 const int mq2GasSensor = A0; // Gas Sensor
```

#### 4.2.2.4 Firebase: Google Cloud Access

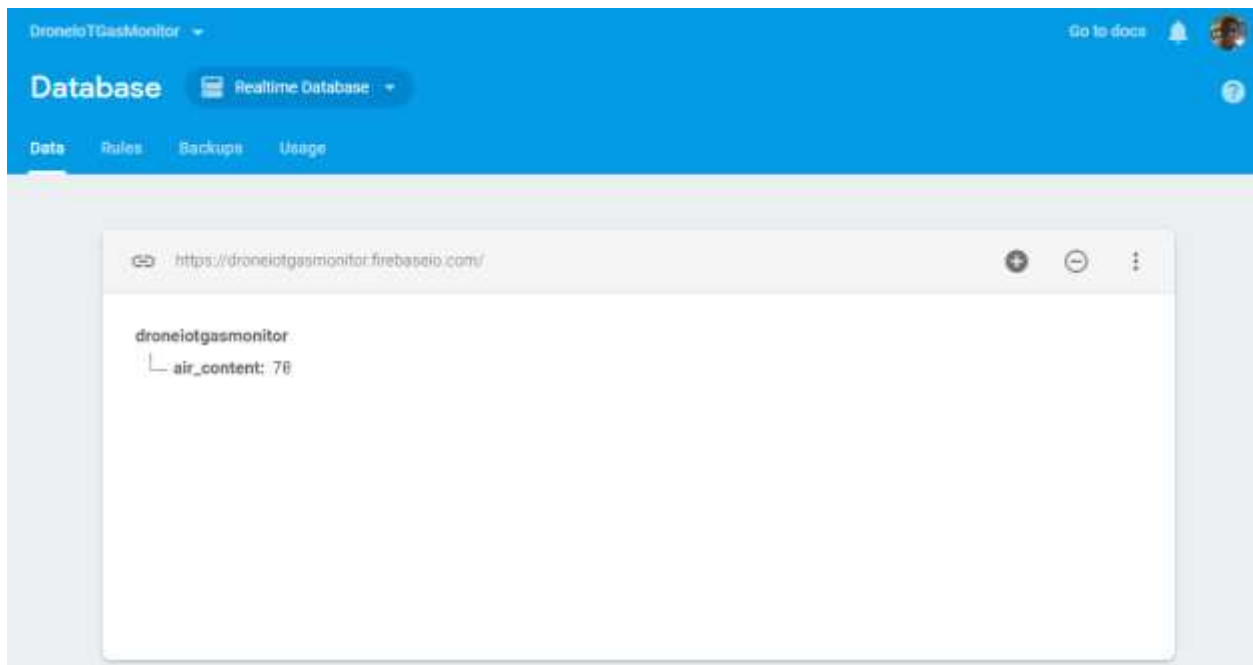


Figure 58: Firebase Cloud Access

#### 4.2.2.5 Android Mobile Application



Figure 59: Android Mobile App Interface

## CHAPTER FIVE

### 5. RESULTS AND DISCUSSIONS

This section describes the working of all the components that have been discussed in Chapter five. Among the key elements that will be addressed are:

- i. Calibration of the sensors and actuators
- ii. PID Tuning
- iii. Testing
- iv. Performance analysis and evaluation

#### 5.1 CALIBRATION OF THE SENSORS AND ACTUATORS

Any devices that give varying parameters requires calibration before any use. Calibration ensures that the acquired values are within the range and it also allows someone to determine the accuracy and the precision of the device. A system is likely to fail or become so tricky during testing since it may have very low or high preset values which are far much away from the optimal operating costs. The following components need an initial setup or calibration before performing any operation to guarantee safe testing and services:

- i. MPU6050 (Gyro)
- ii. Transmitter
- iii. ESC
- iv. Motors and Propellers

### 5.1.1 Gyro Calibration

MPU6050 is the primary sensing device in a quadcopter that gives the quadcopter ability to sense its rotational angles and acceleration which sums up to the determination of position and altitude among other elements. Calibrating this sensor facilitates proper mounting and also the most appropriate values during the implementation of the complementary filters for damping the disturbance effects.

During the calibration, the program loops at the rate of 250Hz giving it an available time

of  $4000\mu s$ . The refresh rate of the sensor is pre-set at the initial stage of the program. This value can be changed based on the performance of the sensor. Reading the data of the angular gyro rate takes  $300\mu s$ . It will always be good practice and recommended to have the MPU6050 sensor calibrated while the quadcopter is at a flat surface. A flat surface minimizes the errors and also over compensation and future carried and prolonged failures.

A well configured Gyro will allow the drone to start without any error or problem. In case the problem is encountered, the quadcopter cannot begin to, and the LED will remain on one state. Based on the quadcopter initial checks of the program, this ensures that the drone is fine and fit for flight.

```

/*
*****
*-----MPU6050 READ FUNCTION-----
*****
*/
//Requesting raw values from MPU6050.
void readSensor() {
    Wire.beginTransmission(MPU_ADDRESS); // Start
    Wire.write(0x3B); // Send
    Wire.endTransmission(); // End t
    Wire.requestFrom(MPU_ADDRESS, 14); // Reque

    // Wait until all the bytes are received
    while(Wire.available() < 14);

    acc_raw[X] = Wire.read() << 8 | Wire.read();
    acc_raw[Y] = Wire.read() << 8 | Wire.read();
    acc_raw[Z] = Wire.read() << 8 | Wire.read();
    temperature = Wire.read() << 8 | Wire.read();
    gyro_raw[X] = Wire.read() << 8 | Wire.read();
    gyro_raw[Y] = Wire.read() << 8 | Wire.read();
    gyro_raw[Z] = Wire.read() << 8 | Wire.read();
}

```



*Figure 60: Gyro Configuration*

Below are the results of the gyro calibration of a quadcopter:

[illegible]

### a) Roll & Pitch Error

(b) Roll Error

(c) Pitch Error

(d) Fully Balance

### 5.1.2 ESCs Calibration

ESC calibration depends upon the availability of adequate power supplied to the quadcopter ESCs among other factors like the state of the gyro and also if the transmitter and receiver are bound. In case there is any problem among the three mentioned, the quadcopter ESCs cannot be

armed if they were initially disarmed or vice versa. A successful calibration is determined the two beeping sound during the initial powering of the quadcopter followed by other two beeping sound after there is a perfect connection between the receiver and the transmitter. The maximum pulse length of the ESC is  $2000\mu s$ .

ESC Calibration program loop:

- i. Read the gyro angular rate data.
- ii. Calculate the PID corrections.
- iii. Calculate the pulse for every second.
- iv. Compensate every pulse for voltage drop.
- v. Send the calculated pulse to the ESCs.

### 5.1.3 Transmitter Initial Set-up

Transmitter requires to have initially configured based on the program to be executed by the flight controller. In this case, the transmitter is configured on MODE 2 which has been explained in the previous chapter. Failure to have a proper configuration, the transmitter and receiver may bind, but the commands may fail to execute or end up being executed wrongly. This is a crucial initial set-up step that is crucial in having a safe flight operation. Based on the code to be completed, the transmitter may be allowed to operate in more than one mode. This project has the provision of only MODE 2. Addition of other methods requires unique connections to the receiver channel 5 and expansion of the functionalities in the code to allow operation of several different ways.

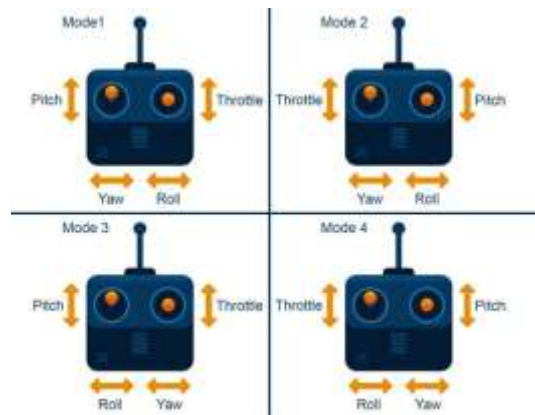


Figure 61: Transmitter MODE Configurations

### 5.1.4 Motors and Propellers Calibration

Two motors in a quadcopter run in clockwise while the other two runs in the anti-clockwise direction. The same configuration of the rotor applies to propellers. Failure to adhere to these



configurations makes it difficult for the quadcopter to achieve thrust, roll, yaw, or pitch independently. This could also result in losing propeller caps due to propellers becoming loose. When the propellers are loosely attached, it is difficult even to attain the efficiency needed.

## 5.2 PID TUNING

Every custom-made quadcopter has different PID controller parameters due to variations on various components as discussed in the previous chapters. Although the PID controller parameters could be changed, there exist several PID tuning methods which include:

- i. Manual Tuning
- ii. Ziegler-Nichols
- iii. Tyreus Luyben
- iv. Cohencoon
- v. Astrom-Hagglund

Among the five techniques, the most popular and widely used include the: Ziegler-Nichols and manual tuning. The benefit of using Ziegler-Nichols is that it is a proven method while manual adjustment requires no mathematics. The limitations of manual tuning are that experienced personnel is needed.

Despite Ziegler-Nichols being popular among scientists and engineers, it has the following disadvantages:

- i. Some trial and error process is involved
- ii. Very aggressive tuning
- iii. The process may tend to be upset

When having PID tuning, it should be noted that:

Proportional (P): It is for Immediate Correction

Integral (I): It is for overtime or steady-state correction

Derivative (D): It is for “Take it easy” correction

### 5.2.1 Manual PID Tuning Procedure

- i. Start by setting all values to 0 except  $P_{\text{roll}} = 1$  and  $P_{\text{yaw}} = 1$ . Examine the behavior.
- ii. Set  $P_{\text{roll}}$  to 0 and start incrementing  $P_{\text{yaw}}$  with 0.5 until a stable point is attained.
- iii. Maintain the value of  $P_{\text{yaw}}$  and start setting  $I_{\text{yaw}}$ . Since the Integral has higher effect use steps of 0.001.
- iv. After  $I_{\text{yaw}}$  it attained, leave  $D_{\text{yaw}}$  at 0 and proceed to tune the roll values.  $D_{\text{yaw}}$  does not need tuning since there is much drag from the props; hence the effect is automatically canceled.
- v. Do the same for the  $P_{\text{roll}}$  and  $I_{\text{roll}}$ .
- vi. For  $D_{\text{roll}}$ , take steps of 1.0 for the few steps, and if the effects are minimal, you can decide to take steps of 5.0 until a stable point is attained.
- vii. Since (Pitch values) = (Roll Values), the pitch values are determined by only using the roll values.



*Figure 62: PID Response Testing*



*Figure 63: PID Controller Manual Tuning*

## 5.3 SYSTEM TESTING

### 5.3.1 Quadcopter PID Controller

After several calibrations and testing of the PID controller, it was finally concluded that the quadcopter operated in a stable mode at:

```
void pidController() {  
    //These are the Gain Values that are tuned accordingly  
    //Every slight change in the quadcopter will require tuning.  
    float Kp[3]      = {3, 1.3, 1.3};           // P coefficients : Yaw, Pitch, Roll  
    float Ki[3]      = {0.045, 0.03, 0.03};     // I coefficients : Yaw, Pitch, Roll  
    float Kd[3]      = {0, 15, 15};             // D coefficients : Yaw, Pitch, Roll
```

This is the most rigorous process in designing and implementing a PID Controller in a quadcopter since it requires precision in everything that one does. Every slight change in the quadcopter hardware configuration was resulting in a massive problem on the PID Controller. This stage, therefore, requires a lot of precision handling of the quadcopter to get the most accurate results. These values of the PID Controller were able to ensure stable command of pitch, yaw, roll, and thrust are achieved.



*Figure 64: PID Controller Testing*



*Figure 65: Quadcopter Flying*



*Figure 66: Final Quadcopter Prototype*

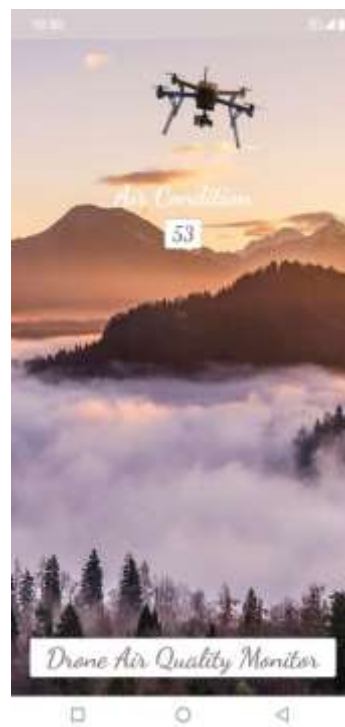
### 5.3.2 Air Monitor



The readings are not available hence indicated as N/A.

This could be due to:

1. Loss of Internet connection.
2. The cloud resources are not accessible.
3. The sensor and the ESP8266 are offline.

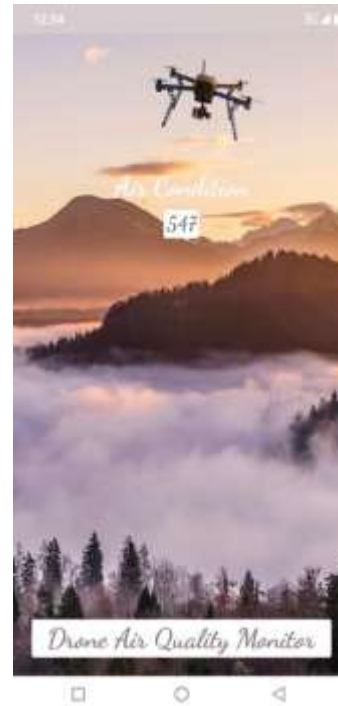


The readings are available. The low value indicates low concentration of the pollutants.





There is a slight increase of the value due to change of environment. From outdoors to indoors in a room where cooking is happening.



There is a drastic increase of the value due to availability of a high pollutant gas. This happened after LPG was tested by the use of lighter through sparking it near the sensor.

## 5.4 PERFORMANCE ANALYSIS AND EVALUATION

### 5.4.1 PID Gains Analysis and Evaluation

After several tests, it was observed that:

*i. Too Low P-Gains are:*

It is Easy to overcorrect but difficult to control. This makes the stick response to be prolonged, imprecise and easy to turn by the use of the hand.

*ii. Correct P-Gains are:*

Easy to control and correction is always stable. It results in a sharp response of sticks and cannot be easily turned by hand. Although the response is good, there are ordinarily slight oscillations after sticks have been released.

*iii. Too high P-Gain are:*

Its altitude is hard to control.

Gains heights quickly which makes it hard to hover.

Motors produces an oscillating sound.

Has rapid oscillations.

iv. *Too High I-Gains are:*

The effects are the same as those of Too high P-Gain, but the difference is that they oscillate with shallow frequency.

v. *Good P-Gain and Too Low I-Gain:*

There is much smooth coordination due to factors like gravity since the sticks move back to center after the release of the stick. It is also experienced that there is ease while pushing it by hand over a long duration of time.

vi. *Good P-Gain and Good I-Gain:*

The quadcopter responds according to, and there are no unnecessary movements due to stick release. It is also tough to be affected by external disturbances for a long time.

vii. *Good P-Gain and Too High I-Gain:*

There are many oscillations but occurring at a lower frequency than when with too high P-Gain.

viii. *Effects of D-Gain*

D-Gain ensures settling and achievement of the foreseen point is attained quickly. It should be noted that too high D-Gain may have catastrophic damage to motors due to overheating.

Weight of the quadcopter must be centered achieve a stable flight. It has also been observed that it becomes tough to control an imbalanced quadcopter. Effects of unbalanced quadcopter result into the fast-rising of the lighter side hence resulting in leaning sideways and finally toppling when on the throttle.

Values received from the gyro plays a significant role in a PID controller and auto-correction effects. The more the precise the acquired values, the higher the accuracy of the gyroscope. It should also be noted that Low P-Gain works so well when tested under calm conditions but has a much worse response when tested on a windy or high disturbance environment. Similarly, High P-Gain keeps the quadcopter calm in a very high windy environment. It is therefore advisable to strike a point that works optimum in both environmental conditions.

**NOTE:** The orientation is never an essential factor to consider so long as the z-axis is vertical (perpendicular to the surface) and the edges of the gyro are aligned with the sides of the quadcopter. The software or the program used in controlling the gyro should be able to detect and invert gyro orientation when needed. Gyros are best when mounted by the double-sided tape unlike when using any dampening material such as foam tape since it increases the vibrations decreasing the performance of the gyroscope.

#### **5.4.2 Common Mistakes when Operating a Quadcopter**

Based on the results that were achieved from this project, several mistakes were identified which resulted in not getting the most appropriate and desired flight. These mistakes are common to all similar projects and have many impacts when testing or during actual quadcopter operations.

These mistakes include:

- i. Powering ON a quadcopter when it is not on a level surface.
- ii. Lose of orientation: Always be at the back of the quadcopter with the transmitter pointed at the perfect center of the quadcopter.
- iii. Crashing while taking-off: Giving a momentarily increasing thrust has this effect hence recommended to start the quadcopter with enough throttle.
- iv. Deficient power battery: Battery should always operate with more than 20% full charge.
- v. Landing too fast: It is recommended to land gently so long as the PID controller responds correctly.



## CHAPTER SIX

## 6. PROJECT SCHEDULE AND BUDGET

## 6.1 PROJECT SCHEDULE

## MULTI-PURPOSE QUADCOPTER | EC/09/14

Project Schedule

	Sept - Oct 2018	Nov - Dec 2018	Jan - Feb 2019	March 2019	April 2019	May 2019
<b>Project Selection</b>						
<b>Literature Review</b>						
<b>Design &amp; Construction</b>						
<b>Testing</b>						
<b>Presentation &amp; Report</b>						

## 6.2 BUDGET

Component	Description	Quantity	Total Cost
Frame	Carbon 450mm	1	4500
Motors (Brushless)	2200/1000KV	4@1500	6000
Propellers	10 in	4@300	1200
ESCs	30A	4@1800	7200
Controller	Arduino Uno	1	1200
Rx/Tx	2.4GHz	1	8000
Battery (Li-Po)	2200mAh	2	6500
PDB	2 Channel	1	450
Landing Gear	Carbon Fibre	1 pair	450
MPU6050	Sensor	1	1000
Miscellaneous	License, transportation, internet access etc		13,500
<b>TOTAL</b>			<b>50,000</b>

## CHAPTER SEVEN

### 7. CHALLENGES, CONCLUSION, AND RECOMMENDATIONS

#### 7.1 CHALLENGES

- i. Delay of the project.
- ii. Faulty IMU (sensor) makes it hard in controls.
- iii. Power Problems (battery selection).
- iv. Obtaining a precise Balanced System.
- v. Costly project.
- vi. Significant Scope for one person based on the short time available.

#### 7.2 CONCLUSION

In conclusion, the project is a success having accomplished to design the Arduino based custom quadcopter PID flight controller and IoT based air detection and monitoring system. Designing the quadcopter was simple but building it requires a high level of precision in balancing every component. The PID Controller responded well despite challenges of unevenly balanced masses of the quadcopter. Thrust, pitch, roll, and yaw has been able to be commanded independently and also the quadcopter is ready to auto-level when on thrust motion. Many lessons have been learned through the projects as seen through discussions in different chapters of the report. The air quality detection and monitoring system have responded well allowing successful data processing from end-to-end. The ESP8266 was able to take the reading from the sensor and upload them to the cloud where the real-time database was hosted. The mobile application received the values on a real-time basis making the system a complete success.

### 7.3 RECOMMENDATIONS

- i. Proper planning is needed for such a project to give the testing of the quadcopter enough time.
- ii. Adequate resources are required especially 3D printers to ensure quality designs are achieved.
- iii. Due to limitations of licensing, it is advisable to have the license before starting doing the project.
- iv. Acquire quality sensors and other components to guarantee precision and accuracy is achieved.
- v. Any student planning to do such a project should seek a reliable team to collaborate with since the project complexity requires a multi-disciplinary approach.

### 7.4 FUTURE WORKS

Drones technologies provide massive opportunities both in design and implementation based on specific needs. With the advanced sensors and other control electronics, designing custom drone is possible and customized to suit specific needs in the society. Cutting edge technologies like AI, cloud computing, data analytics, blockchain, among others has exposed drones to massive manipulation which are likely to solve societal problems in better ways. Modeling of a custom drone with recommended features and specific from customers is a worthy exploration for future researchers, companies, and individuals willing to make their drones possible. Application these trending technologies could future uses of drones a reality.

## REFERENCES

Abas, N., Legowo, A., and Akmeliawati, R. (2011), "Parameter Identification of an Autonomous Quadrotor," *4th International Conference on Mechatronics*, IEEE, Kuala Lumpur, Malaysia.

Raza, A. Syed, and Gueaieb, Wail. 2010. *Intelligent Flight Control of an Autonomous Quadrotor*. Science, Technology and Medicine. Doi: 10.5772/6968.

Schmidt, M. David. 2011. *Simulation and Control of a Quadrotor Unmanned Aerial Vehicle*. The University of Kentucky Library.

Andreas, R. (2010), "Dynamics Identification & Validation, and Position Control for a Quadrotor," MS Thesis, Autonomous Systems Lab, Swiss Federal Institute of Technology, Zurich, Switzerland.

Balas, C. (2007), "Modelling and Linear Control of a Quadrotor," MS Thesis, Cranfield University, Bedford, UK.

Beauregard, B. (2012), "Arduino (open source) PID Library," available at

Bresciani, T. (2008), "Modeling, Identification and Control of a Quadrotor Helicopter," MS Thesis, Department of Automatic Control, Lund University, Lund, Sweden.

DiCesare, A., Gustafson, K., and Lindenfeler, P. (2009), "Design Optimization of a Quadrotor Capable of Autonomous Flight," BS Report, Aerospace, and Mechanical Dept., Worcester Polytechnic Institute, Worcester, MA.

Domingues, J. M. B. (2009), "Quadrotor prototype," MS Thesis, Mechanical Engineering Department, Technical University of Lisbon, Lisbon, Portugal.

Hurd, M. B. (2013), "Control of a Quadcopter Aerial Robot Using Optic Flow Sensing," MS Thesis, Mechanical Engineering Department, University of Reno, Reno, CA.

Lee, G., Jeong, D. Y., Khoi, N. D., and Kang, T. (2011), "Attitude Control System for a Quadrotor Flying Robot," *8th International Conference on Ubiquitous Robots and Ambient Intelligence*, URAI, Incheon, Korea, pp. 74-78.

Magnussen, O., and Sjonhaug, K. E. (2011), "Modeling, Design and Experimental Study for a Quadcopter System Construction," MS Thesis, Engineering Dept., Agder Univ., Kristiansand, Norway.

Martinez, V. M. (2007), "Modeling of the Flight Dynamics of a Quadrotor Helicopter," MS Thesis, Aerospace Sciences Department, Cranfield Univ., Bedford, United Kingdom.

Miller, D. S. (2011), "Open Loop System Identification of a Micro Quadrotor Helicopter from Closed Loop Data," MS Thesis, Aerospace Engineering Dept., Maryland Univ., College Park, MD.

Parker, M., Robbiano, C., and Bottorff, G. (2011), "Quadcopter," BS, Electrical and Computer Engineering Department, Colorado State Univ., Fort Collins, CO.

Sa, I., and Corke, P. (2012), "System Identification, Estimation and Control for a Cost-Effective Open-Source Quadcopter," *Internal Conference on Robotics and Automation*, IEEE, Saint Paul, Minnesota, pp. 2202-2209.

Saakes, D. Choudhary, V., Sakamoto D., Inami, M., and Igarashi, T. (2013), "A Teleoperating Interface for Ground Vehicles using Autonomous Flying Cameras," *23rd International Conference on Artificial Reality and Telexistence*, IEEE.

Schreier, M., and Darmstadt, T. (2012), "Modeling and Adaptive Control of a Quadrotor," *International Conference on Mechatronics and Automation*, IEEE, Chengdu, China, pp. 383-390

Siebert, S., and Teizer, J. (2014), "Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system", *Automation in Construction*, IEEE, Vol. 41.

Sorensen, A. F. (2010), "Autonomous Control of a Miniature Quadrotor Following Fast Trajectories," Control Engineering Department, Aalborg University, Aalborg, Denmark.

Tischler, M. and Rempl, R. K. (2006), "CIFER: Student version," available at

B. L. Stevens and F. L. Lewis, Aircraft Control and Simulation. Hoboken, New Jersey: John Wiley & Sons, Inc., 2nd ed., 2003.

M. Rauw, FDC 1.2 - A SIMULINK Toolbox for Flight Dynamics and Control Analysis, February 1998. Available at <http://www.mathworks.com/>.

<https://www.droneomega.com/drone-motor-essentials/>

<https://metrarocketclub.org/thrust-to-weight-charts/>

<https://www.draw.io/>

Drone Laws in Kenya: <https://www.uavsystemsinternational.com/drone-laws-by-country/kenya-drone-laws/>

Drone Laws: <https://uavcoach.com/drone-laws-in-kenya/>

<https://github.com/lobodol/drone-flight-controller>

<https://www.arduino.cc/en/Reference/PortManipulation>

<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>

[http://www.brokking.net/ymfc-3d\\_v2\\_main.html](http://www.brokking.net/ymfc-3d_v2_main.html)

PID Controller:

[https://en.wikipedia.org/wiki/PID\\_controller#Ziegler%E2%80%93Nichols\\_method](https://en.wikipedia.org/wiki/PID_controller#Ziegler%E2%80%93Nichols_method)


<https://oscarliang.com/quadcopter-pid-explained-tuning/>

Android Studio: <https://developer.android.com/studio>

Firebase: <https://firebase.google.com>

## APPENDICES

## KCAA DRONE LICENSE



**KENYA CIVIL AVIATION AUTHORITY**

**KCAA/OPS/2117/5** **27 FEBRUARY, 2019**

**Alex Githinji Muriungi**  
Moi University  
P. O. Box 3900-30100  
**ELDORET**

Dear Sir,

**RE: USE OF DRONE PARTS FOR ENGINEERING PROJECT**

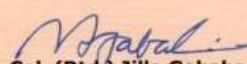
Reference is made to your letter of application dated 30 October, 2018 requesting for the use of drone parts for your final year engineering project.


The Authority has reviewed your request and hereby grants you a temporary authorization to import and use the drone parts for the aforementioned purposes for a period of 3 months from 27 February, 2019 to 27 May, 2019.

This authorization does not override any other government requirements and further indemnifies the Authority from any liabilities related to any incidents and/or accidents as a result of the operations and that the operator shall take full responsibility for such incidents and/or accidents.

The Authority retains the right to vary, cancel/ suspend and/or revoke this authorization.

Yours sincerely,

  
**Col. (Rtd.) Jillo Gababo**  
**FOR: DIRECTOR GENERAL**



P.O. Box 30163 - 0100 GPO Nairobi  
Tel: +254 020 827470 - 5,  
Fax : + 254 020 827 808, 822 300

Website: [www.kcaa.or.ke](http://www.kcaa.or.ke)  
E-mail: [info@kcaa.or.ke](mailto:info@kcaa.or.ke)  
Telegrams 'DIRECTAIR' Nairobi. Telex: 25239 KCAA Hqs KE



## AIR DETECTION AND MONITORING ARDUINO CODE

```

1  /*ECE 590: ENGINEERING PROJECT II
2  * MULTI-PURPOSE QUADCOPTER
3  * : Gas Quality/Leakage Monitoring
4  * EC/09/14
5  * KITHINJI MURIUNGI
6  * 2018/19
7  * Supervisor: Mr. S.B.Kifalu
8  */
9
10 //MQ2 GAS SENSOR
11 #include <ESP8266WiFi.h> //
12 #include <FirebaseArduino.h>
13
14 // Set these to run example.
15 /*#define FIREBASE_HOST "droneiotgasmonitor.firebaseio.com"
16 #define FIREBASE_AUTH "J45Fu0kzUjCUwaKZ8Dg83GFIKF6NBpD4es7No123"*/
17
18
19 #define FIREBASE_HOST "home-automation-7f123.firebaseio.com"
20 #define FIREBASE_AUTH "KKDcaIEctWOHGvHHdtlemDkf1Q37alqqpUcwb123"
21
22 #define WIFI_SSID "Panthers" //Your Wi-Fi SSID
23 #define WIFI_PASSWORD "qwerty90" //Your Wi-Fi Password
24
25 const int mq2GasSensor = A0; // Gas Sensor
26
27 void setup() {
28   Serial.begin(9600);
29
30   pinMode(mq2GasSensor, INPUT);
31
32   // connect to wifi.
33   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
34   Serial.print("connecting");
35   while (WiFi.status() != WL_CONNECTED) {
36     Serial.print(".");
37     delay(500);
38   }
39   Serial.println();
40   Serial.print("connected: ");
41   Serial.println(WiFi.localIP());
42
43   Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
44   }
45
46 void loop() {
47   float newGasReading = analogRead(mq2GasSensor);
48   Firebase.setFloat("air_condition", newGasReading);
49
50   delay(200);
51 }/*END OF THE CODE

```

## QUADCOPTER ARDUINO CODE

```

1  /*ECE 590: ENGINEERING PROJECT II
2  * MULTI-PURPOSE QUADCOPTER
3  * : Arduino Based PID Controller
4  * EC/09/14
5  * KITHINJI MURIUNGI
6  * 2018/19
7  * Supervisor: Mr. S.B.Kifalu
8  */
9  /*
10 *****
11 *-----HARDWARE / CIRCUIT WIRING-----*
12 *****
13 *Motors:(Purple) | Channels: (White) | LED -> 13
|
MPU6050
14 *I -> 4 | I -> 8 | VCC -> Red
|
SDA -> A4 (White)
15 *II -> 6 | II ->9 | GND -> Green
|
SCL -> A5 (Purple)
16 *III -> 7 | III ->10 | Vin -> White
|
Vcc -> Red (5V)
17 *IV -> 5 | IV -> 11 | Batt Monitor -> A0
(Yellow) |
18 */
19 /*
20 *****
21 *-----DECLARATION-----*
22 *****
23 */
24 #include <Wire.h> // Library to enable MPU6050 configurations and
calibrations
25 #define CHANNEL1 0 //Pin 8 : Roll on Mode 2
26 #define CHANNEL2 1 //Pin 9 : Pitch on Mode 2
27 #define CHANNEL3 2 //Pin 10 : Throttle on Mode 2
28 #define CHANNEL4 3 //Pin 11 : Yaw on Mode 2
29
30 #define YAW 0
31 #define PITCH 1
32 #define ROLL 2
33 #define THROTTLE 3
34
35 #define X 0 // X axis
36 #define Y 1 // Y axis
37 #define Z 2 // Z axis
38 #define MPU_ADDRESS 0x68 // I2C address of the MPU-6050
39 #define FREQ 250 // Sampling frequency ///Equivalent to 4000us
: (1/250)

```

```

40  #define SSF_GYRO      65.5  // Sensitivity Scale Factor of the gyro from
datasheet
41
42  #define STOPPED      0
43  #define STARTING    1
44  #define STARTED      2
45  // ----- Receiver variables -----
-----46

  /**
47

  * Received
  flight
  instructions
  formatted
  with
  good
  units,
  in
  that
  order
  : [Yaw,

Pitch, Roll, Throttle]
48  * Units:
49  * - Yaw      : degree/sec
50  * - Pitch    : degree
51  * - Roll     : degree
52  * - Throttle :  $\mu$ s
53  */
54  float instruction[4];
55
56  // Previous state of each channel (HIGH or LOW)
57  volatile byte previous_state[4];
58
59  // Duration of the pulse on each channel of the receiver in  $\mu$ s (must be
within
1000 $\mu$ s & 2000 $\mu$ s)
60  volatile unsigned int pulse_length[4] = {1500, 1500, 1000, 1500};
61
62  // Used to calculate pulse duration on each channel
63  volatile unsigned long current_time;
64  volatile unsigned long timer[4]; // Timer of each channel
65
66  // Used to configure which control (yaw, pitch, roll, throttle) is on
which channel
67  int mode_mapping[4];
68  // ----- MPU variables -----
-----69

  //

```

```

The
RAW
values
got
from
gyro
(in
°/sec)
in
that
order:
X,
Y,
Z
70

    int
gyro_raw[3] = {0,0,0};
71
72 // Average gyro offsets of each axis in that order: X, Y, Z
73 long gyro_offset[3] = {0, 0, 0};
74
75 // Calculated angles from gyro's values in that order: X, Y, Z
76 float gyro_angle[3] = {0,0,0};
77
78 // The RAW values got from accelerometer (in m/sec²) in that order: X,
Y, Z
79 int acc_raw[3] = {0 ,0 ,0};
80
81 // Calculated angles from accelerometer's values in that order: X, Y, Z
82 float acc_angle[3] = {0,0,0};
83
84 // Total 3D acceleration vector in m/s²
85 long acc_total_vector;
86
87 /**
88  * Real measures on 3 axis calculated from gyro AND accelerometer in
that order :
Yaw, Pitch, Roll
89  * - Left wing up implies a positive roll
90  * - Nose up implies a positive pitch
91  * - Nose right implies a positive yaw
92  */
93 float measures[3] = {0, 0, 0};
94
95 // MPU's temperature
96 int temperature;
97
98 // Init flag set to TRUE after first loop
99 boolean initialized;
100 // ----- Variables for servo signal generation -----
-----101
    unsigned
int period; // Sampling period

```

```

102 unsigned long loop_timer;
103 unsigned long now, difference;
104
105 unsigned long pulse_length_esc1 = 1000,
106 pulse_length_esc2 = 1000,
107 pulse_length_esc3 = 1000,
108 pulse_length_esc4 = 1000;
109
110 // ----- Global variables used for PID controller -----
-----111
    float
errors[3]; // Measured errors (compared to instructions) :
[Yaw, Pitch, Roll]
112 float error_sum[3] = {0, 0, 0}; // Error sums (used for integral
component) :
[Yaw, Pitch, Roll]
113 float previous_error[3] = {0, 0, 0}; // Last errors (used for
derivative component)
: [Yaw, Pitch, Roll]
114 // -----
-----115
/**
116  * Status of the quadcopter:
117  *   -> 0 : stopped
118  *   -> 1 : starting
119  *   -> 2 : started
120  */
121 int status = STOPPED;
122 // -----123
    int
battery_voltage;
124 // -----125

126
/**
127  *****
128  *-----INITIALIZATION-----*
129  *****
130  */
131 void setup() {
132 //Serial.begin(115200);
133
134 // Start I2C communication
135
136 Wire.begin();
137 TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
138
139 // Turn LED on during setup
140 pinMode(13, OUTPUT);
141 digitalWrite(13, HIGH);
142
143 // Set pins #4 #5 #6 #7 as outputs
144 DDRD |= B11110000;
145

```

```

146  setupMpu6050Registers();
147
148  calibrateMpu6050();
149
150  configureChannelMapping();
151
152  // Configure interrupts for receiver
153  //NO DigitalRead or DigitalWrite due to memory and unnecessary delays
in the
program
154  PCICR |= (1 << PCIE0); // Set PCIE0 to enable PCMSK0 scan
155  PCMSK0 |= (1 << PCINT0); // Set PCINT0 (digital input 8) to trigger an
interrupt
on state change
156  PCMSK0 |= (1 << PCINT1); // Set PCINT1 (digital input 9) to trigger an
interrupt
on state change
157  PCMSK0 |= (1 << PCINT2); // Set PCINT2 (digital input 10)to trigger an
interrupt
on state change
158  PCMSK0 |= (1 << PCINT3); // Set PCINT3 (digital input 11)to trigger an
interrupt
on state change
159
160  period = (1000000/FREQ) ; // Sampling period in µs
161
162  // Initialize loop_timer
163  loop_timer = micros();
164
165  // Turn LED off now setup is done
166  digitalWrite(13, LOW); // To monitor the status especially the initial
configs :
Arming, Battery, Config
167  }
168
169  /*
170  *****
171  *-----MAIN PROGRAM LOOP-----*
172  *****
173  */
174  void loop() {
175  // 1. First, read raw values from MPU-6050
176  readSensor();
177
178  // 2. Calculate angles from gyro & accelerometer's values
179  calculateAngles();
180
181  // 3. Translate received data into usable values
182  getFlightInstruction();
183
184  // 4. Calculate errors comparing received instruction with measures
185  calculateErrors();
186
187  if (isStarted()) {

```

```

188 // 5. Calculate motors speed with PID controller
189 pidController();
190
191 compensateBatteryDrop(); //Ensures safe flight with enough power:
Failure to
which it will shutdown motors
192 }
193
194 // 6. Apply motors speed
195 applyMotorSpeed(); // Apply motor speeds accordingly as per the
command:
Yaw,Pitch,Roll,Throttle
196 */*
197 *****
198 *-----END MAIN LOOP-----*
199 *****
200 */
201
202 /**
203  * Generate servo-signal on digital pins #4 #5 #6 #7 with a frequency
of 250Hz (4ms
period).
204  * Direct port manipulation is used for performances.... Without this
instant/realtime control isn't possible. This could cause delays in commands.
205  *
206  * This function might not take more than 2ms to run, which lets 2ms
remaining to do
other stuff. ----THIS IS THE SOLE REASON OF ADDRESSING REGISTERS instead of
ordinary ports
207  *https:// www.arduino.cc/en/Reference/PortManipulation
208  */
209
210 /*
211 *****
212 *****
213 *****
214 *****
215 *****
216 *****
217 *****
218 *****
219 *****
220 */
221 void applyMotorSpeed() {
222 // Refresh rate is 250Hz: send ESC pulses every 4000µs
223 while ((now = micros()) - loop_timer < period);
224

```

```

225 // Update loop timer
226 loop_timer = now;
227
228 // Set pins #4 #5 #6 #7 HIGH
229 PORTD |= B11110000;
230
231 // Wait until all pins #4 #5 #6 #7 are LOW
232 while (PORTD >= 16) {
233     now = micros();
234     difference = now - loop_timer;
235
236     if (difference >= pulse_length_esc1) PORTD &= B11101111; // Set pin #4
LOW
237     if (difference >= pulse_length_esc2) PORTD &= B11011111; // Set pin #5
LOW
238     if (difference >= pulse_length_esc3) PORTD &= B10111111; // Set pin #6
LOW
239     if (difference >= pulse_length_esc4) PORTD &= B01111111; // Set pin #7
LOW
240 }
241 }
242
243 /*
244     *****
245     *-----MPU6050 READ FUNCTION-----*
246     *****
247 */
248 //Requesting raw values from MPU6050.
249 void readSensor() {
250     Wire.beginTransaction(MPU_ADDRESS); // Start communicating with the
MPU-6050
251     Wire.write(0x3B); // Send the requested starting register
252     Wire.endTransmission(); // End the transmission
253     Wire.requestFrom(MPU_ADDRESS, 14); // Request 14 bytes from the MPU-6050
254
255     // Wait until all the bytes are received
256     while(Wire.available() < 14);
257
258     acc_raw[X] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the acc_raw[X] variable
259     acc_raw[Y] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the acc_raw[Y] variable
260     acc_raw[Z] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the acc_raw[Z] variable
261     temperature = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the temperature variable
262     gyro_raw[X] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the gyro_raw[X] variable

```



```

263   gyro_raw[Y] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the gyro_raw[Y] variable
264   gyro_raw[Z] = Wire.read() << 8 | Wire.read(); // Add the low and high
byte to
the gyro_raw[Z] variable
265   }
266
267   /*
268   *****
269   *-----CALCULATING ANGLES FUNCTION-----*
270   *****
271   */
272   //Calculate real angles from gyro and accelerometer's values
273   void calculateAngles()
274   {
275     calculateGyroAngles();
276     calculateAccelerometerAngles();
277
278     if (initialized) {
279       // Correct the drift of the gyro with the accelerometer
280       //These can be changed accordingly based on the accuracy of the Sensor
used
281       gyro_angle[X] = gyro_angle[X] * 0.9996 + acc_angle[X] * 0.0004;
282       gyro_angle[Y] = gyro_angle[Y] * 0.9996 + acc_angle[Y] * 0.0004;
283     } else {
284       // At very first start, init gyro angles with accelerometer angles
285       resetGyroAngles();
286
287       initialized = true;
288     }
289
290     // To dampen the pitch and roll angles a complementary filter is used
291     //Software implementation of Complimentary Filter: HPF & LPF
292     //Gyro -- Integral -- HPF
293     //Accelerometer -- Angle Conversion -- LPF
294     measures[ROLL] = measures[ROLL] * 0.9 + gyro_angle[X] * 0.1;
295     measures[PITCH] = measures[PITCH] * 0.9 + gyro_angle[Y] * 0.1;
296     measures[YAW] = -gyro_raw[Z] / SSF_GYRO; // Store the angular motion
for this
axis
297   }
298
299   /*
300   *****
301   *-----CALCULATE GYRO ANGLES ONLY FUNCTION-----*
302   *****
303   */
304   //Calculate pitch & roll angles using only the gyro.
305   void calculateGyroAngles()
306   {
307     // Subtract offsets
308     gyro_raw[X] -= gyro_offset[X];
309     gyro_raw[Y] -= gyro_offset[Y];

```

```

310 gyro_raw[Z] -= gyro_offset[Z];
311
312 // Angle calculation using integration
313 gyro_angle[X] += (gyro_raw[X] / (FREQ * SSF_GYRO));
314 gyro_angle[Y] += (-gyro_raw[Y] / (FREQ * SSF_GYRO)); // Change sign to
match the
accelerometer's one
315
316 // Transfer roll to pitch if IMU has yawed
317 gyro_angle[Y] += gyro_angle[X] * sin(gyro_raw[Z] * (PI / (FREQ *
SSF_GYRO *
180)));
318 gyro_angle[X] -= gyro_angle[Y] * sin(gyro_raw[Z] * (PI / (FREQ *
SSF_GYRO *
319 }
320
321 /*
180)));
322 *****
323 *-----CALCULATE ACCELEROMETER ANGLES ONLY FUNCTION-----*
324 *****
325 */
326 //Calculate pitch & roll angles using only the accelerometer.
327
328 void calculateAccelerometerAngles()
329 {
330 // Calculate total 3D acceleration vector :  $v(X^2 + Y^2 + Z^2)$ 
331 acc_total_vector = sqrt(pow(acc_raw[X], 2) + pow(acc_raw[Y], 2) +
pow(acc_raw[Z], 2));
332
333 // To prevent asin to produce a NaN, make sure the input value is
within [-1;+1]
334 if (abs(acc_raw[X]) < acc_total_vector) {
335 acc_angle[X] = asin((float)acc_raw[Y] / acc_total_vector) * (180 / PI);
//
asin gives angle in radian. Convert to degree multiplying by 180/pi
336 }
337
338 if (abs(acc_raw[Y]) < acc_total_vector) {
339 acc_angle[Y] = asin((float)acc_raw[X] / acc_total_vector) * (180 / PI);
340 }
341 }
342
343 /**
344 *Motors and MPU6050 Configurations
345 *
346 * (A) (B) x
347 * \ / z ↑
348 * X \ |
349 * / \ +-----> y
350 * (C) (D)
351 *
352 *X = Roll
353 *Y = Pitch

```

```

354     *Z = Yaw
355     *
356     * Motors A & D run clockwise.
357     * Motors B & C run counter-clockwise.
358     *
359     * Each motor output value range is about 1000µs to 2000µs
360     */
361
362     /*
363     *****
364     *-----PID CONTROLLER FUNCTION-----*
365     *****
366     */
367
368     void pidController() {
369         //These are the Gain Values that are tuned accordingly
370         //Every slight change in the quadcopter will require tuning.
371         float Kp[3] = {3, 1.3, 1.3}; // P coefficients : Yaw,
Pitch, Roll
372         float Ki[3] = {0.045, 0.03, 0.03}; // I coefficients : Yaw, Pitch,
Roll
373         float Kd[3] = {0, 15, 15}; // D coefficients : Yaw,
Pitch, Roll
374         float delta_err[3] = {0, 0, 0}; // Error deltas : Yaw, Pitch,
Roll
375         float yaw_pid = 0;
376         float pitch_pid = 0;
377         float roll_pid = 0;
378
379         // Initialize motor commands with throttle
380         pulse_length_esc1 = instruction[THROTTLE];
381         pulse_length_esc2 = instruction[THROTTLE];
382         pulse_length_esc3 = instruction[THROTTLE];
383         pulse_length_esc4 = instruction[THROTTLE];
384
385         // Do not calculate anything if throttle is 0
386
387         //PID Calculations
388
389         if (instruction[THROTTLE] >= 1012) {
390             // Calculate sum of errors : Integral coefficients
391             error_sum[YAW] += errors[YAW];
392             error_sum[PITCH] += errors[PITCH];
393             error_sum[ROLL] += errors[ROLL];
394
395             // Calculate error delta : Derivative coefficients
396             delta_err[YAW] = errors[YAW] - previous_error[YAW];
397             delta_err[PITCH] = errors[PITCH] - previous_error[PITCH];
398             delta_err[ROLL] = errors[ROLL] - previous_error[ROLL];
399
400             // Save current error as previous_error for next time
401             previous_error[YAW] = errors[YAW];
402             previous_error[PITCH] = errors[PITCH];
403             previous_error[ROLL] = errors[ROLL];

```

```

404
405 // PID = e.Kp + ∫e.Ki + Δe.Kd ---- PID MODEL----
406 //This is the final calculated PID Model value
407 yaw_pid = (errors[YAW] * Kp[YAW]) + (error_sum[YAW] * Ki[YAW]) +
(delta_err[YAW] * Kd[YAW]);
408 pitch_pid = (errors[PITCH] * Kp[PITCH]) + (error_sum[PITCH] *
Ki[PITCH]) +
(delta_err[PITCH] * Kd[PITCH]);
409 roll_pid = (errors[ROLL] * Kp[ROLL]) + (error_sum[ROLL] * Ki[ROLL]) +
(delta_err[ROLL] * Kd[ROLL]);
410
411 // Calculate pulse duration for each ESC
412 //Due to the nature of the PID, the pulse length of any ESC does not
matter.
413 pulse_length_esc1 = instruction[THROTTLE] + roll_pid + pitch_pid -
yaw_pid;
414 pulse_length_esc2 = instruction[THROTTLE] - roll_pid + pitch_pid +
yaw_pid;
415 pulse_length_esc3 = instruction[THROTTLE] + roll_pid - pitch_pid +
yaw_pid;
416 pulse_length_esc4 = instruction[THROTTLE] - roll_pid - pitch_pid -
yaw_pid;
417 }
418 //The minimum value helps in having the lowest speed and torque just to
indicate
that the rotors are spinning correctly and the drone is fully armed
419 pulse_length_esc1 = minMax(pulse_length_esc1, 1100, 2000); //Min-Max
Mapping
Accordingly 1100/1000 - 2000
420 pulse_length_esc2 = minMax(pulse_length_esc2, 1100, 2000);
421 pulse_length_esc3 = minMax(pulse_length_esc3, 1100, 2000);
422 pulse_length_esc4 = minMax(pulse_length_esc4, 1100, 2000);
423 }
424
425 /*
426 *****
427 *-----ERROR CALCULATION FUNCTION-----*
428 *****
429 */
430 //Calculate errors of Yaw, Pitch & Roll: this is simply the difference
between the
measure and the command.
431 void calculateErrors() {
432 //This facilitates in Closed-Loop Feedback system
433 errors[YAW] = instruction[YAW] - measures[YAW];
434 errors[PITCH] = instruction[PITCH] - measures[PITCH];
435 errors[ROLL] = instruction[ROLL] - measures[ROLL];
436 }
437
438 /**
439 * Calculate real value of flight instructions from pulses length of
each channel.
440 *
441 * - Roll : from -33° to 33°

```

```

442  * - Pitch      : from -33° to 33°
443  * - Yaw        : from -180°/sec to 180°/sec
444  * - Throttle   : from 1000µs to 1800µs
445  */
446
447  /*
448  *****
449  *-----FLIGHT INSTRUCTIONS FUNCTION-----*
450  *****
451  */
452  void getFlightInstruction() {
453  instruction[YAW] = map(pulse_length[mode_mapping[YAW]], 1000, 2000, -
180,
180);
454  instruction[PITCH] = map(pulse_length[mode_mapping[PITCH]], 1000, 2000,
33,
-33); //Since the operates on "Reverse Mode"
455  instruction[ROLL] = map(pulse_length[mode_mapping[ROLL]], 1000, 2000, -
33,
33);
456  instruction[THROTTLE] = map(pulse_length[mode_mapping[THROTTLE]], 1000,
2000,
457  }
1000, 1800); // Get some room to keep control at full speed
458
459  /**
460   * Customize mapping of controls: set here which command is on which
channel and call
461   * this function in setup() routine.
462   */
463
464  /*
465  *****
466  *-----CHANNEL MAPPING FUNCTION-----*
467  *****
468  */
469  void configureChannelMapping() {
470  //Mapping according to the TRANSMITTER Configuration
471  //Using these mapping, confirm that the Transmitter is set to MODE 2
(LEFT:
Throttle & Yaw ---- RIGHT: Pitch & Roll)
472  mode_mapping[YAW] = CHANNEL4;
473  mode_mapping[PITCH] = CHANNEL2;
474  mode_mapping[ROLL] = CHANNEL1;
475  mode_mapping[THROTTLE] = CHANNEL3;
476  }
477
478  /*
479  *****
480  *-----MPU6050 CONFIGURATION FUNCTION-----*
481  *****
482  */
483
484  /**

```

```

485     * Configure gyro and accelerometer precision as following:
486     * Check from the MPU6050 Datasheet
487     * - accelerometer:  $\pm 8g$ 
488     * - gyro:  $\pm 500^\circ/s$ 
489     */
490     void setupMpu6050Registers() {
491         // Configure power management
492         Wire.beginTransmission(MPU_ADDRESS); // Start communication with MPU
493         Wire.write(0x6B); // Request the PWR_MGMT_1 register
494         Wire.write(0x00); // Apply the desired configuration to the
register
495         Wire.endTransmission(); // End the transmission
496
497         // Configure the gyro's sensitivity
498         Wire.beginTransmission(MPU_ADDRESS); // Start communication with MPU
499         Wire.write(0x1B); // Request the GYRO_CONFIG register
500         Wire.write(0x08); // Apply the desired configuration to the
register :  $\pm 500^\circ/s$ 
501         Wire.endTransmission(); // End the transmission
502
503         // Configure the accelerometer's sensitivity
504         Wire.beginTransmission(MPU_ADDRESS); // Start communication with MPU
505         Wire.write(0x1C); // Request the ACCEL_CONFIG register
506         Wire.write(0x10); // Apply the desired configuration to the
register :  $\pm 8g$ 
507         Wire.endTransmission(); // End the transmission
508
509         // Configure low pass filter
510         Wire.beginTransmission(MPU_ADDRESS); // Start communication with MPU
511         Wire.write(0x1A); // Request the CONFIG register
512         Wire.write(0x03); // Set Digital Low Pass Filter about ~43Hz
513         Wire.endTransmission(); // End the transmission
514     }
515
516     /*
517     *****
518     *-----MPU6050 CALIBRATION FUNCTION-----*
519     *****
520     */
521
522     /**
523     * Calibrate MPU6050: take 2000 samples to calculate average offsets.
524     * During this step, the quadcopter needs to be static and on a
horizontal surface.
525     *
526     * This function also sends low throttle signal to each ESC to init and
prevent them
beeping annoyingly.
527     *
528     * This function might take ~2sec for 2000 samples.
529     */
530     void calibrateMpu6050()
531     {

```

```

532 // This takes place at the initial stage when the Quad is first tuned
ON.
533 //This is Indicated by the continuous lighting of the GREEN LED.
534 //When the GREEN LED Goes off, it indicates that the configurations are
over.
535 int max_samples = 2000;
536
537 for (int i = 0; i < max_samples; i++) {
538   readSensor();
539
540   gyro_offset[X] += gyro_raw[X];
541   gyro_offset[Y] += gyro_raw[Y];
542   gyro_offset[Z] += gyro_raw[Z];
543
544   // Generate low throttle pulse to init ESC and prevent them beeping
545   PORTD |= B11110000; // Set pins #4 #5 #6 #7 HIGH
546   delayMicroseconds(1000); // Wait 1000µs
547   PORTD &= B00001111; // Then set LOW
548
549   // Just wait a bit before next loop
550   delay(3);
551 }
552
553 // Calculate average offsets
554 gyro_offset[X] /= max_samples;
555 gyro_offset[Y] /= max_samples;
556 gyro_offset[Z] /= max_samples;
557 }
558
559 /**
560  * Make sure that given value is not over min_value/max_value range.
561  * @param float value      : The value to convert
562  * @param float min_value  : The min value
563  * @param float max_value  : The max value
564  */
565 float minMax(float value, float min_value, float max_value) {
566   if (value > max_value) {
567     value = max_value;
568   } else if (value < min_value) {
569     value = min_value;
570   }
571
572   return value;
573 }
574
575 /*
576  *****
577  *-----WHEN QUAD IS STARTED FUNCTION-----*
578  *****
579  */
580
581 /**
582  * Return whether the quadcopter is started.

```

```

583     * To start the quadcopter, move the left stick in bottom left corner
then, move it
back in center position.
584     * To stop the quadcopter move the left stick in bottom right corner.
585     */
586     bool isStarted()
587     {
588         // When left stick is moved in the bottom left corner
589         if (status == STOPPED && pulse_length[mode_mapping[YAW]] <= 1012 &&
pulse_length[mode_mapping[THROTTLE]] <= 1012) {
590             status = STARTING;
591         }
592
593         // When left stick is moved back in the center position
594         if (status == STARTING && pulse_length[mode_mapping[YAW]] == 1500 &&
pulse_length[mode_mapping[THROTTLE]] <= 1012) {
595             status = STARTED;
596
597             // Reset PID controller's variables to prevent confusion during the
start
598             resetPidController();
599
600             resetGyroAngles();
601         }
602
603         // When left stick is moved in the bottom right corner
604         if (status == STARTED && pulse_length[mode_mapping[YAW]] >= 1988 &&
pulse_length[mode_mapping[THROTTLE]] <= 1012) {
605             status = STOPPED;
606             // Make sure to always stop motors when status is STOPPED
607             stopAll();
608         }
609
610         return status == STARTED;
611     }
612
613     /*
614     *****
615     *-----RESET GYRO ANGLES FUNCTION-----*
616     *****
617     */
618     //Reset gyro's angles with accelerometer's angles.
619     void resetGyroAngles()
620     {
621         gyro_angle[X] = acc_angle[X];
622         gyro_angle[Y] = acc_angle[Y];
623     }
624
625     /*
626     *****
627     *-----STOPPING ALL MOTOR FUNCTION-----*
628     *****
629     */
630     //Reset motors' pulse length to 1000µs to totally stop them.

```



```

631 void stopAll()
632 {
633     pulse_length_esc1 = 1000;
634     pulse_length_esc2 = 1000;
635     pulse_length_esc3 = 1000;
636     pulse_length_esc4 = 1000;
637 }
638
639 /*
640  *****
641  *-----RESET PID CONTROLLER FUNCTION-----*
642  *****
643  */
644 //Reset all PID controller's variables.
645 void resetPidController()
646 {
647     errors[YAW] = 0;
648     errors[PITCH] = 0;
649     errors[ROLL] = 0;
650
651     error_sum[YAW] = 0;
652     error_sum[PITCH] = 0;
653     error_sum[ROLL] = 0;
654
655     previous_error[YAW] = 0;
656     previous_error[PITCH] = 0;
657     previous_error[ROLL] = 0;
658 }
659
660 /*
661  *****
662  *-----BATTERY COMPENSATION FUNCTION-----*
663  *****
664  */
665 //Compensate battery drop applying a coefficient on output values
666 void compensateBatteryDrop() {
667     if (isBatteryConnected()) {
668         pulse_length_esc1 += pulse_length_esc1 * ((1240 - battery_voltage) /
669 (float)
3500);
669         pulse_length_esc2 += pulse_length_esc2 * ((1240 - battery_voltage) /
670 (float)
3500);
670         pulse_length_esc3 += pulse_length_esc3 * ((1240 - battery_voltage) /
671 (float)
3500);
671         pulse_length_esc4 += pulse_length_esc4 * ((1240 - battery_voltage) /
672 (float)
3500);
672     }
673 }
674
675 /*
676  *****

```

```

677  *-----READ BATTERY VOLTAGE FUNCTION-----*
678  ****
679  */
680  //Read battery voltage & return whether the battery seems connected
681  bool isBatteryConnected() {
682  // A complementary filter is used to reduce noise.
683  // 0.09853 = 0.08 * 1.2317.
684  battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) *
0.09853;
685
686  return battery_voltage < 1240 && battery_voltage > 800;
687  }
688
689  /*
690
691  *---INTERRUPTS TO ENABLE TRANSMITTER-RECEIVER-CONTROLLER COMMUNICATE--
--*
692
693  */
694
695  /**
696  * This Interrupt Sub Routine is called each time input 8, 9, 10 or 11
changed state.
697  * Read the receiver signals in order to get flight instructions.
698  * This routine must be as fast as possible to prevent main program to
be messed up.
699  * The trick here is to use port registers to read pin state.
700  * It is less convenient but more efficient, which is the most
important here.
701  */
702  ISR(PCINT0_vect) {
703  current_time = micros();
704
705  // Channel 1 -----706
    if
(PINB & B00000001) { // Is input 8 high ?
707  //Doing (PINB & B00000001) is the same as digitalRead(8) with the
advantage of
using less CPU loops.
708  if (previous_state[CHANNEL1] == LOW) { // Input 8
changed from 0 to 1 (rising edge)
709  previous_state[CHANNEL1] = HIGH; // Save current
state
710  timer[CHANNEL1] = current_time; // Save current
time
711  }
712  } else if (previous_state[CHANNEL1] == HIGH) { // Input 8
changed from 1 to 0 (falling edge)
713  previous_state[CHANNEL1] = LOW; // Save current
state
714  pulse_length[CHANNEL1] = current_time - timer[CHANNEL1]; // Calculate
pulse duration & save it

```

```

715     }
716
717     // Channel 2 -----718
    if
(PINB & B00000010) { // Is input 9
high ?
719     //Doing (PINB & B00000001) is the same as digitalRead(9) with the
advantage of
using less CPU loops.
720     if (previous_state[CHANNEL2] == LOW) { // Input 9
changed from 0 to 1 (rising edge)
721     previous_state[CHANNEL2] = HIGH; // Save current
state
722     timer[CHANNEL2] = current_time; // Save current
time
723     }
724     } else if (previous_state[CHANNEL2] == HIGH) { // Input 9
changed from 1 to 0 (falling edge)
725     previous_state[CHANNEL2] = LOW; // Save current
state
726     pulse_length[CHANNEL2] = current_time - timer[CHANNEL2]; // Calculate
pulse duration & save it
727     }
728
729     // Channel 3 -----730
    if
(PINB & B00000100) { // Is input 10
high ?
731     //Doing (PINB & B00000001) is the same as digitalRead(10) with the
advantage
of using less CPU loops.
732     if (previous_state[CHANNEL3] == LOW) { // Input 10
changed from 0 to 1 (rising edge)
733     previous_state[CHANNEL3] = HIGH; // Save current
state
734     timer[CHANNEL3] = current_time; // Save current
time
735     }
736     } else if (previous_state[CHANNEL3] == HIGH) { // Input 10
changed from 1 to 0 (falling edge)
737     previous_state[CHANNEL3] = LOW; // Save current
state
738     pulse_length[CHANNEL3] = current_time - timer[CHANNEL3]; // Calculate
pulse duration & save it
739     }
740
741     // Channel 4 -----
742     if (PINB & B00001000) { // Is input 11
high ?
743     //Doing (PINB & B00000001) is the same as digitalRead(11) with the
advantage
of using less CPU loops.
744     if (previous_state[CHANNEL4] == LOW) { // Input 11
changed from 0 to 1 (rising edge)

```

```
745  previous_state[CHANNEL4] = HIGH; // Save current
state
746  timer[CHANNEL4] = current_time; // Save current
time
747  }
748  } else if (previous_state[CHANNEL4] == HIGH) { // Input 11
changed from 1 to 0 (falling edge)
749  previous_state[CHANNEL4] = LOW; // Save current
state
750  pulse_length[CHANNEL4] = current_time - timer[CHANNEL4]; // Calculate
pulse duration & save it
751  }
752  }/*END OF THE CODE
753  *
754  *-----*
755  *-----Signature-----*
756  *-----*
757  *
758  *MULTI-PURPOSE QUADCOPTER
759  *:Arduino-Based PID Controller
760  *FINAL YEAR ENGINEERING
761  *ECE 590 : ENGINEERING PROJECT II
762  *KITHINJI MURIUNGI
763  *EC/09/14
764  *2018/19
765  *Supervised By: Mr. S. B. Kifalu
766  */
767
```