الآيـــــــــة

بِسۡــــمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيــــمِ :

﴿قَالَ الَّذِي عِندَهُ عِلْمٌ مِّنَ الْكِتَابِ أَنَا آتِيكَ بِهِ قَبْلَ أَن يَرْتَدَّ إِلَيْكَ طَرْفُكَ ۚ فَلَمَّا رَآهُ مُسْتَقِرًّا عِندَهُ قَالَ هَٰذَا مِن فَضْلِ رَبِّي لِيَبْلُوَنِي أَأَشْكُرُ أَمْ أَكْفُرُ ۖ وَمَن شَكَرَ فَإِنَّمَا يَشْكُرُ لِنَفْسِهِ ۖ وَمَن كَفَرَ فَإِنَّ رَبِّي غَنِيٌّ كَرِيمٌ﴾

النمل – الآية40

Abstract

The control system of UAV represents the heart of the UAV; the ability of UAV to do a certain task is mainly dependent on accuracy of control system inside it. This thesis is a project to build a control system for a Quadcopter platform.

This is a report of a two section project: theoretical analysis section which contains the Quadcopter modeling and the developed PID algorithm for controlling the Quadcopter, and hardware implementation where the components of the Quadcopter were chosen and the interface between them was accomplished

The Final code was developed then implemented into Arduino Uno microcontroller and the Quadcopter was able to hover. To achieve better response and more stable flight it’s recommended to use both the gyro and accelerometer.

التجريد

نظام التحكم للطائرة بدون طيار يمثل قلبها، وتعتمد قدرة الطائرة على أداء مهمة ما على دقة نظام التحكم بداخلها. هذه الأطروحة عبارة عن مشروع لبناء نظام تحكم لنموذج طائرة بدون طيار رباعية الدفع.

المشروع ينقسم إلى قسمين اساسيين، التحليل النظري والتطبيق العملي لنظام التحكم في الطائرة. احتوى التحليل النظري على نمذجة الطائرة وتطوير خوارزمية المتحكم التناسبي التكاملي التفاضلي للتحكم في الطائرة. أما التطبيق العملي تضمن اختيار مكونات الطائرة والربط بينها ماديا وبرمجيا.

البرنامج النهائي تم تحميله في المعالج الدقيق وتمكنت الطائرة من التحليق. ولكي نحصل على طيران اكثر استقراراً يوصى باستخدام حساس الجايرو ومقياس التسارع الخطي معا.

Acknowledgment

First of all, all our gratitude is for Allah, for giving us the knowledge and lighting our way through, then our sincere appreciation for our supervisor Dr. Osman Imam for his advice, support and guidance through the entire project. Special thanks to Lecturer Raheeg Osama Wahbi who shared with us the main principles of a successful preparation for a final project, and reviewing the context of this project with us.

We would love to thank Aeronautical Research Center (ARC) for sharing with us both knowledge and materials gracefully. And to the seniors who provided us with basic requirements of the project.

Dedication

To those who helped us at reaching our target. To the University and Department that enlightened us through the years and for their deep concern and efforts. To the families, for always being there for us and encouraging us all the way. And to the Dear friend Omer Mukhtar who had the main role of providing us with the components of the project from China. Last but not least, to all dear friends who gave us hope in times of despair.

Contents

الآيـــــــــة I

Abstract II

التجريد III

Acknowledgment IV

Dedication V

Contents VI

List of figures X

List of tables XII

Glossary XIII

List of Symbols XV

CHAPTER ONE: INTRODUCION 1

1.1 Overview 1

1.2 Motivation 1

1.3 Objectives 2

1.4 Problem statement 2

1.5 Proposed solution 2

1.6 Methodology 2

1.7 Outline 3

CHAPTER TWO: LITERATURE REVIEW 4

2.1 History and Background 4

2.1.1 Quadcopter operation 4

2.1.2 Advantages of Quadcopter 6

2.1.3 Disadvantages of Quadcopter 6

2.2 History 7

2.3 Arduino microcontroller 11

2.3.1 Features of Arduino 11

2.3.2 Arduino IDE 12

2.4 Methods of control 12

2.4.1 Remote control UAV 12

2.4.2 Autonomous UAV 15

CHAPTER THREE: MODULING AND PID CONTROLLER DESIGN 18

3.1 Modeling of Quadcopter 18

3.2 PID controller 21

3.2.1 Characteristics of P, I, and D controllers 21

3.2.2 Manual Tuning of PID 22

3.2.3 PID Controller in Quadcopter 23

3.2.4 Classic PID equations 26

3.2.5 Transformation of the PID controller equations 26

3.2.6 PID output for Quadcopter movements 27

CHAPTER FOUR: HARDWARE IMPLEMENTATION 29

4.1 Frame 29

4.2 Battery 30

4.3 Brushless DC motor (BLDC) 30

4.3.1 Basic concepts when selecting Motors 31

4.4 Propellers 32

4.5 Transmitter and Receiver 33

4.5.1 Receiver connection with Arduino UNO.. 34

4.6 Control board 35

4.7 ITG3200 Gyroscope.. 37

4.7.1 Features of ITG3200 Gyroscope... 37

4.7.2 Gyro connection with Arduino UNO.. 37

4.8 Electronic Speed Controlled (ESC) 38

4.8.1 ESCs connection with Arduino... 39

4.9 Bluetooth module connection with Arduino UNO 40

CHAPTER FIVE: QUADCOPTER SOFTWARE.. 41

5.1 The transmitter and receiver 42

5.1.1 Transmitter and signal transmission 42

5.1.2 Receiver interface with Arduino UNO 43

5.2 Gyroscope (ITG3200).. 45

5.2. 1 **I2C** Inter- Integrated Circuit 45

5.2.2 Connecting to sensor procedure.. 46

5.2.3 Configuration of gyro parameters... 46

5.2.4 Obtaining readings.. 48

5.2.5 Gyro calibration 49

5.2.6 Filtering... 49

5.3 ESCs interface with Arduino UNO.. 51

5.3.1 Final outputs of ESCs 51

5.4 Control the Quadcopter by mobile phone.. 52

5.4.1 Program algorithm 52

5.4.2 Application.. 52

5.4.3 The operation 53
CHAPTER SIX: CONSLUSION AND RECOMMENDATION 55
6.1 Conclusion 55
6.2 Recommendations 56
6.3 Future work 57
6.4 References 58
6.5 Appendixes 59
Appendix A: the Quadcopter schematic 59
Appendix B: Arduino Pinout diagram 60
Appendix C: The code of Quadcopter 61

List of figures

Figure 1: Yaw, pitch and roll rotations of a common Quadrotor....................................... 5

Figure 2: Illustration of the various movements of a Quadrotor. 5

Figure 3: Etienne Oehmichen’s second prototype. 8

Figure 4: Convertawings Model “A” helicopter 8

Figure 5: V-22 Ospray 9

Figure 6: Quadrotors in the Private Sector and in Universities 10

Figure 7: Generic PWM Pulse 14

Figure 8: Partial PPW Pulse.. 14

Figure 9: Block diagram of Quadcopter control system 21

Figure 10: PID output of the transformed PID equations 27

Figure 11: F450 PCB Frame 29

Figure 12: 3S 11.1V 2600MAH 30C LiPo battery 30

Figure 13: A 2212 1000 KV BLDC (Brushless DC Motor).. 31

Figure 14: 1045 fixed-pitch, Carbon fiber Propeller 32

Figure 15: 8 channels Futaba RC.. 33

Figure 16: Receiver... 33

Figure 17: RC interface with Arduino Uno 34

Figure 18: Arduino UNO 35

Figure 19: GY-85 IMU (Inertial Measurement Unit 36

Figure 20: GY-85 connected to I^2 C port of Arduino.. 37

Figure 21: 30A Brushless ESC 38

Figure 22: ESCs interface with Arduino Uno... 39

Figure 23: Bluetooth communication 40

Figure 24: Control system’s flow chart 41

Figure 25: Schematic architecture of $\mathbf{I2C}$ interface 45

Figure 26: Recursive Filter block diagram 50

Figure 27: Command format.. 53

Figure 28: Control system’s Block Diagram 54

List of tables

Table 1: Characteristics of PID gains 22

Table 2: Specifications of A2212 / 920 KV out runner motor 32

Table 3: Specification for 30A Brushless ESC.. 38

Table 4: RC readings 42

Table 5: Register 22- DLPF, Full Scale.. 46

Table 6: LPF Bandwidth & internal Sample Rate 47

Table 7: Register 21 bits configuration... 47

Table 8: Registers 27 to 34 48

Glossary

ADC	Analog to digital converter
BLDC	Brushless DC Motor
BT	Bluetooth
DAC	Digital to Analog Converter
DLPF	Digital Low Pass Filter
ESCs	Electronic Speed Controllers
FHSS	Frequency-Hopping Spread Spectrum
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift-Keying
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LAN	Local Area Network
LQ	Linear Quadratic
MEMS	Micro Electronic Mechanical System
PAN	Personal Area Network
PID	Proportional Integral Derivative
PPM	Pulse Width Modulation
PWM	Pulse Position Modulation
RC	Remote Control

RPM	Revolution Per Minute
SCL	Serial Clock Line
UAV	Unmanned Aerial Vehicles
SDA	Serial Data Line
USB	Universal Serial Bus
VTOL	Vertical Take-Off and Landing

List of Symbols

Symbol	Unit	Description
θ	rad	Pitch angle
$\theta^{\cdot}$	rad.s^{-1}	Pitch angle rate
ρ	kg.m^{-3}	Air density at sea level and 20◦C
τs	-	Time constant
∅	rad	Roll angle
$∅^{\cdot}$	rad.s^{-1}	Roll angle rate
φ	rad	Yaw angle
$φ^{\cdot}$	rad.s^{-1}	Yaw angle rate
ω	rad.s^{-1}	Propeller angular velocity
ω^{-}	rad.s^{-1}	Quadrotor's angular velocities (P Q R)
D_P	m	Propeller diameter
F_{net}	N	Combination of all the forces acting on the quadrotor F_{net}= ($FxFyFz$)
F_P	N	Total thrust generated by the propellers F_P = (F_{px} F_{Py} F_{Pz})
g	$m.s^{-2}$	Earth's gravity (constant value of 9.81$m.s^{-2}$)
I	$kg.m^2$	Inertia matrix of the Quadrotor.
M_{net}	-	Sum of all the moments acting on the Quadrotor M_{net} = (Mx My Mz).
m	kg	Mass of the Quadrotor

P	$rad.s^{-1}$	Angular speed around the *ux0* axis of the Quadrotor
$P^{\cdot}$	$rad.s^{-2}$	Angular acceleration of the Quadrotor along the *ux0* axis of Inertial reference frame
Q	$rad.s^{-1}$	Angular speed around the *uy0* axis of the Quadrotor.
$Q^{\cdot}$	$rad.s^{-2}$	Angular acceleration of the Quadrotor along the *uy0* axis of the inertial reference frame
R	$rad.s^{-1}$	Angular speed around the *uz0* axis of the Quadrotor
$R^{\cdot}$	$rad.s^{-2}$	Angular acceleration of the Quadrotor along the *uz0* axis of the inertial reference frame
S	-	Rotation matrix (also known as direction cosine matrix)
T	N	Propeller thrust
$V^{\cdot}$	$m.s^{-2}$	Linear acceleration of the Quadrotor along the *uy* axis of the inertial reference frame
$W^{\cdot}$	$m.s^{-2}$	Linear acceleration of the Quadrotor along the *uz* axis of the inertial reference frame
$U^{\cdot}$	$m.s^{-2}$	Linear acceleration of the Quadrotor along the *ux* axis of the inertial reference frame

CHAPTER ONE: INTRODUCION

1.1 Overview

A quad-rotor, or quad-rotor helicopter, is an aircraft that becomes airborne due to the lift force provided by four rotors usually mounted in cross configuration, hence its name. It is an entirely different vehicle when compared with a helicopter controlled by adjusting rotor pitch but quad-copter is controlled changing the speeds of rotation of motors.

At present, there are three main areas of quad-rotor development: military, transportation (of goods and people) and Unmanned Aerial Vehicles (UAVs). UAVs can be classified into two major groups heavier-than-air and lighter-than-air. These two groups self-divide in many other that classify aircrafts according to motorization, type of liftoff and many other parameters. Vertical Take-Off and Landing (VTOL) UAVs like Quadrotor have several advantages over fixed-wing airplanes. They can move in any direction and are capable of hovering and fly at low speeds. In addition, the VTOL capability allows deployment in almost any terrain while fixed-wing aircraft require a prepared airstrip for takeoff and landing

1.2 Motivation

UAVS (unmanned aerial vehicle) are the state of art in aeronautical engineering especially Avionics, the control and stability of UAV represent a challenge that needs to be met.

Rotorcrafts have witnessed an incredible evolution in the last years. Universities, students and researchers continuously work to introduce more robust controllers and modeling techniques, so that they can provide detailed and accurate representations of real-life quad rotors. Accordingly, a lot of information, knowledge and greater understanding can be gained by studying the researches in this topic.

The implementation in simulation software takes place in ideal conditions; the results may be ideal in the simulation, but results will experience internal and external noise when applied to the outside world giving inaccurate outputs. The real implementation will give actual result, and give the opportunity to deal with these problems and fix them; once the problems are fixed further improvements can be added.

1.3 Objectives

- Developing a PID controller for the Quadcopter.
- Build the Quadcopter using available materials and hardware components.

1.4 Problem statement

In theory, the control and stability of UAV is ongoing challenge furthermore stability and control is additional task for Quadcopter for complexity, and addition of four rotors to be controlled to achieve maneuver, lift and thrust.

1.5 Proposed solution

The PID controller is proposed to carry out the control task for Quadcopter.

1.6 Methodology

An analytical and theoretical approach is adopted to design a PID controller. Hardware implementation of Quadcopter is used to implement the PID designed, a collection of hardware components were chosen to build up the model, tasks carried out on the build Quadcopter with proposed PID controller.

1.7 Outline

In chapter one, introduction included the definition of Quadcopter, the motivations for this thesis and the objectives in addition to the problem statement with the proposed solutions. The methodology that the project was sequenced and executed according to was finally illustrated.

Moving to chapter two which the operation, advantages, and disadvantages of Quadcopter were illustrated. A historical review of Quadcopter was provided, followed by the used microcontroller, methods of control of Quadcopter illustrated.

Chapter three included the modeling of the Quadcopter stating the forces, moments, and dynamics of Quadcopter then PID controller concept and development.

Followed by chapter four where the chosen components of the Quadcopter were stated in addition to the connections between them.

In chapter five the code which used to interface between components was illustrated.

Chapter six included the conclusion of the final implementation of the Quadcopter and the outcomes, followed by the recommendations regarding the limitations and proposed solutions for encountered problems. At last the future work based on this thesis was stated.

CHAPTER TWO: LITERATURE REVIEW

2.1 History and Background

2.1.1 Quadcopter operation

"Each rotor in a Quadrotor is responsible for a certain amount of thrust and torque about its center of rotation, as well as for a drag force opposite to the rotorcraft's direction of flight. The Quadrotor's propellers are not all alike. In fact, they are divided in two pairs, two pusher blades and two puller blades, which work in contra-rotation. As a consequence, the resulting net torque can be null if all propellers turn with the same angular velocity, thus allowing for the aircraft to remain still around its center of gravity.

In order to define an aircraft's orientation (or attitude) around its center of mass, aerospace engineers usually define three dynamic parameters, the angles of yaw, pitch and roll. This is very useful because the forces used to control the aircraft act around its center of mass, causing it to pitch, roll or yaw.

Changes in the pitch angle are induced by contrary variation of speeds in propellers 1 and 3 (see Figure 1), resulting in forward or backwards translation. If we do this same action for propellers 2 and 4, we can produce a change in the roll angle and we will get lateral translation. Yaw is induced by mismatching the balance in aerodynamic torques (i.e. by offsetting the cumulative thrust between the counter-rotating blade pairs). So, by changing these three angles in a Quadrotor we are able to make it maneuver in any direction (Figure 2)."[1]

---

[1]Domingues, J. M. B. (2009). "Quadcopter prototype." Grau de Mestre emEngenharia Mecânica,(2009, Oct).

[2]Henriques, B. S. M. (2011). Estimation and control of a quadrotor attitude, Master's thesis, Instituto

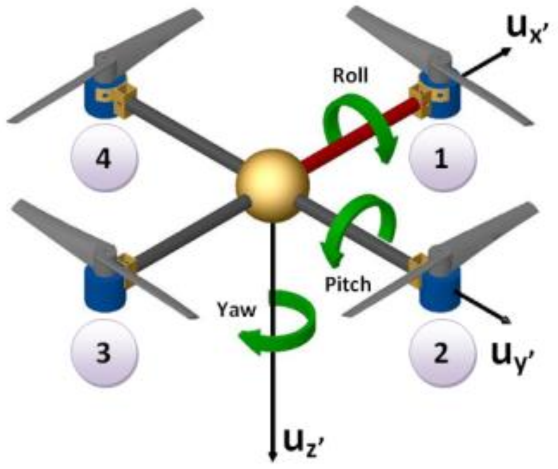


Figure 1: Yaw, pitch and roll rotations of a common Quadrotor.

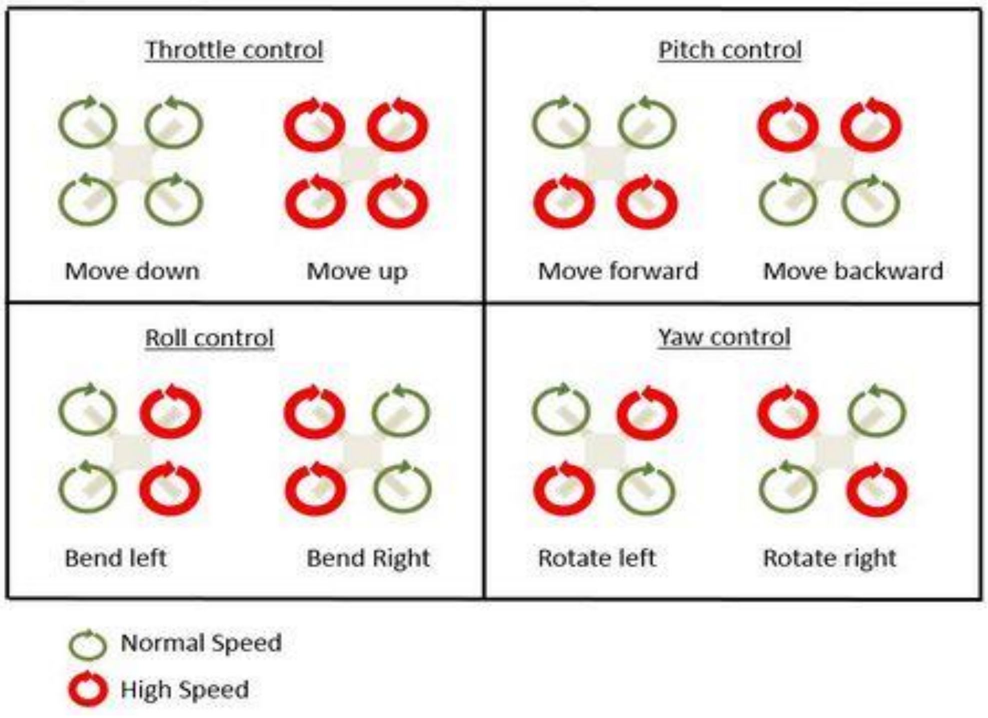


Figure 2: Illustration of the various movements of a Quadrotor.

2.1.2 Advantages of Quadcopter

"There are many advantages to Quadcopter compared to other aircrafts. A Quadcopter does not require a large area to obtain lift, like a fixed wing aircraft does. The Quadcopter creates thrust with four evenly distributed motors along its frame. A helicopter suffers from torque issue due to its main rotor. The design of the Quadcopter does not suffer from the same torque issues as the helicopter. The counter balancing forces of the spinning motors cancel out the torque forces caused by each motor causing the Quadcopter to balance itself. Because the Quadcopter uses four rotors instead of one main rotor, it requires less kinetic energy per rotor for the same amount of thrust when compared to the helicopter. Due to this and its symmetrical design, a Quadcopter's maintenance and manufacturing costs are relatively lower than other aircrafts"[2].

At a small size, Quadcopters are cheaper and more durable than conventional helicopters due to their mechanical simplicity. Their smaller blades are also advantageous because they possess less kinetic energy, reducing their ability to cause damage. For small-scale Quadcopter, this makes the vehicles safer for close interaction. It is also possible to fit Quadcopter with guards that enclose the rotors, further reducing the potential for damage.

2.1.3 Disadvantages of Quadcopter

As size increases, fixed propeller Quadcopter develop disadvantages over conventional helicopters. Increasing blade size increases their momentum. This means that changes in blade speed take longer, which negatively impacts control. At the same time, increasing blade size improves efficiency as it takes less energy to generate thrust by moving a large mass of air at a slow speed than by moving a small mass of air at high speed. Therefore, increasing efficiency comes at the cost of control. Helicopters do not

[2]Henriques, B. S. M. (2011). Estimation and control of a quadrotor attitude, Master's thesis, Instituto Superior Técnico.

experience this problem as increasing the size of the rotor disk does not significantly impact the ability to control blade pitch[3].

2.2 History

The concept of a Quadrotor is actually not new; amongst the first sketches of rotorcrafts, appeared the first ideas for the concept of Quadrotors. In 1907, Louis and Jacques Breguet associated to Professor Charles Richet developed the “Gyroplane No.1”, propelled by four 4-blade biplane rotors mounted on the extremity of a cross-shaped structure. To obtain stability, the rotorcraft counted on diagonally opposed rotors to rotate in opposite directions thus being able to cancel the torque produced by each pair with the other pair of rotors. Although able to achieve lift, the stability necessary to consider it a proper flight was not attained and the short 60cm flight was only made possible thanks to the four arms supporting and stabilizing the craft. Later that year, a free flight was accomplished, reaching the height of over a meter. However the pilot had no steering nor forward propulsion means and the gyroplane could not be considered to be controllable nor perfectly stabilized.

In the 1920’s, Etienne Oehmichen underwent several experiments concerning rotorcrafts. His second prototype (see Figure 3) baptized “Oehmichen No.2” had four rotors and eight propellers mounted on a cross shaped frame. Five propellers were used to obtain the lift and lateral stability thanks to the change in the angle of the blades, two were used to give horizontal propulsion and an additional propeller was used to steer the vehicle. With this configuration flights of several minutes were made possible and a 1Km close-circuit controlled flight was achieved.

[3]https://en.wikipedia.org/wiki/Quadcopter

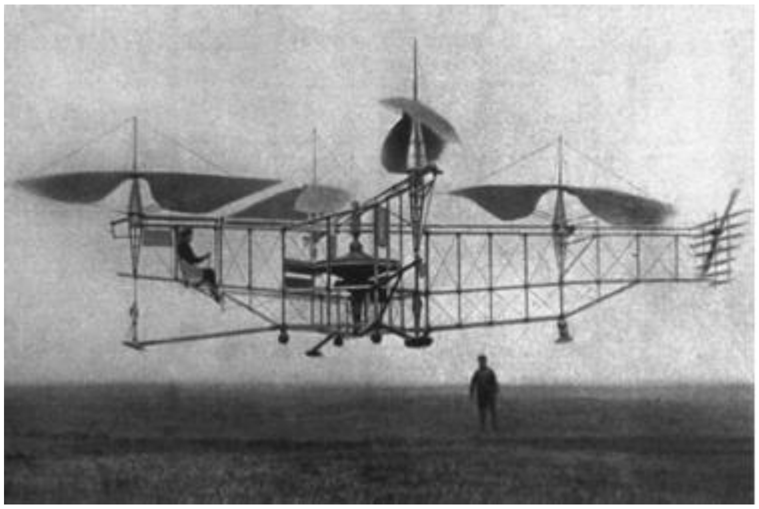


Figure 3: Etienne Oehmichen's second prototype.

Later in 1956, a Quadrotor helicopter prototype called "Convertawings Model A" (see Figure 4) was designed both for military and civilian use. It was controlled by varying the thrust between rotors, and its flights were a success, even in forward flight. The project ended mainly due to the lack of demand for the aircraft.

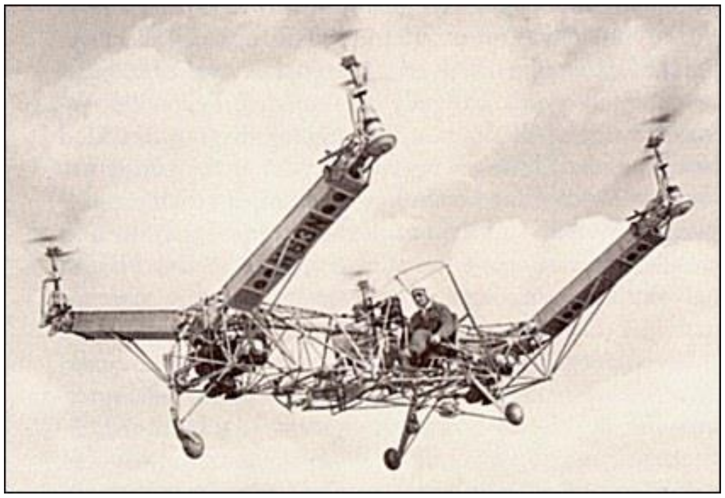


Figure 4: Convertawings Model "A" helicopter

Recently there has been an increasing interest in Quadrotor designs. Bell is working on a quad tilt rotor to overcome the V-22 Ospray (see Figure 5), capable of carrying a large payload, achieving high velocity and while using a short amount of space for Vertical Take-Off and Landing (VTOL). Much of its systems come directly from the V-22 except for the number of engines.

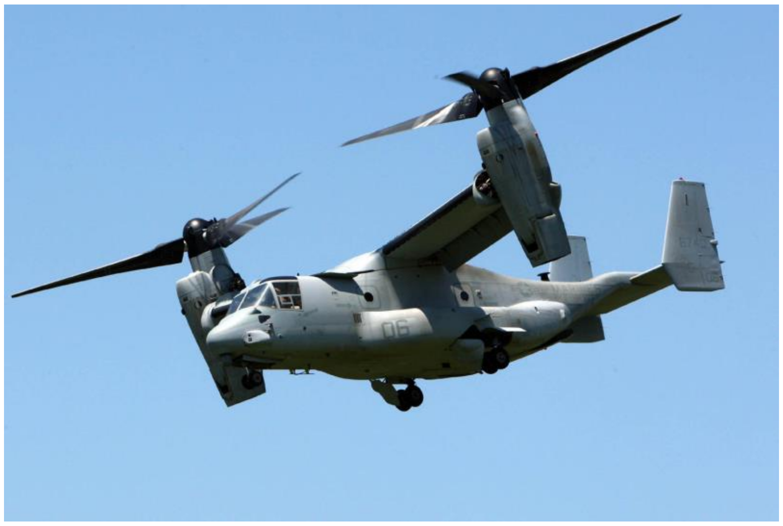


Figure 5: V-22 Ospray

At the same time that military organizations encourage the creation of larger and heavier Quadrotor, the interest in unmanned Quadrotor has been growing in the academic milieu, where these vehicles appear as challenging platforms to develop new solutions for control, estimation, communication problems, and 3D orientation and navigation algorithms. The attention given to these objects has been increasing thanks to miniaturization and the development of cheaper components. With these platforms the algorithms can be tested, improved and can then be implemented on larger objects.

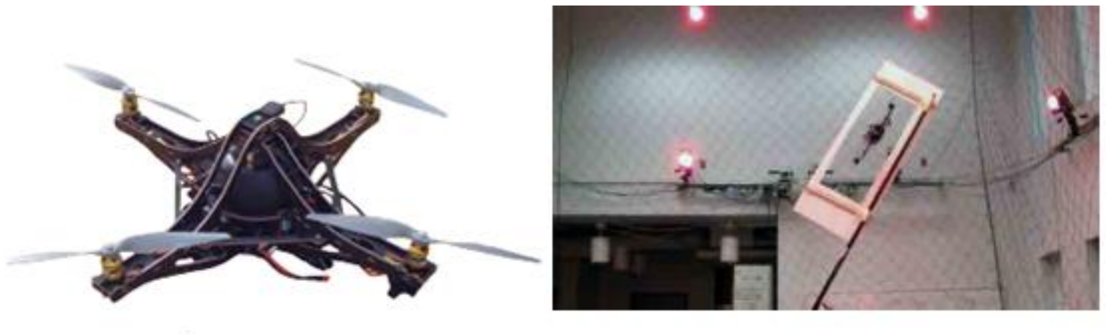
For example one might think of the problem of having a flying object autonomously landing on a moving platform as developed by the Aalborg University. The idea could be applied to helicopters landing on aircraft carriers.[4]

[4]Domingues, J. M. B. (2009). "Quadcopter prototype." Grau de Mestre emEngenharia Mecânica,(2009, Oct).

The private sector has also found applications for Quadrotor. As an example one might think of UAVison®, a Portuguese company which entered a growing market with its ”U4 QuadCopter” shown in Figure 6(a), thinking of Quadrotors not only as military devices but finding some civilian applications. For instance, the company proposes the usage of these vehicles as a broadcasting tool, accessing difficult angles for image acquisition in sport or outdoors events or even as an aerial surveillance tool for law enforcement, among others.

Quadrotors are vehicles capable of aggressive maneuvers allowing them to fit a large set of applications[2]. As an example, Figure 6(b) presents a Quadrotor passing through a window, demonstrating the potential of this vehicle. It becomes clear that the concept of the platform is promising and allows variations.

However it is important to note that each set of sensors, each set of actuators may imply a very different approach, therefore justifying the continuous improvements and research dedicated to this area.[5]



(a) UAVison prototype: U4 QuadCopter. (b) Aggressive Maneuvers.

Figure 6: Quadrotors in the Private Sector and in Universities

[5]Henriques, B. S. M. (2011). Estimation and control of a quadrotor attitude, Master’s thesis, Instituto Superior Técnico.

2.3 Arduino microcontroller

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

2.3.1 Features of Arduino

1- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms.

2- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

3- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well.

4- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers

5- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money

2.3.2 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them. Programs written using Arduino Software (IDE) are called sketches.

The Arduino IDE supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consist of two functions that are compiled and linked with a program stub *main()* into an executable cyclic executive program.

setup(): a function that runs once at the start of a program and that can initialize settings.

loop(): a function called repeatedly until the board powers off.

After compiling and linking with the GNU "toolchain", also included with the IDE distribution, the Arduino IDE employs the program "*avrdude*" to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

2.4 Methods of control

2.4.1 Remote control UAV

A radio transmitter is an electronic device which, when connected to an antenna, produces an electromagnetic signal such as in radio and television broadcasting, two way communications or radar. Heating devices, such as a microwave oven, although of similar design, are not usually called transmitters, in that they use the electromagnetic energy locally rather than transmitting it to another location.

A radio transmitter design has to meet certain requirements. These include the frequency of operation, the type of modulation, the stability and purity of the resulting signal, the efficiency of power use, and the power level required to meet the system design objectives.

2.4.1.1 Modulation used in RC

PWM and PPM are two common words used in the R/C industry. PWM stands for Pulse Width Modulation and PPM stands for Pulse Position Modulation. Some devices that use PWM for control are ESC's (electronic speed controls) and servos. PWM is a technique used to relay data in the form of a varying pulse width.

You may be already familiar with binary, 1's and 0's; where a 1 is represented as 'on' and a 0 as 'off'. An example of this would be a light switch. Turning the switch on would indicate a 1, off a 0. In the case of a PWM/PPM signal, a voltage applied indicates a 1 and vice versa. However, in the case of R/C electronics this 'on/off' data is not enough; this is where the pulse width comes in.

The way we relay data to a servo for instance is the time the pulse is on. In the case of R/C electronics this time is usually around 1-2 milliseconds. A servo or ESC will monitor this pulse and begin counting when the pulse is detected and stop counting when the pulse stops. The time the pulse is on will determine the servo position. For example, sending a servo a 1ms pulse will make the servo swing completely left while a 2ms pulse will swing the arm completely right.

Generally in R/C equipment an entire PWM pulse will last a total of 20ms. The entire pulse is called a frame. A complete frame will include both the time the pulse is high (1-2ms) and the time the pulse is low. Figure (7) represents a typical PWM frame.

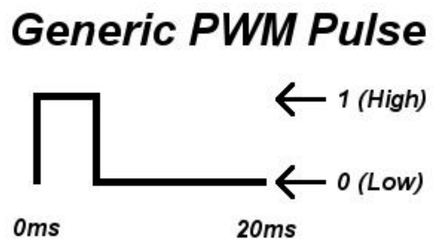


Figure 7: Generic PWM Pulse

Although the frame lasts 20ms the important part of the pulse is the time the pulse is on; 1-2ms. Although the time between pulses is not as important it does play an important role. Usually keeping the time between pulses around 20ms is best. If the delay is longer, a servo for example will lose holding power. A pulse can be generated much faster but 20ms is best for most situations.

This is an R/C specific and will help understand PPM. PPM basically is several PWM signals lined up back to back. A PPM frame looks like this:

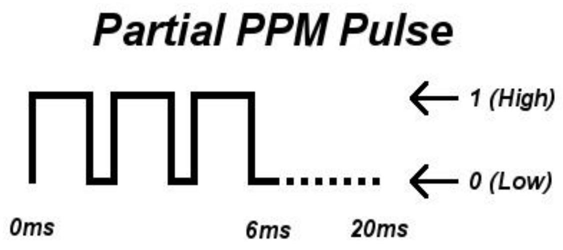


Figure 8: Partial PPW Pulse

Aside from the gaining servo holding power, the reason for the 20ms frame is just having the ability to line up several PWM signals in the same frame. Like I said before, the time the pulse is on is what is important because we are able to strip out this relevant data from a PPM frame to re-generate a PWM frame. For example, if a radio only sent 1 PWM signal at a time, it would take 20ms per channel. If you have an 8 channel radio each update would take 160ms. The same data can be packed into a PPM frame and only take 20ms per update. Transmitters and receivers are the two most common R/C devices that use PPM.

2.4.1.2 Number of channels

Each channel allows one individual thing on the aircraft to be controlled. For example, one channel for throttle, one channel for turning right and left, one channel for pitching forward and backward, one for rolling left and right. Four channels is a minimum for a Quadcopter (pitch, roll, throttle, yaw).

2.4.2 Autonomous UAV

An autonomous robot is a robot that performs behaviors or tasks with a high degree of autonomy, which is particularly desirable in fields such as space exploration, household maintenance (such as cleaning), waste water treatment and delivering goods and services.

Some modern factory robots are "autonomous" within the strict confines of their direct environment. It may not be that every degree of freedom exists in their surrounding environment, but the factory robot's workplace is challenging and can often contain chaotic, unpredicted variables. The exact orientation and position of the next object of work and (in the more advanced factories) even the type of object and the required task must be determined. This can vary unpredictably (at least from the robot's point of view).

One important area of robotics research is to enable the robot to cope with its environment whether this be on land, underwater, in the air, underground, or in space.

A fully autonomous robot can:

1. Gain information about the environment
2. Work for an extended period without human intervention
3. Move either all or part of itself throughout its operating environment without human assistance.
4. An autonomous robot may also learn or gain new knowledge like adjusting for new methods of accomplishing its tasks or adapting to changing surroundings.

Like other machines, autonomous robots still require regular maintenance.

Not far away, drones have involved in this field strongly. UAVs obviously have no pilot on board, but are often flown by pilots on the ground who monitor and command the aircraft through remote control. Although this type of operation may be acceptable for short duration flights where the pilot has adequate information from on-board sensors to make real-time decisions, to relieve the pilot of most aspects of decision-making by automating as many flight and data gathering processes as possible. The advantages of autonomous operation include: reduced personnel requirements and costs, consistent decision-making based on pre-programmed rules, and greater scientific productivity due to flexible, optimized, intelligent flight and payload operations.

2.4.2.1 UAV Autonomy Capabilities

A UAV becomes more autonomous as more and more decision-making functionality is transferred from the human operator to the UAV system. Taking advantage of the unique capabilities of UAVs requires autonomy functionality in both the aircraft and in the payload. For example, even though the aircraft may nominally be flown from a ground station by a pilot, it could be programmed to follow pre-determined waypoints. The payload instruments must also operate without hands-on control, either through remote control or built-in functionality. Introducing autonomy also requires special risk-mitigation strategies. To ensure safety, the aircraft must be programmed to follow some course of action if communication is lost with the pilot on the ground. Beyond these minimal requirements, autonomous operations can improve the productivity of a mission, for example by allowing dynamic replanning of the flight path. Various autonomy capabilities are being developed for UAVs, including: Aircraft health system monitoring, including fuel level, state of communications link, and payload health. On-board payload information processing to reduce data or direct UAV operations .Flying a pre-programmed flight profile, including lat/long, altitude, time on station.

Autonomous requires Goal-directed, tactical flight profile based on real-time on-board sensor information (e.g. payload). Automated, strategic revising plan based on input from scientists or pilot on the ground. Automated coordination of multiple UAV operations, a vision based navigation approach multirotor has even developed making the navigation process more autonomous depending on camera’s sensor.

Finally, Remote controlled systems are less complex than autonomous systems; in case of simplicity the RC is more suitable for applications. in closed environment the use of autonomous systems is favorable considering that the environment is defined then the vehicle will operate properly.

In conclusion, the type of control method applied to the UAV depends on the type of application required.

CHAPTER THREE: MODULING AND PID CONTROLLER DESIGN

3.1 Modeling of Quadcopter

The first step before the control stage is the adequate modeling of the system dynamics**;** It will provide us with a better understanding of the overall system capabilities and limitations. The current chapter will guide us through the equations and techniques used to model our Quadrotor and its motors.

To mathematically write the movement of an aircraft we must employ Newton's second law of motion. As such, the equations of the net force and moment acting on the Quadrotor's body (respectively F***net*** and M***net***) are provided:

$$F_{net} = \frac{d}{dt}(mv)_b + \omega^{-} \times (mv)_b \text{.................................}\mathbf{1}$$

$$M_{net} = \frac{d}{dt}(I\omega^{-})_b + \omega^{-} \times (I\omega^{-})_b \text{.................................}\mathbf{2}$$

Where **I** is the inertia matrix of the Quadrotor, **v** is the vector of linear velocities andω^{-} is the vector of angular velocities. If the equation of Newton's second law is to be as complete as possible, we should add extra terms such as the force of gravity (F***g***) which is too significant to be neglected, thus it is defined by

$$F_g = mS[0\ \ 0\ \ g]^T = mg[-sin\theta\ cos\theta sin\emptyset\ \ cos\theta cos\emptyset]^T{}_b \ \text{...............}\mathbf{3}$$

Where **S** is the rotation matrix

$$S=\begin{bmatrix} \cos\theta\cos\varphi & \cos\theta\sin\varphi & -\sin\theta \\ \sin\varphi\sin\theta\cos\varphi - \cos\emptyset\sin\varphi & \cos\emptyset\ \cos\ \varphi + \sin\emptyset\ \sin\theta\sin\varphi & \sin\emptyset\cos\theta \\ \cos\emptyset\sin\theta\cos\varphi + \sin\emptyset\sin\varphi & \sin\theta\cos\emptyset\sin\varphi - \sin\emptyset\cos\varphi & \cos\theta\cos\emptyset \end{bmatrix}$$

The force of gravity together with the total thrust generated by the propellers (F_P) have therefore to be equal to the sum of forces acting on the Quadcopter:

$F_{net} = F_p + F_g$**................................4**

Combining equations 1, 3 and 4, the vector of linear accelerations acting on the vehicle's body can be written as:

$$\begin{bmatrix} U^{\cdot} \\ V^{\cdot} \\ W^{\cdot} \end{bmatrix} = \begin{bmatrix} 0 & -RQ \\ R & 0 \; P \\ -Q & P \; 0 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_{px} \\ F_{py} \\ F_{pz} \end{bmatrix} + \begin{bmatrix} -sin\theta \\ cos\theta sin\emptyset \\ cos\theta cos\emptyset \end{bmatrix} g \text{......................5}$$

Where [$F_{P\,x} F_{P\,y} F_{P\,z}$] are the vector elements of $\mathbf{F}_P$.

The forces and moments acting on Quadcopter of (×) configurations

$F_{pz} = -(T_1 + T_2 + T_3 + T_4)$.................................6

$M_x = l(-T_1 - T_2 + T_3 + T_4)$ 7

$M_y = (-T_1 + T_2 + T_3 - T_4)$...........................8

$M_z = K_{TM}(-T_1 + T_2 - T_3 + T_4)$....................9

where

l is the distance to the aircrafts COG ,and $K_{\,TM}$ is a constant that relates moment and thrust of a propeller

Assuming the Quadcopter is a rigid body with constant mass and axis aligned with the principal axis of inertia, then the tensor I becomes a diagonal matrix containing only the principal moments of inertia:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \text{.............................10}$$

Combine equation 9 and 10 result:

$$M_{net} = \begin{bmatrix} I_{xx}P^{\cdot} \\ I_{yy}Q^{\cdot} \\ I_{zz}R^{\cdot} \end{bmatrix} + \begin{bmatrix} (I_{zz} - I_{yy})QR \\ (I_{xx} - I_{zz})PR \\ (I_{yy} - I_{xx})PQ \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots 11$$

$$M_{net} = \begin{bmatrix} I_{xx} 0 & 0 \\ 0 \; I_{YY} & 0 \\ 0 \;\; 0 & I_{ZZ} \end{bmatrix} \begin{bmatrix} P^{\cdot} \\ Q^{\cdot} \\ R^{\cdot} \end{bmatrix} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} I_{xx} 0 & 0 \\ 0 \; I_{YY} & 0 \\ 0 \;\; 0 & I_{ZZ} \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \ldots\ldots\ldots 12$$

Then

$$\begin{bmatrix} P^{\cdot} \\ Q^{\cdot} \\ R^{\cdot} \end{bmatrix} = \begin{bmatrix} \frac{M_x}{I_{xx}} \\ \frac{M_y}{I_{yy}} \\ \frac{M_z}{I_{zz}} \end{bmatrix} - \begin{bmatrix} \frac{(I_{zz} - I_{yy})QR}{I_{xx}} \\ \frac{(I_{xx} - I_{zz})PR}{I_y} \\ \frac{(I_{yy} - I_{xx})PQ}{I_{zz}} \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 13$$

Here are the steps to summary the modeling:

1- Calculate the T,M for each motor from equations ($T = k_T \omega^2$, $M = k_m \omega^2$).

2- Calculate the forces and moments applied on the Quadcopter by propeller using equations (6, 7, 8, 9).

3- Compute the linear and angular acceleration using equations (5,13).

4- Output vector $[R^{\cdot} P^{\cdot} Q^{\cdot} U^{\cdot} V^{\cdot} W^{\cdot}]$

.

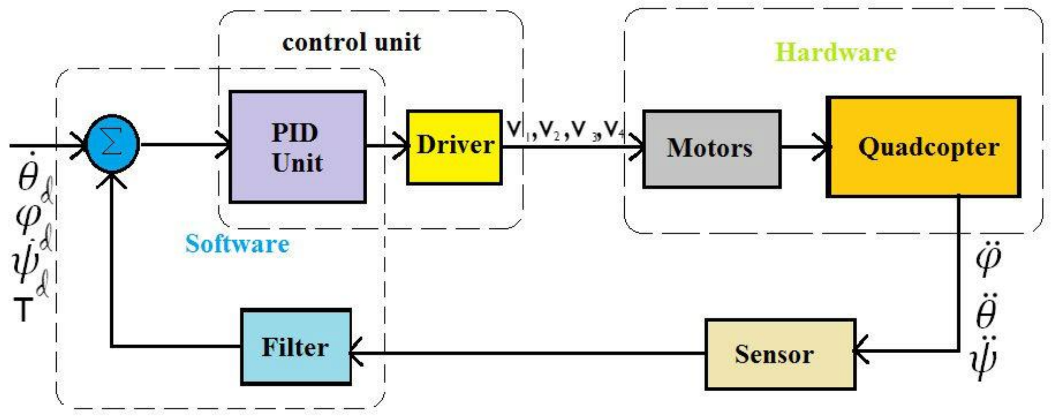


Figure 9: Block diagram of Quadcopter control system

3.2 PID controller

Proportional-integral-derivative controller is a common control feedback mechanism broadly used in industrial control system. A PID controller estimates an "error" value as the difference between a measured process variable and a desired set point.

The controller attempts to minimize the error by altering the process control inputs.

3.2.1 Characteristics of P, I, and D controllers

The present error is dependent on P, past error accumulates on I, and the future error is forecasted by D, based on current rate of change.

Proportional controller (Kp) will have the effect of reducing the rise time and will reduce, but never eliminate, the steady-state error. An integral control (Ki) will have the effect of eliminating the steady-state error, but it may make the transient response worse. A derivative control (Kd) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response. Effects of each of controllers Kp, Kd, and Ki on a closed-loop system are summarized in the table (1).

Table 1: Characteristics of PID gains

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small Change	Decrease	Decrease	Small Change

3.2.2 Manual Tuning of PID

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing P, I, and D based on the dynamic model parameters.

If the system must remain online, one tuning method is to first set Ki and Kd values to zero. Increase the Kp until the output of the loop oscillates, then the Kd should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase Ki until any offset is corrected in sufficient time for the process. However, too much Ki will cause instability. Finally, increase Kd, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much Kd will cause excessive response and overshoot.[6]

There other methods for PID tuning which are not mentioned here.

As it is obvious, tuning a control loop is the correction of its control parameters (proportional gain, integral gain, derivative gain) to the optimum values for the preferred control response. Stability is the basic requirement while designing any controller for the system.

3.2.3 PID Controller in Quadcopter

As any other control system drones need an observer and controller to measure the necessary parameter needed for the control unit to be processed and then decide the proper control action to be performed , but also it's important to give Quadrotor control system special care since it encounters a big error margin relatively during measurement process rises from sensors used furthermore, aerodynamic problems rise from the inconsistent airflow around Quadrotor which makes drone control even more difficult hence a lot of concern should be given for error elimination when selecting sensors and designing observer and controller for such system.

Many projects researchers suggest using specific observer and controller over others. There are many projects that use different strategies in order to smoothly control the Quadrotor attitude and behavior:(pitch, roll, yaw, altitude) while flying ,those strategies vary according to many aspects : need for simplicity , need for precision drone, cost of developed controllers , limitations of hardware used which necessitate employing a specific controller , knowledge exists of control system engineering, etc.

Jun Li et al,[7] in their scientific paper presented the PID controller which aims to regulate the posture (position and orientation) of the 6dof. Quadrotor. The dynamic model is

[6]https://en.wikipedia.org/wiki/PID_controller

[7]J. Li and Y. Li., “Dynamic Analysis and PID Control for a Quad rotor,” *Proc. of IEEE International Conference onMechatronics and Automation*, Beijing, China, pp. 573– 578, August, 2011.

implemented in Matlab/Simulink simulation, and the PID control parameters are obtained according to the simulation results. The simulation results show that the system overshoot is small, at the same time the steady-state error is almost zero, and the system response is fast, which is to say that the performance can be improved by PID controller. So the system simulations verify the effectiveness of the design of the control method. The experiment results show that the Quadrotor can achieve attitude stabilization if the PID parameters are appropriate.

The PID tuning is a complex problem, even though there are only three parameters and in principle is simple to describe, because it must assure complex criteria within the limitation of PID control. The present Quadrotor modes is a highly nonlinear and so parameters that work well at full load conditions don't work initially when the process is starting up from no-load; this can be corrected by gain scheduling. PID controllers generally provide acceptable control using default tunings, but performance can usually be improved by adequate and careful tuning.

Quadrotor controls for many applications have been studied for a long time and considerable progress has been made in the field. Although quad rotors are now able of autonomous flight and aggressive maneuvering, a number of important issues remain open, with no single accepted solution. Research presented in [8] [9] presents a PID controller for attitude stabilization of a Quadrotor UAV.

Having mathematical dynamic model of vehicle is essential for designing a good controller. Nevertheless, UAV model always includes some uncertainties. As the model becomes simpler, the controller becomes more complicated and a model with more details leads to more confident controller. UAV modeling procedure includes determining the dynamic equations of the vehicle body and the structure of uncertain

---

[8]J. Li and Y. Li., "Dynamic Analysis and PID Control for a Quad rotor," *Proc. of IEEE International Conference on Mechatronics and Automation*, Beijing, China, pp. 573– 578, August, 2011.

[9]ZulAzfar and D. Hazry., "Simple Approach on Implementing IMU Sensor Fusion in PID Controller for Stabilizing Quadrotor Flight Control," *IEEE 7thInternational Colloquium on Signal Processing and its Applications*, Penang, Malaysia, March, 2011.

dynamics, specifying the relationship between control inputs and outputs of actuators, and finally dynamics of actuators and sensors must be considered.

The universal and common scheme to design a control system is to analyze and compute the dynamic model of the system. The system model is a set of mathematical equation that includes all the forces that act or perform on the system at the given time. Different control technologies have been compared by various researchers[10] [11].

In majority of cases Quadrotor prove to be a dynamic vehicle with major challenges because of it's under actuated nature. Modeling and control of Quadrotor is currently a common area of research and application, with different levels of model complexity and control design described all through recent literature. The modeling of the vehicle dynamics is classically kept relatively uncomplicated. Control methodologies range from linear proportional-integral-derivative (PID) [12], to linear quadratic (LQ) optimal[13] to a range of adaptive and semi-adaptive schemes[14] [15]. To feedback linearization, back stepping, and dynamic inversion, among a variety of others. The aims and the level of control developed vary, with some sources looking for only to control the orientation of the vehicle, and others seeking to control orientation and position.

[10]Benallegue A., Mokhtari A. and Fridman L., "Feedback Linearization and High order Sliding Mode Observer for a Quadrotor UAV," *Proceedings of IEEE international Work- Shop on variable structure system*, pp. 365 – 372, Alghero, June, 2006.

[11][11] Zhang Z., Cong M., "Controlling Quad rotors Based on Linear Quadratic Regulator," *Applied Science and Technology*, pp. 38-42, 2011.

[12]T. Buchholz, D. Gretarsson, and E. Hendricks, "Construction of a Four rotor Helicopter Control System," *Technical University of Denmark*, 2009.

[13]Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, "Design, Modelling, Estimation and Control for Aerial Grasping and Manipulation," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2668–2673, 2011

[14]Salih A.L., Moghavvemi and Mohammed A.F., "Flight PID Controller Design for a UAV Quad rotor," *Scientific Research and Essays*, pp.3660-3667, 2010.

[15]Yoonsoo Kim, Da-Wei Gu, and Ian Postlethwaite, "Real- Time Optimal Mission Scheduling and Flight Path Selection," *IEEE Transactions on Automatic Control*, Vol. 52, No.6, pp.1119-1122, June, 2007.

3.2.4 Classic PID equations

The equations of classic PID controller are:

1- Proportional controller:

$$A(t) = Kp * e(t)$$

2- integral controller

$$A(t) = Ki * \int_0^t e(t)dt$$

3- Derivative controller

$$A(t) = Kd * \frac{de(t)}{dt}$$

The total equation of classic PID controller is:

$$u(t) = Kp * e(t) + Ki * \int_0^t e(t)dt + Kd * \frac{de(t)}{dt}$$

3.2.5 Transformation of the PID controller equations

The microcontroller can't understand the integration and derivative equation, thus the solution was the conversion of integration and derivative equation into summation and difference equations.

Before transforming the equations, e(t) was substituted as the difference between the Gyro reading and receiver signal as follow:

e(t) = gyro –receiver

Then the transformations of the three equations become:

1- Proportional controller

$$p_{out} = (gyro - receiver) \times K_p$$

2- Integral controller

$$I_{out} = I_{out} + (gyro - receiver) \times K_i$$

3- Derivative controller

$$D_{out} = \left(gyro - receiver - gyro_{prev} - receiver_{prev}\right) \times K_d$$

Then the PID output of the transformed equations is as in (Figure 10):

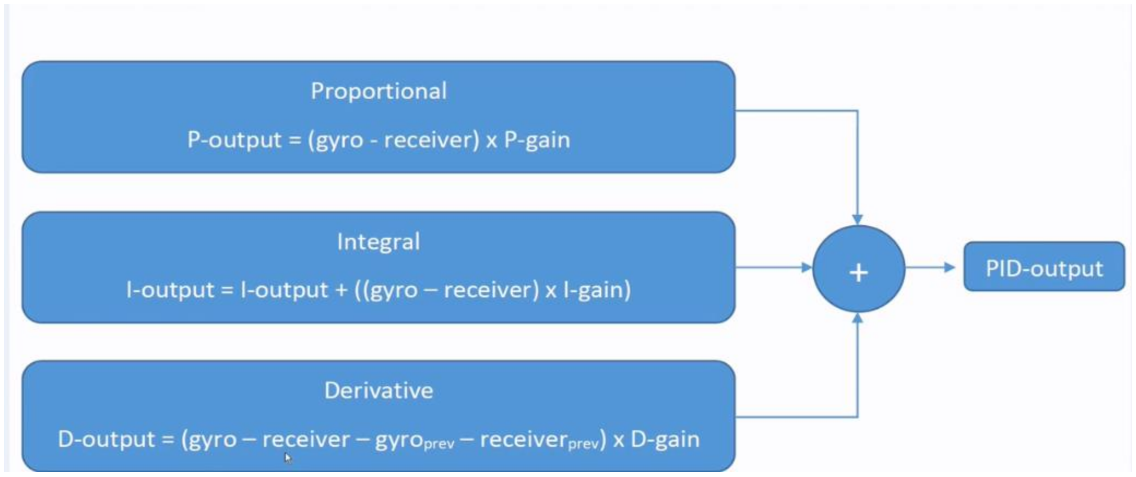

Proportional
P-output = (gyro - receiver) x P-gain
Integral
I-output = I-output + ((gyro – receiver) x I-gain)
Derivative
D-output = (gyro – receiver – gyro_prev – receiver_prev) x D-gain
+
PID-output

Figure 10: PID output of the transformed PID equations

3.2.6 PID output for Quadcopter movements

As mentioned previously, the Quadcopter will have three PID outputs each output is for one movement of the Quadcopter, then the final PID outputs are:

1- The PID output of roll is:

$$PID_{out\_roll} = (gyro - receiver)_{roll} \times K_p + I_{out} + \left(gyro - receiver - gyro_{prev} - receiver_{prev}\right)_{roll} \times K_d$$

2- The PID output of Pitch:

$$PID_{out\_pitch} = (gyro - receiver)_{pitch} \times K_p + I_{out} + \left(gyro - receiver - gyro_{prev} - receiver_{prev}\right)_{pitch} \times K_d$$

3- The PID output of Yaw:

$$PID_{out\_yaw} = (gyro - receiver)_{yaw} \times K_p + I_{out} + \left(gyro - receiver - gyro_{prev} - receiver_{prev}\right)_{yaw} \times K_d$$

CHAPTER FOUR: HARDWARE IMPLEMENTATION

The essential components of the Quadcopter are:

1- Frame
2- Battery
3- Brushless DC motor
4- Propellers
5- Transmitter and receiver
6- Control board
7- Electronic Speed Controller (ESC)

4.1 Frame

The airframe is the mechanical structure of an aircraft that supports all the components, much like a “skeleton” in Human Beings. Designing an airframe from scratch involves important concepts of physics, aerodynamics, materials engineering and manufacturing techniques to achieve certain performance, reliability and cost criteria. The main purpose of this thesis is not airframe design, so, because construction time of the Quadrotor is critical, it is preferable to acquire, if possible, parts already available for sale. The chosen airframe for the Quadrotor was the “F450 PCB Frame” model (Figure 11).

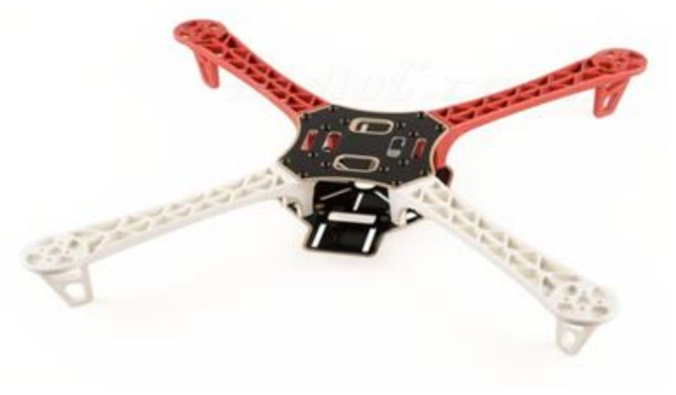


Figure 11: F450 PCB Frame

4.2 Battery

Quadrotor typically uses LiPo batteries which come in a variety of sizes and configuration. 3 S1P batteries were used, which indicate 3 cells in parallel. Each cell is 3.7 volts, so this battery is rated at 11.1 volts. LiPo batteries also have a C rating and a power rating in mAh (which stands for milliamps per hour). The C rating describes the rate at which power can be drawn from the battery, and the power rating describes how much power the battery can supply. A general rule of thumb is that doubling the battery power will get you 50% more flight time, assuming your Quadrotor can lift the additional weight. For this Quadrotor [10], A3S 11.1V 2600MAH 30C LiPo batterywas chosen, shown in (Figure 12).

Figure 12: 3S 11.1V 2600MAH 30C LiPo battery

4.3 Brushless DC motor (BLDC)

Brushless DC motors consist of a permanent magnet rotor with a three-phase stator winding. As the name implies, BLDC motors do not use brushes for commutation; instead, they are electronically commutated. Brushless DC (BLDC) motors are rapidly gaining popularity. They are a bit similar to normal DC motors in the way that coils and magnets are used to drive the shaft. Though the brushless motors do not have a brush on the shaft which takes care of switching the power direction in the coils, and that’s why they are called brushless, instead the brushless motors have three coils on the inner (center) of the motor, which is fixed to the mounting.

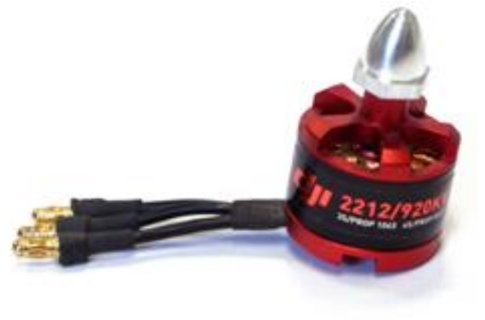


Figure 13: A 2212 1000 KV BLDC (Brushless DC Motor)

On the outer side, it contains a number of magnets mounted to a cylinder that is attached to the rotating shaft. So the coils are fixed which means wires can go directly to them and therefore there is no need for a brush. They offer longer life and less maintenance than conventional brushed DC motors. Some other advantages over brushed DC motors and induction motors are: better speed versus torque characteristics, noiseless operation and higher speed ranges. And in addition, the ratio of torque delivered to the size of the motor is higher, making them useful in applications where space and weight are critical factors.

The speed and torque of the motor depend on the strength of the magnetic field generated by the energized windings of the motor, which depend on the current through them. Therefore adjusting the rotor voltage (and current) will change the motor speed.

4.3.1 Basic concepts when selecting Motors

Motor selection depends on how much weight you are planning to take and the thrust required to lift the Quadrotor, the general rule is that you should be able to provide twice as much thrust than the weight of the Quadrotor. If the thrust provided by the motors are too little, the Quadrotor will not respond well to your control, even has difficulties to take off, But if the thrust is too much, the Quadrotor might become too agile and hard to control.

The BLDC motor chosen was A2212/ 920KV brushless out runner motor shown in Table (2):

Table 2: Specifications of A2212 / 920 KV out runner motor

K V	920 rpm/V
Voltage	DC 7-12V
Type	22*12
Weight	60g
Max power	180 W

4.4 Propellers

Propeller is a set of rotating blades design to convert the power (torque) of the engine in to thrust.

The Quadrotor consists of four propellers coupled to the brushless motor. Among these four propellers, two clockwise and the remaining other two are counter clockwise. Clockwise and anticlockwise propellers cancel their torque from each other.

Propellers are specified by their diameter and pitch. The propeller used is 1045 fixed-pitch, symmetric, tapered Normal Rotation Carbon fiber Propeller, shown in (figure 14):

Figure 14: 1045 fixed-pitch, Carbon fiber Propeller

4.5 Transmitter and Receiver

The RC transmitter used was 8 channels RC, only four channels was needed to provide freedom to control Throttle, yaw, roll and pitch individually.

Figure 15: 8 channels Futaba RC

Figure 16: Receiver

4.5.1 Receiver connection with Arduino UNO

The Receiver connection with Arduino microcontroller is illustrated in (figure 17)

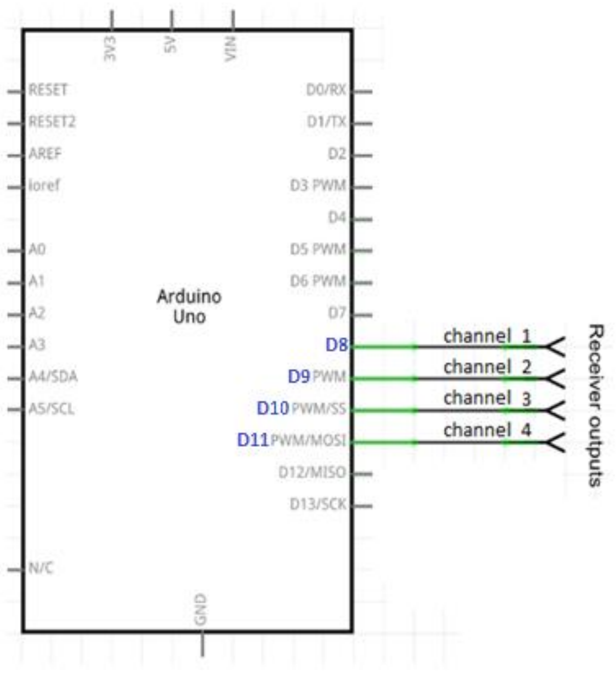


Figure 17: RC interface with Arduino Uno

Four receiver channels are required to receive the signals of the four RC transmitting channels, the 4 channel receiver interface with the Arduino microcontroller is illustrated in (figure 17)..

The first receiver channel is for roll and was connected in Arduino pin number 8, the second receiver channel is for pitch, connected in Arduino pin number 9, the third channel is for throttle, connected in Arduino pin number 10 and the fourth channel for yaw connected in Arduino pin number 11.

4.6 Control board

The flight control board is the ‘brain’ of the Quadrotor. It houses the microcontroller and sensors such as gyroscopes and accelerometers that determine how fast each of the Quadrotor’s motors spin. The microcontroller takes the received signals, and takes the readings from the sensors to generate signal to the ESC to control the brushless motor.

Our control board consists of:

1- Microcontroller (Arduino UNO microcontroller)

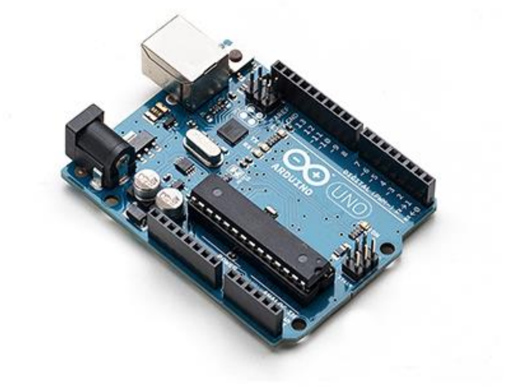


Figure 18: Arduino UNO

2- IMU (Inertial Measurement System) , GY-85 consist of (gyro ITG3205 + accelerometer ADXL345 + magnetometer HMC5883L)
To obtain a feedback of the state of the Quadrotor we use a MEMS technology gyroscope. This type of powerful sensors in such a way of providing accurate reading of quad copter angular velocities with small error, the sensor chip is in fact an inertial measurement unit (IMU) the type used here is GY-85 which contains three sensors, measuring acceleration, orientation and Earth's magnetic field. Values can be gathered using the I2C protocol. The X-axis and the Y-axis are horizontal and the Z-axis is vertical.

The three sensors of GY-85 are:

1. Accelerometer (ADXL345)

   The accelerometer used on the GY-85 is the ADXL345. It measures acceleration for all three axis (x, y, z) and has a resolution up to 13 bit (detects changes less than 1.0°).

2. Magnetometer (HMC5883L)

   Measurement instruments used for two general purposes: to measure the magnetization of a magnetic material like a ferromagnet, or to measure the strength and, in some cases, the direction of the magnetic field at a point in space.HMC5883L is a 3-axis digital magnetometer. The chip is most commonly used as a digital compass to sense the angle from magnetic north (not true north) in degrees.

3. Gyroscope (ITG3200)

   The GY-85 uses InvenSense's ITG3200 to measure orientation. It can sense rotational motion on all three axis and the sensor values are digitalized using a 16 bit ADC. In addition it also has an integrated temperature sensor.

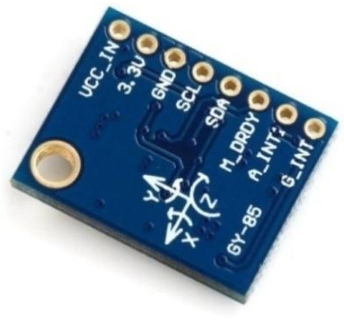


Figure 19: GY-85 IMU (Inertial Measurement Unit

4.7 ITG3200 Gyroscope

Gyroscope (ITG3200) was the only sensor used in the IMU to measure the rate of acceleration of the Quadcopter.

4.7.1 Features of ITG3200 Gyroscope

The ITG-3200 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I^2C (400 kHz) interface. Additional features include an embedded temperature sensor and a 2% accurate internal oscillator. This breakthrough in gyroscope technology provides a dramatic 67% package size reduction, delivers a 50% power reduction, and has inherent cost advantages compared to competing multi-chip gyro solutions.

4.7.2 Gyro connection with Arduino UNO

the gyro is connected to arduino with four pins as shown in figure (20)

3.3v   : 3.3v

GND: GND

SDA: A4

SCL: A5

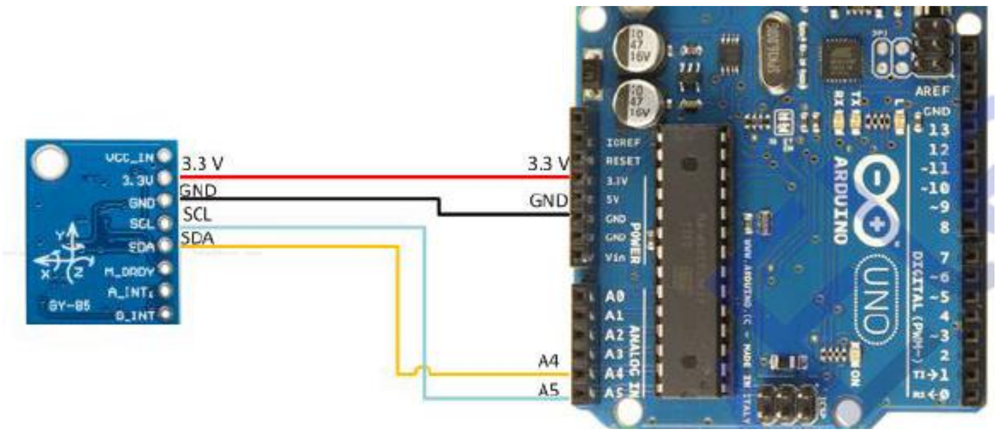


Figure 20: GY-85 connected to I^2 C port of Arduino

4.8 Electronic Speed Controlled (ESC)

The speed of a brushless motor is controlled by an Electronic Speed Controllers (or ESC). This hardware receives the power from the battery and drives it to the motor according to a PWM (Pulse Width Modulation) signal that is provided by the controller unit.

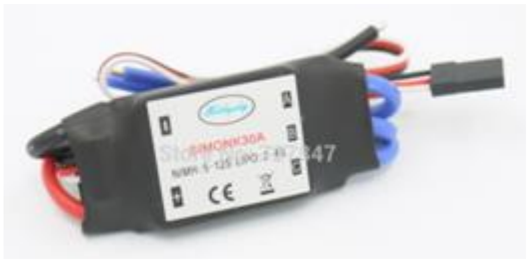


Figure 21: 30A Brushless ESC

Table 3: Specification for 30A Brushless ESC

Cont Current:	30A
Burst Current	35A
BEC Model	Linear mode
BEC Output	5V3A
Weight	25g
Size	32*24*7mm

4.8.1 ESCs connection with Arduino

ESC 1 for motor 1 is connected to Digital pin 4 of the Arduino Uno, consequently ESC 2 for motor 2 connected in Arduino Digital pin 5, ESC 3 in Digital pin 6 and ESC 4 in Digital port 7 as in the figure (22):

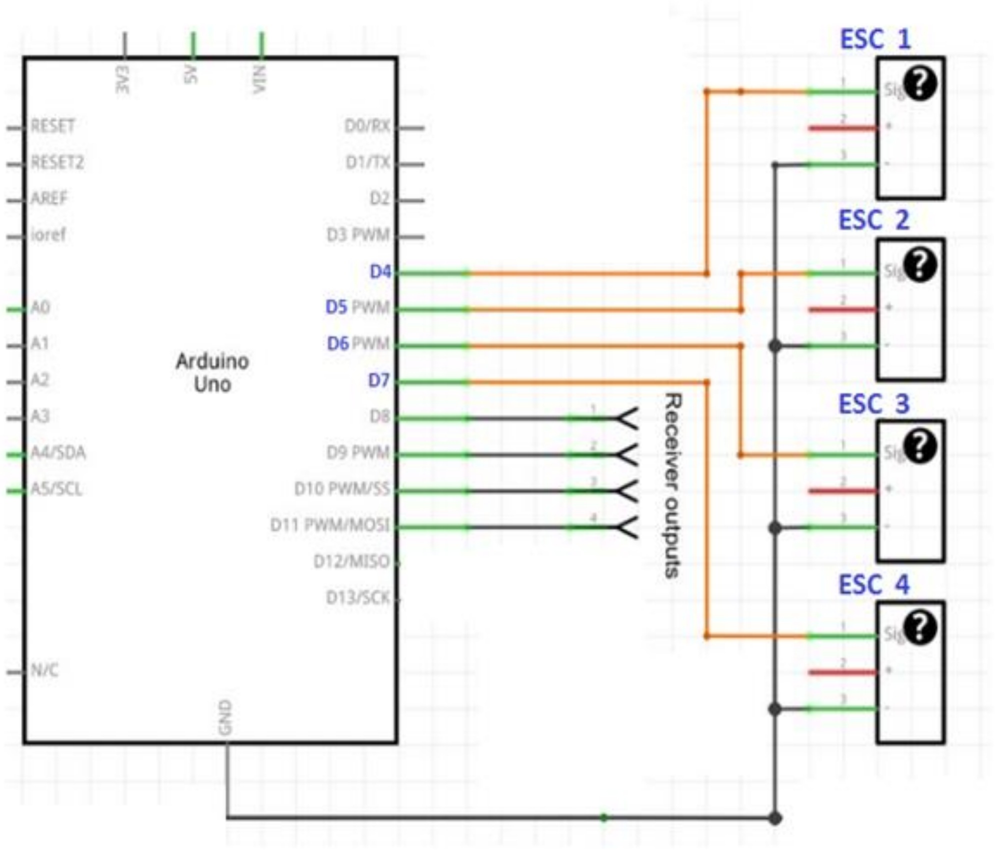


Figure 22: ESCs interface with Arduino Uno

4.9 Bluetooth module connection with Arduino UNO

Bluetooth module connection is as shown in figure (23)

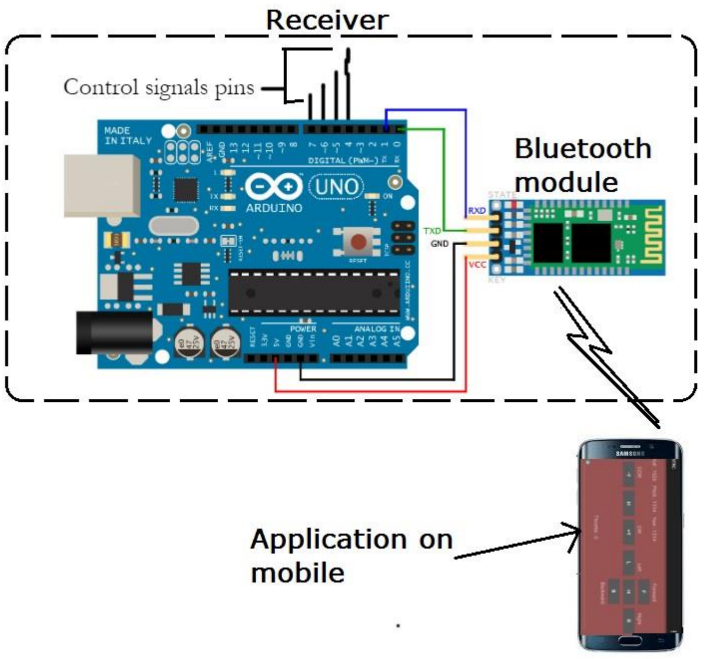


Figure 23: Bluetooth communication

VCC:	5V
GND:	GND
TXD:	PIN 0
RXD:	PI

CHAPTER FIVE: QUADCOPTER SOFTWARE

This chapter will illustrate the code of Quadcopter and the interface between components, the flow chart shown in figure (24) describes the sequence of the operation of code for the Quadcopter.

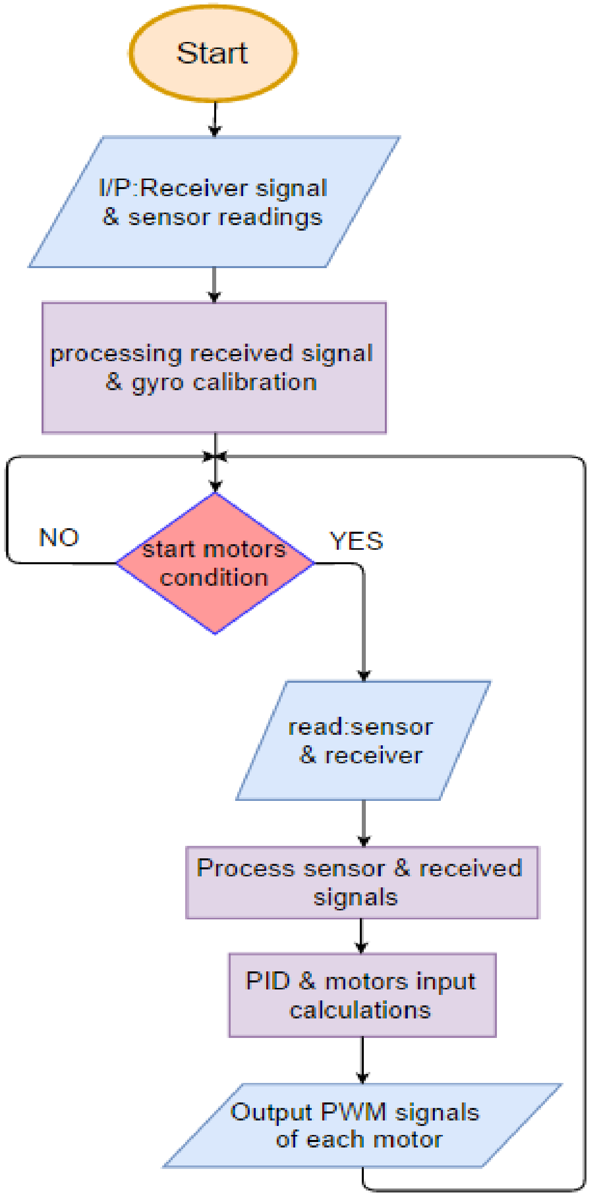

Start
I/P:Receiver signal & sensor readings
processing received signal & gyro calibration
start motors condition
NO
YES
read:sensor & receiver
Process sensor & received signals
PID & motors input calculations
Output PWM signals of each motor

Figure 24: Control system's flow chart

5.1 The transmitter and receiver

5.1.1 Transmitter and signal transmission

Figure (24) will provide guidance and the sequence of this chapter, starting with The RC transmitter used, which has four channels; the need to use a four channel RC is to provide freedom to control Throttle, yaw, roll and pitch individually. Before the execution of transmission, the set points must be determined, in addition to minimum and maximum ranges for each of the four channels in order to adjust the accurate response.

The following readings of PWM for each one of the four channels were recorded and according to these readings the code for throttle roll, pitch and yaw was developed, these reading are as shown in (table 4):

Table 4: RC readings

	Roll (PWM)	Pitch (PWM)	Yaw (PWM)
Maximum readings (micro second)	1930-1932	1932-1936	1932-1936
Set point limitations (micro second)	1524-1528	1512-1516	1512-1516
Minimum readings (micro second)	1104-1108	1108-1112	1108-1108

Throttle max pulse width is **1900** while minimum throttle pulse width is **1100**

5.1.2 Receiver interface with Arduino UNO

5.1.2.2 Interrupt routine

The selected Arduino pins for receiver channels need to hold special notation to allow the receiver to receive signals from the RC into the Arduino, before illustrating this notation a description of the main loop program of the Arduino is required.

The main loop is the main body of the program, it takes place in sequence, and hence in order to receive the input signals from the RC, the main loop needs to be interrupted in order to be able to receive the signals from the RC.

Arduino pins on default do not allow interrupt, the pins can be used to interrupt the main loop and receive an input signal only if these pins where declared in the program to be able to interrupt, the Digital pins 8, 9, 10, 11 of Arduino UNO were chosen to represent the interrupt pins for the four receiver channels respectively for simplicity and to be in orderly manner.

Before declaring the interrupt pins, interrupt mode need to be activated, and the following statement was used:

```
PCICR |= (1 << PCIE0);
```

After activation, the pins 8, 9, 10, 11 can be declared as interrupt pins using the code:

```
PCMSK0 |= (1 << PCINT0);

PCMSK0 |= (1 << PCINT1);

PCMSK0 |= (1 << PCINT2);

PCMSK0 |= (1 << PCINT3);
```

5.1.2.3 Arduino processing of the received signals

When the RC sends the pulse through one or more of the RC channels the interrupt routine is activated allowing the corresponding receiver channels to receive the pulse, As the signal is being received from the RC the Arduino program calculates the time elapsed from the beginning of the program and subtracts it from the time the signal of the RC stops; the time difference between the two is considered the input of that receiver channel. This part of the code describes this process:

```
current_time = micros();

//Channel 1=========================================

if (PINB & B00000001){

  if (last_channel_1 == 0){

  last_channel_1 = 1;

 timer_1 = current_time;                    }}

else if (last_channel_1 == 1){

last_channel_1 = 0;

receiver_input_channel_1 = current_time - timer_1;
```

This code is for channel 1 the code for the rest channels is illustrated in the final code at the appendixes.

5.2 Gyroscope (ITG3200)

Following (figure 24), the received signal needs to be compared with the actual readings of the Quadcopter to determine the error. Thus Gyroscope (ITG3200) sensor was used to measure the rate of acceleration of the Quadcopter.

5.2. 1 I^2C Inter- Integrated Circuit

The gyro is connected to I^2C interface (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus invented. It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. Figure (25) shows the schematic architecture of I^2C interface.

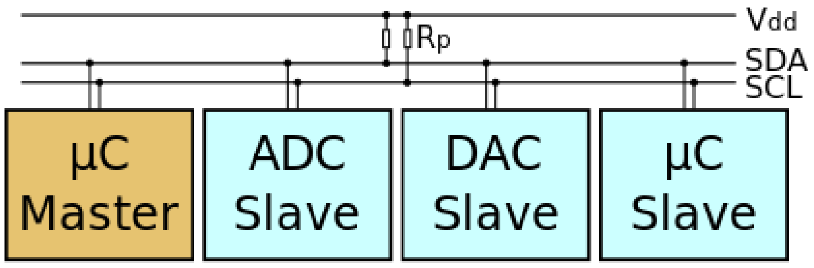


Figure 25: Schematic architecture of I^2C interface

Figure (25) shows A sample schematic with one master, three slave nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors R_p.

Master node is the node that generates the clock and initiates communication with slaves, Slave node is a node that receives the clock and responds when addressed by the master.

I^2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL).

5.2.2 Connecting to sensor procedure

To communicate with the gyroscope sensor Wire library is included to allow Arduino to use the I^2C and make connection to that sensor.

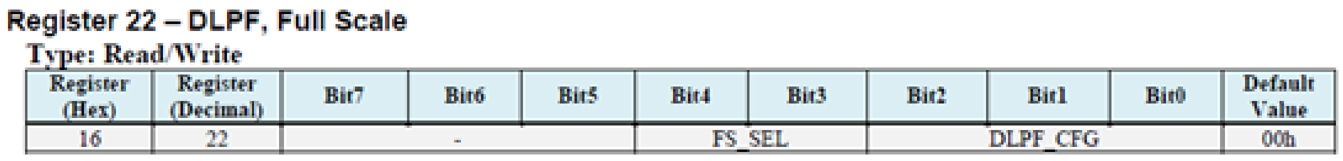
First start a connection to the gyro: master (Arduino) transmit mode by sending a start bit followed by the 7-bit address of the slave (GY-85 which has address of 0x68). so that only the gyro sensor is chosen .hence to start transmission in C language we use the following statement:

```
Wire.beginTransmission(0x68);            //connecting to ITG3200
```

5.2.3 Configuration of gyro parameters

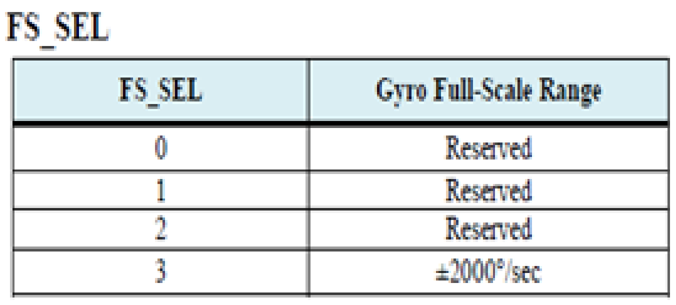
1- Range of output.
2- Sampling rate.
3- The internal Digital Low Pass Filter (DLPF).

Referring to data-sheet of GY-85 set the gyro output scale to ± 2000 deg/s which is not set by default is done by writing the value $(3)_{\text{decimal}}$ to the 3th and 4th bits of (22) gyro. Register



Register 22 – DLPF, Full Scale
Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
16	22		-		FS_SEL			DLPF_CFG		00h



FS_SEL

FS_SEL	Gyro Full-Scale Range
0	Reserved
1	Reserved
2	Reserved
3	±2000°/sec

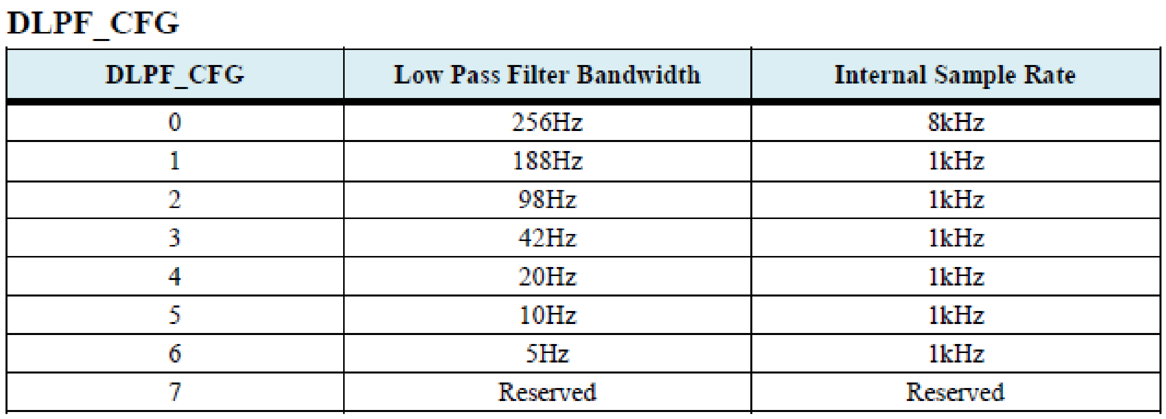
Table 5: Register 22- DLPF, Full Scale

The code for that is:

```
Wire.write(22);              // calling the register of Full Scale
Wire.write(3<<3);          // write 3 then shift it to left  of Full Scale reg.
Wire.endTransmittion();   // necessary to end each call to register
```

Sampling rate which is the rate of output readings of gyro is so important, since we want to have 250 readings per second. Setting gyro rate is done by setting up sampling rate register, referring to gyro. data sheet we find out that there are two registers to configure the sampling rate, the first one is register (22), to be more specific the first three bit Bit0,Bit1 and Bit2. It is responsible for determining the internal sampling rate which has only two sampling rates (1 KHz, 8 KHz).

Table 6: LPF Bandwidth & internal Sample Rate

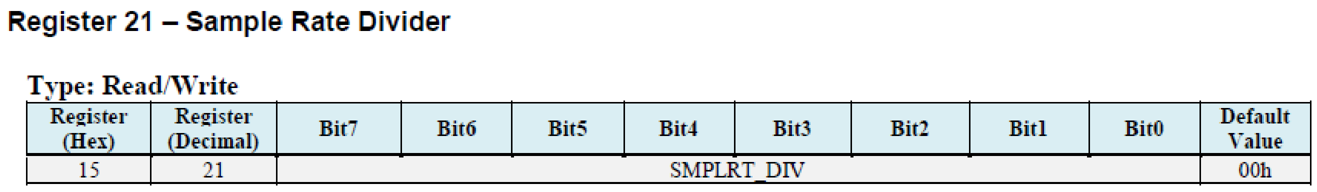


DLPF_CFG

DLPF_CFG	Low Pass Filter Bandwidth	Internal Sample Rate
0	256Hz	8kHz
1	188Hz	1kHz
2	98Hz	1kHz
3	42Hz	1kHz
4	20Hz	1kHz
5	10Hz	1kHz
6	5Hz	1kHz
7	Reserved	Reserved

Then comes the register (21) which also called the sampling rate divider register (SMPLRT-DIV) .

Table 7: Register 21 bits configuration



Register 21 – Sample Rate Divider

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
15	21	SMPLRT_DIV								00h

This divider is set to any value that satisfies the following equation:

$$F_{\text{sample}} = F_{\text{internal}}/(\text{divider} + 1)$$

Where:

F_{sample}:is the sample rate ,F_{internal}:internal rate determined by reg.(22) which is either (1KHz,8KHz) , divider determined by reg.(21)

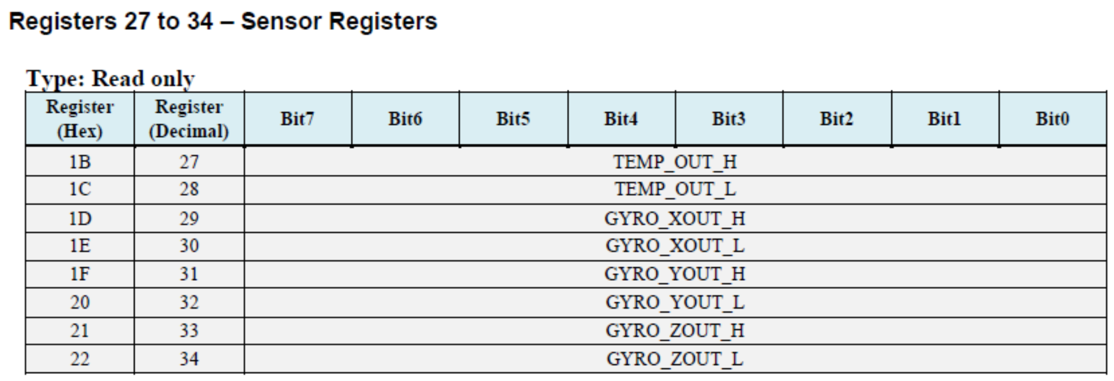
Hence, to get 250 reading out of that gyro, reg.(22) need to be set to zero which is already the default value for that register so no need to manipulate it ,for the divider ,reg.(21) need to be set to $(31)_{\text{Dec}}$, simply be writing that value to that register.

```
Wire.begingTransmittion(0x68);

Wire.write(21);

Wire.write(31);

Wire.endTransmittion();
```

5.2.4 Obtaining readings

After configuring sensor register, the gyro is ready to provide readings through the registers (29-34) ,readings are ready in registers to be picked by the microcontroller any time.

Table 8: Registers 27 to 34



Registers 27 to 34 – Sensor Registers

Type: Read only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	TEMP_OUT_H							
1C	28	TEMP_OUT_L							
1D	29	GYRO_XOUT_H							
1E	30	GYRO_XOUT_L							
1F	31	GYRO_YOUT_H							
20	32	GYRO_YOUT_L							
21	33	GYRO_ZOUT_H							
22	34	GYRO_ZOUT_L							

As you can see in (table 8) that the three angular velocity about axis x, y and z ,each one of them consist of two bytes (high byte, low byte) in two different registers

A dedicated function has been built for picking readings from gyro which can be referred to on appendixes.

Starting by calling the sensor first output register (29), then requesting the6 bytes of sensor o/p registers, those values are being received in array, after that high and low bytes of each axis angular velocity are combined into one variable to end up with three variables containing raw data but not in the form of (deg/s) hence they are being divided by gyro sensitivity $14.375 LSB$ per $(°/sec)$ to give values in deg/s.

5.2.5 Gyro calibration

The readings collected usually have consistent off-set errors which differ in the value from an axis to another. Offset error can be eliminated by calculating the average value for a fair amount of readings which in turn contains the offset error storing this into variables (gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal) these values are subtracted from each reading taken hence eliminating the gyro offset error.

Since calibration is only performed one time, the routine responsible for calibration has been placed in setup loop, refer back to appendixes to see this routine.

5.2.6 Filtering

The calibrated reading taken from gyro are still have noise which can't be easily eliminated by getting rid of offset error, this noise is actually a measurement noise (vibrations caused by motors' propellers) and needs to be processed by necessity since these gyros are so sensitive to the small change of motion. Furthermore, the white noise generated internally even without any external moving force which also needs to be filtered, sometimes sensor also provide completely wrong readings which have to be eliminated.

This Quadcopter system is using a very simple filter which have proven to provide accurate output results, its block diagram is shown in (figure 26).

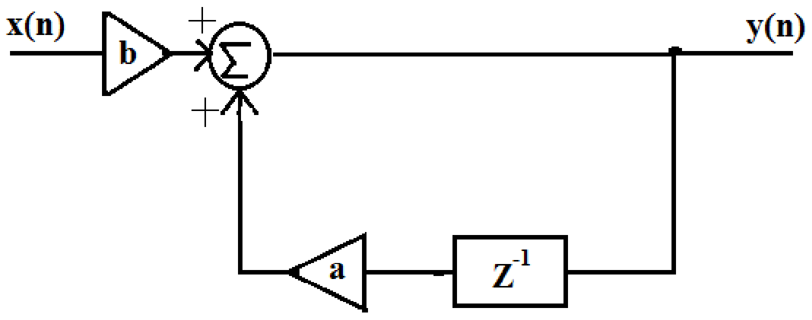


Figure 26: Recursive Filter block diagram

The equation that describes the BD is:

$$y(n) = ay(n-1) + bx(n) where : a = 1 - b$$

As you can see this type of filter take the input x(n) and sum it up with the feedback y(n-1) , a & b are the gains to be tuned to give the required output response .since (a and b) are related to each other by that equation, hence it's easy to tune.

Realization of the filter is straight forward hence the equation is obvious then we can write it down as the following code:

```
gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll ) * 0.2);
```

Where:

gyo_roll: is the filter input value of the roll angular velocity.

gyro_roll_input: the one to the right hand-side of equation is the previously stored value of roll angular velocity, one to the left hand-side of equation is the filter output.

Applying filter to the three inputs of system (roll, pitch, yaw) coming from sensor, we can extend the above code to include them as well. [Appendixes

5.3 ESCs interface with Arduino UNO

Again returning to figure (24) at the beginning of this chapter, the RC signal was transmitted, received and processed by the Arduino then the Gyro sensor provided the angular accelerations the Quadcopter which were filtered, calibrated and processed in the microcontroller.

Before calculating the total received input signal, the output ports of the Arduino Uno connected to the ESCs needs to be declared. This is done using the following code.

```
DDRD |= B11110000;
```

5.3.1 Final outputs of ESCs

The ESCs output depend on the PID output as in chapter 3 , Since the ESCs control the motors, the ESCs output which satisfies the basic movements of Quadcopter mentioned previously are shown in the following code:

esc_1 = throttle -PID_{out_pitch}+PID_{out_roll}- PID_{out_yaw}.

esc_2 = throttle + PID_{out_pitch}+ PID_{out_roll}+ PID_{out_yaw}.

esc_3 = throttle + PID_{out_pitch}- PID_{out_roll}- PID_{out_yaw}.

esc_4 = throttle - PID_{out_pitch}-PID_{out_roll}+PID_{out_yaw}.

The positive and negative signs for the PID outputs in the ESCs equations are set according to the basic movements of the Quadcopter.

The last step, is tuning the PID gains (k_P, k_i, k_d) to provide a smooth and stable response of the Quadcopter.

5.4 Control the Quadcopter by mobile phone

A mobile application has been developed in order to control the Quadcopter instead of remote control, this application was designed to imitate the real RC but it rather sends command signals through Bluetooth channel established between the mobile phone and a receiver module carried by the Quadcopter. The following sections will demonstrate this process.

5.4.1 Program algorithm

The program algorithm used to control the based on RC concept; where Quad-rotor is designed to receive interrupt signals transmitted by RC which are pulse-width-modulated signals through four channels (channel for each controlled variable roll, pitch, yaw, throttle). These signals are received at 4 interrupt pins of the microcontroller "Arduino Uno". But the received signals by BT module is in the form of serial bits containing the command which are not PWM signals, therefore there is a need to process these serial data to regenerate the four pulse width modulated signals and hence can be transmitted to flight controller interrupt pins, it also allows to evade any need for modification or adding new routines to the main program algorithm controlling the Quadcopter. This process is carried out by another μcontroller (could be Arduino Uno as in this case), which receives the BT module signals through serial port and then process this data to regenerate 4 PWM signals (roll, pitch, yaw, throttle) to be delivered by pins 4,5,6,7 of port b to the flight controller interrupt pins.

5.4.2 Application

The application was designed so that it copies the core functions of most Remote Controllers .It sends four command signals for controlling the roll, pitch ,yaw and throttle of the quad rotor ,it is designed to run on Android operating system version 3 (or as it is usually called "Honeycomb") or any other version that follows this one.

Java and XML languages were used to in the process of writing the application's code, Java usually used to provide the underlying functions whereas the XML is basically used to design the graphical user interface of the application.

5.4.2.1 Command format

Commands sent through this application has a special format .In order for these commands to be extracted at receiver device a specific format have to be defined, composed of letters and numbers see figure (27) which elaborate the format used, the necessary information to be sent are the command type (roll, pitch, yaw or throttle) and the degree of roll, pitch. yaw or throttle, simple format is used to represent these information consist of a letter to determine the command type followed by another letter to determine direction separated by a coma mark"," after that comes a decimal number represent the degree of that command .Other marks are also used for starting, ending and separating purposes clarified in the figure (27) (e.g. *R , r | 1750 #) where capital R stands for Roll command and small r stands for right direction and the number 1750 for degree of roll which is in fact the pulse width time represented in microsecond.

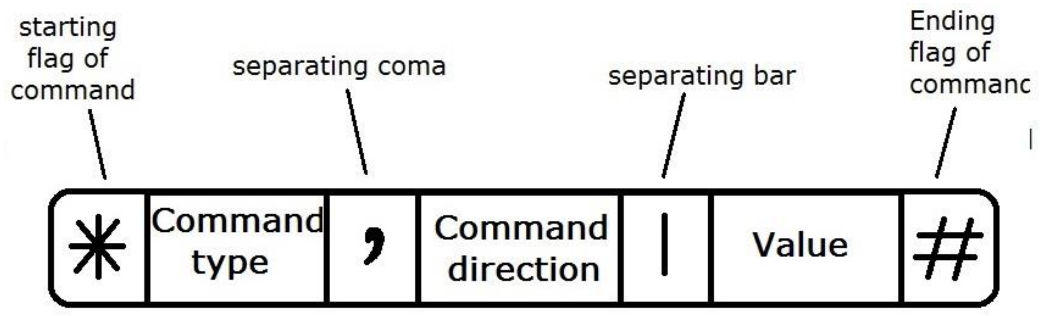


Figure 27: Command format

5.4.3 The operation

The application is enhanced by the operating system (Android) built-in routines and functions to establish a connection with another Bluetooth device (module), firstly by allowing user to search for any nearby Bluetooth device. User then chooses a device from the search result list to pair it's phone with .Once the device has been paired ,it will be added to paired devices list hence there is no need for another search to find that device next time.

Secondly, after device has been paired a connection will be established with that chosen device, the app. is ready to send command signals.

Figure 28 illustrates the block diagram of the control system of the Quadcopter.

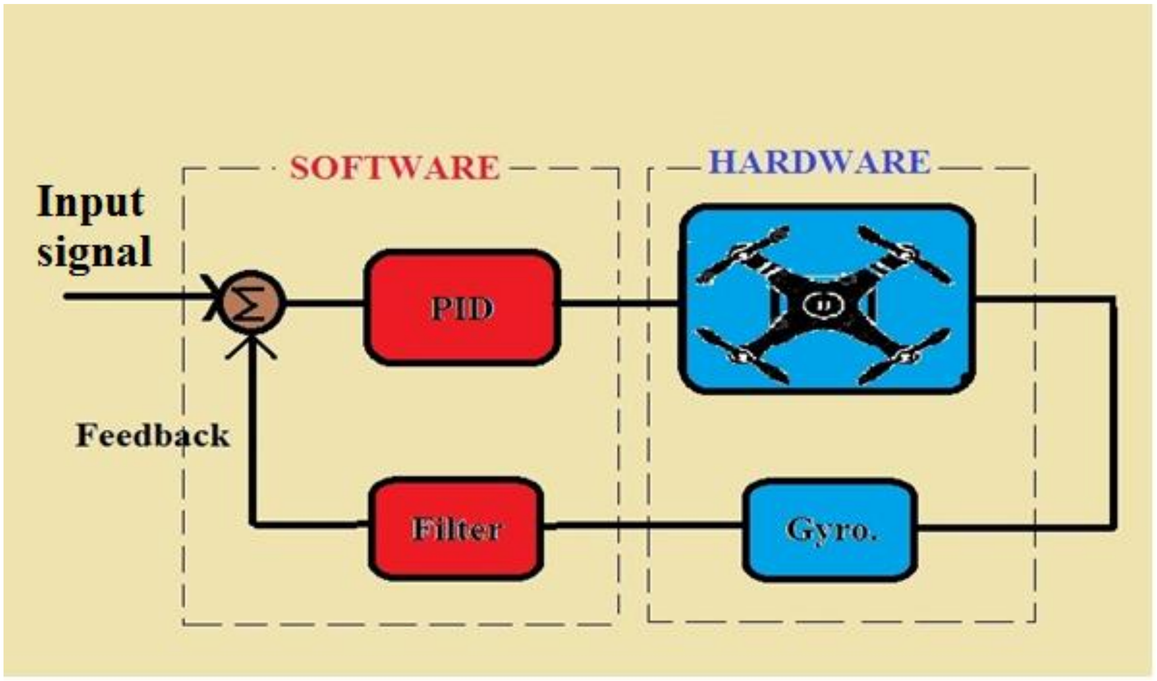


Figure 28: Control system’s Block Diagram

CHAPTER SIX: CONSLUSION AND RECOMMENDATION

6.1 Conclusion

The main objectives of this thesis are to build control system for Quadcopter platform and develop algorithm for control the Quadcopter and implement it in microcontroller platform.

The right components have been chosen then the interface between components has been done, finally the algorithm was developed and the code was implemented in Arduino Uno platform.

After implementation, the angular velocities of brushless DC motors have been controlled by the RC, then the angular rates readings from the gyro were calibrated, obtained, filtered and processed then the PID controller gains were applied and the Quadcopter was able to roll, pitch and yaw.

The flight test has been executed, and the Quadcopter was hovering but it experienced some drifts. This is due to the error caused by MEMS gyro.

6.2 Recommendations

In the project when using the MEMS gyroscope was the only sensor used as feedback to give the angular rates around the 3 axes of space in deg/s. but the problem is that all common MEMS gyroscopes used with Arduino have a drift. It means that even if you stay steady (stationary), the sensor output values will deviate from zero. So in order to achieve better outputs it’s recommended to use gyro and accelerometer together as feedback.

In this project the PID controller was chosen, there are other types of control could be recommend like optimal controller (LQR). A Recursive filter was chosen in this project, for future projects Kalman Filter is recommend to be used.

Finally, when the components of Quadcopter are to be chosen, the weight of Quadcopter is preferred to be as minimum as possible to provide easy control.

6.3 Future work

In this project the control of Quadcopter was made using the RC then mobile phone as the input. For future studies based on this project, Autonomous Quadcopter control will provide a challenge to be met, considering the great advantages that Autonomous flight provides in various fields and that most of the application arising today involves mostly Autonomous flight, Since autonomous control is not controlled by an RC; the Quadcopter needs Advanced control techniques like (vision system, Optimal control, navigations system, mapping.. etc.).

6.4 References

1- Domingues, J. M. B. (2009). "Quadcopter prototype." Grau de Mestre emEngenharia Mecânica,(2009, Oct).

2- Henriques, B. S. M. (2011). Estimation and control of a quadrotor attitude, Master’s thesis, Instituto Superior Técnico.

3- Domingues, J. M. B. (2009). "Quadcopter prototype." Grau de Mestre emEngenharia Mecânica,(2009, Oct).

4- Henriques, B. S. M. (2011). Estimation and control of a quadrotor attitude, Master’s thesis, Instituto Superior Técnico.

5- J. Li and Y. Li., “Dynamic Analysis and PID Control for a Quad rotor,” *Proc. of IEEE International Conference on Mechatronics and Automation*, Beijing, China, pp. 573– 578, August, 2011.

6- ZulAzfar and D. Hazry., “Simple Approach on Implementing IMU Sensor Fusion in PID Controller for Stabilizing Quadrotor Flight Control,” *IEEE 7thInternational Colloquium on Signal Processing and its Applications*, Penang, Malaysia, March, 2011.

7- Benallegue A., Mokhtari A. and Fridman L., “Feedback Linearization and High order Sliding Mode Observer for a Quadrotor UAV,” *Proceedings of IEEE international Work-Shop on variable structure system*, pp. 365 – 372, Alghero, June, 2006.

8- [11] Zhang Z., Cong M., “Controlling Quad rotors Based on Linear Quadratic Regulator,” *Applied Science and Technology*, pp. 38-42, 2011.

9- T. Buchholz, D. Gretarsson, and E. Hendricks, “Construction of a Four rotor Helicopter Control System,” *Technical University of Denmark*, 2009.

10- Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, “Design, Modelling, Estimation and Control for Aerial Grasping and Manipulation,” in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2668–2673, 2011

11- Salih A.L., Moghavvemi and Mohammed A.F., “Flight PID Controller Design for a UAV Quad rotor,” *Scientific Research and Essays*, pp.3660-3667, 2010.

12- Yoonsoo Kim, Da-Wei Gu, and Ian Postlethwaite, “Real- Time Optimal Mission Scheduling and Flight Path Selection,” *IEEE Transactions on Automatic Control*, Vol. 52, No.6, pp.1119-1122, June, 2007.

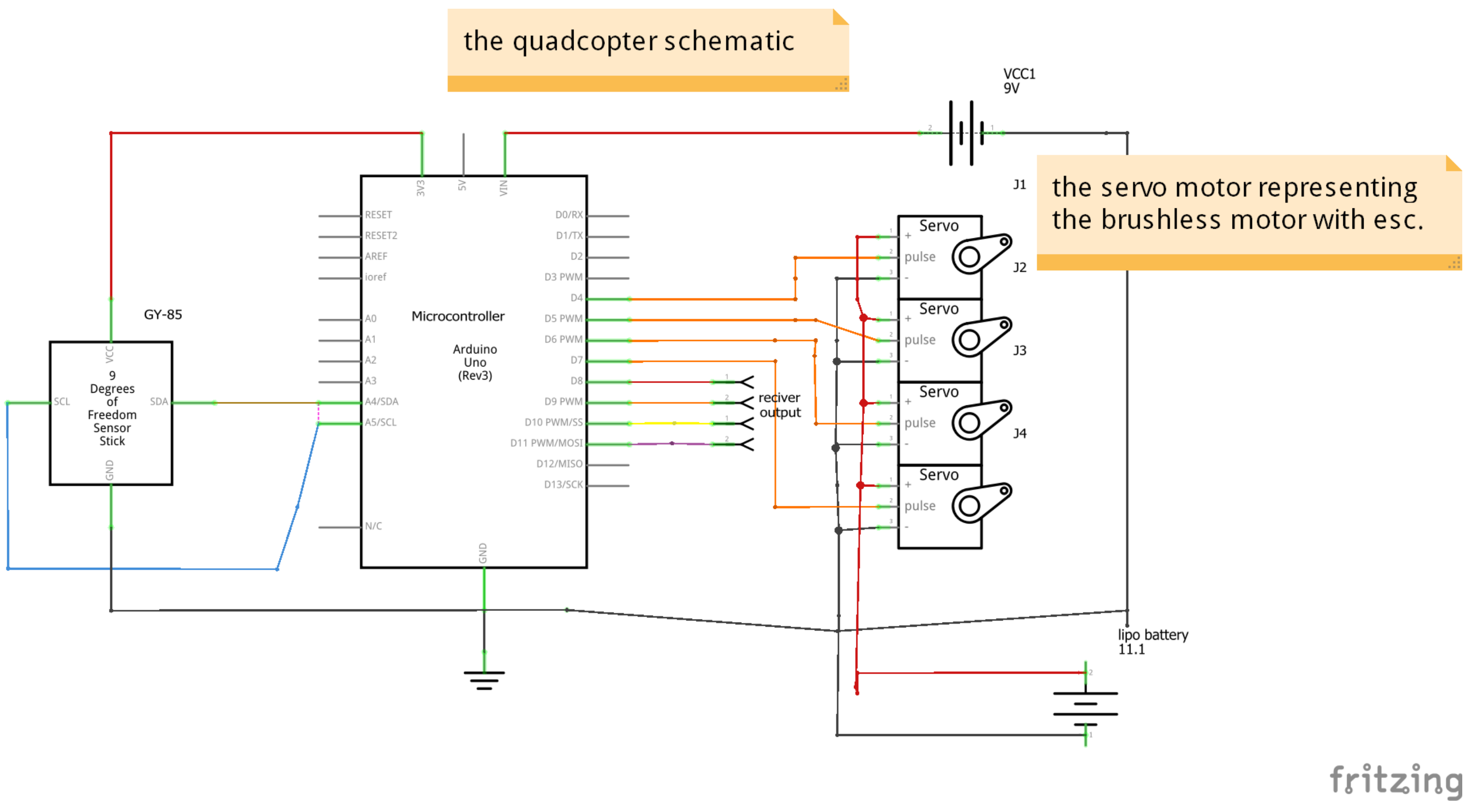
References

1- https://en.wikipedia.org/wiki/Quadcopter

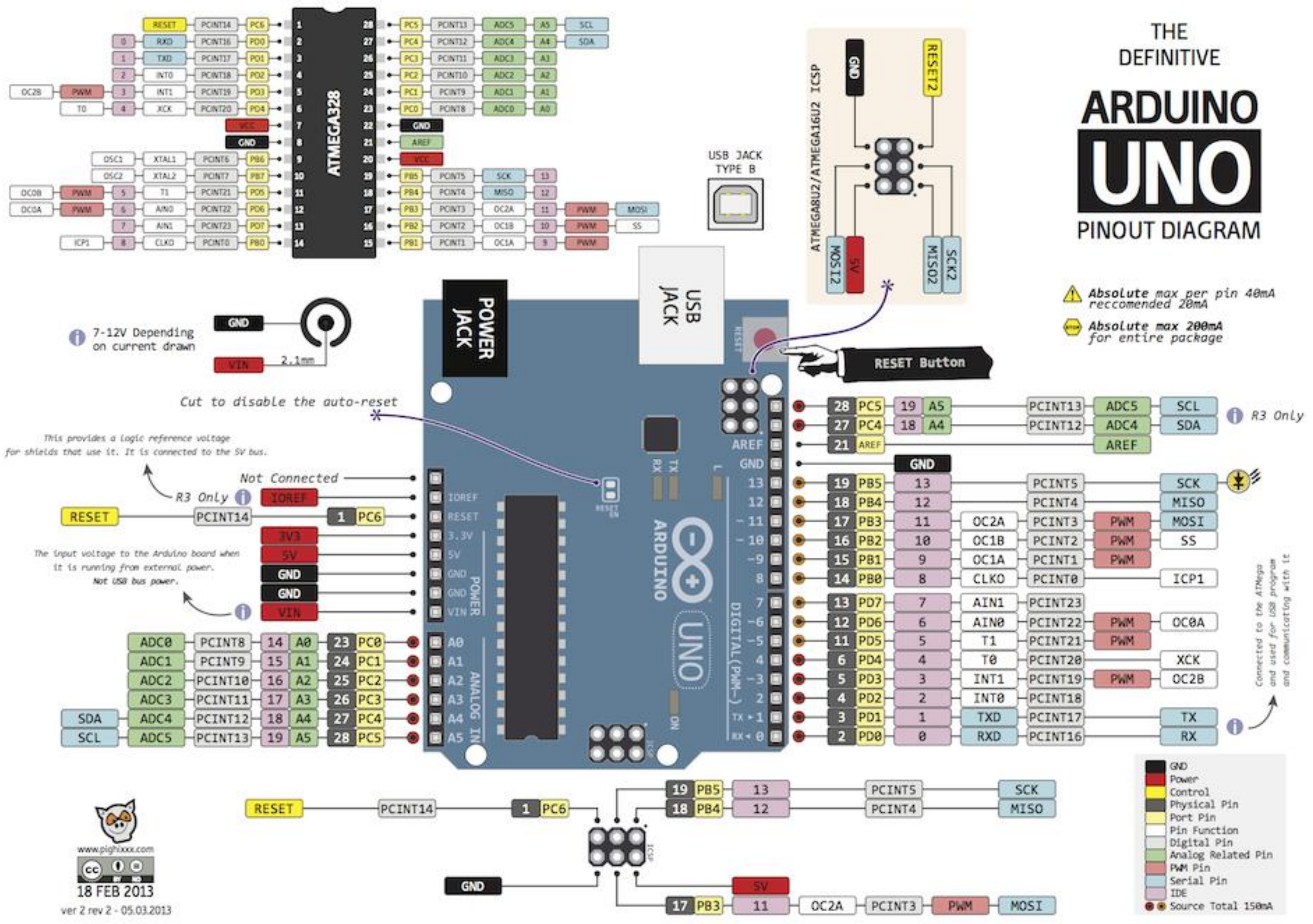
2- https://en.wikipedia.org/wiki/PID_controller

6.5 Appendixes

Appendix A: the Quadcopter schematic



Appendix B: Arduino Pinout diagram


THE DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
POWER JACK
USB JACK
RESET Button
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
7-12V Depending on current drawn
Cut to disable the auto-reset
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
Connected to the ATMega and used for USB program and communicating with it
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013

Appendix C: The code of Quadcopter

```
#include <Wire.h>//Include the Wire.h library so we can communicate with the gyro.

///////////////////////////////////////////////////////////////////////////////////////////////

//PID gain and limit settings

///////////////////////////////////////////////////////////////////////////////////////////////

floatpid_p_gain_roll = 1 ;//Gain setting for the roll P-controller (1.3)

floatpid_i_gain_roll = 0;             //Gain setting for the roll I-controller (0.3)

floatpid_d_gain_roll = 2;           //Gain setting for the roll D-controller (15)

intpid_max_roll = 200;                  //Maximum output of the PID-controller (+/-)


floatpid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.

floatpid_i_gain_pitch  =pid_i_gain_roll; //Gain setting for the pitch I-controller.

floatpid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.

intpid_max_pitch = pid_max_roll;         //Maximum output of the PID-controller (+/-)


floatpid_p_gain_yaw = 3 ;              //Gain setting for the pitch P-controller. //4.0

floatpid_i_gain_yaw = 0.02;             //Gain setting for the pitch I-controller. //0.02

floatpid_d_gain_yaw = 0;              //Gain setting for the pitch D-controller.

intpid_max_yaw = 200;                  //Maximum output of the PID-controller (+/-)


///////////////////////////////////////////////////////////////////////////////////////////////
```

```
//Declaring Variables

///////////////////////////////////////////////////////////////////////////////////////////////////

byte last_channel_1, last_channel_2, last_channel_3, last_channel_4,last_channel_5;

int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;

// int receiver_input_channel_11, receiver_input_channel_22, receiver_input_channel_33, receiver_input_channel_44;

intcounter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;

int esc_1, esc_2, esc_3, esc_4,x;

int throttle, battery_voltage;

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

unsigned long timer_1, timer_2, timer_3, timer_4, current_time/*,timer_5*/;

intcal_int, start;

unsigned longloop_timer;

floatgyro_pitch, gyro_roll, gyro_yaw;

floatgyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;

int buff[6];

floatpid_error_temp;

floatpid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
```

```
floatpid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch,
pid_last_pitch_d_error;

floatpid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw,
pid_last_yaw_d_error;


////////////////////////////////////////////////////////////////////////////////////////////////////

//Setup routine

////////////////////////////////////////////////////////////////////////////////////////////////////

void setup(){

Serial.begin(9600);

Wire.begin();                                   //Start the I2C as master.


  DDRD |= B11110000;//Configure digital port 4, 5, 6 and 7 as output.

  DDRB |= B00110000;                            //Configure digital port 12 and 13 as
output.

 //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as
inputs.


  //Use the led on the Arduino for startup indication

digitalWrite(12,HIGH);  //Turn on the warning led.

delay(3000);    //Wait 2 second before continuing.
```

```
  PCICR |= (1 << PCIE0);                                //Set PCIE0 to enable PCMSK0 scan.
  PCMSK0 |= (1 << PCINT0);        //Set PCINT0 (digital input 8) to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT1);   //Set PCINT1 (digital input 9)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT2);//Set PCINT2 (digital input 10)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT3);          //Set PCINT3 (digital input 11)to trigger an interrupt on state change.
 // Wait until the receiver is active and the throttle is set to the lower position.
while(receiver_input_channel_3 < 1720 ){
start ++;                                          //While waiting increment start with every loop.
delay(5);                                          //Wait 3 milliseconds before the next loop.
if(start == 125){                                      //Every 125 loops (500ms).
digitalWrite(12, !digitalRead(12));                    //Change the led status.
start = 0;                                       //Start again at 0.
   }
  }
  //ITG3200 GYRO=>Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit with a sensitivity of
  //14.375 LSBs per °/sec and a full-scale range of ±2000°/sec
Wire.beginTransmission(0x68);                               //connecting to ITG3200
Wire.write(22);                                  //write to DLPF
Wire.write(3<<3);                                  //setting up the Full scale range parameter

Wire.endTransmission();

delay(250);                                        //Give the gyro time to start.

  ////////////////////////////////////////////////////

  //gyro calibration process

for (cal_int=0 ; cal_int< 2000 ; cal_int ++)

{

//Take 2000 readings for calibration.

if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12));

gyro_signalen();

delay(3);

      }

if(cal_int==2000) {

gyro_roll_cal /= 2000;                             //Divide the roll total by 2000.

gyro_pitch_cal /= 2000;                              //Divide the pitch total by 2000.

gyro_yaw_cal /= 2000;                                //Divide the yaw total by 2000.

  }

  //////////////////////////////////////////////////////

  //ESC calibration process

esccal();                                  //ESC calibration

//////////////////////////////////////////////////////
```

```
start=0;                                   //Set start back to 0.

  //When everything is done, turn off the led.

digitalWrite(12,LOW);                                  //Turn off the warning led.

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

//Main program loop

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

voidloop(){

//Let's get the current gyro data and scale it to degrees per second for the pid calculations.

gyro_signalen();                                               //Read the gyro output.

gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll ) * 0.2);          //Gyro pid input is
deg/sec.

gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch ) * 0.2);         //Gyro pidinput
isdeg/sec.

gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw ) * 0.2);                       //Gyro
pidinput isdeg/sec.



//For starting the motors: throttle low and yaw left (step 1).

if(receiver_input_channel_3 < 1150 && receiver_input_channel_4 < 1150)start = 1;



 //When yaw stick is back in the center position start the motors (step 2).
```

```
if(start == 1 && receiver_input_channel_3 < 1150 && receiver_input_channel_4 >
1450){

start = 2;

   //Reset the pid controllers for a bumpless start.

pid_i_mem_roll = 0;

pid_last_roll_d_error = 0;

pid_i_mem_pitch = 0;

pid_last_pitch_d_error = 0;

pid_i_mem_yaw = 0;

pid_last_yaw_d_error = 0;


  }


  //Stopping the motors: throttle low and yaw right.


if(start == 2 && receiver_input_channel_3 < 1150 && receiver_input_channel_4 >
1750){start = 0;}


  //The PID set point in degrees per second is determined by the roll receiver input.

  //In the case of dividing by 3 the max roll rate is aprox 164 degrees per second ( (500-
8)/3 = 164d/s ).

pid_roll_setpoint = 0;
```

```
  //We need a little dead band of 16us for better results.


if(receiver_input_channel_1 > 1534)pid_roll_setpoint = (receiver_input_channel_1 - 1534)/4.0;

else if(receiver_input_channel_1 < 1518)pid_roll_setpoint = (receiver_input_channel_1 - 1516)/4.0;


  //The PID set point in degrees per second is determined by the pitch receiver input.

  //In the case of dividing by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).

pid_pitch_setpoint = 0;

  //We need a little dead band of 16us for better results.

if(receiver_input_channel_2 > 1524)pid_pitch_setpoint = (receiver_input_channel_2 - 1524)/4.0;

else if(receiver_input_channel_2 < 1506)pid_pitch_setpoint = (receiver_input_channel_2 - 1506)/4.0;


//The PID set point in degrees per second is determined by the yaw receiver input.

  //In the case of dividing by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).

pid_yaw_setpoint = 0;

  //We need a little dead band of 16us for better results.

if(receiver_input_channel_3 > 1130){ //Do not yaw when turning off the motors.
if(receiver_input_channel_4 > 1524)pid_yaw_setpoint = (receiver_input_channel_4 - 1524)/4.0;

else if(receiver_input_channel_4 < 1504)pid_yaw_setpoint = (receiver_input_channel_4 - 1504)/4.0;

  }


//PID inputs are known. So we can calculate the pid output.

calculate_pid();



  //The battery voltage is needed for compensation.

  //A complementary filter is used to reduce noise.



throttle = receiver_input_channel_3;                                   //We need the throttle signal as a base signal.


if (start == 2){                                                    //The motors are started.

if (throttle > 1800) throttle = 1800;                                //We need some room to keep full control at full throttle.

   esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 1 (front-right - CCW)
   esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate
the pulse for esc 2 (rear-right - CW)

   esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the
pulse for esc 3 (rear-left - CCW)

   esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the
pulse for esc 4 (front-left - CW)

if (esc_1 < 1200) esc_1 = 1000;                    //Keep the motors running.

if (esc_2 < 1200) esc_2 = 1000;                    //Keep the motors running.

if (esc_3 < 1200) esc_3 = 1000;                    //Keep the motors running.

if (esc_4 < 1200) esc_4 = 1000;                    //Keep the motors running.


if(esc_1 > 1900)esc_1 = 1900;                              //Limit the esc-1 pulse to
2000us.

if(esc_2 > 1900)esc_2 = 1900;                              //Limit the esc-2 pulse to
2000us.

if(esc_3 > 1900)esc_3 = 1900;                              //Limit the esc-3 pulse to
2000us.

if(esc_4 > 1900)esc_4 = 1900;                              //Limit the esc-4 pulse to
2000us.

  }


else{

   esc_1 = 1000;                                    //If start is not 2 keep a 1000us
pulse for ess-1.
```

```
    esc_2 = 1000;                                           //If start is not 2 keep a 1000us
pulse for ess-2.

    esc_3 = 1000;                                           //If start is not 2 keep a 1000us
pulse for ess-3.

    esc_4 = 1000;                                           //If start is not 2 keep a 1000us
pulse for ess-4.

  }




//All the information for controlling the motor's is available.

  //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.

while(micros() - loop_timer< 4000);                                 //We wait until 4000us are
passed.

loop_timer = micros();                                      //Set the timer for the next loop.


  PORTD |= B11110000;                                        //Set digital outputs 4,5,6 and
7 high.

  timer_channel_1 = esc_1 + loop_timer;     //Calculate the time of the faling edge of the
esc-1 pulse.

  timer_channel_2 = esc_2 + loop_timer;     //Calculate the time of the faling edge of the
esc-2 pulse.

  timer_channel_3 = esc_3 + loop_timer;     //Calculate the time of the faling edge of the
esc-3 pulse.
```

```
  timer_channel_4 = esc_4 + loop_timer;                        //Calculate the time of the
faling edge of the esc-4 pulse.


while(PORTD >= 16){                                          //Stay in this loop until output
4,5,6 and 7 are low.

esc_loop_timer = micros();                                //Read the current time.

if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;               //Set digital
output 4 to low if the time is expired.

if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;               //Set digital
output 5 to low if the time is expired.

if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;               //Set digital
output 6 to low if the time is expired.

if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;               //Set digital
output 7 to low if the time is expired.

  }

}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

//This routine is called every time input 8, 9, 10 or 11 changed state

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

ISR(PCINT0_vect){

current_time = micros();
```

```
  //Channel 1=========================================
if(PINB & B00000001){                                    //Is input 8 high?
if(last_channel_1 == 0){                              //Input 8 changed from 0 to 1
     last_channel_1 = 1;                               //Remember current input state
     timer_1 = current_time;                            //Set timer_1 to current_time
   }
  }
elseIf(last_channel_1 == 1){                             //Input 8 is not high and changed from 1
to 0
   last_channel_1 = 0;                                 //Remember current input state
   receiver_input_channel_1 = current_time - timer_1;        //Channel 1 is current_time -
timer_1
  }
  //Channel 2=========================================
if(PINB & B00000010 ){                                   //Is input 9 high?
if(last_channel_2 == 0){                              //Input 9 changed from 0 to 1
     last_channel_2 = 1;                               //Remember current input state
     timer_2 = current_time;                            //Set timer_2 to current_time
   }
  }
else if(last_channel_2 == 1){                            //Input 9 is not high and changed from 1
to 0
```

```
    last_channel_2 = 0;                                  //Remember current input state

    receiver_input_channel_2 = current_time - timer_2;          //Channel 2 is current_time -
timer_2

  }

  //Channel 3=========================================

if(PINB & B00000100 ){                                    //Is input 10 high?

if(last_channel_3 == 0){                               //Input 10 changed from 0 to 1

      last_channel_3 = 1;                               //Remember current input state

      timer_3 = current_time;                              //Set timer_3 to current_time

    }

  }

else if(last_channel_3 == 1){                             //Input 10 is not high and changed from
1 to 0

    last_channel_3 = 0;                                  //Remember current input state

    receiver_input_channel_3 = current_time - timer_3;          //Channel 3 is current_time -
timer_3


  }

  //Channel 4=========================================

if(PINB & B00001000 ){                                    //Is input 11 high?

if(last_channel_4 == 0){                               //Input 11 changed from 0 to 1

      last_channel_4 = 1;                               //Remember current input state
      timer_4 = current_time;                                   //Set timer_4 to current_time
    }
  }
else if(last_channel_4 == 1){                                  //Input 11 is not high and changed from
1 to 0
    last_channel_4 = 0;                                       //Remember current input state
    receiver_input_channel_4 = current_time - timer_4;         //Channel 4 is current_time -
timer_4
  }
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////
//Subroutine for reading the gyro
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////
voidgyro_signalen(){
Wire.beginTransmission(0x68);
Wire.write(0x1D);
Wire.endTransmission();
Wire.beginTransmission(0x68);
Wire.requestFrom(0x68, 6);                                     //Request 6 bytes from the gyro
uint8_ti=0;
```

```
while(Wire.available() ){
buff[i]=Wire.read();
i++;
   }                              //Wait until the 6 bytes are received
Wire.endTransmission();


gyro_roll  = ((buff[0]<< 8) | buff[1])/14.375;
gyro_pitch = ((buff[2]<< 8) | buff[3])/14.375;
gyro_yaw    = ((buff[4]<< 8) | buff[5])/14.375;                  //Only compensate after the
calibration*/
 //gyro_roll  *=-1;
 //gyro_pitch *=-1;
if (cal_int<2000){
gyro_roll_cal += gyro_roll;                               //Ad roll value to gyro_roll_cal.
gyro_pitch_cal += gyro_pitch;                             //Ad pitch value to gyro_pitch_cal.
gyro_yaw_cal += gyro_yaw;                                 //Ad yaw value to gyro_yaw_cal.
 }
else if (cal_int==2000){
gyro_roll-=gyro_roll_cal;
gyro_pitch-=gyro_pitch_cal;
gyro_yaw-=gyro_yaw_cal;
```

```
    }

}


/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

//Subroutine for calculating pid outputs

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////

voidcalculate_pid(){

  //Roll calculations

pid_error_temp = gyro_roll_input - pid_roll_setpoint;

pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;

if(pid_i_mem_roll>pid_max_roll)pid_i_mem_roll = pid_max_roll;

else if(pid_i_mem_roll<pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;


pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll
* (pid_error_temp - pid_last_roll_d_error);

if(pid_output_roll>pid_max_roll)pid_output_roll = pid_max_roll;

else if(pid_output_roll<pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;


pid_last_roll_d_error = pid_error_temp;
```

```
  //Pitch calculations
pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
if(pid_i_mem_pitch>pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
else if(pid_i_mem_pitch<pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

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

pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
if(pid_i_mem_yaw>pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
else if(pid_i_mem_yaw<pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_last_yaw_d_error);
if(pid_output_yaw>pid_max_yaw)pid_output_yaw = pid_max_yaw;
```

```
else if(pid_output_yaw<pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

pid_last_yaw_d_error = pid_error_temp;

}

  /******esc calibration routine********/
voidesccal() {

while(receiver_input_channel_1<1600)
  {
while(micros() - loop_timer< 4000);                                 //We wait until 4000us are
passed.
loop_timer = micros();                                   //Set the timer for the next loop.

  PORTD |= B11110000;                                      //Set digital outputs 4,5,6 and
7 high.
  timer_channel_1          =          receiver_input_channel_3          +          loop_timer;
//Calculate the time of the faling edge of the esc-1
-  while(PORTD >= 16){                                   //Stay in this loop until output
4,5,6 and 7 are low.
esc_loop_timer= micros();                                //Read the current time.
if(timer_channel_1 <= esc_loop_timer)

PORTD &= B00001111; //Set digital output 4 to low if the time is expired.

                    }

                }

         }

  ////////////////////////////////////////the end//////////////////////////////
```