DYNAMIC QUADROTOR TRAFFIC OFFENSE MONITORING SYSTEM.

By

ADRIAN Z RUWODO R156897Q

Submitted in partial fulfilment of the Requirements for the degree of

BACHELOR OF SCIENCE (HONS)

TELECOMMUNICATION SYSTEMS

Department of Applied Physics & Telecommunications in the

Faculty of Science and Technology at the

Midlands State University

Gweru

June, 2018

DECLARATION

I, ADRIAN Z RUWODO (R156897Q) hereby declare that I am the sole author of this thesis. I authorise the Midlands State University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature:………………………. Date:………/…………/..……….

APPROVAL

This dissertation entitled “**Dynamic quadrotor traffic offense monitoring system**” by **Adrian Z Ruwodo** meets the regulations governing the award of the degree the **BSC in Applied Physics and Telecommunications Hounors** of the **Midlands State University**, and is approved for its contribution to knowledge and literal presentation.

SUPERVISOR………………………………..

DEDICATION

This project is dedicated to the agricultural, national security and mining sectors to which the technology can become a cornerstone of at large and my telecommunications lecturers that made it possible for me to ensure the completion of these educational endeavours by imparting knowledge to me by any means possible.

ABSTRACT

The unavailability of strong technologies in the police force as far as traffic offenses are concerned has led to many law violators going unpunished, and the root cause of this is the limitation that all traffic road blocks around the country are totally dependent on human effort or power. The need for cheaper, smarter and efficient traffic monitoring system facilities in the police force has brought to introduction of small unmanned copters that can execute human tasks with less resources being used. The document will outline a quadcopter based surveillance and supplies' transport system using Proportional, Integral and derivate(PID), Radio frequency technologies (RF) with Kalman filtering algorithms to control a quadcopter and Human Machine Interface(HMI) using processing 2.2.1 to show video surveillance and aid in controls. An Ardu-pilot (microprocessor) was implemented for the flight system.

ACKNOWLEDGEMENT

My deepest gratitude goes to all those who made it possible for me to complete the project on which this report is based. To my supervisor Mr. C.Mudzingwa, I am truly grateful for his continuous technical guidance, idea inculcation and support with regards to this project and for all the knowledge I acquired during this project. I owe my gratitude to my mum for believing in me and for all the financial assistance she inputted for this project to materialize. I would like to extend my gratitude to my fiancé or all the support and encouragement. Finally, we would like to extend our gratitude to all my friends and family members for the great motivation l received from them during the course of this project.

Thanks to All!

TABLE OF CONTENTS

List of Figures

CHAPTER 1 - INTRODUCTION

1.1 Background of the study

Research and development of unmanned aerial vehicle (UAV) and micro aerial vehicle
(MAV) is increasing nowadays, since the application of UAV and MAV apply to a wide range of areas such as search and rescue missions, military surveillance, film making, agriculture and others. In U.S. Coast Guard maritime search and rescue missions, teams use UAVs that are attached with infrared cameras to assist the mission to search the target [1]. Quadcopters or quad-rotor aircrafts are one of the UAVs that are major focuses of active researches in recent years. Compare to terrestrial mobile robot that often possible to limit the model to kinematics, Quadcopters required dynamics in order to account for gravity effect and aerodynamic forces [2]. Quadcopters are operated by thrust that is produced by four motors that are attached to its body. It has four input forces and six output states (x, y,z, θ, ψ, ω) and it is an under-actuated system, such it enables the Quadcopter to carry more load [3]. A Quadcopter has advantages over the conventional helicopter were the mechanical design is simpler besides that, Quadcopters change direction by
Manipulating the individual propeller's speed and do not require cyclic and collective
pitch control [4].

A quadcopter is an aerial vehicle that uses four rotors for lift, steering, and stabilization. Unlike other aerial vehicles, the quadcopter can achieve vertical flight in a more stable condition. The main advantages of the quadrotor over the helicopter is that it is not affected by the torque issues that a helicopter experiences due to the main rotor such investment in these small planes is invaluable in modern day research. Furthermore, due to the quadcopter's cyclic design, it is easier to construct and maintain [5].
Various groups such as the military, engineers, researchers, and hobbyists have been developing quadcopters to understand different technical areas. For example, quadcopters can be used for reconnaissance and collecting data. This could range from searching for survival victims in a disaster area to checking the state of electrical power lines. Some quadcopters in production today can hold light payloads, such as food and medical supplies, and deliver them to areas where normal planes cannot reach [6].

Figure 1: Generic Quadrotor

1.2 Problem Definition

“A total of 269 vehicles have been impounded since 1st October 2017 for moving on the country’s roads whilst the owners has not effected change of ownership as required in terms of the Vehicle Registration and Licensing Act (Chapter 13: 14). The ZRP as a law enforcement agency has noted that, there has been an increase in flagrant disregard of traffic laws by motorists and the commission of criminal acts by elements using these unregistered vehicles or those without registration number plates. 67 vehicles were impounded for operating as Public Service Vehicle yet they were not registered in terms of the Road Motor Transportation Act (Chapter 13: 10).” Nyathi said [7].

First hand detail from the Bulawayo Police sites that, they have impounded more than 100 vehicles without registration number plates amid reports that unregistered cars are linked to serious crimes in the city [8].

October 18, 2011 – There are several cars in Harare that do not have registration numbers while most are suspected to have been used in criminal activities, making it difficult for police to trace them. Police impounded 118 unregistered vehicles. This has resulted in members of public losing valuables to robbers using vehicles without number plates after being offered lifts. Armed robbers have been using some of these vehicles as getaway cars after robbing members [9].

1.3 Proposed Solution

Design and Implementation of a quadrotor traffic system which is programmed to remotely capture real time vehicle images. The videos and images will be transmitted to a base/ control station (CS) located on the ground for image processing.

1.4 Aim

To design an aerial system that remotely monitors the activities of traffic in the CBD where the ground personnel are not deployed. To design a UAV that captures and relay information to a main control station.

1.5 Project objectives

1. Design a remotely controlled fly system with detachable payload.
2. Create camera payload interface for surveillance i.e. images and videos.
3. Image processing system that matches vehicles number plates with existing database for identification.

1.6 Project scope/ constrains

The scope includes: weather conditions, distance and space:

(a) Quadcopter can only operate in sunny day or dry condition.
(b) This Quadcopter design operates within a distance not more than 100m line of sight from the wireless receiver.
(c) Quadcopter is controlled by Arduino base microcontroller.
(d) Quadcopter will be driven by brushless motor control via the electronic speed controllers.

REFERENCES

[1] Allison Ryan and J. Karl Hedrick (2005). “A mode-switching path planner for

UAV- assisted search and rescue.” *44th IEEE Conference on Decision and Control, and the European Control Conference 2005*.

[2] Atheer L. Salih, M. Moghavvemil, Haider A. F. Mohamed and Khalaf Sallom Gaeid (2010). “Flight PID controller design for a UAV Quadcopter.” *Scientific Research and Essays Vol. 5(23), pp. 3660-3667, 2010.*

[3] A. Zul Azfar and D. Hazry (2011). “Simple GUI Design for Monitoring of a Remotely Operated Quadcopter Unmanned Aerial Vehicle.” *2011 IEEE 7th International Colloquium on Signal Processing and its Applications.*

[4] Kong Wai Weng (2011). “Quadcopter” *Robot Head To Toe Magazine September 2011 Volume 3, pp. 1-3.*

[5] R. Mahony, P. Pounds, and P. Corke. (2006, December) Modelling and control of a quad-rotor robot. In the Proceedings of the Australasian Conference on Robotics and Automation. Auckland, New Zealand. Accessed: September 2012. [Online]. Available: http://www.araa.asn.au/acra/acra2006/papers/paper 5 26.pdf

[6] V. Ross. (2011, August) In development: Networks of unmanned quadcopters to ferry medicine to isolated areas. Discover Magazine. Accessed: September 2012. [Online]. Available: http://blogs.discovermagazine.com/80beats/2011/08/30/in-development-networks-of-%20unmanned-quadcopters-to-ferry-medicine-to-isolated-areas/#.UTqfsxxgCXg

[7]P.Nyathi, ‘Police impound unregistered vehicles’,2017.[Online]. Available: http://www.dailynews.co.zw/articles/2017/10/06/police-impund-unregistered-vehicles.[Accessed: 19- Dec- 2017].

[8] N. Tshili, ‘Police impound over 100 unregistered vehicles’, 2016. [Online]. Available: http:/www.chronicle.co.zw/police-impound-over-100-unregistered-vehicles/. [Accessed: 19- Dec- 2017].

[9] J. Sabau, ‘Police impound 118 unregistered vehicles’, 2011. [Online]. Available: http://www.chronicle.co.zw/police-impound-118-unregistered-vehicles/. [Acessed: 19- Dec-2017].

CHAPTER 2 - LITERATURE REVIEW AND THEORETICAL BACKGROUND OF THE PROJECT

2.1 Motivation

Unmanned aerial vehicles are very much applicable in various area as they portray significant characteristics which include maneuverability, small size, low cost of maintenance and manufacturing. They can maneuver very well in tight areas; can be used to carry load depending on the quadcopter's capacity. e. g. medical supplies.

2.2 History of the Quadrotor

The quadrotor protocol architecture comprises of both software and hardware components. Communication between the remote and the quadcopter is about 200m apart, on average, with a power consumption of 100mA. A fundamental consideration was to design a quadrotor that has a battery life that is prolonged, could fly up to 1km, and had the capabilities of taking surveillance for an extended period of time [1].

A quadrotor is a UAV which consists of four electric motors with propellers attached to them. Basically, for any kind of maneuver, there are four upward rotors: for it to fly in its region. Each rotor plays an important role in direction and balance of the quadrotor as well as torque about its center rotation, plus a drag force opposite to the quadrotor's direction of flight. All the motors on the quadrotor are identical. Adjacent propellers are oriented opposite to each other: when a propeller is spinning in the clockwise direction, then the two adjacent propellers will be spinning in an anticlockwise direction. This is to ensure that torque are balanced i.e. if all propellers are spinning at the same speed [2].

In this project, I used a commercial frame building on it with the electronics required. Together with the frame – attached were motors and propellers. These components determined how much space could be available for the electronics as well as the amount of weight the quadrotor would be able to lift without any complications. Another special component was the microcontroller, which was an open source Arduino board on which the software (own software) was to be placed. Sensor boards were also added to the microcontroller, which would be required to achieve flight. On the sensor board, were three different sensors namely the gyroscope, compass, accelerometer, which all work hand-in-hand to maintain stability of flight while in motion or hovering. Finally, a Lipo battery was employed due to its best ratio of weight to power. This particular battery proved sufficiency to completion of the design, assembly, as well as testing of the quadrotor system and experiments have shown that as this quadrotor had plenty of thrust, it needed a larger battery for the mission flight so as improving flight time.

The camera type for surveillance capability for the UAV is a color camera that employs a transistor- transistor-level (TTL) logic signal. The camera is capable displaying a series of images through a serial communication output as well as 30 frames per second (fps) National Television System Committee (NTSC) formatted output. All communication of all sensors and electric hardware in this project is done over a TTL serial connection, including the wireless Bluetooth module used. The main reason for choosing this type of a camera is the ability to integrate the video over the serial connection seamlessly. Additional reasons include the fact that it operated from a 5V power supply, just like the rest of the sensors, and the power consumption was 10W at <100mA. The camera has the ability to capture VGA, QVGA, QQVGA picture formats and also allowing the image to be compressed with various compression degrees. This low for the shrinking of the image file size to under 30kb per image frame which is small enough to allow frame rate of about 2.5 fps while transmitting at 115 200 bps. This frame rate should be sufficient so as to guide navigation and perform surveillance [3]. The Arduino processor board controls the camera. A series of hex commands are sent from the Arduino to the camera: to initialize and then begin a series of image collections. The images are transmitted from the camera serially in hex format to the Arduino and the transmitted to the ground-based computer via the Bluetooth modules for processing.

The Oxford Dictionary defines a drone as a ‘remote-less controlled piloted aircraft or missile.’ Drones were first built after the World War *II* in which unmanned jets, such as the Ryan Fire bee began field operation and from then the number of drones employed in the military sector has increased greatly to such an extent that NEW York Time decided to refer to it as a new paradigm for warfare. The US-military sector was the first to implement the idea of aerial military surveillance as far as during the Civil War, although other countries followed suit in the use of UAVs. Also a flashback reveals that even before the Wright brothers taught the fledgling aviation secrets of controlled flight and also other efforts towards unmanned combat vehicles existed such as balloons, which were used by the Austrian army in an attack on Venice in 1849 as well as the Japanese forces in the Fu-go bombings in 1945. The first individual to patent a remote-control of unmanned vehicles is Nicola Tesla. He described it as ‘tele-automation’ and it developed to be one of the fundamental principles for today’s UAVs and smaller drones such as quadrotors. Its popularity became more vast in part after the presentation by Professor Vijar Kumar at TED and following outputs from the cutting-edge research that was being performed in the laboratories at the University of Pennsylvania and ETH Zurich.

A futuristic perspective shows that drones are expected to enter into civil use in which these vehicles can carry passengers without an onboard human aerial supervisor/ pilot, thus, bringing to an end an almost 90-year old tradition of planes piloted by human beings or quadrotors patrolling city streets being introduced, marking a very big step forward. Considerably in police forces and fire services, the small size and portability of the quadrotor (small drones/ unmanned aerial vehicles) has become an appealing feature; to study their adoption might be feasible for their own aerial surveillance purposes.

2.3 How much impact traffic offenses has on Community and Police force

According to Allafrica.com, the Zimbabwe Broadcasting Corporation (ZBC) employed new strategies in tightening bolts on motorists concerning radio licenses to the Zimbabwe National Road Administration (ZINARA). This move, was put to effect due to the increase of motorists evading radio license payments. Another incident reported in 2017 was of 67 vehicles that were impounded by police for operating as Public service vehicles yet not being registered in terms of the Road Motor Transportation Act (Chapter 13:10).

On the 3rd of January 2014, according to the Herald (press): Botswana was reported (with immediate effect) to have banned all imported second-hand vehicles from passing through its territory, a development that saw an effect on imports from Europe through Namibia. South Africa initiated this ban stating that cars from Asia that came through Durban Port were supposed to be ferried on vehicle carriers up to Beitbridge. This significantly increased the cost of importing a vehicle into Zimbabwe. Botswana’s Transport and Communication Ministry said this ban was imposed after realization that most unregistered vehicles’ road worthiness was unknown, thus posing a danger to the public. It further pointed out that the imported vehicles in transit were often uninsured and this posed problems in the event of an accident.

Unregistered vehicles were reported to have (up to date) been used in several crimes and there is no way of identifying the culprits or perpetrators. Ferrying a car by carrier through Botswana costs anywhere between US $1 000 and US $1 500 depending on type of vehicle

According to the ZIMSTAT (Zimbabwe National Statistics Agency) publication on the 21st of April 2016: the number of motorists arrested for driving without due care increased from 2 201 in 2010 to 37 419 in 2015 while unlicensed drivers rose to 13 800 the previous year from 819 in 2010. Also the number of vehicles recorded for operating without insurance in 2010 was 2 678 while in 2015 the figure rose to 28 033 cases. The number of motorists operating vehicles without the licensing Act increased from 4 356 in 2010 to 42 615 cases the previous year [4].

2.4 The Main Components Used In Implementation of The Quadrotor

2.4.1 Microcontroller

An Arduino board that has a single-board microcontroller is employed for the implementation of a quadrotor and this single Arduino board comes with simple open source hardware board designed around an 8-bit Atmel AVR microcontroller. Either C or C++ programming language serial communication interface. The IMU (an electronic device which measures and reports on an aircraft's velocity, orientation, and gravitational forces using a combination of accelerometer and gyroscopes, sometimes also magnetometers) is interfaced with the microcontroller.

2.4.2 Comparison and Choosing microprocessor or controller

Determining through comparison which type of microprocessor is ideal for a particular project or device in this case a quadrotor is based on the following.

- System requirements (performing, size, power dissipation)
- Reliability,
- Maintainability,
- Flexibility,

- Environmental constraints,
- Software support,
- Cost, with a manufacturer's track record and history being vital factors to be considered whenever a system is to be implemented using a microprocessor which is the heart of the device.

In general, a microprocessor is a programmable device that accepts digital data as input, processes it with regards to instructions stored in its memory. It is also multipurpose in nature and it provides results as output. It is an example of sequential digital logic, as it has internal memory.

Microprocessors are classified into three classes:-

- Microcontrollers – For control applications and embedded system with low cost and high speed.
- Digital Signal Processors – For control applications (embedded systems) with relatively low cost and flexible I/O configuration.
- General Purpose Processors – General purpose computers PCs and workstations that require high speed.

Based on the architecture of the microprocessor we can categorize the microprocessors into five classes i.e. the 4 – bit, 8 – bit, 16 – bit microprocessors. S1C 60 family is the CMOS 4 – bit single chip microcontroller, an integration of the various kinds of peripheral circuits like RAM, ROM, I/O port, LCD driver, and etc. into a single chip centering on very powerful 4-bit CPU core. Yet, it is characterized by low voltage as well as low power consumption technologies Epson takes pride in it. The S1C88 family is Epson's powerful 8 – bit core; its arrangement integrates a broad choice of ROM as well as RAN size, LCD drivers, serial ports and other high – performance peripheral circuits into a single chip design. In addition, all the devices contain the power – saving, low voltage technology (Epson devices most commonly known for).

Epson's new 16 - bit microcontroller has small size as well as low power consumption equivalent to an 8-bit microcontroller, even with the 16MB address space. The S1C17 family of original 16-bit MCUs integrate a vast variety of interfaces, giving room for connection with multiple sensors and a number of peripheral circuits such as an LCD controller, EPD driver and driver supporting low to medium level resolutions. The S1C17 family has RISC architecture to achieve high-speed operation plus low power consumption. This makes it ideal for mobile devices. The family also includes a broad set-up of built-in Flash ROM products. Shorten design turnaround time is brought about by the high quality development environment as well as on-chip ICE function.

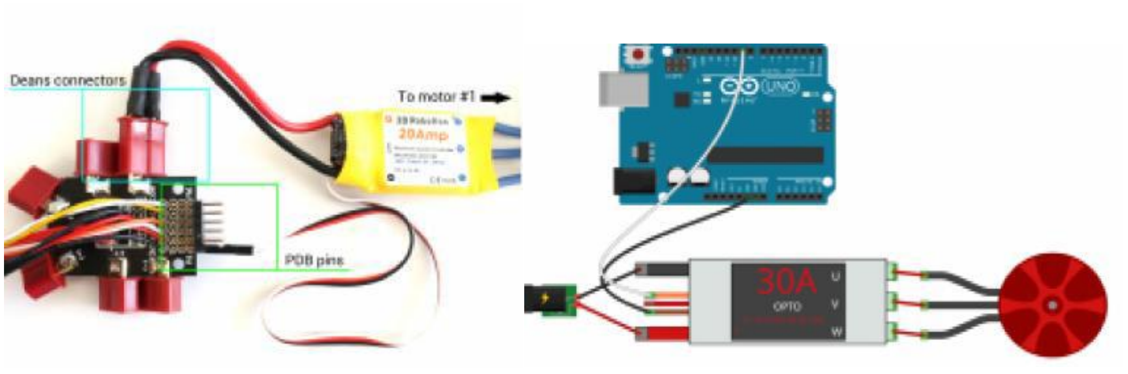
The S1C33 family is Epson's original 32-bit RISC CPU [5]. The product incorporates the peripheral functions; rapid DMA, programmable timer, PLL, multichannel and pre-scaler. Constituting of fast operation and extremely low power consumption, this family matches well to the OA equipment such as printers and the portable equipment, such as PDAs, toys and cellular phones. Furthermore, as this family incorporates an A/D converter and PWM timer, when the middle ware, is combined to it – it can realize the digital signal processing such as voice processing by single chip. They also apply in Telecommunications, high-speed operations in robotics, intelligent control systems, automobiles and image processing.

Where multiple programs, high-speed response and refresh rates are supposed to be run, 64-bit microprocessors are implemented. Also where real-time feedback is needed, it is applicable e.g. in computers, space crafts and Artificial intelligent systems, they work best on 64-bit software.

Most common examples of MCU include, the Intel MCS48, 51 as well as 96 families, whereas the Motorola MC68HC11 family and the Zilog z8. Most of the MSCs comprise of an 8-bit word size (except the MCS-96 with a 16-bit word size), at least 64-bytes of R/W memory and 1KB of ROM. The range of I/O lines ranges between 16 - 40 lines. It is practice that each individual manufacturer has their own unique instruction set and register set hence microprocessors and microcontrollers are incompatible with each other. Intel 8051 is under 8-bit microprocessor family. ROM ranges from NIL to 8KB, RAM size of 128 or 256 bytes (depending on the specific port number) Clock frequency is rated up to 12MHz. UV light erases data and special electrical programmer writes new data. Architecture comprises of four bi-directional I/O ports of 8-bits each [6].

After a thorough consideration between different processors and microcontroller plus relativity, to our high-speed refresh rates and real-time algorithms, I decided to use Arduino UNO microprocessor. This board has a gyroscope as well as magnetometer sensors mounted on it so as to allow connection of a RF Module, GPS module and all the accessories that the board could use, it also supports Bluetooth and ZigBee communication. In this project I did not mount the Ardu- pilot and GPS module.

2.5 ESC (Electronic Speed Controller)



(a) (b)

Figure 2.1: Electronic Speed Controllers

In this thesis, an electronic speed controller (ESC) actuates the quadcopter motors. The control board (which is the brain and main component) controls the ESCs. The purpose of an ESC is to vary the speed of the electric motor, its direction and acts as dynamic brake where required to. It also tells the motor at a particular time at what speed (how fast) it is supposed to spin. Since the quadcopter has for motors, it simply means each motor will have an ESC assigned to it independently.

ESCs are inexpensive and directly linked to the battery input. Some of the ESCs available have a built-in battery eliminator circuit, hence power can be supplied to the RR and FCB without having to connect them directly to the battery. Since the motors need precise rotation speed to achieve accurate flight, the ESCs becomes very vital. Nowadays, a built-in firmware Simonk comes with the ESC, which allows us to change the refresh rate of the ESC so the motors can get more information per second from the ESC. Therefore, it provides more control on the behavior of the quadrotor. [7]

The rotor’s position must be considered initially to electronically communicate a permanent magnet motor. To achieve this sensor-less driving techniques or Hall effect can be used.

Sensored motor operations makes simpler the driving complexity though they will result in heavier and more costly motor as shown below Fig 3.8.

In place of using sensors, sensor-less techniques e.g. back electro-motive force (BEMF) zero-cross detection and field oriented control can be implemented [8][9]. Sensor-less operation is preferred because sensor-less motors have reduced weight, cost, and complexity. As a result this makes sensor-less brushless common in quadcopters. Therefore, a DC to AC 3 phase sensor-less motor driver is required to drive these motors. Generally these are commonly referred to as electronic speed controllers (ESC). ESCs overally represent an integral part of the quadcopter system architecture as their output controls the orientation of the quadcopter by varying the speed of the propellers. A motor turns by reason of the magnetic forces created by the windings and the magnets within the motor. For a brushless motor, the speed of rotation of the motor will depend on the frequency of the winding drive sequence. On a basic brushless motor, there are normally three, pulse width modulated (PWM) signals. To create the necessary magnetic forces needed to turn the rotor, two windings will be driven at a time. Changing the pulse width of the signals adjusts the frequency of the signals. Smaller pulse widths cause an increase in the frequency of a PWM signal because more pulses can be transmitted to the windings in the same time duration, the reverse is true for large pulse widths.

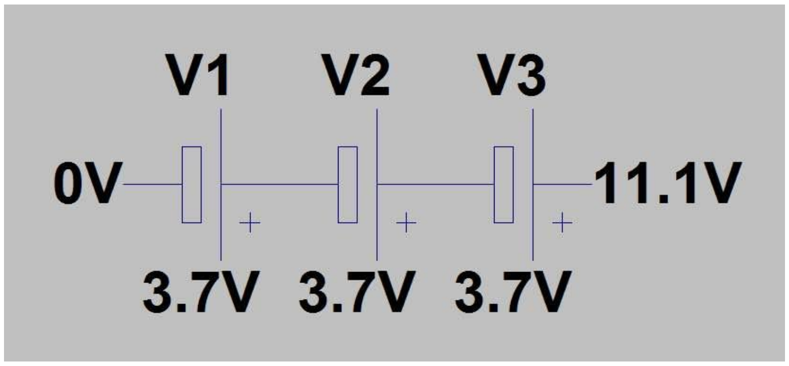
2.6 Control Board

In ordered to meet autonomous control objective on the quadcopter system, additional sensors are required such as Lidar, GPS and Sonar. Quadrotors are now beginning to be applied in commercial fields for surveillance and aerial videography by using GPS to sense the position of the vehicle. A GPS module is used to determine the actual (current) position of the quadcopter. The GPS module needed interfacing with the control board so as to collect data. The quadcopter requires a GPS module that is relatively smaller in size, fast and accurate, yet consuming as little power as possible and overally with easiness. The United States of America Department of Defense operates a GPS (which is a satellite network). This network of satellites transmits data concerning its current location and time. A GPS receiver passively retrieves this data from multiple satellites so as to estimate its position. A GPS receiver can actually determine its current position three dimensions by estimating the distance between more than three satellites.

2.7 Battery/ Power Supply

For this project there was need for a power supply of relatively lower cost, long battery life yet rechargeable and that can last up to 30 minutes of flight. From a thorough research, I discovered that there are currently three main types of rechargeable batteries available commercially for radio controlled modules, namely; Nickel-cadmium (NiCad), nickel-metal hydride (NiMH), and lithium polymer (LiPo) batteries. I decided to use the 3.7 per cell lithium polymer batteries due to their low weight and high capacity. The Lipo batteries are rated according to their current and their current discharge is specified by this current rating for example with a battery of rating 12C, the expected discharge is 12 times the battery's capacity. The above mentioned quadcopter components were ordered from China through Ali-express, online.

(a)



(b)

Figure2.2: LiPo Battery

2.8 Video System for Surveillance at 5.8GHz

Many different opinions are available for the camera. One considerable option was to mount an IP camera to the fuselage of the quadrotor so that it can produce a high - resolution image with its own transmitter. Such a camera requires connection to a network; it cannot function without an internet connection and would not be applicable in wilderness areas/ regions. On overall the setback is on cost – they are expensive.

The alternative option was to use a smart phone and having to install the IP WEBCAM android application and also employ a wide local area network (WLAN) to stream the video via a browser on the computer. This is what was employed in this project as it appeared more cost effective and reduced need for rigorous coding. The camera needed to be as light as possible so as to allow the UAV to fly unabated and be compact enough so that it will not interfere with the landing gear and rotors. The video system must be able to transmit the captured data over a suitable distance over open space with minimal loss of signal and interference. For the prototype design I opted for 1 000m as a suitable range, could transmit up to 5 000m.

2.9 RF Remote Controller

The controller is a RF type transmitter which creates the link between the user interface and the UAV which uses the Radio frequency protocol for transmission; this mode of communication uses a 2.4 GHz frequency similar to a WAN communication, they can connect to all 2.4 GHz RF system.

2.10 Communication protocol to interface remote controller and computer.

A connection must be established when creating a live interface between the computer and the on-board camera needed to establish a connection and also to control the quadrotor another communication protocol had to be evaluated, with the RF communication protocol that utilizes Radio Frequency.

Radio Frequency is a specification for a suite of high level communication protocols using tiny, low-power omnidirectional antennas.

A list of RF characteristics:

- The energy is the signal which can radiate off a conductor into space.
- It tends to flow via routes that contain insulating material, e.g. those found is a capacitor (dielectric)
- Full duplex radiating signal can be supported
- Incorporates power saving mechanisms for all device classes
- Various transmission options including broadcast
- Security key generation mechanism
- Requires specialized transmission lines so that it cannot reflect from discontinuities in the cable.

The key advantages of RF are:

- It is applicable in various medical fields: used in Diathermy instrument for surgery
- It is used in MRI for capturing images of human body as well as in skin tightening
- It is used in radar for object defection
- It is used in satellite communication
- It is used in microwave line-of-sight communication systems such as walls of buildings or houses based on the frequency hence used for radio and television transmission and also I n cellular mobile phone service

The main disadvantages of Radio Frequency include:

- As RF waves are available both in LOS and non-LOS regions of transmitter, it can be prone to intrusion by hackers and as a result crucial personal/ official data can be decoded for malicious motives. In order to counter for such a problem, radio frequency wave based transmission is (implemented with highly secured algorithms such as AES, WEP, WPA etc. RF signals can also be modulated either using frequency hopping or spread spectrum techniques to avoid this kind of eavesdropping.
- In order to create a live video feed from the UAV; an FPV system had to be implemented as it works on radio frequency 5.8GHz - which is a fast transmission rate. It could handle video feed transmission and for the remote controller RF communication protocol theoretically proved to be the most efficient and was employed at a different frequency and affective in Zimbabwe.

2.11 Gyroscope, accelerometer and magnetometer sensors

The autonomous characteristics of the rotor is basically brought about by these sensors: for balancing of the quadrotor in mid-air, instant feedback from sensors is needed. This as a result when combined together gives continuous angle, speed and height correction; through the implementation of the microprocessor which gives real-time algorithms that aids the data to be processed from the sensors'. A DSP is a part of the microprocessor which - serves to process the data from the sensors (row data), and then it is sent to the ESC in order to transform the frequency of the motors in the four parts of the quadrotor to match the specific required angular velocities to balance the system applying laws of centripetal forces.

Inside are 3 sensors, one being a classic 3-axis accelerometer, which serves to detect the direction which would be down towards the earth (by gravity measurement) or the speed of acceleration of the board in 3D space. The second one is a 3-axis magnetometer, which senses the source of the strongest magnetic force: generally detects magnetic north. The third being a 3-axis gyroscope, which measures spin as well as twist. The sensor has a digital (12C) interface. The 9 DOF sensor has SDA/ SCL pins and sew ESCs are mostly applied on electrically powered radio-controlled models, with variety most often used for brushless motors essentially providing an electrically generated three-phase electric power low voltage source of energy for the motors[10]

Irrespective of the type used, the ESC interprets control information and not a mechanical motion like servo motor but by varying the switching rate of network of FETs' conductive thread from the 3V, SDA, SCL and GND pins. These enable connection to the APM2.5 board.

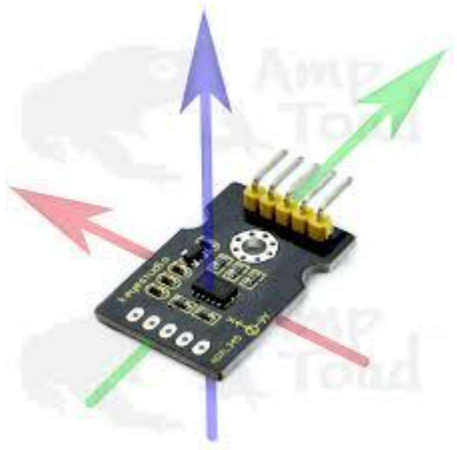


Figure 2.3: Gyroscopic sensor

2.12 Motors

The design is comprised of a symmetric array of four motors, which are commonly attached with an 'X' shaped frame. The spinning direction of each motor is alternated; opposite motors spin in the same direction, so as to counteract the reaction torque produced by the rotors. [11]

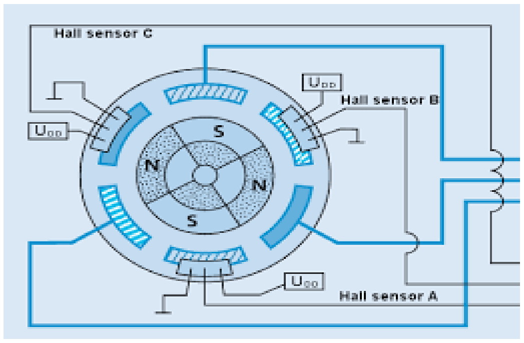


Figure 2.4: Motor Circuit

Brushless DC electric motor (BLDC motors), also called electrically commutated motors, are synchronous motors that are powered by a DC electric source with the use of an integrated inverter/ switching power supply, which produces an AC electric signal for driving the motors. They regulate the amount of power/ speed of the electric RC motor. These kind have become more popular with radio controlled airplanes because of their efficiency, power, longevity and light weight comparing with the traditional brushed motors.

2.13 Quadcopter's flight control mechanism

Quadcopters can be described as small vehicles with four motors which are evenly attached to rotors located at the cross frame. The main aim for having a fixed pitch rotors is to control the motion of the vehicle. The four rotors have independent rotating speeds. Due to the presence of independent, pitch, roll and yaw attitude of the vehicle can be controlled easily. Pitch, roll and yaw attitude of the Quadcopter can be visualized from Figure 2.1.5, 2.1.6 and 2.1.7 below.

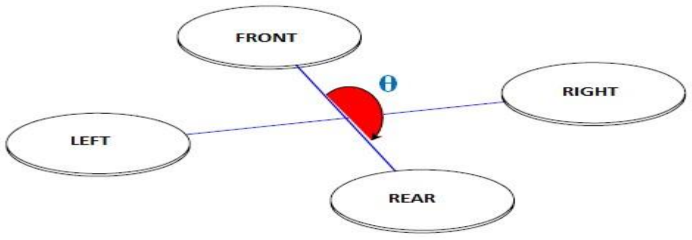


Figure 2.5: Pitch direction of Quadcopter

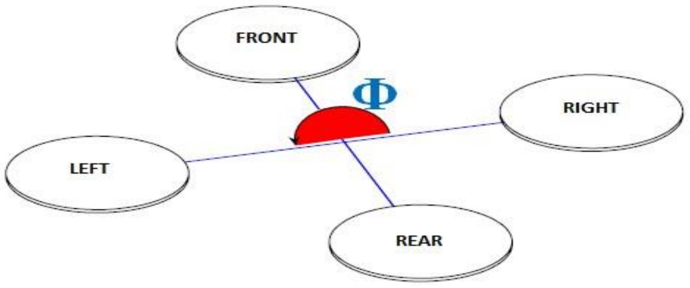


Figure 2.6: Roll direction of Quadcopter

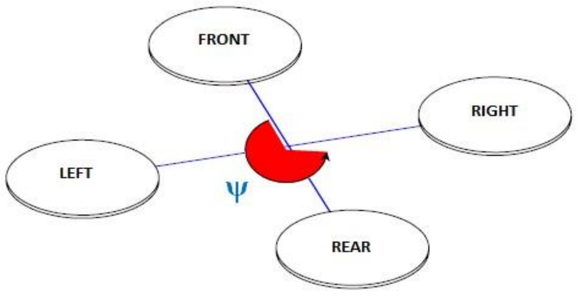


Figure 2.7: Yaw direction of Quadcopter

Quadcopters have four input forces and basically the thrust that is produced by the propellers connected to the rotor. It is this Thrust that lifts the copter using the drive from the motors.

2.13.1 Take-off and landing motion mechanism

The arrows represent the rotation direction whereas the spinning rotors are represented by the circles. Motors labelled FRONT and REAR rotate in a clockwise direction with the use of pusher rotors whereas motors labelled RIGHT and LEFT rotate in a counter-clockwise direction using puller rotors. A thrust and torque about the center of the quad-copter is produced by each motor. Because of oppositely spinning directions of the motors, the overall/ net torque is ideally zero about the center of the quad-copter, producing zero angular acceleration: this deals away with one's need for stabilization

Take-off is the elevation from ground to hover position whilst landing position is versa of take-off position. Take-off/ landing motions are controlled by increasing/ decreasing the speed of four rotors simultaneously, which means altering the vertical motion. Figure 2.1.8 and 2.1.9 illustrated the take-off and landing motion of Quad-copter respectively. By increasing the speed of all the motors by the same amount of throttle - a vertical force is created; as gravitational force of the earth is overcome by the vertical forces, the quad-copter will begin to rise in altitude.

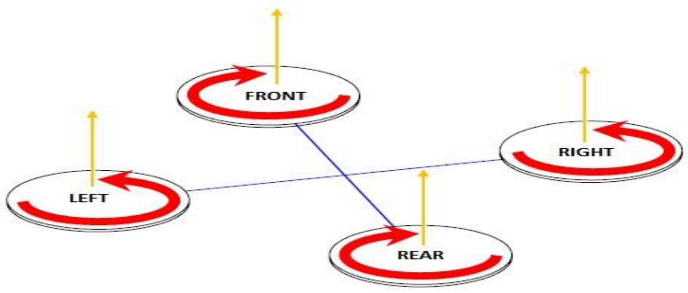


Figure 2.8: Take-off motion

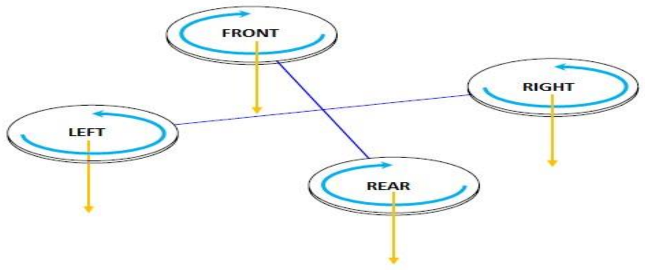


Figure 2.9: Landing motion

2.13.2 Forward and backward motion

Forward/ backward motion is controlled by increasing/ decreasing speed of rear/ front rotor. Decreasing/ increasing the rear/ front rotor speed simultaneously has an effect on the pitch angle of the Quadcopter. The forward and backward motions of Quadcopter are shown in Figure 2.1.10 and 2.1.11 respectively.

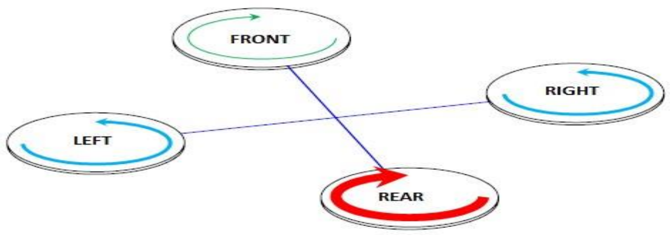


Figure 2.10: Forward motion

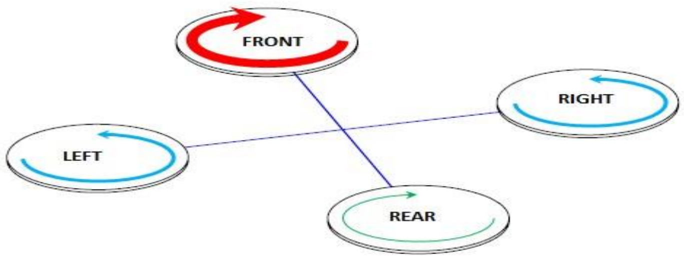


Figure 2.11: Backward motion

2.13.3 Left and right motion

For left and right motion, it can be controlled by changing the yaw angle of Quadcopter. Yaw angle is manipulated by increasing (decreasing) counter-clockwise rotors speed while decreasing (increasing) clockwise rotor speed. Figure 2.12 and 2.13 show the right and left motion of Quadcopter.

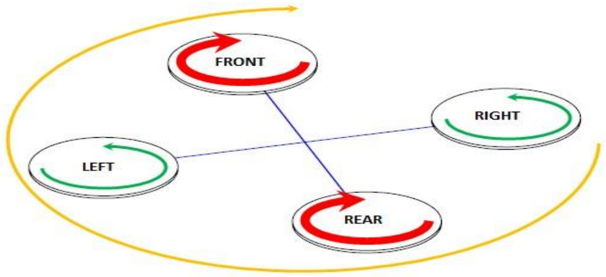


Figure 2.12: Right motion

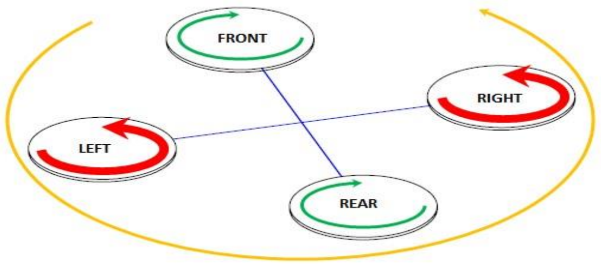


Figure 2.13: Left motion

2.13.4 Hovering or static position

The hovering or static position of Quadcopter is done by two pairs of rotors rotating in clockwise and counter-clockwise directions respectively at same speed. By two rotors rotating in clockwise and counter-clockwise position, the total sum of reaction torque is zero and this allows the Quadcopter to hover on one position.

2.14 Advantages of Quadcopter

Quadcopter has many advantages as compared to other aircrafts. It doesn’t necessarily require much space/ area to obtain elevation (lift), as compared to a fixed wing aircraft does. Thrust is created by the quadcopter with all four evenly distributed motors along its frame. Typically helicopters encounter torque issues due to its main rotor unlike quadcopters. By having counter balancing forces of the rotating motors allows cancelation out of torque forces caused by each motor causing the quadcopter to obtain self-balance. Less Kinetic energy is required per rotor for the same amount of thrust when put in comparison with the helicopter; this is because a quadcpoter uses four rotors instead of one. Maintenance and manufacturing costs are relatively lower than other aircrafts due to the above factors as well as its symmetrical design.

2.15 PID (Proportional Integral Derivation) - Controller

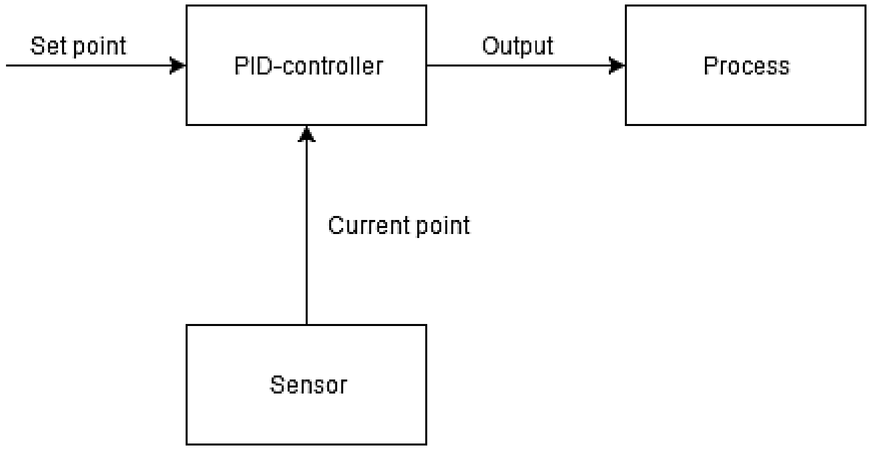


Figure 2.14: PID diagram

In this project, a proportional- integral-derivative-controller (PID) was implemented to deal with system auto stabilization. A PID controller gives certainty of matching where we intend to be - and actual position. This is known as the set-point (SP) an the current-point (CP). For example if we set the SP at 22^0C and the CP would be 24^0C. The controller therefore is required to regulate so that CP matches the SP (i.e. will lower the temperature). This results in the PID-controller giving an output. PID-controller has three variables as displayed in the figure above. The main job the PID-controller is employed for is to look at CP and the SP therefore determining the output.

The mathematical expression for a simple PID-controller is as below:

$$\text{Output} = \mathrm{k_p} e(t) + \mathrm{k_I} \int e(t)\, dt + \mathrm{k_D} \frac{d}{dt} e(t) \tag{2.1}$$

Where e = Setpoint – Input

In the equation Input = CP

Each term should be explained individually to determine which effect they have on the end result. The P-term is a factor multiplied by the error, the I-term is a factor multiplied by the error over time and the D-term is a factor multiplied by the difference in error. In this equation, a variation in time is used (this was not implemented in this project since the algorithm would run at a specific interval. The P-term is only dependent on the error and the k_p for instance given by 10 and error as 6, the P-term would equal to 60. A high gain helps the control loop to have a fast response. Though a very large gain will result in oscillation around the SP, which tends out to be very unsuitable for flight controller. The worst scenario would be not just a small oscillation but also unstabilization of the whole system; a crash in simpler terms.

The I-term sums the error over time. This means that even a small error will increase the I-term pushing it towards a steady-state. If continuing wind for example, the P-term would try to move the CP towards its SP, it might overshoot or undershoot, either way results in oscillation. This is where the I-term is large, it sums the errors to match the continuous external force keeping the system at a steady-state.

The D-term is proportional to the rate of change; does not favour change. This means that it will try to counter any change.

This is an overall basis of the PID-controller operation, though for the context of being used in a flight controller some modifications might be required.

A typical PID response curve is illustrated in Fig below:

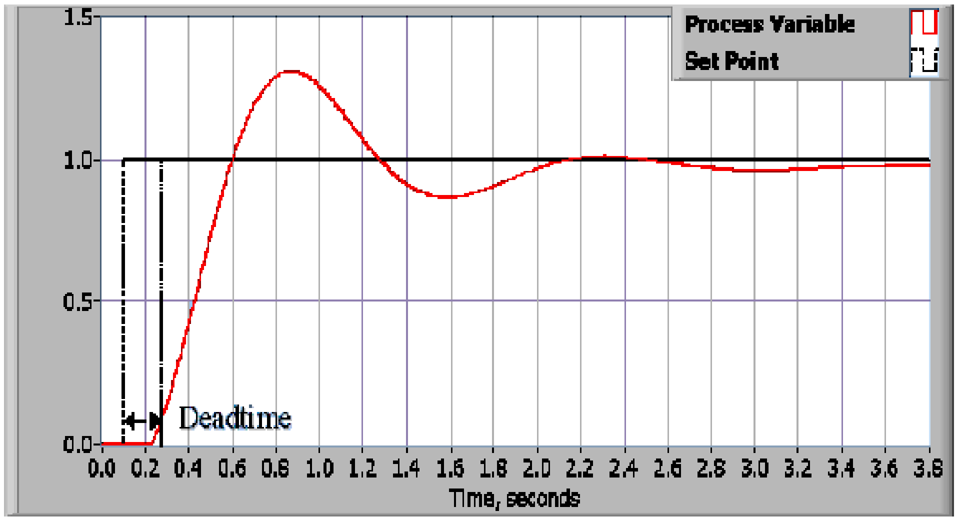


Figure 2.15: PID graph

Here the process variable equals the CP. The dead-time in Fig 2.1.15 will not always appear and will not be relevant in the context of this project. It would be as a result of delay from changing the output to the actual process reacting.

This algorithm can be applied in auto stabilization of the quadrotor system. For instance, if there be a wind gust that moves the quadrotor away from its SP, the algorithm will use the CP read from the IMU to calculate an output, which the flight controller can use to adjust the appropriate motors. This is auto stabilization at any SP, meaning it can be used to perform movement of a quadrotor as well.

2.16 Quad-copter mathematical modelling

The schematic movement of Quadcopter is represented in Figure 2.16 and based on this schematic, the Quadcopter mathematical modeling is derived as below [12]:

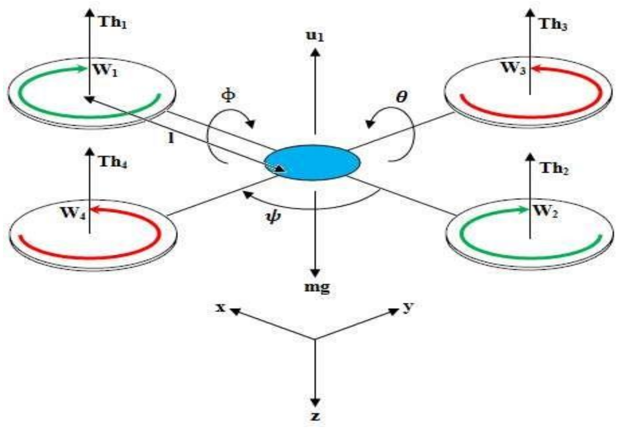


Figure 2.16: Schematic of Quadcopter

Where,

U_1 = sum of the thrust of each motor

Th_1= thrust generated by front motor

Th_2= thrust generated by rear motor

Th_3= thrust generated by right motor

Th_4= thrust generated by left motor

m = mass of Quadcopter

g = the acceleration of gravity

l = the half length of the Quadcopter

x, y, z = three position

θ, ϕ, ψ = three Euler angles representing pitch, roll, and yaw

The dynamics formulation of Quadcopter moving from landing position to a fixed point in the space is given as:

$$D = Rxyz \begin{bmatrix} C\phi C\Theta & C\phi S\Theta S\Psi - S\phi C\Psi & C\phi S\Theta C\Psi + S\phi S\Psi \\ C\phi S\Theta & S\phi S\Theta S\Psi + C\phi C\Psi & S\phi S\phi C\Psi - C\phi S\Psi \\ -S\Theta & C\Theta S\Psi & C\Theta C\Psi \end{bmatrix} \quad (2.2)$$

The D matrix is a description of the transformation Earth- fixed coordinates → body- fixed coordinates.

Where,

R = matrix transformation

$S\Theta = \text{Sin}(\theta)$, $S_\phi = \text{Sin}(\phi)$, $S\Psi = \text{Sin}(\psi)$

$C\Theta = \text{Cos}(\theta)$, $C_\phi = \text{Cos}(\phi)$, $C\Psi = \text{Cos}(\psi)$

By applying the force and moment balance laws, the Quadcopter motion equation are given in Equation (2.3) till (2.5) and Pythagoras theorem is computed as Figure 2.1.11.

$$\ddot{x} = u_1 (\text{Cos}\phi \text{Sin}\theta \text{Cos}\psi + \text{Sin}\phi \text{Sin}) - K_1\dot{x}/m \quad (2.3)$$

$$\ddot{y} = u_1 (\text{Sin}\phi \text{Sin}\theta \text{Cos}\psi + \text{Cos}\phi \text{Sin}) - K_2\dot{y}/m \quad (2.4)$$

$$\ddot{z} = u_1 (\text{Cos}\phi \text{Cos}\psi) - g - K_3/m \quad (2.5)$$

Where, Ki = drag coefficient (Assume zero since drag is negligible at low speed)

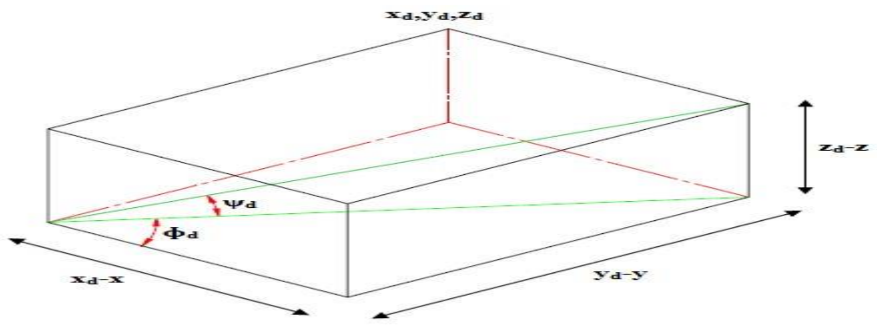


Figure 2.17: Angle movement of Quadcopter

The angle ϕ_d and ψ_d in Figure 2.17 are determined using Equation (2.6) and (2.7) respectively.

$$\phi_d = tan^{-1}\left(\frac{y_d-y}{xd-x}\right) \tag{2.6}$$

$$\psi_d = tan^{-1}\left(\frac{z\,d-z}{\sqrt{(xd-x)2+(yd-y)^2}}\right) \tag{2.7}$$

Quadcopters have four controller input forces U_1, U_2, U_3, and U_4 that will affect certain sides of the Quadcopter. U_1 will affect the attitude of the Quadcopter, U_2 affects the rotation in roll angle, U_3 affects the pitch angle and U_4 control the yaw angle. To control the Quadcopter movement is done by controlling each input variable. The equations of the forces are as below:

$$U \begin{cases} U_1 = (Th_1 + Th_2 + Th_3 + Th_4) / m \\ U_2 = l\,(-Th_1-Th_2+Th_3+Th_4) / I_1 \\ U_3 = l\,(-Th_1+Th_2+Th_3-Th_4) / I_2 \end{cases} \tag{2.8}$$

$$U_4 = 1(Th_1+Th_2+Th_3+Th_4) / I_3$$

Where,

Th_i = thrust generated by four motor

C = the force to moment scaling factor

I_i = the moment of inertia with respect to the axes

Then the second derivatives of each angle are:

$$\ddot{\theta} = u_2 - 1k_4\dot{\theta} \,/\, I_1 \tag{2.9}$$

$$\ddot{\Psi} = U_3 - lK_5\,\dot{\psi}/I_2 \tag{2.10}$$

$$\ddot{\phi} = U_1 - lK_6\dot{\phi}/I_3 \tag{2.11}$$

2.1.5.1 Direction Cosine Matrix

Vector rotation around x can be described as matrix

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Phi & sin\Phi \\ 0 & -sin\Phi & cos\Phi \end{bmatrix} \tag{2.12}$$

And around y as

$$R_y = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{2.13}$$

And around axis z as

$$R_z = \begin{bmatrix} cos\Psi & sin\Psi & 0 \\ -sin\Psi & cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

The D matrix describes the transformation Earth- fixed coordinates → body-fixed coordinates. The first set of state equation is describing the change of position according to quadrotor's attitude in its velocity measured in the body frame:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = D^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.15}$$

2.1.5.2 Angular rates transformation

The transformation between angular rates in Earth –fixed frame to body-fixed frame is given by equation [13]

$$\begin{bmatrix} p \\ q \\ \dot{r} \end{bmatrix} = \mathrm{E} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} \tag{2.16}$$

Where
$$\mathrm{E} = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\Phi & sin\Phi cos\theta \\ 0 & -sin\Phi & cos\Phi cos\theta \end{bmatrix} \tag{2.17}$$

Then the second set of state equations describing change of attitude according to rotation in the body frame is

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \mathrm{E}^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.18}$$

2.1.5.2 Linear acceleration

The linear acceleration in Earth-fixed frame is described by the Newton's Second Law

$$\mathrm{F} = \mathrm{m}\dot{V} \tag{2.19}$$

Where m is the quadrotor's mass which is constant and V is the Velocity vector in the body frame. The speeds u,v and w are measured in body-fixed coordinates and the body frame velocity vector can rotate and change its magnitude at the same time. This leads to total derivation of vector V [14]

$$\mathrm{F} = \mathrm{m}\dot{V} + \omega \times \mathrm{mV} \tag{2.20}$$

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + m \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.21}$$

After expanding the cross product and reorganizing

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \mathrm{m} \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{2.22}$$

Neglecting the aerodynamic force then the eternal forces acting in the quadrotor's body are thrust of the propellers T and weight force W.

Thrust is always acting in the body z axis while the weight force is projected according to the attitude of the quadrotor.

$$\begin{bmatrix} W_x \\ W_y \\ W_z - T \end{bmatrix} = \mathrm{m}\begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \qquad (2.23)$$

The weight force is always acting in the Earth's frame z axis. Conversionto body-fixed frame is done by the direction cosine matrix (2.4)

$$\mathrm{D}\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \mathrm{m}\begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \qquad (2.24)$$

After recognizing

$$\dot{u} = rv - qw - g\sin\phi$$

$$\dot{\mathrm{v}} = \mathrm{pw} - \mathrm{ru} + \mathrm{g}\cos\theta\sin\phi \qquad (2.25)$$

$$\dot{\mathrm{w}} = \mathrm{qu} - \mathrm{pv} + \mathrm{g}\cos\Phi\cos\theta - \frac{\mathrm{T}}{\mathrm{m}}$$

Considering no motor dynamics the thrust of all rotors is (thrust is proportional to the square of the propeller's angular rate) [15]

$$\mathrm{T} = \mathrm{b}\,(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \qquad (2.26)$$

Where b is a thrust coefficient and Ω_i is speed of each rotor.

This leads to another set of state equations

$$\dot{u} = rv - qw - g\sin\phi$$

$$\dot{\mathrm{v}} = \mathrm{pw} - \mathrm{ru} + \mathrm{g}\cos\theta\sin\phi \qquad (2.27)$$

$$\dot{\mathrm{w}} = \mathrm{qu} - \mathrm{pv} + \mathrm{g}\cos\Phi\cos\theta - \frac{\mathrm{b}}{\mathrm{m}}\,(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$$

2.1.5.3 Angular acceleration

Application of an external torque will change the angular momentum of the quadrotor

$$M = \dot{H} \tag{2.28}$$

And

$$M = \dot{H} + \omega \times H \tag{2.29}$$

$$H = I\,\omega \tag{2.30}$$

Where ω is the change of the attitude and I is the moment of inertia of the quadrotor. The quadrotor is a rigid body symmetric about its xz and yz plane, and the rotation axes coincidences with the principal axis, then the moment of the inertia tensor is

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.31}$$

Then

$$M = I\dot{\omega} + \omega \times I\omega \tag{2.32}$$

After expanding

$$\begin{aligned} M_x &= \dot{p}I_x + qr(I_z - I_y) \\ M_y &= \dot{q}I_y + pr(I_x - I_z) \\ M_z &= \dot{r}I_z + pq(I_y - I_x) \end{aligned} \tag{2.33}$$

And because of the xz and yz symmetry

$$I_x \approx I_y \tag{2.34}$$

The equation can be simplified to:

$$\begin{aligned} M_x &= \dot{p}I_x + qr(I_z - I_y) \\ M_y &= \dot{q}I_y + pr(I_x - I_z) \\ M_z &= \dot{r}I_z \end{aligned} \tag{2.35}$$

The external torque is produced by the thrust and drag of the propellers. Neglecting the propeller's inertia and aerodynamic torques, then the external torques can be written as:

$$M_x = lb(\Omega_2^2 - \Omega_4^2)$$

$$M_y = lb\,(\Omega_1^2 - \Omega_3^2) \tag{2.36}$$

$$M_z = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 1\Omega_3^2)$$

Where d is the drag factor of the rotors and l is the distance of the propeller from the CG. Then the last set of equations of motion is

$$\dot{p} = \frac{lb}{I_x}(\Omega_2^2 - \Omega_4^2) - qr\frac{I_z - I_y}{I_x}$$

$$\dot{q} = \frac{Ib}{I_y}(\Omega_1^2 - \Omega_3^2) - p\frac{rI_x - I_z}{I_y} \tag{2.37}$$

$$\dot{r} = \frac{d}{I_z}(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2)$$

2.2 Expanding the equations of motion

2.2.1 Gyroscopic moments of the propellers

The previous equations of motion are simplified. They do not take into account aerodynamic and gyroscopic forces and moments and the motor dynamics. Here I'm going to expand the already derived equations of motion by the terms associated with motor dynamics and gyroscopic moments.

Appending the gyroscopic moments to the moment equations leads to [16]

$$M_x = \dot{p}I_x + qr(I_z - I_y) + H_x + H_z q - H_y r$$

$$M_y = \dot{q}I_y + pr(I_x - I_z) - H_y + H_x r - H_z p \tag{2.38}$$

$$M_z = \dot{r}I_z + \dot{H}_z + H_y p - H_x q$$

Where H_x, H_y, H_z are total angular momentums of spinning masses with angular rates in x,y and z direction in the body frame.

$$H_x = \sum_{i=1}^{4} I_{xi}\ \omega_{xi} \tag{2.39}$$

$$H_y = \sum_{i=1}^{4} I_{yi}\ \omega_{yi} \tag{2.40}$$

$$H_z = \sum_{i=1}^{4} I_{zi}\ \omega_{zi} \tag{2.41}$$

The angular rates of the rotors are presented only in the z -axis (in the body frame) and there are no more rotating masses than them, the equations can be simplified to

$$\begin{aligned} M_x &= \dot{p}I_x + qr(I_z - I_y) + H_z q \\ M_y &= \dot{q}I_x + pr(I_x - I_z) - H_z p \\ M_z &= \dot{r}I_z + H_z \end{aligned} \tag{2.42}$$

The state equations for the angular rates with the propeller’s gyroscopic moments added are

$$\begin{aligned} \dot{p} &= \frac{lb}{I_x}(\Omega_2^2 - \Omega_4^2) - qr\frac{I_z - I_y}{I_x} - \frac{H_z}{I_x} q \\ \dot{q} &= \frac{Ib}{I_y}(\Omega_1^2 - \Omega_3^2) - pr\frac{I_x - I_z}{I_y} + \frac{H_z}{I_y} p \\ \dot{r} &= \frac{d}{I_z}(\mathit{\Omega}_2^2 + \mathit{\Omega}_4^2 - \mathit{\Omega}_1^2 - \mathit{\Omega}_3^2) \end{aligned} \tag{2.43}$$

2.2.2 Engine dynamics

The motors propelling the quadrotor have their own dynamics. The equations of motions are the well -known equations of DC motor with aerodynamic damping added.

$$L\frac{di}{dt} = u – Ri - k_e\omega_m \tag{2.44}$$

$$J_r\dot{\omega}_m = k_i i - d_m\omega_m - f(\omega_m)$$

Where

- L – inductance of the coil in the engine
- i – current flowing through the engine
- R – resistance of the coil and wirings
- k_e – back EMF constact
- ω_m – motor angular rate
- d_m – bearing damping constant
- J_r – moment of inertia of the rotor
- k_i – torque constant
- $f(\omega_m)$ – nonlinear drag torque function for the given propeller

2.2.3 Engine dynamics identification, introduction

In the above chapter 2.3.2 I highlighted equations outlining a general DC electric engine. It is a set of two first order equation, which means the overall system is of order two. The engines used n the quadrotor are very small thus having small inductances and back EMF constant. This means that I can discard the dynamics of the current without any bigger impact of the model precision. This leads to only one differential equation

$$0 = u - Ri$$

$$J_r\dot{\omega}_m = k_i i - d_m\omega_m = -d_m\omega_m - f(\omega_m) + \frac{k_i}{R}u \qquad (2.45)$$

A general LTI (linear Time Invariant) system of first order can be described by the following transfer function:

$$G(s) = \frac{k}{(\tau s+1)} \qquad (2.46)$$

Where k stands for the DC (steady-state) gain and τ for time constant of the system.

The differential equations of the engine are nonlinear, which means that the linear transfer function will only be valid in close vicinity of an arbitrary trim point. The most convenient trim point is the hovering quadrotor.

REFERENCES

[1] G. D. R. K. Anudepp M, "Design of a Quadcopter and Fabrication," International Journal of Innovations in Engineering and Technology(IJIET*)*, vol. 4, no. 1, pp. 1-8, August 2014.

[2] T. W. (. P. Micheal Hoang, "Final Report Design, Implementationt, and Testing of a UAV Quadcopter," G11_Final_Report_2013, vol. I, no. 1, pp. 14-18, 2013.

[3] Q. Arena, "the-history-of-drones-and-quadcopter," 05 January 2017. [Online]. Available: http://quadcopterarena.com/the-history-of-drones-and-quadcopters/. [Accessed 23 March 2017].

[4] ZIMSTAT, "Statistical Presentation Data," 6 September 2016. [Online]. Available: www.statsontraffic.com/pagehandler/en-us/statistics/recordonfile. [Accessed 16 May 2018].

[5] Kong Wai Weng (2011). "Quadcopter" Robot Head To Toe Magazine September 2011 Volume 3, pp. 1-3.

[6] MicroChip, "PIC 16F877 Datasheet," 6 September 1998. [Online]. Available: www.microchip.com/pagehandler/en-us/products/picmicrocontrollers. [Accessed 16 April 2018].

[7] Sam, 2014. Electronic Speed Controller for Multirotor. Drone Test. Available: http://www.dronetrest.com/t/what-to-consider-when-buying-a-esc-for-your-multirotor/1305. [Accessed 11 May 2018].

[8] NXP," 3-Phase BLDC Motor Control with Sensorless Back EMF Zero Crossing Detection Using 56F80x," 2016, [Online]. Available: cache.freescale.com/files/product/doc/AN1914.pdf.

[9] NXP, "Sensorless PMSM Field-Orientated Control," 2016, [Online]. Available: http://www.nxp.com/doc/DRM148. [Accessed 20 March 2018]

[10] Atheer L. Salih, M. Moghavvemil, Haider A. F. Mahamed and Khalaf Sallom Gaeid (2010). "Flight PID Controller Design for a UAV Quadcopter." Scientific Research and Essays Vol.5 (23), pp. 3660-3667, 2010

[11] Ashfaq Ahmad Mian, Wang Daobo (2007). "Nonlinear Flight Control Strategy for an Underactuated Quadrotor Aerial Robot" 2007 IEEE Journal.

[12] Frank Haffman, Niklas Goddemeier, Torsten Bertam (2010). "Attitude estimation and control of Quadcopter" 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[13] Cook, M.V., 1997. *Flight Dynamics Principles* 1st ed., John Wiley & Sons.

[14] Blakelock, J.H., 1991. *Automatic Control of Aircraft and Missiles* 2nd ed., Wiley-Interscience.

[15] Hoffmann, G.M. et al., 2007. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*. p. 1–20.

[16] Stepaniak, M.J., 2008. *A Quadrotor Sensor Platform*. Ohio University

CHAPTER 3: METHODOLOGY

3.1 Overview of the system

The main thrust behind this research is to reduce the amount of traffic violation offenses, which has a massive impact on the police force as well as the society as a whole. Using UMV the Quadrotor has a RF communication remote module and a camera mounted to it. The police force personnel operating the system can monitor the traffic as well as receiving real time images of vehicles. A microcontroller with a code embedded into it, is mounted on this Quadrotor. This code plays an important role in enabling the user to control the Quadrotor by simply transmitting commands to the Quadrotor from the base/ station (on ground) via the RF communication protocol. A display interface/ screen is placed at the base/ station where the operator can view the images or live video feeds transmitted back from the remotely located hovering Quadrotor. The Quadrotor has a remote interface with which the controlling officer can remotely control its take-off from the ground. The serial-to-RF telemetry adapter is linked to the microcontroller through the microcontroller's serial port. The user's command or instructions are received by the microcontroller and hence configures the lights accordingly. This system was designed to control eight lights, being hard-wired to the microcontroller. These lights tend to display the status of communication and the current update so that the controller can monitor communication even without the use of the HMI [1].

3.2 Quadrotor Block Diagram and Explanation

This block diagram of the Quadrotor sighting the major components in the design of the hardware. The main components include; microcontroller RF module, brushless motors, power supply, and surveillance camera. The microcontroller - embedded code is programmed in C and C++ programming languages. This code allows the controller to receive signals from the gyro sensors and the microcontroller transmits the signals to the four motors so as to either increase or decrease the angular velocity of the motors. The Quadrotor can stabilize itself in air by either increasing or decreasing the speed of rotation of the rotor blades (or moving in either direction from its original position). The code embedded in the microcontroller also aids establishment of communication link between the remote control at the base/ station and the RF module on the rotor. The C or C++ code running on the microcontroller allows the remote controller to transmit data or commands entered by the user to the RF module inbuilt in the remote control. This module will then transmit this information to the remote RF receiver module through the RF communication protocols.

This is the Block diagram of the Quadcopter sighting the major components in the design of the Hardware.

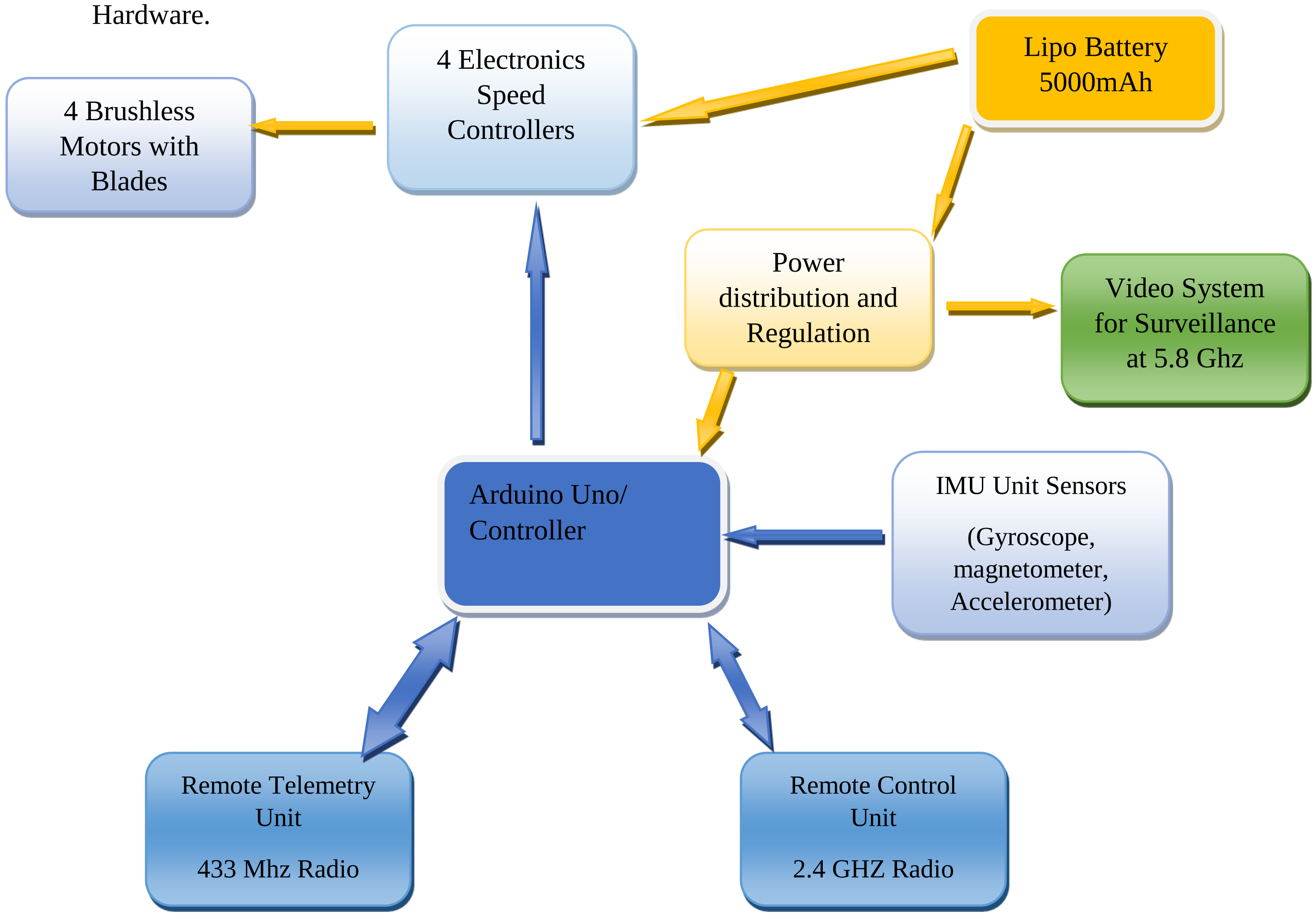

4 Brushless Motors with Blades
4 Electronics Speed Controllers
Lipo Battery 5000mAh
Power distribution and Regulation
Video System for Surveillance at 5.8 Ghz
Arduino Uno/ Controller
IMU Unit Sensors
(Gyroscope, magnetometer, Accelerometer)
Remote Telemetry Unit
433 Mhz Radio
Remote Control Unit
2.4 GHZ Radio

Figure 3.1: Quadcopter Block Diagram

The microcontroller was interfaced with the RF receiver module, having the received data transferred to the microcontroller serially through the serial port. The code embedded in the microcontroller allows the microcontroller to interpret and manipulate data received by the RF module as well as process it with the instruction of the code. From the processed data, the microcontroller transmits an output signal to its output port - which is interfaced to the camera. Therefore, a bidirectional communication is established between the Quadrotor and the remote control. The power supply unit supplies the required amount of regulated voltage to all components of the rotor. The ESC connects the motors to the microcontroller and regulates the angular velocity of each motor.

3.3 System Description

The minimum components I carefully selected them, and these include a control board, four brushless motors, four propellers, batteries (rechargeable), Quadrotor chassis, the long-range satellite transmitter, video camera, and remote controller. The controller can be a receiver and application on a computer that will be used by the rotor for its motion commands e.g. up or down and the video camera that will be getting the main point of concentration of this project. It will be a CMOS camera connected via a FPV. System at 5.8GHz frequency will be transmitting live feed to the station. The Quadrotor will carry the camera (wireless) and will be controlled using an RF transmitter. The long-range transmitter is an already purchased transmitter using a 2.4GHz antenna. The antenna will focus the signal from the station to the rotor and increase signal range. More emphasis is on weight and stability of the Quadrotor [2]. The Quadrotor is designed to be of lightweight as much as possible so as to maximize its flight time as based on the basic work law, more weight – more power required to lift the weight. The frame of the rotor is designed in such a way that it balances the force of the components mounted to it with the power of the rotors that lift the copter.

3.4 System Software Development

The different software components were developed using various programming languages and compilers due to the differences in purpose fitting.

The microcontroller code was built in C and C++ languages on Mikro-Basic PRO for PIC, a full-featured Basic compiler which makes Microchip PIC development ideal for all. Its environment has a vast range of features namely; easy-to-use IDE, very compact and efficient code, hardware and software libraries, software simulator, hardware debugger support, comprehensive documentation, COFF file generation etc. [3]. The most recent version (v4.15) supports Enhanced mid-range PIC16 family, sped up compilation time 3.5 times, and unveiled new libraries and much more. Some of the advantages of this IDE are:

- Free Life-time product technical support
- Free updates of new compiler versions
- Above 415 PIC microcontrollers are supported
- Several hardware and software libraries
- Numerous ready-to-use practical examples
- User-friendly IDE with additional tools
- Easy-to-understand documentation
- ANSI C compiler with minor modifications

The rest of the system software were developed using the below mentioned software tools:

- Proteus software for schematic design and simulation
- Visual Studio for the design of the User Interface (UI)
- MDK Keil Uvision 5 software for the flight testing and simulation
- Mission Planner software for calibration and testing of the gyroscope, as well as the calibration of the motors
- Taulabs software for beta testing

3.5 Circuit Schematic Connection Simulation

For the purposes of transmission testing, the use of a hyper-terminal was utilized to see if there was
any data transmitted between an external transceiver and the control circuit developed above. LEDs and
switches were included in the schematic to highlight any data transmission lines and to simulate any sensor
port inputs.

3.6 Hardware and software implementation

3.6.1 Hardware Implementation.

This phase of the project was conducted in multiple stages

3.6.2 Circuit Schematic Design

Upon the acquisition of the requirements for the physical aspects of the project, the first step undertaken was the simulation of design of the intended circuit on Proteus 8.5 Suite. A control card for testing was developed with a microcontroller and connecting ports for interfacing with the sensors on the frame. The screen shot below clearly shows the test circuit with a microcontroller at its heart with an RF module for testing the transmitter for a new design and gyroscopes. This circuit was developed to replicate the intended final control circuit of the Transmitter.

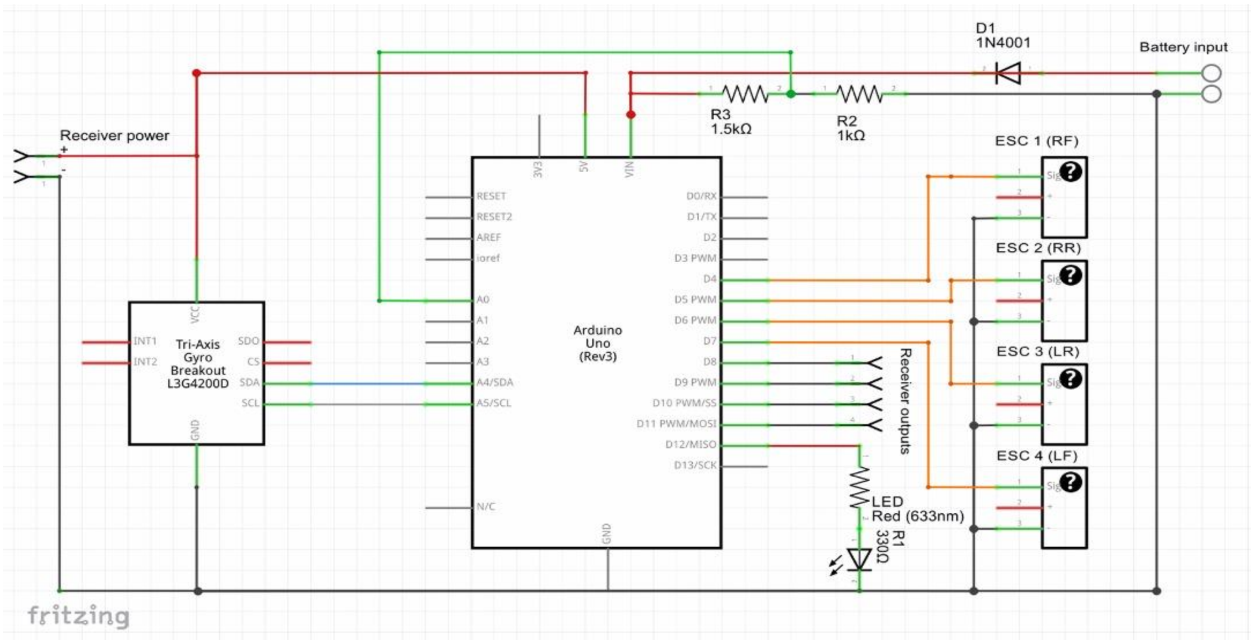


Figure 3.2 Schematic in Frit zing design

3.6.2 Circuit Schematic Connection Simulation

For the purposes of transmission testing, the use of a hyper-terminal was utilized to see if there was any data transmitted between an external transceiver and the control circuit developed above. LEDs and switches were included in the schematic to highlight any data transmission lines and to simulate any sensor port inputs.

3.6.3 Testing the Hardware

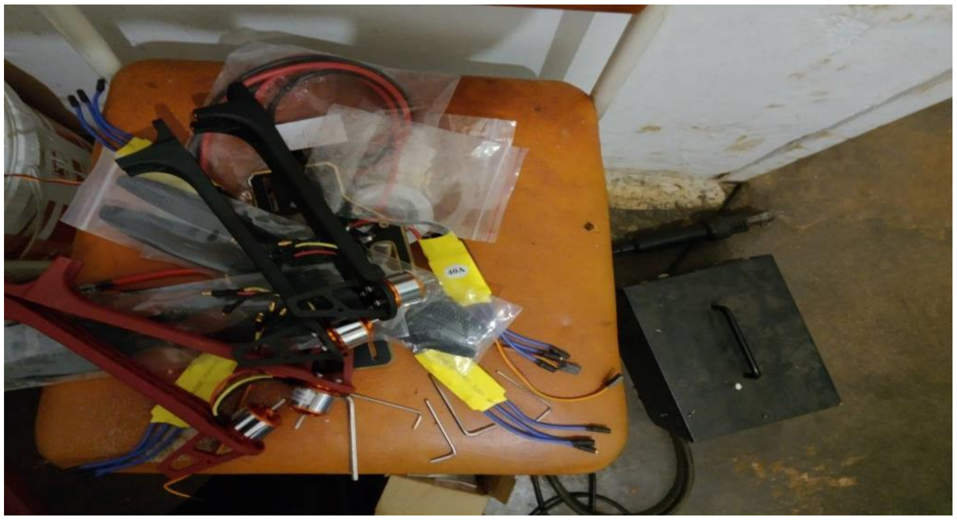


Figure 3.4: Actual components after shipping

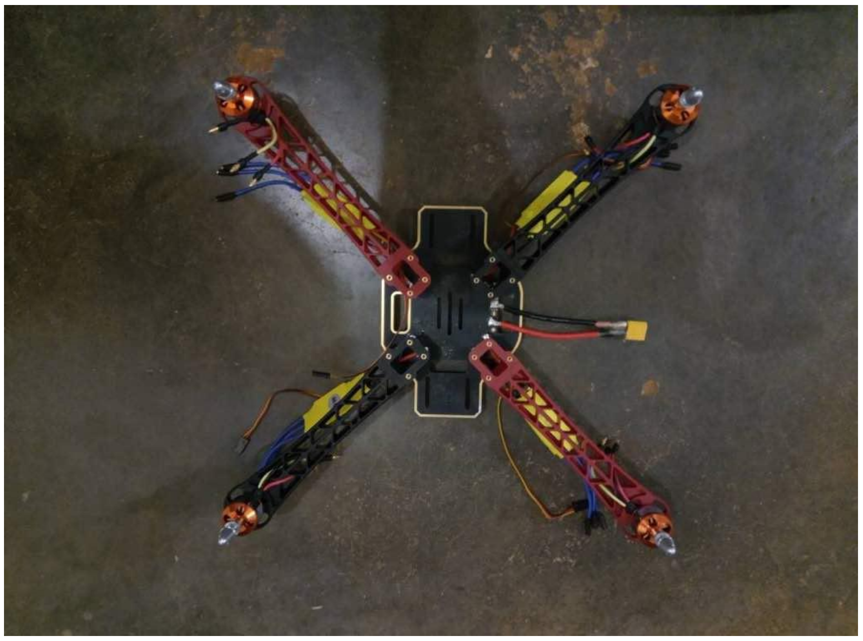


Figure 3.5: Initial building stage

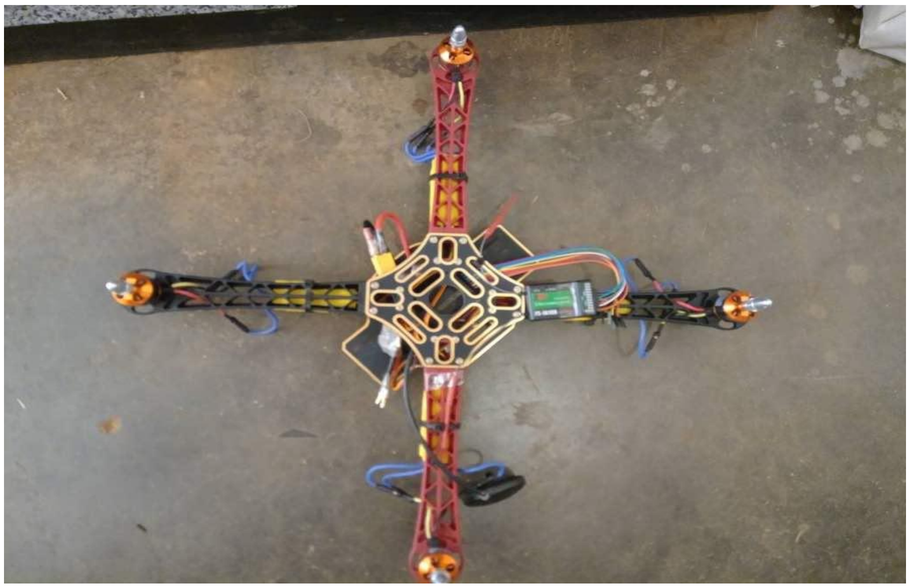


Figure 3.6: The second building stage

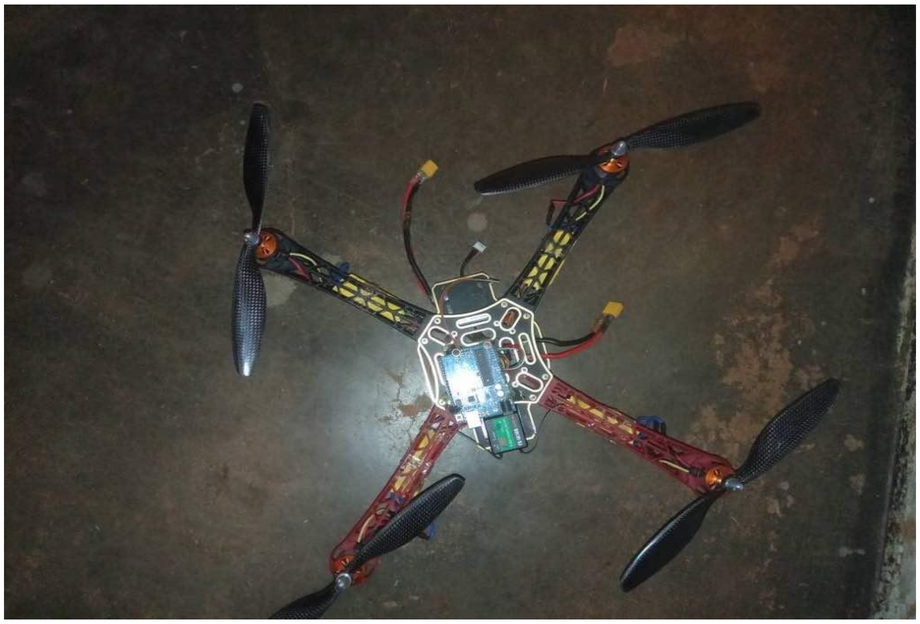


Figure 3.7: The third building stage

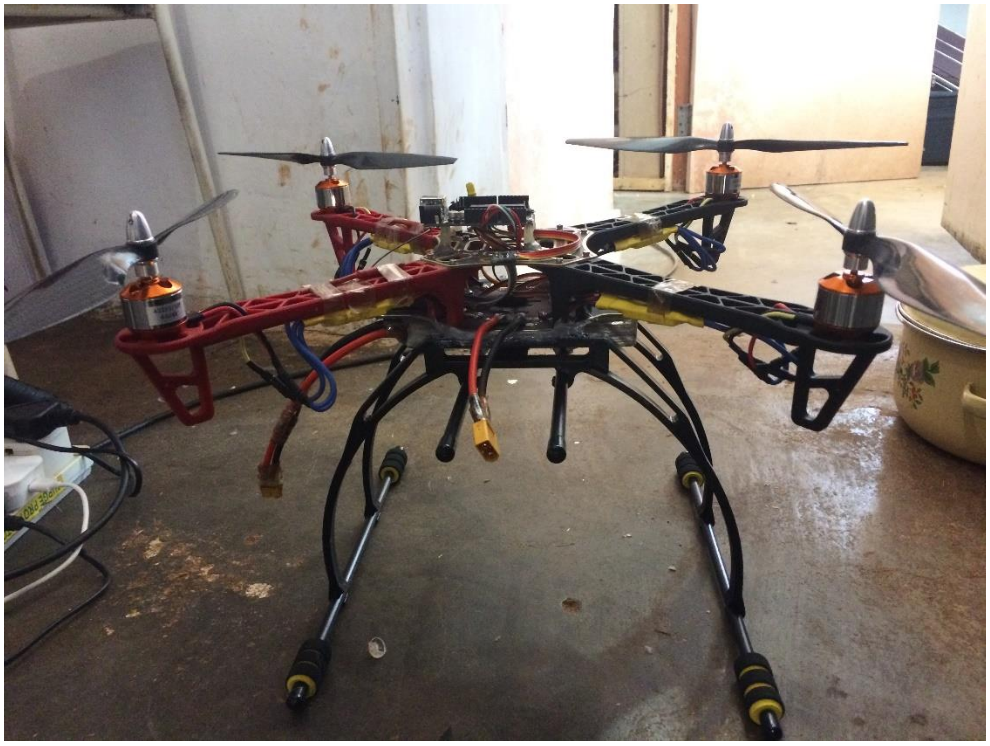


Figure 3.8: The realized hardware during troubleshooting

REFERENCES

[1] T. S. Alderete, “Simulator aero model implementation.” NASA Ames Research Center, Moffett Field, California, http://www.aviationsystemsdivision.arc.nasa.gov/publications/hitl/rtsim/Toms.pdf.

[2] H. Bouadi and M. Tadjine, “Nonlinear observer design and sliding mode control of four rotors helicopter,” Proceedings of World Academy of Science, Engineering and Technology, vol. 25, pp. 225–230, 2007.

[3] K. M. Zemalache, L. Beji, and H. Marref, “Control of an under-actuated system: Application to a four rotors rotorcraft,” IEEE International Conference on Robotic and Biomimetics, pp. 404–409, 2005.

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, results of the procedures carried out in this project are presented from the observed outcomes, with discussions attempting to explain their deviations, if any, from the expected results

4.2 Results: Expected versus Actual

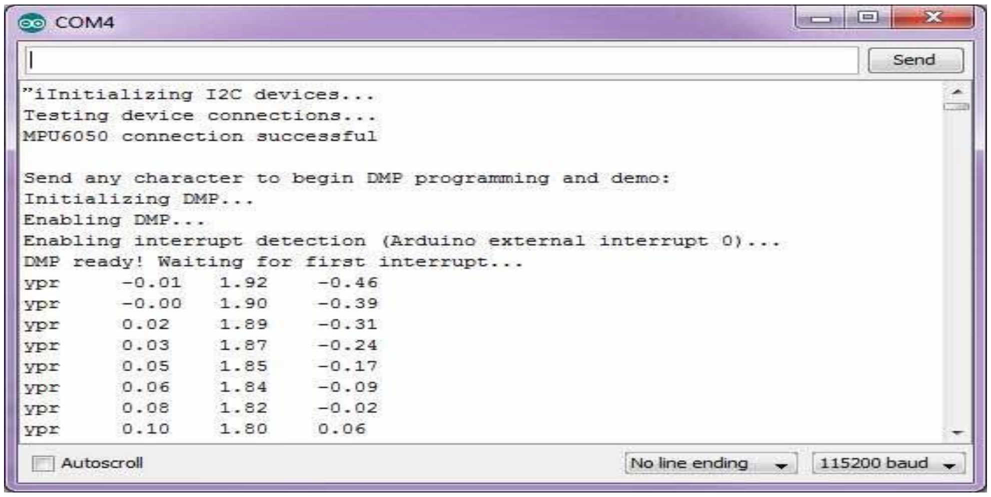
At the onset, the system was expected to be able to:

- Stabilize in the air
- Be controlled by the user and maneuver
- Able to send a live video to the user PC using the FPV system.

Typical Control commands and their responses were expected as follows:

REMOTE CONTROL	COMMAND ACKNOWLEDGED	ERROR REPORT
UP	Copter Going UP	No command given!
DOWN	Copter Going Down	No command given!
LEFT	Copter Going Left	No command given!
RIGHT	Copter Going Right	No command given!

The actual results were met on the above table as the copter could accept user input and respond as per the instruction sent from the RF remote controller. The PID tests that were done are shown in Figure4.1 below for the Yaw, Pitch and Roll.



```
"iInitializing I2C devices...
Testing device connections...
MPU6050 connection successful

Send any character to begin DMP programming and demo:
Initializing DMP...
Enabling DMP...
Enabling interrupt detection (Arduino external interrupt 0)...
DMP ready! Waiting for first interrupt...
ypr     -0.01   1.92    -0.46
ypr     -0.00   1.90    -0.39
ypr     0.02    1.89    -0.31
ypr     0.03    1.87    -0.24
ypr     0.05    1.85    -0.17
ypr     0.06    1.84    -0.09
ypr     0.08    1.82    -0.02
ypr     0.10    1.80    0.06
```

Figure 4.1: PID Test Result

The Seismometer in Fig4.2 shows the variation in the PID response to stability of the quadrotor system. The graph shows random spikes due to the instabilities mainly caused by external factors such as gusts of wind, then followed by attempts by the PID to stabilize the rotor.

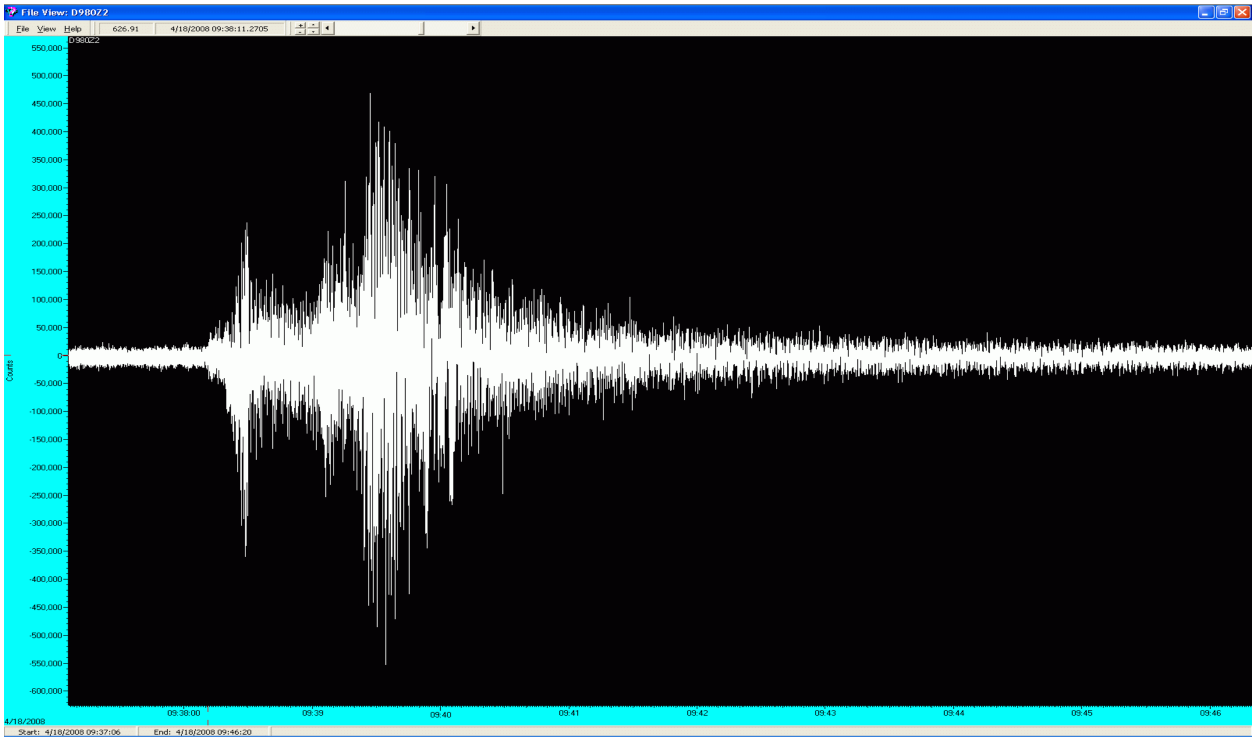


Figure 4.2: Seismometer Test Presentation

4.3 Discussion and Challenges faced

The major challenge that was faced was of having the quadrotor to take off steadily and vertically as well as putting it to hovering state. During the test of the quadrotor, after it took off very well (vertically as expected) – due to the PID instability. One of the frame arms was damaged and had to be replaced. Fig below shows the damaged frame:

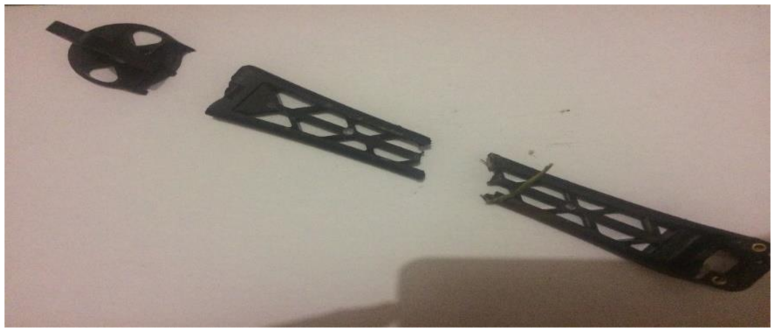


Fig 4.3: Broken frame

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Overall achievement/conclusion of the project

The overall aim of the project was to realize a functional microprocessor-based quad-copter capable of surveillance purposes using a camera mounted on its frame and then providing instant video feedback to a computer implementing the RF communication protocol at different frequencies. This system generally achieved its purpose, although the challenge of replacing the broken frame which meant that the re-configuration of the entire system.

5.2 Individual Conclusions

At the onset of the project, a number of key objectives were set to be achieved, as clear yardsticks of the progress and subsequent success (or otherwise) of the project. A recap, with completion markers would show this progress against these objectives as follows

- To design a quad-copter which is controlled wirelessly using a remote- this was successfully achieved.
- To create a live interface between the quad copter and the computer- this was achieved through the FPV camera interface.
- To design a quad-copter that demonstrates stability in the air- the main challenge with this objective was maintaining rotor speeds equivalent on all four rotors, a task solved through coupled PWM (Pulse Width Modulation). Initially, an attempt at achieving this through C revealed that four pulse width were required, something that was attained through the use of Arduino code.
- To transmit instant telemetry information from the quadcopter to the Computer via the ground and air station modules at 433Mhz and this was successfully achieved.

5.3 Recommendations

The biggest recommendation would be for the integration of a renewable energy source to allow for longer flying time and active time, with solar energy being a particular area of great encouragement. The communication protocol can be replaced in higher budget models to allow for greater ranges of influence and operation. This is an area of great potential at relatively lower cost than conventional methods of traffic offense monitoring.

In addition, high quality ESCs need to be used in the future such you can avoid the ESCs burn mystery associated with the low quality available ESCs.

7. APPENDIX

7.1 QUADROTOR CODE

```
/**
* system.c: initialize and configure all parts of the Quadcopter
*/

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

#include <EEPROM.h>          //Include the EEPROM.h library so we can store information onto
the EEPROM

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

//PID gain and limit settings

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

float pid_p_gain_roll = 2.00;//3.5; //4; //my base is 2.2, joop> 1.3;  //Gain setting for the roll P-
controller

float pid_i_gain_roll = 0.04;//0.05; // 0.04;             //Gain setting for the roll I-controller

float pid_d_gain_roll = 30;//30; //25;//35;// 18.0;    //Gain setting for the roll D-controller

int pid_max_roll = 400;                  //Maximum output of the PID-controller (+/-)


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

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

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

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


float pid_p_gain_yaw = 4.0;               //Gain setting for the pitch P-controller. //4.0

float pid_i_gain_yaw = 0.02;              //Gain setting for the pitch I-controller. //0.02
float pid_d_gain_yaw = 0.0;                //Gain setting for the pitch D-controller.

int pid_max_yaw = 400;                     //Maximum output of the PID-controller (+/-)

boolean auto_level = true;                 //Auto level on (true) or off (false)

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

//Declaring global variables

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

byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;

byte eeprom_data[36];

byte highByte, lowByte;

int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3,
receiver_input_channel_4;

int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4,
loop_counter;

int esc_1, esc_2, esc_3, esc_4;

int throttle, battery_voltage;

int cal_int, start, gyro_address;

int receiver_input[5];

int temperature;

int acc_axis[4], gyro_axis[4];

float roll_level_adjust, pitch_level_adjust;


long acc_x, acc_y, acc_z, acc_total_vector;
```

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

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

unsigned long loop_timer;

double gyro_pitch, gyro_roll, gyro_yaw;

double gyro_axis_cal[4];

float pid_error_temp;

float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;

float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch, pid_last_pitch_d_error;

float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;

float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;

boolean gyro_angles_set;

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

//Setup routine

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

void setup()

{

//  Serial.begin(57600);                    //Copy the EEPROM data for fast access data.

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

  start = 0;                                                    //Set start back to zero. gyro_address = eeprom_data[32];

                                                         //Store the gyro address in the variable.
```

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

TWBR = 12;                                 //Set the I2C clock speed to 400kHz.

//Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs.
DDRD |= B11110000;                         //Configure digital poort 4, 5, 6 and 7 as output.

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

                                           //Use the led on the Arduino for startup indication.

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

 //Check the EEPROM signature to make sure that the setup program is executed.

while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] != 'B')delay(10);

 //The flight controller needs the MPU-6050 with gyro and accelerometer

 //If setup is completed without MPU-6050 stop the flight controller program

  if(eeprom_data[31] == 2 || eeprom_data[31] == 3)delay(10);

set_gyro_registers();                                //Set the specific gyro registers.

for (cal_int = 0; cal_int < 1250 ; cal_int ++){              //Wait 5 seconds before continuing.

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

 delayMicroseconds(1000);                                     //Wait 1000us.

 PORTD &= B00001111;                                          //Set digital poort 4, 5, 6 and 7 low.

 delayMicroseconds(3000);                                     //Wait 3000us.

  }


//Let's take multiple gyro data samples so we can determine the average gyro offset (calibration).

for (cal_int = 0; cal_int < 2000 ; cal_int ++){               //Take 2000 readings for calibration.
if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12)); //Change the led status to indicate calibration.
gyro_signalen();                                        //Read the gyro output.
gyro_axis_cal[1] += gyro_axis[1];                       //Ad roll value to gyro_roll_cal.
gyro_axis_cal[2] += gyro_axis[2];                       //Ad pitch value to gyro_pitch_cal.
gyro_axis_cal[3] += gyro_axis[3];                       //Ad yaw value to gyro_yaw_cal.
//We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
PORTD |= B11110000;                                     //Set digital poort 4, 5, 6 and 7 high.
delayMicroseconds(1000);                                //Wait 1000us.
PORTD &= B00001111;                                     //Set digital poort 4, 5, 6 and 7 low.
delay(3);                                               //Wait 3 milliseconds before the next loop.
  } //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
  gyro_axis_cal[1] /= 2000;                             //Divide the roll total by 2000.
  gyro_axis_cal[2] /= 2000;                             //Divide the pitch total by 2000.
  gyro_axis_cal[3] /= 2000;                             //Divide the yaw total by 2000.
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 throtle is set to the lower position.

while(receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 ||
receiver_input_channel_4 < 1400){

receiver_input_channel_3 = convert_receiver_channel(3);       //Convert the actual receiver
signals for throttle to the standard 1000 - 2000us

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

 start ++;                                        //While waiting increment start whith every
loop.

//We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while
waiting for the receiver inputs.

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

delayMicroseconds(1000);                          //Wait 1000us.

PORTD &= B00001111;                               //Set digital poort 4, 5, 6 and 7 low.

delay(3);                                         //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.

    }

  }

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

//Load the battery voltage to the battery_voltage variable.
```

```
//65 is the voltage compensation for the diode.
//12.6V equals ~5V @ Analog 0.
//12.6V equals 1023 analogRead(0).
//1260 / 1023 = 1.2317.
//The variable battery_voltage holds 1050 if the battery voltage is 10.5V.
battery_voltage = (analogRead(0) + 65) * 1.2317;
loop_timer = micros();                                    //Set the timer for the next loop.
                                                          //When everything is done, turn off the led.
digitalWrite(12,LOW);                                     //Turn off the warning led.
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main program loop
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop(){
//65.5 = 1 deg/sec (check the datasheet of the MPU-6050 for more information).
gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_roll / 65.5) * 0.3);     //Gyro pid input is deg/sec.
gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_pitch / 65.5) * 0.3);//Gyro pid input is deg/sec.
gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_yaw / 65.5) * 0.3);        //Gyro pid input is deg/sec.
//////////////////////////////////////////////////////////////////////////////////////////////////////
//This is the added IMU code from the videos:
```

```
//https://youtu.be/4BoIE8YQwM8
//https://youtu.be/j-kE0AMEWy4
////////////////////////////////////////////////////////////////////////////////////////////////


//Gyro angle calculations                                   //0.0000611 = 1 / (250Hz / 65.5)
angle_pitch += gyro_pitch * 0.0000611;                      //Calculate the traveled pitch angle and add this to the angle_pitch variable.
angle_roll += gyro_roll * 0.0000611;                        //Calculate the traveled roll angle and add this to the angle_roll variable.
 //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);  //If the IMU has yawed transfer the roll angle to the pitch angel.
angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066);   //If the IMU has yawed transfer the pitch angle to the roll angel.
//Accelerometer angle calculations
acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));       //Calculate the total accelerometer vector.
//57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;            //Calculate the pitch angle.
angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;           //Calculate the roll angle.
//Place the MPU-6050 spirit level and note the values in the following two lines for calibration.
angle_pitch_acc -= 0.5;                                     //Accelerometer calibration value for pitch.
angle_roll_acc -= 2;                                        //Accelerometer calibration value for roll.
```

```
//  Serial.print(angle_pitch_acc);
//  Serial.print("\t");
//  Serial.println(angle_roll_acc);
angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;          //Correct the drift of the
gyro pitch angle with the accelerometer pitch angle.
angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;               //Correct the drift of the
gyro roll angle with the accelerometer roll angle.
pitch_level_adjust = angle_pitch * 15;                                    //Calculate the pitch angle
correction
roll_level_adjust = angle_roll * 15;                              //Calculate the roll angle correction
if(!auto_level){                                         //If the quadcopter is not in auto-level mode
pitch_level_adjust = 0;                                   //Set the pitch angle correction to zero.
roll_level_adjust = 0;                                    //Set the roll angle correcion to zero.
  }
//For starting the motors: throttle low and yaw left (step 1).
if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;
//When yaw stick is back in the center position start the motors (step 2).
if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1450){
start = 2;
angle_pitch = angle_pitch_acc;                                //Set the gyro pitch angle equal to the
accelerometer pitch angle when the quadcopter is started.
angle_roll = angle_roll_acc;                                   //Set the gyro roll angle equal to the
accelerometer roll angle when the quadcopter is started.
gyro_angles_set = true;                                   //Set the IMU started flag.
//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 < 1050 && receiver_input_channel_4 > 1950)start =
0;

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

//In the case of deviding 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 > 1508)pid_roll_setpoint = receiver_input_channel_1 - 1508;

else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = receiver_input_channel_1 - 1492;

pid_roll_setpoint -= roll_level_adjust;                              //Subtract the angle correction from
the standardized receiver roll input value.

pid_roll_setpoint /= 3.0;                                            //Divide the setpoint for the PID roll
controller by 3 to get angles in degrees.

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

//In the case of deviding 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 > 1508)pid_pitch_setpoint = receiver_input_channel_2 - 1508;

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


pid_pitch_setpoint -= pitch_level_adjust;                              //Subtract the angle correction
from the standardized receiver pitch input value.

pid_pitch_setpoint /= 3.0;                                    //Divide the setpoint for the PID pitch
controller by 3 to get angles in degrees.

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

 //In the case of deviding 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 > 1050){ //Do not yaw when turning off the motors.

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

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

calculate_pid();                              //PID inputs are known. So we can calculate the pid
output.

//The battery voltage is needed for compensation.

//A complementary filter is used to reduce noise.

//0.09853 = 0.08 * 1.2317.

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

//Turn on the led if battery voltage is to low.
```

```
if(battery_voltage < 1000 && battery_voltage > 600)digitalWrite(12, HIGH);

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

//throttle -= 1400;      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 (battery_voltage < 1240 && battery_voltage > 800){              //Is the battery connected?

esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);               //Compensate the esc-1 pulse
for voltage drop.

esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);               //Compensate the esc-2 pulse
for voltage drop.

esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);               //Compensate the esc-3 pulse
for voltage drop.

esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);               //Compensate the esc-4 pulse
for voltage drop.    }

if (esc_1 < 1050) esc_1 = 1050;                                        //Keep the motors running.

if (esc_2 < 1050) esc_2 = 1050;                                        //Keep the motors running.
if (esc_3 < 1050) esc_3 = 1050;                                   //Keep the motors running.

if (esc_4 < 1050) esc_4 = 1050;                                   //Keep the motors running.


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

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

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

if(esc_4 > 2000)esc_4 = 2000;                               //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.

}

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

//Creating the pulses for the ESC's is explained in this video:

//https://youtu.be/fqEkVcqxtU8

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

//! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

//Because of the angle calculation the loop time is getting very important. If the loop time is
```

```
//longer or shorter than 4000us the angle calculation is off. If you modify the code make sure

//that the loop time is still 4000us and no longer! More information can be found on

//the Q&A page:

//! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

if(micros() - loop_timer > 4050)digitalWrite(12, HIGH);          //Turn on the LED if the
loop time exceeds 4050us.

//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.

//There is always 1000us of spare time. So let's do something usefull that is very time
consuming.

//Get the current gyro and receiver data and scale it to degrees per second for the pid
calculations.

gyro_signalen();
```

```
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. This is used to read the
receiver signals.

//More information about this subroutine can be found in this video:

//https://youtu.be/bENjl1KQbvo

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

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.

}

  }

else if(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[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[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[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[4] = current_time - timer_4;                //Channel 4 is current_time - timer_4.
  }
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void gyro_signalen(){
//Read the MPU-6050
if(eeprom_data[31] == 1){
Wire.beginTransmission(gyro_address);                          //Start communication with the gyro.
Wire.write(0x3B);                                              //Start reading @ register 43h and auto
increment with every read.
Wire.endTransmission();                                        //End the transmission.
Wire.requestFrom(gyro_address,14);                             //Request 14 bytes from the gyro.
receiver_input_channel_1 = convert_receiver_channel(1);               //Convert the actual receiver
signals for pitch to the standard 1000 - 2000us.
receiver_input_channel_2 = convert_receiver_channel(2);               //Convert the actual receiver
signals for roll to the standard 1000 - 2000us.
receiver_input_channel_3 = convert_receiver_channel(3);               //Convert the actual receiver
signals for throttle to the standard 1000 - 2000us.
receiver_input_channel_4 = convert_receiver_channel(4);               //Convert the actual receiver
signals for yaw to the standard 1000 - 2000us.
while(Wire.available() < 14);                                  //Wait until the 14 bytes are received.
acc_axis[1] = Wire.read()<<8|Wire.read();                      //Add the low and high byte to the acc_x
variable.
acc_axis[2] = Wire.read()<<8|Wire.read();                      //Add the low and high byte to the acc_y
variable.
acc_axis[3] = Wire.read()<<8|Wire.read();                //Add the low and high byte to the
acc_z variable.

temperature = Wire.read()<<8|Wire.read();                //Add the low and high byte to the
temperature variable.

gyro_axis[1] = Wire.read()<<8|Wire.read();                    //Read high and low part of the
angular data.

gyro_axis[2] = Wire.read()<<8|Wire.read();                    //Read high and low part of the
angular data.

gyro_axis[3] = Wire.read()<<8|Wire.read();                    //Read high and low part of the
angular data.

  }

if(cal_int == 2000){

gyro_axis[1] -= gyro_axis_cal[1];                               //Only compensate after the
calibration.

gyro_axis[2] -= gyro_axis_cal[2];                               //Only compensate after the
calibration.

gyro_axis[3] -= gyro_axis_cal[3];                               //Only compensate after the
calibration.

  }

gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];                     //Set gyro_roll to the
correct axis that was stored in the EEPROM.

if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;              //Invert gyro_roll if the MSB
of EEPROM bit 28 is set.

gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];                 //Set gyro_pitch to the
correct axis that was stored in the EEPROM.
```

```
if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;                    //Invert gyro_pitch if the
MSB of EEPROM bit 29 is set.

gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];                     //Set gyro_yaw to the
correct axis that was stored in the EEPROM.

if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;                        //Invert gyro_yaw if the
MSB of EEPROM bit 30 is set.

acc_x = acc_axis[eeprom_data[29] & 0b00000011];                        //Set acc_x to the correct
axis that was stored in the EEPROM.

if(eeprom_data[29] & 0b10000000)acc_x *= -1;                      //Invert acc_x if the MSB of
EEPROM bit 29 is set.

acc_y = acc_axis[eeprom_data[28] & 0b00000011];                        //Set acc_y to the correct
axis that was stored in the EEPROM.

if(eeprom_data[28] & 0b10000000)acc_y *= -1;                      //Invert acc_y if the MSB of
EEPROM bit 28 is set.

acc_z = acc_axis[eeprom_data[30] & 0b00000011];                        //Set acc_z to the correct
axis that was stored in the EEPROM.

if(eeprom_data[30] & 0b10000000)acc_z *= -1;                      //Invert acc_z if the MSB of
EEPROM bit 30 is set.

}

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

//Subroutine for calculating pid outputs

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

//The PID controllers are explained in part 5 of the YMFC-3D video session:

//https://youtu.be/JBvnB0279-Q

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

```
void calculate_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;

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;
```

```
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;
}
//This part converts the actual receiver signals to a standardized 1000 – 1500 – 2000 microsecond value.
//The stored data in the EEPROM is used.
int convert_receiver_channel(byte function)
{
byte channel, reverse;                                  //First we declare some local variables
int low, center, high, actual;
int difference;


channel = eeprom_data[function + 23] & 0b00000111;                   //What channel corresponds with the specific function
if(eeprom_data[function + 23] & 0b10000000)reverse = 1;          //Reverse channel when most significant bit is set
else reverse = 0;                                       //If the most significant is not set there is no reverse
```

```
actual = receiver_input[channel];                    //Read the actual receiver value for
the corresponding function

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

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

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

if(actual < center){                           //The actual receiver value is lower than the
center value

if(actual < low)actual = low;                   //Limit the lowest value to the value that
was detected during setup

difference = ((long)(center - actual) * (long)500) / (center - low);     //Calculate and scale the
actual value to a 1000 - 2000us value

if(reverse == 1)return 1500 + difference;                   //If the channel is reversed

 else return 1500 - difference;                                   //If the channel is not
reversed

  }

else if(actual > center){                                    //The actual receiver value
is higher than the center value

if(actual > high)actual = high;                    //Limit the lowest value to the value
that was detected during setup

difference = ((long)(actual - center) * (long)500) / (high - center);     //Calculate and scale the
actual value to a 1000 - 2000us value

if(reverse == 1)return 1500 - difference;               //If the channel is reversed

else return 1500 + difference;                      //If the channel is not reversed
```

```
  }
else return 1500;
}
```

Drone Code MSU.c

Displaying Drone Code MSU.c.