DESIGN AND IMPLEMENTATION OF THE CLOSED LOOP CONTROL OF A QUAD ROTOR UAV FOR STABILITY

A PROJECT REPORT

submitted by

CB107EE103	ADITYA SREEKUMAR
CB107EE118	P. HITHESAN
CB107EE119	M. KRISHNA ANAND

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

AMRITA SCHOOL OF ENGINEERING, COIMBATORE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE 641 105

May 2011

ACKNOWLEDGMENT

First and foremost, we would like to thank our beloved **AMMA** – whose divine help and support has been the backbone of our project.

We express our sincere gratitude to our guide, **Prof. V. Radhamani Pillay**, Professor, Department of Electrical and Electronics Engineering, for her continuous support and guidance throughout the course of the project.

We also express our heartfelt gratitude to **Dr. J. Chandrasekhar**, Chairman, Department of Aerospace Engineering for guiding us throughout the project and being a constant source of motivation.

We thank **Dr. M. P. Chandrasekharan**, Dean – Engineering, for his support and trust, which have been a deciding factor in the completion of this project.

We thank **Dr. T. N. P. Nambiar**, Chairman, Department of Electrical and Electronics Engineering and **Dr. K. K. Sasi**, Vice – Chairman, Department of Electrical and Electronics Engineering for giving us the opportunity and motivation to proceed with our project.

We would also like to thank **Mr. Sarath S. Nair**, Assistant Professor, Department of Electrical and Electronics Engineering, for his support and valuable technical advice which threw light on various aspects of our project.

Our gratitude also extends to all the staff of the Embedded Systems Lab, Power Electronics Lab, Aerodynamics Lab and the Special Machines Lab especially **Mr. Sudarshan, Mr. Sidhan** and **Mr. Sreeprasad** who have helped us in overcoming various technical aspects of our project.

We would also like to thank our project group mates from the Aerospace Engineering Department - **Akhil Mulloth**, **Arun Gopal**, **Arun Vishnu**, **Arjun M. Nambiar** and **Vivek Mohan** without whose help, the completion of the project would have not been a reality.

We finally wish to thank our families for their constant support and encouragement which have been a motivating factor throughout the project. Our heartfelt gratitude also extend to all our friends who have stood by us and helped us through our hardships.

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE - 641105

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**Design and Implementation of the Closed Loop Control of a Quad Rotor UAV for Stability**" submitted by

"**Aditya Sreekumar**	**CB107EE103**
P. Hithesan	**CB107EE118**
M. Krishna Anand	**CB107EE119**"

in partial fulfillment of the requirements for the award of the **Degree of Bachelo**r **of Technology** in **ELECTRICAL AND ELECTRONICS ENGINEERING** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Coimbatore.

Ms. V. Radhamani Pillay

Supervisor

Electrical and Electronics Engineering Department

Dr. T. N. P. Nambiar

Head of Department

Electrical and Electronics Engineering Department

This project report was evaluated by us on:-

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

Quad rotor vehicles are gaining prominence as a platform for Unmanned Aerial Vehicles (UAVs) owing to their simplicity in construction, ease of maintenance, ability to hover and their vertical takeoff and landing capabilities (VTOL).Owing to this, they are being widely developed for applications relating to reconnaissance, security, mapping of terrains and buildings, etc. We wish to study and implement the control strategy of a quad rotor UAV with this project. The design is based on a project undertaken by an interdisciplinary group of the 2006 batch of the Amrita School of Engineering. The work carried out is the selection and purchase of components, the quad rotor design and fabrication and the development of the motor control module. We will be analyzing the implemented motor control for any discrepancies and rectifying them. Following this, we will go on to design and implement the closed loop control strategy, while simultaneously simulating it. Our aim is to understand the complexities involved in such a control and to realize a closed loop control scheme for it.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Symbol	Description
F_i	Force due to i^{th} rotor
I_{xx}	Moment of inertia about x axis
I_{yy}	Moment of inertia about y axis
I_{zz}	Moment of inertia about z axis
J_r	Rotor Moment of inertia
PWM_i	Pulse Width Modulation signal corresponding to i^{th} motor
U_1	Net rotor thrust in vertical direction
U_1^*	Effective force in vertical direction
U_2	Component of force responsible for pure roll
U_3	Component of force responsible for pure pitch
U_4	Component of force responsible for pure yaw
a_i	x intercept of force vs pwm graph corresponding to i^{th} motor
b_i	Slope of force vs pwm graph corresponding to i^{th} motor
b_{cw}	Thrust coefficient of rotor rotating in clock wise direction
b_{aw}	Thrust coefficient of rotor rotating in anti - clock wise direction
b	Thrust coefficient of rotor
d_{cw}	Drag coefficient of rotor rotating in clock wise direction
d_{aw}	Drag coefficient of rotor rotating in anti - clock wise direction
$e_{altitude}$	Altitude error
e_{roll}	Roll error
e_{pitch}	Pitch error
e_{yaw}	Yaw error
g	Acceleration due to gravity
l	Arm length
$k_{p_{alt}}$	Proportional constant for altitude
$k_{p_{roll}}$	Proportional constant for roll
$k_{p_{pitch}}$	Proportional constant for pitch
$k_{p_{yaw}}$	Proportional constant for yaw
$k_{i_{alt}}$	Integral constant for altitude
$k_{i_{roll}}$	Integral constant for roll
$k_{i_{pitch}}$	Integral constant for pitch
$k_{i_{yaw}}$	Integral constant for yaw
$k_{d_{alt}}$	Derivative constant for altitude
$k_{d_{roll}}$	Derivative constant for roll
$k_{d_{pitch}}$	Derivative constant for pitch
$k_{d_{yaw}}$	Derivative constant for yaw
m	Mass
u_1	Actuation signal from altitude controller
u_2	Actuation signal from roll controller
u_3	Actuation signal from pitch controller
u_4	Actuation signal from yaw controller

$\ddot{x}$	Acceleration in the x direction
$\ddot{y}$	Acceleration in the y direction
$\ddot{z}$	Acceleration in the z direction
ϕ	Angular movement about the x axis, Roll
θ	Angular movement about the y axis, Pitch
ψ	Angular movement about the x axis, Yaw
τ_i	Torque due to the i^{th} motor
Ω_i	RPM of the i^{th} motor
a_x	Acceleration in x axis
a_y	Acceleration in y axis
a_z	Acceleration in z axis

CHAPTER 1. INTRODUCTION

1.1 Overview

The quad rotor is an aerial vehicle whose motion is based on the speed of four motors. Due to its ease of maintenance, high maneuverability, vertical takeoff and landing capabilities (VTOL), etc., it is being increasingly used. The constraint with the quad rotor is the high degree of control required for maintaining the stability of the system.

It is an inherently unstable system. There are six degrees of freedom – translational and rotational parameters. These are being controlled by 4 actuating signals. The x and y axis translational motion are coupled with the roll and pitch. Thus we need to constantly monitor the state of the system, and give appropriate control signals to the motors. The variation in speeds of the motors based on these signals will help stabilize the system.

The thrust produced by the motors should lift the quad rotor structure, the motors themselves and the electronic components associated with quad rotor control. An optimum quad rotor design using light and strong materials can help reduce the weight of the quad rotor. This will be one of the challenges faced during the course of the project. The hardware assembly should also be as accurate as possible to avoid any vibrations which will affect the sensors. This will make the control system perform more effectively.

The complexity in the control system of the quad rotor is accounted for in the minimal mechanical complexity of the system. Less maintenance is required as well. As 4 small rotors are being used instead of one big rotor, there is less kinetic energy and thus, less damage in case of accidents. There is also no need of rotor shaft tilting.

With these advantages, the quad rotor is an optimum platform for unmanned aerial vehicles.

1.2 Objectives

The primary objective of the project is to develop a control system for the quad rotor to maintain its stability for conditions including hovering, and basic translational motion.

The goals of the project can be listed as:-

1. Construction of the quad rotor structure
2. Simulation of the control system using Matlab Simulink
3. Implement open loop control
4. Achieve stable hovering condition
5. Perform basic translational motion while maintaining stability

CHAPTER 2. LITERATURE REVIEW

2.1 Background

The theory studied about the quad rotor will be discussed in this chapter.

2.1.1 Aerial Vehicles

Aerial Vehicles can be broadly classified into Rotary wing and Fixed Wing aircrafts. A rotorcraft or rotary wing aircraft is a heavier-than-air flying machine that uses lift generated by wings, called rotor blades, which revolve around a mast. Helicopters, gyro dynes and auto gyros are examples of rotorcrafts. A fixed-wing aircraft, typically called an aero plane, airplane or simply plane, is an aircraft capable of flight using forward motion that generates lift as the wing moves through the air. Fighter jets and gliders are examples of fixed wing aircrafts.

2.1.2 **Unmanned Aerial Vehicles**

A UAV is defined as a powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload [19].

UAVs have far reaching applications. A few examples are:-

- Target and decoy – providing ground and aerial gunnery a target that simulates an enemy aircraft or missile
- Reconnaissance – providing battlefield intelligence
- Combat – providing attack capability for high-risk missions
- Logistics – UAVs specifically designed for cargo and logistics operation
- Research and development – used to further develop UAV technologies to be integrated into field deployed UAV aircraft
- Civil and Commercial UAVs – UAVs specifically designed for civil and commercial applications.

2.1.3 Quad Rotor UAVs

The quad rotor is an aerial vehicle whose motion is based on the speed of four motors. Due to its ease of maintenance, high maneuverability, vertical takeoff and landing capabilities (VTOL), etc., it is being increasingly used. The constraint with the quad rotor is the high degree of control required for maintaining the stability of the system.

It is an inherently unstable system. There are six degrees of freedom – translational and rotational parameters. These are being controlled by 4 actuating signals. The x and y axis translational motion are coupled with the roll and pitch. Thus we need to constantly monitor the state of the system, and give appropriate control signals to the motors. The variation in speeds of the motors based on these signals will help stabilize the system.

The thrust produced by the motors should lift the quad rotor structure, the motors themselves and the electronic components associated with quad rotor control. An optimum quad rotor design using light and strong materials can help reduce the weight of the quad rotor. This will be one of the challenges faced during the course of the project. The hardware assembly should also be as accurate as possible to avoid any vibrations which will affect the sensors. This will make the control system perform more effectively.

The complexity in the control system of the quad rotor is accounted for by the minimal mechanical complexity of the system. Less maintenance is required as well. As 4 small rotors are being used instead of one big rotor, there is less kinetic energy and thus, less damage in case of accidents. There is also no need of rotor shaft tilting.

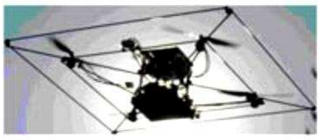
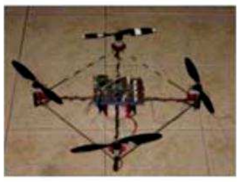
Due to these advantages, the quad rotor is considered as an optimum platform for unmanned aerial vehicles.

2.2 Literature

In 1907, the Breguet Brothers constructed the first quad-rotor named Gyroplane [1]. The flight was a good work to show the principle of a quad rotor. In 1922, Georges de Bothezat built a quad rotor with a rotor located at each end of a truss structure of intersecting beams, placed in the shape of a cross. Experimental aircrafts X-19 and Bell X-22A are also designed as quad-tilt rotor aircrafts. In time due to the tremendous improvements in manufacturing techniques and innovations in metallurgical material knowledge more precise and smaller sensors can now be produced. The Micro-electromechanical Systems (MEMs) technology now allows the production of machine components such as gears with sizes in 10^{-6} meter range. Using this MEMS technology very small accelerometers, gyros and magnetometers are also produced, which caused the production of smaller strap down inertial navigation systems.

Table 2.1 shows a comparative study of different control strategies employed in various quad rotors. The information is obtained from literature [24].

Table 2.1 : Comparison of Control Strategies in different quad rotors

Project	Control Technique	Picture
STARMAC, Stanford University, 2005 Waslander et al. (2005)	Reinforcement Lrn.	
OS4, EPFL, December 2006 Bouabdallah (2007)	Backstepping	
Pennsylvania State University, Hanford, 2005 Hanford (2005)	PI	
Helio-copter, Brigham Young University, Fowers, 2008 Fowers (2008)	Visual feedback	
HMX-4, Pennsylvania State University, 2002 Altug et al. (2002)	Feedback Lin.	
Quad-Rotor UAV, University of British Columbia Chen and Huzmezan (2003)	MBPC and H∞	
Quad-Rotor Flying Robot, Universiti Teknologi Malaysia Weng and Shukri (2006)	PID	

As a result of this improvement in technology very small quad rotors are developed around the world such as Mesicopter (Figure 2.1) and Hoverbot. There is also commercially available quad rotor named DraganFlyer (Figure 2.2). Recently, there are several different studies in the literature about quad rotors. These works utilized different controllers, equipments and materials.

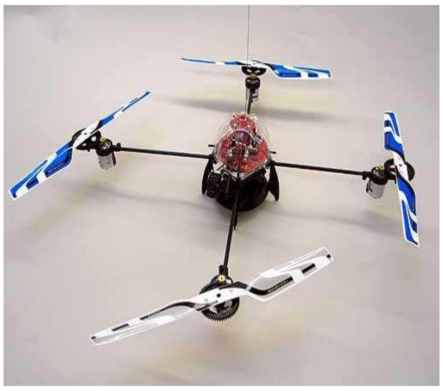


Fig 2.1 : Mesicopter

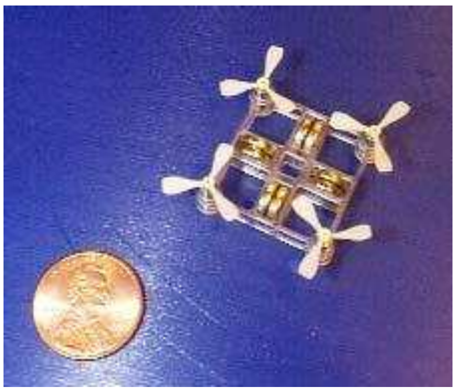


Fig 2.2 : Dragan Flyer

In Pennsylvania State university two different studies had been done on quad rotors [3], [4]. First is a master thesis (Figure 2.3) that had been done about a quad rotor test bench. The inertial measurement unit consists of three gyros from Analog Devices (ADXRS150EB), one accelerometer (ADXL210EB). Attitude of the quad rotor is controlled with PI control law.

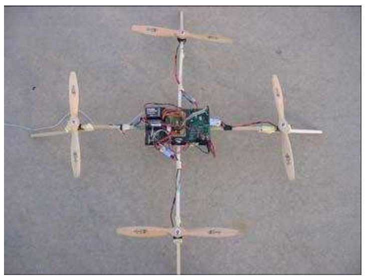


Fig 2.3 : Quad rotor designed at Pennsylvania State University

Second work done in university of Pennsylvania utilizes DraganFlyer as test bed. It has external pan-tilt ground and on-board cameras in addition to the three onboard gyroscopes. One camera placed on the ground captures the motion of five 2.5 cm colored markers present underneath the DraganFlyer, to obtain pitch, roll and yaw angles and the position of the quad rotor by utilizing a tracking algorithm and a conversion routine. In other words, two-camera method has been introduced for estimating the full six degrees of freedom (DOF) pose of the helicopter. Algorithm routines ran in an off board computer. Due to the weight limitations, GPS or other accelerometers could not be added on the system. The controller obtained the relative positions and velocities from the cameras only. Two methods of control are studied – one using a series of mode-based, feedback linearizing controllers and the other using a back-stepping control law. The helicopter was restricted with a tether to vertical, yaw motions and limited x and y translations. Simulations performed in MATLAB-Simulink show the ability of the controller to perform output tracking control even when there are errors on state estimates.

The X4-Flyer developed in ANU consists of a HC-12 a single board computer, developed at QCAT that was used as the signal conditioning system [6]. This card uses two HC-12 processors and outputs PWM signals that control the speed drivers directly, inputs PWM signals from an R700 JR Slimline RC receiver allowing direct plot input from a JP 3810 radio transmitter and has two separate RS232 serial channels, the first used to interface with the inertial measurement unit (IMU) and second used as an asynchronous data linked to the ground based computer. As an IMU, the most suitable unit considered was the EiMU embedded inertial measurement unit developed by the robotics group in QCAT, CSIRO weighs 50-100g. Crossbow DMU-6 is also used in the prototype. It weighs 475g. The rotor used is an 11’’ diameter APC-C2 2.9:1 gear system included 6’’ per revolution pitch with a maximum trust of 700grams. The motors are Johnson 683 500 series motors. The speed controller used is MSC30 B with a weight 26g rated 30A at 12V The pilot augmentation control system is used. A double lead compensator is used for the inner loop. The final setup is shown in Figure 2.4.

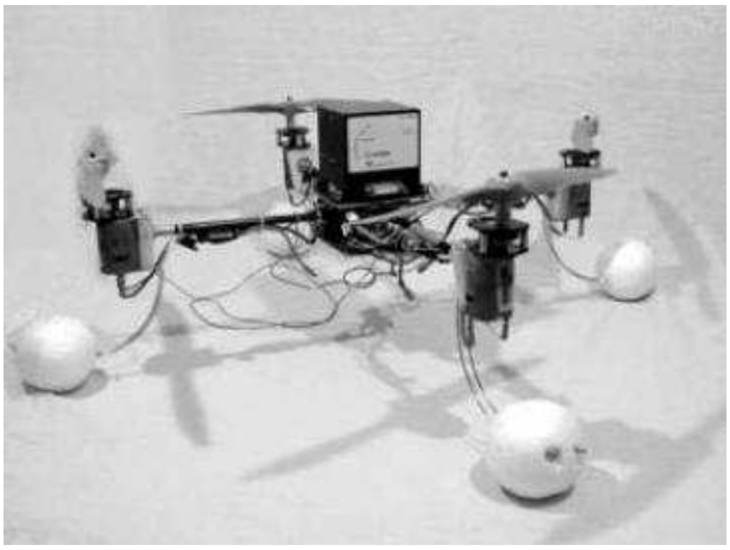


Fig 2.4 : The X4 Flyer developed in FEIT, ANU

The name of the project that is worked on in Stanford University is called STARMAC [8]. STARMAC consists of four X4-flyer rotorcraft that can autonomously track a given waypoint trajectory. This trajectory generated by novel trajectory planning algorithms for multi agent systems. STARMAC project aims a system fully capable of reliable autonomous waypoint tracking, making it a useful test bed for higher level algorithms addressing multiple-vehicle coordination.

The base system is the off-the-shelf four-rotor helicopter called the DraganFlyer III. The open-loop system is unstable and has a natural frequency of 60 Hz, making it almost impossible for humans to fly. An existing onboard controller slows down the system dynamics to about 5 Hz and adds damping, making it pilotable by humans. It tracks commands for the three angular rates and thrust. An upgrade to Lithium-polymer batteries has increased both payload and flight duration, and has greatly enhanced the abilities of the system. For attitude measurement, an off-the-shelf 3-D motion sensor developed by Microstrain, the 3DM-G was used. This all in one IMU provides gyro stabilized attitude state information at a remarkable 50 Hz. For position and velocity measurement, Trimble Lassen LP GPS receiver was used. To improve altitude information, a downward-pointing sonic ranger (Sodar) by Acroname was used, especially for critical tasks such as takeoff and landing. The Sodar has a sampling rate of 10 Hz, a range of 6 feet, and an accuracy of a few centimeters, while the GPS computes positions at 1 Hz, and has a differential accuracy of about 0.5 m in the horizontal direction and 1 m in the vertical. To obtain such accuracies, DGPS planned be implemented by setting up a ground station that both receives GPS signals and broadcasts differential correction information to the flyers.

All of the onboard sensing is coordinated through two Microchip 40 MHz PIC microcontrollers programmed in C. Attitude stabilization were performed on board at 50 Hz, and any information was relayed upon request to a central base station on the ground. Communication is via a Bluetooth Class II device that has a range of over 14 150 ft. The device operates in the 2.4 GHz frequency range, and incorporates band hopping, error correction and automatic retransmission. It is designed as a serial cable replacement and therefore operates at a maximum bandwidth of 115.2 kbps. The communication scheme incorporates polling and sequential transmissions, so that all flyers and the ground station simultaneously operate on the same communication link. Therefore, the bandwidth of 115.2 kbps is divided among all flyers. The base station on the ground performs differential GPS and waypoint tracking tasks for all four flyers, and sends commanded attitude values to the flyers for position control. Manual flight is performed via standard joystick input to the ground station laptop. Waypoint control of the flyers was performed using Labview on the ground station due to its ease of use and on the fly modification ability. Control loops have been implemented using simple PD controllers. The system while hovering is shown in Figure 2.5.

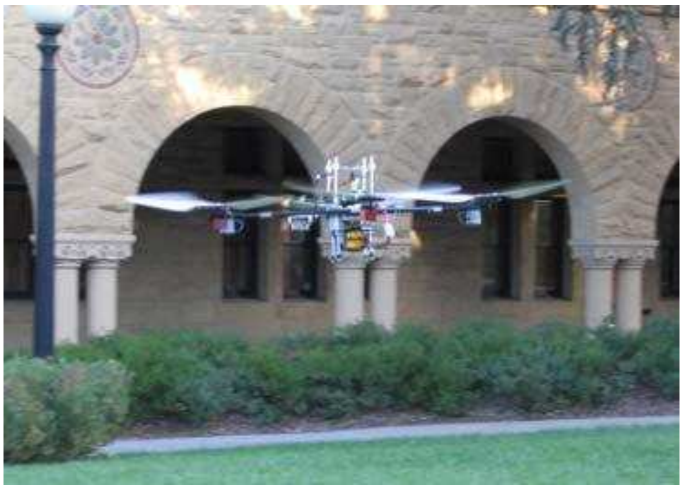


Fig 2.5 : Quad rotor designed in Stanford University

2.3 Findings

The data obtained from the literature regarding the operation of the quad rotor are summarized below. Further details are presented in the modules in the subsequent chapters.

2.3.1 Structure

A Quad rotor is an aerial vehicle that generates lift with four rotors. The craft is controlled by varying the rpm and not by using any mechanical actuators like in a helicopter. This makes it particularly suitable for UAVs. The craft requires active control of six degrees of freedom to fly and is inherently unstable. The layout of a quad rotor is shown in the figure.

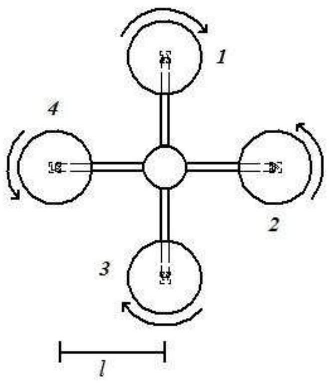


Fig 2.6 : Quad Rotor Design

The Quad Rotor layout is shown Figure 2.6. There are two arms, each having motors at its ends. The motors 1 and 3, which are mounted on the same arm, rotate in the clockwise direction while the motors 2 and 4, mounted on the second arm, rotate in the anti-clockwise arrangement. Both motors at opposite ends of the same arm should rotate in same direction to prevent torque imbalance during linear flight.

2.3.2 Quad Rotor Dynamics

Generally, the quad rotor can be modeled with four rotors in cross configuration. The whole model can be considered as a rigid body. Figure 2.7 illustrates the basic motion control of the quad rotor. In this section, the thrust produced is proportional to the thickness of the arrows in the figures.

Two of the motors rotate in the clockwise direction while the other two rotate in the anticlockwise direction. The reference axes are represented in the figure.

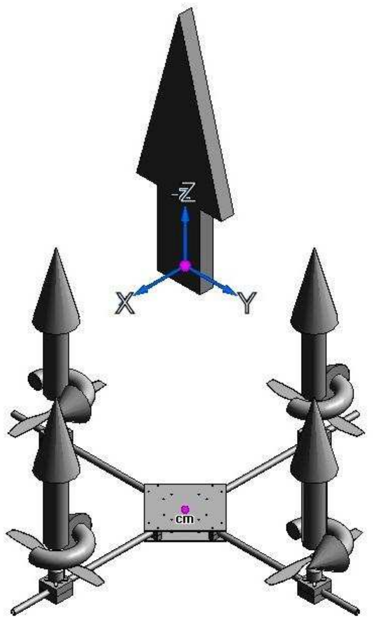


Fig 2.7 : Quad Rotor Control Mechanism

2.3.2.1 Altitude Motion

The throttle movement is provided by increasing (or decreasing) the speed of all the rotors by the same amount. It leads a vertical force u_1 $[N]$ with respect to body-fixed frame which raises or lowers the quad rotor.

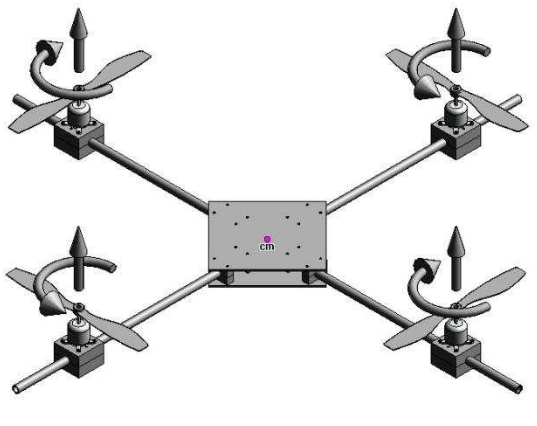


Fig 2.8 : Decreasing Altitude

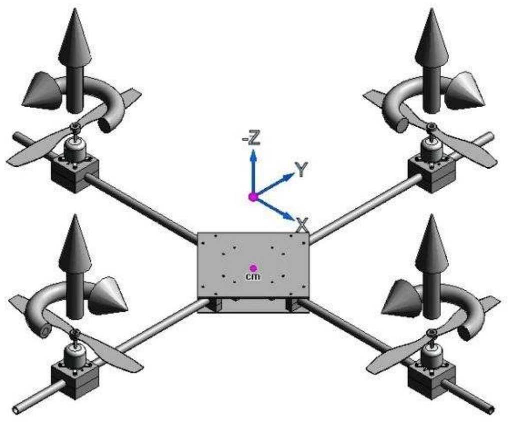


Fig 2.9 : Hovering Condition

2.3.2.2 Roll Motion

The roll movement is provided by increasing (or decreasing) the left rotor's speed and at the same time decreasing (or increasing) the right rotor's speed. It leads to a torque with respect to the central axis as shown in Fig 2.10 which makes the quad rotor roll. The overall vertical thrust is the same as in hovering.

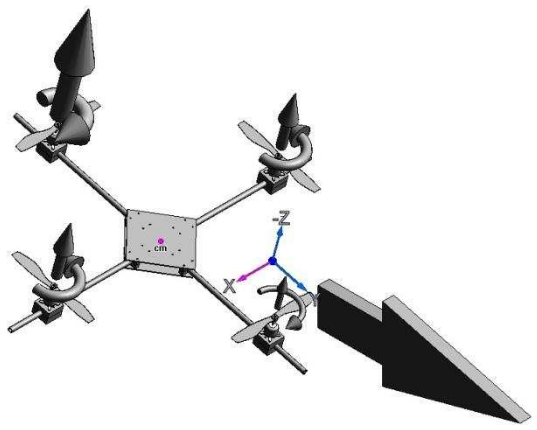


Fig 2.10 : Roll Motion

2.3.2.3 Pitch Motion

The pitch movement is provided by increasing (or decreasing) the front rotor's speed and at the same time decreasing (or increasing) the back rotor's speed. It leads to a torque with respect to the central axis. The overall vertical thrust is the same as in hovering.

2.3.2.4 Yaw Motion

The yaw movement is provided by increasing (or decreasing) the front-rear rotors' speed and at the same time decreasing (or increasing) the left-right couple. It leads to a torque which makes the quad rotor turn in horizon level. The overall vertical thrust is the same as in hovering.

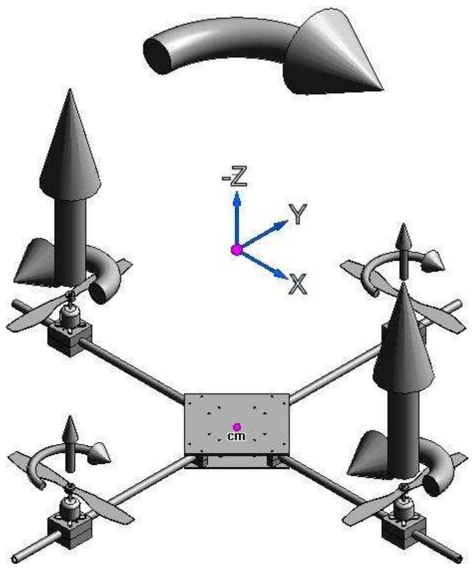


Fig 2.11 : Yaw Motion

2.3.2.5 Conclusion

The dynamics of the quad rotor was thus obtained from the literature available and modules required for completion of the project were listed out.

CHAPTER 3. BACKGROUND STUDY

3.1 Modules

The project can be split into different modules. These include the simulation modules to the hardware modules. They are:-

1. Mathematical analysis of the Quad Rotor UAV
2. Quad Rotor Simulation
3. Motor Control using Electronic Speed Controller
4. Arduino – Simulink Interfacing
5. Control System Implementation

3.2 Mathematical analysis of the quad rotor UAV

The quad rotor has six degrees of freedom that can be divided into two parts:

- Translational – translational motion occurs in x, y, and z directions.
- Rotational – rotational motion occurs about x, y and z directions and named as roll (ϕ), pitch (θ), and yaw (ψ).

The frames of reference used are:

- Earth's frame (an inertial frame of reference)
- Body-axis (a non-inertial frame of reference)

The model derived is based on the following assumptions:

- The structure is rigid and symmetrical
- The center of gravity and the body fixed frame origin coincide
- The propellers are rigid
- Thrust is directly proportional to the square of the propeller's speed

3.2.1 Translational Motion:

In order to derive the equations representing the translation motion, the basic equation used is:

$$\sum F = Ma$$

Where, $\sum F$ represents the sum of all the external forces on the system.

M is the total mass of the system.

a is the total acceleration vector of the system.

The forces acting on the system are:

1. Thrust due to four propellers
2. Force of gravity

The equations obtained are:

$$ma_z = mg - (\cos\theta\cos\phi)U_1 \quad \text{-----------} \; (3.1)$$

$$ma_x = (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)U_1 \quad \text{-----------} \; (3.2)$$

$$ma_y = (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)U_1 \quad \text{-----------} \; (3.3)$$

3.2.2 Rotational Motion:

In order to derive the equations representing the rotational motion, the basic equation used is:

$$\dot{h} = I\alpha = I\dot{\omega}$$

The equations obtained are:

$$I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) + \mathrm{J}_r\Omega_r\dot{\theta} + lU_2 \quad \text{-----------} \; (3.4)$$

$$I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - \mathrm{J}_r\Omega_r\dot{\phi} + lU_3 \quad \text{-----------} \; (3.5)$$

$$I_{zz}\ddot{\psi} = \dot{\theta}\dot{\phi}(I_{xx} - I_{yy}) + \mathrm{J}_r\dot{\Omega}_r \quad \text{-----------} \; (3.6)$$

3.2.3 Multiple Input Multiple Output (MIMO) Approach

The above 6 equations which represent the fundamental equations of the Quadrotor are coupled. To perform control the Quadrotor either a MIMO approach or a SISO approach can be made. For MIMO approach the following procedure can be followed [9]

3.2.3.1 State Space Equations:

Now, in order to study the stability of the system the system is written in the form: $\dot{X} = AX + BU$

$Y = CX + DU$

3.2.3.2 Linearization:

The system of equations 2.1 to 2.6 are non-linear coupled equations. Therefore, the system of equations was linearized using 'small perturbation theorem'.

The states of the system are:

X= [u v w p q r] ^{T}

Inputs to the system are:

U= $[U_1 U_2 U_3 \Omega_r \dot{\Omega}_r]^T$

The perturbation was given only to the states (velocities).

The new perturbation technique was adopted by introducing the disturbance in positions (ϕ, θ, and ψ) and acceleration due to the disturbance in velocity.

Equations 3.1 to 3.3 represent translational motion and equation 3.4 to 3.6 represent rotational motion. For simplification purpose, the following constants were assumed in the equations:

$\frac{I_{yy}-I_{zz}}{I_{xx}} = b_1$ ----------- (3.7)

$\frac{I_{zz}-I_{xx}}{I_{yy}} = b_2$ ----------- (3.8)

$\frac{I_{xx}-I_{yy}}{I_{zz}} = b_3$ ----------- (3.9)

$\frac{J_r}{I_{xx}} = c_1$ ----------- (3.10)

$\frac{J_r}{I_{yy}} = c_2 \frac{J_r}{I_{zz}} = c_3 \frac{l}{I_{xx}} = d_1$ ----------- (3.11)

$\frac{l}{I_{yy}} = d_2$ ----------- (3.12)

Perturbation is marked by the suffix 'e'. Hence, perturbation in w is written as

$\frac{d}{dt}(w_0 + w_e) = g - [\cos(\theta + \theta_e)\cos(\phi + \phi_e)]\frac{U_1}{m}$ ----------- (3.13)

$\dot{w}_0 + \dot{w}_e = g - [(cos\theta - \theta_e sin\theta)(cos\phi - \phi_e sin\phi)]\frac{U_1}{m}$ ----------- (3.14)

$\dot{w}_0 + \dot{w}_e = g - [cos\theta cos\phi - \theta_e sin\theta cos\phi - \phi_e sin\phi cos\theta]\ \frac{U_1}{m}$ ----------- (3.15)

Since

$\dot{w}_o = g - (cos\theta cos\phi)\frac{U_1}{m}$ ----------- (3.16)

$\dot{w}_e = (\theta_e sin\theta cos\phi + \phi_e sin\phi cos\theta)\frac{U_1}{m}$ ----------- (3.17)

$\dot{w}_e = \theta_e a_1 w + \phi_e a_2 w$ ----------- (3.18)

Where $a_{1w} = sin\theta cos\phi$; $a_{2w} = sin\phi cos\theta$

Perturbation in u:

$\dot{u} = (cos\phi sin\theta cos\psi + sin\phi sin\psi)\frac{U_1}{m}$ ----------- (3.19)

$$u_0^{\cdot} + \dot{u}_e = [(cos\phi - \phi_e sin\phi)(sin\theta + \theta_e cos\theta)(cos\psi - \psi_e sinc\psi) + (sin\phi + \phi_e cos\phi)\big(sin\psi + \psi_{ecos\psi}\big)]\frac{U_1}{m}$$

$$= [(cos\phi sin\theta cos\psi + \theta_e cos\theta cos\phi cos\psi - \phi_e sin\theta sin\phi cos\psi - \psi_e sin\psi cos\phi sin\theta) + (sin\phi sin\psi + \phi_e cos\phi sin\psi + \psi_e cos\psi sin\phi)]\frac{U_1}{m}$$

Since, $\dot{u_0} = cos\phi sin\theta cos\psi + sin\phi sin\psi$

$$\dot{u_e} = (\theta_e cos\theta cos\phi cos\psi - \phi_e sin\theta sin\phi cos\psi - \psi_e sin\psi cos\phi sin\theta + \phi_e cos\phi sin\psi + \psi_e cos\psi sin\phi)\frac{U_1}{m}$$

$$\dot{u_e} = [\phi_e(cos\phi sin\psi - sin\theta sin\phi cos\psi) + \theta_e(cos\theta cos\phi cos\psi) + \psi_e(cos\psi sin\phi - sin\psi cos\phi sin\theta)]\frac{U_1}{m}$$

$\dot{u_e} = \theta_e a_{1u} + \phi_e a_{2u} + \psi_e a_{3u}$ ----------- (3.20)

Where, $a_{1u} = cos\theta cos\phi cos\psi$

$a_{2u} = cos\phi sin\psi - sin\theta sin\phi cos\psi$

$a_{3u} = cos\psi sin\phi - sin\psi cos\phi sin\theta$

Perturbation in v:

$\dot{v} = (cos\phi sin\theta sin\psi - sin\phi cos\psi)\frac{U_1}{m}$ ----------- (3.21)

$$\dot{v_0} + \dot{v_e} = [(cos\phi - \phi_e sin\phi)(sin\theta + \theta_e cos\theta)(sin\psi + \psi_e cos\psi) - (sin\phi + \phi_e cos\phi)(cos\psi - \psi_e sin\psi)]\frac{U_1}{m}$$

$$= cos\phi sin\theta sin\psi + \theta_e cos\phi cos\theta sin\psi - \phi_e sin\phi sin\theta sin\psi + \psi_e cos\psi cos\phi sin\theta - sin\phi cos\psi + \psi_e sin\psi sin\phi - \phi_e cos\phi cos\psi]\ \frac{U_1}{m}$$

Since,

$\dot{v_0} = [\theta_e(cos\phi cos\theta sin\psi) - \phi_e(sin\phi sin\theta sin\psi + cos\phi cos\psi) + \psi_e(cos\psi cos\phi sin\theta + sin\psi sin\phi)]\frac{U_1}{m}$

$\dot{v_e} = \theta_e a_{1v} + \phi_e a_{2v} + \psi_e a_{3v}$ ----------- (3.22)

Where, $a_{1v} = cos\phi cos\theta sin\psi$

$a_{2v} = -(sin\phi sin\theta sin\psi + cos\phi cos\psi)$

$a_{3v} = cos\phi cos\psi sin\theta$+sinψsinϕ

Perturbation in rotational velocity p:

$\dot{p} = qr\frac{(I_{yy}-Izz)}{I_{xx}} + \frac{J_r}{I_{xx}}\Omega_r q + \frac{l}{I_{xx}}U_2$ ----------- (3.23)

$$\dot{p_0} + \dot{p_e} = (q+q_e)(r+r_e)b_1 + c_1(q+q_e)\Omega_r + d_1U_2$$

$$= (qr + q_e r + r_e q)b_1 + (c_1 q + c_1 q_e)\Omega_r + d_1 U_2$$

Since, $\dot{p_0} = qrb_1 + c_1\Omega_r q + d_1U_2$

$\dot{p_e} = q_e r b_1 + q_e c_1 \Omega_r + r_e q b_1$ ----------- (3.24)

Perturbation in q:

$\dot{q} = pr\frac{I_{zz}-I_{xx}}{I_{yy}} - \frac{J_r}{I_{yy}}p\Omega_r + \frac{l}{I_{yy}}U_3$ ----------- (3.25)

$$\dot{q_0} + \dot{q_e} = (p+p_e)(r+r_e)b_2 - c_2(p+p_e)\Omega_r + d_2U_3$$

$$= (pr + p_e r + r_e p)b_2 - c_2(p+p_e)\Omega_r + d_2U_3$$

Since, $\dot{q_0} = prb_2 - p\Omega_r c_2 + d_2U_3$

$\dot{q_e} = p_e r b_2 + r_e p b_2 - c_2 p_e \Omega_r$ ----------- (3.26)

$$\dot{r} = qp\frac{I_{xx} - I_{yy}}{I_{zz}} + \frac{J_r}{I_{zz}}\dot{\Omega_r}$$

$$\dot{r_0} + \dot{r_e} = (q+q_e)(p+p_e)b_3 + c_3\dot{\Omega_r}$$

$$= (pq + q_e p + p_e q)b_3 + c_3\dot{\Omega_r}$$

Since, $\dot{r_0} = qpb_3 + c_3\dot{\Omega_r}$

$\dot{r_e} = q_e p + p_e q$ ----------- (3.27)

The state space equation was thus formed as.

$$\dot{X} = AX + BU$$

The corresponding Eigen values of the A matrix was also found.

I_{xx}, I_{yy} and I_{zz} were used to obtain the matrices A and B.

3.2.4 Single Input Single Output (SISO) approach

The fundamental equations are decoupled based on certain assumptions as follows [19]:-

- The quad rotor is assumed to have very low linear and angular velocities when in motion and assumed not to tilt beyond 15^o in pitch and roll. The quad rotor is always flying at near hovering conditions and Coriolis and rotor moment of inertia terms can be neglected.
- Attitude is controlled by manipulating the four degrees of freedom involved – viz. altitude, roll, pitch and yaw.
- The equations representing the four degrees of freedom are:-

$$\ddot{z} = -g + (\cos(\phi)\cos(\theta))\frac{U_1}{m}$$

$$\ddot{\phi} = \frac{l}{I_{xx}} U_2$$

$$\ddot{\theta} = \frac{l}{I_{yy}} U_3$$

$$\ddot{\psi} = \frac{1}{I_{yy}} U_4$$

3.3 Quad rotor simulation

The Quad Rotor model can be represented using MATLAB SIMULINK. This implementation will allow us to identify the state of the quad rotor when we give any disturbance or input. We can split the quad rotor into separate segments and integrate it to obtain the overall view.

3.3.1 Implementation

The whole simulation has been divided into different subsystems which have been specified as follows:

1. Reference model creator block: Block to obtain the values of position in translational axes(x, y, z) and rotational axes (roll, pitch and yaw).
2. Controller Block: Block used to get the desired translational and rotational parameters and produce the control inputs in altitude, roll, pitch and yaw.
3. PWM controller Block: Block used for obtaining the PWM input for the four motors from the four control inputs.
4. Quad Rotor Dynamics Block: Block implementing the mathematical model of the quad rotor. It takes the thrust of the four motors and obtains the change in position of the quad rotor system with the change in the thrusts.
5. IMU Block: Block used to implement the 3 axis gyroscope and 3 axis accelerometer. The accelerations in various rotational and translational parameters are measured and given as feedback to the system.

The following blocks were linked together to obtain the state of the quad rotor with change in inputs.

3.4 Motor control using Electronic Speed Controller

This chapter deals with the Electronic Speed Controller used to control the speed of the BLDC motor.

3.4.1 Introduction

For our purpose, a method which is less complex, occupies minimal space and is simpler to control is required. An Electronic Speed Controller (ESC) is thus adopted for controlling the speed of the motor.

The specifications of the motors and ESCs are :-

S. No	Component	Quantity
1.	Topband Out runner Brushless Motor 11.1V 1400Kv For Airplane (AE2806-01B)	4
2	Dual-sky XC2512BA Speed Controller	4
3	1600mAh 3cell, Hyperion Gen-3 25C/45C Packs - 40A Continuous	4

3.4.2 Working of Electronic Speed Controller

An electronic speed control or ESC is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake.

For a BLDC motor, the ESC performs the function of the motor driver and also causes the commutation. When a PWM signal is fed to the ESC, it converts this PWM signal to a 3 phase PWM of the supply voltage. We are thus able to vary the supply voltage to the BLDC motor by varying the PWM input to the ESC, in turn controlling the speed of the motor.

3.4.3 Implementation of the ESC for BLDC motor control

Using a PIC16F877A microcontroller, we generated a set of 16 PWM signals which could be fed to the ESC. The signal could be changed by a switching arrangement consisting of 4 input pins to the PIC. We were thus able to vary the speed of the BLDC motor between minimum and maximum throttle values and plot the duty ratio v/s speed graph for the BLDC motor. The results are shown in the later chapter.

3.5 Arduino – Simulink interfacing

The Arduino processor to Simulink interfacing is discussed in this chapter.

3.5.1 Introduction

The Arduino Mega 2560 board is an Atmel based board which will be the onboard processor for the quad rotor. It is capable of all the processing required for implementing the control system.

We interface the Arduino board with Simulink. This allows us to develop models using Simulink and directly implement it using the Arduino board. We can also connect the two systems while performing the control operation.

3.5.2 Software Required

The software required for interfacing Simulink with Arduino are [21]:-

1. Real Time Workshop
2. Real Time Workshop Embedded Center
3. Matlab 2010a or above
4. Arduino 0022 IDE
5. Microsoft Visual Studio 2008

3.5.3 Conclusion

Using this interface, we can develop our control system using Simulink. This can be loaded onto the Arduino and used directly on our quad rotor.

3.6 Control System Implementation

The Quadrotor stabilization is done by a PID control system. The Quadrotor is autonomous and hence the way points can be programmed in to it. Depending on the subsequent way point and the present position the distance to move in the x, y and z directions in the inertial coordinate system can be determined; this will serve as a reference input for the trajectory control loop. The Quadrotor has an inner attitude stabilization control loop and an outer trajectory control loop. The outer trajectory control loop will generate the reference values for the inner loop.

Hardware wise the Quadrotor has Arduino Mega 2560 and ArduV2 flat IMU. The ArduV2 flat IMU (Inertial measurement unit) is a six degree of freedom IMU. It serves as the feedback. The IMU has an accelerometer, a dual axis gyro, a single axis gyro and an Atmega 328 processor. The Gyroscope gives the angular velocity as output in terms of mV. And the accelerometer gives the linear acceleration as output in terms of mV. The signals can be converted to orientation information by the DCM algorithm explained in the literature [25].

3.6.1 Attitude control loop

The attitude control loop controls roll, pitch, yaw and altitude independently with their respective PID controllers. The PID algorithm will generate the four actuation signals which are then used to calculate the appropriate PWM signals. The ESCs upon receiving the PWM signal controls the RPM of the motor accordingly. The thrust produced by the rotors vary as the RPM varies and the attitude of the Quadrotor changes.

The IMU senses this change in attitude in terms of the sensor outputs. A DCM algorithm is used to calculate the attitude of the Quadrotor based on the feedback signal. The calculated attitude is then sent to the PID controller where it serves the role of error.

3.6.2 Trajectory control loop

The same feedback received by the attitude PID controller is also received by the trajectory PID controller. The trajectory PID controller modifies the reference values going into the attitude control loop to rectify the trajectory error.

3.6.3 Algorithm implementation in hardware

The attitude control loop is vital to the stability of the Quadrotor. The DCM algorithm executes in the ArduV2 IMU board itself. The Quadrotor orientation values are then transmitted to the Arduino Mega Board where the control loops use the values as feedback. PWM signals are generated and sent to the ESCs from the Arduino Mega.

3.6.4 IMU calibration at start up

The first set of values over a period of 1 second is recorded and the mean is calculated. This mean value is taken as the bias and is subsequently subtracted from all the subsequent measurements.

3.6.5 Conclusion

Steps were taken as mentioned above to implement the hardware. The literatures in [26] and [27] were used as reference throughout the coding and have been modified according to the hardware for executing the control.

CHAPTER 4. IMPLEMENTATION

4.1 Quad rotor simulation

We are able to control the speed of the motor using PWM signals as shown earlier. The state of the system when there is a change in input is our next area of concern. The mathematical model of the quad rotor was implemented using Matlab Simulink. Matlab Function Blocks were used for different modules of the simulation.

4.1.1 SIMULINK Model

The following blocks were used to implement the mathematical model:-

4.1.1.1 Reference Model Creator Block (Manual input block):

This block is used to obtain the reference values for rotational parameters - roll, pitch and yaw –and the translational parameter– z, which are manual inputs in the simulation. The block can be modified to accept input from an outer control loop for position in future.

4.1.1.2 Controller Block

This block is used to compare the reference values of the rotational parameters - roll, pitch and yaw - and the translational parameter, z, with the actual values obtained as feedback from the quad rotor's mathematical model. The error obtained from the comparison is given to a PID control block which performs the control action and the necessary control signals are generated. Four PID controllers control the parameters of the system. The outputs from this block are [4]:-

u_1=actuation signal from altitude controller

u_2=actuation signal from roll controller

u_3=actuation signal from pitch controller

u_4=actuation signal from yaw controller

The PID controller equations obtained from Matlab Simulink are:-

$$\frac{u_1(s)}{e_{altitude}} = k_{p_{alt}} + k_{i_{alt}}\frac{1}{s} + k_{d_{alt}}\frac{s}{0.1s+1}$$

$$\frac{u_2(s)}{e_{roll}} = k_{p_{roll}} + k_{i_{roll}}\frac{1}{s} + k_{d_{roll}}\frac{s}{0.1s+1}$$

$$\frac{u_3(s)}{e_{pitch}} = k_{p_{pitch}} + k_{i_{pitch}}\frac{1}{s} + k_{d_{pitch}}\frac{s}{0.1s+1}$$

$$\frac{u_4(s)}{e_{yaw}} = k_{p_{yaw}} + k_{i_{yaw}}\frac{1}{s} + k_{d_{yaw}}\frac{s}{0.1s+1}$$

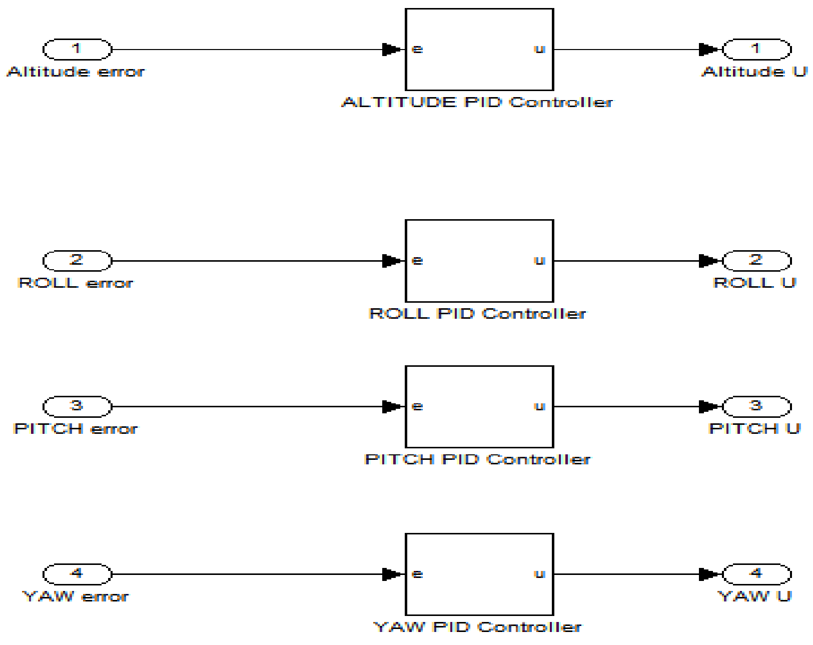


Fig 4.1 : Controller Block

4.1.1.3 PWM Signal Generation Block

The signals from the controller block are given to the PWM Signal Generation block. This block generates the PWM signals to be given to the four motors of the quad rotor. The equations used for this calculation of individual motor thrusts are available in literature and are as follows [4]:-

$$F_1 = \frac{1}{4}u_1 + \frac{1}{2}u_3 + \frac{b}{4d}u_4$$

$$F_2 = \frac{1}{4}u_1 - \frac{1}{2}u_3 + \frac{b}{4d}u_4$$

$$F_3 = \frac{1}{4}u_1 + \frac{1}{2}u_2 - \frac{b}{4d}u_4$$

$$F_4 = \frac{1}{4}u_1 - \frac{1}{2}u_2 - \frac{b}{4d}u_4$$

The thrust factor for the clockwise and anti-clockwise rotating propellers is different, and the equations were modified as:-

$$F_1 = \frac{1}{4}u_1 + \frac{1}{2}u_3 + \frac{b_{cw}}{4dcw}u_4$$

$$F_2 = \frac{1}{4}u_1 - \frac{1}{2}u_3 + \frac{b_{aw}}{4d_{aw}}u_4$$

$$F_3 = \frac{1}{4}u_1 + \frac{1}{2}u_2 - \frac{b_{cw}}{4d_{cw}}u_4$$

$$F_4 = \frac{1}{4}u_1 - \frac{1}{2}u_2 - \frac{b_{aw}}{4d_{aw}}u_4$$

The PWM signal to be generated based on the input to the controller [4]:-

$$PWM_1 = \frac{1}{b_1}(F_1 - a_1)$$

$$PWM_2 = \frac{1}{b_2}(F_2 - a_2)$$

$$PWM_3 = \frac{1}{b_3}(F_3 - a_3)$$

$$PWM_4 = \frac{1}{b_4}(F_4 - a_4)$$

4.1.1.4 Motor Dynamics Block

This block is used for estimating the thrust generated by the motor for a particular input PWM signal. The block uses a first order algebraic equation, and converts the input PWM signal to equivalent force.

The relation between the PWM signal and the thrust developed by the motor is [4]:-

$$F_i = b_i.PWM_i + a_i$$

The output from the block represents the resultant force acting on the system resolved into components responsible for altitude, roll, pitch and yaw. To achieve this equations are obtained from the literature [2], [4], [5], [8], [6]:-

$$U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$$

$$U_2 = b(-\Omega_2^2 + \Omega_4^2)$$

$$U_3 = b(-\Omega_1^2 + \Omega_3^2)$$

$$U_4 = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)$$

The equations were modified for implementation as follows.

The force and torque produced by the motors can be related to their RPM, Ω as:-

Force due to the ith motor $= F_i = b_{cw}\Omega_i^2$

Torque due to the ith motor $= \tau_i = d_{cw}\Omega_i^2$

Where i = 1 or 3 (clockwise motors)

Force due to the ith motor $= F_i = b_{aw}\Omega_i^2$

Torque due to the ith motor $= \tau_i = d_{aw}\Omega_i^2$

Where i = 2 or 4 (anti clockwise motors)

The output from the block is:-

$$U_1 = b_{cw}(\Omega_1^2 + \Omega_3^2) + b_{aw}(\Omega_2^2 + \Omega_4^2)$$
$$U_1^* = U_1 - mg$$
$$U_2 = b_{aw}(-\Omega_2^2 + \Omega_4^2)$$
$$U_3 = b_{cw}(-\Omega_1^2 + \Omega_3^2)$$
$$U_4 = -b_{cw}(\Omega_1^2 + \Omega_3^2) + b_{aw}(\Omega_2^2 + \Omega_4^2)$$

4.1.1.5 Quad Rotor Block

The forces from the motor dynamics block are the inputs to the mathematical model block. This block gives the positional parameters of the system. Three of the decoupled transfer functions that represent roll, pitch and yaw are explained in the literature [4], [6]:-

$$\frac{\phi(s)}{U_2(s)} = \frac{1}{s^2 I_{xx}}$$

$$\frac{\psi(s)}{U_4(s)} = \frac{1}{s^2 I_{zz}}$$

A transfer function for the altitude was introduced. This is:-

$$\frac{z(s)}{U_1^*(s)} = \frac{\cos(\phi)\cos(\theta)}{s^2 m}$$

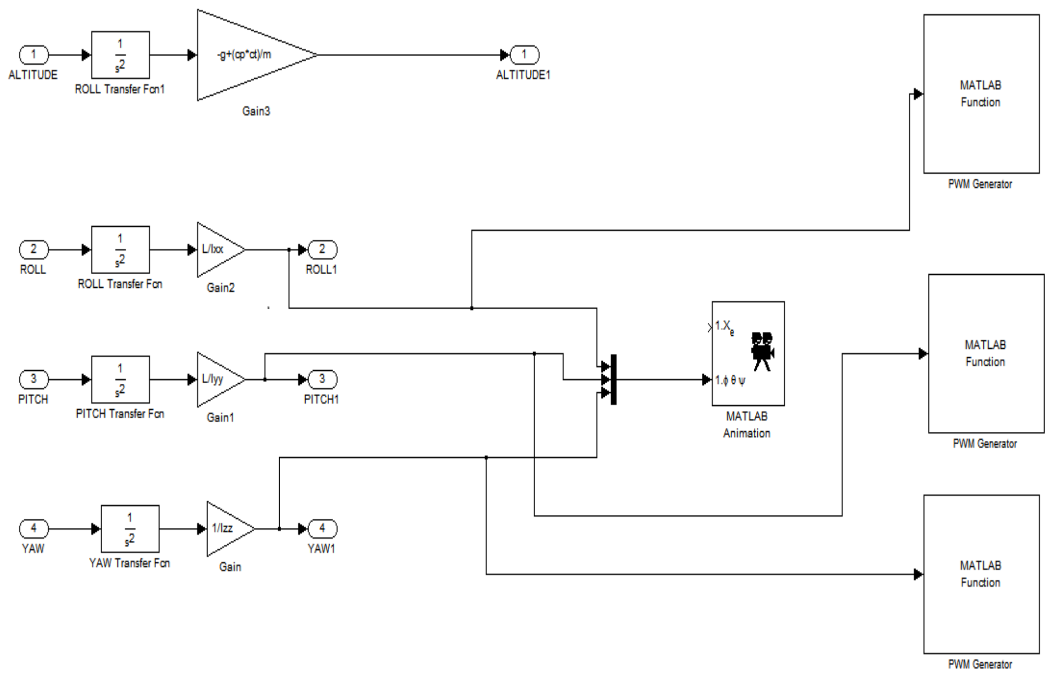


Fig 4.2 : Quad Rotor Block

4.1.1.6 Inertial Measurement Unit Block

The acceleration and torque of the system is interpreted in terms of rotational and translational parameters using this block. It is used for providing feedback to the controller.

4.2 Motor Thrust Testing

A test stand was constructed for a detailed comparative study of a GWS 10x6 three bladed propellers and a Dragonflyer 8x4.5 two bladed propeller. The study revealed the adverse effects of vibration without improvement in the thrust value when using a 10x6 three bladed propeller.

The two test structures used for testing are displayed below. Fig 4.3 is used for estimating the thrust produced due to a single motor. Fig 4.4 is the structure for testing the interference between 2 motors are varying distances and recognizing the optimum distance.

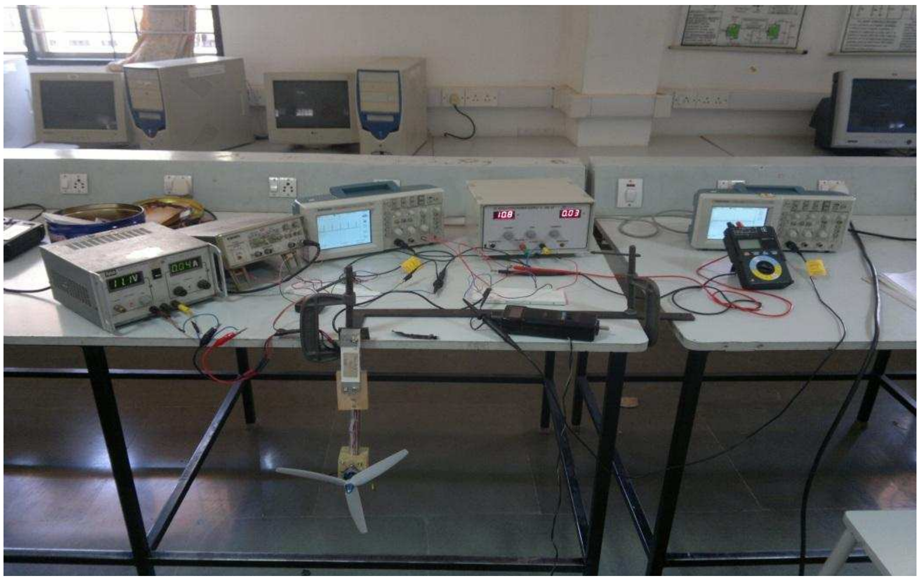


Fig 4.3 : Thrust Testing Setup

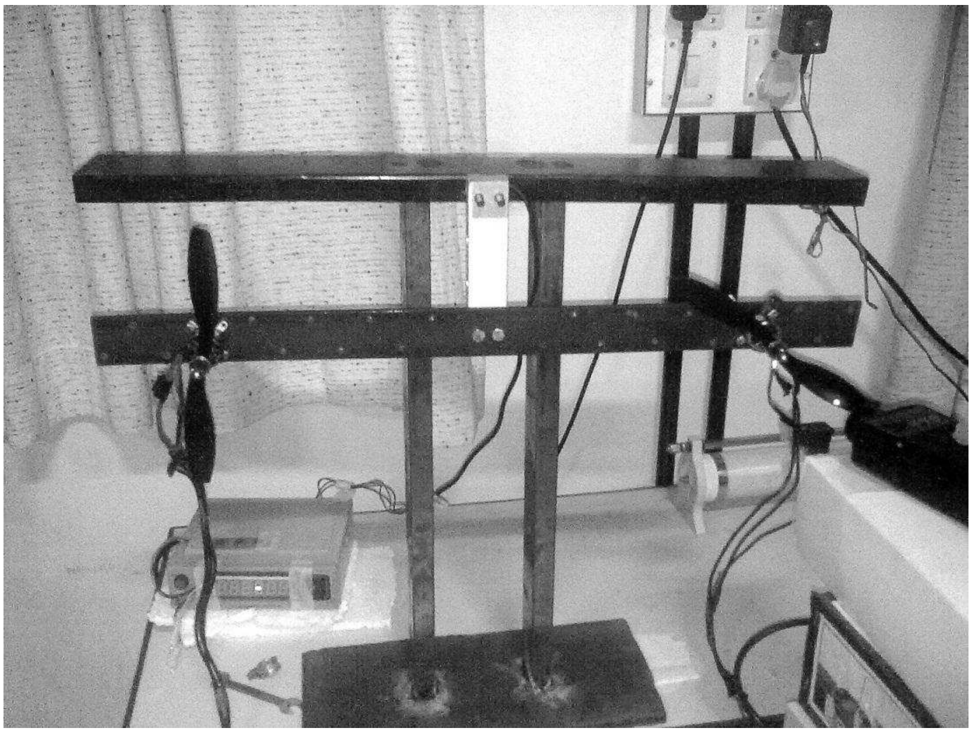


Fig 4.4 : Interference Testing

CHAPTER 5. RESULTS

5.1 BLDC motor control simulation

We have simulated the PIC code for generating the input signal required for the ESC. This is done using delay signals based on the number of cycles required to produce a specified delay.

We use the relation:-

$$x = \frac{Delay}{t_{cyc}}$$

Where,

$$t_{cyc} = \frac{1}{f_{cyc}}$$

A C++ program was implemented for carrying out this calculation for as many times, as the number of bits we assign for PWM signal selection. In our implementation scheme, we use 4 bits, thus giving us 16 values. The C++ code is flexible for any number of bits.

The sample outputs for 2 cases of 0000 and 0001 are depicted:-

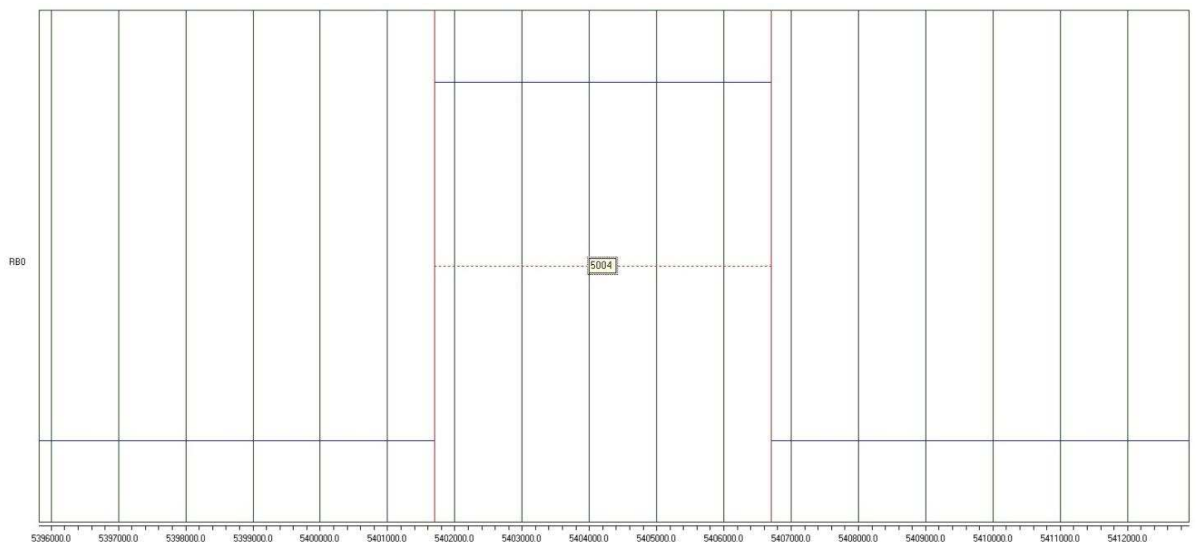


Fig 5.1 : PWM signal corresponding to 1ms on time

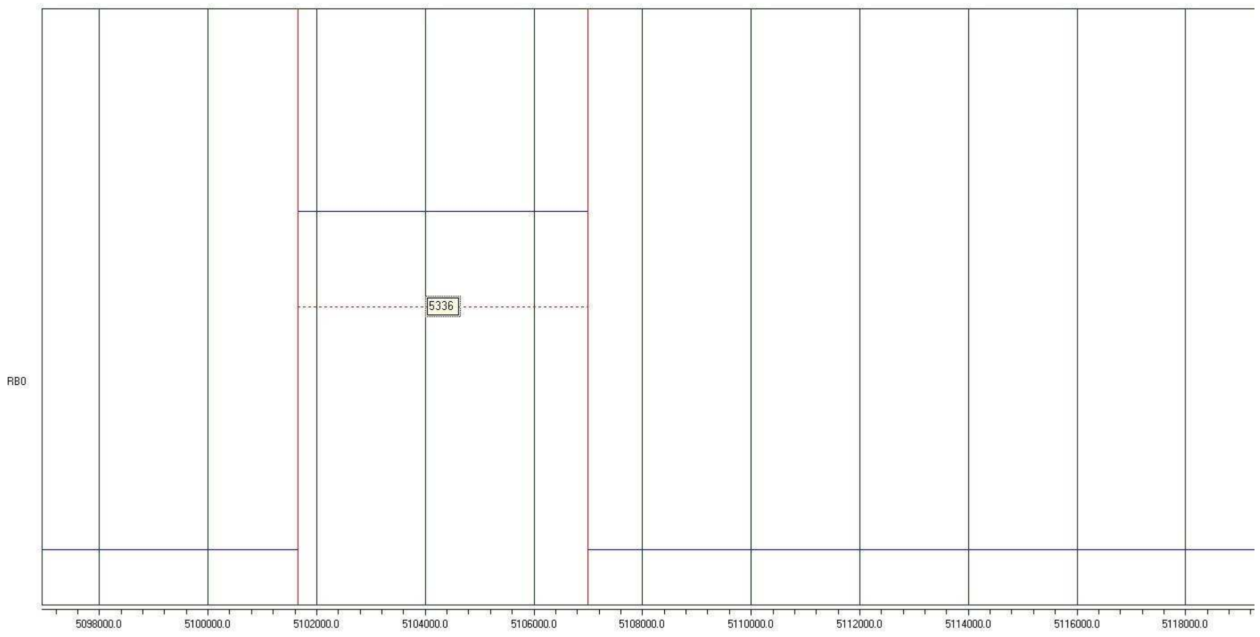
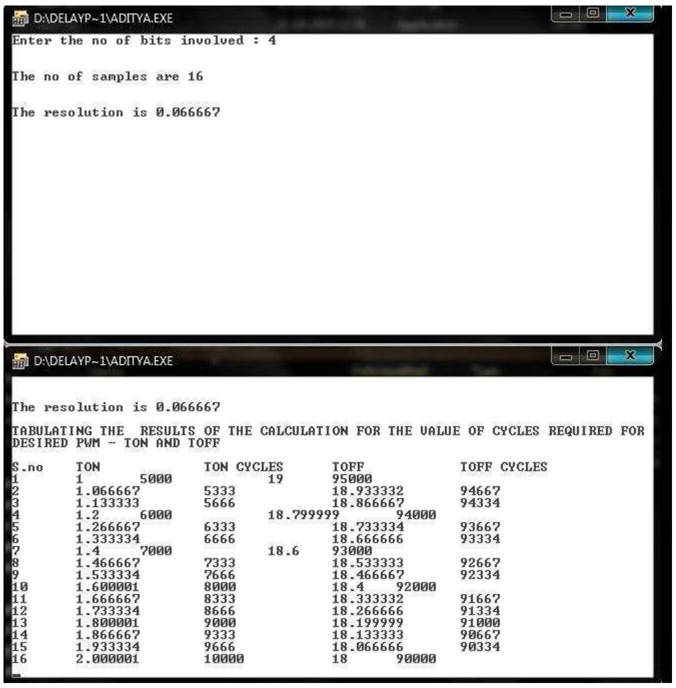


Fig 5.2 : PWM signal corresponding to 1.06ms on time

The values of delay are obtained from the C++ code are :-



```
Enter the no of bits involved : 4

The no of samples are 16

The resolution is 0.066667
```

```
The resolution is 0.066667

TABULATING THE  RESULTS OF THE CALCULATION FOR THE VALUE OF CYCLES REQUIRED FOR
DESIRED PWM - TON AND TOFF

S.no    TON             TON CYCLES      TOFF            TOFF CYCLES
1       1       5000            19      95000
2       1.066667        5333            18.933332       94667
3       1.133333        5666            18.866667       94334
4       1.2     6000            18.799999       94000
5       1.266667        6333            18.733334       93667
6       1.333334        6666            18.666666       93334
7       1.4     7000            18.6    93000
8       1.466667        7333            18.533333       92667
9       1.533334        7666            18.466667       92334
10      1.600001        8000            18.4    92000
11      1.666667        8333            18.333332       91667
12      1.733334        8666            18.266666       91334
13      1.800001        9000            18.199999       91000
14      1.866667        9333            18.133333       90667
15      1.933334        9666            18.066666       90334
16      2.000001        10000           18      90000
```

Fig 5.3 : C++ Code for computing on cycles

On comparison we see that the programmed values match with the values we obtain as output of the PWM code. We will thus proceed with the implementation of this code with the ESC.

5.2 Motor control using Electronic Speed Controller

We have obtained a method of varying the PWM signal input to the ESC and the motor rating is known to us as well. But, the relation between the PWM signals and the motor speed has to be obtained. This value is of great significance, as the control of the quad rotor depends on the speed of the motors. We thus performed experiments with the motor and the results are shown below.

5.2.1 Observation

Using an RC transmitter-receiver arrangement, we were able to get an approximate value of the minimum and maximum throttle signals. We then varied the PWM between the extreme values to obtain more specific results.

Table 5.1 : PWM Signal Limits

	On Time
Minimum Throttle	0.92ms
Maximum Throttle	1.92ms

We checked the current input and voltage output of the ESC at the minimum and maximum speeds.

Table 5.2 : PWM - Speed Characteristics

PWM On Time	**Speed**	**Current Drawn**	**Voltage O/P from ESC**
0.92ms	0 rpm	0.02A	0V
0.94 ms	2800 rpm	0.2A	2V
1.92ms	11320 rpm	0.7A	8V

5.3 Quad rotor simulation

The PID parameters were tuned and the values were obtained. The plots illustrate the ability of the control system in stabilizing the quad rotor. The four single input single output loops for altitude, roll, pitch and yaw, work simultaneously to effectively control the quad rotor. In each of the plots shown, the x axis represents time. The duration of the simulation is 35s. The plot in Figure 5.4 represents the reference input given for the altitude and the graph in Figure

5.5 represents the actual altitude of the system. Here, the y axis represents the altitude of the quad rotor in metres.

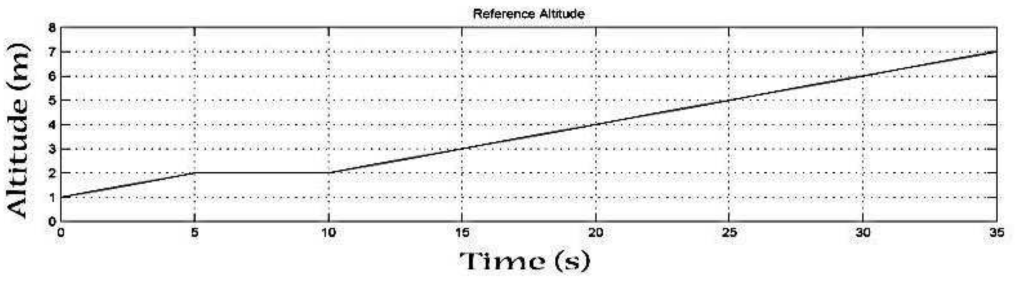


Fig 5.4 : Reference Altitude

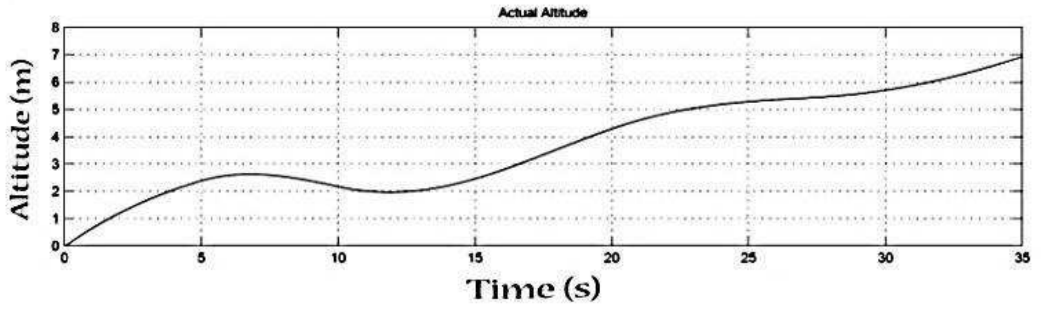


Fig 5.5 : Actual Altitude

The plots in Figures 5.6 to 5.9 represent the reference values and the actual values of the rotational parameters – viz. roll, pitch and yaw. The y axis represents the angle of the respective parameter in radians.

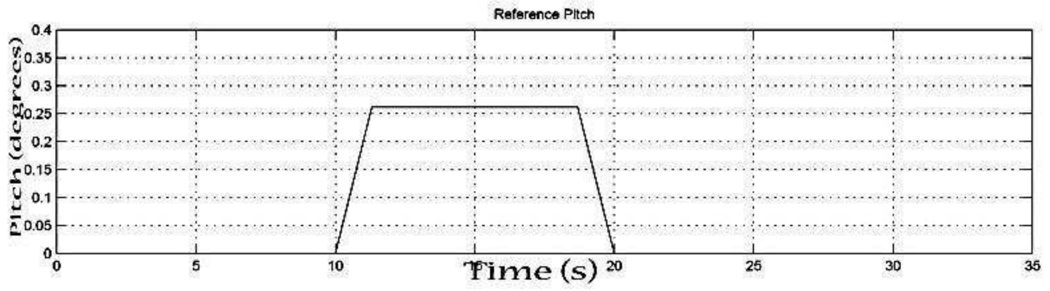


Fig 5.6 : Reference Pitch

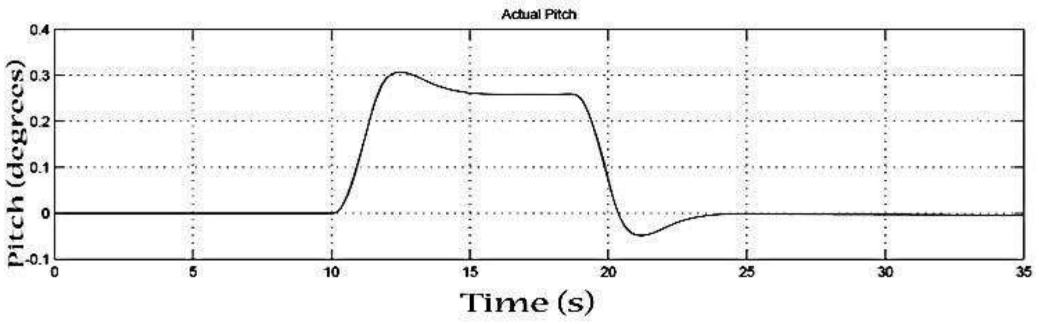


Fig 5.7 : Actual Pitch

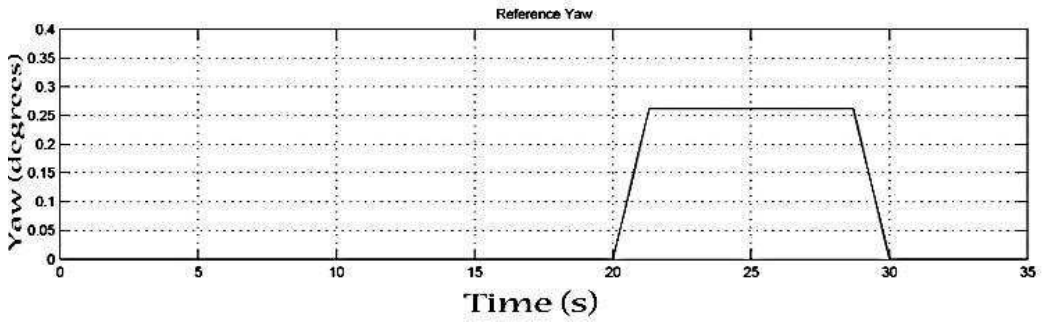


Fig 5.8 : Reference Yaw

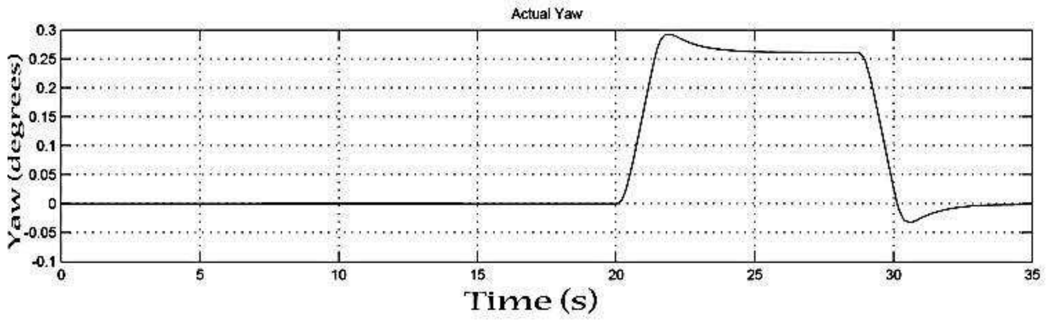


Fig 5.9 : Actual Yaw

5.4 Motor Thrust Testing

The BLDC motors are tested along with their propellers for obtaining the values required for design. The sections below deal with testing them for different arrangements and the results are mentioned.

5.4.1 Aim

To devise experimental setup and perform experiments to determine properties of motor-propeller combination (2 bladed and 3 bladed Propellers) and also to suggest spacing between the propellers which result in minimum aerodynamic interference.

5.4.2 Observation

The trend between the variations of Thrust v/s RPM and Thrust v/s current for the 2 and 3 bladed propellers are given in the figures below. These were used to obtain the quad rotor constants. There is also the plot of the aerodynamic interference.

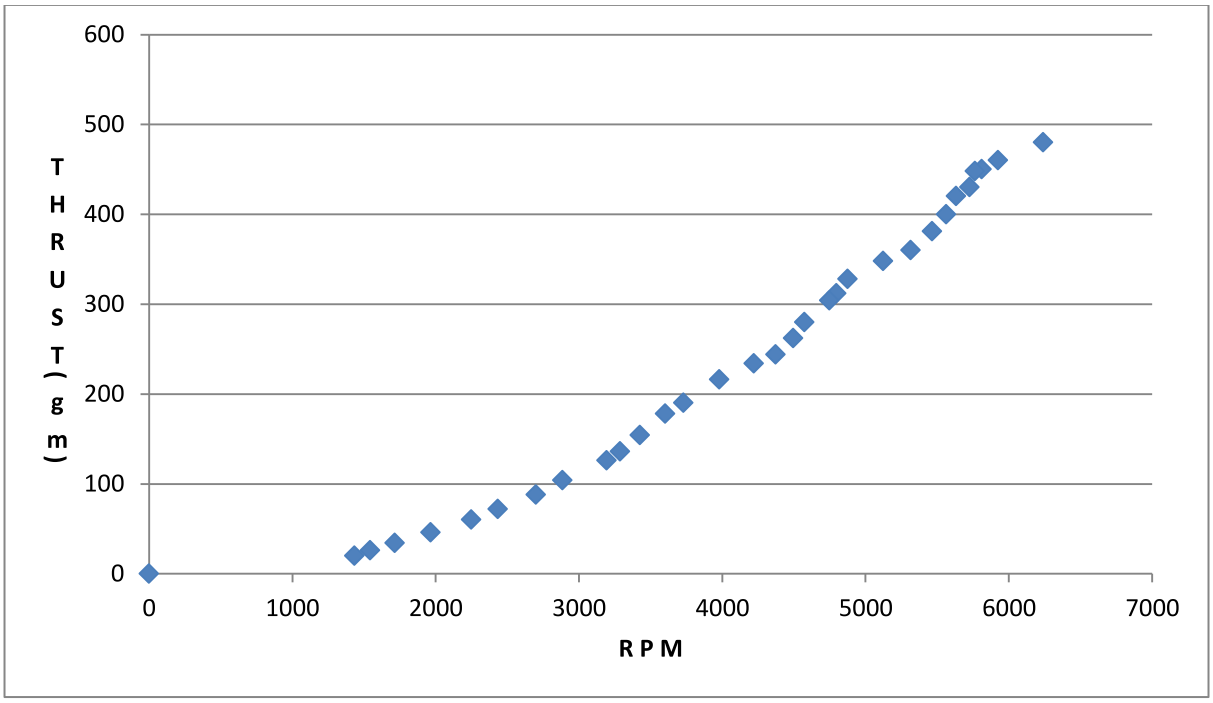


Fig 5.10 : Two Blades - Load vs. RPM

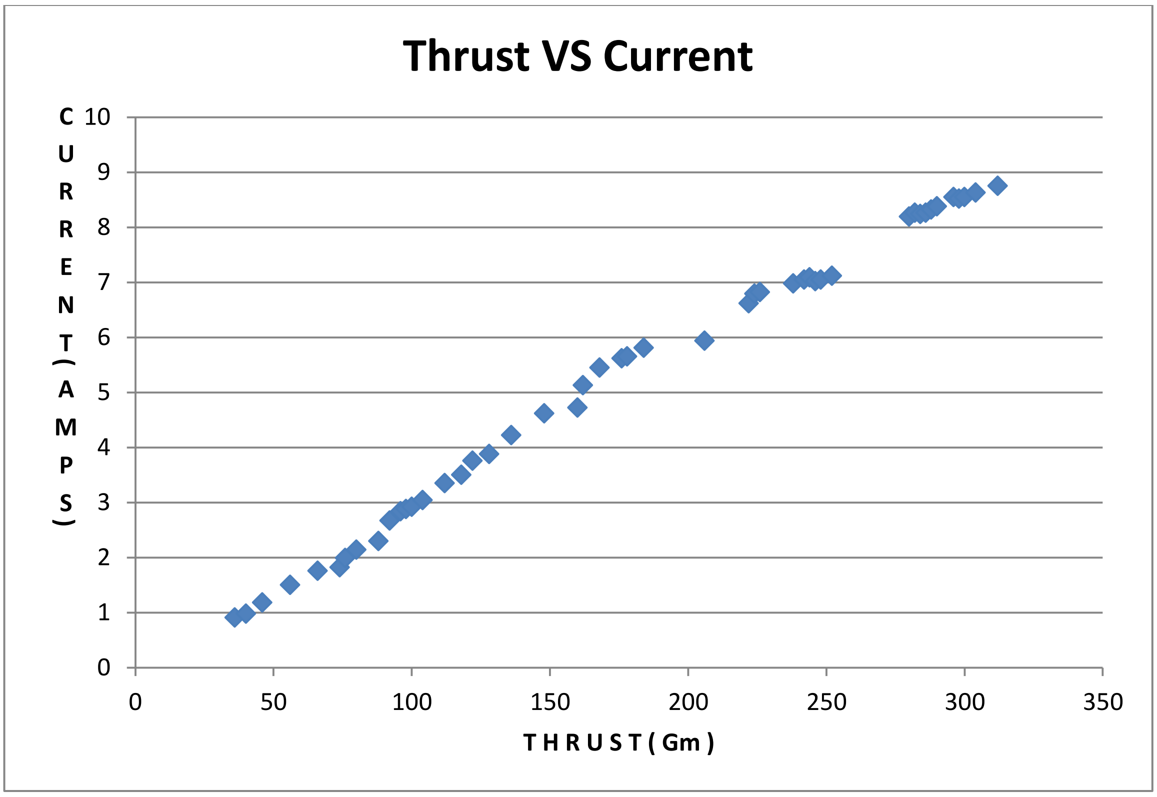


Fig 5.11 : Two Blades - Thrust vs. Current

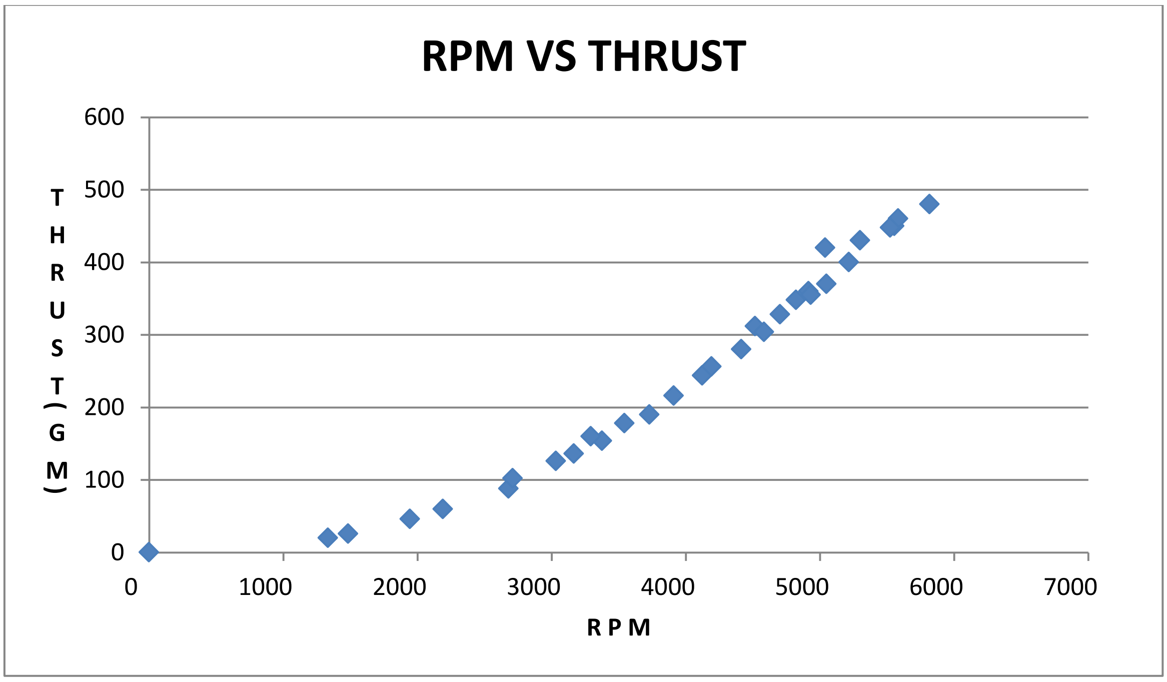


Fig 5.12 : Three blade - Thrust vs. RPM

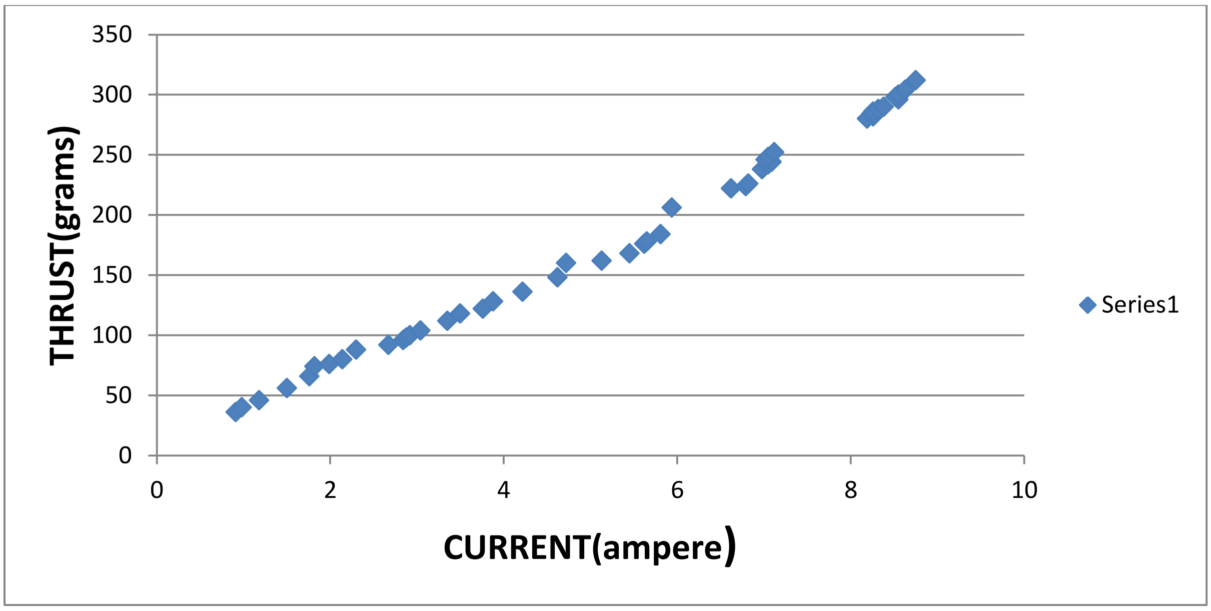


Fig 5.13 : Three Blade - Thrust vs. Current

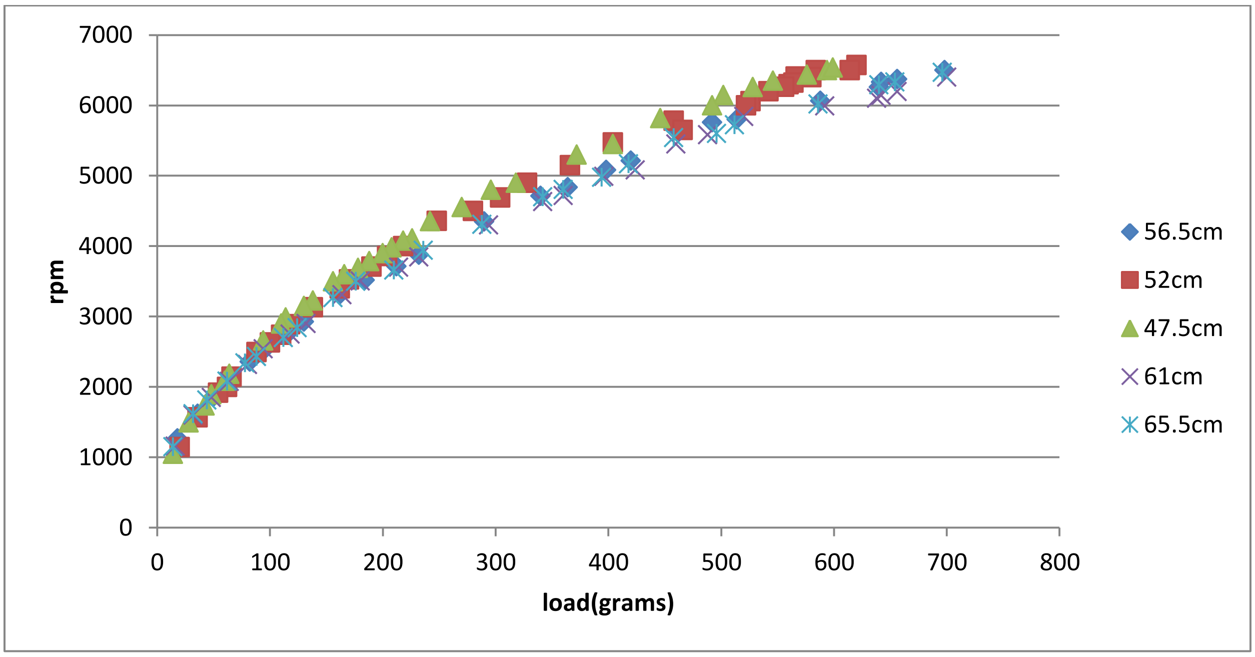


Fig 5.14 : Aerodynamic Interference

5.4.3 Inference

1) The current vs. thrust and thrust vs. rpm characteristics of all the 4 motor-propeller combination is slightly different.

2) Though there is variation in the maximum thrusts, it will not affect the Quadrotor performance because the operational range of thrust is below the maximum value.

3) At hovering condition, the time of flight achieved using the dual blades is 23% more than that achieved using the three bladed propellers.

4) Since the time of flight is an important consideration in our project and since two blade produces almost similar thrust as the 3 blade propeller at low currents, we chose the 2 blade dragon fly 8x propeller over a 3 blade propeller

5) The aerodynamic interference was minimal when the propellers were kept at a distance of 61cm (which is thrice the diameter of our 2 bladed propellers).

CHAPTER 6. CONCLUSION

6.1 Work Completed

The study of various quad rotors were carried out following which the optimal set of modules required and the hardware configuration was finalized. Modeling of the quad rotor was performed based on information from the literature. The modeling was used as a base for developing a simulation of the quad rotor. This simulation incorporated response of the system for various perturbations. Control of motor speed and thrust using the Arduino was performed which paved way for testing and further construction. Motor thrust tests were performed to compare different propellers and obtain the optimal distance between propellers placed at the ends of each arm. The interference was the factor to be minimized. These tests then led to the finalization of the quad rotor structure which was assembled using carbon fiber rods and nylon mounts and supports. A test stand was designed for the purpose of testing the quad rotor giving freedom about the 3 axes in the rotational planes. All testing was carried out on this. We then implemented the control system code which was studied from literature and modified to suit our needs. A stable hovering was not yet possible, but all the pre-requisites have been fulfilled.

6.2 Future Work

Future work would include fine tuning the PID control system which has been implemented but not yet optimized. Implementation of other control strategies can also be performed and the code re-written on further study.

With inclusion of further sensors, an automatic heading system and point to point autonomous navigation can be implemented.

A tele-command system for controlling the motion of the UAV can also be designed. A telemetry system can enhance the functionality of the system for applications such as video surveillance, reconnaissance, etc.

REFERENCES

[1] Akhil M, M. Krishna Anand, Aditya Sreekumar, P. Hithesan, “Simulation of the Mathematical Model of a Quad Rotor Control System using Matlab Simulink,” Proceedings of the 2011 International Conference on Mechanical and Aerospace Engineering, March 21-23, 2011, New Delhi, India, p. 389 - 393

[2] Guilherme V. Raffo, Manuel G. Ortega, Francisco R. Rubio, “MPC with Nonlinear H∞ Control for Path Tracking of a Quad-Rotor Helicopter,” Proceedings of the 17th World Congress of The International Federation of Automatic Control, July 6-11, 2008, Seoul, Korea, http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/2578.pdf

[3] S. Bouabdallah and R. Siegwart, “Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor,” Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005, pp. 2247-2252

[4] M.B. Srikanth, Z.T. Dydek, A.M. Annaswamy, and E. Lavretsky, “A robust environment for simulation and testing of adaptive control for mini-UAVs,” American *Control Conference, 2009. ACC’09.*, IEEE, 2009, p. 5398–5403

[5] Y. Wu, “Development and Implementation of a Control System for a Quadrotor UAV,” MSc. Thesis, University of Applied Science Ravensburg, Weingarten, Baden-Wurttemberg , Germany, March, 2009

[6] Robert Barclay, “A Generic Simulator for Quad-Rotor Unmanned Aerial Vehicles,” Undergraduate Project Dissertation, University of Sheffield, United Kingdom, 2005

[7] M. Önkol and M. Efe, “Experimental Model Based Attitude Control Algorithms for a Quadrotor Unmanned Vehicle,” 2nd International Symposium on Unmanned Aerial Vehicles, June 8-10, 2009, Nevada, USA, http://www.onderefe.etu.edu.tr

[8] Tommaso Bresciani, “Modelling, identification and control of a Quadrotor helicopter,” Master Thesis, Lund University, Lund, Sweden, October, 2008 , www.roboticsclub.org/redmine/attachments/467/Quadrotor_Bible.pdf

[9] I.D. Cowling, O.A. Yakimenko, J.F. Whidborne, and A.K. Cooke, “A Prototype of an Autonomous Controller for a Quadrotor UAV,” Proceedings of the European Control Conference, Kos, Greece, 2007, http://www2.engr.arizona.edu/~sprinkjm/research/c2wt/uploads/Main/paper1.pdf

[10] Smruti Ranjan Dora, Neerav Harsh, S. Maitreyi, “Design and Implementation of an Indoor Quad Rotor”, Undergraduate Project Dissertation, Amrita Vishwa Vidyapeetham, Coimbatore, Tamil Nadu, India, 2010

[11] Preethi Kumar, Diwakar Mandal, “Design and Development of a Compact Integrated Telecommand and Telemetry System for an Unmanned Aerial Vehicle”, Undergraduate Project Dissertation, Amrita Vishwa Vidyapeetham, Coimbatore, Tamil Nadu, India, 2010

[12] J.G.Leisman, Evolution of Helicopter Flight, http:// www.100yearsofflight.com, retrieved December 2006.

[13] S.Sassen, P. Uhleman, “Quattrocopter A unique Micro-Aerial Vehicle,” European Aeronautic Defense and Space Company Corporate Research Centre, November 2003.

[14] Scott D. Hanford, Lyle N. Long, Joseph F. Horn, “A Small Semi-Autonomous Rotary-Wing Unmanned Air Vehicle (UAV)”, Infotech@Aerospace Conference, American Institute of Aeronautics and Astronautics, Paper no 2005-7077, December 2005.

[15] E.Altuğ, “Vision based control of unmanned aerial vehicles with applications to an autonomous four rotor helicopter, quadrotor,” PhD thesis, University of Pennsylvania, 2003.

[16] B.Çamlıca, “Demonstration of a stabilized hovering platform for undergraduate laboratory,” Master Thesis, Middle East Technical University, December 2004.

[17] P.Pounds , R.Mahony, P. Hynes, J. Roberts, “Design of a four-rotor aerial robot,” Australasian Conference on Robotics and Automation Auckland, 27-29 November 2002.

[18] M. Chen, M. Huzmezan, “A Combined MBPC/ 2 DOF H∞ Controller for a Quad Rotor UAV, Department of Electrical and Computer Engineering,” University of British Columbia Vancouver, BC, Canada, V6T 1Z4, 2003.

[19] G.Hoffmann, D.Dostal, S.Waslander, J.Jang, C.Tomlin, “Stanford Test bed of Autonomous Rotorcraft for Multi-Agent Control(STARMAC),” Stanford University, October 28th, 2004.

[20] http://en.wikipedia.org/wiki/Unmanned_aerial_vehicle#cite_note-0, accessed 2nd January 2011

[21] www.copleycontrols.com/motion/pdf/Field-Oriented-Control.pdf

[22] http://www.mathworks.com/matlabcentral/fileexchange/24675-arduino-target

[23] http://de.nanotec.com/schrittmotor_animation.html?motor=motor_bldc_block_delta

[24] Syed Ali Raza, Wail Gueaieb, “ Intelligent Flight Control of an Autonomous Quadrotor , " Motion Control, Federico Casolo (Ed.), ISBN: 978-953-7619-55-8, InTech, Available from: http://www.intechopen.com/articles/show/title/intelligent-flight-control-of-an-autonomous-quadrotor

[25] William Premerlani and Paul Bizard, “Direction Cosine Matrix IMU : Theory”, gentlenav.googlecode.com/files/DCMDraft2.pdf

[26] http://aeroquad.com/

[27] http://code.google.com/p/ardu-imu/wiki/HomePage

APPENDIX

Appendix A : Test stand design and fabrication

Introduction

After successfully controlling the motor, we have to test the thrust developed by the motor propeller arrangement. For this purpose, we have developed a new test stand. This will allow us to test the thrust of a single motor or thrust of 2 motors coupled together. The 2-motor arrangement uses a rider to vary the distance between the 2 motors.

Design

The design is as show below:-

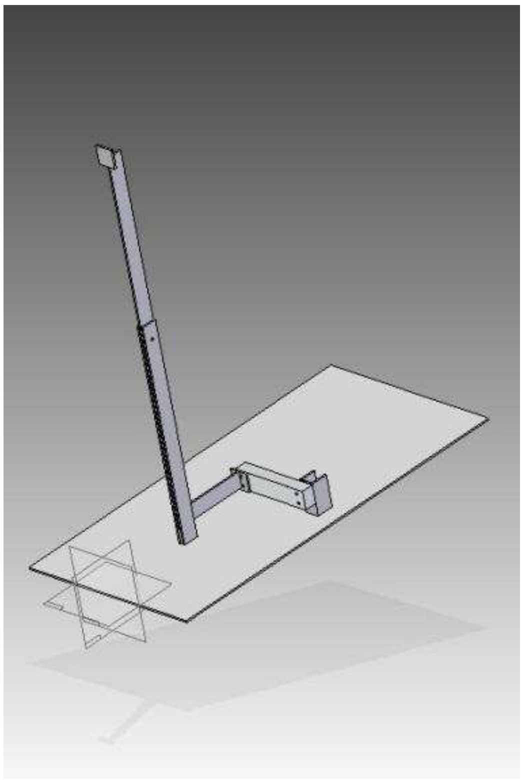


Fig 1 : Single Motor Test Stand

The test stand shown in figure was designed to test the motor thrust. The stand was designed to minimize the interference with the propeller wake.

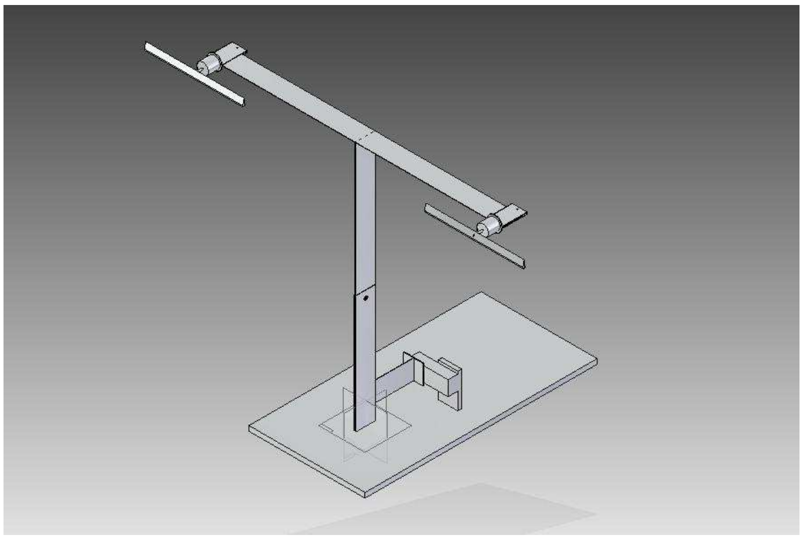


Fig 2 : Two Motor Test stand

The two-motor test stand was designed to test the effect of aerodynamic interference with varying spacing between the motor axes.

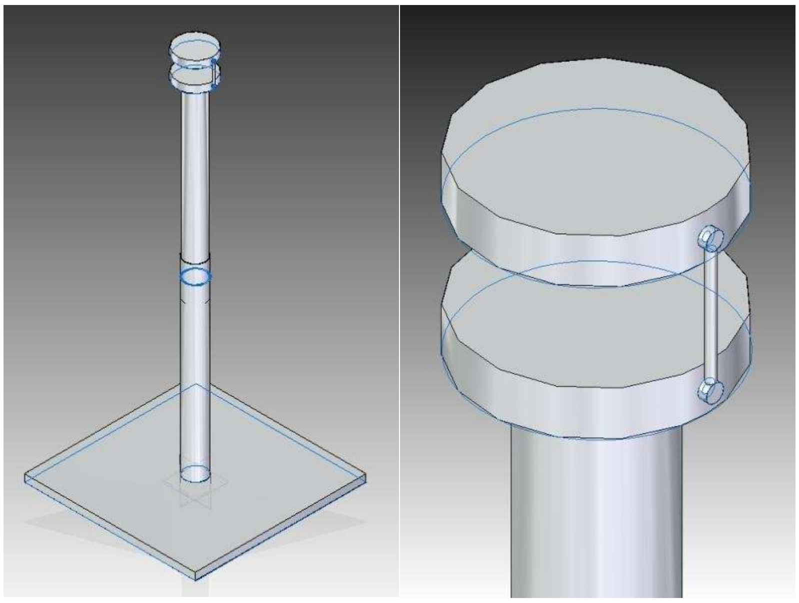


Fig 3 : Gimbal test stand

The test stand above was designed to tune the control system of the Quadrotor. The test stand allow 3 degrees of freedom which are rotations about x, y and z.

Appendix B : Length of Arm

Aerodynamic Considerations

The studies conducted on dual rotor interference indicate that the wake of two rotors in tandem configuration is independent if the spacing between the rotor axes is twice the rotor diameter. It is illustrated in the figure. Theoretically when the rotor wakes interfere the induced power coefficient will have to be considered. The aerodynamic effects of wake interference has not been fully explored and understood hence it was decided to maintain a minimum rotor axis-axis distance of twice the rotor diameter. This meant that the diagonally opposite rotors will be spaced at a distance of approximately 3 times rotor diameter.

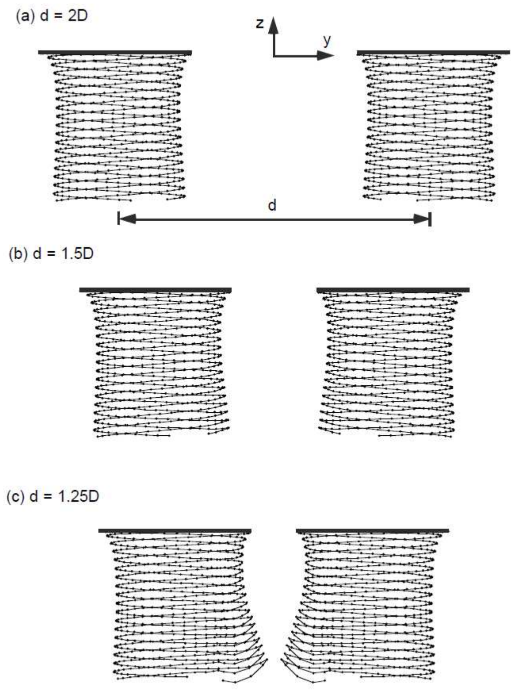
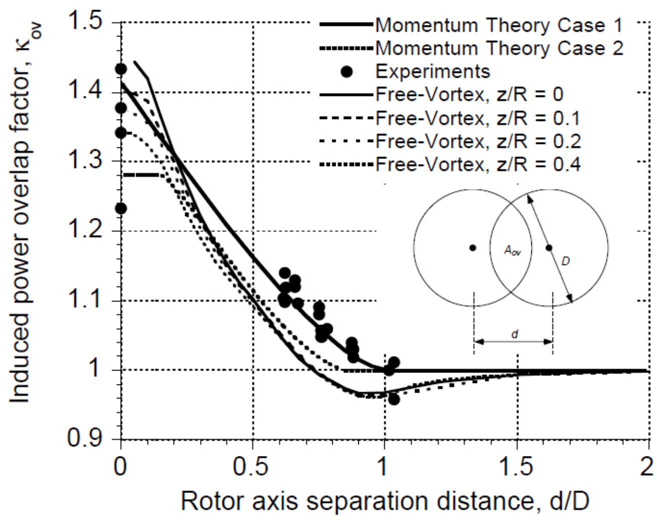


Fig 1 : Tandem Wake Interference



Controllability Considerations

The effectively control the Quadrotor the control loop by the Quadrotor to respond to a control should function faster than the time taken input i.e. the time constant. The time constant associated with each axis is proportional to the square root of the corresponding [24] moment of inertia.

$$M \sim I\frac{\theta}{t^2}$$

$$t \sim \sqrt{\frac{\theta}{M} I}$$

Where I is the moment of Inertia, θ is the angle of rotation, M is the moment and t is the time constant [24]. Therefore: The moment of inertia increases with the arm length.

Safety and Mission Considerations

The indoor mission profile requires the Quadrotor the fly through doors. The width of a typical door was is 80 cm. For the Quadrotor to pass through the door was easily it should fit in a square 80cm x 80cm. If propeller diameter =D

2*D*1.41+D<80 cm hence.

Propeller diameter, D<= 8 in.

Appendix C : Propulsion System Selection

The thrust requirement of the Quadrotor for hovering was estimated to be 1100g with an additional 200g of thrust for maneuvering. The propulsion system requirement was thus a minimum of 1300g of thrust.

Propellers

The quad rotor system requires two contra-rotating pair of propellers. The Quadrotor was being designed for indoor flight and hence a smaller size was desirable. The Quadrotor designed to hover and fly at near hovering conditions hence a thin low pitch propeller was desired. While hovering the propellers operate under static thrust condition. The static thrust is independent of propeller pitch

Table 1 : Power variation with pitch for static thrust

Diameter (in)	Pitch	No of blades	Static thrust (g)	RPM	Power required (kW)
8	3	2	350	8000	0.033
8	4	2	350	8000	0.044
8	5	2	350	8000	0.055
8	6	2	350	8000	0.066

Table 2 : Static thrust variation with diameter and rpm

Diameter (in)	Pitch	No of blades	Static thrust (g)	RPM	Power required (kW)
5	4	2	370	21000	0.122
6	4	2	340	14000	0.075
7	4	2	390	11000	0.067
8	4	2	350	8000	0.044
9	4	2	370	6500	0.037

The size of the propellers is chosen to be 8’’ by 4’’ as they are easily available in market and will be able to generate the required thrust as per our calculations. Two propellers were shortlisted:

- APC 8*3.8 slow flyer
- Dragon flyer 8*4.5

The Dragon flyer propeller being thin was suitable for hovering and also packed less kinetic energy and moment of inertia hence was selected. The Dragon flyer propeller was also used in a very popular Quadrotor.

Motors

The motor was selected based on the power and rpm requirements of the propeller. The motors selected for this purpose are Brushless DC motors rated for 7.2-11.1 volts and 7A continuous currents. These motors are out-runner motors that are characterized by their high efficiency. These motors create significant torque and can thus drive direct props without the need for a gearbox.

TOPBAND AE2806-01B 1400Kv Model: AE2806-01B Weight(g): 35 KV: 1400 Shaft Diameter (mm): 3.0 No load current (11.1V): 0.7A Max	Current (60s):13A Max Speed (rpm):8000 Max Thrust (g):600 Prop:8×4,7×5,6×5.5 (3S) Prop:9x5, 10x4.7(2S)

Electronic Speed Controller

An appropriate electronic speed controller was also selected. Dual-sky XC2512BA Speed Controller. An electronic speed control or ESC is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake.

For a BLDC motor, the ESC performs the function of the motor driver and also causes the commutation. When a PWM signal is fed to the ESC, it converts this PWM signal to a 3 phase PWM of the supply voltage. We are thus able to vary the supply voltage to the BLDC motor by varying the PWM input to the ESC, in turn controlling the speed of the motor.

Batteries

The battery is selected on the basis of power requirements for the selected motor combination. We have opted for a battery of the lithium-polymer variety, despite the fact that it is considerably more expensive than other batteries providing the same power, because this battery provides the best power-to-weight ratio. Our battery choice is a 1600mAh 11.1V Li-polymer battery.

Appendix D : Structural Design

The Quadrotor has a plus shaped configuration with motors mounted near the outer ends of the four arms. The CAD diagrams of the design are shown in the following pages.

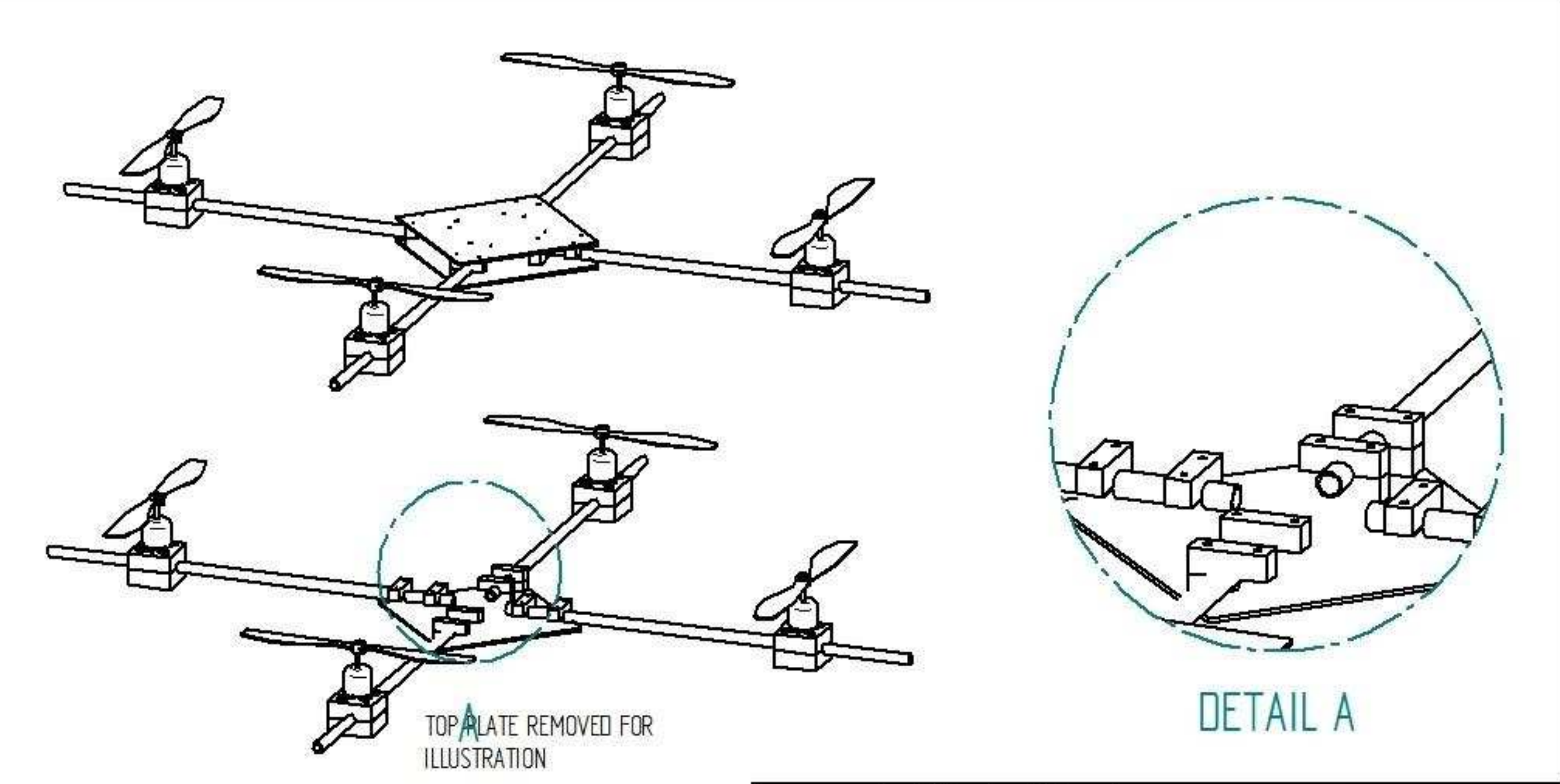
Location of Center of Gravity

The Quad rotor's intended flight domain is indoors hence is not expected to be affected by sever wind. The center of gravity location is chosen to be as close as possible to the plane of rotation of the rotor. The center of gravity when located below the plane of rotation of the propeller improves the stability of the Quadrotor by providing a restoring torque when perturbed. Such a configuration is suitable for outdoor flight. The location of center of gravity in plane to the rotor translates a larger fraction of the perturbations into the better manageable linear accelerations rather than to the adverse angular accelerations.

Carbon fiber Arm design

The carbon fiber Arm was designed with the help of Solid Edge Software. The relevant data are shown below.

Modulus of Elasticity:	117246.3158 MPa
Modulus of Rigidity:	3997.85 MPa
Specific Mass:	1550.0744 kg/m^3
Beam Weight	0.019 kg
Reaction at Support	185.651 mN
Moment at Support	-0.0401 Nm
Rotation at Support	0.00 deg
Deflection at Load	-0.03 mm



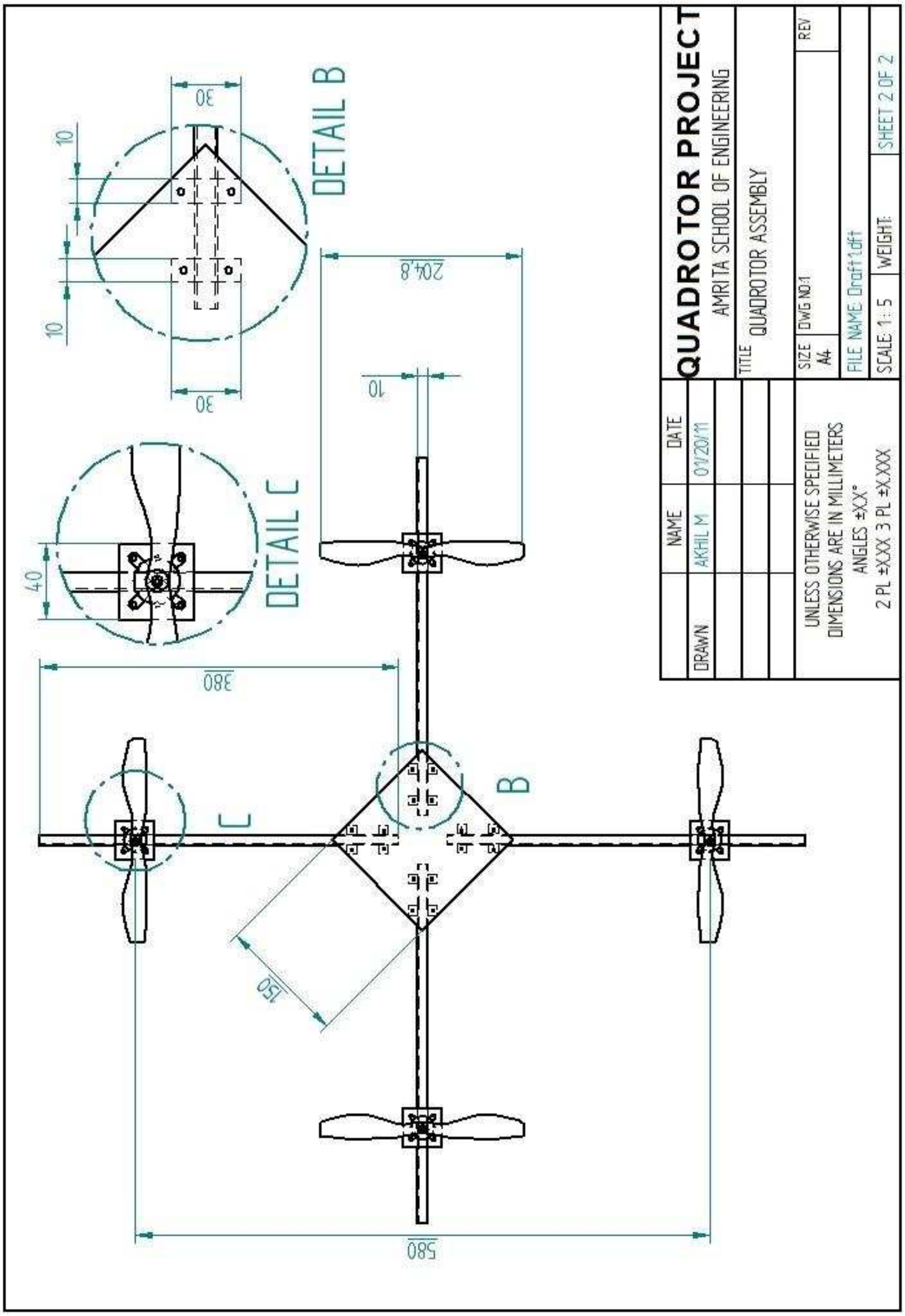
	NAME	DATE
DRAWN	AKHIL M	01/20/11

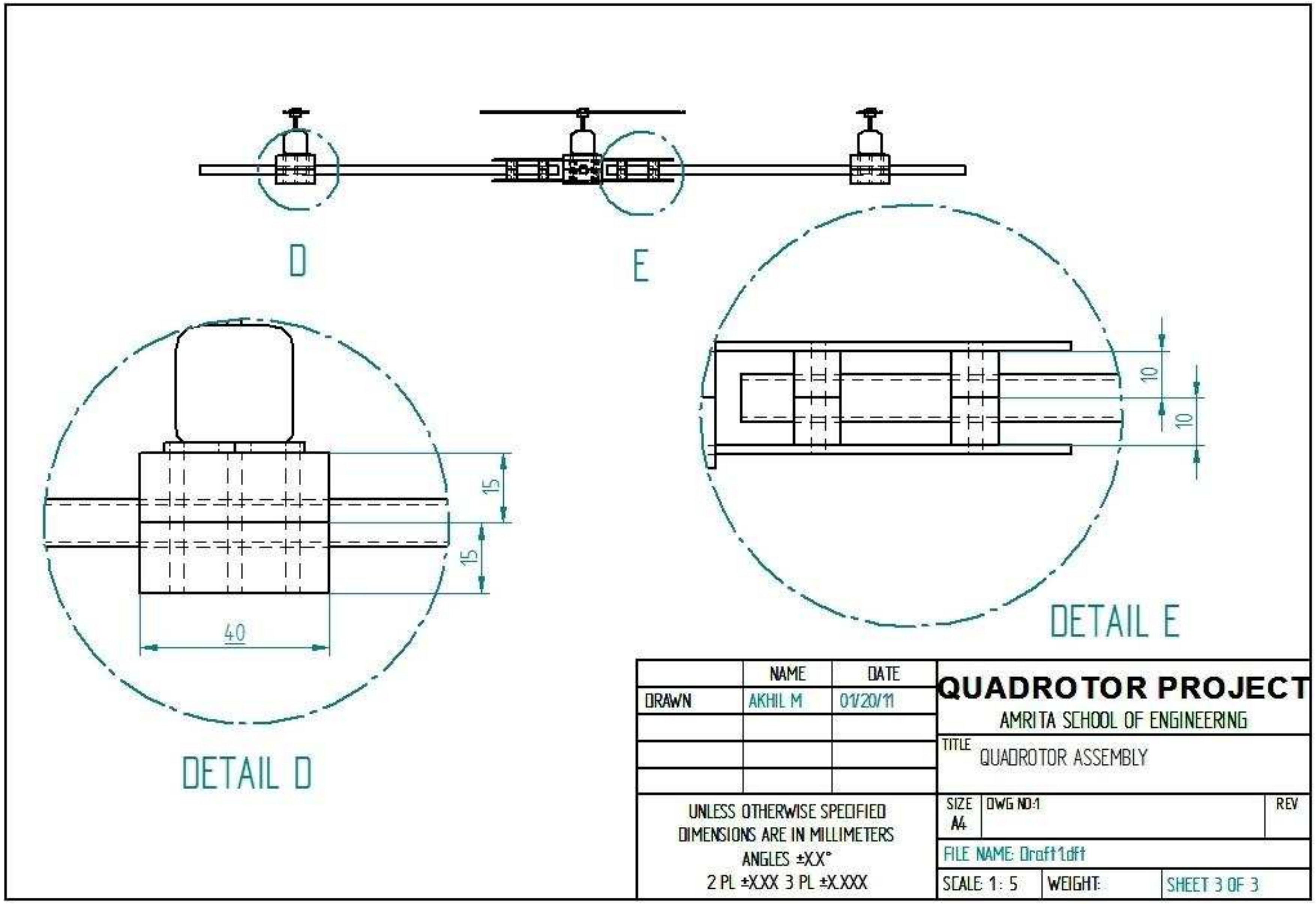
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX

QUADROTOR PROJECT

AMRITA SCHOOL OF ENGINEERING

TITLE QUADROTOR ASSEMBLY		
SIZE A4	DWG NO:1	REV
FILE NAME: Draft1.dft		
SCALE: 1 : 5	WEIGHT:	SHEET 1 OF 2


QUADROTOR PROJECT
AMRITA SCHOOL OF ENGINEERING
TITLE QUADROTOR ASSEMBLY
DRAWN
NAME
DATE
AKHIL M
01/20/11
SIZE A4
DWG NO:1
REV
FILE NAME: Draft1.dft
SCALE 1 : 5
WEIGHT:
SHEET 2 OF 2
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±XX°
2 PL ±X.XX 3 PL ±X.XXX
DETAIL B
DETAIL C
B
C
30
30
10
10
204,8
10
40
380
150
580


D
E
DETAIL D
40
15
15
DETAIL E
10
10
NAME
DATE
DRAWN
AKHIL M
01/20/11
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
QUADROTOR PROJECT
AMRITA SCHOOL OF ENGINEERING
TITLE
QUADROTOR ASSEMBLY
SIZE
A4
DWG NO:1
REV
FILE NAME: Draft1.dft
SCALE: 1 : 5
WEIGHT:
SHEET 3 OF 3

Appendix E : Detailed Weight Estimation

The basic frame is shown in the figure below.

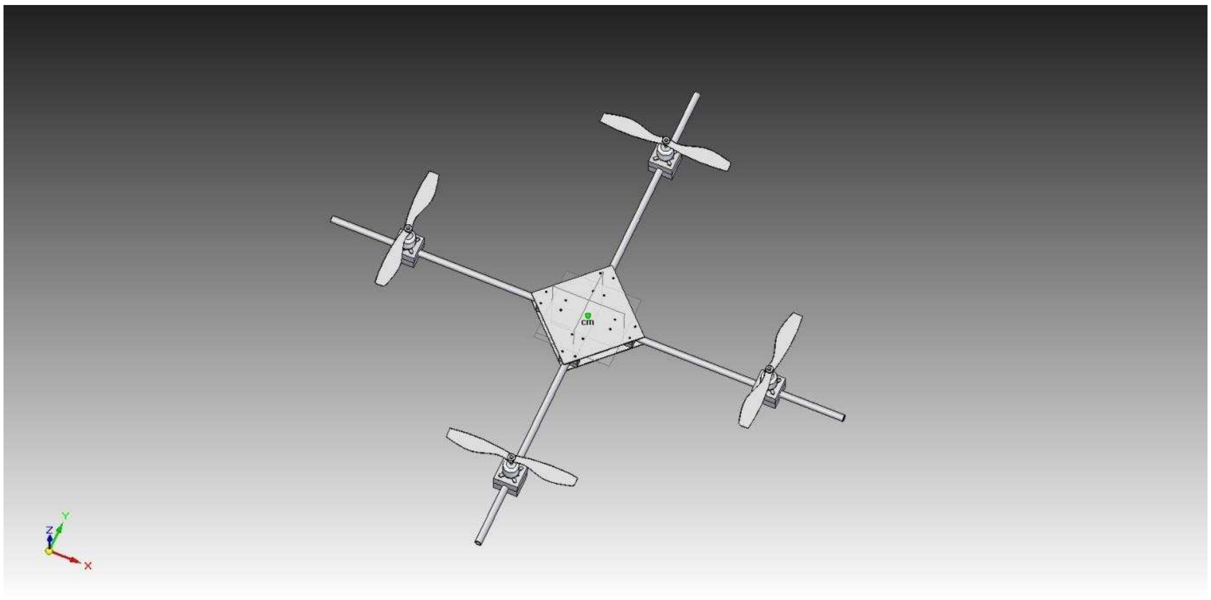


Fig 1 : Quad Rotor Design

The different components of the frame contribute to the weight of the system and have been explicitly mentioned below:-

Table 1 : Weight of the system

PART	Material	WEIGHT PER PART	NO OF PARTS
Quad rotor		0.940 kg	
Carbon fiber Arm	Carbon fiber	0.017 kg	4
Central Hub	Carbon fiber	0.056 kg	2
hub holder	Nylon, general purpose	0.006 kg	8
Motor Base	Aluminum, 1060	0.002 kg	4
Motor mount bottom half	Nylon, general purpose	0.027 kg	8
Motor		0.024 kg	4
Propeller	Carbon fiber	0.010 kg	4
Battery		0.080 kg	2
ESC		0.020 kg	4
Arduino		0.50	1

Appendix F : Code for PWM generation for the Arduino

```
void setup()
{
// port directions: 1=input, 0=output
pinMode(13,OUTPUT);
pinMode(2,INPUT);
pinMode(3,INPUT);
pinMode(4,INPUT);
pinMode(5,INPUT);
pinMode(6,INPUT);
//TRISC = 0x00;
//Taking TRISC as ouput only for testing purposes
//Taking TRISC as input for practical purposes


}


void pwmgen(float x)
{
  digitalWrite(13,HIGH);
  delay(x);
  digitalWrite(13,HIGH);
  delay(20-x);
}
void loop()
{
RC1=0;
RC2=1;
RC3=0; */
```

```
                    }


int inmain=digitalRead(2);
int in1=digitalRead(3);
int in2=digitalRead(4);
int in3=digitalRead(5);
int in4=digitalRead(6);

//We trigger RC4 to 1 to enable the PWM operation
//RC4=1;

digitalWrite(13,LOW);

          if(inmain==1)          //RC4 is used to switch on the PWM signal generator
          {
          if(in1==0)
          {
               if(in2==0)
               {
                    if(in3==0)
                    {
                         if(in4==0)
                         {
                                   pwmgen(0.92);
                         }
                         else
                         {
                                   pwmgen(0.9825);
                         }
                    }
```

```
        else
        {
                if(in4==0)
                {
                                pwmgen(1.045);
                }
                else
                {
                                pwmgen(1.1075);
                }
        }
}

else
{
        if(in3==0)
        {
                if(in4==0)
                {
                                pwmgen(1.17);
                }
                else
                {
                                pwmgen(1.2325);
                }
        }

        else
        {
                }
                if(in4==0)
                {
                        pwmgen(1.295);
                }
                else
                {
                        pwmgen(1.3575);
                }
            }
        }

}

else
    {
        if(in2==0)
        {
            if(in3==0)
            {
                if(in4==0)
                {
                        pwmgen(1.42);
                }
                else
                {
                        pwmgen(1.4825);
                }
            }
        else
        {
                if(in4==0)
                {
                                pwmgen(1.545);
                }
                else
                {
                                pwmgen(1.6075);
                }
        }
}

else
{
        if(in3==0)
        {
                if(in4==0)
                {
                                pwmgen(1.67);
                }
                else
                {
                                pwmgen(1.7325);
                }
        }

        else
        {
```

```
                                        if(in4==0)
                                        {
                                                    pwmgen(1.795);
                                        }
                                        else
                                        {
                                                    pwmgen(1.92);
                                        }
                                  }
                            }


                }
          }

          }
```

Appendix G : Code for delay value generation

```
#include<iostream.h>

#include<math.h>

#include<conio.h>

main()

{

//initialising the variables

intbits,count=0,ton[100];

float samples=1,i,res,j,toff;

unsigned long inttoffc;


clrscr();


//Read the no of bits which are to be assigned for ESC control in the PIC

cout<<"Enter the no of bits involved : ";

cin>>bits;


//Calculate no of samples we obtain using the no of bits

for(i=1;i<=bits;i++)

{

samples=2*samples;

}

cout<<"\n\nThe no of samples are "<<samples<<"\n";

samples-=1;

getch();

//Obtain the resolution of the on time variation
res=1/samples;

cout<<"\n\nThe resolution is "<<res<<"\n\n";

getch();


cout<<"TABULATING THE  RESULTS OF THE CALCULATION FOR THE VALUE OF CYCLES
REQUIRED FOR DESIRED PWM - TON AND TOFF \n\nS.no\tTON\t\tTON CYCLES\tTOFF\t\tTOFF
CYCLES\n";


i=1.000;



//calculation and display of the ton relation with the corr delay value to be assigned

for(j=0;j<=samples;j++)

{

ton[j]=i*5000;

toffc=100000-ton[j];

toff=20-i;

cout<<j+1<<"\t"<<i<<"\t"<<ton[j]<<"\t\t"<<toff<<"\t"<<toffc<<"\t\n";

i+=res;

}

getch();

}
```

Appendix H : PWM generation block

```
function PWM=PWM (u)
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
PWM=(B^-1)*((c^-1)*u-A)
```

Appendix I : Motor dynamics block

```
function F=FORCE (PWM)
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
F=A+B*PWM
```

Appendix J : PWM signal generation block

```
function u=U (F)
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
u=c*F
```

Appendix K : Code for Φ

```
function phi(phi)
global cp
global sp
cp=cos(phi);
sp=sin(phi);
```

Appendix L : Code for θ

```
global ct
global st
ct=cos(theta);
st=sin(theta);
```

Appendix M : Code for ψ

```
function si(si)
global cs
global ss
cs=cos(si);
ss=sin(si);
```

Appendix N : Code for the Angle

```
function angle (phi,theta,si)
cp=cos(phi);
ct=cos(theta);
cs=cos(si);
sp=sin(phi);
st=sin(theta);
```

Appendix O : Initialization Code

```
Ixx=0.017247
Iyy=0.017674
Izz=0.030082
b=1.5500e-006
d=b/10
g=9.81;
L=.3;
m=0.9;
Kiroll=-0.0070
Kdroll=0.0987
Kproll=0.0763
Kipitch=-0.0070
Kdpitch=0.0987
Kppitch=0.0763
Kiyaw=-0.0070
Kdyaw=0.0987
Kpyaw=0.0763
Kialtitude=-0.0099
Kdaltitude=-0.0831
Kpaltitude=-0.0195
Kix=-0.0099
Kdx=-0.0831
Kpx=-0.0195
Kiy=-0.0099
Kdy=-0.0831
Kpy=-0.0195
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
global cs
global ss
global ct
global st
global cp
global sp
cs=0
ss=0
ct=0
st=0
cp=0
sp=0
```

Appendix P : Arduino Code for Implementing the Control System

```
/*                    JJ ArduIMU Quadcopter                              */
/*                                                                */
/* Code based on ArduIMU DCM code from Diydrones.com                    */
/* Date : 22-02-2010                                                 */
/* Author : Jose Julio                                              */
/* Version : 1.20 (mini)                                             */
/* This version supports the optional magnetometer                   */
/* With the magnetometer we eliminate the yaw drift                   */
/* ******************************************************************** */

#include <inttypes.h>
#include <math.h>
#include <Wire.h>   // For magnetometer readings

// QuadCopter PID GAINS
#define KP_QUAD_ROLL 1.8 // 2.2   //1.75
#define KD_QUAD_ROLL 0.42 // 0.54  //0.45
#define KI_QUAD_ROLL 0.5 // 0.45  //0.5
#define KP_QUAD_PITCH 1.8 // 2.2   //1.75
#define KD_QUAD_PITCH 0.42 // 0.54  //0.45
#define KI_QUAD_PITCH 0.5 // 0.45  //0.5
#define KP_QUAD_YAW 3.8 // 4.6  //3.2 //2.6
#define KD_QUAD_YAW 1.3 // 0.7  //0.8 //0.4
#define KI_QUAD_YAW 0.15 // 0.2  //0.15

#define KD_QUAD_COMMAND_PART 18.0    // for special KD implementation (in two
parts)
```

```
// The IMU should be correctly adjusted : Gyro Gains and also initial IMU offsets:

// We have to take this values with the IMU flat (0º roll, 0ºpitch)

#define acc_offset_x 508

#define acc_offset_y 504

#define acc_offset_z 501       // We need to rotate the IMU exactly 90º to take this value

#define gyro_offset_roll 370

#define gyro_offset_pitch 373

#define gyro_offset_yaw 380


// We need to now the number of channels to read from radio

// If this value is not well adjusted you will have bad radio readings... (you can check the
radio at the begining of the setup process)

#define MAX_CHANNELS     7        // Number of radio channels to read (7 is the number if
you use the PPM encoder from store.diydrones.com)


#define MIN_THROTTLE 1037       // Throttle pulse width at minimun...

#define CHANN_CENTER 1500


#define SPEKTRUM 1  // Spektrum radio


/*
*************************************************************************
************** */


// ADC : Voltage reference 3.3v / 10bits(1024 steps) => 3.22mV/ADC step

// ADXL335 Sensitivity(from datasheet) => 330mV/g, 3.22mV/ADC step => 330/3.22 =
102.48

// Tested value : 101

#define GRAVITY 101 //this equivalent to 1G in the raw data coming from the accelerometer
```

```
#define Accel_Scale(x) x*(GRAVITY/9.81)//Scaling the raw data of the accel to actual
acceleration in meters for seconds square


#define ToRad(x) (x*0.01745329252)  // *pi/180

#define ToDeg(x) (x*57.2957795131)  // *180/pi


// LPR530 & LY530 Sensitivity (from datasheet) => 3.33mV/º/s, 3.22mV/ADC step => 1.03

// Tested values : 0.96,0.96,0.94

#define Gyro_Gain_X 0.92 //X axis Gyro gain

#define Gyro_Gain_Y 0.92 //Y axis Gyro gain

#define Gyro_Gain_Z 0.94 //Z axis Gyro gain

#define Gyro_Scaled_X(x) x*ToRad(Gyro_Gain_X) //Return the scaled ADC raw data of the
gyro in radians for second

#define Gyro_Scaled_Y(x) x*ToRad(Gyro_Gain_Y) //Return the scaled ADC raw data of the
gyro in radians for second

#define Gyro_Scaled_Z(x) x*ToRad(Gyro_Gain_Z) //Return the scaled ADC raw data of the
gyro in radians for second


#define Kp_ROLLPITCH 0.0125  //0.010 // Pitch&Roll Proportional Gain

#define Ki_ROLLPITCH 0.000010 // Pitch&Roll Integrator Gain

#define Kp_YAW 1.2 // Yaw Porportional Gain

#define Ki_YAW 0.00005 // Yaw Integrator Gain


/*Min Speed Filter for Yaw drift Correction*/

#define SPEEDFILT 3 // >1 use min speed filter for yaw drift cancellation, 0=do not use


/*For debugging purposes*/

#define OUTPUTMODE 1   //If value = 1 will print the corrected data, 0 will print
uncorrected data of the gyros (with drift), 2 Accel only data


uint8_t sensors[6] = {6,7,3,0,1,2};  // For Hardware v2 flat
```

```
//Sensor: GYROX, GYROY, GYROZ, ACCELX, ACCELY, ACCELZ
int SENSOR_SIGN[]={1,-1,-1,1,-1,1,-1,-1,-1}; //{1,-1,-1,-1,1,-1}

float AN[6]; //array that store the 6 ADC filtered data
float AN_OFFSET[6]; //Array that stores the Offset of the gyros

float G_Dt=0.02;    // Integration time for the gyros (DCM algorithm)

float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a vector
float Accel_Vector_unfiltered[3]= {0,0,0}; //Store the acceleration in a vector
float Accel_magnitude;
float Accel_weight;
float Gyro_Vector[3]= {0,0,0};//Store the gyros rutn rate in a vector
float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
float Omega_P[3]= {0,0,0};//Omega Proportional correction
float Omega_I[3]= {0,0,0};//Omega Integrator
float Omega[3]= {0,0,0};

float errorRollPitch[3]= {0,0,0};
float errorYaw[3]= {0,0,0};
float errorCourse=0;
float COGX=0; //Course overground X axis
float COGY=1;

float roll=0;
float pitch=0;
float yaw=0;
```

```
};


//Magnetometer variables
int magnetom_x;
int magnetom_y;
int magnetom_z;
float MAG_Heading;

unsigned int counter=0;

float DCM_Matrix[3][3]= {
  {
    1,0,0  }
  ,{
    0,1,0  }
  ,{
    0,0,1  }
};
float Update_Matrix[3][3]={{0,1,2},{3,4,5},{6,7,8}}; //Gyros here



float Temporary_Matrix[3][3]={
  {
    0,0,0  }
  ,{
    0,0,0  }
  ,{
    0,0,0  }
};

//#define roll_offset 0.0
```

```
//#define pitch_offset 0.0


// PPM Rx signal read (ICP) constants and variables
#define Servo1Pin  9           // Servo pins...
#define Servo2Pin 10
#define icpPin     8          // Input Capture Pin (Rx signal reading)
//#define MAX_CHANNELS    4       // Number of radio channels to read
#define SYNC_GAP_LEN    8000    // we assume a space at least 4000us is sync (note clock
counts in 0.5 us ticks)
#define MIN_IN_PULSE_WIDTH (750)  //a valid pulse must be at least 750us (note clock
counts in 0.5 us ticks)
#define MAX_IN_PULSE_WIDTH (2250) //a valid pulse must be less than  2250us
static volatile unsigned int Pulses[ MAX_CHANNELS + 1];  // Pulse width array
static volatile uint8_t  Rx_ch = 0;
int Neutro[MAX_CHANNELS+1];    // Valores para los neutros en caso de fallos...
byte radio_status=0;           // radio_status = 1 => OK, 0 => No Radio signal
static volatile unsigned int ICR1_old = 0;
static volatile unsigned int Timer1_last_value;  // to store the last timer1 value before a reset
int ch1;    // Channel values
int ch2;
int ch3;
int ch4;
int ch_aux;
int ch_aux2;
int ch1_old;    // Channel values
int ch2_old;
int ch3_old;
int ch4_old;
int ch_aux_old;
int ch_aux2_old;
```

```
// PPM Rx signal read END


// Servo Timer2 variables (Servo Timer2)
#define SERVO_MAX_PULSE_WIDTH 2000
#define SERVO_MIN_PULSE_WIDTH 900
#define SERVO_TIMER2_NUMSERVOS 4          // Put here the number of servos. In this
case 4 ESC´s
typedef struct {
  uint8_t pin;
  int value;
  uint8_t counter;
}  servo_t;
uint8_t num_servos=SERVO_TIMER2_NUMSERVOS;
servo_t Servos[SERVO_TIMER2_NUMSERVOS];
static volatile uint8_t Servo_Channel;
static volatile uint8_t ISRCount=0;
static volatile unsigned int Servo_Timer2_timer1_start;
static volatile unsigned int Servo_Timer2_timer1_stop;
static volatile unsigned int Servo_Timer2_pulse_length;
// Servo Timer2 variables END


// Servo variables (OC1 and OC2) for standard servos [disabled in this version]
unsigned int Servo1;
unsigned int Servo2;



//GPS variable definition
```

```
union long_union {
        int32_t dword;
        uint8_t  byte[4];
} longUnion;

union int_union {
        int16_t word;
        uint8_t  byte[2];
} intUnion;

// Flight GPS variables
int gpsFix=0; //This variable store the status of the GPS
float lat=0; // store the Latitude from the gps
float lon=0;// Store guess what?
float alt_MSL=0; //This is the alt.
long iTOW=0; //GPS Millisecond Time of Week
long alt=0; //Height above Ellipsoid
float speed_3d=0; //Speed (3-D)
float ground_speed=0;// This is the velocity your "plane" is traveling in meters for second,
1Meters/Second= 3.6Km/H = 1.944 knots
float ground_course=90;//This is the runaway direction of you "plane" in degrees
char data_update_event=0;
byte GPS_Error=1;

// GPS UBLOX
byte ck_a=0;     // Packet checksum
byte ck_b=0;
byte UBX_step=0;
byte UBX_class=0;
byte UBX_id=0;
byte UBX_payload_length_hi=0;
byte UBX_payload_length_lo=0;
byte UBX_payload_counter=0;
byte UBX_buffer[40];
byte UBX_ck_a=0;
byte UBX_ck_b=0;



// Navigation control variables
//float alt_error;
//float course_error;
//float course_error_old;

// ADC variables
volatile uint8_t MuxSel=0;
volatile uint8_t analog_reference = DEFAULT;
volatile uint16_t analog_buffer[8];
volatile uint8_t analog_count[8];
int an_count;

// Attitude control variables
float comando_rx_roll=0;        // comandos recibidos rx
float comando_rx_roll_old;
float comando_rx_roll_diff;
float comando_rx_pitch=0;
float comando_rx_pitch_old;
float comando_rx_pitch_diff;
float comando_rx_yaw=0;
float comando_rx_yaw_diff;
int control_roll;          // resultados del control
int control_pitch;
int control_yaw;
float K_aux;
float roll_I=0;
float roll_D;
float err_roll;
float err_roll_ant;
float pitch_I=0;
float pitch_D;
float err_pitch;
float err_pitch_ant;
float yaw_I=0;
float yaw_D;
float err_yaw;
float err_yaw_ant;

// AP_mode : 1=> Radio Priority(manual mode) 2=>Stabilization assist mode 3=>Autopilot
mode
byte AP_mode = 2;

long t0;
int num_iter;
float aux_debug;

#define MAX_ROLL_ANGLE 18
#define MAX_PITCH_ANGLE 18
```

```
/* ************************************************************ */
// ROLL, PITCH and YAW PID controls...
void Attitude_control(){

  // ROLL CONTROL
  err_roll_ant = err_roll;
  if (AP_mode==2)        // Stabilization mode
    err_roll = comando_rx_roll - ToDeg(roll);
  else
    err_roll = 0;
  err_roll = constrain(err_roll,-30,30);  // to limit max roll command...

  roll_I += err_roll*G_Dt;
  roll_I = constrain(roll_I,-50,50);
  // D term implementation => two parts: gyro part and command part
  // To have a better (faster) response we can use the Gyro reading directly for the Derivative term...
  // Omega[] is the raw gyro reading plus Omega_I, so it´s bias corrected
  // We also add a part that takes into account the command from user (stick) to make the system more responsive to user inputs
  roll_D = comando_rx_roll_diff*KD_QUAD_COMMAND_PART - ToDeg(Omega[0]);  // Take into account Angular velocity of the stick (command)

  // PID control
  control_roll = KP_QUAD_ROLL*err_roll + KD_QUAD_ROLL*roll_D + KI_QUAD_ROLL*roll_I;

  // PITCH CONTROL
  err_pitch_ant = err_pitch;
  if (AP_mode==2)        // Stabilization mode
```

```
    err_pitch = comando_rx_pitch - ToDeg(pitch);
  else
    err_pitch = 0;
  err_pitch = constrain(err_pitch,-30,30);  // to limit max pitch command...

  pitch_I += err_pitch*G_Dt;
  pitch_I = constrain(pitch_I,-50,50);
  // D term
  pitch_D = comando_rx_pitch_diff*KD_QUAD_COMMAND_PART - ToDeg(Omega[1]);

  // PID control
  control_pitch = KP_QUAD_PITCH*err_pitch + KD_QUAD_PITCH*pitch_D + KI_QUAD_PITCH*pitch_I;

  // YAW CONTROL
  err_yaw_ant = err_yaw;
  if (AP_mode==2){        // Stabilization mode
    err_yaw = comando_rx_yaw - ToDeg(yaw);
    if (err_yaw > 180)    // Normalize to -180,180
      err_yaw -= 360;
    else if(err_yaw < -180)
      err_yaw += 360;
    }
  else
    err_yaw = 0;
  err_yaw = constrain(err_yaw,-60,60);  // to limit max yaw command...

  yaw_I += err_yaw*G_Dt;
  yaw_I = constrain(yaw_I,-50,50);
  yaw_D = comando_rx_yaw_diff*KD_QUAD_COMMAND_PART - ToDeg(Omega[2]);


  // PID control
  control_yaw = KP_QUAD_YAW*err_yaw + KD_QUAD_YAW*yaw_D + KI_QUAD_YAW*yaw_I;
}



int channel_filter(int ch, int ch_old)
{
  int diff_ch_old;
  int result;


  result = ch;
  diff_ch_old = ch - ch_old;      // Difference with old reading
  if (diff_ch_old<0)
    {
    if (diff_ch_old<-30)
      result = ch_old-30;        // We limit the max difference between readings
    }
  else
    {
    if ((diff_ch_old>30)&&(ch_old>900))     // We take into account that ch_old could be 0 (not initialized)
      result = ch_old+30;
    }
  return((ch+ch_old)/2);     // small filtering
}
```

```
long timer=0; //general porpuse timer
long timer_old;

void setup(){

  int i;
  int aux;

  Serial.begin(38400);
  pinMode(2,OUTPUT); //1Serial Mux
  digitalWrite(2,HIGH); //Serial Mux
  pinMode(8,INPUT);    // Rx Radio Input
  pinMode(5,OUTPUT);   // Red LED
  pinMode(6,OUTPUT);   // BLue LED
  pinMode(7,OUTPUT);   // Yellow LED
  pinMode(9,OUTPUT);   // Servo1
  pinMode(10,OUTPUT);  // Servo2  Right Motor
  pinMode(11,OUTPUT);  // Servo3  Left Motor
  pinMode(12,OUTPUT);  // Servo4  Front Motor
  pinMode(13,OUTPUT);  // Servo5  Back Motor

  ch1=MIN_THROTTLE;
  ch2=MIN_THROTTLE;
  ch3=MIN_THROTTLE;
  ch4=MIN_THROTTLE;

  delay(100);
  comando_rx_yaw = 0;
  Servo1 = 1500;
```

```
 Servo2 = 1500;
 Serial.println();
 Serial.println("JJ ArduIMU Quadcopter 1.20 mini PID");
 RxServoInput_ini();
 delay(1000);

// Take neutral radio values...
Neutro[1] = RxGetChannelPulseWidth(1);
Neutro[2] = RxGetChannelPulseWidth(2);
Neutro[3] = RxGetChannelPulseWidth(3);
Neutro[4] = RxGetChannelPulseWidth(4);
Neutro[5] = RxGetChannelPulseWidth(5);
Neutro[6] = RxGetChannelPulseWidth(6);
for (i=0; i<80; i++)
  {
  Neutro[1] = (Neutro[1]*0.8 + RxGetChannelPulseWidth(1)*0.2);
  Neutro[2] = (Neutro[2]*0.8 + RxGetChannelPulseWidth(2)*0.2);
  Neutro[3] = (Neutro[3]*0.8 + RxGetChannelPulseWidth(3)*0.2);
  Neutro[4] = (Neutro[4]*0.8 + RxGetChannelPulseWidth(4)*0.2);
  Neutro[5] = (Neutro[5]*0.8 + RxGetChannelPulseWidth(5)*0.2);
  Neutro[6] = (Neutro[6]*0.8 + RxGetChannelPulseWidth(6)*0.2);
  delay(25);
  }

Serial.print("Rx values: ");
Serial.print(Neutro[1]);
Serial.print(",");
Serial.print(Neutro[2]);
Serial.print(",");
Serial.print(Neutro[3]);
Serial.print(",");
Serial.println(Neutro[4]);
Serial.print(",");
Serial.println(Neutro[5]);
Serial.print(",");
Serial.println(Neutro[6]);

// Roll, Pitch and Throttle have fixed neutral values (the user can trim the radio)
#if SPEKTRUM==1
  Neutro[3] = CHANN_CENTER;
  Neutro[2] = CHANN_CENTER;
  Neutro[1] = MIN_THROTTLE;
#else
  Neutro[1] = CHANN_CENTER;
  Neutro[2] = CHANN_CENTER;
  Neutro[3] = MIN_THROTTLE;
#endif

// Assign pins to servos
num_servos = 4;
Servos[0].pin = 10;       // Left motor
Servos[1].pin = 11;       // Right motor
Servos[2].pin = 12;       // Front motor
Servos[3].pin = 13;       // Back motor
Servo_Timer2_set(0,MIN_THROTTLE);   // First assign values to servos
Servo_Timer2_set(1,MIN_THROTTLE);
Servo_Timer2_set(2,MIN_THROTTLE);
Servo_Timer2_set(3,MIN_THROTTLE);
```

```
Servo_Timer2_ini();              // Servo Interrupt initialization

Analog_Reference(EXTERNAL);
Analog_Init();
I2C_Init();
delay(100);

// Magnetometer initialization
 Compass_Init();

Read_adc_raw();
delay(20);

// Offset values for accels and gyros...
AN_OFFSET[3] = acc_offset_x;
AN_OFFSET[4] = acc_offset_y;
AN_OFFSET[5] = acc_offset_z;
AN_OFFSET[0] = gyro_offset_roll;
AN_OFFSET[1] = gyro_offset_pitch;
AN_OFFSET[2] = gyro_offset_yaw;

// Take the gyro offset values
for(int i=0;i<600;i++)
   {
   Read_adc_raw();
   for(int y=0; y<=2; y++)   // Read initial ADC values for offset.
     AN_OFFSET[y]=AN_OFFSET[y]*0.8 + AN[y]*0.2;
   delay(20);
   }
```

```
Serial.print("AN[0]:");
Serial.println(AN_OFFSET[0]);
Serial.print("AN[1]:");
Serial.println(AN_OFFSET[1]);
Serial.print("AN[2]:");
Serial.println(AN_OFFSET[2]);
Serial.print("AN[3]:");
Serial.println(AN_OFFSET[3]);
Serial.print("AN[4]:");
Serial.println(AN_OFFSET[4]);
Serial.print("AN[5]:");
Serial.println(AN_OFFSET[5]);

delay(1500);

// Wait until throttle stick is at bottom
#if SPEKTRUM==1
while (RxGetChannelPulseWidth(1)>(MIN_THROTTLE+50)){
#else
while (RxGetChannelPulseWidth(3)>(MIN_THROTTLE+50)){
#endif
  //Serial.println("Move throttle stick to bottom to start !!!");
  Serial.print("Radio Channels:");
  for (i=1;i<=MAX_CHANNELS;i++)
    {
    Serial.print(RxGetChannelPulseWidth(i));
    Serial.print(",");
    }
    Serial.println();
    delay(100);
  }

  Read_adc_raw();   // Start ADC readings...
  timer = millis();
  delay(20);

  digitalWrite(7,HIGH);
 }

void loop(){

  int aux;
  float aux_float;

  if((millis()-timer)>=14)   // 14ms => 70 Hz loop rate
  {
    counter++;
    timer_old = timer;
    timer=millis();
    G_Dt = (timer-timer_old)/1000.0;      // Real time of loop run
    num_iter++;

    // IMU DCM Algorithm
    Read_adc_raw();

    if (counter > 7)  // Read compass data at 10Hz... (7 loop runs)
      {
```

```
  counter=0;
  Read_Compass();    // Read I2C magnetometer
  Compass_Heading(); // Calculate magnetic heading
  }

Matrix_update();
Normalize();
Drift_correction();
Euler_angles();
// *****************

// Telemetry data...

aux = ToDeg(roll)*10;
Serial.print(aux);
Serial.print(",");
aux = (ToDeg(pitch))*10;
Serial.print(aux);
Serial.print(",");
aux = ToDeg(yaw)*10;
Serial.print(aux);
/*
Serial.print(magnetom_x);
Serial.print(",");
Serial.print(magnetom_y);
Serial.print(",");
Serial.println(magnetom_z);
*/
```

```
if (radio_status == 1){
  radio_status=2;   // Radio frame read
  ch1_old = ch1;
  ch2_old = ch2;
  ch3_old = ch3;
  ch4_old = ch4;
  ch_aux_old = ch_aux;
  ch_aux2_old = ch_aux2;
  #if SPEKTRUM==1
    ch1 = channel_filter(RxGetChannelPulseWidth(2),ch1_old);   // Aileron
    ch2 = channel_filter(RxGetChannelPulseWidth(3),ch2_old);   // Elevator
    ch3 = channel_filter(RxGetChannelPulseWidth(1),ch3_old);   // Throttle
    ch4 = channel_filter(RxGetChannelPulseWidth(4),ch4_old);   // Ruder
    ch_aux = channel_filter(RxGetChannelPulseWidth(6),ch_aux_old);  // Aux
    ch_aux2 = channel_filter(RxGetChannelPulseWidth(5),ch_aux2_old);  // Aux2
  #else
    ch1 = RxGetChannelPulseWidth(1);   // Read radio channel values
    ch2 = RxGetChannelPulseWidth(2);
    ch3 = RxGetChannelPulseWidth(3);
    ch4 = RxGetChannelPulseWidth(4);
    //ch_aux = RxGetChannelPulseWidth(6);  // Aux
    //ch_aux2 = RxGetChannelPulseWidth(5);  // Aux
  #endif

  // Commands from radio Rx...
  // Stick position defines the desired angle in roll, pitch and yaw
  comando_rx_roll_old = comando_rx_roll;
  comando_rx_roll = (ch1-CHANN_CENTER)/15.0;
  comando_rx_roll_diff = comando_rx_roll-comando_rx_roll_old;
  comando_rx_pitch_old = comando_rx_pitch;
  comando_rx_pitch = (ch2-CHANN_CENTER)/15.0;
  comando_rx_pitch_diff = comando_rx_pitch-comando_rx_pitch_old;
  aux_float = (ch4-Neutro[4])/200.0;
  comando_rx_yaw += aux_float;
  comando_rx_yaw_diff = aux_float;
  if (comando_rx_yaw > 180)         // Normalize yaw to -180,180 degrees
    comando_rx_yaw -= 360.0;
  else if (comando_rx_yaw < -180)
    comando_rx_yaw += 360.0;

  // I use K_aux to adjust gains linked to a knob in the radio... [not used now]
  K_aux = K_aux*0.8 + ((ch_aux-1500)/100.0 + 0.6)*0.2;
  if (K_aux < 0)
    K_aux = 0;

  AP_mode = 2;   // Stabilization assist mode
  }
else if (radio_status==0)
  {  // Radio_status = 0 Lost radio signal => Descend...
  ch3--;   // Descend  (Reduce throttle)
  if (ch3<MIN_THROTTLE)
    ch3 = MIN_THROTTLE;
  comando_rx_roll = 0;     // Stabilize to roll=0, pitch=0, yaw not important here
  comando_rx_pitch = 0;
  Attitude_control();
  // Quadcopter mix
  Servo_Timer2_set(0,ch3 - control_roll - control_yaw);    // Right motor
  Servo_Timer2_set(1,ch3 + control_roll - control_yaw);    // Left motor
  Servo_Timer2_set(2,ch3 + control_pitch + control_yaw);   // Front motor
  Servo_Timer2_set(3,ch3 - control_pitch + control_yaw);   // Back motor
  }

// Attitude control
//comando_rx_yaw = 0;
Attitude_control();

// Quadcopter mix
if (ch3 > (MIN_THROTTLE+40))  // Minimun throttle to start control
  {
  Servo_Timer2_set(0,ch3 - control_roll - control_yaw);    // Right motor
  Servo_Timer2_set(1,ch3 + control_roll - control_yaw);    // Left motor
  Servo_Timer2_set(2,ch3 + control_pitch + control_yaw);   // Front motor
  Servo_Timer2_set(3,ch3 - control_pitch + control_yaw);   // Back motor
  }
else
  {
  roll_I = 0;  // reset I terms...
  pitch_I = 0;
  yaw_I = 0;
  Servo_Timer2_set(0,MIN_THROTTLE);  // Motors stoped
  Servo_Timer2_set(1,MIN_THROTTLE);
  Servo_Timer2_set(2,MIN_THROTTLE);
  Servo_Timer2_set(3,MIN_THROTTLE);
  // Initialize yaw command to actual yaw
  comando_rx_yaw = ToDeg(yaw);
  comando_rx_yaw_diff = 0;
  }
```

```
    Serial.print(",");
    Serial.print(K_aux);
    Serial.print(",");
    Serial.print(timer-timer_old); // G_Dt*1000

    /*
    Serial.print(",");
    Serial.print(ch1);
    Serial.print(",");
    Serial.print(ch2);
    Serial.print(",");
    Serial.print(ch3);
    Serial.print(",");
    Serial.print(ch4);
    */

    Serial.println();  // Line END
    }
  }
```

Appendix Q : Code for Analog to Digital Conversion

```
// We are using an oversampling and averaging method to increase the ADC resolution

// The theorical ADC resolution is now 11.7 bits. Now we store the ADC readings in float
format

void Read_adc_raw(void)

{

  int i;

  int temp1;

  int temp2;


  // ADC readings...

  for (i=0;i<6;i++)

    {

    temp1= analog_buffer[sensors[i]];   // sensors[] maps sensors to correct order

    temp2= analog_count[sensors[i]];

    if (temp1 != analog_buffer[sensors[i]])   // Check if there was an ADC interrupt during
readings...

      {

      temp1= analog_buffer[sensors[i]];      // Take a new reading

      temp2= analog_count[sensors[i]];

      }

    AN[i] = float(temp1)/float(temp2);

    }


  // Initialization for the next readings...

  for (int i=0;i<8;i++){

    analog_buffer[i]=0;

    analog_count[i]=0;

    if (analog_buffer[i]!=0) // Check if there was an ADC interrupt during initialization...
```

```
      {
      analog_buffer[i]=0;    // We do it again...
      analog_count[i]=0;
      }
  }
}

float read_adc(int select)
{
  if (SENSOR_SIGN[select]<0)
    return (AN_OFFSET[select]-AN[select]);
  else
    return (AN[select]-AN_OFFSET[select]);
}

//Activating the ADC interrupts.
void Analog_Init(void)
{
 ADCSRA|=(1<<ADIE)|(1<<ADEN);
 ADCSRA|= (1<<ADSC);
}

//
void Analog_Reference(uint8_t mode)
{
  analog_reference = mode;
}

//ADC interrupt vector, this piece of code
```

```
//is executed everytime a convertion is done.
ISR(ADC_vect)
{
  volatile uint8_t low, high;
  low = ADCL;
  high = ADCH;
  analog_buffer[MuxSel] += (high << 8) | low;   // cumulate analog values
  analog_count[MuxSel]++;
  MuxSel++;
  MuxSel &= 0x07;   //if(MuxSel >=8) MuxSel=0;
  ADMUX = (analog_reference << 6) | MuxSel;
  // start the conversion
  ADCSRA|= (1<<ADSC);
}
```

Appendix R : Code for Implementing the DCM Algorithm

```
void Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //eq.19

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq.19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq.19

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b
//eq.20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq.21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq.21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq.21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/**************************************************/
```

```
void Drift_correction(void)

{

  //Compensation the Roll, Pitch and Yaw drift.

  float mag_heading_x;

  float mag_heading_y;

  float errorCourse;

  static float Scaled_Omega_P[3];

  static float Scaled_Omega_I[3];

  float Accel_magnitude;

  float Accel_weight;


  //*****Roll and Pitch***************


  // Calculate the magnitude of the accelerometer vector

  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] +
Accel_Vector[1]*Accel_Vector[1] + Accel_Vector[2]*Accel_Vector[2]);

  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.

  // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)

  Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);


  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]);
//adjust the ground of reference

  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);


  Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weight);

  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);


  //*****YAW***************

  // We make the gyro YAW drift correction based on compass magnetic heading
  mag_heading_x = cos(MAG_Heading);

  mag_heading_y = sin(MAG_Heading);

  errorCourse=(DCM_Matrix[0][0]*mag_heading_y)                                          -
(DCM_Matrix[1][0]*mag_heading_x);  //Calculating YAW error

  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw correction to
the XYZ rotation of the aircraft, depeding the position.


  Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);

  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.


  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);

  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the Omega_I


}

/**************************************************/

void Accel_adjust(void)

{


 Accel_Vector[1] += Accel_Scale(speed_3d*Omega[2]);  // Centrifugal force on Acc_y =
GPS_speed*GyroZ

 Accel_Vector[2] -= Accel_Scale(speed_3d*Omega[1]);  // Centrifugal force on Acc_z =
GPS_speed*GyroY


}

/**************************************************/


void Matrix_update(void)

{

  Gyro_Vector[0]=Gyro_Scaled_X(read_adc(0)); //gyro x roll

  Gyro_Vector[1]=Gyro_Scaled_Y(read_adc(1)); //gyro y pitch

  Gyro_Vector[2]=Gyro_Scaled_Z(read_adc(2)); //gyro Z yaw
  // Low pass filter on accelerometer data (to filter vibrations)
  Accel_Vector[0]=Accel_Vector[0]*0.4 + read_adc(3)*0.6; // acc x
  Accel_Vector[1]=Accel_Vector[1]*0.4 + read_adc(4)*0.6; // acc y
  Accel_Vector[2]=Accel_Vector[2]*0.4 + read_adc(5)*0.6; // acc z



  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);//adding integrator
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]);//adding proportional

  //Accel_adjust();//adjusting centrifugal acceleration. // Not used for quadcopter

 #if OUTPUTMODE==1
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
  #endif
  #if OUTPUTMODE==0
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
```

```
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
  #endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c

  for(int x=0; x<3; x++)  //Matrix Addition (update)
  {
    for(int y=0; y<3; y++)
    {
      DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
    }
  }
}

void Euler_angles(void)
{
  #if (OUTPUTMODE==2)         // Only accelerometer info (debugging purposes)
    roll = atan2(Accel_Vector[1],Accel_Vector[2]);   // atan2(acc_y,acc_z)
    pitch = -asin((Accel_Vector[0])/(float)GRAVITY); // asin(acc_x)
    yaw = 0;
  #else        // Euler angles from DCM matrix
    pitch = asin(-DCM_Matrix[2][0]);
    roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
    yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
  #endif

}
```

Appendix S : Code for receiving the signal from the RC Receiver

```
// R/C RADIO PPM SIGNAL READ (USING TIMER1 INPUT CAPTURE AND
OVERFLOW INTERRUPTS)

// AND SERVO OUTPUT ON OC1A, OC1B pins [Disabled now]

// SERVO OUTPUT FUNCTION USING TIMER2 OVERFLOW INTERRUPT


// Timer1 Overflow

// Detects radio signal lost and generate servo outputs (overflow at 22ms (45Hz))

ISR(TIMER1_OVF_vect){


  //TCNT1 = 20000;        // at 16Mhz (0.5us) (65535-20000) = 22 ms (45Hz)

  //OCR1A = 20000 + (Servo1<<1);  // Output for servos...

  //OCR1B = 20000 + (Servo2<<1);


  //TCCR1A = 0xF0;        // Set OC1A/OC1B on Compare Match

  //TCCR1C = 0xC0;        // Force Output Compare A/B (Start Servo pulse)

  //TCCR1C = 0x00;

  //TCCR1A = 0xA0;  // Clear OC1A/OC1B on Compare Match


  TCNT1 = 0;

  Timer1_last_value=0xFFFF;  // Last value before overflow...


  // Radio signal lost...

  radio_status = 0;

}


// Capture RX pulse train using TIMER 1 CAPTURE INTERRUPT

// And also send Servo pulses using OCR1A and OCR1B [disabled now]
// Servo output is synchronized with input pulse train
ISR(TIMER1_CAPT_vect)
{
  if(!bit_is_set(TCCR1B ,ICES1)){       // falling edge?
          if(Rx_ch == MAX_CHANNELS) {        // This should be the last pulse...
             Pulses[Rx_ch++] = ICR1;
          radio_status = 1;              // Rx channels ready...
           }
       TCCR1B = 0x42;              // Next time : rising edge
       Timer1_last_value = TCNT1;  // Take last value before reset
       TCNT1 = 0;                  // Clear counter

       // Servo Output on OC1A/OC1B... (syncronised with radio pulse train)
       //TCCR1A = 0xF0;        // Set OC1A/OC1B on Compare Match
       //TCCR1C = 0xC0;        // Force Output Compare A/B (Start Servo pulse)
       //TCCR1C = 0x00;
       //TCCR1A = 0xA0;       // Clear OC1A/OC1B on Compare Match
  }
  else {                                        // Rise edge
           if ((ICR1-ICR1_old) >= SYNC_GAP_LEN){   // SYNC pulse?
                  Rx_ch = 1;               // Channel = 1
             Pulses[0] = ICR1;
             }
        else {
          if(Rx_ch <= MAX_CHANNELS)
               Pulses[Rx_ch++] = ICR1;    // Store pulse length
          if(Rx_ch == MAX_CHANNELS)
            TCCR1B = 0x02;            // Next time : falling edge
          }
```

```
   }
   ICR1_old = ICR1;
}



// Servo Input PPM Initialization routine
void RxServoInput_ini()
{
  pinMode(icpPin,INPUT);
  Rx_ch = 1;
  //TCCR1A = 0xA0;          // Normal operation mode, PWM Operation disabled, clear
OC1A/OC1B on Compare Match
  TCCR1A = 0x00;       // COM1A1=0, COM1A0=0 => Disconnect Pin OC1 from
Timer/Counter 1 -- PWM11=0,PWM10=0 => PWM
  TCCR1B = 0x42;       // TCNT1 preescaler/8 (16Mhz => 0.5useg, 8 Mhz => 1useg),
Rising edge
  TIMSK1 = _BV(ICIE1)|_BV (TOIE1);   // Enable interrupts : Timer1 Capture and Timer1
Overflow

  Neutro[1] = 1037;
  Neutro[2] = 1037;
  Neutro[3] = 1037;
  Neutro[4] = 1037;
  Servo1 = 1037;
  Servo2 = 1037;
}

int RxGetChannelPulseWidth( uint8_t channel)
{
  unsigned int result;
  unsigned int result2;
  unsigned int pulso_ant;

  unsigned int pulso_act;


  // Because servo pulse variables are 16 bits and the interrupts are running values could be
corrupted.

  // We dont want to stop interrupts to read radio channels so we have to do two readings to be
sure that the value is correct...

  result =  Pulses[channel];

  result2 =  Pulses[channel];

  if (result != result2)

    result =  Pulses[channel];   // if the results are different we make a third reading (this
should be fine)

  pulso_act = result;


  pulso_ant = Pulses[channel-1];

  result2 =  Pulses[channel-1];

  if (pulso_ant != result2)

     pulso_ant = Pulses[channel-1];   // if the results are different we make a third reading (this
should be fine)


    result = (result - pulso_ant)>>1;        // Restamos con el valor del pulso anterior y pasamos
a microsegundos (reloj a 0.5us)


  if ((result > MIN_IN_PULSE_WIDTH)&&(result < MAX_IN_PULSE_WIDTH))  // Out of
range?

    return result;

  else

    {

    return Neutro[channel];

    }

}
```

```
// SERVO SIGNAL OUTPUT (USING TIMER2 OVERFLOW INTERRUPT AND TIMER1
READINGS)
ISR (TIMER2_OVF_vect)
{
  int us;
  int aux;


  if (Servo_Channel < num_servos){
    Servo_Timer2_timer1_stop = TCNT1;       // take the timer1 value at this moment


    // Now we are going to calculate the time we need to wait until pulse end
    if (Servo_Timer2_timer1_stop>Servo_Timer2_timer1_start)     // Timer1 reset during the
pulse?
      Servo_Timer2_pulse_length = Servo_Timer2_timer1_stop-Servo_Timer2_timer1_start;
    else
      Servo_Timer2_pulse_length = ((long)Servo_Timer2_timer1_stop + Timer1_last_value) -
(long)Servo_Timer2_timer1_start;
    us = (Servos[Servo_Channel].value) - (Servo_Timer2_pulse_length>>1);
    us -= 2;  // Adjust for the time of this code
    if (us>1)
      {
      us <<= 2; // Translate us to the 4 cyles loop (1/4 us)
      __asm__ __volatile__ (   //  4  cycles  loop  =  1/4  us   (taken  from  delayMicroSeconds
function)
            "1: sbiw %0,1" "\n\t"            // 2 cycles
            "brne 1b" : "=w" (us) : "0" (us) // 2 cycles
            );
      }
    digitalWrite( Servos[Servo_Channel].pin,LOW);    // pulse this channel low
    Servo_Channel++;                                 // increment to the next channel
    }
  else
    Servo_Channel = 0;                // SYNC pulse end => Start again on first channel


  if (Servo_Channel == num_servos){           // This is the SYNC PULSE
    TCCR2B = _BV(CS20)|_BV(CS21)|_BV(CS22);      // set prescaler of 1024 => 64us
resolution (overflow = 16384uS)
    TCNT2 = 0x04;   //64usx4 = 256us
    }
  else{
    TCCR2B = _BV(CS20)|_BV(CS22);                 // Set prescaler of 128  (8uS resolution at
16Mhz)
    TCNT2 = Servos[Servo_Channel].counter;           // Set the clock counter register for the
overflow interrupt
    Servo_Timer2_timer1_start = TCNT1;           // we take the value of Timer1 at the start of
the pulse
    digitalWrite(Servos[Servo_Channel].pin,HIGH);  // Pulse start. Let´s go...
    }
}


void Servo_Timer2_ini()
{
  // Servos must have correct values at this moment !! Call First Servo_Timer2_set()
function...
  // Variable initialization
  Servo_Channel = 0;
  TIMSK2 = 0;  // Disable interrupts
  TCCR2A = 0;  // normal counting mode
  TCCR2B = _BV(CS20)|_BV(CS22);              // Set prescaler of 128  (8uS resolution at
16Mhz)
  TCNT2 = Servos[Servo_Channel].counter;      // Set the clock counter register for the
overflow interrupt
```

```
  TIFR2 = _BV(TOV2);  // clear pending interrupts;
  TIMSK2 =  _BV(TOIE2) ; // enable the overflow interrupt
}

void Servo_Timer2_set(uint8_t servo_index, int value)
{
  int aux;

  if (value > SERVO_MAX_PULSE_WIDTH)
    value = SERVO_MAX_PULSE_WIDTH;
  else if (value < SERVO_MIN_PULSE_WIDTH)
    value = SERVO_MIN_PULSE_WIDTH;

  Servos[servo_index].value = value; // Store the desired value on Servo structure

  value = value - 20;  // We reserve 20us for compensation...

  // Calculate the overflow interrupt counter (8uS step)
  aux = value>>3;  // value/8
  Servos[servo_index].counter = 256 - aux;
}
```

Appendix T : Code for dot product calculation of vectors

```
//Computes the dot product of two vectors
float Vector_Dot_Product(float vector1[3],float vector2[3])
{
  float op=0;

  for(int c=0; c<3; c++)
  {
  op+=vector1[c]*vector2[c];
  }

  return op;
}

//Computes the cross product of two vectors
void Vector_Cross_Product(float vectorOut[3], float v1[3],float v2[3])
{
  vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
  vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
  vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}

//Multiply the vector by a scalar.
void Vector_Scale(float vectorOut[3],float vectorIn[3], float scale2)
{
  for(int c=0; c<3; c++)
  {
   vectorOut[c]=vectorIn[c]*scale2;
  }
}

void Vector_Add(float vectorOut[3],float vectorIn1[3], float vectorIn2[3])
{
  for(int c=0; c<3; c++)
  {
     vectorOut[c]=vectorIn1[c]+vectorIn2[c];
  }
}
```

Appendix U : Code for Matrix Multiplication

```
//Multiply two 3x3 matrixs. This function developed by Jordi can be easily adapted to multiple n*n matrix's. (Pero me da flojera!).

void Matrix_Multiply(float a[3][3], float b[3][3],float mat[3][3])
{
  float op[3];
  for(int x=0; x<3; x++)
  {
    for(int y=0; y<3; y++)
    {
      for(int w=0; w<3; w++)
      {
       op[w]=a[x][w]*b[w][y];
      }
      mat[x][y]=0;
      mat[x][y]=op[0]+op[1]+op[2];

      float test=mat[x][y];
    }
  }
}
```