ABSTRACT

Title of Thesis: DESIGN OF AVIONICS AND CONTROLLERS FOR AUTONOMOUS TAKEOFF, HOVER AND LANDING OF A MINI-TANDEM HELICOPTER

Shafiq Ur Rahman, Master of Science, 2010

Thesis directed by: Professor Gilmer Blankenship
Department of Electrical and Computer Engineering

Robotics autonomy is an active research area these days and promises very useful applications. A lot of research has been carried out on Vertical Takeoff and Landing (VTOL) vehicles especially single rotor small scale helicopters. This thesis focuses on a small scale twin rotor helicopter. These helicopters are more useful because of their power efficiency, scalability, long range of center of gravity, shorter blades and most importantly their "all lift" feature. By "all lift" we mean that unlike single rotor helicopters where tail rotor's power is wasted just to cancel the torque of the main rotor both of its rotors are used for generating lift. This makes twin rotors ideal for lifting heavy weights.

This thesis considers avionics systems and the controllers development for a twin rotor. It involves electronic component selection and integration, software development, system identification and design of zero rate compensators. The compensators designed are responsible for autonomous take-off, hover and landing of this unmanned aerial vehicle (UAV). Both time and frequency domain system identification approaches were evaluated and a selection was made based on hardware limitations. A systematic approach is developed to demonstrate that a rapid prototyping UAV can be designed from cheap off-the-shelf components that are readily available and functionally compatible. At the end some modifications to existing mechanical structure are proposed for more robust outdoor hovering.

DESIGN OF AVIONICS AND CONTROLLERS FOR AUTONOMOUS TAKEOFF, HOVER AND LANDING OF A MINI-TANDEM HELICOPTER

by

Shafiq Ur Rahman

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2010

Dedication

To my beloved parents for their continuous support and inspiration

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, Professor Gilmer Blankenship for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past year. He has been very co-operative and considerate throughout this period.

I would also like to thank Dr. Gaurav Bajpai at Techno-Sciences Inc. His guidance made this thesis practically possible. Thanks are due to Professor Christopher Davis and Professor Agis A. Iliadis for agreeing to serve on my thesis committee and for sparing their invaluable time reviewing the script.

I would also like to thank Mr. Pratik Gupta for his insightful ideas in developing electronics for the helicopter. He always lent a helping hand and gave very useful suggestions during the entire design process.

I would also like to acknowledge help and support from some of the staff members. Jay Renner at the Autonomous Systems Lab has been very kind and always went out of the way to help me.

I owe my deepest thanks to my family - my mother, father and three brothers (Muhammad Qaisar, Altaf Gohar, Saif Ur Rehman) who have always stood by me and guided me through my career. They always supported me to realize my dreams and placed my choice ahead of theirs.

I can't forget here my useful discussions on almost every aspect of the life with my friends: Abdul Rehman, Ghulam Ali, Malik Muhammad Usman Gul, Waqas Ahmad and Zaheen Gul.

My housemates at my place of residence have been a crucial factor in finishing my work smoothly. I'd like to express my gratitude to Hamza Shakeel, Tamoor Gandapur, Muhammd Akbar, George Zoto, Awais Khawar and Waseem Ansar Malik for their friendship and support. I would like to thank Waseem bahi specially for his sincere advices during my stay at University of Maryland.

Thank you all!

Table of Contents

List of Tables

List of Figures

List of Abbreviations

UAV	Unmanned Aerial Vehicle
ASL	Autonomous Sytems Lab
UMD	University of Maryland College Park
CP	Collective Pitch
DCM	Direction Cosine Matrix
CCPM	Collective and Cyclic Pitch Mixing
VTOL	Vertical Takeoff and Landing
MEMS	Microelectromechanical Systems
LiPo	Lithium Polymer
LAN	Local Area Network
MAN	Metropolitan Area Network
PAN	Personal Area Network
DSSS	Direct Sequence Spread Spectrum
OEM	Original Equipment Manufacturers
RF	Radio Frequency

Chapter 1

Introduction

1.1 Background and Motivation

An Unmanned Aerial Vehicle (UAV) as the name suggests is a remote-controlled or completely autonomous vehicle designed to carry out a pre-specified task in a particular way. The vehicle is either programmed or trained beforehand to accomplish such a task [1].

Development of remotely controlled vehicles is as old as the 1950s. However, the history of Vertical Take-Off and Landing (VTOL) UAVs started in early 1960s when the US Navy studied the feasibility of such a vehicle for the first time. Some close and short range (50km and 200km respectively) vehicles were developed. Recent advances in controls, micro-electronics, microelectromechanical systems (MEMS) and wireless communication have led to the development of long range UAVs. Currently long range i.e. endurance aircraft is being developed that is meant to be used for cinematography, search, surveillance and transportation.

Research is being carried out in different universities to develop a completely autonomous vehicle that is able to perform complex tasks on its own and different groups are exploring new techniques to make this concept practical. Towards this end engineers are exploring different configurations of VTOLs to exploit specific advantages associated with their particular design. For example 'Autonomous Helicopter' group at Carnegie Mellon University is working on vision based stability. A group at Stanford University is trying to achieve acrobatic maneuvers using apprenticeship and reinforcement learning. In simple words, *reinforcement learning* involves adaptive algorithms that a machine learns by observing actions taken by an intelligent teacher agent, which in most cases is human, in certain environmental situations [25].

Previously significant contributions in this area are [2] and [3]. Pieter Abbeel worked on acrobatic maneuvers using machine learning while Vladislav Gavrilets used a conventional approach for system identification and controller development. But each of the above researchers had focused on single rotor helicopters. I, in this thesis, have attempted to work on the stability of twin rotor tandem helicopters and will discuss some stability issues particular to this class of helicopter. The basic reason for choosing this vehicle is its all-lift feature. By that I mean, unlike single rotor helicopters where tail rotor's energy is wasted in counteracting the main rotor's torque, both rotors in tandem helicopter are used for generating lift. That is why tandem helicopters are very useful in lifting heavy weights. As an example, according to the Boeing website, a Chinook (CH-47D) helicopter is able to lift another Chinook equal in size and weight.

1.2 Main Challenges

In designing a system like a UAV one need to take care of certain issues. The basic consideration in designing an autonomous or remote control aircraft is the choice of electronics. It should have minimum but sufficient electronics to carry out complex maneuvering tasks. The weight and placement of electrical components also play an important role and should be distributed carefully about the center of gravity. The system also needs to have a robust communication link because in case of an autonomous vehicle it would be utilized in sending important information back to a base station.

The major challenge in this thesis was integration of different off-the-shelf components and modifying their firmware to meet the timing and rate synchronization requirements among them. Even in the design phase i.e. system identification, which is not needed for the vehicle once the model is established, the hard thing was to keep track of input/output time periods for analysis purposes.

Another issue was to mount a rigid platform on the helicopter that can carry all the electronics and sensors. Again this platform has a weight and care was taken to mount it so that the overall weight distribution remained unchanged.

Since the helicopter was built starting only from the mechanical structure a lot of things were adjusted manually and by performing experiments, which included but are not limited to adjustment of:

(1) the servos/swashplates linkages for collective pitch

(2) the position given to each servo for varying collective and cyclic pitch

(3) the calibration of collective and cyclic pitch using a pitch gauge

(4) the throttle given to brushless DC motor

(5) the gear ratio for enough RPM to take-off

(6) the position of Inertial Measurement Unit (IMU) etc

1.3 Thesis Overview

The objective of this thesis was to demonstrate that one can develop a small scale autonomous vehicle having many interesting applications without designing custom electronics for it. The hardware used should be inexpensive and readily available in the market. By attempting to focus on a twin rotor helicopter I want to prove that making it autonomous would bring a lot more exciting applications to small scale UAVs, which include search, surveillance and inexpensive traffic monitoring etc. Make it large and it can be used to rescue an injured person or a person in a life threatening situation like someone stuck on a cliff or in flood where other UAVs fail. A more frequent utilization would be to transport stuff from one location to other.

In Chapter 2, I will explain in detail the electrical components used, their specifications and their integration in the over all system along with the practical details about the issues faced in this process. The reasons for choosing any particular component will follow its description.

In Chapter 3, I will illustrate how I developed the model for the system. I will also list what time and frequency domain techniques for system identification were employed and why I relied on a particular technique.

Finally in chapter 4, I will mention possible future work that can lead this project to become a more useful reality. I would also recommend some modifications in the mechanical structure of the helicopter that I think can be useful for prototyping of the vehicle.

In general, I will follow the same approach that I used to design the vehicle, listing the components in the same way as I selected them so that it can be made easy for the reader to understand.

Chapter 2

Hardware Design

The choice of hardware in any UAV is dependent upon a number of criteria, which include, but are not limited to, compatibility with other components, light weight, cost, ease of integration in the system and the flexibility in firmware. Fortunately most of these criteria are design considerations of companies like Sparkfun, a company based in Boulder Colorado, whose products I have used in this project. Below is the list of all the hardware (mechanical and electronics) components that I have used and which I will subsequently explain in detail:

(1) Twinn Rexx Kit

(2) Brushless DC Motor

(3) Electronic Speed Controller (ESC)

(4) Battery Eliminator Circuit (BEC)

(5) Metal Gear Servos

(6) Ultrasonic PING sensors

(7) Arduino Mega

(8) Inertial Measurement Unit (IMU)

(9) Power Board

(10) eLogger

(11) Battery

2.1 Hardware Components Detail

2.1.1 Twinn Rexx Kit

The helicopter kit, shown in Figure 2.1, known as Twinn Rexx was bought from Tech Model Products. The Twinn Rexx is basically a mini tandem helicopter designed to use the very popular T-Rex rotor heads and drive components. It has as much 3D flying capabilities as possible that are inherent with the tandem configuration [10].

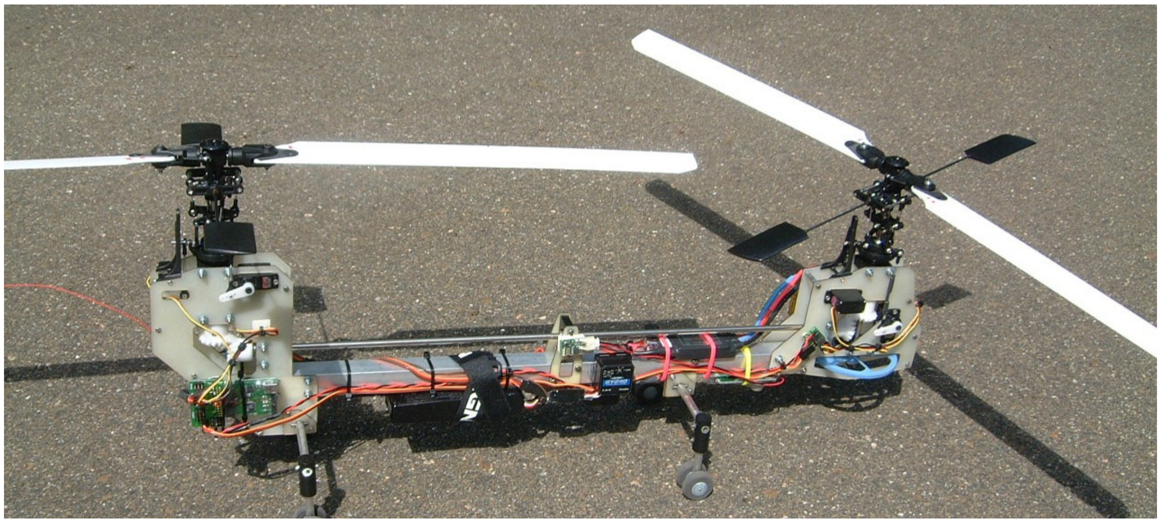


Figure 2.1: Twinn Rexx Kit [10]

Also, the model is designed to give a sport scale CH47 Chinook appearance with special attention given to the accessibility of the internal subsystems. The model originally had the following components:

- Motor: Medusa 028-040-2500 with 12T pinion
- ESC: Align 35A w/Tech Ferrite Ring Glitch Filter
- Battery: TP 4S1P 2070 mAh 25C
- Servos: 6 HS65HB
- Receiver: Futaba R146ip
- Antenna: Tech Micro Antenna
- Yaw Axis Gyro: Futaba GY240 (AVCS off)
- Pitch Axis Gyro: Futaba GY240 (AVCS on) DCP Channel
- BEC: 3.5 amp Capacity at 5 volts; Medusa MR-BEC-45035
- Controller: Tech TH-2 (Tandem SW Version 2B005)
- Rotor heads: Stock Align 450 XL TRex
- Blades: Tech 325mm High Performance Blades
- Length: Shaft to shaft distance is 20.25 inches

Since I had to design whole of the electronics of the helicopter again, I replaced all the control and communication related parts like yaw and pitch gyros, antenna and receiver etc with new parts. All the difficult looking terms/abbreviations above will be explained in detail in later sections. Here is the detail of the parts that I used:

2.1.2 Brushless DC Motor

Brushless Direct Current motors (or BLDC motors) also known as electronically commutated motors are synchronous electric motors powered by DC electricity and having electronic commutation systems, rather than mechanical commutators and brushes. The current-to-torque and voltage-to-speed relationships of BLDC motors are linear [25].

BLDC motors may also be described as *Stepper motors*, with fixed permanent magnets and possibly more poles on the stator than the rotor. However, the term stepper motor tends to be used for motors that are designed specifically to be operated in a mode where they are frequently stopped with the rotor in a defined angular position [25].

BLDC motors are more power efficient than their corresponding brushed counterparts which means that they can utilize electric current better to convert it to mechanical rotation. This improvement in the performance is mainly due to the absence of electrical contacts between the stationary and rotating parts of the electrical connection. There is no frictional loss in BLDC motors that resulted in increased efficiency which is better in the no-load and low-load region of the motor's electric current-to-rotation curve. However it is interesting to note that under high mechanical loads, BLDC motors and high-quality brushed motors have similar performance.

Here are a few considerations that motivate choosing the specific brushless DC motor:

2.1.2.1 Difference between Brushed and Brushless Motor

A brushed motor utilizes stationary and moving metallic contacts that brush against each other. These contacts are used to transfer electric current from a stationary battery plug to a rotating armature. A brushless motor, on the other hand has no electrical contacts at all. It has stationary coils inside a cylinder with a couple of permanent magnets that rotate around the coil. The cylinder is connected to the output shaft where usually a gear is mounted. The coils are grouped together into phases, and an electronic motor controller powers up each coil in sequence thus making it an electromagnet, causing the external cylinder and shaft to rotate. [4]

2.1.2.2 RPM, Kv and Current Rating

RPM stands for the number of rotations per minute or revolutions per minute, and is a measure of how fast a motor spins. Brushless motors are also given a *Kv* rating, which is basically number of RPM (K) per volt. It indicates how fast the motor will rotate with a given voltage supplied to it.

A 980Kv motor powered by an 11.1 volt battery would spin at 980 x 11.1 = 10878 RPM with no load. The *Current Rating* specifies the maximum continuous and/or burst current that the motor is able to handle. When choosing a battery and electronic speed controller, we choose ones with continuous current ratings equal to or greater than that of the motor.[4]

2.1.2.3 Inrunner vs Outrunner

Brushless DC motors can be classified into two types based on their design. An *inrunner motor* has stationary coils attached to cylindrical structure while the shaft is connected to the rotating magnet at the center. An *outrunner motor* has stationary coils at the center and the rotating magnet along with the cylindrical structure on outside is connected to the shaft. Outrunner motors generally have lower Kv ratings, meaning they run at a lower speed, but have more torque, which would allow one to directly drive larger props without needing a gearbox. So they are ideal for use in helicopters and planes. However, most RC cars and boats would require an inrunner brushless motor. [4]

2.1.2.4 Tractor vs Pusher

A *Tractor motor* is used in Radio Controlled (or RC) planes for driving the plane and are usually located on the nose of the plane. The *Pusher motors*, on the other hand, are used for pushing the aircraft forward and are usually mounted at the tail of the plane. Based on the position of the motor in plane, a design is sometimes referred as a pusher/tractor configuration.

2.1.2.5 560TH Outrunner-Tractor BLDC motor

Keeping the above criteria in mind, I used a 560TH tractor BLDC motor made by Just Go Fly (shown in Figure 2.2). Its a 5mm axle, 1650Kv motor which is specifically designed for use in RC aircrafts and VTOLs.

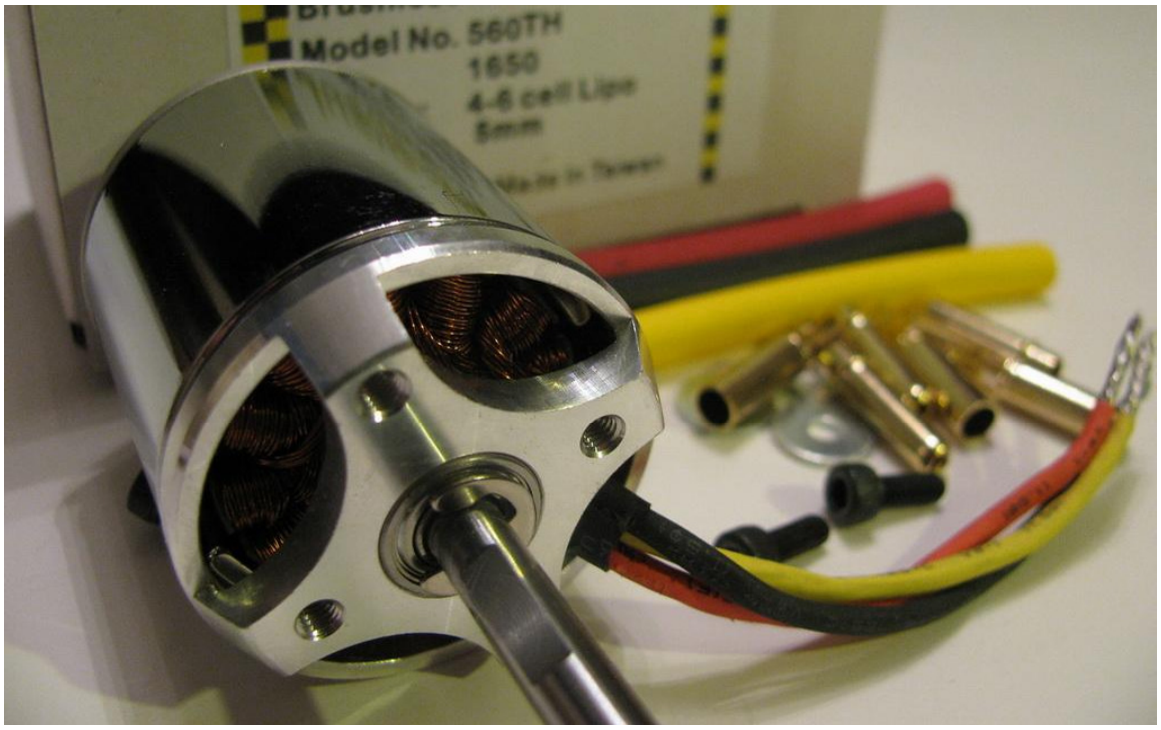


Figure 2.2: 560TH Brushless DC (BLDC) Motor

As discussed above the motor can run at a top speed of 24420 RPM at 14.8 volts. Since I am using a 12 teeth pinion and my drive gear has 150 teeth (came installed on the kit), the max prop RPM that i can achieve is 1953 without load. This RPM can be increased or decreased by using more or less teeth on the pinion gear but care should be taken as this would affect the torque of the motor and hence the lift of helicopter. This motor has amperage of 2.5 amp at no load and up to 45 amp at maximum load.

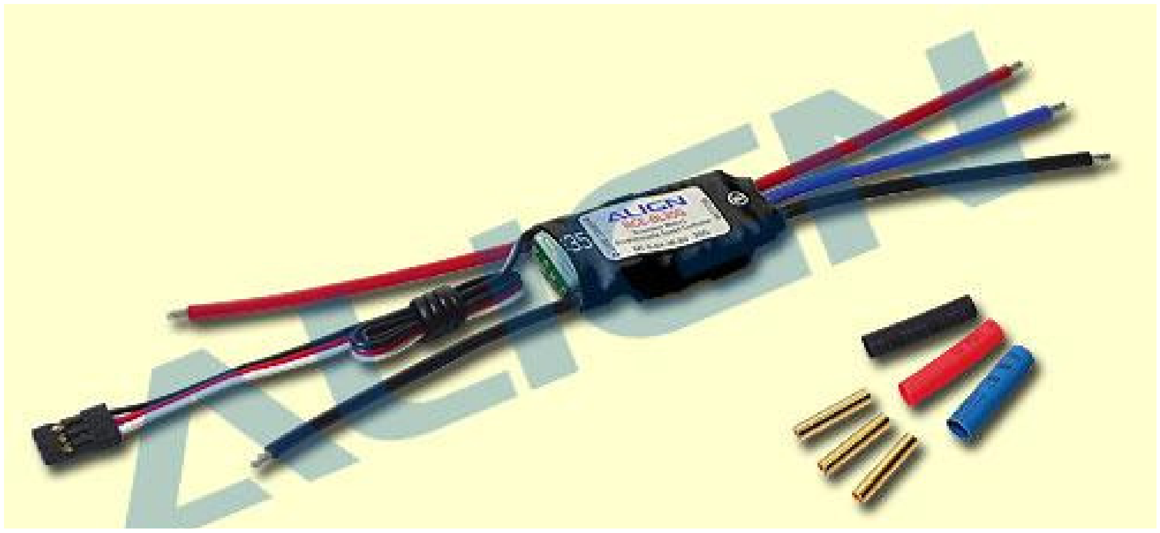


Figure 2.3: RCE-BL35G Electronic Speed Controller (ESC)

2.1.3 Electronic Speed Controller (ESC)

An Electronic Speed Controller (or ESC) is an electronic circuit designed to convert a DC voltage to three square waves, each 120^o out of phase with respect to the other, for driving the motor. The properties of the output are dictated by a controller's Pulse Width Modulation (PWM) signal. It can also reverse the direction of the motor and apply dynamic brakes. This gives user a fine control and makes the motor run smoother than the corresponding brushed counterpart [25].

Most modern ESCs have a buit-in battery eliminator circuit (BEC) to regulate voltage for the receiver, removing the need for extra receiver batteries. BECs are usually either linear or switched mode voltage regulators.

2.1.3.1 ESC Working

DC ESCs are basically PWM controllers for brushless DC motors. Controlling an ESC is similar to controlling a servo motor. A user needs to provide a pulse at the rate of 50 Hz or a pulse every 20 milli-seconds (ms) to the ESC. The on-time or pulse width of the wave varies from 1 ms to 2 ms which encodes the control information. When supplied with a 1 ms pulse width at 50 Hz which corresponds to 0 percent duty cycle, the ESC turns off the DC motor. A 1.5 ms pulse-width input signal results in a 50 percent duty cycle, ESC outputs signal that drives the motor at approximately half the maximum speed. When presented with a 2.0 ms input signal, the motor runs at full speed due to the 100 percent duty cycle. In some ESCs the pulse width can be varied up to 2.5ms. Arduino has a 'Servo' library that can generate and keep providing a pulse of commanded width in micro-seconds (μs) every 20ms.

2.1.3.2 RCE-BL35G features

I chose ALIGN ESC RCE-BL35G, shown in Figure 2.3, which is capable of supplying 35A continuous current and burst currents up to 45A. It supports 2 - 10 poles in and out-runners and a maximum RPM of 630,000 for a 6 pole motor. All of these specifications are well above the range of specs of 560TH. In the following, I will explain some key features of this ESC, as given in [15], and how to setup different modes:

1. **Brake**: No brake/Soft brake/Hard brake

2. **Electronic Timing**: Low Timing/Mid Timing/High Timing

   Electronic timing is used to either fine tune a bit more power or a bit more efficiency out of the electronic system. With brushless motors, all of the commutation, including the timing, is done in the ESC not the motor.

3. **Battery Protection**: 70%/65%/60%

   This feature is meant to avoid over discharging of battery. So for a 3S Lithium Polymer (LiPo) battery with 4.2 volts per cell the net voltage is = 4.2 x 3 = 12.6 volts. When battery voltage drops below 12.6 x 60% = 7.56 volts the ESC decreases the power and lets the aircraft land slowly.

4. **Aircraft Option**: Normal Airplane/Helicopter 1/Helicopter 2

   Normal airplane is applied to general airplanes and gliders. For helicopters we can use helicopter 1 or helicopter 2. Helicopter 1 mode has soft start feature while helicopter 2 mode has a governor mode as well.

   Governor Mode: In this mode the ESC maintains a constant RPM for the helicopter by varying collective pitch and throttle in a controlled way.

5. **Thermal Protection**: When the temperature of ESC increases to 80°F the ESC decreases the power

6. **Safe Power Alarm**: The alarm will sound if the receiver is turned on and there is no transmitter signal at all. This is helpful in fixing the problem before one flies the machine.

7. **Aircraft Locator**: If aircraft lands in an unexpected location where it is difficult

to find it, switching transmitter off will sound alarm in the aircraft after 30 sec.

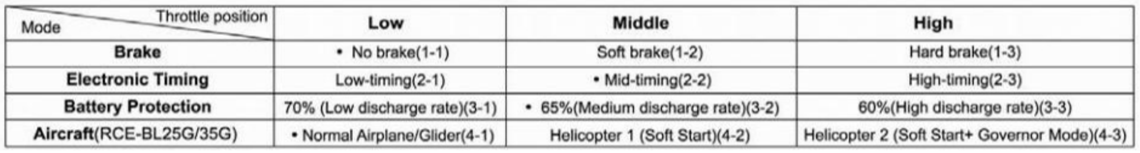
It is important to note that braking feature of ESC is not used for RC helicopters because it can cause severe damage to the rotor assembly and meshing gears. Also there is no reverse motor direction during the operation. Usually the direction of a DC motor is reversed manually by interchanging any two of its three input wires.

2.1.3.3 Modes of Operation

There are two operating modes in which ESC functions: Setup Mode and Using Mode. In the following, I will briefly explain what each means.

1. Setup Mode: In order to enter ESC in setup mode we need to set the transmitter throttle stick to full throttle and then turn on the ESC in the receiver, it will produce a sequence of sounds to confirm the setup mode. The user can now place the throttle stick at low/mid/high positions to choose among three different choices for each of the following settings: Brake, Electronic timing, battery protection and aircraft type. The possible settings in the case of this ESC are shown in Figure 2.4.

2. Operating Mode: In order to put the ESC in operating mode we have to first keep the transmitter threshold stick at its lowest position and then turn on the receiver. The ESC will emit an operating mode confirmation sounds followed by a number of sounds confirming the current settings of the ESC. After that one



Mode \ Throttle position	Low	Middle	High
Brake	• No brake(1-1)	Soft brake(1-2)	Hard brake(1-3)
Electronic Timing	Low-timing(2-1)	• Mid-timing(2-2)	High-timing(2-3)
Battery Protection	70% (Low discharge rate)(3-1)	• 65%(Medium discharge rate)(3-2)	60%(High discharge rate)(3-3)
Aircraft(RCE-BL25G/35G)	• Normal Airplane/Glider(4-1)	Helicopter 1 (Soft Start)(4-2)	Helicopter 2 (Soft Start+ Governor Mode)(4-3)

Figure 2.4: RCE-BL35G Setup Mode [15]

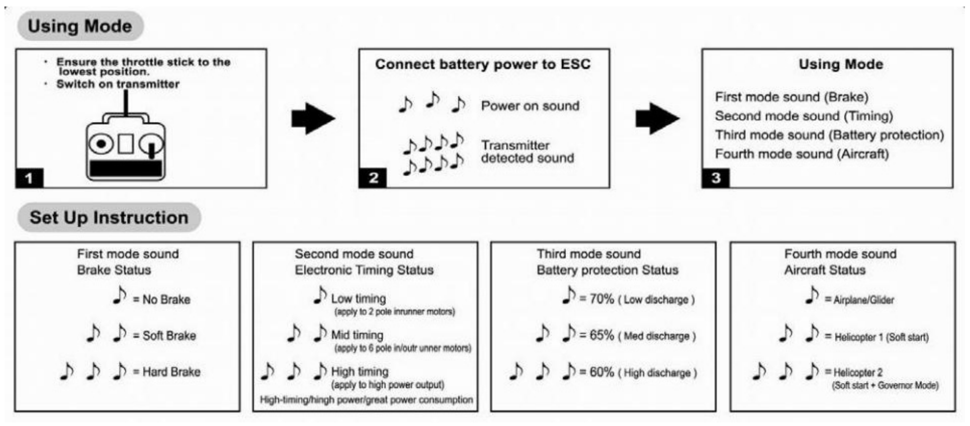

Using Mode
· Ensure the throttle stick to the lowest position.
· Switch on transmitter
1
Connect battery power to ESC
Power on sound
Transmitter detected sound
2
Using Mode
First mode sound (Brake)
Second mode sound (Timing)
Third mode sound (Battery protection)
Fourth mode sound (Aircraft)
3
Set Up Instruction
First mode sound
Brake Status
= No Brake
= Soft Brake
= Hard Brake
Second mode sound
Electronic Timing Status
Low timing
(apply to 2 pole inrunner motors)
Mid timing
(apply to 6 pole in/outr unner motors)
High timing
(apply to high power output)
High-timing/hingh power/great power consumption
Third mode sound
Battery protection Status
= 70% (Low discharge)
= 65% (Med discharge)
= 60% (High discharge)
Fourth mode sound
Aircraft Status
= Airplane/Glider
= Helicopter 1 (Soft start)
= Helicopter 2
(Soft start + Governor Mode)

Figure 2.5: RCE-BL35G Using Mode [15]

can vary the transmitter throttle to vary the speed of the brushless DC motor. The sound sequences the ESC should produce are given in Figure 2.5:

2.1.3.4 Transmitter Emulator for ESC Setup

In my project, since I was not using a standard RC transmitter and receiver I had to emulate the functioning and output of the transmitter. In my setup, shown

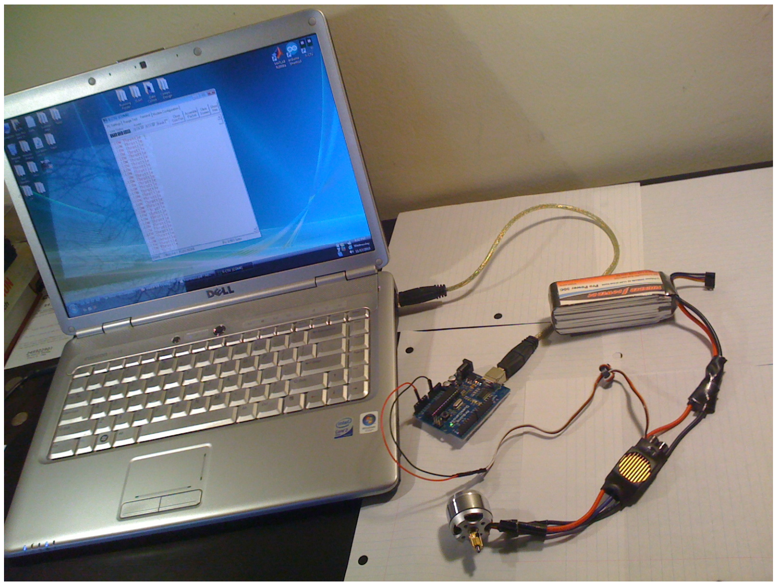
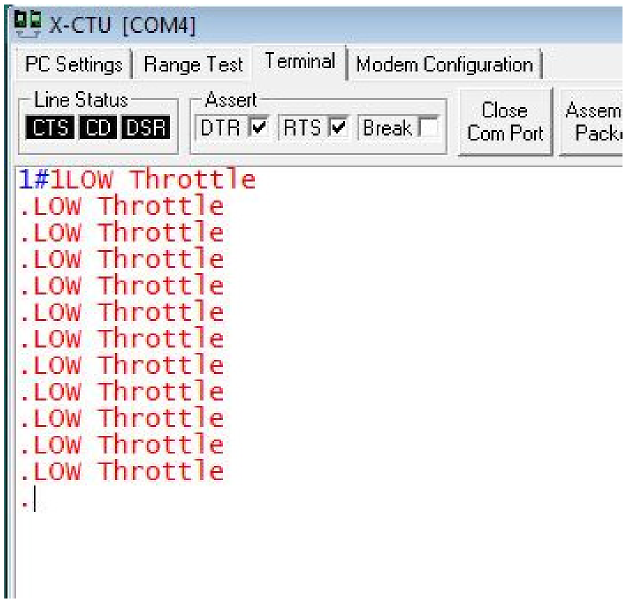


Figure 2.6: Transmitter Emulator Circuit

in Figure 2.6, I hooked up my ESC to an Arduino board for PWM signal on one side and to the motor and battery on other side. The Arduino is connected serially to the computer where a terminal program, X-CTU in my case, is used to communicate command signals to Arduino to output a particular signal to the ESC signal input.

I have written an Arduino code, attached in Appendix A.1, for transmitter emulation. This code serially receives a command in the format of 1#, 2# or 3# that corresponds to low throttle (Transmitter stick at lowest position), Mid Throttle (Transmitter stick at middle stick) and High Throttle (Transmitter stick at highest position). I follow all the steps listed above for Operating and Setup mode, however, instead of the throttle stick position I input a command and the Arduino then outputs the corresponding emulated signal to the ESC. A snapshot of the X-CTU window is shown in Figure 2.7.



```
1#1LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.LOW Throttle
.
```

Figure 2.7: Transmitter Emulator Output

2.1.4 Battery Eliminator Circuit (BEC)

BEC stands for battery eliminator circuit. This circuit is commonly used for powering the servos and receiver. A few amps of current is used depending upon the servos in the circuit when those are moved. It is directly connected to the battery and gives out regulated voltage. The current rating of BECs are usually high, which means that they are capable of providing high currents at low voltages. They are usually equipped with a circuitry which senses low battery voltage. It enables them to reduce the power to the propellers so that the system can land safely.

2.1.4.1 MR-BEC 45035 Specifications

I used *Medusa Research MR-BEC 45035* shown in Figure 2.8. This 3.5 amp BEC is very useful for larger models since it is capable of enduring currents up to

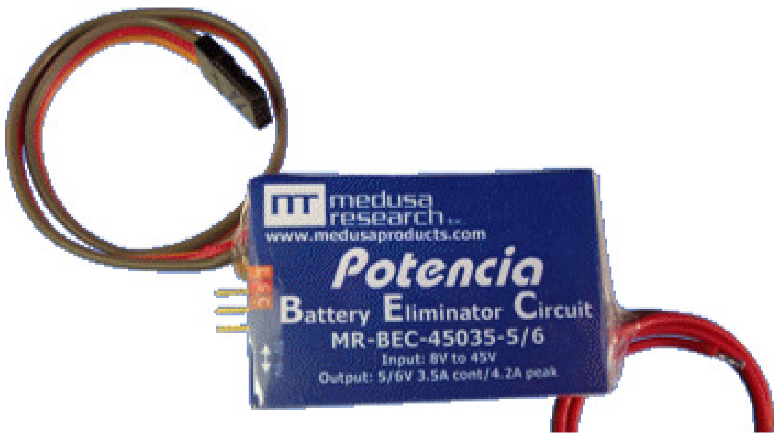


Figure 2.8: MR-45035 Battery Eliminator Circuit (BEC)

3.5 amp and input voltages up to 45 volts. Moreover, it is very light, weighing only 13 grams while a typical receiver and servo battery pack weighs approximately 7 times more. It can therefore boost the performance of the aircraft by improving its thrust to weight ratio. It is also efficient in terms of space as its size is 30x50mm. However, it requires a minimum input voltage of 8 volts so it is used with battery packs of 2 or more cells.

If separate battery packs are used for receiver and servo, they are mostly not equipped with battery failure protection. BECs provide a solution to this problem by tying them with the primary battery whose voltage protection is already incorporated in the ESC.

2.1.5 Metal Gear Servos

A servo or servomechanism is an automatic device that uses feedback to correct the performance of a mechanism. This term correctly applies only to systems where the feedback or error-correction signals help control mechanical position or other parameters. A common type of servo provides position control [5]. Servos calculate their current positions by some sort of transducer and by subtracting from the desired position they get the error that they have to traverse back using negative feedback.

RC servos are used for actuation of various mechanical systems like controlling position of a shaft etc. They are composed of an electric motor linked to a potentiometer. PWM signals sent to the servo are translated into position commands by electronics inside the servo. When a servo is asked to rotate to a particular position the potentiometer readings are constantly converted to equivalent position and PID control algorithms are used to lock it to the desired position. Servos are used in robotics applications because of their reliability and simplicity of control.

2.1.5.1 Servo Working

A servo has usually three wires: ground, power, and signal. The servo traverses an angle according to the pulses received over the signal wire, which in a practical situation sets the angle of an actuator. Theoretically, a pulse indicating a particular position should be sent every 20 ms. The width of the servo pulse dictates the range of the servo's angular motion.

The width of a servo pulse usually varies between 1 ms to 2 ms. A pulse of 1.5 ms therefore places it to its center position or 45^o, a pulse of 1.25 ms could set it to 0^o and a pulse of 1.75 ms to 90^o. The physical limits of the servo hardware

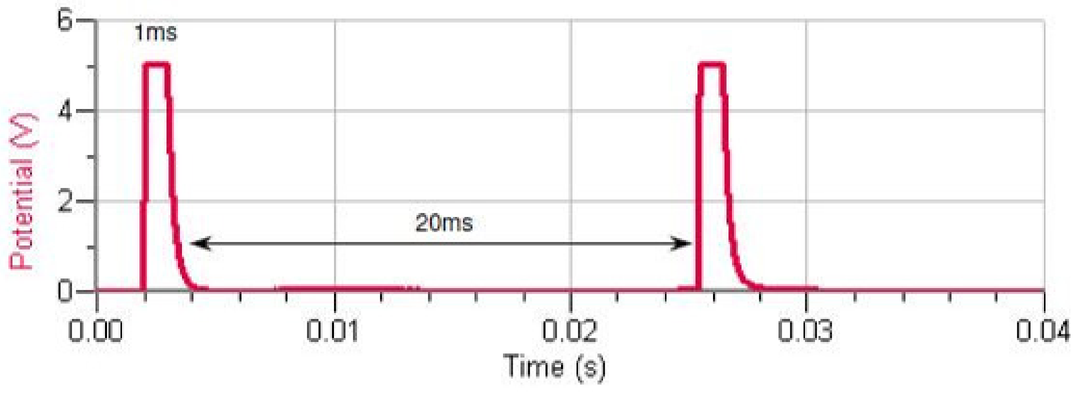


Figure 2.9: A train of 1ms pulses with 20ms delay between them

varies between brands and models, but a general servo's angular motion will travel somewhere in the range of 90^o - 120^o and the neutral position is almost always at 1.5 ms. Figure 2.9 shows a train of servo's input pulses of 1ms separated by 20ms graphically.

2.1.5.2 Servo Speed

Servo speed is usually specified in seconds. A servo has a 0-60 degree time just as a car has a 0-60 mph time. (Figure 2.10). The lower the time to reach 60 degrees the faster the servo operates. The servo speed for HS-65MG is 0.11 sec @ 60 degree which means this servo can cover a 60 degree commanded angle at 6 volts in 0.11 sec. This speed is very important for us as it tells us the limitations of our helicopter

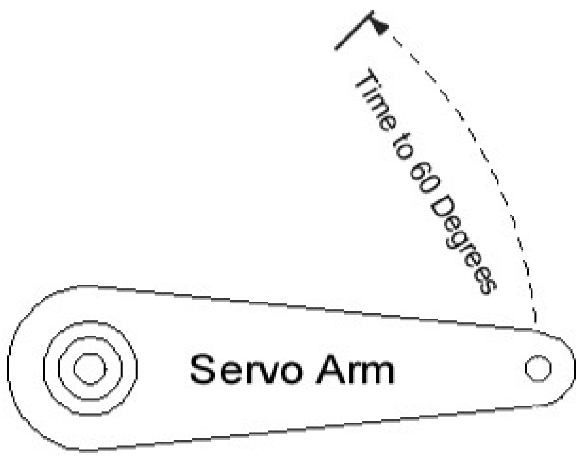


Figure 2.10: Servo Speed

agility. This number has a one to one relationship with the speed of variation of collective and cyclic pitch of rotors. The particular consideration for the choice of this servo was its metal gears that have least wear and tear.

2.1.5.3 Servo Power

Servo power is stated as ounce-inches (oz-in). This is the maximum amount of power the given servo will apply with a one inch arm on an actuator. For example, if a servo has a power rating of 16 oz-in, the maximum amount of power it will be able to apply with a 1" arm will be 1 lb. If you want to find out how many pounds a servo will lift or push with a 1" arm, simply divide the oz-in. number by 16. (Note: 16 ounces = 1 lb as shown in Figure 2.11).

Since I am using a 1" arm with my servo HS-65MG which has a power of 31 ounces/in, this means that it can lift 31/16 = 1.9375 lbs of weight and since I am using three such servos at 120^o apart so total weight of 3x1.9375 = 5.8125 lbs can

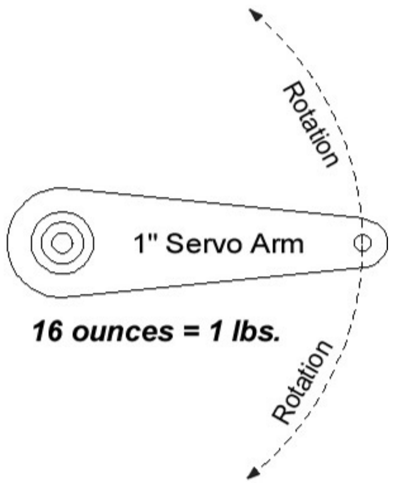


Figure 2.11: Servo Power

be lifted. This indicates that the three servos have enough power to lift the rotors as the total weight of the helicopter used is 4.5 lbs.

2.1.5.4 HS-65MG Metal Gear Servo

The HS-65MG Metal Gear feather servo, shown in Figure 2.12, is considered as one of the most powerful servos in the micro class. The HS-65 is offered with a choice of “shock resistant” metal or “zero wear” Karbonite gears. The 65 series features a top-ball bearing for long life and positive centering, 31 ounces/inch of power at 6 volts and is an excellent choice for higher performance micro helicopters, electric park flyers, and 1/18 scale cars [22].

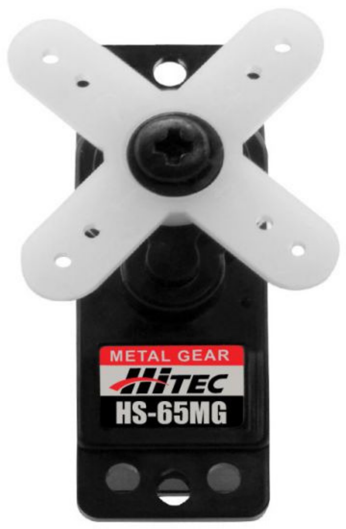


Figure 2.12: HS-65MG Metal Gear Servo

2.1.6 Ultrasonic PING sensors

Parallax's PING or commonly referred PING))) ultrasonic sensor (shown in Figure 2.13) provides a very low-cost and easy method of distance measurement. This sensor measures distance between moving and/or stationary objects. Naturally, robotics applications are very popular but it also finds applications in security systems or as an infrared replacement if so desired. It has a single I/O pin and an activity status LED.

The PING sensor measures distance using sonar. It generates an ultrasonic pulse which bounces back from the target. The time the wave hits the sensor back is noted and the time difference between the transmitted and received wave is calculated by an external processor which in turn accounts for twice the distance to the obstacle. In other words, the output from the PING sensor is a variable-width

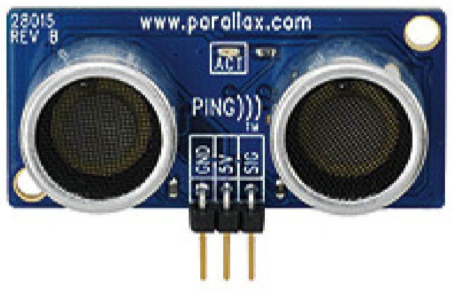


Figure 2.13: Parallax Ultrasonic PING))) sensor

pulse that corresponds to the distance to the target.

Since PING has only one pin that is used both as input and output therefore it is difficult to interface. Initially the pin is used to trigger the ping sensor so its an input pin. As soon as the ultrasonic pulse is generated the pin switches to an output pin and starts listening for echo return pulse. An onboard three pin header allows it to be plugged into a solderless breadboard and to be connected to its host through a standard three-pin servo extension cable [34]. Figure 2.14 shows pictorially how PING sensor measures distance.

2.1.6.1 Key Features

Here are the key features of the PING sensor:

1. Provides precise, non-contact distance measurements within a 2 cm to 3 m range

2. Simple pulse in/pulse out communication

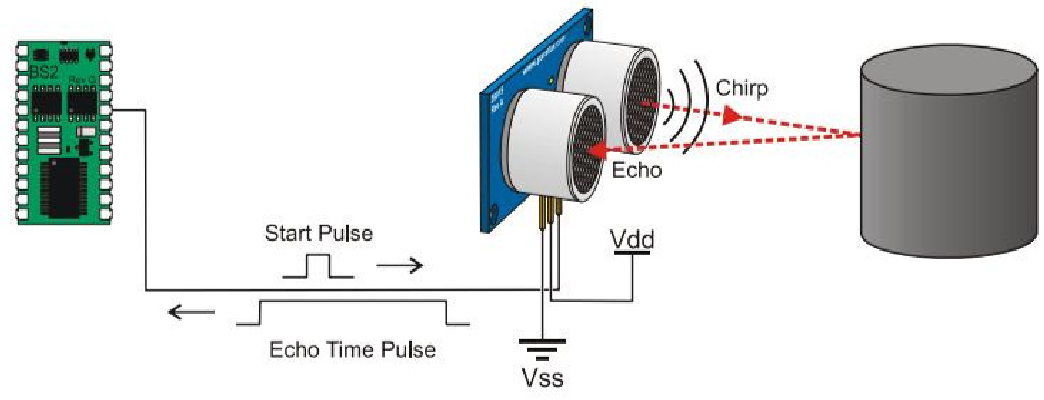


Figure 2.14: PING distance measurement [34]

3. Input Trigger: 2μs to 5μs pulse typically

4. Echo Pulse: positive TTL pulse, 115μs min - 18.5ms max

5. Burst indicator LED shows measurement in progress

6. 20 mA power consumption

7. Narrow acceptance angle

8. 3-pin header makes it easy to connect using a servo extension cable, no soldering required

9. Power requirements: +5 volts DC

10. Communication: Positive TTL pulse

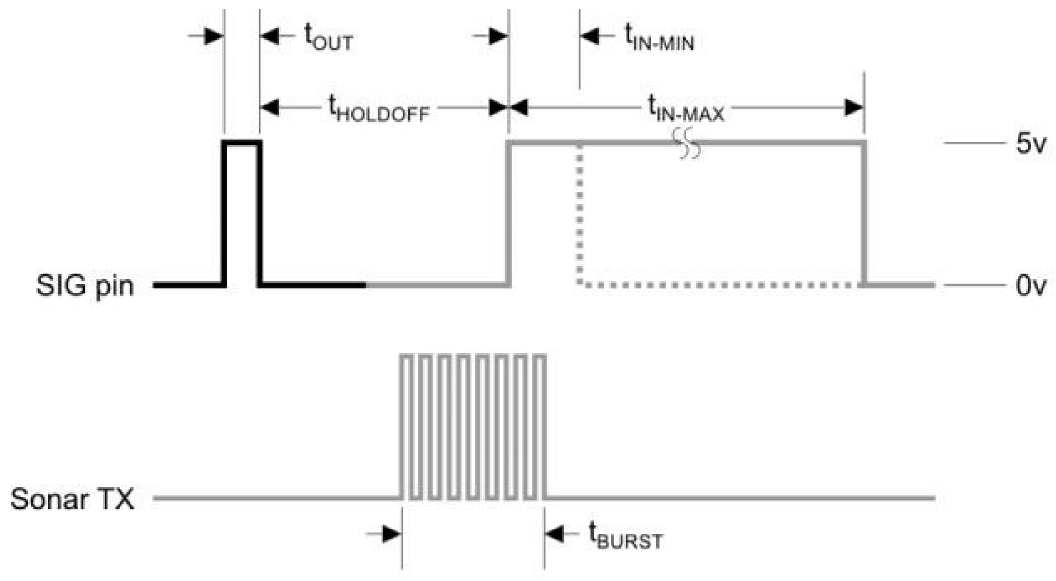


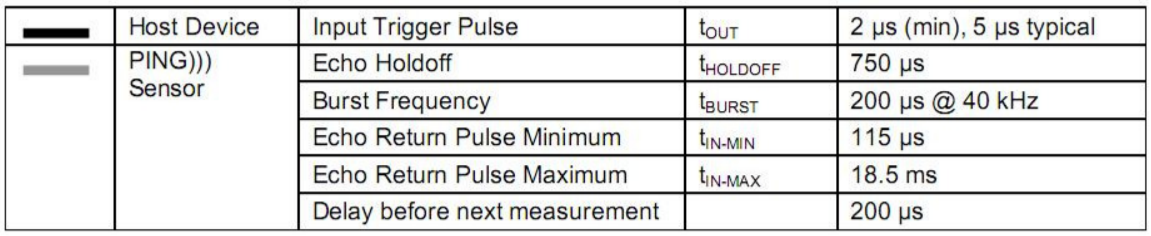
Figure 2.15: PING communication protocol [34]

2.1.6.2 Communication Protocol

The PING sensor is manufactured by Parallax Inc. It calculates the distance to an object by emitting a 40 kHz ultrasonic wave and listening to it. The time difference from the point the ultrasonic wave leaves the sensor to the point when it hits it back is noted and corresponds to the distance between the sensor and the object. Figure 2.15 shows the timing diagram of the protocol and Figure 2.16 shows the maximum time limits for which the controller waits before calculating distance.

2.1.7 Arduino

The Arduino is both an open source electronics platform and and an open source environment. The Arduino environment is used to program Arduino electronics platform which is basically a breakout board for specific Atmel microcontroller.



▬	Host Device	Input Trigger Pulse	t_{OUT}	2 μs (min), 5 μs typical
▭	PING))) Sensor	Echo Holdoff	$t_{HOLDOFF}$	750 μs
		Burst Frequency	t_{BURST}	200 μs @ 40 kHz
		Echo Return Pulse Minimum	t_{IN-MIN}	115 μs
		Echo Return Pulse Maximum	t_{IN-MAX}	18.5 ms
		Delay before next measurement		200 μs

Figure 2.16: Protocol time limits [34]

It is a flexible, easy-to-use and inexpensive hardware. The board has a number of features which will be explained later. This board can be directly programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer [16].

Here I will list two particular type of Arduino boards that are used in my system:

2.1.7.1 Arduino Duemilanove

The Arduino Duemilanove (2009) is an Atmega168 or Atmega328 based microcontroller board, shown in Figure 2.17. 'Duemilanove' means 2009 in Italian and is named after the year of its release. The Duemilanove is the latest in a series of USB Arduino boards.

1. **Summary**

   Microcontroller: Atmega168

   Operating Voltage: 5 volts

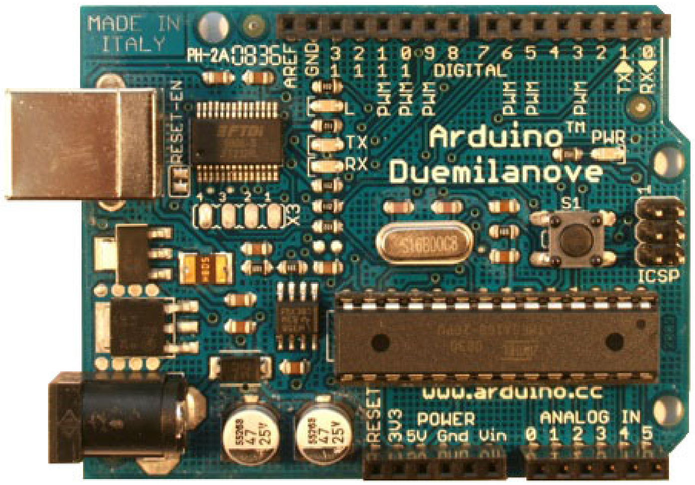


Figure 2.17: Arduino Duemilanove

Input Voltage (recommended): 7-12 volts

Input Voltage (limits): 6-20 volts

Digital I/O Pins: 14 (6 PWM)

Analog Input Pins: 6

DC Current per I/O Pin: 40 mA

DC Current for 3.3V Pin: 50 mA

Flash Memory: 16 KB (Atmega168) or 32 KB (Atmega328)

SRAM: 1 KB (Atmega168) or 2 KB (Atmega328)

EEPROM: 512 bytes (Atmega168) or 1 KB (Atmega328)

Clock Speed: 16 MHz

2. **Power**

The Arduino Duemilanove has three power input alternatives. Either it can be powered by USB, power jack or two power-in pins. The board has a USB Type B port that can be used to power the board while observing the data in and out of an Arduino board. The second option is to use an AC-to-DC adapter with 6-20 volts DC output. The board is compatible with a 2.1 mm center positive jack. Finally we can also power using the Vin and Gnd pins on the board if we have a battery. The maximum input voltage range is 6-20 volts but the recommended range is 7-12 volts [16].

3. **Memory**

   The Atmega168 has 16 KB of flash memory and Atmega328 has 32 KB of flash memory. In fact Atmel processors are named like AtmegaXXY where XX indicates the amount of flash memory they have and Y in some cases denotes how many bit architecture its processor is. Additionally, The ATmega168 has 1 KB of SRAM and 512 bytes of EEPROM; the ATmega328 has 2 KB of SRAM and 1 KB of EEPROM.

4. **Inputs and Outputs**

   There are 14 digital pins on the Arduino Duemilanove and six analog pins. Each of the digital pins has an internal pull-up resistor of 20-50 kOhms and can provide a maximum of 40 mA. The pins can be used as input/output by the use of a few functions like pinMode(), digitalWrite(), and digitalRead(). Moreover, Analog pins can also be used as digital pins by using numbers 14-19 for them instead of 0-5 in the above mentioned functions. Six of the digital pins also provide PWM output. However if needed PWM can also be produced by *bit banging* on other digital pins. Bit banging is basically generation of PWM manually by timing the switching of an output pin between 5 and 0 volts in a periodic fashion.

5. **Serial:**

   Arduino Duemilanove has one serial port, 0 (Rx) and 1 (Tx). Rx is used to receive and Tx is used to transmit TTL serial data. These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.

6. **External Interrupts:**

   Digital pins 2 and 3 of the board can be used as the external interrupts. The interrupt service routine (ISR) connected to these interrupts can be configured to be called on a rising edge (0-1 logic), falling edge (1-0 logic) or on change in value (0-1 or 1-0 logic). The attachInterrupt() function is used to attach a particular pin to an ISR.

7. **PWM:**

   Digital pins 3, 5, 6, 9, 10, and 11 can be used to provide PWM output by using analogWrite() function. This function takes a value from 0 to 255 (8 bit) that is translated to the duty cycle of the resulting PWM. 0 means 0% duty cycle and 255 means 100% duty cycle.

8. **SPI:**

   SPI stands for Serial Peripheral Interface. It is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. The 4 digital pins associated with it are 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library. These are all useful to burn the bootloader on a micro-controller.

9. **LED:**

   Pin 13 is connected to a built-in LED. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

   The Duemilanove has *6 analog inputs*, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the *AREF pin* and the analogReference() function. We put a voltage reference on to the AREF pin and change default reference to the new one in analogReference() function.

   Additionally, some pins have specialized functionality:

10. **I2C:**

    I2C stands for inter-IC communication. Pin 4 is Serial Data line (SDA) and pin 5 is serial clock line (SCL). Using these two bus lines data can be transferred between ICs at 100kbps in standard mode, 400 kbps in fast mode and 1 mbps in fast mode plus and up to 3.4 mbps in high speed mode.

    There are a couple of other pins on the board:

11. **Reset:**

    Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one already on the board.

12. **USB Overcurrent Protection**:

The Arduino Duemilanove has a resettable polyfuse that protects a computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed [16].

2.1.7.2 Arduino Mega

The Arduino Mega is a microcontroller board based on the ATmega1280 (shown in figure 2.18). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. The Mega is compatible with most shields designed for the Arduino Duemilanove or Arduino Diecimila.

Below is a summary of the important components of the Mega board:

Microcontroller: ATmega1280

Operating Voltage: 5 volts

Input Voltage: 7-12 volts

Input Voltage (limits): 6-20 volts

Digital I/O Pins: 54 (14 provide PWM)

Analog Input Pins: 16

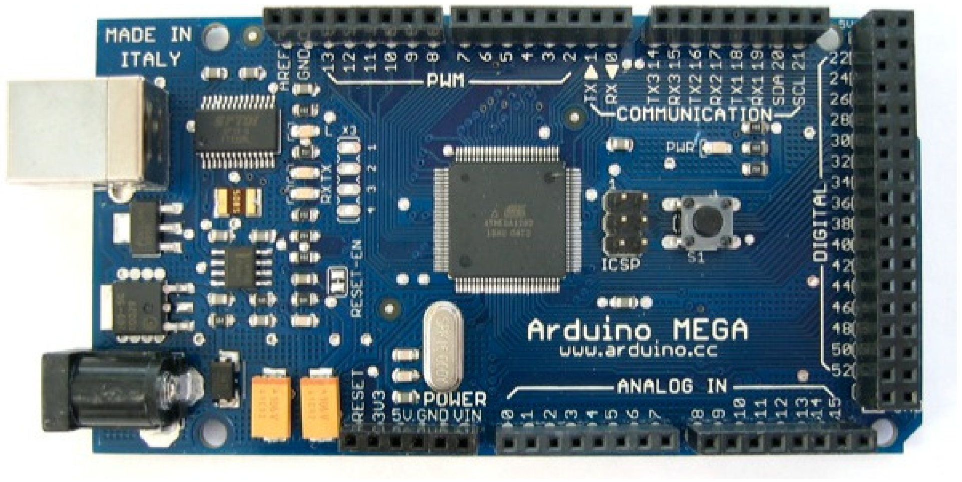


Figure 2.18: Arduino Mega (ATmega 1280)

DC Current per I/O Pin: 40 mA

DC Current for 3.3V Pin: 50 mA

Flash Memory: 128

SRAM: 8 KB

EEPROM: 4 KB

Clock Speed: 16 MHz

2.1.7.3 Arduino Environment

The microcontroller on the Arduino board is programmed using the Arduino programming language and the Arduino development environment. The Adruino language is based on Wiring language and the Arduino environment is based on

Processing environment. I will briefly explain below what these terms mean.

Wiring Language

Wiring is an open source programming environment that is used for exploring computer programming and prototyping electronics. It combines the two realms of programming and electronics by physically interacting with the intangible media and design. Wiring started at the Interaction Design Institute Ivrea in Italy and it is currently developed at the Universidad de Los Andes in Colombia. This project was initiated by Hernando Barragan at Universidad de Los Andes.

Major contribution of wiring is done by Hernando but it also involve contribution by some other researchers. Wiring builds on *Processing* so portions of the code are copyrighted by Ben Fry and Casey Reas. Who are these guys, I will explain in the next section. What is processing and how it was evolved?.

Processing Environment

Processing is an open project initiated by Ben Fry (Broad Institute) and Casey Reas (UCLA Design, Media Arts). It originated from experiments and concepts developed in the Aesthetics and Computation Group at the MIT Media Lab. Its an open source programming language and environment for people who want to create images, animations, and interactions. Motivated from teaching programming concepts visually to computer engineering students Processing consequently has established as a language that is capable of being used for developing complete products. Today, there are a lot of students, designers and hobbyists who use Processing for learning, prototyping, and production [6].

Ben Fry is founder of *Fathom*, a design and software consultancy located in

Boston. He received his doctoral degree from the *Aesthetics and Computation Group* at the MIT Media Laboratory, where his research focused on combining fields such as computer science, statistics, graphic design, and data visualization as a means for understanding information. Processing is the product that combines these fields beautifully.

With Casey Reas of UCLA, he currently develops Processing, an open source programming environment for teaching computational design and sketching interactive media software.

2.1.8 Inertial Measurement Unit (IMU)

An inertial measurement unit (IMU) is a sensor that is used to measure linear acceleration and orientation (roll, pitch and yaw angles and rates). Linear acceleration is measured with the use of a set of 3 accelerometers that are placed orthogonally to one another and the orientation is measured using a gyroscope [18].

Since accelerometers give us acceleration of the object they are mounted on. Linear velocity can be found by differentiating their readings. These are important to accurately determine the location and stability of robot.

The Razor IMU (shown in Figure 2.19) has four sensors a single axis gyro (LY530AL), a double axis gyro (LPR530AL), a triple-axis magnetometer (HMC5843) and a triple axis accelerometer (ADXL345). The outputs of all sensors are processed by an on-board Atmega328 and are sent out over a serial interface. With the work of Jordi Munoz and many others, the 9DOF Razor has become an Attitude and

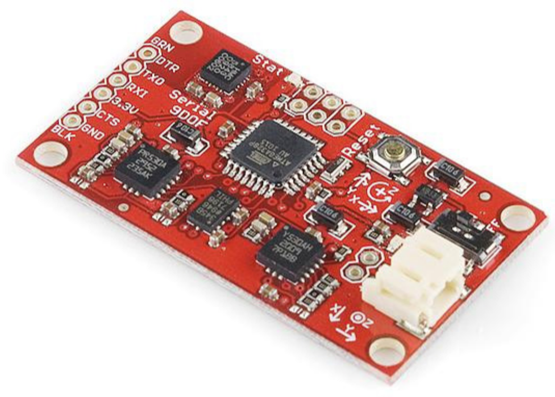


Figure 2.19: Sparkfun Razor IMU

Heading Reference System (AHRS). This enables the Razor IMU to become a very powerful control mechanism for UAVs, autonomous vehicles and image stabilization systems.

The board comes pre-programmed with the 8MHz Atmega328 and has a firmware that checks all of the on-board sensors and if those are in a working condition at that particular time to produce outputs, displays output serially. For observing raw sensor data we can simply connect to the serial Tx and Rx pins with a 3.3 volts FTDI Basic Breakout board. Using any terminal program we can interact with the firmware to see the data from any particular sensor or all of them at the same time. Moreover, this firmware can be changed in a similar way as one would program an Arduino. We just need to select the 'Arduino Pro or Pro Mini (3.3 volts, 8mhz) w/Atmega328' as the board.

The IMU operates at 3.3 volts DC; any power supplied to the white JST connector will be regulated down to this operating voltage - LiPo batteries are an excellent power supply choice.

2.1.8.1 Features

The IMU on a single flat board has following important components:

LY530ALH - 300^o/s single-axis gyro

LPR530ALH - 300^o/s dual-axis gyro

ADXL345 - 13-bit resolution, 16g, triple-axis accelerometer

HMC5843 - triple-axis, digital magnetometer

Input - 3 volts DC

2.1.8.2 Working and Terminology

Let me first briefly explain the basic components of an IMU and their functionality and then I will elaborate on its operation.

A *Gyroscope* or simply a gyro is a device for measuring or maintaining orientation, based on the principles of conservation of angular momentum. Gyroscopes can be mechanical or electrical. The mechanical gyroscope is essentially a spinning wheel whose axis is free to take any orientation. Gyroscopes based on electrical principles include MEMS based gyros, solid state optic laser gyros, fibre optic gyros and extremely sensitive quantum gyros.

An *Accelerometer* measures the change in linear velocity. It can be single or multi-axis. It can be used to sense acceleration or vibration. Instead of the standard m/sec^2 units here the accelerations are measured in terms of g-force [25]. An accelerometer at rest on the surface of the earth indicates a 1g upwards acceleration. In order to get the acceleration in a particular direction this gravity offset must be subtracted.

Now we need IMU for sensor fusion to get the advantage of each sensor. Single accelerometer measures the acceleration of an object along one axis while a 3-axis accelerometer can do so for all three axis and hence can be used to tell the orientation of the object relative to earth surface. If the object is moving then measurement becomes difficult. For example for a free falling body accelerating towards earth, accelerometer will show zero reading. Similarly during a coordinated banking it can show up to 2g vertical acceleration although aircraft is maintaining a particular altitude.

As discussed above, a gyro measures rate of rotation about an axis. Gyros will read 0 readings while plane is not tilting in any direction but if the plane tilts and stop at a tilted position, gyros will again give us zero readings which means that gyros alone can't be used to measure orientation. One can approximate the roll angle by integrating the roll rate over time but it is erroneous because gyros drift over time. This error will keep accumulating and produce more and more incorrect angle estimates over time. So gyros alone can't be used for orientation of aircraft.

So, in a nutshell: Accelerometers are right in the long term but noisy in the short term. Gyros are right in the short term but drift in the long term. One needs both, each to calibrate the other, to be right all the time.

But even that only works for pitch and roll only. For yaw, which is orthogonal to gravity, the accelerometers can't help, so one needs something else to correct the drifting yaw gyro. That's either magnetometers or GPS.

GPS has a relatively slow update rate (up to 10 Hz) and is subject to short term errors. It is possible to use GPS alone to keep a very stable and slow flying airframe on a particular track on a calm day.

Therefore to fuse all of this information into a more useful data we need an IMU which combines information from two or more sensors, such as gyros, accelerometers, magnetometer, and/or GPS, to determine orientation and velocity vector relative to the earth. The computations are fairly complex, and special filtering is often required to eliminate the measurement noise these silicon devices are subject to, so a "low cost" IMU with decent specs can easily cost \$1000 to \$5000.

An important terminology in this regard is AHRS. An *Attitude Heading Reference System* consists of sensors on three axes that provide heading, attitude and yaw information for aircraft. AHRS consist of either solid-state or MEMS gyroscopes, accelerometers and magnetometers on all three axes. The key difference between an IMU and an AHRS is the addition of an on-board processing system in an AHRS which provides solved attitude and heading solutions versus and IMU which just delivers sensor data to an additional device that solves the attitude solution. A form of non-linear estimation such as a Kalman filter is typically used to compute the solution from these multiple sources. AHRS differ from traditional *Inertial Navigation Systems (INS)* by attempting to estimate only attitude (i.e. roll, pitch, yaw) states, rather than attitude, position and velocity as is the case with an INS [25].

The Sparkfun Razor IMU that I am using uses Direction Cosine Matrix (DCM)

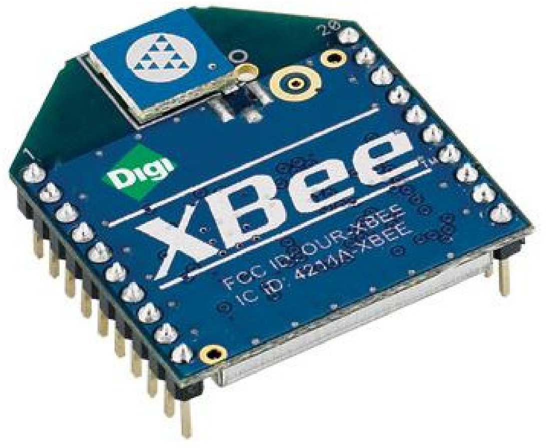


Figure 2.20: Digi International Xbee PRO RF Module [40]

approach for sensor fusion which I will explain in the next chapter. It is developed by Doug Weibel and Jose Julio who also contributed to arduPilot, a project of DIY drones, the first company to successfully launch its iPhone controlled quadcopter recently with a lot of exciting features like live video streaming etc.

2.1.9 Xbee/Xbee Shield

The XBee-PRO ZNet 2.5 OEMs (Original Equipment Manufacturers) RF module, shown in the Figure 2.20, satisfies the unique needs of low-cost, low-power wireless mesh sensor networks. They operate on IEEE standard 802.15.4. Let me explain this standard first:

2.1.9.1 IEEE standard 802.15.4

IEEE (Institute for Electrical and Electronics Engineers) 802.15.4 is a standard for wireless communication. The IEEE is a technical professional association that has written numerous standards to promote growth and inter-operability of existing and emerging technologies. As a few examples, the IEEE 802.11 standard defines communication for wireless LAN and 802.16 defines communication for broadband wireless Metropolitan Area Networks (MAN).

Each of these standards has different considerations in its design. For example, 802.11 and 802.16 were developed to meet the needs of having a higher bandwidth internet access. But 802.15.4 was developed for low data rate applications with limited battery power in mind. It operates in the 868-868.8 MHz, the 902-928 MHz or the 2.400-2.4835 GHz range. Usually 2.4 GHz band is used because it is available in all parts of the world [40].

The 802.15.4 standard specifies that communication should occur in 5 MHz channels ranging from 2.405 to 2.480 GHz. In the 2.4 GHz band, a maximum over-the-air data rate of 250 kbps is specified, but due to the overhead of the protocol the actual theoretical maximum data rate is approximately half of that. While the standard specifies 5 MHz channels, only approximately 2 MHz of the channel is consumed with the occupied bandwidth. At 2.4 GHz, 802.15.4 specifies the use of Direct Sequence Spread Spectrum and uses an Offset Quadrature Phase Shift Keying (O-QPSK) with half-sine pulse shaping to modulate the RF carrier [40].

Digi's XBee 802.15.4 OEM RF modules can be set up to operate in a point-to-point, point-to-multipoint or a peer-to-peer configuration. While standard 802.15.4 always requires a coordinator, the Digi radios are set up so that a coordinator is not required.

2.1.9.2 Zigbee Protocol

Zigbee is a protocol that is based on the specifications of 802.15.4 standard and adds some additional functionalities that are particularly suited for low power applications like mesh networking. This protocol is developed by *Zigbee Alliance* which is a group of companies who cooperate to produce network protocols for different applications.

Since low power devices don't have enough power range. *Mesh Networking* is especially useful as it allows an intermediate device to route data between source and destination while not using enough power. The ZigBee protocol within the radios will take care of retries, acknowledgements and data message routing. ZigBee also has the ability to self-heal the network. If the radio at any point in the network was removed for some reason, a new path would be used to route messages among other nodes [8].

There are three configuration in which a Zigbee module can be used i.e. a Coordinator, a Router, or an End Device. Each of these have their own specifications and unique function in the network. A *Coordinator*, for example, is responsible for starting the network, connecting different networks and allowing devices to connect to it and form a network. It has the most memory of all other network components, selects the frequency channel for the network to operate on and can bridge among different Zigbee networks. There is only one coordinator in any network. The *Router*, is the second most powerful device in Zigbee networks. It is mainly used for routing messages between coordinator and end devices but it can also run applications. Finally an *End Device* is mainly used for running a particular application. It usually forms the ends of an network and hence got its name [8].

2.1.9.3 Key Features

Indoor/Urban: up to 300 feet (100 m).

Outdoor line-of-sight: up to 1 mile (1.6 km).

Transmit Power Output: 100 mW (20 dBm).

Receiver Sensitivity: -102 dBm

RF Data Rate: 250 kbps

Tx Current: 295 mA (@3.3 volts)

Rx Current: 45 mA (@3.3 volts)

Power-down Current: $1\mu A$@25^oC

2.1.9.4 Point-to-Point Network Setup

Point-to-point communications or in short P2P generally refers to a connection restricted to two endpoints. Though Zigbee is capable of P2P and P2MP (Point-to-multipoint) communications but in this thesis I will focus only on P2P network as that is the only case needed for the helicopter and the base station communication. I will briefly list the procedure I used for creating the network.

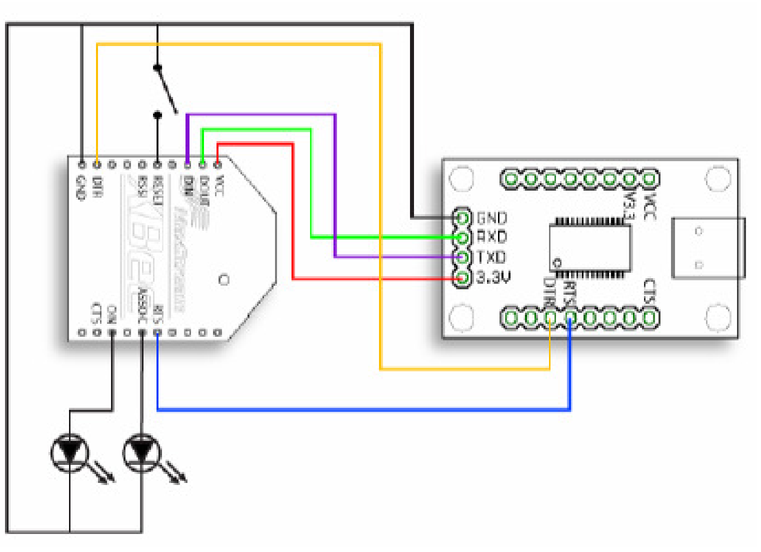


Figure 2.21: FT232 breakout board with Xbee ZNet Module

1. **Circuit for Xbee Configuration:**

   In order to configure an Xbee module to a particular configuration one needs to build a circuit. For this we need an XBee Breakout Board and an Xbee module. Circuit should be set up as shown in Figure 2.21.

   Since an Xbee is a 3.3 volt device and it pulls 40 mA so we can power it directly by the FT232RL output which can provide up to 50 mA at this output voltage. Another way is to power it using the 3.3 volts output pin of an Arduino.

2. **Setting Up X-CTU:**

   Next we need to plug the FT232 Breakout into computer using a USB-mini Type B cable. Both LEDs on the Xbee will turn on. Now using the Xbee configuration software provided by Digi International i.e. X-CTU, we can select the port to which Xbee is connected. If initially its difficult to figure out which port is Xbee connected to we can select a port and click test/query. Once we found the port we can switch to the ‘Modem Configuration’ tab to see what configuration the Xbee is already configured to. Finally we can change certain fields to configure it to perform a particular function.

3. **Router Setup:**

   Among many options available, we need to configure a few in the following way:

   Function Set: ZNet 2.5 Router/End Device AT.

   Node Identifier: RouterX.

   PAN Id: 3337 or whatever 4 digit number you like. This is used to detect the Private Area Network (PAN) when other networks are operating nearby.

   After this we can test new Xbee configuration using following AT commands:

   +++: Xbee ready for command mode, usually Xbee replies with OK to confirm.

   ATVR: This AT command asks for the new version of Xbee software uploaded to Xbee.

   ATID: The Xbee will return the PAN Id that we set in the previous step.

   ATNI: Xbee will give Network Identifier back.

   ATCN: To exit command mode.

4. **Coordinator Setup:**

Here we have to repeat step 3 for another Xbee and configure it as the coordinator. But in this case the configuration needs to be different:

Function Set: ZNet 2.5 Coordiator AT.

Node Identifier: CoordinatorX.

PAN Id: 3337 or whatever 4 digit number you like. This is used to detect your Xbee network when other networks are operating nearby.

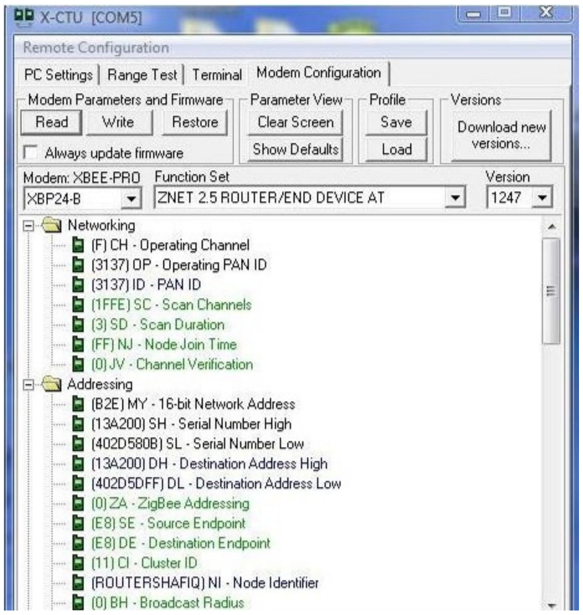
DH: Destination High (DH) needs to be set same as the Source High (SH) i.e. the high address of RouterX

DL: Destination Low (DL) needs to be set same as the Source Low (SL) i.e. the low address of RouterX

As an example I have shown Router setup in figure 2.22 pictorially. After making these changes we are done with the setup of our PAN. We can now use X-CTU to communicate between two Xbees.

2.1.10 Power Board

In addition to the company manufactured components, I had to design a power board that is capable of supplying 14.8 volts to the arduino Mega, 5 volts regulated to the rest of the circuit like servos etc and 3.3 volts regulated to the IMU. It also has some indicator LEDs that signal to the pilot critical steps taken by the on-board processor.



Modem: XBEE-PRO XBP24-B
Function Set: ZNET 2.5 ROUTER/END DEVICE AT
Version: 1247

Networking
- (F) CH - Operating Channel
- (3137) OP - Operating PAN ID
- (3137) ID - PAN ID
- (1FFE) SC - Scan Channels
- (3) SD - Scan Duration
- (FF) NJ - Node Join Time
- (0) JV - Channel Verification

Addressing
- (B2E) MY - 16-bit Network Address
- (13A200) SH - Serial Number High
- (402D580B) SL - Serial Number Low
- (13A200) DH - Destination Address High
- (402D5DFF) DL - Destination Address Low
- (0) ZA - ZigBee Addressing
- (E8) SE - Source Endpoint
- (E8) DE - Destination Endpoint
- (11) CI - Cluster ID
- (ROUTERSHAFIQ) NI - Node Identifier
- (0) BH - Broadcast Radius

Figure 2.22: Configuration of Router Xbee

2.1.11 eLogger

I used eLogger V3 from Eagle Tree Systems, shown in Figure 2.23, which is an in-flight data logger that continuously records volts, amps, watts, and milli-amp-hours to enable a pilot to analyze the performance of his/her electric power systems. In addition, it supports a wide variety of optional sensors to record altitude, airspeed, component temperatures, RPM, and more.

An eLogger is a small device that is interfaced with many different sensors whose values are continuously digitized and stored in the internal memory of the device. A software comes with the device that can be used to download all the data from the eLogger and plot it on a computer. There are a number of sensors that can be used with the eLogger like air speed sensor, servo current sensor, GPS module, altitude meter, temperature sensor, RPM sensor and Power Panel display. I used

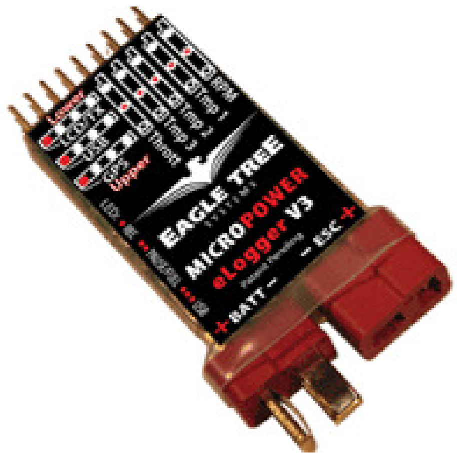


Figure 2.23: eLogger V3 (Eagle Tree Systems)

only the last three. The variety of sensors and the ability to use so many of them at the same time is one of the major strengths of this product.

2.1.11.1 Key Features:

- Logs peak currents up to +/- 100 amps and voltage from about 5 volts to 70 volts.
- Great for bench monitoring of battery charging.
- Accepts optional inexpensive sensors for three Temperatures, RPM (brushless, magnetic or optical), Airspeed, Altitude, GPS, Servo Current, Throttle Movement, and more
- Adjustable logging rate (1-10 samples/second) and lossless data compression for long log times
- Integrated USB with included cable

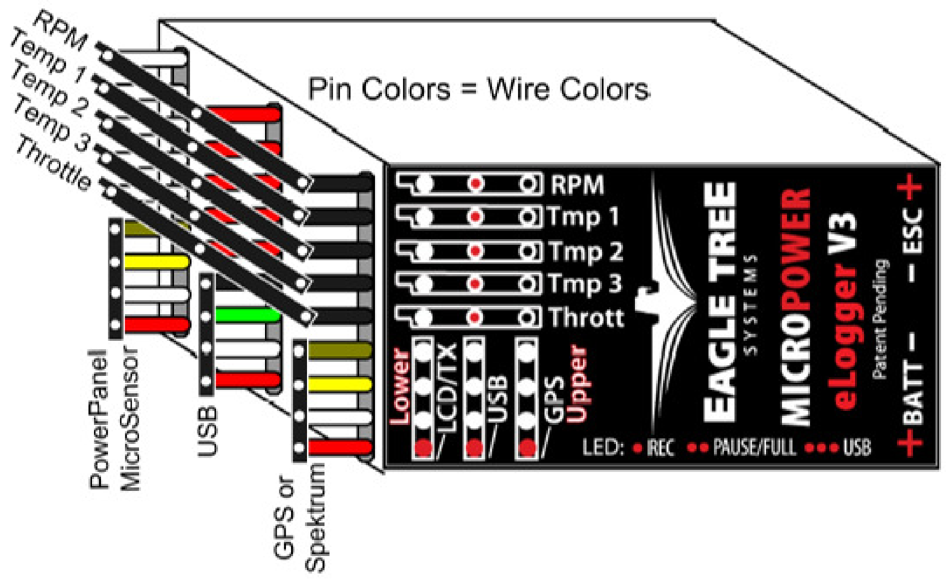


Figure 2.24: eLogger V3 Pinout

- Weighs about 0.7 oz (20 grams)
- Powerful Graphing software has advanced charting features

2.1.11.2 Installation of Elogger and related Sensors

The eLogger is normally connected such that the connectors or leads marked “Batt” are connected to the battery pack, and the connectors or leads marked “ESC” are connected to the ESC’s power input. The red wires of the “wire lead” version of the eLogger should be connected to the positive side of the load and source. A pin out of the eLogger is shown in Figure 2.24.

1. **Temperature Sensor:**

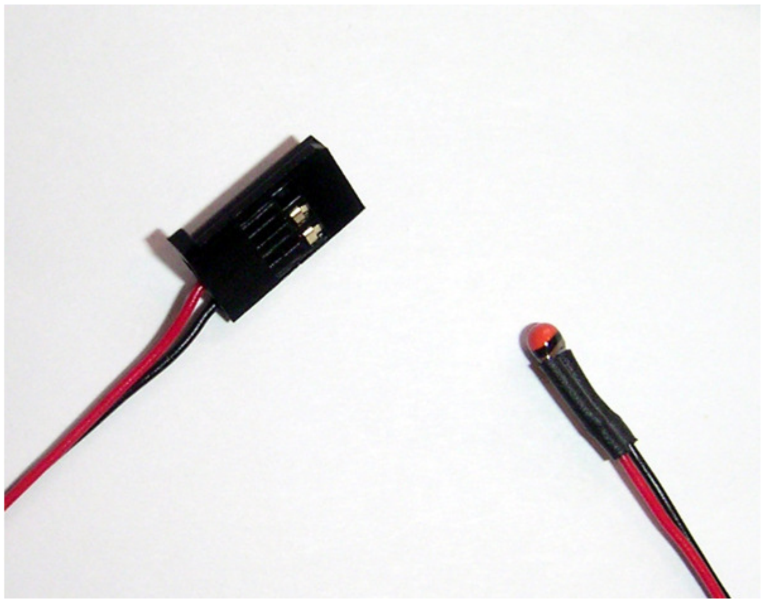


Figure 2.25: micro temperature sensor

The micro-temperature sensor, shown in Figure 2.25, can be installed by placing its tip at the surface whose temperature is to be measured. It can be tapped on, glued or fixed to the surface. eLogger V3 can use up to three temperature sensors simultaneously. A micro-temperature sensor can actually be used for measuring temperatures up to 250^{o}F.

2. **RPM Sensor:**

   The Brushless Motor RPM Sensor, shown in Figure 2.26, works with eLogger to measure RPM via pulses from any two of the wires leading from ESC to the motor. The sensor works with all known brushless motors. If we have a brushed motor, the sensor will not work. Brushed motor RPM can be measured with magnetic or optical RPM sensors.

3. **Power Panel LCD Display:**

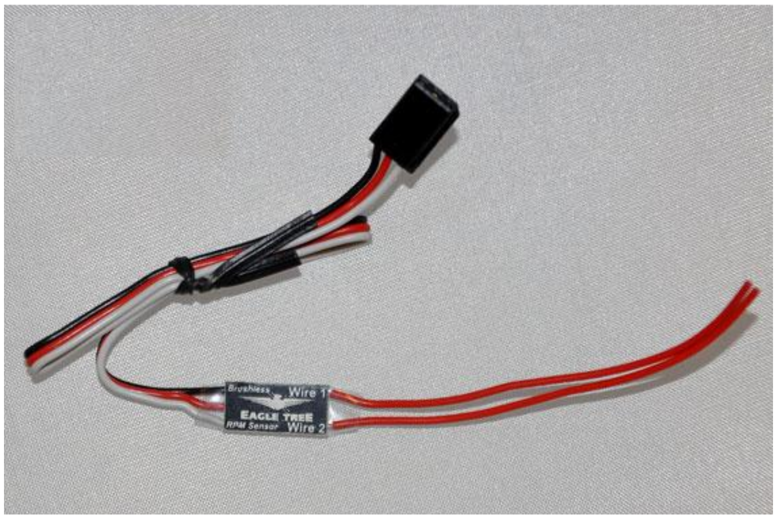


Figure 2.26: Brushless Motor RPM sensor

The ultra thin Liquid Crystal Display (LCD), shown in Figure 2.27, connects to the eLogger with a data cable, and displays maximum and live data without a PC, just like the heavier, non-logging Watt-meters on the market. The display is fully programmable to show live and max voltage, amperage, wattage, temperatures, RPM, mAh used etc. Since one can know mAh of system after a session, one knows approximately how much time the battery would survive.

Since the Power Panel is light, it can easily be attached to one side of the model for live/max display all the time.

2.1.12 Battery

I used the *Thunder Power* (TP) 2250 mAh Lithium Polymer (LiPo) battery. It is 14.8 volts and is of type 4s1p. It has a maximum discharge rate of 30C (68

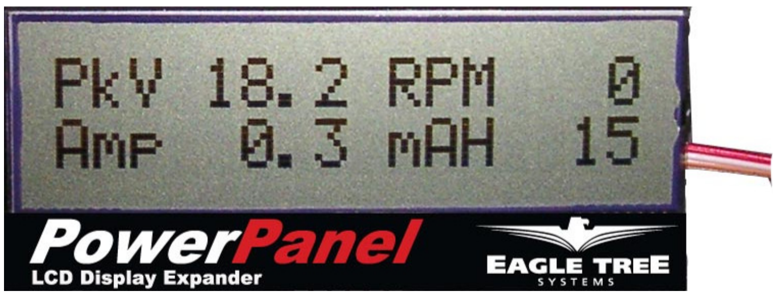


Figure 2.27: LCD Power Panel Display

amp) with a 60C (135 amp) burst rating. The dimensions are 32 mm x 106 mm x 32 mm. It weighs about 235 grams and has a built in balancing connector. Let me explain a few terms used above.

2.1.12.1 milli-Amp-hour (mAh)

It is a unit for measuring electric power over time. mAh is commonly used to describe the total amount of energy a battery can store at one time. A higher mAh rating means that a charged battery can power a device that consumes more power and/or for a longer amount of time before becoming depleted and needing to be recharged. For example, a battery rated at 2250 mAh can power a device drawing 2.25 amps for 1 hour, or a device drawing 13.5 amps for 10 minutes which is what we need for the helicopter used.

2.1.12.2 LiPo configurations

LiPo batteries usually come in packs that means different 3.7 volts small batteries are hooked up in different configurations to make up high voltage or high

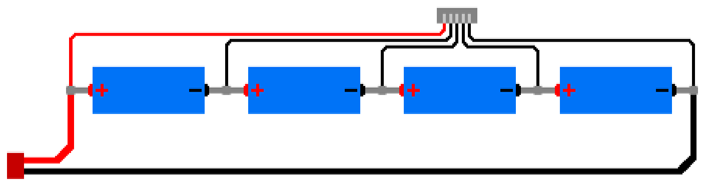


Figure 2.28: 4s1p LiPo battery configuration

current rated batteries. These two objectives are achieved by simple electrical principles like two batteries connected in parallel will give high current but same voltage and two batteries connected in series will give same current and high voltage. It depends on one's application to choose the one that is more suitable like in driving micro-controllers low current batteries are enough but for electric motors high current batteries are used.

Usually those come in XsYp configuration where X and Y can be positive integers. 's' stands for serial and 'p' stands for parallel. So a 4s1p LiPo means that the battery contains 4 cells connected in series and therefore they make up one parallel branch only. It is pictorially shown in Figure 2.28.

2.1.12.3 'C' rating

'C' stands for the capacity of the battery and works with mAh rating explained above. It tells how much current is safe to draw from the battery. The charge and discharge current of a battery is measured in C-rate. Most portable batteries are rated at 1C. This means that a 1000mAh battery would provide 1000mA for one hour if discharged at 1C rate. The same battery discharged at 0.5C would provide 500mA for two hours. At 2C, the 1000mAh battery would deliver 2000mA for 30 minutes. 1C is often referred to as a one-hour discharge; a 0.5C would be a two-hour, and a 0.1C a 10-hour discharge. Lithium-ion/polymer batteries are electronically protected against high load currents. Depending on battery type, the discharge is limited to between 1C and 2C.

In my case, the battery is 2250mAh and 30C that means that maximum continuous discharge current can be up to 30 x 2.25 amps, which is 67.5 amps. while burst current can go up to 135 amps. These specifications are pretty safe as the maximum continuous current the BLDC motor takes is 15-16 amps for hovering and less than 20 amps for maneuvering.

2.2 Overall Circuit Diagram

To get an overall idea of how each component is linked with the other, Figure 2.29 shows the final circuit diagram of the all the components that I used in the helicopter and which are discussed in this chapter.

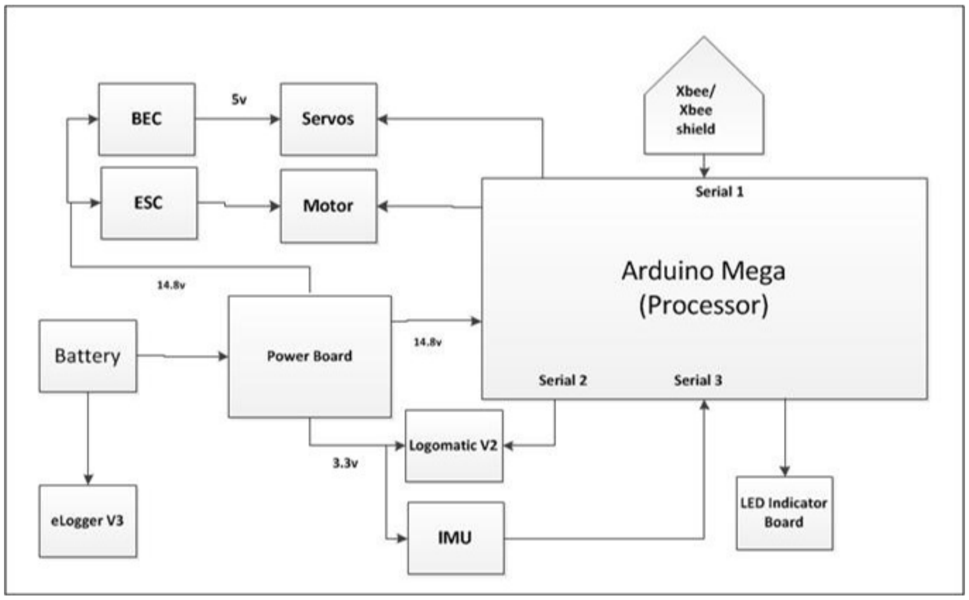


Figure 2.29: The Overall Circuit Diagram of the Helicopter

Chapter 3

Development of System Model and Compensators for Hover

The techniques and algorithms used to model the system, which in this case is a small scale twin rotor helicopter, are explained in detail in this chapter. It is important to note here that the purpose of this thesis is to document my experience with this experiment and to facilitate its use in future projects. My primary objective here is not to get a completely autonomous aerial vehicle, but rather to study the control issues involved in this endeavor. In last chapter, I described the way I developed the hardware required for a twin rotor helicopter. In this chapter the software, system identification procedure and compensator design will be explained. It also involves description of some time and frequency domain system approaches, each of which has its own advantages and disadvantages. What different inputs were chosen and why some of those were used while others were discarded?. I will also state the results obtained and will explain those with graphical reasoning. Finally I will explain how I developed the unit feedback controllers for the system and what were the performance criteria.

3.1 Basic Steps in Identification Process

Each identification process consists of a series of basic steps. Some of them may be hidden or selected without the user being aware of his/her choice. Clearly, this can result in poor or suboptimal results. In order to avoid that the following actions should be taken in each session:

- Gather information about the system.
- Select one of model structures to represent the system.
- Choose the model parameters to fit the model
- Verify the selected model.

Each of these points is discussed in more detail below:

3.1.1 Gather Information about the System

In order to build a model for a system one needs to collect as much information about the system as possible. This actually helps to understand the system and its response better. While setting experiments one has to be very careful regarding a number of issues like giving system as many degrees of freedom as possible and selecting a collection of inputs that better represent the real inputs the system may face once in operation. This also affects the quality of the final result. I, during system identification process I used, also chose a number of inputs but then selected one for the reasons I will explain in later sections.

3.1.2 Select a Model Structure to Represent the System

Another important thing is to select a model structure for the system which one thinks would represent one's system best under some assumptions. Again a number of possibilities exist, such as

3.1.2.1 Parametric versus Nonparametric Models

The difference between a parametric and a nonparametric model is as follows: In a parametric model the modeler tries to estimate directly some of the model parameters based on which he/she can characterize system response to inputs. Whereas, in nonparametric models one can either measure the system response at all possible frequencies or at a large number of points (in the time domain) of interest. A filter for example described by its poles and zeros is an example of a parametric model but the same filter when described by its impulse response at a large number of points is a representation of a nonparametric model [7].

For modeling systems like an RC helicopter nonparametric models are easier to estimate than the parametric ones because nonparametric models' estimation needs little information about the system. Moreover, they mainly depend on the choice of inputs and a careful observation of resulting outputs. Parametric models, on the other hand, need an insight into the working knowledge of subsystems of a system and observation of some behavior-defining characteristics like resonant frequency etc. This thesis focuses on development of a nonparametric model for the system in particular by observing the response of the helicopter to a lot of different frequency inputs.

3.1.2.2 White Box Models versus Black Box Models

White box modeling means the modeling that is based on the physical laws that govern the system and the specialized knowledge of different fields involved in the functioning of the overall system. Moreover the experimenter has a good grasp of specific areas of these fields and he is able to formulate the provided information. As an example we can take modeling of a loud speaker. It requires extensive understanding of mechanical, electrical and acoustic phenomena.

Another approach is to extract a *black box model* from the data. This is more practical and physically realizable technique because this allows one to draw a mathematical model of the system based only on the input and output responses of the system. Unlike the white box approach, it doesn't require any physical insight into the system.

There is a trade off for choosing any particular strategy. A modeler needs to prioritize constraints that suit him best depending on his/her working knowledge of the system, precision needed for use, time available and the information he/she has for the project.

Although as a rule of thumb, it is advisable to include as much prior knowledge about the system as possible during the modeling process, it is not always easy to do. If we know, for example, that a system is stable, it is not simple to express this information if the polynomial coefficients are used as parameters [7].

3.1.2.3 Linear Models versus Nonlinear Models

Only a few systems in real life are linear, all others are nonlinear. But fortunately most of the nonlinear systems operate near an equilibrium point, where they can be approximated by a linear system. This conversion is usually made because the theory of nonlinear systems is complicated and is generally avoided. Even if the nonlinear model is simple it would take a lot more time to model it than for the corresponding linearized one. This linearized model has same features as the original nonlinear one within some thresholds of operation defined while linearizing.

For example, a nonlinear model is needed to describe the distortion of an amplifier, but a linear model will be sufficient to represent its transfer characteristics if the linear behavior is dominant and is of main interest [7].

3.1.2.4 Linear/Nonlinear-in-the-parameters

A model is called linear-in-the-parameters if the error is linearly related to the parameters of the output to input relationship. The system in this case may/may not be linear itself. For example $\epsilon = y - (a_1 u + a_2 u^2)$ is linear in parameters a_1 and a_2 where ϵ is the error between observed output y and its true value but describes a nonlinear system. On the other hand,

$$\epsilon(j\omega) = Y(j\omega) - \frac{a_o + a_1(j\omega)}{b_o + b_1(j\omega)} U(j\omega) \tag{3.1}$$

describes a linear system but the model is nonlinear in b_1 and b_2 parameters. Linearity in the parameters is a very important aspect of models because it has a strong impact on the complexity of the estimators if a weighted least squares cost function is used. In that case, the problem can be solved analytically for models that are linear in the parameters so that an iterative optimization is avoided [7].

3.1.3 Relate the Selected Model Structure to the Measurements

After choosing a system model the next thing to do is to match it to the available information. The best thing to do in this regard is to minimize some form of error between the measured value and the true model value. Usually, mean squared error is used as a parameter of goodness. The choice of this criterion is very important as it determines the major properties of the resulting model.

3.1.4 Verify the Selected Model

Finally one should verify that the model selected explains the system response well. If not then we need to modify the model to meet the missing information. It is interesting to note here that the model with minimum errors is usually adopted but is not always preferred. The choice of this cost function depends on the application.

The model verification test should consider providing the same conditions to the system as are expected when in operation. Extrapolation of the model characteristics should be avoided as much as possible.

This brief overview of the identification process indicates that it is a time consuming and sensitive task. In order for the system to behave in the way expected, it is necessary that care should be taken in completing all these steps and any others if needed.

3.2 IMU Firmware

In chapter 2 we discussed the specifications of the Razor IMU and its general theory of operation which involves the necessity of incorporation of different sensors. Now is the time to discuss how are these readings fused together to get a useful outcome which can subsequently be used in the control algorithms for stabilizing the helicopter.

3.2.1 Original Firmware

Initially the Sparkfun's Razor IMU that I am using has firmware that was used just to read the raw data from gyros, accelerometers and magnetometers. The board comes programmed with the 8MHz Arduino Atmega168 bootloader and an example firmware that tests the outputs of all the sensors. It tests the outputs of all the components and if working outputs the data serially. The data sent is raw data which means it directly comes from the sensor and not being modified at all. Since the data sent is raw this means two things. Firstly the gyro outputs will naturally have a drift in their values, which needs to be corrected. Secondly for future control algorithms we might also needs Euler angles so the firmware needs to be changed. The Razor IMU has a micro-controller on-board and algorithms already exist that can convert the raw sensor information to Euler angles, I relied on the on-board controller. This helps in reducing the space of an extra controller and therefore its weight. The algorithm that I used for this purpose is the DCM algorithm.

Let me first explain what DCM means?

3.2.2 DCM Based Firmware

The Direction Cosine Matrix (or DCM) can be used to track the motion of an object in space. The first and most significant attempt in this regard is of William Premerlani [11]. Below is the summary of the algorithm by which gyro drift is compensated:

1. The gyros are used as primary source of orientation information. The non-linear kinematics equation that relates the time rate of change of orientation of aircraft to its rotation rate and its present orientation is integrated. This is done at 50 Hz which is enough to give servos updated position information within their 20 msec inter-command interval.

2. Recognizing that numerical integration errors will gradually violate the orthogonality condition of the rotation matrix we need to make continuous adjustments.

3. A negative feedback PI controller is used to correct for numerical errors, gyro drifts and gyro offsets. Magnetometers are used for yaw rate and accelerometers are used for pitch and roll rate corrections.

3.3 Choice of Model and System Identification Approach

After considering the basic steps in the identification process discussed in detail in Section 3.1 and the information available about the system to be modeled, listed below are some of the reasons for choosing different characteristics of the model:

3.3.0.1 Nonparametric:

In case of small-scale helicopters, we know a little about the dynamics of the helicopter. Probably the only good assumption one can make is that there is no significant interaction between the rotor wake and the fuselage of the helicopter which will have the effect of reducing the complexity in modeling [37]. So no useful insight into the working dynamics of the system is present therefore it is better to evaluate the system at as many different points of interest as possible. This pertains to nonparametric modeling.

3.3.0.2 Linear:

In order to develop a nonlinear model for the helicopter, a complete experimental study of the forces/torques acting on the helicopter should be performed and mathematical relationships for these should be developed [38]. Since this is only a masters thesis and I had spent a lot of time working on the avionics of the system, it was not possible to develop a complete mathematical description of the helicopter dynamics due to the limited time available. So a reasonable approach was to find a linear model that would be sufficient enough to explain the response of the helicopter at hover.

3.3.0.3 Black Box:

Since a black box approach allows one to draw a mathematical model of the system based on the inputs and outputs of the system. A black box approach is more reasonable to follow due to less required knowledge about the system and the time required for using this approach.

3.3.0.4 Linear in parameters:

Finally, as we are developing a linear nonparametric model of the system it comes out that it would be linear in parameters as well. This in turn reduces the complexity of the transfer function that may have been incorporated due to nonlinear parameters. Nonlinearity in parameters actually increases the complexity of the cost function if a weighted least square function is used. So for simplicity we will choose linear parameters and linear models.

3.3.1 Why Second Order?

According to [41], the response of a mini-helicopter can be represented by a fourth order transfer function which has two poles farther from the imaginary axis of the s-plane called *Short Period Poles* and two poles (complex conjugate) that are located close to the imaginary axis called *Phugoid Poles.* These poles are responsible for two primary forms of oscillations. The first form is thePhugoid mode of oscillation, which is a long-period, slow oscillation of the aircraft. This oscillation can generally be controlled by pilot. The second oscillation is a short-period variation of the angle of attack. Usually, this oscillation decreases very quickly with no pilot effort. A study of the root locus of these transfer function shows Phugoid poles try to move towards the right half plane while the short period poles move further into the left half plane. Thus from a compensator design point of view Phugoid poles are important because they can make the system unstable. This motivates us to further simplify the transfer function consisting of the Phugoid poles only which means modeling the helicopter as a second order system can account for a reasonable representation of system dynamics.

3.3.2 System Identification Approach and Limiting Factors

Though there are a lot of system identification approaches out there in theory [32] for LTI systems but I will employ some simple techniques mainly because my system is quite complex and its better to start up with simple techniques so that setting up the experiment for it should be inexpensive. The techniques that I used are explained below:

3.3.2.1 Frequency Response Analysis

In frequency response analysis, we input a sine wave to the system and see the output response of the resulting sine wave. It pertains to fundamental physical interpretation of a transfer function $\mathbf{G}(\mathbf{s})$ that the complex number $\mathbf{G}(\mathbf{j}\omega)$ bears the information of what happens to an input sine wave of a particular frequency 'ω' when passed through an LTI system. Mathematically, we know that if we give following input to an LTI system:

$$\mathbf{u(t) = cos(\omega t)} \tag{3.2}$$

or

$$\mathbf{u(t) = Re(e^{j\omega t})} \tag{3.3}$$

The corresponding output will be

$$\mathbf{y(t)} = \sum_{\mathbf{k=1}}^{\infty} \mathbf{g(k)Re(e^{j\omega(t-k)})} = \mathbf{Re}\sum_{\mathbf{k=1}}^{\infty} \mathbf{g(k)e^{j\omega(t-k)}} = \mathbf{Re}\left(\mathbf{e^{\omega t}}\sum_{\mathbf{k=1}}^{\infty} \mathbf{g(k)e^{-j\omega k}}\right) \tag{3.4}$$

$$=\mathbf{Re}\left(\mathbf{e^{j\omega t}.G(e^{j\omega})}\right) = \left|\mathbf{G(e^{j\omega})}\right| \mathbf{cos(\omega t} + \phi) \tag{3.5}$$

where

$$\phi = \arg G(e^{j\omega}) \tag{3.6}$$

These equations tell us that the output to an LTI system will also be a cosine of the same frequency as input, but with an amplitude amplified by $|G(e^{j\omega})|$ and a phase shift of $\arg G(e^{j\omega})$ radians.

The complex number $G(e^{j\omega})$ indicates a transfer function evaluated at the point $z = e^{j\omega}$ and therefore gives full information about the behavior of the system to a sinusoid input of frequency ω.

This function is usually then plotted as $\log |G(e^{j\omega})|$ and $\arg G(e^{j\omega})$ against $\log \omega$ which is called a *bode plot*.

Based on the bode plots obtained we can find parameters of a standard second order system [42] given below:

$$\mathbf{G(s)} = \frac{\mathbf{a}}{\mathbf{s^2 + bs + c}} = \frac{\mathbf{K\omega_n{}^2}}{\mathbf{s^2 + 2\zeta\omega_n s + \omega_n{}^2}} \tag{3.7}$$

1. **Estimating Order**

   The order of the system is inferred from the magnitude of dips in the phase plot of the Bode plot. A dip of 90^o means a real pole and a dip of 180^o means a complex conjugate pair.

2. **DC Gain**

   We can calculate the DC gain of the system from the magnitude plot of bode plot by noting the magnitude of the output at $s = 0$ i.e. M_o and is given by

   $$K = 10^{M_0/20} \tag{3.8}$$

3. **Natural Frequency**

   The natural frequency of a second order system is the value of frequency when the phase of the output is -90^o.

4. **Damping Ratio**

   The damping ratio of the system determines how fast the oscillations will decay after a disturbance to the system is made. The damping ratio can be calculated from the bode plot if we know the dc gain of the system and the gain when the phase of the system is -90^o i.e. M_{-90^o} and can be calculated by

   $$\zeta = \frac{K}{2 \cdot 10^{M_{-90^o}/20}} \tag{3.9}$$

3.3.2.2 Transient Response Analysis

Another way to find the model of a second order system is by analyzing the step response of a system. The following equations determine the important parameters i.e. damping ratio, DC gain and natural frequency etc that are defined in the previous subsection.

$$\zeta = \frac{-\ln(\%\mathbf{OS}/\mathbf{100})}{\sqrt{\pi^{\mathbf{2}} + \ln^{\mathbf{2}}(\%\mathbf{OS}/\mathbf{100})}} \tag{3.10}$$

where %OS indicates the percent overshoot (will be defined later in the chapter).

$$\mathbf{K} = \frac{\mathbf{SS_o}}{\mathbf{SS_i}} \tag{3.11}$$

where SS_o is the steady state output value and the SS_i is the steady state input value which is simply the magnitude of step.

$$\omega_{\mathbf{n}} = \frac{\omega_{\mathbf{d}}}{\sqrt{\mathbf{1} - \zeta^{\mathbf{2}}}} \tag{3.12}$$

where ω_d is the damped frequency which is $2\pi/\Delta t$ where Δt is the time interval between the two peaks of step response.

3.3.2.3 Limiting/Deciding Factors

In order to decide which approach is more suitable for my experiment the following reasons were taken into account:

1. The IMU refresh rate was the most important deciding factor because the maximum refresh rate of the IMU was 87Hz. It means if I sample the gyro rate of helicopter, which was the output of the system in this case, at least

twice (as required by Nyquist Criterion) the maximum allowed frequency of the input wave can be 43Hz.

2. The response of the helicopter over a prolonged period of time is more prone to errors due to gyro drifts. This discourages the step response analysis because in order to find the characteristics of the step response we need to see the output till it settles down to a steady state value.

Motivated by these compelling reasons, it was reasonable to further pursue frequency response analysis for system identification.

3.3.3 Design of Experiment

In the following, setup for the experiment and design of input will be explained.

3.3.3.1 Experiment Setup

The setup to carry out system identification on is shown in Figure 3.1.

Basically two changes were made to the vehicle. Firstly, a light weight hollow plastic rod was tied firmly to its center of gravity. This was done to prevent rotors from striking on ground in case of any imbalance. Secondly, all resulting four corners were tied to the ground using strings in such a way that it could not hit the ground while hovering or while the input is applied to it.

3.3.3.2 System Block Diagram

The block diagram of the system is shown in Figure 3.2. As the objective of

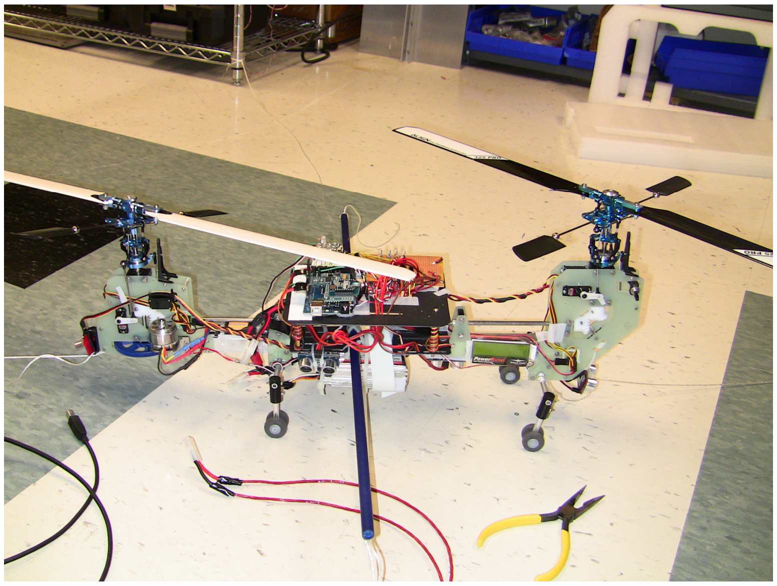


Figure 3.1: Setup for System Identification

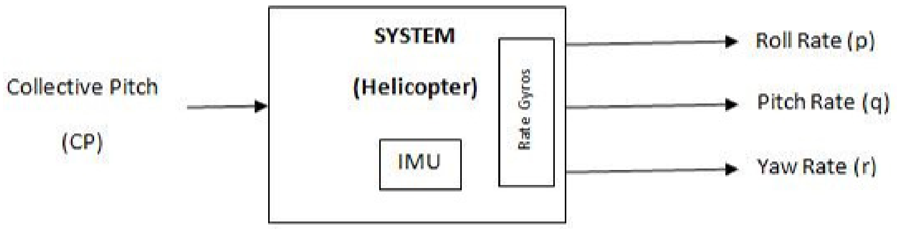


Figure 3.2: Input/Outputs of System

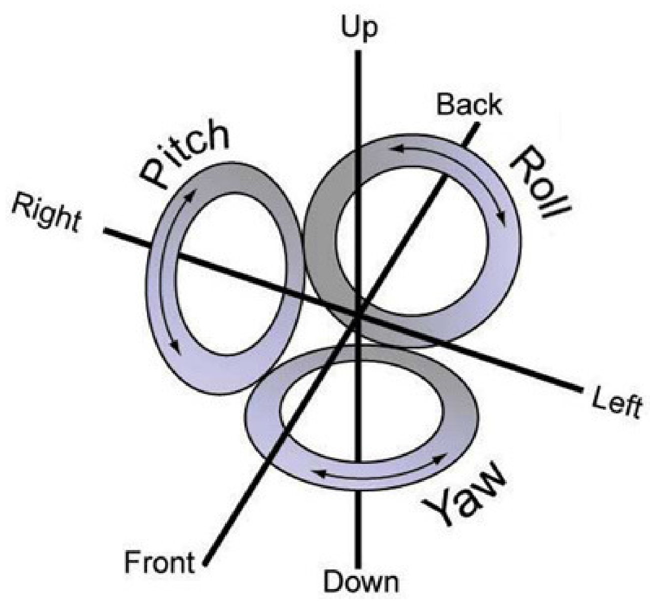


Figure 3.3: Roll, Pitch and Yaw of Helicopter

the thesis is to takeoff, hover and land the helicopter and no sideways maneuvering thus the only input we are interested in is collective pitch (CP). As we know that increasing/decreasing CP results in the helicopter flying up/down but at the same time since it is not stable will try to tilt in other directions. The magnitude of tilt in one direction is in accordance with the CP applied. In fact, we want to find exactly what is the relationship between the two. Now considering the helicopter as a rigid body, tilt can be along any axis in the three dimensional space. These are named roll, pitch and yaw depending on the rotation about an axis with reference to the aircraft as shown in Figure 3.3. The roll, pitch and yaw rates indicate how fast the orientation of heli is changing and are measured in degrees/sec or rad/sec.

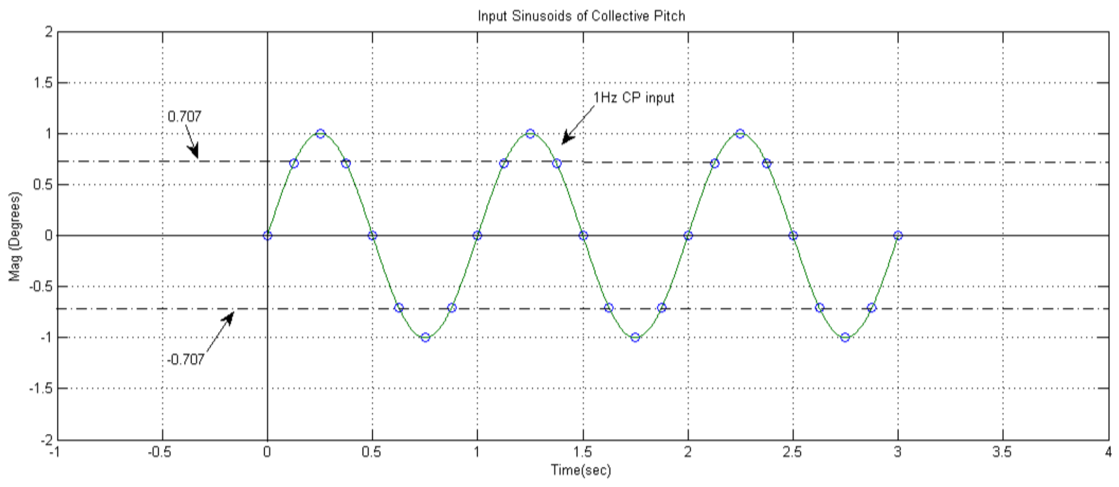


Figure 3.4: Input Sinusoids of Collective Pitch

3.3.3.3 Input Design and Data Logging

1. **Input**

   The next step was to design CP sinusoids of different frequencies from 1Hz to 43Hz. I have written Arduino code to achieve this. The code is responsible for changing the positions of the servos in such a manner that the collective pitch varies between hover pitch ± 1 degree. The frequency is varied by varying the position of servos within the time period corresponding to that particular frequency. The values (or offsets about the helicopter hover position) used are listed in table 3.1 and three sinusoids of input of 1 Hz are pictorially shown in Figure 3.4.

   where $H_p = 10^o$ indicates the CP when the helicopter is hovering. Similarly $S_o = 180$ indicates the corresponding servos offset in microseconds for that CP. The + or - in other values means that these values are added or subtracted

Angle (rad)	0	$\pi/4$	$\pi/2$	$3\pi/4$	π	$5\pi/4$	$3\pi/2$	$7\pi/4$
CP(deg)	H_p	$+0.707$	$+1$	$+0.707$	H_p	-0.707	-1	-0.707
Servo Offset	S_o	$+8$	$+12$	$+8$	S_o	-8	-12	-8

Table 3.1: Servo Offsets to generate one CP sinusoid at hover position

from the horizontal hover values. Three such sinusoids were generated so that a maximum phase difference of 6π can be detected. Also the amplitude of the resulting output could be averaged out over three sinusoids to reduce the effect of noise.

2. **Logomatic V2**

   At output the data is collected twice as fast as the input frequency, that is 2 samples of each of three gyro output are collected within each cycle of the output sinusoids. This is the maximum sampling rate possible in order to vary input up to 43 Hz. This data is initially logged to Logomatic V2, shown in Figure 3.5, having a mini-SD card as long as the helicopter is in testing mode. After testing the data is copied to a computer for analysis.

3.3.4 Relation of Model with the Measurements

The resulting output data is plotted in terms of both the amplitude and phase against each frequency. This will generate the bode plots for roll, pitch and yaw as shown in Figures 3.6, 3.7 and 3.8 respectively. It is important to note that since the gyro data has significantly high variance so the smooth curves in the above

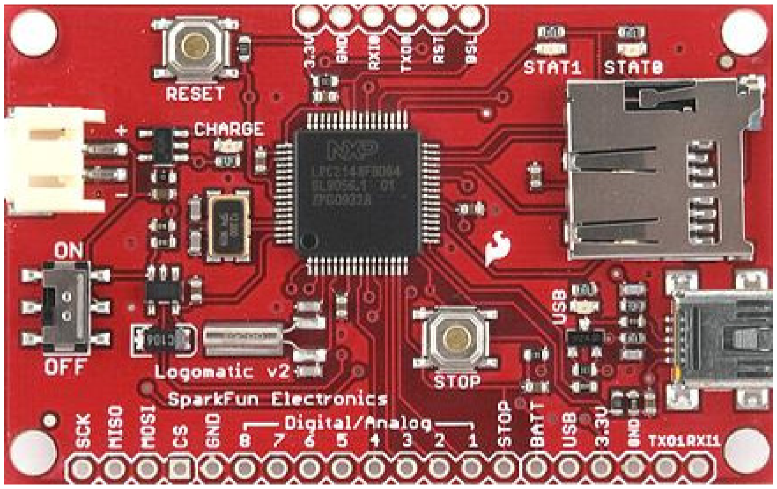


Figure 3.5: Logomatic V2 (Sparkfun)

mentioned figures are obtained by curve fitting. The order of the polynomial used for curve fitting is determined quantitatively by comparing the Mean Squared Error (or MSE) of several different orders as mentioned in table 3.2.

From these three figures, parameters for the second order transfer function for each of roll, pitch and yaw to CP were determined as discussed in previous section. Those parameters are listed in table 3.3.

Following are the transfer functions found from the parameters calculated in table 3.3. The pole locations are also listed in table 3.3 and are shown graphically in Figures 3.9, 3.10, 3.11, 3.12, 3.13 and 3.14.

$$\mathbf{H_{Roll}(s) = \frac{423.122}{s^2 + 19.0332s + 267.7315}} \tag{3.13}$$

$$\mathbf{H_{Pitch}(s) = \frac{228.1936}{s^2 + 11.6224s + 120.125}} \tag{3.14}$$

$$\mathbf{H_{Yaw}(s) = \frac{80.73}{s^2 + 19.058s + 267.69458}} \tag{3.15}$$

MSE	**Roll**	**Pitch**	**Yaw**
Order 2	16.5749	23.3842	18.0643
Order 3	10.7900	23.7381	17.5785
Order 4	10.4416	23.5595	14.3477
Order 5	10.4690	21.8562	14.9597
Order 6	10.6201	21.3851	15.4878
Selected	3	5	4

Table 3.2: Choice of order of polynomial for curve fitting in Bode Plots

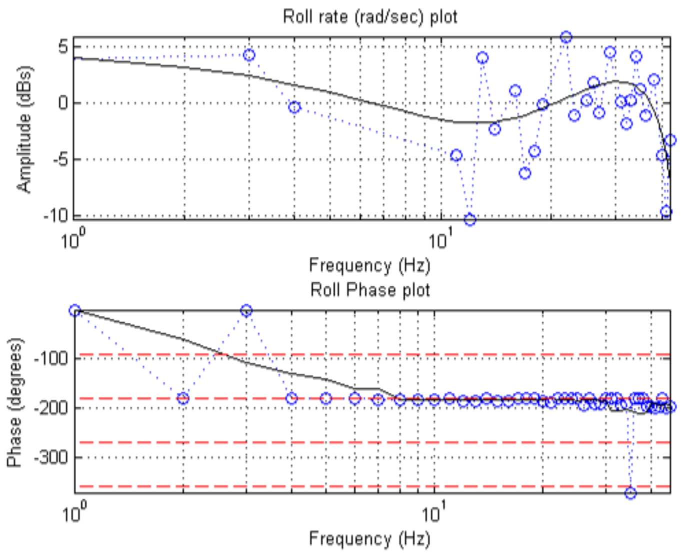


Figure 3.6: Roll Rate Bode Plot

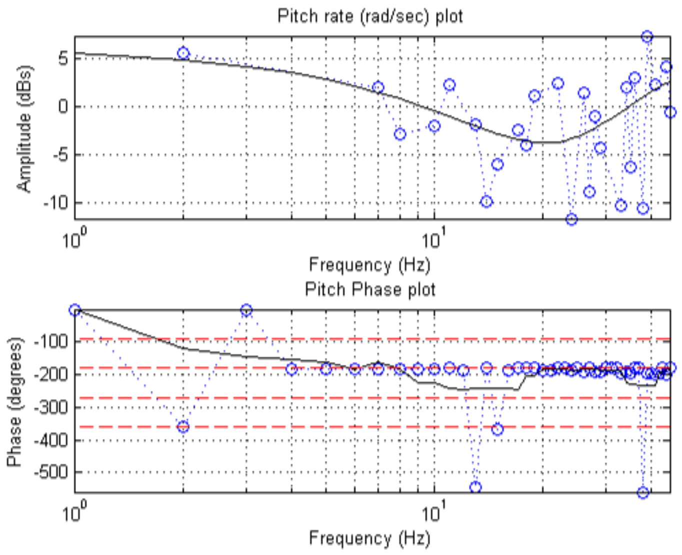


Figure 3.7: Pitch Rate Bode Plot

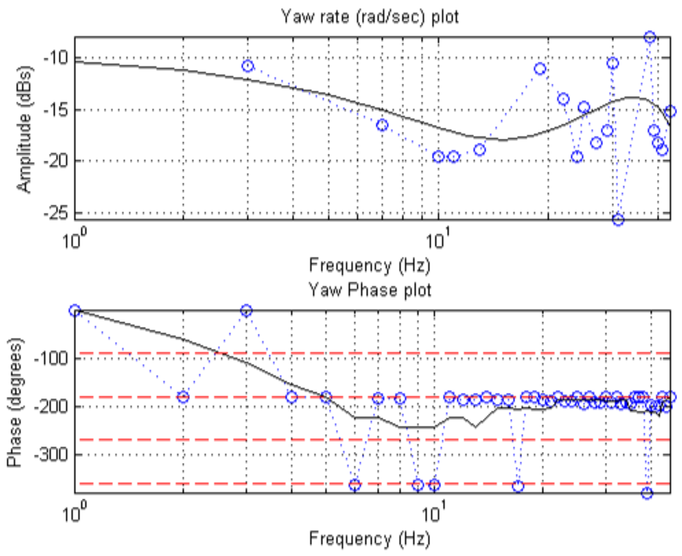


Figure 3.8: Yaw Rate Bode Plot

Parameters	**Roll**	**Pitch**	**Yaw**
K	1.58039	1.89962	0.301577
$\mathbf{f_n}$	2.6042	1.7443	2.603993
$\omega_{\mathbf{n}}$	16.3625	10.9602	16.36
ζ	0.5816	0.5302	0.58243
Pole 1	$-9.5166 + 13.3104i$	$-5.8112 + 9.2928i$	$-9.5294 + 13.2998i$
Pole 2	$-9.5166 - 13.3104i$	$-5.8112 - 9.2928i$	$-9.5294 - 13.2998i$

Table 3.3: Calculated Parameters of Roll, Pitch and Yaw Transfer Functions

3.3.5 Verification of Obtained Model

It was noticed that the comparison of the approximated and the curve drawn with the transfer functions obtained have a MSE of 10^{-3} as far as the magnitude response of the bode plot is concerned. Phase plot in this case has almost no error as phase was just used to find the order of the transfer function. This verifies that the model is reasonable under the assumptions made earlier.

3.4 PID Introduction

A Proportional-Integral-Derivative (or PID) is a well known feedback control mechanism widely applicable in almost all types of industrial processes. This popularity of PID controllers can be attributed partly to its robust performance in a wide range of operating conditions and partly to its functional simplicity [21].

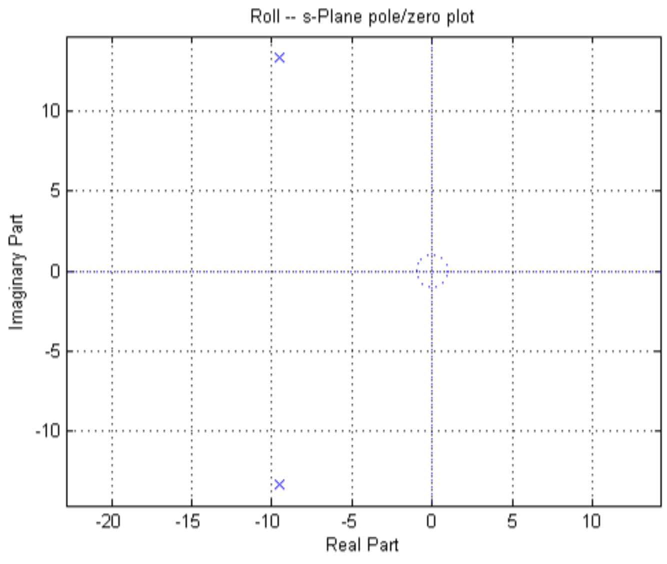


Figure 3.9: Roll Transfer Function Poles in s-plane

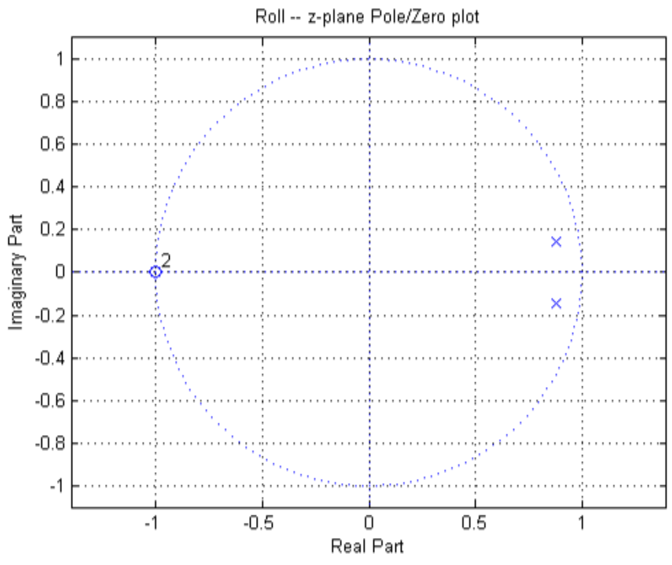


Figure 3.10: Roll Transfer Function Poles in z-plane

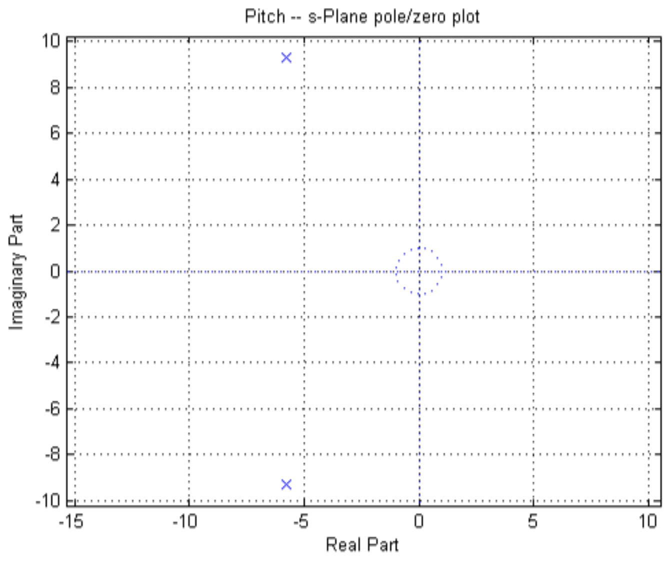


Figure 3.11: Pitch Transfer Function Poles in s-plane

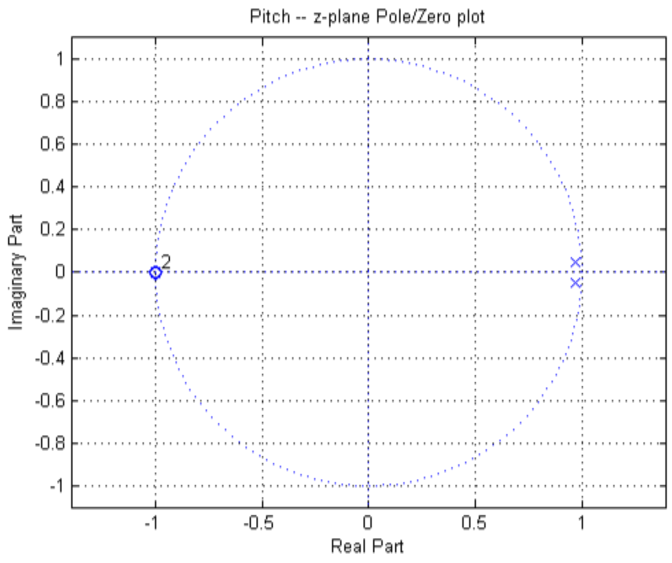


Figure 3.12: Pitch Transfer Function Poles in z-plane

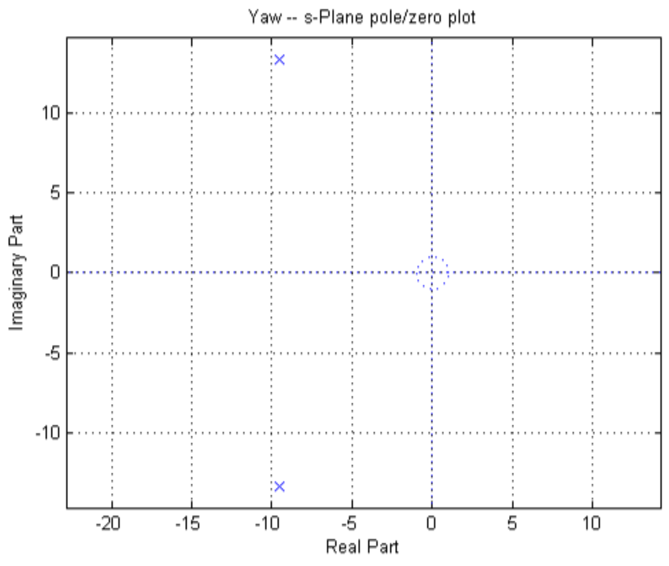


Figure 3.13: Yaw Transfer Function Poles in s-plane

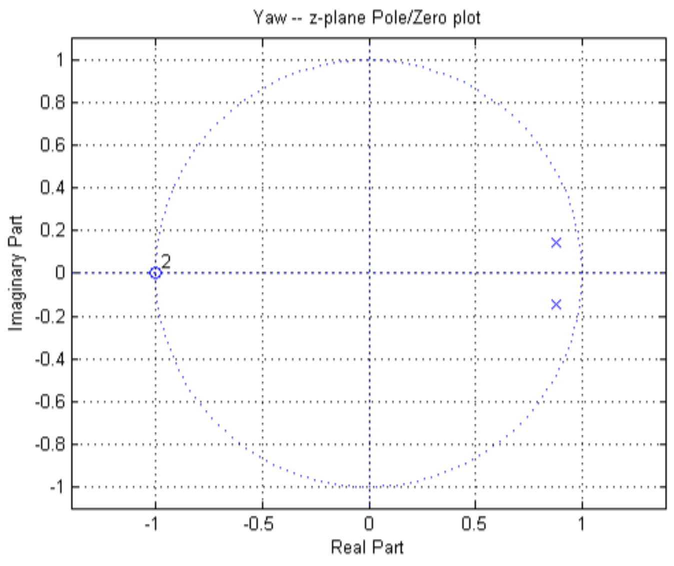


Figure 3.14: Yaw Transfer Function Poles in z-plane

The basic idea behind a PID controller is to read a sensor and then compute the desired actuator output by calculating proportional, integral and derivative responses of error and summing those components up. Let us first get familiar with the terminology:

3.4.1 Terminology related to PID

Following are a few important terms related to PID and will be used frequently later on so are important to be mentioned here.

3.4.1.1 Process Variable

In a typical control system *Process Variable* is a system parameter that needs to be controlled, such as temperature, pressure or in our case a particular gyro rate.

3.4.1.2 Set Point

Set Point is a desired or commanded value for a process for example 50^oC in case of temperature or a vector of zero gyro rates i.e, $\mathbf{a} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ in our case.

3.4.1.3 Step Response

The control system design process begins by defining the performance requirements. The performance is often evaluated by observing the response of a step input to the system. Here are a few parameters that are usually important:

1. **Rise Time**

Rise Time is the time taken by the system to move from 10% to 90% of the steady state value.

2. **Percent Overshoot**

   Percent Overshoot is the percentage amount by which the process variable has gone above or below the desired steady state value.

3. **Settling Time**

   The amount of time required by the system to get into a ball of certain radius (usually 5%) encompassing the steady state value.

4. **Steady-State Error**

   Steady State Error as the name suggests is the final difference between the value the system reaches and the actual value commanded.

All of these parameters are grapically shown in Figure 3.15.

3.4.1.4 Disturbance Rejection

This is a measure of how well the designed closed loop control system is able to overcome any unexpected disturbance.

3.4.1.5 Loop Cycle

The interval between calls to the control algorithm to calculate output and apply adjustment is called *Loop Cycle*. In our case the loop cycle should never exceed 20 ms as this is the time after which the servo needs fresh angle information

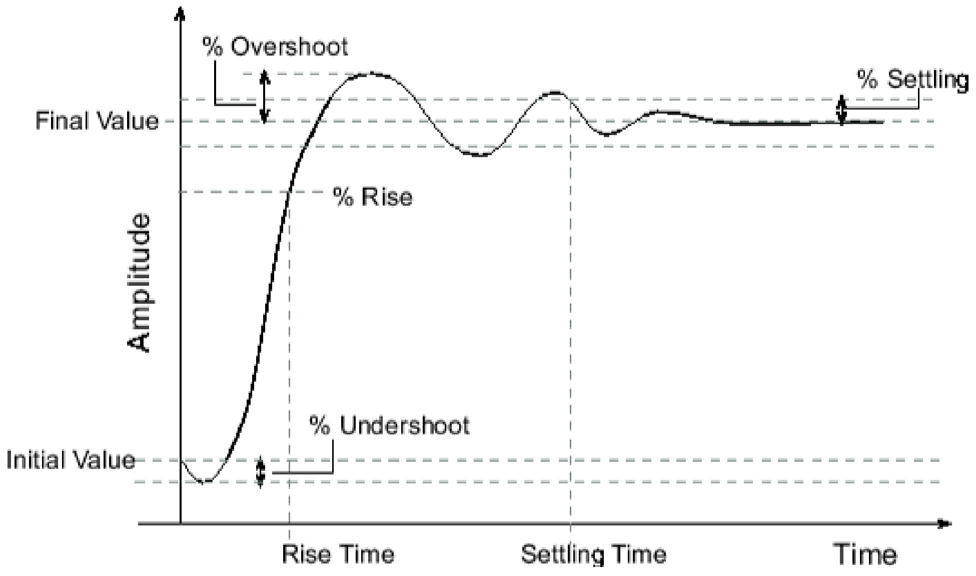


Figure 3.15: Step Response Related Terminology [24]

to maintain correct position.

3.4.2 Choice of Suitable Controller

Below I will describe functioning of each of P, I and D controllers and their importance for helicopter stabilization. Figure 3.16 shows a general PID controller block diagram that will be used later for reference.

3.4.2.1 Proportional (P) Response

Proportional response only depends on the current error i.e. difference between the process variable current value and the set point. The *Proportional Gain (K_p)* determines what is the error to controller output response.

$$\mathbf{P_{out} = K_p e(t)} \tag{3.16}$$

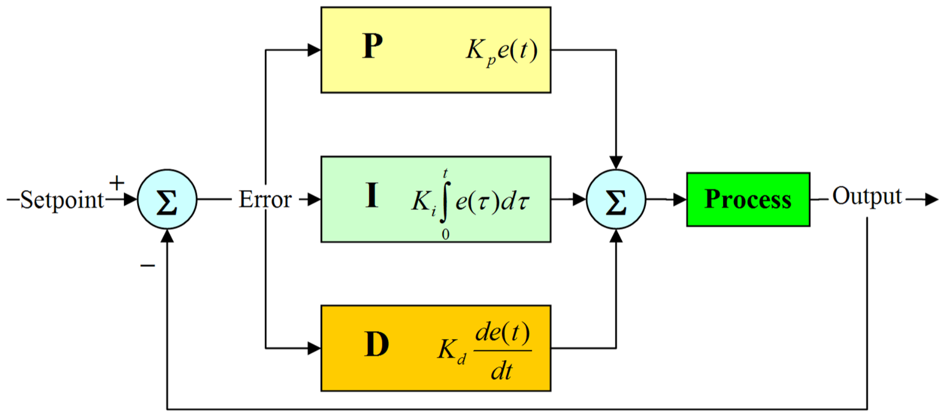


Figure 3.16: General PID block diagram [25]

The calculation of constant K_p depends upon the expected error range and the corresponding expected output that would serve as input to actuators like servos. Increasing the K_p will have the effect of a faster response, that is rise time will decrease but if not done in a controlled way it can make the system unstable by driving the poles to right-half plane (RHP) of the s-Plane. Another problem with the P controller is that it gives a steady state error and never reaches the desired value.

3.4.2.2 Integral (I) Response

Unlike P response the integral response depends on the previous errors as well. In other words it accumulates the errors over time and then generates output based on history of errors. The integral output will continuously increase over time thus increasing the speed of actuators depending on its response to previous controller outputs. The output start to decrease once the overshoot occurs and continues in this periodic fashion till error reaches zero. Given below is the equation for the Integral control only:

$$\mathbf{I_{out}} = \mathbf{K_i} \int_{\mathbf{0}}^{\mathbf{t}} \mathbf{e}(\tau)\, \mathbf{d}\tau \tag{3.17}$$

The I controller accelerates the system response and eliminates the steady state error however it increases the percent overshoot.

3.4.2.3 Derivative (D) Response

The derivative term causes the output to decrease if the process variable is increasing rapidly. The D response is proportional to the rate of change of process variable.

$$\mathbf{D_{out}} = \mathbf{K_d} \frac{\mathrm{d}}{\mathrm{d}\,\mathbf{t}} \mathbf{e}(\mathbf{t}) \tag{3.18}$$

Increasing the $\mathbf{K_d}$ will have the effect of a stronger reaction to the error. Practical systems uses either very small or no $\mathbf{K_d}$ because this response is highly sensitive to noise as it reacts to error at current time only. So if the sensor measuring the process variable is prone to noise or system response is slow, D controller is not used because it can make the system go unstable [23].

3.4.2.4 Proportional-Integral (PI) Response

The PI controller is a balance of complexity and capability that makes it the most widely used algorithm in process control applications [27]. PI controllers have two parameter to adjust which makes them challenging to implement. However

Integral Action enables the PI controller to remove steady state error while keeping the swift response property of P controller intact. Given below is the equation of output to error response of PI controller:

$$\mathbf{C_{out}} = \mathbf{K_p e(t)} + \mathbf{K_i} \int\limits_{\mathbf{0}}^{\mathbf{t}} \mathbf{e}(\tau)\, \mathbf{d}\tau \tag{3.19}$$

Integral action has the effect of continuously resetting the bias value to eliminate offset as operating level changes. There are however two major challenges :

1. The interaction of two tuning parameters, $\mathbf{K_d}$ and $\mathbf{K_i}$, makes tuning difficult.
2. The integral term tends to increase the oscillatory or rolling behavior of the closed loop system.

3.4.2.5 Proportional-Derivative (PD) Response

From the properties of P and D controllers we can infer that this is the least used controller. To state explicitly, due to P controller the final output will still have an offset and due to D controller it is highly sensitive to sensor noise.

3.4.2.6 Proportional-Integral-Derivative (PID) Response

PID controller is more complex. It has three variables to adjust. That is, its activity and performance is based on the values chosen for three tuning parameters, one of which is associated with the proportional, integral and derivative terms. The addition of a third tuning parameter makes the tuning of PID controller a hard task. The addition of derivative however to the PI controller has a positive effect of decreasing the overshoot of response.

Parameter	**Rise Time**	**Overshoot**	**Settling Time**	**SS Error**
Kp	Dec	Inc	Small Change	Dec
Ki	Dec	Inc	Inc	Eliminate
Kd	Small Change	Dec	Dec	Small Change

Table 3.4: Effect of Change in Individual Gain on PID

$$\mathbf{C_{out}} = \mathbf{K_p e(t)} + \mathbf{K_i} \int_{\mathbf{0}}^{\mathbf{t}} \mathbf{e}(\tau)\, \mathbf{d}\tau + \mathbf{K_d} \frac{\mathrm{d}}{\mathrm{d}\,\mathbf{t}} \mathbf{e(t)} \tag{3.20}$$

Table 3.4 lists in detail the effect of variation of a particular gain within PID controller. A pictorial illustration of these effects can be seen in Figures 3.17, 3.18 and 3.19.

3.4.2.7 Controller Chosen: PI

After explaining all different forms of PID control, we can now make a choice as to what controller to implement in case of helicopter hover stabilization. There are a few important system related properties and/or limitations that played an important role in controller selection. They are listed in the order of high to low priority below:

1. Different forms of noise is introduced in the gyro output signals by the vibration of helicopter platform and neighboring electronic components therefore the controller should not be much sensitive to noise.

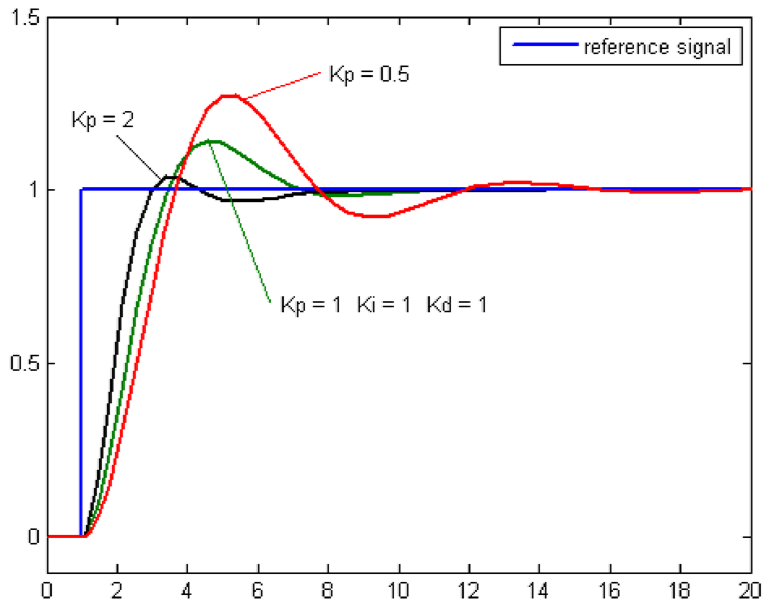


Figure 3.17: Effect of changing Kp in PID [25]

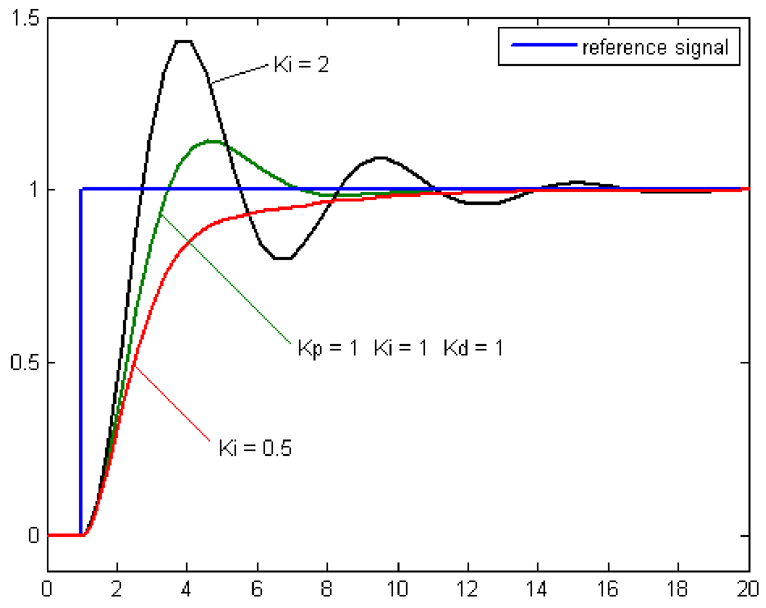


Figure 3.18: Effect of changing Ki in PID [25]

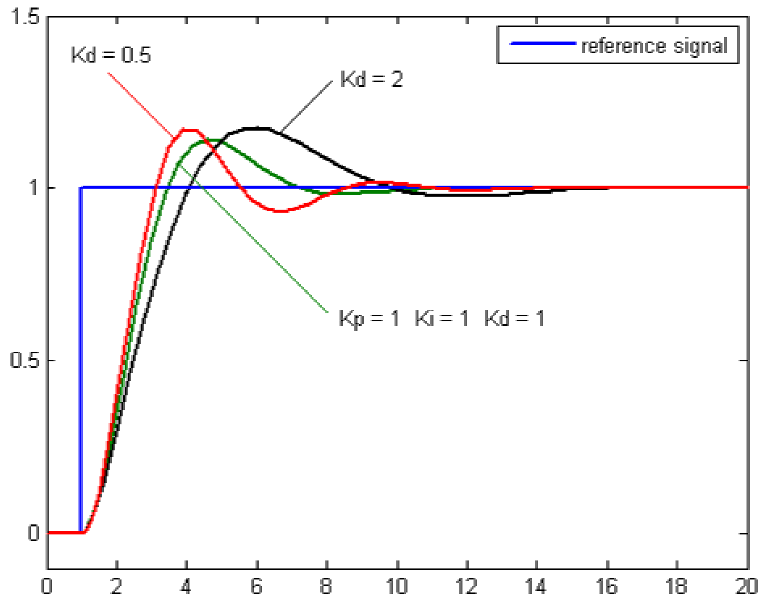


Figure 3.19: Effect of changing Kd in PID [25]

2. The time for computation of controller output and performing other task like acoustic sensor distance measurement, speed and pitch variation of rotors and fetching/processing data from IMU should not exceed 20ms.

3. The number of gains to be tuned should be as less as possible because tuning requires a hovering-helicopter to be a part of closed loop and could be both unsafe and tiresome while experimenting.

Based on these criteria, the discussion of this section about different form of PID control and the reasons for rejecting other controllers given below, I came to the conclusion that PI controller is best suited for my application. Table 3.5 lists the reasons why other controllers were not good options.

Controller	**Reason/Reasons of Rejection**
P	There is a constant steady state error
I	High overshoot and prolonged oscillatory behavior
D	Too sensitive to noise
PD	Disadvantages of both P and D
PID	Difficult to tune, more time for calculating output

Table 3.5: Reasons for not choosing other forms of PID

3.4.2.8 Development of PID Library for Arduino

Since Arduino provides an open source environment a researcher can contribute to different problems of practical interests by developing codes for them and making them available for others to use. As a matter of fact there exist a PID library for Arduino already but following were the reasons that motivated me to develop one of my own.

1. The existing library doesn't allow one to change the P, I and D gains on the go which makes tuning process very difficult.

2. There is no minimum or maximum integral state limits to prevent integral windups.

3. There is no way for verification of what values of gains etc are commanded and what are in fact used. In other words the gains once declared at the start of the program can't be accessed individually later on.

The library that i developed takes care of these issues.

3.5 Design of Compensators

As discussed in the previous sections I chose PI compensator to be used for the 'zero-rate-hover' objective. Matlab's SISOTOOL was used for compensator design. The transfer functions developed in the previous section were used as the open loop systems. Each of the compensators was developed assuming that there is no mechanical coupling between the individual systems.

3.5.1 Limits on Parameters of Interests

Since our input (CP) is varied in steps when vehicle takes off or lands therefore an evaluation of step response of CP on the outputs (Roll, Pitch and Yaw rates) is very important and defines the behavior of the vehicle. Since each of the outputs has different sensitivity to input so the performance criteria for each compensator were decided based on the corresponding open loop response.

3.5.1.1 Roll Compensator Design

Figure 3.20 shows the behavior of the open loop step response for roll of the vehicle. The most important consideration in this case is to decrease the percent overshoot $\%OS$ because a twin rotor helicopter has a natural tendency to tilt sideways when the front and back rotor collective pitches are perfectly synchronized. Therefore a higher $\%OS$ means vehicle will tilt very aggressively to one side, which is not desirable. As we can see the $\%OS$ is 29.2% which is fairly high. A PI compensator with the following considerations was developed for roll i.e. $\%OS < 20\%$

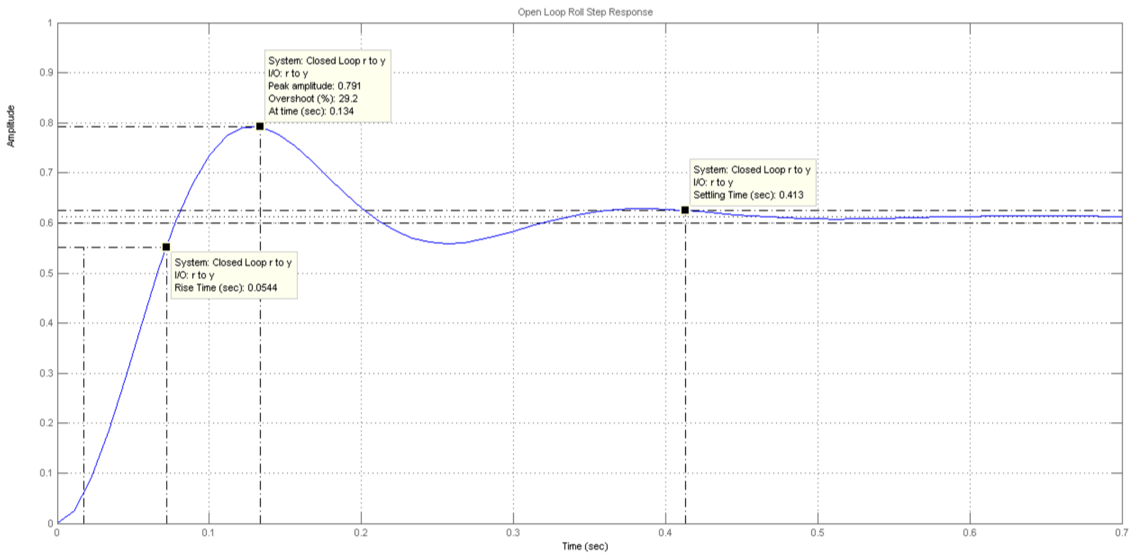


Figure 3.20: Open Loop Step Response of Roll-CP System

and $settlingtime < 0.78sec$ (shown in Figure 3.21). The corresponding new closed loop step response of the system is shown in Figure 3.22. It is also important to note that the steady state error seen in Figure 3.20 is also eliminated by the compensator as is visible in Figure 3.22.

3.5.1.2 Pitch Compensator Design

Figure 3.23 shows the open loop step response for the pitch of the vehicle. Pitch rate is the most sensitive among the three rates and most important one because the cyclic pitch used to correct any pitch perturbation is coupled mechanically with the cyclic pitches for roll and yaw. Therefore again the more the $\%OS$, the more are the system's chances to induce perturbations by itself. Because of the sensitivity of

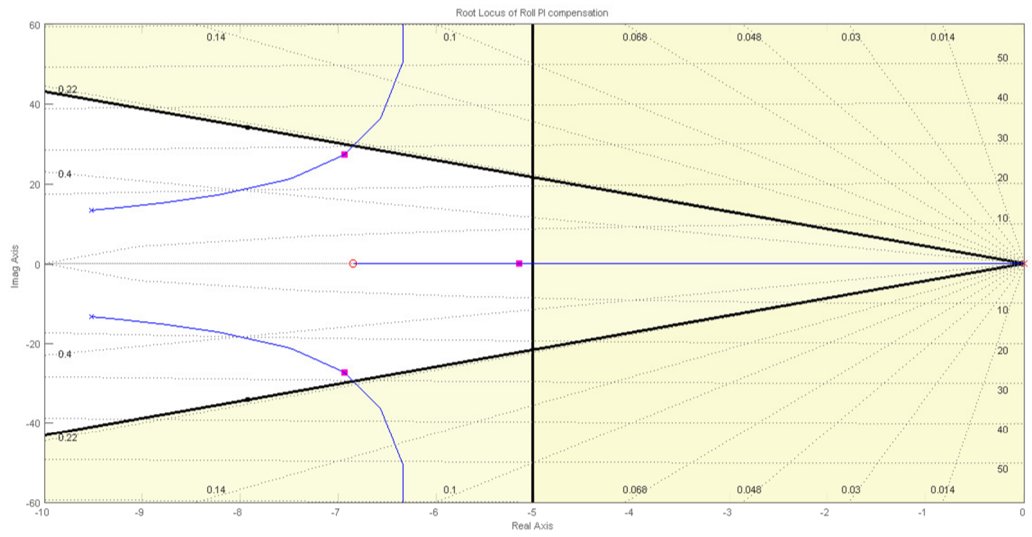


Figure 3.21: Root Locus of PI Compensator developed for Roll

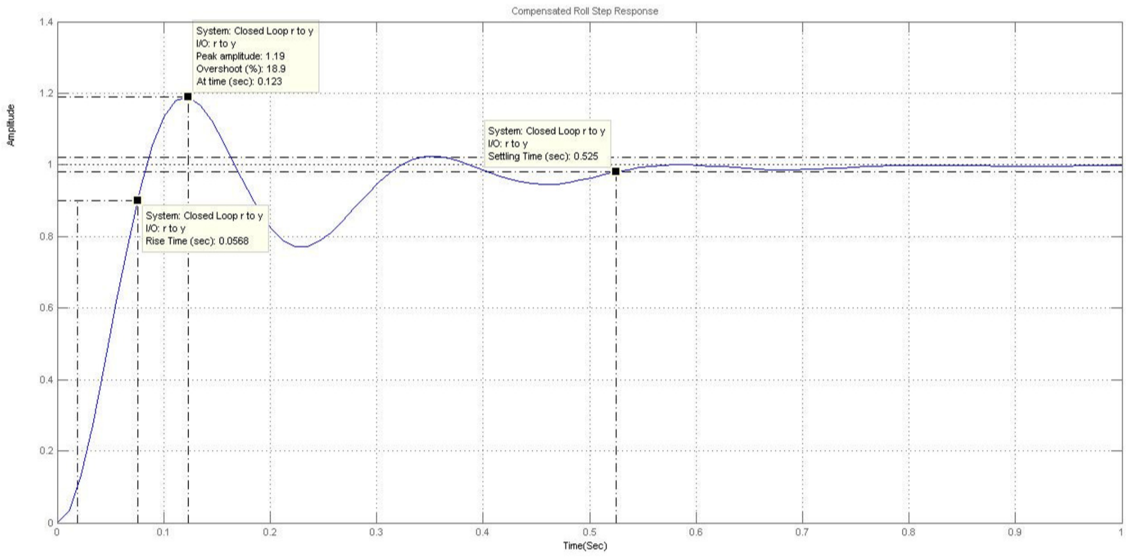


Figure 3.22: Closed Loop Step Response of Roll-CP System

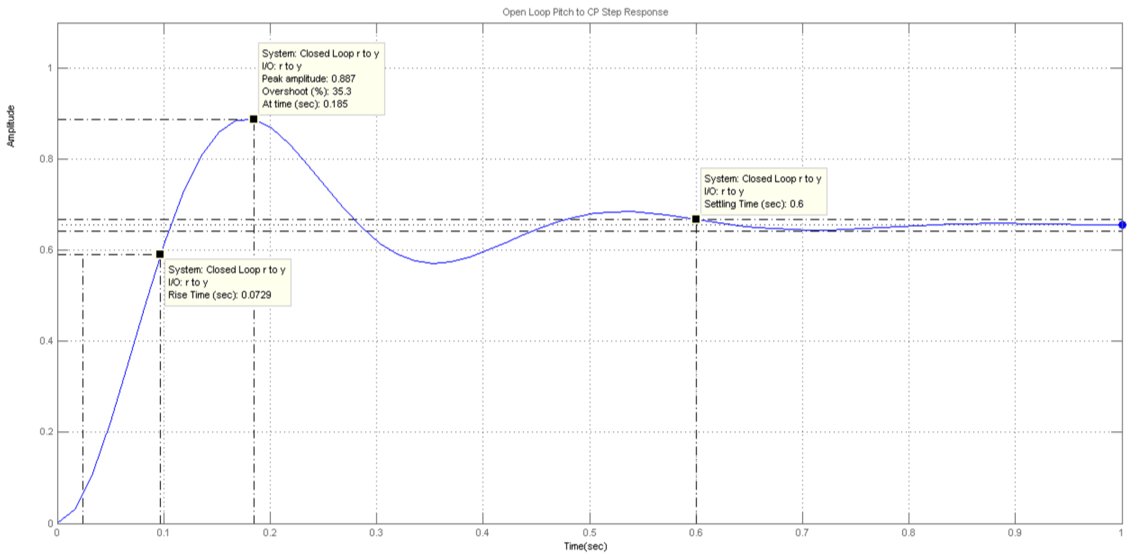


Figure 3.23: Open Loop Step Response of Pitch-CP System

pitch, the $\%OS$ is designed to be negligible i.e. 3.32% as shown in Figure 3.25.

A PI compensator developed for pitch is shown in Figure 3.24.

Again, the steady state error seen in Figure 3.23 is eliminated by the compensator as is visible in Figure 3.25.

3.5.1.3 Yaw Compensator Design

Though the $\%OS$ of the open loop step response of yaw of the vehicle is already within an acceptable range i.e. 10.5%, Figure 3.26, it is better to prevent yaw movement from oscillations. Therefore, no OS was allowed in the design of the yaw compensator as shown in Figure 3.28. The steady state error is eliminated as well. A PI compensator developed for yaw is shown in Figure 3.27.

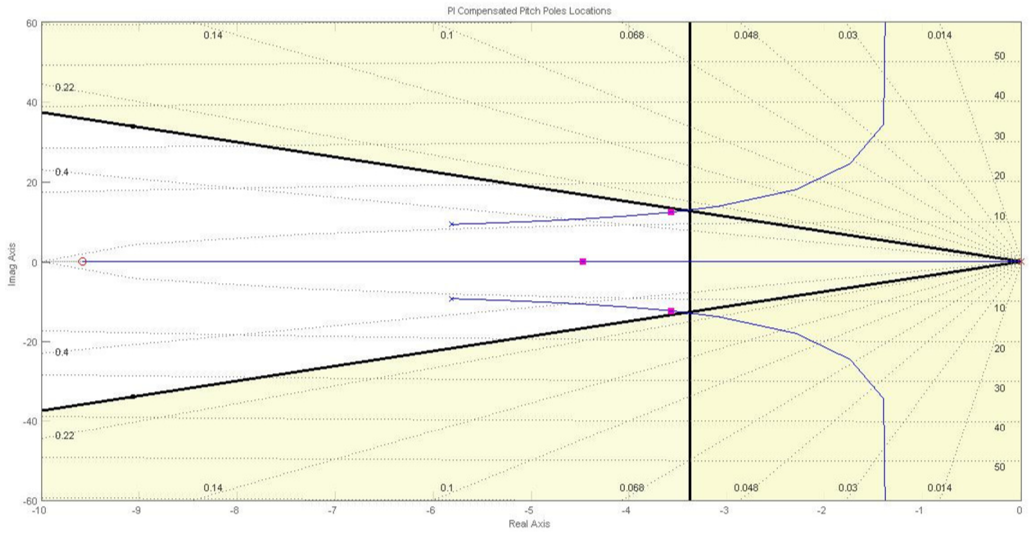


Figure 3.24: Root Locus of PI Compensator developed for Pitch

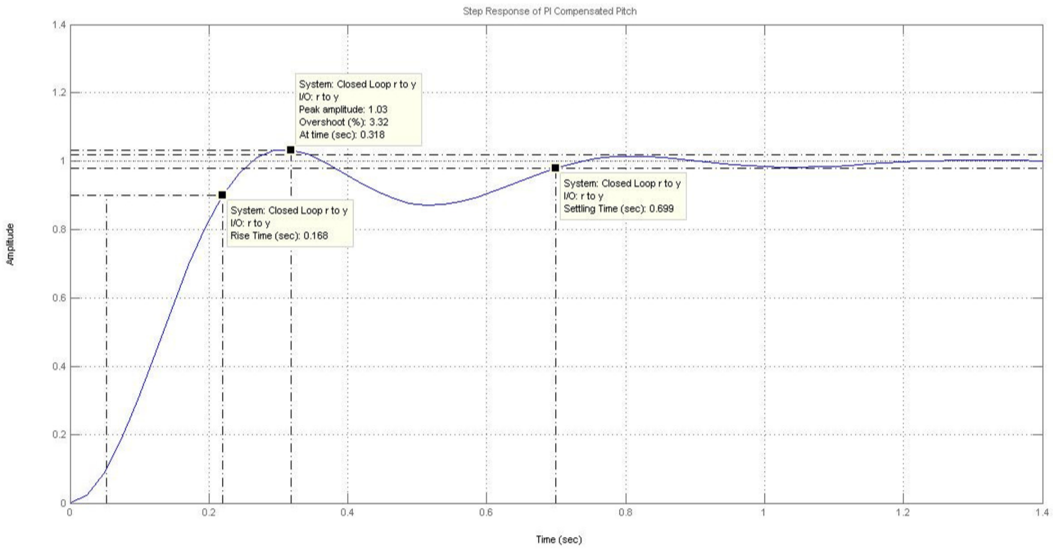


Figure 3.25: Closed Loop Step Response of Pitch-CP System

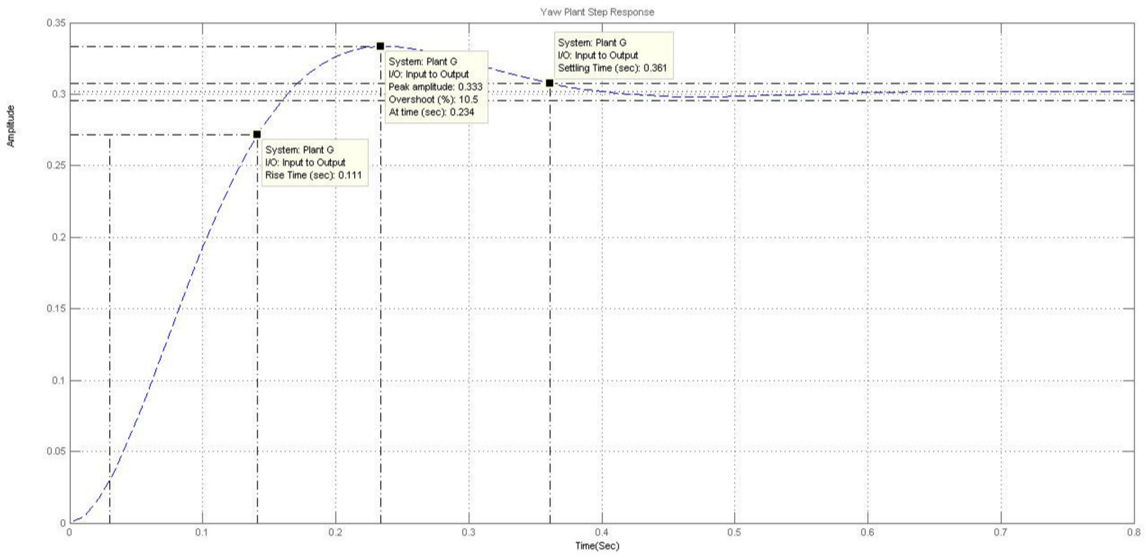


Figure 3.26: Open Loop Step Response of Yaw-CP System

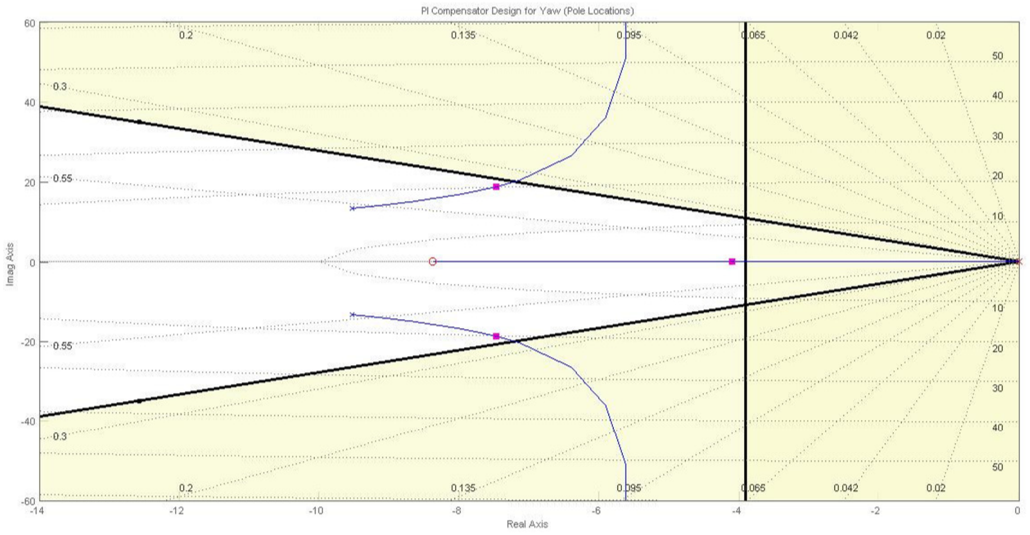


Figure 3.27: Root Locus of PI Compensator developed for Yaw

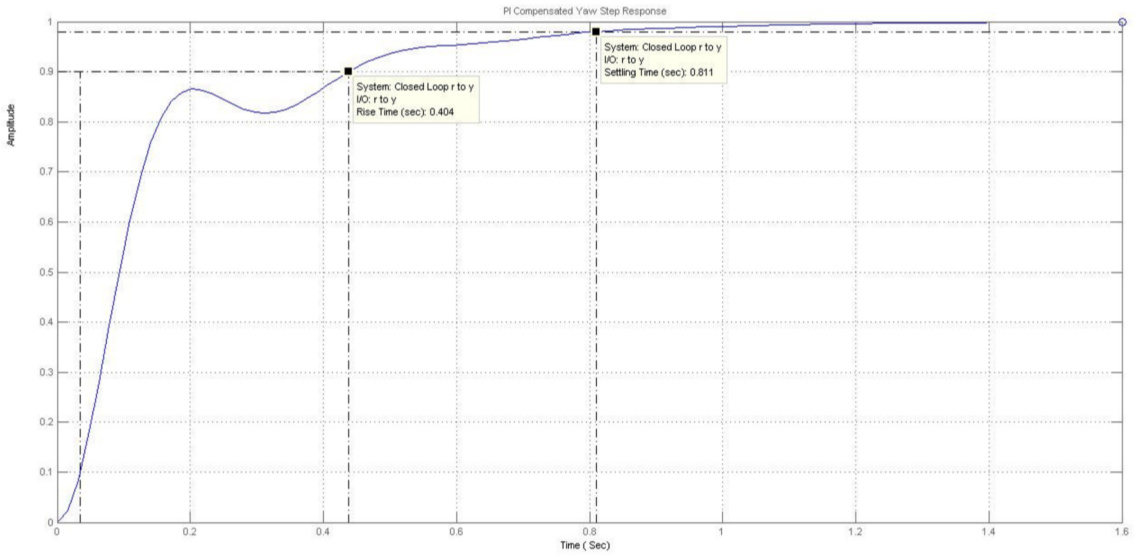


Figure 3.28: Closed Loop Step Response of Yaw-CP System

3.5.2 Practical Implementation of Compensators

Table 3.6 shows the gains obtained for proportional and integral compensators designed above.

The compensators designed in previous subsection were practically implemented on the helicopter using the PID library that I developed. The detailed algorithm in the form of a flow graph is shown in Figure 3.29. It can be noticed that there are priorities given to each controller based on its sensitivity. Moreover compensators are implemented in a cascade fashion not in parallel which means that instead of applying all compensators on the same IMU output, an IMU output is processed from one compensator to another in a serial fashion so that the transition between them should be smooth .

Gains	K_p	K_i
Roll	1.4081	9.6455
Pitch	0.33835	3.2417
Yaw	2.4585	20.6047

Table 3.6: Gains of Compensators Designed

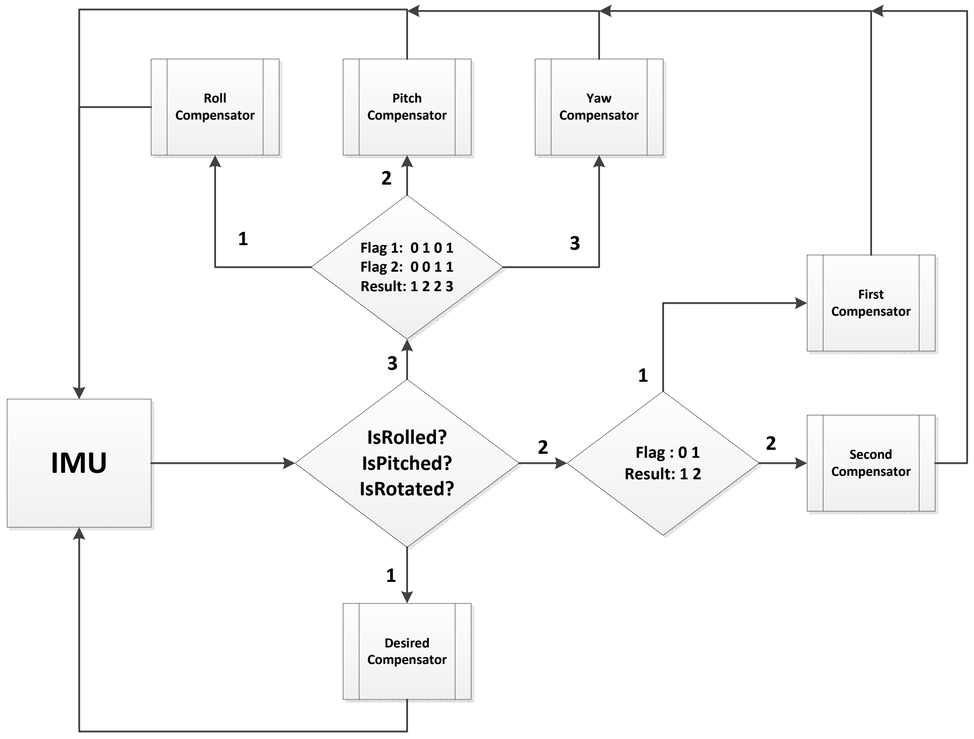


Figure 3.29: Compensators' Algorithm Flow Graph

Gains	**OldK_p**	**OldK_i**	**NewK_p**	**NewK_i**
Roll	1.4081	9.6455	1	8
Pitch	0.33835	3.2417	0.33835	4
Yaw	2.4585	20.6047	2	17

Table 3.7: Adjusted Gains of Compensators

3.5.3 Fine Tuning of Compensators' Gains

As each of the compensator was developed independently therefore when implemented in the system the gains needed to be adjusted. Moreover, the cyclic pitches used for individual orientation corrections are mechanically coupled and will therefore impact each other which were neglected during the design phase as these are difficult to predict in advance. Due to these reasons one needs to fine tune the compensator gains. These gains were adjusted by keeping the hovering helicopter in the loop and changing the gains step by step around the values listed in table 3.6. The new gains obtained that resulted in the final stable hover of the helicopter are given in table 3.7 which are not much different than the ones obtained during the design process therefore confirming that the model developed correctly represents the dynamics of the system.

3.5.4 Quantitative Analysis of Stability Achieved

The resulting system was tested for stability during takeoff, hover and landing. A quantitative analysis is carried out to measure the stability achieved by the use of

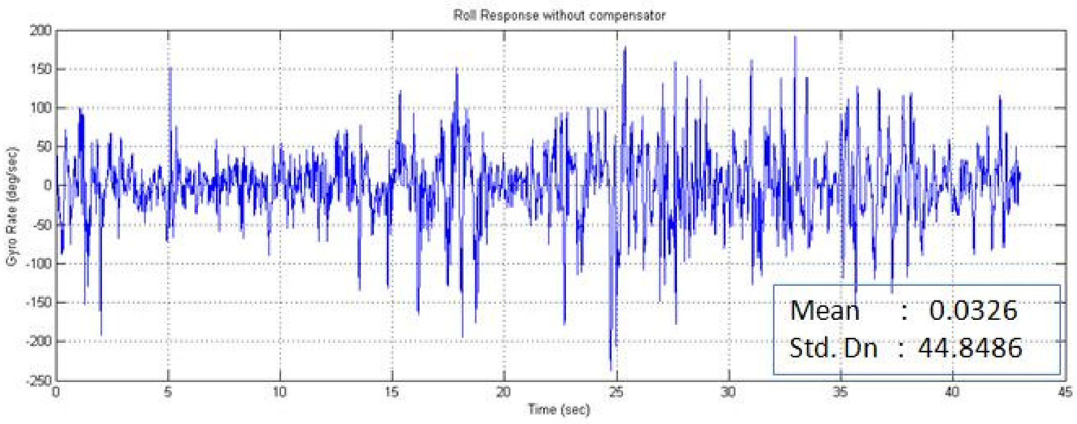


Figure 3.30: Roll Gyro Output With Compensators Off

compensators. Since the measure of the instability of gyro outputs is the deviation of the gyro values from their values when the vehicle is stationary i.e. all zeros. Therefore, standard deviation is a reasonable measure to show how those variations are reduced once the controllers are applied. Figure 3.30 and Figure 3.31 show the Roll gyro output without and with compensators while the helicopter is hovering.

Figure 3.32 and Figure 3.33 show the Pitch gyro output without and with compensators.

Figure 3.34 and Figure 3.35 show the Pitch gyro output without and with compensators.

It is clear from these figures that the tilts in any direction whether it is roll, pitch or yaw is reduced considerably. This is also confirmed by the behavior of the helicopter when compensators are applied to it while hovering. The testing however was carried out indoors as the effect of air gusts were not taken into account. Moreover since a professional platform as used in helicopter design was not available

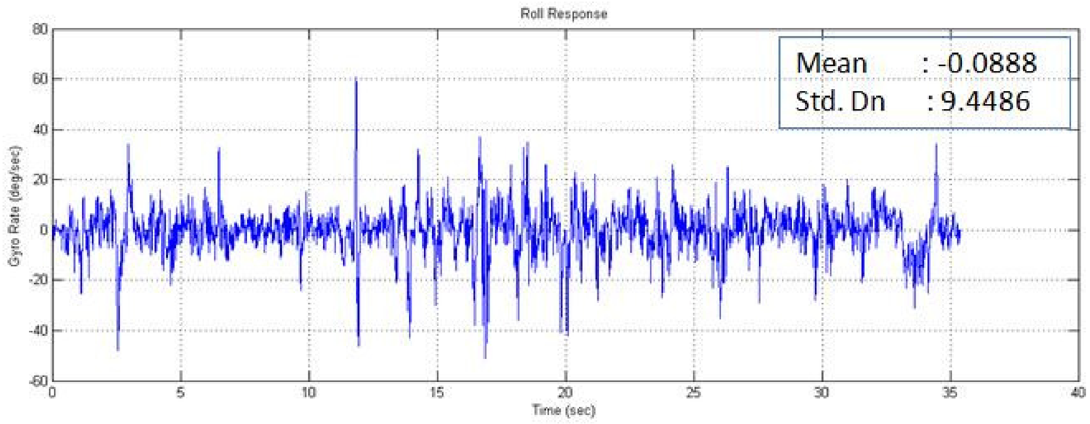


Figure 3.31: Roll Gyro Output With Compensators On

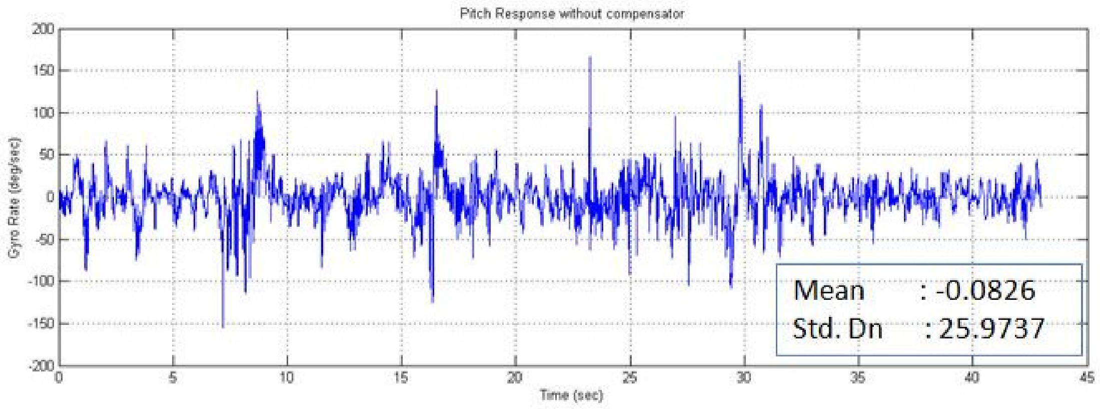


Figure 3.32: Pitch Gyro Output With Compensators Off

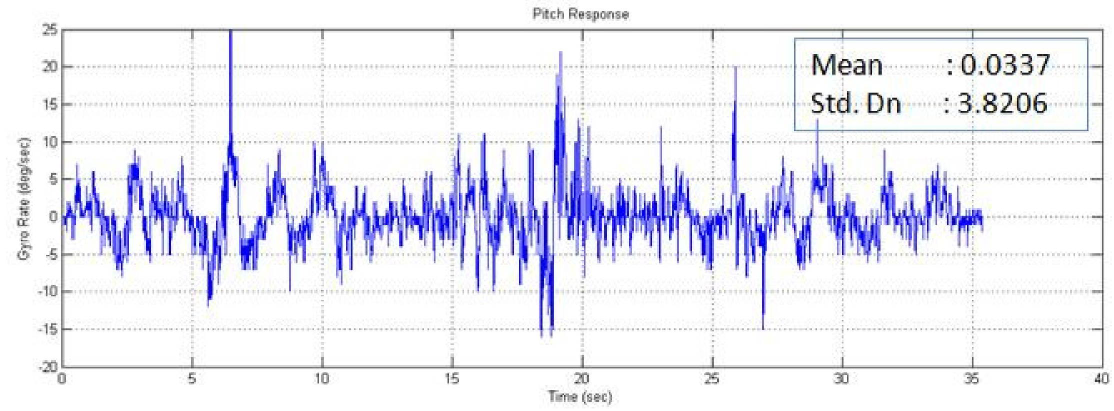


Figure 3.33: Pitch Gyro Output With Compensators On

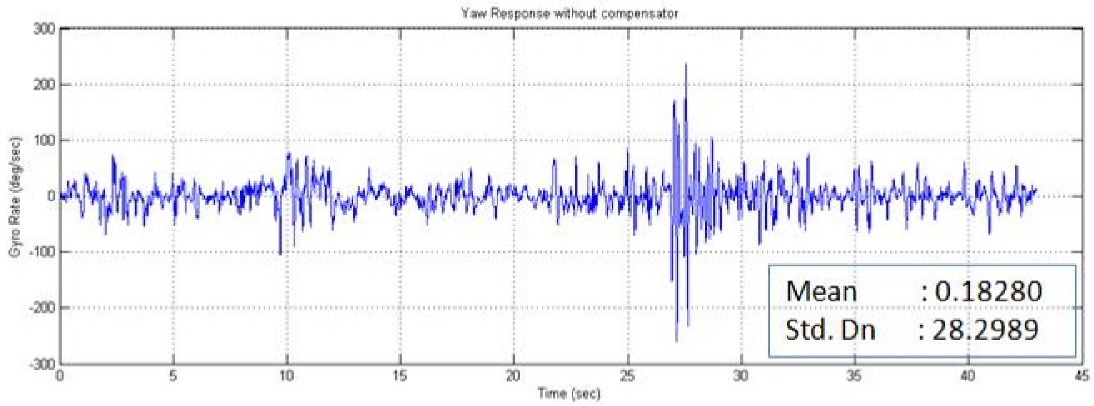


Figure 3.34: Yaw Gyro Output With Compensators Off

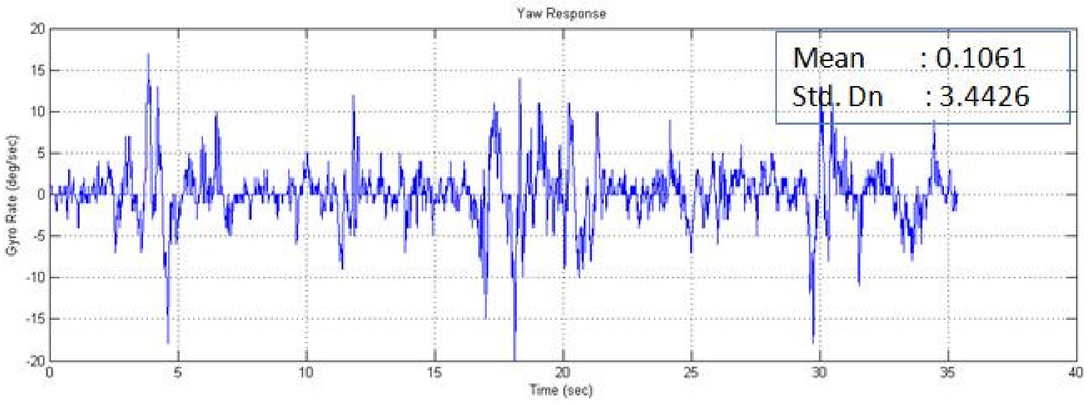


Figure 3.35: Yaw Gyro Output With Compensators On

and the helicopter was tied to ground with strings, designing was performed for close-to-ground hovering and needs adjustment for hovering at few feet above the ground. I will list in detail some hardware modifications in the structure in the next chapter towards that end.

3.6 Project Challenges and their Solutions

At the end, I would like to discuss some common problems that I faced during the project and how those were debugged and finally solved. I learned a lot of things during the project which are not directly related to electronics and control theory but they have to be done in order to complete the project. This section is very important from a development point of view because many things are confronted in the projects that may be theoretically correct or missing in datasheets but they can still halt the progress of one's project. A few of those problems are listed below:

3.6.1 ESC Setup Emulation

Detailed ESC setup is explained in Chapter 2. However it is necessary to mention here that ESC is usually programmed using a standard transmitter. The Twinn Rexx is controlled by 2.4GHz Spektrum DX-5e 5 channel Transmitter. I will not explain again all the setup procedure but it is important to note that since I developed my own communication network of Xbees so I have to make my own transmitter and receiver. This means that I cannot use regular transmitter to program ESC anymore. So I wrote my own code that I used to upload on my transmitter controller whenever I wanted to program ESC (The Arduino code can be found in Appendix A.1). The rest of the details are mentioned in Chapter 2.

3.6.2 CCPM Setup Emulation

CCPM or Cyclic/Collective Pitch Mixing is a complex structure of mechanical linkages that transfer motion of servos from a stationary part to the rotating part (i.e. rotors). All of this is achieved by the swash plate, which is shown in Figure 3.36.

3.6.2.1 Collective Pitch (CP) Adjustment

Collective Pitch is referred as the angle in degrees that rotor blades' edges make with respect to an abstract horizontal disk spanning rotors. It is also called *angle of attack*. The CP can be positive or negative depending on the angle of attack. Figure 3.37 shows the side view of one of the rotor blades both with positive

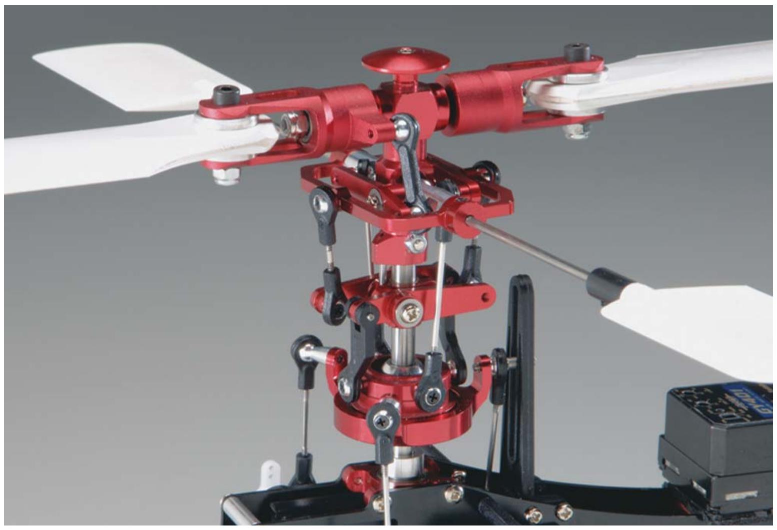


Figure 3.36: Rotor Assembly of an RC Helicopter including Swash plate

and negative CP.

The problem with my helicopter was that it had different front and back rotor assemblies. Therefore in order to set same collective pitch I needed to provide different offsets to the three servos of each rotor. All of this adjustment was done manually using a pitch gauge. The values of these offsets above or below their horizontal or zero pitch positions along with the rotors CP are shown in figure 3.38.

3.6.2.2 Cyclic Pitch Adjustment

The cyclic pitch control is designed in such a way that movement of the cyclic in any direction, will decrease collective pitch on the blade that is pointing in that direction and will increase the collective pitch of rotor in the opposite direction. This means that for forward movement of the helicopter the pitch will be smaller in the

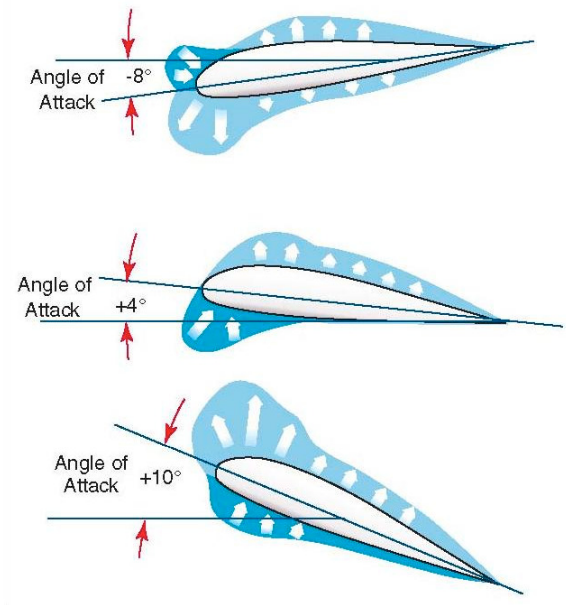


Figure 3.37: Positive and Negative Collective Pitch

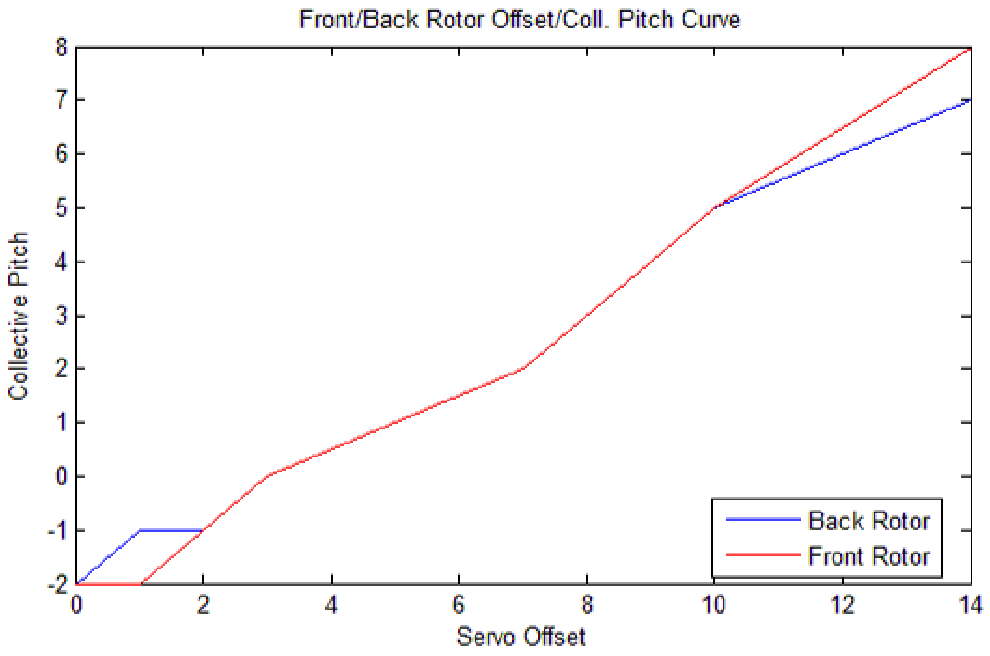


Figure 3.38: Relationship of Offset to Front/Back Rotor CP

Stick Position	Low Stick	2	Mid Stick	4	High Stick
Throttle(%)	0	40	80	90	100
Pitch(deg)	-2	4	7	10	10

Table 3.8: Normal Heli Mode

forward direction as compared to the aft direction of helicopter. This changing of cyclic pitch during each revolution of the rotor blades is known as *Cyclic feathering.*

The Cyclic pitch was also adjusted for forward, backward, left and right directions using pitch gauge.

3.6.2.3 Collective-Throttle Mixing

The *collective throttle mixing* or *pitch curves* are different standard helicopter modes of operation that are preprogrammed usually in an RC helicopter. They account for beginner level or more advanced 3D heli maneuvers. The two most common modes are following:

1. **Normal Mode**

   In normal mode we want to start at slightly negative pitch lets say -2^o at zero throttle to high positive pitch usually 10^o at full throttle. The increase is almost exponential.

   The values of pitch in Normal mode are different depending upon particular helicopter weight and rotor size. I found the values listed in table 3.8 specifically for this particular Twinn Rexx I am using by different experiments.

Stick Position	Low Stick	2	Mid Stick	4	High Stick
Throttle(%)	100	90	80	90	100
Pitch(deg)	-10	5	0	5	10

Table 3.9: Idle Up Heli Mode

2. **Idle Up 1**

   Idle Up Mode 1, is used for more aggressive flying. It is important to note that the idle up mode is always engaged when the heli is already hovering otherwise it can cause heavy damage to the servos and driving motor. The common values of pitch and throttle that I found by experimentation for this mode for my helicopter are listed in table 3.9.

3.6.3 Driving BLDC Motor

Driving a BLDC motor without a proper transmitter and receiver is a hectic task. First of all programming ESC needs to be taken care of which I explained earlier. Another practical problem was of wire gauge.

3.6.3.1 ESC Wire Gauge Problem

Since I was using a power supply for powering the helicopter, I tied up the ESC and the Arduino board with the same wire gauge thinking that theoretically the wire gets hot if it is not able to carry enough current. The problem in this case was that motor was running but it was never able to reach enough rpm to take off.

Some times as it starts to take off the RPM suddenly decrease and it slams back onto ground. Finally after testing different power supplies, I replaced the wire with a really heavy gauge wire and it started to fly. The previous wire was not able to carry more than 10A while the helicopter was pulling the 15A needed to hover. So this was one of the problems where theory was correct but the problem has to be debugged by trial and error procedure.

3.6.4 Electronics Platform and Battery Mounting

I have mounted an aluminium made platform for mounting circuits and controllers but it needed to be corrected and a more better and rigid platform should be built and mounted. This platform is not rigid so it is contributing to the vibrational noise in gyros.

3.6.5 Changing IMU Firmware

As discussed in detail in Chapter 3 in earlier sections that original firmware on Razor IMU was giving out raw sensor data which was of no use. So I needed to use new firmware, which has turned IMU into an AHRS, written by Doug Weibel and Jose Julio and hosted by google code [29].

3.6.6 Refresh Rate of IMU

The final and important problem that I face was of low IMU rate after updating new firmware. I discovered this problem during system identification when I was not able to vary the reference frequency beyond 25Hz in accordance with Nyquist Criterion as the maximum IMU rate was 50Hz. So I needed to increase this rate somehow. I did two modifications to the AHRS code taken from [29]:

1. Increased the Serial baud rate from 57600bps to 115200bps

2. Deleted any extra information that was being printed, which in this case was Euler angles. I could do it without any problem because this step is needed only for system identification phase and can be inverted for controller use later which operates at 50Hz. This was done because Arduino built-in Serial.print() function takes more cycles than any other routine function.

As a result of these actions I was able to increase the refresh rate of IMU to 87Hz which enabled me to vary the reference frequency of input up to 43Hz which was good enough for identification purposes.

Chapter 4

Future Work

In this thesis I developed a linear model for the Twinn Rexx helicopter and the compensators for its rate control for stable hovering but there are a few things that I would suggest to make this project practically more useful.

1. **Hover in Outdoor environments**

   A lot of work needs to be done to make this hover stable for outdoor environments. In heavy gusts of air which can stop UAV at a point other than horizontal position for some time, this controller is not sufficient as the gyros set point (zero rates) will now be indicated at a tilted location. Additionally, gyros are more prone to errors than the Euler angles. The AHRS algorithm used in this thesis to find Euler angles can be employed to close the loop over the closed loop system developed for angle compensation. This outer relatively slower loop will give more hover stability and a firm control over cyclic pitch for lateral/longitudinal maneuvers.

2. **Altitude Control**

   For a more practical flight a controller for maintaining a constant altitude has to be designed. If the intended flight is within 3 meters then the already mounted acoustic sensors can be used, however, for higher altitudes some more sophisticated sensor like a laser sensor can be used.

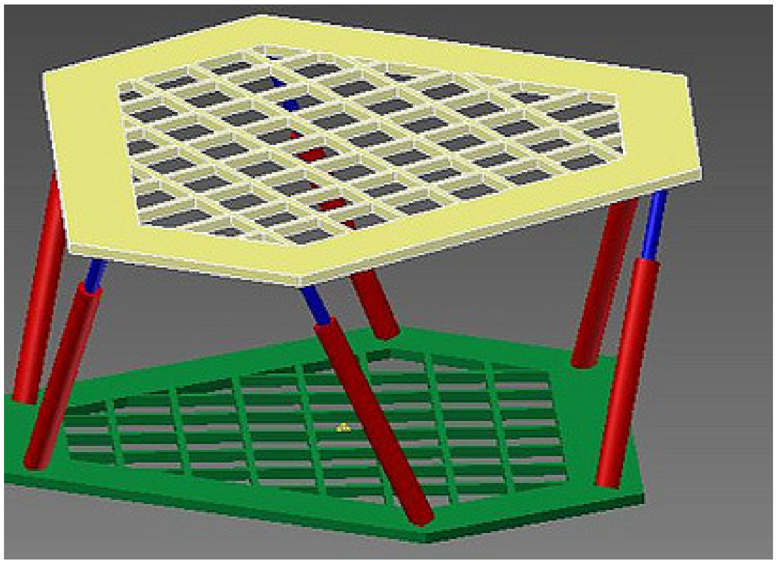


Figure 4.1: 6 DOF Motion Platform

3. **Mechanical Design Improvement**

   There are a few things that needed to be added to the helicopter so that the handling and mounting of circuits and battery would become easier. One needs to design a custom aluminium or fiber glass mount and mount it on the vehicle without affecting the center of gravity. Moreover for higher altitude flights, we need to have a high power motor. Right now we are using 400 watts motor but it is not capable of supporting high collective pitch. A BLDC motor of 750 watts would be better choice in that case.

4. **Six Degrees of Freedom (DoF) Mechanical Platform**

   Though I relied on the strings connected to the helicopter for system identification but it is not the proper way to do that. A 6 DoF motion platform, shown in Figure 4.1, should be used and recommended for system identification. Not only will it give more accurate results but also a safer way to model the helicopter.

A.1 ESC Setup Emulator Code

```
#include <Servo.h>

boolean CmdStatus = false;

Servo myServoObj;

int ServoPin = 50;


#define DelayBtwCmd  10

#define KeepMotorRunning 2000

#define DelayBtwMotorStartNconfigure 1000

#define ReceivedMsgLength 16

#define LineEnd '#'


char messageRecvd[ReceivedMsgLength];

char LstCmd = '\0';


int LowThrottle = 1000;

int MidThrottle = 1500;

int FullThrottle = 2500;


int TempMaxSpeed = 2500;


void setup()

{

  pinMode(ServoPin,OUTPUT);

  myServoObj.attach(ServoPin);

  Serial.begin(9600);

  //Serial.println("Entered SetUp");

  Serial.flush();
```

```
void loop()
{
    // Serial.println("Entered Loop");
     int inputLength = 0;

     do
     {
          while (!Serial.available())
          {
             //myServoObj.write(LowThrottle);
             if (LstCmd == '1')
             {
              myServoObj.writeMicroseconds(LowThrottle);
              Serial.println("LOW Throttle");
              delay(10);
             }
             else if(LstCmd == '2')
             {
               myServoObj.writeMicroseconds(MidThrottle);
               Serial.println("MID Throttle");
               delay(10);
             }
             else if(LstCmd == '3')
             {
              myServoObj.writeMicroseconds(FullThrottle);
              Serial.println("FULL Throttle");
              delay(10);
             }
             else if(LstCmd == '5')
             {
                myServoObj.writeMicroseconds(2500);
                Serial.println("TempMaxSpeed");
                delay(10);
               }
           }; // wait for input
           messageRecvd[inputLength] = Serial.read(); // read it in

      } while (messageRecvd[inputLength] != LineEnd && ++inputLength < ReceivedMsgLength);
     // Serial.println("Exited both while: Msg Read\n");
      messageRecvd[inputLength] = 0; //  add null terminator
      Serial.print(messageRecvd);
      LstCmd = messageRecvd[0];
      HandleCommand(messageRecvd, inputLength);
}

void HandleCommand(char* input, int length)
{
  if (length < 1)
      { return;}
  int value = 0;
  if (length > 1)
      { value = atoi(&input[1]);  }
  int* command = (int*)input;
  switch(*command)
    {
      //SETUP MODE
      case '1':
                  CmdStatus = false;
                  myServoObj.writeMicroseconds(LowThrottle);
                  //Serial.println(" -- Low Throttle Given");
```

```
            delay(10);
            break;
case '2':
            CmdStatus = false;
            myServoObj.writeMicroseconds(MidThrottle);
           // Serial.println(" -- Mid Throttle Given");
            delay(10);
            break;
case '3':
            CmdStatus = false;
            myServoObj.writeMicroseconds(FullThrottle);
           // Serial.println(" -- Full Throttle Given");
            delay(10);
            break;
//Operating MODE
case '4': // Put throttle in LOW position
             CmdStatus = false;
             break;
case '5':
              for(int ind = LowThrottle; ind < TempMaxSpeed; ind++)
              {
                myServoObj.writeMicroseconds(ind);
                Serial.println(ind);
                delay(10);
              }
              Serial.println(" -- END");
              break;
case '6':
              CmdStatus = true;
              Serial.println("CMD stopped");
```

```
                break;
        default:
                break;
  }
}
```

Bibliography

[1] Jategaonkar, Ravindra V. *Flight Vehicle System Identification - A Time Domain Methodology - Progress in Astronautics and Aeronautics, Volume 216.. American Institute of Aeronautics and Astronautics.* (2006)

[2] Vladislav Gavrilets, *Avionics System Development for Small Unmanned Aircraft* (Master's Thesis, MIT, June 1998)

[3] Pieter Abbeel,Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control (August 2008)

[4] *http://www.align-trex-parts.com*

[5] *http://www.basicx.com/Products/robotbook/servo%20intro.pdf*

[6] Casey Reas, Ben Fry *Getting Started with Processing - O'Reilly Media/Make* (June 2010)

[7] Rik Pintelon, Johan Schoukens *Sytem Identification A Frequency Domain Approach – IEEE Press* (2001)

[8] Jared S. Napora *Implementation, Evaluation and Applications of Mobile Mesh Networks for Platforms in Motion* (Master's thesis, University of Maryland College Park, 2009)

[9] Michael Stanley *Implementation of Kalman Filter to Tracking Custom Four-Wheel-Drive Four-Wheel-Steering Robotic Platform* (Master's thesis, University of Maryland College Park, 2010)

[10] *http://www.tech-mp.com/twinn_rexx.htm*

[11] William Premerlani, Paul Bizard Direction Cosine Matrix IMU: Theory (May 2009)

[12] Christian A. Trott*Electronics Design for an Autonomous Helicopter* (Master's thesis, MIT, 1997)

[13] Robert Mahony, Tareq Hamel et. al *Nonlinear Complimentary Filters on the Special Orthogonal Group - IEEE Transactions on Automatic Control Vol.53, No. 5*(June 2008)

[14] Mark Eusten, Paul Coote et. al *A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV - IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008)

[15] *Electronic Speed Controller for Brushless Motor Product Manual- ALIGN*

[16] *http://www.arduino.cc/*

[17] Robert Mahony, Sung-Han Cha et. al *A coupled estimation and control analysis for attitude stabilisation of mini aerial vehicles - Australasian conference on Robotics and Automation* (December 2006)

[18] Siegwart, Roland and Nourbakhsh, Illah R. *Introduction to Autonomous Mobile Robots - MIT Press* (2004)

[19] Grant Baldwin, Robert Mahony et. al *Complementary filter design on the Special Euclidean group SE(3) - European Control Conference* (2007)

[20] Johann Borenstein, Lauro Ojeda et al *Heuristic Reduction of Gyro Drifts for Personal Tracking Systems - Journal of Navigation Vol. 62, No. 1, pp. 41-58* (January 2009)

[21] Graham C. Goodwin, Stephan F. Graebe et al *Control System Design - Prentice Hall PTR* (January 2007)

[22] *http://www.servocity.com/html/hs65mg_mighty_feather.html*

[23] Johan W. Webb, Ronald A. Reis *PID Control of Continuous Processes - Prentice Hall PTR* (September 2006)

[24] *http://www.mathworks.com*

[25] *http://en.wikipedia.org*

[26] Aidan O'Dwyer Handbook of PI and PID Controller Tuning Rules - Imperial College Press (2009)

[27] Doug Cooper et al. Practical Process Control - Proven Methods and Best Practices for Automatic PID Control

[28] *http://www.youtube.com/watch?v=dbMm6LJnVng&feature=related*

[29] *http://code.google.com/p/sf9domahrs/*

[30] *http://wiring.org.co/*

[31] J. Ryan Miller *A 3D Color Terrain Modelling System for Small Autonomous Helicopter* (PhD Dissertation, Carnegie Mellon University, Spring 2002)

[32] Lennart Ljung *SYSTEM IDENTIFICATION: Theory for the User - Prentice Hall PTR* (1987)

[33] A Rahideh, M H Shaheed *Mathematical Dynamic Modelling of a Twin-rotor Multiple Input Multiple Output System - IMechE Journal of Systems and Control Engineering, Vol. 221, No. 1, pp. 89-101* (2007)

[34] *http://www.parallax.com/tabid/768/ProductID/92/Default.aspx*

[35] R A Stuckey *Mathematical Modelling of Helicopter Slung-Load Systems - Technical Report (DSTO)*

[36] Ganesh Rajagopalan, et al *Experimental and Computational Study of Interaction between a Tandem-Rotor Helicopter and a Ship - Conference on Aeromechanics* (2004)

[37] Vladislav Gavrilets *Autonomous Aerobatic Maneuvering of Miniature Helicopters* (PhD Dissertation, MIT, 2003)

[38] J.C. Avila Vilchis, B. Brogliato et al *Nonlinear Modelling and Control of Helicopters - Automatica 39, 1583-1596* (2003)

[39] Joao P. Haspanha *Topics in Undergraduate Control Systems Design, Draft* (April 2010)

[40] *http://www.digi.com/*

[41] Brian L. Stevens, Frank L. Lewis *Aircraft Control and Simulation - Wiley-IEEE* (2003)

[42] *http://www.me.cmu.edu/ctms/modeling/tutorial/systemidentification /mainframes.htm* (online)

[43] Bernard Mettler , Mark B. Tischler,et al. System Identification of Small-Size Unmanned Helicopter Dynamics - American Helicopter Society (1999)