

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/331298873>

Autopilot Design for a Quadcopter

Thesis · October 2017

DOI: 10.13140/RG.2.2.17020.80008

CITATIONS

3

READS

14,650

2 authors:



Amir Hussein

Johns Hopkins University

21 PUBLICATIONS 154 CITATIONS

SEE PROFILE



Rayyan Abdalla

Arizona State University

7 PUBLICATIONS 3 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Autopilot Design for a Quad-copter [View project](#)



ArabicSpeech [View project](#)

AUTOPILOT DESIGN FOR A QUADCOPTER

By

AMIR NASIR BABIKER HUSSEIN

INDEX NO.124029

Supervisor

Prof.Sharief Fadul Babikir

A REPORT SUBMITTED TO

University Of Khartoum

In partial fulfillment for the degree of

**B.Sc. (HONS) Electrical and Electronics Engineering
(ELECTRONICS and COMPUTER ENGINEERING)**

Faculty of Engineering

Department of Electrical and Electronics Engineering

October 2017

DECLARATION OF ORIGINALITY

I declare this report entitled “Autopilot Design for a Quadcopter” is my own work except as cited in references. The report has been not accepted for any degree and it is not being submitted currently in candidature for any degree or other reward.

Signature: _____

Name: _____

Date:_____

DEDICATION

To my parents

ACKNOWLEDGEMENT

I would like first to present enormous gratitude and thanks to my supervisor Professor Sharif F. Babiker for his guidance, motivation and unlimited support.

Sincere thanks go to Ust. Azza Eljaily for her unlimited help, patients and valuable advices and Ust. Dania Alsadiq for her help in flight dynamics and mechanics.

I would also like to thank Aeronautics Research Center for providing test setups and guidance.

Society of Electric and Electronic Engineering as well as Mohammad Abbas Elhaj supplied us with the hardware components we needed for this project, I am grateful to them all.

Finally, I want to express a special appreciation and thanks to the one of the most important persons in my life, my partner and fiancée “Rayyan Salah”. Thank you for your patient and hard working. Thank you for providing unfailing support, continuous encouragement and believing in me. Without you this accomplishment would not have been possible.

ABSTRACT

Autopilot systems play a very crucial role in Unmanned Aerial vehicles (UAVs) control by making the flight safer, easier and more efficient. These systems ensure UAVs' stability, disturbance rejection and trajectory tracking without human intervention. Moreover, they are the base of autonomous flight navigation which has very important applications especially in hazardous environments that are dangerous for humans.

The work in this thesis presents full approach of the concerning design and implementation of control system for six degrees of freedom, four rotors unmanned aerial vehicle known as quadcopter. Firstly, different forces acting on the system and aerodynamic effects are analyzed. Then, detailed mathematical nonlinear model of the quadcopter is formulated based on Newton-Euler equations. Next, the model is developed and tested in Matlab\Simulink environment. After that it is linearized around hovering point. Then, improved PID controller is designed for the linear model. Controllers are designed using Simulink SISO Tool and verified in the nonlinear simulated model. Designed controllers are used in building full autopilot system for translation as well as orientation with significant improvements that are not found in other literatures. The components of the quadcopter are carefully selected according to desired specifications. Then, several experiments are done to verify their performance. The quadcopter is assembled and the designed attitude controller system is implemented using APM 2.7 flight controller. The performance of the controller regarding stability and reference tracking is tested through three degree of freedom experimental configuration. Ground station system is developed through which signals are sent to computer via Bluetooth module which allows real time debugging and analysis of a system. Finally, conclusion is made based on comparison between experimental and theoretical results and thus recommendations for future work are stated.

المستخلص

تقوم أنظمة تحكم الطيران الآلي بدور مصيري في توجيه الطائرات بدون طيار و ذلك لجعل الطيران اسهل و اكثر امانا و كفاءة. هذه الانظمة تضمن إستقرار المركبة, تقليل تأثير الاضطرابات و تتبع المسار المعين دون التدخل البشري. إضافة إلى ذلك هذه الأنظمة تعتبر الاساس للملاحة الجوية و التنقل المستقل و ذات اهمية في التطبيقات المختلفة خصوصا في البيئات الخطرة على حياة الانسان.

هذا البحث يعرض المنهجية الشاملة فيما يخص تصميم و تطبيق نظام تحكم لطائرة ذات أربعة محركات دوارة و ستة درجات حرية تدعى "الكوادكوبتر". أولاً تم تحليل جميع القوى و تأثيرات الديناميكا الهوائية, ثم اشتق النموذج الرياضي اللاخطي اعتمادا على طريقة نيوتن-اويلر. ثانيا اختبر النموذج المشتق باستخدام برنامج الماتلاب\سيمولنك و اشتق النموذج الخطي حول نقطة الاستقرار. بعد ذلك تم تصميم و تحسين نظام تحكم خطي للنموذج باستخدام اداة "السيسو" خاصة ببرنامج السيمولنك و من ثم التحقق من أدائها في النموذج اللاخطي. استخدم متحكم تناسبي, تكاملي و تفاضلي في بناء نظام الطيار الآلي الكامل لأغراض التنقل و التوجيه مع عدد من التحسينات المهمة التي لم تذكر في الابحاث الاخرى. اختيرت أجزاء نظام "الكوادكوبتر" اعتمادا على متطلبات النظام و أجريت العديد من التجارب للتحقق من أدائها. تم تركيب الطائرة و طبق عليها نظام التحكم المصمم مسبقا على لوحة متحكم الطيران. اختبر نظام التحكم فيما يخص الاستقرار و المتابعة بواسطة منصة اختبار ذات ثلاث درجات حرية. تم تطوير محطة تحكم ارضية معنية بارسال بيانات الطيران للكمبيوتر باستخدام البلوتوث و ذلك بدوره يسمح بتحليل و تتبع أداء النظام. أخيرا اجريت مقارنة بين النتائج التجريبية و النظرية للنظام و بناء عليها استخلصت التوصيات فيما يخص العمل المستقبلي

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY	ii
DEDICATION	iii
ACKNOWLEDGEMENT	iv
ABSTRACT.....	v
المستخلص	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES	xi
LIST OF TABLES	xiv
LIST OF ABBREVIATIONS.....	xv
CHAPTER 1	1
INTRODUCTION	1
1.1 History of Quadcopters:	1
1.2 Problem Statement:	1
1.3 Motivation and Objectives:	2
1.4 Outline of The Thesis:	3
CHAPTER 2	4
LITERATURE REVIEW	4
2.1 Theoretical Background:	4
2.2 State of The Art:	7
2.3 Quadcopter Modeling:.....	9
2.3.1 Reference Frames:	9
2.3.2 Quadcopter Kinematics:.....	10
2.3.3 Quadcopter Dynamics:.....	12

2.3.4	Newton Euler Equations:	14
2.3.5	External Forces, Torques and Aerodynamic Effects:	18
2.4	Control Inputs:.....	22
CHAPTER 3		25
SYSTEM SIMULATION AND CONTROL DESIGN.....		25
3.1	System Simulation:	25
3.1.1	Quadcopter Dynamics Simulation:	25
3.1.2	Motor Dynamics Modeling:	26
3.1.3	Motor Mixer:	26
3.2	Open Loop Simulation:	27
3.3	Controller Design:	28
3.3.1	Linearization of Quadcopter:	29
3.3.2	Altitude Control:	32
3.3.3	Attitude and Heading Control:.....	32
3.3.4	Overall System Control:	34
3.4	Simplified PID Controller	35
3.5	Improving PID Controller	36
3.5.1	Integrator Windup	36
3.5.2	Derivative Kick.....	37
3.6	PID Controller Implementation in Simulink.....	37
3.6.1	Linearization	37
3.6.2	Altitude PID Controller Design:	38
3.6.3	Attitude Controller Design.....	40
3.6.4	Position Controller	43

CHAPTER 4	45
HARDWARE AND DESIGN IMPLEMENTATION	45
4.1 Components Specifications:	45
4.1.1 AutoPilot Board:	45
4.1.2 DJI F450 Frame:	46
4.1.3 Battery:	47
4.1.4 RC Transmitter and Receiver:	48
4.1.5 Motors:	48
4.1.6 Propellers:	49
4.1.7 Electronic Speed Controllers (ESC):	49
4.1.8 Bluetooth module:	50
4.1.9 Connection Setup:	50
4.1.10 Power Regulator:	51
4.2 Installation and Components Placement:	51
4.3 Autopilot Implementation:	54
4.3.1 Setting up the Local IDE:	54
4.3.2 Developing the Autopilot:	55
4.3.3 Reading RC Inputs:	56
4.3.4 Reading the IMU and AHRS:	56
4.3.5 Sending Motor Commands:	57
4.3.6 Implementing The Control System:	57
4.3.7 PID Implementation	57
4.3.8 Putting The Code All Together:	58
CHAPTER 5	59

RESULTS DISCUSSION.....	59
5.1 Simulation Results:	59
5.2 Controller Implementation Results:	60
CHAPTER 6	64
CONCLUSION.....	64
REFERENCES	66
Appendix A:.....	68
Appendix B:	70
Appendix C	75
Appendix D:.....	84
Appendix E:	89
Appendix F:	95

LIST OF FIGURES

Figure 2.1: Structure of Quadcopter.	5
Figure 2.2: Quadcopter Euler Angles.	5
Figure 2.3: Quadcopter +-Configuration and X-Configuration.	6
Figure 2.4: The Quadcopter Reference Frames.	10
Figure 2.5: Logical Diagram of Quadcopter Vectors Calculation.	13
Figure 2.6: Logical Diagram of Quadcopter Mathematical Model	14
Figure 2.7: Quadcopter Thrust, Moments and Torques.	19
Figure 3.1: Quadcopter Dynamics	25
Figure 3.2: BLCD Motor Model	26
Figure 3.3 : Simulink Model of a Motor Mixer	27
Figure 3.4: Open loop Quadcopter System.	27
Figure 3.5 : Simplified Comprehensive System Control	29
Figure 3.6 : Altitude Controller	32
Figure 3.7 : Attitude and Heading Controller	33
Figure 3.8 : Simulink Model for Translating Position (x,y) to Pitch and Roll Angles	34
Figure 3.9 : Position and Heading Controller	35
Figure 3.10: Simplified PID Controller	36
Figure 3.11: Improved PID Controller.	37
Figure 3.12: Linearized System with Altitude Controller	38
Figure 3.13: Linear Altitude Velocity and Position Response	39
Figure 3.14 : Altitude Controller Implemented in Nonlinear System	39
Figure 3.15 : Nonlinear Altitude Velocity and Position Response	40
Figure 3.16: Pitch, Roll Linearized System Controller.	41

Figure 3.17 :Linearized System Roll Angular Velocity and Angle Response	41
Figure 3.18 :Roll Angle Controller Implemented in Nonlinear system.....	41
Figure 3.19: Nonlinear System Roll Angular Velocity and Angle Response.....	42
Figure 3.20: Yaw linearized System Controller	42
Figure 3.21 Linearized System Yaw Angular Velocity and Angle Response	42
Figure 3.22: Yaw Angle Controller Implemented in Nonlinear System	43
Figure 3.23: Nonlinear System Yaw Angular Velocity and Angle Response.....	43
Figure 3.24 :X Coordinate Controller.....	43
Figure 3.25 : Y Position Tracking Response	44
Figure 3.26 : X Position Tracking Response	44
Figure 3.27 :Quadcopter Full Autopilot Model	44
Figure 4.1: APM Flight Controller	46
Figure 4.2: DJI F450 Quadcopter Frame	47
Figure 4.3: LiPo Battery	47
Figure 4.4: RC Transmitter and Receiver	48
Figure 4.5: D3542/6 BLDC Motor	49
Figure 4.6: GWS 10 x 4.7 Propellers.....	49
Figure 4.7: Afro 30A ESC	50
Figure 4.8:JY-MCU Bluetooth Module.....	51
Figure 4.9: APM insulation layer.....	52
Figure 4.10:APM Input and Output Connections	52
Figure 4.11: Battery placement.....	53
Figure 4.12: ESC Mounting.....	53
Figure 4.13: ArduPilot IDE Setup.	55

Figure 4.14: Quadcopter + Configuration.....	56
Figure 5.1: Test Setup for Attitude Control.....	60
Figure 5.2 :Barometer Zero Altitude Readings.....	61
Figure 5.3 :Pitch Angle Tracking.....	61
Figure 5.4 :Roll Angle Tracking.....	62
Figure 5.5 : Yaw Angle Tracking	62
Figure A.1: Transformation Frames	68
Figure B.1: Verification of Inertia Analytic Results Using Inertia Calculator	73
Figure D.1: Verification of Inertia Analytic Results Using Inertia Calculator	86
Figure D.2: RC Calibration.....	88
Figure E.1: Motor Test Arduino Code.....	90

LIST OF TABLES

Table 5.1: PID Controller Results.....	59
Table 5.2: PID Controller Implemented on Real System	63
Table B.1:Quadcopter Parameters	73

LIST OF ABBREVIATIONS

<i>A/D</i>	Analog to Digital
<i>AHRS</i>	Attitude and Heading Reference System
<i>API</i>	Application Programming Interface
<i>APM</i>	Ardu Pilot Mega
<i>BLDC</i>	Brushless Direct Current
<i>CoG</i>	Center of Gravity
<i>DC</i>	Direct Current
<i>DIY</i>	Do It Yourself
<i>DOF</i>	Degree Of Freedom
<i>ESC</i>	Electronic Speed Controller
<i>GCS</i>	Ground Control Station
<i>GPS</i>	Global Position System
<i>HAL</i>	Hardware Abstraction Layer
<i>IDE</i>	Integrated development Environment
<i>IMU</i>	Inertial Measurement Unit
<i>LIPO</i>	Lithium Polymer
<i>LQR</i>	Linear Quadratic Regulator
<i>MEMS</i>	Electro-Mechanical-Systems
<i>PID</i>	Proportional ,Integral, Derivative
<i>PWM</i>	Pulse Width Modulation
<i>RC</i>	Radio Controlled

<i>RPM</i>	Rotation Per Minute
<i>RX</i>	Receiver
<i>SISO</i>	Single Input Single Output
<i>SPP</i>	Serial Port Profile
<i>TTL</i>	Transistor Transistor Logic
<i>TX</i>	Transmitter
<i>UAV</i>	Unmanned Aerial Vehicle
<i>USB</i>	Universal Serial Bus
<i>VTOL</i>	Vertical Takeoff and Landing

LIST OF SYMBOLS

C_D	Drag Force Coefficient
C_T	Thrust Coefficient
C_H	Hub Torque Coefficient
$\overrightarrow{V^b}$	Body Frame Vector
$\overrightarrow{V^i}$	Inertial Frame Vector
Ω	Speed of Rotor
\vec{F}	Net Force
F_d	The Drag Force
F_g	Gravitational Force
H_i	Hub Torque
K_D	Derivative Gain
K_H	Hub Torque Proportionality Constant
K_I	Integral Gain
K_P	Proportional Gain
K_T	Thrust Proportionality Constant
\vec{L}	Angular Momentum
\vec{p}	Linear Momentum
\vec{r}	Position Vector
ρ	Air Density
S	Orthogonal Projection of Surface

K_d	Drag Force Proportionality Constant
$\vec{\tau}$	Net Torque
g	Gravitational Acceleration
J_{xx}, J_{yy}, J_{zz}	Inertial Components
l	Length of Quadcopter Arm
l_2	ESC Distance from Center of the Quadcopter
m	Mass
p	Body Frame x Angular Rate
q	Body Frame y Angular Rate
r	Body Frame z Angular Rate
R	Rotor Radius
R_b^i	Rotation Matrix from Body Frame to Inertial Frame
R_i^b	Rotation Matrix from Inertial Frame to Body Frame
T	Thrust Vector
u	Body Frame x Axis Velocity
u_1	Total Thrust
u_2	Roll Moment
u_3	Pitch Moment
u_4	Yaw Torque
v	Body Frame y Axis Velocity
x	Earth Frame x Axis Position
y	Earth Frame y Axis Position
w	Body Frame z Axis Velocity
z	Earth Frame z Axis Position

θ	Euler Pitch Angle
φ	Euler Roll Angle
ψ	Euler Yaw Angle
ω	Angular Velocity in The Body Frame

CHAPTER 1

INTRODUCTION

This chapter illustrates a general overview on scope and the aim behind writing this thesis. A brief quadcopter historical background, problem statement, motivation and objectives are introduced and then the layout of thesis is presented.

1.1 History of Quadcopters:

Louis Breguet built first rotary wing aircraft "Gyroplane n: 01" in 1907 [1]. It had very limited flight capabilities and was controlled by people on ground for stabilization. Later, Lawrence and Sperry (USA) introduced UAV in 1916 that was able to fly for a distance of 30 miles. In 1920, Etienne Oehmichen suggested a new design with four rotors and eight propellers all driven by a single engine. During that time, the aircraft exhibited significant increase in control stability and accuracy. Years later, the concept of Vertical Take-off and landing (VTOL) has become very popular research area and thus the study of quadcopter structure has evolved. This concept was firstly applied on VZ-7 multirotor aircraft designed by Curtiss-Wright Company for the US army in 1958. After that much improved designs with better performance were introduced. Nowadays, open source systems are available for people who are interested in developing their own quadcopters with light and inexpensive electronic materials.

1.2 Problem Statement:

Quadcopters have many advantages such as a high maneuverability, cheap price and simple construction. However, several problems must be solved for use in complex applications. The major problem is the control of quadcopter, since it is a nonlinear and underactuated system with highly unstable nature and control difficulties. Conventional motion theories are not enough to describe the flight behavior of quadcopter vehicle as it operates in different reference frames. The aerodynamic effects exerted on the quadcopter require to be considered seriously. In order to design autopilot controller properly many environmental uncertainties must be considered. Appropriate control algorithm must be selected according to the ability of stabilizing the system, reject the disturbance as well as accurately track the commanded signals in the area of application. Another important autopilot design consideration is the proper selection of the

system hardware. Components employed in the hardware implementation must provide the necessary compatibility and reliability. The selected flight controller board must allow the implementation of designed control algorithm and sensors integration. Some of the modern flight boards have integrated built in IMU. The work in this thesis takes into account all previously stated problems.

1.3 Motivation and Objectives:

This thesis work concentrates on modeling, simulation, controller design and implementation of quadcopter autopilot. In the last decade, advances in technology specially Micro and Nano-Electro-Mechanical-Systems MEMS and microprocessors made it possible to build reliable, cheap and efficient multirotor systems. In addition, multirotor systems exhibit greater maneuverability and hovering ability and solve the problems that helicopter pilots had with making vertical flights. As a result quadcopter configuration became popular for small unmanned aerial vehicles and have been used in different applications. However, these applications are highly dependent on designing an autopilot system that is extremely challenging research field. The practical solutions provided encounter some limitations that need to be eliminated. Moreover the study of quadcopter model allows the deep understanding and application of many control engineering theoretical principles.

The main objectives of this thesis can be summarized as follows:

- Deriving an accurate mathematical model of the quadcopter using classical mechanics, aerodynamics principles and different system identification techniques.
- Building the simulation based on the mathematical model with MATLAB-Simulink environment and performs system verification.
- Obtaining the Linearized model of the system around hovering.
- Designing PID controller for the linearized model and examining the system response with the controller.
- Performing the necessary improvements to the designed PID controller.
- Obtaining the appropriate quadcopter components and assembling the quadcopter hardware platform.

- Implementing the designed autopilot system in the physical system and perform experimental setup to examine its response.
- Comparing between the quadcopter simulation and real system response and thus observing limitations and recommending improvements.

1.4 Outline of The Thesis:

The contents of the next chapters are summarized as follows:

Chapter 2: Firstly, a brief theoretical background about the quadcopter concept is presented. Secondly, previous work of the literature is introduced with some analysis. Lastly, the quadcopter model is derived and its dynamics and kinematics are discussed in details.

Chapter 3: In this chapter, the quadcopter model is built using Matlab-Simulink environment. The open loop testing is performed for verification. Then, the system is linearized around hovering. The linear PID controller is designed and implemented on the nonlinear model. The system response is examined and after making some improvements to the PID controller.

Chapter 4: This chapter introduces technical guidance about the hardware and software implementation of the quadcopter. The system components selection, installation steps, details regarding building the autopilot code and PID implementation.

Chapter 5: This chapter discusses the system simulation results while achieving comparison between linearized and nonlinear system. Furthermore, the real system is tested using experimental testbench and its response is compared to the simulated response.

Chapter 6: This chapter gives a summary of the work done in the previous chapter as well as explains the system limitations and recommends some future work.

CHAPTER 2

LITERATURE REVIEW

This chapter introduces a theoretical background of quadcopter concept, physics of flight, structure and design configurations. Then numerous papers and theses dealing with the problem of modeling, controlling and implementing the quadcopter are discussed briefly. Finally the quadcopter mathematical model is shown with the details of reference frames, quadcopter kinematics, dynamics and aerodynamic effects.

2.1 Theoretical Background:

The Quadcopter is UAV with four identical rotors. The quadcopter changes the rotational speed of these rotors to control the thrust and torque which achieve the desired motion. The main advantage of quadcopter over conventional helicopter is that quadcopter propellers are relatively smaller than helicopter propellers. This enables the quadcopter to fly with less risk and more stability in challenging environments.

The main components of a quadcopter are frame and rotors. The frame carries controller, sensor, power source and other advanced gadgets. Rotors are located on the tip of the frame arm. The structure of the quadcopter is cross-like structure. Each rotor consists of a propeller fitted to a separately powered DC motor. **Figure 2.1** shows the structure of quadcopter. Propellers 1 and 3 rotate in the same direction while propellers 2 and 4 rotate in an opposite direction. This rotation style leads to balancing the total system torque and cancelling the aerodynamics torques in stationary flights [3] [2].

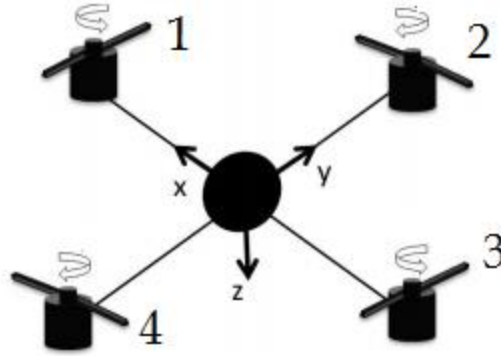


Figure 2.1: Structure of Quadcopter.

The quadcopter has 6 degree of freedom. This means that 6 variables are needed to express its position and orientation in space (x , y , z , ϕ , θ and ψ). The x , y and z variables represent the distances of quadcopter's center of mass along the x , y and z axes respectively from fixed reference frame. The other three variables are the three Euler angles which represent the quadcopter orientation. (ϕ) is the angle about the x axis and is called roll angle, (θ) is the angle about y axis and is called pitch angle and (ψ) is the angle about z axis and is called yaw angle.

Figure 2.2 explains Euler angles of a quadcopter. The roll and pitch angles are referred to as the attitudes of the quadcopter while the yaw angle is called the heading of the quadcopter. The quadcopter distance from the ground is called the quadcopter altitude.

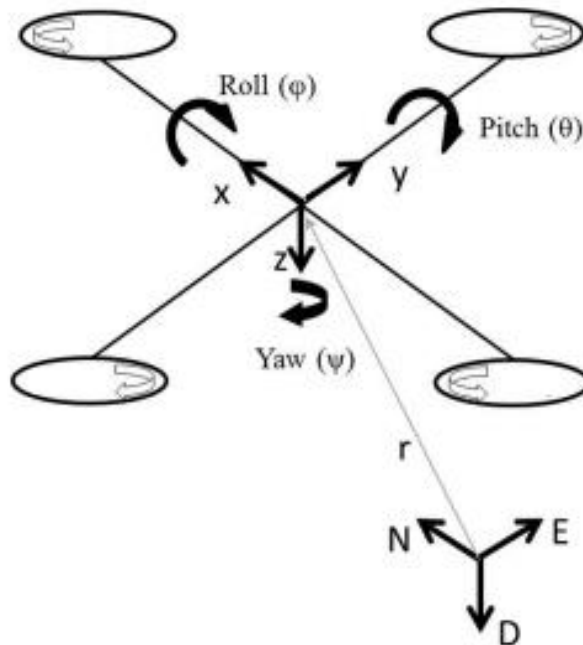


Figure 2.2: Quadcopter Euler Angles.

The quadcopter is underactuated system, where four inputs are used to control six degrees of freedom. A quadcopter with four rotors has two design configurations, X-configurations and +-configurations. The thrust and yaw dynamics are similar for both, however, the roll and pitch dynamics are different. In other words, in +-configurations, only two rotors generate the roll and pitch moment while the four motors generate these moments in X-configurations. **Figure 2.3** shows X- and +-configurations. Note the direction of axes corresponding to each configuration.

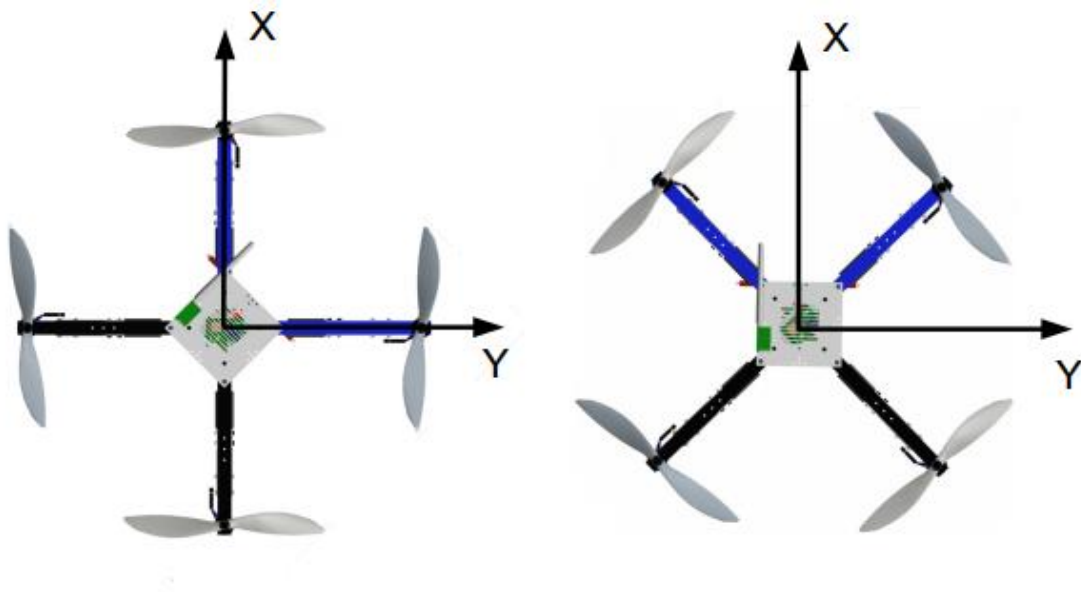


Figure 2.3: Quadcopter +-Configuration and X-Configuration.

For both configurations, vertical upward motion is generated by increasing the rotors speed whereas vertical downward motion is generated by decreasing rotors speed. For +-configuration, pitch rotation is realized by varying the speed of the front and back rotors while roll rotation is realized by varying the speed of left and right rotors. For X-configurations, the speed of four rotors is varied according to the desired rotation. This thesis adopts the +-configuration under study.

2.2 State of The Art:

During the recent period, studies and surveys regarding quadcopter modeling, design, control and implementation have become of considerable interest to researchers. This is properly due to numerous advantages and simple theoretical of the quadcopter. This section introduces the literature of the work on quadcopter along with the evolution of its technology.

Attention is drawn to the fact that multicopters are more advantageous compared to classic UAVs due to vertical takeoff and landing approach (VTOL) [4] [6] [5] [8].

The quadcopter modeling requires the use of Newton-Euler or Euler-Lagrange formalisms, these formalisms provides assumptions about the quadcopter structure and operation which simplifies the derivation of quadcopter kinematics and dynamics. Newton-Euler equations are the most common method adopted due to its realistic assumptions [6] [4] [7] [9].

Deriving quadcopter kinematic equations requires the definition of reference frames. Mehmet Ikulak [6], Ruth Tesfaye [5] and Alexander Lebedev [7] employ the fact that quadcopter motion is described according to two frames, body frame and inertial frame [4] [10]. However, others researchers like Wei Zhong Fum [9] assume the need for additional frame corresponding to quadcopter motion which is the vehicle frame.

The quadcopter dynamics equations are derived by considering the net external forces, torques and aerodynamic effects affecting the quadcopter motion in free space. The external forces and torques leading the quadcopter are normally conventional while aerodynamic effects vary according to certain applications and different environments. Researchers have different perspective regarding considering aerodynamic effects, most of them just neglect those effects [9] [11] [5] [10]. However, Alexander Lebedev assumed that the drag force effect is the most considerable aerodynamic effect and has negative drawbacks when neglected while S. Bouabdallah [20] takes gyroscopic effect into account.

Modeling the quadcopter requires paying attention to the design configuration. Researchers have already studied the dynamics of both +- or X-configuration. S. Bouabdallah and Alexander Lebedev who employ +-configuration assume that this design configuration is simpler and easier to explain the quadcopter flight mechanisms [4] [7] [5]. However, real applications favor the use of X-configuration due to intensive stability [6].

Quadcopter autopilot design is the dominant of all research fields. The main objective is to find the best controlling techniques and thus achieving the maximum possible stability and performance improvements. Wei Zhong Fum, Mehmet Ikulak and Wil Selby [12] have assumed that the quadcopter is nonlinear system that can be linearized around hovering conditions and thus linear control mechanisms can simply be employed [4] [9] [7] [12] [13]. However, S. Bouabdallah, Ruth Tesfaye and Tommaso Bresciani have designed controllers that apply to the nonlinearity behavior of quadcopter [4] [5] [6] [14].

The simplest controller design approach is the conventional PID controller. This controller is adopted by many researchers due to simple tuning, acceptable stability and easier code implementation. The applied PID controller can be simple PID [7] [12] [13] or PID with gain scheduling [4]. This controller can be implemented to quadcopter attitude, altitude and position in the same approach with the ability to nest the control loops for further stability [3] [13] [6] [12]. However, PID controller has a reliable performance only in the linear region of quadcopter operation and thus the employment of nonlinear controller techniques is highly advisable for real applications [20] [14] [4] [11]. Many modern researches have introduced more advanced controller techniques than the conventional PID, Linear quadratic regulator (LQR) and Sliding mode technique have become of interest due to their further improvements to system stability [6] [14] [5] [4].

Implementation of the quadcopter in real physical system requires paying attention to the flight controller through which control setting will be applied. The most basic approach is the use of simple microcontroller and relying on the code base to apply controller setting. Many researchers have made use of Arduino platform to implement the quadcopter [15] [16]. However, implementation using Arduino requires the integration of quadcopter sensor modules and which have no yet reliable Arduino libraries to apply. Recently, researchers tend to implement their controllers using compact flight controllers with built-in sensors modules. These flight controllers provide reliable solution since they are designed with the consideration of delay introduced and provision of abstract libraries. The most common flight controllers adopted for research applications are those provided by Ardupilot organization [17]. These flight controllers come with software that provides a wizard through which the user can specify the system properties. They are open-sourced and code base approach can be used instead. APM flight

controller, for example, allows the implementation of custom PID controllers using abstract libraries defined by some developers. That's why most researchers implementations are based on APM flight controller [6] [12] [11]. Ardupilot organization provides more advanced controllers with much improved features along with full quadcopter kits employing those flight controllers, as example, Iris quadcopter kit which employs Pixhawk flight controller is generally utilized for some research applications [9]. Other alternatives of flight controller also exist. K.K.2 flight controller is a simple and preferable alternative for light weight applications [18].

Discussion of all previous surveys has lead to decision that modeling of the quadcopter is easier done with Newton-Euler formalism. +-configuration provides a better understandable mean. The control of the quadcopter can be done utilizing PID controller and response is visualized using Matlab and Simulink environment. The implementation of controller is done using APM flight control and through code base approach.

2.3 Quadcopter Modeling:

This section presents the modeling of the quadcopter system. First of all, the reference frames that formulate the coordinates of the quadcopter are discussed. Secondly, quadcopter Kinematics and dynamics equation are derived based on Newton-Euler formalism. Then, the state equations are written for the purpose of control design and control inputs are determined.

2.3.1 Reference Frames:

There are two reference frames essential to describe the behavior of a quadcopter, the body frame and inertial frame. The reason why two frames are needed is that the quadcopter has many sensors, like gyroscope and accelerometer, that give readings with respect to body frame whereas it has other sensors, like GPS and magnetometer, that give readings with respect to inertial frame. Therefore, a mean of transformation is required to derive system equations according to a single frame.

The body frame is associated with the quadcopter and its origin corresponds to the center of gravity (CoG) of the vehicle. Since the quadcopter has a symmetrical shape with a central core and four identical rotors attached at its arms, the quadcopter CoG is located at the core center.

The inertial frame, on the other hand, is generally represented by North-East-Down coordinates system. The origin of the inertial frame is chosen to be a fixed point on Earth, generally the landing point is selected as the origin [19]. **Figure 2.4** shows the quadcopter reference frames.

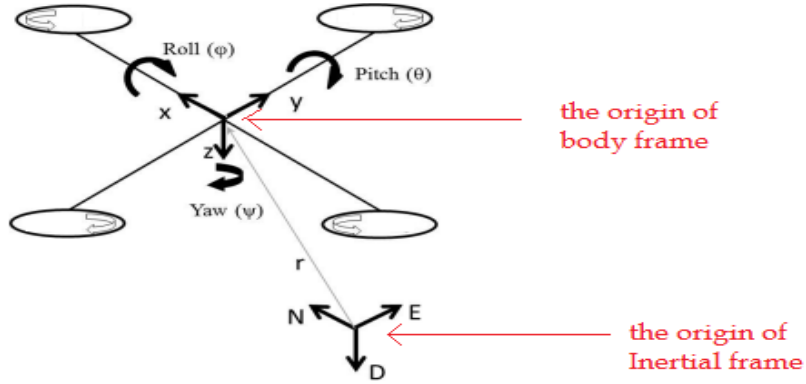


Figure 2.4: The Quadcopter Reference Frames.

2.3.2 Quadcopter Kinematics:

Kinematics is the subject that deals with motion of objects without taking into account the forces and moments acting on these objects. Transformations between coordinate systems are a function of kinematics.

In order to derive quadcopter dynamics equations, the quadcopter states must be introduced as well as means of transformation between their different coordinate frames. These states express the motion of the quadcopter in free space. This motion can be translational; expressed by linear position and velocity, or rotational; expressed by the angular rates and Euler angles of the quadcopter.

- Quadcopter position in the earth frame is expressed by the vector $[r_x \ r_y \ r_z]^T$.
- Linear velocity in the body frame is expressed by the vector $[u \ v \ w]^T$.
- Angular rates in the body frame are expressed by the vector $[p \ q \ r]^T$.
- Euler angles, which are also called roll, pitch and yaw angles are expressed as $[\phi \ \theta \ \psi]^T$

To obtain the representation of a body frame vector \vec{V}^b in the inertial frame \vec{V}^I , rotation matrix R_b^I is used as follows:

$$\vec{V}^i = R_b^i \vec{V}^b \quad (1)$$

An inverse transformation of inertial frame vector to body frame vector can be obtained by multiplying with R_i^b which is the inverse of R_b^i . The inverse of the rotation matrix R_b^i is found to be its transpose. The properties and details of rotation matrix derivation is clearly shown in appendix A.

$$\vec{V}^b = R_i^b \vec{V}^i = (R_b^i)^{-1} \vec{V}^i = (R_b^i)^T \vec{V}^i \quad (2)$$

$$R_b^i = \begin{bmatrix} c(\theta)c(\psi) & s(\varphi)s(\theta)c(\psi) - c(\varphi)s(\psi) & c(\varphi)s(\theta)c(\psi) + s(\varphi)s(\psi) \\ c(\theta)s(\psi) & s(\varphi)s(\theta)s(\psi) + c(\varphi)c(\psi) & c(\varphi)s(\theta)s(\psi) - s(\varphi)c(\psi) \\ -s(\theta) & s(\varphi)c(\theta) & c(\varphi)c(\theta) \end{bmatrix} \quad (3)$$

Where **s** and **c** represent sin() and cos() respectively.

Thus, the relationship between the linear velocity of the body frame and the linear position in the inertial frame is expressed by the following equation:

$$\frac{d}{dt} \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = R_b^i \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (4)$$

The Euler angles $[\varphi \ \theta \ \psi]^T$ are expressed in different frames. When an object rotates, the body frame rotates about the X axis by amount of φ (roll) angle. Then the resulting position of the object rotates about the Y axis by an angle θ and finally the resulting position rotates about the Z axis by an angle ψ . Each angle is expressed according to a certain position during the rotation. Thus the procedure of transforming angular rate vector $[p \ q \ r]^T$ of the body frame to Euler

angles vector $[\varphi \ \theta \ \psi]^T$ is a bit complex [19]. The transfer matrix, \mathbf{T} , is used instead of Rotation matrix for this transformation purpose.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = T^{-1} \frac{d}{dt} \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix} = T^{-1} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (5)$$

Where T^{-1} matrix transforms Euler angles rates into the angular rates of the body frame and its inverse matrix is \mathbf{T} . The details of deriving Transfer matrix \mathbf{T} is shown in appendix A.

$$T = \begin{bmatrix} 1 & \sin(\varphi) \tan(\theta) & \cos(\varphi) \tan(\theta) \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) \sec(\theta) & \cos(\varphi) \sec(\theta) \end{bmatrix} \quad (6)$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\varphi) & \sin(\varphi) \cos(\theta) \\ 0 & \sin(\varphi) & \cos(\varphi) \cos(\theta) \end{bmatrix} \quad (7)$$

Once transformation matrices are obtained, kinematic equations can be expressed as follows:

$$\frac{d}{dt} \begin{bmatrix} r_x \\ r_y \\ r_z \\ \varphi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} R_b^i & 0_{3 \times 3} \\ 0_{3 \times 3} & T \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \quad (8)$$

2.3.3 Quadcopter Dynamics:

Dynamic equations express the body physical motion as a function of time while taking all forces and torques into account. These dynamic equations are found using Newton-Euler method. This method states the following assumptions [23]:

- The quadcopter structure is assumed to be rigid.
- The quadcopter structure is assumed to be symmetrical.
- The quadcopter center of mass and the body frame origin are assumed to coincide with each other.
- The propellers are assumed to be rigid.
- The thrust force and hub moment are proportional to square propellers speed.
- The inertial matrix is time invariant.

As mentioned earlier, the quadcopter has four vectors; each has three components (12 states) to describe its motion. Description of quadcopter motion with respect to inertial frame requires the two vectors, the position vector and the orientation vector. Calculation of both position vector and orientation vector requires prior knowledge of all initial conditions of these vectors. Additionally, net force \vec{F} and net torque $\vec{\tau}$ that are acting on the quadcopter must be known.

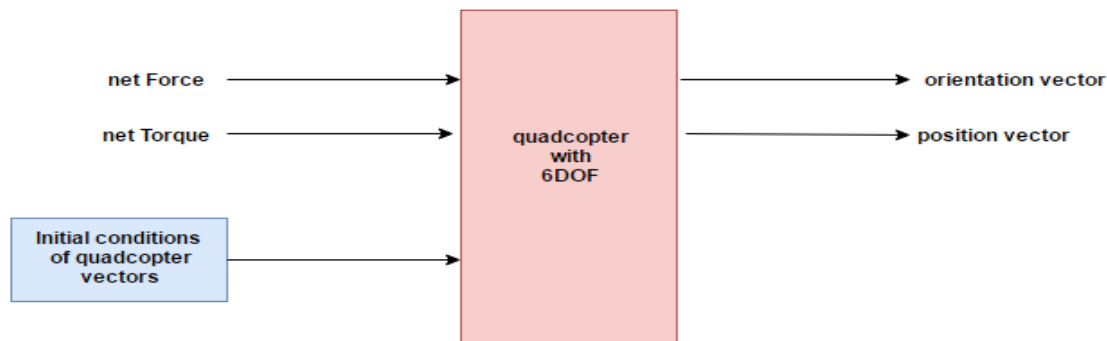


Figure 2.5: Logical Diagram of Quadcopter Vectors Calculation.

The main sources of net force are gravitational force, considerable aerodynamic effects and thrust force. Thrust force is generated by propellers rotation while the main sources of net torque are the torques and moments generated by propellers rotation. The autopilot system controls the speed of the rotors through speed controllers which generate the essential force and torque to drive the quadcopter. **Figure 2.5** shows the logical diagram of quadcopter mathematical model.

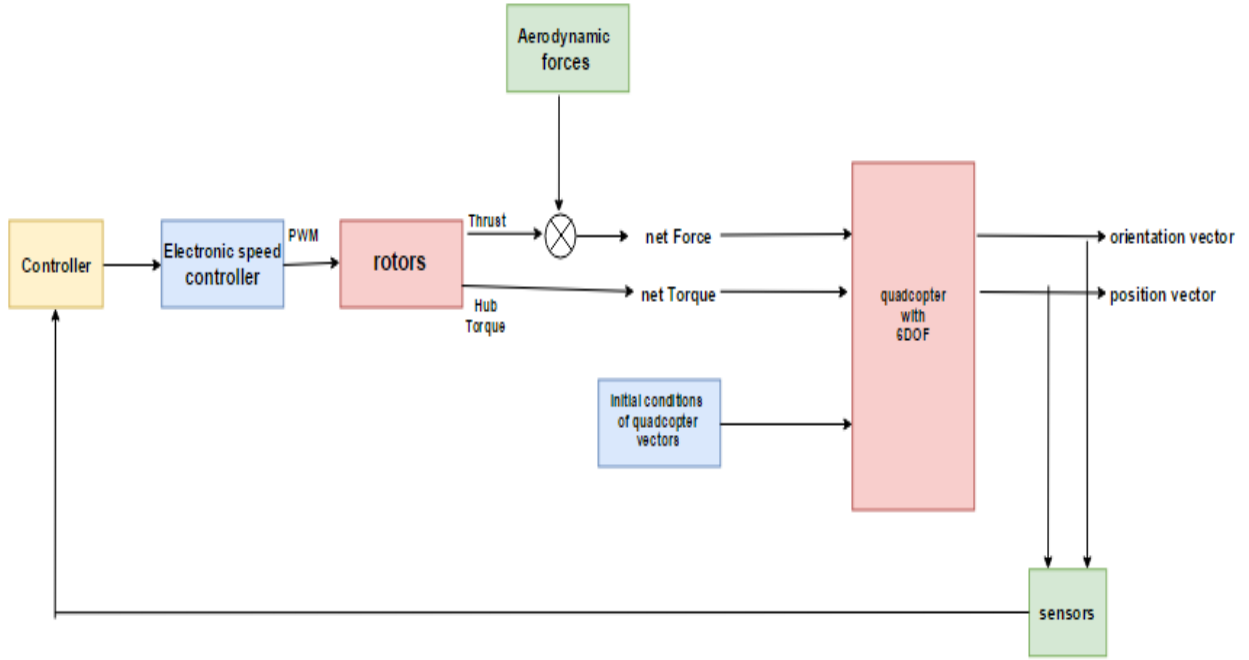


Figure 2.6: Logical Diagram of Quadcopter Mathematical Model

2.3.4 Newton Euler Equations:

A translational motion of a body can be described by the following equation:

$$\vec{F} = \frac{d\vec{p}}{dt} \quad (9)$$

Where \vec{p} is the linear momentum of the body. The translational motion is described with respect to a fixed reference with an origin. Thus the location of the body according to the reference origin is described by the vector \vec{r} . Given that the body mass is m , the linear momentum is \vec{p} found by:

$$\vec{p} = m * \frac{d\vec{r}}{dt} \quad (10)$$

Then, the net force equation becomes:

$$\vec{F} = m * \frac{d^2\vec{r}}{dt^2} \quad (11)$$

$$\frac{d^2\vec{r}}{dt^2} = \frac{\vec{F}}{m} = \vec{\ddot{r}} \quad (12)$$

Separating into position vector components:

$$\frac{\vec{F}_x}{m} = \ddot{r}_x, \quad \frac{\vec{F}_y}{m} = \ddot{r}_y, \quad \frac{\vec{F}_z}{m} = \ddot{r}_z \quad (13)$$

These net forces are calculated with respect to inertial frame.

An Angular motion of a body around a fixed point can be described as:

$$\vec{\tau} = \frac{d\vec{L}}{dt} \quad (14)$$

Where \vec{L} is the angular momentum of the body. For a rotating body with a collection of equal masses and each mass dm is located at distance ρ from the origin of the reference frame, the angular momentum can be found as follows:

$$\vec{L} = \iiint \rho \times \overrightarrow{v_{dm}} dm \quad (15)$$

$\overrightarrow{v_{dm}}$ Represents the linear velocity of single point of mass dm .

$$\overrightarrow{v_{dm}} = \vec{\omega} \times \vec{\rho} \quad (16)$$

Where $\vec{\omega}$ represents the angular velocity of the body. Substituting equation (16) into equation (15) results into the following relationship:

$$\vec{L} = J * \vec{\omega} \quad (17)$$

J represents the inertia matrix. J is a scalar matrix that expresses the inertia of the body along all axes and different rotational motions.

$$J = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{bmatrix} \quad (18)$$

The first index of J elements corresponds to the index of \vec{L} and the second index corresponds to index of $\vec{\omega}$. Since $\vec{\omega}$ and \vec{L} cannot be of different direction, the off-diagonal elements of J matrix reduce to zero [21]. And the relationship between \vec{L}, J and $\vec{\omega}$ is expressed by (19). The details of deriving J matrix is found in appendix B.

$$\vec{L} = \begin{bmatrix} L_{xb} \\ L_{yb} \\ L_{zb} \end{bmatrix} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} * \begin{bmatrix} p \\ q \\ r \end{bmatrix} = J \times \vec{\omega} \quad (19)$$

The rotation of a vector on a reference frame while this reference frame is also rotating about another fixed frame introduces additional component to the derivative of this vector. This additional component depends mainly on the angular velocity of the rotating reference frame [23]. This effect is known as Coriolis Effect and it leads to the so-called Euler equation [22]:

$$\vec{\tau} = \frac{d\vec{L}}{dt} = \frac{d(J * \vec{\omega})}{dt} + \vec{\omega} \times (J * \vec{\omega}) \quad (20)$$

The net torque is expressed in body frame. The above equation can be detailed as:

$$\vec{\tau} = \begin{bmatrix} \tau_{xb} \\ \tau_{yb} \\ \tau_{zb} \end{bmatrix} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} * \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} * \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (21)$$

In a shorter form:

$$\begin{aligned} \tau_{xb} &= J_{xx} * \dot{p} + (J_{zz} - J_{yy}) * qr \\ \tau_{yb} &= J_{yy} * \dot{q} + (J_{xx} - J_{zz}) * pr \\ \tau_{zb} &= J_{zz} * \dot{r} + (J_{yy} - J_{xx}) * pq \end{aligned} \quad (22)$$

In order to find the change of angular velocity:

$$\begin{aligned} \dot{p} &= \frac{\tau_{xb} - (J_{zz} - J_{yy}) * qr}{J_{xx}} \\ \dot{q} &= \frac{\tau_{yb} - (J_{xx} - J_{zz}) * pr}{J_{yy}} \\ \dot{r} &= \frac{\tau_{zb} - (J_{yy} - J_{xx}) * pq}{J_{zz}} \end{aligned} \quad (23)$$

Equation (6) can be used to obtain the Euler angular rate vector which gives the quadcopter orientation.

2.3.5 External Forces, Torques and Aerodynamic Effects:

- 1- The gravitational force: this force is applied to the CoG of the quadcopter. It is directed toward the center of earth and is written as:

$$F_g = m * \vec{g} \quad (24)$$

F_g is the gravitational force, m is the quadcopter mass and \vec{g} is the gravitational acceleration.

- 2- Forces of the propellers: this is the controllable force of the quadcopter since it is directly dependent on the speed of the rotors. The thrust force of a single rotor is proportional to the square speed of the rotor.

$$T_i = K_T * \Omega_i^2 \quad (25)$$

Where T_i is the thrust of the rotor i . Ω_i is the angular speed of the rotor and K_T is the thrust proportionality constant [24]. The formula of finding K_T is shown in appendix B. The subscript i corresponds to the index of the rotor.

- 3- Hub Torque of the propellers: This is the torque responsible for yaw rotation of the quadcopter. It is dependent on the square angular speed of the quadcopter. For a single rotor, the hub torque can be found as follows:

$$H_i = K_H * \Omega_i^2 \quad (26)$$

Where H_i is the hub torque of the motor i . Ω_i is the angular speed of the rotor and K_H is the hub torque proportionality constant [24]. The formula of finding K_H is shown in appendix B. The subscript i corresponds to the index of the rotor.

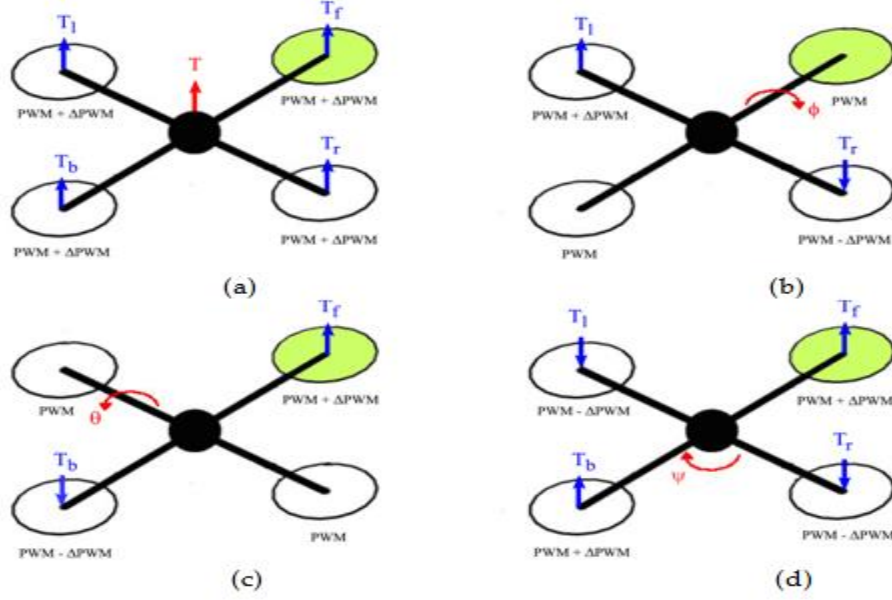


Figure 2.7: Quadcopter Thrust, Moments and Torques.

- 4- Aerodynamic effects: These are the forces external to the quadcopter system and which have a parasitic effect on quadcopter flight and are generally introduced by environmental conditions. Some aerodynamic effects must be taken seriously. The drag force is of the most importance due to the non-elliptical shape of the quadcopter structure. Drag force is directly proportional to the system linear velocity and contributes to the total forces acting on the quadcopter.

$$F_d = K_d * v^2 = K_d * |v| * \vec{v} \quad (27)$$

Where F_d is the drag force, $\vec{v} = [u \ v \ w]$ is the linear velocity of the quadcopter and K_d is drag force proportionality constant. The details regarding drag constant derivation are found in appendix B.

Other aerodynamic effects: beside drag force, many aerodynamic effects are relevant to quadcopter motion. The gyroscopic moment has a direct effect on the rotational motion of the quadcopter, the ground effect, which results at vehicle take off due to air distribution below the propellers, is sometimes significant. Propellers may also result in a

gyro effect due to air resistance. However, all these effects will be neglected in the short term as the structure under study is quite simple.

Once the total forces and torques are now known. The final form of quadcopter dynamic equations can be obtained.

The total thrust generated by rotors is found by:

$$T = T_1 + T_2 + T_3 + T_4 \quad (28)$$

The total thrust has only one component in the \mathbf{Z} axis of the body frame since it results in the vertical motion of the quadcopter.

$$\vec{T}_{xyz} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = R_b^i * \vec{T} = R_b^i * \vec{T}_b = R_b^i * \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \quad (29)$$

Where \vec{T}_{xyz} is the thrust force in the inertial frame and \vec{T}_b is the thrust force in body frame.

$$\vec{T}_{xyz} = \begin{bmatrix} \cos(\psi) \sin(\theta) + \cos(\theta) \sin(\varphi) \sin(\psi) \\ \sin(\theta) \sin(\psi) - \sin(\varphi) \cos(\theta) \cos(\psi) \\ \cos(\varphi) \cos(\theta) \end{bmatrix} * T \quad (30)$$

Thus, equation (13) can be re-written as:

$$\begin{aligned} \ddot{r}_x &= \frac{(\cos(\psi) \sin(\theta) + \cos(\theta) \sin(\varphi) \sin(\psi)) * T - K_d u^2}{m} \\ \ddot{r}_y &= \frac{(\sin(\theta) \sin(\psi) - \sin(\varphi) \cos(\theta) \cos(\psi)) * T - K_d v^2}{m} \\ \ddot{r}_z &= \frac{(\cos(\varphi) \cos(\theta)) * T - m * g - K_d w^2}{m} \end{aligned} \quad (31)$$

The above equations give a complete description of translational motion considering that all the forces affecting quadcopter are known.

The net torque $\vec{\tau}$ components are the main sources of roll, pitch and yaw rotations of the quadcopter. The roll and pitch rotations are generated by the thrust moment around the center of the quadcopter while the yaw rotation is generated by the hub torque of the rotors. According to **figure 2.1** the net torque components can be expressed as follows:

$$\vec{\tau} = \begin{bmatrix} \tau_{xb} \\ \tau_{yb} \\ \tau_{zb} \end{bmatrix} = \begin{bmatrix} l * (\vec{T}_4 - \vec{T}_2) \\ l * (\vec{T}_1 - \vec{T}_3) \\ (\vec{H}_1 + \vec{H}_3) - (\vec{H}_2 + \vec{H}_4) \end{bmatrix} \quad (32)$$

Where l is the length of the quadcopter arm. Thus the equation (23) can be re-written as:

$$\begin{aligned} \dot{p} &= \frac{l * (\vec{T}_4 - \vec{T}_2) - (J_{zz} - J_{yy}) * qr}{J_{xx}} \\ \dot{q} &= \frac{l * (\vec{T}_1 - \vec{T}_3) - (J_{xx} - J_{zz}) * pr}{J_{yy}} \\ \dot{r} &= \frac{(\vec{H}_1 + \vec{H}_3) - (\vec{H}_2 + \vec{H}_4) - (J_{yy} - J_{xx}) * pq}{J_{zz}} \end{aligned} \quad (33)$$

Note that the net torque is expressed in the body frame just like angular velocity components so that no need for rotation matrix.

It is found that equation (31) and equation (33) are used to give the state of the system with the aid of some transformations and integration to obtain the rest of the system states.

2.4 Control Inputs:

As mentioned earlier, the quadcopter has twelve states that determine its motion in space. The quadcopter has only four inputs to four different rotors, each of these inputs corresponds to a certain type of motion. The controllable parameter of the quadcopter is the angular speed of its rotors.

The control inputs vector: $U = [u_1 \ u_2 \ u_3 \ u_4]$ where each input:

$$u_1 = K_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \quad (34)$$

$$u_2 = K_T(-\Omega_2^2 + \Omega_4^2)$$

$$u_3 = K_T(\Omega_1^2 - \Omega_3^2)$$

$$u_4 = K_H(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2)$$

Where:

$u_1 \equiv$ Total thrust that is responsible for quadcopter altitude.

$u_2 \equiv$ thrust moment which results in roll attitude.

$u_3 \equiv$ thrust moment which results in pitch attitude.

$u_4 \equiv$ difference in hub torque that results in yaw attitude.

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} K_T & K_T & K_T & K_T \\ 0 & -K_T & 0 & K_T \\ K_T & 0 & -K_T & 0 \\ K_H & -K_H & K_H & -K_H \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \quad (35)$$

If the rotor velocities are needed to be calculated from the control inputs, then an inverse relationship can be defined:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4K_T} & 0 & \frac{1}{2K_T} & \frac{1}{4K_H} \\ \frac{1}{4K_T} & -\frac{1}{2K_T} & 0 & -\frac{1}{4K_H} \\ \frac{1}{4K_T} & 0 & -K_T & \frac{1}{4K_H} \\ \frac{1}{4K_T} & \frac{1}{2K_T} & 0 & -\frac{1}{4K_H} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \quad (36)$$

Taking the square root of rotors angular speed:

$$\begin{aligned} \Omega_1 &= \sqrt{\frac{1}{4K_T} u_1 + \frac{1}{2K_T} u_3 + \frac{1}{4K_H} u_4} \\ \Omega_2 &= \sqrt{\frac{1}{4K_T} u_1 - \frac{1}{2K_T} u_2 - \frac{1}{4K_H} u_4} \\ \Omega_3 &= \sqrt{\frac{1}{4K_T} u_1 - \frac{1}{2K_T} u_3 + \frac{1}{4K_H} u_4} \\ \Omega_4 &= \sqrt{\frac{1}{4K_T} u_1 + \frac{1}{2K_T} u_2 - \frac{1}{4K_H} u_4} \end{aligned} \quad (37)$$

The control vector can be seen as two main components, altitude input \mathbf{U}_1 and attitude input \mathbf{U}_2 .

$$\mathbf{U} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \quad (38)$$

$$U_1 = [u_1] \tag{39}$$

$$U_2 = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

CHAPTER 3

SYSTEM SIMULATION AND CONTROL DESIGN

In this chapter, the quadcopter model, which is formulated using physical system parameters and dynamics equations, will be implemented in Matlab Simulink environment. The Simulink model is used for open loop testing, verification and controller design. The system is linearized around specific operating point. Further, the process of designing and tuning PID controller for linear system is illustrated. Finally, the linear controller is applied to the nonlinear model.

3.1 System Simulation:

3.1.1 Quadcopter Dynamics Simulation:

In order to implement the quadcopter dynamics equations in Simulink, a level-2 M-file S-function application programming interface (API) is used. This function allows using Matlab scripting language to create Simulink custom blocks and specify their properties and behavior [27]. All dynamics and Kinematics equations derived in the previous chapter are written in Matlab script and linked to quadcopter dynamic block. The steps of creating level-2 s-function and Matlab script are mentioned in detail in appendix C.

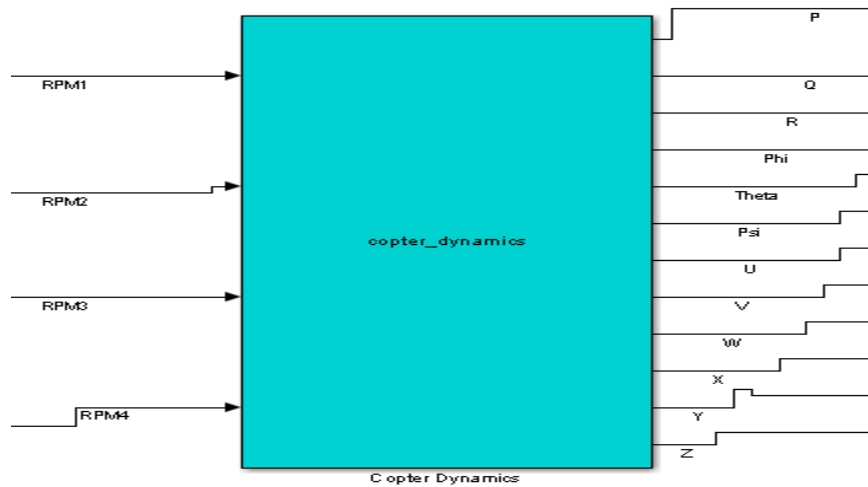


Figure 3.1: Quadcopter Dynamics

Figure (write figure number) shows the quadcopter dynamics block. This block has the motors rotation speed as its inputs while it outputs the twelve states of the quadcopter based on the previously derived equations.

3.1.2 Motor Dynamics Modeling:

It is mentioned previously that the quadcopter achieve desired motion through controlling the rotors. The rotor is mainly a Brush less DC motor (BLDC) and a propeller. The quadcopter motor has its own control system through the electronic speed controller (ESC). The motor receives an input and is expected to produce a certain output regardless of its control mechanism. Therefore, a black box modeling is used to model the motor dynamics. A linear formula that describes relationship between the motor input, which is the desired throttle, and motor output, which is the motor rotational speed, is found and implemented in Simulink as shown in **Figure (3.2)**. The details regarding motor linear formula derivation is found in appendix E.

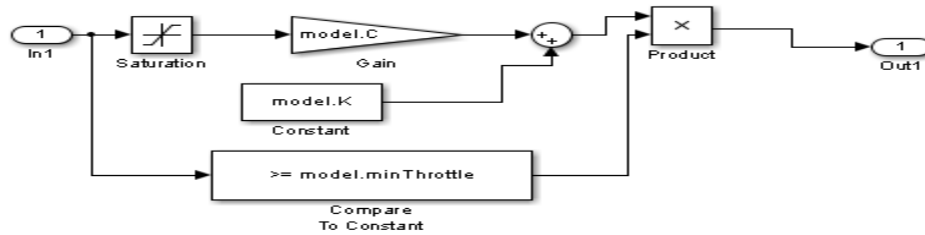


Figure 3.2: BLCD Motor Model

There are two additional blocks in the Simulink motor model. The saturation block, which represents physical limitations of the motor, and minimum throttle compare block, which represents the minimum percentage at which motor start to rotate.

3.1.3 Motor Mixer:

The purpose of motor mixer block is to determine the motors input and thus the rotation speed for each motor according to which the required thrust and moments are generated achieving the desired attitude and altitude.

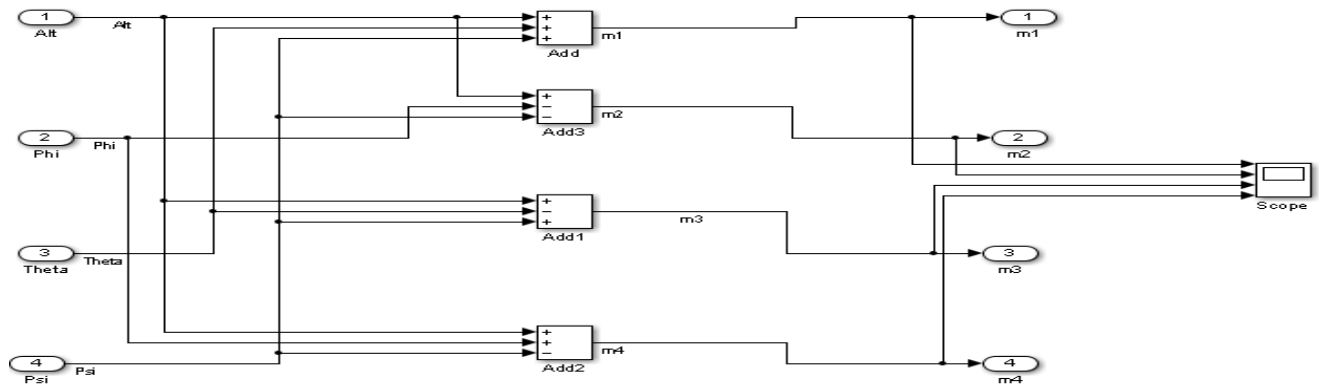


Figure 3.3 : Simulink Model of a Motor Mixer

3.2 Open Loop Simulation:

This simulation method is used to verify that the system built in Simulink is working and responding as expected. Generally it is the first step in verification.

Figure (3.4) shows block diagram of open loop system simulation.

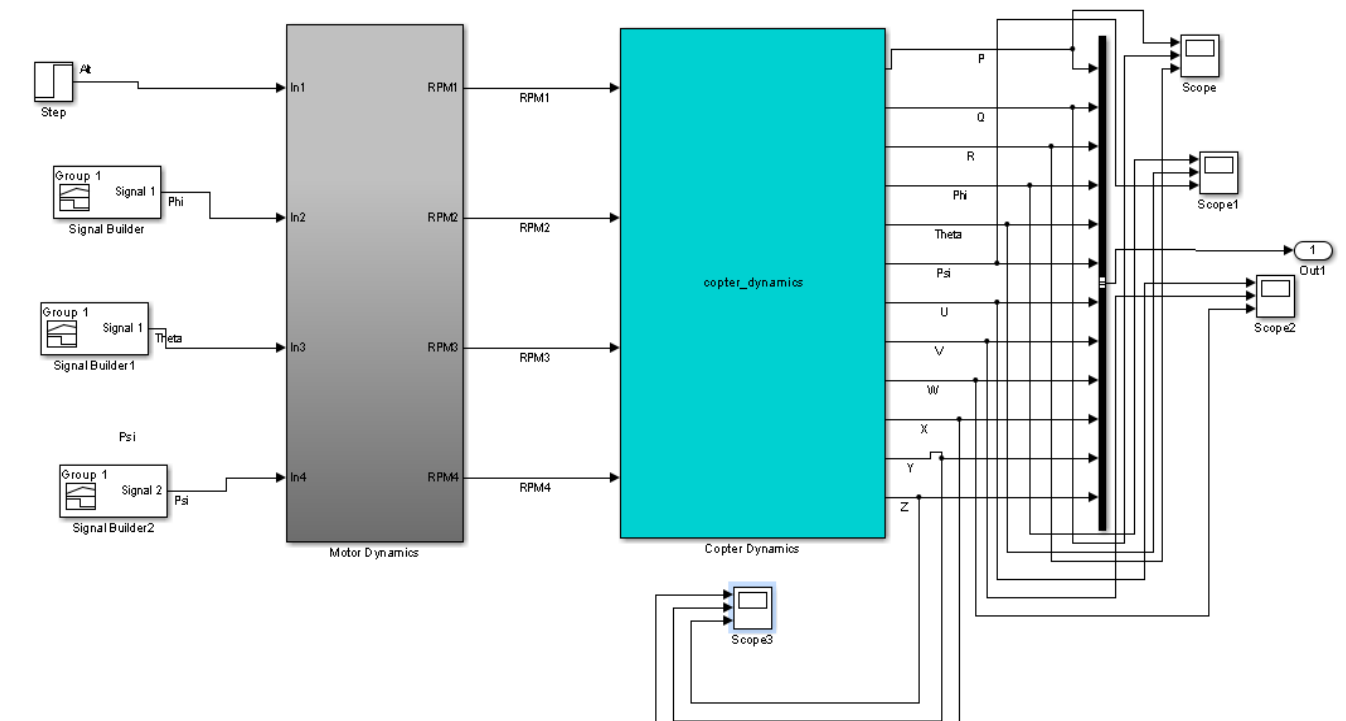


Figure 3.4:Open loop Quadcopter System

Performing open loop Simulation leads to the following observations:

- Changing the thrust of each motor equally only changes the altitude of the quadcopter. Thus, hovering condition results when the total thrust is equal to the quadcopter weight. Hovering condition can be expressed by the following equation:

$$m * g = 4 * \vec{T} = 4K_T\Omega^2 \quad (40)$$

- When the total thrust is equal to the quadcopter weight and two rotors of the same diagonal rotate faster the other two rotors; only the yaw angle changes.
- When changing the roll and pitch angles; the system oscillates and gets unstable due to nonlinearity and absence of the controller.

These observations comply with the mathematical model derived and thus first step in verification is achieved.

3.3 Controller Design:

As previously mentioned, the quadcopter is underactuated system which has four control inputs and six DOF. While quadcopter can move in vertical axes without changing irrelevant states, it needs to change attitude when moving in x or y axes.

The control system consists of two main loops; inner loop, which corresponds to orientation control, and outer loop which corresponds to the position control. The main inner loop consists of two cascaded loops as well; one loop for tracking the angular rate and another loop for tracking the angle. The inner loop must be at least ten times faster than the outer loop [7]. Specific roll and pitch angles, which are calculated according to desired (x, y) coordinates, are compared to the actual angles measured by inertial measurement unit (IMU) of the system.

In the outer loop, the actual position is compared to the desired position and thus U_1 is calculated [28]. The control input vector components as shown in equation (38) serve as inputs to the control system as shown in the **Figure 3.5**

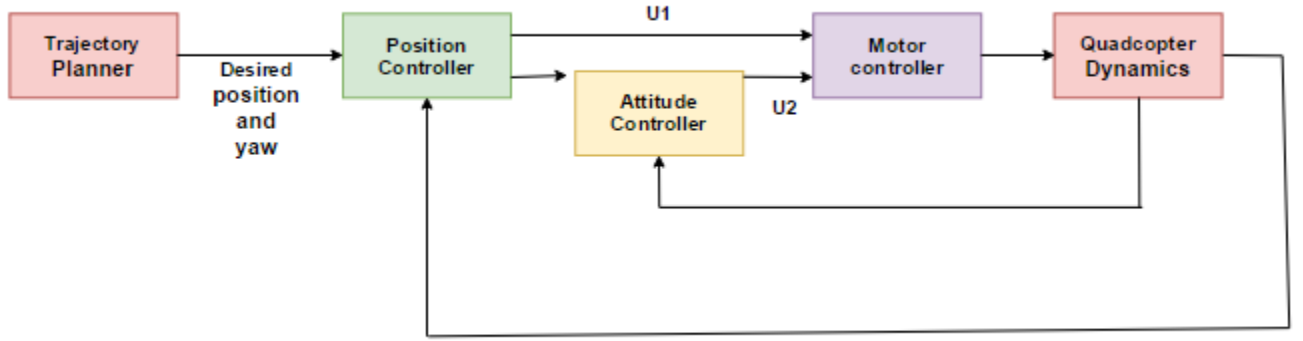


Figure 3.5 : Simplified Comprehensive System Control

3.3.1 Linearization of Quadcopter:

Designing a linear controller requires linearizing the nonlinear model as a first step. Next, linearized equations are used to design the PID controller. The designed controller is applied to the nonlinear system and its consequent response is compared to the linearized model response. The linearization is performed around equilibrium operating point (hovering) [7].

It is noticeable that the source of nonlinearity in previously derived equations is $\sin()$ and $\cos()$ functions of varying roll and pitch angles. When the quadcopter is observed around equilibrium hover configuration, its states reduce as follows:

$$r = r_0, \psi = \psi_0, \varphi = 0, \theta = 0, U_1 = m * g, U_2 = 0 \text{ and all derivatives} = 0 \quad (41)$$

Linear equations can be derived using Taylor's series assumptions when the system is near equilibrium configuration.

$$\sin(\theta) = \theta, \sin(\varphi) = \varphi, \cos(\theta) = 1 \text{ and } \cos(\varphi) = 1 \quad (42)$$

Recalling equation (5), it can be re-written as follows:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = T^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\varphi) & \sin(\varphi) \cos(\theta) \\ 0 & \sin(\varphi) & \cos(\varphi) \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (43)$$

Applying equation (42) to T^{-1} matrix:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\theta \\ 0 & 1 & \varphi \\ 0 & \varphi & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (44)$$

$$p = \dot{\phi} - \dot{\psi}\theta \quad (45)$$

$$q = \dot{\theta} + \dot{\psi}\varphi$$

$$r = \dot{\varphi}\theta + \dot{\psi}$$

Substituting $\dot{\psi}\theta \approx \dot{\psi}\varphi \approx \dot{\varphi}\theta \approx 0$ leads to:

$$p = \dot{\phi} \quad (46)$$

$$q = \dot{\theta}$$

$$r = \dot{\psi}$$

Expressing equation (23) in term of control inputs:

$$\dot{p} = \frac{u_2 - (J_{zz} - J_{yy}) * qr}{J_{xx}} \quad (47)$$

$$\dot{q} = \frac{u_3 - (J_{xx} - J_{zz}) * pr}{J_{yy}}$$

$$\dot{r} = \frac{u_4 - (J_{yy} - J_{xx}) * pq}{J_{zz}}$$

The product of two terms around zero is equal to zero, this leads to:

$$qr \approx pr \approx pq \approx \dot{\phi}\dot{\theta} \approx \dot{\theta}\dot{\psi} \approx \dot{\psi}\dot{\phi} \approx 0 \quad (48)$$

Then:

$$\dot{p} = \frac{u_2}{J_{xx}} = \ddot{\phi} \quad (49)$$

$$\dot{q} = \frac{u_3}{J_{yy}} = \ddot{\theta}$$

$$\dot{r} = \frac{u_4}{J_{zz}} = \ddot{\psi}$$

As a result, movement along the x and y axes can be controlled by changing the values roll and pitch angles.

This is only true for small angles assumption, i.e. in range (30° to -30°). However, in simulation and real implementation; this range is limited to (20° to -20°).

Using previous assumptions, the position equations become:

$$m * \ddot{r}_x = \theta * \cos(\psi) + \phi * \sin(\psi) * U_1 \quad (50)$$

$$m * \ddot{r}_y = \theta * \sin(\psi) - \phi * \cos(\psi) * U_1$$

$$m * \ddot{r}_z = -m * g + U_1$$

In Simulink environment, **Linear Analysis Tool** is used for the purpose of linearization, linear models extraction and relating each input to the corresponding output of the system. Then, these models are used for linear controller design.

3.3.2 Altitude Control:

The actual quadcopter altitude is measured using barometer sensor. This measured altitude is then compared to the desired altitude. The error obtained is fed to the altitude controller to generate U_1 signal.

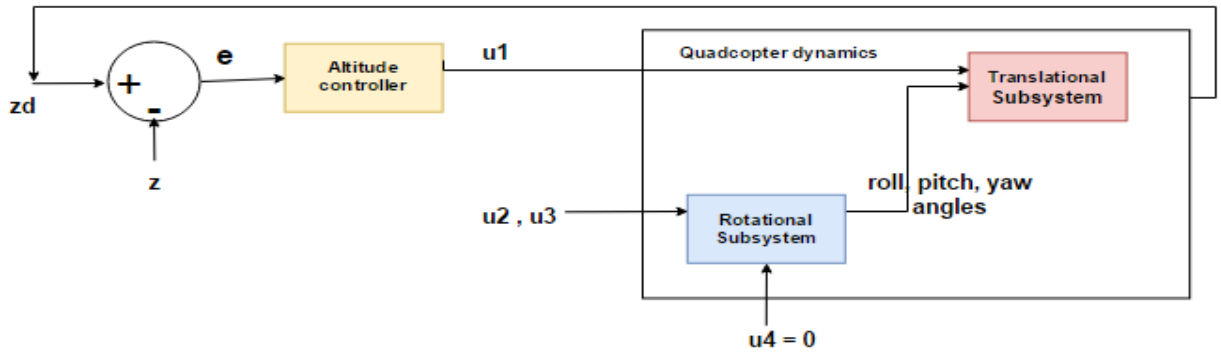


Figure 3.6 : Altitude Controller

From equation:

$$U_1 = m * (\ddot{r}_z + g) \quad (51)$$

3.3.3 Attitude and Heading Control:

Once U_1 is calculated by the outer loop, the inner loop commands should now be calculated. Since the movement along x and y axes is controlled through roll and pitch angles. The equations of (50) are re-arranged to give:

$$\begin{bmatrix} \theta_{des} \\ \phi_{des} \end{bmatrix} = \begin{bmatrix} -\sin(\psi) & -\cos(\psi) \\ \cos(\psi) & \sin(\psi) \end{bmatrix} * \frac{m}{U_1} * \begin{bmatrix} \ddot{r}_x \\ \ddot{r}_y \end{bmatrix} \quad (52)$$

$$\psi_{des} = \psi \quad (53)$$

The previous equation allows the design of inner feedback loop by simply specifying U_2 using attitude controller. In real applications, this is done via the radio control.

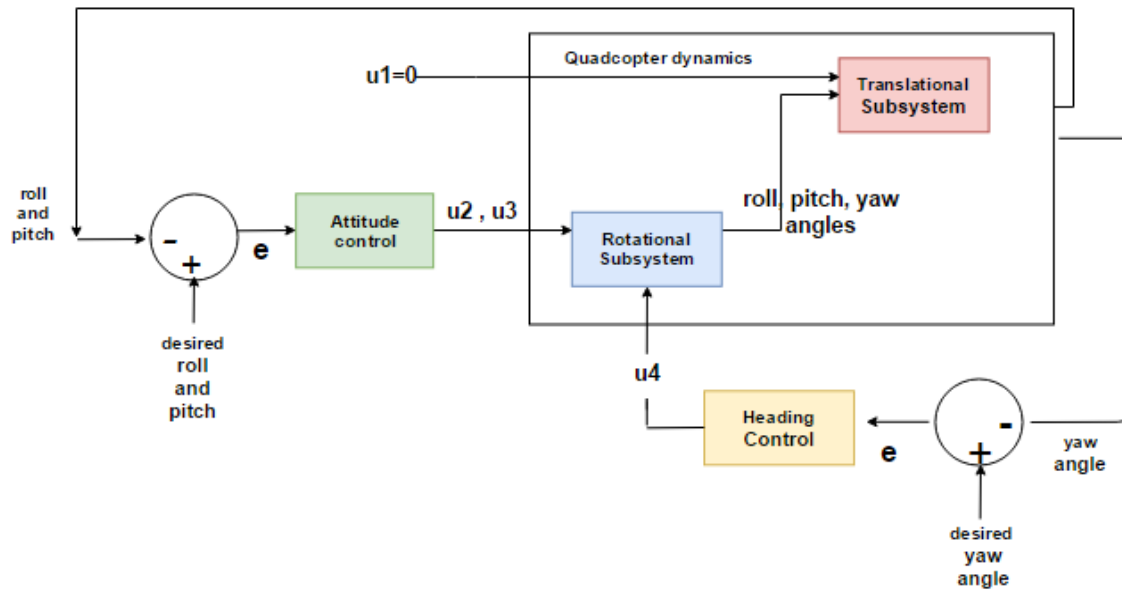


Figure 3.7 :Attitude and Heading Controller

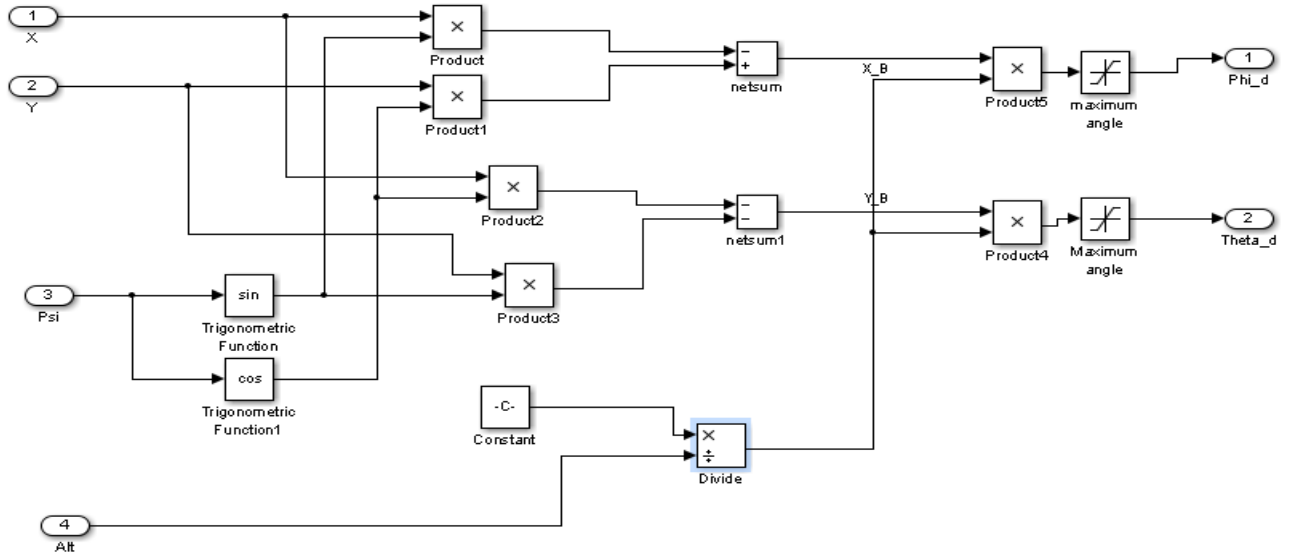


Figure 3.8 :Simulink Model for Translating Position (x,y) to Pitch and Roll Angles

3.3.4 Overall System Control:

The full autopilot controller is obtained by combining all previous controllers together in a single diagram. These controllers can contain any type of control algorithms.

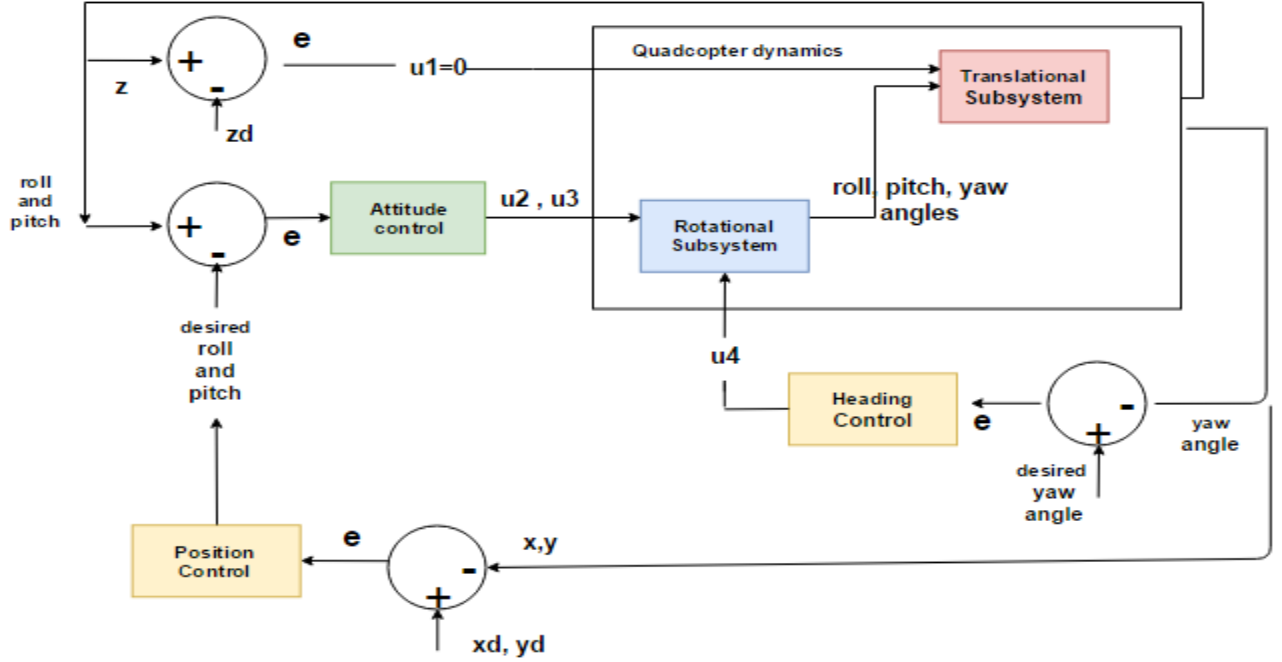


Figure 3.9 :Position and Heading Controller

3.4 Simplified PID Controller

Generally this simplified scheme of controller generates signals depending on the error between the actual and desired state. In this thesis, PID controller is selected to stabilize the quadcopter because it is intuitive to tune and easy to implement in code with little processing power.

PID controller has three tunable gains; proportional gain K_p , integral gain K_I and derivative gain K_D . It can be represented with the following transfer function:

$$G(s) = K_p + \frac{K_I}{s} + K_D * s \quad (54)$$

The proportional term K_p is directly proportional to the error, increasing it will increase the rise time, i.e. making the system response faster. The Integral term K_I is proportional to the sum of the error values accumulated over time; this error often represents a disturbance (wind, modeling

errors, etc). Increasing K_I will decrease steady state error of the feedback system, however, this term introduces a pole at the origin of the S-plane which might make the system oscillate and even go unstable. The derivative K_D term is a multiple of the rate of change of the error. It is used to reduce the overshoot and make the system response slower. K_D acts as a damper which allows the use of larger K_P value and results in improving steady state performance [7] .Since K_D is acting on the derivative of the error, and not the error itself, it is always used in combination with K_P and K_I .

The representation of basic PID controller is shown in **Figure 3.9**[29]

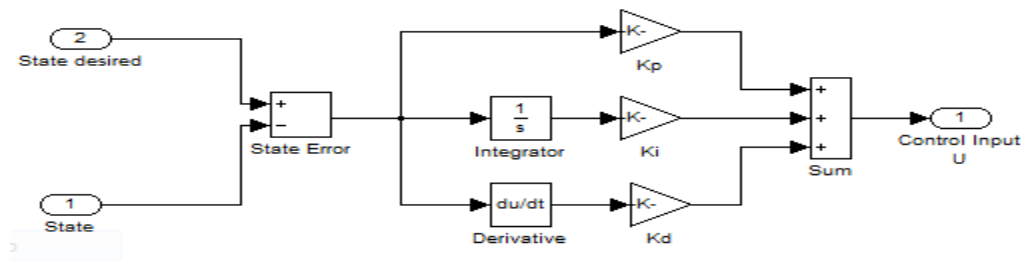


Figure 3.10: Simplified PID Controller

There are various methods to tune PID controller and determine the best values of control gains. In this thesis, the values are first determined for simulation purposes and then verified on the actual system.

3.5 Improving PID Controller

Although PID controller is very simple to design and implement, there are some problems that must be considered in order to improve controller performance and stability. This section presents these problems and their possible solutions.

3.5.1 Integrator Windup

This phenomena happens due to the limitations in actuators (motor has limited speed; a valve cannot be more than fully opened or fully closed, etc). Since the integral term corresponds to the error summation, it might become very large, and the controller will reach actuator limits. As a result the feedback loop will be broken and the actuator will not respond to the controller commands due to error changes. However, It may take a long time before the integrator and the

controller output come inside the saturation range. This nonlinear behavior is called Integral Windup.

There are many ways to solve this problem. The easiest way is to put limits for the summation. When the controller saturates, the summation is suspended until the controller output moves away from the limit [30].

3.5.2 Derivative Kick

This problem occurs when the set point rapidly changes. For most applications, the set point remains relatively constant. When following desired trajectory, the set point may continue to vary which will result in large derivatives and could result in instability.

To solve this problem, it is assumed that the derivative of the set point doesn't often change and its derivative is set to zero [30]. The improved PID controller shown below:

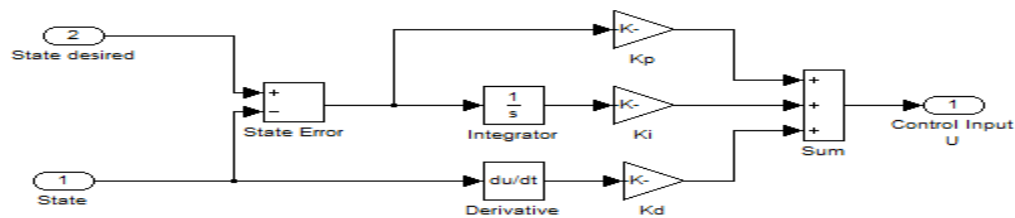


Figure 3.11: Improved PID Controller

3.6 PID Controller Implementation in Simulink

3.6.1 Linearization

One of the main advantages of using Matlab and Simulink environment for simulation is the provision of various tools for system analysis and design. The linearization is performed to extract linear transfer functions that relate each input and output using Simulink **Linear Analysis Tool**. Next, these linear models are used to design linear controller that will be later implemented in the nonlinear system with linear region operation restrictions. In addition, these models can also be used for system verification by comparing them to the linearized equations derived

previously. All controllers are designed using **Simulink SISO tool** .Root locus is used to design the compensators as well as setting the appropriate gain to acquire the desired response.

3.6.2 Altitude PID Controller Design:

Taking Laplace transform of equation (55)

$$\frac{Z(s)}{U_1(s)} = \frac{1}{ms^2} \quad (55)$$

The transfer function extracted using Linear Analysis Tool:

$$\frac{Z(s)}{U_1(s)} = \frac{0.9925}{s^2} \quad (56)$$

The derived PID control law for linear system is as follows:

$$U1 = Kp * e_z + Kd * (V_{des} - V_z) + \sum Ki * e_z \quad , \quad V_{des} = 0 \quad (57)$$

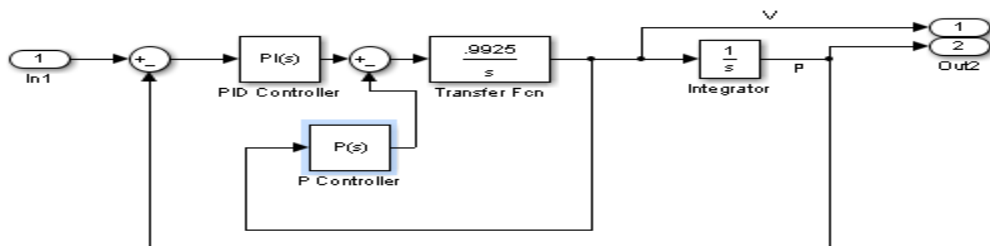


Figure 3.12: Linearized System with Altitude Controller

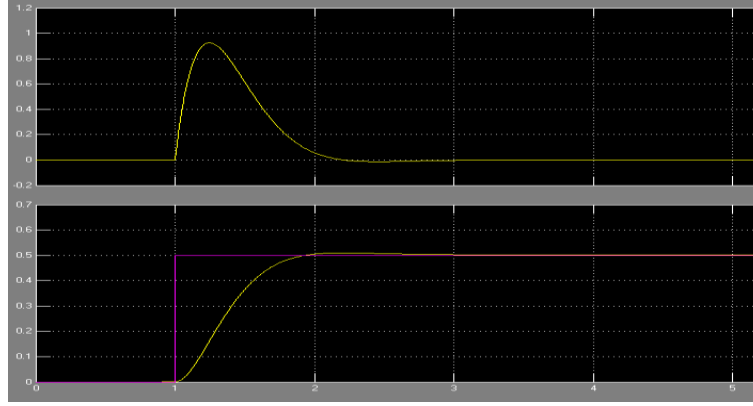


Figure 3.13: Linear Altitude Velocity and Position Response

Although linearizing the system is straight forward process, it directly works only for altitude. Further investigation shows that additional constant signal is needed to the altitude, which is equal to the system weight. This signal is necessary for hovering state simulation. The input signal U_1 is modified as follows:

$$U_1 = \frac{1}{\cos(\theta) * \cos(\varphi)} (Kp * e_z - Kd * V_z + \sum Ki * e_z + mg) \quad (58)$$

It is clear that at hovering state $U_1 = m * g$. The division by term $[\cos(\theta) \cos(\varphi)]$ is due to the fact that when quadcopter rotates with roll and pitch angles, the thrust force has two components; vertical and horizontal. The vertical component is found by: $T_{Component} = m * g * \cos(angle)$, then $m * g = T_{Component} / \cos(angle)$

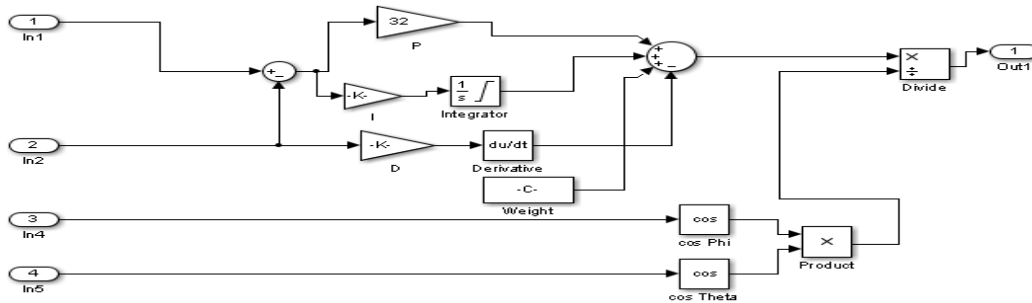


Figure 3.14 :Altitude Controller Implemented in Nonlinear System

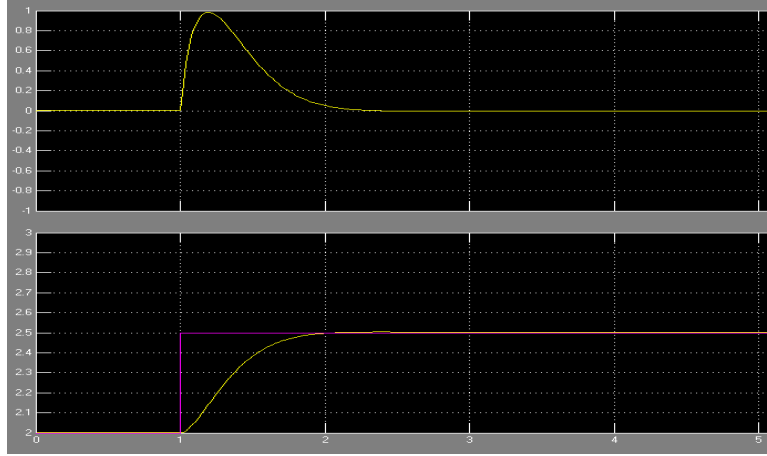


Figure 3.15 : Nonlinear Altitude Velocity and Position Response

3.6.3 Attitude Controller Design

Due to symmetry of the quadcopter, the inertia matrix is diagonal and hence the roll and pitch dynamics are identical. This theory is verified and the verification result is shown in Table 5.1. Therefore, the design steps of only pitch and yaw angles are mentioned.

Taking Laplace transform for the equation (49)

$$\frac{\varphi(s)}{u_2(s)} = \frac{1}{I_{xx} * s^2} \quad (59)$$

The transfer function extracted using Linear Analysis Tool:

$$\frac{\varphi(s)}{u_2(s)} = \frac{12.14}{s^2} \quad (60)$$

The strategy of designing control system for roll, pitch and yaw is the same. As previously mentioned in controller design section, the control system is implemented using two nested loops. This configuration has superior disturbance rejection properties. The input to the attitude controller is the angle error and the output is the desired angular velocity which is fed to the final

control subsystem. The output of this subsystem is the control signal that is mapped to motor speed.

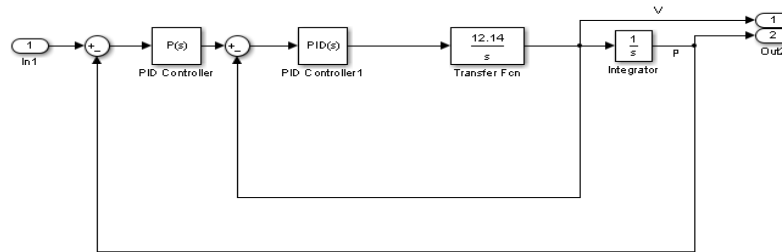


Figure 3.16: Pitch, Roll Linearized System Controller

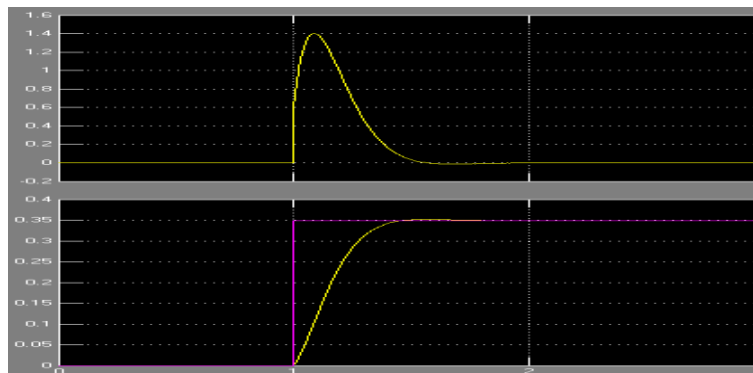


Figure 3.17 :Linearized System Roll Angular Velocity and Angle Response

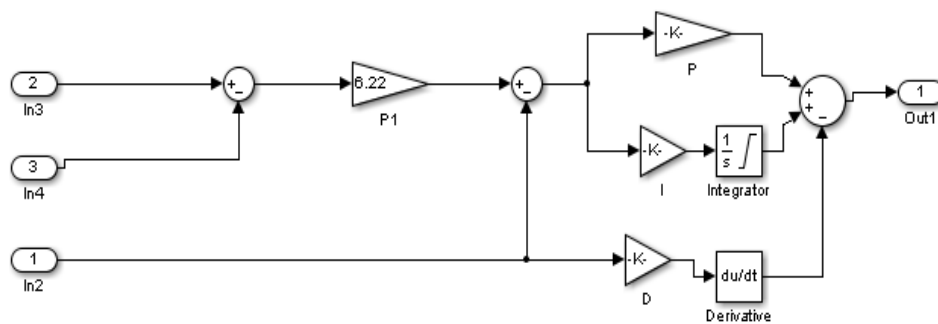


Figure 3.18 :Roll Angle Controller Implemented in Nonlinear system

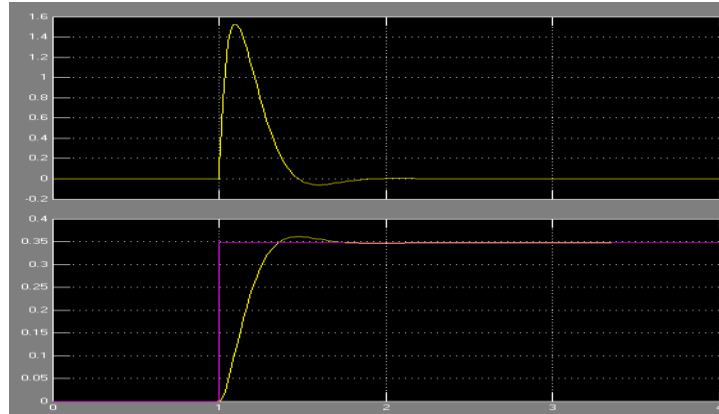


Figure 3.19: Nonlinear System Roll Angular Velocity and Angle Response

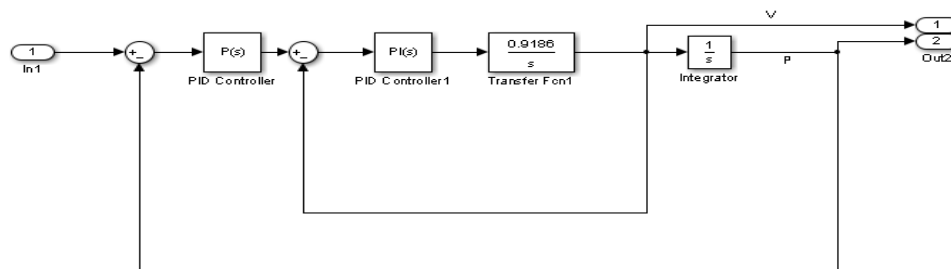


Figure 3.20: Yaw linearized System Controller

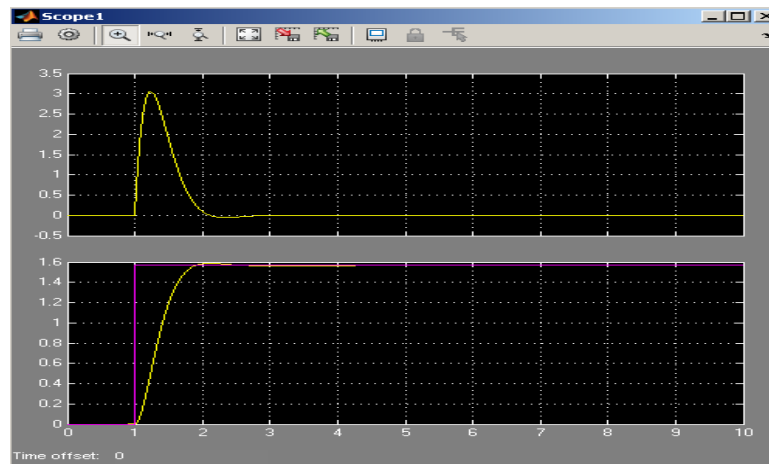


Figure 3.21 Linearized System Yaw Angular Velocity and Angle Response

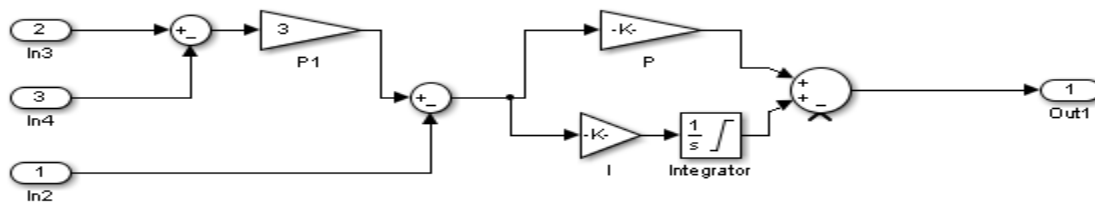


Figure 3.22: Yaw Angle Controller Implemented in Nonlinear System

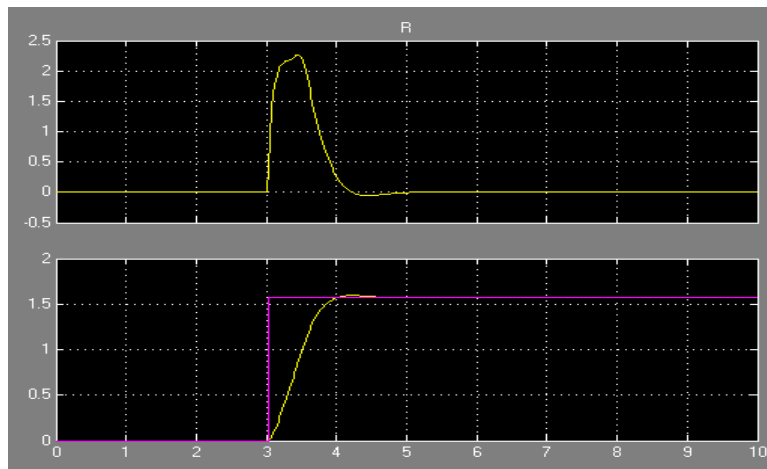


Figure 3.23: Nonlinear System Yaw Angular Velocity and Angle Response

3.6.4 Position Controller

After designing stable controllers for both altitude and attitude, position controller is designed using the same PID equation. Due to the symmetry of the quadcopter, position controllers of both axes are the same. The output of this controller is used to calculate the desired position (X_{des} , Y_{des}).

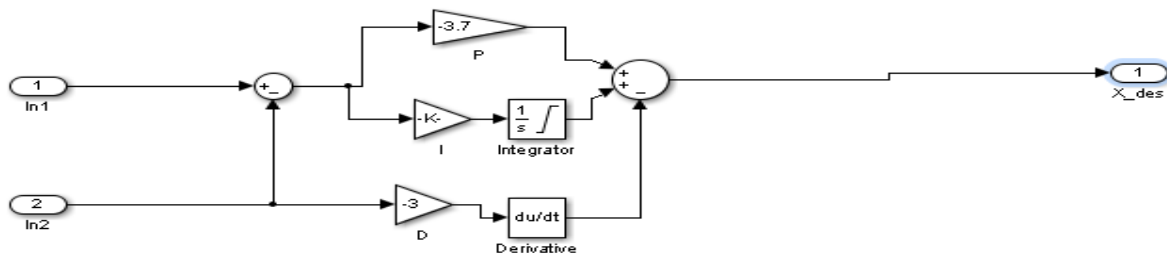


Figure 3.24 :X Coordinate Controller

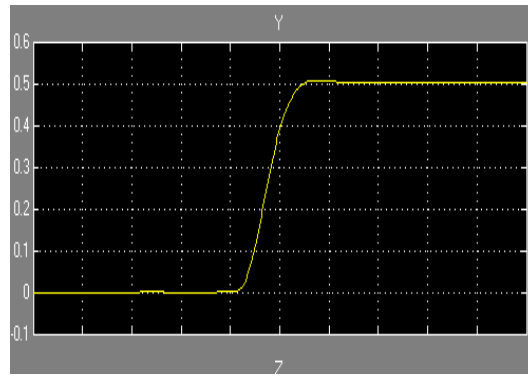


Figure 3.25 : Y Position Tracking Response

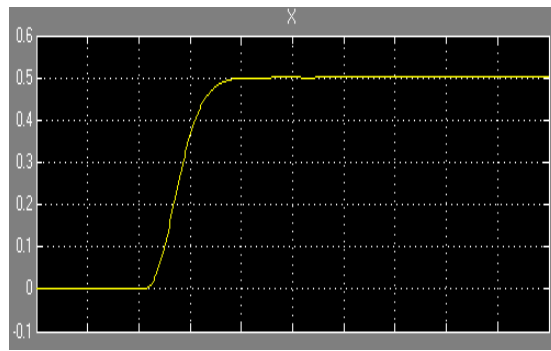


Figure 3.26 : X Position Tracking Response

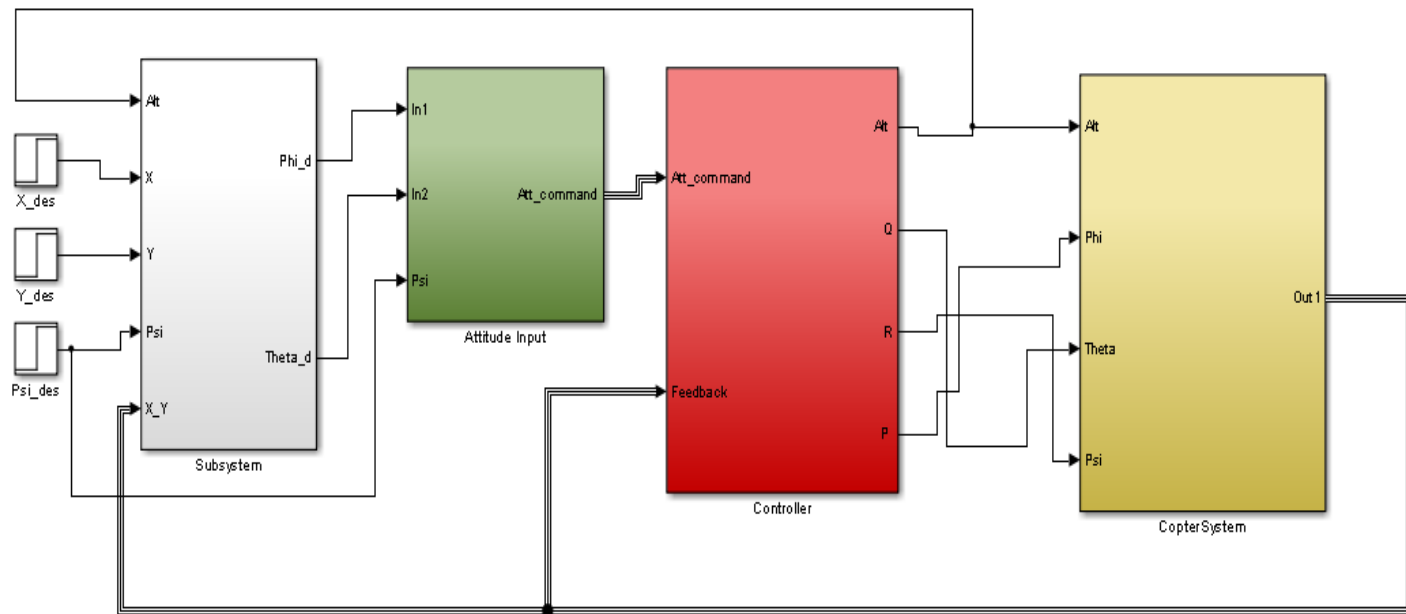


Figure 3.27 :Quadcopter Full Autopilot Model

CHAPTER 4

HARDWARE AND DESIGN IMPLEMENTATION

This chapter presents the hardware implementation of the quadcopter which provides a mean of verifying the mathematical model and controller designed earlier in the previous chapter. Not all of the simulated scenarios were implemented on real hardware due to both the lack of time and some sensor problems encountered.

As a first step, the hardware components used to build the quadcopter are described in details along with presenting the motivations behind these components selection. The second step illustrates the model construction, power distribution, components placements and connections over the quadcopter physical frame. Next the flight controller software is introduced and the procedures necessary to write the user-specified flight controller code with further illustration of the libraries required. The PID controller designed earlier is implemented in this phase. The final step is concerned with real flight test, tracking the attitude of the quadcopter using the wireless Bluetooth module attached to it and analyzing the actual quadcopter behavior for comparison purpose against the expected ideal behavior observed from the simulation environment.

4.1 Components Specifications:

The components used to implement the quadcopter hardware are selected according to certain requirements, specifications and design considerations. The quadcopter payload, flight duration, power consumption, maximum altitude and many other factors are necessarily taken into account during the selection process.

4.1.1 AutoPilot Board:

Ardu Pilot Mega (APM) is a complete open source autopilot system which is based on Arduino. This compact system gives the users chance to develop their own autopilot software. One can design an autopilot for any fixed wing planes, multi rotor vehicles, cars and even boats with APM.

APM has mainly three pin groups which are labeled as outputs, inputs, analogs. The

outputs give PWM signals to drive motors via ESCs. The inputs are connected with RC receiver. The analogs can be used as input or output. Additional sensors such as sonar and infrared range finder can be integrated by using these channels. These analog pins are also configured as digital outputs depending on the application. Beside these pin groups; there exists connector slots for GPS, wireless telemetry, power and micro USB connections.

APM can be programmable with Arduino IDE. Main processor is Atmel's ATMEGA 2560 which has 16-channel and 10 bit A/D converter. The system has a 4 megabyte dataflash memory chip for data logging. The APM includes a six degree of freedom MEMS IMU (MPU-6000) which contains a 3 axis gyroscope (angular velocity measurement), a 3 axis accelerometer (acceleration measurement), and a temperature sensor. Moreover, APM has a MEMS pressure sensor/ barometer (MS5611-01BA) that is used to measure altitude and magnetometer (heading information) which can be integrated internally or via external compass.



Figure 4.1: APM Flight Controller

4.1.2 DJI F450 Frame:

This quadcopter fiber frame is one of the simplest, most popular and most durable frames available. It is as an inexpensive alternative with many unique features. The frame includes a built-in power distribution board in its lower core plate which eliminates the need for ugly DIY wiring harnesses. This frame allows an easy build with the option of adding many other accessories to quadcopter structure. The frame is very light, strong and provides a suitable housing for all other components.



Figure 4.2: DJI F450 Quadcopter Frame

4.1.3 Battery:

The battery gives energy to the system. For the quadcopter, the weight and capacity of the battery bound the flight time and agility. The lithium polymer (LiPo) batteries are the most common power supply for the quadcopter because they have very good power to weight ratio and they are lighter than other types of batteries. The LiPo battery is mainly specified according to cell numbers, cell connection, and discharge rate. Each LiPo cell provides nominally 3.7 V. Capacity of the battery is expressed by milliampere rate (mAh) which shows how much current is supplied to the circuit for one hour after the battery is fully charged. Discharge rate simply indicates how fast the battery is discharged safely. In quadcopter setup, the motors are the main load and other components consume much less power. Typically, a LiPo battery with 3 cells (11.1 V nominal voltages), 2200 mAh capacity and 30 C discharge rate is well suited for the current purpose.



Figure 4.3: LiPo Battery

4.1.4 RC Transmitter and Receiver:

RC stands for Radio Control system. This system uses the radio signal to control a device at a distant place. It has two main parts, the receiver part which is attached to the quadcopter frame, and the transmitter part which sends the control signals to the receiver. Thus, they are used for the manual flight of the quadrotor. The reference signal is converted to radio signals by the transmitter. Then, the receiver at the vehicle collects these signals and generates appropriate PWM values to drive the vehicle. Before the first flight, the transmitter and the receiver should be bound together, so that the receiver recognizes its own transmitter signals. The RC used for the current purpose is FlySky fs-i6 2.4 GHz with 6 channels of control. Each channel of the RC transmitter provides a throttle value in the range of (1000-2000) according to specific sticks movements. The RC must be calibrated to define the desired throttle range of operation. The RC calibration steps are well described in Appendix D.



Figure 4.4: RC Transmitter and Receiver

4.1.5 Motors:

There are two basic types of technologies to choose from, Brushed DC and Brushless DC motors. The Brushless motor differs from the conventional Brushed DC Motors in their concept essentially in that the commutation of the input voltage applied to the armature's circuit is done electronically. Turnigy D3542/6 1100 KV brushless motors were chosen for our quadcopter because of their superior specifications, weight, power output and availability. The term (KV) represents the motor RPM per one volt supply voltage.



Figure 4.5: D3542/6 BLDC Motor

4.1.6 Propellers:

The quadcopter makes use of two-blade propellers rather than 3-blade ones. The former provides better pitch compared to the latter. GWS 10 x 4.7 propellers were used for the stated purpose. A propeller is named by two numbers; the first number represents the diameter of the propeller in inches and the second one is the pitch value that is the distance moved up per one propeller revolution. Propellers come in two shapes, clock-wise rotational and anti clock-wise rotational shapes. These difference shapes provide different pitch direction for the same motor rotation direction.



Figure 4.6: GWS 10 x 4.7 Propellers.

4.1.7 Electronic Speed Controllers (ESC):

Electronic speed control (ESC) is a device that controls the rotational speed of the BLDC motor. Control pins of the ESC are connected to the flight controller output. The autopilot sends the desired PWM values for each motors and the ESC drives the motors according to these values. SimonK Afro 30A ESC is used for the stated purpose. The term (30A) refers to the size of the ESC which is the amount of amps supplied for the motor connected to the ESC. Before the first

flight, the ESC must be calibrated to define the minimum and the maximum pulse widths expected, according to these widths, the ESC determines the minimum and maximum motor speed. The ESC calibration steps are mentioned in Appendix D.



Figure 4.7: Afro 30A ESC

4.1.8 Bluetooth module:

Serial port Bluetooth module is connected to APM board. The Bluetooth module provides real-time remote access to APM flight data during flight time at distances up to 10m while the PC being the receiving end. This wireless communication is important for the purpose of debugging and evaluating the performance of the designed controller. The Bluetooth module utilized for the stated purpose is JY-MCU V1.06. It is a cheaper alternative compared to its counterpart the ArduCopter telemetry and provides many reliable features.

4.1.9 Connection Setup:

The Ardupilot, unlike the case for all other communication modules, does not have specific libraries for Bluetooth module connection setup. A creative user-defined method is used to establish this Bluetooth connection as the JY-MCU V1.06 module operates over SPP. The method stated is first tested on Arduino Uno, the Bluetooth module is connected to the Arduino, and the Arduino is made to send data through the serial cable. Since the Arduino has its RX-TX terminals connected to USB-TO-TTL serial chip, the Bluetooth module copies and transfers data making use of this feature. The same strategy is applied to the APM board as it is based on Arduino controller. The APM GPS port is the only port available for data transfer and is utilized interestingly for Bluetooth connection.



Figure 4.8:JY-MCU Bluetooth Module.

4.1.10 Power Regulator:

The different components of the quadcopter operate with different voltage levels. The main power source during quadcopter flight is the LiPo battery which provides voltage in the range of (12.6-10)V for nominal flight. The ESC accepts a voltage in the range of (16-8.6)V. the APM Board requires a voltage of 5.3V. Once the APM board is powered, it provides the necessary power for other components connected to it, except the ESC since it is powered directly from the battery. For this reason, a voltage regulator is required to provide the necessary and constant voltage to the APM Board.

4.2 Installation and Components Placement:

This section covers the complete quadcopter installation and first-time setup. Making use of all components presented before, the quadcopter is assembled and the autopilot is tested using Ardupilot Mission Planner GCS [31]. Starting with frame assembly, the components are mounted onto the frame with the consideration of load balancing, components sensitivity and vibration damping.

There are two configurations available for quadcopter, X-configuration and +-configuration, according to this, the APM Board is mounted on the frame core to give the correct orientation with respect to control signals received from the user. Vibration is a serious issue that leads to APM maloperation. A thick-layer insulator is placed between the APM board and the frame core upper surface to provide the necessary isolation.

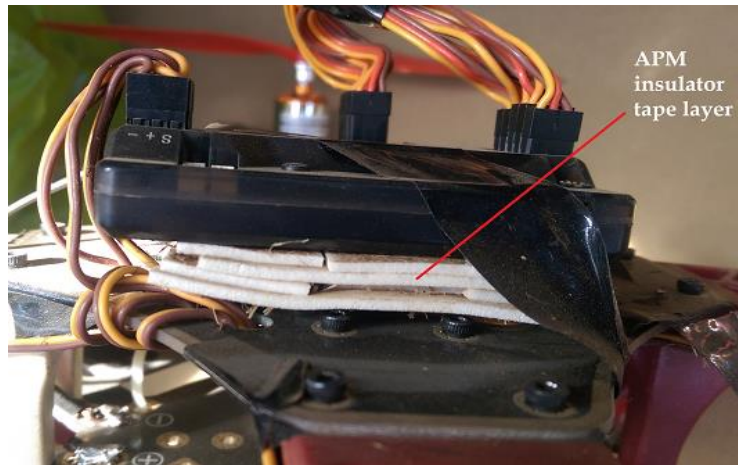


Figure 4.9: APM insulation layer.

The RC receiver is connected to the APM board is placed next to it on the same core surface. The RC channels must be connected in a certain order according to the required operation. For a nominal operation, only 4 channels of the RC are needed. The first channel refers to the roll angle control. The second channel refers to pitch angle control. The third channel is the throttle channel through which the desired altitude is set. The forth channel refers to yaw rate control. The antenna of the receiver is left free and in a position almost parallel to control signals direction.



Figure 4.10:APM Input and Output Connections

The LiPo battery is placed on the frame core lower surface which is the power distribution board. The regulator input terminals are welded to the power distribution board and its output terminals are fed to the APM Board. The ESC power cables are welded to power distribution board and the control wires are connected to the APM board. The motors have three high-power wires which are power, ground and data wires. These wires are connected to ESC high-power wires in the

same order. The ESC provides the essential pulse widths to the motor through the data wire. In order to reverse the motor direction, the motor power and ground wires are swapped. As mentioned earlier, the motors of each quadcopter diagonal must rotate in the same direction and motors of the different diagonals rotate in opposite directions, according to this, the propellers are mounted to specific motor according to its direction of rotation to provide the necessary lifting.

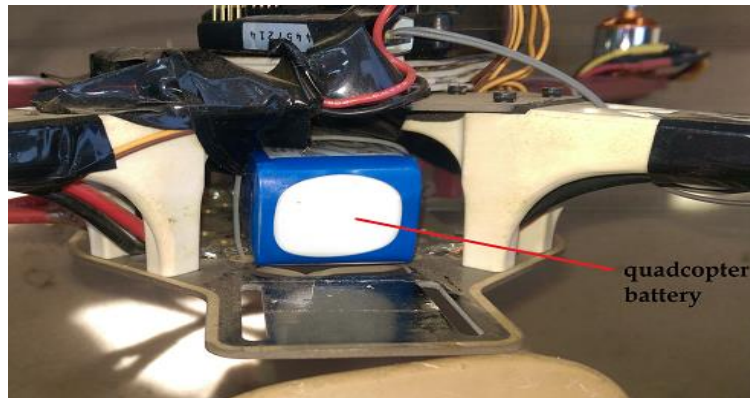


Figure 4.11: Battery placement.

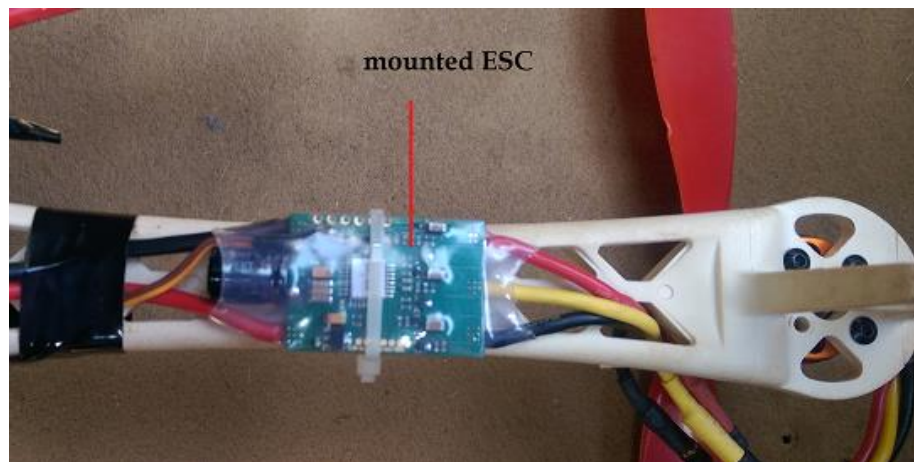


Figure 4.12: ESC Mounting.

As soon as Installation is complete, components calibration is performed. The ESCs are calibrated one by one. The steps of ESC calibration are mentioned in Appendix D. Next, the RC channels are calibrated to specify the necessary pulses width range. The APM sensors (Magnetometer and accelerometer) are calibrated to determine the correct orientation. RC and sensors calibrations require the use of mission planner software (will be described later). These calibrations steps are well described in Appendix D.

The final setup step is the first flight test which is done by the APM mission planner GCS software. This software is developed by Ardupilot organization for full-featured ground station control of autonomous and non-autonomous vehicles. According to the vehicle type and specifications, the Mission planner provides associated firmware for vehicle operation. The firmware can be an ArduCopter (multicopter control), Arduplane (plane control), ArduRover (Rover control) and so on. The ArduCopter firmware is downloaded and loaded into the APM board through Mission planner software. Once firmware is loaded, the quadcopter arming test [32] can be performed to validate the connectivity and validity of components. Then the first flight can be performed to ensure that the quadcopter is ready for user-specified modifications. These modifications are done apart from the mission planner software and rely basically on code-base approach.

4.3 Autopilot Implementation:

This section covers the development of a basic autopilot system utilizing ArduCopter code base and APM 2.7 hardware. This basic autopilot relies extensively on the work presented in the previous chapters regarding system simulation and controller design. The autopilot developed is applied to a real physical system verifying the appropriate operation of system modeling and control. The results expected is hardware and software system that enables users to implement further features and developments.

4.3.1 Setting up the Local IDE:

As a first step, compiling and loading the code onto APM board require the download of ArduPilot version of Arduino IDE along with ArduCopter Libraries. The code developed uses ArduCopter 3 libraries [33]. These libraries are the latest to support Ardupilot Mega board. The ArduCopter libraries give access to the HAL. The HAL interface allows the work with sensors, motors, and ESCs without considering low level code and drivers associated with each piece of hardware. Making use of these libraries provides an easier guide for building the code and comparing the implementation of the controller with MATLAB simulation environment. The libraries should be placed in the sketches folder to be found by Arduino software. Additional

settings require selection of the board type among Arduino types menu and setting up the drivers [34].

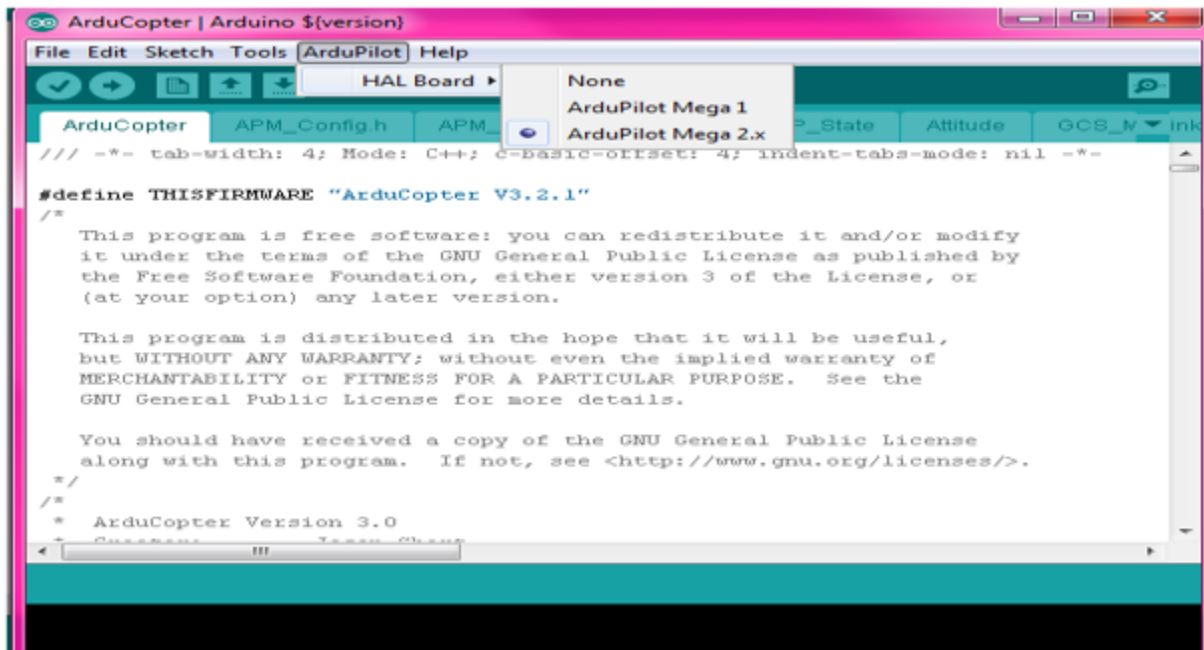


Figure 4.13: ArduPilot IDE Setup.

4.3.2 Developing the Autopilot:

Once the quadcopter is assembled, components are calibrated, the APM board is properly installed, the correct quadcopter “+” configuration is established and the quadcopter is successfully flown before, the quadcopter is now ready for autopilot code development process. This section walks through the autopilot code step by step. A simplified description of the operation states that the autopilot reads commands sent from the pilot via the RC, measures the current position and orientation, calculates the motors commands responding to RC inputs and finally sends these motors commands to motors. Each of these steps is described in the upcoming sections.

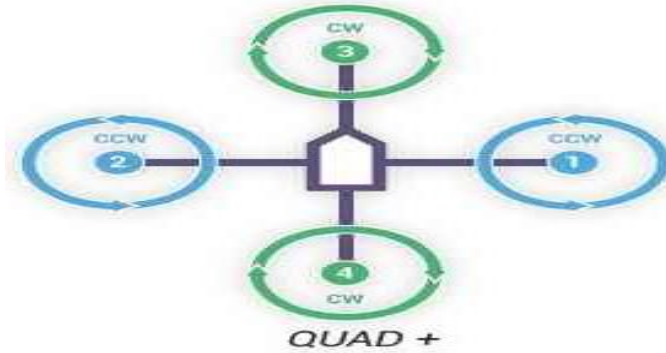


Figure 4.14: Quadcopter + Configuration

4.3.3 Reading RC Inputs:

As stated before and for this basic autopilot, only 4 channels are needed. These channels correspond to roll, pitch, yaw and altitude commands that are sent to the autopilot. RC commands are sent based on PWM frequency. A minimum value is sent as 1000 microseconds and a maximum value is sent as 2000 microseconds. The RC calibration, detailed in appendix D, is required as a first step for the sake of determining the range of minimum and maximum values being sent by RC channels. Once the channels PWM ranges are determined, a map function is used to scale the PWM input value according to the range defined by calibration.

The roll, pitch, and yaw channels inputs are scaled to specific angles ranges. The throttle channel does not need to be scaled. It is important to note that the roll and yaw angles are negative when the stick is to the left while the pitch value should be negative when the stick is forward. Maximum throttle is when the stick is forward. For this notation, some negative signs are required for the outputs when developing the code.

4.3.4 Reading the IMU and AHRS:

The APM has two sensors for determining quadcopter attitude, the gyroscope sensor which is used for angular velocity measurements around each axis and the accelerometer which is used to measure translational acceleration. Accelerometers are sensitive to vibrations while gyroscopes are vibrations resistant. The Accelerometers have slower update frequency while gyroscopes have faster update frequency. Therefore, a sensor fusion algorithm is required to combine both

measurements, minimize the disadvantages of both sensors and get more accurate attitude estimation.

In the autopilot implementation, the GPS and barometer are utilized but not to control translational position. Instead, the IMU is used for direction sensor measurement while the AHRS is used for sensor measurement fusion. The AHRS includes a processing system which provides solved attitude and heading solutions. The code developed utilizes the IMU, GPS and barometer for use in AHRS.

4.3.5 Sending Motor Commands:

The commands are sent to motors via ESCs as PWM signals. The PWM ranges from 1000 microseconds to 2000 microseconds like the RC inputs. The motors commands are sent as a response to the controller signals.

4.3.6 Implementing The Control System:

The control system takes the RC inputs, the current attitude measurements from the IMU and the AHRS and computes the control system outputs. This section implements the control system described in the chapter 3. Position controller is not used here. Instead, the RC inputs are used to compute the desired roll and pitch angles, yaw rate and thrust. Those serve as inputs to attitude controller which and rate controller and will be mixed to compute the final motor commands. The controller implementation requires defining an array to store PID values which can be updated at any time to tune the performance of the autopilot. Attitude controllers are initially proportional controllers while rate controllers are PID controllers. An important note to keep in mind is that the RC input yaw value is used to specify the desired yaw rate. This prevents the quadcopter from returning to its home orientation when the yaw stick is centered.

4.3.7 PID Implementation

Implementation of a controller in a digital system requires the use of emulation method [9]. This method considers designing the controller in continuous time domain and then discretizing the controller for digital implementation. In the case of PID controller, it is very convenient to use this method since the discrete version of PID is well known and very easy to implement. However, ensurance is needed that PID function is called at a regular time points since the

integral and derivative parts depend on change in time. This is done by calling PID function at predetermined samples of time. This approach simplifies the implementation of derivative and integral calculations. As a result, the integration can be approximated as summation where the error of each sample is summed to the previous error value. The derivative term is approximated as a difference between errors. The improvements of PID controller are implemented as explained in the previous chapter. An important issue of digital controller implementation is the selection of appropriate sampling time. The sampling time selected must be, at least, 30 times faster than settling time of the system so the controller appears continuous to the system.[9]. According to simulation result, the minimum settling time for inputs-outputs pairs is 0.41 seconds and thus the corresponding frequency is around 2.5 Hz. The duration of code execution by APM is measured to ensure sampling condition. This duration is found to be around $2 * 10^{-3}$ seconds and thus the frequency is 500Hz. Therefore, the sampling time can be up to 0.002 seconds which is 205 times smaller than the minimum settling time and hence sampling condition is satisfied.

4.3.8 Putting The Code All Together:

The final step is combining all the previous pieces of code with the necessary modifications to provide the overall autopilot code. Those modifications determine when to send commands to motors and control inputs are combined to specific motor. The control commands are only activated when the pilot raises the throttle stick slightly above the minimum value; this is set by determining a minimum throttle range at which the motors start to rotate. This final version of the autopilot code is found in appendix F.

CHAPTER 5

RESULTS DISCUSSION

In this chapter, both software and hardware controller implementations are discussed. The simulation results of both linearized and nonlinear systems are presented and compared for the sake of verification. The controller hardware response is examined using experimental testbench and the behavior observed is compared against the simulated version.

5.1 Simulation Results:

Looking at figures demonstrated in “chapter 3” it is clear that both linearized system and nonlinear system, operating in the linear region, have almost the same response. It is noticeable that both roll and pitch angles as well as the x and y positions have the same response which verifies previous assumptions about symmetry. Both angle and position controllers show excellent reference tracking with minimum overshoot. Nonlinear system response with PID controller is summarized in **Table (5.1)**.

Table 5.1: PID Controller Results

Quadcopter motion	Initial value	Desired Value	PID Controller				Settling time	Overshoot	Steady state error
			Kp (External)	Kp	Ki	Kd			
Altitude(z)	2 m	2.5 m	0	32	0.262	7.057	1.05 sec	11.3%	0
Attitude(θ and ϕ)	0°	20°	6	1.657	0.06616	0.03016	0.41sec	2%	0
Heading(ψ)	0°	30°	3	8.308	3	0	1.2 sec	2.4%	0
Position(x,y)	0 m	0.5 m	0	3.7	0.01	3	1.62 sec	0%	0

Both altitude and attitude equations, which are linearized using Simulink Linear Analysis Tool, result in the same dynamics compared to linearized equations derived theoretically, which verifies the assumptions made in equations simplification and that the simulation is working properly.

5.2 Controller Implementation Results:

Before applying the controller to a real flight test, experimental setup is built in order to examine the controller response, therefore ensuring flight safety. The experimental testbench is shown in **figure(5.1)**. The quadcopter is hanged from the top and fixed at the bottom allowing only three (roll, pitch and yaw) out of six DOF. Various reference signals are sent via RC to test the reference tracking of the system. Data of response is sent via bluetooth module to the computer at a baud rate of 115200 bit/sec. Then, Matlab script is written to record the data for analysis purposes.



Figure 5.1: Test Setup for Attitude Control

Figure (5.2) shows barometer altitude measurement while the altitude is kept at zero position. It is noticeable that large measurement noise exists and the measurement error approaches 0.5m.

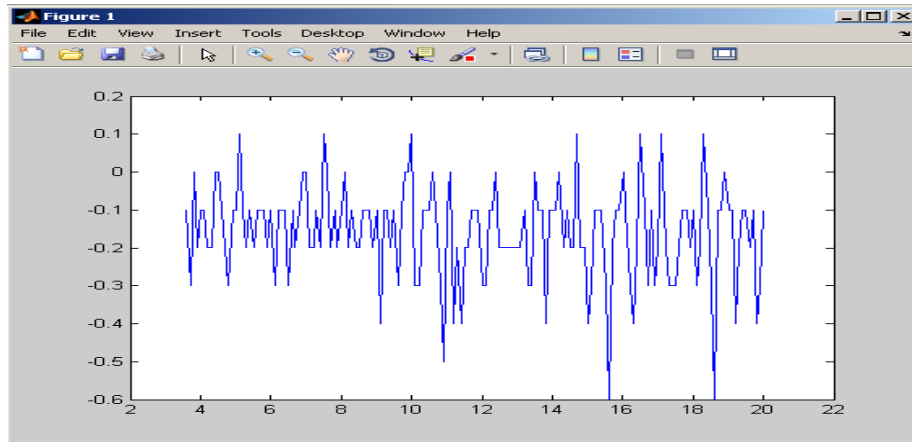


Figure 5.2 :Barometer Zero Altitude Readings

On the other hand, the measurement noise observed is insignificant when reading the roll and pitch angles. The steady state error for both roll and pitch is around 0.03° whereas the yaw steady state error is larger and in the order of 0.3° . Therefore, and due to barometer limitation, only attitude controller of roll, pitch and yaw angles is implemented.

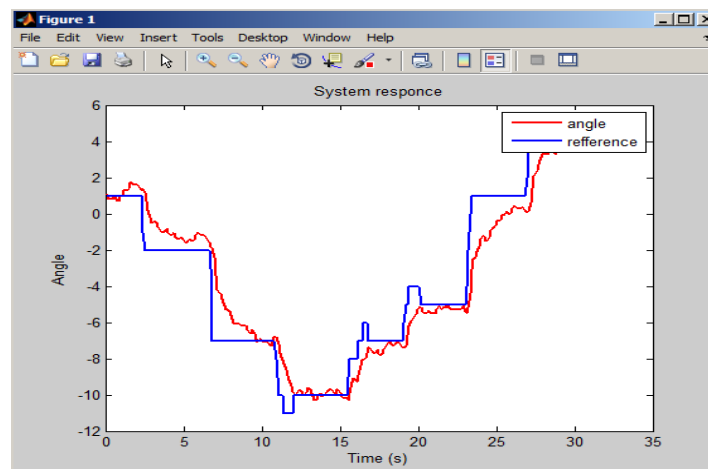


Figure 5.3 :Pitch Angle Tracking

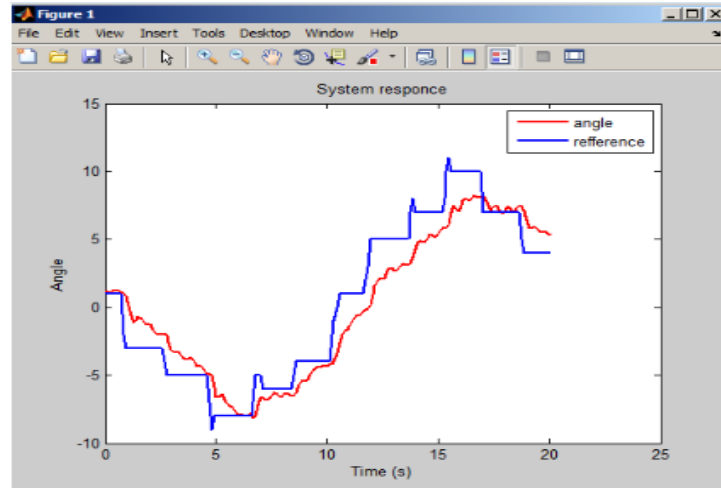


Figure 5.4 :Roll Angle Tracking

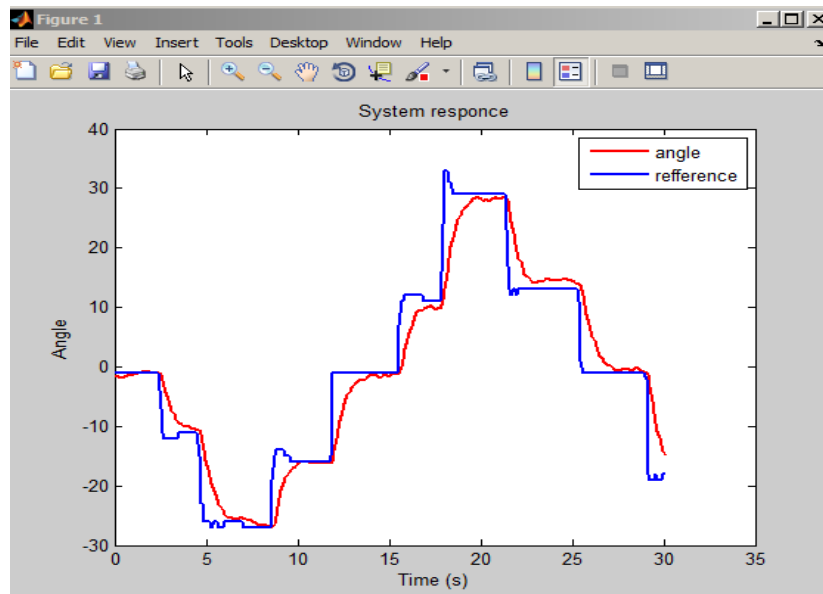


Figure 5.5 : Yaw Angle Tracking

It can be seen that there is a considerable disturbance, typically when measuring pitch and roll angles. This disturbance is due to tension exerted by testbench rods

When rolling and pitching, the quadcopter moves in x and y directions while being held still by testbench ropes, this may result in some vibration and prevent tracking of reference signals. It is clear that the pitch angle response is much better than the roll angle response. In spite of all experimental configuration drawbacks, it shows that both roll and pitch angles controllers have good reference tracking. The yaw angle controller, on the other hand, has excellent reference

tracking and is not affected by vibration due to the fact that the tension exerted by the ropes is through the z-axis.

The results of controllers' experimental setup cannot be directly and quantitatively compared to those obtained from simulation due to the presence of disturbance and mistracking of reference signals. Obtaining more quantitative results requires getting rid of all sources of disturbance and vibration by simple approximation approach. The average of total vibration values is obtained and is approximated as a straight line single value. This leads to more accurate experimental results summarized in **Table (5.2)**.

Table 5.2: PID Controller Implemented on Real System

Angle	PID Controller				Settling time	Overshoot	Steady state error
	K_P (External)	K_P	K_I	K_D			
Roll	4	1.657	0.06616	0.03016	1.2sec	4%	0
Pitch	4	1.657	0.06616	0.03016	1.3sec	6%	0
Yaw	3	8.308	3	0	1.8 sec	2%	0

It is noticeable that the actual approximated response is very close to the simulation response. However, the outside “ K_P ” gain is changed from 6, which is the simulation value, to 4 due to some oscillations. This is reasonable because several approximations are made when dealing with simulation. Firstly, the relation between the motor throttle input and angular rotational speed is approximated as linear. Moreover, all aerodynamics effects are calculated approximately due to lack of the experimental devices. Despite all these uncertainties, the designed controller is capable of stabilizing the system and tracking reference signals. This shows the adaptability designing cascaded PID controller since it compensates for all approximations and uncertainties in the model.

CHAPTER 6

CONCLUSION

The main purposes of this thesis are to build an accurate and reliable simulation representing the real physical system, develop a control algorithm to stabilize all its states and implement the controller in the actual quadcopter to verify its performance on the real system.

Firstly, dynamics and kinematics of quadcopter aircraft were investigated and a mathematical non-linear model was acquired using classical mechanics. Major aerodynamics effects were taken in to consideration, such as thrust, hub and drag, while gyro and ground effects were neglected due to their small contribution to system.

In order to build an accurate model, experimental identification of system parameters was performed. Those experiments include motor model, inertia calculations, quadcopter mass estimation, hub torque, thrust and drag coefficients. Derived dynamics equations and experimental data obtained were used to build full autopilot system on Matlab/Simulink environment. Then, significant improvements, which are hovering condition, the effect of roll and pitch on thrust and the effect of altitude signal on the translation of the position to attitude, were applied to the simulation environment during model testing and analysis. These improvements were not introduced in previous theses.

Linearized model of quadcopter system was obtained for the purpose of using linear control techniques in system analysis and controller design. Improved PID control techniques were applied to stabilize quadcopter, track the reference input signal and reject small disturbance (light wind). The response of nonlinear model operating within linear region was compared to linear model response.

One of the main valuable outputs of this thesis is the provision of simulation environment that enables designing autopilot systems with different controller techniques.

Finally, successful implementation of PID controller was achieved using APM 2.7 in the quadcopter hardware system. Ground station for short distance applications was built by connecting the system with PC via Bluetooth module. Data from the controller was received and

plotted in the real time using Matlab script. The main benefit of that was the ability to debug the system errors and analyze the controller performance in the real time. Although APM 2.7 is a powerful controller with many features, it is noted that better controllers are needed, with more space for coding and data logging, for more complicated applications such as autonomous navigation.

Recommendations and Future work:

In order to improve disturbance rejection and exceed the limitation of operating around linear region, nonlinear controller design techniques are needed to take system nonlinearity into considerations. Examples are: sliding mode controller, backstepping controller and linear quadratic regulator.

The barometer encountered problem which can be solved by combining it with accelerometer $-z$ axis using Kalman filter. Moreover, for autonomous mission planning, full autopilot implementation is needed using GPS for position estimation in outdoor scenarios. Another important thing to mention is the application of obstacle avoidance system that can be implemented using distance sensor, such as infrared sensor, sonar or vision system.

As the entire model parameters were assumed to be accurately known, this is not the case in reality, thus, more tests and experiments are needed, such as wind tunnel and thrust meter, to get more accurate model for simulation and develop an adaptive control algorithms to account for the system uncertainties.

The distance between the quadcopter and the ground station was only about 10m; this distance can be extended using telemetry with antenna.

Sensors' readings noise must be taken into consideration and its effect on system stability must be tested under the effect of the applied controller.

Final important improvement is to discretize the model and design the controller in discrete domain. This helps to estimate effect of time delays of the sensors and microcontroller.

REFERENCES

- [1] <http://terpconnect.umd.edu/~leishman/Aero/Breguet.pdf>, last accessed at 11.8.2017
- [2] Hongning Hou, Jian Zhuang, Hu Xia, Guanwei Wang, and Dehong Yu. A simple controller of minisize quad-rotor vehicle. In Mechatronics and Automation (ICMA), 2010
- [3] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In Intelligent Robots and Systems, 2004.
- [4] Heba talla M. N. ElKholy (2014), Dynamic Modeling and Control of a Quadrotor Using Linear and Nonlinear Approaches. American university in Cairo, School of Science and Engineering, 2014 .
- [5] Ruth Tesfaye (2012). Modeling and Control of a Quad-rotor Unmanned Aerial Vehicle at Hovering Position. Addis Ababa university, School of graduate studies, Addis Ababa Institute of technology, Electrical and computer engineering department, 2012
- [6] Mehmet S. BÜYÜKSARIKULAK, Autopilot design for a quadcopter, The graduate school of natural and applied science of Middle East technical university, 2014.
- [7] Alexander Lebedev, Design and Implementation of a 6DOF Control System for an Autonomous Quadrocopter, Julius Maximilian University of Würzburg, Faculty of Mathematics and Computer Science, 2013
- [8] Camilo Ossa-Gomez, Design, Construction and Control of a Quadrotor Helicopter Using a New Multirate Technique, The Department of Electrical and Computer Engineering, Concordia University Montreal, Quebec, Canada, 2012.
- [9] Fum, Wei Zhong, Implementation of Simulink controller design on Iris+ quadrotor, Naval Postgraduate School, Monterey, California, 2015.
- [10] C Balas, Modelling and Linear Control of a Quadrotor, Cranfield University, 2007.
- [11] Vijaykumar Sureshkumar FNU, Autonomouous control of quadcopter UAV using Fuzzy logic, university of Cincinnati, 2014.
- [12] wilselby.com/research/arducopter, last accessed at 10.10.2017.
- [13] <https://github.com/dch33/Quad-Sim>, last accessed at 11.10.2017
- [14] Andrew Neff, Linear and Non-Linear Control of a Quadrotor UAV, Clemson university, 2007.

- [15] Dean Bawek, Design and implementation of a control system for quadrotor MAV, 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, 2012
- [16] http://www.brokking.net/ymfc-al_main.html, last accessed at 10.10.2017
- [17] <http://ardupilot.org/copter/>, last accessed at 8.10.2017
- [18] David R. Emmanuel, Sava Jaswanth, Jogendra M. Kumar, Design and Implementation of quadcopter drone with kk 2.1.5 flight controller, School of Electronics and Communication Engineering, Lovely professional university, 2016.
- [19] J. L. Boiffier. The dynamics of flight the equations. John Wiley & Sons, 1998
- [20] S. Bouabdallah. Design and control of quadrotors with application to autonomous flying. Ecole Polytechnique Federale De Lausanne, 2007
- [21] E.L. Nikolai, Theory of gyroscopes, OGIZ, Moscow, Russia, 1948
- [22] D. Kleppner, R.J. Kolenkow, An Introduction to Mechanics, McGraw-Hill, New York, USA, 1973
- [23] S. Windnall, J. Peraire, Relative Motion using Rotating Axes, Lecture Notes in Dynamics, MIT, Boston, USA, 2008
- [24] E.B. Mikirtumov, Aircraft Models, DOSAAF, Moscow, Russia, 1956
- [25] K. Ogata, Modern Control Engineering, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010.
- [27] <https://de.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html>, last accessed at 18.6.2017.
- [28] <https://www.coursera.org/learn/robotics-flight>, last accessed at 20.9.2017
- [29] <https://www.wilselby.com/research/arducopter/controller-design/>, last accessed at 15.10.2017
- [30] <http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/>, last accessed at 30.8.2017
- [31] <http://ardupilot.org/copter/docs/initial-setup.html>, last accessed at 10.10.2017
- [32] http://ardupilot.org/copter/docs/arming_the_motors.html, last accessed at 10.10.2017
- [33] <http://ardupilot.org/dev/docs/apmcopter-programming-libraries.html>, last accessed at 12.10.2017
- [34] <http://ardupilot.org/dev/docs/load-the-code-onto-apm2x.htm>

Appendix A:

Transformation Matrices Derivation

This appendix introduces the properties and derivation steps of both rotation matrix and transfer matrix. To express the relationship between the inertial frame and body frame three rotational matrices are required in a certain sequence. These rotation matrices are implemented in a certain way according to the application giving the overall rotation matrix or transfer matrix.

Consider the transformation from body frame to inertial frame. First, the body frame rotates about the X axis by amount of ϕ which is the roll angle. The resulting frame is called a_1 frame which has the same X axis as the body frame but different Y and Z axes. Secondly, the a_1 frame is rotated about Y axis by amount of θ which is the pitch angle. The resulting frame is called a_2 . Finally, a_2 frame is rotated about Z axis by amount of ψ which is the yaw angle to give the inertial frame.

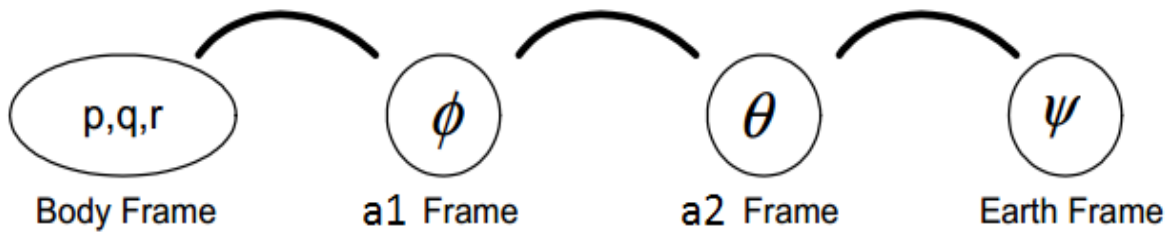


Figure A.1: Transformation Frames

Each rotation between those four frames has its own primary rotation matrix. Those rotation matrices are expressed as follows:

$$R_b^{a_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$
$$R_{a_1}^{a_2} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_{a_2}^i = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The overall rotational matrix that achieves the transformation from body frame to inertial frame is obtained by multiplying the three primary matrices:

$$R_b^i = R_b^{a_1} \times R_{a_1}^{a_2} \times R_{a_2}^i$$

$$R_b^i = \begin{bmatrix} c(\theta)c(\psi) & s(\varphi)s(\theta)c(\psi) - c(\varphi)s(\psi) & c(\varphi)s(\theta)c(\psi) + s(\varphi)s(\psi) \\ c(\theta)s(\psi) & s(\varphi)s(\theta)s(\psi) + c(\varphi)c(\psi) & c(\varphi)s(\theta)s(\psi) - s(\varphi)c(\psi) \\ -s(\theta) & s(\varphi)c(\theta) & c(\varphi)c(\theta) \end{bmatrix}$$

The Transfer matrix, on the other hand, is obtained by a bit complicated procedure. The roll rotation of a body is achieved by only one rotation, the pitch rotation requires two successive rotations and the yaw rotation requires three successive rotations. Thus, each of Euler angles is expressed in a different frame.

Since the transfer matrix T and its inverse are used to achieve transformation between body angular rates and Euler angles rates, the steps of T^{-1} derivation are done as follows:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_b^{a_1} \frac{d}{dt} \begin{bmatrix} \varphi \\ 0 \\ 0 \end{bmatrix} + R_b^{a_1} R_{a_1}^{a_2} \frac{d}{dt} \begin{bmatrix} 0 \\ \theta \\ 0 \end{bmatrix} + R_b^{a_1} R_{a_1}^{a_2} R_{a_2}^i \frac{d}{dt} \begin{bmatrix} 0 \\ 0 \\ \psi \end{bmatrix}$$

These matrices product is done and the inverse transfer matrix T^{-1} is obtained. The Transfer Matrix T can be defined to transform Angular rates of Body to Euler angles rate vector.

$$T^{-1} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\varphi) & \sin(\varphi) \cos(\theta) \\ 0 & -\sin(\varphi) & \cos(\varphi) \cos(\theta) \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & \sin(\varphi) \tan(\theta) & \cos(\varphi) \tan(\theta) \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) \sec(\theta) & \cos(\varphi) \sec(\theta) \end{bmatrix}$$

Appendix B:

Parameters Estimation

Air Density calculation:

The quadcopter thrust, drag force and drag moment coefficients are directly dependent on the air density. The air density with respect to a particular place varies basically upon the variation of humidity and ambient temperature. The atmospheric pressure, determined by height above sea level, has the main role in determining the air density. According to the quadcopter of interest, the location of flight is Khartoum city which at 381 m above sea level. The average ambient temperature is 35°C. The average humidity is about 9%. The air density of interest is found to be 1.135 Kg/m^3 . An air density online calculator can be utilized with the prior knowledge of the previous factors.

Thrust force and Hub Torque constants:

The Thrust and Hub torque constants are found according to the specific propeller used. Each propeller has its own coefficients and these coefficients are obtained through a series of experiments. UIUC university provides database for multiple types and sizes of propellers. According to the propeller of current work which is GWS 1047, those values of coefficients are used.

$$\text{Thrust } (T) = \left(\frac{C_T \times \rho \times R^4}{(2\pi)^2} \right) \omega^2 = K_T \Omega^2$$

$$\text{Hub Torque } (H) = \left(\frac{C_H \times \rho \times R^5}{(2\pi)^2} \right) \omega^2 = K_H \Omega^2$$

Where:

$$C_T \equiv \text{Thrust Coefficient} = 0.137$$

$$C_H \equiv \text{Drag moment Coefficient} = \frac{0.0577}{2\pi} = 0.0092$$

$\rho \equiv \text{Air density} = 1.135 \text{ Kg/m}^3$.

$R \equiv \text{Rotor radius} = 5 \text{ inches} = 12.7 \text{ cm} = 0.127 \text{ m}$.

$K_T \equiv \text{Thurst constant} = 0.82 \times 10^{-5}$

$K_H \equiv \text{Hub torque constant} = 1.4 \times 10^{-7}$

Drag Force:

Drag force constant is found by the following formula:

$$\text{Drag Force } F_D = \left(\frac{1}{2} \times C_D \times \rho \times S\right)V^2 = K_d V^2$$

Where:

$\rho \equiv \text{Air density} = 1.135 \text{ Kg/m}^3$

$C_D \equiv \text{Drag Coefficient} = 0.63 \text{ (Dimensionless)}$

$S \equiv \text{Reference area aka. orthographic projection of surface} = 60 \text{ cm}^2 = 0.006 \text{ m}^2$

$V \equiv \text{Linear velocity in m/s}$.

$K_d \equiv \text{Drag constant} = 0.00215$

The drag coefficient is constant for a certain shape of structure, assuming that the quadcopter has a central cube-shaped core with rotors of negligible drag effect, then the drag coefficient is found equal to 0.63

Determining Physical properties of the quadcopter:

Mass of the Quadcopter:

The Quadcopter mass is essential for modeling purposes. The quadcopter is weighted using analytical weighting scale. The quadcopter must include all the loads that will be carried by it during flight.

The total quadcopter mass can be seen as three collections of masses. The first collection is the core mass (M) which is the mass of the center of the quadcopter structure. It is the heaviest part and it weighs about 372 grams. The second collection is the total mass of motors and propellers

$(4 \times m_1)$ where m_1 represents the mass of single motor with its propeller and weighs about 77 grams. The third collection is the total mass of quadcopter arms along with the ESCs attached to these arms $(4 \times m_2)$ where m_2 represents the mass of single arm and its ESC and weighs about 67 grams.

Inertia Measurement:

The mass moment of inertia of the quadcopter can be measured using two ways, analytically and experimentally. The experimental method is only accurate for heavy weight structures since they are less likely to be affected by motion swings. Analytical method is adopted here to calculate the essential moment of inertia about the main axes of orientations. A mean of verification is provided [13] to test the validity of results obtained

Analytical method:

Since the quadcopter has a symmetric shape, the quadcopter structure can be seen as core sphere located at the center and connected to 4 smaller spheres, i.e. motors, via rectangular rods.

Fortunately, symmetrical shapes have mathematical expressions for measuring the moment of inertia. Using the same mass identification stated in the previous subsection, the inertia around both x and y axes is the same due to symmetry. The inertia around z axis is of a greater value due to the contribution of all mass collections. Hence the calculations are:

$$J_{xx} = J_{yy} = \frac{2}{5} \times M \times R^2 + 2 \times m_1 \times l_1^2 + 2 \times m_2 \times l_2^2 = 0.01122 \text{ Kg.m}^2$$

$$J_{zz} = \frac{2}{5} \times M \times R^2 + 4 \times m_1 \times l^2 + 4 \times m_2 \times l_2^2 = 0.02199 \text{ Kg.m}^2$$

Where:

$J_{xx}, J_{yy}, J_{zz} \equiv \text{Inertia around } x, y, z \text{ respectively.}$

$R \equiv \text{Radius of the core} = 5.5 \text{ cm} = 0.055 \text{ m.}$

$l_1 \equiv \text{length of the quadcopter arm} = 23 \text{ cm} = 0.23 \text{ m.}$

$l_2 \equiv \text{distance of esc from the center of quadcopter} = 14 \text{ cm} = 0.14 \text{ m}$

Verification:

Verification is done by the aid of some inertia calculator designed by a previously working researcher. The calculator takes into account all quadcopter parameters and surrounding conditions and estimates the inertia according to them. Fortunately, the results obtained are very close to the results of analytical calculation.

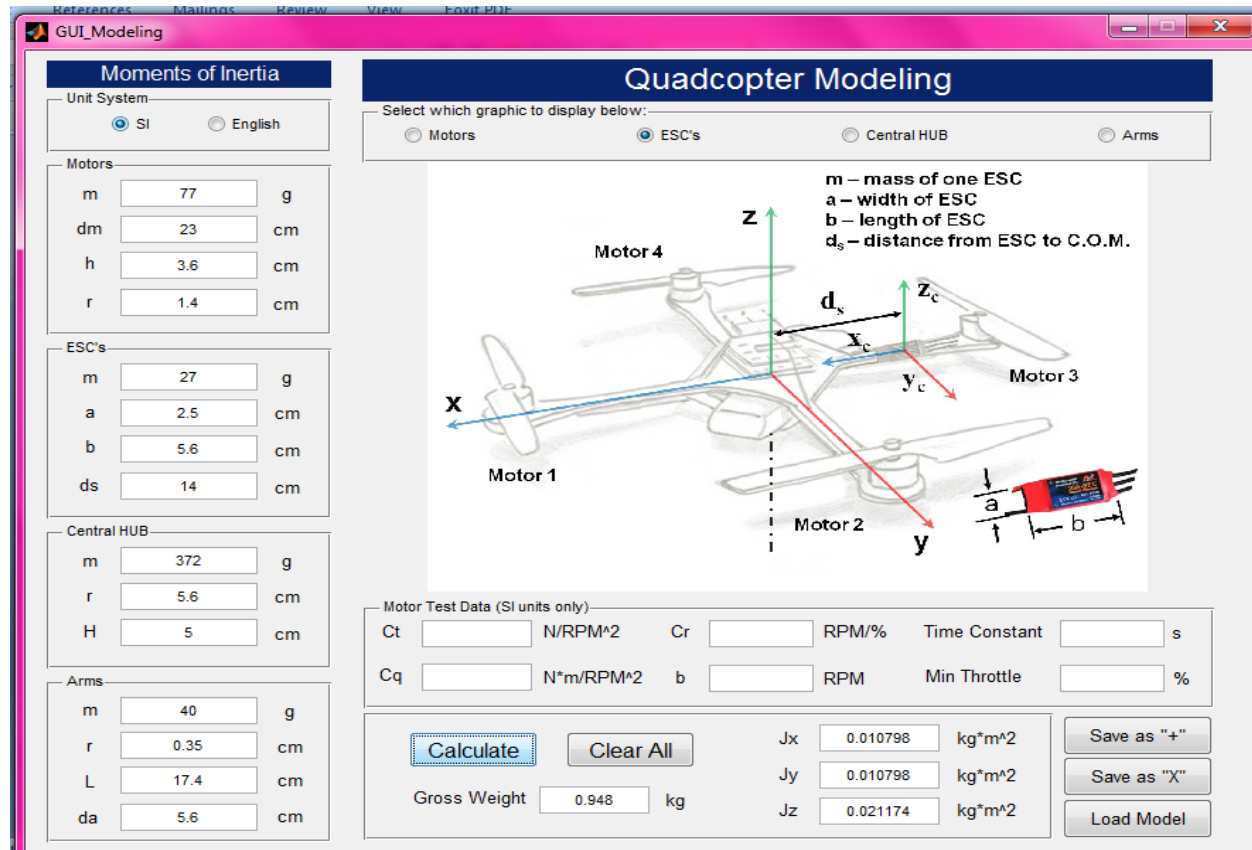


Figure B.1: Verification of Inertia Analytic Results Using Inertia Calculator

Table (B.1) summarizes the main parameters of the quadcopter:

Table B.1:Quadcopter Parameters

parameter	description	value	unit
J_{xx}	MOI of body frame around x axis	0.01122	$Kg.m^2$
J_{yy}	MOI of body frame around y axis	0.01122	$Kg.m^2$

J_{zz}	MOI of body frame around z axis	0.02199	$Kg.m^2$
l	Quadcopter arm length	0.23	m
m	Mass of quadcopter	1.04	Kg
K_T	Thrust constant	0.82×10^{-5}	$N.s^2$
K_H	Hub torque constant	1.4×10^{-7}	$N.m.s^2$
K_d	Drag constant	0.00215	$N.m^{-1}$

Appendix C

Steps for creating S-function

In order to create the S-function, templates need to be used which contain examples of implementations of the required callback methods defined by the Level-2 MATLAB S-function. To create Level-2 M-file S-function a copy of one of the templates must be edited to the desired behavior.

When to Use an S-Function:

- Creating new general purpose blocks
- Describing a system as a set of mathematical equations
- Using graphical animations

Steps in creating S-function:

1. Make a copy of the template such as `msfuntmpl_basic.m` in a new file
2. Initialize the basic S-function characteristics using the setup function (number of inputs, outputs and their properties, the number of S-function dialog parameters)
3. Register the methods that will be used, such as outputs function that is used to calculate the S-function outputs
4. Implement the equations in derivative function block
5. Use these derivatives to update the outputs

[Code]

```
% This Level-2 S-Function simulates the dynamics
```

```
% of a quadcopter based on RPM inputs
```

```
function copter_dynamics(block)
```

```
setup(block);
```

```
%endfunction
```

```
function setup(block)
```

```
% Register the number of ports.
```

```
block.NumInputPorts = 4;
```

```
block.NumOutputPorts = 12;
```

```
% Set up the port properties to be inherited or dynamic.
```

```
block.SetPreCompInpPortInfoToDynamic;
```

```
block.SetPreCompOutPortInfoToDynamic;
```

```
% Override the input port properties.
```

```
for i = 1:4;
```

```
block.InputPort(i).DatatypeID = 0; % double
```

```
block.InputPort(i).Dimensions = 1;
```

```
block.InputPort(i).DirectFeedthrough = 0;
```

```
block.InputPort(i).SamplingMode = 'Sample';
```



```

block.InputPort(1).Complexity = 'Real';

end

% Override the output port properties.

for i = 1:12;

block.OutputPort(i).DatatypeID = 0; % double

block.OutputPort(i).Dimensions = 1;

block.OutputPort(i).SamplingMode = 'Sample';

block.OutputPort(1).Complexity = 'Real';

end


% Register the parameters.

block.NumDialogPrms = 1;

%block.DialogPrmsTunable = {'Tunable','Nontunable','SimOnlyTunable'};%%change


% Set up the continuous states.

block.NumContStates = 12;


% Register the sample times.

% [0 offset] : Continuous sample time

% [positive_num offset] : Discrete sample time

%

```

```
% [-1, 0] : Inherited sample time
```

```
% [-2, 0] : Variable sample time
```

```
block.SampleTimes = [-1 0];%% changed this to Inherited sample time
```

```
function InitializeConditions(block)
```

```
%initial conditions for 12 states
```

```
model= block.DialogPrm(1).Data;
```

```
%convert to rads
```

```
R = model.R*pi/180;
```

```
Q = model.Q*pi/180;
```

```
P = model.P*pi/180;
```

```
Phi = model.Phi*pi/180;
```

```
Theta = model.Theta*pi/180;
```

```
Psi = model.Psi*pi/180;
```

```
%linear velocity
```

```
dX = model.dX;
```

```
dY =model.dY;
```

```
dZ =model.dZ;
```

```
x = model.x;
```

```

y = model.y;
z = model.z;

init_cond = [P,Q,R,Phi,Theta,Psi,dX,dY,dZ,x,y,z];

for i=1:12

block.OutputPort(i).Data = init_cond(i);
block.ContStates.Data(i) = init_cond(i);

end

function Outputs(block)

for i = 1:12;

    block.OutputPort(i).Data = block.ContStates.Data(i);

end

%endfunction

function Derivatives(block)

%% here the derivatives of the states would be calculated

%block.Derivatives.Data = 2*block.ContStates.Data;

model=block.DialogPrm(1).Data;

```

%extracting the 12 states

% first the angular velocity in body frame

P = block.ContStates.Data(1);

Q= block.ContStates.Data(2);

R= block.ContStates.Data(3);

%the angles

Phi=block.ContStates.Data(4);

Theta=block.ContStates.Data(5);

Psi=block.ContStates.Data(6);

%the linear velocity in body frame

dX=block.ContStates.Data(7);

dY=block.ContStates.Data(8);

dZ=block.ContStates.Data(9);

%the linear position

x=block.ContStates.Data(10);

y=block.ContStates.Data(11);

z=block.ContStates.Data(12);

% The rpm values from motors

w1 = block.InputPort(1).Data;

```

w2 = block.InputPort(2).Data;
w3 = block.InputPort(3).Data;
w4 = block.InputPort(4).Data;

w = [w1; w2; w3; w4];
w=w/60*2*pi; %rpm to ang_vel
% disp(w);

T=model.bt*(w.^2); %the force thrust
% disp(T);

H=model.bh*(w.^2); %the hub torque

l=model.l; %copter arm length

T_tot=sum(T); %the total thrust

H_tot=[l*(-T(2)+T(4));
        l*(-T(3)+T(1));
        (H(1)+H(3))-(H(2)+H(4))];

Jx=model.Jx;
Jy=model.Jy;
Jz=model.Jz;

%calculate angular acceleration P,Q,R
dP=(H_tot(1)-(Jz-Jy)*Q*R)/Jx;
dQ=(H_tot(2)-(Jx-Jz)*P*R)/Jy;
dR=(H_tot(3)-(Jy-Jx)*P*Q)/Jz;

```

%calculate euler angular velocities

```
Ro=[1 ,sin(Phi)*tan(Theta) ,cos(Phi)*tan(Theta);
    0, cos(Phi)      ,-sin(Phi);
    0,sin(Phi)/cos(Theta) , cos(Phi)/cos(Theta)];
Uler_W=Ro*[P;Q;R];
```

%Rotation matrix Rt=((Rx*Ry*Rz)^-1) displacement from body to inertia

```
Rt=[cos(Theta)*cos(Psi)-sin(Phi)*sin(Theta)*sin(Psi),-cos(Theta)*sin(Psi),
cos(Psi)*sin(Phi)+cos(Theta)*sin(Phi)*sin(Psi);
    cos(Phi)*sin(Psi)+cos(Psi)*sin(Phi)*sin(Theta) , cos(Phi)*cos(Psi), sin(Theta)*sin(Psi)-
sin(Phi)*cos(Theta)*cos(Psi);
    -cos(Phi)*sin(Theta) , sin(Phi) , cos(Phi)*cos(Theta) ];
```

Vb=[dX;dY;dZ]; %body frame linear velocity

```
VI=Rt*[dX;dY;dZ];
```

```
WB_cross = [ 0,-R, Q;
```

```
    R, 0,-P;
```

```
    -Q, P, 0];
```

%drag force

```
drag=[model.Kd*dX^2;model.Kd*dY^2;model.Kd*dZ^2];
```

%Body frame acceleration

```
ACC=(Rt')*[0;0;-model.g]+[0;0;T_tot]/model.m-WB_cross*Vb-drag/model.m;
```

```
%velocity in inertial frame
```

```
dX=VI(1);
```

```
dY=VI(2);
```

```
dZ=VI(3);
```

```
%acceleration in body frame
```

```
ddX=ACC(1);
```

```
ddY=ACC(2);
```

```
ddZ=ACC(3);
```

```
dPhi=Uler_W(1);
```

```
dTheta=Uler_W(2);
```

```
dPsi=Uler_W(3);
```

```
%[P,Q,R,Phi,Theta,Psi,Xd,Yd,Zd,x,y,z];
```

```
% grounding condition
```

```
if ((z<=0) && (dZ<=0))
```

```
    dZ = 0;
```

```
    block.ContStates.Data(12) = 0;
```

```
end
```

```
dstates = [dP dQ dR dPhi dTheta dPsi ddX ddY ddZ dX dY dZ].';
```

```
%This is the state derivative vector
```

block.Derivatives.Data = dstates:

Appendix D:

Components calibration

ESC Calibration:

ESC calibration is required at first time usage. Calibration means specifying the range in which the ESC expects the PWM signals. This step is only done in the first time usage and the ESC remains calibrated until the process is repeated for another specified range. Since the ESCs are meant to operate with APM board, there are two means of calibration, either all-at-once calibration which is done automatically by the APM board or manual ESC calibration which is done independently of APM board and in one-by-one manner.

The components needed for calibration are:

- Fully charged LiPo battery.
- RC receiver and Transmitter.
- Single ESC and a motor.

Calibration steps:

- The ESC control wires are connected to the throttle channel of the RC receiver.
- The ESC high voltage wires are connected to the motor.
- The transmitter is turned on and throttle stick is put at the maximum.
- The ESC power cables are connected to the LiPo battery.
- Two musical beeps are released by the ESC. After these beeps, the throttle stick is lowered to the minimum position.
- Then a number of beeps followed by a long beep are released by the ESC. The long beep indicates that the ESC is finally set to the specified range.
- The battery is disconnected and so are the ESC control wires.
- The previous steps are repeated for all other ESCs.
- Keep in mind that the RC receiver must be powered synchronously with the ESC; this can be done by utilizing the APM connection itself or by only using the regulator.

IMU Sensors Calibration:

The main IMU sensors to be calibrated are the accelerometer and internal compass. The gyroscope does not need to be calibrated.

Accelerometer Calibration:

The accelerometer requires to be calibrated initially. At the first time, the accelerometer cannot determine the orientation of the quadcopter. Fortunately, the APM board come with an arrow drawn at its upper surface which facilitates determining the mounting direction of the board as well as calibration steps. Assuming that the APM is mounted according to the arrow direction, the calibration can be done by the use of APM mission planner software. The APM is connected to the Mission planner software. This software gives the steps of calibration one by one and evaluates the correctness of calibration process. The accelerometer is found at mandatory hardware bar.

Internal Compass Calibration:

The compass calibration is necessary since the code developed relies on sensors fusion algorithm which makes use of compass and GPS for attitude and altitude determination. As the case with accelerometer, the compass requires calibration at the first time via APM mission planner software. The software provides the necessary calibration steps and validates the correctness of calibration. Similarly, the compass is found at the mandatory hardware bar.

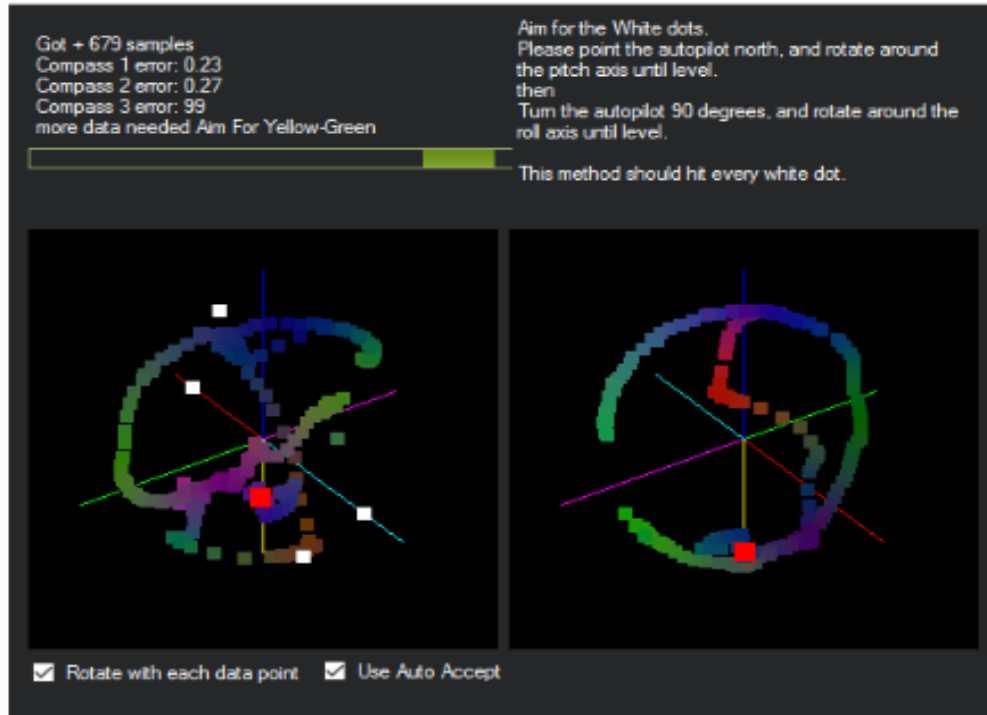


Figure D.1: Verification of Inertia Analytic Results Using Inertia Calculator

RC Setup and Calibration:

The RC transmitter and receiver must be bound together so that the receiver recognizes its own transmitter. Once binding is done, the transmitter requires applying certain setting according to the specified operation. Then the RC must be calibrated using the APM mission planner software or Arduino board.

RC Binding steps:

- The pins labeled BAT in the RC receiver are connected using the binding wire. These signals are data pin and the power pin.
- The receiver is powered using appropriate 5V power supply through any channel of the 6 channels.
- The transmitter bind button is pressed and held while turning the transmitter on.
- Once the transmitter is turned on, a message is displayed on the transmitter screen showing that the binding is done successfully.
- The flashing light of the receiver must go solid at the end of binding process.

Transmitter Setup:

The RC transmitter provides many modes of operation. Since the RC can be used to control many types of aerial vehicles, the specific type of interest must be determined via the setup option which is variable pitch helicopter in this case. The stick mode must be selected. Mode 2 provides the desired configuration according to quadcopter movement. The sticks signals can be reversed according to the desired operation. The throttle curve can be adjusted to determine the throttle values corresponding to certain stick position. The limiting ends of each stick PWM signal can be specified through end points. Many setup options can be provided but those previously mentioned are only the essentials.

RC Calibration:

As mentioned before, calibration can be done using Arduino board or APM mission planner software utilizing APM board. The APM mission planner calibration method is more reliable technique. The APM is connected to the RC receiver channels. The mission planner software provides the necessary steps for RC calibration. When doing calibration, the sticks must be brought to its maximum and minimum positions so that the end points of PWM signals are recorded. At the end of calibration the APM mission planner software provides a table of all channels end points reached. This table can be used to verify the operation of the RC.



Figure D.2: RC Calibration

Appendix E:

Motor test:

The brushless motor operation is completely controlled electronically via ESC. The ESC receives the appropriate control signals via its control wires and thus provides the pulses to motor for the desired speed. However, the motor speed is dependent on another factor which is the available power supplied by the battery, i.e. the speed of the motor at certain voltage level is not the same at another voltage level for the same control signal. This dependency on available power supply affects the expected operation of the motor under the condition of battery continuous discharging. The motor operation is described in terms of its rotational speed, the control signals are represented in term of Throttle percentage. Therefore, an average relation must be established between the input throttle and the output motor rotational speed. This relation must take into account all battery levels that maybe approached during flight time.

Components required:

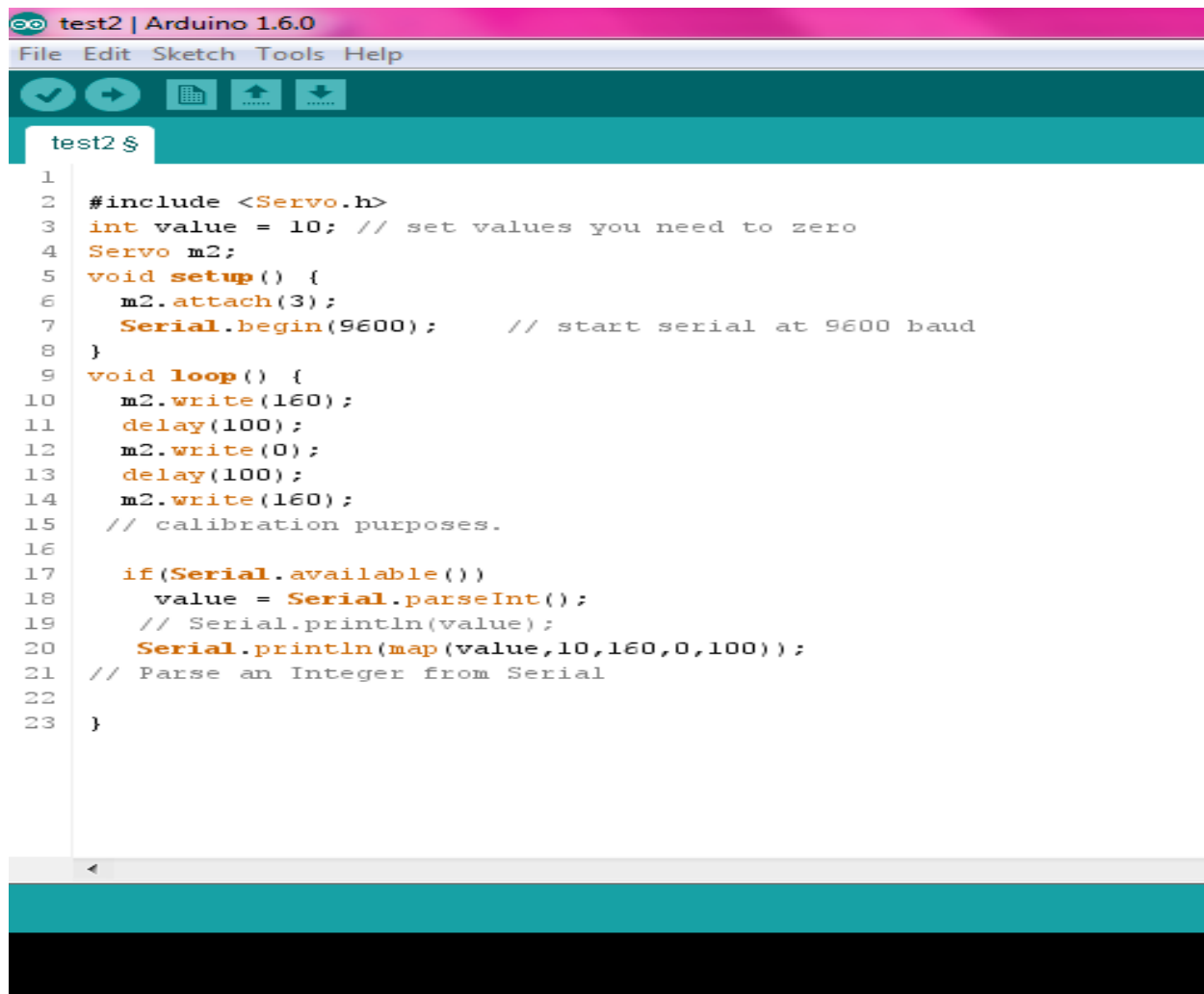
- D3542 motor mounted on a fixed arm or a test bench.
- ESC for motor control.
- LiPo battery for power supply.
- LiPo battery charger for charging and discharging purposes.
- Arduino board (Arduino Uno is suitable).
- Tachometer for rotational speed measurement.

Test Procedure:

Starting with the connection, the motor is connected to the ESC high-power terminals, the ESC power cables are connected directly to the battery and the control wires of the ESC are connected to the Arduino Uno. No need to supply power to the Arduino since it will be connected to PC via USB terminal.

The Arduino code is written to allow the tester to enter the values of the desired throttle manually via Arduino serial monitor, the Arduino testing code makes use of Servo motor library

since the Servo principle of operation relies on PWM just like the ESC. The Arduino code displays the percentage of the throttle entered by the tester in the serial monitor. The tester can then apply any throttle value at primarily specified battery voltage and then measure the RPM speed of the motor using the tachometer. Testing is performed at multiple levels of battery voltage. The data collected are scheduled and processed to give an associated relation between throttle percentage and RPM speed. The final agreed relation is found by taking the average of all the relations obtained.

The image is a screenshot of the Arduino IDE interface. At the top, the title bar reads "test2 | Arduino 1.6.0". Below it is a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". A toolbar with icons for saving, running, and other functions is visible. The main text area shows a C++ sketch for a servo motor. The code includes a header, defines a variable, and sets up a servo and serial communication. The loop function writes PWM values to the servo and prints the received throttle percentage to the serial monitor. Line numbers 1 through 23 are visible on the left side of the code editor.

```
1
2 #include <Servo.h>
3 int value = 10; // set values you need to zero
4 Servo m2;
5 void setup() {
6     m2.attach(3);
7     Serial.begin(9600);    // start serial at 9600 baud
8 }
9 void loop() {
10     m2.write(160);
11     delay(100);
12     m2.write(0);
13     delay(100);
14     m2.write(160);
15     // calibration purposes.
16
17     if(Serial.available())
18         value = Serial.parseInt();
19     // Serial.println(value);
20     Serial.println(map(value,10,160,0,100));
21     // Parse an Integer from Serial
22
23 }
```

Figure E.1: Motor Test Arduino Code

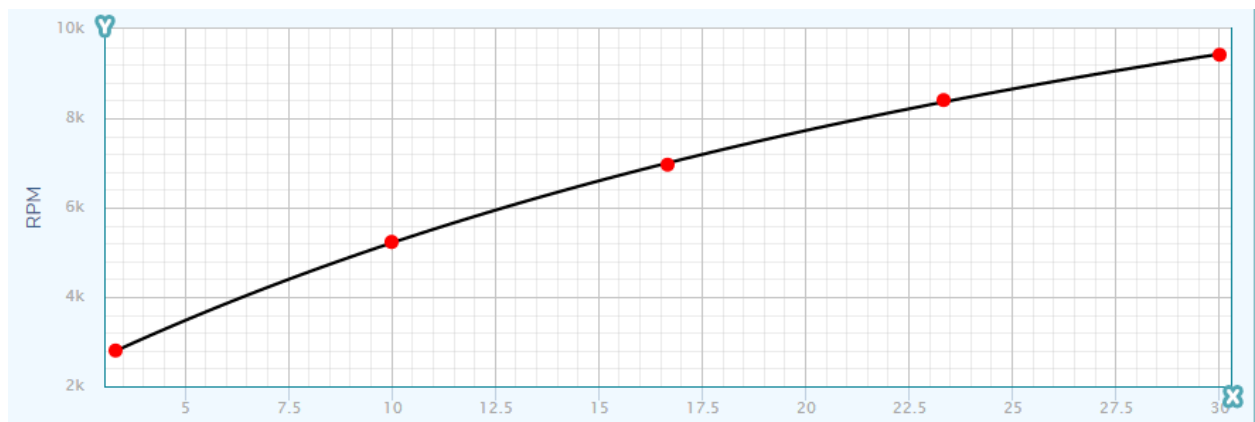
The test starts using fully charged LiPo battery and then the Battery level is discharged every time to a known level using the LiPo charger operating in discharge mode.

The Test Results and Data Collected:

Battery level 12.3:

Throttle %	Servo degree	RPM
3.33	15	2800
10	25	5231
16.66	35	6960
23.33	45	8406
30	55	9420

Point to point plot:



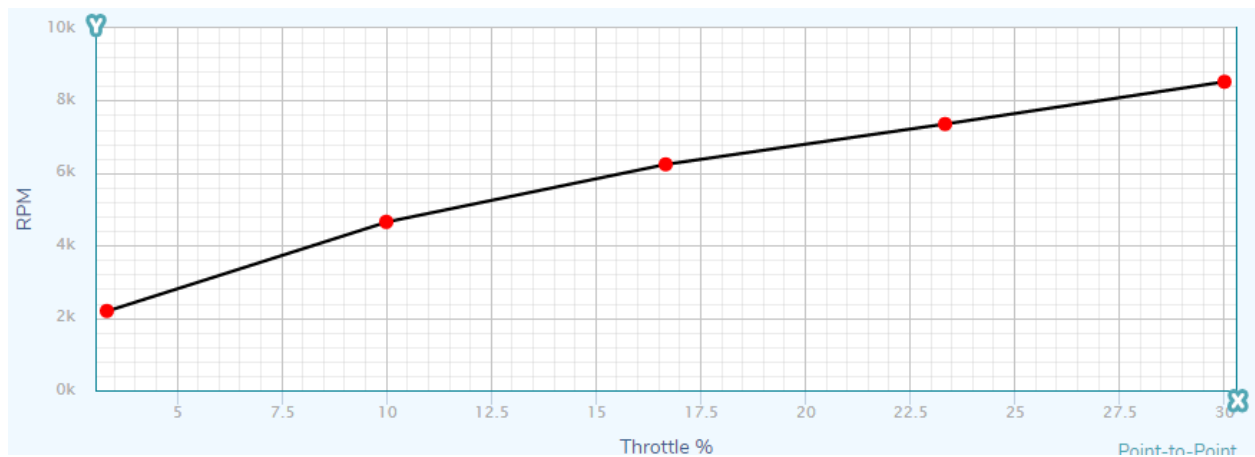
Linear approximation using curve fitting technique:



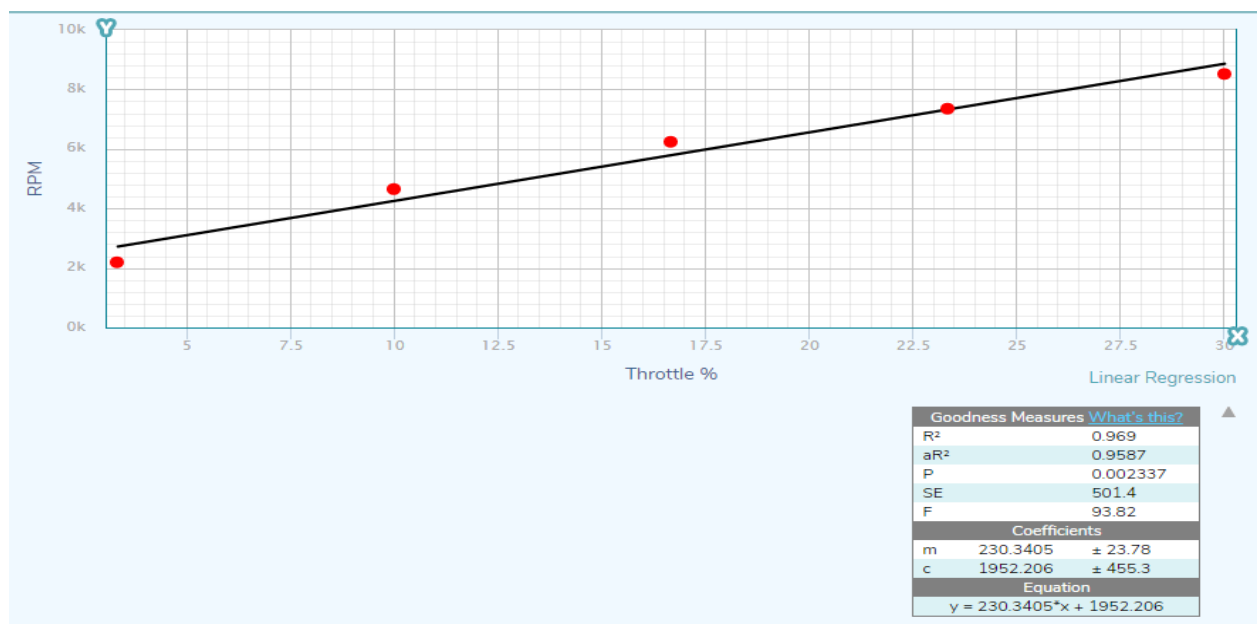
Battery level 11.3:

Throttle %	Servo degree	RPM
3.33	15	2192
10	25	4650
16.66	35	6239
23.33	45	7353
30	55	8519

Point to Point plot:



Linear approximation curve fitting technique:

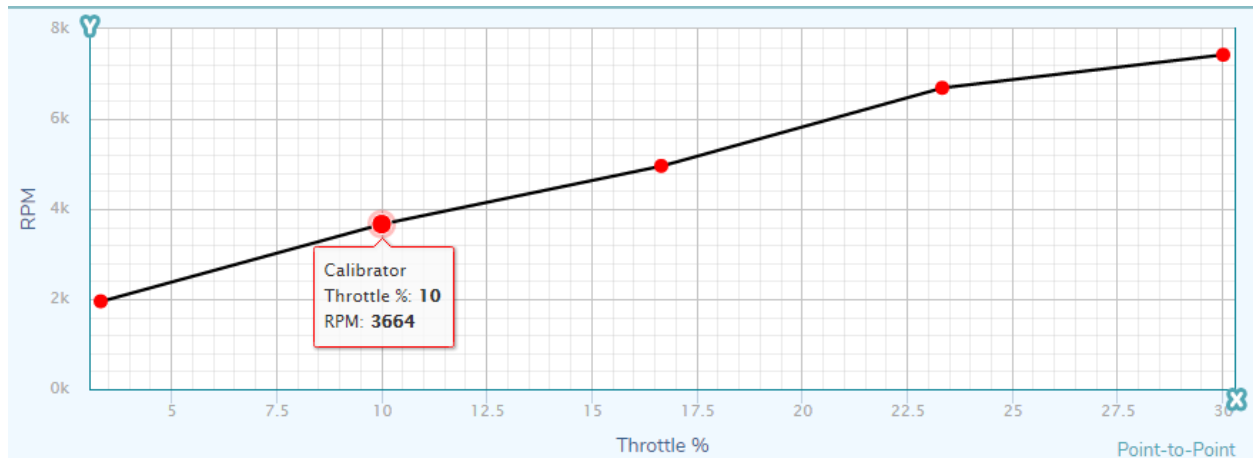


Battery level 10.3:

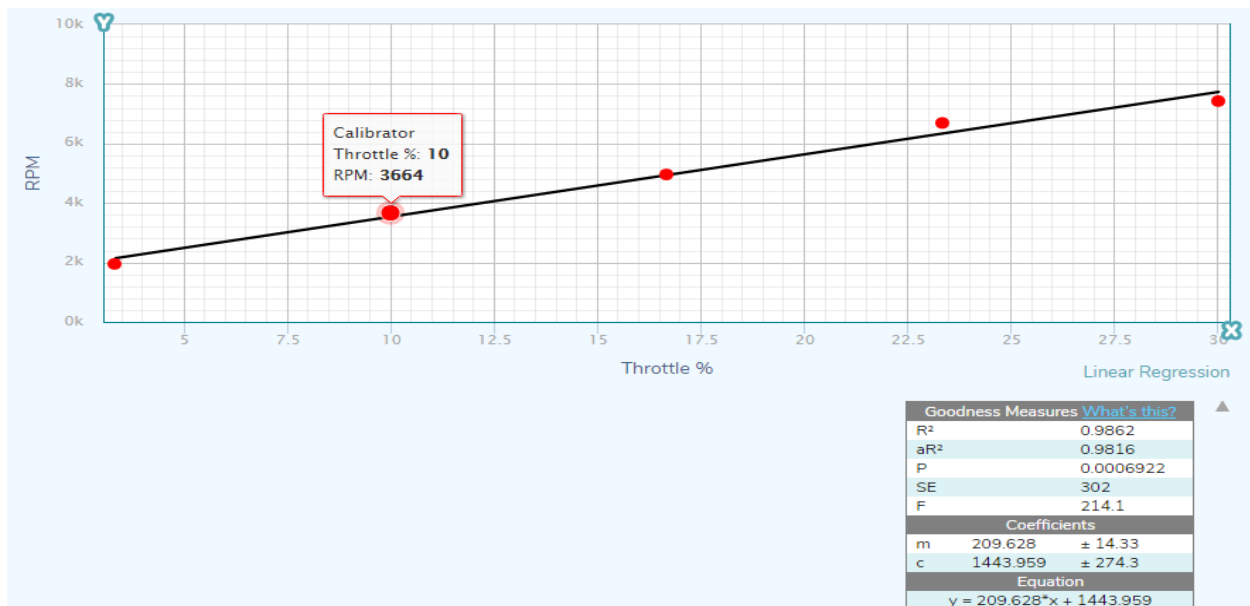
Throttle %	Servo degree	RPM
3.33	15	1950
10	25	3664
16.66	35	4957

23.33	45	6690
30	55	7425

Point to point plot:



Linear approximation curve fitting technique:



The final equation is found by taking the average of the three relations obtained. Keep in mind that (y) represents the RPM speed and (x) represents the throttle percentage

$$RPM\ Speed = 228.73 \times Throttle\% + 1952$$

Appendix F:

Autopilot Code:

```
#include <AP_Common.h>
#include <AP_Param.h>
#include <AP_Program.h>
#include <AP_HAL.h>
#include <AP_HAL_AVR.h>
#include <AP_Math.h>
#include <AP_ADC.h>
#include <StorageManager.h>
#include <AP_InertialSensor.h>
#include <AP_Notify.h>
#include <AP_GPS.h>
#include <AP_Baro.h>
#include <Filter.h>
#include <GCS_MAVLink.h>
#include <AP_AHRS.h>
#include <AP_Airspeed.h>
#include <AP_Vehicle.h>
#include <AP_Compass.h>
#include <AP_NavEKF.h>
#include <PID.h>
#include <AP_Common.h>

/* Defines */
#define MOTOR_F 2 // Front
#define MOTOR_R 0 // Right
#define MOTOR_B 3 // Back
#define MOTOR_L 1 // Left

// PID array
PID pids[6];
#define PID_PITCH_STAB 0
#define PID_ROLL_STAB 1
#define PID_YAW_STAB 2
#define PID_PITCH_RATE 3
#define PID_ROLL_RATE 4
#define PID_YAW_RATE 5

#define wrap_180(x) (x < -180 ? x+360 : (x > 180 ? x - 360 : x))
/* Variables */
const AP_HAL::HAL& hal = AP_HAL_AVR_APM2; // Hardware abstraction layer
AP_InertialSensor ins; // Generic inertial sensor
AP_InertialSensor_MPU6000 imu(ins); // MPU6000 sensor
AP_Baro_MS5611 baro(&AP_Baro_MS5611::spi); // Barometer
AP_GPS gps; // GPS
AP_Compass_HMC5843 compass;
AP_AHRS_DCM ahrs(ins, baro, gps); // AHRS DCM implementation
```

```

/* Functions */
float map(long x, long in_min, long in_max, long out_min, long out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void setup()
{
    hal._uartA->begin(115200); // for using
    hal.rcout->set_freq(0xF, 490);
    // we need to stop the barometer from holding the SPI bus
    hal.gpio->pinMode(40, HAL_GPIO_OUTPUT);
    hal.gpio->write(40, 1);
    // Enable output channels
    for (uint8_t i=0; i<8; i++) {
        hal.rcout->enable_ch(i);
    }

    // Initialize MPU6050 sensor
    ins.init(AP_InertialSensor::COLD_START, AP_InertialSensor::RATE_100HZ);

    // Initialize the accelerometer
    ins.init_accel();

    ahrs.init();
    if( compass.init() ) {
        //hal.console->printf("Enabling compass\n");
        ahrs.set_compass(&compass);
    }
    else {
        // hal.console->printf("No compass detected\n");
    }

    //gps.init(NULL);

    pids[PID_PITCH_STAB].kP(4);
    pids[PID_PITCH_STAB].kI(0);
    pids[PID_PITCH_STAB].kD(0);
    pids[PID_PITCH_STAB].imax(8);

    pids[PID_ROLL_STAB].kP(4);
    pids[PID_ROLL_STAB].kI(0);
    pids[PID_ROLL_STAB].kD(0);
    pids[PID_ROLL_STAB].imax(8);

    pids[PID_YAW_STAB].kP(3);
    pids[PID_YAW_STAB].kI(0);
    pids[PID_YAW_STAB].kD(0);
    pids[PID_YAW_STAB].imax(8);
}

```

```

static float yaw_target = 0;
float pitch_stab_output;
float roll_stab_output;
float yaw_stab_output;
float pitch_output;
float roll_output;
float yaw_output;
uint32_t now = hal.scheduler->micros();
if (last_print == 0) {
    last_print = now;
    return;
}
// Read RC channels and store in channels array
hal.rcin->read(channels, 8);

// Copy from channels array to something human readable - array entry 0 = inr
rcthr = channels[2];
rcyaw = -map(channels[3], 1001, 2003, -50, 50);
rcpit = -map(channels[1], 1001, 2001, -25, 25);
rcroll = -map(channels[0], 1001, 2001, -25, 25);

// Wait for a new measurement
ins.wait_for_sample();

// Read the gyroscope (deg/s)
ins.update();
gyro = ins.get_gyro();
gyropitch = ToDeg(gyro.y);
.....

pids[PID_PITCH_RATE].kP(1.657);
pids[PID_PITCH_RATE].kI(0.06616);
pids[PID_PITCH_RATE].kD(0.03016);
pids[PID_PITCH_RATE].imax(50);

pids[PID_ROLL_RATE].kP(1.657);
pids[PID_ROLL_RATE].kI(0.06616);
pids[PID_ROLL_RATE].kD(0.03016);
pids[PID_ROLL_RATE].imax(50);

pids[PID_YAW_RATE].kP(8.308);
pids[PID_YAW_RATE].kI(3);
pids[PID_YAW_RATE].kD(0);
pids[PID_YAW_RATE].imax(50);
}

void loop()
{
    uint16_t channels[8]; // array for raw channel values
    float rcthr, rcyaw, rcpit, rcroll; // Variables to store rc input

    Vector3f gyro;
    float gyroroll, gyropitch, gyroyaw; // Velocity in deg/s
    float roll, pitch, yaw; // Angle in degrees
    static uint16_t counter;
    static uint32_t last_print, last_compass;
    .....

```

```

gyropitch = ToDeg(gyro.y);
gyroroll = ToDeg(gyro.x);
gyroyaw = ToDeg(gyro.z);

// Read the AHRS to get attitude (deg)
ahrs.update();
roll = ToDeg(ahrs.roll);
pitch = ToDeg(ahrs.pitch);
yaw = ToDeg(ahrs.yaw);
if(rcthr > 1000 + 100){

    // Stabilise PIDS
    pitch_stab_output = constrain_float(pids[PID_PITCH_STAB].get_pid((float)rcpit - pitch, 1), -250, 250);
    roll_stab_output = constrain_float(pids[PID_ROLL_STAB].get_pid((float)rcroll - roll, 1), -250, 250);
    yaw_stab_output = constrain_float(pids[PID_YAW_STAB].get_pid(wrap_180(yaw_target - yaw), 1), -360, 360);

    //Yaw input is rate control, check if pilot is changing yaw rate
    if(abs(rcyaw) > 5) {
        yaw_stab_output = rcyaw;
        yaw_target = rcyaw; // remember this yaw for when pilot stops
    }

    // rate PIDS
    pitch_output = constrain_float(pids[PID_PITCH_RATE].get_pid(pitch_stab_output - gyropitch, 1), -500, 500);
    roll_output = constrain_float(pids[PID_ROLL_RATE].get_pid(roll_stab_output - gyroroll, 1), -500, 500);
    yaw_output = -constrain_float(pids[PID_YAW_RATE].get_pid(yaw_stab_output - gyroyaw, 1), -500, 500);

    // mix pid outputs and send to the motors.

    hal.rcout->write(MOTOR_F, rcthr + pitch_output + yaw_output);
    hal.rcout->write(MOTOR_R, rcthr - roll_output - yaw_output);
    hal.rcout->write(MOTOR_B, rcthr - pitch_output + yaw_output);
    hal.rcout->write(MOTOR_L, rcthr + roll_output - yaw_output);
}
else {
    // motors off
    hal.rcout->write(MOTOR_F, 1000);
    hal.rcout->write(MOTOR_R, 1000);
    hal.rcout->write(MOTOR_B, 1000);
    hal.rcout->write(MOTOR_L, 1000);

    yaw_target = yaw;

    for(int i=0; i<6; i++) // reset PID integrals whilst on the ground
        pids[i].reset_I();
}

// Print to serial monitor at 10Hz
counter++;

if (now - last_print >= 100000 /* 100ms : 10hz */) {

    hal._uartA->printf_P(PSTR("%.2f %.2f %.2f \n "),roll,pitch, yaw);

    last_print = now;
    counter = 0;
}
}

AP_HAL_MAIN(); // special macro that replace's one of Arduino's to setup

```