People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

MASTER

In **Control**

Option: **Control**

Title:

Trajectory Tracking Based On Sliding Mode Control For a Quadrotor

Presented by:

- **BOUANZOUL Mahdi**
- **AOUATI Ayoub**

Supervisor:

Dr. BOUSHAKI Razika

Registration Number:……/2019

Abstract

In order to solve the problem of precise trajectory tracking control for a quadrotor in the presence of external disturbance and system model parameter uncertainty, a nonlinear trajectory tracking controller based on sliding mode for the quadrotor is designed.

The dynamic model is used to design a stable and accurate controller to perform the best tracking and attitude results.

The robustness and effectiveness of the proposed control strategy is verified by simulation in a virtual environment for linear and nonlinear trajectories.

Dedication

I dedicate this work:

To my dear parents for their devotion, and for their endless support.

This is for you, Khaoula, Yacine and Anas. You are so important in my life.

To Mimi and Djedou. Thanks for always being there for me.

In memory of my grandmother Cherifa. You shan't be forgotten.

To all my family and friends.

AOUATI Ayoub

I would like to dedicate this work:
To my father and my mother for their support
To my brothers and my sister
To all my family and friends

BOUANZOUL Mahdi

Acknowledgement

First and foremost, praises and thanks to Allah, the Almighty, for his conciliation in the completion of this humble work.

We are sincerely obliged to many worthy persons, without whom we could have never had the opportunity of learning greatly with interest.

Special gratitude goes to our enthusiastic and amazing supervisor, Dr. R. BOUSHAKI whose guidance has been present every step of our final project study.

We also would like to extend our most profound appreciations to the jury members for offering us time in reading our project report.

Last but by no means least, thanks go to our parents for their love, prayers, caring and sacrifices for educating and preparing us for the future. They are the most important people in our world and we dedicate this work to them. Also, we express our thanks to our sisters, brothers, grandparents and all our families for their support and valuable prayers.

Table of Content

List of Figures

List of Tables

Nomenclatures

BDC	Brushed Direct Current.
BEC	Battery Eliminator Circuit.
BLDC	Brushless Direct Current
CCW	Counter Clock Wise.
CW	Clock Wise.
DC	Direct Current.
DOF	Degree Of Freedom.
ESC	Electronic Speed Controller.
GPS	Global Positioning System.
HALE	Hight Altitude Long -Endurance.
LIPO	Lithium Polymer Batteries.
MALE	Medium-Altitude Long-Enduration.
MAV	Micro UAV.
MUAV	Mini UAV.
NAV	Nano UAV.
PD	Propertional Derivative Controller.
PWM	Pulse Width Modulation.
SMC	Sliding Mode Control.
TUAV	Medium-Range Or Tactical Uav.
VTOL	Vertical Take-Off And Landing

List of symbols

B	Body frame
b	Thrust factor.
d	Drag factor
E	Fixed frame
f	Thrust forces
g	Gravity acceleration
I	Inertia matrix
I_{rotor}	Rotor inertia moment
l	distance between gravity center of the quadrotor and the motor rotation axis
$(x, y, z, \varphi, \theta, \psi)$	Generalizd coordinates system of the quadrotor
$R(\phi, \theta, \psi)$	Rotation matrix
U	The control of signals
τ	Total torque
τ_1	Roll,pitch and yaw torque
τ_2	Gyroscopic effect
θ	Pitch angle
φ	Roll angle
ψ	Yaw angle

Introduction

The last decades have seen a tremendous progress in the development of unmanned aerial vehicles (UAVs). A growing number of research institutes, universities, governments, and commercial entities across the world are developing and employing UAVs for a diverse range of applications, such as aerial photogrammetry [1], agriculture [2], and military missions [3]. One type of aerial vehicle which can accomplish this, is a quadrotor. The quadrotor is an UAV with four rotating blades which enable flight in a similar way to that of a helicopter. Movement is attained by varying the speeds of each blade thereby creating different thrust forces.

In order to accomplish the above-mentioned missions without constant supervision of human operators, the UAV must autonomously follow predefined paths in 2D or 3D space. Usually, the problems of motion control for a single autonomous vehicle are roughly classified into three groups. Namely, point stabilization, where the goal is to stabilize a vehicle about a given target point with a desired orientation; trajectory tracking, where the vehicle is required to track a time parametrized reference; and path-following, where the quadrotor is required to follow a desired geometric path, implying a constraint in space, but not in time. Thus, the time takes the quadrotor to reach the target position does not matter here [4].

Design and analysis of control systems are usually started by carefully considering mathematical models of physical systems. In principle, a quadrotor is dynamically unstable and therefore proper control is necessary to make it stable. However, quadcopter control is a fundamentally difficult and interesting problem. Therefore, several control algorithms have been applied to it. Andrew and Samuel present a review of control algorithms for autonomous quadrotor with a comparison between them [5].

Thus, this project focuses on the quadrotor modeling and the controller designing in order to achieve higher trajectory-tracking accuracy.

In this study, a complete dynamical model of the studied Quadrotor UAV is established using the Newton-Euler formalism that is more realistic as it introduces variable aerodynamics coefficients [6]. Since the quadrotor is a nonlinear type system, sliding mode controller (SMC) is proposed to control the states of the quadrotor. SMC is a robust nonlinear control algorithm that has been used to implement tracking controllers for unmanned aircraft systems that are robust to modeling uncertainty and exogenous disturbances, thereby providing excellent performance for autonomous operation [7]. In

addition, Proportional Derivative (PD) controller was illustrated and simulated for comparison reason and to proof the efficiency of the proposed controller.

In practice however, sliding mode controllers induce chatter in the system, degrading the system's performance and damaging its physical components. Therefore, - in this study- it is imperative to avoid the control chattering by providing continuous and smooth control signals.

The performance of the overall system was tested in a numerical simulation. The first applied simulation being without controllers to show the evidence of using control algorithms for stabilization and control objectives. Whereas, the last applied simulations done with existence of SMC and PD controllers to monitor the system in a closed loop.

Finally, the combination of the obtained results and components description, are used in the hardware and software implementation of the autonomous quadcopter tracking system.

.

1 Quadrotor Generalities

Aerial manipulation has been an active area of research in recent years, mainly because the active tasking of Unmanned Aerial Vehicles (UAV) increases the employability of these vehicles for various applications. This area of research opens a new challenge in a diverse range of fields, such as aeronautics, instrumentation, robotics and control, since they are not only remote-controlled machines, but also real autonomous systems which can interact with the environment. The recent development of the aerial manipulation has found potential applications in both military and civilian domains. Military applications include border patrolling, mine detection, reconnaissance, etc., while civilian applications are in disaster management, bridge inspection, construction, material delivery, search and rescue [8].

1.1 Problem Statement

This project deals with VTOL aircrafts, where the four-rotor helicopter, quadcopter or quadrotor is mainly studied. Such a vehicle is an under-actuated mechanical system that has six Degrees of Freedom (DOF) but only four control inputs namely roll, pitch, yaw and thrust. The vehicle has the capacity to take-off and land in vertical position.

1.2 Objectives

This project focuses on design and control of unmanned, autonomous fly helicopters with application to it. The contribution of this work lies in three fields.

- Dynamic modelling of quadrotors: the goal is to obtain a faithful mathematical representation of the mechanical system for system analysis and control design.
- System control: the aim is to understand and then master the dynamics of quadrotors by applying the appropriate control techniques.
- System implementation: realize the quadrotor in real environment.

1.3 Components of Autonomous Flight

In order to enable autonomous flight, there are a few components/modules that we need in any UAV. They are listed as below:

- Sensors: indicate the position and velocity (including rotation and angular velocity of the robot).
- Controller: Command motors and produce desired actions in order to navigate to the desired state. Therefore, the vehicle must be able to follow a given trajectory.
- Actuators: devices that accept control command (mostly in the form of an electrical signal) and produce a change in the physical system by generating force, motion and so forth [9].

1.4 UAVs Classification

There are different ways to classify UAVs, either according to their range of action, aerodynamic configuration, size and payload or according to their levels of autonomy.

1.4.1 Range of Action Classification

UAVs can be classified into seven different categories based on their maximum altitude and endurance as follows [10]:

1.4.1.1 High-Altitude Long-Endurance (HALE)

They can fly over 15000 m high with an endurance of more than 24 hr. They are mainly used for long-range surveillance missions.

1.4.1.2 Medium-Altitude Long-Endurance (MALE)

They can fly between 5000-15000 m of altitude for a maximum of 24 hr. MALE UAVs are also used for surveillance missions**.**

1.4.1.3 Medium-Range or Tactical UAV (TUAV)

They can fly between 100 and 300 km of altitude. They are smaller and operated with simpler systems that their HALE and MALE counterparts.

1.4.1.4 Close Range UAV

They have an operation range of 100 km. They are mainly used in the civil application such as power-line inspection, crop-spraying, traffic monitoring, homeland security.

1.4.1.5 Mini UAV (MUAV)

They have a weight of about 20 kg and an operating range of about 30 km.

1.4.1.6 Micro UAV (MAV)

They have a maximum wingspan of 150 mm. They are mainly used indoors where they are required to fly slowly and hover

1.4.1.7 Nano Air Vehicles (NAV)

They have a small size of about 10 mm. they are mainly used in swarms for applications such as radar confusion. They are also used for short range surveillance if equipped with an equally small camera

1.4.2 Aerodynamic Configuration Classification

UAVs can be classified into four main categories based on their aerodynamic configuration as follows

1.4.2.1 Fixed-wing UAVs

Require a run-way to take-off and land. They can fly for a long time and at high cruising speeds. They are mainly used in scientific applications such as meteorological reconnaissance and environmental monitoring.

1.4.2.2 Rotary-wing UAVs

They can take off and land vertically. They can also hover and fly with high maneuverability. The Rotary-wing UAVs can be further classified into four groups []:

- **Single-rotor**

They have a main rotor on top and another rotor at the tail for stability, same like the helicopter configuration.

- **Coaxial**

They have two rotors rotating in opposite directions mounted to the same shaft in Figure 1-4(b).

- **Quadrotor**

They have four rotors fitted in a cross-like configuration. Shown in Figure 1 (c).

- **Multi-rotor**

UAVs with six or eight rotors. They are agile type and fly even when a motor fails, as there is redundancy due to the large number of rotors. Shown in Figure 1.1 (d).

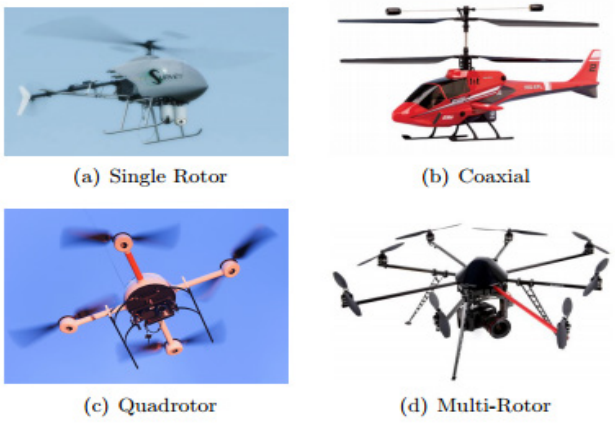


Figure 1.1: Rotary-Wing UAVs [10].

1.5 Quadrotor Application

Quadrotors are being increasingly used in Precision Farming, Construction, Archeology, Photography, Robotic First Responders during Emergency. In precision farming, robots are being used to patrol orchards to do a visual survey in infrared spectrum to assess the quality of yield. In construction industry, quadrotors are being used for inspection and the check the progress of the work. In archaeological sites, these robots are used to inspect the stability of old building that were built thousands of years ago.

Quadrotors have the potential to be used as first responds in case of a calamity or emergency. They can go to places where rescue workers cannot go easily and provide information to the authorities to plan rescue missions. Quadrotors are also being used alongside manipulators to perform cooperative mobile manipulation tasks.

1.6 Outline

The rest of this project is organized as follows:

- **Chapter 2**: This chapter presents the mathematical model of a quadrotor UAV which has been derived based on the Newton-Euler laws. The developed model of the quadrotor is used in the proposed control strategies to control the attitude, altitude and position of the quadrotor in space.
- **Chapter 3:** This chapter presents, the performance of overall system has been tested by simulation in MATLAB/Simulink.
- **Chapter 4:** Finally, in this chapter, the obtained results are used in the implementation of the autonomous quadrotor system.

2 Modeling and Controller Design

2.1 Introduction

Mathematical modeling of an unmanned aerial vehicle, specifically, quadrotor modeling is not an easy task because of its complex structure, nonlinear dynamics and under-actuated nature. In this chapter a dynamic model of a quadrotor has been developed using law of physics and mathematics manipulations. The aim is to model a quadrotor vehicle as realistic as possible. The model is then used to design a SMC and PD controllers' structure to stabilize the roll, pitch, yaw angles and linear position of the quadrotor system.

2.1 Basic Concepts

A quadrotor, also referred to as a quadcopter or simply a drone, is a Vertical Take-Off and Landing (VTOL) Unmanned Aerial Vehicle (UAV) with hovering capability, high maneuverability and agility. It consists of four rotors in cross configuration. The quadrotor movement is controlled by varying the angular velocity of each rotor individually, thereby changing forces, torques and moments can be generated on the body.

The type of rotorcraft used in this thesis is a quad-copter in X configuration. The four rotors spin clockwise and counter-clockwise like it is shown in Figure 2.1. Clockwise (CW) motors, shown in blue, use normal propellers and counter-clockwise (CCW) motors, shown in red, use pusher propellers.

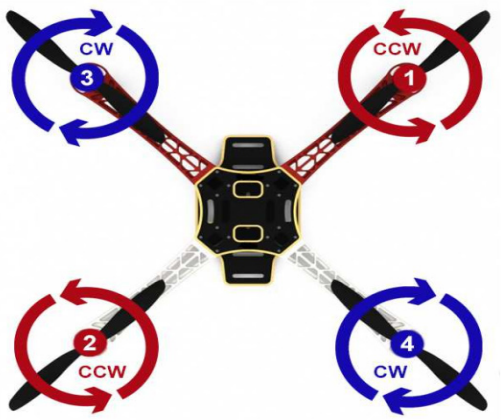


Figure 2.1: Quad-rotor in X configuration.

2.1.1 Reference Frames

The absolute position and orientation of a quadrotor is described with the use of two reference frames (also referred as coordinate systems):

- Earth inertial frame (E-frame).
- Body-fixed frame (B-frame).

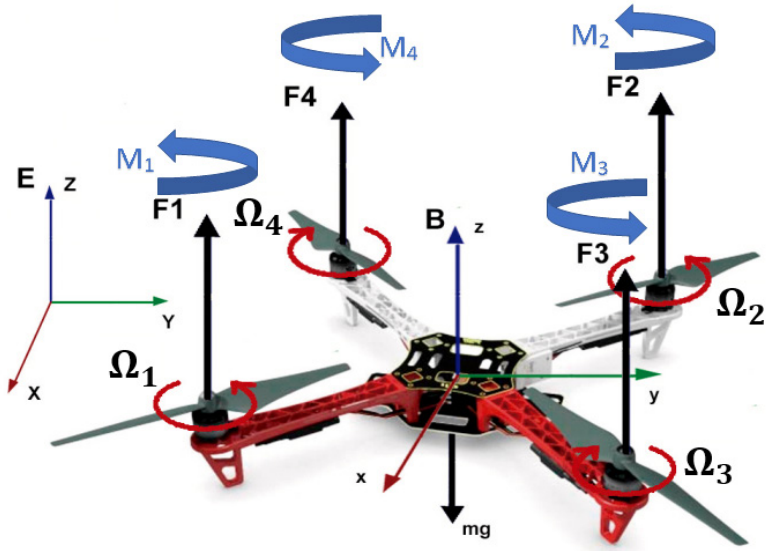


Figure 2.2: Quad-rotor frames of reference.

The axes of the E-frame coordinate system are aligned with North x_E, East y_E and up z_E. The B-frame coordinate system is fixed to the center of gravity (COG) of the quadrotor with x_b in the forward direction, as seen in Figure 2.2.

The position of the quadrotor is described with $\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$, which is the position vector of the center of gravity of the quadrotor, relative to the Earth reference frame E.

The orientation of the quadrotor is described using Tait-Bryan angles with $\eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \in \mathbb{R}^3$, which is the angular position vector of the center of gravity of the quadrotor, relative to the Body reference frame B, where ϕ , θ and ψ are the roll, pitch and yaw respectively, often referred to as Euler angles.

The generalized coordinates of the quadrotor are represented with the vector q that consists the linear and angular position vectors: $q = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \in \mathbb{R}^6$.

2.1.2 Quadrotor Movements

The quadrotor is a 6 DOF object, thus 6 variables are used to express its position in space (x, y, z, ϕ, θ and ψ). x, y, and z represent the distances of the quadrotor's center of mass along the x, y, and z axes respectively from an Earth fixed inertial frame. ϕ, θ and ψ are the three Euler angles representing the orientation of the quadrotor. ϕ is called the roll angle which is the angle about the x-axis, θ is the pitch angle about the y-axis, while ψ is the yaw angle about the z-axis.

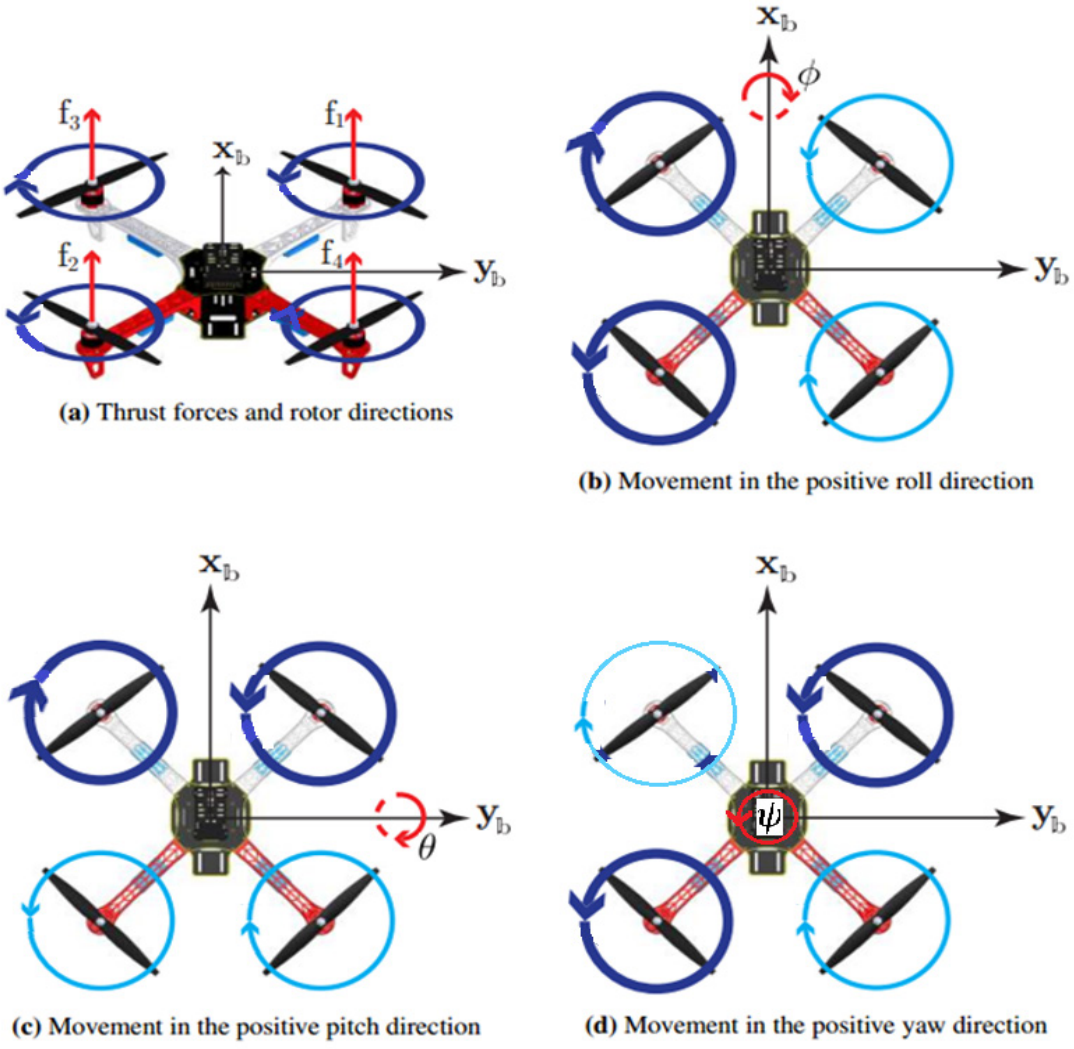


Figure 2.3: Basic quad rotor movements.

Figure 2.3a shows the quadrotor with equal speed on all rotors, resulting in equal forces from each rotor. This allows the quadrotor to move along the z_b axis by increasing or decreasing the rotor speed. With a certain rotor speed the quadrotor will obtain a hovering state.

Figure 2.3b and 2.3c show how the roll ϕ and pitch θ angles are controlled. In both cases two adjacent rotors have increased speed compared to the other rotors, resulting in a movement around the x_b and y_b axes respectively.

Each rotor affects the quadrotor body with a torque in the opposite direction of the rotation of the rotor. These torques are canceled when two rotors rotate clockwise and two rotors rotate counter clockwise with the same speed. The torque from the rotors can be exploited to control the yaw angle ψ, as illustrated in Figure 2.3d, where two diagonal rotors rotate faster than the other rotors. This result is a movement around the z_b axis, in this case in the positive yaw direction.

2.2 Modeling Design

The kinematics and dynamics models of a quadrotor will be derived based on a Newton-Euler formalism [6], [11], [12], [13]. Before analyze the dynamic of the quadrotor, serval assumptions are made:

- The structure is rigid and symmetrical.
- The propellers are rigid.
- The COG of the quadrotor coincides with the body fixed frame origin. Therefore, $I_{xx} = I_{yy}$ and $I_{xy} = I_{yz} = I_{zx} = 0$.
- Thrust and drag are proportional to the square of propeller's speed.

2.2.1 Kinematic Model

Based on the previous subsections, the three single rotations are described separately by:

- $R(x, \phi)$ rotation around x-axis.
- $R(y, \theta)$ rotation around y-axis.
- $R(z, \psi)$ rotation around z-axis.

They are represented by:

$$R(x, \phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\ \phi & -sin\ \phi \\ 0 & sin\ \phi & cos\ \phi \end{bmatrix} \quad (2.1)$$

$$R(y,\theta) = \begin{bmatrix} cos\,\theta & 0 & sin\,\theta \\ 0 & 1 & 0 \\ -sin\,\theta & 0 & cos\,\theta \end{bmatrix} \tag{2.2}$$

$$R(z,\psi) = \begin{bmatrix} cos\,\psi & -sin\,\psi & 0 \\ sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The complete rotation matrix -or transformation from the Body frame to the Earth frame- is the product of the previous three successive rotations:

$$R(\phi,\theta,\psi) = R(x,\phi)R(y,\theta)R(z,\psi)$$

Which results in:

$$R = \begin{bmatrix} c\psi\, c\theta & c\psi\, s\theta\, s\phi - s\psi\, c\phi & c\psi\, s\theta\, c\phi + s\psi\, s\phi \\ s\psi\, c\theta & s\psi\, s\theta\, s\phi + c\psi\, c\phi & s\psi\, s\theta\, c\phi - s\phi s\psi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{2.4}$$

Where: $s = sin$ and $c = cos$.

The rotation matrix R will be used in formulating the dynamics model of the quadrotor, its significance is due to the fact that some states are measured in the body frame (e.g. the thrust forces produced by the propellers) while some others are measured in the inertial frame (e.g. the gravitational forces and the quadrotor's position). Thus, to have a relation between both types of states, a transformation from one frame to the other is needed. Note that, the above rotation matrix is an orthonormal matrix. Then the transformation to obtain the Body frame quantities can be obtained by taking the transpose of the rotation matrix R^T.

To acquire information about the angular velocity of the quadrotor, typically an on-board Inertial Measurement Unit (IMU) is used which will in turn give the velocity in the body coordinate frame. To relate the Euler rates $\dot{\eta} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$ that are measured in the inertial frame and angular body rates $\omega = [p \quad q \quad r]^T$, a transformation is derived as follows:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R(\dot{x},\dot{\phi}) \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R(x,\phi)R(\dot{y},\dot{\theta}) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathrm{R(x,\phi)R(y,\theta)R}(\dot{z},\dot{\psi}) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{2.5}$$

Note that $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$ are small thus $R(\dot{x},\dot{\phi}) = R(\dot{y},\dot{\theta}) = \mathrm{R}(\dot{z},\dot{\psi}) = I$, then

$$\omega = R_r\dot{\eta} \tag{2.6}$$

Where:

$$R_r = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \quad (2.7)$$

Around the hover position, small angle assumption is made where $cos\,\phi \equiv 1$, $cos\,\theta \equiv 1$, and $sin\,\phi = 0$, $sin\,\theta = 0$. Then, one can write $[\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T \approx [p \quad q \quad r]^T$. Thus R_r can be simplified to an identity matrix I.

2.2.2 Dynamic Model

The motion of the quadrotor can be divided into two subsystems; rotational subsystem (roll ϕ ,pitch θ and yaw ψ angles) and translational subsystem (altitude z, and x and y position). The rotational subsystem is fully actuated while the translational subsystem is underactuated.

2.2.2.1 Translational Motion

The translation equations of motion for the quadrotor are based on Newton's second law and they are derived in the Earth inertial frame.

$$m\ddot{\xi} = P + R(\phi, \theta, \psi)F_t \quad (2.8)$$

Where:

$\xi = [x \quad y \quad z]^T$ Quadrotor's distance from the inertial frame.

$P = [0 \quad 0 \quad -mg]^T$ The gravitational force acting on the quadrotor in the body frame.

m Quadrotor's mass.

g Gravitational acceleration $g = 9.81\ m/s^2$.

F_t Nongravitational forces acting on the quadrotor in the body frame.

When the quadrotor is in a horizontal orientation (i.e. it is not rolling or pitching), the only nongravitational forces acting on it is the thrust F_i produced by the rotation of the propellers which is proportional to the square of the angular velocity Ω_i^2 of the propeller as $F_i = b\Omega_i^2$. Thus, the nongravitational forces acting on the quadrotor, F_t, can be expressed as,

$$F_t = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} F_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix} \tag{2.9}$$

The first two rows of the force vector are zeros as there is no forces in the x and y directions, the last row is simply an addition of the thrust forces produced by the four propellers.

F_t is multiplied by the rotation matrix $R(\phi, \theta, \psi)$ to transform the thrust forces of the rotors from the body frame to the inertial frame, so that the equation can be applied in any orientation of the quadrotor.

Noting $U_1 = \sum_{i=1}^{4} F_i$, the translational motion equations are given:

$$\begin{cases} \ddot{x} = (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\dfrac{U_1}{m} \\ \ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\dfrac{U_1}{m} \\ \ddot{z} = -g + (\cos\phi \cos\theta)\dfrac{U_1}{m} \end{cases} \tag{2.10}$$

2.2.2.2 Rotational Motion

The rotational equations of motion are derived in the body frame using the Newton-Euler method with the following general formalism.

$$\tau = I\ddot{\eta} + \dot{\eta} \times I\eta \tag{2.11}$$

Where:

τ The total torque.

I Quadrotor's diagonal inertia Matrix.

η The angular position vector of the center of gravity of the quadrotor.

The first two terms of the above equation, $I\ddot{\eta}$ and $\dot{\eta} \times I\dot{\eta}$, represent the rate of change of angular momentum in the body frame.

The inertia matrix for the quadrotor is a diagonal matrix, the off-diagonal elements, which are the product of inertia, are zero due to the symmetry of the quadrotor.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.12}$$

Where I_{xx}, I_{yy} and I_{zz} are the area moments of inertia about the principle axes in the body frame.

By rewriting equation (2.11),

$$\tau = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} I_{xx}\dot{\phi} \\ I_{yy}\dot{\theta} \\ I_{zz}\dot{\psi} \end{bmatrix}$$

Leads to,

$$\tau = \begin{bmatrix} I_{xx}\ddot{\phi} \\ I_{yy}\ddot{\theta} \\ I_{zz}\ddot{\psi} \end{bmatrix} + \begin{bmatrix} (I_{zz} - I_{yy})\dot{\theta}\dot{\psi} \\ (I_{xx} - I_{zz})\dot{\phi}\dot{\psi} \\ (I_{yy} - I_{xx})\dot{\phi}\dot{\theta} \end{bmatrix} \tag{2.13}$$

The total torques can be written as $\tau = \tau_1 + \tau_2$, where τ_1 is Moments acting on the quadrotor in the body frame, and τ_2 is Gyroscopic moments due to rotors' inertia.

For τ_1, there is a need to define two physical effects which are the aerodynamic forces and moments produced by a rotor. As an effect of rotation, there is a generated force called the aerodynamic force or the lift force and there is a generated moment called the aerodynamic moment. The aerodynamic force F_i and moment M_i produced by the ith rotor can be expressed in simplified way as:

$$F_i = b\Omega_i^2 \tag{2.14}$$

$$M_i = d\Omega_i^2 \tag{2.15}$$

Where b and d are the aerodynamic force and moment constants respectively and Ω_i is the angular velocity of rotor *i*. The aerodynamic force and moment constants can be determined experimentally for each propeller type.

Each rotor causes an upwards thrust force F_i and generates a moment M_i with direction opposite to the direction of rotation of the corresponding rotor *i,* Figure (2.2).

The total moment τ_ϕ, τ_θ, and τ_ψ about the x, y, and z-axis respectively can be expressed as:

$$\begin{cases} \tau_\phi = -lF_1 + lF_2 + lF_3 - lF_4 \\ \tau_\theta = lF_1 - lF_2 + lF_3 - lF_4 \\ \tau_\psi = \mathrm{M}_1 + M_2 - M_3 - M_4 \end{cases} \tag{2.16}$$

Moments acting on the quadrotor in the body frame:

$$\tau_1 = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} bl(-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ bl(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ d(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \end{bmatrix} \tag{2.17}$$

Where l is the moment arm, which is the distance between the rotor and the origin of the body reference frame which should coincide with the center of gravity of the quadrotor.

The gyroscopic moment τ_2 of a rotor is a physical effect in which gyroscopic torques or moments attempt to align the spin axis of the rotor along the inertial z-axis.

$$\tau_2 = -I_r \left(\dot{\eta} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \Omega_r = \begin{bmatrix} -\dot{\theta} I_r \Omega_r \\ \dot{\phi} I_r \Omega_r \\ 0 \end{bmatrix} \tag{2.18}$$

Where:

$\Omega_r = (\Omega_1 + \Omega_2 - \Omega_3 - \Omega_4)$ rotor's relative speed.

I_r rotor's inertia moment.

From equation (2.13), (2.17) and (2.18) the rotational motion equation can be deduced:

$$\begin{cases} \ddot{\phi} = \dot{\theta}\dot{\psi}\left(\dfrac{I_{yy} - I_{zz}}{I_{xx}}\right) + \dfrac{I_r \dot{\theta} \Omega_r}{I_{xx}} + \dfrac{l}{I_{xx}} U_2 \\ \ddot{\theta} = \dot{\phi}\dot{\psi}\left(\dfrac{I_{zz} - I_{xx}}{I_{yy}}\right) - \dfrac{I_r \dot{\phi} \Omega_r}{I_{yy}} + \dfrac{l}{I_{yy}} U_3 \\ \ddot{\psi} = \dot{\phi}\dot{\theta}\left(\dfrac{I_{xx} - I_{yy}}{I_{zz}}\right) + \dfrac{1}{I_{zz}} U_4 \end{cases} \tag{2.19}$$

Where (U_1, U_2, U_3, U_3) are the inputs control.

2.2.3 General Dynamic Model of Motion

To obtain the general dynamic model of motion the translational with rotational motion, so the quadcopter is able the navigate in 6DOF.

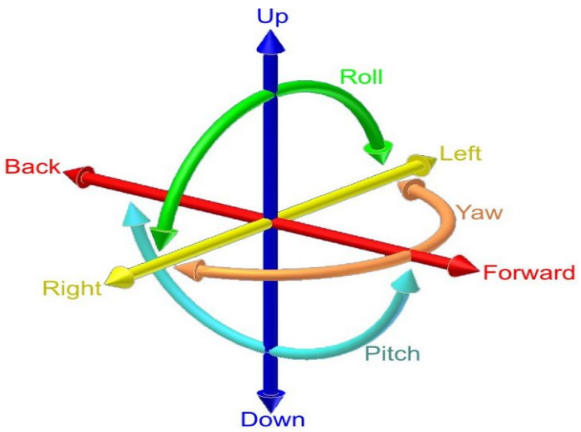


Figure 2.4: Six degree of freedom

From equations (2.19) and (2.10) we obtained the full description of the motion of quadrotor.

$$\begin{cases} \ddot{x} = (cos\ \phi\ sin\ \theta\ cos\ \psi + sin\ \phi\ sin\ \psi)\dfrac{U_1}{m} \\ \ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\dfrac{U_1}{m} \\ \ddot{z} = -g + (\cos\phi \cos\theta)\dfrac{U_1}{m} \\ \ddot{\phi} = \dot{\theta}\dot{\psi}\left(\dfrac{I_{yy} - I_{zz}}{I_{xx}}\right) + \dfrac{I_r\dot{\theta}\Omega_r}{I_{xx}} + \dfrac{l}{I_{xx}}U_2 \\ \ddot{\theta} = \dot{\phi}\dot{\psi}\left(\dfrac{I_{zz} - I_{xx}}{I_{yy}}\right) - \dfrac{I_r\dot{\phi}\Omega_r}{I_{yy}} + \dfrac{l}{I_{yy}}U_3 \\ \ddot{\psi} = \dot{\phi}\dot{\theta}\left(\dfrac{I_{xx} - I_{yy}}{I_{zz}}\right) + \dfrac{1}{I_{zz}}U_4 \end{cases} \tag{2.20}$$

2.2.4 State Space Description

Formulating the acquired mathematical model for the quadrotor into a state space model will help make the control problem easier to tackle.

Defining the state vector of the quadrotor to be,

$$X = (x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \quad x_{10} \quad x_{11} \quad x_{12})^T$$

which is mapped to the degrees of freedom of the quadrotor in the following manner,

$$X = (\phi \quad \dot{\phi} \quad \theta \quad \dot{\theta} \quad \psi \quad \dot{\psi} \quad x \quad \dot{x} \quad y \quad \dot{y} \quad z \quad \dot{z})^T$$

The state vector defines the position of the quadrotor in space and its angular and linear velocities.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_4x_6a_1 + a_2x_4 + b_1U_2 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = x_2x_6a_3 - a_4x_2 + b_2U_3 \\ \dot{x}_5 = x_6 \\ \dot{x}_6 = x_2x_4a_5 + b_3U_4 \\ \dot{x}_7 = x_8 \\ \dot{x}_8 = u_x\dfrac{1}{m}U_1 \\ \dot{x}_9 = x_{10} \\ \dot{x}_{10} = u_y\dfrac{1}{m}U_1 \\ \dot{x}_{11} = x_{12} \\ \dot{x}_{12} = -g + u_z\dfrac{1}{m}U_1 \end{cases} \tag{2.22}$$

To simplify, define,

$$a_1 = \left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right) \qquad b_1 = \frac{l}{I_{xx}} \qquad u_x = (cos\ \phi\ sin\ \theta\ cos\ \psi + sin\ \phi\ sin\ \psi)$$

$$a_2 = \left(\frac{I_r \Omega_r}{I_{xx}}\right)$$

$$a_3 = \left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) \qquad b_2 = \frac{l}{I_{yy}} \qquad u_y = (cos\ \phi\ sin\ \theta\ sin\ \psi - sin\ \phi\ cos\ \psi)$$

$$a_4 = \left(\frac{I_r \Omega_r}{I_{yy}}\right)$$

$$a_5 = \left(\frac{I_{xx} - I_{yy}}{I_{zz}}\right) \qquad b_3 = \frac{1}{I_{zz}} \qquad u_z = (cos\ \phi\ cos\ \theta)$$

It is worthwhile to note in the latter system that the angles and their time derivatives do not depend on translation components. On the other hand, the translations depend on the angles. One can ideally imagine the overall system described by (2.22) as constituted of two subsystems, the angular rotations and the linear translations [14].

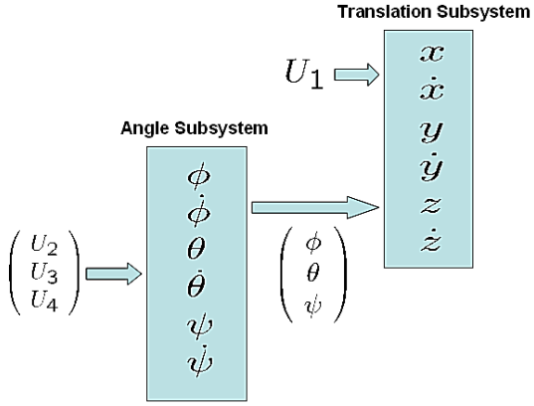


Figure 2.5:Connection of rotations and translations subsystems.

Four control equations are used to keep the quadrotor on the reference value in spite of external disturbances. The signal U_1 is used to guarantee that the altitude follows the reference value, although the signals U_2, U_3 and U_4 are used to control the roll, pitch and yaw of the system. Looking at the equations (2.20), we found four control input signals. The action of these input signals makes the quadrotor moves forwards, backward, to the left, to the right, upwards or down.

A control input vector, U, consisting of four inputs; U_1 through U_4 is defined as [15],

$$U = [U_1 \quad U_2 \quad U_3 \quad U_4]^T$$

Where:

$$\begin{cases} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = b(-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ U_3 = b(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ U_4 = d(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \end{cases} \tag{2.23}$$

Equations (2.23) can be arranged in a matrix form to result in,

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ -b & b & b & -b \\ b & -b & b & -b \\ d & d & -d & -d \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \tag{2.24}$$

This choice of the control vector U decouples the rotational system, where U_1 will generate the desired altitude of the quadrotor, U_2 will generate the desired roll angle, the desired pitch angle will be generated by U_3 whereas U_4 will generate the desired heading.

If the rotor velocities are needed to be calculated from the control inputs, an inverse relationship between the control inputs and the rotors' velocities is needed, which can be acquired by inverting the matrix in Equation (2.24) to give,

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & \frac{-1}{4b} & \frac{1}{4b} & \frac{1}{4d} \\ \frac{1}{4b} & \frac{1}{4b} & \frac{-1}{4b} & \frac{1}{4d} \\ \frac{1}{4b} & \frac{1}{4b} & \frac{1}{4b} & \frac{-1}{4d} \\ \frac{1}{4b} & \frac{-1}{4b} & \frac{-1}{4b} & \frac{-1}{4d} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{2.25}$$

Taking the square root of that, the rotors' velocities can be calculated from the control inputs as follows,

$$\Omega_1 = \sqrt{\frac{1}{4b}U_1 - \frac{1}{4b}U_2 + \frac{1}{4b}U_3 + \frac{1}{4d}U_4}$$

$$\Omega_2 = \sqrt{\frac{1}{4b}U_1 + \frac{1}{4b}U_2 - \frac{1}{4b}U_3 + \frac{1}{4d}U_4}$$

$$\Omega_3 = \sqrt{\frac{1}{4b}U_1 + \frac{1}{4b}U_2 + \frac{1}{4b}U_3 - \frac{1}{4d}U_4}$$

$$\Omega_4 = \sqrt{\frac{1}{4b}U_1 - \frac{1}{4b}U_2 - \frac{1}{4b}U_3 - \frac{1}{4d}U_4}$$

2.3 Controller Design

Due to the fact that the dynamics of the quadrotor is of a nonlinear nature, developing nonlinear/linear control algorithms to be used as flight controllers was necessary. There is a variety of control algorithm applied to quadrotor such as sliding mode control and PD.

In this section, the formulated quadrotor model has been used in control design. Two controllers have been developed: Sliding Mode and PD.

2.3.1 Sliding Mode Controller

2.3.1.1 Basic Concept of SMC

The aim of the proposed flight controller is to achieve asymptotic position and attitude tracking of the quadcopter based on Sliding Mode Control Approach. This is obtained through driving the tracking errors to zero to achieve the required tracking performance.

A SMC is a type of Variable Structure Control (VSC). It uses a highspeed switching control law to force the state trajectories to follow a specified, user defined surface in the states space and to maintain the state trajectories on this surface.

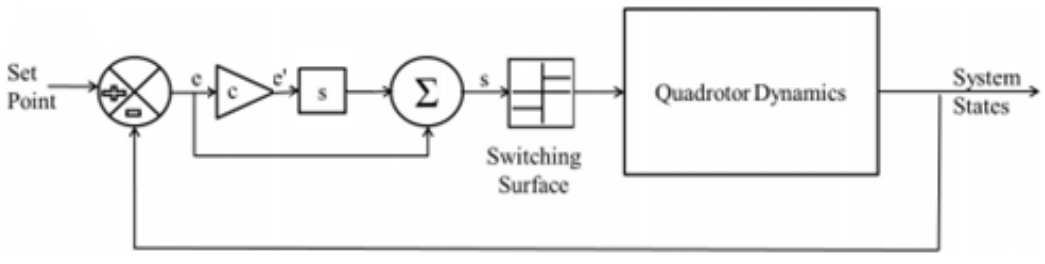


Figure 2.6:Sliding mode control block diagram.

SMC has established itself as an effective robust control technique as robustness is inherent in this control scheme.

The strengths of SMC include:

- Low sensitivity to plant parameter uncertainty.
- Greatly reduced-order modeling of plant dynamics.
- Finite-time convergence (due to discontinuous control law).
- Solution does not depend on plant parameters nor disturbance, and this is called invariance property.
- Its ability to globally stabilize the system.

The weaknesses of SMC include:

- Chattering due to implementation imperfections, fast-switching of controller and discretization chatter due to the fixed sampling rate.

The basic idea of the SMC approach is to attract the system states towards a surface, called sliding surface, suitably chosen, and design a stabilizing control law that keeps the system states on such a surface. For the choice of the sliding surface shape, the general form of Eq. (2.26) was proposed by [16] [17]:

$$S(x) = \left(\lambda_x + \frac{d}{dt}\right)^{n-1} e(x) \tag{2.26}$$

Where x denotes the variable control (state), $e(x)$ is the tracking error defined as $e(x) = x - x_d$ and x_d is the desired trajectory, λ_x is a positive constant that interprets the dynamics of the sliding manifold and presents a design parameter for the SMC method and n is the order of the controlled system. The proposed model is of order $n = 2$, so the surface (sliding manifold) is defined by:

$$S(x) = \lambda_x e(x) + \dot{e}(x) \tag{2.27}$$

Condition, called attractiveness is the condition under which the state trajectory will reach the sliding surface. There are two types of conditions of access to the sliding surface. This study, the Lyapunov based approach is used. It consists to make a positive scalar function, given by Eq. (2.28) and called Lyapunov candidate function, for the system state variables and then choose the control law that will decrease this function: In

$$\dot{V}(x) < 0, and\ V(x) > 0 \tag{2.28}$$

In this case, the Lyapunov function can be chosen as:

$$V(x) = \frac{1}{2}S(x)^2 \tag{2.29}$$

The derivative of this above function is negative when the following expression is checked:

$$\dot{V}(x) = S(x)\dot{S}(x) < 0 \tag{2.30}$$

The purpose is to force the system state trajectories to reach the sliding surface and stay on it despite the presence of uncertainty. The sliding control law contains two terms as follows:

$$u(t) = u_{eq}(t) + u_D(t) \tag{2.31}$$

Where $u_{eq}(t)$ denotes the equivalent control, which is a way to determine the behavior of the system when an ideal sliding regime is established. it is calculated from the following
invariance condition of the surface:

$$\begin{cases} S(x,t) = 0 \\ \dot{S}(x,t) = 0 \end{cases} \tag{2.32}$$

And $u_D(t)$ is a discontinuous function calculated by checking the condition of the attractiveness. It is useful to compensate the uncertainties of the model and often defined as follows:

$$u_D(t) = -\varepsilon sgn\big(S(t)\big) - kS(t) \tag{2.33}$$

where K is a positive control parameter and $sgn(.)$ is the mathematical signum function defined as: $sgn(t) = \begin{cases} -1 \; if \; t < 0 \\ 0 \; if \; t = 0 \\ 1 \; if \; t > 0 \end{cases}$.

One of the most important aims of SMC is to keep the system trajectories on the sliding surface once they arrive to it.

To enforce the sliding mode with the desired dynamics, the slope of the control surfaces should follow the equation:

$$\dot{S}(x,t) = -\varepsilon sgn\big(S(x,t)\big) - kS(x,t) \tag{2.34}$$

Where $\varepsilon > 0$ and $k > 0$ are the sliding surface exponential approach coefficients.

2.3.1.2 Sliding Mode Controller Designs

The quadcopter dynamic model can be divided into two subsystems: translational motion Eq (2.10) and rotational motion (2.19). The controller is split into parts: altitude sliding mode controller designed for translational motion subsystem, attitude sliding mode controller designed for the rotational motion subsystem and sliding mode for the position.

2.3.1.2.1 Attitude Control

The tracking errors represent the difference between the set-point and current values of the state defined as:

$$\begin{cases} e_\phi = \phi - \phi_d \\ e_\theta = \theta - \theta_d \\ e_\psi = \psi - \psi_d \end{cases} \tag{2.35}$$

Where ϕ_d, θ_d, and ψ_d are the desired roll, pitch, and yaw respectively. The sliding surfaces are chosen based on the tracking errors such as:

$$\begin{cases} S_\phi = \lambda_\phi e_\phi + \dot{e}_\phi \\ S_\theta = \lambda_\theta e_\theta + \dot{e}_\theta \\ S_\psi = \lambda_\psi e_\psi + \dot{e}_\psi \end{cases} \tag{2.36}$$

The exponential reaching law for attitude sliding surface are as follow:

$$\begin{cases} \dot{S}_\phi = -\varepsilon_\phi sgn(S_\phi) - k_\phi S_\phi \\ \dot{S}_\theta = -\varepsilon_\theta sgn(S_\theta) - k_\theta S_\theta \\ \dot{S}_\psi = -\varepsilon_\psi sgn(S_\psi) - k_\psi S_\psi \end{cases} \tag{2.37}$$

The derivatives of the sliding surfaces are then equated to the exponential reaching laws and substituting Eq. (2.35) in Eq. (2.36) as follow:

$$\begin{cases} \dot{S}_\phi = -\varepsilon_\phi sgn(S_\phi) - k_\phi S_\phi = \lambda_\phi(\dot{\phi} - \dot{\phi}_d) + (\ddot{\phi} - \ddot{\phi}_d) = \dot{S}_\phi \\ \dot{S}_\theta = -\varepsilon_\theta sgn(S_\theta) - k_\theta S_\theta = \lambda_\theta(\dot{\theta} - \dot{\theta}_d) + (\ddot{\theta} - \ddot{\theta}_d) = \dot{S}_\theta \\ \dot{S}_\psi = -\varepsilon_\psi sgn(S_\psi) - k_\psi S_\psi = \lambda_\psi(\dot{\psi} - \dot{\psi}_d) + (\ddot{\psi} - \ddot{\psi}_d) = \dot{S}_\psi \end{cases} \tag{2.38}$$

Substituting $\ddot{\phi}$, $\ddot{\theta}$, and $\ddot{\psi}$ by its definitions from Eq. (2.19), the control inputs for pitch U_2, roll U_3, and yaw U_4 are calculated:

$$\begin{cases} U_2 = \left[-\lambda_\phi(\dot{\phi} - \dot{\phi}_d) - \frac{I_r\dot{\theta}\Omega_r}{I_{xx}} - \dot{\theta}\dot{\psi}\left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right) + \ddot{\phi}_d - \varepsilon_\phi sgn(S_\phi) - k_\phi S_\phi\right]\frac{I_{xx}}{l} \\ U_3 = \left[-\lambda_\theta(\dot{\theta} - \dot{\theta}_d) + \frac{I_r\dot{\phi}\Omega_r}{I_{yy}} - \dot{\phi}\dot{\psi}\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) + \ddot{\theta}_d - \varepsilon_\theta sgn(S_\theta) - k_\theta S_\theta\right]\frac{I_{yy}}{l} \\ U_4 = \left[-\lambda_\psi(\dot{\psi} - \dot{\psi}_d) - \dot{\phi}\dot{\theta}\left(\frac{I_{xx} - I_{yy}}{I_{zz}}\right) + \ddot{\psi}_d - \varepsilon_\psi sgn(S_\psi) - k_\psi S_\psi\right]I_{zz} \end{cases} \quad (2.39)$$

2.3.1.2.2 Altitude Control

As what has been done with the development of the attitude controllers, an altitude SMC is implemented. The error is defined as:

$$e_z = z - z_d \quad (2.40)$$

Where z_d is the desired altitude of the quadrotor. The sliding surface is defined as:

$$S_z = \lambda_z e_z + \dot{e}_z \quad (2.41)$$

The exponential reaching law for altitude sliding surface are as follow:

$$\dot{S}_z = -\varepsilon_z sgn(S_z) - k_z S_z \quad (2.42)$$

The derivative of the sliding surface is then equated to the exponential reaching law as follows:

$$\dot{S}_z = -\varepsilon_z sgn(S_z) - k_z S_z = \lambda_z(\dot{z} - \dot{z}_d) + (\ddot{z} - \ddot{z}_d) \quad (2.43)$$

Substituting $\ddot{z}$ by its definition from Eq. (2.19), the control input for altitude U_1 is calculated:

$$U_1 = [-\lambda_z(\dot{z} - \dot{z}_d) + g + \ddot{z}_d - \varepsilon_z sgn(S_z) - k_z S_z]\frac{m}{\cos\phi\cos\theta} \quad (2.44)$$

2.3.1.2.3 Position Control

Position tracking of the quadcopter is achieved by calculating the desired rotational angles ϕ_d and θ_d from the translational equations of motion Eq. (2.10).

Since the quadrotor is operating around hover, which means small values for the roll and pitch angles ϕ and θ, we can use the small angle assumption ($\sin\phi_d = \phi_d, \sin\theta_d = \theta_d \; and \cos\phi_d = \cos\theta_d = 1$) to simplify the above equations:

$$\begin{cases} \ddot{x} = (\theta_d \cos\psi + \phi_d \sin\psi)\dfrac{U_1}{m} \\ \ddot{y} = (\theta_d \sin\psi - \phi_d \cos\psi)\dfrac{U_1}{m} \end{cases} \tag{2.45}$$

Which are the angles ϕ_d and θ_d can be derived in a matrix form as:

$$\begin{bmatrix} \phi_d \\ \theta_d \end{bmatrix} = \frac{m}{U_1}\begin{bmatrix} sin\,\psi & -cos\,\psi \\ cos\,\psi & sin\,\psi \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{2.46}$$

The calculated ϕ_d and θ_d have to be limited to the range between -20^o and $+20^o$ to fulfill the small angle assumption used in the derivation and this can be done via a saturation function in the simulation. Other hand, because of the assumption that the quadrotor movement is near the hovering state, the thrust forces U_1 have to nullify the gravitational force allowing for the U_1/m to change with gravitational acceleration, therefore, Eq. (2.46) becomes:

$$\begin{bmatrix} \phi_d \\ \theta_d \end{bmatrix} = \frac{1}{g}\begin{bmatrix} sin\,\psi & -cos\,\psi \\ cos\,\psi & sin\,\psi \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{2.47}$$

Defining the errors for x and y axes as:

$$\begin{cases} e_x = x - x_d \\ e_y = y - y_d \end{cases} \tag{2.48}$$

Where x_d and y_d are the desired position of the quadrotor. And the sliding surfaces are:

$$\begin{cases} S_x = \lambda_x e_x + \dot{e}_x \\ S_y = \lambda_y e_y + \dot{e}_y \end{cases} \tag{2.49}$$

The exponential reaching law for position x and y sliding surfaces are as follow:

$$\begin{cases} \dot{S}_x = -\varepsilon_x sgn(S_x) - k_x S_x \\ \dot{S}_y = -\varepsilon_y sgn(S_y) - k_y S_y \end{cases} \tag{2.50}$$

The derivatives of the sliding surfaces are then equated to the exponential reaching law as follows:

$$\begin{cases} \dot{S}_x = -\varepsilon_x sgn(S_x) - k_x S_x = \lambda_x(\dot{x} - \dot{x}_d) + (\ddot{x} - \ddot{x}_d) = \dot{S}_x \\ \dot{S}_y = -\varepsilon_y sgn(S_y) - k_y S_y = \lambda_y(\dot{y} - \dot{y}_d) + (\ddot{y} - \ddot{y}_d) = \dot{S}_y \end{cases} \tag{2.51}$$

From Eq. (2.50) $\ddot{x}$ and $\ddot{y}$ are derived:

$$\begin{cases} \ddot{x} = -\lambda_x(\dot{x} - \dot{x}_d) + \ddot{x}_d - \varepsilon_x sgn(S_x) - k_x S_x \\ \ddot{y} = -\lambda_y(\dot{y} - \dot{y}_d) + \ddot{y}_d - \varepsilon_y sgn(S_y) - k_y S_y \end{cases} \tag{2.52}$$

Substituting in the matrix of desired roll/pitch provide the following results:

$$\begin{bmatrix}\phi_d\\ \theta_d\end{bmatrix}=\frac{1}{g}\begin{bmatrix}sin\,\psi & -cos\,\psi\\ cos\,\psi & sin\,\psi\end{bmatrix}\begin{bmatrix}-\lambda_x(\dot{x}-\dot{x}_d)+\ddot{x}_d-\varepsilon_x sgn(S_x)-k_x S_x\\ -\lambda_y(\dot{y}-\dot{y}_d)+\ddot{y}_d-\varepsilon_y sgn(S_y)-k_y S_y\end{bmatrix} \quad (2.53)$$

2.3.1.2.4 SMC Chattering Reduction

The chattering problem in SMC approach is the one of the most common handicaps for applying to real-time prototyping [18]. The term chattering describes the phenomenon of finite-frequency, finite-amplitude oscillations appearing in many sliding mode implementations. These oscillations are caused by the high-frequency switching of a sliding mode controller exciting unmodeled dynamics in the closed loop. "Unmodeled dynamics" may be those of sensors and actuators neglected in the principal modeling process because they are generally significantly faster than the main system dynamics [19].

Different techniques were proposed in the literature to address the chattering problem, Slotine and Li proposed the "boundary layer solution" in which the control law in Eq. (2.33) is replaced by a saturation function that approximates the sign term in a boundary layer of the manifold $S(x)=0$, [16].

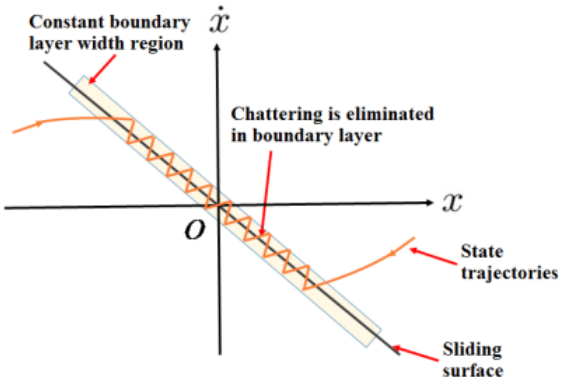


Figure 2.7: Chattering phenomenon and boundary layer concepts in SMC.

The boundary layer approach in SMC changes the control law from discontinuous to continuous. Discontinuity of the proposed control law is the main source of chattering.

The sat function is defined by,

$$sat(S(t))=\begin{cases}S & if\ |S|\le 1\\ sgn(S) & if\ |S|>1\end{cases}$$

Changing the $sign$ function with the $saturation$ function in equations (2.39), (2.44), and (2.53) gives us the following equations:

For the altitude and attitude controllers,

$$\begin{cases} U_1 = [-\lambda_z(\dot{z} - \dot{z}_d) + g + \ddot{z}_d - \varepsilon_z sat(S_z) - k_z S_z]\dfrac{m}{\cos\phi\cos\theta} \\ U_2 = \left[-\lambda_\phi(\dot{\phi} - \dot{\phi}_d) - \dfrac{I_r\dot{\theta}\Omega_r}{I_{xx}} - \dot{\theta}\dot{\psi}\left(\dfrac{I_{yy} - I_{zz}}{I_{xx}}\right) + \ddot{\phi}_d - \varepsilon_\phi sat(S_\phi) - k_\phi S_\phi\right]\dfrac{I_{xx}}{l} \\ U_3 = \left[-\lambda_\theta(\dot{\theta} - \dot{\theta}_d) + \dfrac{I_r\dot{\phi}\Omega_r}{I_{yy}} - \dot{\phi}\dot{\psi}\left(\dfrac{I_{zz} - I_{xx}}{I_{yy}}\right) + \ddot{\theta}_d - \varepsilon_\theta sat(S_\theta) - k_\theta S_\theta\right]\dfrac{I_{yy}}{l} \\ U_4 = \left[-\lambda_\psi(\dot{\psi} - \dot{\psi}_d) - \dot{\phi}\dot{\theta}\left(\dfrac{I_{xx} - I_{yy}}{I_{zz}}\right) + \ddot{\psi}_d - \varepsilon_\psi sat(S_\psi) - k_\psi S_\psi\right] I_{zz} \end{cases} \quad (2.54)$$

And for position controllers,

$$\begin{bmatrix}\phi_d \\ \theta_d\end{bmatrix} = \frac{1}{g}\begin{bmatrix}sin\,\psi & -cos\,\psi \\ cos\,\psi & sin\,\psi\end{bmatrix}\begin{bmatrix}-\lambda_x(\dot{x} - \dot{x}_d) + \ddot{x}_d - \varepsilon_x sat(S_x) - k_x S_x \\ -\lambda_y(\dot{y} - \dot{y}_d) + \ddot{y}_d - \varepsilon_y sat(S_y) - k_y S_y\end{bmatrix} \quad (2.55)$$

2.3.2 Proportional Derivative Controller

PD is stand for proportional, derivative, it is a control loop feedback mechanism controller Figure (2.8) commonly used in industrial control systems. And it is continuously calculating the error value as difference between a desired set point and the measured process variable. The controller attempts to minimize the error over time by adjustment of control variable.

A proportional control variable (K_p) will have the effect of reducing the rise time, the proportional component depends only on the difference between the set-point and the process variable and it is referred as the error term.

A derivative control variable (K_d)have the effect of increasing the stability of the system, reducing the overshoot and improve the transient response.

$$U = K_p(setpoint - mesured\ varible\ ) + K_d(setpoint - measured\dot{}\ variable)$$

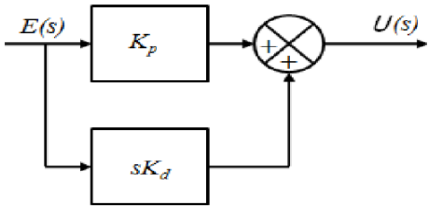


Figure 2.8:PD controller structure.

2.3.2.1.1 Altitude Control

To control translational motion of the quadrotor, A PD controller designed using mathematical model to obtain control signal U_1.

$$U_1 = \left(kp_z(z_d - z) + kd_z(\dot{z_d} - \dot{z})\right) \tag{2.55}$$

Where

kp_z :is the proportional controller gain.

kd_z:is the derivative controller gain.

2.3.2.1.2 Attitude Control

Second PD controller developed for the rotational angles (ϕ, θ, ψ) of the quadrotor. the control signals U_2, U_3, U_4.

$$\begin{cases} U_2 = kp_\phi(\phi_d - \phi) + kd_\phi(\dot{\phi}_d - \dot{\phi}) \\ U_3 = kp_\theta(\theta_d - \theta) + kd_\theta(\dot{\theta}_d - \dot{\theta}) \\ U_4 = kp_\psi(\psi_d - \psi) + kd_\psi(\dot{\psi}_d - \dot{\psi}) \end{cases} \tag{2.56}$$

Where:

$kp_\phi, kp_{\theta,} kp_\psi$ are constants proportional gains.

kd_ϕ, kd_θ, kd_ψ are constants derivatives gains.

2.3.2.1.3 Position Control

Another PD developed to get accurate position (x, y) by computing the desired roll and pitch. Using the same assumption made in previous position controller the following equation are obtained.

$$\begin{bmatrix} \phi_d \\ \theta_d \end{bmatrix} = \left(\frac{1}{g}\right) \begin{bmatrix} s\psi & -c\psi \\ c\psi & s\psi \end{bmatrix} \begin{bmatrix} kp_x(xd - x) + kd_x(\dot{x}_d - \dot{x}) \\ kp_y(yd - y) + kd_y(\dot{y}_d - \dot{y}) \end{bmatrix} \tag{2.57}$$

Where:

$kp_x, kp_{y,}$ are proportional gains.

kd_x, kd_y, are derivatives gains.

2.4 Turning Parameters

For the proceeding control algorithms, tuning the controller constants (gains and different parameters) was done using GA. The objective function of the GA was set to be error between the desired value and the actual value. The optimization toolbox in MATLAB /Simulink used and it includes a built-in command for GA optimization for both controllers sliding mode controller and PD controller [11].

2.5 The Overall Control of Quadrotor

In the previous section three controllers are developed with two techniques (SMC and PD) to get the accurate result and best performance the next figure illustrates the interconnection between the controller and dynamic of the quadrotor.

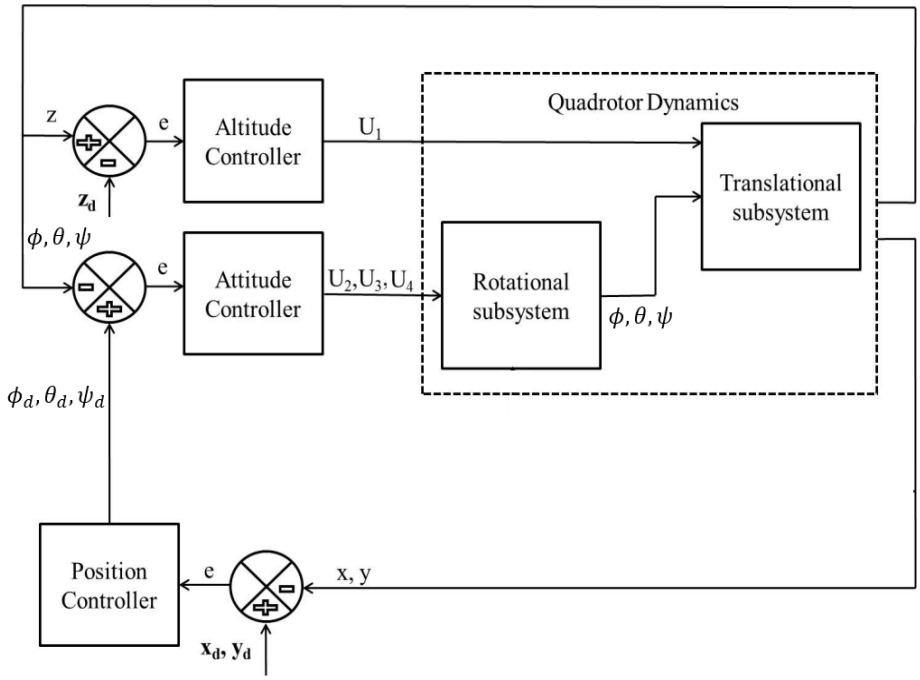


Figure 2.9:Overall control system.

2.6 Conclusion

The goal of this chapter was to derive a mathematical model for the quadrotor Unmanned Aerial Vehicle (UAV) using Newton-Euler formalism with a proposed assumption for simplification, and develop a nonlinear/linear control algorithm represented in Sliding Mode Control and PD to stabilize the states of the quadrotor, which include its attitude, altitude and position in space.

The mathematical model of a quadrotor UAV was developed in details including kinematic model and dynamic model for both translational and rotational motions. A sliding mode controller based on Lyapunov stability theory and proportional derivative controller were developed to stabilize and drive the quadrotor to a desired position for tracking objectives. The main obstacle for application of SMC are two interconnected phenomena: chattering and high activity of control action. The boundary layer approach is then used to reduce and minimize the chattering effect.

Finally, the whole system then is designed with necessary interconnection in a closed loop as illustrated in the last section.

3 Simulation

3.1 Introduction

In this chapter, simulation is performed for open loop and closed loop to test the mathematical model and the controllers developed in previous chapter using MATLAB/ Simulink.

3.2 Parameters Specification

The simulation depends on several parameters briefly described in table (3-1). The computation of those parameters is detailed in Appendix A.

Table 3-1:Quadrotor model parameters

Parameter	Characteristics	Value
b	Thrust factor	4.844×10^{-5}
$\boldsymbol{I_{xx}}$	Inertia value along x axis	$1.318 \times 10^{-2}(kg * m^2)$
I_{yy}	Inertia value along y axis	$1.318 \times 10^{-2}(kg * m^2)$
I_{zz}	Inertia value along z axis	$1.867 \times 10^{-2}(kg * m^2)$
I_r	Inertia of brushless motor	$19.65 \times 10^{-4}(kg * m^2)$
l	Length from the center of the frame	0.23(m)
d	Drag factor	7.50×10^{-7}

3.3 Open Loop Simulation

To verify the mathematical model, an open loop simulation was carried out, Figure (3.1) illustrates the block of developed model in Simulink which has four control input signals and six outputs used to express position in space.

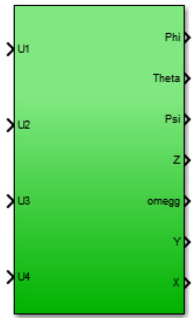


Figure 3.1:Qudrotor dynamic model.

The figure (3.2) shows the interconnection between the rotational and translational parts of the above block.

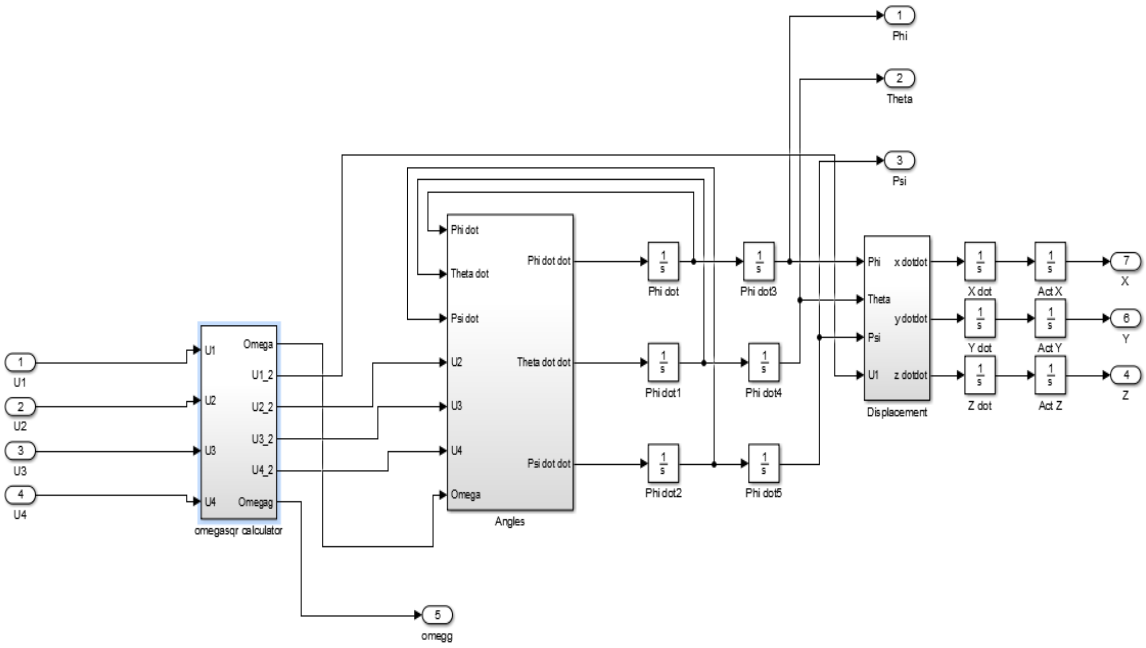


Figure 3.2:Translational and rotational part of the model

By introducing step function as input and collect the outputs to check the stability, knowing that U_1 control the motion the z axis, U_2 control the roll angle, U_3 control the pitch angle and U_4 control the yaw angle.

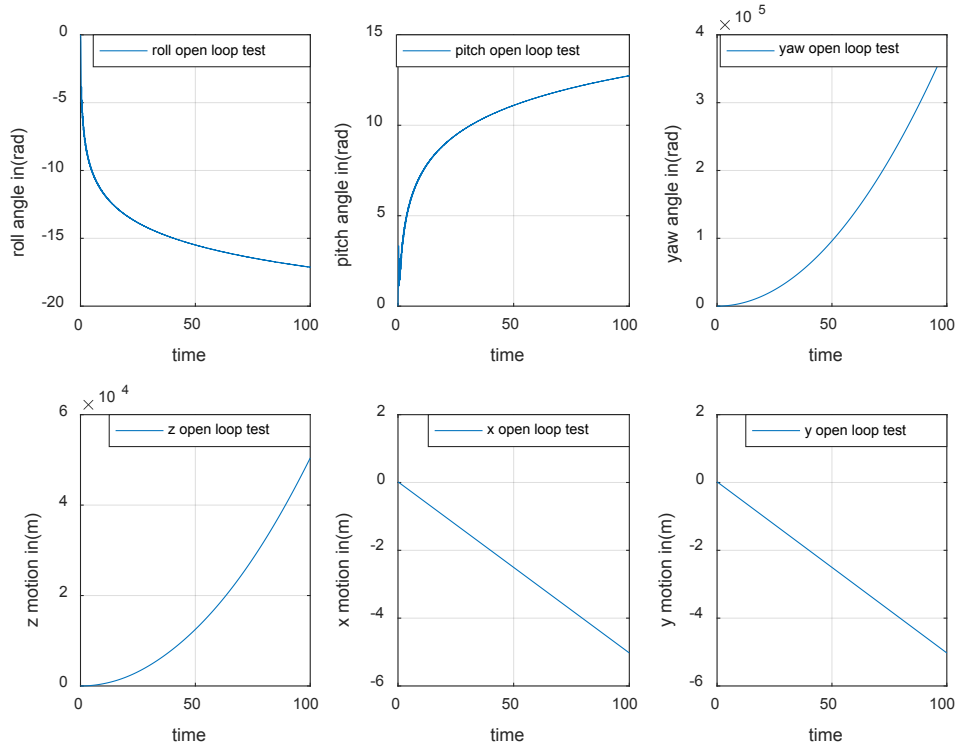


Figure 3.3:Open loop test

The resulting graphs in the previous figure (3.3) demonstrate that the target position is never reached since the values of each output tend to infinity although the input is just a step function. Therefore, the quadrotor system is dynamically unstable.

3.4 Closed Loop Simulation

From the previous section outcome, adding a feedback to the system is then necessary to monitor the progress of actual output. Therefore, to reduce the error between the actual and desired states, two types of controllers are considered in the closed loop and they are simulated in two separate systems for comparison objectives.

3.4.1 SMC Simulation

SMC is the proposed controller that supervises the altitude, attitude and position of quadrotor. The controller response is tested by step input and linear/nonlinear trajectories. The subsystems interconnections are illustrated in figure (3.4).

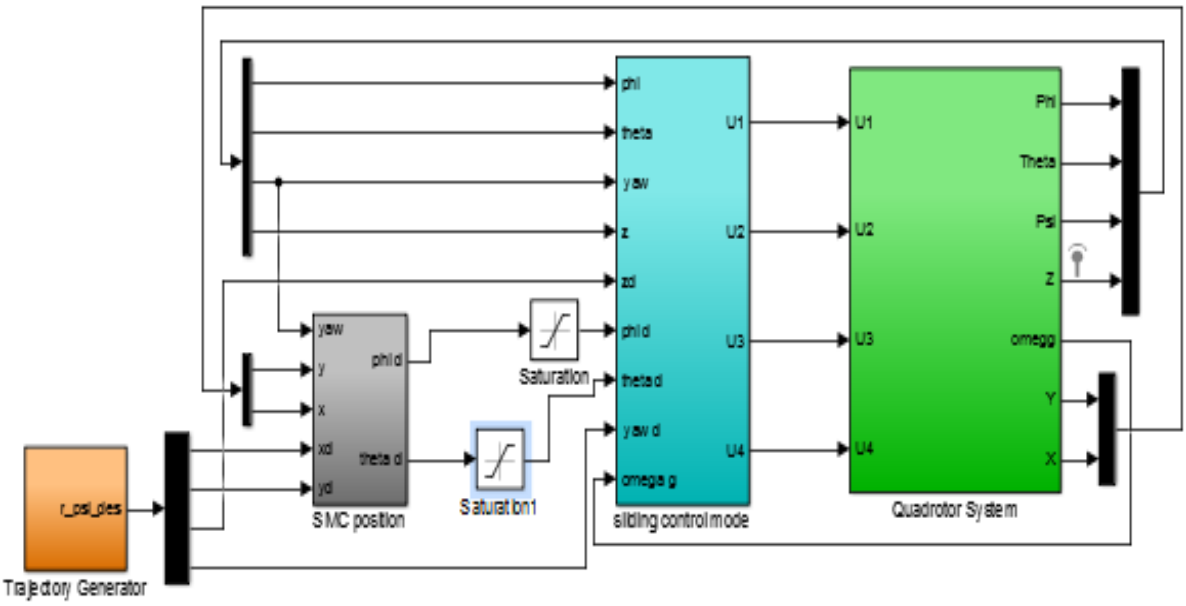


Figure 3.4: Simulink model of a Quadrotor system using SMC controller

3.4.1.1 Step Input Test For SMC

Initially, step input signal is inserted in the desired position states x, y, z *and* yaw, to verify the system performances. Figures (3.5) and (3.6) illustrate the response of each actual state output with the targeted one, and their corresponding control signals respectively.

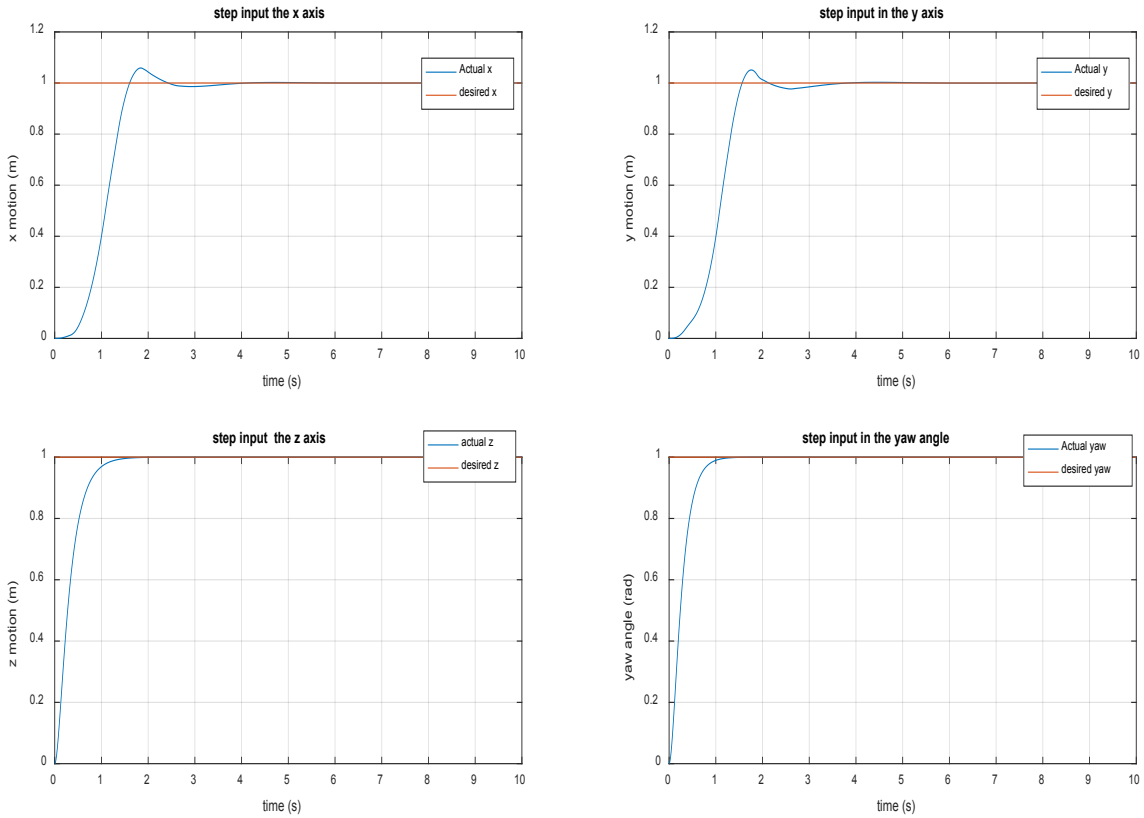


Figure 3.5:Step input test for SMC.

Based on the resulting graphs, some characteristics parameter can be obtained as demonstrated in table (3.2).

Table 3-2:step response performance for SMC.

	X (t)	Y(t)	Z(t)	Yaw (t)
Raise time (s)	1.21	1.19	0.89	0.73
Settling time (s)	2.25	2.12	1.25	1.08
Overshoot (%)	5.32	3.94	0.00	0.00

The outputs of the quadrotor states become more suitable as a result of reducing the error between actual and desired position by an SMC controller that generates appropriate signals to accomplish this task. Therefore, both targeted position and stability are reached. In addition, from figures (3.5) and (3.6), we observe that the responses in the Z axis and yaw attitude are faster with no overshoot in comparison to X and Y directions because the motion in Z axis and yaw attitude are related directly to U_1 and U_4 respectively, whereas, the motion along X and Y axes are associated with both U_2 and U_3.

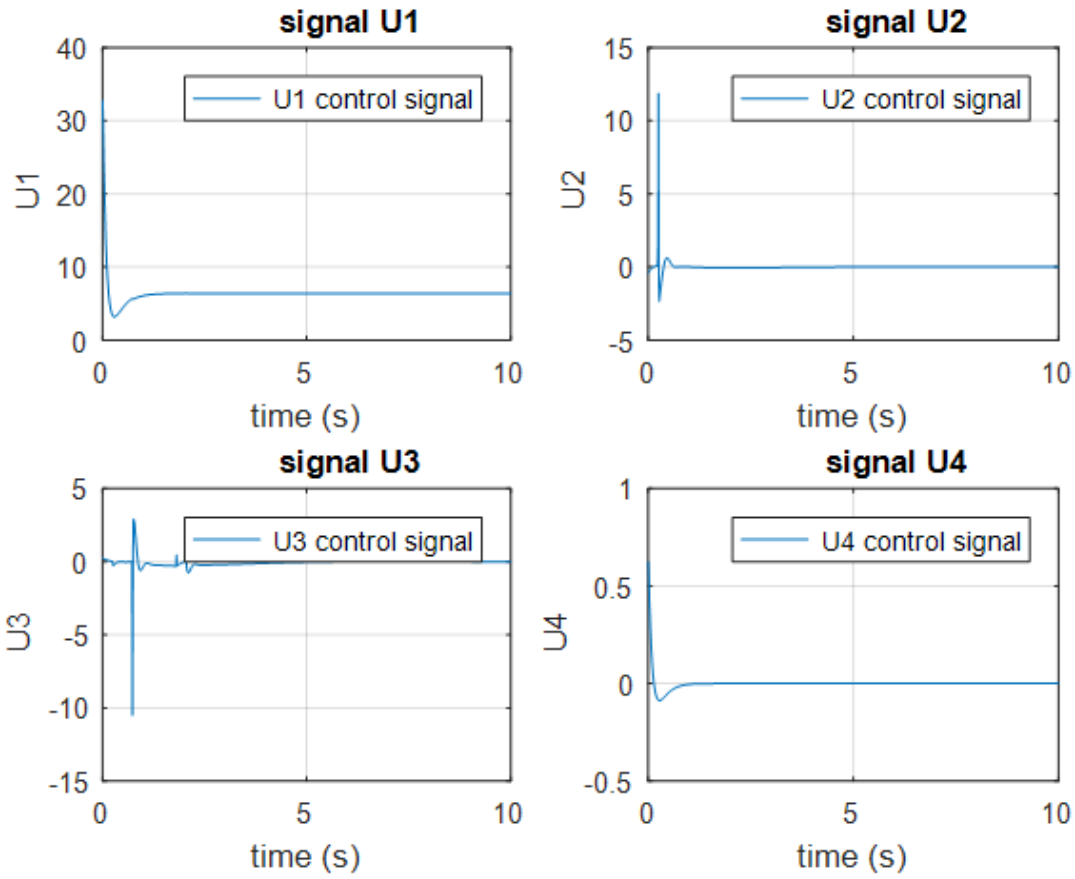


Figure 3.6: The control signal generated by SMC controller.

3.4.1.2 Trajectory Tracking For SMC

The main purpose of this work that the quadrotor is capable of tracking trajectories selected by the user. In this subsection, a linear and nonlinear path are generated and inserted in the controller's inputs.

First, a helix trajectory is selected to evaluate the tracking performances for sliding mode controller in case of a nonlinear path. The helical trajectory is defined as,

$$\begin{cases} x = 4 \times sin(2\pi ft) \\ y = 4 \times sin\left(2\pi ft + \frac{\pi}{2}\right) \\ z = 0.2 \times \mathrm{t} \end{cases} \tag{3.1}$$

The simulation results are shown in Figure (3.7).

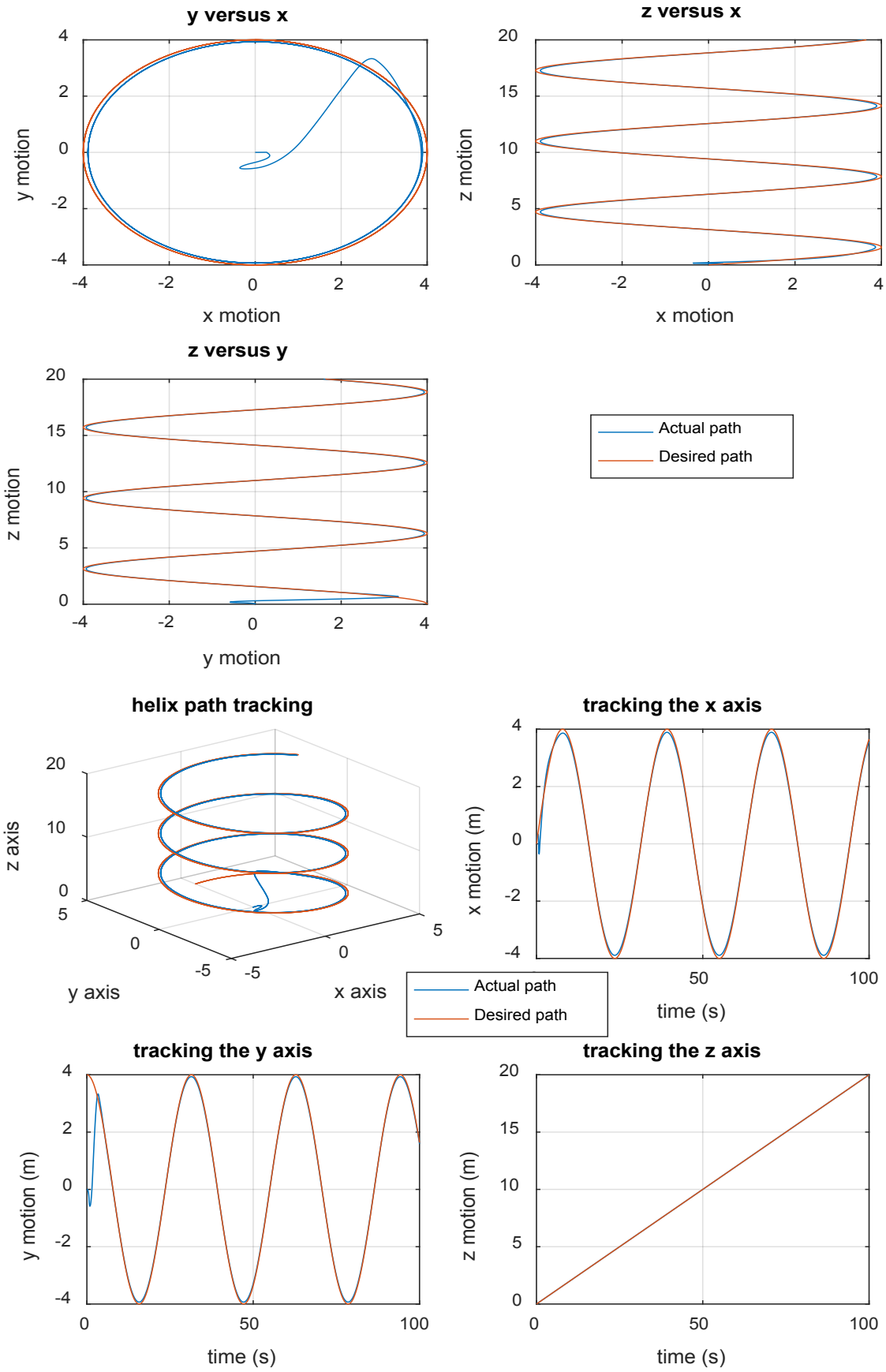


Figure 3.7: Nonlinear trajectory tracking for SMC controller

From the previous figure, we can observe that the initial position of the quadcopter is matching the starting point of the targeted trajectory in x and z axes while it defers in y axis. Therefore, x and z states are identical to their desired values without oscillation from the beginning until mission ends. However, y state converges to its desired value after 4 seconds in cause of difference between initial actual and desired point.

The control signals generated during the trajectory tracking are depicted in figure (3.8).

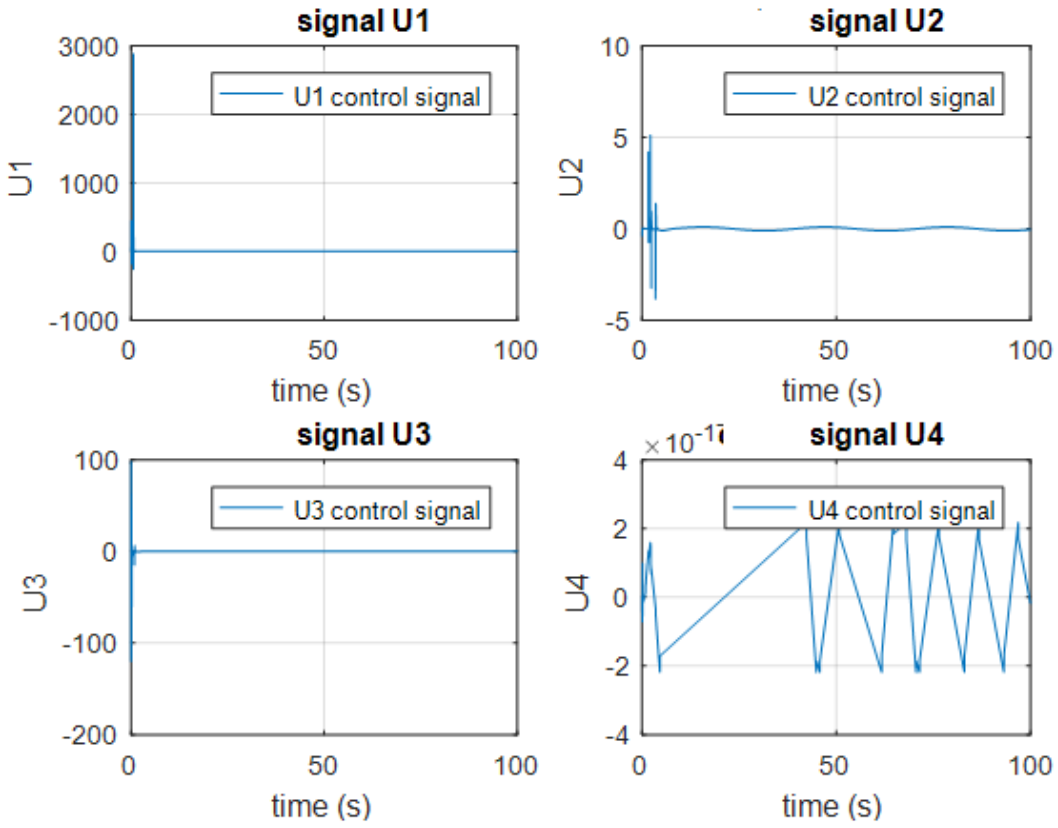


Figure 3.8: Control signal generated during nonlinear trajectory for the SMC

Now, a rectangular trajectory is simulated to evaluate the tracking performances for a linear path. As describes the following set of equations,

$$\begin{cases} z_d = t \quad for \quad t < 10 \\ y_d = (t-10) \quad for \quad t \geq 10 \; and \; t < 20 \\ x_d = (t-20) \quad for \quad t \geq 20 \; and \; t < 30 \\ y_d = 10 - (t-30) \quad for \quad t \geq 30 \; and \; t < 40 \\ x_d = 10 - (t-40) \quad for \quad t \geq 40 \; and \; t < 50 \\ z_d = 10 - (t-50) \quad for \quad t \geq 50 \; and \; t < 60 \end{cases} \tag{3.1}$$

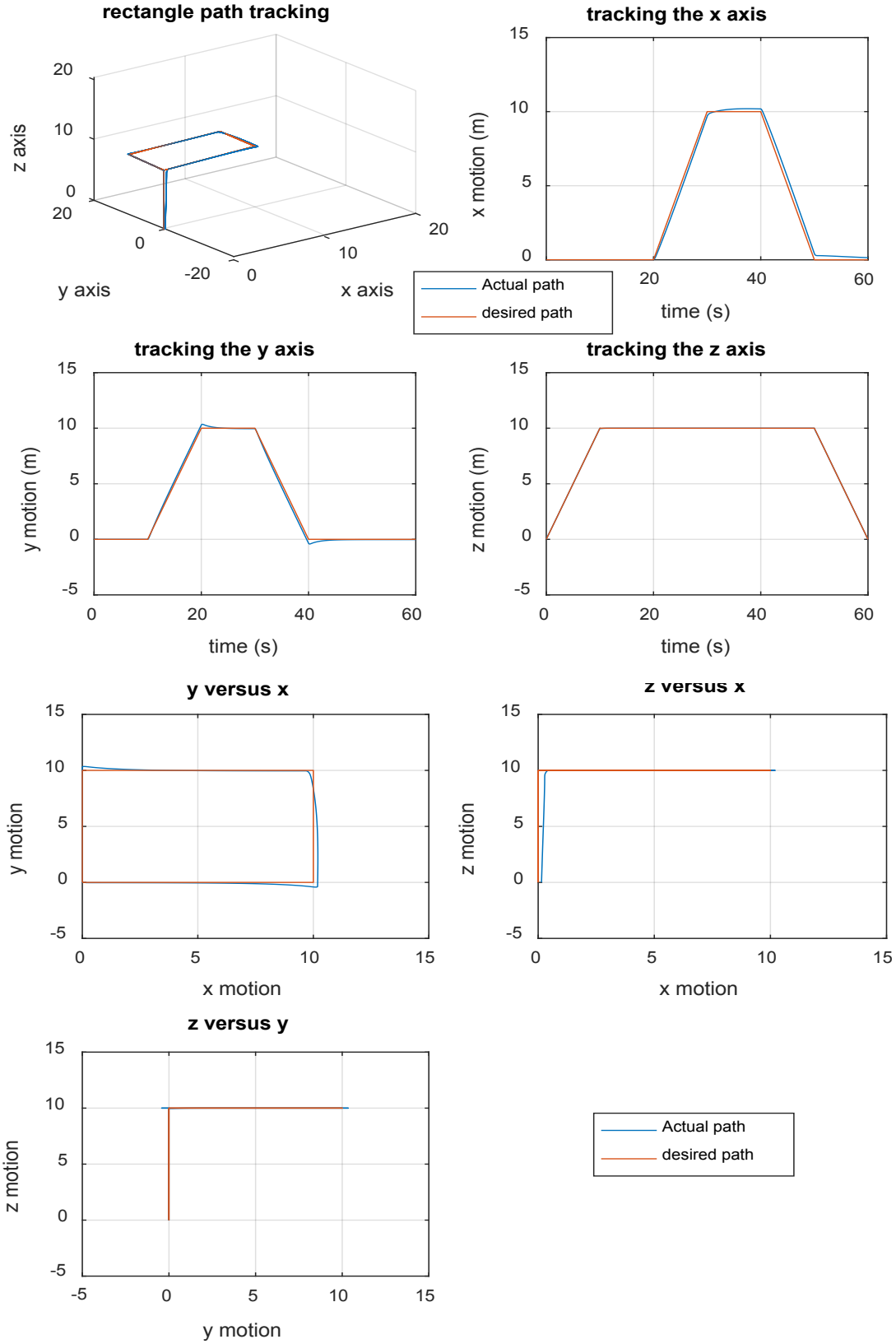
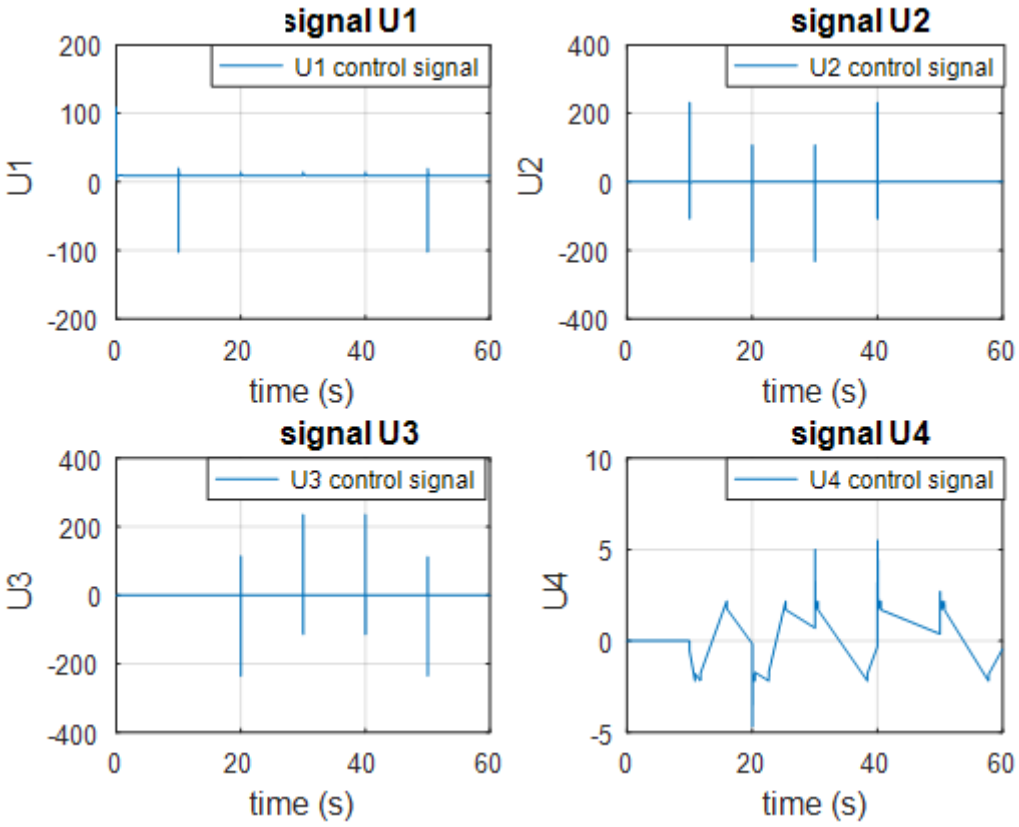


Figure 3.9: Rectangular trajectory for SMC

As shown in figure (3.9), the proposed control scheme can track the desired trajectory accurately after a few seconds. We can observe clearly that the tracking in z axis is the most accurate one, whereas, in x and y axes tracking there are some deviation around the four corners due to the abrupt direction changes.

The next figure (3.10) shows the control signal generated during the path tracking simulation for linear path.



locks

Figure 3.10:Control signal during linear trajectory.

3.4.2.1 Step Input Test For PD

PD tests are performed using step input in the desired position states x, y, z *and* yaw, to verify the system performances. Figures (3.12) and (3.13) illustrate the response of each actual state output with the targeted one, and their corresponding control signals respectively.

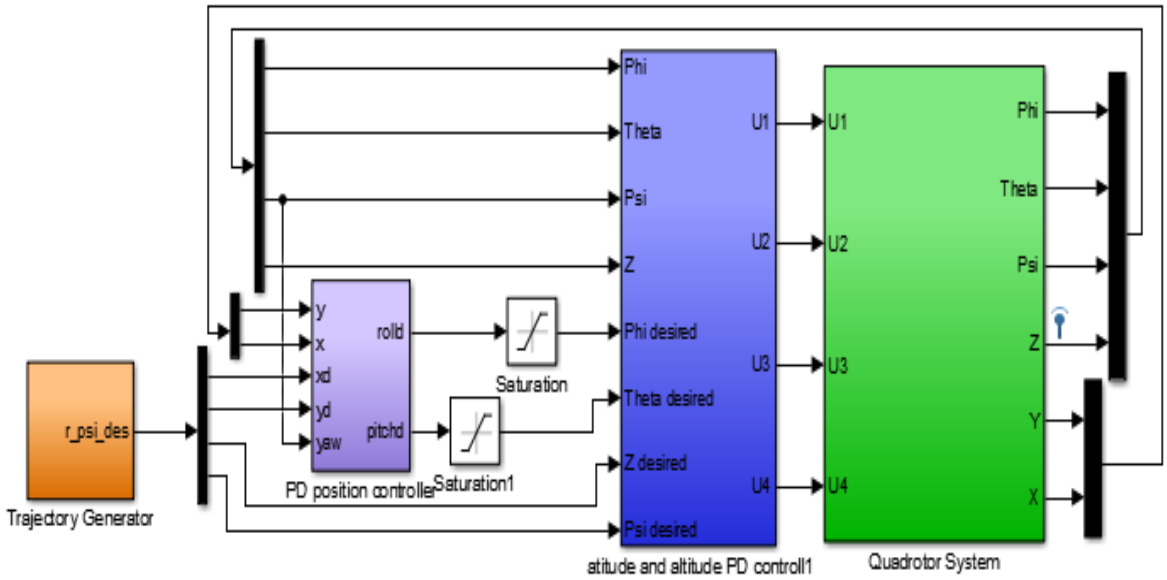


Figure 3.11: Simulink model of a Quadrotor system using PD controller

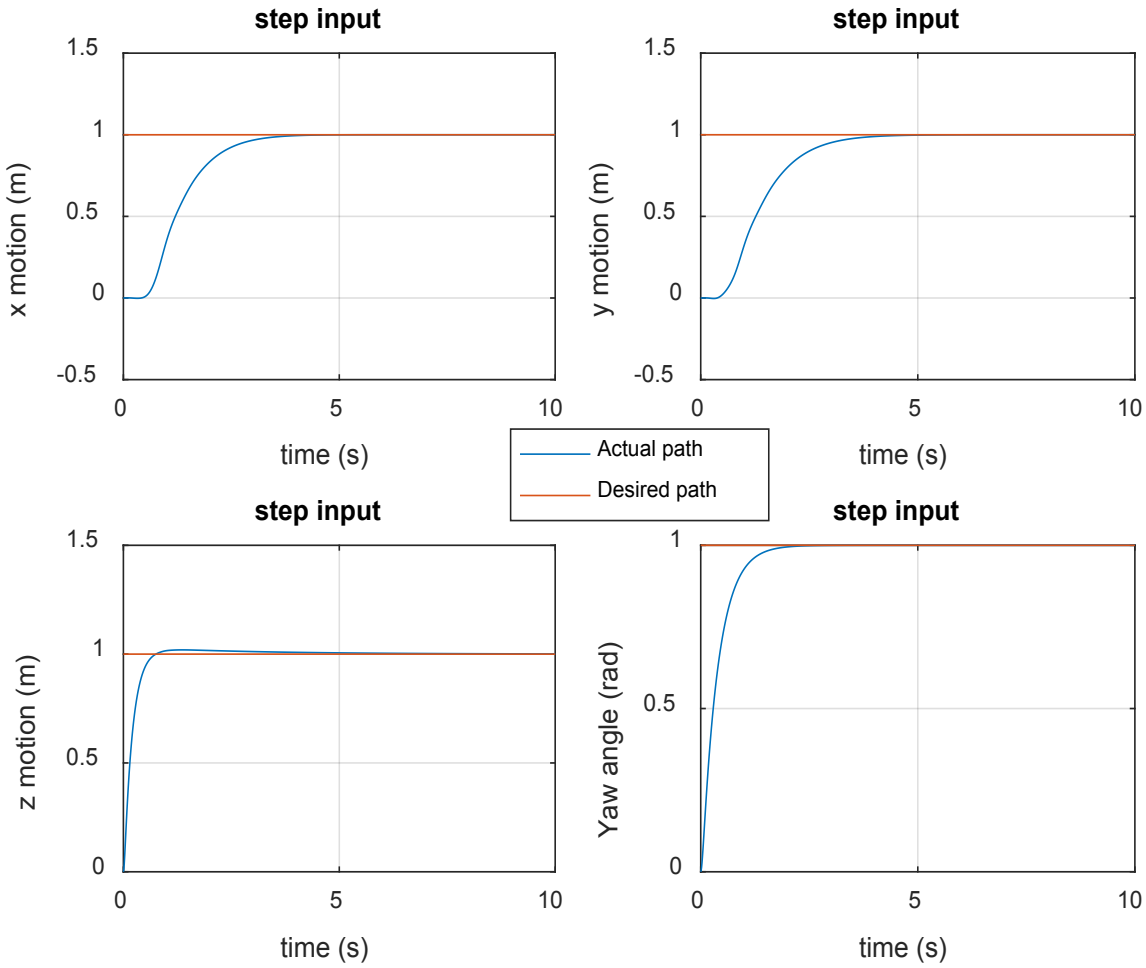


Figure 3.12:Step response of closed loop PD control.

Table (3.3) demonstrates some characteristic parameters of the step response.

Table 3-3:PD Step response performance.

	x (t)	Y(t)	Z(t)	Yaw(t)
Raise time (s)	3.69	3.51	0.55	1.08
Settling time (s)	3.93	3.77	0.74	1.37
Overshoot (%)	0.21	0.24	0.38	0.00

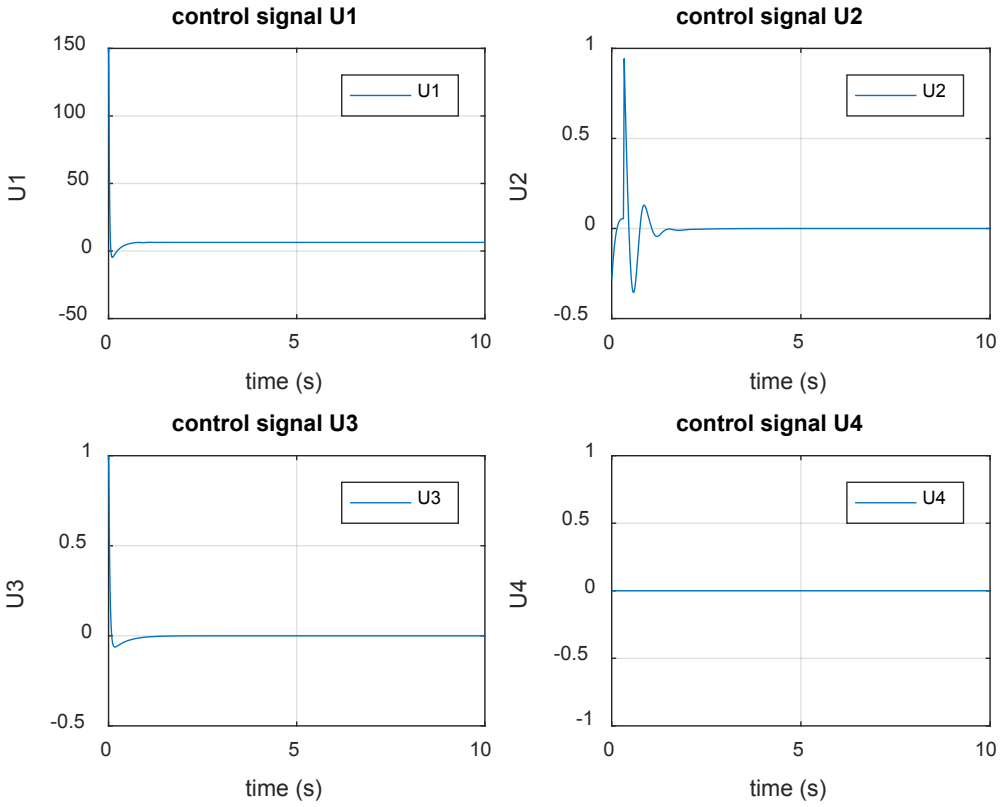


Figure 3.13:Control signals for step input.

3.4.2.2 Trajectory Tracking for PD

With the same procedure used for SMC, two different trajectories are selected as an input to observe PD tracking performance.

First, a helix trajectory is inputted to the quadrotor system, the obtained results are shown in figure (3.14).

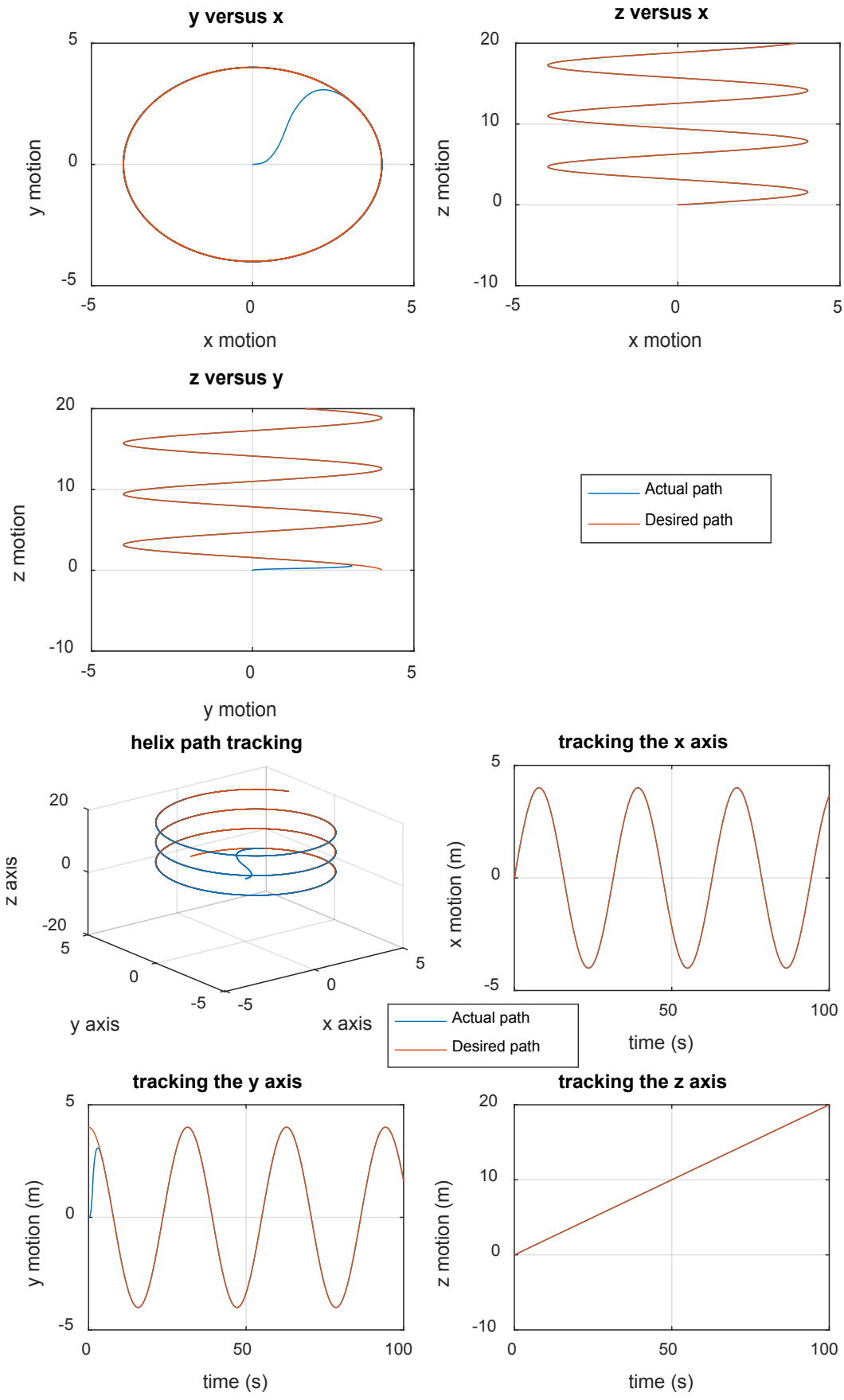


Figure 3.14: Nonlinear trajectory tracking for PD controller

The control signals generated during the trajectory tracking are shown in figure (3.16).

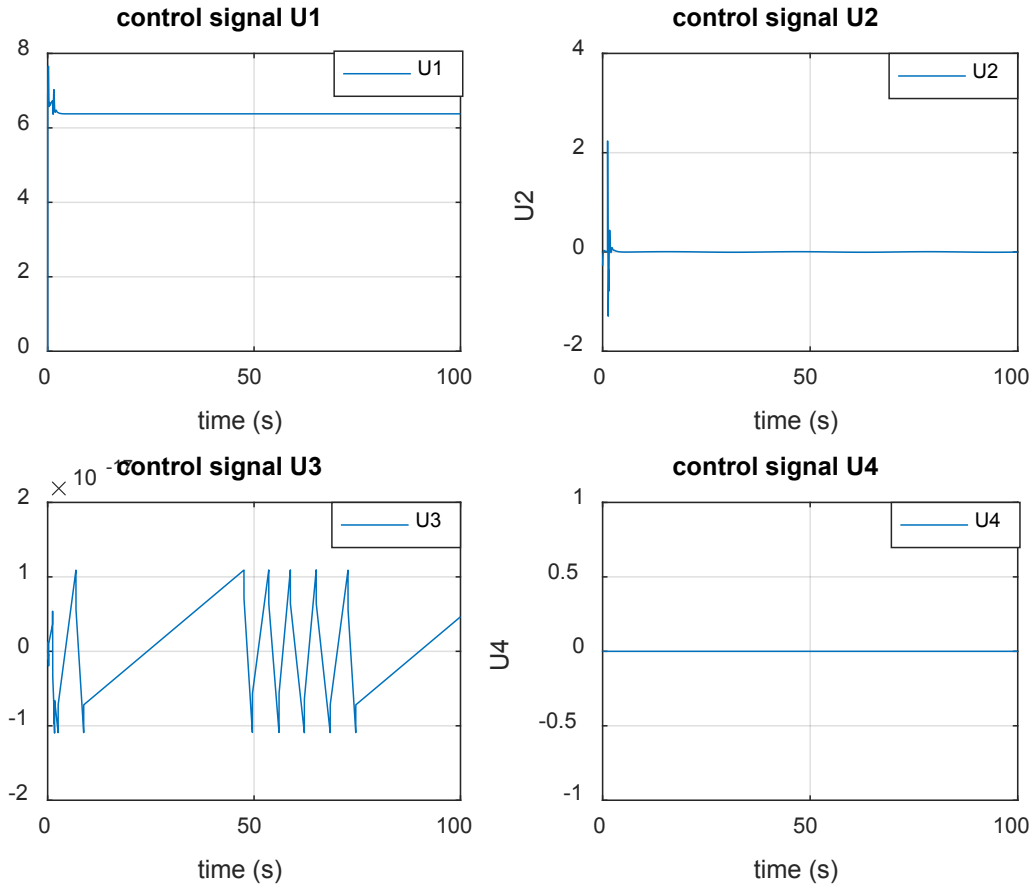


Figure 3.15: Control signal generated during nonlinear trajectory for PD

A rectangular trajectory is simulated to test the tracking performance of a linear path. Figure (3.16) shows the states response of the system.

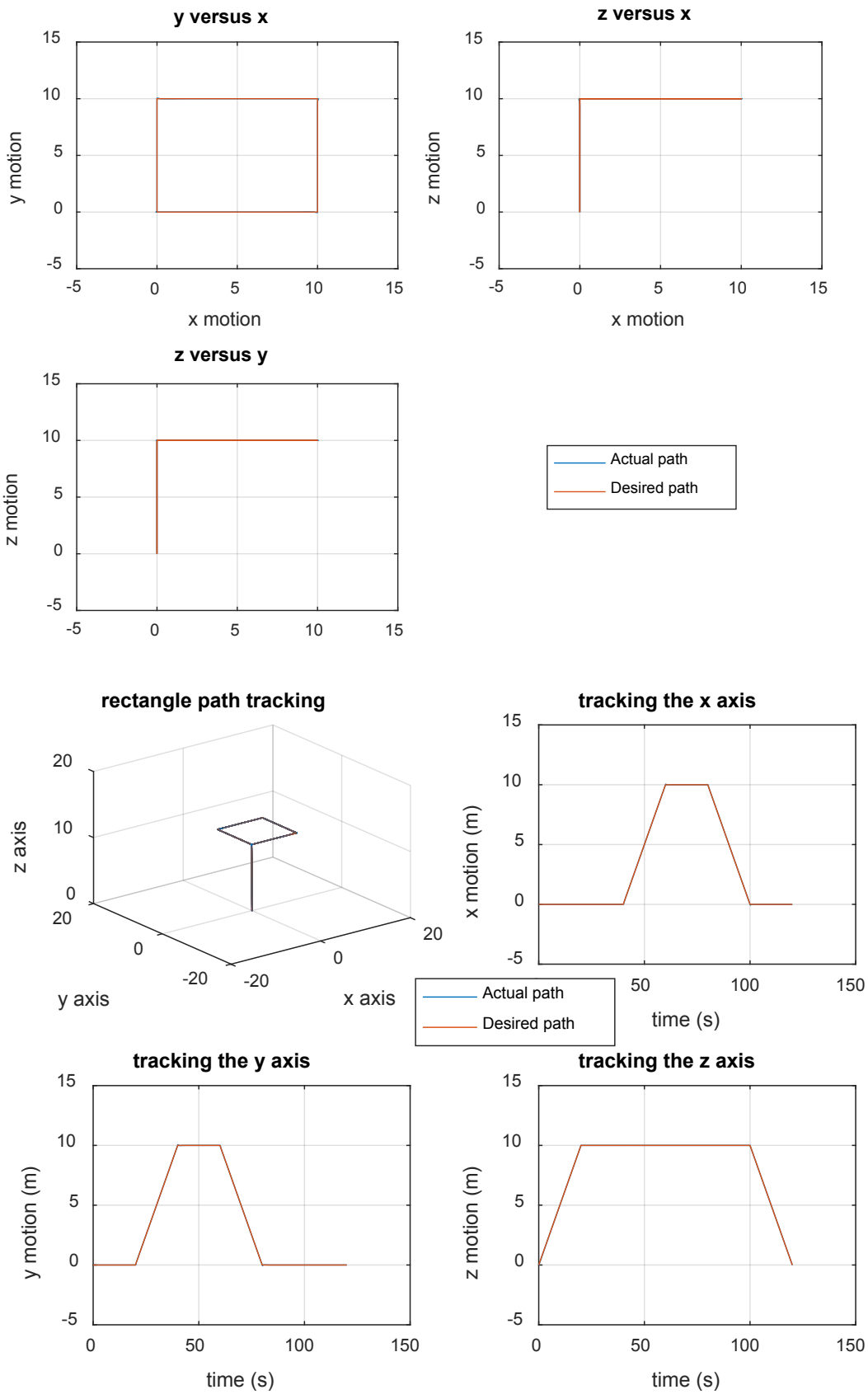


Figure 3.16:Rectangle path tracking for PD

The four control signals generated from altitude and attitude controller is illustrated in figure (3.17).

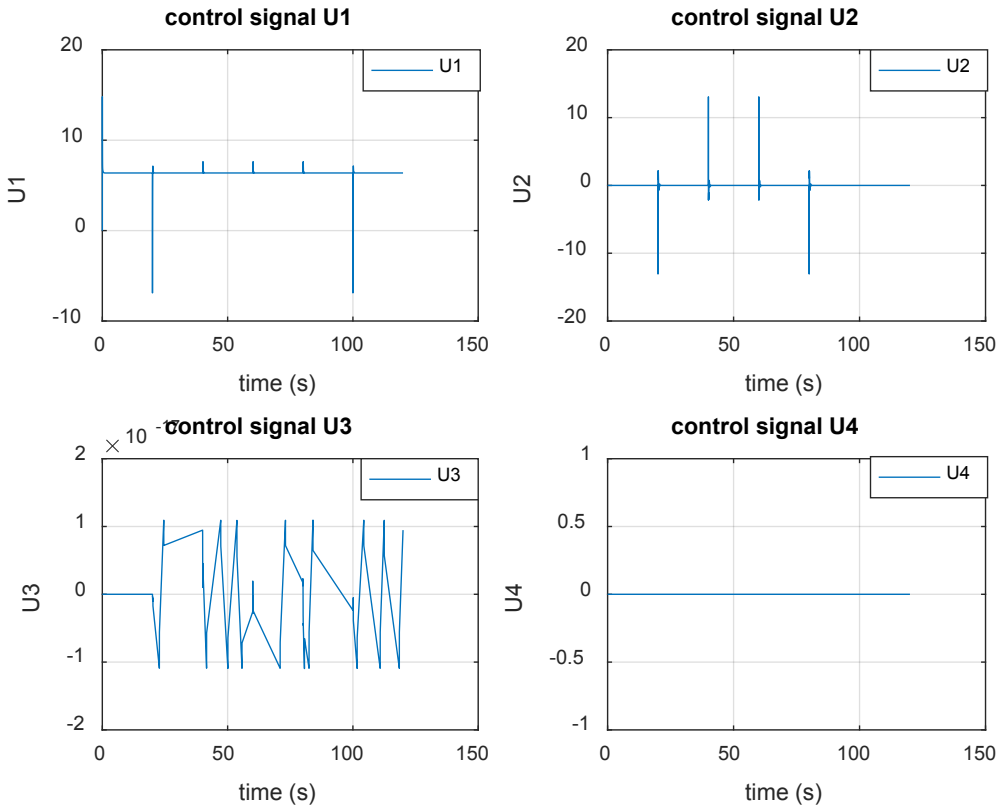


Figure 3.17:Control signal generated during rectangular trajectory.

3.5 SMC and PD Comparison

Table (3.2): summarizes the comparison of the two control algorithms as applied to simulated quadrotor system with all things being equal. The performance of each algorithm depends on many factors that may not even be modeled.

Table 3-4:SMC and PD comparision

Control Algorithm	Characteristic							
	Robust	Tracking ability	Fast convergence/response	Precision	Simplicity	Manual tuning	Noise	Chattering/ energy loss
SMC	1	2	2	1	1	1	0	2
PD	1	1	1	2	2	2	2	0

Legend: 0---low to non; 1---average; 2---high.

3.6 Conclusion

In this chapter, several simulations and tests were performed in order to verify the efficiency of our system with the proposed controllers. At the beginning, the dynamic unitability was proved by an open loop simulation, to state the necessity of adding control algorithm. Next, various testes were carried in a closed loop simulation for SMC/PD controllers starting by step input test to a linear/nonlinear trajectory tracking. Finally, in the last section, a comparison was performed between SMC and PD for several characteristics, therefore, quadrotor tasks and objectives must be carefully selected due to the trade-off between the two control strategies.

4 Quadcopter Implementation

4.1 Introduction

The concept of how quadcopters operate is relatively simple, but implementing each subsystem requires quite a bit of attention to detail for the aircraft to function properly. This chapter highlights the design methodology that was followed for the implementation of the quadcopter and details of how each subsystem works. The first phase takes care of the hardware parts which mainly consist mechanical and electrical construction and flight controller. Whereas, final phase concerned the overall system is implemented with a given circuitry, specifications and software required.

4.2 Requirements

- The quadcopter needs to be medium size, lightweight and rigid.
- The motors should lift twice the weight of the full load quadcopter.
- The quadcopter frame should absorb vibrations.
- Considerable flight time in order to achieve the required application.
- The quadcopter should have a reliable communication with the ground station.

4.3 Mechanical Construction

The mechanical construction of the quadcopter is simple, which is one of the many advantages that made them so popular. However, it is necessary to know the material and the geometrics of the used design. Here the frame and the propellers characteristics will be covered.

4.3.1 Frame

The frame is the main body of the aircraft. All other component, propellers, batteries, computer, etc. Are mounted to the airframe. The quadrotor frames have four arms, each of which are connected to a single motor. They come in a large variety of layouts, sizes, configurations and materials. The two main layouts are: Racing and utility.

The frame size of a quadcopter is the distance from opposite corner motors. It accounts for the diagonal motor to motor distance, that distance (mostly expressed in

millimeters) is the frame size. Generally, small one is from 180 mm to 250 mm, bigger one is from 280 mm to 800 mm even more.

There are two common styles “X” and plus “+” configuration, in addition to that there are “H” and Stretch shapes.

Typical frame can be made from a different material, like: wood, carbon fiber, plastic, aluminum, PCB…etc. Ideally the frame should be reasonably rigid, strong, and lightweight material to be stable with minimal vibration transmission as possible. Quadcopter frame, LIPO battery, motor and propeller size matching table. For the implementation we select a size of 450mm, “X” shape, plastic material and utility layout Appendix B.

Table 4-1:Frame characteristics.

Size (mm)	Propeller Size (inch)	Motor Size	Motor (Kv)	LIPO battery (mAh)
120 or less	3	1104 – 1105	4000 or more	80-800 1s/2s
150– 160	3-4	1306 – 1407	3000 or more	600-900 2s/3s
180	4	1806 – 2204	2600 or more	1000 -1300 3s/4s
210	5	2204 – 2206	2300-2700	1000-1300 3s/4s
250	6	2204 – 2208	2000-2300	1300-1800 3s/4s
330 – 350	7-8	2208 – 2212	1500-1600	2200-3200 3s/4s
450 – 500	9-10	2212 – 2216	800-1000	3300 4s

4.3.2 Propeller

A propeller is a type of fan that transmits power by converting rotational motion into thrust. Propellers come in a variety of diameters and pitches as well as materials such as plastic, reinforced plastic, carbon fiber and wood. The specifications on it are defined in terms of diameter and pitch (D x P). The diameter is the end to end length (in inches) of the propeller.

The pitch of a propeller can be defined as the distance (in inches) travelled per revolution. The larger propellers with lower pitch ratings are more efficient than smaller propellers with higher pitch ratings. Propellers are either designed to rotate clockwise (CW) or counter-clockwise (CCW), these counter-rotations help stabilize the quadcopter.

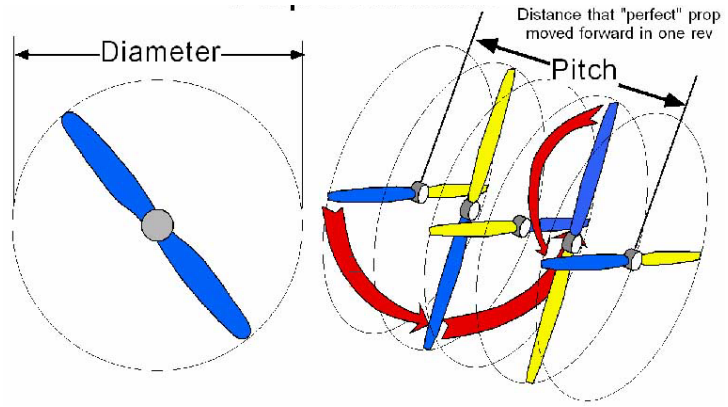


Figure 4.1 : Propeller definition

4.4 Electrical Construction

The electrical and electronic construction of the quadrotor contain the sensors, actuators, battery, microcontroller, and transceiver.

4.4.1 Sensors

Quadrotor uses many different sensors, these are considered the most important component in the stability and control of quadcopter, so carefully selection is needed, and the sensors used in this project are inertial measurement unit (IMU), the barometer, ultrasonic and GPS sensor.

4.4.1.1 Inertial Measurement Unit (IMU)

Is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers, gyroscopes, and magnetometer. For the implemented quadrotor, IMU GY-86 10DOF, ultrasonic HC-SR04 and GPS Ublox NEO-6M are chosen.

4.4.1.1.1 Accelerometer

The accelerometer measures the acceleration relative to the gravitational force. So, the *3*-axis accelerometer basically gives us components of acceleration in all the 3 directions. It can be used to measure the orientation, vibration and shock and hence is critical for the stability of the Quadcopter (in our case). The disadvantage of only using the accelerometer is that it may become extremely sensitive to unwanted vibrations and noise (for example motor vibrations) and may lead to instability of the bot.

4.4.1.1.2 Gyroscope

A gyroscope is a device which measures and maintains the orientation based on the principle of angular momentum. It measures the angular velocity i.e.; how fast something is spinning about an axis as well as rotational acceleration. Unlike accelerometers gyroscopes are not affected by gravity, so they make a great complement to each other. Accelerometer measure acceleration along the specified axes whereas Gyroscopes measure acceleration about the axes.

4.4.1.1.3 Magnetometer

An electronic magnetic compass or simply magnetometer measures the strength and the direction of magnetic fields and hence can be used to determine the orientation of the bot with respect to Earth's magnetic field. This helps in additional Yaw stability.

4.4.1.1.4 Barometer

Barometer, device used to measure atmospheric pressure. Because atmospheric pressure changes with distance above or below sea level, a barometer can also be used to measure altitude [20].

4.4.1.1.5 GPS

Global Positioning System (GPS) is a satellite-based system that uses satellites and ground stations to measure and compute its position on Earth. GPS is also known as Navigation System with Time and Ranging (NAVSTAR) GPS. GPS receiver needs to receive data from at least 4 satellites for accuracy purpose. GPS receiver does not transmit any information to the satellites [21].

4.4.1.1.6 GY-86 10 DOF IMU

This IMU 10DOF is a motion tracking module. Its design is based on the sensor MPU6050, HMC5883L and MS5611.The sensor MPU6050 which is the world's first integrated 6-axis Motion Tracking device, that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP). The Honeywell's HMC5883L which is a 3-axis digital compass. The MS5611 is a high-accuracy chip to detect barometric pressure and temperature.

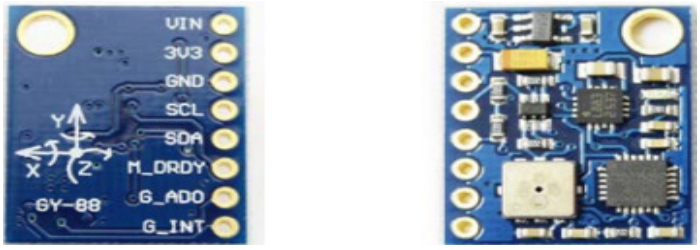


Figure 4.2: GY-86 10DOF IMU.

4.4.1.1.7 GPS (Ublox NEO-6M)

The GPS used in the quadcopter is the *Ublox NEO-6M* which uses UART protocol to communicate with the microcontroller.

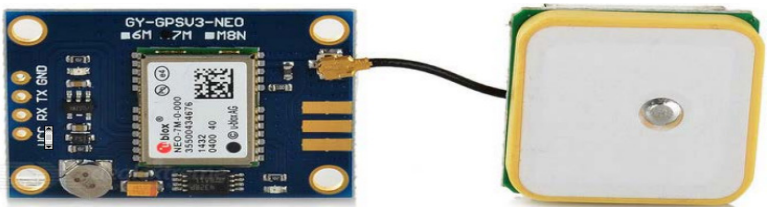


Figure 4.3:GPS Ublox NEO-6M

4.4.1.1.8 Ultrasonic

The barometer is very inaccurate in altitude less than five meter due to humidity and ground air turbulence, for this reason the ultrasonic sensor (HC-SR04) is introduced to calculate the low altitude accurately.

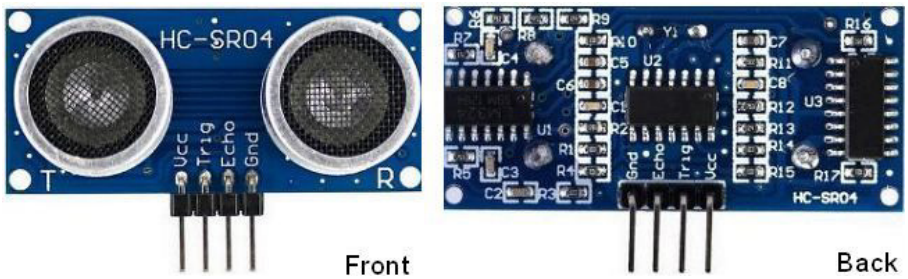


Figure 4.4:Ultrasonic HC-SR04

4.4.2 Microcontroller (Arduino Mega 2560)

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

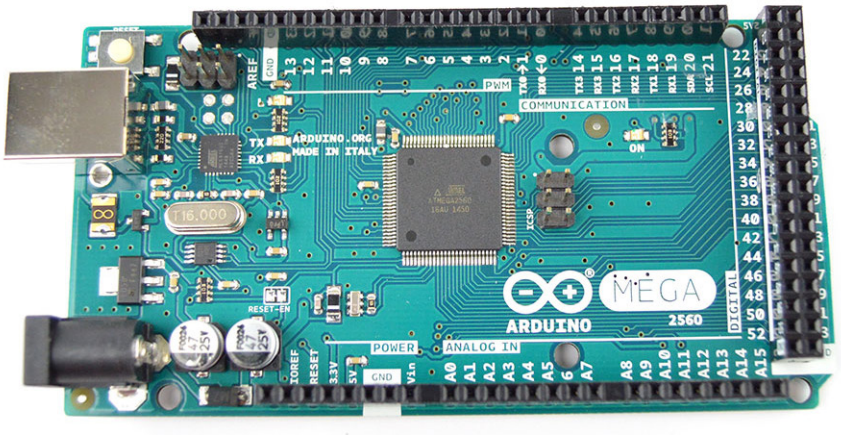


Figure 4.5:Arduino Mega 2560 .

4.4.3 Battery

The purpose of a battery is to store energy and release it at a desired time. Lithium polymer batteries, more commonly known as LiPo, have high energy density, high discharge rate and light weight which make them a great candidate in RC applications.

LiPo batteries used in RC are made up of individual **cells** connected **in series**. Each cell has a nominal voltage of **3.7V**. Therefore, battery voltage is often referred to as how many cells in the battery (aka "S").

Figure 4.6: Typical Lipo battery

LiPo battery is designed to operate within a safe voltage range, from **3V to 4.2V**. However, it's advisable to stop discharging when it reaches 3.5V for battery health reasons.

The capacity of a LiPo battery is measured in mAh is basically an indication of how much current you can draw from the battery for an hour until it's empty. Increasing battery capacity might give you longer flight time, but it will also get heavier in weight and larger in physical size. There is a trade-off between capacity and weight, that affects flight time and agility of the aircraft.

The C rating (or discharge rate) of a LiPo battery is a measure of the safe and continuous maximum discharge current.

$$\text{Max Discharge Current} = \text{C} - \text{Rating} \times \text{Capacity}$$

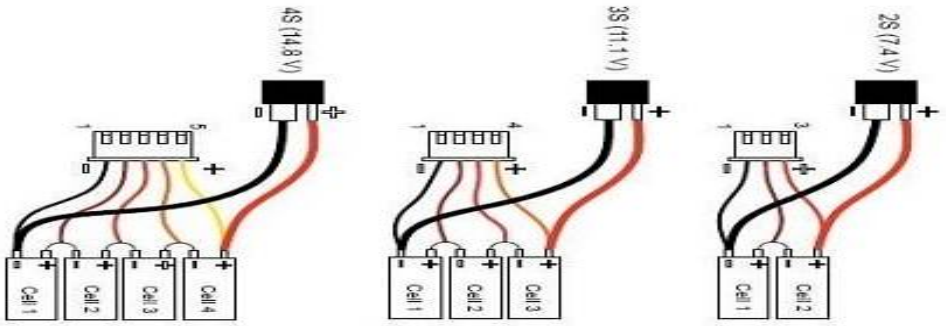


Figure 4.7:Typical wiring in side Lipo battery

Larger capacity batteries can usually supply more current as their internal electrodes have a greater surface area.

4.4.4 Actuators

4.4.4.1 Motors

Direct current (DC) motors are nearly universally used in RC aircraft, helicopters, and multirotor craft. The two main DC motor types are brushed (BDC) and brushless (BLDC). Brushless motors are preferred for use in quadcopters because they do not use carbon brushes, which makes them much easier to maintain. They also rotate at very high speeds, as compared to brushed motors, and produce less electrical noise.

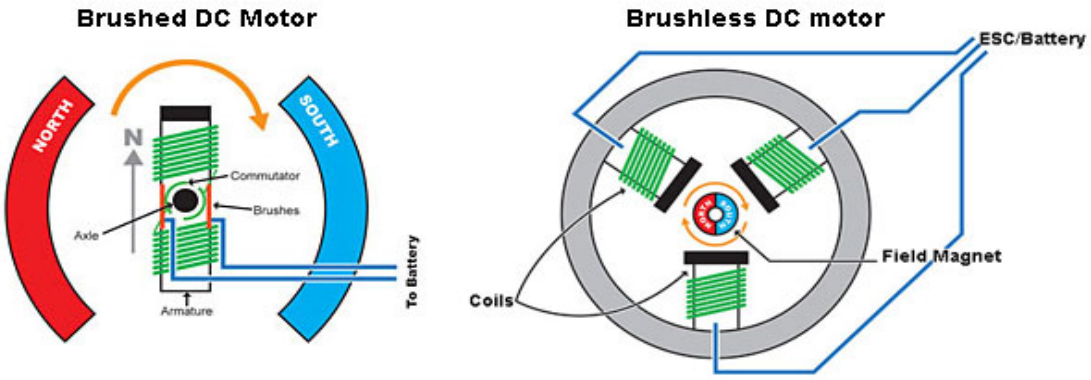


Figure 4.8: Brushed motor vs brushless motor

Brushed motors spin the coil inside a case with fixed magnets mounted around the outside of the casing. Brushless motors do the opposite; the coils are fixed either to the outer casing or inside the casing while the magnets are spun.

There two categories of brushless motor:

- **In-runner**: these have the fixed coils mounted to the outer casing and the magnets are mounted to the armature shaft which spins inside the casing.
- **Out-runner**: these have the magnets mounted on the outer casing which is spun around the fixed coils in the center of the motor casing (the bottom mounting of the motor is fixed).

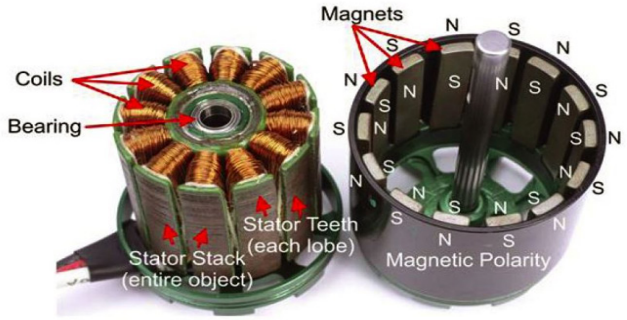


Figure 4.9: Out-runner Brushless motor components

In-runner motors are often used with R/C cars as they can spin much faster than out-runner motors. However out-runner motors can produce more torque which allows them to drive larger propellers used on aircraft and multirotor.

Kv is the number of revolutions per minute (rpm) that the motor will turn when 1V (one Volt) is applied with no load attached to the motor ($Kv = \frac{rpm}{Volt}$).

Kv allows us to get a handle on the torque that can be expected from a motor. Torque is determined by the number of winds on the armature and the strength of the magnets. A low Kv motor has more winds of thinner wire—it will carry more volts at fewer amps, produce higher torque, and swing a bigger propeller. A high Kv motor has fewer winds of thicker wire that carry more amps at fewer volts and spin a smaller propeller at high revolutions.

A rule of thumb is that the dronie should be able to at least provide two times the amount of thrust than the weight of the quadcopter ($Thrust\ per\ motor = \frac{weight \times 2}{4}$).

To explain these things a bit more, let's consider this example. If we have a quadcopter with an overall weight of about 1 Kg, the total thrust that the motors should be able to produce at maximum throttle should be 2 Kg at the minimum or say 500 g per motor in case of a quadcopter.

4.4.4.2 Electronic Speed Controller ESC

An ESC is an electronic circuit that allows the flight controller to control the speed and direction of a motor. The ESC must be able to handle the maximum current which the motor might consume, and be able to provide it at the right voltage. Most ESCs used in the hobby industry only allow the motor to rotate in one direction, though with the right firmware, they can operate in both directions.

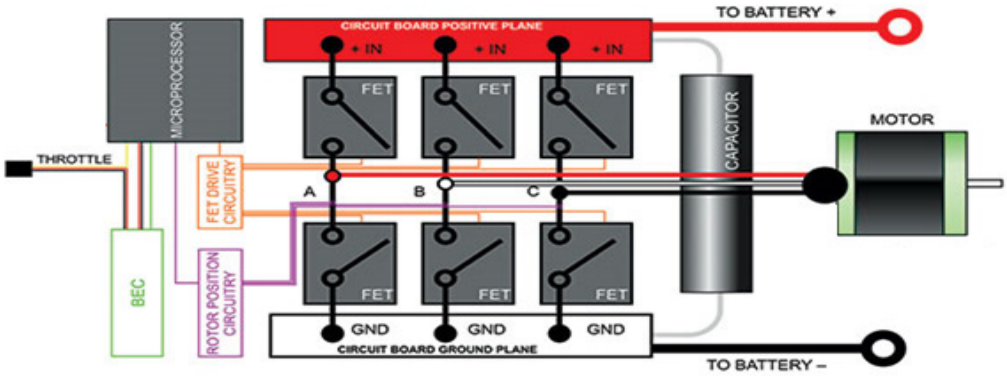


Figure 4.10: Simplified circuit of electronic speed control

The ESC has three sets of wires; one set of thick wires pair (normally, red and black) goes to the batteries, one set include 3-Pin R/C Servo Connector -and it accepts the RC signal- goes the flight controller, and one set of a 3 Bullet Connectors goes to the motors.

There are four main functional groups in an ESC: the power MOSFETs, the MOSFET driver circuitry, the microprocessor, and the motor position detection circuitry. A Battery Eliminator Circuit (BEC) is present in some controllers; it is designed to reduce the voltage of motor batteries to a level that is useful to the flight controller and onboard components of the quadcopter in general. Typically, BEC provides a voltage of 5V with the max current rating defined by the max amps that the motor can draw.

The ESC takes the battery power and ration it to the motor based on control signals and information from a rotor position circuit. The control signals come as PWM (Pulse Width Modulation) from a receiver or intermediate device such as a flight controller.

The ESC has a small microcontroller on board that reacts to that frequency, performs a little logic on it, and sends appropriate power to the motor by managing the operation of field-effect transistors (FETs).

Motor Position Detection Circuitry of the ESC must know the precise location of the rotor magnet(s) to accurately sequence the connections that the FETs make. There are two main ways to go about this: sensored and sensorless. Sensored systems use a magnetic

"Hall Effect" sensor or "Optical" detector in the motor to track the rotor. This requires additional parts in the motor (sensors) and an additional wiring harness to connect the motor sensors to the controller. Sensorless: modern ESCs detect the rotor position through the power wires, while only 2 of the wires are energized by the ESC at any one time. The pole that is not energized at any specific instant will generate a small amount of voltage that is proportional to how fast the motor is spinning; this is known as "Back Electromotive Force" (Back EMF). This small voltage is used by the ESC to determine how fast and in what direction the motor is rotating at any given time. Then the information is used to switch FETs as needed to cause correct magnetic push or pull in the phases.

4.4.5 Bluetooth Wireless Communication

To connect the smartphone as remote controller a Bluetooth wireless communication is established by adding HC-05 Bluetooth to circuitry implementation.

4.4.5.1 Bluetooth HC-05

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. The HC-05 Bluetooth Module can be used in a Master or Slave configuration, making it a great solution for wireless communication.

This serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband.

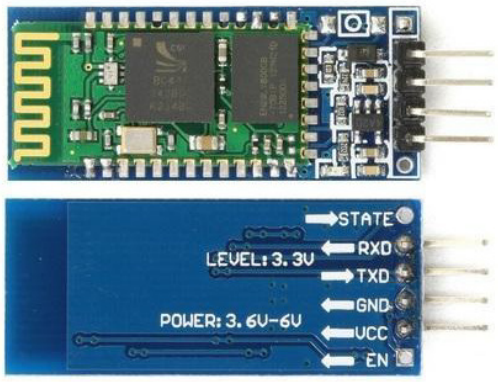


Figure 4.11: Bluetooth module HC-05

4.5 Overall System Implementation

4.5.1 System Circuitry

In this section, all the electronical components are assembled together as shown in the next figure (4.12). The necessary connections are performed between the IMU, GPS, Ultrasonic, Bluetooth, microcontroller and actuator using the suitable techniques of communication. More details are post in Appendix D.

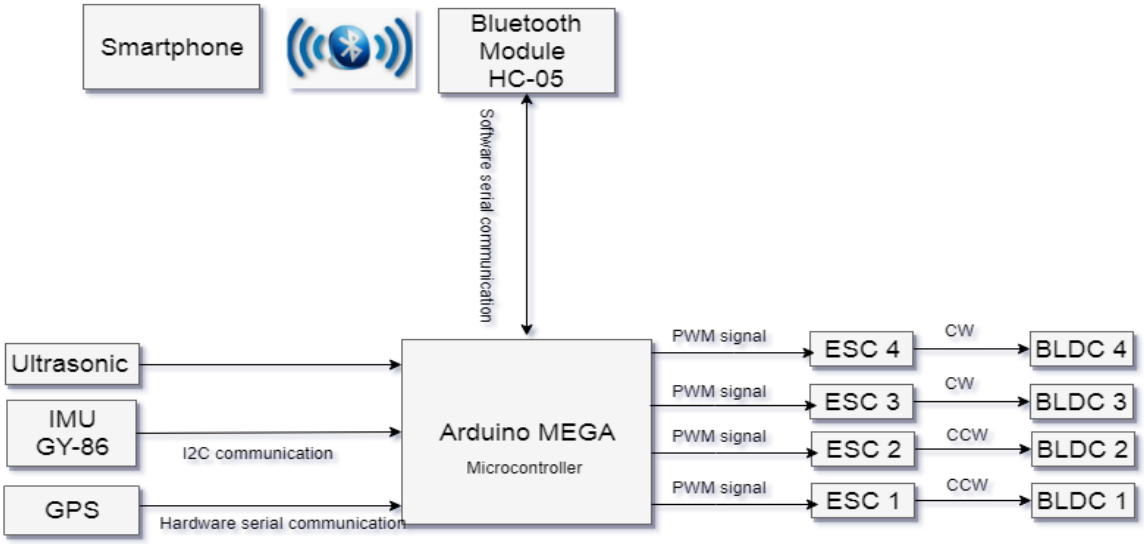


Figure 4.12: Over all hardware connections

4.5.2 Quadcopter Full Specification

After implementing all the quadcopter hardware and uploaded the appropriate software to the flight controller, the summary in table gives the full specifications.

Table 4-2: Quadrotor specifications.

Components	Characteristics	Weight (g)
Frame f450	'X' configuration	285.7
BLDC ×4 A2212/T13	1000kv	52.7×4
ESC ×4	30A	32×4
Propeller ×4	'10'x'45' CW and CCW	7×4
Flight controller	Arduino Mega 2560	36
GY-86	Acetometer, Gyroscope, Barometer, Magnetometer	4
Ultrasonic	5 meter of range limits	8.5
GPS	Ulbox NEO-6M	12
LIPO Battery	4900mah 3S	350
Breadboard	Wire connection	15
Total Weight (g)		1078

4.5.3 Software Development

In the literature overview there exist many open sources code for flight controller of quadrotor (Multiwii, Arduicopter …), that are generally based on PID controllers. However, the purpose of our work is to develop both hardware and software parts of the autonomous quadrotor.

The flowchart in the figure (4.13) illustrates global consecutive steps in the SMC controller algorithm programming (the detailed code is posted in Appendix E). At the beginning, the data is obtained from sensors (GPS and IMU) and desired trajectory from the user (phone application figure 4.14). Then the controller processes the data and wait the order from the user to start navigating (GO button is pushed or GO = true). The quadcopter travels until it reaches the desired altitude, then it surveys all the four linked desired points, after that it returns to the initial position and eventually land-in. However, if STOP button is pressed (STOP = true) for any reason, the controller breaks navigation and landing the quadcopter smoothly.

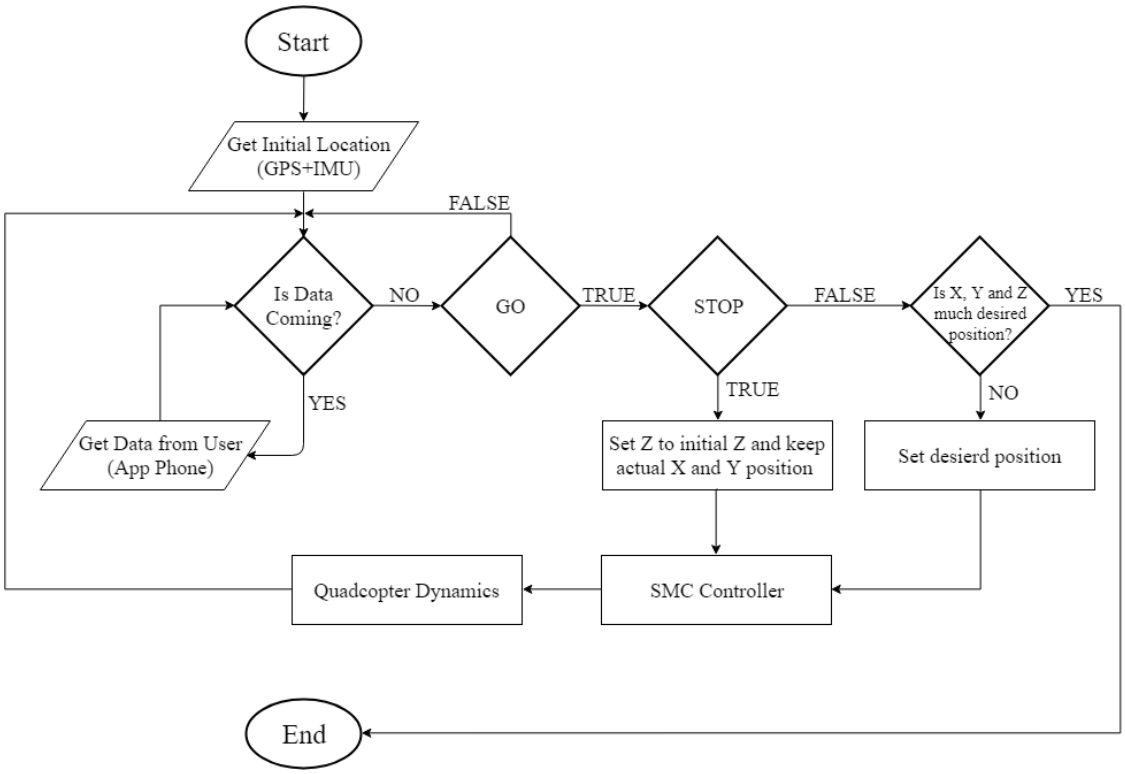


Figure 4.13: Flowchart of the developed code for Quadrotor

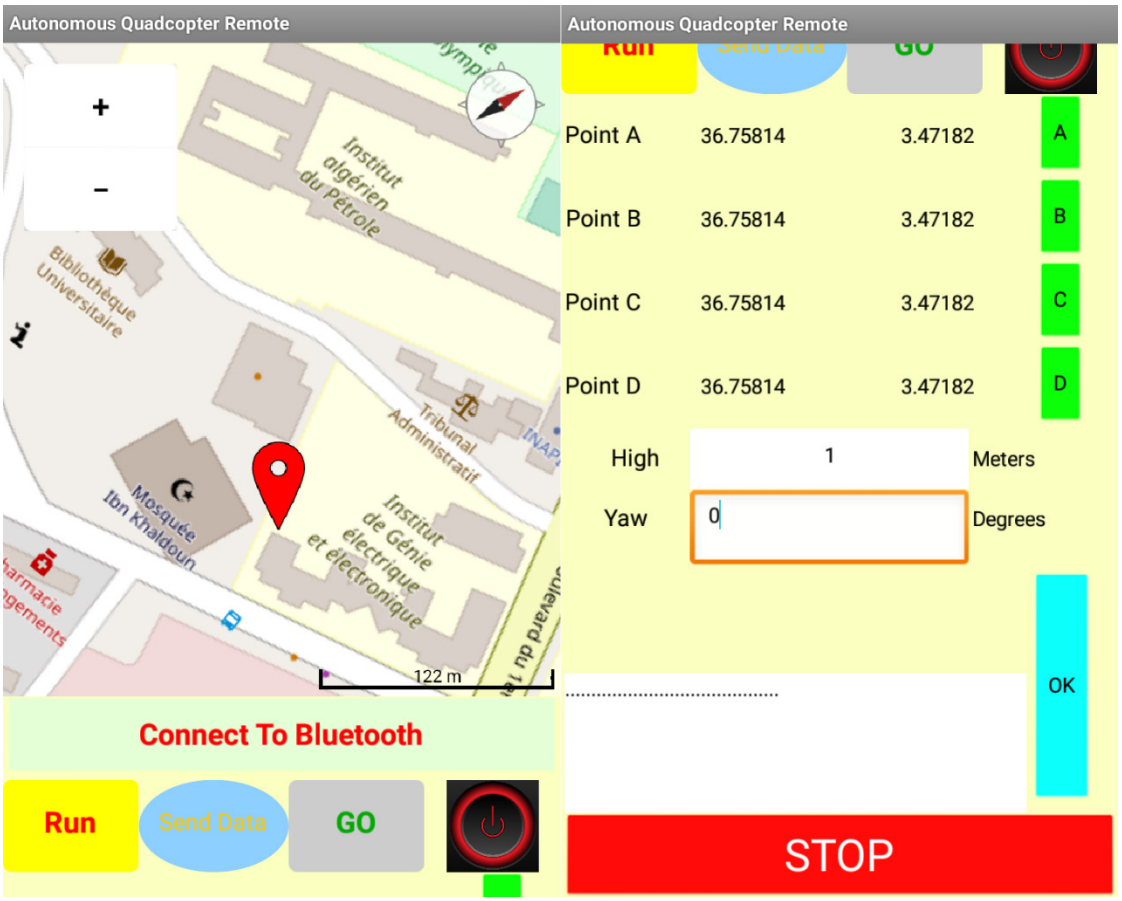


Figure 4.14: Autonomous quadrotor android application.

4.6 Problems and Difficulties

It is imperative to deal with obstacles in any project implementation. In our case, several problems and difficulties were facing us which listed as follow:

- *Cost:* the material and components needed are relatively expensive at the first place, add to that any mistake that may occur increases the total cost.
- *Programming:* obtained data from all required sensors is not an easy task. Every sensor must be working properly and provide acceptable data. Moreover, inserting appropriate equations of the SMC model with all necessary parameters was proved to be quite difficult especially the derivatives' calculation of actual and desired states. Finally, the reference trajectory must be generated from desired points that the user prefers (by Phone Application).
- *Implementation:* the lack of testing environment was marked as a weakness that inhibited the advancements of the project implementation.

4.7 Conclusion

In this chapter, all the components included in our system are perfectly described in order to understand the functionality of each one, and then a proper manipulation is reached. Next, the entire system is assembled together with appropriate circuitry and specifications. Furthermore, programming of the controller and phone application takes considerable part of the implementation process, therefore, software development is a critical phase in our project.

Conclusion

In this project we explored a portion for the use of flying robots with trajectory tracking techniques however this needs a lot of effort in order to achieve a fully autonomous system for the applications discussed in the previous chapters.

To implement the quadcopter, we needed to understand its concept of operation theoretically so we passed by the modeling and controller design methods where we covered the mathematical theory behind the quadrotor kinematics and dynamics, this led to the controller design part where we used two techniques(nonlinear and linear) which are sliding mode control (SMC) and proportional derivative (PD) to achieve the stability of the six DOF of the system.

After discussing the mathematical modelling, a simulation is performed in MATLAB/Simulink to verify and optimize the parameter of the system in the virtual environment.

We interfaced all the sensor and actuator parts with the microcontroller, then we designed the control algorithm that is responsible for stabilizing and maneuvering the quadcopter based (SMC) from hovering to rotations and translations, in addition to a ground control station to communicate with the UAV for automatic piloting.

The contribution of this project was to create a system that consists of a quadcopter with the smartphone that can perform trajectory tracking, this full system can minimize the risks of human being intervening in such missions, it will also minimize the time and cost for this task.

As a future work, we can design a better performance SMC controller, also implementing a reliable communication channel with the ground control station using Xbee modules, and finally adding a depth camera or laser scanner to create a 3D reconstruction environment application like photogrammetry.

References

[1] Mokhtar, M., Matori, A., Yusof, H., Chandio, I., Viet, D.,, "A study of unmanned aerial vehicle photogrammetry for environment mapping: Preliminary observation," *In Advanced Materials Engineering,* vol. 626 of Advanced Materials, pp. 440-444, 2012.

[2] Berni, J., Zarco-Tejada, P., Suarez, L., and Fereres, E, "Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle," *IEEE,* Vols. Transactions on Geoscience and Remote Sensing, vol.47, no.3, pp. 722-738, 2009.

[3] Bednowitz, N., Batta, R., and Nagi., R, "Dispatching and loitering policies for unmanned aerial vehicles under dynamically arriving multiple priority targets.," *Journal of simulation,* vol. 8, no. 1(ISSN: 1747-7778), pp. 9-24, 2014.

[4] Andres Hernandez , Harold Murcia , Cosmin Copot and Robin De Keyser, "Model Predictive Path-Following Control of an AR.Drone Quadrotor," in *Memorias del XVI Congreso Latinoamericano de Control Automático, CLCA 2014*, Cancún, Quintana Roo, México, 2014.

[5] Andrew Zulu*, Samuel John, "A Review of Control Algorithms for Autonomous Quadrotors," *Open Journal of Applied Sciences,* vol. 4, pp. 547-556, 2014.

[6] S. BOUABDALLAH, "Design And Control of Quadrotors With Application To Autonomous Flying," EPFL, Lausanne, 2007.

[7] E. Schulken, "Investigations of Model-Free Sliding Mode Control Algorithms including Application to Autonomous Quadrotor Flight," Department of Mechanical Engineering, Rochester, New York, 2017.

[8] J. U. A. Muñoz, Modeling and control of VTOL, 2017.

[9] P.K. Sekhar, V. Uwizeye, "Review of sensors and actuator mechanisms for bioMEMS.," *Elsevier journal,* vol. MEMS for Biomedical Applications., pp. 46-77, 2012.

[10] Q. Quan, Introduction to mulitirotor control design, Beihang University Beijing China: springer, 2007.

[11] H. t. M. N. ElKholy, Dynamic Modeling and Control of a Quadrotor Using Linear and Nonlinear Approaches,Master thesis, CAIRO, Spring 2014.

[12] J. Tang, S. Yang, Q. Li, "Attitude Control of a Class of Quad-Rotor Based on LADRC," *Lecture Notes in Electrical Engineering,* vol. 460, pp. 157-166, 2018.

[13] L.R.G.Carrillo, R.Lozano, A.E.D.López and C.Pégard, Quad Rotorcraft Control Vision-Based Hovering and Navigation, Mexico, French: Springer London Heidelberg New York Dordrecht, 2013.

[14] R. S. Samir Bouabdallah, "Full Control of a Quadrotor," in *Conference on Intelligent Robots and Systems*, San Diego, CA, USA, 2007.

[15] I. Kugelberg, "Black-Box Modeling and Attitude Control of a Quadcopter," Division of Automatic Control Department of Electrical Engineering Linköping University, Linköping, Sweden, 2016.

[16] J-J.E. Slotine and W. Li, Applied nonlinear control, Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

[17] J.-J. S. a. W. Li, Applied nonlinear control, Prentice-Hall, 1991.

[18] N.B. Ammar · S. Bouallègue, N.B. Ammar, J. Haggège, S. Bouallègue, S. Vaidyanathan, "Chattering Free Sliding Mode Controller Design for a Quadrotor Unmanned Aerial Vehicle," *Studies in Computational Intelligence,* vol. 709, no. DOI 10.1007/978-3-319-55598-0_3, pp. 61-79, 2017.

[19] J. Guldner, V. I. Utkin, "The chattering problem in sliding mode systems," 2015.

[20] John P. Rafferty, "Britannica," The Editors of Encyclopaedia Britannica, 27 jun 2017. [Online]. Available: https://www.britannica.com/technology/barometer.

[21] "Electronicwings," [Online]. Available: https://www.electronicwings.com/sensors-modules/gps-receiver-module.

[22] W. Selby, "Wilselby," [Online]. Available: https://www.wilselby.com/research/arducopter/model-verification/.

[24] M. Walid, N. Slaheddine, A. Mohamed and B. Lamjed, "Robust internal model controller for quadrotor UAV," in *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, Hammamet, Tunisia, June2018.

[25] T. Bresciani, Modelling, Identification and Control, October 2008.

[26] A. P. G. C. C. Marco Herrera, "Sliding Mode Control: An approach to Control a Quadrotor," *Asia-Pacific Conference on Computer Aided System Engineering,* 2015.

[27] S. W. Q. L. Y. Z. S. L. Hang Yu, "Robust Integral Sliding Mode Controller for Quadrotor Flight," *IEEE,* 2015.

[28] S. BOUABDALLAH, "Design and Control of Quadrotors With Application to Autonomous Fly," EPFL, Lausanne, 2007.

Appendix A: Parameter Identification

In order to identify the parameters of the dynamic model an approximate formula is used [22]:

- I_{xx}= quadrotor moment of inertia in $x_b (kg - m^2)$.
- I_{yy}=quadrotor moment of inertia in $y_b (kg - m^2)$.
- I_{zz}=quadrotor moment of inertia in z_b.
- m_s =mass of the stack-up frame and electronics (kg).
- r_s=radius of the stack-up (m).
- r_m=mass of motor (kg).
- r=radius of the motor(m).
- h=height of the motor(m).
- l=length of the arm (m).
- I_r=$\frac{1}{12} m_m (3r^2 + h^2)$.

And the equation stack-up is given by $\frac{2}{3}\ m_s \times r_s^2$ this provide us the following approximation.

$$I_{xx} = I_{yy} = \frac{2}{3} m_s^2 + 2\left(\frac{1}{12} m_m (3r^2 + h^2) + m_m l^2\right).$$

For the inertia the z axis.

$$I_{zz} = \frac{2}{3} m_s r_s^2 + 4\left(\frac{1}{2} m_m r^2 + m_m l^2\right)$$

The thrust factor it is based on the following formula.

$$b = \frac{mg}{4\omega_h}$$

Where:

ω_h :is the angular speed of motor at hovering condition.

m : the mass of the quadrotor.

g :gravitational acceleration.

Appendix B: Quadcopter Component Specifications

B.1 Brushless DC motor

The motors used in this project are 1000KV, a full specification is given in figure (B.1) taken from the datasheet.

No. of Cells:	2 - 3 Li-Poly 6 - 10 NiCd/NiMH
Kv:	1000 RPM/V
Max Efficiency:	80%
Max Efficiency Current:	4 - 10A (>75%)
No Load Current:	0.5A @10V
Resistance:	0.090 ohms
Max Current:	13A for 60S
Max Watts:	150W
Weight:	52.7 g / 1.86 oz
Shaft Diameter:	3.2 mm
Poles:	14
Model Weight:	300 - 800g / 10.5 - 28.2 oz

Figure B. 1: Brushless DC motor full specification

B.2 Electronic Speed Controller

This is fully programmable 30A BLDC ESC with 5V, 3A BEC. Can drive motors with continuous 30Amp load current. It has sturdy construction with two separate PCBs for Controller and ESC power MOSFETs. It can be powered with 2-4 lithium Polymer batteries or 5-12 NiMH / NiCd batteries. It has separate voltage regulator for the microcontroller for providing good anti-jamming capability. It is most suitable for UAVs, Aircrafts and Helicopters.

Figure B. 2: Electronic Speed Controller

With the following specifications:

Output	30A continuous; 40Amps for 10 seconds
Input voltage	2-4 cells Lithium Polymer / Lithium Ion battery
BEC	5V, 3Amp for external receiver and servos
Max Speed	2 Pole: 210,000rpm; 6 Pole: 70,000rpm; 12 Pole: 35,000rpm
Weight	32gms
Size	55mm x 26mm x 13mm

According to the datasheet it has the next features:

- High quality MOSFETs for BLDC motor drive
- High performance microcontroller for best compatibility with all types of motors at greater efficiency

- Fully programmable with any standard RC remote control
- Heat sink with high performance heat transmission membrane for better thermal management
- Three start modes: Normal / Soft / Super-Soft, compatible with fixed wing aircrafts and helicopters
- Throttle range can be configured to be compatible with any remote control

B.3 Frame

The next figure(B-3) provides an overview of the frame specification used in the implementation.

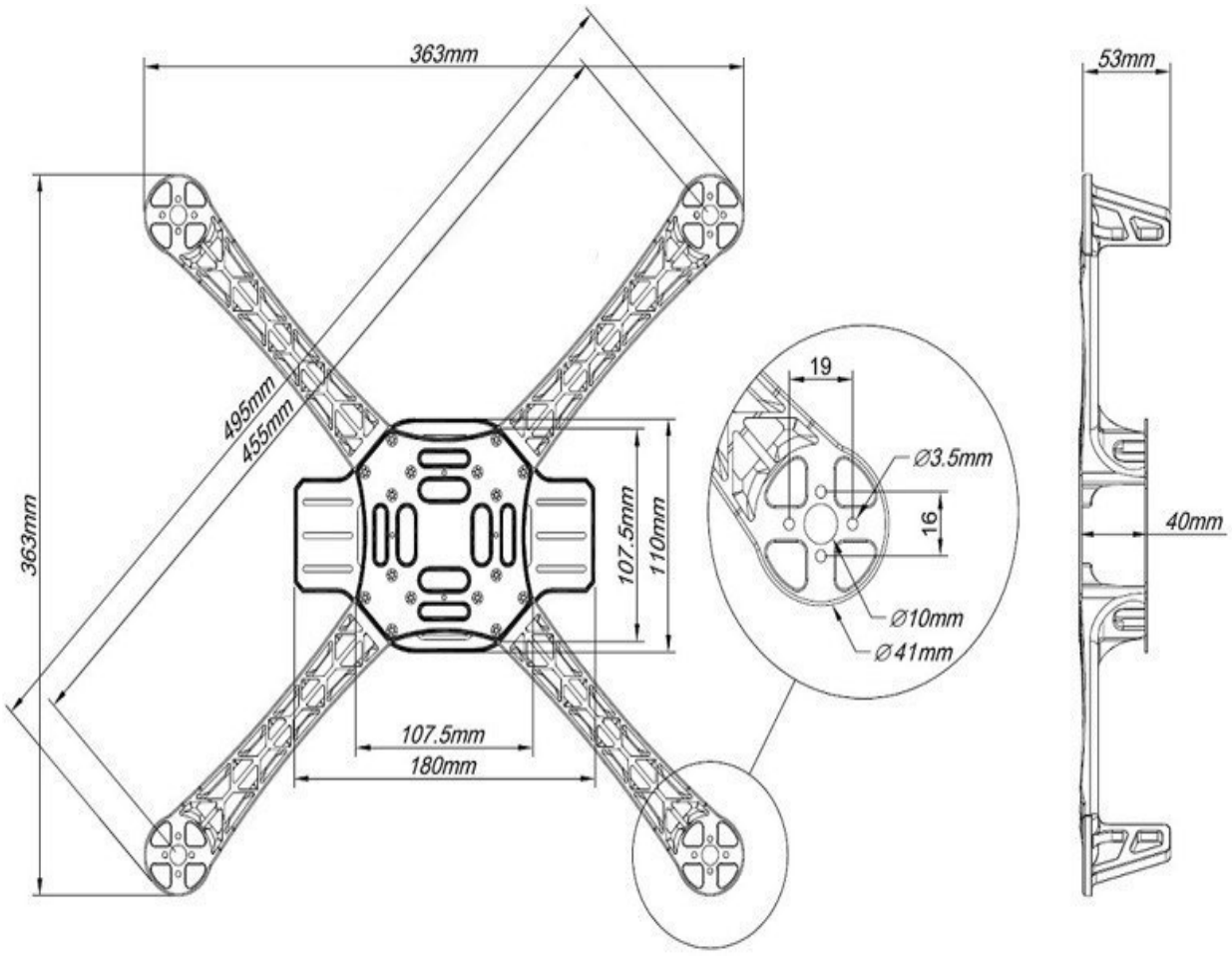


Figure B. 3: Frame specification

Appendix C: Simulation Turning Parameter.

Process optimization of the parameter of both (PD and SMC) are based on the genetic algorithm to get better performance following the next steps:

- Select the output which we are going to optimize as in the next figure (C- 1)

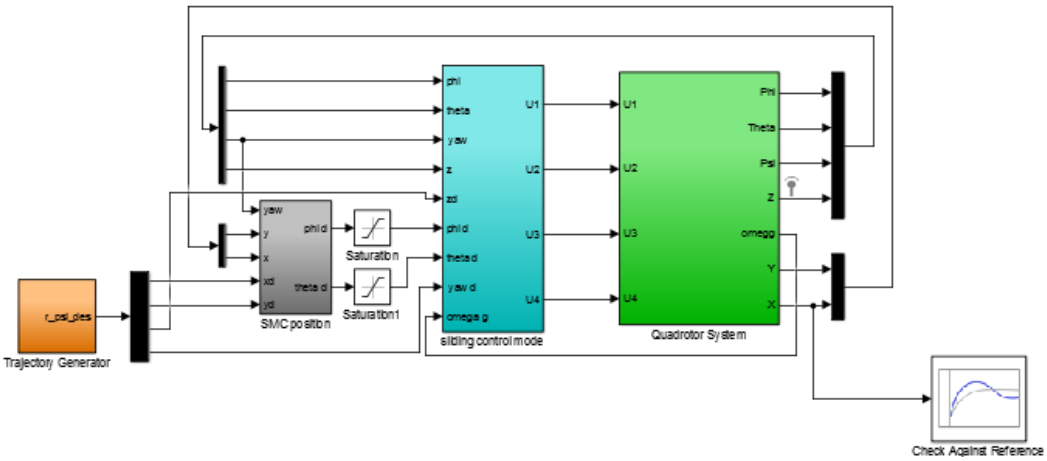


Figure C. 1: System with optimization block diagram

- After that select the parameter of that output as in the figure (C-2).

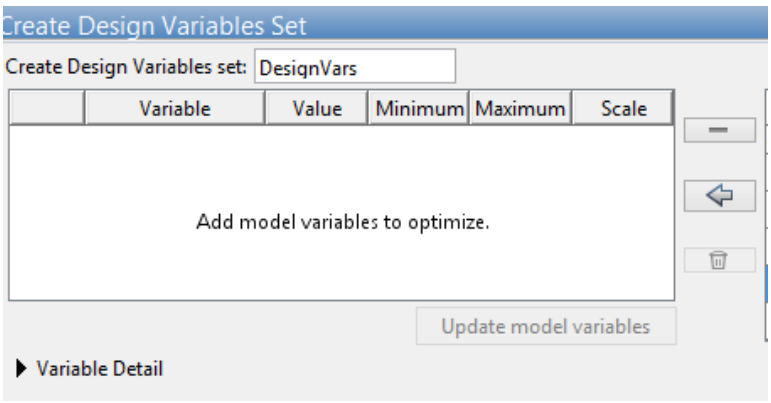


Figure C. 2: Parameters selection for optimization

- Select the GA algorithm as optimization options as in the figure (C-3).

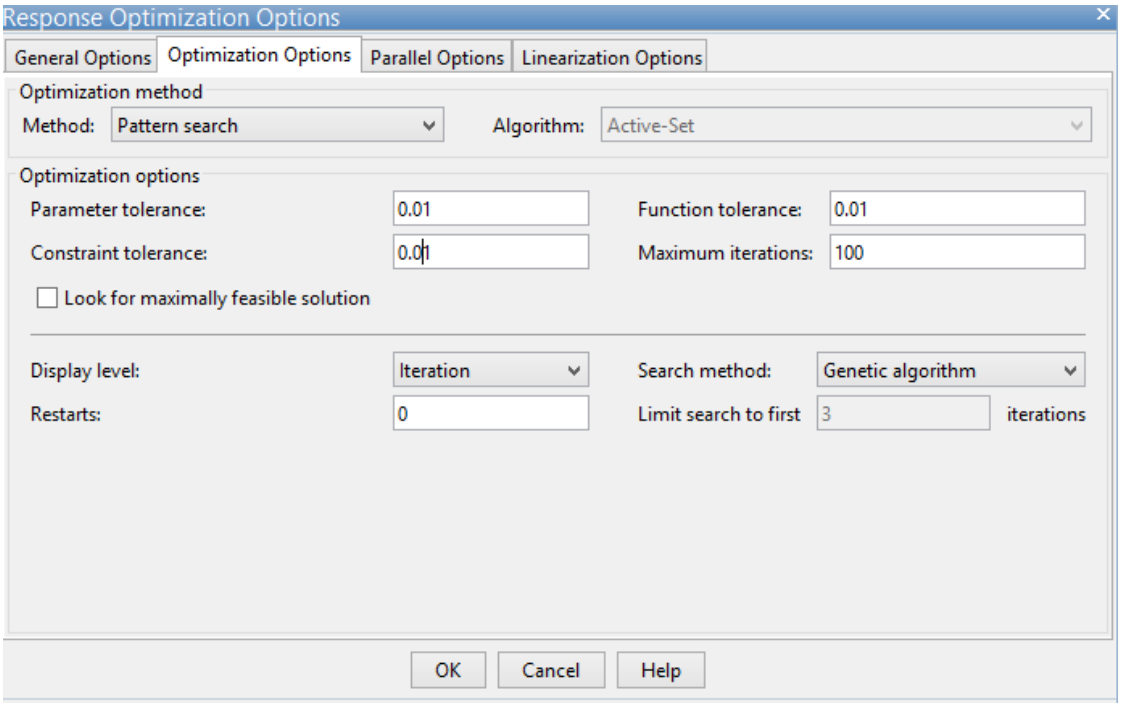


Figure C. 3: Selection of GA algorithm

- Starting the process by pushing the button optimize as in the figure (C-4).

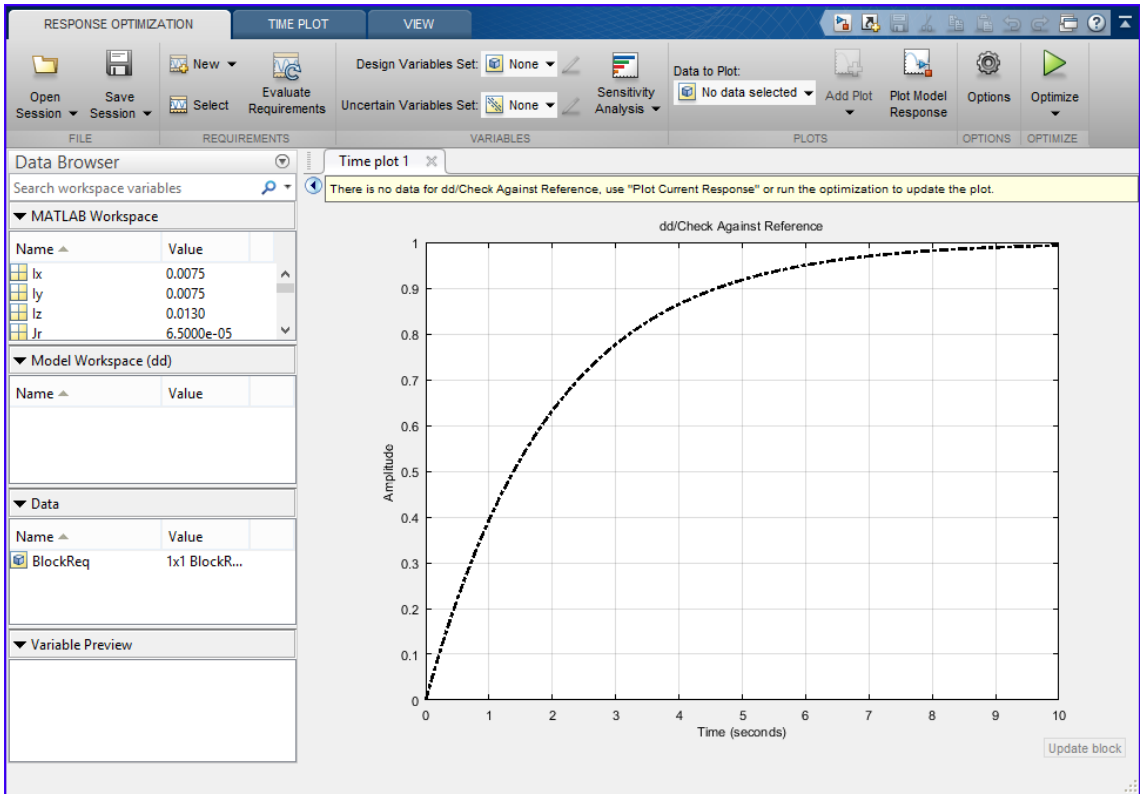


Figure C. 4: The optimization process

Appendix D: Hardware Part.

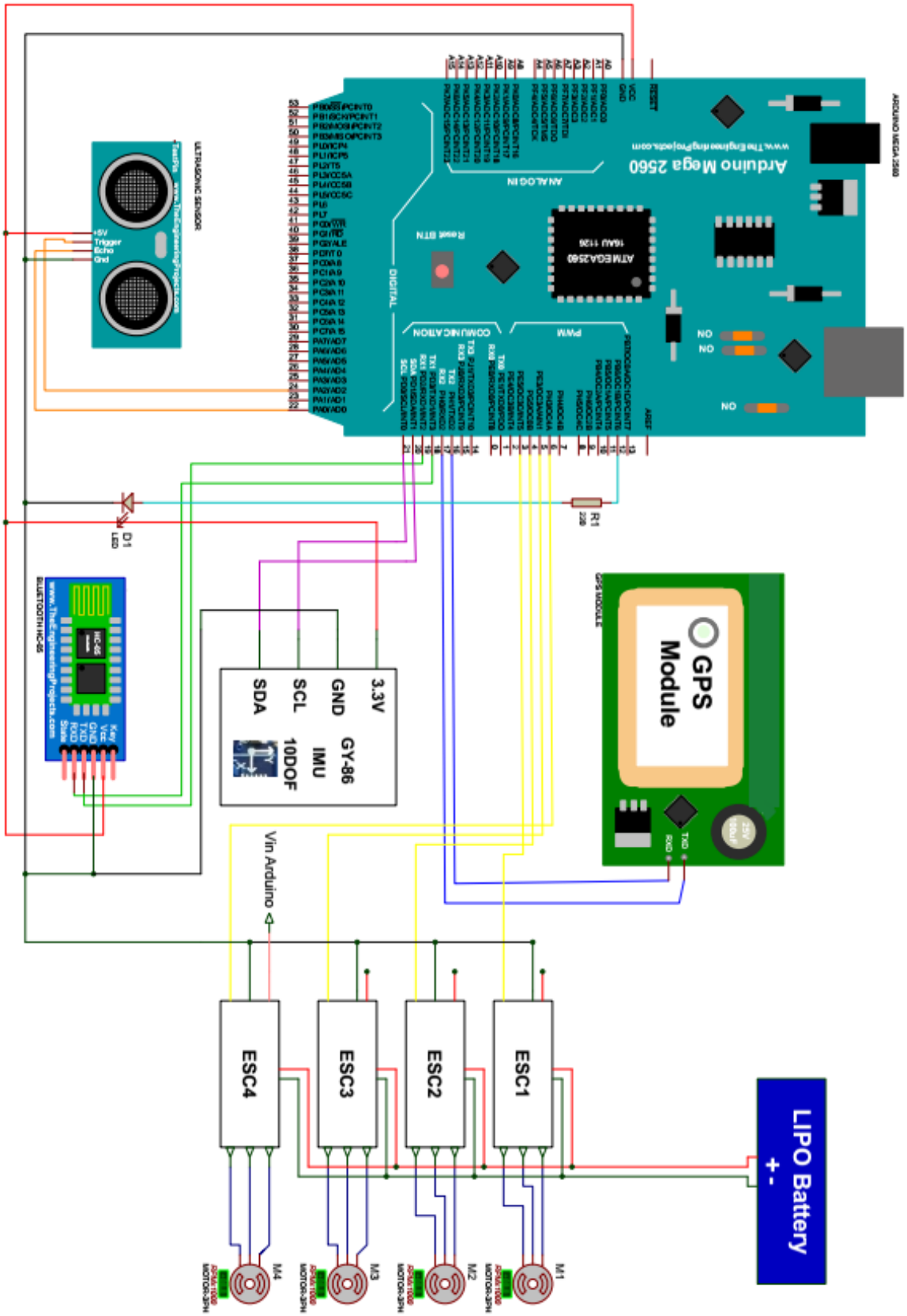


Figure D. 1: Overall circuit implementation

The previous figure (D-1) shows the connection between different components of quadrotor based on the table(D-1).

Table D-1:Hardware implementation pin-out

Devices		Arduino Mega 2650
Name	**Pinout**	**Pinout**
Ultrasonic	ECO	24
	TRIG	22
GY-86	SCL	SCL
	SDA	SDA
GPS Module	RX	TX2
	TX	RX2
Bluetooth HC-05	RX	TX1
	TX	RX1
ESC1	PWM	3
ESC2	PWM	4
ESC3	PWM	5
ESC4	PWM	6
LED	Positive	12

Appendix E: The Programming Code

In this Appendix, the developed controller program is demonstrated.

```
//for ESC devices
#include <Servo.h>
#include "ESC.h"
//for IMU 10dof  sensor
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include "HMC5883L.h"
#include <MS5611.h>
//for IMU sensor fusion
#include "SensorFusion.h" //SF
//for GPS module
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

//Configuration pins:
#define esc1_pin 3
#define esc2_pin 4
#define esc3_pin 5
#define esc4_pin 6

#define SPEED_MIN (1000)
#define SPEED_MAX (2000)
#define ARM_VALUE (500)

#define imu_SCL A5
#define imu_SDA A4

#define trigPin 24
#define echoPin 22

#define LEDs 12

//Declaration of globale variables:
// ESC_Name (PIN, Minimum Value, Maximum Value, Arm Value)
ESC esc1 (esc1_pin, SPEED_MIN, SPEED_MAX, ARM_VALUE);
ESC esc2 (esc2_pin, SPEED_MIN, SPEED_MAX, ARM_VALUE);
ESC esc3 (esc3_pin, SPEED_MIN, SPEED_MAX, ARM_VALUE);
ESC esc4 (esc4_pin, SPEED_MIN, SPEED_MAX, ARM_VALUE);

//IMU and fusion objects
MPU6050 accelgyro;
HMC5883L mag;
MS5611 baro;
SF fusion;

// The TinyGPS++ object
TinyGPSPlus gps;

float Latv, Lonv, Lat_R, Lon_R, Alt, Alt_R;
float Lat[] = {0,0,0,0}, Lon[] = {0,0,0,0}, High;
float X, Y, Z, Xd, Yd, Zd, Roll,Roll_d, Pitch, Pitch_d, Yaw ,Yaw_d, Yaw_R;
float dX, dY, dZ, dX_d=1, dY_d=1, dZ_d=1, ddX_d=0, ddY_d=0, ddZ_d=0;
float X_d[]={0,0,0,0,0,0,0,0},Y_d[]={0,0,0,0,0,0,0,0},Z_d[]={0,0,0,0,0,0,0,0};
float dRoll, dPitch, dYaw, dRoll_d, dPitch_d, dYaw_d, ddRoll_d, ddPitch_d, ddYaw_d=0;
const float C_x=1.02, C_y=1, C_z=2.05, K_x=1, E_x=0.5, K_y=1, E_y=0.5, K_z=2, E_z=9;
const float C_roll=5.01, K_roll=5, E_roll=0.1, I_r=6.5*pow(10,-5), E_pitch=0.1, C_pitch=5;
const float K_pitch=5, E_yaw=0.124, C_yaw=6, K_yaw=8, I_xx=0.013187, I_yy=0.013187, I_zz=0.018697;
float Sx, Sy, Sz, Sroll, Spitch, Syaw;
float U1, U2, U3, U4;

const float Pi=3.141592653589793, g=9.81, m=1, l=0.2275;

float b=4.8444*pow(10,-5),d=7.5*pow(10,-7); //The areodynamic force and moment constants
float Omg1, Omg2, Omg3, Omg4, Omg_r=0; //The RPM speed of the motors
float VO1, VO2, VO3, VO4; //The scaled RPM speed to signal period

boolean GO = false, STOP=false;
```

```
void setup() {
  Wire.begin();
   accelgyro.setI2CMasterModeEnabled(false);
   accelgyro.setI2CBypassEnabled(true);
   accelgyro.setSleepEnabled(false);
  Serial.begin(9600);
   Serial2.begin(9600);
   Serial1.begin(9600);
   accelgyro.initialize();
   mag.initialize();
     // Initialize MS5611 sensor
  while(!baro.begin()){}

   pinMode(LEDs, OUTPUT);
   digitalWrite(LEDs, LOW);

  //Send the Arm value
  esc1.arm();
  esc2.arm();
  esc3.arm();
  esc4.arm();
  esc1.speed(1000);
  esc2.speed(1000);
  esc3.speed(1000);
  esc4.speed(1000);
  //GY-86 pins configuration
  pinMode(imu_SCL, INPUT);
  pinMode(imu_SDA, INPUT);
  //Ultrasonic
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // Reference position
  Alt_R=BarometerAlt();
  while(Latv==0 || Lonv ==0){
  GPSDataLatLon();
  }
  Lat_R=Latv;
  Lon_R=Lonv;
  LatLonAltToXYZ(Latv, Lonv, Alt_R);
  X_d[0]=X_d[6]=X;
  Y_d[0]=Y_d[6]=Y;
  Z_d[0]=Z;
  IMUDataRollPitchYaw();
  Yaw_R = Yaw;
  Serial.println("READY");
  Serial1.write("READY");
}

void loop() {
    float Pretimer=0, timer=0;
    RecieveData();
    digitalWrite(LEDs, LOW);
    while(GO == true && STOP == false){
      digitalWrite(LEDs, HIGH);
      GPSDataLatLon();
      Alt = GetAlt();
      LatLonAltToXYZ(Latv, Lonv, Alt);
      Pretimer=micros(), timer=0;
      while(Z < Z_d[1] && GO==true){
      timer=(micros()-Pretimer)*pow(10,-6);
      Xd=X_d[0];
      Yd=Y_d[0];
      if(timer <= (Z_d[1]-Z_d[0])/dZ_d){
      Zd=dZ_d*timer+Z_d[0];
      }
      else{
        Zd=Z_d[1];
      }
      Yaw_d=Yaw_R;
      dYaw_d=0;
      ControlInputs();
      ControlMotors(U1,U2,U3,U4);
      RecieveData();
     }
     RecieveData();
     if(GO == false){break;}
     Serial1.write("ALT");
     for(int i=1;i<=6;i++){
      Pretimer=micros(), timer=0;
      while((abs(X - X_d[i])!= 2  || abs(Y - Y_d[i]) != 2) && GO == true){
      timer=(micros()-Pretimer)*pow(10,-6);
      if(timer <= max((X_d[i]-X_d[i-1])/dX_d, (Y_d[i]-Y_d[i-1])/dY_d)){
      if(Xd < X_d[i] && X_d[i] > X_d[i-1]){
        Xd=dX_d*timer+X_d[i-1];
      }
      else if(Xd > X_d[i] && X_d[i] < X_d[i-1]){
        Xd=-dX_d*timer+X_d[i-1];
      }
      if(Yd < Y_d[i] && Y_d[i] > Y_d[i-1]){
        Yd=dY_d*timer+Y_d[i-1];
      }
      else if(Yd > Y_d[i] && Y_d[i] < Y_d[i-1]){
        Yd=-dY_d*timer+Y_d[i-1];
      }
      }
      else{
        Xd=X_d[i];
        Yd=Y_d[i];
      }
```

```
      Zd=Z_d[i];
      if(i>1){
      Yaw_d += dYaw_d*timer;
      }
      ControlInputs();
      ControlMotors(U1,U2,U3,U4);
      RecieveData();
      }
      RecieveData();
      if(GO == false){break;}
      if(i==1){Serial1.write("POINT1");}
      else if(i==2){Serial1.write("POINT2");}
      else if(i==3){Serial1.write("POINT3");}
      else if(i==4){Serial1.write("POINT4");}
      else if(i==5){Serial1.write("POINT5");}
      else if(i==6){Serial1.write("POINT6");}
      }
      RecieveData();
      if(GO == false){break;}
      Pretimer=micros(), timer=0;
     while(Z > Z_d[0]){
      timer=(micros()-Pretimer)*pow(10,-6);
      Xd=X_d[6];
      Yd=Y_d[6];
      if(timer <= (Z_d[1]-Z_d[0])/dZ_d){
      Zd=-dZ_d*timer+Z_d[6];
      }
      else{
        Zd=Z_d[0];
      }
      Yaw_d=Yaw_R;
      ControlInputs();
      ControlMotors(U1,U2,U3,U4);
     }
     if(Z <= Z_d[0]){
           GO=false;
           esc1.speed(1000);
           esc2.speed(1000);
           esc3.speed(1000);
           esc4.speed(1000);
           Serial1.write("END");
           break;
      }
   }
}

//Emergency Stop
void EmergencyStop(){
  GPSDataLatLon();
  Alt = GetAlt();
  LatLonAltToXYZ(Latv, Lonv, Alt);
  float x=X, y=Y, z=Z;
  float t=0,pt;
    pt=micros();
    while(Z > Z_d[0]){
    t=(micros() - pt)*pow(10,-6);
    Xd=x;
    Yd=y;
    if(t < (z - Z_d[0])/dZ_d){
    Zd=-dZ_d*t+z;
    }
    else{
      Zd=Z_d[0];
    }
    Yaw_d=Yaw_R;
    ControlInputs();
    ControlMotors(U1,U2,U3,U4);
    }
    Serial1.write("END");
     esc1.speed(1000);
     esc2.speed(1000);
     esc3.speed(1000);
     esc4.speed(1000);
    GO = false;
    STOP = false;
}

//Recieving Data
void RecieveData(){
      while(Serial1.available()){
      String Data = "";
      char DATA[200];
      Data = Serial1.readString();
      Data.toCharArray(DATA, 200);
      if(GO == false){
      GetDesiredData(DATA);
      }
      GetPermission(DATA);
      Data.toCharArray(DATA, 200);
      if(Data.compareTo("") != 0){
        Serial1.write(DATA,200);
      }
      Data="";
      }
}

//Generate the control inputs
void ControlInputs(){
  DivXYZ();
  DivRollPitchYaw();
  //The sliding surfaces:
  Sx=C_x*(X-Xd)+(dX-dX_d);
  Sy=C_y*(Y-Yd)+(dY-dY_d);
  Sz=C_z*(Z-Zd)+(dZ-dZ_d);
  PosController();
  Sroll=C_roll*(Roll-Roll_d)+(dRoll-dRoll_d);
  Spitch=C_pitch*(Pitch-Pitch_d)+(dPitch-dPitch_d);
  Syaw=C_yaw*(Yaw-Yaw_d)+(dYaw-dYaw_d);
  Syaw=C_yaw*(Yaw-Yaw_d)+(dYaw-dYaw_d);
  //The altitude controller:
  U1= (-C_z*(dZ-dZ_d)+g+ddZ_d-E_z*sat(Sz,1)-K_z*Sz)*(m/(cos(Roll)*cos(Pitch)));
  //The atittude controller:
  U2=(-C_roll*(dRoll-dRoll_d)-(I_r*dRoll*Omg_r+dPitch*dYaw*(I_yy-I_zz))/I_xx+ddRoll_d-E_roll*sat(Sroll,1)-K_roll*Sroll)*(I_xx/l);
  U3=(-C_pitch*(dPitch-dPitch_d)+(I_r*dPitch*Omg_r-dRoll*dYaw*(I_zz-I_xx))/I_yy+ddPitch_d-E_pitch*sat(Spitch,1)-K_pitch*Spitch)*(I_yy/l);
  U4=(-C_yaw*(dYaw-dYaw_d)-(dRoll*dPitch*(I_xx-I_yy))/I_zz+ddYaw_d-E_yaw*sat(Syaw,1)-K_yaw*Syaw)*I_zz;
}

```

```
285 //position Controller
286 void PosController(){
287   float xx,yy,xxx,yyy;
288   xx=-C_x*(dX-dX_d)+ddX_d-E_x*sat(Sx,l)-K_x*Sx;
289   yy=-C_y*(dY-dY_d)+ddY_d-E_y*sat(Sy,l)-K_y*Sy;
290   xxx=-(C_x+K_x)*(xx)-E_x*dsat(Sx,l)-C_x*K_x*dX;
291   yyy=-(C_y+K_y)*(yy)-E_y*dsat(Sy,l)-C_y*K_y*dY;
292   Roll_d=(l/g)*(sin(Yaw)*(xx)-cos(Yaw)*(yy));
293   Pitch_d=(l/g)*(cos(Yaw)*(xx)+sin(Yaw)*(yy));
294   Roll_d=sat(Roll_d, 0.349066);
295   Pitch_d=sat(Pitch_d, 0.349066);
296   dRoll_d=(l/g)*(sin(Yaw)*(xxx)-cos(Yaw)*(yyy));
297   dPitch_d=(l/g)*(cos(Yaw)*(xxx)+sin(Yaw)*(yyy));
298   ddRoll_d=(l/g)*(sin(Yaw)*(-(C_x+K_x)*(xxx)-C_x*K_x*(xx))-cos(Yaw)*(-(C_y+K_y)*(yyy)-C_y*K_y*(yy)));
299   ddPitch_d=(l/g)*(cos(Yaw)*(-(C_x+K_x)*(xxx)-C_x*K_x*(xx))+sin(Yaw)*(-(C_y+K_y)*(yyy)-C_y*K_y*(yy)));
300 }
301
```

```
302 //Saturation function
303 float sat(float s, float b){
304   if (s<-b){ return -b;}
305   else if (s>b){ return b;}
306   else {return s;}
307 }
308
309 //Derivative of saturation function
310 float dsat(float s, float b){
311   if (s<-b){ return 0;}
312   else if (s>b){ return 0;}
313   else {return 1;}
314 }
315
316 //Control speed of motors
317 void ControlMotors(float U1, float U2, float U3, float U4){
318   //RPM speed of the motors calculation
319   Omg1=sqrt(abs((1/(4*b))*(U1-U2+U3)+(1/(4*d))*U4));
320   Omg2=sqrt(abs((1/(4*b))*(U1+U2-U3)+(1/(4*d))*U4));
321   Omg3=sqrt(abs((1/(4*b))*(U1+U2+U3)-(1/(4*d))*U4));
322   Omg4=sqrt(abs((1/(4*b))*(U1-U2-U3)-(1/(4*d))*U4));
323   Omg_r = Omg1+Omg2-Omg3-Omg4;
324   // scale it to use it with the ESC
325   VO1= map(Omg1, 10, 235, SPEED_MIN, SPEED_MAX);
326   VO2= map(Omg2, 10, 235, SPEED_MIN, SPEED_MAX);
327   VO3= map(Omg3, 10, 235, SPEED_MIN, SPEED_MAX);
328   VO4= map(Omg4, 10, 235, SPEED_MIN, SPEED_MAX);
329   // sets the ESC speed according to the scaled value
330   esc1.speed(VO1);
331   esc2.speed(VO2);
332   esc3.speed(VO3);
333   esc4.speed(VO4);
334   }
335 //Convert Lat/Lon/Alt cooordinates into X/Y/Z coordinates
336 void LatLonAltToXYZ(float Lat, float Lon, float Alt){
337   float r;
338   r=sqrt(pow(6378100*cos(Lat_R),2)+pow(6356800*sin(Lat_R),2));
339   X=r*(Lat-Lat_R);
340   Y=r*cos(Lat)*(Lon-Lon_R);
341   Z=Alt-Alt_R;
342 }
343 //Get IMU GY-86 data measuments Roll, Pitcch, and Yaw
344 void IMUDataRollPitchYaw(){
345    int16_t rax, ray, raz;
346    int16_t rgx, rgy, rgz;
347    int16_t rmx, rmy, rmz;
348    float ax, ay, az;
349    float gx, gy, gz;
350    float mx, my, mz;
351    float deltat;
```

```
352    accelgyro.getMotion6(&rax, &ray, &raz, &rgx, &rgy, &rgz);
353    mag.getHeading(&rmx, &rmy, &rmz);
354    // convert accelometer reading into G's
355    ax=rax/16384.0;
356    ay=ray/16384.0;
357    az=raz/16384.0;
358    // convert gyroscope reading into Radian/second
359    gx=(rgx/131)*Pi/180;
360    gy=(rgy/131)*Pi/180;
361    gz=(rgz/131)*Pi/180;
362    // convert magnetometer reading into Gauss's
363    mx=rmx/1090.0;
364    my=rmy/1090.0;
365    mz=rmz/1090.0;
366    //Madgwik filter
367    deltat = fusion.deltatUpdate();
368    fusion.MadgwickUpdate(gx, gy, gz, ax, ay, az, mx, my, mz, deltat);
369    //Euler angels
370    Roll = fusion.getRollRadians();
371    Pitch = fusion.getPitchRadians();
372    Yaw = fusion.getYawRadians();
373 }
374 //Get derivatives of Roll, Pitch , and Yaw
375 void DivRollPitchYaw(){
376    float PreRoll, PrePitch, PreYaw;
377    float PreTime,Time;
378    PreTime = micros();
379    IMUDataRollPitchYaw();
380    PreRoll = Roll; PrePitch = Pitch; PreYaw = Yaw;
381    Time = micros();
382    IMUDataRollPitchYaw();
383    float dt=(Time-PreTime)*pow(10,-6);
384    dRoll = Div(dRoll, PreRoll, Roll, dt);
385    dPitch = Div(dPitch, PrePitch, Pitch, dt);
386    dYaw = Div(dYaw, PreYaw, Yaw, dt);
387 }
388 //Get derivatives of X, Y, and Z
389 void DivXYZ(){
390    float PreX, PreY, PreZ;
391    float PreTime,Time;
392    PreTime = micros();
393    GPSDataLatLon();
394    Alt = GetAlt();
395    LatLonAltToXYZ(Latv, Lonv, Alt);
396    PreX = X; PreY = Y; PreZ = Z;
397    Time = micros();
398    GPSDataLatLon();
399    Alt = GetAlt();
400    LatLonAltToXYZ(Latv, Lonv, Alt);
401    float dt=(Time-PreTime)*pow(10,-6);
```

```
    dX = Div(dX, PreX, X, dt);
    dY = Div(dY, PreY, Y, dt);
    dZ = Div(dZ, PreZ, Z, dt);
}
//Get Actual Altitude
float GetAlt(){
  float h;
   h=UlrasonicAlt();
   if(h<4.8){
    return h+Alt_R;
   }
   else{
    return BarometerAlt();
    }
}

//Get Altitude from MS5611 in meters
float BarometerAlt(){
    // Read raw values
  uint32_t rawPressure = baro.readRawPressure();

  // Read true temperature & Pressure
  long realPressure = baro.readPressure();

  // Calculate altitude
  return baro.getAltitude(realPressure);
}
//Get Altitude from Ultrasonic
float UlrasonicAlt(){
  long duration;
  float distance;
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin,
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance= duration*0.034/2;
  return distance/100;
}

//Get GPS Data Latitude  and Longitude
void GPSDataLatLon(){
   while (Serial2.available() > 0){
    if (gps.encode(Serial2.read())){
      if (gps.location.isValid()){
       Latv=gps.location.lat();
       Lonv=gps.location.lng();
       break;
  }
  else  {    continue;  }
  }
  }
}
//Derivative function
float Div(float V, float PreD, float D, float T ){
  return V = (D-PreD)/T;
}

// start and stop operation
void GetPermission(char DATA[]){
  String Testt = "";
   for(int k=0; k<200; k++){
       for(int i =0; i<5; i++){
         Testt += DATA[i+k];
       }
    if(Testt.compareTo("START") == 0){
     GO = true;
     break;
     }
    if(Testt.compareTo("STOP!") == 0){
    STOP = true;
    EmergencyStop();
    break;
    }
    Testt="";
   }
   memset(DATA, 0, sizeof(DATA));
}

// Latitude, Longitude, altitude and yaw data
void GetDesiredData(char DATA[]){
  boolean Do=false;
  String Testt = "";
  for(int k=0; k<200; k++){
       for(int i =0; i<4; i++){
         Testt +=DATA[i+k];
       }
       int t;
       if(Testt.compareTo("Lata")==0){
         t=0;
         Testt="";
         while(DATA[4+k+t] != 'b'){
           Testt +=DATA[4+k+t];
           t++;
         }
         Lat[0]=Testt.toFloat();
         Testt="";
         t++;
         while(DATA[4+k+t] != 'c'){
           Testt +=DATA[4+k+t];
           t++;
         }
        Lat[1]=Testt.toFloat();
         Testt="";
         t++;
         while(DATA[4+k+t] != 'd'){
           Testt +=DATA[4+k+t];
           t++;
         }
         Lat[2]=Testt.toFloat();
         Testt="";
         t++;
         while(DATA[4+k+t] != 'e'){
           Testt +=DATA[4+k+t];
           t++;
         }
        Lat[3]=Testt.toFloat();
         Testt="";
       }
       if(Testt.compareTo("Lona")==0){
         t=0;
         Testt="";
         while(DATA[4+k+t] != 'b'){
           Testt +=DATA[4+k+t];
           t++;
         }
         Lon[0]=Testt.toFloat();
         Testt="";
         t++;
         while(DATA[4+k+t] != 'c'){
           Testt +=DATA[4+k+t];
           t++;
         }
        Lon[1]=Testt.toFloat();
         Testt="";
         t++;
         while(DATA[4+k+t] != 'd'){
           Testt +=DATA[4+k+t];
           t++;
         }
         Lon[2]=Testt.toFloat();
         Testt="";
         t++;
```

```
          while(DATA[4+k+t] != 'e'){
            Testt +=DATA[4+k+t];
            t++;
          }
         Lon[3]=Testt.toFloat();
          Testt="";
        }
        if(Testt.compareTo("High")==0){
          t=0;
          Testt="";
          while(DATA[4+k+t] != 'H'){
            Testt +=DATA[4+k+t];
            t++;
          }
          High=Testt.toFloat();
          Testt="";
          break;
        }
        if(Testt.compareTo("Yawa")==0){
          t=0;
          Testt="";
          while(DATA[4+k+t] != 'Y'){
            Testt +=DATA[4+k+t];
            t++;
          }
          dYaw_d=Testt.toFloat();
          Testt="";
          Do=true;
        }
        Testt="";
      }
      Alt=Alt_R+High;
     if(Do == true){
    for(int i=1; i<=4;i++){
      LatLonAltToXYZ(Lat[i-1],Lon[i-1],Alt);
      X_d[i]=X;
      Y_d[i]=Y;
      Z_d[i]=Z;
    }
    X_d[5]=X_d[1];
    Y_d[5]=Y_d[1];
    Z_d[5]=Z_d[1];
    Z_d[6]=Z_d[1];
    Do=false;
   }
}
```