LAB University of Applied Sciences
Technology Lappeenranta
Mechanical Engineering and Production Technology

Muhammad Usman

Quadcopter Modelling and Control With MATLAB/Simulink Implementation

Thesis 2020

Abstract

Muhammad Usman
Quadcopter Modelling and Control with MATLAB/Simulink Implementation, 68 pages
LAB University of Applied Sciences
Technology Lappeenranta
Mechanical Engineering and Production Technology
Bachelor’s Thesis 2019
Instructors: Senior University Lecturer Kiviluoma Panu, Aalto University
Supervisor: Jukka Nisonen, LAB University of Applied Sciences;

The objective of the project was to design a Proportional, Integral and Derivative (PID) based controller in MATLAB/Simulink to achieve attitude control of the quadcopter.

A controller built upon the mathematical model of kinematics and dynamics of the vehicle was Implemented and tested on an Arduino hardware for data collection and control system evaluation. Internet was the primary source of information and study material on the project. Besides internet, books, articles and online training programmes related to the topic were also consulted.

The simulation results of the designed model were quite satisfactory. On the real hardware the controller could lift the quadcopter from the ground while moving in random directions to achieve stability for hovering, as the controller was not designed to maintain the take-off position of the vehicle. However, because of hardware limitations, processing signals scaling and limiting time for PIDs tuning the take-off behaviour of the quadcopter was not as expected from simulation studies of the model.

Further PIDs tuning is required for improved control and smoother flights. In addition to this hardware board with more computational power, real-time wireless communication and better compatibility with MATLAB/Simulink can be introduced to make tuning process easy and safe.

Keywords: Quadcopter, control system, PID, MATLAB/Simulink, mathematical model, kinematics, dynamics, Arduino, simulation, computation, communication, tuning

Abbreviations

PID – Proportional, Integral, Derivative
UAV – Unmanned Aerial Vehicle
ESC – Electronic Speed Controller
PWM – Pulse Width Modulation
PPM – Pulse Position Modulation
RPM – Rotations Per Minute
Hz – Hertz
MHz – Megahertz
LQR – Linear Quadratic Regulator
DC – Direct Current
ID – Identification
IMU – Inertial Measurement Unit
LiPo – Lithium Polymer
NED – North East Down
GPS – Global Positioning System
BLDC – Brushless DC
ESP – Espressif System
TCP – Transmission Control Protocol
IP – Internet Protocol
I^2C – Inter-integrated circuit
SCL – Serial Clock
SDA – Serial Data
VOTL – Vertical Take-off and Landing
MCU – Microcontroller Unit
FLA – Fast Lightweight Autonomy
KV – Constant Velocity
MEMS – Micro Electromechanical Sensor
3-D – Three Dimensions
6-DOF – Six-degrees of Freedom
SRAM - Static Random-Access Memory
AVR - Alf and Vegard's RISC processor
RISC - Reduced Instruction Set Computer

Table of Contents

1 Introduction

1.1 Quadcopter

A quadcopter also known as a quadrotor is a multi-rotor unmanned aerial vehicle (UAV). Quadcopters falls in the category of vertical take-off and landing (VTOL) UAVs. A quadcopter has four rotors in square formation at the equal distance from the centre of mass of the vehicle. Speed of the rotors is manipulated to perform different maneuvers, hovering, take-off and landing.

Before GPS and internet, drones were only available for military use, but with continuous development in the UAV technology and exceptional growth rate in the previous ten years the drones have become very popular among the civil sector. With growth in popularity drones market was valued at 18.14 billion USD and it is expected to reach 52.30 billion USD by 2025 (Markets 2018.)

Advancement and introduction of an impressive technology in drones has developed new fields of applications for it. Today drones are being used in several areas for various purposes. Stated below are some of common areas of drones applications. (Fiaz and Mukarram 2018a; VBROADCAST LIMITED 2019.)

- Aerial photography
- Search and rescue
- Agriculture
- Shipping and delivery
- Engineering applications
- 3-D mapping
- Research and science
- Aerial surveillance
- Minerals exploration
- Military use, etc

1.2 Background Studies

Quadcopter as a testing platform for different control strategies attracts people related to control engineering. A lot of research is happening on quadcopters to achieve desired maneuverability and hovering while using various control strategies such as Linear Fuzzy Logic, Quadratic Regulator (LQR) control, Adaptive control, Predictive control, Robust control etc. (Fiaz and Mukarram 2018b.)

In Munich Technical University Computer Vision Group is doing research on an autonomous quadcopter. The autonomous quadcopter will be able to, localize and navigate in the 3D environment. They are interested in using monocular stereo and RGB-D cameras as the main sensor. (Technical University of Munich 2014.)

A team of researchers is working on Fast Light weight Autonomy program in DARPA (AGENCY 2017). The purpose of the researchers is to enable quadcopter to fly through obstacles and buildings at a speed of 20 m/s. Drone would be able to use onboard camera and sensor as eyes to see around. An advanced algorithm will be developed which will use data from camera and sensors to make decision without human interference. (DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 2017.)

In Middle East Technical University altitude control was developed by using Linear Quadratic Regulator (LQR) (ÇAMLICA 2004).

1.3 Scope of Thesis

The objective of the project is to utilize the existing material to understand dynamic equations and behaviour of quadcopters. Depending on the dynamic equations of the quadcopter a Proportional, Integral and Derivative (PID) based MATLAB/Simulink control system will be designed and implemented to achieve control of the quadcopter. The designed controller will be able to control attitude of the vehicle (*Roll, Pitch* and *Yaw*). This paper will explain the PID controllers tuning process and integration of the designed controller with real hardware in detail. The project is primarily focused on the PID controller, other control strategies are not explained in this project. Altitude control and autonomous navigation are not part of the project, altitude and position of the vehicle in an inertial frame will be controlled by the pilot commands.

Hardware board, Sensors, Electrical and Mechanical components will be selected, and their general working and compatibility to each other will be discussed for implementation of the controller on the actual system. However, components manufacturing process, materials and detailed working are not concerned with the purpose of the project.

2 Vehicle Dynamic

It is very important to have the concept of 6-degrees of freedom (6-DOF) to describe the motion of the quadcopter in terms of equations. *6-DOF* concept gives the position and the orientation of the vehicle in three dimensions (3-D). Vehicles dynamic equations simply represent the change in orientation and position over time. We assume that in our case the earth is flat, the force of gravity is constant, centre of mass is equal to center of gravity and quad-copter structure is rigid (Tytler 2017a.)

2.1 Six-Degrees of Freedom (6-DOF)

The position of the vehicle can be described easily by three coordinates in space, but to achieve the control on the vehicle we must be aware of the orientation of the vehicle in space as well. To describe the dynamics (Position vs time and Orientation vs time) in space we need to describe the position of all points on vehicles body, which can be done with six-Coordinates (Two Frames of References), originating the concept of six-degrees of freedom (6-DOF) (Figure 1). (Tytler 2017a.)

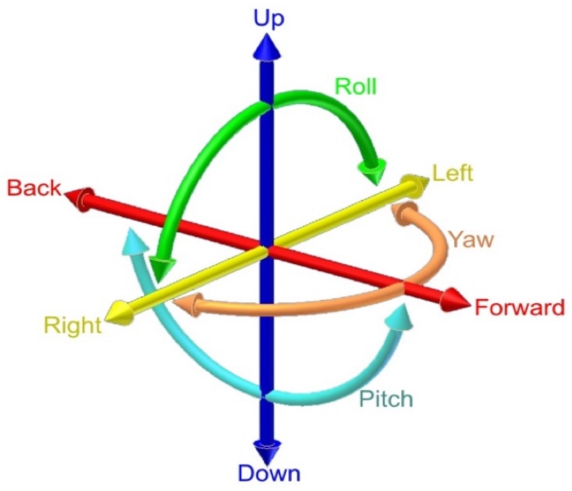


Figure 1. Six-Degrees of Freedom (Wikipedia 2019)

2.2 Orientation and Frames of References

Position of the vehicle can be described by *x, y* and *z* coordinates, giving the respective distance from an origin fixed to the earth known as the inertial frame of reference, and coordinates are usually in the cardinal directions *North, East* and *Down.* To represent the orientation or attitude of the vehicle *φ, θ* and *ψ* angles are used with respect to the body frame reference which is a coordinate system with its origin at body center of gravity (Cg). Figure 2 is the representation of the frames of references for visualization and better understanding.

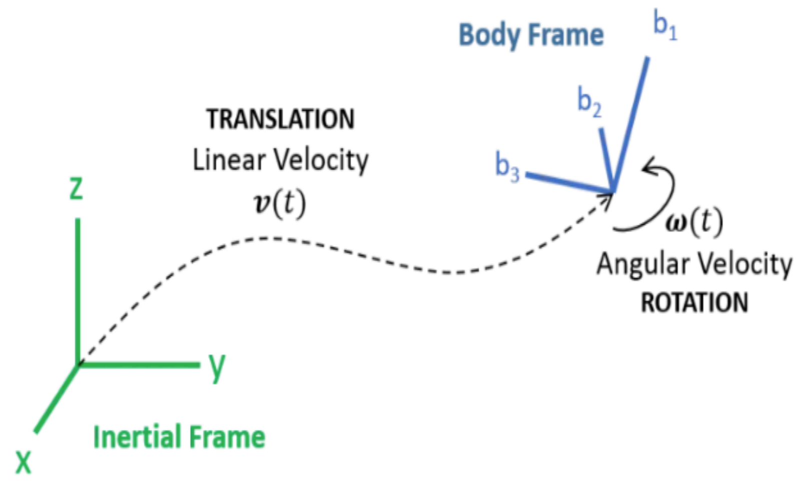
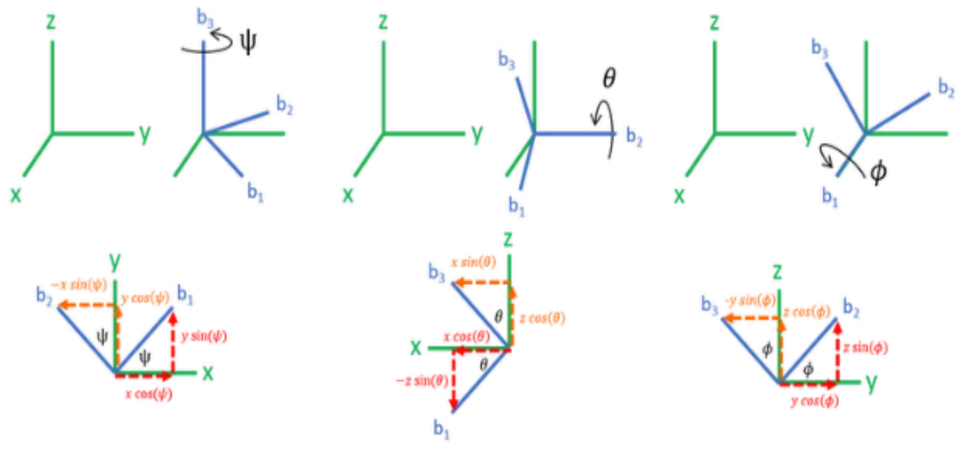


Figure 2. Inertial and Body Frames of References (Tytler 2017b)

2.3 Euler Angels

For this project, we are mainly focusing on the attitude control of the quadcopter. There are some ways to get the orientation of the vehicle, for example, Trigonometric functions, Euler angles and Quatrains. For the sake of simplicity, we will use Euler angles even though there are some singularity issues. By Euler angles we can find the final orientation of the vehicle with-respect to the body frame by using an inertial to body frame transformation matrix. It is convenient to use matrix multiplication for vector transformation. Let us say *(x, y* and *z)* is an inertial frame of reference and *(b1, b2* and *b3)* is a body frame of reference. Rotation around one of the body frame axis results in the displacement of the other two body frame axises with-respect to inertial frame axes. At the same time the rotation axis remains parallel to the corresponding inertial axis. Rotated body frame axises can be represented as inertial trigonometric function equations. Trigonometric equations can be later expressed in matrix forms. Similarly, we can get two other matrices by rotating other two body frame axises. Figure 3 shows the rotation of all three-body frame axises *b1, b2* and *b3* with respect to corresponding inertial frame axises, inertial trigonometric function equations and their matrix representations.



$$b_1 = x\cos(\psi) + y\sin(\psi) \quad b_2 = -x\sin(\psi) + y\cos(\psi) \quad b_3 = z$$

$$b_1 = x\cos(\theta) - z\sin(\theta) \quad b_2 = y \quad b_3 = x\sin(\theta) + z\cos(\theta)$$

$$b_1 = x \quad b_2 = y\cos(\phi) + z\sin(\phi) \quad b_3 = -y\sin(\phi) + z\cos(\phi)$$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Figure 3. Euler Angles (Tytler 2017a)

Orderly combination of these three matrices (Figure 3) gives the inertial to body frame transformation matrix. Order of the operations is really important.

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} cos(\Psi) & sin(\Psi) & 0 \\ -sin(\Psi) & cos(\Psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (1)$$

All three matrices representing the trigonometric equations are multiplied to get a single transformation matrix C_n^b shown in equation 2 which will be used for inertial to body frame transformation.

$$\begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} \boldsymbol{cos(\theta)\,cos(\Psi)} & \boldsymbol{cos(\theta)\,sin(\Psi)} & \boldsymbol{-\,sin(\theta)} \\ \boldsymbol{-cos(\phi)\,sin(\Psi) + cos(\Psi)\,sin(\theta)\,sin(\phi)} & \boldsymbol{cos(\Psi)\,cos(\phi) + sin(\theta)\,sin(\phi)\,sin(\Psi)} & \boldsymbol{cos(\theta)\,sin(\phi)} \\ \boldsymbol{sin(\Psi)\,sin(\phi) + cos(\Psi)\,cos(\phi)\,sin(\theta)} & \boldsymbol{-\,sin(\phi)\,cos(\Psi) + cos(\phi)\,sin(\theta)\,sin(\Psi)} & \boldsymbol{cos(\theta)\,cos(\phi)} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} \quad \boldsymbol{(2)}$$

For transformation from body to inertial frame theoretically the inverse of the trans- formation matrix C_b^n is used. As transformation matrices are orthonormal, we can simply use the transpose of the transformation matrix R^t=R^{-1}. Equation 3 shows transpose of the equation 2 matrix.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} cos(\theta)\,cos(\Psi) & -cos(\phi)\,sin(\Psi) + cos(\Psi)\,sin(\theta)\,sin(\phi) & sin(\Psi)\,sin(\phi) + cos(\Psi)\,cos(\phi)\,sin(\theta) \\ cos(\theta)\,sin(\Psi) & cos(\Psi)\,cos(\phi) + sin(\theta)\,sin(\phi)\,sin(\Psi) & -\,sin(\phi)\,cos(\Psi) + cos(\phi)\,sin(\theta)\,sin(\Psi) \\ -\,sin(\theta) & cos(\theta)\,sin(\phi) & cos(\theta)\,cos(\phi) \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (3)$$

Now using the equations 2 and 3, we can find the orientation of the vehicle with respect to body frame and inertial frame. (Tytler 2017a.)

2.4 Equations of Motion

2.4.1 Variables

Now we can define the position and orientation of the vehicle in space. To get equations of motion of the vehicle we need variables (Table 1) to represent the linear and angular velocity of the quadcopter. Using right-hand rule and old convention referring *X, Y* and *Z* to *North, East* and *Down* respectively, we can derive the equations of dynamics. (Tytler 2017b.)

Table 1. Variables

Linear velocity variables	
V^b	Linear velocity in body frame
u	Longitudinal velocity
v	Lateral velocity
w	Normal velocity
Rotational velocity variables	
$\mathcal{W}^b$	Rotational velocity in body frame
p	Roll rate
q	Pitch rate
r	Yaw rate
Forces	
F_x	Force in X direction
F_Y	Force in Y direction
F_Z	Force in Z direction
Euler angles	
ϕ	Roll angle
θ	Pitch angle
Ψ	Yaw angle
Moments or Torques	

L	Rotational moment along x axis
M	Rotational moment along y axis
N	Rotational moment along Z axis

2.4.2 Inertial Motion in Body Frame

As we have assumed that vehicle body is rigid, there is no movement in the body parts regarding body frame. Our sensors and propellers are attached to the body, so we want to derive equations of motion in the body frame coordinates independent of an inertial frame. Later these equations could be used with any inertial starting point. (Tytler 2017b.)

2.4.3 The Chain Rule

The chain rule represents the inertial motion in body frame and simultaneously gives the mathematical representation. We take the derivative of inertial frame vectors to represent them in body frame. Chain rule of derivation gives the derivative of both the change because of the time derivative of the vector within the coordinate frame, as well as the time derivative of the coordinate frame rotation. Where $\hat{b}_1$, $\hat{b}_2$ and $\hat{b}_3$ in equation 4 are unit vectors associated with body frame to represent the inertial frame velocity in the body frame.

$$V^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b = u\hat{b}_1 + v\hat{b}_2 + w\hat{b}_3 \quad (4)$$

Applying the Chain rule on equation 4 we get equation 5.

$$\left(\frac{dV}{dx}\right)_{inertial} = \left(\frac{du}{dt}\hat{b}_1 + \frac{dv}{dt}\hat{b}_2 + \frac{dw}{dt}\hat{b}_3\right) + \left(u\frac{d\hat{b}_1}{dt} + v\frac{d\hat{b}_2}{dt} + w\frac{d\hat{b}_3}{dt}\right) \quad (5)$$

In equation 5 above, the first set of parentheses represent inertial velocity in the body coordinates. The second set represent velocity change due to coordinate frame rotation. The equation 5 above can be written as equation 6 using the Coriolis Theorem, which is cross product of angular velocity with velocity vector representing frame rotation. (Tytler 2017b.)

$$\dot{V}_{inertial} = \dot{V}^b + \mathcal{W}_n^b \times V^b \quad (6)$$

Further it can be represented as a skew-symmetric matrix as shown in the equation 7.

$$\mathcal{W} \times v = [\mathcal{W}\times]v = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_z & 0 \end{bmatrix} V \quad (7)$$

2.5 Newton's 2nd Law of Motion

So far, we can switch between the inertial and body frame, and can get inertial motion in the body frame of reference. To derive linear and rotational equations of motion we use Newtons Second Law of motion summarized as force (F) equals change in momentum (P) or mass (m) times acceleration (a) or derivative of velocity (v), represented in equation 8.

$$F = \frac{dP}{dt} = m\frac{dV}{dt} = ma \quad (8)$$

The equation 8 giving linear momentum, can be used to get angular momentum (Equation 9) by multiplying with a position vector ($\vec{r}$).

$$H = \vec{r} \times F = \vec{r} \times m\frac{dV}{dt} \quad (9)$$

Further simplified as the moment (M) equals change in angular momentum (H) which can be replaced with the product of moment of inertia (I) for continuously mass distributed objects and angular acceleration (Ω), where Ω is the derivative of angular velocity (ω). The equation 9 becomes as follows (Equation 10). (Tytler 2017a.)

$$M = \frac{dH}{dt} = I\frac{d\omega}{dt} = I\Omega \quad (10)$$

We will derive linear and rotational equations separately.

2.5.1 External Forces and Moments

For translation equations of motion, we need to know the external forces and moments acting on the body of the vehicle

2.5.2 Thrust

Thrust produced by the propellers of the quadcopter acts perpendicular to the vehicle and moment acts at the centre of gravity of the vehicle. Figure 4 shows the distance of propellers from the centre of gravity (CG) of the vehicle. These distances are later used to calculate the moment around the centre of gravity of the vehicle.

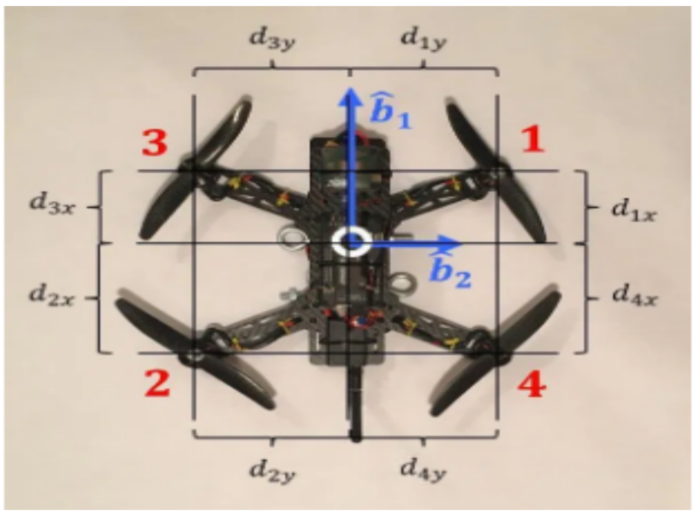


Figure 4. Propellers Distance from centre of gravity (CG) (Tytler 2017b)

Following equation 11 represent the thrust equation of the vehicle.

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -F_1 - F_2 - F_3 - F_4 \end{bmatrix} \quad (11)$$

Moments are simple product of forces and distance from the centre of gravity around all three axes of the body frame.

Equation 12 shows moment along *X* axis of the vehicle.

$$L = F_1 d_{1y} - F_2 d_{2y} - F_3 d_{3y} + F_4 d_{4y} \quad (12)$$

Equation 13 shows moment along *Z* axis of the vehicle.

$$M = -F_1 d_{1x} + F_2 d_{2x} - F_3 d_{3x} + F_4 d_{4x} \quad (13)$$

Yaw is produced in quadcopter by the rotation of the propellers. Two propellers rotate clockwise and other two rotate counter-clockwise to balance the yaw moment (Figure 5). Equation 14 shows how to calculate yaw moment of the quadcopter.

$$N = -T(F_1 d_{1x} d_{1y}) + -T(F_2 d_{2x} d_{2y}) + T(F_3 d_{3x} d_{3y}) + T(F_4 d_{4x} d_{4y}) \quad (14)$$

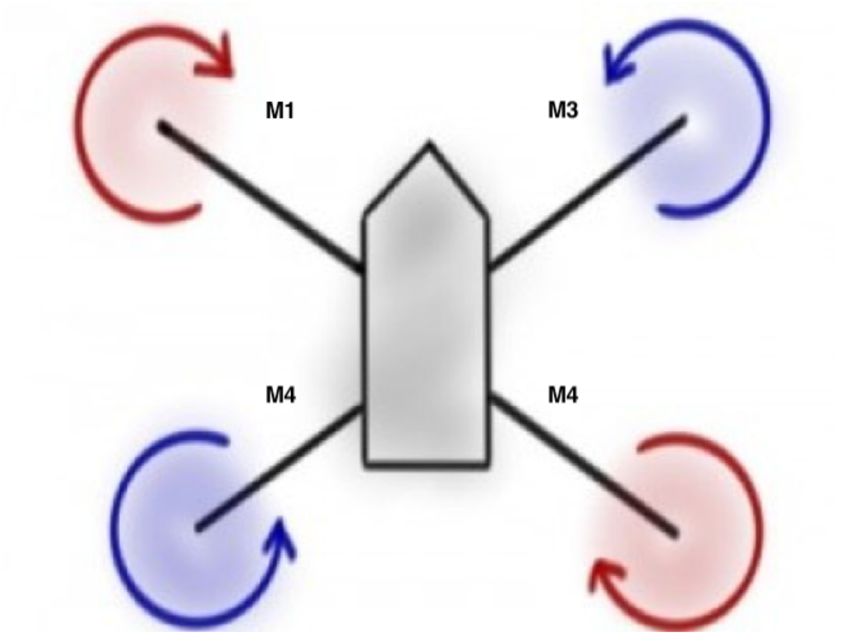


Figure 5. Motors rotation direction for yaw moment (Black Tie Aerial 2014)

2.5.3 Gravity in Body Frame

The force of gravity acting on the vehicle can be represented in the body frame using Euler angles. Transformation matrix C_n^b from equation 2 can be used for this purpose. Where C_n^b is an inertial to body frame transformation matrix, F_g^n and F_g^b represents the force of gravity in inertial and body frames of references respectively. Equation 15 shows the force of gravity in an inertial frame of reference.

$$F_g^n = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \quad (15)$$

Equation 15 is multiplied with inertial to body frame transformation matrix C_n^b to get force of gravity in body frame of reference (Equation 16).

$$F_g^b = C_n^b F_g^n = \begin{bmatrix} cos(\theta)\,cos(\Psi) & cos(\theta)\,sin(\Psi) & -sin(\theta) \\ -cos(\phi)\,sin(\Psi) + cos(\Psi)\,sin(\theta)\,sin(\phi) & cos(\Psi)\,cos(\phi) + sin(\theta)\,sin(\phi)\,sin(\Psi) & cos(\theta)\,sin(\phi) \\ sin(\Psi)\,sin(\phi) + cos(\Psi)\,cos(\phi)\,sin(\theta) & -sin(\phi)\,cos(\Psi) + cos(\phi)\,sin(\theta)\,sin(\Psi) & cos(\theta)\,cos(\phi) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \quad (16)$$

After multiplication, equation 16 simplifies as equation 17.

$$F_g^b = \begin{bmatrix} -mg\,sin(\theta) \\ mg\,sin(\phi)\,co\,s(\theta) \\ mgcos(mg\,cos(\phi)\,co\,s(\theta)) \end{bmatrix} \quad (17)$$

2.5.4 Moments of Inertia

Moment of inertia gives the amount of moment needed to rotate a still object and moment needed to stop a rotating object. We need moment around all three axes of the vehicle, so we will present it in matrix form (Equation 18). Moment of inertia is the square of distance from the centre of mass of the body.

$$I = \begin{bmatrix} I_{xx} & -I_{yx} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \quad (18)$$

For a symmetrical body the moment of inertia on opposite sides of the vehicle cancel each other. We have assumed our vehicle is symmetrical, So our inertia matrix will be simplified as follows (Equation 19). (Tytler 2017b.)

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (19)$$

2.5.5 Linear Acceleration and Motion

Using Coriolis theorem (Equation 6), we can convert inertial acceleration into rotating body frame. Equation 20 shows the inertial acceleration conversion into body frame acceleration.

$$\dot{V}_b = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix}^b + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b = \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \quad (20)$$

By putting forces and acceleration in Newton's 2nd law *F=ma* we can get linear motion (Equations 21).

$$\begin{bmatrix} -mg\,sin(\theta) \\ mg\,sin(\phi)\,co\,s(\theta) \\ -F_1 - F_2 - F_3 - F_4 mgcos(mg\,cos(\phi)\,co\,s(\theta)) \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \quad (21)$$

2.5.6 Rotational Acceleration

Using Coriolis Theorem (Equation 6) again as above in equation 20, we can get rotational acceleration (Equation 22), the difference is that, here we use the angular velocity instead of linear velocity.

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{q} \end{bmatrix}^{b} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (22)$$

After addition and multiplication, equation 22 simplifies as equation 23.

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \dot{p} I_{xx} \\ \dot{q} I_{yy} \\ \dot{q} I_{zz} \end{bmatrix} + \begin{bmatrix} -I_{yy}qr + I_{zz}qr \\ I_{xx}pr - I_{zz}pr \\ -I_{xx}pq + I_{yy}pq \end{bmatrix} \quad (23)$$

2.5.7 Angular Velocity and Euler Angles

We know, how the angular velocities *p, q* and *r* are changing over time and a gyro will give us these values, but these are not the same as Euler angles ϕ, θ and Ψ. Angular velocity is the rate of change of angles with body axis, while Euler angles are rotation in their own frame of reference. Using the coordinate transformation, angular velocity can be represented as Euler angle derivative (Equation 24). (Tytler 2017b.)

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix} \begin{bmatrix} c(\theta) & 0 & -s(\theta) \\ 0 & 1 & 0 \\ s(\theta) & 0 & c(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\Psi} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \quad (24)$$

Equation 24 simplifies as Equation 25.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} -\dot{\Psi}\, s(\theta) \\ \dot{\Psi}\, c(\theta)\, s(\phi) \\ \dot{\Psi}\, c(\phi)\, c(\theta) \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\theta}\, c(\phi) \\ \dot{\theta}\, s(\phi) \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \quad (25)$$

Finally, the equations look as follows

$$p = \dot{\phi} - \dot{\Psi}\, sin(\theta) \quad (26)$$

$$q = \dot{\theta} cos\phi + \dot{\Psi}\, cos(\theta)\, sin(\phi) \quad (27)$$

$$r = \dot{\Psi} cos\phi\, cos(\theta) + \dot{\theta} sin(\phi) \quad (28)$$

2.5.8 Inertial Coordinate

To calculate position of the vehicle in an inertial frame, we simply use Euler angle transformation matrix C_b^n (Equation 3) to convert body frame velocities to an inertial coordinate position (Equation 29). Where $\dot{x}^E$, $\dot{y}^E$ and $\dot{z}^E$ represents a first derivative of *X, Y* and *Z* positions of the vehicle in an inertial frame. (Tytler 2017b.)

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ -\dot{z}^E \end{bmatrix} = \begin{bmatrix} cos(\theta)\, cos(\Psi) & -cos(\phi)\, sin(\Psi) + cos(\Psi)\, sin(\theta)\, sin(\phi) & sin(\Psi)\, sin(\phi) + cos(\Psi)\, cos(\phi)\, sin(\theta) \\ cos(\theta)\, sin(\Psi) & cos(\Psi)\, cos(\phi) + sin(\theta)\, sin(\phi)\, sin(\Psi) & -sin(\phi)\, cos(\Psi) + cos(\phi)\, sin(\theta)\, sin(\Psi) \\ -sin(\theta) & cos(\theta)\, sin(\phi) & cos(\theta)\, cos(\phi) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}^{b} \quad (29)$$

2.5.9 Non-Linear Equations

By simple mathematical operations on the matrices above, we can rewrite the equations in non-linear form and can modify them as we want. Equations are represented below. (Tytler 2017b.)

$$\dot{u} = -gsin(\theta) + rv - qw \quad (30)$$

$$\dot{v} = -gsin\phi cos(\theta) - ru + pw \quad (31)$$

$$\dot{w} = \frac{1}{m}(-Fz) + gcos\phi cos(\theta) - qu - pv \quad (32)$$

$$\dot{p} = \frac{1}{I_{xx}}\left(L + (I_{yy} - I_{zz})qr\right) \quad (33)$$

$$\dot{q} = \frac{1}{I_{yy}}(M + (I_{zz} - I_{xx})pr) \quad (34)$$

$$\dot{r} = \frac{1}{I_{zz}}\left(N + (I_{xx} - I_{yy})pq\right) \quad (35)$$

$$\dot{\phi} = (p + (qsin\phi - rcos\phi)tan\theta) \quad (36)$$

$$\dot{\theta} = qcos\phi - rsin\phi \quad (37)$$

$$\dot{\Psi} = (qsin\phi - rcos\phi)sec\theta \quad (38)$$

$$\dot{x}^E = (qsin\phi - rcos\phi)sec\theta \quad (39)$$

Non-Linear equations represent the dynamic behaviour of the quadcopter, but we still need to represent the equations including motors rpm, propellers thrust, the mass of the quadcopter, torque functions and inertial variables values. These calculations and variable values can be found in Appendix 1(A1, A2, A3, A4, A5 and A6). With these equations we can start the Simulink modeling.

3 Components

The components (Table 2) are selected considering the performance and compatibility with other selected components. The most suitable components are selected online considering the application and budget of the project to build the actual model.

Table 2. Components

Frame	S500 Quadcopter Frame (1)
Propellers/Rotors	10x4.5 Propellers (4)
Motors	TURNIGY 950Kv BLDC motors (4)
ESCs	Afro Racing Spec 20A ESCs (4)
Battery	3S LiPo Battery (1)
Transmitter	TURNIGY TGY i6S Transmitter (1)
Receiver	TURNIGY 6S Channel Receiver (1)
Inertial Measurement Unit (IMU)	Adafruit BNO055 IMU (1)
Arduino Board	Arduino Mega 2560 (1)

3.1 Frame

Frame of the quadcopter is chosen depending on the purpose of the quadcopter. Usually, small drones are used for acrobats and racing, because they are twitchy and considerably unstable. Contrary to this, bigger drone are used for aerial photography as they are more stable with smoother flight. An S500 with 480mm motor centres (Figure 6) is chosen to achieve smoother flights and for clearer observation of control systems performance. After choosing the frame recommended motors and propellers can be chosen from the description sheet.

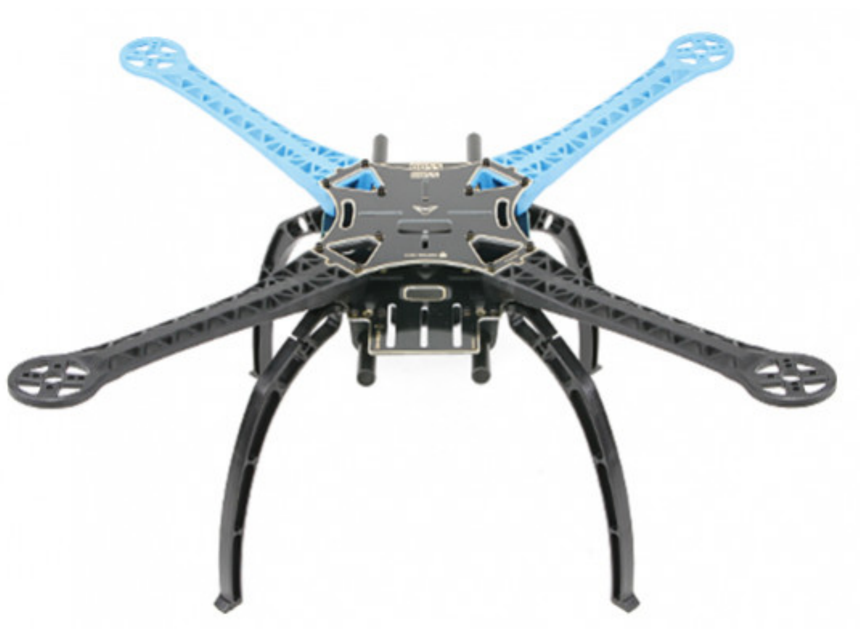


Figure 6. S500 Quadcopter Frame (HobbyKing 2019c)

3.2 Propellers/Rotors

Propellers are selected based on the amount of thrust they can produce depending on motor RPM to obtain the desired performance and altitude of the vehicle. Each propeller should at least provide thrust force equals to the quarter of the weight of the vehicle while rotating within the RPM range of the motor to perform hovering. In a quadcopter two of the propellers rotate clockwise and the remaining two rotate counter-clockwise to balance the torque produced by the rotation of propellers. Propellers can be selected by diameter and pitch representing length and distance covered in one rotation respectively, usually given in inches.(Pro Maker 2016.)

Four 10x4.5 propellers (Figure 7) are selected for this project as they can provide a sufficient amount of thrust to perform hovering and other maneuvers easily.

Figure 7. Propellers 10x4.5 (HobbyKing 2019b)

3.3 Motors

Brushless DC motors (BLDC) motors are used for the quadcopter, as they provide more torque than the brushed motors and they do not need maintenance that often, as brushed motors. There are two types of BLDCs, in-runner and out-runner with rotating inner and outer part of the motor, respectively. In-runners rotate faster as compared to out-runners while the out-runners provide more torque making them popular for quadcopters. Motors are available with different *Kv* ratings which gives the maximum RPM of the motor when multiplied by the maximum voltage provided by the battery. Usually for large propellers motors with low *Kv* ratings and for small propellers with high *Kv* ratings are selected.(Pro Maker 2016.)

Figure 8 shows 950Kv motors selected for 10x4.5 propellers (Figure 7) to obtain the desired thrust and table 3 shows the motors specifications.

Table 3. Table of Motors Specifications

Battery	2~4 Cell /7.4~14.8V
RPM	950kv
Max current	23.2A
No load current	1A
Max power	243W
Internal resistance	0.070 ohm
Weight	70g (including connectors)
Diameter of shaft	4mm
Dimensions	28x36m
Prop size	7.4V/12x6 14.8V/9x6
Max thrust	850g

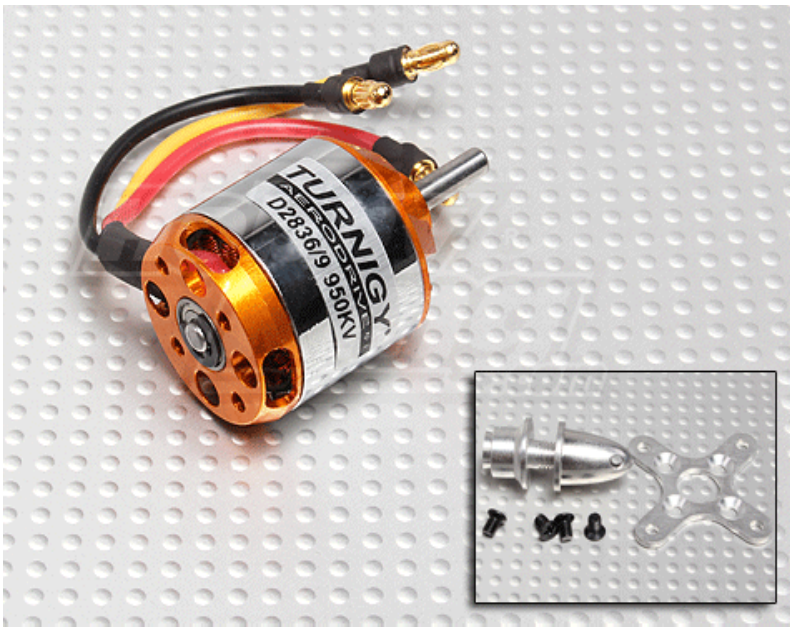


Figure 8. TURNIGY 950Kv BLDC Motor (HobbyKing 2019d)

3.4 Electronic Speed Controllers (ESCs)

ESCs control the speed of BLDC motors according to the signals from the controller for maneuvering and hovering. ESCs are selected based on the maximum current drawn by the motors. It can be seen in the motors specifications (Table 2), that each motor can draw a maximum of 23.2 amperes of current while producing a thrust of 850g. As our vehicle is not aimed for racing and it is very light, there will be hardly an event in which we need the maximum thrust resulting motors to draw 23.2 amps. Considering these factors 20A ESCs (Figure 9) are selected to control the motors. Table 4 presents the specifications of the selected ESCs.

Table 4. ESC's Specifications

Current Draw	20A Continuous
Voltage Range	2-4s Lipoly
Motor wire/plugs	18AWG/Female 2mm
Input Freq	1KHz
Firmware	afro_nfet.hex
Discharge wire/plugs	18AWG/Male 2mm
Weight	10.7g (Included wire, plug, heat shrink)
Size	25 x 20 x 6.5mm

Figure 9. Afro Racing Spec 20A ESC (HobbyKing 2019a)

3.5 Battery

Batteries have a significant effect on the flight of a quadcopter, such as flight time, speed and thrust. Selecting more powerful battery will certainly improve the above-mentioned flight characteristics. However, it is always important to use recommended batteries (Table 2) for the selected motors, as more powerful batteries can cause overheating of motor, finally burning the motors. Despite of the fact that LiPo batteries are fragile and sensitive to overcharging and over-discharging, three to four cell LiPo batteries are quite popular among the hobbyists because of high discharge rate and reasonable weight for long and smooth flight. A battery is chosen based on the maximum current the quadcopter can draw. A three cells LiPo battery (Figure 10) is selected for this project to meet the power requirements of the quadcopter. This battery provides 11.1 V with a continuous discharge rate of 66 amps which can provide a subsequent amount of power to the quadcopter. (Pro Maker 2016.)

Figure 10. Three Cells LiPo Battery (HobbyLinna 2019)

3.6 Transmitter

A transmitter is a way of communication between a pilot and a drone. The transmitter converts the human commands to signals for the vehicle, for different flight operations. While selecting a transmitter the most important thing to be considered is the number of channels. For this project the minimum amount of transmission channels needed is four to perform basic *roll, pitch, yaw* and *take-off* operations.

Figure 11. TURNIGY TGY i6S Transmitter (HobbyKing 2019e)

TURNIGY TGY i6S Transmitter (Figure 11) is selected for this project. This is a 6 channel (PWM) or a 10 channel (PPM) radio transmitter with 2.4GHz transmitting frequency.

3.7 Receiver

A receiver receives the radio signal from a transmitter and converts it into the electrical signal for the microcontroller. The number and type of channels are the main things to be considered in the selection of the receiver. A TURNIGY 6 channel receiver (Figure 12) provided with the transmitter is used in this project. (Pro Maker 2016.)

Figure 12. TURNIGY 6S Channel Receiver (HobbyKing 2019e)

3.8 Flight Controller

A flight controller is the most important component of the quadcopter, it is considered as brain of the vehicle. A flight controller receives data from sensors and performs calculations, designed in control strategy to control the behaviour of the vehicle while performing different maneuvers. An Arduino based flight controller is used in this project consisting of Arduino mega 2560 and an Adafruit BNO055 Inertial Measurement Unit (IMU). Controller is built following the schematics in Figure 13. IMU and MCU transfer data by I^2C communication. (Circuit Basics 2016.)

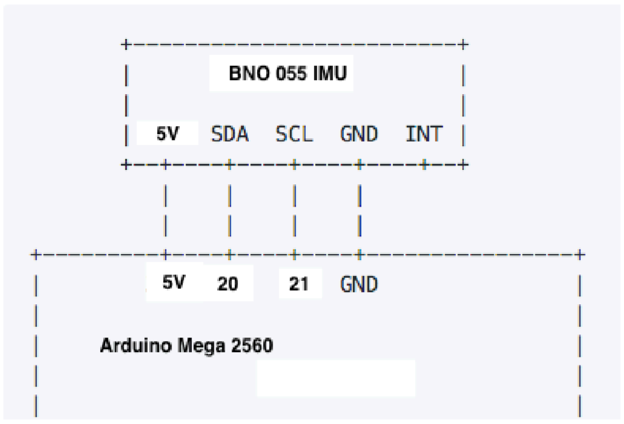


Figure 13. Flight Controller Schematics

3.8.1 Inertial Measurement Unit (IMU)

IMU is an electrical device, which estimates the orientation of the body by measuring the forces of acceleration and angular velocity acting on the small parts of the body (Technaid 2019). An Adafruit BNO055 IMU with 9-DOF is used in this project (Figure 14). BNO055 is a 9-axis absolute orientation sensor with sensor fusion and raw sensor data (Adafruit 2019).

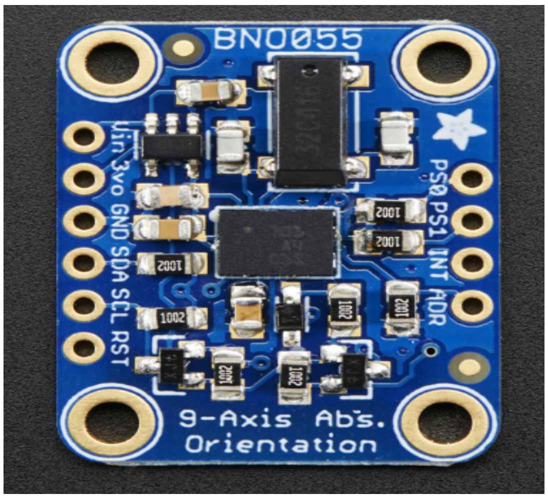


Figure 14. Add a Fruit BNO 055 IMU (Adafruit 2019)

Most of the IMUs (MEMS) consist of a gyroscope, an accelerometer and a magnetometer to estimate orientation, acceleration and angular rates. An accelerometer measures the acceleration by measuring the change in capacitance. A microstructure consisting of some mass is suspended between the electrically charged plates when a force is exerted in a certain direction the mass moves in that direction changing the capacitance between the plates (Figure 15).

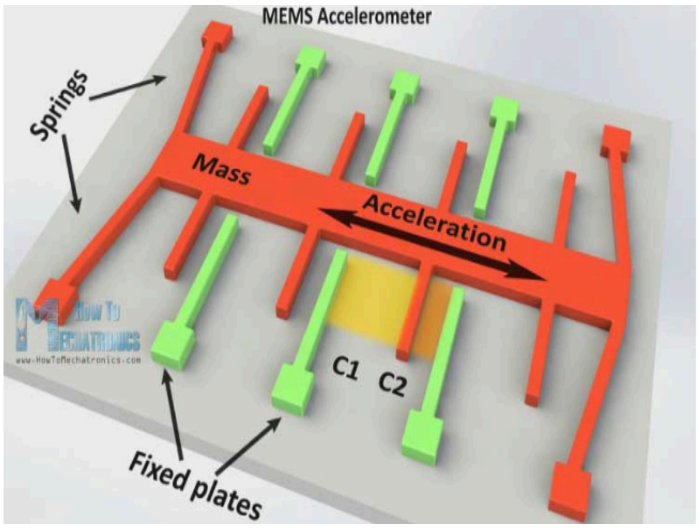


Figure 15. IMU Accelerometer (How to mechatronics 2016)

Similar to accelerometer, a gyroscope also measures the change in capacitance to give angular rate, using Coriolis Effect. External angular rate on a moving body results in a change in capacitance to measure angular rate (Figure 16). A magnetometer measures the change in the magnetic field using Hall Effect. (How to mechatronics 2016.)

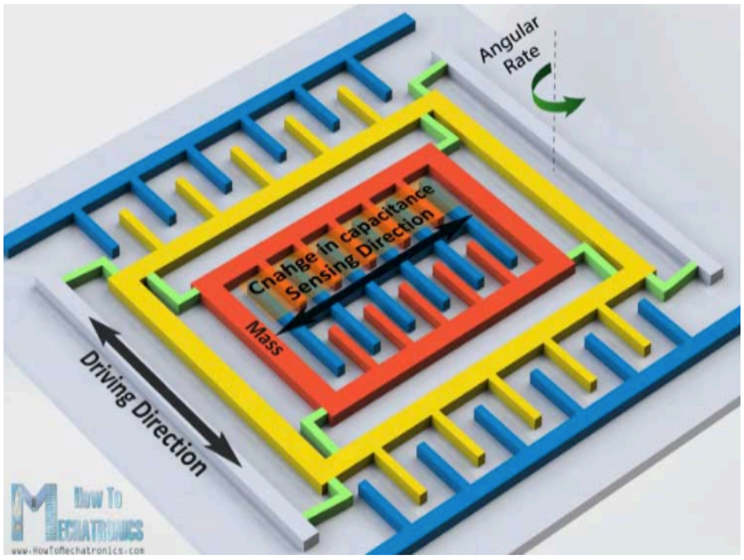

Angular Rate
Cnahge in capacitance Sensing Direction
Mass
Driving Direction
How To Mechatronics
www.HowToMechatronics.com

Figure 16. IMU Gyroscope (How to mechatronics 2016)

3.8.2 Arduino Electronics Platform and Microcontroller Unit (MCU)

Electronics platform used in this project is an Arduino board based on ATmega 2560 microcontroller (Figure 17). ATmega 2560 is a low-power 8-bit microcontroller based on the AVR enhanced RISC architecture. Arduino was chosen, because it's easy to use for beginners, material support on internet and compatibility with different operating systems.(ARDUINO 2019.)

A microcontroller is an integrated circuitry to perform different operations. Microcontrollers usually consists of processors, memory, ram etc. They can also be programmed (TECTARGET 2019).

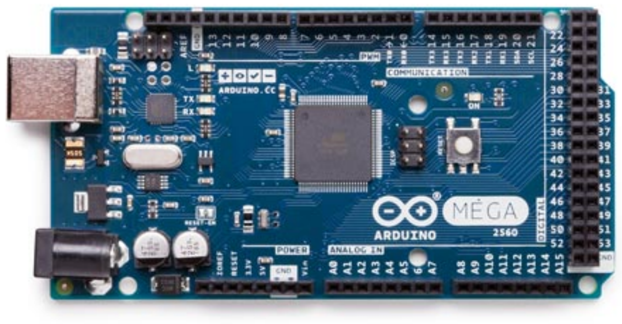


Figure 17. Arduino Mega 2560 (ARDUINO 2019)

3.8.3 I^{2}C Communication

I^{2}C communication can connect single master to multiple slaves and multiple master to single or multiple slaves (Figure 18). To establish I^{2}C communication between MCU and IMU only two wires are needed, SCL and SDA. All data is transferred through one wire SDA in the form of bits, while SCL synchronize the data with the clock signal which is always controlled by master.(Circuit Basics 2016.)

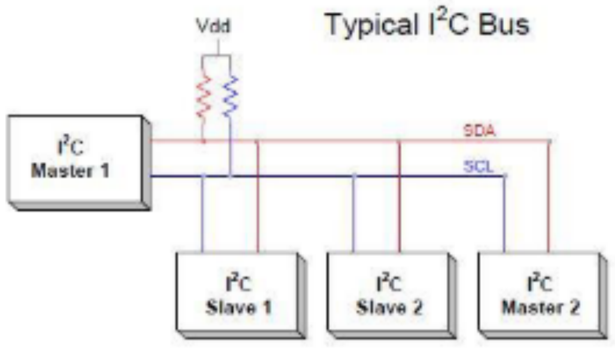


Figure 18. I^{2}C Schematics (Circuitdigest 2019)

3.8.4 Quadcopter Schematics

The quadcopter electrical connections are made following the schematics in Figure 19. The connections are made by soldering and jumper wires are used to make connections to the Arduino.

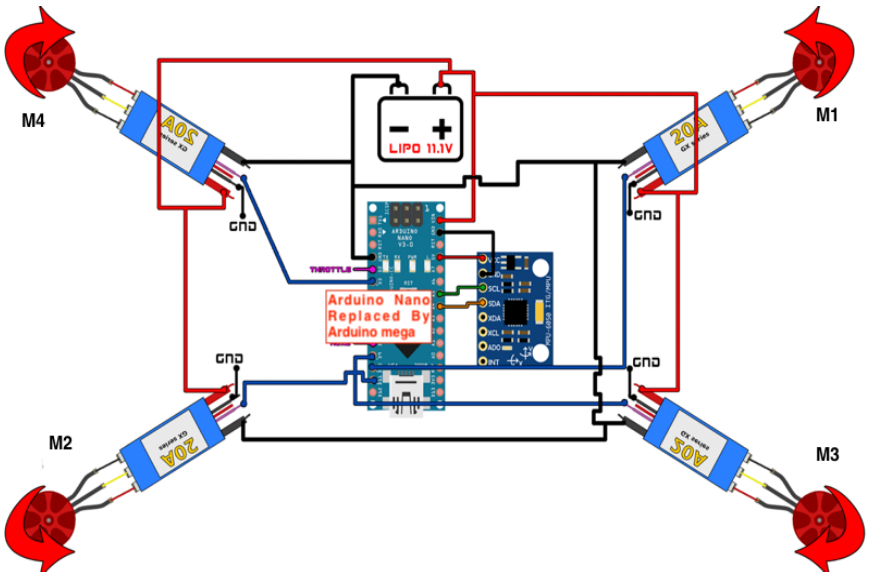


Figure 19. Quadcopter Electrical Connections Schematics (Electronobs 2019)

After connecting all of the mechanical and electrical components and making electrical connections the Quadcopter looks as follows (Figure 20).

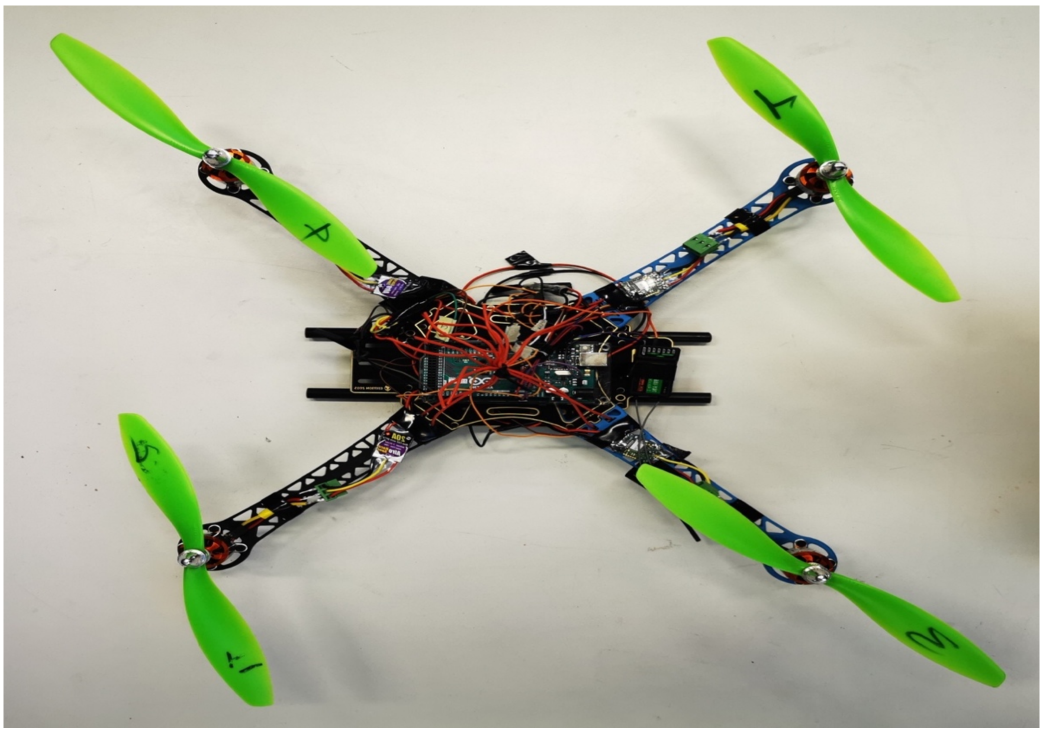


Figure 20. Quadcopter

4 Simulink Modelling

4.1 Plant Model

Now we have started building the simple plant model of the vehicle in Simulink depending on the non-linear equations. A subsystem for the plant modelling is created with *x, y, h* or *z*, and *Ψ* as outputs and *M1, M2, M3* and *M4* as inputs (Figure 21). Plant model subsystem will have several subsystems to represent different parameters depending on their equations. The purpose is to create a model consisting of all the dynamic equations of the vehicle and then build a control system for the model.

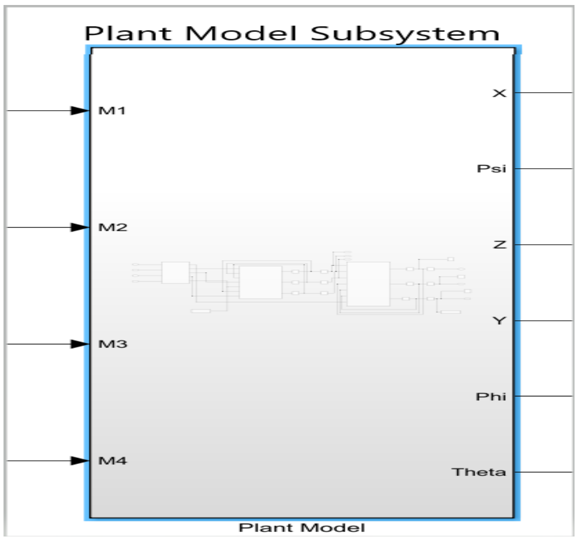


Figure 21. Plant Model Subsystem

On opening the plant model subsystem, it looks as follows (Figure 22).

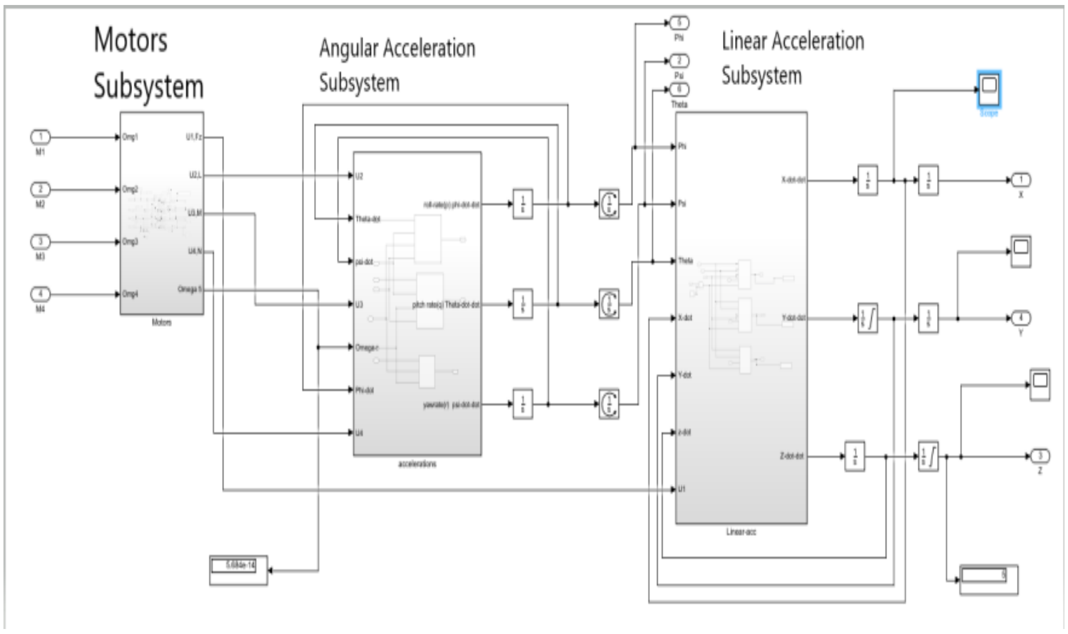


Figure 22. Plant Model Subsystem Opened

4.1.1 Orientation and Motors

Orientation and Motors is a subsystem (Figure 23) under Plant Model subsystem. It depends on the moment equation 12, 13, and 14 for the specific orientation of the vehicle, the quadcopter has two orientations + and *X*. We have derived equations for *X* orientation of our vehicle.

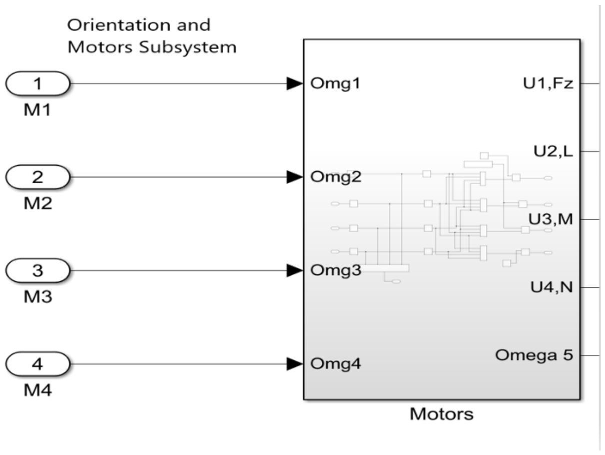


Figure 23. Orientation and Motors Subsystem

The Orientation and Motors subsystem looks as follows on opening (Figure 24). This subsystem takes the motor commands and converts them to moments to control the vehicles orientation and altitude. Omega is the total angular velocity of the quadcopter; the propellers rotate in the opposite direction to balance the total angular velocity.

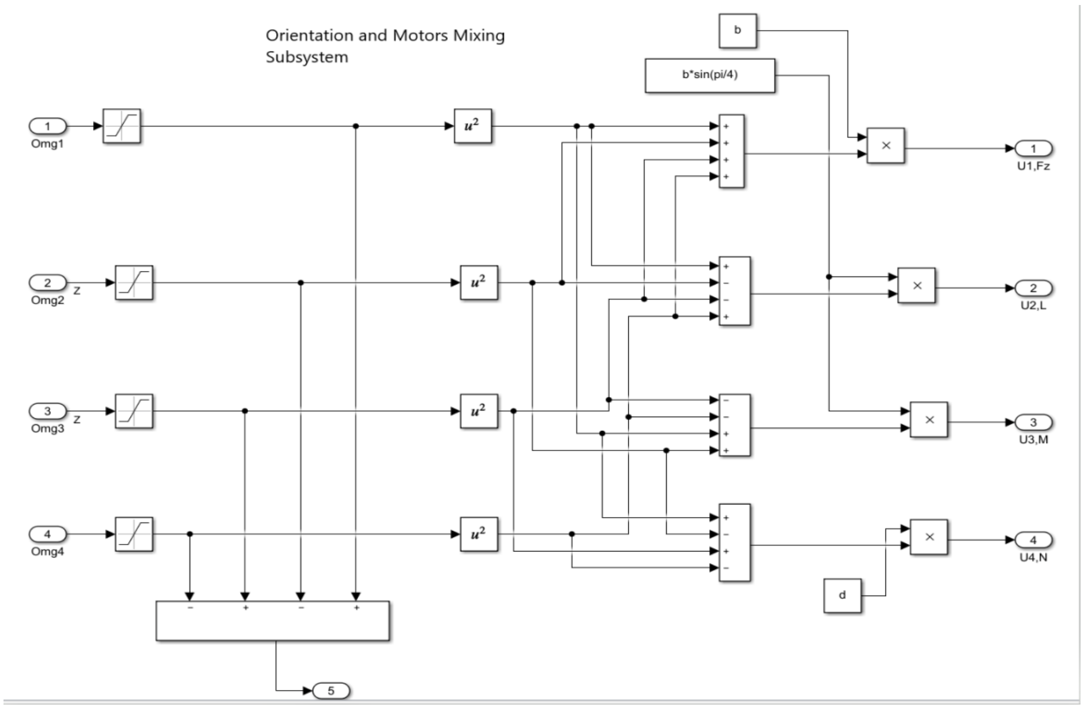


Figure 24. Orientation and Motors Subsystem Inside

4.1.2 Rotational Dynamics

This subsystem (Figure 25) is based on equation 33, 34 and 35 in which motors thrust is converted into angular rates. It takes *U2, U3, U4* and *omega* from motors subsystem and outputs *p, q* and *r*. While *U1* is fed straight to the next subsystem, as *U1* is the total thrust of all motors in the *Z* direction.

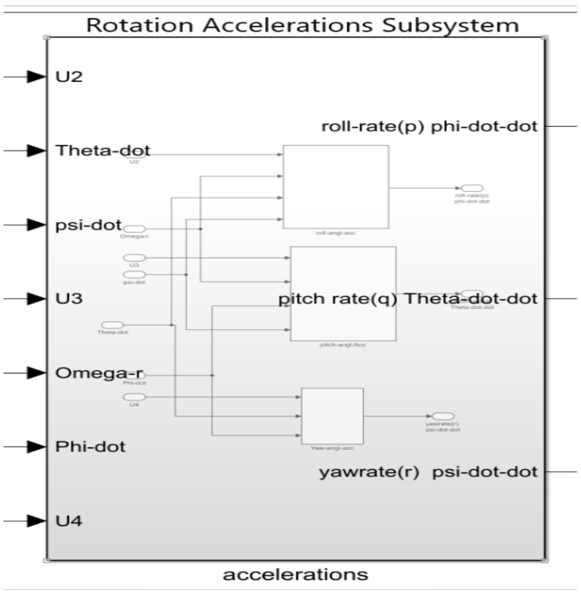


Figure 25. Angular Acceleration Subsystem

Angular acceleration subsystem has three more subsystems to control *roll, pitch* and *yaw* acceleration separately, containing the equations. Subsystems can be opened to see the equations and their arrangement (Figure 26).

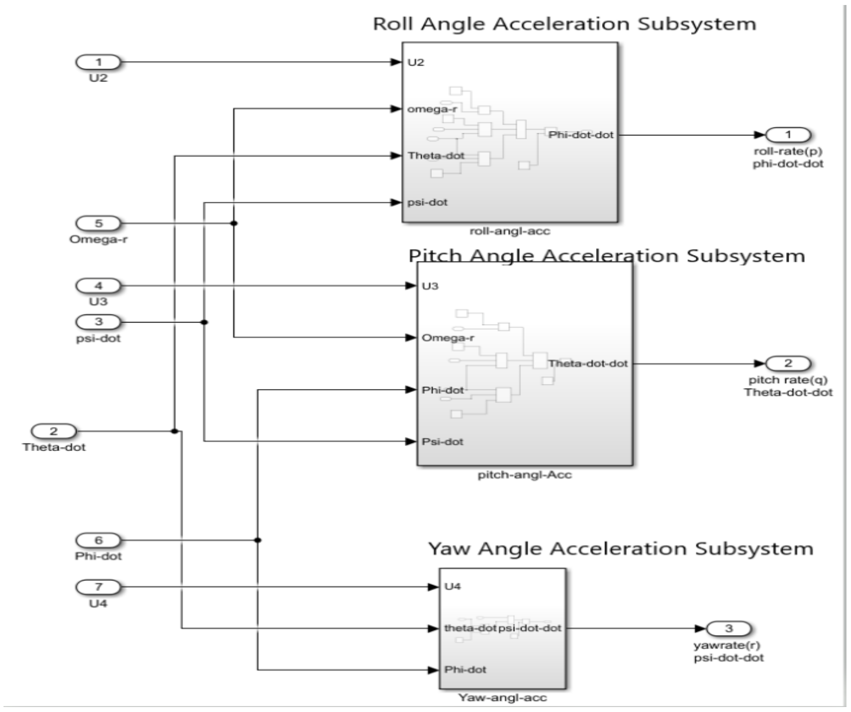


Figure 26. Angular Acceleration Subsystem Inside

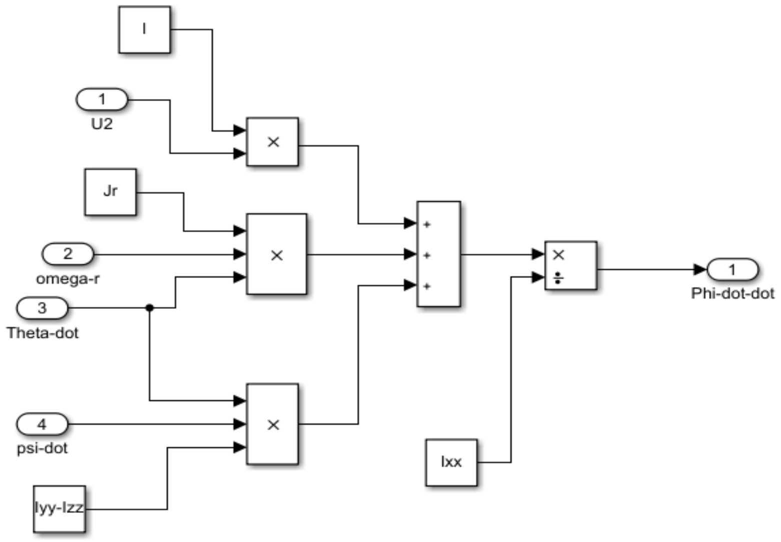


Figure 27. Roll Angle Acceleration Subsystem

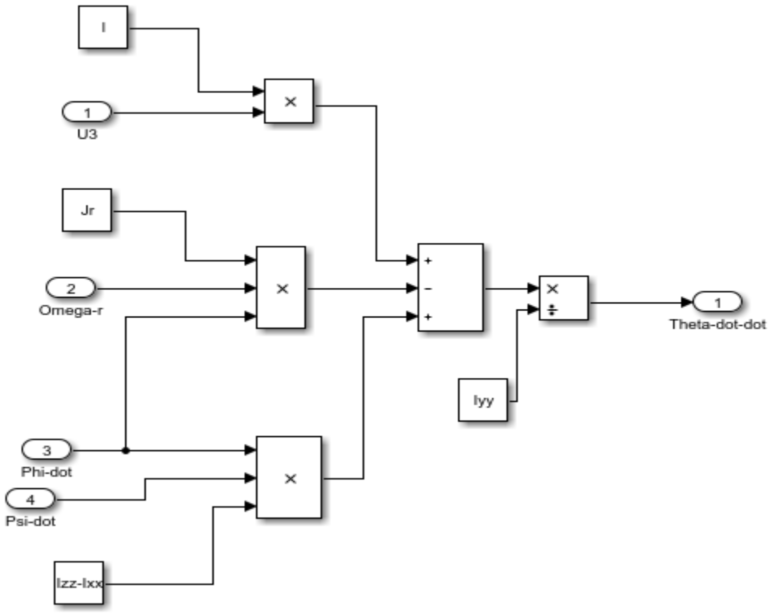


Figure 28. Pitch Angle Acceleration Subsystem

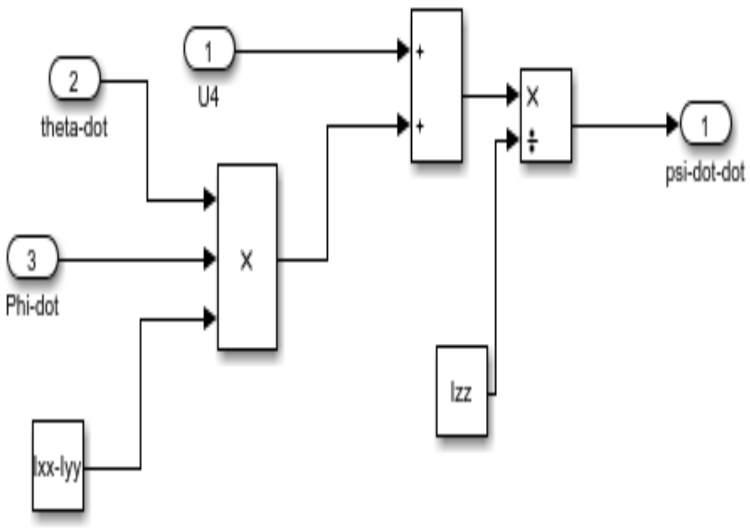


Figure 29. Yaw angle Subsystem

4.2 Linear Dynamics

The angular rates from the previous subsystem are converted into angles using integrators. Now the linear acceleration subsystem (Figure 30) converts angles into the linear acceleration in the inertial frame depending on navigation coordinate equations 36, 37 and 39. To navigate the vehicle in the inertial coordinate frame.

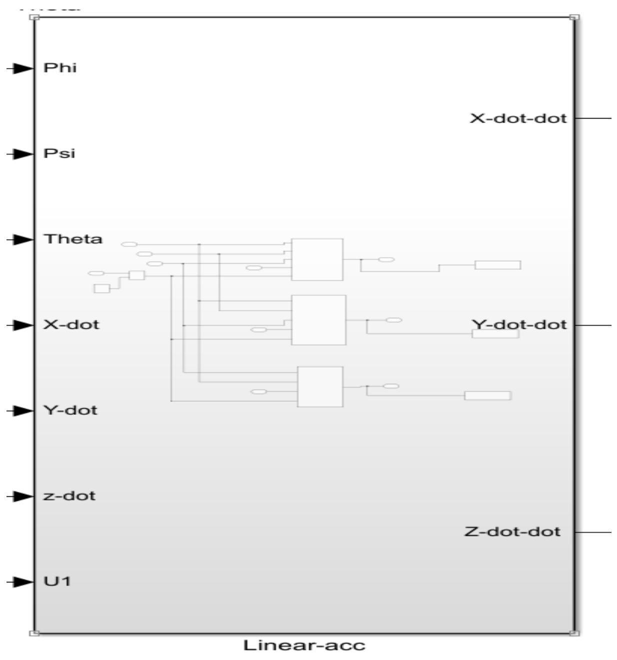


Figure 30. Linear Acceleration Subsystem

Linear acceleration subsystem also has three subsystems for *x, y* and *z* acceleration separately, containing equations for each (Figure 31, 32, 33 and 34).

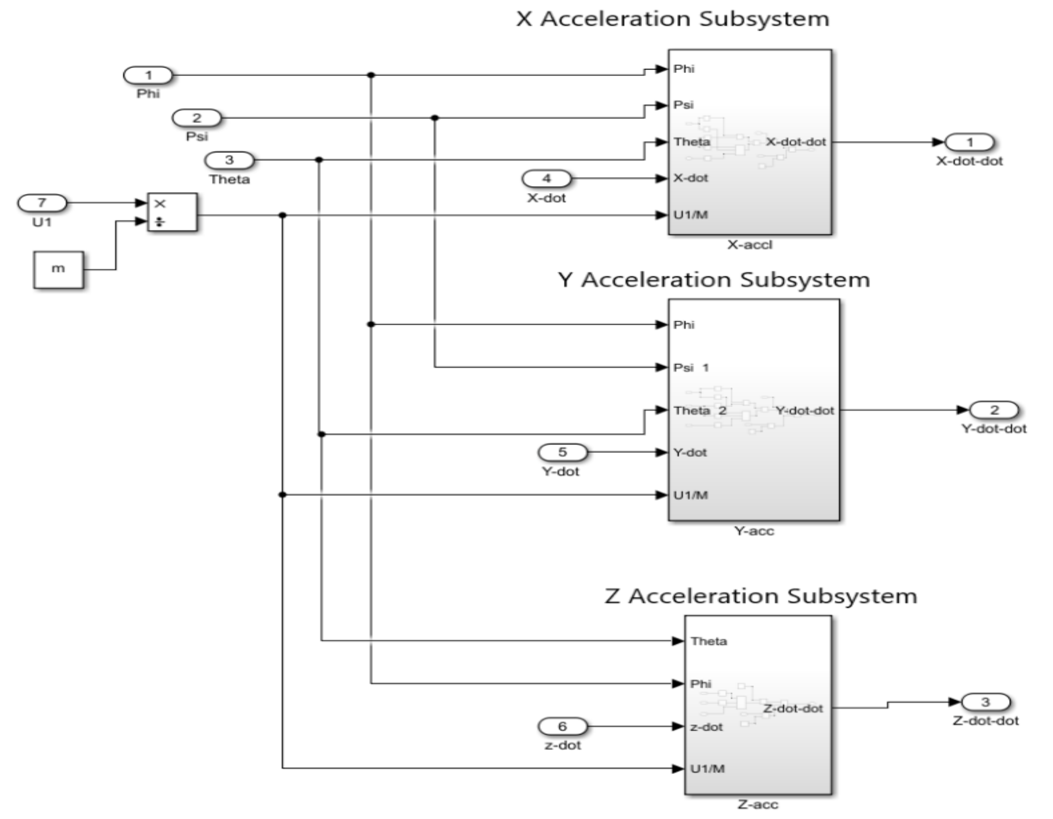


Figure 31. Linear Acceleration Subsystem Inside View

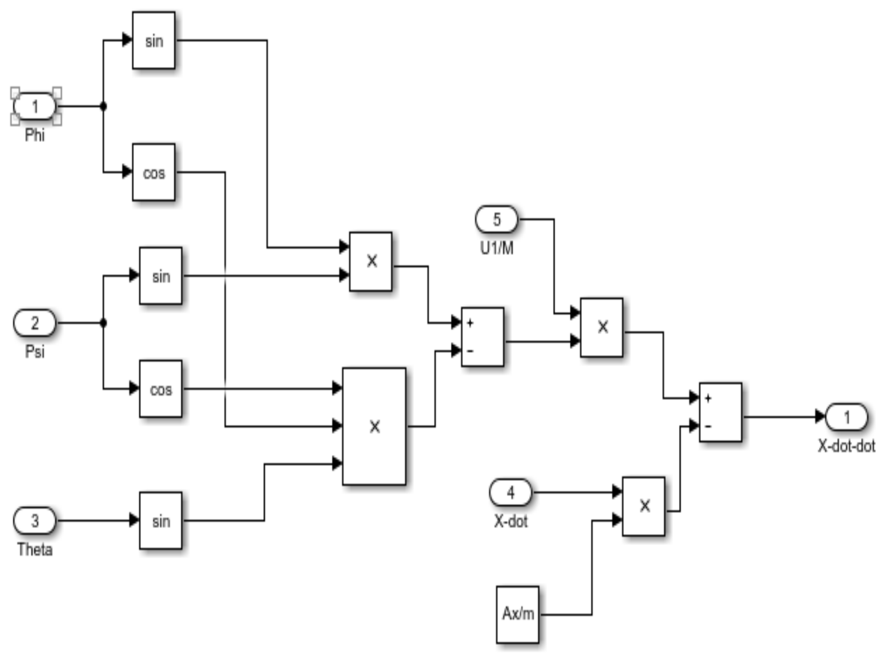


Figure 32. X-Acceleration Subsystem

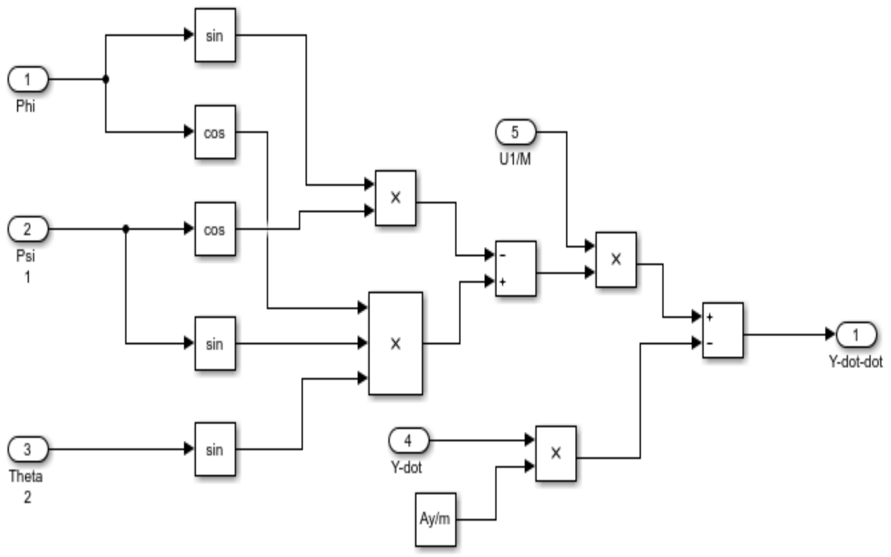


Figure 33. Y-Acceleration Subsystem

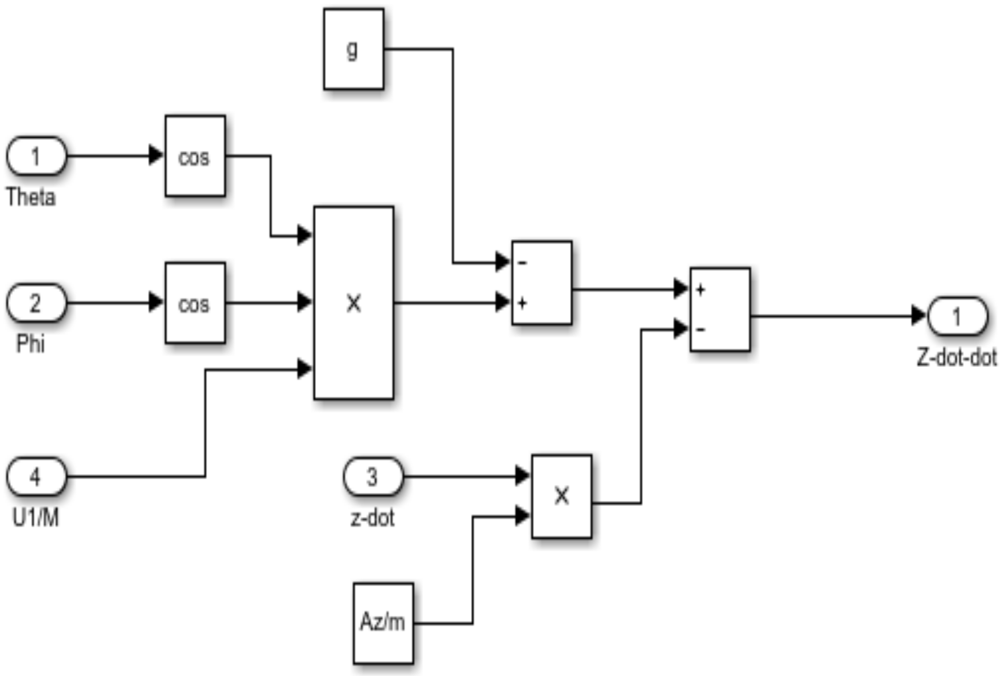


Figure 34. Z-Acceleration Subsystem

4.3 Constraints

Now, we have complete model representing our vehicle, but simulation does not consider real world effects, so we introduce some limitations in our model to have real world effects in our simulation. First of all, the *h* is limited by introducing zero as lower limit in the integrator which converts *z* velocity into *z* position as the vehicle cannot go below the earth's surface. Motors speeds are limited by saturators which eliminates the negative signals to avoid rotation in the opposite of the selected direction. In some quadcopters rotors can rotate in both directions to perform flips and to fly upside down, which is not the focus in our case. *Roll, pitch* and *yaw* angles are also limited between pi and -pi. (Ferry 2017.)

4.4 Model Verification

The Simulink model represents the quadcopter obeying real world limits. We can verify the model by simply applying motor speed inputs. *Simulink scope block* is used for the visualization instead of plotting the data in MATLAB, as it is more convenient and time saving. *Altitude, phi, theta* and *psi* are plotted in the same scope for comparison and to observe the change in behavior. First thrust is provided by rotating all the motors at the same speed. Vehicle response is shown in Figure 35. It can be seen that the *altitude* is changed while *roll, pitch* and *yaw* angles remain same.

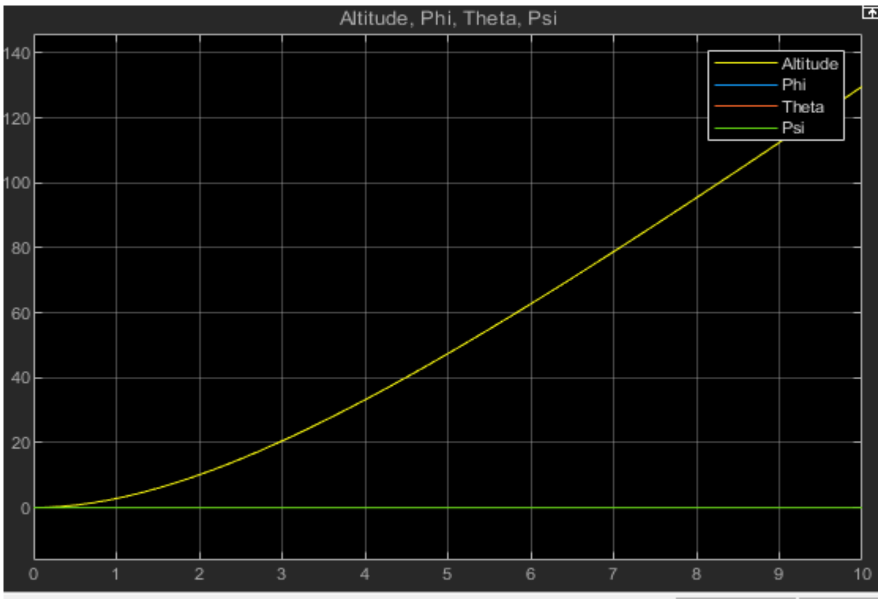


Figure 35. Model Verification Altitude

Changing *M2* and *M4* or *M1* and *M3* speeds at the same time will cause the change of *pitch angle*. Similarly, changing *M1* and *M2* or *M3* and *M4* speeds at the same time will cause *roll angle* change. For verification M2 and M4 are increased, results in affecting *pitch angle theta* (Figure 36). Changing all 4 motors speed accordingly affects the *yaw angle psi*. This verifies that the model is built correctly and works properly.

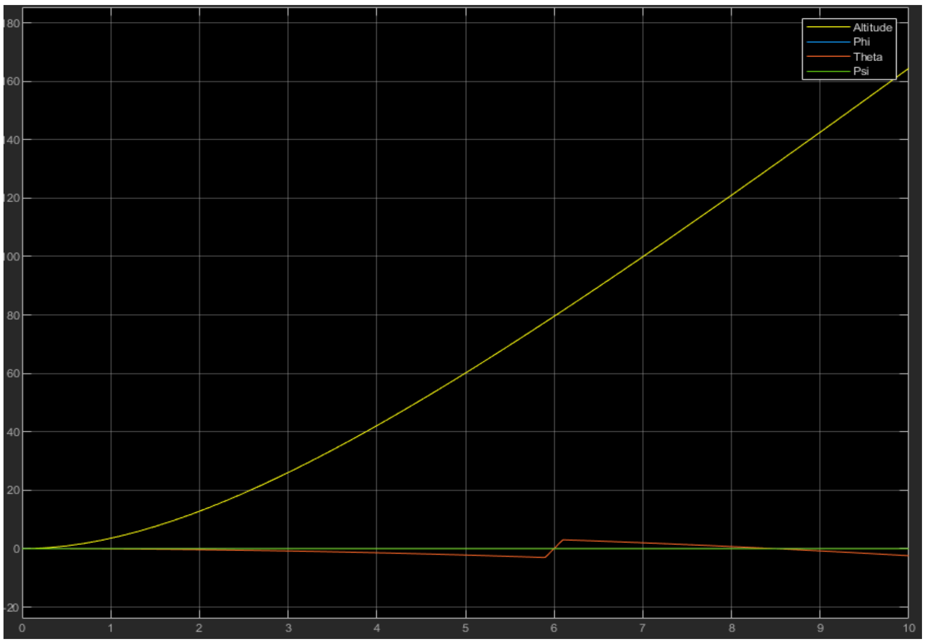


Figure 36. Model Verification Theta

5 PID Controller

Controller is very important for a vehicle to work properly. Without a controller the quadcopter will be unstable. There are many control strategies and algorithms but for this project, a feedback control system comprising PID controllers is utilized.

A PID (Proportional Integral and Derivative) controller has three control terms (Figure 37). The input to the PID controller is an error signal (difference between actual and desired value), which is multiplied by PID gains to acquire the control of the plant or process. Proportional term compares desired and actual value and simply multiply the error value to a constant. Integral term removes steady state error by integrating the error value over time until it reaches zero. Derivative term estimates about the future error and removes it. It estimates the error by the rate of change of error with time and multiply it with derivative constant. (Bright Hub Engineering 2019.)

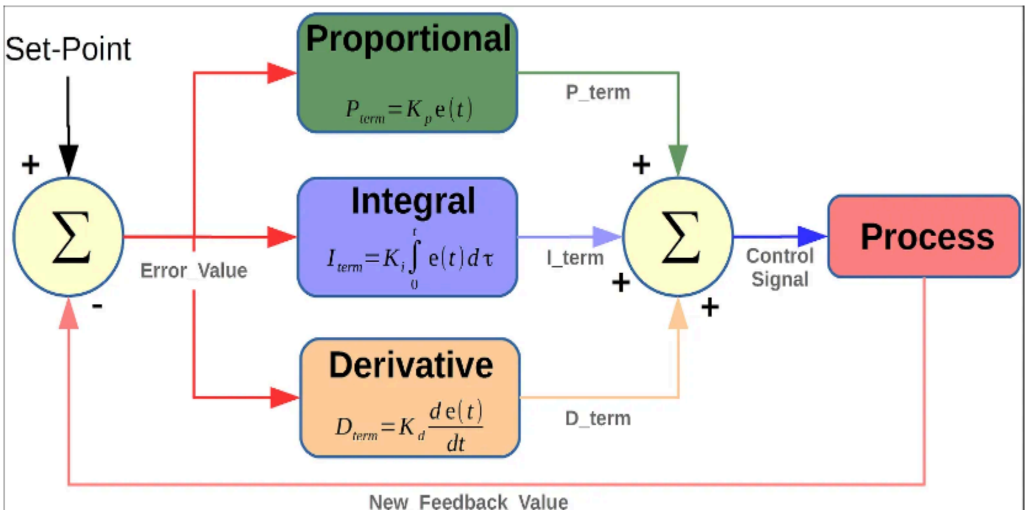


Figure 37. PID Controller (Bright Hub Engineering 2019)

5.1 Plant Model Controller

We are familiar with the basic working principle of the PID controller and have verified the plant model obeying the real-world constraints. The purpose of the project is to control the attitude of the copter, but a controller for the altitude is also used because it is necessary to control the altitude to tune the attitude (*roll, pitch* and *yaw*) controllers. A PID controller can be built by using simple gains in Simulink and there is also a built-in PID block in Simulink. Built-in PID blocks are utilized in the controller building of the vehicle. A PID based controller is applied to control the actual values obtained from the plant model according to the desired input values. In total, we have four PIDs for *altitude, roll*, *pitch* and *yaw*. Then, the output of the PID controllers' splits into motor commands for *x* orientation of the quadcopter. The plant model with controller looks as follows (Figure 38 and 39).

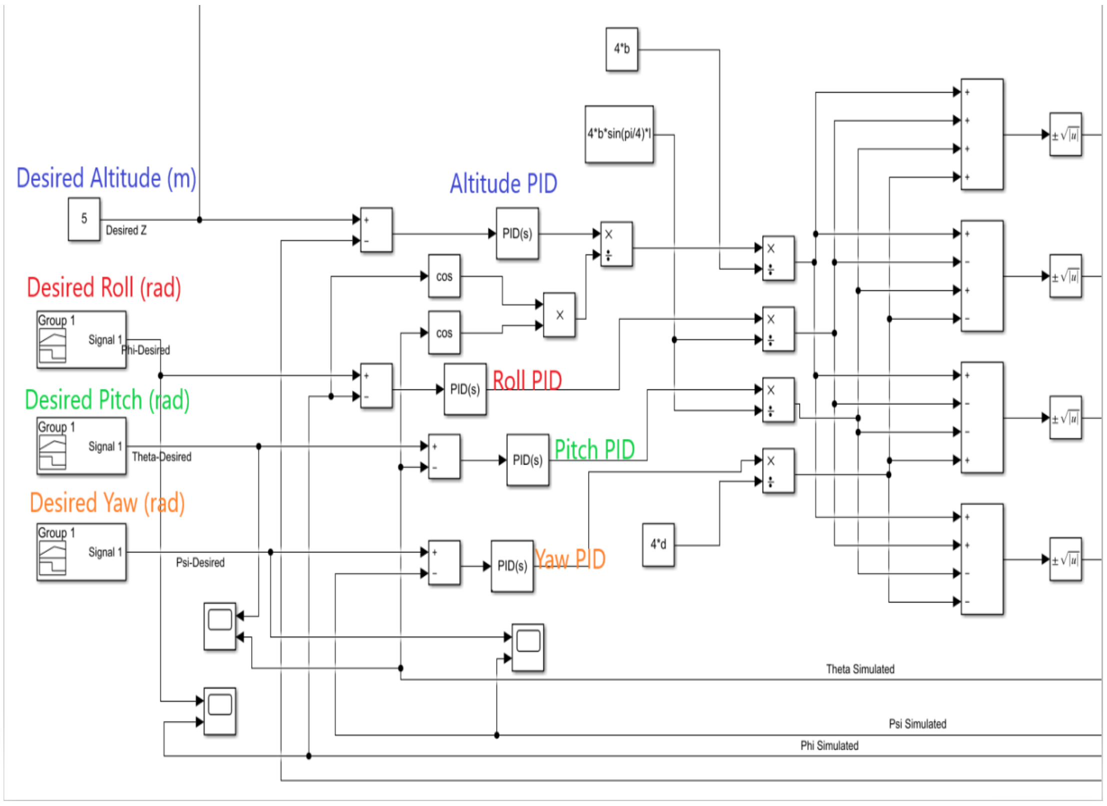


Figure 38. Input commands and Controller

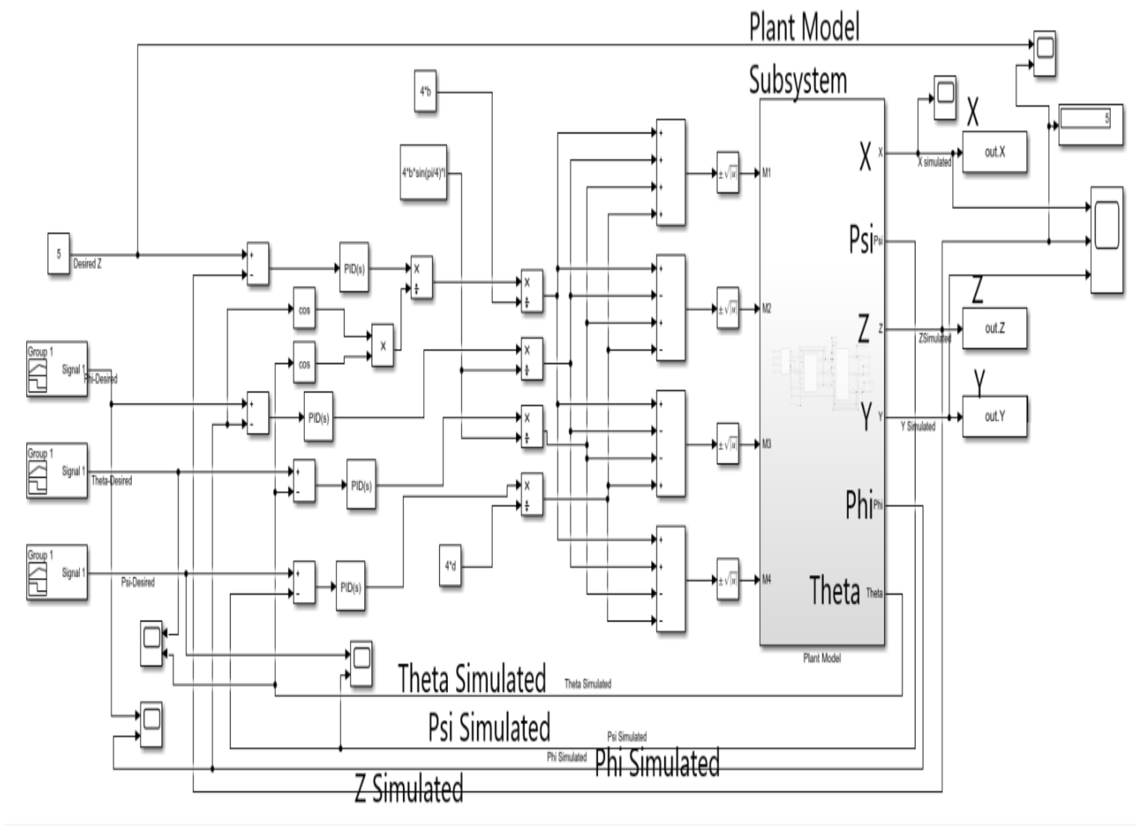


Figure 39. Controller with Plant Model

5.2 Controller Tuning

A PID controller can be tuned manually or using auto-tuning function in Simulink. Each of the PID is tuned individually for each of the desired input control. While tuning a PID controller manually, a simple procedure can be followed but still the process can take a very long time and can be iterative.

5.3 Tuning Process

5.3.1 Altitude PID Tuning

The process is to tune the *P* gain first, while *I* and *D* gains are zero, and keep increasing the gain value until the actual value overshoots the desired value (Figure 40). But this is not the tuned value for *P* gain, it just indicates the range of the gain value, which might be adjusted when tuning *I* and *D* gains. Again, we are using scope for the visualization, as it saves time.

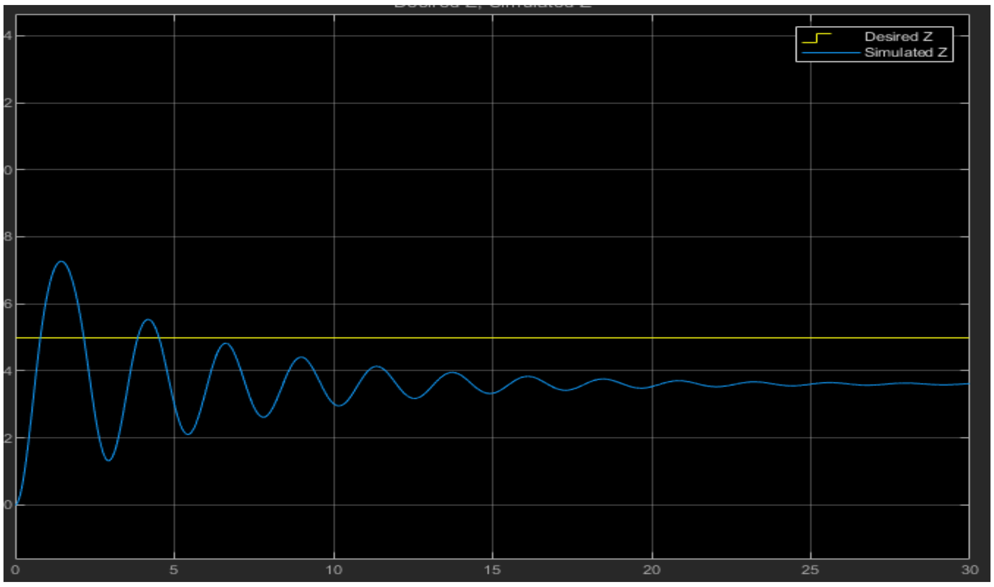


Figure 40. Altitude P Gain tuning

After *P* we tune *D* gain, it predicts and removes future errors. Increase the gain value slightly and observe its effect on the results. It will remove the future error, which will result in decreasing the oscillations in the simulation, resulting in smoothing the response (Figure 41).

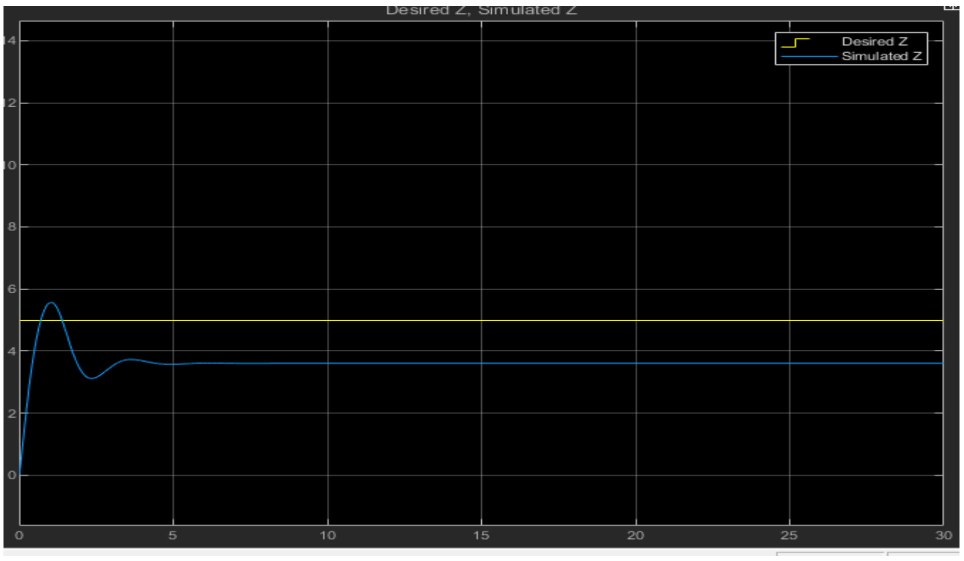


Figure 41. Tuning Altitude D Gain

Now, the actual value response is very smooth, and overshooting is negligible but there is the continuous error, and the actual response value is not the same as the desired value. This continuous error can be removed by the integrator, which accumulates the error value over time and tries to reach the desired value. It can be observed in figure 42 how *I* gain decrease the steady state error.

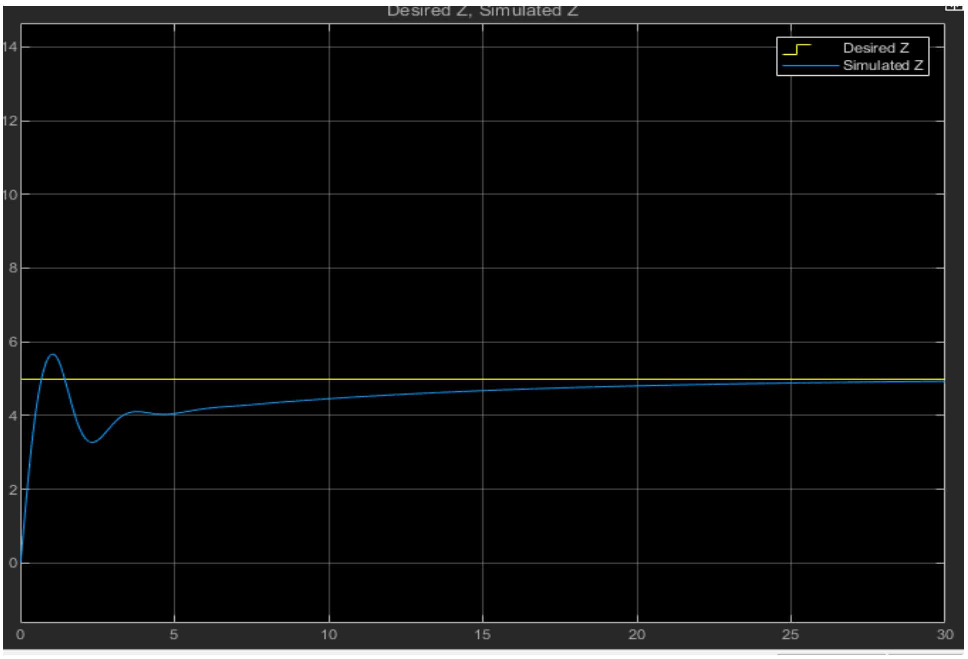


Figure 42. Altitude I Gain Tuning

After doing some adjustments and fixings, the fully tuned PID gave the following results (Figure 43). The results are quite satisfying, the quadcopter is stabilizing at an altitude of 5 m in less than 10 seconds. Same goals can be achieved in even less time, but the behavior of the vehicle will be very aggressive.

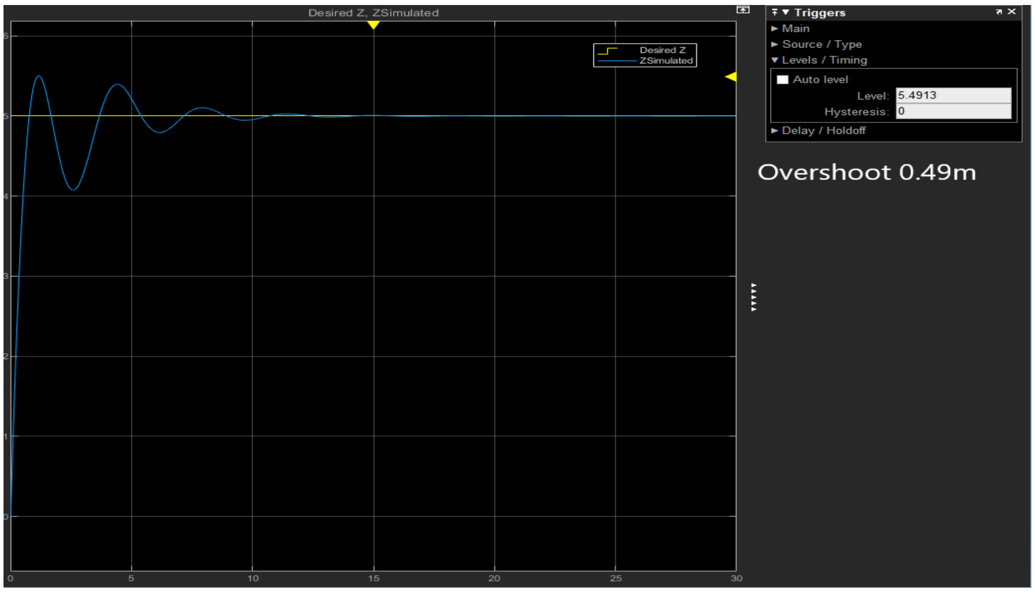


Figure 43. Tuned Altitude PID Controller Response

Figure 44 gives a more realistic representation of the vehicle behavior. Previously (Figure 43), 5 m altitude command was given to the vehicle from the beginning of the simulation which is unlikely to happen, contrary to this in the real world a pilot will take off rather slowly, to mimic this in simulation a ramp block (Figure 45) is used as an altitude command, signal starts from zero and keeps increasing with a slope of 1.5. It can be seen (Figure 44), that in the beginning for about three seconds the quadcopter does not take-off because the signal is insufficient. After three seconds it takes-off very smoothly and stabilizes itself according to the continuously increasing altitude command signal within 10 seconds. Table 5 represents the PID gains of tuned *altitude* PID controller.

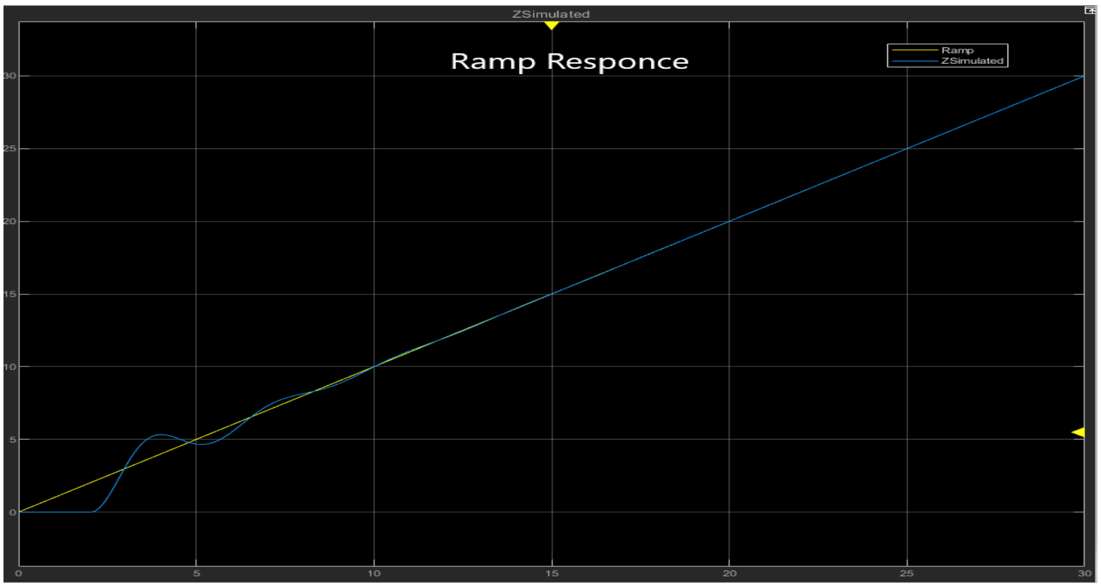
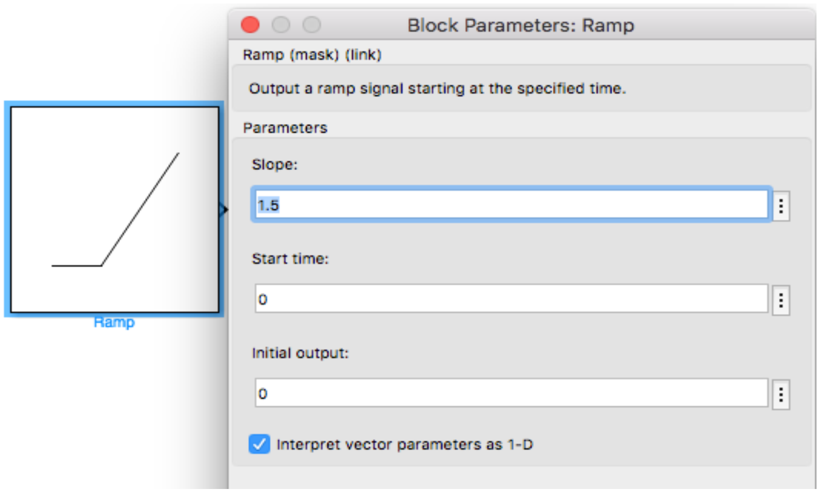


Figure 44. Tuned Altitude Real World Behaviour

Ramp is a Simulink block to create a signal, that can be started at desired time in the simulation and can be changed at a specific rate (MathWorks 2019d).



Block Parameters: Ramp

Ramp (mask) (link)

Output a ramp signal starting at the specified time.

Parameters

Slope: 1.5

Start time: 0

Initial output: 0

✓ Interpret vector parameters as 1-D

Figure 45. Ramp Block

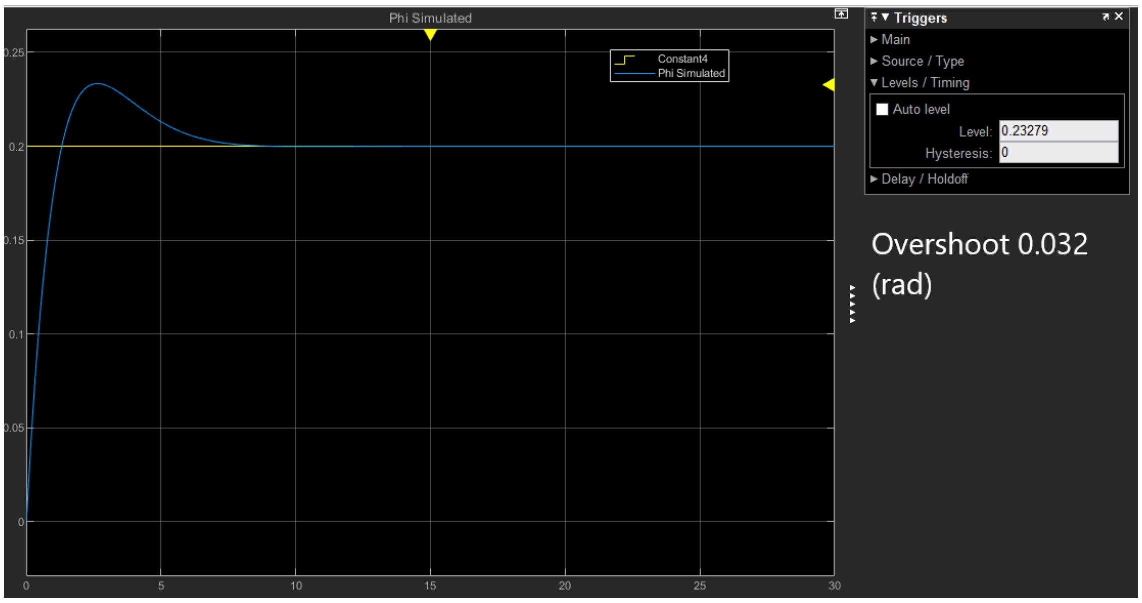
Table 5. Altitude PID gains

P	4.3
I	4.5
D	2

PID controllers for *roll, pitch* and *yaw* are also tuned, following the same procedure.

5.3.2 Roll (Phi) and Pitch (Theta) PID Tuning

Roll and *pitch* PIDs have same gain values and the same responses, as they work on the same principle. Figure 46 shows the *roll or pitch* response of the vehicle, when the desired angle is 12 degrees and the vehicle is trying to attain the altitude of 1m at the same time. Table 6 shows the PID gains of tuned *roll* and *pitch* PID controller.



Auto level

Level: 0.23279

Hysteresis: 0

Overshoot 0.032 (rad)

Figure 46. Roll and Pitch PID Controllers Tuned Response

Table 6. Roll and Pitch PID Gains

P	0.006
I	0.000009
D	0.013

5.3.3 Yaw (Psi) PID Tuning

Yaw PID values differ from *roll* and *pitch*, needed to be turned separately. It is difficult to tune for the *yaw* PID, as it depends on torque produced by all four rotors. Figure 47 shows the tuned response of the *yaw* controller, when the desired *yaw* rate is 12 degrees and the vehicle altitude command is 1 m at the same time. Table 7 gives the gains values of a tuned *yaw* PID controller.

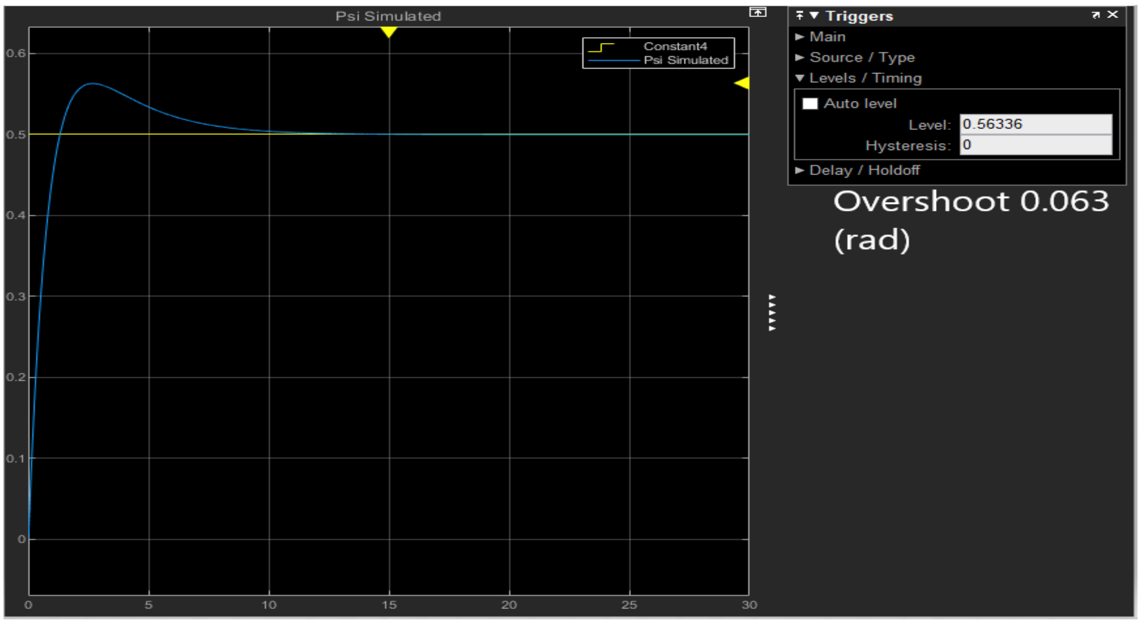


Figure 47. Tuned Yaw Controller Response

Table 7. Yaw PID Gains

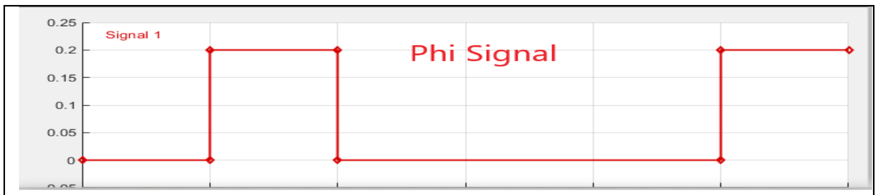
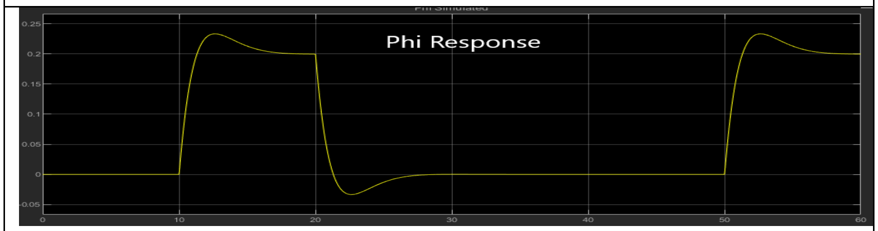
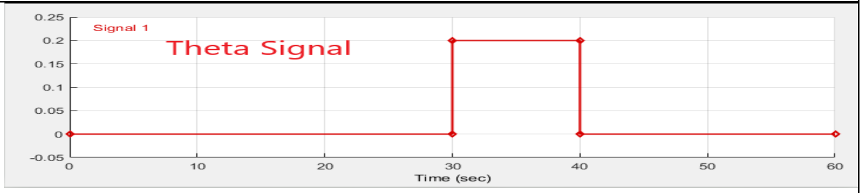
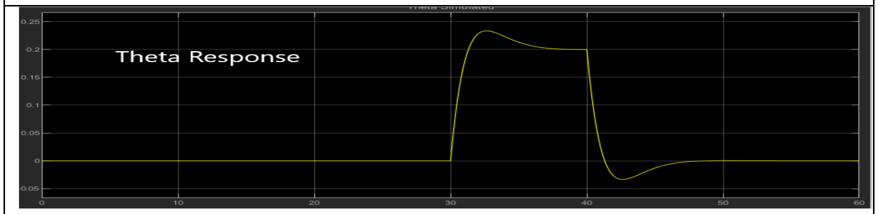
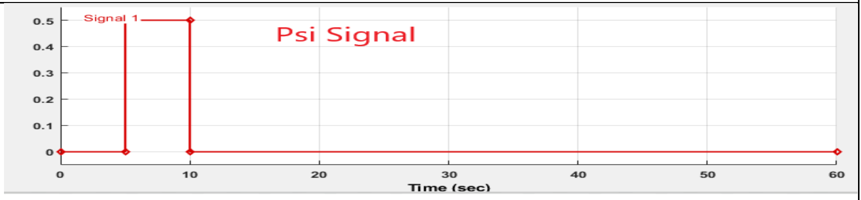
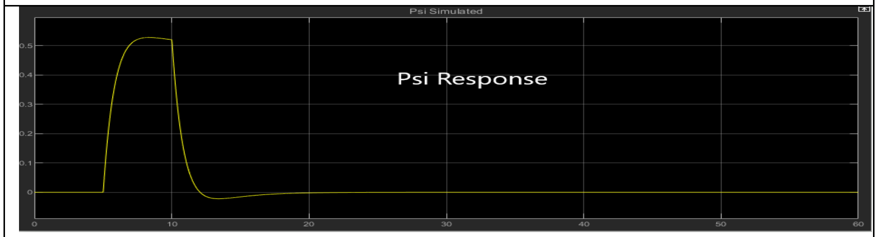
P	0.01
I	0.0002
D	0.03

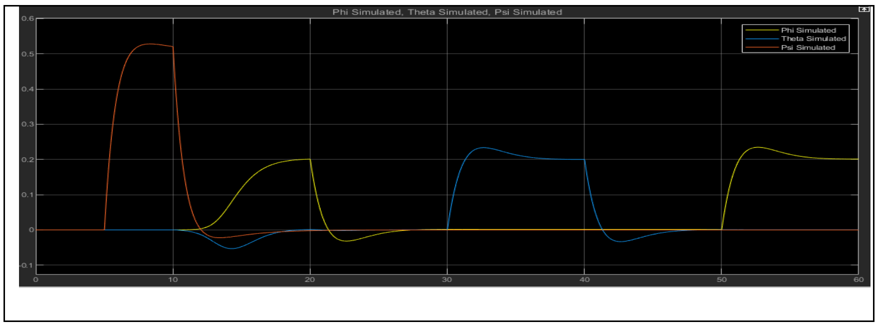
So far, all the PIDs of the controller are tuned, which works fine in the simulation, but implementation on real hardware may need to change these values slightly, because with these gain values the quadcopters response to each command will be a bit aggressive.

6 Trajectory

A trajectory is planned using a signal-builder block (Figure 48) in Simulink, to visualize the performance of the vehicle, when over one signal is applied at the same time. The inertial trajectory is random, because *roll, pitch* and *yaw* are controlled in body coordinate system, instead of *X, Y* and *Z* in inertial coordinate system. Different signal commands and simulation responses to those commands are as follows (Table 8). Simulation is run for 60 seconds and altitude command is a ramp signal (Figure 45) with a slope of 1.5.

Table 8. Signal Commands and Simulation Responses



A Group of signals can be generated, using signal builder block in Simulink. Signals are piecewise linear and interchangeable (MathWorks 2019f).

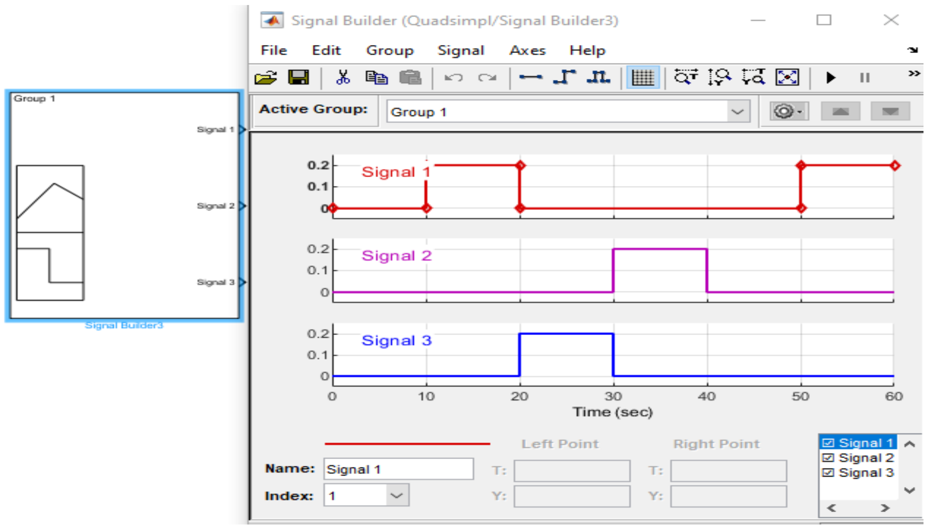


Figure 48. Signal Builder Block

It can be seen that the vehicle is responding to the commands very well (Table 8), verifying that the PID's are tuned properly. Response graphs (Table 8 and Figure 44), make sure that the controller is working and vehicle should fly obeying the commands, but we do not know, how the quadcopter will behave in the real world, to see this vehicle's trajectory is plotted in 3D (Figure 49) and an animation is created (Figure 50).

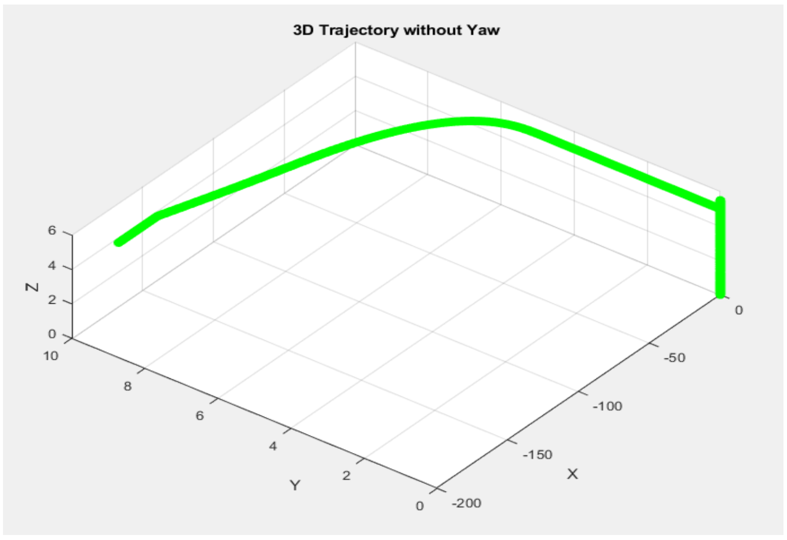


Figure 49. Quadcopter's 3D Trajectory

Animation (Figure 50) can be animated, in the word file by right click and then play from dropdown menu. The Video (Figure 50) has no sound.

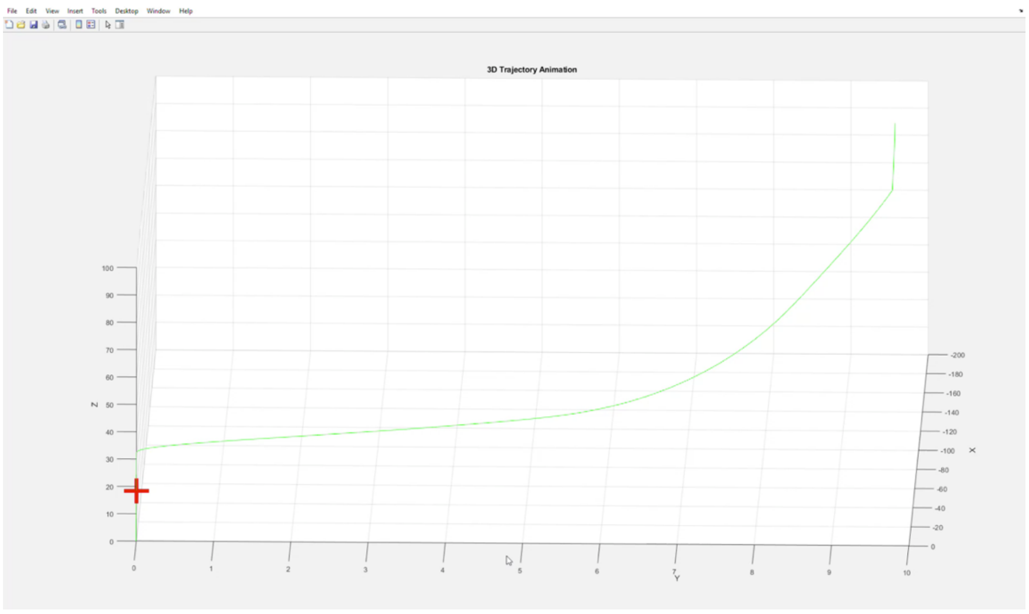


Figure 50. Trajectory Animation Video

7 Implementation

The entire simulation model is not uploaded on the hardware, as the quadcopter model based on the dynamic equations mimics the vehicle in the simulation, to develop the controller for it, so just the controller part of the Simulink model is uploaded on the real hardware. The implementation model looks as follows (Figure 51).

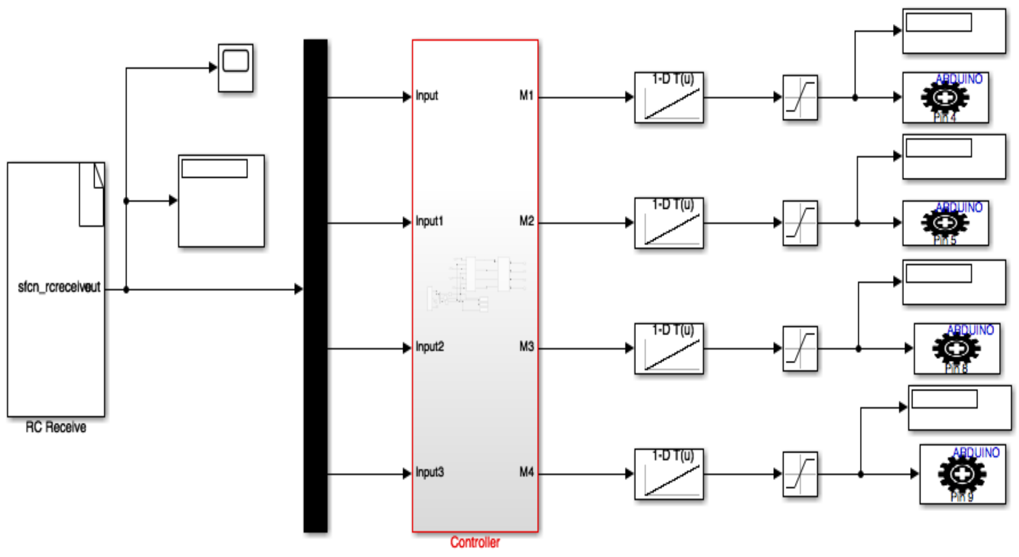


Figure 51. Implementation Model

7.1 Simulink Receiver Model

We are moving from software to real world, need an algorithm to represent the receiver model in Simulink. Receiver model will receive and process the transmitter signals to control the vehicle. An RC receiver model developed by MathWorks is used to meet the needs. The model is downloaded with the supporting files and a compiler is also downloaded for the configuration of the block. The block is developed to read four channels of the transmitter, used to control *roll, pitch, yaw* and *throttle* signal. The block is used with minute changes in the existing one (Figure 52). (Scharler 2018.)

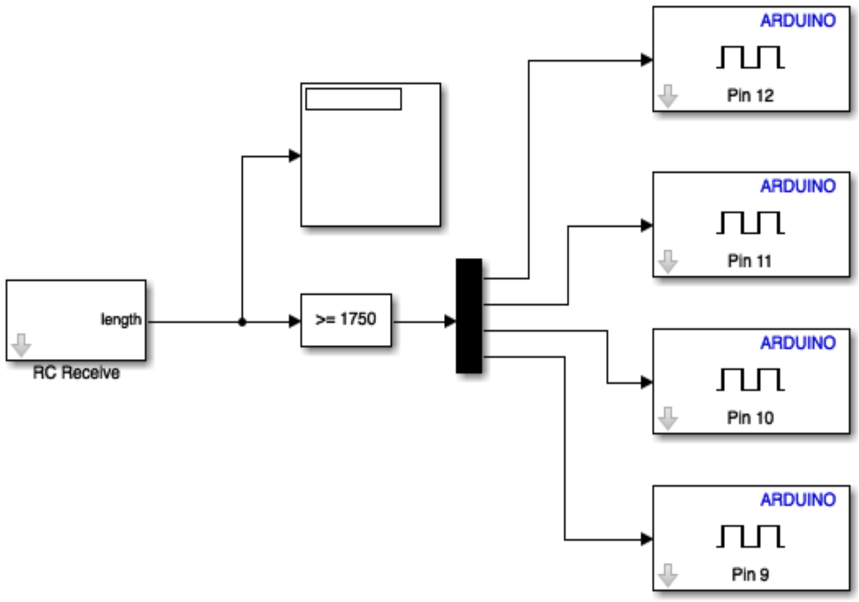


Figure 52. RC Receiver Block (Scharler 2018)

7.2 Controller

The controller subsystem of the implemented model on the real hardware looks as follows (Figure 53). It contains PIDs from the simulation model to control the vehicle and motor mixing algorithm, which control the rotor's speed according to the command signals.

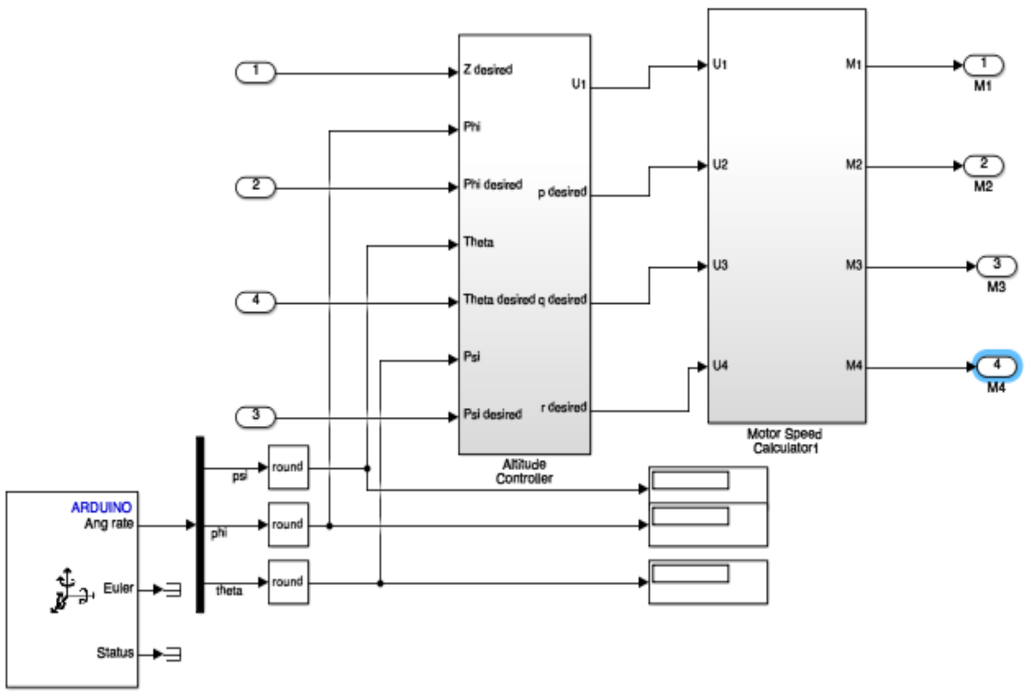
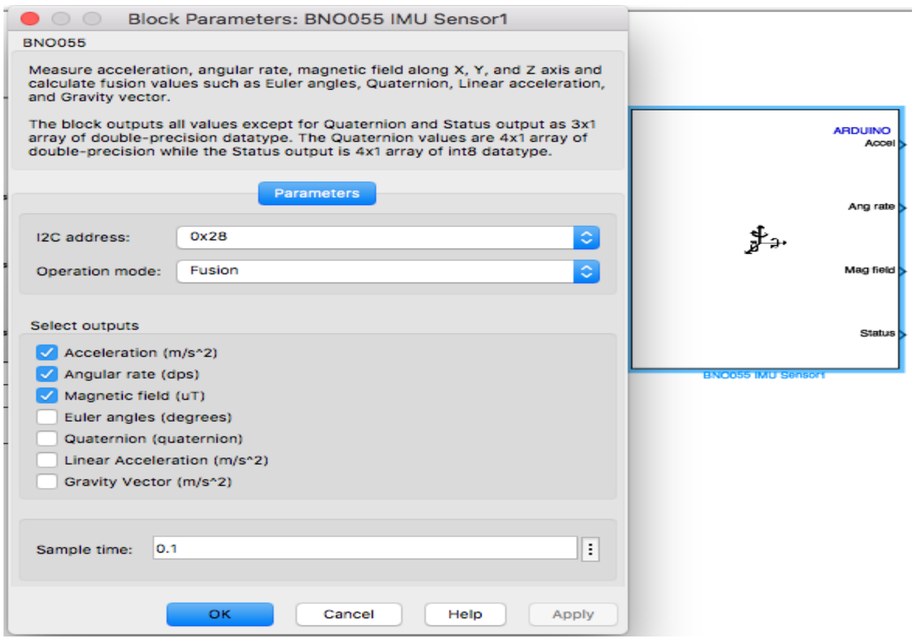


Figure 53. Implementation Model Controller Subsystem Inner View

Arduino Adafruit IMU Block

The Simulink BNO055 Arduino block (Figure 54) reads the data from the BNO055 IMU sensor attached to the hardware. The block operates in fusion and non-fusion mode, that can be selected from the block parameters window. In fusion mode block outputs calibrated values of Euler angles, Quaternions, linear acceleration, gravity vector and sensor status, while in non-fusion mode the output data is in raw form. (MathWorks 2019b.)


Block Parameters: BNO055 IMU Sensor1

BNO055

Measure acceleration, angular rate, magnetic field along X, Y, and Z axis and calculate fusion values such as Euler angles, Quaternion, Linear acceleration, and Gravity vector.

The block outputs all values except for Quaternion and Status output as 3x1 array of double-precision datatype. The Quaternion values are 4x1 array of double-precision while the Status output is 4x1 array of int8 datatype.

Parameters

I2C address: 0x28

Operation mode: Fusion

Select outputs

- [x] Acceleration (m/s^2)
- [x] Angular rate (dps)
- [x] Magnetic field (uT)
- [ ] Euler angles (degrees)
- [ ] Quaternion (quaternion)
- [ ] Linear Acceleration (m/s^2)
- [ ] Gravity Vector (m/s^2)

Sample time: 0.1

Figure 54. BNO055 IMU in Simulink

Figure 55 shows the default orientation of the IMU sensor.

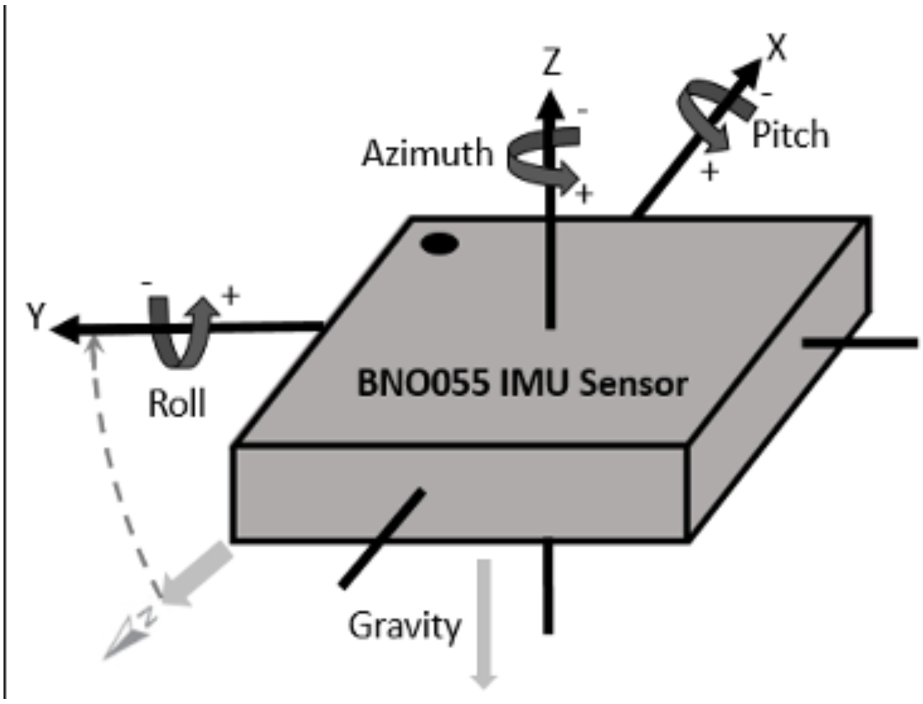


Figure 55. BNO055 Default Orientation (MathWorks 2019)

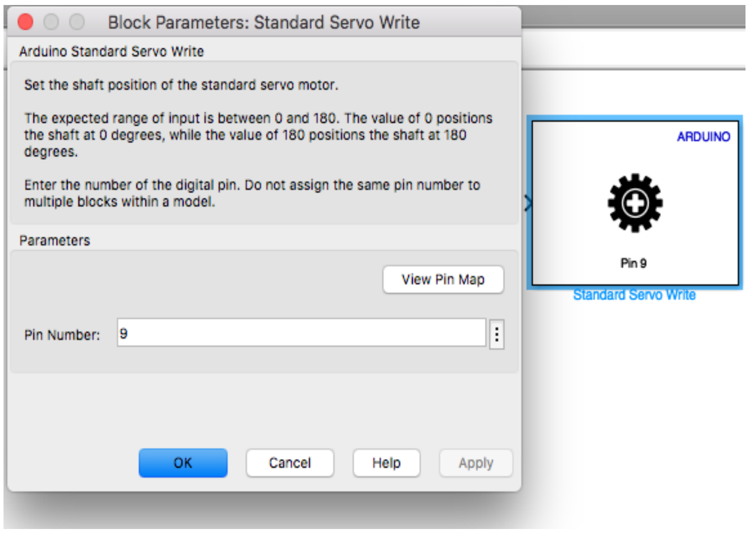
We can get calibrated values in fusion mode, we need not to filter the data. Angular rates are used to provide the attitude signal to the controller.

7.3 Scaling

Brushless Direct Current (BLDC) motors are usually controlled by electronic speed controllers (ESCs), because BLDC motors are three-phase motors. An

ESC has two wires on one end for voltage and ground, three wires on the other end to control the three-phase motor. Another set of wires in ESCs is to connect with the electronic-board's signal wire and ground. ESCs use same signal which is used to control servos, a 50HZ PWM signal with the pulse width varying from 1 to 2 milliseconds.(How to mechatronics 2019.)

So, instead of using PWM output pins (0-255 range) with different output frequency, we use standard servo write (Figure 56) pins (0-180 range) output frequency 50HZ to control BLDC motors via ESCs (MathWorks 2019c) (MathWorks 2019g). Same 50HZ PWM signal is received from receiver block in the Simulink varying from 1000 to 2000 microseconds (1 to 2 milliseconds), where 1000 and 2000 representing 0% and 100% of pulse width respectively. When this signal passes through the controller and motor mixing algorithm, it gets multiplied with PID gains. It changes range from the default value, which is not acceptable by the ESCs, so we rescaled it to make the motors working.


Block Parameters: Standard Servo Write

Arduino Standard Servo Write

Set the shaft position of the standard servo motor.

The expected range of input is between 0 and 180. The value of 0 positions the shaft at 0 degrees, while the value of 180 positions the shaft at 180 degrees.

Enter the number of the digital pin. Do not assign the same pin number to multiple blocks within a model.

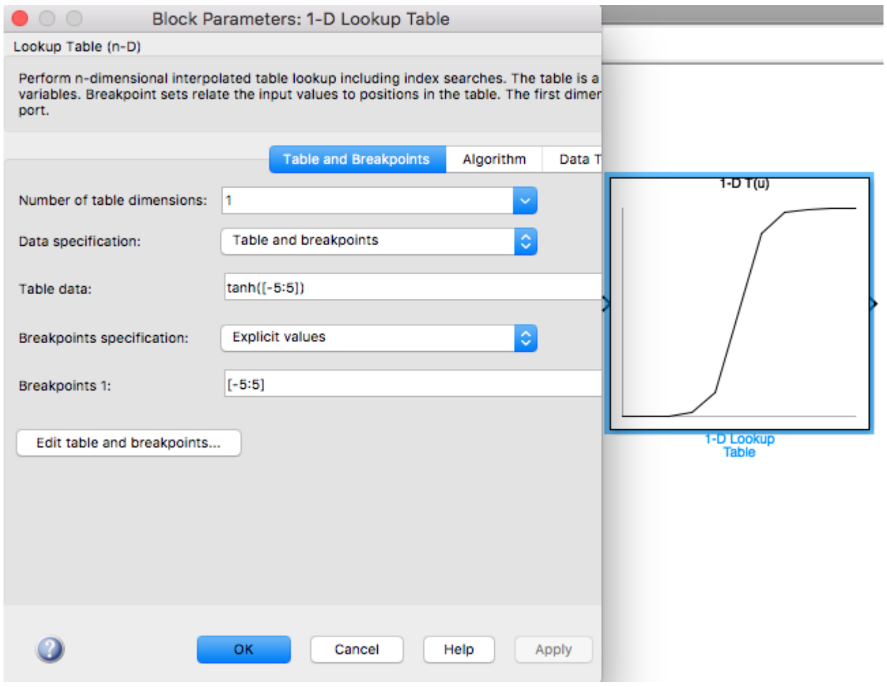
Parameters

Pin Number: 9

Figure 56. Standard Servo Write in Simulink

Receiver Signal Scaling

An ESC can use 1000 to 2000 pulse width range to control the motor, when it is directly connected to the actual receiver in hardware, but in our case it is connected to the receiver via micro-controller (ATmega 2560), using standard servo write pins with acceptable signal value range from 0 to 180. So, we mapped the motor mixing output values from 0 to 180 using 1-D lookup tables (Figure 57).1-D lookup table uses one-dimensional array data as input and outputs the mapped values accordingly in the mapping range. The block looks up or interpolate a table of values defined in the block parameters to map input values to output values. (MathWorks 2019a.)


Block Parameters: 1-D Lookup Table

Lookup Table (n-D)

Perform n-dimensional interpolated table lookup including index searches. The table is a variables. Breakpoint sets relate the input values to positions in the table. The first dimer port.

Number of table dimensions: 1

Data specification: Table and breakpoints

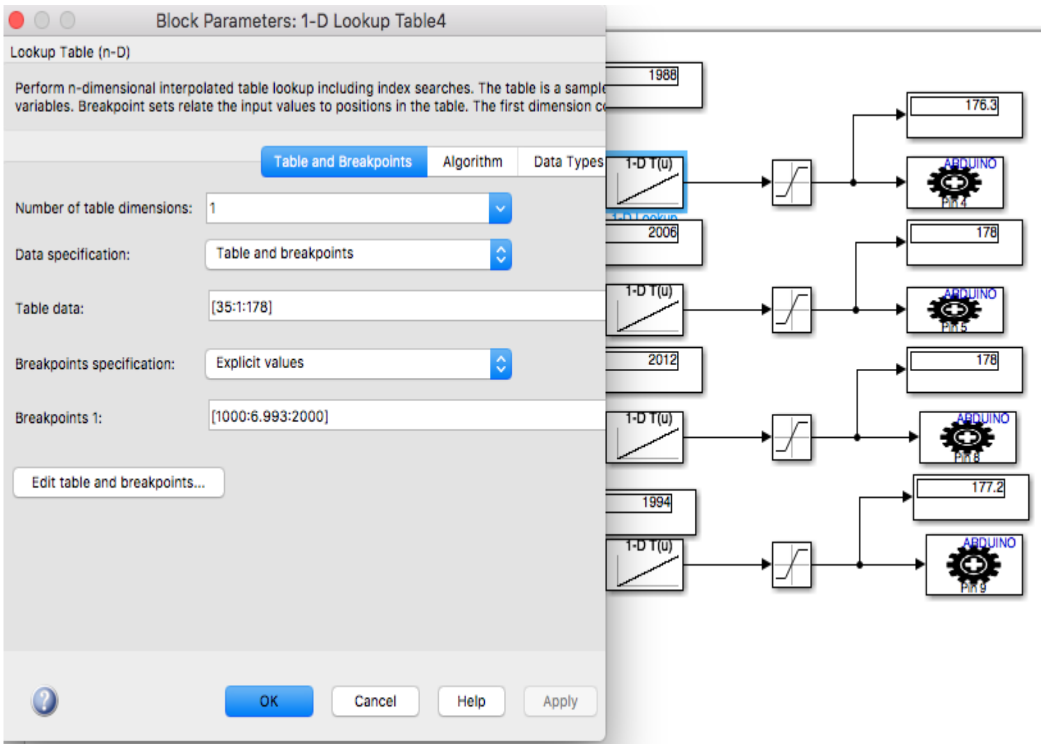
Table data: tanh([-5:5])

Breakpoints specification: Explicit values

Breakpoints 1: [-5:5]

Figure 57. 1-D Outlook Block in Simulink

Theoretically, it should work. But in reality, it was not true. ESCs were not calibrating at the same time and the motors were not running at the same speed, which is useless for this project application. Output values were mapped and tested again and again varying the range between 0 and 180. Finally, a sweet spot was found, where all the motors were calibrating at the same time and all the rotors started rotating at the same time with same speed. The signal value from 1000 to 2000 is mapped between 35 to 178 considering the experimental experience to be on the safe side. Figure 58 shows the 1D lookup table data and explicit values used for the mapping.


Block Parameters: 1-D Lookup Table4

Lookup Table (n-D)

Perform n-dimensional interpolated table lookup including index searches. The table is a sampl
variables. Breakpoint sets relate the input values to positions in the table. The first dimension c

Table and Breakpoints | Algorithm | Data Types

Number of table dimensions: 1

Data specification: Table and breakpoints

Table data: [35:1:178]

Breakpoints specification: Explicit values

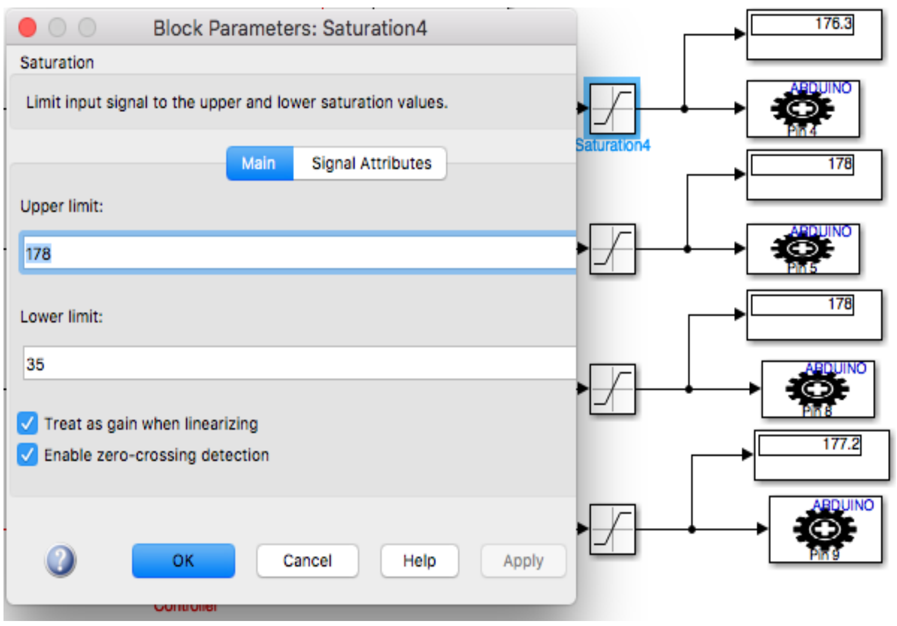
Breakpoints 1: [1000:6.993:2000]

Edit table and breakpoints...

OK | Cancel | Help | Apply

Figure 58. Lookup Table Explicit Values and Break Points

Values can exceed the mapped range leading to an error, burned out ESCs, overheated motors and unpredictable behavior of the vehicle. Depending on the input signal and state of the vehicle, signal value can go below 1000 (35 after mapping) and above 2000 (178 after mapping), to avoid this saturation blocks (Figure 59) are used which limit signal values with a lower limit 35 and upper limit 178 (MathWorks 2019e).


Block Parameters: Saturation4

Saturation

Limit input signal to the upper and lower saturation values.

Main | Signal Attributes

Upper limit: 178

Lower limit: 35

☑ Treat as gain when linearizing
☑ Enable zero-crossing detection

OK | Cancel | Help | Apply

Figure 59. Saturation Blocks for Limiting Signals Values

7.4 Implementation Model Tuning

Even though, we have tuned controller for our plant model. The gain values are not working on the real hardware as expected, this could be because of signal scaling, different range of input values, Arduino output pins fixed range of acceptable signal values and gust factor. So, we have to tune PIDs again for our actual hardware, while the controller is running on the hardware. This is done by running a real time simulation. The tuning process is the same as discussed before. Different methods are applied to tune the PIDs, while the program is running on the quadcopter some of them are shown in Figure 60 and 61 below.

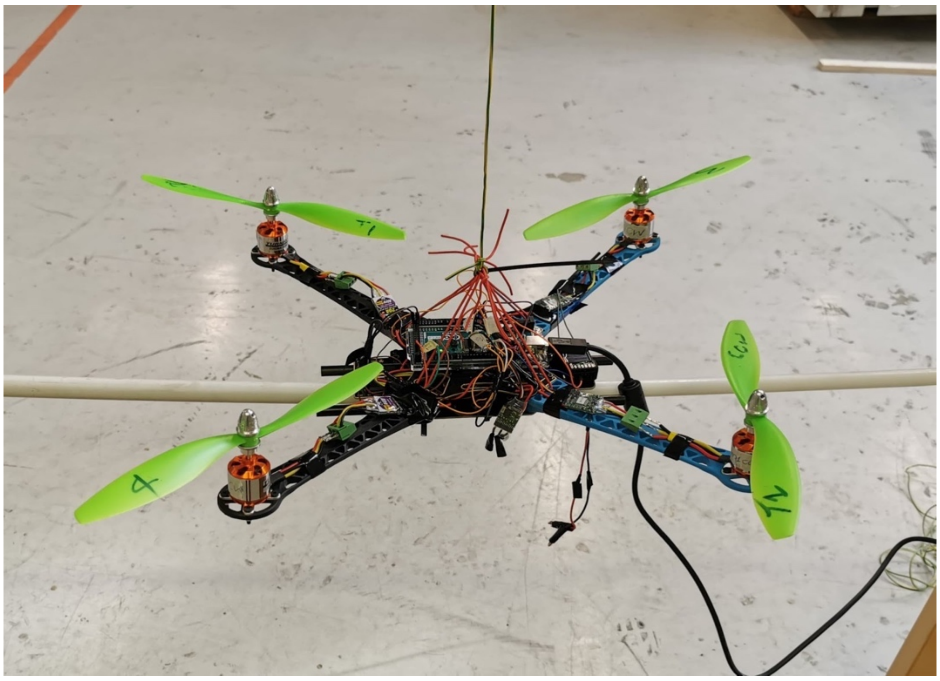


Figure 60. Tuning Roll and Pitch

We fixed the quadcopter on a stick, so it cannot fly, the stick allows the quadcopter to rotate around the axis of the stick. We fixed the quadcopter on the stick for both *roll* and *pitch* orientation and tried to tune the PIDs while the program was running on the hardware (Figure 60).

Then we fixed the quadcopter with ropes from below and above, so it cannot fly, but it can rotate around the axis of its body. This way we tried to tune the PID for yaw control (Figure 61).

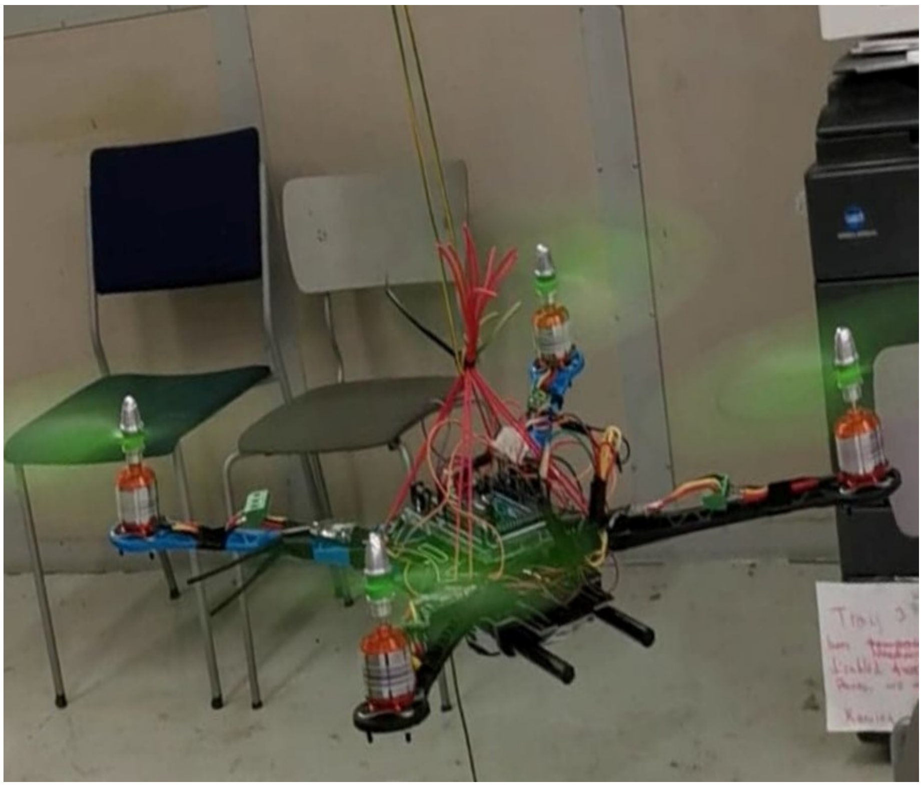


Figure 61. Yaw Tuning

Sometimes the quadcopter could take-off, but with a very poor control. Controller needs more tuning for better control of the vehicle.

8 Conclusion

Quadcopters with accelerating industry are an interesting platform to perform different control strategies. The goal of the project was to develop a PID based Simulink controller and its implementation on the real hardware to control the attitude of the vehicle.

The vehicle was built successfully using the selected components, compatible to each other. The vehicle's Simulink model was made, based on derived differential equations of motion. Simulink model simulated perfectly and was verified by giving the motors RPM commands to confirm that the model is depicting the vehicle's dynamics. A PID based controller was designed and tuned for the quadcopter's Simulink model, comprising of a separate PID controller block for each variable to be controlled. A 3D trajectory animation was created for visualization.

The controller was modified for real world implementation according to the selected electronics platform and was implemented on the hardware to check its performance. Signals from the radio receiver were scaled so, that they were within the acceptable signal ranges of input and output pins, of electronics platform, while having the frequency acceptable by ESCs. All the ESCs were calibrated at the same time, which was very important to meet other goals of the project.

A simulink based controller worked on the hardware, but because of signals scaling and the different range of input signal values to the ESCs the controller needs to be tuned again. Because of unreliable real-time data visualization, hardware limitations and lack of time, PIDs are not tuned properly. In an existing condition the quadcopter can take off from the ground, but it is not very stable and keeps moving in random directions.

9 Future Work

Better control can be achieved, with some advancements in the existing Simulink model of the vehicle and controller. Equations for propellers and BLDC motors can be derived instead of using constants. The existing Simulink model of the vehicle can be optimized, by introducing the propellers and motors Simulink models, to get dynamic behavior of the model closer to vehicle behavior in the real world. Further a control strategy can be used to control the BLDC motors, depending on the equations derived for motors to get precise control of the motors. Existing Simulink controller for the vehicle can also be made more effective by adding more PID controller blocks. The IMU sensors' data can be filtered using Kalman filter to get more accurate data from sensors. PID controllers' gains can be tuned more to get better results.

An ESP8266 WI-FI module can be connected with Arduino to observe real-time data with safety, or an Arduino with built in Wi-Fi module can be used to meet the same purpose. This will help in tuning gains of PIDs easily maintaining a safe distance from the vehicle. The micro-controller can be replaced with raspberry-pi or ArduCopter with better processing power and storage Static Random-Access Memory (SRAM).

The above-mentioned improvements would be enough to meet the main goals of this project. A lot can be done in this project to achieve different objectives. We can introduce a camera, navigation control or any other sensor and equipment depending on the aim.

Appendices 1

A.1 Weight of The Vehicle

Weight of the vehicle can be estimated by adding weight of all of the hardware components.

- Arduino mega=37g
- BNO055 IMU=5g
- Brad Board=10g
- LIPO Battery=208.2g
- RX Receiver=9.8g
- 4 ESCs=42.8g
- Frame=405g
- 4 Motors=280g
- 4 Propellers=20g

Total weight of the vehicle=m=992.8g=0.9928Kg

A.2 Inertia of the Vehicle

Moment of inertia estimation is very important to control the vehicle it limits the amount of torque to produce angular acceleration around an axis. Moment of inertia can be estimated in various ways. One way is to calculate the moment of inertia of each component separately and then add the moment of inertia of components to get moment of inertia for *X, Y* and *Z* orientation of the vehicle. Here we have used a rather simple approach we know the mass of the vehicle and we have assumed that the vehicle is rigid and uniform so we can simply distribute the mass equally for each arm of the quadcopter to find moment of inertia of each arm which can be added accordingly to find moment of inertia for *X, Y* and *Z* orientation of the quadcopter.

A.2.1 Yaw Moment of Inertia

Moment of inertia in Z direction can also be called as yaw moment of inertia as it will resist the yaw moment. It is simply the sum of moment of inertia of all four arms of the quadcopter (Ferry 2017).

- Mass of the vehicle =M=0.9928kg
- Length of the arm=L=0.24m

- Moment of inertia=I_{zz}

$$I_{zz} = \left(\frac{1}{3} \times ml^2\right) \quad (40)$$

$$I_{zz} = \left(\frac{1}{3} \times \frac{0.9928}{4} \times 0.24^2\right) \times 4 \quad (41)$$

$$I_{zz} = 0.019kgm^2 \quad (42)$$

A.2.2 Roll and Pitch Moment of Inertia

Moment of inertia in *X* and *Y* direction is same as the vehicle is symmetrical It can be found using the same equation as above the only thing changed in this is the length of the arm instead of using the length of the arm we use perpendicular distance of the motors from the center of the vehicle.

$$I_{xx} = I_{yy} = 0.00963kgm^2 \quad (43)$$

A.3 Propeller Moment of Inertia

Propellers moment of inertia is taken from Tomas Jiinec's master thesis. As he has calculated the moment of inertia of the same propellers 10x4 as we are using in our project. He has estimated the moment of inertia by dividing the propeller into small sections and then calculating the moment of inertia by using parallel axis theorem.(Selby 2019.)

$$b = 0.04439kgm^2 \quad (44)$$

A.4 Thrust Coefficient

Thrust coefficient depends on weight of vehicle, motor RPM and propeller pitch and diameter. Thrust coefficient can be obtained by analyzing the thrust value at different RPMs of the motor. It is simpler to calculate the thrust coefficient when the quadcopter is hovering. When quadcopter is hovering the thrust of all four rotors is equal to the weight of the vehicle. So, thrust of one rotor is a quarter of the weight of the quadcopter.(Selby 2019.)

Thrust=thrust coefficient x motor rotation speed

$$b = \frac{mg}{4} \div (omega)^2 \quad (45)$$

As we do not have the omega value at hovering for our vehicle, so we have estimated the omega value from a model developed for another quadcopter. In which same propellers are used as in our project with different weight of the vehicle. The estimated omega for our rotor is around 260.

$$b = \frac{0.9928 \times 9.8}{4} \div (260)^2 \quad (46)$$

$$b = 3.59 \times 10^{-5}kgm \quad (47)$$

A.5 Drag Torque Coefficient

Drag coefficient depends on the Inertia of the vehicle, motor RPM and propellers pitch and diameter. Drag coefficient is estimated as a percentage of the drag coefficient of another model with same propellers but different moment of inertia.

$$d = 2.0810 \times 10^{-6} \frac{kgm^2}{s^2} \quad (48)$$

A.6 Air Resistance

Measuring the air resistance is extremely difficult so a simple estimated value is used which do not affect the behavior of the quad significantly in closed environment. (Ferry 2017.)

$$d = 0.1 \frac{kg}{s} \quad (49)$$

Figures

Tables

References

Adafruit (2019) *BNO055 Absolute Orientation Sensor*.
https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/
Accessed in June 2019.

AGENCY, D. A. R. P. (2017) *Fast Lightweight Autonomy (FLA)*.
https://www.darpa.mil/program/fast-lightweight-autonomy
Accessed in June 2019.

ARDUINO (2019) *ARDUINO MEGA 2569*.
https://store.arduino.cc/arduino-mega-2560-rev3
Accessed in June 2019.

Black Tie Aerial (2014) *The Physics of Quadcopter Flight*.
https://blacktieaerial.com/the-physics-of-quadcopter-flight/
Accessed in June 2019.

Bright Hub Engineering (2019) *PID Controller Tuning*.
https://www.brighthubengineering.com/robotics/119653-pid-loop-tuning-methods-for-robotics/
Accessed in August 2019.

ÇAMLICA, F. B. (2004) *DEMONSTRATION OF A STABILIZED HOVERING PLATFORM FOR*. MIDDLE EAST TECHNICAL UNIVERSITY.
http://citeseerx.ist.psu.edu/
Accessed in June 2019.

Circuit Basics (2016) *BASICS OF THE I2C COMMUNICATION PROTOCOL*.
http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
Accessed in July 2019.

Circuitdigest (2019) *I2C Communication*.
https://circuitdigest.com/microcontroller-projects/arduino-i2c-tutorial-communication-between-two-arduino
Accessed in July 2019.

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY (2017) *Smart Quadcopters Find their Way without Human Help or GPS*.
https://www.darpa.mil/news-events/2017-06-28
Accessed in June 2019.

Electronobs (2019) *Arduino Multiwii flight controller*.
https://www.electronoobs.com/eng_robotica_tut5_3.php
Accessed in July 2019.

Ferry, N. (2017) *Quadcopter Plant Model and Control System Development With MATLAB/Simulink Implementation*. Rochester Institute of Technology.
http://www.ritravvenlab.com
Accessed in June 2019.

Fiaz, U. A. and Mukarram, A. (2018a) *Altitude Control of a Quadcopter*. Pakistan Institute of Engineering & Applied Sciences. https://www.researchgate.net/publication/309486306_Altitude_Control_of_a_Quadcopter

Fiaz, U. A. and Mukarram, A. (2018b) *Altitude Control of a Quadcopter*. Pakistan Institute of Engineering and Applied Sciences (PIEAS). https://www.researchgate.net/publication/309486306_Altitude_Control_of_a_Quadcopter

HobbyKing (2019a) *Afro 20A Race Spec Mini ESC Opto*. https://hobbyking.com/en_us/afro-race-spec-mini-20amp-opto-multi-rotor-speed-controller.html
Accessed in June 2019.

HobbyKing (2019b) *HobbyKing Slowfly Propeller 10x4.5 Black (CW/CCW) (4pcs)*. https://hobbyking.com/en_us/10x4-5-sf-props-2pc-cw-2pc-ccw-green.html
Accessed in June 2019.

HobbyKing (2019c) *S500 Glass Fiber Quadcopter Frame 480mm - Integrated PCB Version*. https://hobbyking.com/en_us/s500-glass-fiber-quadcopter-frame-480mm-integrated-pcb-version.html
Accessed in June 2019.

HobbyKing (2019d) *Turnigy D2836/9 950KV Brushless Outrunner Motor*. https://hobbyking.com/en_us/turnigy-d2836-9-950kv-brushless-outrunner-motor.html
Accessed in June 2019.

HobbyKing (2019e) *Turnigy TGY-i6S Digital Proportional Radio Control System*. https://hobbyking.com/en_us/i6s-afhds-2a-white-mode2-6ch-radio-with-colour-box.html?___store=en_us
Accessed in June 2019.

HobbyLinna (2019) *Li-Po Battery 3S 11,1V 2200mAh 30C T-Connector*. https://www.hobbylinna.fi/product/li-po-11-1v-2200mah-30c-dean-air/VPLP020FD/
Accessed in June 2019.

How to mechatronics (2016) *MEMS Accelerometer Gyroscope Magnetometer & Arduino*. https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/
Accessed in July 2019.

How to mechatronics (2019) *Arduino Brushless Motor Control Tutorial | ESC | BLDC*. https://howtomechatronics.com/tutorials/arduino/arduino-brushless-motor-control-tutorial-esc-bldc/
Accessed in July 2019.

Markets, M. (2018) *Unmaned Aerial Vehicle (UAV) Market*. Hadapsar, Pune-411013, India.
https://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html
Accessed in June 2019.

MathWorks (2019a) *1-D Lookup Table*.
https://se.mathworks.com/help/simulink/lookup-tables.html
Accessed in August 2019.

MathWorks (2019b) *BNO055 IMU Sensor*.
https://se.mathworks.com/help/supportpkg/arduino/ref/bno055imusensor.html
Accessed in August 2019.

MathWorks (2019c) *PWM*.
https://se.mathworks.com/solutions/power-electronics-control/pulse-width-modulation.html
Accessed in July 2019.

MathWorks (2019d) *Ramp*.
https://se.mathworks.com/help/simulink/slref/ramp.html?searchHighlight=Ramp&s_tid=doc_srchtitle
Accessed in August 2019.

MathWorks (2019e) *Saturation*.
https://se.mathworks.com/help/simulink/slref/saturation.html
Accessed in August 2019.

MathWorks (2019f) *Signal Builder*.
https://se.mathworks.com/help/simulink/slref/signalbuilder.html
Accessed in August 2019.

MathWorks (2019g) *Standard Servo Write*.
https://se.mathworks.com/help/supportpkg/arduino/ref/standardservowrite.html
Accessed in July 2019.

Pro Maker (2016) *DIY Drone: How to Build a Quadcopter*.
https://maker.pro/custom/projects/diy-quadcopter-tutorial
Accessed in July 2019.

Scharler, H. (2018) *R/C Controller for Arduino and Simulink, MIT*.
https://www.hackster.io/matlab-makers/r-c-controller-for-arduino-and-simulink-be5aee
Accessed in June 2019.

Selby, W. (2019) *Model Verification*.
https://www.wilselby.com/research/arducopter/model-verification/
Accessed in July 2019.

Technaid (2019) *IMU Working Principles*.
https://www.technaid.com/support/research/imu-working-principles/
Accessed in July 2019.

Technical University of Munich (2014) *Micro Aerial Vehicles (MAVs)*.
https://vision.cs.tum.edu/research/quadcopter
Accessed in June 2019.

TECTARGET (2019) *MICROCONTROLLER*.
https://internetofthingsagenda.techtarget.com/definition/microcontroller
Accessed in August 2019.

Tytler, C. (2017a) *Modeling Vehicle Dynamics – Euler Angles*.
https://charlestytler.com/modeling-vehicle-dynamics-euler-angles/
Accessed in July 2019.

Tytler, C. (2017b) *Modeling Vehicle Dynamics – Quadcopter Equations of Motion*. https://charlestytler.com/quadcopter-equations-motion/
Accessed in July 2019.

VBROADCAST LIMITED (2019) *Drones Applications at Present and Future*, *Wondershare*.
https://filmora.wondershare.com/drones/drone-applications-and-uses-in-future.html
Accessed in June 2019.

Wikipedia (2019) *Six degrees of freedom*.
https://en.wikipedia.org/wiki/Six_degrees_of_freedom
Accessed in June 2019.