

Modeling, PID Control & Simulation of Quadcopter

FINAL PROJECT REPORT

Table of Contents

1	Introduction
2	Equations of Motion
3	Simulink Walkthrough
3.1	Quadcopter Block
3.1.1	Quadcopter before transformation without assumption block
3.1.2	Transformation Frames
3.1.3	Altitude Function
3.2	Controller Block
3.2.1	Altitude Controller
3.2.2	Pitch Controller
3.2.3	Roll Controller
3.2.4	Yaw Controller
3.2.5	Sum Block
3.3	Results from Scopes
4	FlightGear Visualization
4.1	Importing Geometry
4.1.1	Saving Geometry to AC3D
4.1.2	Importing to FlightGear
4.2	Linking Simulink to FlightGear
4.2.1	Flat Earth to LLA block
4.2.2	Flight Gear Preconfigured 6DOF
4.2.3	Generating .bat file
4.2.4	Simulation Run
5	Robust Control

1 Introduction:

Sensing and actuating technologies developments make, nowadays, the study of mini Unmanned Air Vehicles (UAVs) very interesting. Among the UAVs, the VTOL (Vertical Take Off and Landing) systems represent a valuable class of flying robots thanks to their small area monitoring and building exploration.

We are studying the behavior of the quadrotor. This flying robot presents the main advantage of having quite simple dynamic features. Indeed, the quadrotor is a small vehicle with four propellers placed around a main body. The main body includes power source, sensors and control hardware. The four rotors are used to controlling the vehicle. The rotational speeds of the four rotors are independent. Thanks to this independence, it's possible to control the pitch, roll and yaw attitude of the vehicle. Then, its displacement is produced by the total thrust of the four rotors whose direction varies according to the attitude of the quadrotor. The vehicle motion can thus be controlled. However, a closed loop control system is required to achieve stability and autonomy.

The aim of this project is to control the position and the yaw, roll and pitch angles of quadcopter using PID (proportional-integral-derivative) Simulink model developed with linking it with a visual simulator Flight Gear simulator.

The notations being used:

- p_n = the inertial (north) position of the quadrotor along \hat{i}^i in \mathcal{F}^i ,
- p_e = the inertial (east) position of the quadrotor along \hat{j}^i in \mathcal{F}^i ,
- h = the altitude of the aircraft measured along $-\hat{k}^i$ in \mathcal{F}^i ,
- u = the body frame velocity measured along \hat{i}^b in \mathcal{F}^b ,
- v = the body frame velocity measured along \hat{j}^b in \mathcal{F}^b ,
- w = the body frame velocity measured along \hat{k}^b in \mathcal{F}^b ,
- ϕ = the roll angle defined with respect to \mathcal{F}^{v2} ,
- θ = the pitch angle defined with respect to \mathcal{F}^{v1} ,
- ψ = the yaw angle defined with respect to \mathcal{F}^v ,
- p = the roll rate measured along \hat{i}^b in \mathcal{F}^b ,
- q = the pitch rate measured along \hat{j}^b in \mathcal{F}^b ,
- r = the yaw rate measured along \hat{k}^b in \mathcal{F}^b .

2 Equations of Motion:

After getting familiar with the pervious notations, it is clear there are different frame of reference. We simply can divide them into body frame of reference and inertial frame of reference. One example from these transformations the attached image. As here we are transforming from the angular rates on the body frame to the angular rate but in the inertial frame.

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

First of all, we are considering the translational part of the motion following Newton Second law

$$m \frac{d\mathbf{v}}{dt_i} = \mathbf{f},$$

where m is the mass of the quadrotor, f is the total applied to the quadrotor, and the time derivative in the inertial frame.

Using Coriolis effect as the body frame might have a motion we reached to this form.

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{m} \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix},$$

Which after transformation changes to this following equation

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \begin{pmatrix} -g \sin \theta \\ g \cos \theta \sin \phi \\ g \cos \theta \cos \phi \end{pmatrix} + \frac{1}{m} \begin{pmatrix} 0 \\ 0 \\ -F \end{pmatrix},$$

To finishing the translational motion. We need to connect the translational motion of the body frame and on the inertial frame as this.

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = \begin{pmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ s\theta & -s\phi c\theta & -c\phi c\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

For the Rotational motion. Newton's second law stated, where \mathbf{h} is the angular momentum and \mathbf{m} is the applied torque.

$$\frac{d\mathbf{h}^b}{dt_i} = \mathbf{m},$$

Putting Coriolis Effect into consideration, we reached to this form

$$\frac{d\mathbf{h}}{dt_i} = \frac{d\mathbf{h}}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{h} = \mathbf{m}.$$

Also we can define that $\mathbf{h}^b = \mathbf{J}\boldsymbol{\omega}_{b/i}^b$ in the body coordinates.

Simply, the inertia matrix is defined by

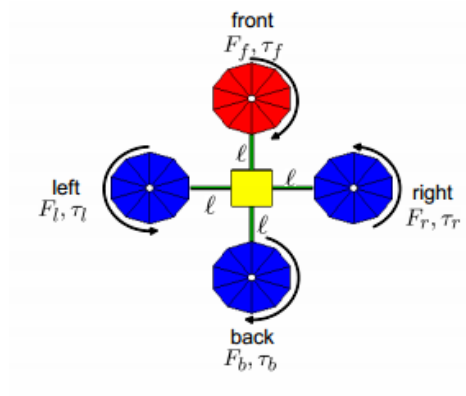
$$\mathbf{J} = \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix}.$$

So, From the Newton's second law for the rotational motion we simply can reach and also we should transform the Body motion into the inertial frame axes:

$$I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (I\boldsymbol{\omega}) = \boldsymbol{\tau}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} + \begin{pmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} \right]$$

Defining Forces and moments:



The total force acting of the quadcopter is

$$F = F_f + F_r + F_b + F_l.$$

The rolling torque is produced by the forces of the right and left motors as

$$\tau_\phi = \ell(F_l - F_r).$$

Similarly, the pitching torque is produced by the forces of the front and back motors as

$$\tau_\theta = \ell(F_f - F_b).$$

Due to Newton's third law, the drag of the propellers produces a yawing torque on the body of the quadrotor. The direction of the torque will be in the apposite direction of the motion of the propeller. Therefore, the total yawing torque is given by

$$\tau_\psi = \tau_r + \tau_l - \tau_f - \tau_b.$$

So finally, the Rotational equations of motions in the inertial frame can be given by.

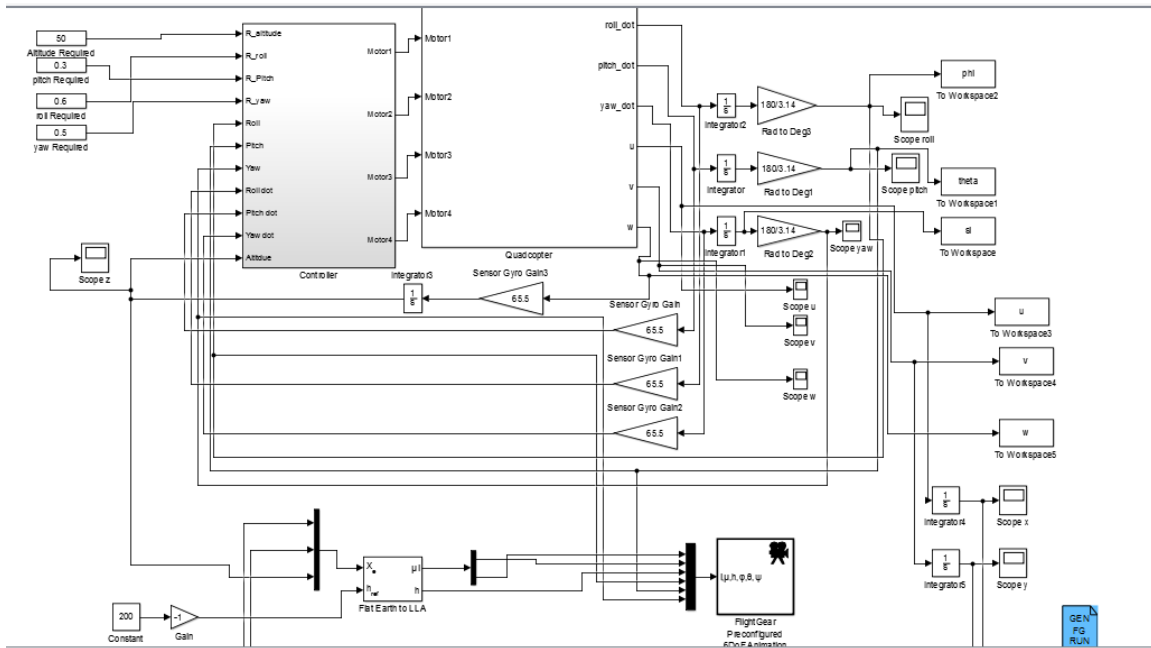
$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix},$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{pmatrix} + \begin{pmatrix} \frac{1}{J_x} \tau_\phi \\ \frac{1}{J_y} \tau_\theta \\ \frac{1}{J_z} \tau_\psi \end{pmatrix}.$$

3 Simulink Walkthrough

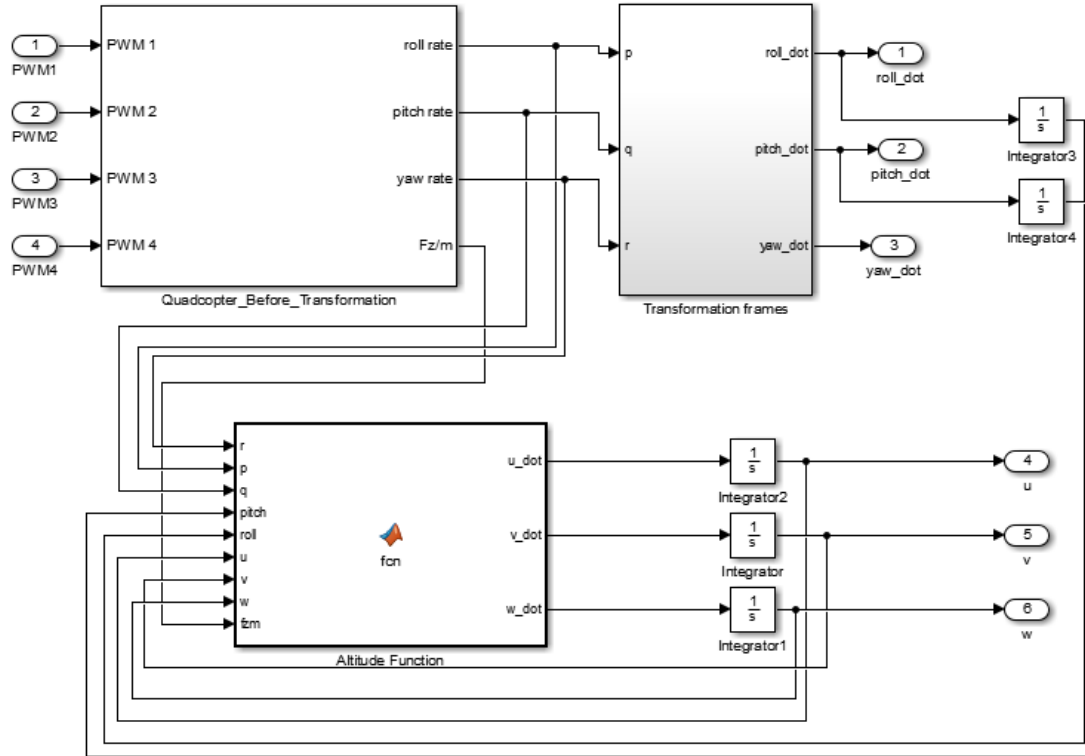
At first, we model the quadcopter equations of motion in the quadcopter block which receives the motor pulse width modulations as inputs and gives the derivatives of the states ($\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}$). From those outputs, we get the states (z, ϕ, θ, ψ).

The reference altitude and pitch, roll, and yaw angles are given as inputs. We control ($z, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$) in the controller block and get the pulse width modulations of the motor to be once again input in the quadcopter block and so on.



3.1 Quadcopter Block

In this block, we need to transform the pulse width modulations of the four motors to $(u, v, w, \dot{\phi}, \dot{\theta}, \dot{\psi})$. While (p, q, r) are roll rate, pitch rate and yaw rate which we get from the body frame directly from motors, so we need to transform them to the inertial frame.

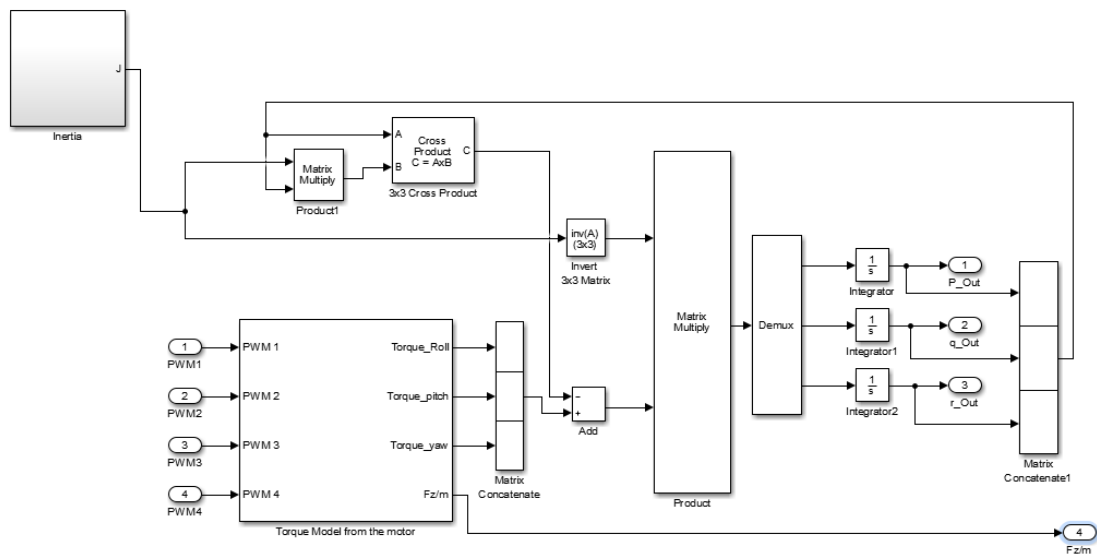


3.1.1 Quadcopter before transformation without assumption block

We need to get $(p, q, r, \frac{F_z}{m})$ from the pulse width modulations of the motors using this equation:

$$I\dot{\omega} + \omega \times (I\omega) = \tau$$

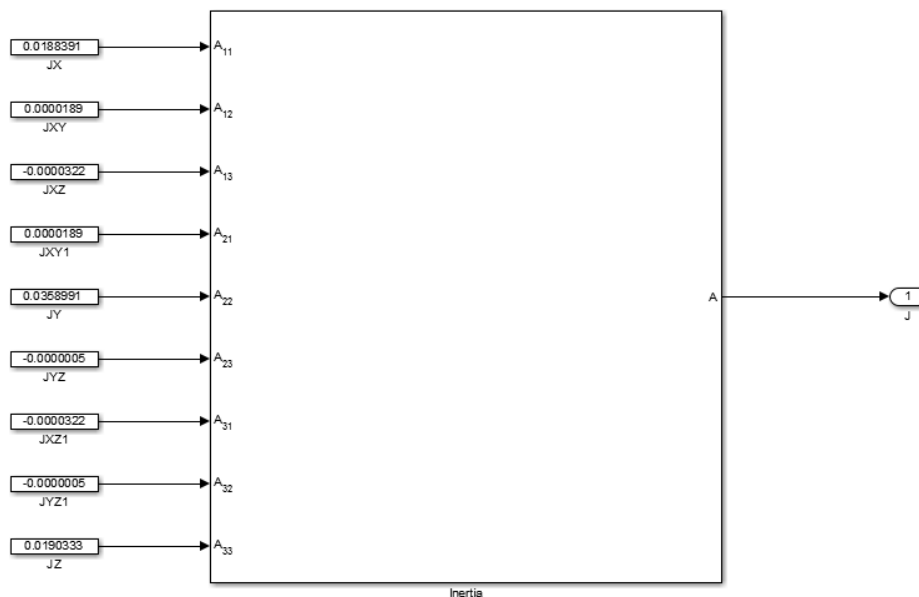
$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} + \begin{pmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} \right]$$



3.1.1.1 Inertia

We assume that the quadcopter is symmetric about all three axes, then $J_{xy}=J_{xz}=J_{zy}=0$. Which shows that:

$$\mathbf{J} = \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix}$$



The values of J_{xy}, J_{xz}, J_{zy} aren't precisely zeros as the Simulink does a matrix inversion, so to overcome the singular matrix we put it with these small values.

3.1.1.2 Torque motor from the rotor

We need to transform the pulse width modulations of the motor to $(\tau_\phi, \tau_\theta, \tau_\psi, \frac{F_z}{m})$, where:

$$F = F_f + F_r + F_b + F_l.$$

$$\tau_\phi = \ell(F_l - F_r).$$

$$\tau_\theta = \ell(F_f - F_b).$$

$$\tau_\psi = \tau_r + \tau_l - \tau_f - \tau_b.$$

Where

$$F_* = k_1 \delta_*$$

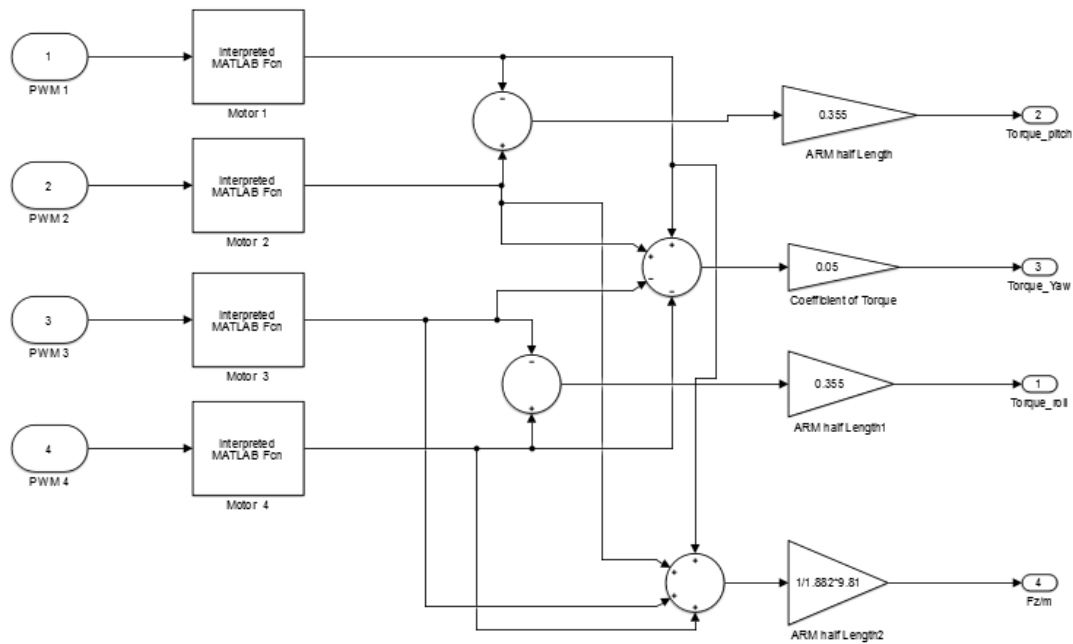
$$\tau_* = k_2 \delta_*,$$

In which k_1 and k_2 are constants that need to be determined experimentally and δ_* is the motor command signal.

In our model: $k_2=0.05$, $Lk_1=0.355$

Then using this equation:

$$\begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} k_1 & k_1 & k_1 & k_1 \\ 0 & -\ell k_1 & 0 & \ell k_1 \\ \ell k_1 & 0 & -\ell k_1 & 0 \\ -k_2 & k_2 & -k_2 & k_2 \end{pmatrix} \begin{pmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{pmatrix}$$



Where

```

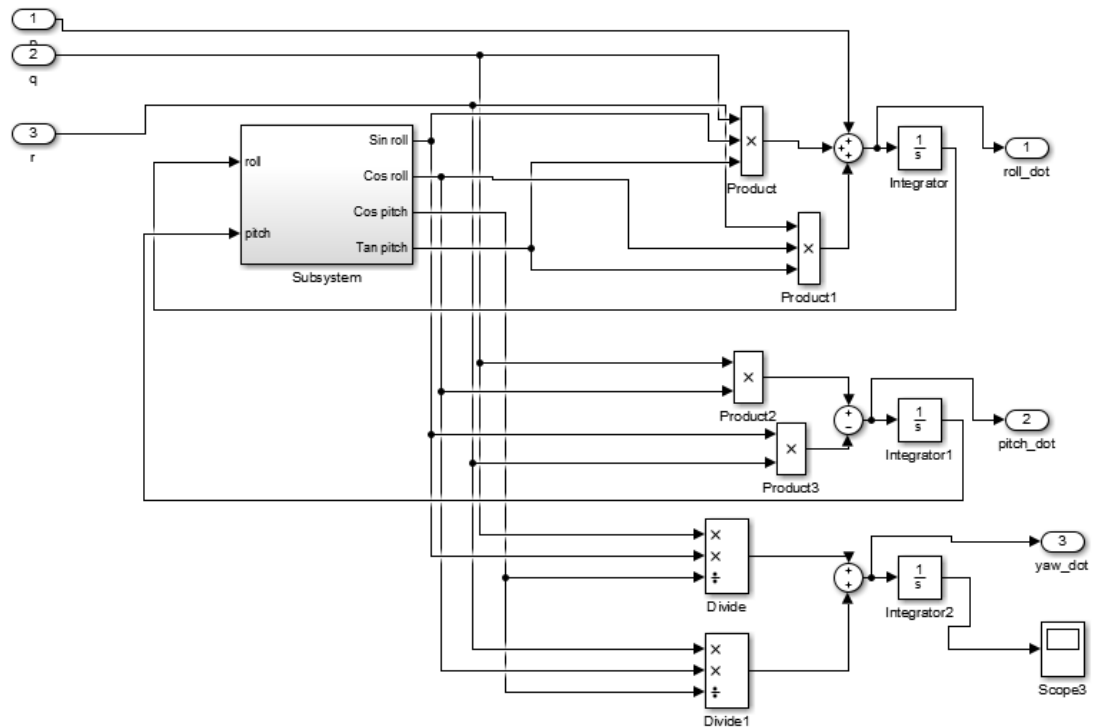
Motor.m
1 function Weight = Motor(PWM)
2
3     load Data.mat;
4
5     Weight = fittedmodel(PWM)*9.81*0.001;

```

3.1.2 Transformation Frames

From p, q, r we can get $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ through this equation:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$



3.1.3 Altitude Function

Through the equations of translational motion,

$$m \frac{d\mathbf{v}}{dt_i} = m \left(\frac{d\mathbf{v}}{dt_b} + \boldsymbol{\omega}_{b/i} \times \mathbf{v} \right) = \mathbf{f},$$

we can get:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \begin{pmatrix} -g \sin \theta \\ g \cos \theta \sin \phi \\ g \cos \theta \cos \phi \end{pmatrix} + \frac{1}{m} \begin{pmatrix} 0 \\ 0 \\ -F \end{pmatrix}$$

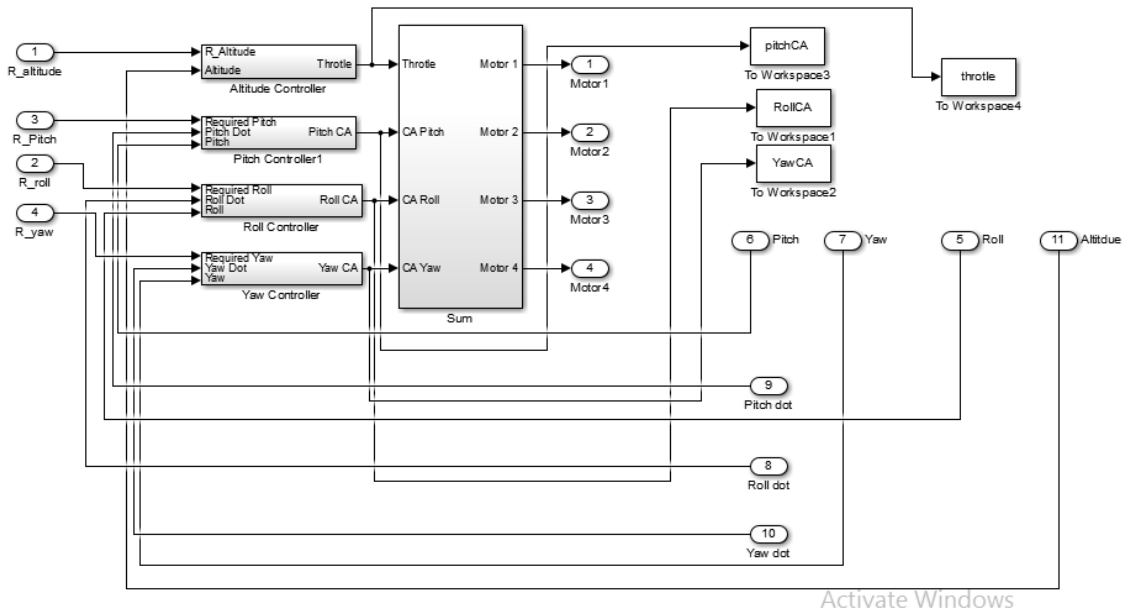
```

Quadcopter/Altitude Function  x  +
1  function [u_dot,v_dot,w_dot] = fcn(r,p,q,pitch,roll,u,v,w,fzm)
2  %#codegen
3  u_dot=(r*v-q*w)-9.81*sin(pitch)+0;
4  v_dot=(p*w-r*u)+9.81*cos(pitch)*sin(roll)+0;
5  w_dot=(q*u-p*v)+9.81*cos(pitch)*cos(roll)-fzm;
6

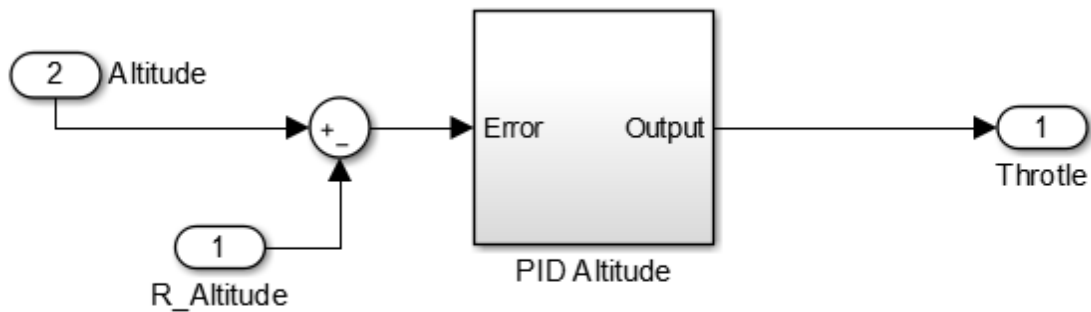
```

3.2 Controller Block

In this block, we compare the states (z , ϕ , θ , ψ) to the reference inputs of those states. We control those errors, and then get the pwms of the motors as outputs. In order to preserve more control, we are controlling the angles and its rates.

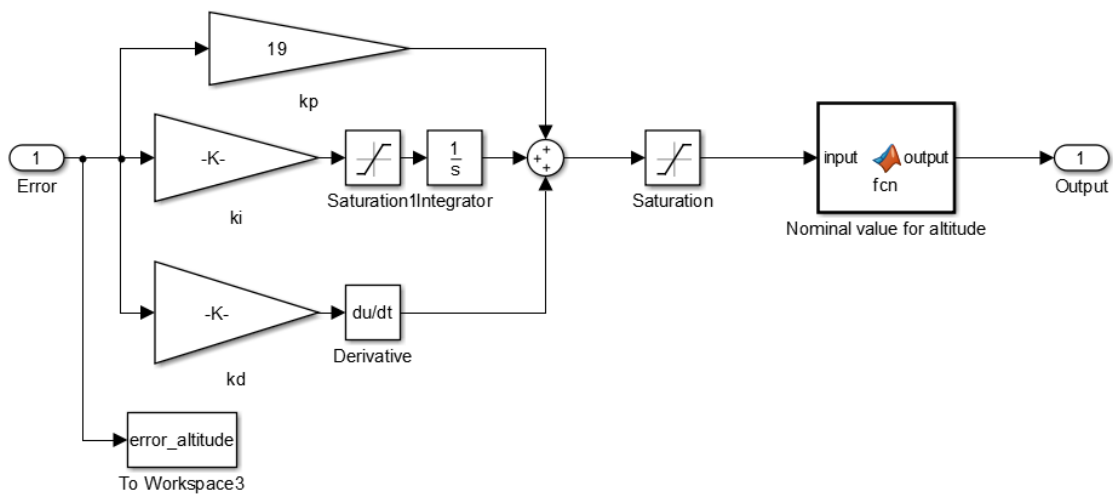


3.2.1 Altitude Controller



3.2.1.1 PID Altitude block:

$K_p = 19$; $K_i = 0.1/0.012 \cdot 0.15$; $K_d = 5.99 \cdot 0.012/0.15$;

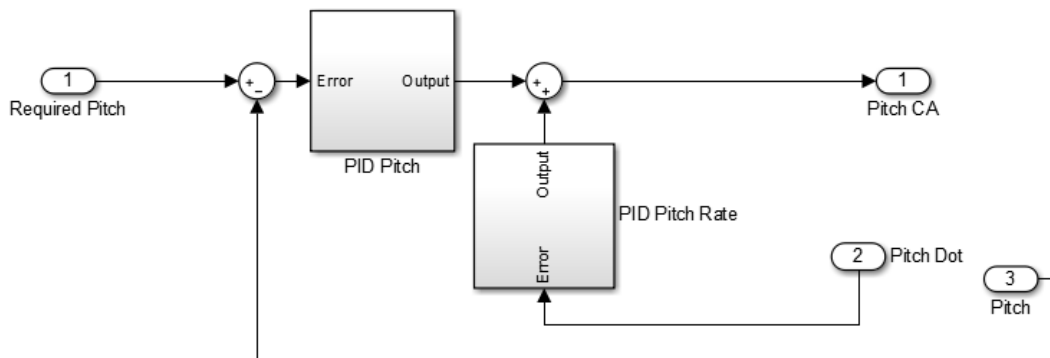


```

Controller/Altitude Controller/PID Altitude/Nominal value for altitude*
1 function output = fcn(input)
2
3 output=1000+33*input;
4

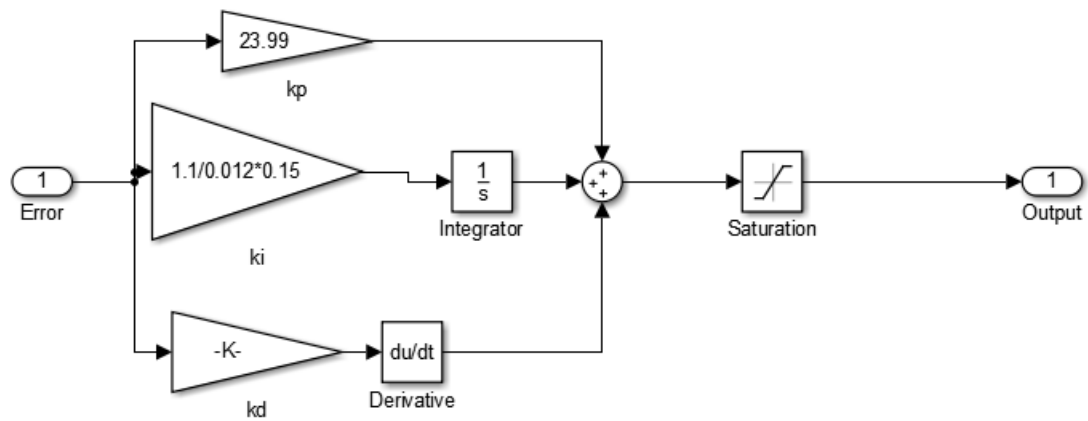
```

3.2.2 Pitch Controller



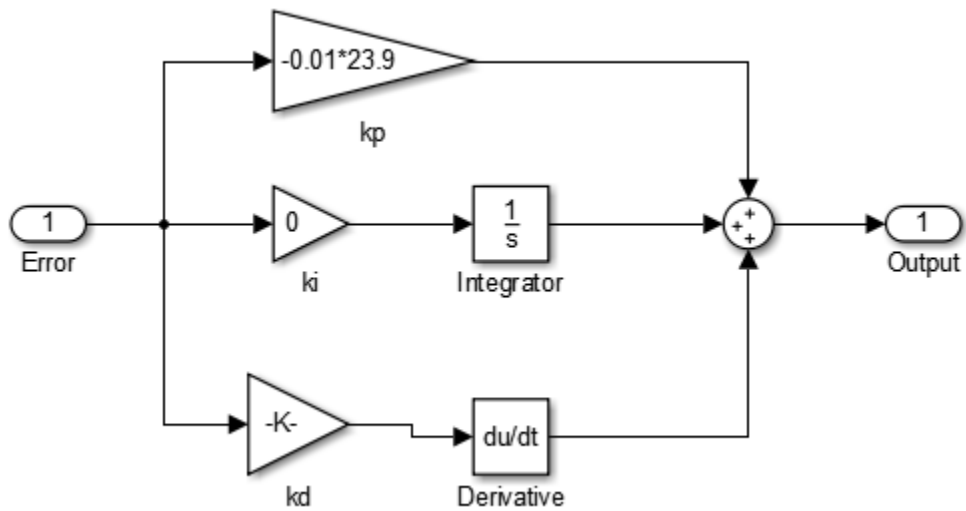
3.2.2.1 PID Pitch block:

$K_p = 23.99$, $K_d = 99.99 \cdot .012 / .15$, $K_i = 1.1 / .012 \cdot .15$

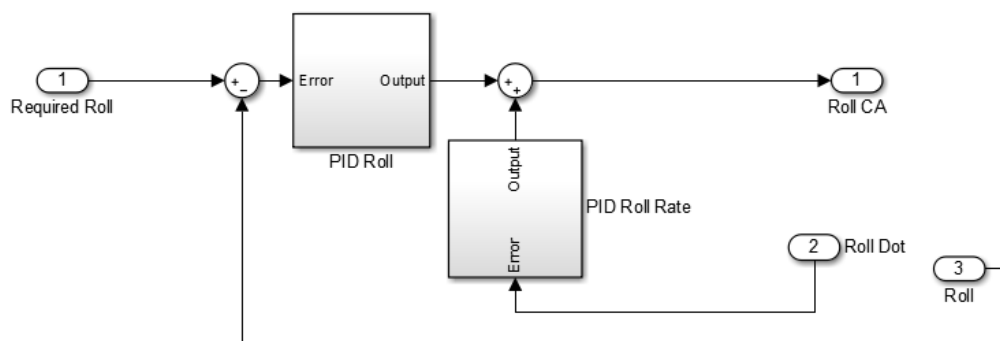


3.2.2.2 PID Pitch Rate block:

$K_p = -0.01 \cdot 23.9$, $K_i = 0$, $K_d = -89.99 \cdot 0.01 \cdot 0.012 / 0.15$

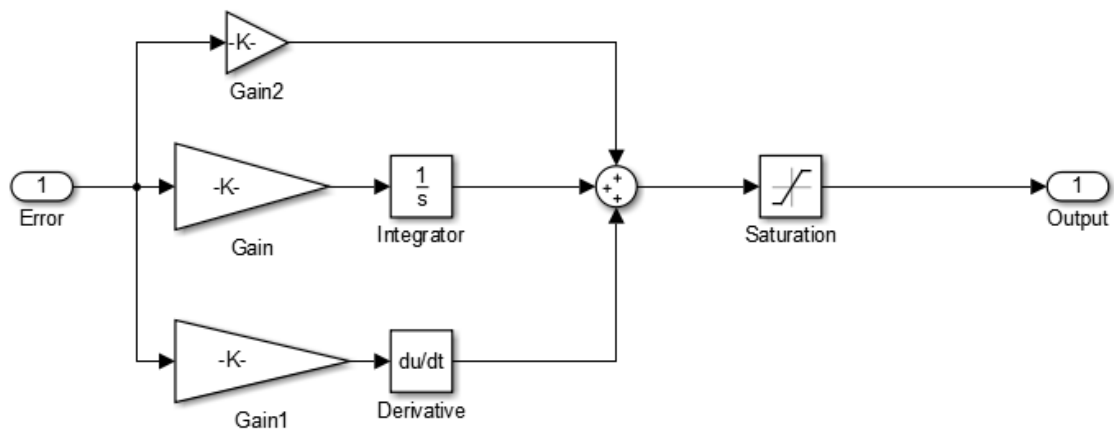


3.2.3 Roll Controller

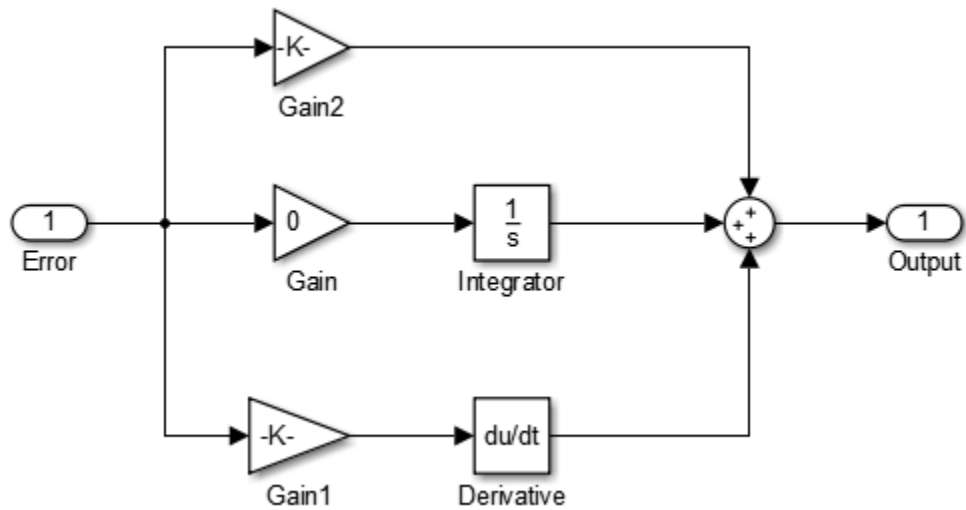


3.2.3.1 PID Roll block:

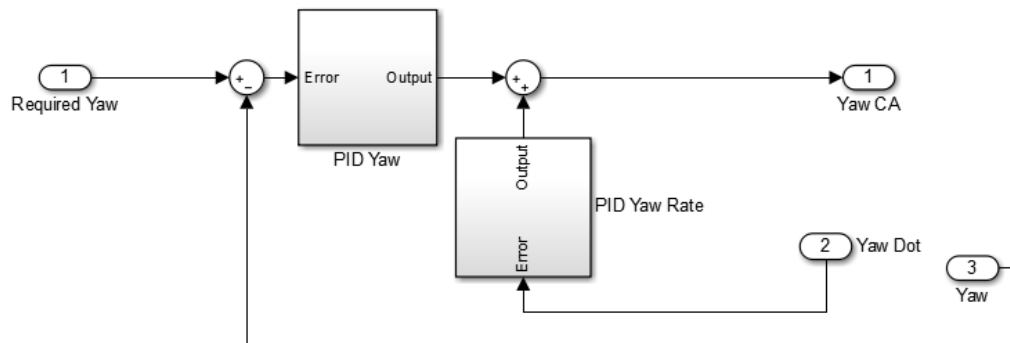
$K_p = 23.99$, $K_d = 99.99 * .012 / .15$, $K_i = 1.1 / .012 * .15$



3.2.3.2 PID Roll Rate block:

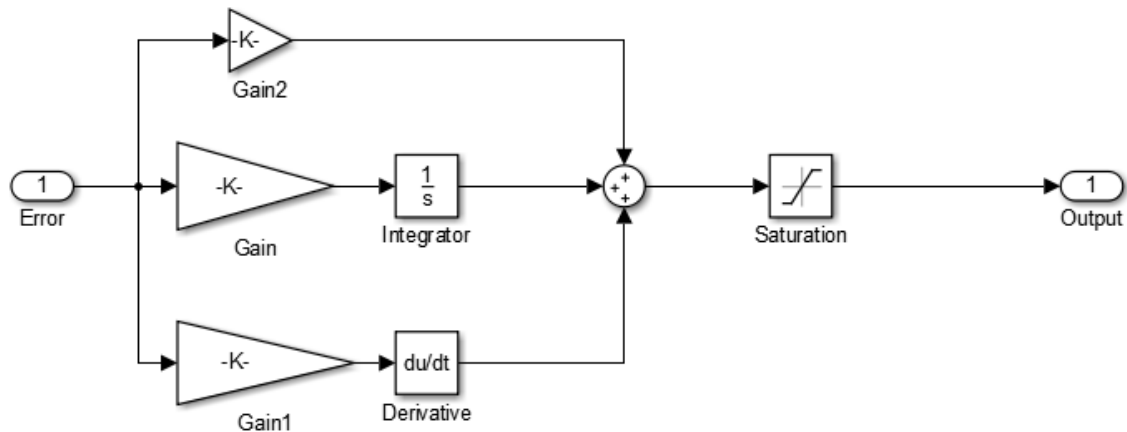


3.2.4 Yaw Controller



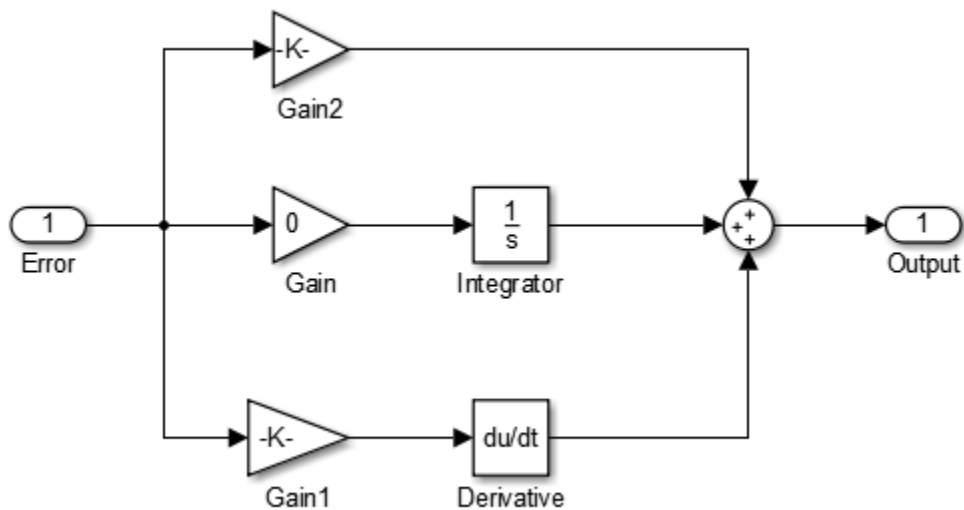
3.2.4.1 PID Yaw block:

$K_p = 23.99$, $K_d = 99.99 * .012 / .15$, $K_i = 1.1 / .012 * .15$



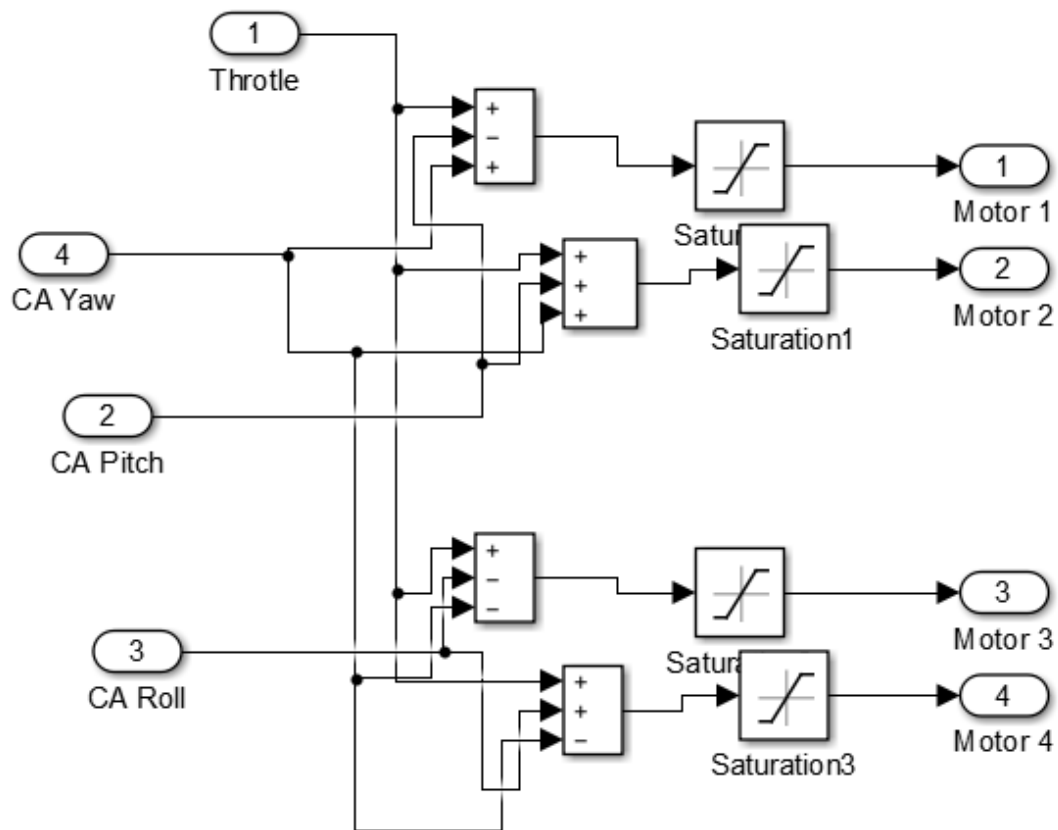
3.2.4.2 PID Yaw Rate block:

$K_p = 23.99$, $K_d = 99.99 * .012 / .15$, $K_i = 1.1 / .012 * .15$

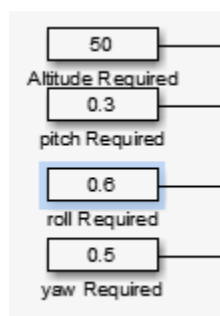


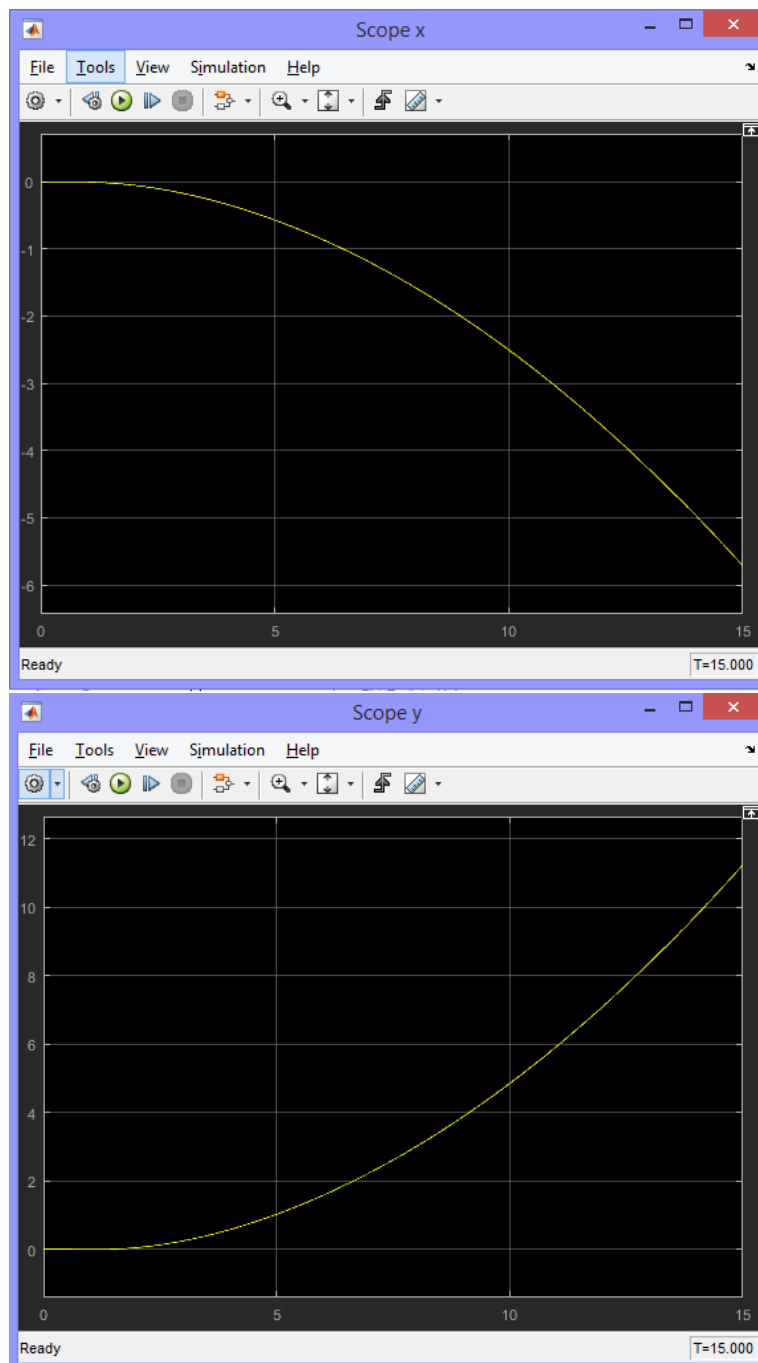
3.2.5 Sum Block

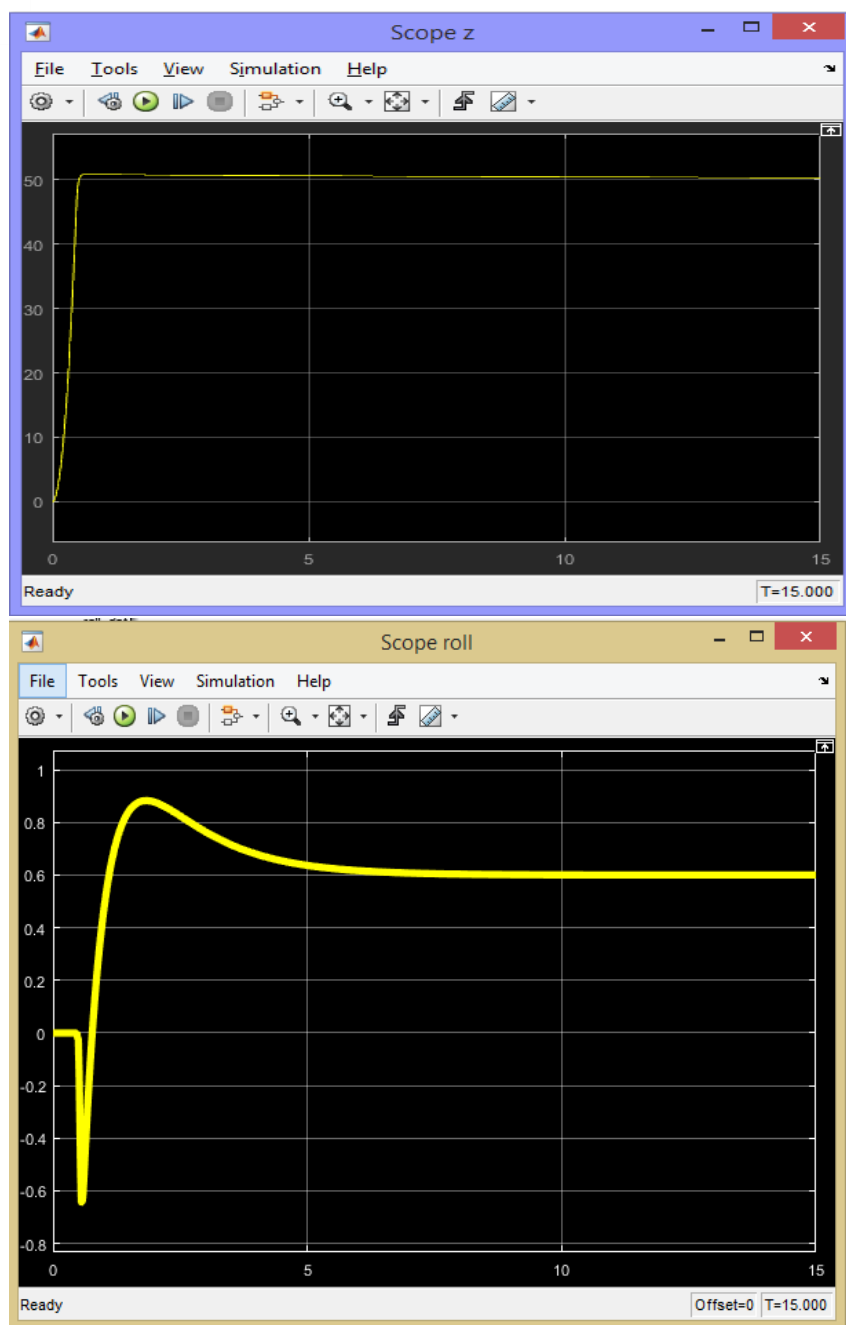
Here we are representing different mode in referring the responsible propellers. Like the thrust is always coming by adding the thrust from the four propellers. While pitching and rolling uses only two different propellers, unlike yawing which uses the hole propellers.

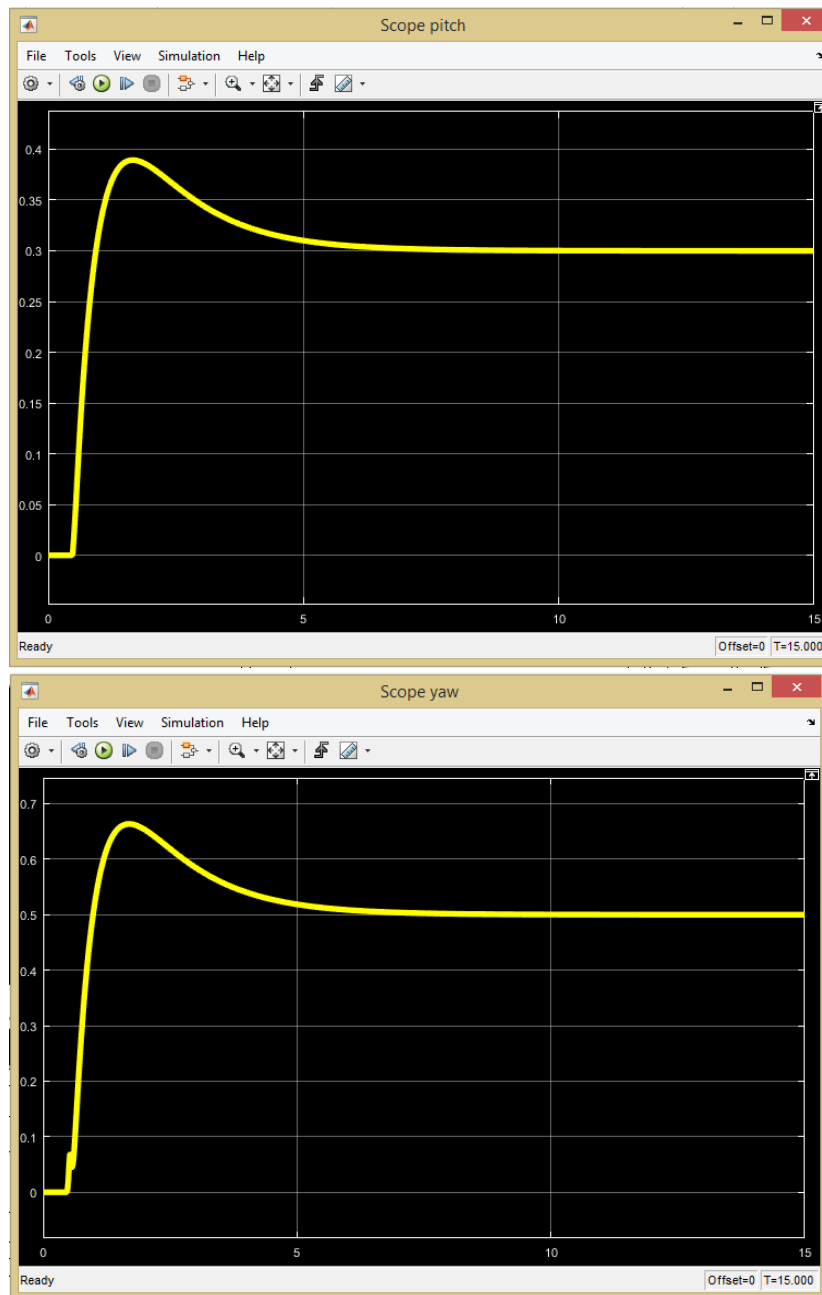


3.3 Results from Scopes









4 FlightGear Visualization

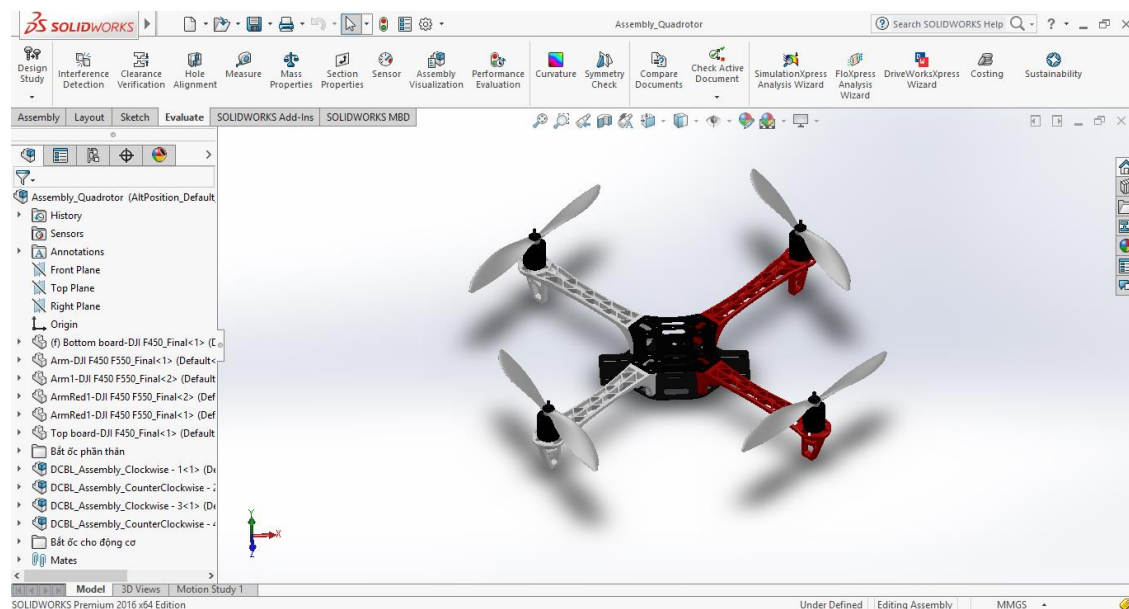
At first, we imported a quadcopter geometry to Flight Gear airplanes, and then we linked our Simulink with Flight Gear.

4.1 Importing Geometry

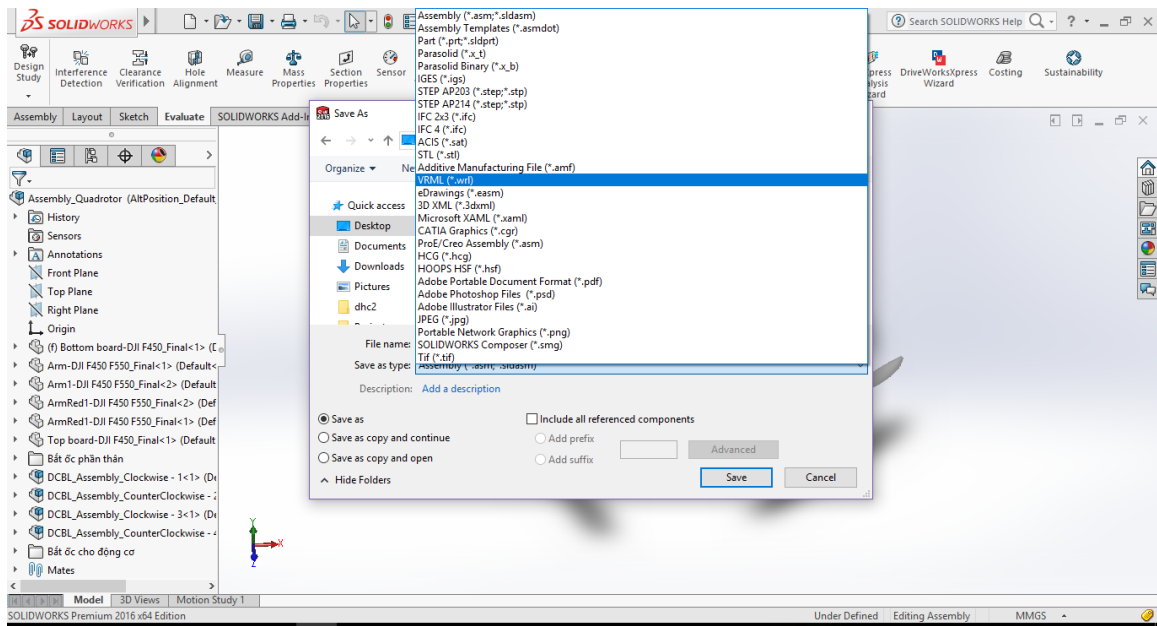
4.1.1 Saving Geometry to AC3D

Since our initial geometry file was in SolidWorks format, we had to change it into AC3D format for FlightGear to be able to simulate it.

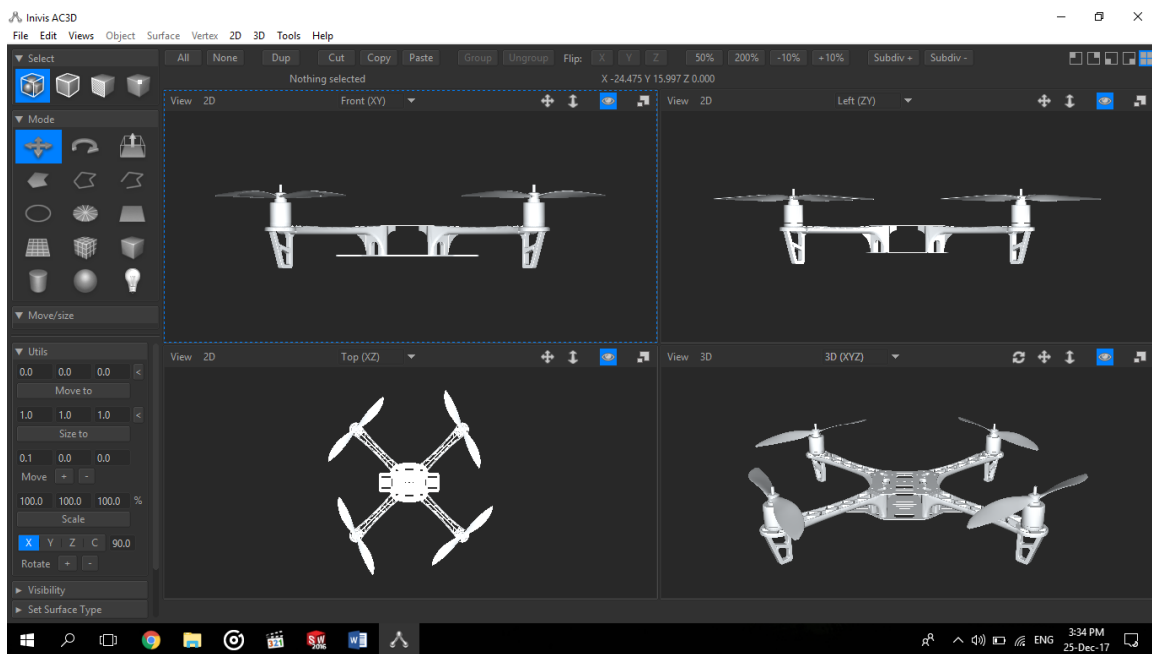
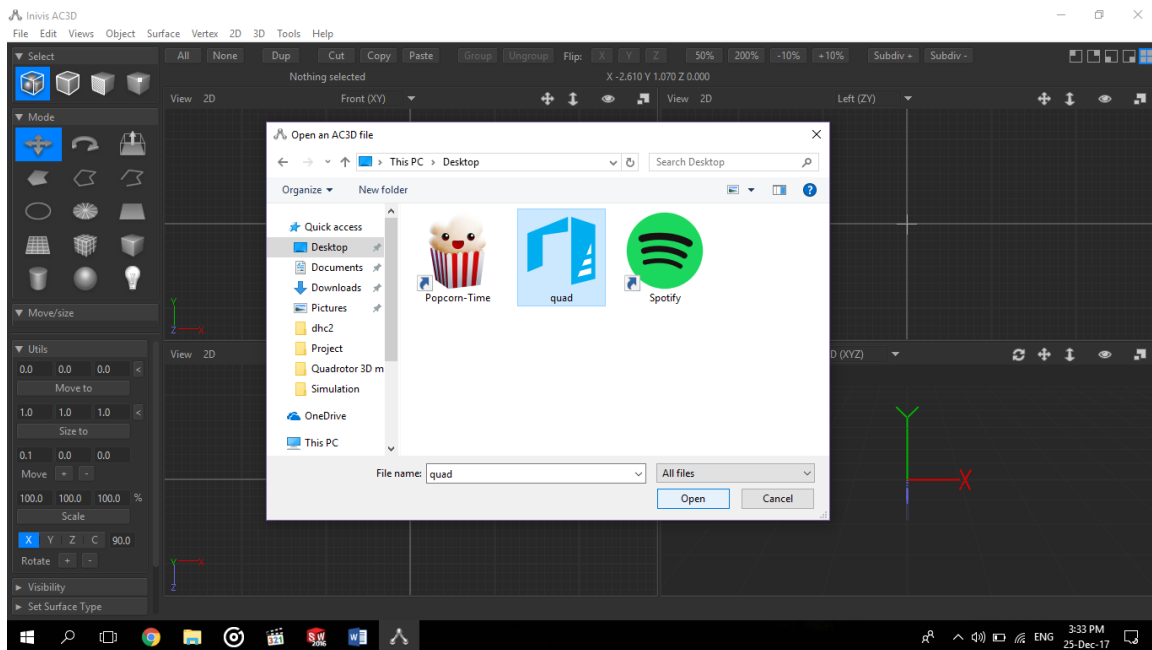
Initial geometry:



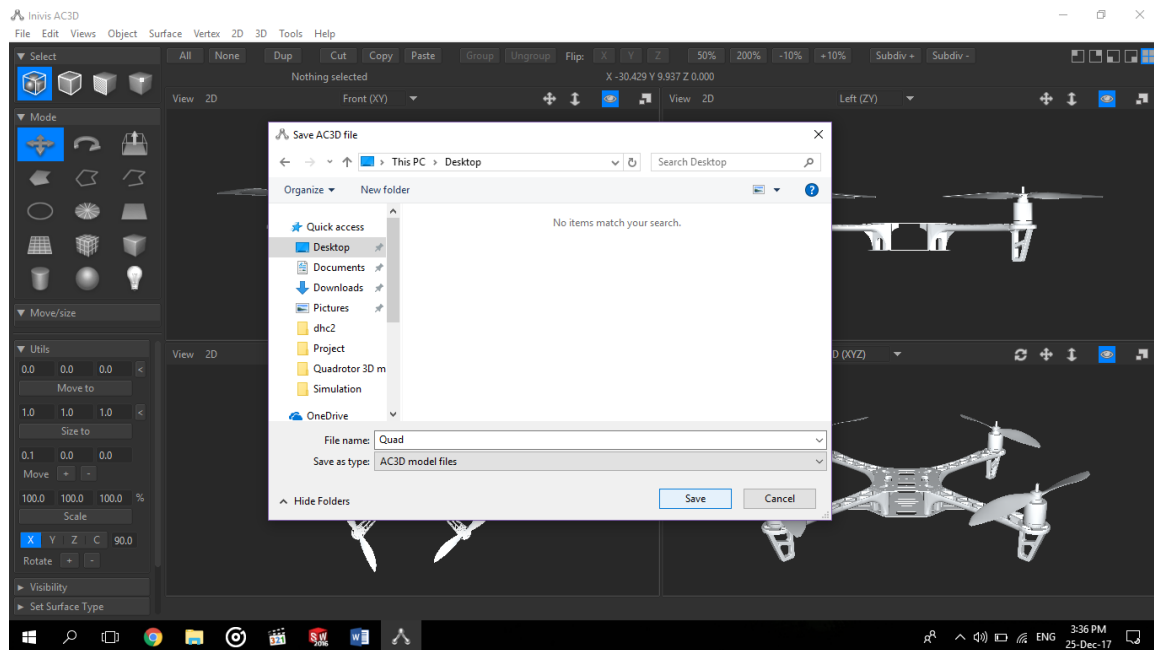
Change to VRML format



Open using AC3D



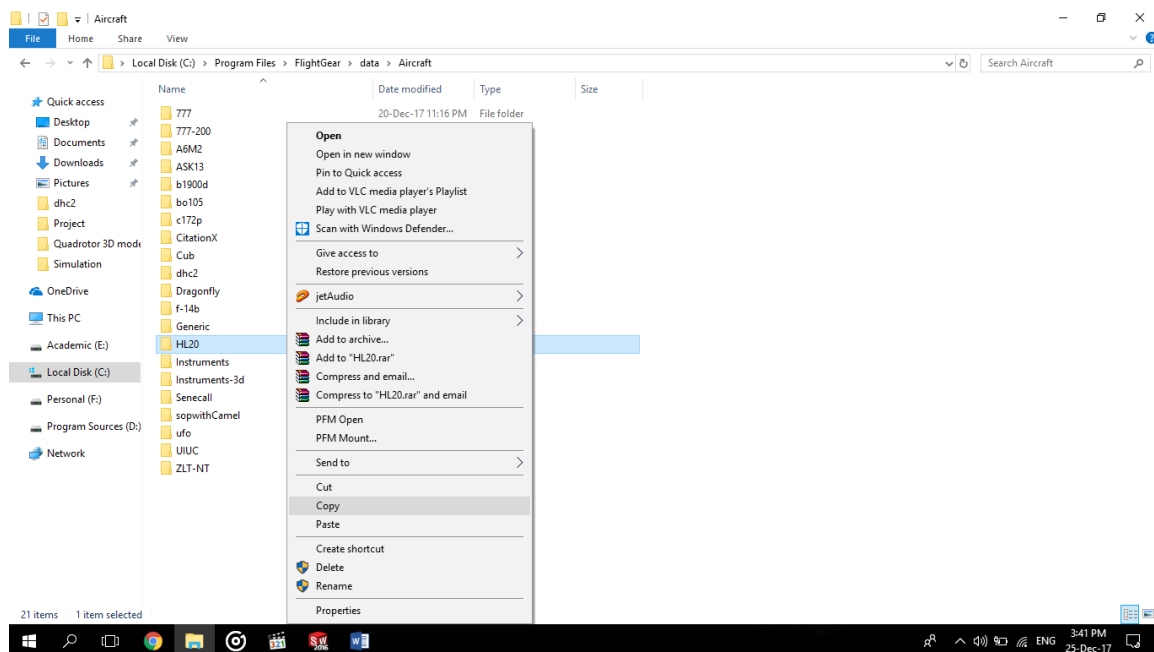
Saving to AC3D format



4.1.2 Importing to FlightGear

To place the geometry with settings that FlightGear can access to run the simulation, we had to use files available in an airplane that is already in FlightGear. We used files available for the HL20 and changes all directories to new folder we named “Quad”.

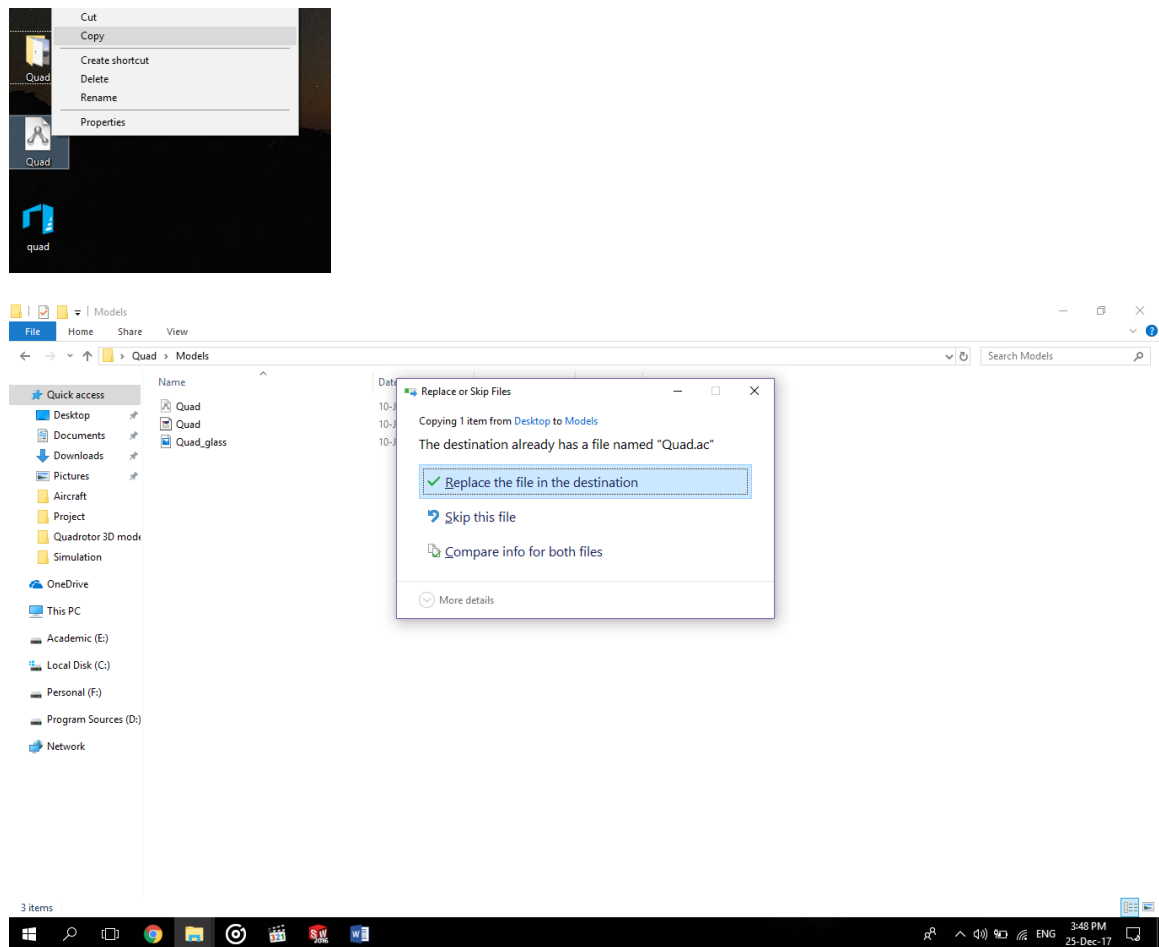
Copying HL20 files



Renaming all

Models	25-Dec-17 3:42 PM	File folder		Models	25-Dec-17 3:42 PM	File folder	
HL20-set	10-Jun-05 12:51 AM	XML File	1 KB	Quad-set	10-Jun-05 12:51 AM	XML File	1 KB
thumbnail	27-Mar-07 9:05 PM	JPG File	7 KB	thumbnail	27-Mar-07 9:05 PM	JPG File	7 KB
HL20	10-Jun-05 12:51 AM	AC3D geometry	80 KB	Quad	10-Jun-05 12:51 AM	AC3D geometry	80 KB
HL20	10-Jun-05 12:51 AM	XML File	5 KB	Quad	10-Jun-05 12:51 AM	XML File	5 KB
HL20_glass	10-Jun-05 12:51 AM	BMP File	193 KB	Quad_glass	10-Jun-05 12:51 AM	BMP File	193 KB

Copy created geometry file into models



Edit XML files

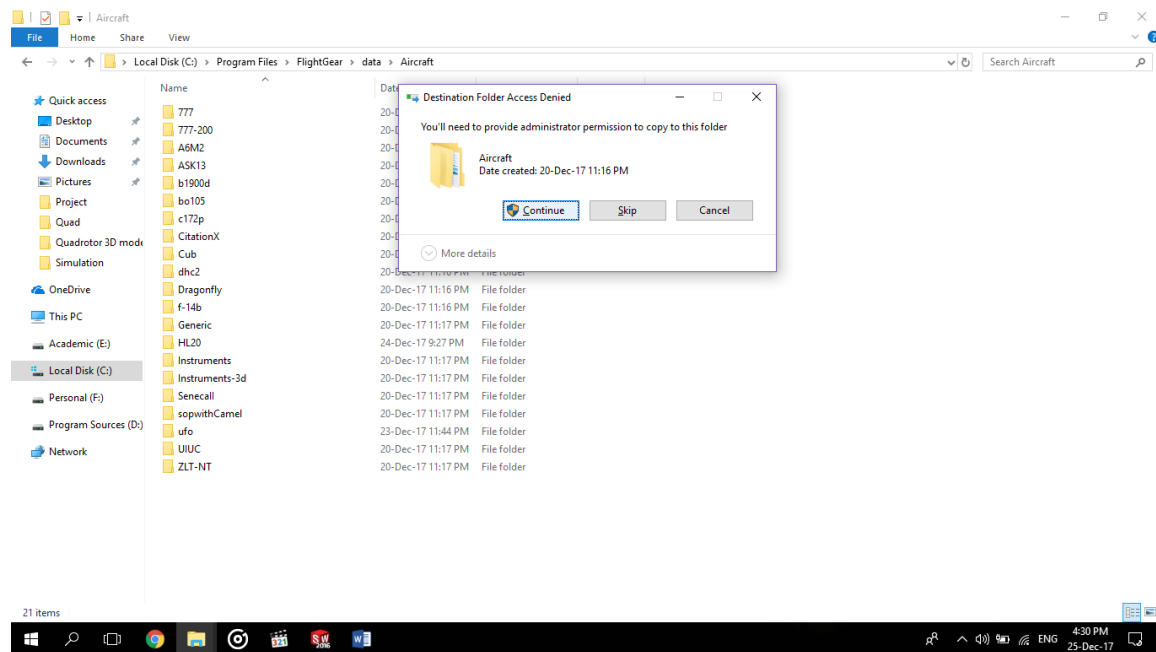
```






: sim>
<description>NASA:HL-20 Personnel Launch System</description>

<description>NASA:Quad Personnel Launch System</description>

```

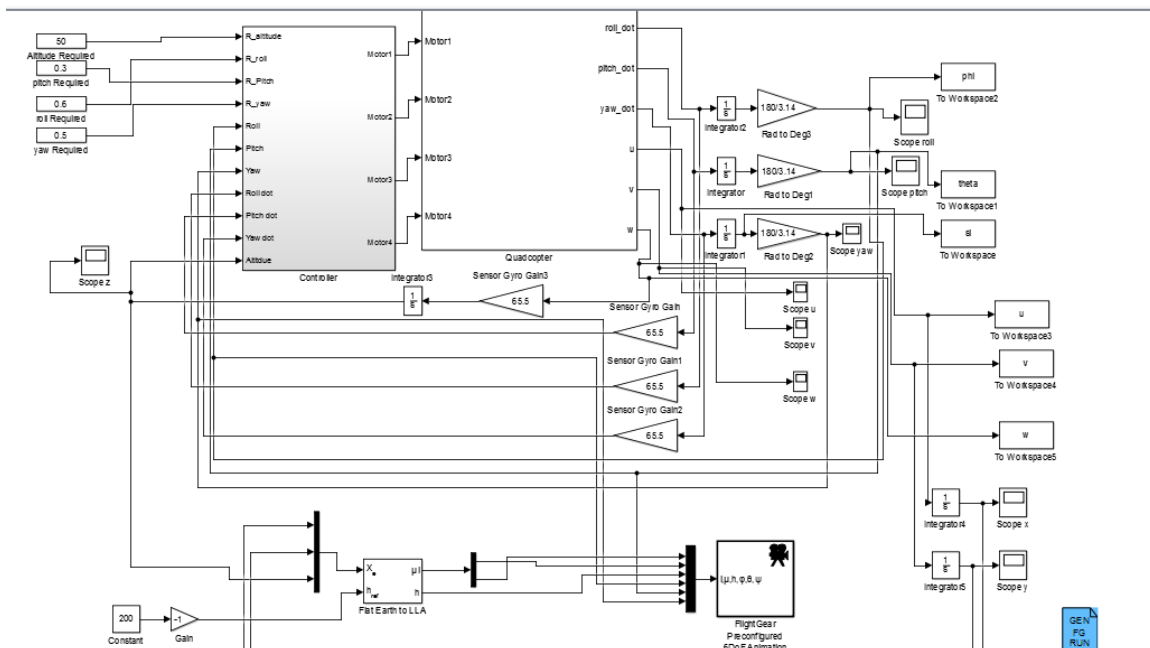
Copy finished folder to FlightGear Airplanes



 Instruments
 Instruments-3d
 Quad
 Senecall
 sopwithCamel

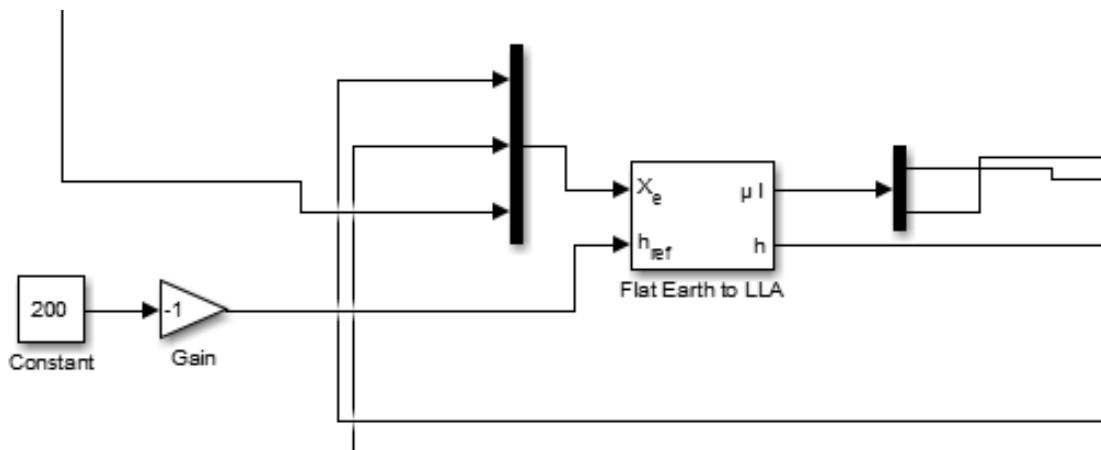
Now, the Quadcopter should be available for FlightGear to access through our Simulink model.

4.2 Linking Simulink to Flight Gear



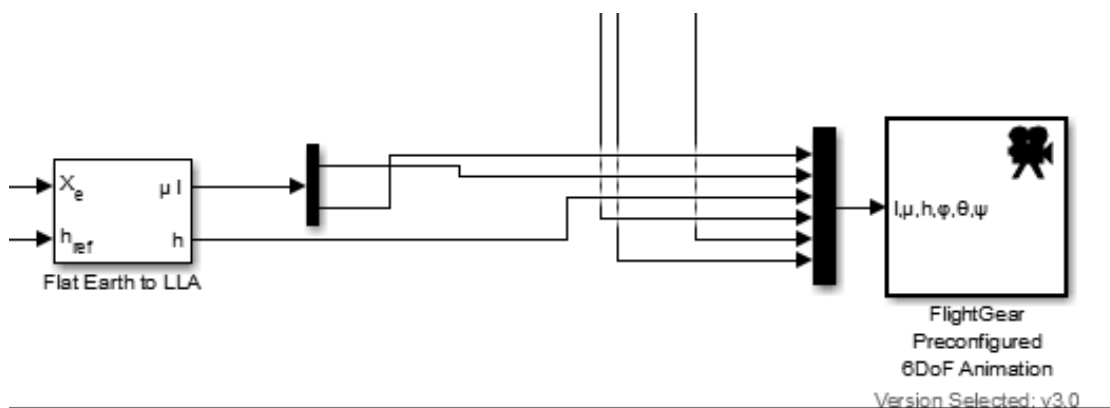
4.2.1 Flat Earth to LLA block

This block is to convert between x, y , and z of the quadcopter to latitude, longitude, and z . The input to X_e (port 1) is the (x, y, z) states, and to h_{ref} is negative the height that we want to start at.



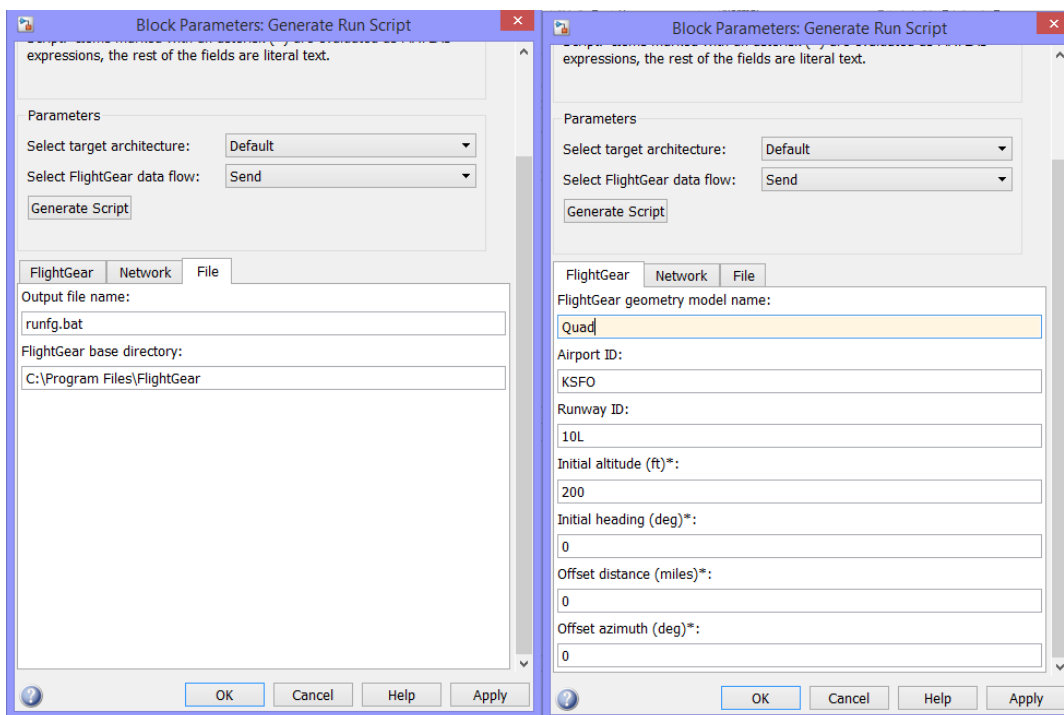
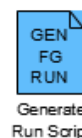
4.2.2 Flight Gear Preconfigured 6DOF

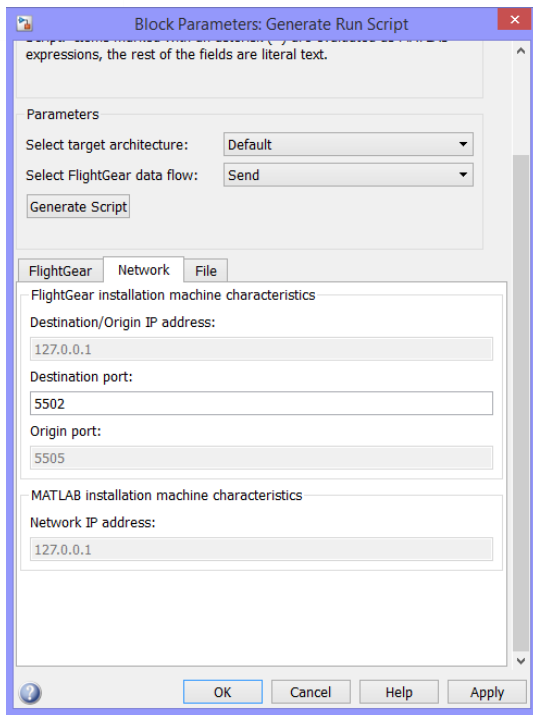
The inputs to this block are longitude, latitude and altitude from the Flat Earth to LLA block and the roll, pitch and yaw angles.



4.2.3 Generating .bat file

Using this block, we can generate the bat file which later should be run in Matlab to start flight gear with our choice of airplane,..etc.





The file will look something like this:

```
C:\Program Files\FlightGear
SET FG_ROOT=C:\Program Files\FlightGear\data
.\bin\win64\fgfs --aircraft=Quad --
fdm=network,localhost,5501,5502,5503 --fog-fastest --
disable-clouds --start-date-lat=2004:06:01:09:00:00 --
disable-sound --in-air --enable-freeze --airport=KSFO --
runway=10L --altitude=200 --heading=0 --offset-distance=0
--offset-azimuth=0
```

After setting those properties, we need to click “Generate Script” and save it in the same directory of our Simulink file.

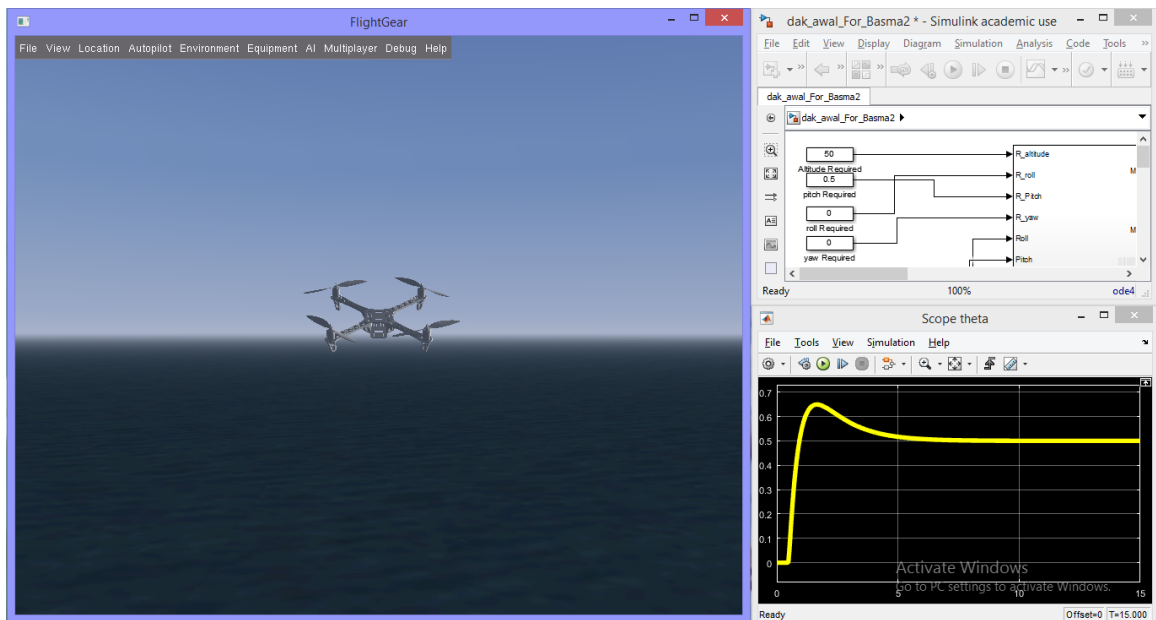
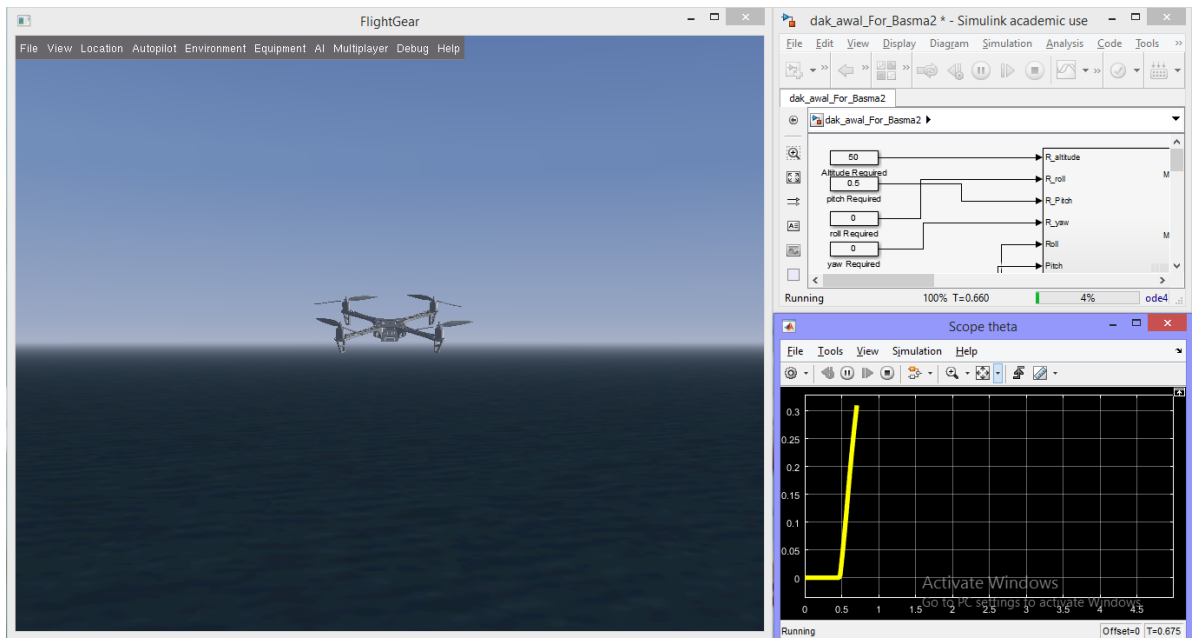
4.2.4 Simulation Run

In order to run the simulation, we need to first write this line in Matlab command window.

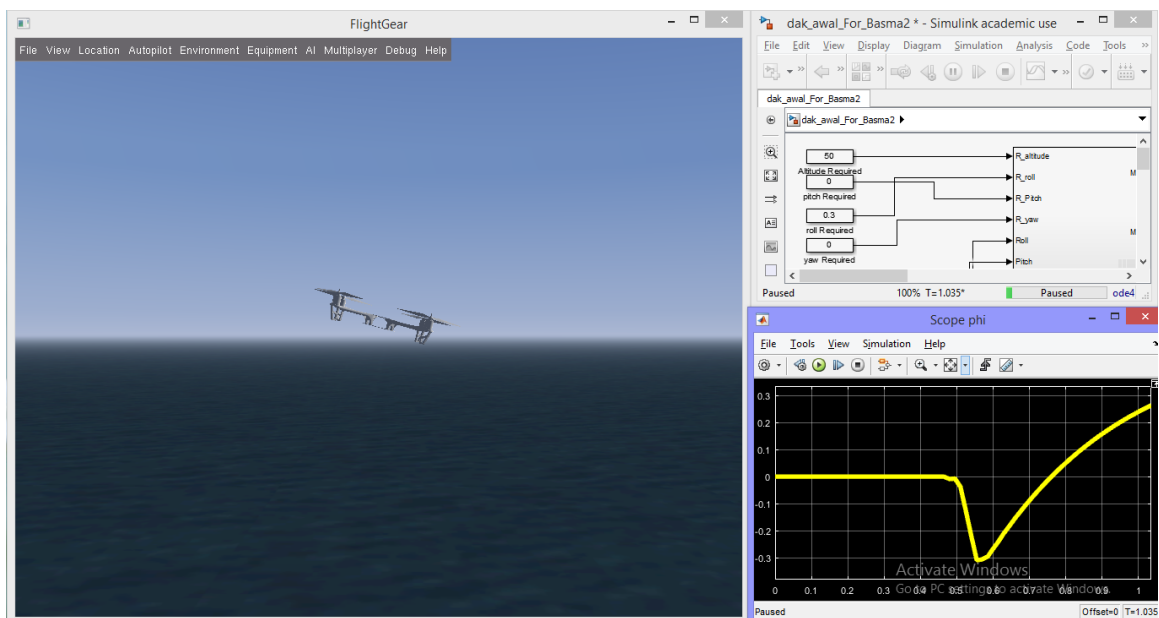
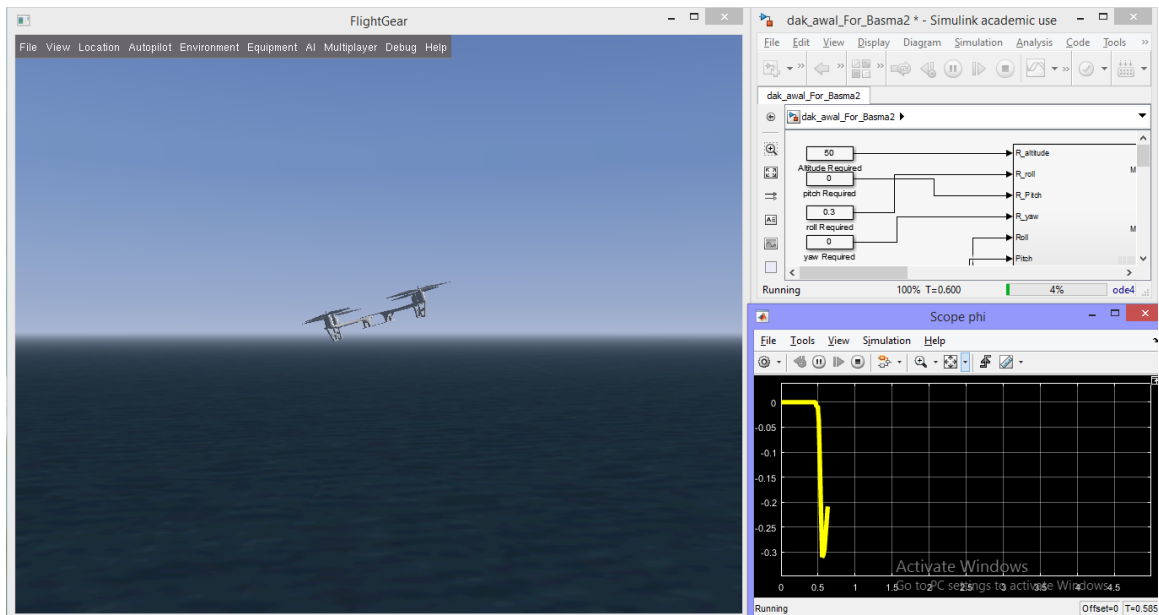
```
>> dos('runfg.bat')
```

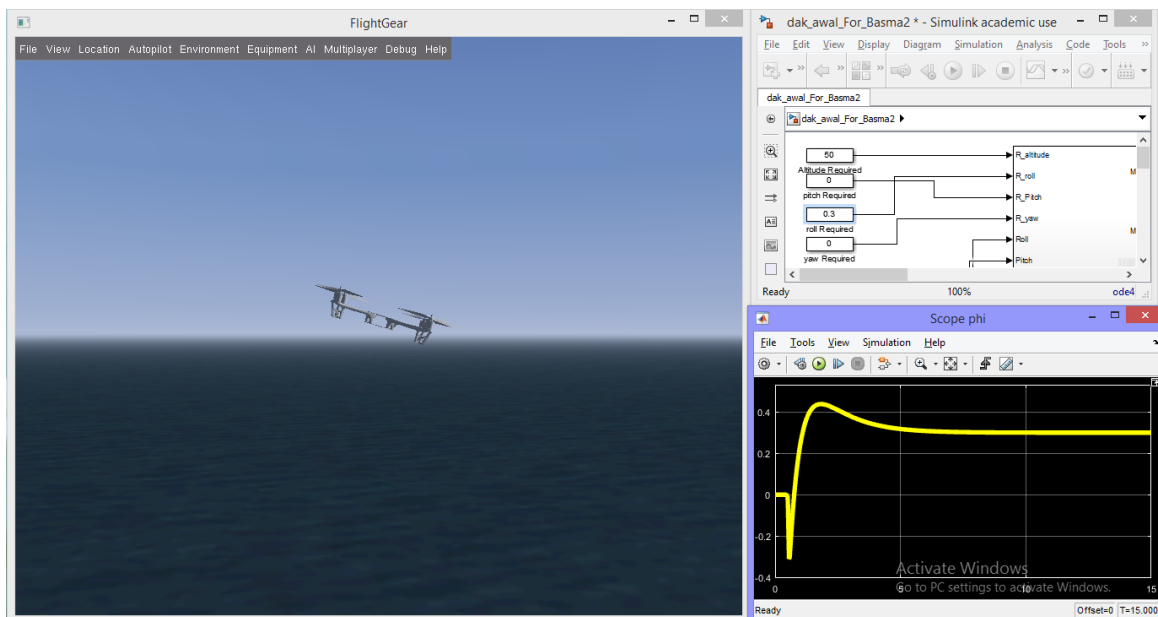
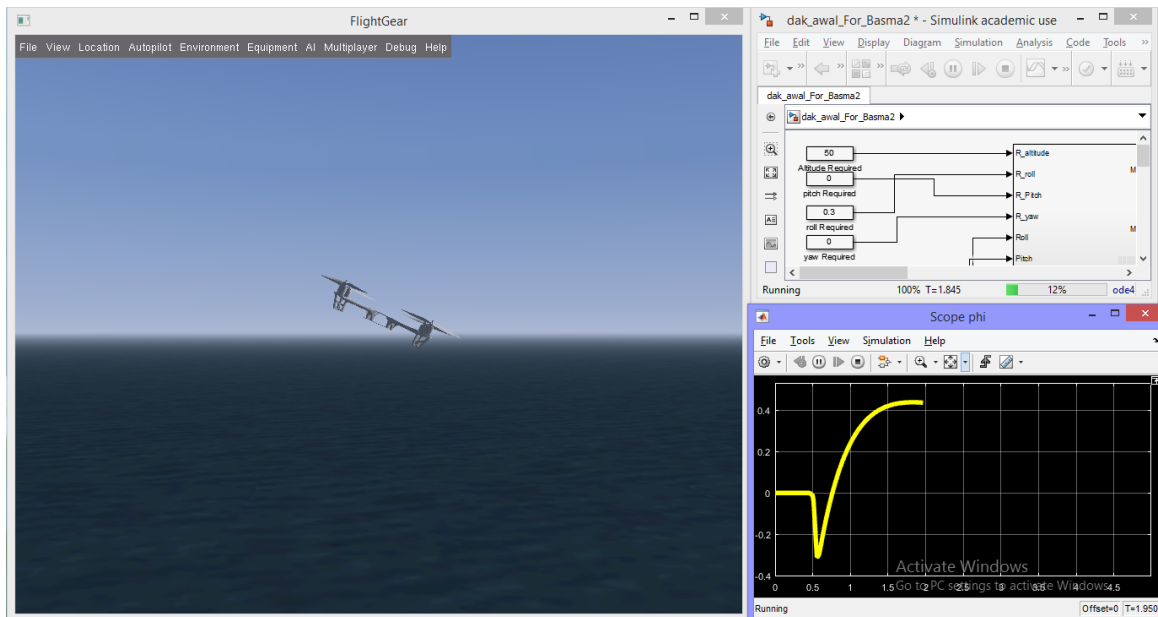
After Flight Gear loads, we can then run the Simulink and compare the graphs in the scopes to the motion of the plane.

Pitch

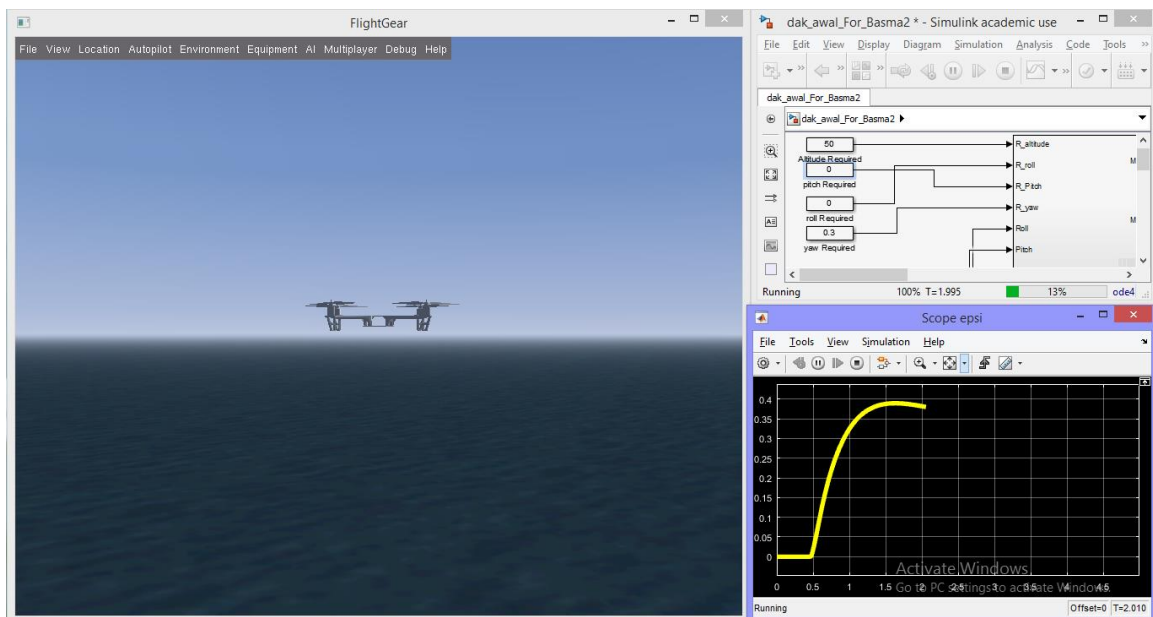
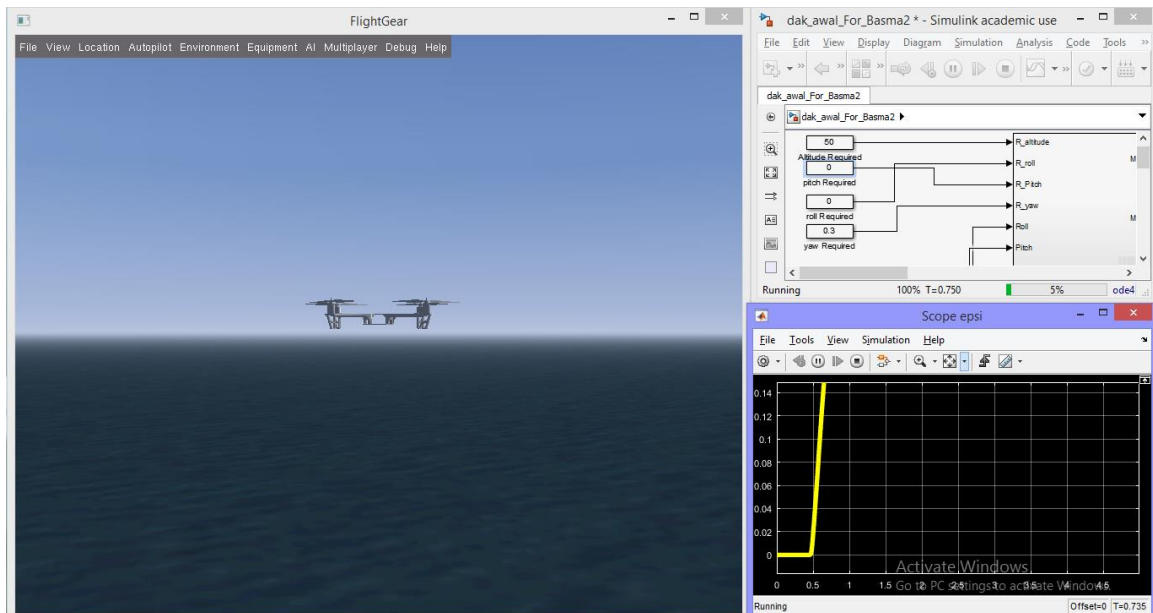


Roll





Yaw



5 Robust PID of the Quadcopter

One of the challenging we have is the problems during the automatic takeoff and landing. Because of disturbances like the wind we may face difficulties to control our model. To improve controller performances in take-off, hovering and trajectory tracking missions is to design a controller which is robust against these disturbances. Based on the time-scale separation approach, we could achieve the automatic take-off, hovering and trajectory tracking missions for a quadcopter. The designed controller consists of a position controller and an attitude controller. The position controller controls the desired pitch and roll angles based on the errors we get from our system which is related to the longitudinal and lateral positions. On the other hand, the attitude controller is designed to generate the desired tracking of the attitude.

