ME 425 HW 2

Demir Demirel 20586

***Abstract*— This report include implementation of dynamical model of quadrotor and apply hover control, trajectory tracking control. To achieve these we design various plants and controllers. Implementations were done by Simulink, MATLAB. Designed system consist of both Simulink blocks and MATLAB functions. The aim of this assignment is achieve vertical take-off and going to a certain point with calculated trajectory. In this assignment we ignore the effect of the disturbances in order to get less noisy data.**

I. INTRODUCTION

Vertical Take-off is a method that applied in most of the UAV's. This VTOL feature enable vehicles to take-off from very small areas instead long runways. In the first part of this report we will be talking about VTOL application on Quadrotors.
Control quadrotor to be follow certain trajectory is another part of this report. Which gives us the flexibility of going to o certain point without any human control over the quadrotor. For these reasons, the common thing that we need to achieve is quadrotor dynamics.

II. QUADROTOR DYNAMICS

In order to get Dynamic equations of the quadrotor we need to understand the Fig. II 1. As it is seen in the figure below our model has linear velocities $(\dot{X},\dot{Y},\dot{Z})$ and angular velocities(p,q,r) determined from the angles of ϕ,θ,ψ. The total thrust is represented as U_1 which is the thrust with respect to the quadrotor body frame. The movement of roll would be U_2, pitch would be U_3, yaw would be U_4 as seen in the Fig. II 2.

Fig. 1. Quadrotor Dynamics

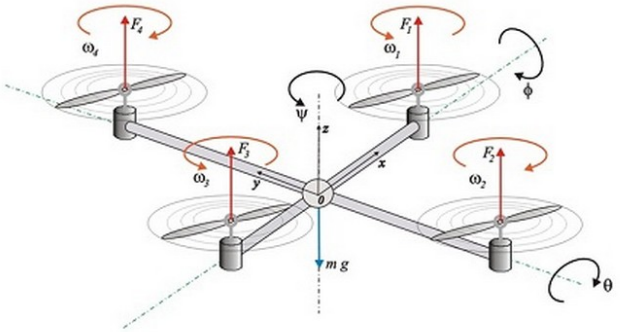
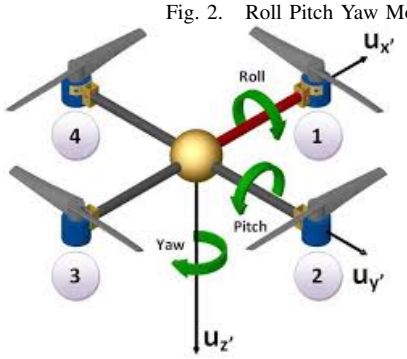


Fig. 2. Roll Pitch Yaw Movements



$$\begin{aligned}
\ddot{X} &= (sin\psi sin\theta + cos\psi sin\theta cos\phi)\frac{U_1}{m} \\
\ddot{Y} &= (-cos\psi sin\theta + sin\psi sin\theta cos\phi)\frac{U_1}{m} \\
\ddot{Z} &= -g + (cos\theta cos\phi)\frac{U_1}{m} \\
\dot{p} &= \frac{I_{yy}-I_{zz}}{I_{xx}}qr - \frac{J_{TP}}{I_{xx}}q\omega + \frac{U_2}{I_{xx}} \\
\dot{q} &= \frac{I_{zz}-I_{xx}}{I_{yy}}pr + \frac{J_{TP}}{I_{yy}}p\omega + \frac{U_3}{I_{yy}} \\
\dot{r} &= \frac{I_{xx}-I_{yy}}{I_{zz}}pq + \frac{U_4}{I_{zz}}
\end{aligned} \tag{1}$$

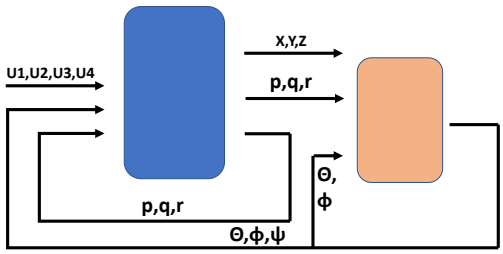
where U_1,U_2,U_3,U_4 are could be determined with the following formulas.

$$\begin{aligned}
U_1 &= m(g + \ddot{Z}_d + k_p e_z + k_d \dot{e_z} + k_I \int e_k dt) \\
U_2 &= I_x x(k_d \dot{e_\phi} + k_p e_\phi + k_I \int e_\phi dt) \\
U_3 &= I_y y(k_d \dot{e_\theta} + k_p e_\theta + k_I \int e_\theta dt) \\
U_3 &= I_z z(k_d \dot{e_\psi} + k_p e_\psi + k_I \int e_\psi dt)
\end{aligned} \tag{2}$$

From the equation 2 there are some gyroscopic terms as $\frac{J_{TP}}{I_{yy}}p\omega$ and $\frac{J_{TP}}{I_{xx}}q\omega$ which are very small and negligible. After apply that formula in the Simulink we need to determine the angles of ϕ,θ,ψ. The relation between p,q,r and ϕ,θ,ψ could be represented by Jacobian transformation given below. The s,c,t is written instead of sin,cos,tan functions.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} [I_{3x3}] & & [0_{3x3}] & \\ & 1 & s\phi t\theta & c\phi t\theta \\ [0_{3x3}] & 0 & c\phi & -s\phi \\ & 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \tag{3}$$

Fig. 3. Dynamics Block



As seen in the Fig: 3, the block is designed to get X,Y,Z and ϕ,θ,ψ as well with the current U_1,U_2,U_3 and U_4 values. This block is common for the both 4 DOF Hover Control and 6 DOF Trajectory Tracking Control since it give us the dynamics of our quadrotor. In the following sections Hover Control and Trajectory Tracking Control will talked with details.

A. Hover Control

Hovering mode is very essential for a quadrotor to move since it couldn't move on the ground. So that we need to design a SIMULINK blocks for that purpose. Dynamic model of the quadrotor is similar with the mentioned model in section ii "Quadrotor Dynamics". But there are some minor differences between trajectory tracking. In hover control we can find out the $\ddot{\phi}$,$\ddot{\theta}$,$\ddot{\psi}$ and $\ddot{Z}$ values from the following equation 4:

$$\begin{aligned} \ddot{Z} &= -g + (cos\theta cos\phi)\frac{U_1}{m} \\ \ddot{\phi} &= \frac{U_2}{I_{XX}} \\ \ddot{\theta} &= \frac{U_3}{I_{YY}} \\ \ddot{\psi} &= \frac{U_4}{I_{ZZ}} \end{aligned} \tag{4}$$

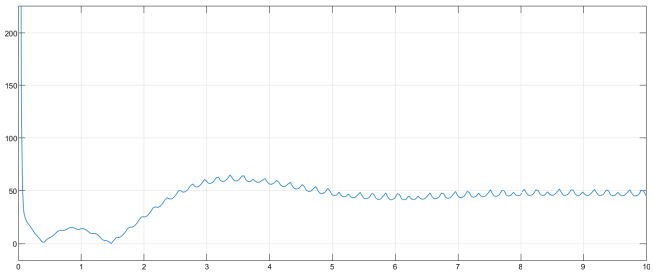
The inertia values I_{XX} ,I_{YY} , I_{ZZ} and some constants defined as followed in the MATLAB file initializations.m. Since the

Symbol	Description	Magnitude
m	Mass	4.5 kg
g	Gravity	9.8 m/sec^2
I_{XX}	Moment of inertia along x axis	0.405kg m^2
I_{YY}	Moment of inertia along y axis	0.405kg m^2
I_{ZZ}	Moment of inertia along z axis	0.72 kg m^2

TABLE I
MODELING PARAMETERS

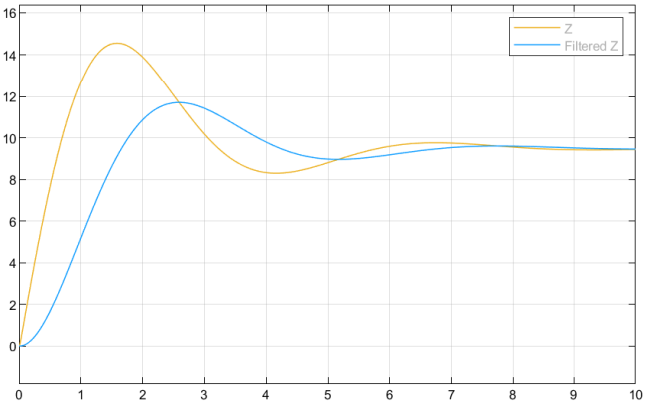
aim is just hovering, desired values of the Z ϕ θ, ψ would be given as a constant value. With the disturbances in the real life scenario, the values for pitch, yaw and roll could a bit tricky but since we are ignoring the disturbances, the desired values of ϕ θ would be 0,0 and ψ would be any value determined by user. Also $Z_{desired}$ value would be fixed value which is determined by the user and our system would try to achieve that references. After quadrotor reach the given altitude it need to be stay at there. If we assume that there is no disturbances, we can apply hard coded, fuzzy control over the quadrotor; we can stay at that altitude with U_1 total thrust would be equal to m*g. This approach gives us the desired hovering but control would be not applied on the open air and with disturbances. When look at the figure 4 we can conclude that the quadrotor try to stay in the air in certain altitude. If we apply fuzzy control over this probably it would move upward slightly. Under that conditions I also

Fig. 4. Total Thrust in Hover Control



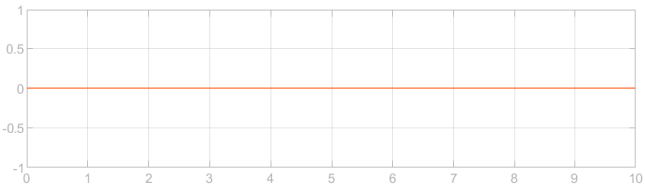
gather such plots that shows desired altitude is achieved and there is such a control over the quadrotor. In figure 5 there

Fig. 5. Z in Hover Control



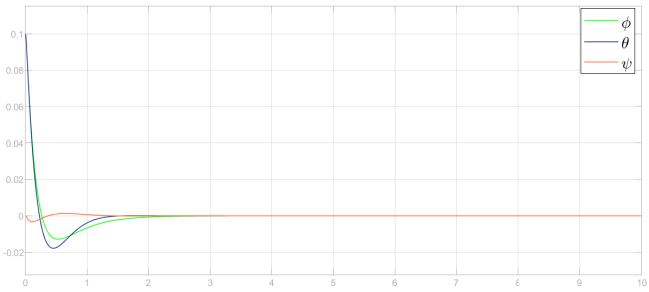
is two lines data points since I apply filter over the signal to receive more successful data. and it shows that our system is overshoot at the beginning but within a time this overshot is over and stays in the desired altitude. From figure 6 we can

Fig. 6. ϕ $\theta$and ψ



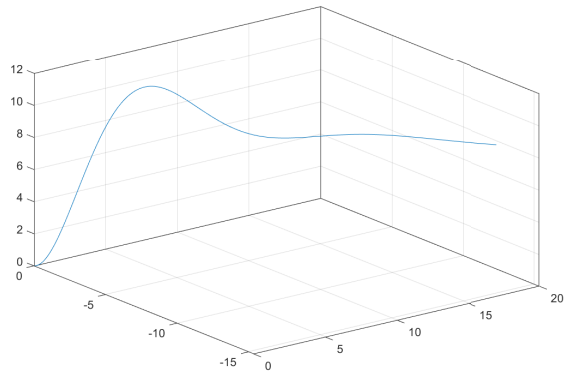
say that there is no tilting or rotating over the time interval. The main reason of this is there is no such disturbances or anomaly at the initial position of the quadrotor. When we have ϕ θ as 0.1 radian initialy, the following figure 7 would be the results. System tries to fix its rotation. When we generate the figure correspond to xy plane, we can see there

Fig. 7. ϕ $\theta$with small angles



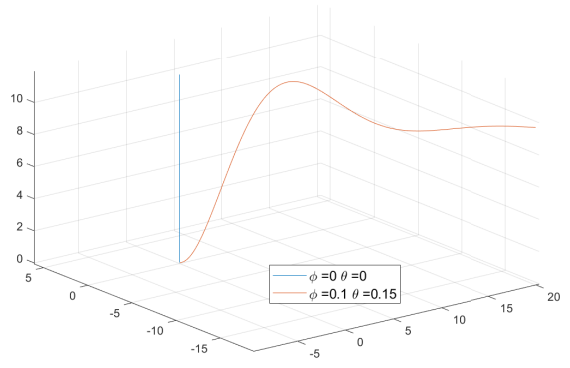
is no control over xy plane. This is a bit problematic thing since we just need to hover the quadrotor. The slipping to a

Fig. 8. Trajectory with ϕ θ are small angles



certain position could be seen when the initial angles of ϕ and θ is not equal to 0. When I encounter with that problem I just need to plot the xyz data for initial conditions as 0, and obtain the graph below. When we analyze it it looks perfect movement. And this experiment shows that since there is no control over the x,y acceleration it is not possible us to control the slippage in hover control. And this graph makes

Fig. 9. Trajectory comparison with different angles



us to ask "then what should we done to fix it?". The answer is very simple: Trajectory tracking control.

B. Trajectory Tracking Control

In trajectory tracking control we couldn't give certain angles and altitudes. About the angles; the quadrotor should decide what is the desired angle for the movement. Since the thrust is a very big role in roll and pitch movements we need to determine what should be the roll and pitch angle to move that way. So that need a block for just control the thrust and roll, pitch angles. For these reason we need to design an attitude controller. The relation between ϕ ,θ, u_1 is written in the formulas below:

$$\begin{aligned} u_1 &= m\sqrt{\mu_x^2+\mu_y^2+(\mu_z+g)^2} \\ \phi_d &= asin(\tfrac{sin(\psi_d)\mu_x - cos(\psi_d)\mu_y}{\sqrt{\mu_x^2+\mu_y^2+(\mu_z+g)^2}}) \\ \theta_d &= asin(\tfrac{cos(\psi_d)\mu_x + sin(\psi_d)\mu_y}{cos(\phi_d)\sqrt{\mu_x^2+\mu_y^2+(\mu_z+g)^2}}) \end{aligned} \tag{5}$$

where μ_x, μ_y, μ_z expressed as below under PD control but not PID control.

$$\begin{aligned} \mu_x &= \ddot{X} + K_{dx}\dot{e}_x + K_{px}e_x \\ \mu_y &= \ddot{Y} + K_{dy}\dot{e}_y + K_{px}e_y \\ \mu_z &= \ddot{Z} + K_{dz}\dot{e}_z + K_{pz}e_z \end{aligned} \tag{6}$$

In the equation terms of $\dot{e}_x$ and e_x are represent the error in the feedback control system to determine these values we need to calculate them in terms of x,y and z:

$$\begin{aligned} \ddot{e}_x &= \ddot{X}_d - \ddot{X} \quad \dot{e}_x = \dot{X}_d - \dot{X} \quad e_x = X_d - X \\ \ddot{e}_y &= \ddot{Y}_d - \ddot{Y} \quad \dot{e}_y = \dot{Y}_d - \dot{Y} \quad e_y = Y_d - Y \\ \ddot{e}_z &= \ddot{Z}_d - \ddot{Z} \quad \dot{e}_Z = \dot{Z}_d - \dot{Z} \quad e_z = Z_d - Z \end{aligned} \tag{7}$$

With all these equations we can follow any desired trajectory given to us. But there is no such trajectory to follow. So when it comes to generate certain trajectories, we need I apply quintic polynomial equation for generate the trajectory. I use following steps to generate a trajectory.

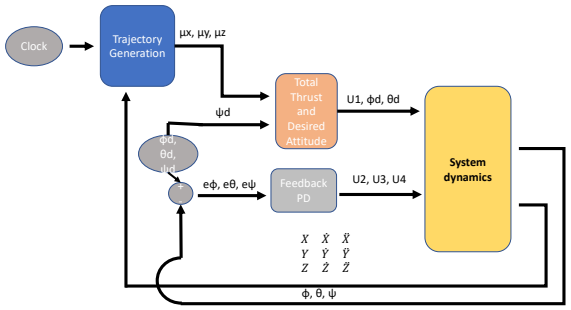
$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_0^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ a_0 \\ q_f \\ v_f \\ a_f \end{bmatrix} \tag{8}$$

Where q_0,v_0,a_0 are initial position,velocity and acceleration respectively and q_f,v_f,a_f are final position,velocity and acceleration respectively. After that find the constant a_0....a_5 for x,y and z. To apply this I take inverse of the matrix and multiply with the position,velocity, acceleration vector. The reason for these operations are determine the values of a_0....a_5 and put them into the format of:

$$q(t) = a_0 1 + a_1 t + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 \tag{9}$$

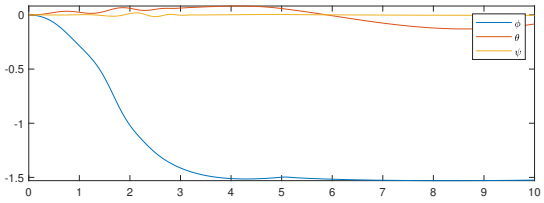
When we have this equation and take the derivative of it to get the velocity of it, we have a curve to represent the trajectory, also when we look at the acceleration o this formula we have 3^{rd} order polynomial for representing the trajectory. If we have linear or constant acceleration the simulations would not be as stable as what we get now.

Fig. 10. Trajectory Tracking Block



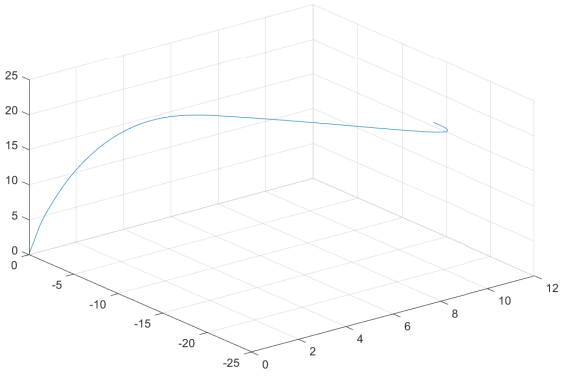
After we get the $\mu_{x,y,z}$ values we need to determine the values for u_1,ϕ_d and θ_d with the formula 5. With that approach, I achieve the graph below. The quadrotor follow the certain trajectory which I gave it to follow. We can understand

Fig. 11. Trajectory Tracking Euler Angles



that movement towards the given point is achieved with the rotation around the body. Since quadrotors mostly use its own body to move around to the target achieving that result is show that it is successful.

Fig. 12. Trajectory Tracking to a certain point

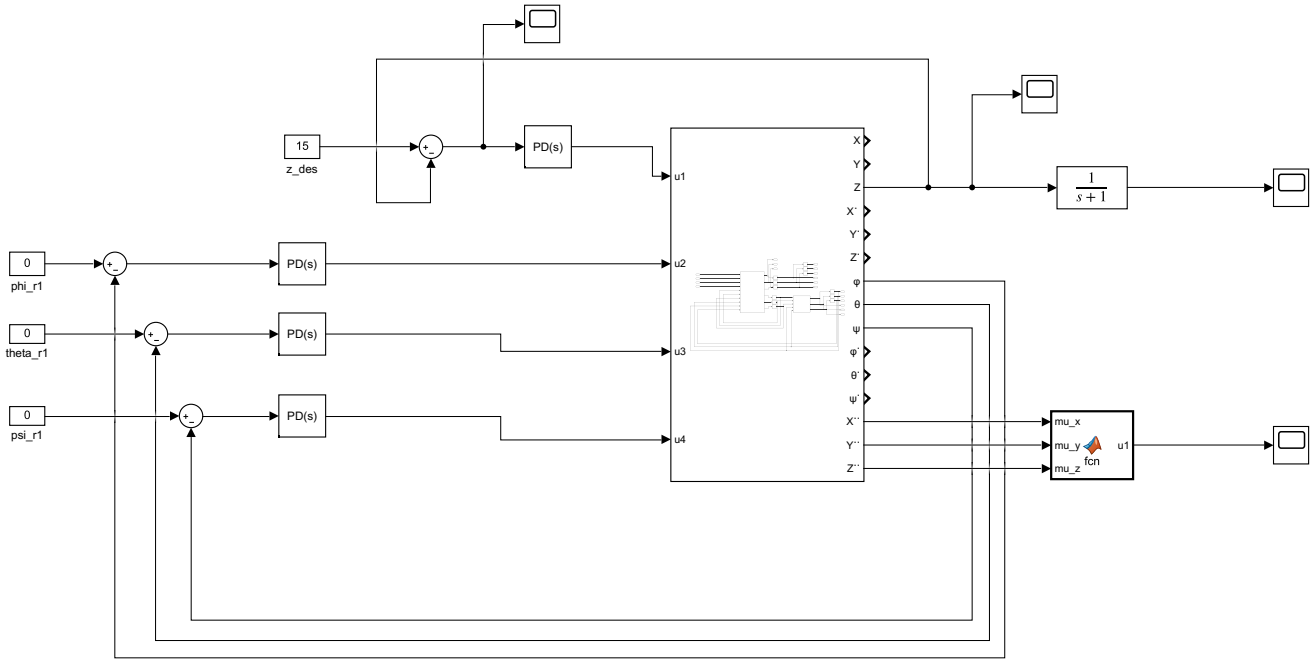


C. CONCLUSION

As a result the quadrotor control should be talked in 2 manner. First one is the hovering and the other one is trajectory tracking. In this assignment I achieve both of the movement with the Simulink and Matlab environment. The movements could be vary with more spesific location and environmental informations. This model could be updated with the disturbances and examine the responses of the system under the external effects. Also with an IMU module we can determine the actual Euler angles and it would makes our system more stable under the external effects.

APPENDIX

Fig. 13. Hover Control Simulink Model



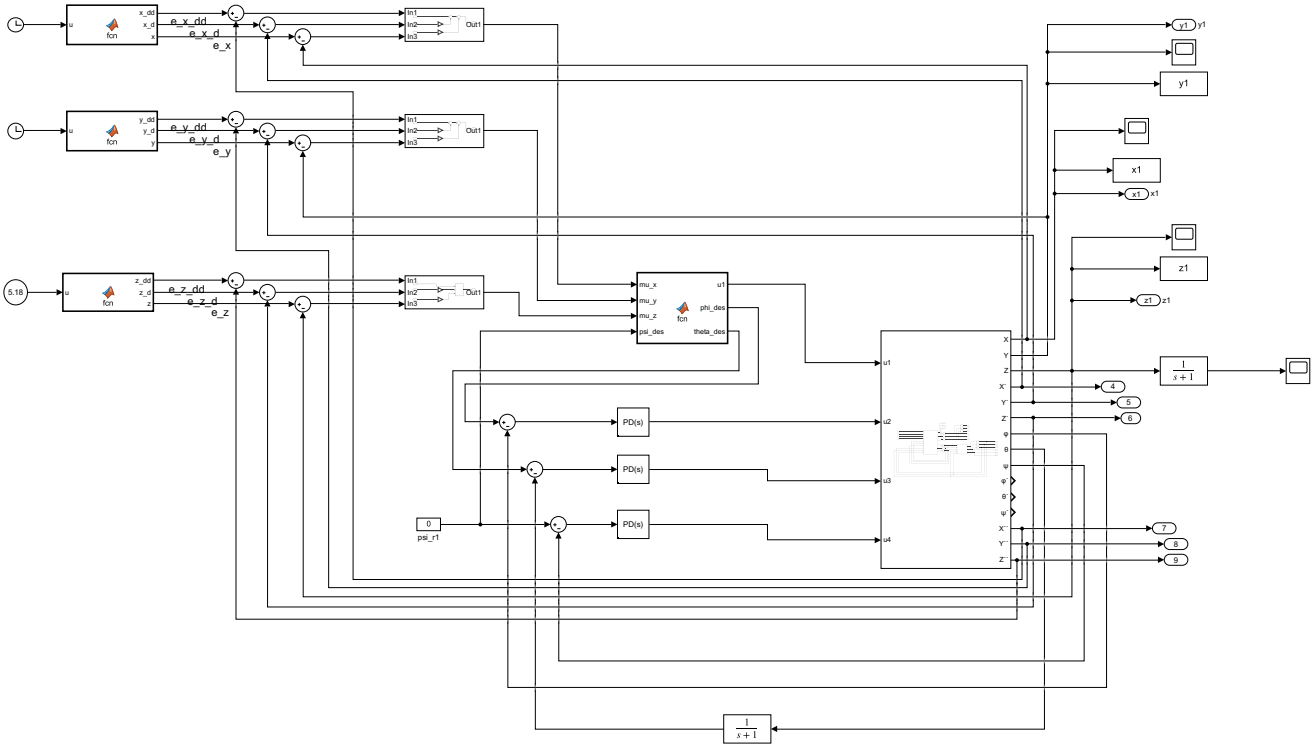
Listing 1. Total Thrust and Desired Attitude Angle generation

```
function [u1,phi_des,theta_des] = fcn(mu_x,mu_y,mu_z,psi_des)

u1=m*sqrt(mu_x^2+mu_y^2+(mu_z+g)^2);
phi_des=asin((sin(psi_des)*mu_x-cos(psi_des)*mu_y)/(u1/m));
theta_des=asin((cos(psi_des)*mu_x+sin(psi_des)*mu_y)/(cos(phi_des)*u1/m));

end
```

Listing 2. initializations.m

```
g= 9.8;%m/sec^2
m= 4.5; %kg
l_s=0.3;%m rotor distance between centre of gravity along y and x axis
L_l=0.3;
I_xx=0.405;%kg m^2
I_yy=0.405;
I_zz=0.72;
lambda = 0.01;%Nm/N
```

Fig. 14. Trajectory Tracking Simulink Model



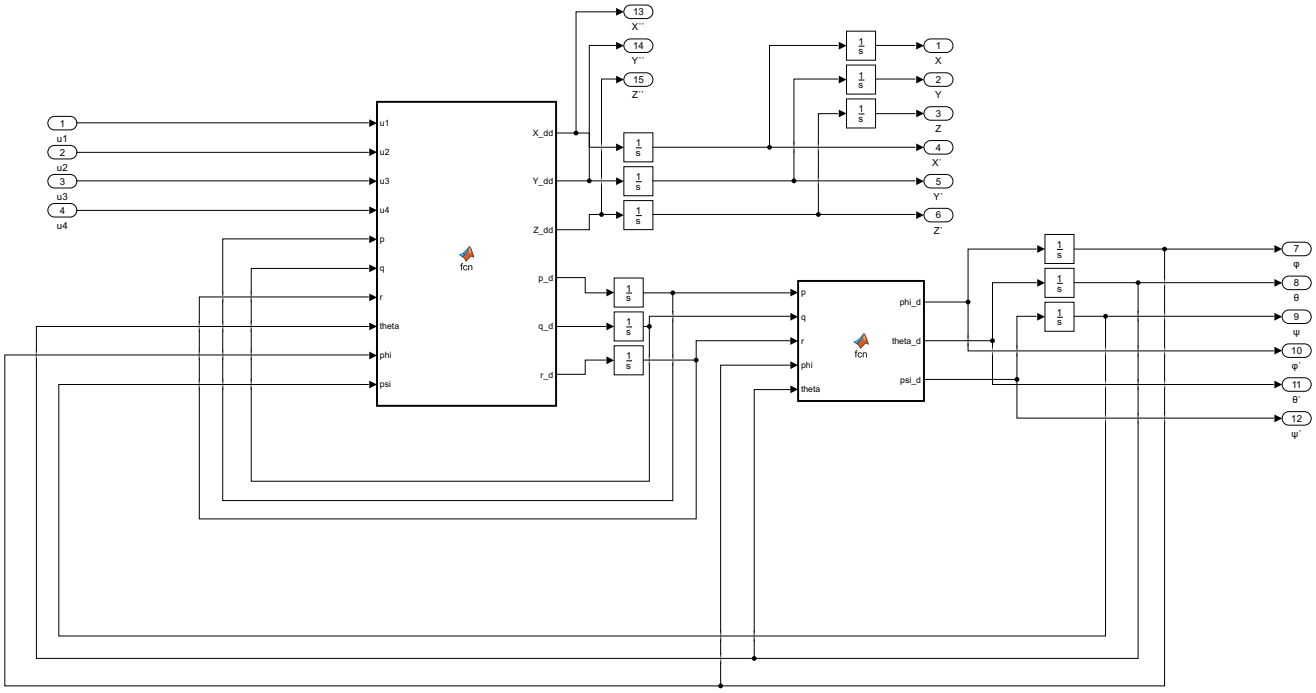
Listing 3. Helper function for Trajectory Generating

```
syms t_f t_0;
t_0=5;
t_f=10;
A= [1 t_0 t_0^2 t_0^3 t_0^4 t_0^5;
     0 1 2*t_0 3*t_0^2 4*t_0^3 5*t_0^4;
     0 0 2 6*t_0 12*t_0^2 20*t_0^3;
     1 t_f t_0^2 t_f^3 t_f^4 t_f^5;
     0 1 2*t_f 3*t_f^2 4*t_f^3 5*t_f^4;
     0 0 2 6*t_f 12*t_f^2 20*t_f^3;];
A_inv= inv(A);
X=[0;0;0;15;0;0];
Y=[0;0;0;15;0;0];
Z=[20;0;0;0;0;0];
xd=A_inv*X;
yd=A_inv*Y;
zd=A_inv*Z;
```

Listing 4. Trajectory generation for x axis

```
function [x_dd,x_d,x] = fcn(u)
if u>5
     x=-0.8595*1+ 0.6654* u +-0.1996*u^2 +0.0288*u^3 + -0.0020*u^4 +0.0001*u^5;
     x_d=0+  0.6654*1 +-0.1996*2*u +0.0288*3*u^2+ -0.0020*4*u^3+ 0.0001*5*u^4;
     x_dd=0+ 0+ -0.1996*2 +0.0288*6*u + -0.0020*12*u^2+ 0.0001*20*u^3;

else
```

Fig. 15. Dynamic Model of Quadrotor




```
    x=0.1*u^3- 0.015*u^4+ 0.0006*u^5;
    x_d=0.1*3*u^2- 0.015*4*u^3+ 0.0006*5*u^4;
    x_dd= 0.1*6*u- 0.015*12*u^2+ 0.0006*20*u^3;
end
end
```

Listing 5. Trajectory generation for y axis

```
function [y_dd,y_d,y] = fcn(u)
if u>5
    y=-0.8595*1+ 0.6654* u +-0.1996*u^2 +0.0288*u^3 + -0.0020*u^4 +0.0001*u^5;
    y_d=0+  0.6654*1 +-0.1996*2*u +0.0288*3*u^2+ -0.0020*4*u^3+ 0.0001*5*u^4;
    y_dd=0+ 0+ -0.1996*2 +0.0288*6*u + -0.0020*12*u^2+ 0.0001*20*u^3;
else
    y=0;
    y_dd=0;
    y_d=0;
end
end
```

Listing 6. Trajectory generation for z axis

```
function [z_dd,z_d,z] = fcn(u)

if u≤5
```

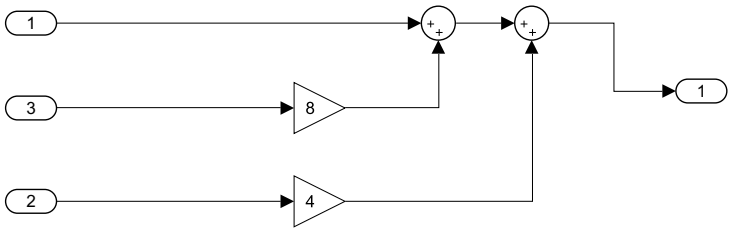


Fig. 16. $k_p x, k_d x$ controller gains

```
    z=1.6*u^3- 0.48*u^4+ 0.0384*u^5;
    z_d=1.6*3*u^2- 0.48*4*u^3+ 0.0384*5*u^4;
    z_dd= 1.6*6*u- 0.48*12*u^2+ 0.0384*20*u^3;
else
    z=-21.1460*1-0.8872* u +0.2662*u^2 -0.0384*u^3 + -0.0020*u^4 +0.0001*u^5;
    z_d=-0.8872*1 +0.2662*2*u -0.0384*3*u^2+ -0.0020*4*u^3+ 0.0001*5*u^4;
    z_dd=0+ 0+0.2662*2 -0.0384*6*u + -0.0020*12*u^2+ 0.0001*20*u^3;
end
end
```