Appendix

Appendix A1: Quadcopter plant

```
function quadplant(block)
setup(block);

function setup(block)

  block.NumInputPorts  = 4 ;

  block.NumOutputPorts = 12;

  for i = 1:4; % These are the motor inputs
  block.InputPort(i).Dimensions = 1;
  block.InputPort(i).DirectFeedthrough = false;
  block.InputPort(i).SamplingMode = 'Sample'; end


  for i = 1:12;
  block.OutputPort(i).Dimensions       = 1;
  block.OutputPort(i).SamplingMod      = 'Sample';
  e
  end

  % Register the parameters.
  block.NumDialogPrms     = 0; %fromtemplate

  % Set up the continuous states.
 block.NumContStates = 12; %notintemplate

  block.SampleTimes = [0 0];

  block.SetAccelRunOnTLC(false);

  block.SimStateCompliance = 'DefaultSimState';

  block.RegBlockMethod('InitializeConditions', @InitializeConditions);

  block.RegBlockMethod('Outputs', @Outputs);

  block.RegBlockMethod('Derivatives', @Derivatives);
  block.RegBlockMethod('Terminate', @Terminate); % Required


function InitializeConditions(block)

% P, Q, R are in
rad/s P=0; Q=0; R=0;

% Phi, The, Psi are in rads
```

```
Phi=10*pi/180; The=12*pi/180; Psi=10*pi/180;

U=0; V=0; W=0;
X=0; Y=0; Z=2;

init = [P,Q,R,Phi,The,Psi,U,V,W,X,Y,Z];

for i=1:12
block.OutputPort(i).Data = init(i);
block.ContStates.Data(i) = init(i);
end

function Outputs(block)
for i = 1:12;
  block.OutputPort(i).Data =
block.ContStates.Data(i); end

function Derivatives(block)

% P Q R in units of rad/sec
P = block.ContStates.Data(1);
Q = block.ContStates.Data(2);
R = block.ContStates.Data(3);
% Phi The Psi in radians
Phi = block.ContStates.Data(4);
The = block.ContStates.Data(5);
Psi = block.ContStates.Data(6);
% U V W in units of m/s
U = block.ContStates.Data(7);
V = block.ContStates.Data(8);
W = block.ContStates.Data(9);
% X Y Z in units of m
X = block.ContStates.Data(10);
Y = block.ContStates.Data(11);
Z = block.ContStates.Data(12);
% w values in rev/min! NOT radians/s!!!!
w1 = block.InputPort(1).Data;
w2 = block.InputPort(2).Data;
w3 = block.InputPort(3).Data;
w4 = block.InputPort(4) .Data;
w = [w1; w2; w3; w4];

% CALCULATE MOMENT AND THRUST FORCES

%find k,d,l
k=2.98e-06; d=.0382; l=0.225;

%find m,Ixx,Iyy,Izz,Ir
m=0.468; Ixx=4.856e-03;Iyy=4.856e-03;Izz=8.801e-03;Ir=3.357e-05;
Ax=.3; Ay=0.3; Az=0.25; Ar=0.2;
T1= k*w1^2;
T2= k*w2^2;
T3= k*w3^2;
T4= k*w4^2;
```

```
T = T1+T2+T3+T4; %total thrust
Mphi= l*(T4-T2); %torques
Mthe= l*(T3-T1);
Mpsi= d*(-T1+T2-T3+T4);

Omega=w1-w2+w3-w4;

dP=  ((Iyy-Izz)/Ixx)*Q*R  -  Ir/Ixx  *  Q*Omega  +  Mphi/Ixx  -
Ar/Ixx*P; dQ= ((Izz-Ixx)/Iyy)*P*R + Ir/Iyy * P*Omega + Mthe/Iyy
- Ar/Iyy*Q; dR= ((Ixx-Iyy)/Izz)*P*Q + Mpsi/Izz -Ar/Izz*R;

dPhi= P+ sin(Phi)*tan(The)*Q + cos(Phi)*tan(The)*R;
dTheta= cos(Phi)*Q - sin(Phi)*R;
dPsi= sin(Phi)/cos(The)*Q + cos(Phi)/cos(The)*R;

dU= ( sin(Phi)*sin(Psi) + cos(Phi)*sin(The)*cos(Psi) )*T/m - Ax/m*U;
dV= ( -sin(Phi)*cos(Psi) + cos(Phi)*sin(The)*sin(Psi) )*T/m -
Ay/m*V; dW= -9.8 + cos(Phi)*cos(The)*T/m - Az/m*W;

vb = [U;V;W];

dX = U;
dY = V;
dZ = W;

f = [dP dQ dR dPhi dTheta dPsi dU dV dW dX dY dZ].';

  %This is the state derivative vector
block.Derivatives.Data = f;

function Terminate(block)

%endfunction
```

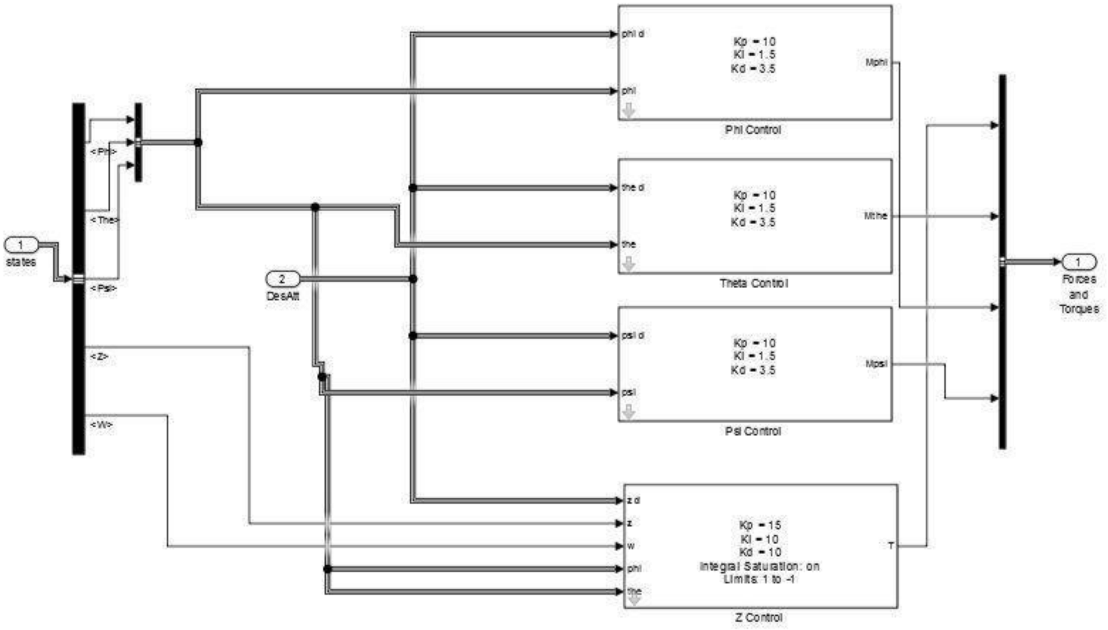
Appendix A2: Quadcopter plant with a failed rotor

```
 function quadplant2(block)
setup(block);

function setup(block)

  block.NumInputPorts  = 3;

  block.NumOutputPorts = 12;


  for i = 1:3; % These are the motor inputs
  block.InputPort(i).Dimensions = 1;
  block.InputPort(i).DirectFeedthrough = false;
  block.InputPort(i).SamplingMode = 'Sample'; end


  for i = 1:12;
  block.OutputPort(i).Dimensions       = 1;
  block.OutputPort(i).SamplingMod      = 'Sample';
  e
  end

  % Register the parameters.
  block.NumDialogPrms     = 0; %fromtemplate

  % Set up the continuous states.
 block.NumContStates = 12; %notintemplate

  block.SampleTimes = [0 0];

  block.SetAccelRunOnTLC(false);

  block.SimStateCompliance = 'DefaultSimState';

  block.RegBlockMethod('InitializeConditions', @InitializeConditions);

  block.RegBlockMethod('Outputs', @Outputs);

  block.RegBlockMethod('Derivatives', @Derivatives);
  block.RegBlockMethod('Terminate', @Terminate); % Required


function InitializeConditions(block)
% P, Q, R are in
rad/s P=0; Q=0; R=0;

% Phi, The, Psi are in rads
Phi=10*pi/180; The=12*pi/180; Psi=10*pi/180;

U=0; V=0; W=0;
X=0; Y=0; Z=2;
```

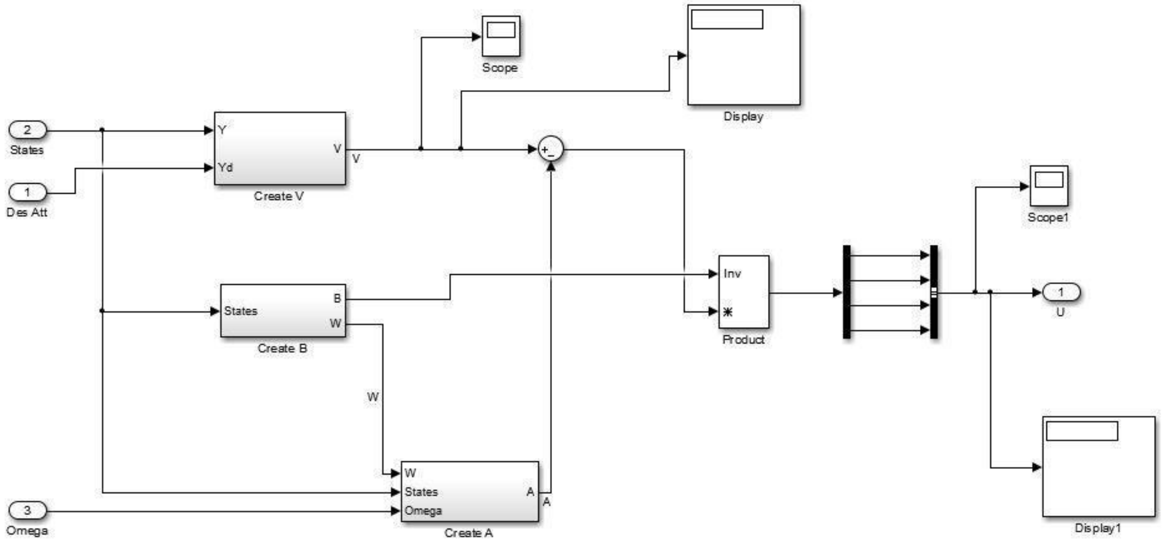
```
init = [P,Q,R,Phi,The,Psi,U,V,W,X,Y,Z];

for i=1:12
block.OutputPort(i).Data = init(i);
block.ContStates.Data(i) = init(i);
end

function Outputs(block)
for i = 1:12;
  block.OutputPort(i).Data =
block.ContStates.Data(i); end

function Derivatives(block)

% P Q R in units of rad/sec
P = block.ContStates.Data(1);
Q = block.ContStates.Data(2);
R = block.ContStates.Data(3);
% Phi The Psi in radians
Phi = block.ContStates.Data(4);
The = block.ContStates.Data(5);
Psi = block.ContStates.Data(6);
% U V W in units of m/s
U = block.ContStates.Data(7);
V = block.ContStates.Data(8);
W = block.ContStates.Data(9);
% X Y Z in units of m
X = block.ContStates.Data(10);
Y = block.ContStates.Data(11);
Z = block.ContStates.Data(12);
% w values in rev/min! NOT radians/s!!!!
w1 = block.InputPort(1).Data;
w3 = block.InputPort(2).Data;
w4 = block.InputPort(3).Data;
w = [w1; w3; w4];

%find k,d,l
k=2.98e-06; d=.03825; l=0.225;

%find m,Ixx,Iyy,Izz,Ir
m=0.468; Ixx=4.856e-03;Iyy=4.856e-03;Izz=8.801e-03;Ir=3.357e-
05; Ax=.3; Ay=0.3; Az=0.25; Ar=0.1;

T1= k*w1^2;
%T2= k*w2^2;
T3= k*w3^2;
T4= k*w4^2;

Fmat= [ 1 1 1; -l l 0; -d -d d;];
Fmat1=inv(Fmat);

mat1= [T1;T3;T4];
mat2= Fmat*mat1;
```

```
T = mat2(1); %total thrust
Mthe= mat2(2);%torques
Mpsi= mat2(3);

%Mphi is not used as control input but appears later in
eq %Substitute for Mphi
Mphi= 0.5*l*(T-Mpsi/d);
Omega=w1+w3-w4; %or opp signs check.

dP=  ((Iyy-Izz)/Ixx)*Q*R  -  Ir/Ixx  *  Q*Omega  +  Mphi/Ixx  -
Ar/Ixx*P; dQ= ((Izz-Ixx)/Iyy)*P*R + Ir/Iyy * P*Omega + Mthe/Iyy
- Ar/Iyy*Q; dR= ((Ixx-Iyy)/Izz)*P*Q + Mpsi/Izz -Ar/Izz*R;

dPhi= P+ sin(Phi)*tan(The)*Q + cos(Phi)*tan(The)*R;
dTheta= cos(Phi)*Q - sin(Phi)*R;
dPsi= sin(Phi)/cos(The)*Q + cos(Phi)/cos(The)*R;

dX = U;
dY = V;
dZ = W;

dU= ( sin(Phi)*sin(Psi) + cos(Phi)*sin(The)*cos(Psi) )*T/m - Ax/m*U;
dV= ( -sin(Phi)*cos(Psi) + cos(Phi)*sin(The)*sin(Psi) )*T/m -
Ay/m*V; dW= -9.8 + cos(Phi)*cos(The)*T/m - Az/m*W;



vb = [U;V;W];
Rib = [cos(Psi)*cos(The) cos(Psi)*sin(The)*sin(Phi)-
sin(Psi)*cos(Phi) cos(Psi)*sin(The)*cos(Phi)+sin(Psi)*sin(Phi);
       sin(Psi)*cos(The) sin(Psi)*sin(The)*sin(Phi)+cos(Psi)*cos(Phi)
sin(Psi)*sin(The)*cos(Phi)-cos(Psi)*sin(Phi);
       -sin(The)         cos(The)*sin(Phi)
cos(The)*cos(Phi)];
% i_dp = Rib*vb;
%dX = i_dp(1);
%dY = i_dp(2);
%dZ = i_dp(3);
f = [dP dQ dR dPhi dTheta dPsi dU dV dW dX dY dZ].';

  %This is the state derivative vector
block.Derivatives.Data = f;

function Terminate(block)

%endfunction
```

Appendix A3: Layout for PID controller

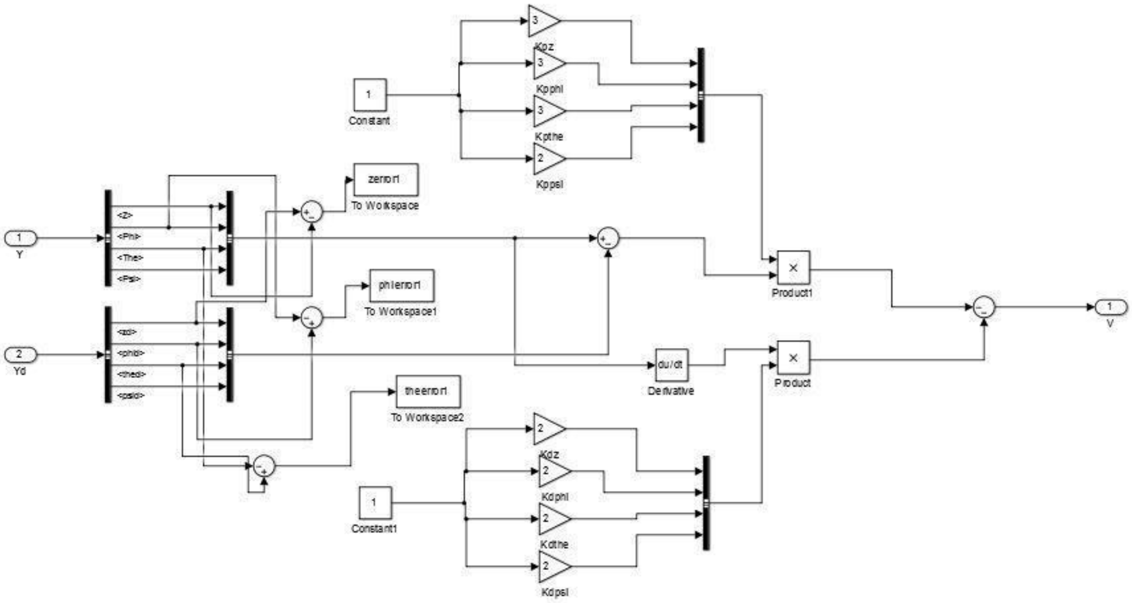


Appendix A4: Layout for FBL+PD controller



Appendix A5: Inside FBL blocks

Inside Create V block



Matrices used for Feedback linearization

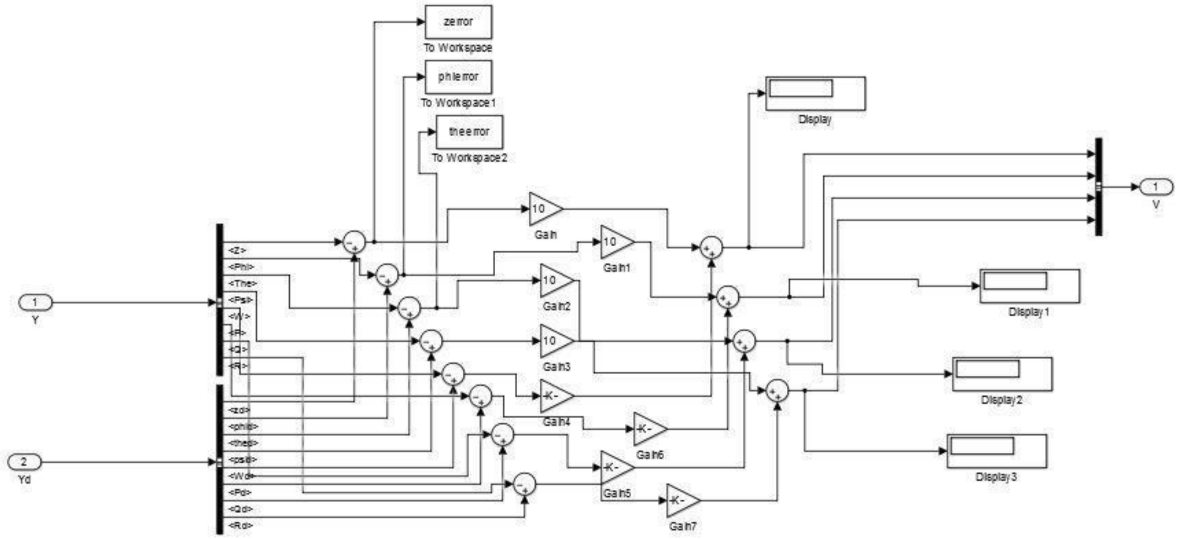
```
function B = MatW(phi,the)
B=[ 1 0 0 0; 0 1 sin(phi)*tan(the) cos(phi)*tan(the); 0 0 cos(phi) -
sin(phi); 0 0 sin(phi)/cos(the) cos(phi)/cos(the)];
end
```

```
function B = MatD(phi,the)
m=0.468; Ixx=4.856e-03;Iyy=4.856e-03;Izz=8.801e-03;
B=[ 1/m*cos(phi)*cos(the) 0 0 0; 0 1/Ixx 0 0; 0 0 1/Iyy 0; 0 0 0
1/Izz]; end
```

```
function B = MatC(P,Q,R,Omega,W)
m=0.468; Ixx=4.856e-03;Iyy=4.856e-03;Izz=8.801e-03;Ir=3.357e-
05;Ar=0.2; Az=.25;
B=[-9.8 - Az/m*W; (Iyy-Izz)/Ixx*Q*R - Ir/Ixx*Q*Omega - Ar/Ixx*P; (Izz-
Ixx)/Iyy*P*R + Ir/Iyy*P*Omega - Ar/Iyy*Q; (Ixx-Iyy)/Izz*P*Q - Ar/Izz*R
]; end
```

```
function B = MatA(W,Wdot,C,Q);
B = W*C+Wdot*Q;
end
```

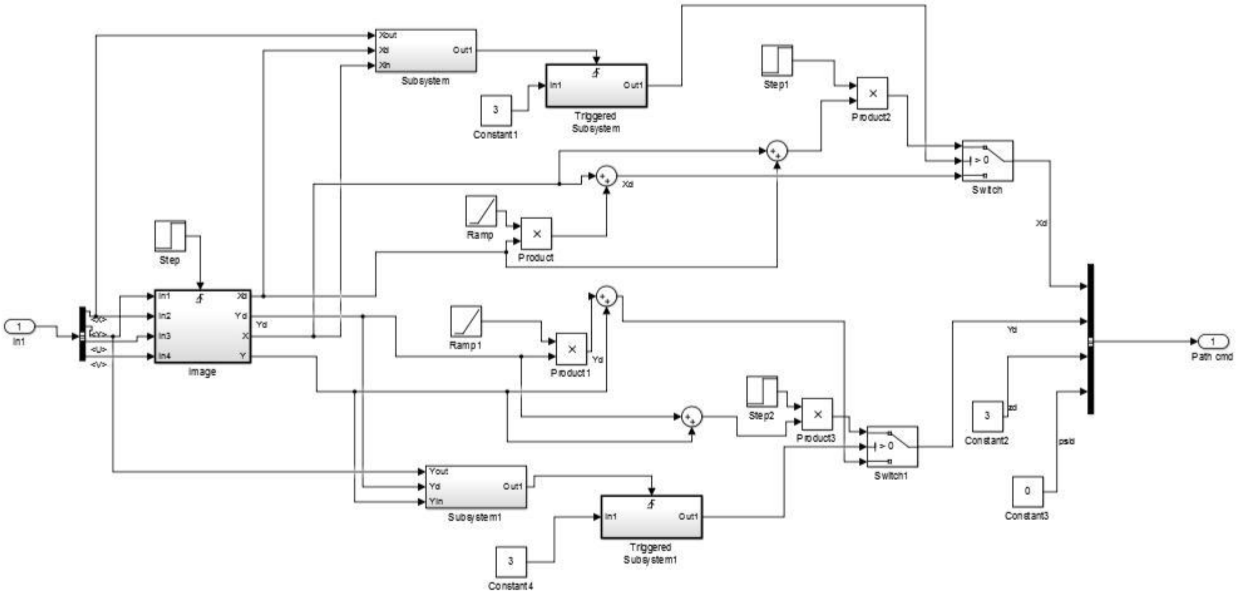
Appendix A6: Layout of LQR



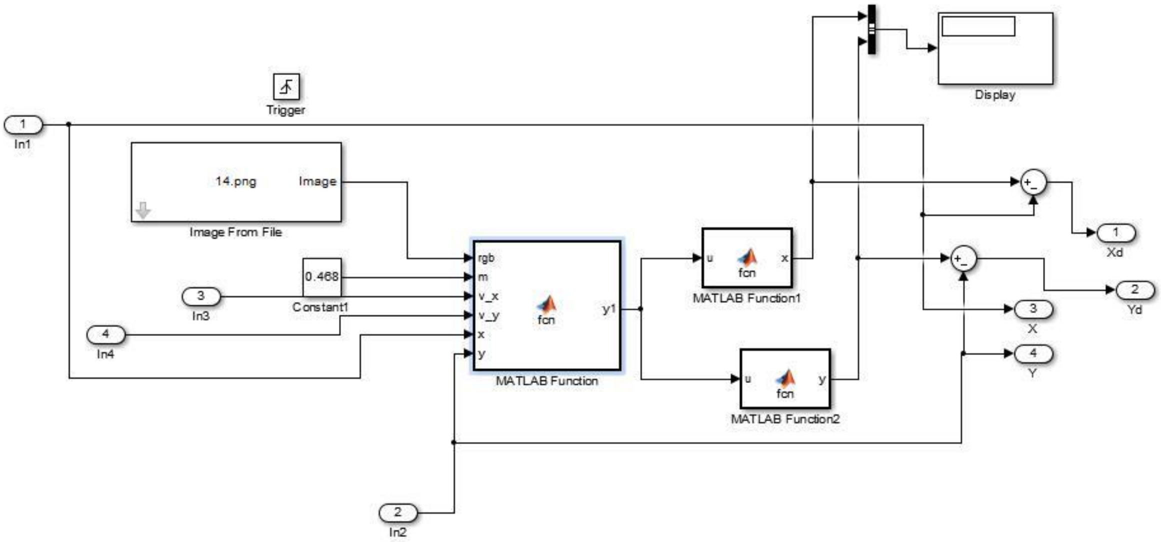
Appendix A7: Linriz.m function

```
A=[00001000;00000100;00000010;00000001;0000-
.5341000;00000-41.18600;000000-41.1860;0000000
22.725];
B=[0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0;2.137 0 0 0;0 205.93 0 0;0 0 205.93 0;0
0 0 29788.5];
C=[10000000;01000000;00100000;00010000]; D=zeros(4);

sys_ss = ss(A,B,C,D);
co = ctrb(sys_ss);
controllability = rank(co);
Q = C'*C;
R=eye(4);
K = lqr(A,B,Q,R);

%Increasing the weights to improve performance
Q(1,1)=100;
Q(2,2)=100;
Q(3,3)=100;
Q(4,4)=100;

K = lqr(A,B,Q,R);
```

Appendix A8: Inside image processing block



Inside the image block



Appendix A9: Pathgen.m function

```
function y= pathgen(L,m,v_x,v_y,x,y)
omega=1000;
F=4*2.98*10^(-6)*omega^2;
theta=pi/4;
% m=0.468;
a=F*sin(theta)/m;
% v_x=10;
% v_y=10;
s=sqrt(v_ x^2+v_y^2);
% x=150;
% y=150;

k=1;
rows=size(L);
while k<=rows(1,1)
    theta1=atan2(L(k,2)-y,L(k,1)-x);
    theta2=atan2(v_y,v_x);
    theta=theta1-theta2;
     v_a=abs(s*cos(theta));
     v_p=abs(s*sin(theta));

d=sqrt((x-L(k,1))^2 + (y-L(k,2))^2);

if theta == 0 || s ==0
    t(k)=(1/a)*(sqrt(v_a^2+2*a*d)-v_a);

elseif theta == pi || theta == -pi
    t(k)=(1/a)*(sqrt(v_a^2+2*a*d)+v_a);


else
syms u v
[solv, solu] = solve(u^2 + v^2 == a^2, (-2*v_a*v_p)/u + (2*v_p^2*v)/(u^2)
== d);

p=1;

while p<=length(solv)
if isreal(solv(p))==1
 if (L(k,1)-x)*solv(p)>=0
    a_x=solu(p);
    a _y=solv(p);
 end
end
p=p+1;
end

t(k)=abs(-2*v_p/a_y);
end
k=k+1;
end
```

```
min = t(1);r=1;
l=2;
while l<=length(t)
    if t(l)<=min
        min=t(l);
        r=l;
    end
    l=l+1;
end
y=L(r,:);
```

Appendix A10: Circle2.m function

```
function y= circle2(RGB)
% imshow(RGB);

I= rgb2gray(RGB);
% bw = imbinarize(I);
bw=im2bw(I,0.3);

 bw2 = bwmorph(~bw, 'dilate',2);
 bw = bwareaopen(bw2,500);
 se = strel('disk',2);
bw = imclose(bw,se);
bw = imfill(bw,'holes');

[B,L] = bwboundaries(bw,'noholes');



% Display the label matrix and draw each
boundary imshow(label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
  boundary = B{k};
  plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth',
2) end

stats = regionprops(L,'Area','Centroid');
threshold = 0.99;
centroid=zeros(length(B),2);


% loop over the
boundaries i=1;
for k = 1:length(B)

  % obtain (X,Y) boundary coordinates corresponding to label
  'k' boundary = B{k};
  % compute a simple estimate of the object's
  perimeter delta_sq = diff(boundary).^2;
  perimeter = sum(sqrt(sum(delta_sq,2)));

  % obtain the area calculation corresponding to label
  'k' area = uint32(stats(k).Area);


  % compute the roundness metric
  metric =(4*pi*area)/perimeter^2;

  % display the results
  metric_string = sprintf('%2.2f',metric);

  % mark objects above the threshold with a black
  circle if metric > threshold
    centroid(i,:)=stats(k).Centroid;
     plot(centroid(1),centroid(2),'ko');


  end
  i=i+1;

  text(boundary(1,2)-35,boundary(1,1)+13,metric_string,'Color','y',...
       'FontSize',14,'FontWeight','bold');

end
% disp(centroid);
p=1;
q=0;
while p<=length(B)

    if centroid(p,1)~=0

        q=q+1;
    end
    p=p+1;

end
x=zeros(q,2);
p=1;
q=1;
while p<=length(B)

    if centroid(p,1)~=0
        x(q,:)=centroid(p,:);
        q=q+1;
    end
    p=p+1;

end
y=x;
% title(['Metrics closer to 1 indicate that ',...
%        'the object is approximately round']);
%   imshow(bw);
```

Appendix A11: dijkstra.m function

```
%-------------------------------------------------
% Dijkstra Algorithm
% author : Dimas Aryo
% email : mr.dimasaryo@gmail.com
%
% usage
% [cost rute] = dijkstra(Graph, source, destination)
%
% example
% G=[0390000;
%      0007100;
%      0207000;
%      0000028;
%      0045090;
%      0000004;
%      0000000;
%      ];
% [e L] = dijkstra(G,1,7)
%-------------------------------------------------
function [e L] = dijkstra(A,s,d)

if s==d
    e=0;
    L=[s];
else

A = setupgraph(A,inf,1);

if d==1
    d=s;
end
A=exchangenode(A,1,s);

lengthA=size(A,1);
 W=zeros(lengthA)
        ;
 for i=2 : lengthA
    W(1,i)=i;
    W(2,i)=A(1,i);
end

for i=1 : lengthA
    D(i,1)=A(1,i);
    D(i,2)=i;
end

D2=D(2:length(D),:);
L=2;
```

```
while L<=(size(W,1)-1)
    L=L+1;
    D2=sortrows(D2,1);
    k=D2(1,2);
    W(L,1)=k;
    D2(1,:)=[];
    for i=1 : size(D2,1)
        if D(D2(i,2),1)>(D(k,1)+A(k,D2(i,2)))
            D(D2(i,2),1) = D(k,1)+A(k,D2(i,2));
            D2(i,1) = D(D2(i,2),1);
        end
    end

    for i=2 : length(A)
        W(L,i)=D(i,1);
    end
end
if d==s
    L=[1];
else
    L=[d];
end
e=W(size(W,1),d);
L = listdijkstra(L,W,s,d);
end
```

Appendix A12: Functions called by Dijkstra.m

(save as separate files)

```
function L = listdijkstra(L,W,s,d)

index=size(W,1);
while index>0
    if W(2,d)==W(size(W,1),d)
        L=[L s];
        index=0;
    else
        index2=size(W,1);
        while index2>0
            if W(index2,d)<W(index2 -1,d)
                if W(index2,1)==s
                    L=[L1];
                else
                    L=[L W(index2,1)];
                end
                L=listdijkstra(L,W,s,W(index2,1));
                index2=0;
            else
                index2=index2-1;
            end
            index=0;
        end
    end
end


function G = exchangenode(G,a,b)

%Exchange element at column a with element at column b;
buffer=G(:,a);
G(:,a)=G(:,b);
G(:,b)=buffer;

%Exchange element at row a with element at row b;
buffer=G(a,:);
G(a,:)=G(b,:);
G(b,:)=buffer;


function G = setupgraph(G,b,s)

if s==1
    for i=1 : size(G,1)
        for j=1 :size(G,1)
            if G(i,j)==0
                G(i,j)=b;
            end
        end
    end
end
if s==2
    for i=1 : size(G,1)
        for j=1 : size(G,1)
            if G(i,j)==b
                G(i,j)=0;
            end
        end
    end
end
```

Appendix A13: pathcr.m function

```
% function d_path = pathcr(m)
m=imread('path5.png');

n=rgb2gray(m);
a=im2bw(n);
p=ones(50,50);
for i=1:1:50
for j=1:1:50
if a(i,j)==0
    p(i,j)=5;
```

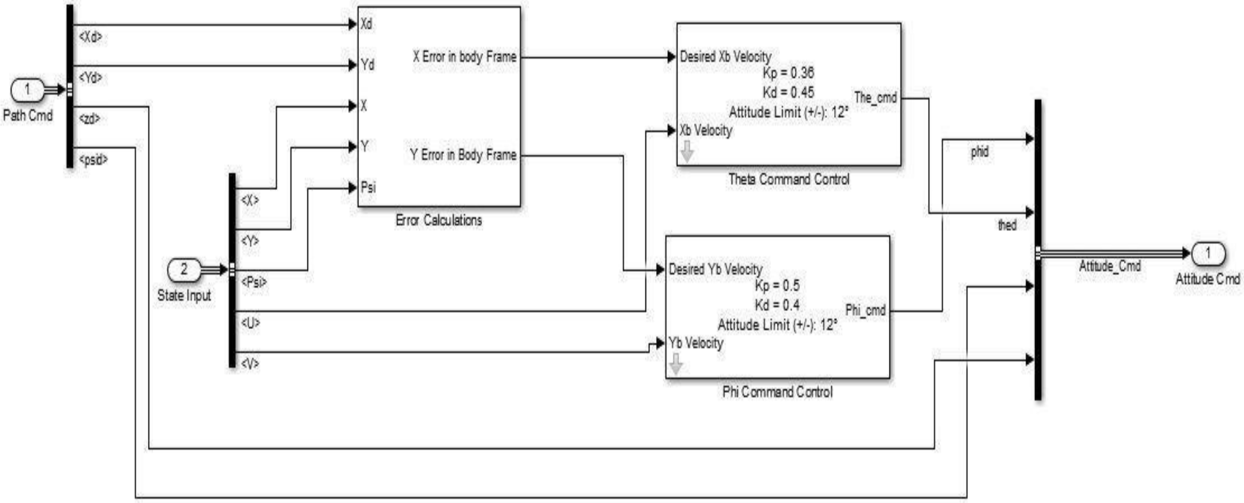
```
else
    p(i,j)=1;
end
end
end
%for l=1:1:100
for i=1:1:50
for j=1:1:50

    if p(i,j)~=5

if i==1 && j==1
        p(i,j)=(p(i,j+1)+p(i+1,j))/2;
elseif i==1 && j==50
        p(i,j)=(p(i,j-1)+p(i+1,j))/2;
    elseif i==50 && j==50
        p(i,j)=(p(i,j-1)+p(i-1,j))/2;
    elseif i==50 && j==1
        p(i,j)=(p(i,j+1)+p(i-1,j))/2;
    elseif i==1 && j~=50 && j~=1 p(i,j)=(p(i,j-
        1)+p(i+1,j)+p(i,j+1))/3;
    elseif i==50 && j~=50 && j~=1 p(i,j)=(p(i,j-
        1)+p(i- 1,j)+p(i,j+1))/3;
elseif j==1 && i~=50 && i~=1 p(i,j)=(p(i-
        1,j)+p(i,j+1)+p(i+1,j))/3;
elseif j==50 && i~=50 && i~=1 p(i,j)=(p(i-
        1,j)+p(i,j-1)+p(i+1,j))/3;
    else
        p(i,j)=(p(i-1,j)+p(i,j-1)+p(i+1,j)+p(i,j+1))/4;
end
    end

end
end
%end
q=1;
k=1;
n=zeros(50,50);
for i=1:1:50
    for j=1:1:50
        n(i,j)=q;
        q=q+1;
    end
end
A=zeros(2500,2500);
for i=1:1:50
    for j=1:1:50

            if i==1 && j==1
       if p(i,j+1)~=5
           A(n(i,j),n(i,j+1))=p(i,j+1); end
       if p(i+1,j)~=5
           A(n(i,j),n(i+1,j))=p(i+1,j); end
       if p(i+1,j+1)~=5
           A(n(i,j),n(i+1,j+1))=p(i+1,j+1); end
            elseif i==1 && j==50
```

```
if p(i,j-1)~=5
    A(n(i,j),n(i,j-1))=p(i,j-1); end
if p(i+1,j)~=5
    A(n(i,j),n(i+1,j))=p(i+1,j); end
if p(i+1,j-1)~=5
    A(n(i,j),n(i+1,j-1))=p(i+1,j-1); end
     elseif i==50 && j==50
if p(i,j-1)~=5
    A(n(i,j),n(i,j-1))=p(i,j-1); end
if p(i-1,j)~=5
    A(n(i,j),n(i-1,j))=p(i-1,j); end
if p(i-1,j-1)~=5
    A(n(i,j),n(i-1,j-1))=p(i-1,j-1); end
     elseif i==50 && j==1
if p(i,j+1)~=5
    A(n(i,j),n(i,j+1))=p(i,j+1); end
if p(i-1,j)~=5
    A(n(i,j),n(i-1,j))=p(i-1,j); end
if p(i-1,j+1)~=5
    A(n(i,j),n(i-1,j+1))=p(i-1,j+1); end
     elseif i==1 && j~=50 && j~=1
if p(i,j+1)~=5
    A(n(i,j),n(i,j+1))=p(i,j+1); end
if p(i+1,j)~=5
    A(n(i,j),n(i+1,j))=p(i+1,j); end
if p(i,j-1)~=5
    A(n(i,j),n(i,j-1))=p(i,j-1); end
if p(i+1,j-1)~=5
    A(n(i,j),n(i+1,j-1))=p(i+1,j-1);
end if p(i+1,j+1)~=5
    A(n(i,j),n(i+1,j+1))=p(i+1,j+1); end
     elseif i==50 && j~=50 && j~=1
if p(i,j+1)~=5
    A(n(i,j),n(i,j+1))=p(i,j+1); end
if p(i-1,j)~=5
    A(n(i,j),n(i-1,j))=p(i-1,j); end
if p(i,j-1)~=5
    A(n(i,j),n(i,j-1))=p(i,j-1); end
if p(i-1,j-1)~=5
    A(n(i,j),n(i- 1,j-1))=p(i-1,j-1); end
if p(i-1,j+1)~=5
    A(n(i,j),n(i-1,j+1))=p(i-1,j+1); end
     elseif j==1 && i~=50 && i~=1
if p(i+1,j)~=5
    A(n(i,j),n(i+1,j))=p(i+1,j); end
if p(i,j+1)~=5
    A(n(i,j),n(i,j+1))=p(i,j+1); end
if p(i-1,j)~=5
    A(n(i,j),n(i- 1,j))=p(i-1,j); end
if p(i-1,j+1)~=5
    A(n(i,j),n(i- 1,j+1))=p(i-1,j+1);
end if p(i+1,j+1)~=5
    A(n(i,j),n(i+1,j+1))=p(i+1,j+1); end
     elseif j==50 && i~=50 && i~=1
if p(i+1,j)~=5
    A(n(i,j),n(i+1,j))=p(i+1,j); end
if p(i,j-1)~=5
```

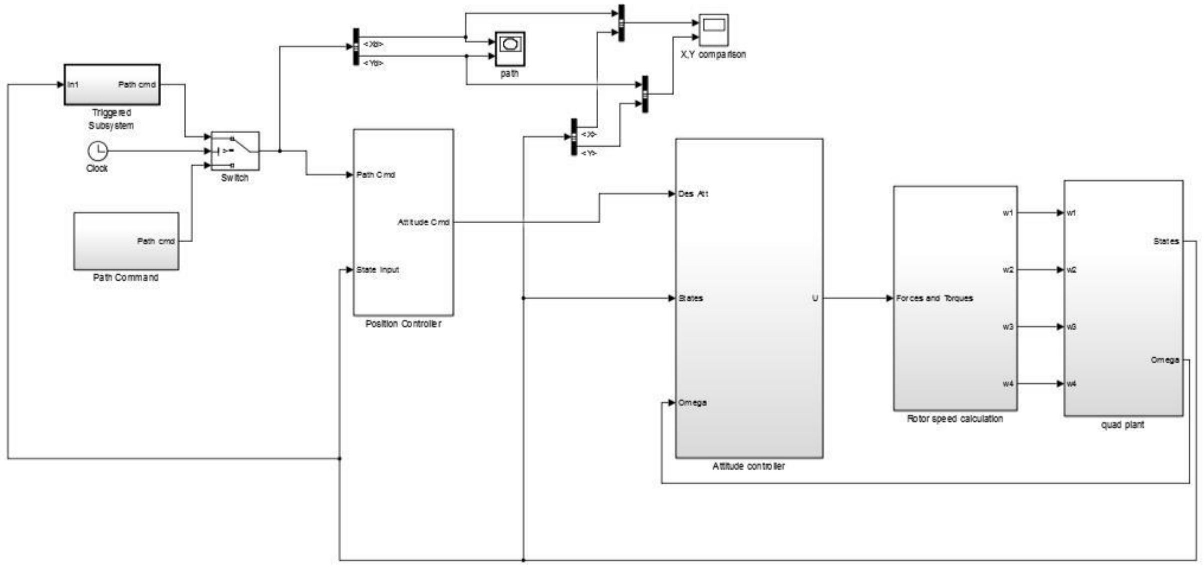
```
           A(n(i,j),n(i,j-1))=p(i,j-1); end
       if p(i-1,j)~=5
           A(n(i,j),n(i- 1,j))=p(i-1,j); end
       if p(i-1,j-1)~=5
           A(n(i,j),n(i- 1,j-1))=p(i-1,j-1);
       end if p(i+1,j-1)~=5
           A(n(i,j),n(i+1,j-1))=p(i+1,j-1); end
            else
       if p(i,j+1)~=5
           A(n(i,j),n(i,j+1))=p(i,j+1); end
       if p(i+1,j)~=5
           A(n(i,j),n(i+1,j))=p(i+1,j); end
       if p(i,j-1)~=5
           A(n(i,j),n(i,j-1))=p(i,j-1); end
       if p(i+1,j-1)~=5
           A(n(i,j),n(i+1,j-1))=p(i+1,j-1); end
        if p(i+1,j+1)~=5
           A(n(i,j),n(i+1,j+1))=p(i+1,j+1); end
        if p(i-1,j)~=5
           A(n(i,j),n(i-1,j))=p(i-1,j); end
        if p(i-1,j-1)~=5
           A(n(i,j),n(i-1,j-1))=p(i-1,j-1); end
        if p(i-1,j+1)~=5
           A(n(i,j),n(i-1,j+1))=p(i-1,j+1); end
            end
    end
end
[cost, t_route] = dijkstra(A,1,2500);
j=ones(50,50);
l=length(t_route);
x=zeros(1,l);
y=zeros(1,l);
for i=1:1:l
    j(t_route(i))=0;
    x(i)=mod(t_ route(i),50);
    if x(i)==0
        x(i)=50;
    end
    y(i)=ceil(t_route(i)/50);
end
x=fliplr(x);
y=fliplr(y);
for i=1:1:10
    x(l+i)=x(l);
    y(l+i)=y(l);
end

w=m;
for i=1:1:l+10
    m(y(i),x(i))=0;
end
p=0;
tim=zeros(1,l+10);
for i=2:1:l+10
    key_x=0;
    key_y=0;
    if x(i)-x(i-1)==1
```

```
        key_x=1;
    end
    if y(i)-y(i-1)==1
        key_y=1;
    end
        if key_x==1&&key_ y==1
            p=p+sqrt(2);
        else
            p=p+1;
        end
        tim(i)=p;
end
  time = 150*tim/tim(l+10) ;
 imshow(m);
    ts_x = timeseries(x,time);
    ts_y = timeseries(y,time);
    path = struct('x',ts_x,'y',ts_y);
    d_path=path;
% save('x_time.mat',path);
```

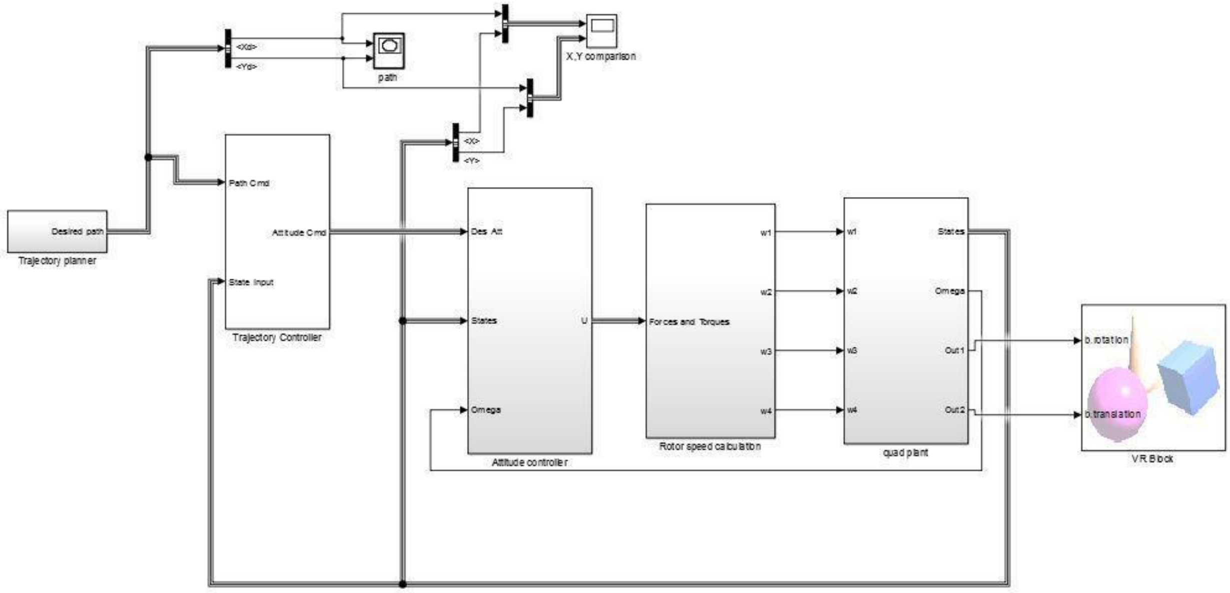
Appendix A14: Trajectory controller



Appendix A15: Complete layout for trajectory control simulation 1



Appendix A16: Complete layout for trajectory control simulation 2


<Xd>
<Yd>
path
X,Y comparison
<X>
<Y>
Path Cmd
Attitude Cmd
State Input
Trajectory Controller
Desired path
Trajectory planner
Des Att
States
U
Omega
Attitude controller
Forces and Torques
w1
w2
w3
w4
Rotor speed calculation
w1
w2
w3
w4
States
Omega
Out1
Out2
quad plant
b.rotation
b.translation
VR Block