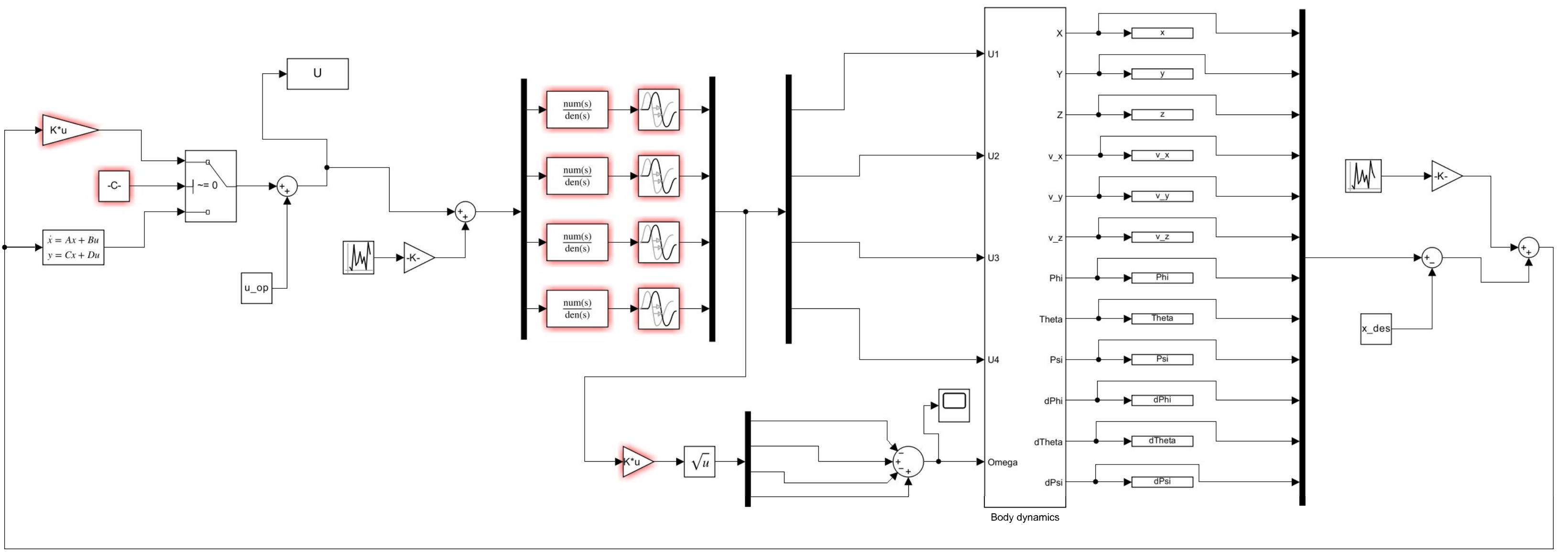
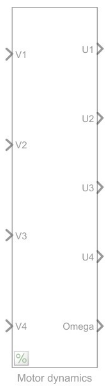
```
%% Quadcopter Robust Control
clear; close all; clc;
% state = [x y z xd yd zd phi th psi phid thd psid]
% control = [u1 u2 u3 u4];


% uncertain parameters
g = 9.807; % m/s^2
m_nom = 2.1; % kg
l_nom = 0.422; % m
tau_nom = 0.001; % s 0.1
gamma_nom = 0.001;
lx = ureal('lx', l_nom, 'Percentage', 30); % m
ly = ureal('ly', l_nom, 'Percentage', 30); % m
m = ureal('m', m_nom, 'Percentage', 30); % kg
Ix = ureal('Ix', 2.1385, 'Percentage', 30); % kg m^2
Iy = ureal('Iy', 2.1385, 'Percentage', 30); % kg m^2
Iz = ureal('Iz', 3.7479, 'Percentage', 30); % kg m^2
tau = ureal('tau', tau_nom, 'Percentage', 30);
gamma = ureal('gamma', gamma_nom, 'Percentage', 30);
% operating point
U1_op = m_nom*g;
psi_op = 0;
phi_op = 0;
th_op = 0;
x_op = [0; 0; 0; 0; 0; 0; phi_op; th_op; psi_op; 0; 0; 0];
u_op = [U1_op; 0; 0; 0];

% linearized dynamics
A = umat(zeros(12, 12));
A(1:3, 4:6) = eye(3);
A(4:5, 7:8) = [sin(psi_op)*U1_op/m, cos(psi_op)*U1_op/m;...
              -cos(psi_op)*U1_op/m, sin(psi_op)*U1_op/m];
A(7:9, 10:12) = eye(3);
B = umat(zeros(12, 4));
B(4, 1) = (phi_op*sin(psi_op) + th_op*cos(psi_op))/m;
B(5, 1) = (-phi_op*cos(psi_op) + th_op*sin(psi_op))/m;
B(6, 1) = 1/m;
B(10:12, 2:4) = diag([lx/Ix; ly/Iy; 1/Iz]);
C = eye(12);
states = {'dx', 'dy', 'dz', 'dxd', 'dyd', 'dzd', 'dphi', 'dth', 'dpsi', 'dphid',↙
'dthd', 'dpsid'};
inputs = {'du1', 'du2', 'du3', 'du4'};
outputs = {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio',↙
'dphido', 'dthdo', 'dpsido'};
P = ss(A, B, C, 0, 'statename', states, 'inputname', inputs, 'outputname', outputs);

% sensor noise
sensor_noise_weights = [0.1, 0.1, 0.1, 0.01, 0.01, 0.01, 0.0873, 0.0873, 0.0873, ↙
0.00873, 0.00873, 0.00873];
Wn = ss(diag(sensor_noise_weights));

% actuator perf
Wc = ss(diag([0.4, 1, 1, 10]));

% actuator lag and time delay
Wact_del = append(tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]), ...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]), ...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]), ...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]));

% external input disturbance
input_disturbance = [2, 0.4, 0.4, 0.05];
Wdist = ss(diag(input_disturbance));

% label block I/O's
Wn.u = 'noise'; Wn.y = 'Wn';
Wc.u = {'u1', 'u2', 'u3', 'u4'}; Wc.y = 'z_act';
Wdist.u = 'dist'; Wdist.y = 'Wdist';
Wact_del.u = {'u1', 'u2', 'u3', 'u4'}; Wact_del.y = {'u11', 'u22', 'u33', 'u44'};

% specify summing junctions
Sum1 = sumblk('%u_dist = %u + Wdist', {'du1', 'du2', 'du3', 'du4'}, {'u11', 'u22',↙
'u33', 'u44'});
Sum2 = sumblk('ymeas = %y + Wn', {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo',↙
'dphio', 'dtho', 'dpsio', 'dphido', 'dthdo', 'dpsido'});
Sum3 = sumblk('err = ymeas - dx_des', 12);

% connect everything
% M = connect(P, Wn, Wc, Wdist, Sum1, Sum2, Sum3,...
%     {'dist', 'noise', 'dx_des', 'u1', 'u2', 'u3', 'u4'}, ...
%     {'z_act', 'err1', 'err2', 'err3', 'err4', 'err5', 'err6', 'err7', 'err8', ↙
'err9', 'err10', 'err11', 'err12'});

M = connect(P, Wn, Wc, Wdist, Wact_del, Sum1, Sum2, Sum3, ...
    {'dist', 'noise', 'dx_des', 'u1', 'u2', 'u3', 'u4'}, ...
    {'z_act', 'err', 'ymeas'});
% M = connect(P, Wn, Wc, Wdist, Sum1, Sum2,...
%     {'dist', 'noise', 'u1', 'u2', 'u3', 'u4'}, ...
%     {'z_act', 'dxm', 'dym', 'dzm', 'dxdm', 'dydm', 'dzdm', 'dphim', 'dthm', ↙
'dpsim', 'dphidm', 'dthdm', 'dpsidm'});


% D-K iteration
disp('Start D-K');
opts = dksynOptions('MixedMU','on');
[k,clp,bnd,dkinfo] = dksyn(M,12,4,opts);
disp('End D-K');
```

```
save('QuadrotorModelCLXPro/K.mat', 'k', 'clp', 'bnd', 'Sum1', 'Sum2', 'Sum3', 'P',↙
'Wn', 'Wc', 'Wdist', 'x_op', 'u_op', 'A', 'B', 'C', 'm', 'Ix', 'Iy', 'Iz', 'g',↙
'm_nom', 'sensor_noise_weights', 'input_disturbance', 'l_nom', 'lx', 'ly', 'tau',↙
'gamma', 'Wact_del', 'dkinfo');

%% Plot

% state = [x y z xd yd zd phi th psi phid thd psid]
% control = [u1 u2 u3 u4];
clear; close all; clc;

load('QuadrotorModelCLXPro/K.mat');


k.InputName = {'err'};
k.OutputName = {'u1', 'u2', 'u3', 'u4'};
% lsim simulate
t = 0:0.1:100;
num = 30;

if num ~= 0
    Parray = usample(P, num);
end
Pnom = P.nom;
yks = zeros(length(t), 16, num+1);
x_des = [0.5; 0.5; 0.5; 0; 0; 0; 0; 0; pi/12; 0; 0; 0]';
for i = 1:num
    cl = connect(Parray(:, :, i), Wn, Wdist, Wact_del, k, Sum1, Sum2, Sum3, {'dist',↙
'noise', 'dx_des'}, {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho',↙
'dpsio', 'dphido', 'dthdo', 'dpsido', 'u1', 'u2', 'u3', 'u4'});
    U = normrnd(0, 1, length(t), 16)./10;
    U = [U, repmat(x_des, length(t), 1)];
    % shift by operating point
    yks(:, :, i) = lsim(cl, U, t) + [x_op', u_op'];
end
% nominal model and no noise
cl = connect(Pnom, Wn, Wdist, Wact_del, k, Sum1, Sum2, Sum3, {'dist', 'noise',↙
'dx_des'}, {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio',↙
'dphido', 'dthdo', 'dpsido', 'u1', 'u2', 'u3', 'u4'});
U = normrnd(0, 1, length(t), 16).*0;
U = [U, repmat(x_des, length(t), 1)];
yks(:, :, end) = lsim(cl, U, t) + [x_op', u_op'];


% plot 3D
figure; hold on;
for i = 1:num
    samp = plot3(yks(:, 1, i), yks(:, 2, i), yks(:, 3, i), '-', 'Color',  [0↙
0.4470    0.7410], 'LineWidth', 1);
end
nom = plot3(yks(:, 1, end), yks(:, 2, end), yks(:, 3, end), '-', 'Color',  [0.8500↙
0.3250    0.0980], 'LineWidth', 3);
grid on; box on; axis equal;
xlabel('$$X\ [m]$$', 'interpreter', 'latex');
ylabel('$$Y\ [m]$$', 'interpreter', 'latex');
zlabel('$$Z\ [m]$$', 'interpreter', 'latex');
if num ~= 0
    legend([nom, samp], 'Nominal', 'Monte Carlo Samples');
else
    legend(nom, 'Nominal');
end

% plot control inputs
choose = 1;
figure;
subplot(4,1,1);
plot(t, yks(:, 13, choose));
ylabel('$$U_1$$', 'interpreter', 'latex');
grid on; box on;
subplot(4,1,2);
plot(t, yks(:, 14, choose));
grid on; box on;
ylabel('$$U_2$$', 'interpreter', 'latex');
subplot(4,1,3);
plot(t, yks(:, 15, choose));
grid on; box on;
ylabel('$$U_3$$', 'interpreter', 'latex');
subplot(4,1,4);
plot(t, yks(:, 16, choose));
grid on; box on;
ylabel('$$U_4$$', 'interpreter', 'latex');

% plot states
figure;
for choose = 1:num
    subplot(3,2,1); hold on;
    plot(t, yks(:, 1, choose), '-', 'Color',  [0    0.4470    0.7410], 'LineWidth',↙
1);
    plot([t(1), t(end)], [x_des(1), x_des(1)], 'k-');
    grid on; box on;
    ylabel('$$X\ [m]$$', 'interpreter', 'latex');
    subplot(3,2,3); hold on;
    plot(t, yks(:, 2, choose), '-', 'Color',  [0    0.4470    0.7410], 'LineWidth',↙
1);
    plot([t(1), t(end)], [x_des(2), x_des(2)], 'k-');
    grid on; box on;
    ylabel('$$Y\ [m]$$', 'interpreter', 'latex');
    subplot(3,2,5); hold on;
    plot(t, yks(:, 3, choose), '-', 'Color',  [0    0.4470    0.7410], 'LineWidth',↙
1);
    plot([t(1), t(end)], [x_des(3), x_des(3)], 'k-');
    grid on; box on;
    ylabel('$$Z\ [m]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

    subplot(3,2,2); hold on;
    sample = plot(t, rad2deg(yks(:, 7, choose)), '-', 'Color',  [0    0.4470↙
0.7410], 'LineWidth', 1);
    target = plot([t(1), t(end)], rad2deg([x_des(7), x_des(7)]), 'k-');
    grid on; box on;
    ylabel('$$\phi\ [^\circ]$$', 'interpreter', 'latex');
    subplot(3,2,4); hold on;
    plot(t, rad2deg(yks(:, 8, choose)), '-', 'Color',  [0    0.4470    0.7410],↙
'LineWidth', 1);
    plot([t(1), t(end)], rad2deg([x_des(8), x_des(8)]), 'k-');
    grid on; box on;
    ylabel('$$\theta\ [^\circ]$$', 'interpreter', 'latex');
    subplot(3,2,6); hold on;
    plot(t, rad2deg(yks(:, 9, choose)), '-', 'Color',  [0    0.4470    0.7410],↙
'LineWidth', 1);
    plot([t(1), t(end)], rad2deg([x_des(9), x_des(9)]), 'k-');
    grid on; box on;
    ylabel('$$\psi\ [^\circ]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');
end
% plot nominal
subplot(3,2,1); hold on;
plot(t, yks(:, 1, end), '-', 'Color',  [0.8500    0.3250    0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(1), x_des(1)], 'k-');
grid on; box on;
    ylabel('$$X\ [m]$$', 'interpreter', 'latex');

subplot(3,2,3); hold on;
plot(t, yks(:, 2, end), '-', 'Color',  [0.8500    0.3250    0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(2), x_des(2)], 'k-');
grid on; box on;
    ylabel('$$Y\ [m]$$', 'interpreter', 'latex');

subplot(3,2,5); hold on;
plot(t, yks(:, 3, end), '-', 'Color',  [0.8500    0.3250    0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(3), x_des(3)], 'k-');
grid on; box on;
    ylabel('$$Z\ [m]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

subplot(3,2,2); hold on;
```

```
nom = plot(t, rad2deg(yks(:, 7, end)), '-', 'Color',  [0.8500    0.3250    0.0980],↙
'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(7), x_des(7)]), 'k-');
if num ~= 0
    legend([nom, sample, target], 'Nominal', 'Monte Carlo Samples', 'Target');
else
    legend([nom, target], 'Nominal', 'Target');
end
grid on; box on;
    ylabel('$$\phi\ [^\circ]$$', 'interpreter', 'latex');

subplot(3,2,4); hold on;
plot(t, rad2deg(yks(:, 8, end)), '-', 'Color',  [0.8500    0.3250    0.0980],↙
'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(8), x_des(8)]), 'k-');
grid on; box on;
    ylabel('$$\theta\ [^\circ]$$', 'interpreter', 'latex');

subplot(3,2,6); hold on;
plot(t, rad2deg(yks(:, 9, end)), '-', 'Color',  [0.8500    0.3250    0.0980],↙
'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(9), x_des(9)]), 'k-');
grid on; box on;
    ylabel('$$\psi\ [^\circ]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

% plot bounds
figure;
opts = bodeoptions('cstprefs');
opts.Grid = 'on';
opts.MagUnits = 'abs';
bodemag(dkinfo{1}.MussvBnds(1,1),opts);
grid on; box on;
title('Structured Singular Value');
```


U
K*u
-C-
~= 0
$\dot{x} = Ax + Bu$
$y = Cx + Du$
u_op
-K-
num(s)/den(s)
num(s)/den(s)
num(s)/den(s)
num(s)/den(s)
K*u
$\sqrt{u}$
U1
U2
U3
U4
Omega
X
Y
Z
v_x
v_y
v_z
Phi
Theta
Psi
dPhi
dTheta
dPsi
x
y
z
v_x
v_y
v_z
Phi
Theta
Psi
dPhi
dTheta
dPsi
Body dynamics
-K-
x_des

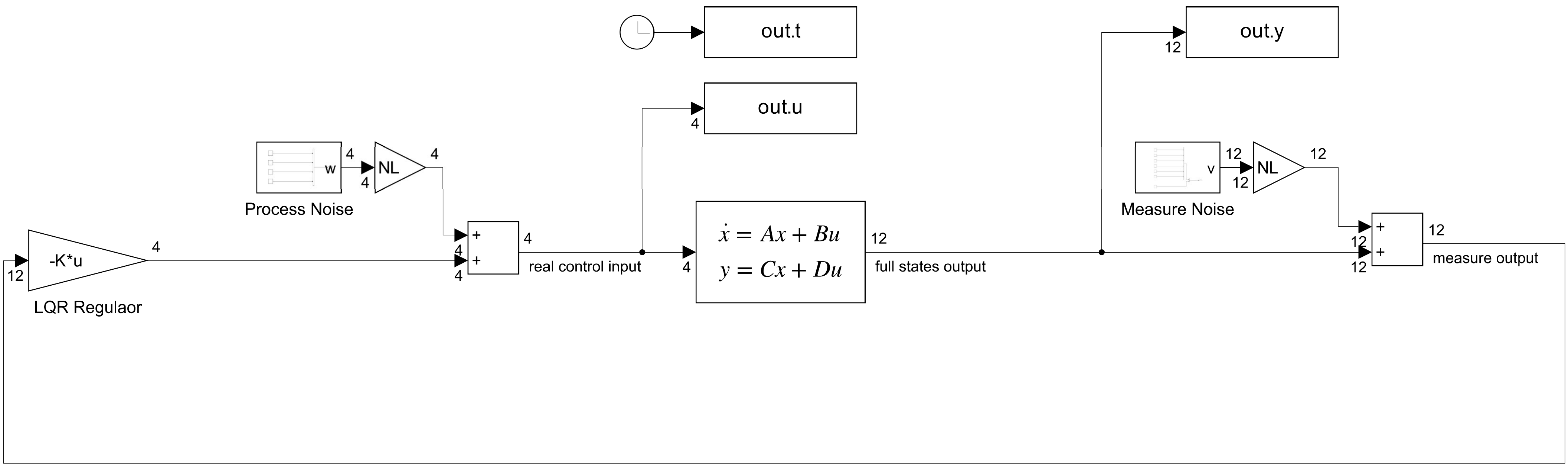

V1
V2
V3
V4
U1
U2
U3
U4
Omega
Motor dynamics

```
%% LQR control
%% Setup Drone
m = 0.2;
I = [[0.1,0,0];[0,0.1,0];[0,0,0.15]];

% sample time
ts = 0.01;

% Initial States
Euler_0 = [0;0;0];
XYZ_0 = [0;0;0];
body_rate_0 = [0;0;0]; % initial pqr

% Environment (North-East-Down coordinate)
g = [0;0;9.8];

%% Linear Model
% Option 1: Load default "Full States" Linear Model
load('LinearModel1');

% Option 2: Get Linear Model from Simulink model.
% model = 'DroneControl_Linear1';
% opspec = operspec(model);
%
% % Specify operating states: Z position
% opspec = addoutputspec(opspec,'DroneControl_Linear1/6dof_system',3);
% opspec.Outputs(1).Known = true;
% opspec.Outputs(1).y = 0;
%
% % Find Operating point
% op1 = findop(model,opspec);
%
% % Get IO from simulink
% io = getlinio(model);
% % Set Inputs
% io(1) = linio('DroneControl_Linear1/Thrust',1,'input');
% io(2) = linio('DroneControl_Linear1/M_roll',1,'input');
% io(3) = linio('DroneControl_Linear1/M_pitch',1,'input');
% io(4) = linio('DroneControl_Linear1/M_yaw',1,'input');
%
% io(5) = linio('DroneControl_Linear1/6dof_system',1,'output');
% io(6) = linio('DroneControl_Linear1/6dof_system',2,'output');
% io(7) = linio('DroneControl_Linear1/6dof_system',3,'output');
% io(8) = linio('DroneControl_Linear1/6dof_system',4,'output');
% io(9) = linio('DroneControl_Linear1/6dof_system',5,'output');
% io(10) = linio('DroneControl_Linear1/6dof_system',6,'output');
% io(11) = linio('DroneControl_Linear1/6dof_system',7,'output');
% io(12) = linio('DroneControl_Linear1/6dof_system',8,'output');
% io(13) = linio('DroneControl_Linear1/6dof_system',9,'output');
% io(14) = linio('DroneControl_Linear1/6dof_system',10,'output');
% io(15) = linio('DroneControl_Linear1/6dof_system',11,'output');
% io(16) = linio('DroneControl_Linear1/6dof_system',12,'output');
%
% % Linearize
% sys = linearize(model,op1,io);


%% LQR regulator
% Set Q and R. Q: Performance Cost; R: Control Cost.

Q = sys.C'*sys.C;
R = eye(4);

%% Get K controller gain
K = lqr(sys.A,sys.B,Q,R);

%% Noise Charactersitic
% Process noise covariance
Qn = 1e-03*diag([0 0 0 0 0 0 1 1 1 1 1 1]);
% Measure noise covariance
Rn = 1e-04;

% Noise Level (0: no noise, 1: with noise)
NL = 0; %(default:0)

%% Simulation

% Task A or Task B (1: Task A, 2: Task B)
Task = 1;

if Task == 1
    % Specify initial states (NED Coordinate)
    x0 = zeros(12,1);
    x0(4) = 1;     % Initial X position
    x0(5) = 1.5;     % Initial Y position
    x0(6) = 2;     % Initial Z position

    % simulation time
    stime = 8;
    % Run LQG.slx Simulink Model.
    out = sim('LQR',stime);

    t = out.t;
    u = out.u;
    y = out.y;
    viewlimit = 2;

else
```

```
    % pecify initial states (NED Coordinate)
    x0 = zeros(12,1);

    % simulation time
    stime = 30;

    % Load WayPts data
    load('WayPts_Test.mat');

    % Generate RefX,Y,Z signal
    XYZsignal;

    % Run LQG.slx Simulink Model.
    out = sim('LQR_Test',stime);

    t = out.t;
    u = out.u;
    y = out.y;
    viewlimit = 3.5;
end

%% 3D Animation

animation_LQR;

%% 2D Plot Performance, Control Effort

% Briefly estimate Battery power cost as inputs^2.
%Power = u(:,1).*u(:,1) + u(:,2).*u(:,2) + u(:,3).*u(:,3) + u(:,4).*u(:,4);
Power = 0*t;
for i = 1:length(t)
    if i == 1
        Power(i) = abs(y(i,7) - 0) + abs(y(i,8) - 0) + abs(y(i,9) - 0) + abs(y(i,10) ↙
- 0) + abs(y(i,11) - 0) + abs(y(i,12) - 0);
    else
        Power(i) = abs(y(i,7)-y(i-1,7)) + abs(y(i,8)-y(i-1,8)) + abs(y(i,9)-y(i-1,9)) ↙
+ ...
                   abs(y(i,10)-y(i-1,10)) + abs(y(i,11)-y(i-1,11)) + abs(y(i,12)-y(i- ↙
1,12))  ;
    end
end


figure;
set(gcf,'position',[0,0,1200,600])
sub1=subplot(3,3,1);
axis([0 stime min(y(:,4)) max(y(:,4))]);
ylabel('X')
grid on
```

```
ani1=animatedline('Color','b','LineWidth',2);

subplot(3,3,4)
axis([0 stime min(y(:,5)) max(y(:,5))]);
ylabel('Y')
grid on
ani2=animatedline('Color','g','LineWidth',2);

subplot(3,3,7)
axis([0 stime min(y(:,6)) max(y(:,6))]);
ylabel('Z')
grid on
ani3=animatedline('Color','r','LineWidth',2);

subplot(3,3,3)
axis([0 stime min(u(:,1)) max(u(:,1))]);
ylabel('Thrust (Perturbation)')
grid on
ani4=animatedline('Color','black','LineWidth',2);

subplot(3,3,2)
axis([0 stime min(u(:,2)) max(u(:,2))]);
ylabel('M roll')
grid on
ani6=animatedline('Color','black','LineWidth',2);

subplot(3,3,5)
axis([0 stime min(u(:,3)) max(u(:,3))]);
ylabel('M pitch')
grid on
ani7=animatedline('Color','black','LineWidth',2);

subplot(3,3,8)
axis([0 stime min(u(:,4)) max(u(:,4))]);
ylabel('M yaw')
grid on
ani8=animatedline('Color','black','LineWidth',2);


subplot(3,3,6)
axis([0 stime min(Power) max(Power)]);
ylabel('Power (Perturbation)')
grid on
ani5=animatedline('Color','black','LineWidth',2);

pause(1);

current_time = 0;
Energy = 0;
```

```
% text1 = annotation('textbox',[0.15 0.65 0.3 0.15],'String',{ 'Energy ',[' =' ↙
num2str(Energy)] } );
text(0, -0.5, 'Time:', 'FontSize', 14, 'HorizontalAlignment',↙
'left','Units','normalized');
TME = text(0.5, -0.5, num2str(current_time, '%.1f'), 'FontSize', 14,↙
'HorizontalAlignment', 'left','Units','normalized');
text(0, -1, 'Energy:', 'FontSize', 14, 'HorizontalAlignment',↙
'left','Units','normalized');
ENG = text(0.5, -1, num2str(Energy, '%.2f'), 'FontSize', 14, 'HorizontalAlignment',↙
'left','Units','normalized');

%%

for k=1:length(t)
    addpoints(ani1,t(k),y(k,4));
    addpoints(ani2,t(k),y(k,5));
    addpoints(ani3,t(k),y(k,6));
    addpoints(ani4,t(k),u(k,1));
    addpoints(ani5,t(k),Power(k,1));
    addpoints(ani6,t(k),u(k,2));
    addpoints(ani7,t(k),u(k,3));
    addpoints(ani8,t(k),u(k,4));

    current_time = num2str(t(k), '%.1f');
    Energy = Energy + 100*abs(Power(k))*ts;
    TME.String = current_time;
    ENG.String = num2str(Energy, '%.1f');

    drawnow
    pause(0.01);
end
```



x_dot = A* x + B * u
y_dot = C*x + D * u

NL: 0 (by default)

```
getQuadrotorDynamicsAndJacobian;
```

```
ans = function_handle with value:
    @QuadrotorStateFcn
ans = function_handle with value:
    @QuadrotorStateJacobianFcn
```

```
nx = 12;
ny = 12;
nu = 4;
nlobj = nlmpc(nx, ny, nu);
```

```
In standard cost function, zero weights are applied by default to one or more OVs because
there are fewer MVs than OVs.
```

```
nlobj.Model.StateFcn = "QuadrotorStateFcn";


nlobj.Jacobian.StateFcn = @QuadrotorStateJacobianFcn;
```

```
rng(0)
validateFcns(nlobj,rand(nx,1),rand(nu,1));
```

```
Model.StateFcn is OK.
Jacobian.StateFcn is OK.
No output function specified. Assuming "y = x" in the prediction model.
Analysis of user-provided model, cost, and constraint functions complete.
```

```
Ts = 0.1;
p = 18;
m = 2;
nlobj.Ts = Ts;
nlobj.PredictionHorizon = p;
nlobj.ControlHorizon = m;
nlobj.MV = struct('Min',{0;0;0;0},'Max',{12;12;12;12});
nlobj.Weights.OutputVariables = [1 1 1 1 1 1 0 0 0 0 0 0];
nlobj.Weights.ManipulatedVariables = [0.1 0.1 0.1 0.1];
nlobj.Weights.ManipulatedVariablesRate = [0.1 0.1 0.1 0.1];

% Specify the initial conditions
x = [7;-10;0;0;0;0;0;0;0;0;0;0];
% Nominal control that keeps the quadrotor floating
nloptions = nlmpcmoveopt;
nloptions.MVTarget = [4.9 4.9 4.9 4.9];
mv = nloptions.MVTarget;
Duration = 20;
hbar = waitbar(0,'Simulation Progress');
xHistory = x';
lastMV = mv;
uHistory = lastMV;
for k = 1:(Duration/Ts)
    % Set references for previewing
    t = linspace(k*Ts, (k+p-1)*Ts,p);
    yref = QuadrotorReferenceTrajectory(t);
    % Compute the control moves with reference previewing.
    xk = xHistory(k,:);
```

```matlab
    [uk,nloptions,info] = nlmpcmove(nlobj,xk,lastMV,yref',[],nloptions);
    uHistory(k+1,:) = uk';
    lastMV = uk;
    % Update states.
    ODEFUN = @(t,xk) QuadrotorStateFcn(xk,uk);
    [TOUT,YOUT] = ode45(ODEFUN,[0 Ts], xHistory(k,:)');
    xHistory(k+1,:) = YOUT(end,:);
    waitbar(k*Ts/Duration,hbar);
end
close(hbar)
plotQuadrotorTrajectory;
```

```matlab
animateQuadrotorTrajectory;
```

This is for Trajectory Graph

```
% This script plots the closed-loop responses of the
nonlinear MPC
% controller used in the quadrotor path following
example.

% Copyright 2019 The MathWorks, Inc.

% Plot the closed-loop response.
time = 0:Ts:Duration;
yreftot = QuadrotorReferenceTrajectory(time)';

% Plot the states.
figure('Name','States')

subplot(2,3,1)
hold on
plot(time,xHistory(:,1))
plot(time,yreftot(:,1))
grid on
xlabel('time')
ylabel('x')
legend('actual','reference','Location','southeast')
title('Qruadrotor x position')

subplot(2,3,2)
hold on
plot(time,xHistory(:,2))
plot(time,yreftot(:,2))
grid on
xlabel('time')
ylabel('y')
legend('actual','reference','Location','southeast')
title('Qruadrotor y position')

subplot(2,3,3)
hold on
plot(time,xHistory(:,3))
plot(time,yreftot(:,3))
grid on
xlabel('time')
ylabel('z')
legend('actual','reference','Location','southeast')
title('Qruadrotor z position')

subplot(2,3,4)
```

```
hold on
plot(time,xHistory(:,4))
plot(time,yreftot(:,4))
grid on
xlabel('time')
ylabel('phi')
legend('actual','reference','Location','southeast')
title('Qruadrotor phi angle')

subplot(2,3,5)
hold on
plot(time,xHistory(:,5))
plot(time,yreftot(:,5))
grid on
xlabel('time')
ylabel('theta')
legend('actual','reference','Location','southeast')
title('Qruadrotor theta angle')

subplot(2,3,6)
hold on
plot(time,xHistory(:,6))
plot(time,yreftot(:,6))
grid on
xlabel('time')
ylabel('psi')
legend('actual','reference','Location','southeast')
title('Qruadrotor psi angle')

% Plot the manipulated variables.
figure('Name','Control Inputs')

subplot(2,2,1)
hold on
stairs(time,uHistory(:,1))
ylim([-0.5,12.5])
plot(time,nloptions.MVTarget(2)*ones(1,length(time)))
grid on
xlabel('time')
legend('actual','reference')
title('Input 1')

subplot(2,2,2)
hold on
stairs(time,uHistory(:,2))
ylim([-0.5,12.5])
plot(time,nloptions.MVTarget(2)*ones(1,length(time)))
```

```
grid on
xlabel('time')
title('Input 2')
legend('actual','reference')

subplot(2,2,3)
hold on
stairs(time,uHistory(:,3))
ylim([-0.5,12.5])
plot(time,nloptions.MVTarget(2)*ones(1,length(time)))
grid on
xlabel('time')
title('Input 3')
legend('actual','reference')

subplot(2,2,4)
hold on
stairs(time,uHistory(:,4))
ylim([-0.5,12.5])
plot(time,nloptions.MVTarget(2)*ones(1,length(time)))
grid on
xlabel('time')
title('Input 4')
legend('actual','reference')
```

For Animation

```
figure('Name','QuadrotorAnimation');
hold on
for i=1:size(xHistory,1)
    clf;
    animateQuadrotor(time(i), xHistory(i,:));
    pause(Ts);
end
```