ECEN 5413
OPTIMAL CONTROL
Instructor: Dr. Martin Hagan

Ball and Beam Model Final Project Report

Prepared by: Srinivas Sambaraju
May 08, 2015

Introduction

The project involves studying various solution approaches in a classical Ball and Beam control system model. The system is inherently unstable without control but as long as it is controllable and observable the system can be stabilized. The Simulink model simulates a non-linear open loop response and a linearized Simulink model is generated to analyze the system.

An optimal controller is designed using the (Linear Quadratic Regulator) LQR solution. Finally an observer is designed by pole positioning followed by an optimal observer that is coupled with the optimal controller. These various solutions are analyzed and compared to get an understanding of the problem. This problem is similar to many real world systems like stabilizing an aircraft during turbulence etc.

Model of the Ball and Beam control system

In this model, a beam is attached to a DC motor by a shaft and the beam rotates clockwise or anti clockwise depending the voltage. The voltage range is + or – 12 volts. A ball is placed on the beam and the general tendency of the ball is to roll off the beam when an external disturbance is applied on the ball. The control system should make sure that the ball does not roll off the beam as long as the disturbance is within the design parameters of the system. Four non-linear state equations are provided after balancing the forces on the system. These equations are further linearized so that state space modeling can be used to design the control system.

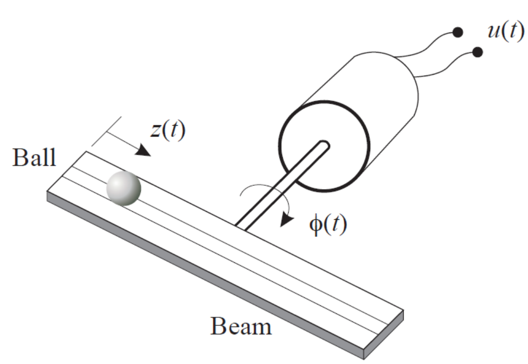


Fig 1. Ball and Beam system

States

x_1 = Beam velocity

x_2 = Beam Angle(∅)

x_3 = Ball Velocity

x_4 = Ball Position (z)

u = Motor Voltage

τ = Torque

$$\dot{x_1} = \frac{\tau - mgcos(x_2)x_4 - 2mx_1x_3x_4}{J_{beam} + J_{ball} + mx_4^2}$$

$$\tau = \frac{K_t}{R_a}(u - K_b x_1)$$

$$\dot{x_2} = x_1$$

$$\dot{x_3} = \frac{mx_1^2x_4 - mgsin(x_2)}{\frac{J_{ball}}{r^2} + m}$$

$$\dot{x_4} = x_3$$

The constants used in the equations are given below.

K_b = 0.0269 (Back EMF)

K_t = 0.025 (Torque constant)

R_a = 13.5 ohms (Armature Resistance)

m = 0.0027 Kg (Ball Mass)

r = 0.02 m (Ball Radius)

g = 9.81 $^{m}/_{s^2}$ (Gravitational constant)

J_{ball} = 7.2 x 10^{-7} Kg – m^2 (Ball Inertia)

J_{beam} = 6.8129 x 10^{-5} Kg – m^2 (Beam Inertia)

Non Linear Simulink model

The differential equations were solved in Simulink using the non-linear model and a schematic of the model is included below. On clicking on the scope, the states – beam angle and ball position are displayed and we can observe that the Beam Angle state goes to positive infinity and the Ball position goes to negative infinity which suggests that the non-linear system is unstable in open loop. The ball falls off the beam.

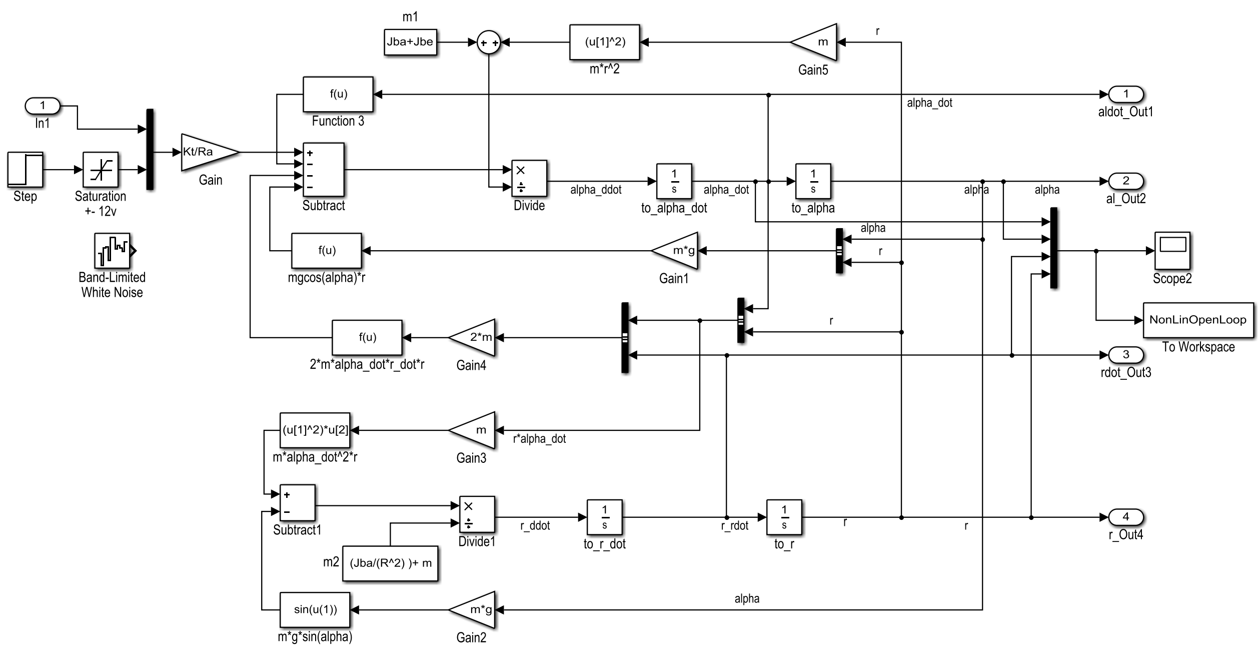
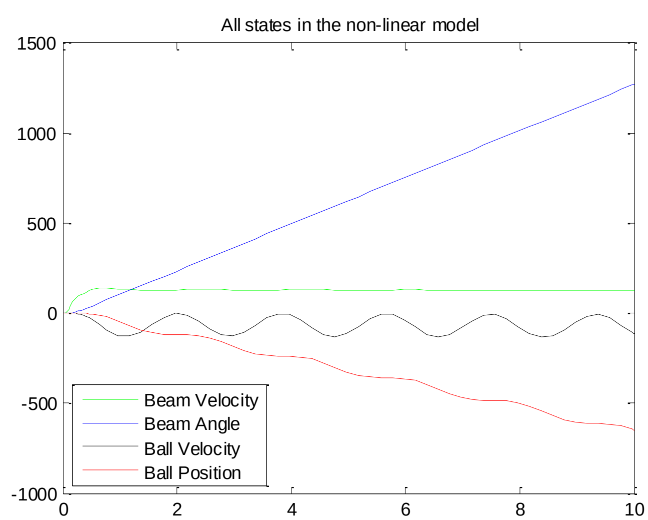


Fig. Simulink model of the non-linear system



Linear Simulink model

The ***linmod*** command is used in Matlab to get a linearized model of the Simulink system. The A, B, C and D matrices were obtained and used for linear analysis. Comparing this open loop linear model with the response of the non-linear Simulink model from above, the response is similar and the states go to infinity and a plot of the states is similar.

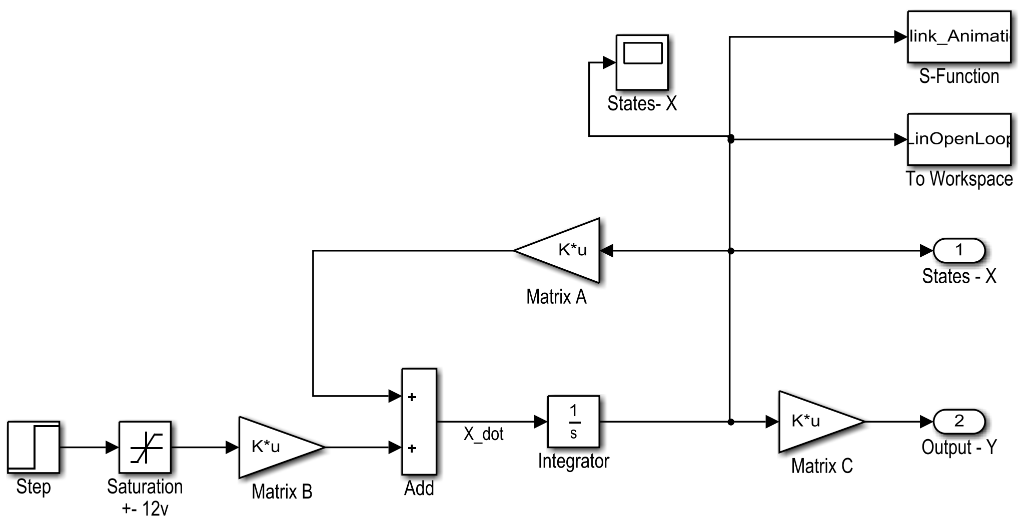
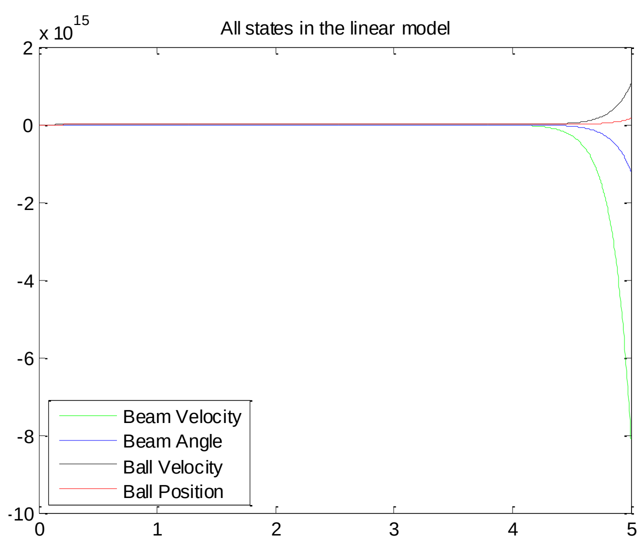


Fig. Simulink model of the linear system



Model Linearized at equilibrium point

By linearizing the model at the equilibrium point where all the states and input were equal to zero, the A, B, C and D matrices were deduced and the numeric value of the matrices matches with those obtained by the Matlab ***linmod*** command. Below are the A, B, C and D matrices after linearization.

$$A = [\left(\frac{-K_t K_b}{R_a(J_{beam} + J_{ball})}\right) 0\ 0\ \left(\frac{-mg}{J_{beam} + J_{ball}}\right); 1\ 0\ 0\ 0; 0 \left(\frac{-mg}{\frac{J_{ball}}{r^2} + m}\right) 0\ 0; 0\ 0\ 10]$$

$$B = [\frac{K_t}{R_a(J_{beam} + J_{ball})}\ 0\ 0\ 0]$$

$$C = [0\ 1\ 0\ 0; 0\ 0\ 0\ 1]$$

$$J_x = A\delta x + B\delta u$$

$$J_y = C\delta x + D\delta u$$

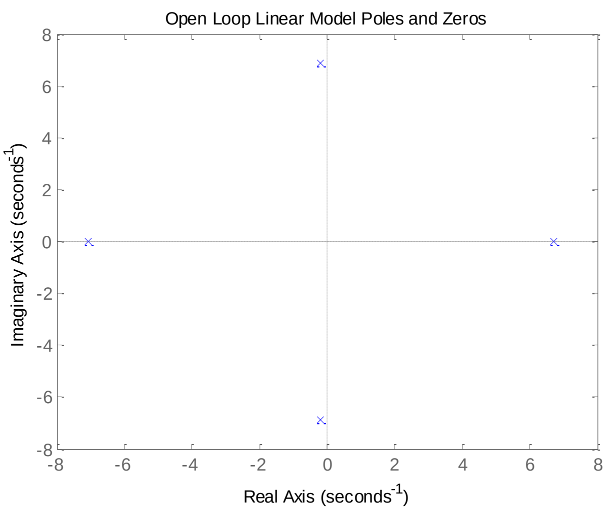
These are the matrices with values.

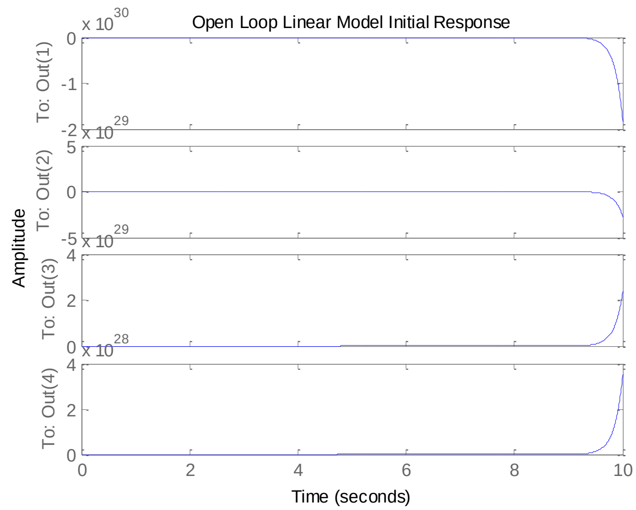
```
A_lin = [ -0.7235   0        0       -384.7115
           1.0000   0        0          0
           0       -5.8860   0          0
           0        0        1.0000     0 ]
B_lin = [ 26.8973;  0;       0;         0 ]
C_lin = [  1  0  0  0;  0  1  0  0;  0  0  1  0;  0  0  0  1 ]
D_lin = [ 0 ]
```

Stability of the system

The first step to understand the stability of the system is to generate a pole zero map from the state space model of the linear system. It shows that there is a pole in the right half plane ($e^{+ve\ value}$) and that explains why the system is unstable in open loop.

From the Simulink non-linear model and the linearized model's it is clear that the ball and beam system in the current state is open loop unstable. Stability guarantees that the system output remains finite if the input is finite. So to be able to stabilize the system in closed loop, we need to check if the system is controllable and observable.





Controllability

A system is controllable if by means of input it can be transferred from an initial state to any other state in a finite time. From Matlab command (ctrb) we can confirm the system is Controllable as the Matrix [B AB] has full rank.

Observability

A system is observable if its state at any time can be determined from the knowledge of the input and the output over a finite period of time. From Matlab command (obsv) we can confirm the system is Observable as the Matrix [C; CA] has full rank.

Therefore the system can be stabilized in closed loop.

Optimal state variable feedback controller

For this case it is assumed that all states are measurable but some iteration should be involved in the choice of performance index. The performance index to minimize is

$$J = \int (Z^T R3Z + U^T R2U)\, dt$$
$$Z = DX \; and \;\; Z^T R3Z = X^T D^T R3DX$$
$$if \; D^T R3D = R1 \;\; Z^T R3Z = X^T R1X$$

The final performance index to minimize is given as below.

$$J = \int (X^T R1X + U^T R2U)\, dt$$

The controller is designed using the LQR command in Matlab which solves the steady state Ricatti equation. The gain was obtained by iterating with different values of R2. R1 was fixed by using weights on x2 and x4 states. The input (u) was also checked if it was within + and -12 volts. Since Lyapunov equation will not be used, the mean square error $E[X^2]$ and mean square input $E[U^2]$ were not calculated.

Using different values of R1 and R2 iteratively a performance index in terms of R1 and R2 was chosen and checked if poles are moving to the left half plane while input (u) is still within limits.

```
K_1 = lqr(A, B, R1, R2)
R1 = [ 0 0 0 0; 0 1 0  0; 0 0 0 0; 0 0 0 2 ]
R2 = .004
```

R1 was chosen such that approximately a 1 degree error in beam angle is equal to 2 cm error in the ball position. The max beam angle is 90 degrees and the max beam length is 1 meter. Using the ratio of r11/r22, pi^2/4 ~= 2.5. This value was iterated and finally a value of 2 was chosen. There could be an error in these assumption and the ratio may be updated in future in a more thorough analysis. A value of performance index was also calculated in Matlab along with the input plot, J was equal to **9.5644** for the optimal controller. This value was lowest among the multiple iterations using R1 and R2. The ball will remain on the beam in this case.

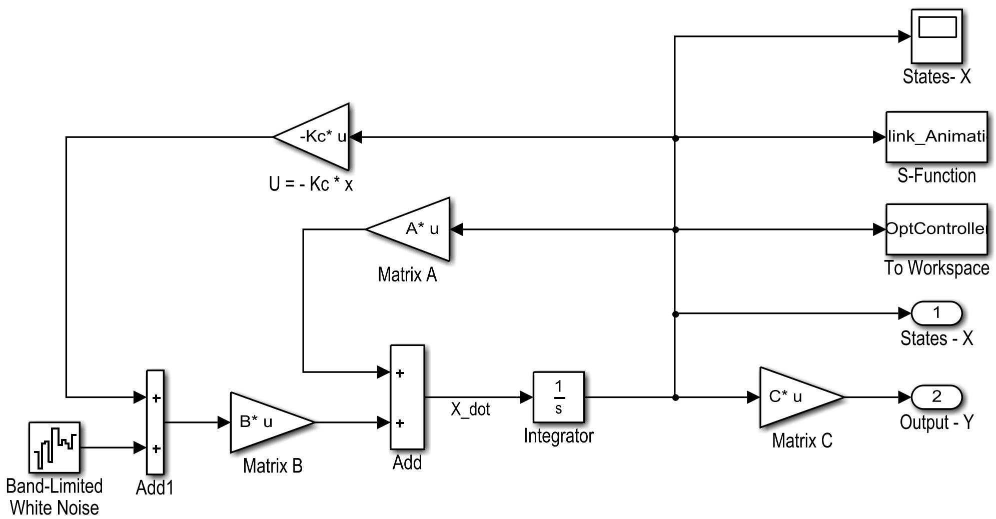
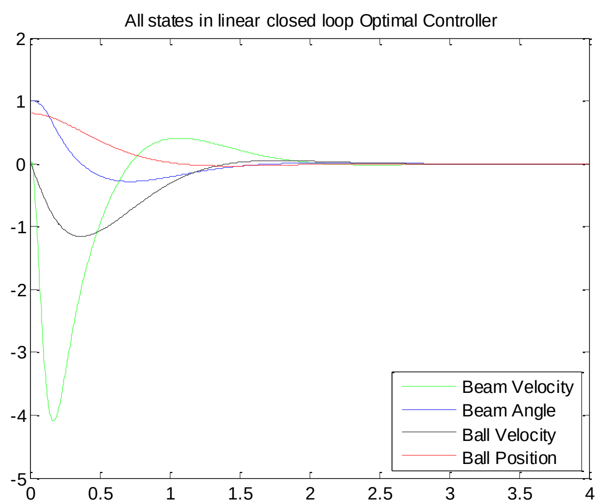
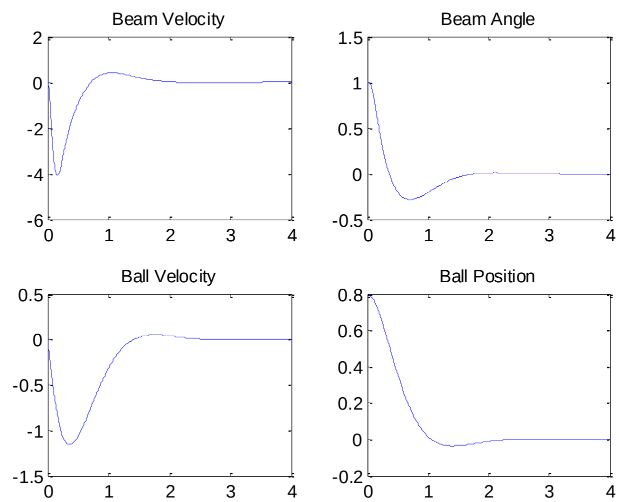
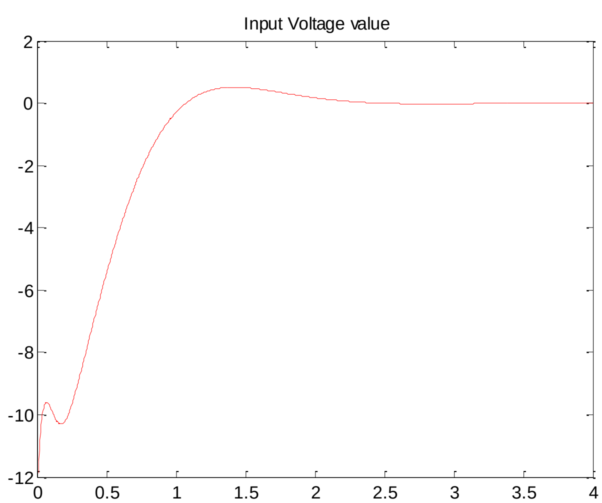
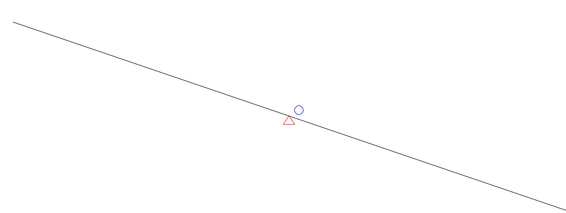


Fig. Simulink model of the linear optimal controller

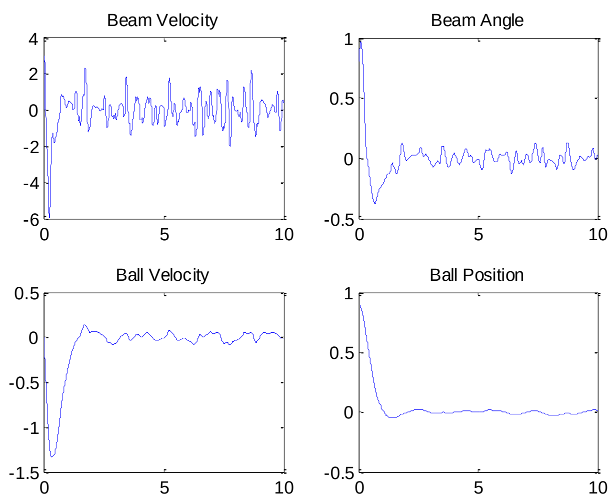


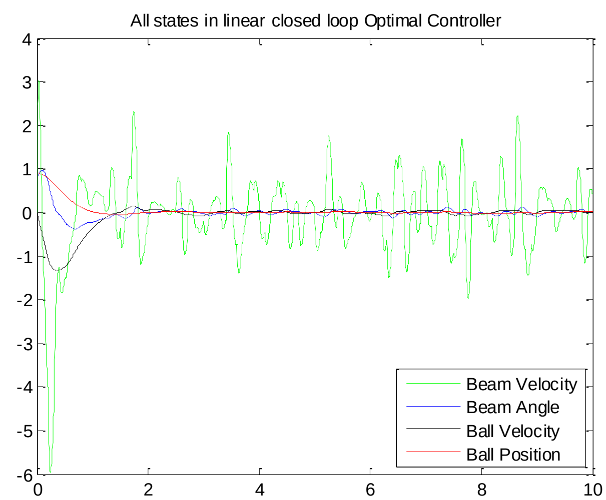


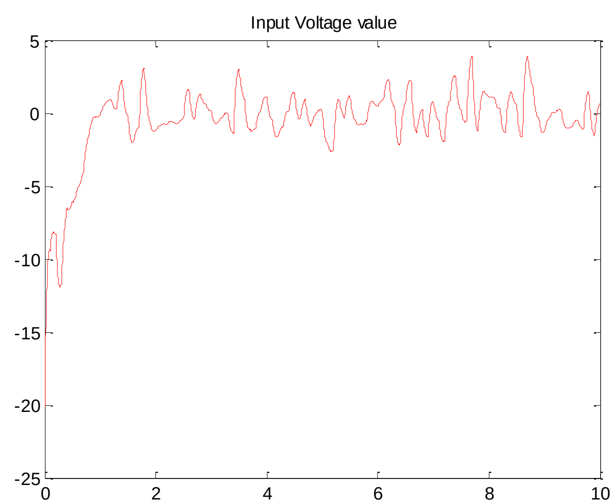




The plots of the states and input with noise added are displayed below. The Beam velocity and angle states had lot of oscillation and were not driven to zero because of the noise. Depending on the steady state error requirements, the controller may have to be deigned to handle noise more effectively.







Output Feedback: using an Observer to estimate the states

In this section an observer is used to estimate the states. The observer design is explained and the results of the simulations are also shown. Further the optimal observer design is compared to the pole positioning method. For this case, only the outputs (ball position and beam angle) are measurable. The Observer equations are given below.

$$\dot{x} = Ax + Bu$$
$$\dot{\hat{x}} = A\hat{x} + Bu + K_o\,(y - \hat{y}\,)$$
$$u = -K_c * x$$
$$y = Cx$$
$$\hat{y} = C\hat{x}$$
$$e = x - \hat{x}$$

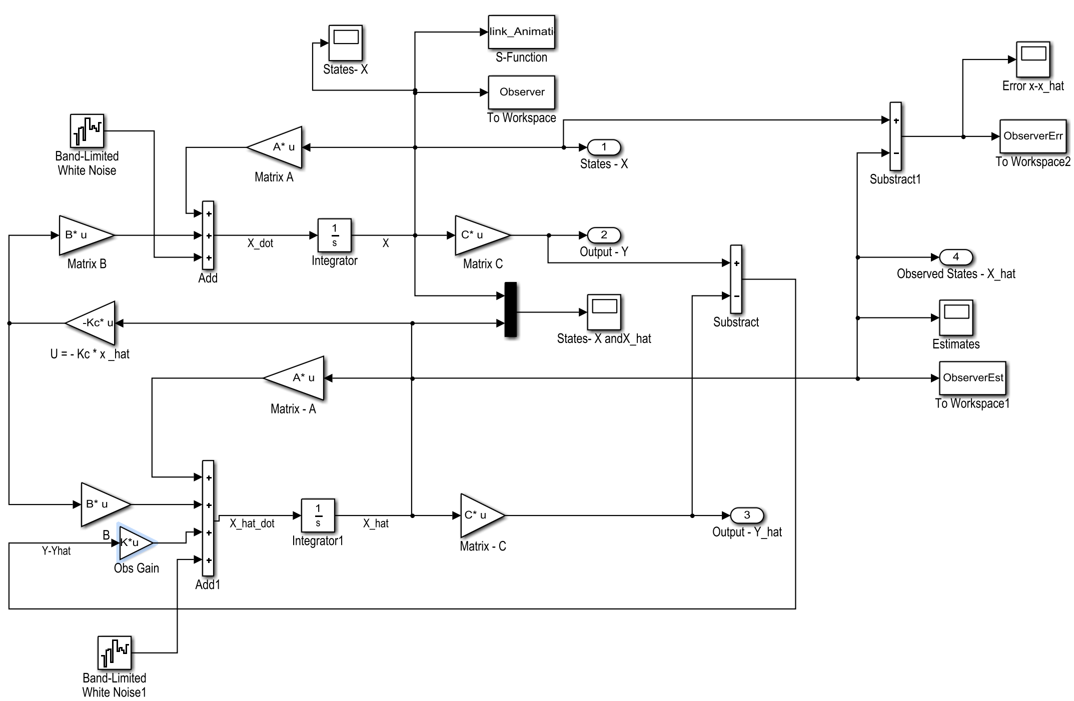


Fig. Simulink model of the linear optimal observer

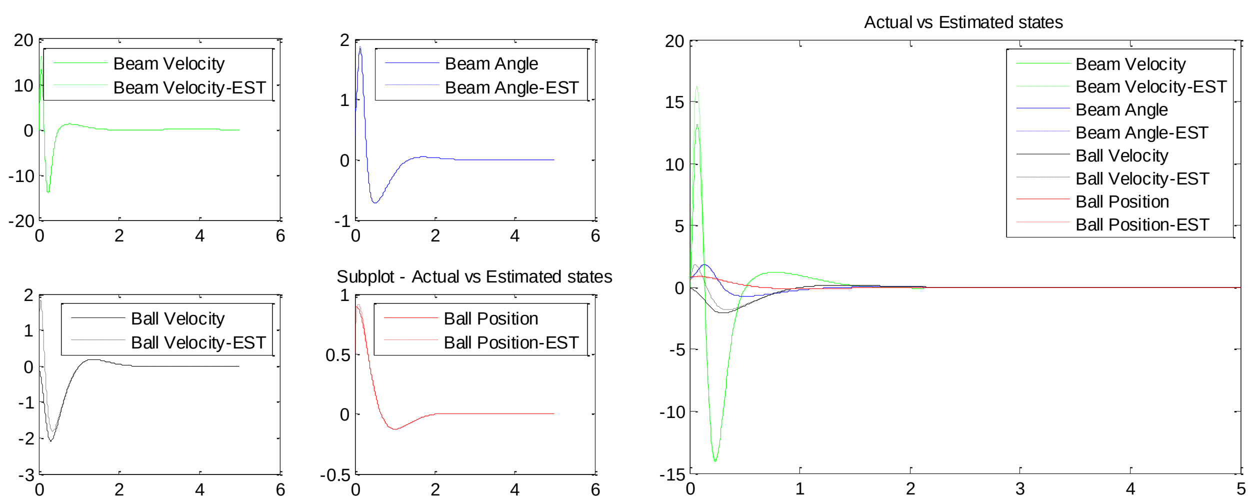
Design of the Optimal Observer

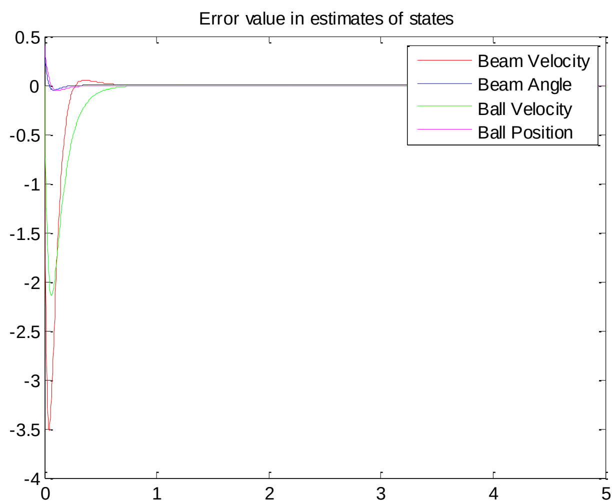
The controller poles and chosen first and the observer poles were assigned to be left of the controller poles. The standard rule of assigning them at 4 x controller poles was given up after initial trials as the noise level in the estimates has very high using the rule and the poles had high negative values. The LQR command was used with V1, V2, A & C matrices.

The value of V2 depends on the sensor accuracy and there is no scope to adjust it in a real model. So once V2 was chosen and decent response was possible, it was fixed. V1 was further iterated using different weights on the states to reduce the noise in the observer and the focus was to reduce the observer error in all the states simultaneously. Over all the observer error was high in the Beam velocity and the Ball velocity. Below are the chosen values of V1 and V2.

```
K_init_1 = lqr(A', C', V1, V2);
V1 = [ 40 0 0 0; 0 1 0 0; 0 0 60 0; 0 0 0 1 ]
V2 = [ 0.0005 0; 0 0.001 ]
```

In this case, there is disturbance into the system in the form of friction, linearization of a non- linear system is also a form of disturbance. Some small movements in the ball beam system frame or a user touching the ball are all input disturbances. There is also measurement noise in the Beam and Ball velocity states (x1 and x3) as sensors cannot directly measure those values. These can be adjusted using the V1 and V2 matrices.



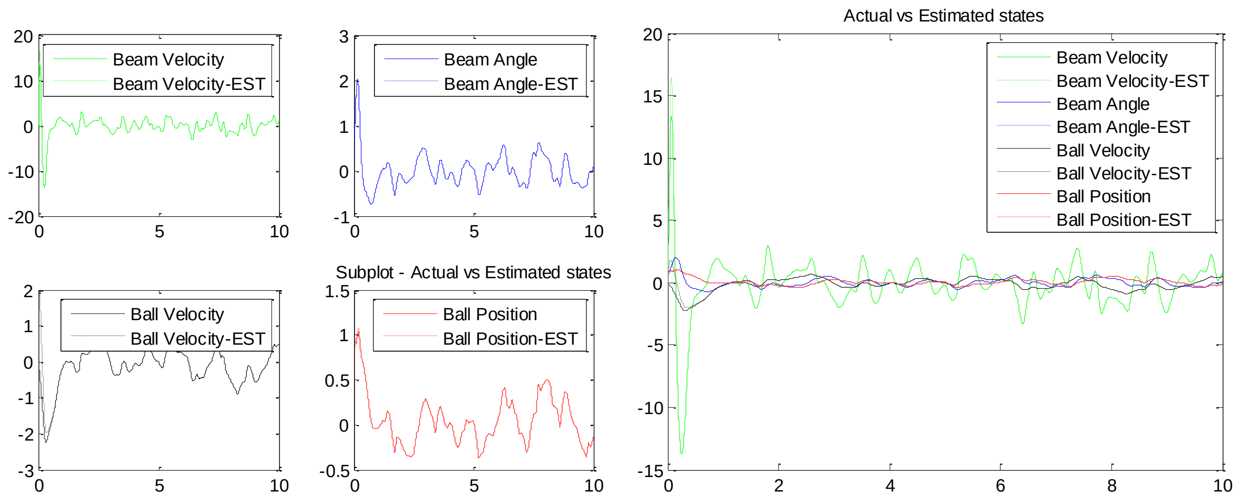


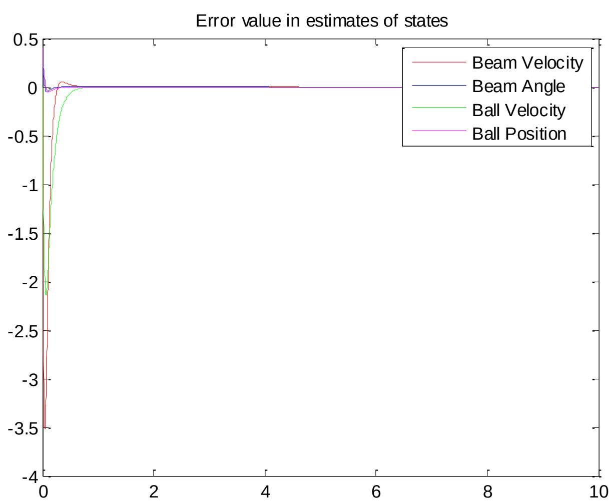
Initial conditions, Poles and Gain values

```
X_CO =  [  0    0.8000         0    0.9000 ]
X_OB = [0 0.4 0 0.5];

Contoller_Poles =
 [ -14.4925+14.4835i -14.4925-14.4835i -2.2372+2.2372i  -2.2372-2.2372i ]
Observer_Poles =
 [-45.7049+0.0000i -20.6492+0.5839i  -20.6492-0.5839i -7.7130+0.0000i ]

Kc = [ 1.2171   20.8016  -13.6973  -40.8468 ]
Ko = [ 572.7408 -357.9835; 55.8069 -3.9413;
      -24.9379 244.6136; -7.8826   38.1859 ]
```

Optimal observer when disturbance and measurement noise was added

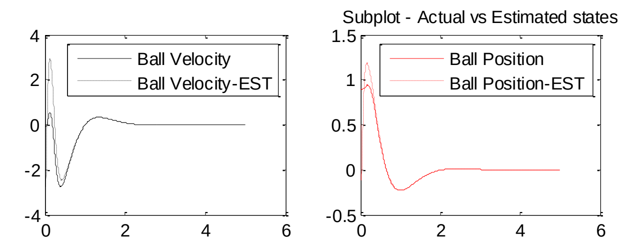
In this case, the observer matched the states very closely and the error was low. The plots of the error are similar to the case without any noise. This proves that the optimal observer with the optimal controller can reject external disturbances and measurement noise very effectively. The states were not completely driven to zero.

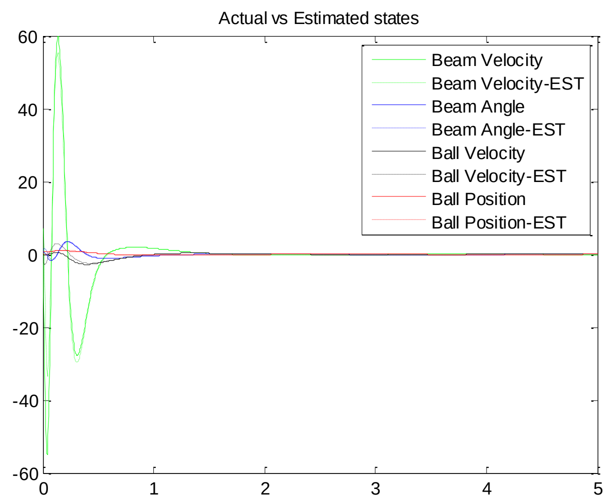


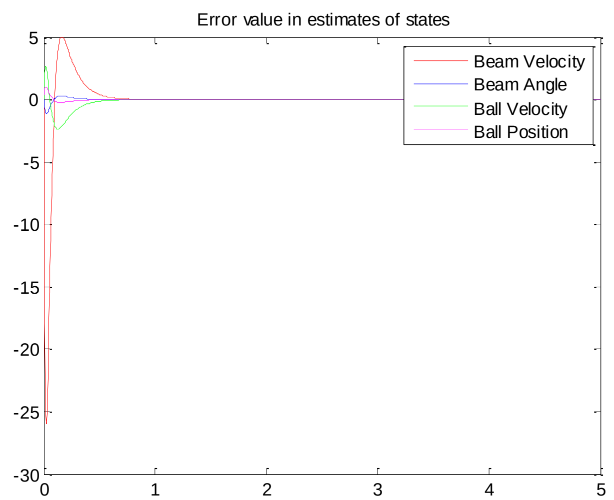


Using Pole Positioning to design the Sub-Optimal observer

When the performance of this sub optimal control is compared with the optimal control from the previous section we can see the error between the states and the estimates increased. The estimates are not as precise as earlier though the same pole locations were chosen for better comparison. The controller gain was similar but the observer gain is different in this case.





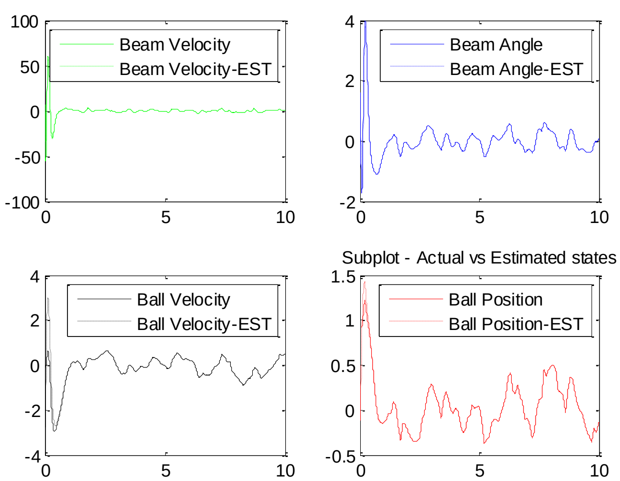


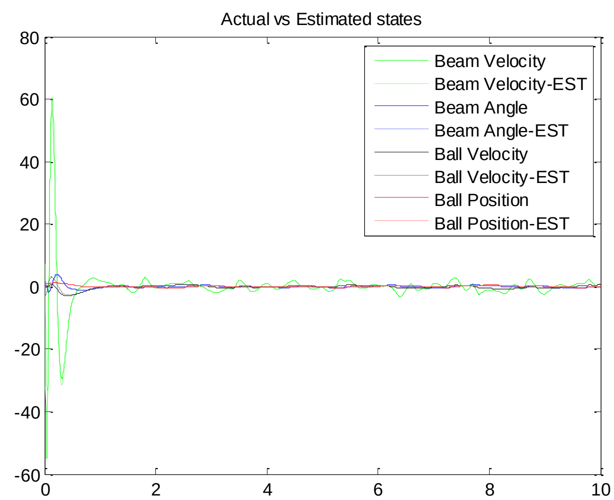
Initial conditions, Poles and Gain values

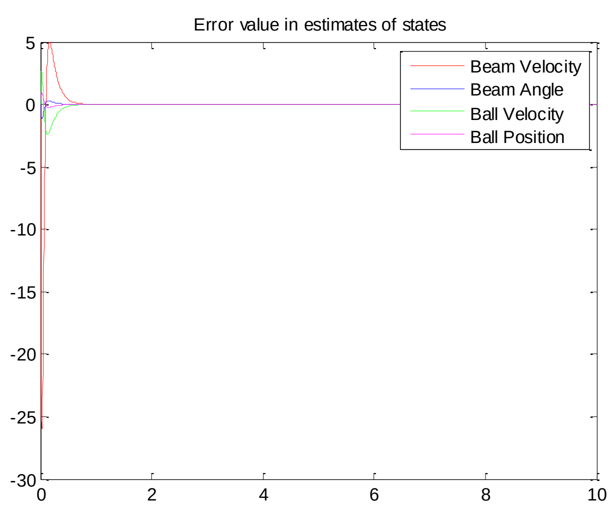
```
X_CO = [     0     0.8000      0     0.9000 ]
X_OB = [    0     0.4000           0     0.5000 ]

Contoller_Poles =
[ -14.4925-14.4835i -14.4925+14.4835i  -2.2372-2.2372i  -2.2372+2.2372i ]
Observer_Poles =
[ -45.7049+0.0000i -20.6492-0.5839i -20.6492+0.5839i  -7.7130+0.0000i ]

Kc = [ 1.2171   20.8015  -13.6973  -40.8464 ]
Ko =   1.0e+03 *
    [ 3.3679  3.3679;  0.1949 0.1949;  -0.4506 -0.4506; -0.1009 -0.1009 ]
```

Sub-Optimal observer with disturbance and measurement noise

In this case, the observer overall matched the states closely and the error increased from the earlier case where there was no noise. This also proves pole positioning method is not as effective the optimal observer though the exact same poles were chosen to begin with. The gains are not able to handle the noise inputs in the system effectively.







Summary

In the poles for the controller and observer the real part is greater than or equal to the imaginary part and because of this the poles are moving along the 45 degree line with less than 5% overshoot. The optimal control also places the two sets of poles at the same locations and because of that the system is critically damped and the settling time is small.

The V1 and V2 values for the Observer should be understood better to fine tune the observer performances. Noise was just used to test the performance of the observer but a real control system has to be robust in the presence of disturbances and measurement noises and accurate modeling of these noises is required.

The optimal observer and controller are excellent methods for optimizing the control system performance with minimum gain and mean square error. In this project Lyapunov equation was not used so the mean square error and mean square input were not calculated. The E[X^2] and E[U^2] terms will give better insights into the system.

The ball remains on the beam for the optimal case and also for the sub-optimal case as long as the initial condition is reasonable. For the non linear and linear open loop cases, the ball rolls off the beam and is not even visible in the picture. The Animation works for the optimal gain scenario with low oscillation but when there is high oscillation, it does not work properly and the ball seems like it is moving independent of the beam. I need to better understand the animation speed, and analyze the angle of the beam and the position of the ball as per the co-ordinate positions in the figure to get it to work for all cases.

References

Linear Optimal Control by Jeffery B. Burl
ECEN 5413 class notes
http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlStateSpace
https://www.youtube.com/watch?v=Yjmk2uFJ6fI
http://web.mit.edu/2.14/www/Handouts/PoleZero.pdf
http://cds.cern.ch/record/1100534/files/p73.pdf

Appendix

Instructions for running the programs

The programs have be run in Matlab first before running the Simulink model so that the values of the variables are accessed from the workspace. After running the Simulink model, the allplots.m method should be run with the correct runCase to plot the results.

a) The ***NonLinearInput_1.m*** program should be run before ***simulink_NonLinear_OpenLoop.mdl***
b) The ***Linearized_Model_2.m*** program should be run before ***simulink_Linear_OpenLoop.mdl***
c) The Optimal_Controller_Observer_3 program should be run before running the controller and observer models ***simulink_Linear_ClosedLoop_Controller.mdl*** and ***simulink_Linear_ClosedLoop_Controller_Observer.mdl***
d) The ***Sub_Optimal_Observer_PolePositioning_4.m*** should be run before running ***simulink_Linear_ClosedLoop_Controller_Observer.mdl*** for the sub-optimal case. Both the optimal and sub-optimal cases can be run using the same model with different inputs.
e) The ***simulink_Animation_0.m*** function is called automatically from Simulink model to run the animations.
f) The ***PolePositioning_4_4.m*** is called automatically from ***Sub_Optimal_Observer_PolePositioning_4.m***
g) The ***allPlots.m*** should be called after running each Simulink model by setting the correct ***runCase*** to plot the results. There are four cases to plot data from Simulink. First input the constants, matrices and gains into Simulink and get the states, estimates, errors etc back and plot the results. To plot each case follow the below steps.
   1. Run the m-files to store the matrices to the workspace and to calculate gains.
   2. Run the simulink model.
   3. Click on the Scope to ensure there are results, the ToWorkSpace block exports data back to the Matlab workspace.
   4. Set the flag in this file to the correct runCase that matches the simulink program and run this file to get the plots.

Matlab Programs

```
%-----------------------------------------------------------------------%
% 1 - Non-Linearized Model Inputs
%-----------------------------------------------------------------------%
clc; clear; close all; format compact;
Kb  = 0.0269
Kt  = 0.025
Ra  = 13.5
Jbe = 6.8129 *10^-5
m   = 0.0027
R   = 0.02   %r - in paper
Jba = 7.2 * 10^-7
g   = 9.81


%-----------------------------------------------------------------------%
% 2 - Linearized Model
%-----------------------------------------------------------------------%

clc; clear; close all; format compact;

Kb  = 0.0269
Kt  = 0.025
Ra  = 13.5
Jbe = 6.8129 *10^-5
m   = 0.0027
R   = 0.02   %r - in paper
Jba = 7.2 * 10^-7
g   = 9.81

% Tau = Kt*(u-Kb*x1)/Ra
% x1_dot = Tau - (m*g*cos(x2)*x4) - (2*m*x1*x3*x4) * 1/(Jbe+Jba+m*x4^2)
% x2_dot = x1
% x3_dot = (m*x1^2 *x4) - (m*g*sin(x2))* 1/((Jba/R^2)+m)
% x4_dot = x3
% x1_dot_eval = vpa(x1_dot)

%My linearized model
A_lin = [(-1*Kt*Kb)/(Ra*(Jba+Jbe)) 0 0 (-1*m*g)/(Jba+Jbe);
          1 0 0 0 ;
          0 (-1*m*g)/((Jba/(R^2))+m) 0 0;
          0 0 1 0]

B_lin = [(Kt/Ra) *1/(Jba+Jbe) ; 0 ;0 ;0 ]
C_lin = [ 1  0  0  0; 0  1  0  0; 0  0  1  0; 0  0  0  1] %[ 0  1  0  0; 0  0  0
1]
D_lin = [ 0 ]

%linmod from simulink
%[A_sim, B_sim, C_sim, D_sim] = linmod('simulink_NonLinear_OpenLoop')
```

```
% A_sim = [ -0.7235   0        0   -384.7115;
%            1.0      0        0      0;
%            0       -5.8860   0      0;
%            0        0        1.0    0 ]
% B_sim = [ 26.8973; 0; 0; 0 ]
% C_sim = [ 1  0  0  0; 0  1  0  0; 0  0  1  0; 0  0  0  1 ]
% D_sim = [ 0 ]

open_loop_poles = eig(A_lin)
sys_lin = ss(A_lin, B_lin, C_lin, D_lin);

figure
pzmap(sys_lin)
title('Open Loop Linear Model Poles and Zeros')

disp('From the map, the system is open loop unstable as there is a pole in the
RHP')
disp('(Right Half Plane) at (6.7242 + 0.0000i). This can also be seen from the
simulink model.')
disp('as one state goes to +inf and other goes to -inf - ball position and beam
angle')

Ctr = ctrb(A_lin, B_lin);
unCtr = length(A_lin)-rank(Ctr); % Number of uncontrollable states
if(unCtr == 0)
    disp('system is controllable')
end
Ob = obsv(A_lin,C_lin);
unObs = length(A_lin)-rank(Ob); % Number of unobservable states
if(unObs == 0)
    disp('system is observable')
end
% Though the system is unstable its observable and controllable

sim_time = 0:0.01:10;
%simulate initial condition - lsim can also do this
init_Cond_mat = [0; 1 ; 0 ;1];
figure
initial(sys_lin, init_Cond_mat, sim_time)
title('Open Loop Linear Model Initial Response')

u_lin = 0.2*ones(size(sim_time));
[Y, TSim] = lsim(sys_lin, u_lin, sim_time, init_Cond_mat);
figure
plot(TSim,Y(:,1),'r', TSim,Y(:,2),'b', TSim,Y(:,3),'g', TSim,Y(:,4),'m')
title('Open Loop Linear Model States')

%-----------------------------------------------------------------------%
%To set up the variables for the simulink Open loop simulation, run this file first
% A_lin, B_lin, C_lin, D_lin matrices and initial conditions
X_OL_INITIAL = [0 ; .8 ; 0 ; 0.8]


%-----------------------------------------------------------------------%
% 3 - optimal state variable feedback controller and observer
%-----------------------------------------------------------------------%
```

```
clc;clear; close all;
format compact %loose  % command window output is compact

A = [-0.7235         0         0      -384.7115 ;
      1.0000         0         0           0 ;
      0            -5.8860     0           0 ;
      0              0         1.0000      0 ]
B = [ 26.8973; 0; 0 ; 0 ]
%C = [0 0 0 0; 0 1 0 0; 0 0 0 0; 0 0 0 1]
C = [ 0 1 0 0; 0 0 0 1]
D = 0; %[ 0 ]

%x2 = pi/2; x4 = 1m; x2/x4
%R1 = [ 1 0  0  0; 0 10 0  0; 0 0 1 0; 0 0  0 10 ]
R1 = [ 0 0 0 0; 0 1 0  0; 0 0 0 0; 0 0 0 2 ]
%having zero weight on x1 and x3 causes oscillations as the imaginary part
magnitude is >=
%real part

%change R2 to adjust gain
%increasing R2 to 10 for ex is also increasing oscillation bcoz imag part increases
R2 = 1
disp('----optimal control phase 1----');
K_1 = lqr(A, B, R1, R2)
AK_1 = A-B*K_1;
disp('----eigen values in optimal control phase 1----');
eig(AK_1)
[evec, eval] = eig(AK_1);
% after the first step of lqr, using gain K moved the pole to LHP

disp('----optimal control phase 2----');
R2 = .1
K_2 = lqr(A, B, R1, R2) %start with A instead of ac_1
AK_2 = A-(B*K_2); %ac_1
eig(AK_2) % controller poles
[evec, eval] = eig(AK_2);

disp('----optimal control phase 3----');
R2 = .004
K_3 = lqr(A, B, R1, R2) %start with A instead of ac_1
AK_3 = A-(B*K_3); %ac_1
eig(AK_3) % controller poles
[evec, eval] = eig(AK_3);

%reducing R2 is reducing performance index, very marginal reduction in magnitude of
poles
%oscillation remains

SS_inpA = AK_3;
%does not matter what R1 and R2 are used, the oscillation cannot be minimized
sys_1 = ss(SS_inpA, B, C, D);
time = 0:0.1:10;
r_inp = .2*zeros(size(time)); %r_inp = -12:0.06:12;
%initial condition plays a role in the value of input voltage
init_Cond_mat = [ 0; 1 ; 0 ; 0.8];

[Y,Tsim,X]=lsim(sys_1, r_inp, time, init_Cond_mat);
```

```
figure
plot(Tsim,X(:,1), 'g',Tsim,X(:,2), 'b', Tsim,X(:,3), 'k', Tsim,X(:,4),'r' )
legend('Beam Velocity', 'Beam Angle', 'Ball Velocity', 'Ball Position')

%CtrlGainForInp
Kc = K_3;
u_inp =  Kc * X';
figure
plot(Tsim,u_inp,'r' )

% %perfidx_value
lenX = size(X); %401 rows
e_X2 = [];
for idx = 1:lenX(1)
   temp_1 = X(idx, :) * R1 * X(idx, :)';
   e_X2 = [ e_X2 temp_1];
end

lenU = size(u_inp);
e_U2 = [];
for idx = 1:lenU(2)
   temp_2 = u_inp(1, idx) * R2 * u_inp(1,idx);
   e_U2 = [ e_U2 temp_2];
end
perf_idx = e_X2 + e_U2;
disp('total value of performance index, J = ')
J = sum(perf_idx)


%----------------------------------------------------------------------------------%
% %X'*R1*X + u_inp'*R2*u_inp;
% Code for the Optimal Observer
%----------------------------------------------------------------------------------%

%Z = D_wgt * x
D_wgt = [ 0 1 0 0; 0 0 0 1 ]; %Here D = C
R3 = 1; R1 = 1; R2 = 1;
%use the weight matrix to apply costs on the x2(beam angle) and x4(ball posn)

AObs = A';
BObs = C'

%V1 - disturbance
%V1 = [ 21 0 0 0; 0 0 0 0; 0 0 41 0; 0 0 0 0 ]  %more penalty on x1 and x3
%Case1 - small voltage issues, ball in center, ball velocity user interference is
disturbance
%disturbance modeling error
V1 = [ 40 0 0 0; 0 1 0 0; 0 0 30 0; 0 0 0 1 ]
%Case 2 - small voltage issues, ball not in center X1, X3 and X4

%V2 = measurement noise weigh X1 and X3
%V2 - sensor noise in accelerations only, no weight on position and angle
%V2 = [ 0.1 0; 0 0.3 ] %c 2x2 % > 1 high oscillation
%V2 = [ 0.001 0; 0 0.005 ] %c 2x2
V2 = [ 0.0005 0; 0 0.001 ] %c 2x2

%---------------
```

```
K_init_1 = lqr(AObs, BObs, V1, V2);
Ko_1 = K_init_1'
eig(A-Ko_1*C)

%observer phase -2
%Adjust V2 to make x_hat less noisy and faster than x ; only x3 is not as required
here
%-------------------------------------------------------------------------------
---------5
%V1 = [ 30 0 0 0; 0 1 0 0; 0 0 20 0; 0 0 0 1 ] %err 25, poles bad, obs ok
%V1 = [ 0 0 0 0; 0 202 0 0; 0 0 0 0; 0 0 0 10 ] %err 14, poles ok, obs not that
good, low os
%  V1 = [ 0 0 0 0; 0 80 0 0; 0 0 0 0; 0 0 0 20 ] %no imag poles, ball vel bad erro
- 3.5
%V1 = [ 2 0 0 0; 0 0 0 0; 0 0 20 0; 0 0 0 0 ] %more oscill obs is good, imag more,
err 25
%  V1 = [ 40 0 0 0; 0 4 0 0; 0 0 60 0; 0 0 0 1 ] %**err - 5, poles ok obs ok
  V1 = [ 40 0 0 0; 0 1 0 0; 0 0 60 0; 0 0 0 1 ]
%V1 = [ 20 0 0 0; 0 1 0 0; 0 0 10 0; 0 0 0 1 ] %more oscill obs is good, imag more,
err 25
%  V1 = [ 20 0 0 0; 0 1 0 0; 0 0 10 0; 0 0 0 1 ] %low oscill obs is good, imag
more, err 8
%  V1 = [ 200 0 0 0; 0 0 0 0; 0 0 1 0; 0 0 0 0 ] %poles - 45, vel obs bad, err 7
%V1 = [ 1 0 0 0; 0 1 0 0; 0 0 10 0; 0 0 0 10 ] % obs ok, err 25, poles bad ok osc

%for low v2
K_init_2 = lqr(AObs, BObs, V1, V2);
Ko_2 = K_init_2'
eig(A-Ko_2*C)

%from optimal controller file
%Kc = [1.2171 20.8016 -13.6973 -40.8468 ] %[ 0.6965    4.6980   -1.7554   -0.6675 ]

Ko = Ko_2
Ace = [ A-(B*Kc)  (B*Kc);  % for A-BKc use the
       zeros(size(A)) (A-Ko*C)];
Be = [    B
       zeros(size(B)) ];
Ce = [ C    zeros(size(C)) ];

Contoller_Poles = eig(A-(B*Kc))
Observer_Poles = eig(A-Ko*C)
%-----------------------------------%----------------------------------
disp('gains for the simulink model')
X_CO = [ 0; 0.8 ; 0 ; 0.9] %angle in radians
%Kc = [ 3.4087   24.2399  -12.3993  -31.7551 ]
Kc
Ko
X_OB = [0 0.4 0 0.5];


%---------------------------------------------------------------------%
% 4 - Sub-Optimal Observer pole positioning
%---------------------------------------------------------------------%

clc;clear; close all;
format compact %loose  % command window output is compact
```

```
A = [ -0.7235   0        0   -384.7115;
           1.0      0        0      0;
           0       -5.8860   0      0;
           0        0        1.0    0 ]
B = [ 26.8973; 0; 0; 0 ]
%C = [ 1  0  0  0; 0  1  0  0; 0  0  1  0; 0  0  0  1 ]
C = [ 0  1  0  0; 0  0  0  1]
D = [ 0 ]

%pass A, C matrices to pole_pos observer without transposing
%the code can handle then transposing.

%Lmda_inp_Con = [-10.5080+2.8i ; -4.0257+0.8i ; -2.0988 + 1.9233i;   -3.0988 -
1.9233i]
% the pole positioning code works better without complex poles
%Lmda_inp_Con = [-10.5080 ; -3.0257 ; -2.0988 ;   -1.0988 ]

% Using the same poles from the Optimal case for comparision
Lmda_inp_Con = [ -14.4925+14.4835i; -14.4925-14.4835i; -2.2372+2.2372i; -2.2372-
2.2372i ]
%Lmda_inp_Obs = [ -45.7049 + 0.0000i; -20.6492 + 0.5839i; -20.6492 - 0.5839i; -
7.7130 + 0.0000i ]

Kc_1 = PolePositioning_4_4(A, B, Lmda_inp_Con, 'CTRL');
Kc = double(Kc_1)

sys_co = ss(A-B*Kc, B, C, D);
time_1 = 0:0.01:10;
u_in = 0.2*ones(size(time_1));
X_CO = [ 0; 0.8 ; 0 ; 0.9]%initial conditions
[Yc, Tsimc, Xc] = lsim(sys_co, u_in, time_1, X_CO);
u_inpController =  Kc * Xc';
figure
plot(Tsimc, u_inpController,'r' )
xlabel('Time (sec)'); ylabel('Input voltage (volts)'); title('Input voltage vs
Time');

%----------------------------------------------%
%Observer Poles
%Lmda_inp_Obs = 4* Lmda_inp_Con
%same poles from optimal case
Lmda_inp_Obs = [ -45.7049 + 0.0000i; -20.6492 + 0.5839i; -20.6492 - 0.5839i; -
7.7130 + 0.0000i ]
Ko_1 = PolePositioning_4_4(A, C, Lmda_inp_Obs, 'OBS');
Ko = double(Ko_1);

At = [ A-B*Kc            B*Kc
       zeros(size(A))    A-Ko*C ];
Bt = [    B %*Nbar
       zeros(size(B)) ];
Ct = [ C    zeros(size(C)) ];

disp('Poles and Gain values')

Contoller_Poles = eig(A-B*Kc)'
Observer_Poles = eig(A-Ko*C)'
```

```
Kc
Ko

X_OB = [0 0.4 0 0.5]
xoc_0 = [ X_CO ; X_OB' ];
sys_co = ss(At, Bt, Ct, 0);


%-----------------------------------------------------------------------%
% 4 - Pole poistioning Gains program for Controller and Observer
%     can be used for a 4 x 4 A matrix
%-----------------------------------------------------------------------%

function y = PolePositioning_4_4(A_inp, BC_inp, Lmda_inp, CtrlObs)

    if(strcmp(CtrlObs, 'CTRL'))
        disp('CTRL_')
        A = A_inp;
        B = BC_inp; % B will be input in this case
    elseif(strcmp(CtrlObs, 'OBS'))
        disp('OBS_')
        A = A_inp';
        B = BC_inp'; % C will be input in this case
    end
    syms s
    Lmda = Lmda_inp;

    size_A = size(A);
    Iden = eye(size_A(1), size_A(1) );
    Phi_temp = (s*Iden -A);
    Phi = inv(Phi_temp);

    Psi_Lmda = Phi * B;

    %desired poles
    Lmda_1 = Lmda(1);
    Lmda_2 = Lmda(2);
    Lmda_3 = Lmda(3);
    Lmda_4 = Lmda(4);

    Psi_Lmda_1 = subs(Psi_Lmda, 's', Lmda_1);
    %Psi_Lmda_1_eval = eval(Psi_Lmda_1)
    Psi_Lmda_2 = subs(Psi_Lmda, 's', Lmda_2);
    Psi_Lmda_3 = subs(Psi_Lmda, 's', Lmda_3);
    Psi_Lmda_4 = subs(Psi_Lmda, 's', Lmda_4);
    %v1 = 1; v2 = 1; v3 = 1; v4 = 1;
    row_col = size(Psi_Lmda);
    if(row_col(2) == 1)
        v1 = 1; v2 = 1; v3 = 1; v4 = 1;
    else
        v1 = [1; 1] ; v2 = [1; 1]; v3 = [1; 1]; v4 = [1; 1];
    end

    M = [ Psi_Lmda_1*v1  Psi_Lmda_2*v2 Psi_Lmda_3*v3 Psi_Lmda_4*v4];
    V = [v1 v2 v3 v4];
    K = -1 *V * inv(M);
```

```
    %AK=[];
    if(strcmp(CtrlObs, 'CTRL'))
        disp('CTRL__')
        AK = (A-B*K);
    elseif(strcmp(CtrlObs, 'OBS'))
        disp('OBS__')
        K  =  K';
        AK = (A'-K*B'); %its just A-KoC, B=C' earlier
    end

    %check
    sI_Minus_AK = (s*Iden - AK);
    EigVal_AK = det(s*Iden - AK);
    EigeVal = solve(EigVal_AK, s);
    disp('CTRL/OBS input Poles combined')
    double(EigeVal)
    %EigeVal values are in the descending order
    %return value
    y=K;
end


%---------------------------------------------------------------------%
% 0 - Simulink Animation program
%---------------------------------------------------------------------%

function [sys, x0, str, ts]=simulink_Animation_0(t, x, u, flag) %(t, x, uu, flag)

% Declare global variables.  Appropriate coordinates for drawings
% will be stored in these.
global  xBeam yBeam xBall yBall

% Global variables for handles of drawings
global Beam_2D Ball_2D
global  xRotateBeam yRotateBeam xShiftBall yShiftBall prevBallDisp

%---------------
xBeamInit = [0 40];
yBeamInit = [19.5 19.5];
xBallInit = 20;
yBallInit = 20;
% Global variable for the handle of the animation figure.
global AnimDemoFigure

% Set variables str and ts according to S-function specifications
str=[];
% ts=[time between samples, start time]  Decreasing time between
% samples will slow the simulation down if it runs too fast.
ts=[.0045 0]; %WAS 0.045

% Set the initial position for each drawing
% Check the value of flag
if flag==2
    u_inp=u;
    size(u_inp);
    angle = u_inp(2); %beam angle - theta
    balldisp = u_inp(4); % ball position
```

```
    % Update - update figure according to input.
    % Make sure correct figure is selected and bring it to the front.
    if any(get(0,'Children')==AnimDemoFigure)
        set(0, 'CurrentFigure', AnimDemoFigure);
        if any(get(gca, 'Children')==Beam_2D) %Rotate

            % Calculate new coordinates for each figure.
            %Total Length L = 1 m
            xBeam = xBeamInit; %[0 40] ;%;* cos(u);
            displ = 0.1*tan(angle);
            yBeam_bef = yBeam;
            %yBeam = [yBeamInit(1)-displ yBeamInit(2)+displ] %[0 1] * sin(u);
            yBeam = [yBeam_bef(1)-displ yBeam_bef(2)+displ];
            %xBall = xBallInit + balldisp*2; %(no vertical disp now)
balldisp/cos(u)

            if(balldisp ~= prevBallDisp)
                %xBall = xBallbef + balldisp;
                xBall =  xBallInit + balldisp*10;
            end
            prevBallDisp = balldisp;
            yBall = yBallInit +  (0.1*(xBallInit - xBall) * tan(angle));
            %-------------------------------
            set(Beam_2D, 'XData', xBeam, 'YData', yBeam);
            set(Ball_2D, 'XData', xBall,  'YData', yBall);
            drawnow
        end
    end
    % Specify sys according to s-function specifications.
    sys=[];
elseif flag==0
    sizes=simsizes
    sizes.NumInputs = 4;%2
    sizes.NumSampleTimes = 1
    % Initialization - setup figure, create and draw base shapes.
    % Check for existing figure.
    [fig, flag]=figflag('Animation Demo Figure', 0);

    if flag % If figure exists, clear it.
        AnimDemoFigure=fig;
        cla reset;
    else % If not, create new figure.
        AnimDemoFigure=figure;
    end

    % Set title of figure.
    set(AnimDemoFigure, ...
        'Name', 'Animation Demo Figure',...
        'NumberTitle', 'on')

    % Set properties and limits of the axes.
    set(gca, ...
       'Visible', 'off',...
       'DrawMode','fast',...
       'XLim', [-20 20],...
       'YLim', [-20 20]);
```

```
   % 'XLim', [0 24],...        'YLim', [0 36]);  // 0 40]  0 40]

        xBeam = xBeamInit; % = [0 50]
        yBeam = yBeamInit; % = [19.5 19.5]

        xBall = xBallInit ;%10; %(no vertical disp now) balldisp/cos(u)
        yBall = yBallInit ;%20;
        %---------------

   % Draw base shapes at initial positions.
   hold on; axis([0 40 0 40]); %axis([-10 10 0 inf])
  % Create base shapes.

   Ball_2D = plot(xBall, yBall, 'bo' )
   Beam_2D = plot(xBeam, yBeam, 'k' )
   plot(20, 19, 'r^')

   % Define sys and x0 according to S-function specification.
   % sys=[0 0 0 (# of inputs) 0 0 1]
   %sys=[0 0 0 1 0 0 1];
   sys = simsizes(sizes)
   x0=[];
end




%--------------------------------------------------------------------------%
% 1 - program for plotting the results exported to workspace from simulink
%--------------------------------------------------------------------------%

close all;
%There are four cases to plot data from simulink
%input the constants, matrices and gains into simulink
%get the states, estimates, errors etc back and plot the results.

% To plot each case follow the below steps.
% 1. Run the m-files to store the matrices to the workspace and to calculate gains
% 2. Run the simulink model
% 3. click on the scope to ensure there are results, the to WorkSpace block exports
data
%    back to the Matlab workspace
% 4. set the flag in this file to the correct ase and run this file to get the
plots.

% Select the correcrt runcase that matches the simulink program.
% runCase = 'NonLinearOpenLoop'
% runCase = 'LinearOpenLoop'
% runCase = 'OptimalController'
 runCase = 'SubOptimal_and_OptimalObserver'

% pre-requisite - run NonLinearInput.m and set runCase to NonLinearOpenLoop
if(strcmp(runCase,'NonLinearOpenLoop'))

    size(NonLinOpenLoop.signals.values)
    Tsim = NonLinOpenLoop.time;
    BeamVelocity = NonLinOpenLoop.signals.values(:,1);
```

```
    BeamAngle    = NonLinOpenLoop.signals.values(:,3);
    BallVelocity = NonLinOpenLoop.signals.values(:,5);
    BallPosition = NonLinOpenLoop.signals.values(:,6);

    %figure; plot(NonLinOpenLoop.time, NonLinOpenLoop.signals.values);
    figure
    plot(Tsim,BeamVelocity,'g', Tsim,BeamAngle, 'b', Tsim,BallVelocity, 'k',
Tsim,BallPosition,'r' )
    legend('Beam Velocity', 'Beam Angle', 'Ball Velocity', 'Ball Position',
'Location','SouthWest')
    title('All states in the non-linear model')

elseif(strcmp(runCase,'LinearOpenLoop'))

    size(LinOpenLoop.signals.values)
    Tsim = LinOpenLoop.time;
    BeamVelocity = LinOpenLoop.signals.values(:,1);
    BeamAngle    = LinOpenLoop.signals.values(:,2);
    BallVelocity = LinOpenLoop.signals.values(:,3);
    BallPosition = LinOpenLoop.signals.values(:,4);

    figure
    plot(Tsim,BeamVelocity,'g', Tsim,BeamAngle, 'b', Tsim,BallVelocity, 'k',
Tsim,BallPosition,'r' )
    legend('Beam Velocity', 'Beam Angle', 'Ball Velocity', 'Ball Position',
'Location','SouthWest')
    title('All states in the linear model')

elseif(strcmp(runCase,'OptimalController'))

    size(OptController.signals.values)
    Tsim = OptController.time; %simout
    BeamVelocity = OptController.signals.values(:,1);
    BeamAngle    = OptController.signals.values(:,2);
    BallVelocity = OptController.signals.values(:,3);
    BallPosition = OptController.signals.values(:,4);

    figure
    plot(Tsim,BeamVelocity,'g', Tsim,BeamAngle, 'b', Tsim,BallVelocity, 'k',
Tsim,BallPosition,'r' )
    %plot(Tsim,X(:,1), 'g',Tsim,X(:,2), 'b', Tsim,X(:,3), 'k', Tsim,X(:,4),'r' )
    legend('Beam Velocity', 'Beam Angle', 'Ball Velocity', 'Ball Position',
'Location','SouthEast')
    title('All states in linear closed loop Optimal Controller')

    figure   %simout.signals.values(:,1)    legend boxoff;
    % column format %subplot(4,1,1) subplot(4,1,2) subplot(4,1,3) subplot(4,1,4)
    % 2x2 array %subplot(2,2,1);
    subplot(2,2,1); plot(Tsim, BeamVelocity); title('Beam Velocity'); %line1 =
    subplot(2,2,2); plot(Tsim, BeamAngle);    title('Beam Angle')
    subplot(2,2,3); plot(Tsim, BallVelocity); title('Ball Velocity')
    subplot(2,2,4); plot(Tsim, BallPosition); title('Ball Position')
    %title('Subplot - All states in linear closed loop Optimal Controller')

    %CtrlGainForInp
    size(Kc)
    u_inp =  Kc * OptController.signals.values'; %X';
```

```
    figure
    plot(Tsim,u_inp,'r' )
    title('Input Voltage value')

elseif (strcmp(runCase,'SubOptimal_and_OptimalObserver')) %SubOptimal -
PolePositionObserver

    %use the same code to plot the sub optimal and optimal observers
    Tsim = Observer.time;
    BeamVelocity = Observer.signals.values(:,1);
    BeamAngle    = Observer.signals.values(:,2);
    BallVelocity = Observer.signals.values(:,3);
    BallPosition = Observer.signals.values(:,4);

    BeamVelocityEst = ObserverEst.signals.values(:,1);
    BeamAngleEst    = ObserverEst.signals.values(:,2);
    BallVelocityEst = ObserverEst.signals.values(:,3);
    BallPositionEst = ObserverEst.signals.values(:,4);

    BeamVelocityErr = ObserverErr.signals.values(:,1);
    BeamAngleErr    = ObserverErr.signals.values(:,2);
    BallVelocityErr = ObserverErr.signals.values(:,3);
    BallPositionErr = ObserverErr.signals.values(:,4);

    figure % all in one plot
    plot(Tsim,BeamVelocity,'-g', Tsim,BeamVelocityEst,':g', Tsim,BeamAngle,'-b',
Tsim,BeamAngleEst,':b', ...
         Tsim,BallVelocity,'-k', Tsim,BallVelocityEst,':k', Tsim,BallPosition,'-r',
Tsim,BallPositionEst,':r');
    legend('Beam Velocity', 'Beam Velocity-EST', 'Beam Angle', 'Beam Angle-EST',
...
           'Ball Velocity', 'Ball Velocity-EST', 'Ball Position', 'Ball Position-
EST');
    title('Actual vs Estimated states')

    figure %separate states into subplots
    subplot(2,2,1); plot(Tsim,BeamVelocity,'-g', Tsim,BeamVelocityEst,':g'); leg_1
= legend('Beam Velocity', 'Beam Velocity-EST');
    subplot(2,2,2); plot(Tsim,BeamAngle,'-b', Tsim,BeamAngleEst,':b'); leg_2 =
legend('Beam Angle', 'Beam Angle-EST');
    subplot(2,2,3); plot(Tsim,BallVelocity,'-k', Tsim,BallVelocityEst,':k'); leg_3
= legend('Ball Velocity', 'Ball Velocity-EST');
    subplot(2,2,4); plot(Tsim,BallPosition,'-r', Tsim,BallPositionEst,':r'); leg_4
= legend('Ball Position', 'Ball Position-EST');
    title('Subplot - Actual vs Estimated states')

    figure
    plot(Tsim,BeamVelocityErr,'r', Tsim,BeamAngleErr,'b', Tsim,BallVelocityErr,'g',
Tsim,BallPositionErr,'m') ;
    legend('Beam Velocity', 'Beam Angle', 'Ball Velocity', 'Ball Position')
    title('Error value in estimates of states');

end
```