King Fahd University of Petroleum & Minerals

Modern Control Engineering

Bin Mohaya, Turki
201569090

Designing A Linear Controller to Compensate A Non-linear System

April 11, 2019

INTRODUCTION

This project aims to analyze and design a linear controller for a non-linear system. The system utilized is the twin rotor multi-input multi-output (MIMO) system. The objective is linearizing the non-linear model, and thus provide a setpoint input and try to control the output accordingly. The conventional control techniques will be utilized to reach out this objective. We will design a Proportional-Integral-Derivative compensator to do such task. The main tools used are the S-function in MATLAB as well as the SISO-tool. The system has two inputs and will provide two outputs. The state space model of the system is a six-state space. We will provide all related plots to the design procedure such as step responses and root locus. Furthermore, details of the controller choice will be discussed.

PROCEDURE

Our main objective is to model the non-linear system with the following mathematical model with parameters defined in Table 1:

$$\dot{\psi} = -\omega_{\psi} \tag{1}$$

$$\dot{\omega}_{\psi} = -\frac{M_{g1}}{J_1} sin\psi + \frac{M_{g2}}{J_1}\cos\psi - B_{1\psi}\omega_{\psi} - B_{2\psi}sgn(\omega_{\psi}) + \frac{b_1}{J_1}\tau_1 + \frac{a_1}{J_1}\tau_1^2 + K_{gy}\dot{\phi}cos\phi \tag{2}$$

$$\tau_1 = \frac{k_1}{T_{11}s + T_{10}}u_1 \tag{3}$$

$$\tau_2 = \frac{k_2}{T_{21}s + T_{20}}u_2 \tag{4}$$

$$\dot{\phi} = \omega_{\phi} \tag{5}$$

$$\dot{\omega}_{\phi} = -B_{1\phi}\omega_{\phi} - B_2 sgn(\omega_{\phi}) + \frac{b_2}{J_2}\tau_2 + \frac{a_2}{J_2}\tau_2^2 \tag{6}$$

We will stabilize this system and give a desired output ϕ = -0.6 rad and second output ψ = 0.4 rad.

The linearized model is given by:

$$\dot{x}_\psi = A_\psi x_\psi + B_\psi u_1 \quad (7)$$
$$y_\psi = C_\psi x_\psi \quad (8)$$

$$\dot{x}_\phi = A_\phi x_\phi + B_\phi u_2 \quad (9)$$
$$y_\phi = C_\phi x_\phi \quad (10)$$

$$A_\psi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -0.06 & 1.51 \\ 0 & 0 & -0.91 \end{bmatrix}, B_\psi = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} C_\psi = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad (11)$$

$$A_\phi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -0.006 & 4.5 \\ 0 & 0 & -1 \end{bmatrix}, B_\phi = \begin{bmatrix} 0 \\ 0 \\ 0.8 \end{bmatrix} C_\phi = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad (12)$$

Table 1: Parameters of Pendulum-Cart System

Parameters	Values
J_1	6.12×10^{-2}
J_2	2×10^{-2}
M_{g1}	0.32
M_{g2}	0.48
a_1	1.35×10^{-2}
b_1	9.24×10^{-2}
a_2	2×10^{-2}
b_2	9×10^{-2}
$B_{1\psi}$	6×10^{-2}
$B_{2\psi}$	1×10^{-2}
$B_{1\phi}$	6×10^{-3}
$B_{2\phi}$	1×10^{-3}
K_{gy}	5×10^{-2}
k_1	1.1
k_2	0.8
T_{11}	1.1
T_{10}	1
T_{21}	1
T_{20}	1
k_c	-0.2
T_p	2
T_0	3.5

First, the S-function code is utilized to model the non-linear system in MATLAB as follows:

```
function Project_316(block)
%MSFUNTMPL_BASIC A Template for a Level-2 MATLAB S-Function
%   The MATLAB S-function is written as a MATLAB function with the
%   same name as the S-function. Replace 'msfuntmpl_basic' with the
%   name of your S-function.
%
%   It should be noted that the MATLAB S-function is very similar
%   to Level-2 C-Mex S-functions. You should be able to get more
%   information for each of the block methods by referring to the
%   documentation for C-Mex S-functions.
%
%   Copyright 2003-2010 The MathWorks, Inc.

%%
%% The setup method is used to set up the basic attributes of the
%% S-function such as ports, parameters, etc. Do not add any other
%% calls to the main body of the function.
%%
setup(block);

%endfunction

%% Function: setup
%% ===================================================
%% Abstract:
%%   Set up the basic characteristics of the S-function block such
as:
%%   - Input ports
%%   - Output ports
%%   - Dialog parameters
%%   - Options
%%
%%   Required         : Yes
%%   C-Mex counterpart: mdlInitializeSizes
%%
function setup(block)

% Register number of ports
block.NumInputPorts  = 2;
block.NumOutputPorts = 2;
block.NumContStates = 6;

% Setup port properties to be inherited or dynamic
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;

% Override input port properties
block.InputPort(1).Dimensions        = 1;
block.InputPort(1).DatatypeID  = 0;  % double
block.InputPort(1).Complexity  = 'Real';
block.InputPort(1).DirectFeedthrough = true;

% Override output port properties
block.OutputPort(1).Dimensions       = 1;
block.OutputPort(1).DatatypeID  = 0; % double
block.OutputPort(1).Complexity  = 'Real';
```

```
% Register parameters
block.NumDialogPrms     = 0;

% Register sample times
%  [0 offset]            : Continuous sample time
%  [positive_num offset] : Discrete sample time
%
%  [-1, 0]               : Inherited sample time
%  [-2, 0]               : Variable sample time
block.SampleTimes = [0 0];

% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume
DefaultSimState
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'CustomSimState',  < Has GetSimState and SetSimState methods
%    'DisallowSimState' < Error out when saving or restoring the
model sim state
block.SimStateCompliance = 'DefaultSimState';

%% -----------------------------------------------------------------
-
%% The MATLAB S-function uses an internal registry for all
%% block methods. You should register all relevant methods
%% (optional and required) as illustrated below. You may choose
%% any suitable name for the methods and implement these methods
%% as local functions within the same file. See comments
%% provided for each function for more information.
%% -----------------------------------------------------------------
-

block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
block.RegBlockMethod('InitializeConditions',
@InitializeConditions);
block.RegBlockMethod('Start', @Start);
block.RegBlockMethod('Outputs', @Outputs);     % Required
block.RegBlockMethod('Update', @Update);
block.RegBlockMethod('SetInputPortSamplingMode',
@SetInpPortFrameData);
block.RegBlockMethod('Derivatives', @Derivatives);
block.RegBlockMethod('Terminate', @Terminate); % Required


function SetInpPortFrameData(block, idx, fd)
block.InputPort(idx).SamplingMode = fd;
for i = 1:2
  block.OutputPort(idx).SamplingMode = fd;
end
%end setup

%%
%% PostPropagationSetup:
%%   Functionality    : Setup work areas and state variables. Can
%%                      also register run-time methods here
%%   Required         : No
%%   C-Mex counterpart: mdlSetWorkWidths
%%
function DoPostPropSetup(block)
block.NumDworks = 1;
```

```
  block.Dwork(1).Name            = 'x1';
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;      % double
  block.Dwork(1).Complexity      = 'Real'; % real
  block.Dwork(1).UsedAsDiscState = true;


%%
%% InitializeConditions:
%%   Functionality    : Called at the start of simulation and if it
is
%%                      present in an enabled subsystem configured
to reset
%%                      states, it will be called when the enabled
subsystem
%%                      restarts execution to reset the states.
%%   Required         : No
%%   C-MEX counterpart: mdlInitializeConditions
%%
function InitializeConditions(block)
x0=[0 ; 0 ; 0; 0; 0; 0];
block.ContStates.Data=x0;


%end InitializeConditions


%%
%% Start:
%%   Functionality    : Called once at start of model execution. If
you
%%                      have states that should be initialized
once, this
%%                      is the place to do it.
%%   Required         : No
%%   C-MEX counterpart: mdlStart
%%
function Start(block)

block.Dwork(1).Data = 0;

%end Start

%%
%% Outputs:
%%   Functionality    : Called to generate block outputs in
%%                      simulation step
%%   Required         : Yes
%%   C-MEX counterpart: mdlOutputs
%%
function Outputs(block)


x=block.ContStates.Data(1:6);
block.OutputPort(1).Data=x(1);
block.OutputPort(2).Data=x(5);
```

```
%end Outputs

%%
%% Update:
%%   Functionality    : Called to update discrete states
%%                      during simulation step
%%   Required         : No
%%   C-MEX counterpart: mdlUpdate
%%
function Update(block)

block.Dwork(1).Data = block.InputPort(1).Data;

%end Update

%%
%% Derivatives:
%%   Functionality    : Called to update derivatives of
%%                      continuous states during simulation step
%%   Required         : No
%%   C-MEX counterpart: mdlDerivatives
%%
function Derivatives(block)
x=block.ContStates.Data(1:6);
u1=block.InputPort(1).Data;
u2=block.InputPort(2).Data;
j1=6.12e-2;
j2=2e-2;
mg1=0.32;
mg2=0.48;
a1= 1.35e-2;
b1= 9.24e-2;
a2= 2e-2;
b2= 9e-2;
b1y= 6e-2;
b2y= 1e-2;
b1o= 6e-3;
b2o= 1e-3;
kgy= 5e-2;
k1= 1.1;
k2= 0.8;
t11= 1.1;
t10 = 1;
t21 = 1;
t20 =1;
xdot(1)=-x(2);
xdot(2)= -mg1/j1*sin(x(1))+mg2/j1*cos(x(1))-b1y*x(2)-
b2y*sign(x(2))+b1/j1*x(3)+a1/j1*x(3)*x(3)+kgy*x(6)*cos(x(5));
xdot(3)= k1*u1/t11-t10*x(3)/t11;
xdot(4)= k2*u2/t21-t20*x(4)/t21;
xdot(5)= x(6);
xdot(6)= -b1o*x(6)-b2o*sign(x(6))+b2/j2*x(4)+a2/j2*x(4)*x(4);
xdot=xdot';
block.Derivatives.Data=xdot;


%end Derivatives
```

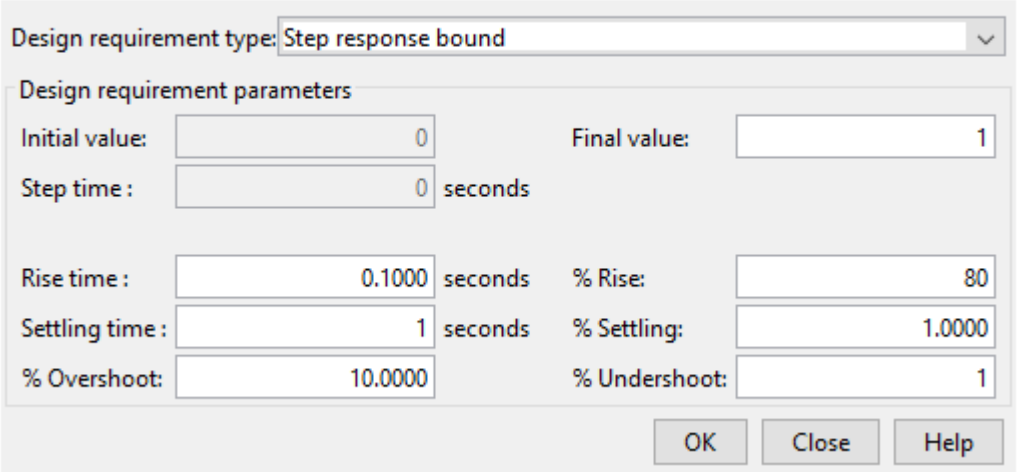
```
%%
%% Terminate:
%%   Functionality    : Called at the end of simulation for cleanup
%%   Required         : Yes
%%   C-MEX counterpart: mdlTerminate
%%
function Terminate(block)

%end Terminate
```

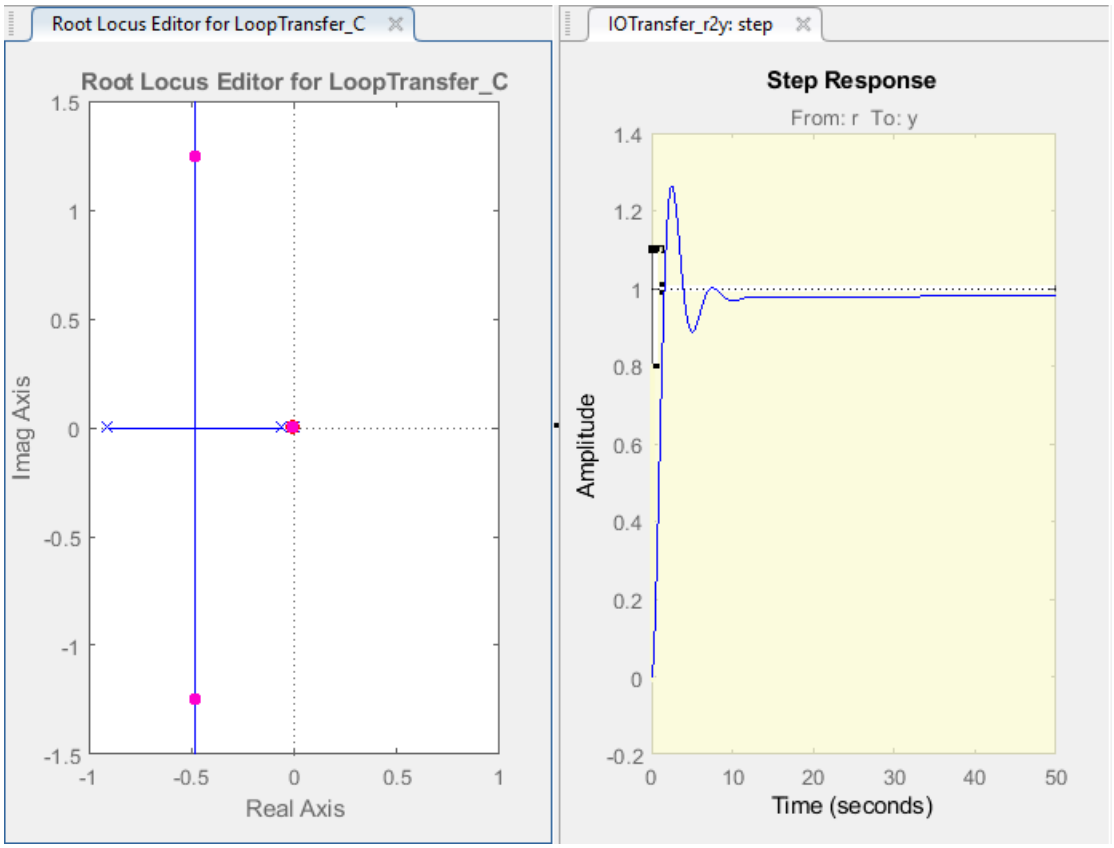
As shown above, we have used the S-function to define the six-state space model of the non-linear system by defining the equations as indicated in Derivatives. Furthermore, the initial conditions are set to be zeros. In other words, we will study only the forced response of the system regardless of what natural response it has.

The next code segment is related to the procedure of defining the following linearized model and converting it from state space representation to a transfer function and, thus, using SISO-tool to design a controller that can change the behavior of the system and make it acceptable.

```
Ay= [0 1 0;
    0 -0.06 1.51
    0 0 -0.91]
By= [0;
    0;
    1]
Cy= [1 0 0];

[numy, deny]= ss2tf(Ay,By,Cy,0);
Gy= tf(numy,deny)


Ao= [0 1 0;
    0 -0.006 4.5
    0 0 -1];
Bo= [0;
    0;
    1];
Co= [1 0 0];

[numo,deno]= ss2tf(Ao,Bo,Co,0);
Go= tf(numo,deno)
```

As shown above, two transfer functions are defined from the linearized model of the system. These two transfer functions are then analyzed through SISO-tool. The following figures shows the root locus as well as the step response of the first transfer function:



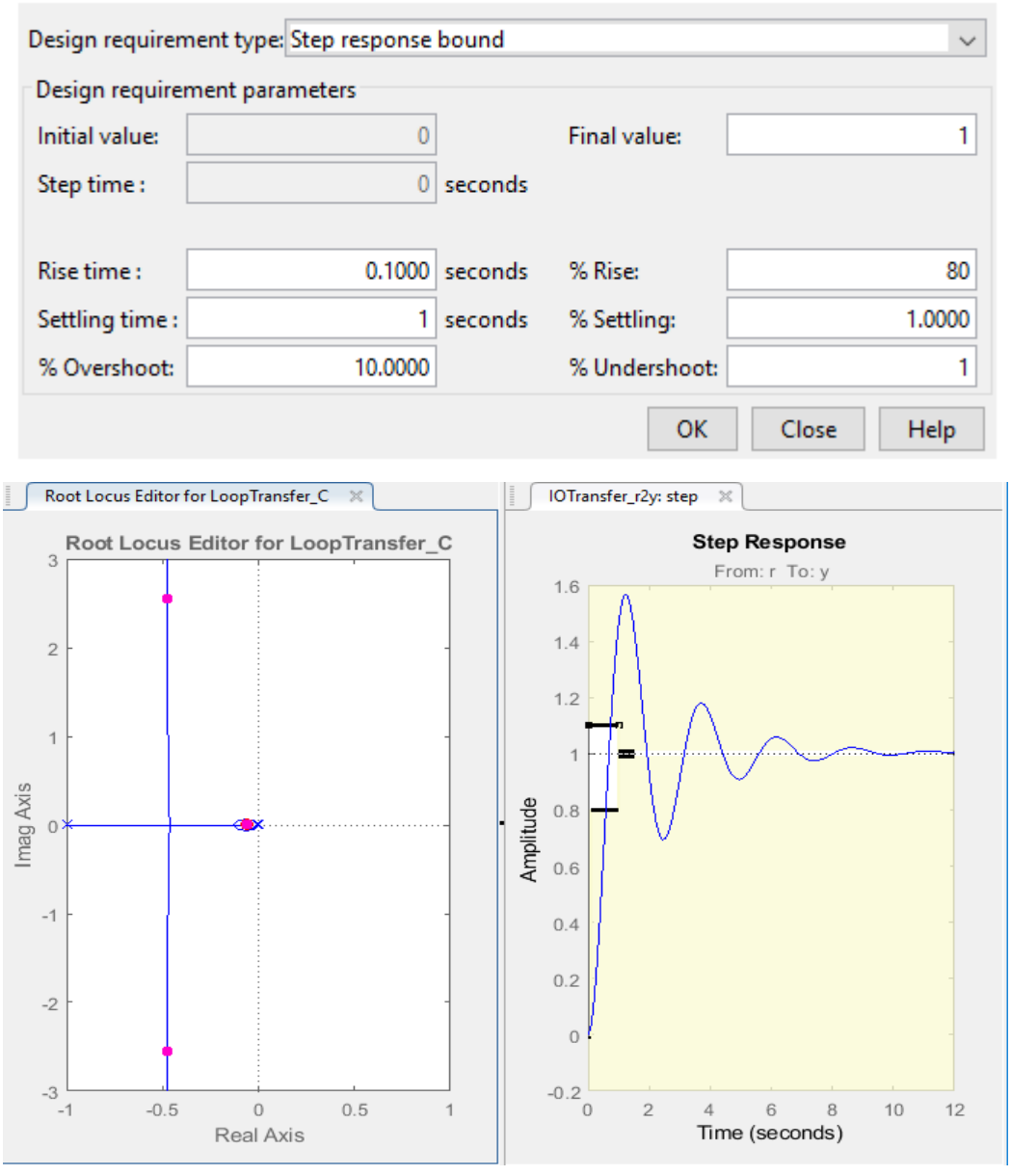


These results are provided using a Proportional-Derivative controller. The reason for not applying a Proportional-Derivative-Integral controller is that the system has three pools and no zeros and one of the pools is located at

the origin. In other words, the transfer function itself has an integrator, therefore, no need to increase the system type more, specially when it represents the linearized model. The controller acquired after this analysis has the following transfer function:

$$G(s)=0.2(s+0.05872)$$
$$Kd=0.2 \quad Kp=0.01$$

The analysis of the second transfer function can be summarized in the following figure which shows the step response as well as the root locus of the compensated transfer function:



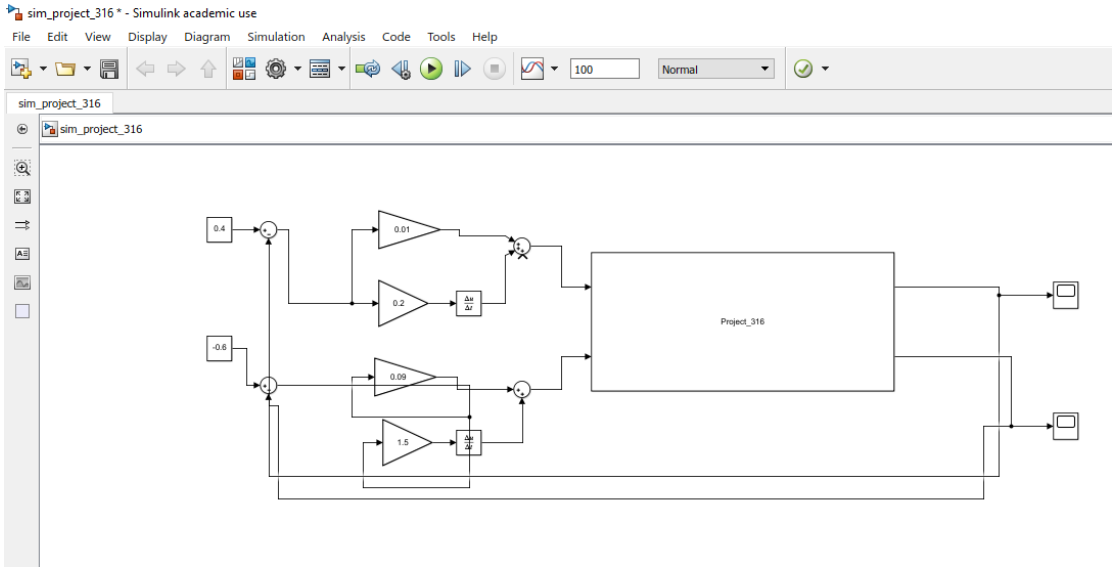
Although the step response shows a high overshot, the actual output of the combined system will exhibit zero overshot and will be illustrated in the following section of the report. This analysis is obtained from the following controller transfer function:

$$G(s)=1.5(s+0.06)$$
$$Kd=1.5 \quad Kp=0.09$$

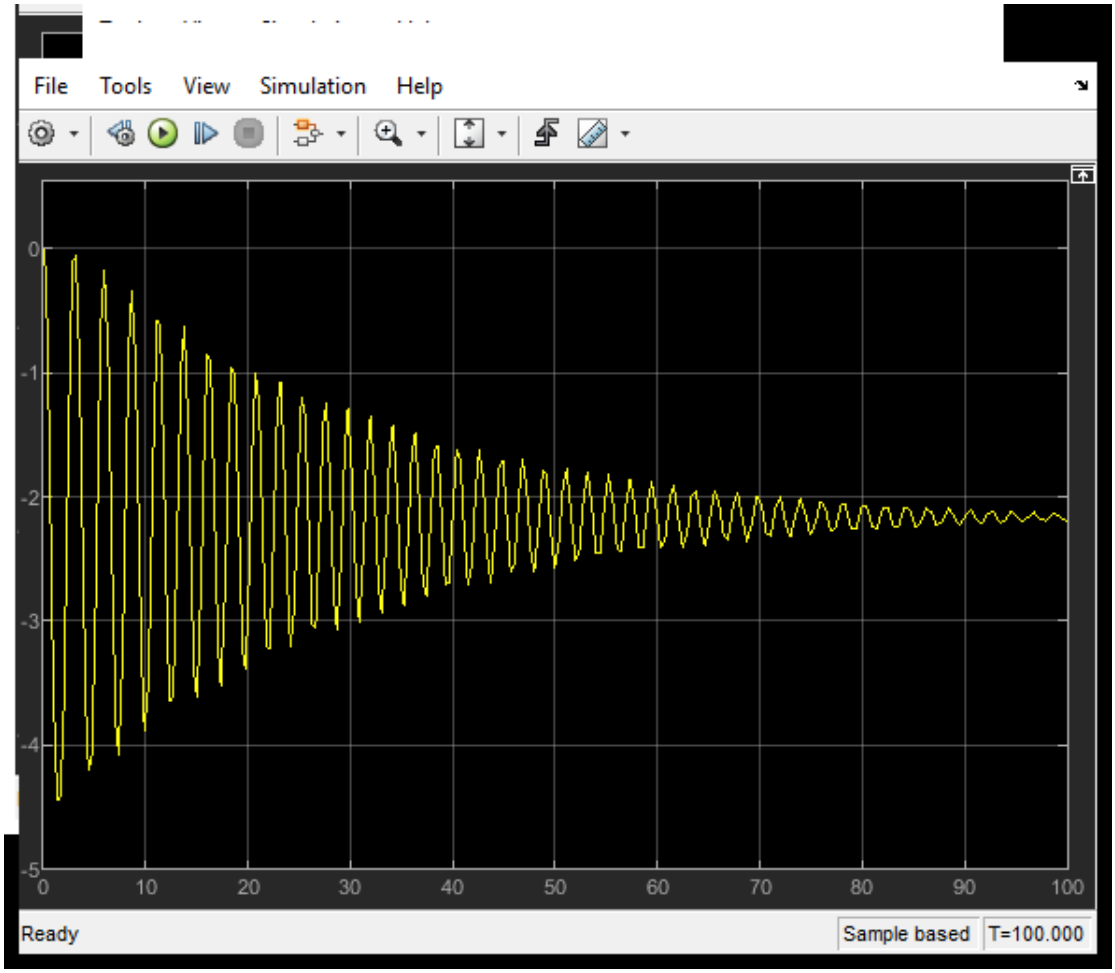
As shown, the controller is also a Proportional-Derivative and the reason is that the transfer function also contains an integrator, therefore no need to add a pole at the origin to eliminate the steady-state error.

After explaining the procedure of analyzing and designing the linear controllers based on the state space models, let us examine the effect of those controllers on the actual system simulated by the S-function file.

The following figure illustrate the Simulink Diagram:

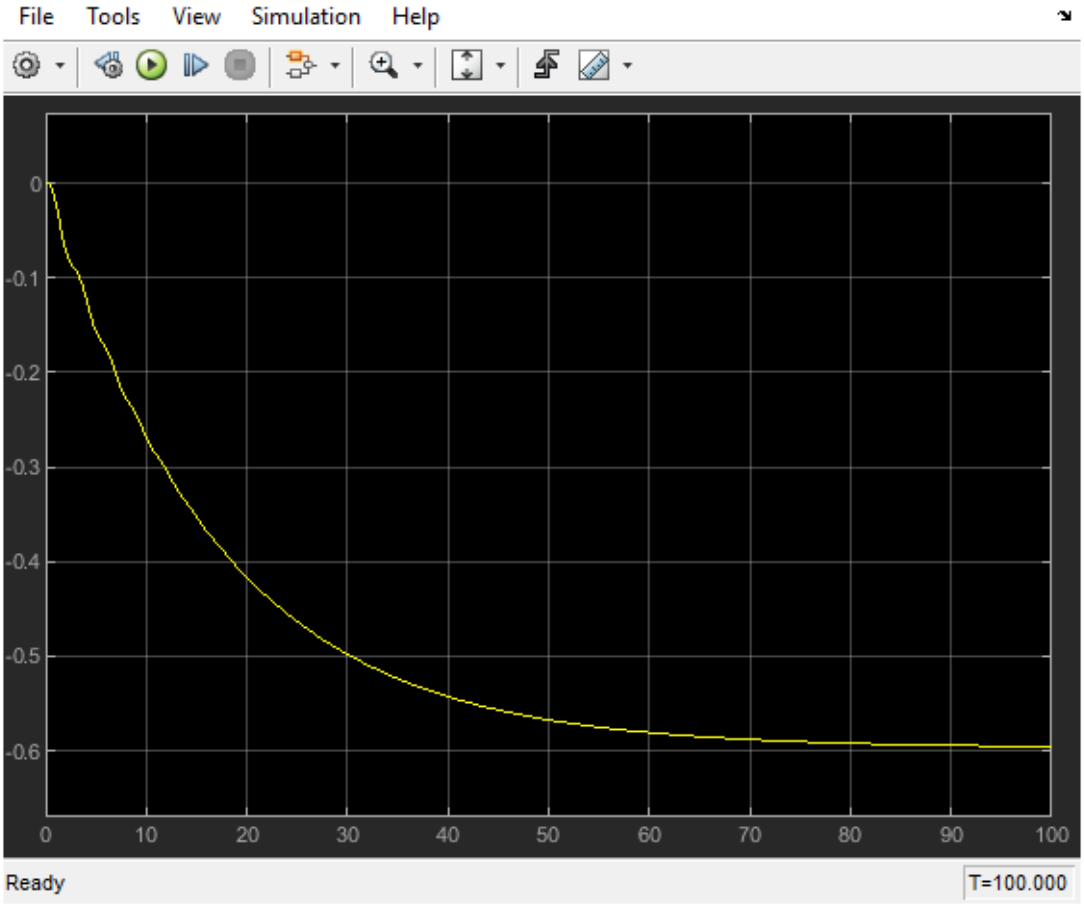


Now let us examine the first output shown below of the actual non-linear system that is describe by the above Simulink diagram:



Although the output exhibits a high error and overshot characteristics, the linear controller is still able to stabilize the first output. The error is estimated to be 240% which means that the linear controller cannot drive the actual output to the desired input.

Now let us investigate the second output of the actual system:



This output is satisfactory since the steady state error is approximately zero and the response exhibits no overshot at all (even though the linear controller shows 20% overshot!).

CONCLUSION

In this project we have investigated wither or not a non-linear system can be compensated using a linear controller after finding its linearized model. Our analysis was limited to a step input for both inputs of the system that is limited between -2.5 and 2.5. The two output of the actual system shows the following:

- A linear controller can be used to compensate a non-linear system; however it will not guarantee the desired output or even stability.

- The output overshot is independent of the linear controller as shown in the second output of the system.
- It appears that the two linear controllers internally affect both outputs simultaneously indicating that separating the two outputs using a linearized model is naïve.
- Furthermore, tiny changes in the controller parameters can lead to an unstable behavior for the actual system.