GINA CODY
SCHOOL OF ENGINEERING
AND COMPUTER SCIENCE

COEN 6331: Neural Networks

ASSIGNMENT -2

Submitted by:

Rohan Kodavalla (40196377)

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

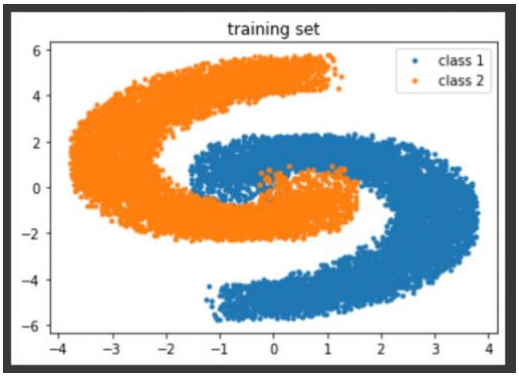
February 24, 2022

ABSTRACT

The objective of this assignment is to study a two-dimensional classification problem that involves nonconvex decision regions via Counter Propagation Network (CPN).

1. Generating two non-convex regions for training:

In order to generate the pattern above in the question a neural network architecture is designed with Keras, that will be used as the data to train the network.



As shown above data obtained from pattern is the dataset that contains two different spirals. Now from this data, choosing the samples in each of the regions, it is possible to train them and study CPN mechanism.

2. Selecting samples for the non-convex regions:

In this case, the number of samples selected are 7000 (random value) that is appended to value ‘n_points’ in the code.

The pattern generated is appended to a variable ‘n’ and from this ‘n’ value, the data sets for classes are derived and named ‘d1x’ & ‘d1y’ respectively in the code.

Further, to generate the datasets for training, all the points belonging to the pattern are normalized. But this normalization is done by arranging the data values in ‘np.hstack’, ‘np.vstack’ i.e., arranging horizontally and vertically in array stacks (just for simplification of coding process).

counter_propagation.py | config.py | generator_distributions.py | distributions.py

```python
import numpy as np
from sources import generator_distributions,distributions
# famous data
P_trust = 0.95
t_critical_student = 2.26
number_drills = 20
number_stoutnes = 24

# for hi_square
decrease_of_freedom = number_stoutnes - 1
level_main = 0.05
hi_critical_23_005 = 35.5  # 23 and 0.05
hi_critical_11_005 = 19.7

n_points=7000
noise=1.5

n = np.sqrt(np.random.rand(n_points,1)) * 400 * (1.10*np.pi)/290
d1x = -np.cos(n)*n + np.random.rand(n_points,1) * noise -1
d1y = np.sin(n)*n + np.random.rand(n_points,1) * noise -1

# X_values data
drills = np.vstack((np.hstack((d1x,d1y)),np.hstack((-d1x,-d1y)))) ### combiing 2 rows
#print(d1x[0:5])
#drills=[item for sublist in drills_ for item in sublist]
# generated data from function generate_normal_distribution()
#normal_distribution = generator_distributions.generate_normal_distributions(batch = len(d1x), size = len(d1x))
#normal_distribution=distributions.generate_normal_distribution(size=len(d1x))
normal_distribution= []
# result from comparison with data from table and generated normal distribution.
# every number belong to data from table.
# 1 - distribution is normal
# 0 - distribution isn't normal
#print(drills[0:5],d1y[0:5])
y_train = np.hstack((np.zeros(n_points),np.ones(n_points))) #### normalization is happening here,, combing 2 cols
```

Code for configuring data sets values

counter_propagation.py | config.py | generator_distributions.py | distributions.py

```python
'''
    Counter propagation neural network.
'''
import numpy as np
from sources.generator_distributions import mix_distributions, generate_normal_distributions, generate_expon_distributions, get_y_train
from sources import config
from math import sqrt
import time

X_values = config.drills
y_values = config.y_train

def sum_squares(arr):
    return sum([x ** 2 for x in arr])

# function normalization for vectors X
def normalization(arr_x, arr_y):
    sum_y_squares = sum_squares(arr_y)
    result = []

    for row_x in arr_x:
        middle_result = []
        sum_x_squares = sum_squares(row_x)
        for x in row_x:
            middle_result.append(round(x / sqrt(sum_x_squares + sum_y_squares), 2))

        result.append(middle_result)
    return result
```

Normalization

3. CPN weights, Kohonen & Grossberg Neurons:

a) Feeding the pattern data set as input to the learning of CPN network –

The pattern obtained in initial phase is now translated into values stored in d1x, d1y and are appended in an array to use it as a vector.

b) Selection of Kohonen, Grossberg Neurons-

In this case, number of Kohonen neurons chosen are 2 , and Grossberg neurons are 1.

c) Generating random weights-

Initially the weights for these 2 layers are chosen randomly, based on length of the vector (set to 4 in our example). These weights are denoted as ‘kohonen_weights’ and ‘grossberg_weights’ respectively.

```
30 class Counter_Propagation:
31     def __init__(self, X_values, y_values, kohonen_neurons = 2, grossberg_neurons = 1, len_x_vector = 4):
32         self.X_values = X_values
33         self.y_values = y_values
34         self.kohonen_neurons = kohonen_neurons
35         self.grossberg_neurons = grossberg_neurons
36         self.kohonen_weights = self.generate_weights(kohonen_neurons, len_x_vector)
37         self.grossberg_weights = self.generate_weights(grossberg_neurons, len_x_vector)
38 
39     # generate weights for each part of network
40     def generate_weights(self, num_neurons = 1, length=4):
41         yy = np.random.rand(num_neurons, length)
42         result = np.asarray(yy)
43         print("Generating Random weights------", result)
44 
45         if len(result) == 1:
46             return result[0]
47         return result
48 
```

Code for b), c) points above

d) Computing Euclidean distance-

Firstly, in order to compute this, the total number of samples in this case was chosen to be 7000 for which each of the samples in vector compute the shortest distance to given 2 neurons in Kohonen layer and 1 neuron in Grossberg layer.

The output of above (Euclidean distance) is stored in an array which generates almost 5000 values, and the winning node is printed corresponding to layer.

Post this the vectors of weights are updated with respect to the layer.

```python
# for shorter Evklide's way
def calculate_evklid_way(self, w_vector, x_vector):
    zz= sum([ ((w-x) ** 2) for w, x in zip(w_vector, x_vector)])
    return zz


# calculate net for Grossberg lay
def sum_activation(self, k_vector, w_vector):
    return sum([ k*w for k, w in zip(k_vector, w_vector)])

# update vector kohonen weights
def update_kohonen_weights(self, x_vector, w_vector, learning_rate = 0.7):
    weights = []

    for x, w in zip(x_vector, w_vector):
        w_new = w + learning_rate * (x - w)
        weights.append(w_new)
    return np.asarray(weights)

# update weights for grossberg lay
def update_grossberg_weights(self, y_value, w_value, learning_rate = 0.1, k = 1):
    w_new = w_value + learning_rate * (y_value - w_value) * k
    #print(f'grossberg update: {w_value}, {w_new}')
    return w_new

```

Code for d) point

e) <u>Training counter propagation neural network-</u>

The parameters chosen for this process is: - learning rate for kohonen layer , learning rate for Grossberg layer, epochs. Now the training of CPN network is done from below code snippets.

```python
# training counter propagation neural network
def fit(self, lr_kohonen = 0.7, lr_grossberg = 0.1, epochs = 10):
    print("kohonen_neurons used-----", self.kohonen_neurons, " || grossberg_neurons used------",self.grossberg_neurons )
    for epoch in range(epochs):
        if epoch % 5 == 0 and lr_kohonen > 0.1 and lr_grossberg > 0.01:
            lr_kohonen-= 0.05
            lr_grossberg -= 0.005

        good_counter = 0
        for x_vector, y_value in zip(self.X_values, self.y_values):
            kohonen_neurons = []

            for w_iter in range(len(self.kohonen_weights)):
                kohonen_neurons.append(self.calculate_evklid_way(x_vector, self.kohonen_weights[w_iter]))

            neuron_min = min(kohonen (method) index: (__value: Any, __start: SupportsIndex = ..., __stop: SupportsIndex = ...) -> int
            index = kohonen_neurons.index(neuron_min)

            for i in range(len(kohonen_neurons)):
                if i == index:
                    kohonen_neurons[i] = 1
                else:
                    kohonen_neurons[i] = 0

            self.kohonen_weights[index] = self.update_kohonen_weights(x_vector, self.kohonen_weights[index], learning_rate= lr_kohonen)

            # grossberg neurons
            #print("index -", index)
            self.grossberg_weights[index] = self.update_grossberg_weights(y_value, self.grossberg_weights[index], learning_rate=lr_grossberg)
            grossberg_neuron_out = int(round(self.sum_activation(kohonen_neurons, self.grossberg_weights)))
            good_counter += self.good_count(y_value, grossberg_neuron_out)

            #print(f'{y_value} : {grossberg_neuron_out}')

        print(f'Success training {epoch+1} epoch: {int(good_counter/len(self.y_values) * 100)}%')
```

```
    # work network on test values
    def evaluate(self, X_values, y_values):
        self.X_values = X_values
        self.y_values = y_values
        rr =[]
        rr1=[]
        good_counter= 0
        for x_vector, y_value in zip(self.X_values, self.y_values):
            kohonen_neurons = []

            for w_iter in range(len(self.kohonen_weights)):
                kohonen_neurons.append(self.calculate_evklid_way(x_vector, self.kohonen_weights[w_iter]))

            neuron_min = min(kohonen_neurons)
            index = kohonen_neurons.index(neuron_min)
            rr.append(neuron_min)
            rr1.append(index)
            for i in range(len(kohonen_neurons)):
                if i == index:
                    kohonen_neurons[i] = 1
                else:
                    kohonen_neurons[i] = 0

            grossberg_neuron_out = int(round(self.sum_activation(kohonen_neurons, self.grossberg_weights)))
            good_counter += self.good_count(y_value, grossberg_neuron_out)

        print(f'Success evaluate: {int(good_counter/len(self.y_values) * 100)}%')
        print("Shortest Euclidean Distance for each Training value------", rr)
        print("Winning node for each Training value------", rr1)
```

```
# work neural network
if __name__ == '__main__':
    #print(X_values[0:5],y_values[0:5],y_values[-5:-1])\


    net = Counter_Propagation(X_values, y_values, kohonen_neurons=2, grossberg_neurons=1, len_x_vector=len(X_values[0]))

    t_start = time.perf_counter()
    lrk=0.7
    lrg= 1
    print("Learning rate for Kohonen ----", lrk , " || Learning rate for Grossberg----" , lrg)
    net.fit(lr_kohonen=0.7, lr_grossberg=1, epochs=5)
    t_stop = time.perf_counter()

    print(f"Time of fit: {round(t_stop - t_start, 3)}")

    # testing on synthetic values
    normal_distr = generate_normal_distributions(1000, 24)
    expon_distr = generate_expon_distributions(450, 24)

    X_test = mix_distributions(normal_distr, expon_distr)
    y_test = get_y_train(X_test)

    print("Evaluate")
    net.evaluate(X_test, y_test)

```

f) Studying effects of learning rate, epochs for training accuracy and winning nodes :

Number of Kohonen neurons	Number of Grossberg neurons	Learning rate for Kohonen layer	Learning rate for Grossberg layer	Epochs	Time taken (min)	Training Accuracy	Winner nodes count for 0^{th} & 1^{st} Neuron
2	1	0.7	1	5	01:316	99%	0- 1000 1- 450
2	1	0.7	0.1	5	01:198	100%	0- 998 1- 452
2	1	1.4	1	10	2:52	99%	1-1450

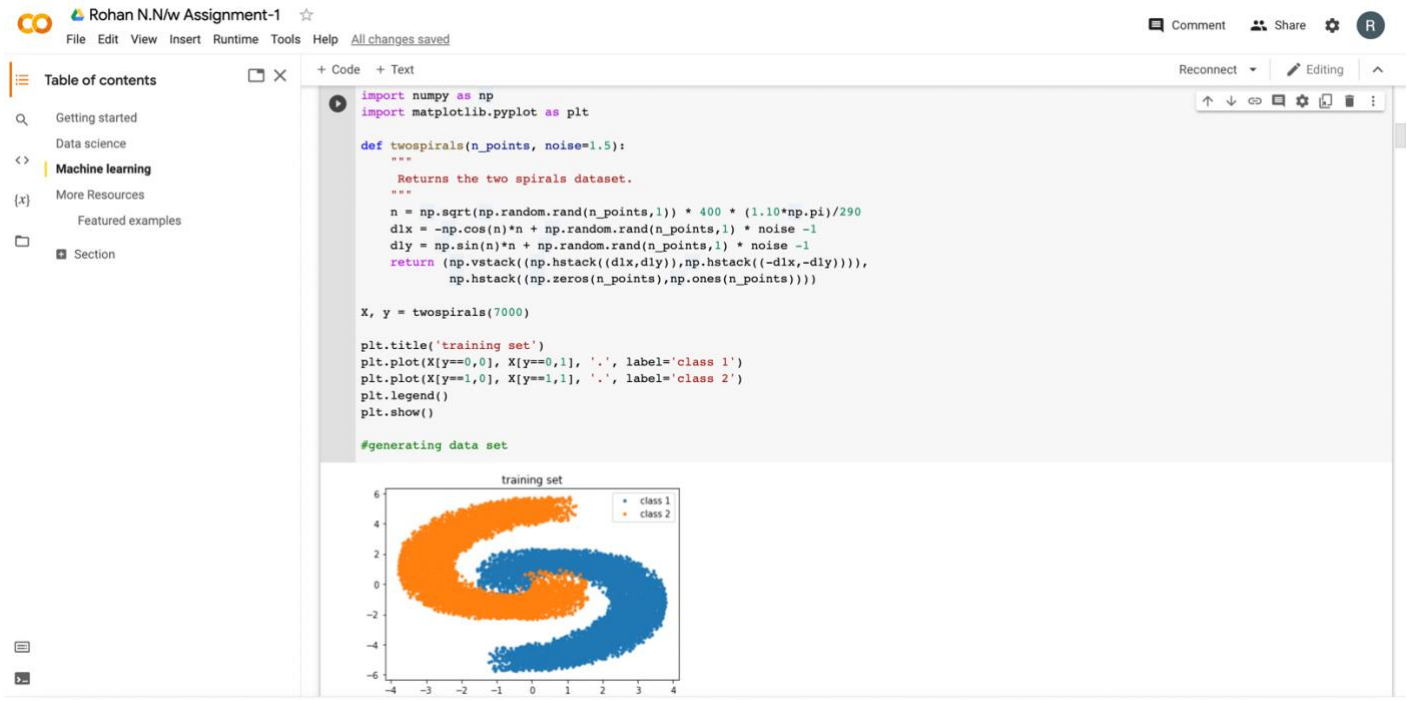
2	1	0.1	2	10	2.584	99%	0- 1000 1- 450

So overall, in this case we observe that for a same number of neurons, time taken for 5 and 10 epochs brings no change in training accuracy. Varying the learning rate for both layers only affects the number of winning nodes for the layers by a small amount.

So overall, maintaining the maximum accuracy and by varying learning rates for fixed number of neurons in Kohonen layer and Grossberg layer, the effects and working of a Counter Propagation Network CPN is studied.

APPENDIX: OUTPUTs // CODE

Generating data set –



Code output -1 :

```
!python3 /content/Neural_Networks/counter_propagation.py
```

```
Generating Random weights------ [[0.10451289 0.27258738]
 [0.42827559 0.00331772]]
Generating Random weights------ [[0.78307658 0.07353826]]
Learning rate for Kohonen ---- 0.7  || Learning rate for Grossberg---- 1
kohonen_neurons used----- 2  || grossberg_neurons used------ 1
Success training 1 epoch: 100%
Success training 2 epoch: 100%
Success training 3 epoch: 100%
Success training 4 epoch: 100%
Success training 5 epoch: 100%
Time of fit: 1.316
Evaluate
Success evaluate: 100%
Shortest Euclidean Distance for each Training value------ [7.0725088636247015, 452.2080374625433, 511.48328711717915, 538.5469875750558, 7.072
Winning node for each Training value------ [1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
```

```
[22] import numpy as np
     a = np.array([1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1
     unique, counts = np.unique(a, return_counts=True)
     dict(zip(unique, counts))

     {0: 1000, 1: 450}
```

Code output -2 :

```
!python3 /content/Neural_Networks/counter_propagation.py
```

```
Generating Random weights------ [[0.43753171 0.24688073]
 [0.79827069 0.54397452]]
Generating Random weights------ [[0.14482283 0.12450256]]
Learning rate for Kohonen ---- 0.7  || Learning rate for Grossberg---- 0.1
kohonen_neurons used----- 2  || grossberg_neurons used------ 1
Success training 1 epoch: 99%
Success training 2 epoch: 99%
Success training 3 epoch: 99%
Success training 4 epoch: 99%
Success training 5 epoch: 99%
Time of fit: 1.198
Evaluate
Success evaluate: 100%
Shortest Euclidean Distance for each Training value------ [482.43719246539933, 521.4448656117901, 528.0170317224772, 381.7258889739552, 488.38
Winning node for each Training value------ [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
```

```
[23] [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,

import numpy as np
a = np.array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1
unique, counts = np.unique(a, return_counts=True)
dict(zip(unique, counts))

{0: 998, 1: 452}
```

Code output -3 :

```
[8] !python3 /content/Neural_Networks/counter_propagation.py

Generating Random weights------ [[0.92550424 0.03577393]
 [0.56863496 0.29513009]]
Generating Random weights------ [[0.23586267 0.89567496]]
Learning rate for Kohonen ---- 1.4  || Learning rate for Grossberg---- 1
kohonen_neurons used----- 2  || grossberg_neurons used------ 1
Success training 1 epoch: 99%
Success training 2 epoch: 99%
Success training 3 epoch: 99%
Success training 4 epoch: 99%
Success training 5 epoch: 99%
Success training 6 epoch: 99%
Success training 7 epoch: 99%
Success training 8 epoch: 99%
Success training 9 epoch: 99%
Success training 10 epoch: 99%
Time of fit: 2.52
Evaluate
Success evaluate: 100%
Shortest Euclidean Distance for each Training value------ [463.46378153047146, 19.769211374042282, 458.3508770353984, 479.70528450649624, 19.7
Winning node for each Training value------ [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,


import numpy as np
a = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
unique, counts = np.unique(a, return_counts=True)
dict(zip(unique, counts))

{1: 1450}
```

Code output -4:

```
!python3 /content/Neural_Networks/counter_propagation.py
```

```
Generating Random weights------ [[0.02824559 0.65261106]
 [0.19534586 0.00488573]]
Generating Random weights------ [[0.30654624 0.14717478]]
Learning rate for Kohonen ---- 0.1  || Learning rate for Grossberg---- 2
kohonen_neurons used----- 2  || grossberg_neurons used------ 1
Success training 1 epoch: 99%
Success training 2 epoch: 99%
Success training 3 epoch: 99%
Success training 4 epoch: 99%
Success training 5 epoch: 99%
Success training 6 epoch: 99%
Success training 7 epoch: 99%
Success training 8 epoch: 99%
Success training 9 epoch: 99%
Success training 10 epoch: 99%
Time of fit: 2.584
Evaluate
Success evaluate: 100%
Shortest Euclidean Distance for each Training value------ [17.3115775475725, 319.1260375394163, 17.3115775475725, 260.8723078472197, 329.61607
Winning node for each Training value------ [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
```

```
[19] import numpy as np
     a = np.array([1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0
     unique, counts = np.unique(a, return_counts=True)
     dict(zip(unique, counts))
```

```
{0: 1000, 1: 450}
```

Magnetic levitation is a highly advanced technology. Magnetic Levitation is a process in which an object is suspended with no support other than magnetic field. The common point in all applications involving magnetic levitation is the lack of contact and thus no wear and friction. This increases efficiency, reduces maintenance costs, and increases the useful life of the system. The magnetic fields are utilized to reverse or balance the gravitational force and some other counter accelerations. Maglev can make frictionless, productive, out of sight advancements. The control system design and analysis methods are only applicable on linear systems. A Maglev system is a **nonlinear system** because the output is not proportional to the input. The control system should be treated as a linear system in a specific range so that the nonlinear properties can be compensated and the system can be linearized by utilizing various methods.

In control theory, there are two main divisions, classical and modern which directly apply case to case. **Classical control theory** is applicable to Linear time invariant systems and SISO systems which implies frequency domain approach. Proportional Integral Derivative (PID) controllers are the most common controllers that are programmed using classical control theory. **Modern control theory** is applicable to Linear, Non-Linear, time variant and time invariant systems. It makes use of time domain approach. State Feedback controllers are the most common controllers that are programmed using modern control theory.

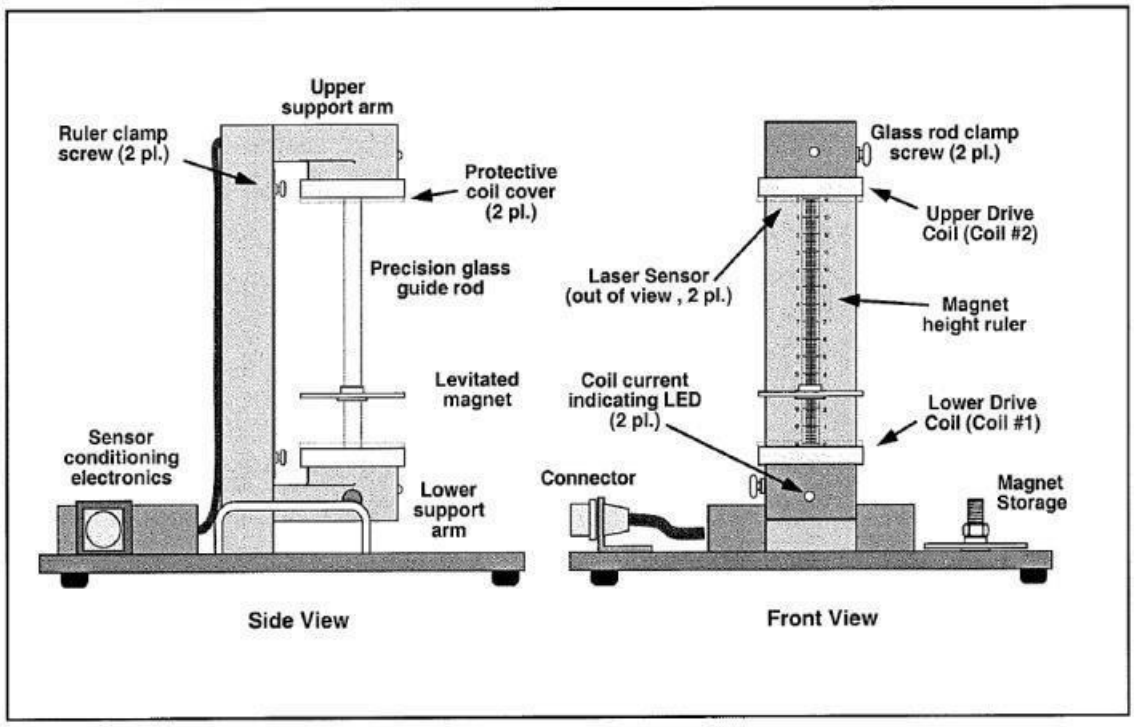


Figure 1: Magnetic Levitation Apparatus

Magnetic levitation method has been carried out in different applications for quite a while; significantly it has been generally utilized in transportation all over the world and in different fields, for example, in aviation applications, home, industry and in a few other engineering disciplines.

2 - BACKGROUND

Magnetic Levitation Model 730 consists of an upper coil and lower coil along with two suspended magnets. To levitate a single magnet, we make use of a repulsive force from the lower coil so that an open loop SISO system is created. If two coils are used then a MIMO system is implemented.

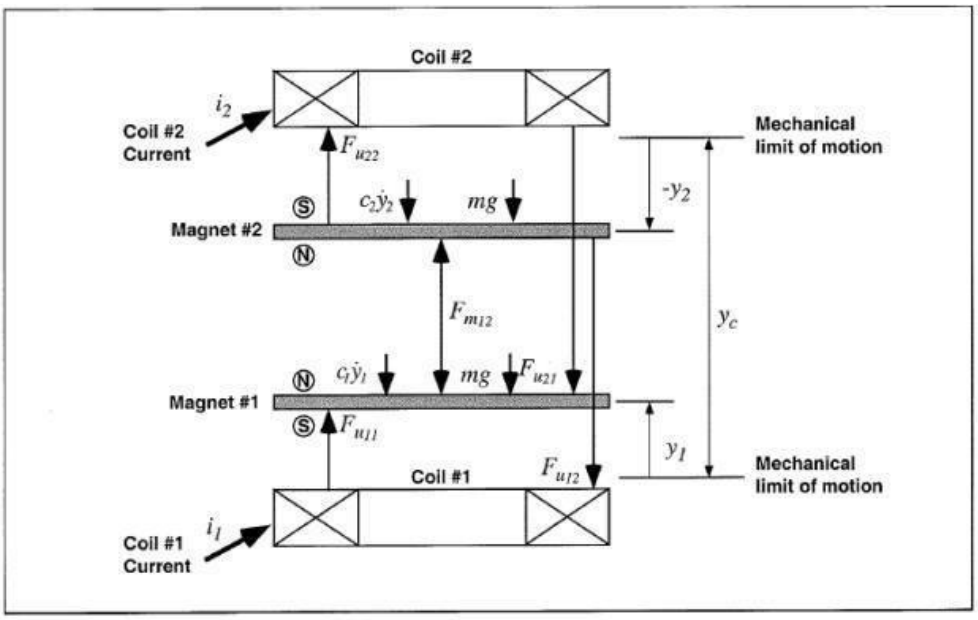


Figure 2: Free body diagram with dynamic configuration of the given plant

There are 5 forces acting in the system.

Fm12 is the force between each magnet.

F11 is the force applied to magnet 1 from coil 1.

F12 is the force applied to magnet 1 from coil 2.

F21 is the force applied to magnet 2 from coil 1.

F22 is the force applied to magnet 2 from coil 2.

i1 and i2 are the current inputs through the respective coils.

The mass of the magnet(m) is 0.12kg.

y1 is the distance between magnet 1 and coil 1.

y2 is the distance between magnet 2 and coil 2;

yc is the distance between the coils.

3 - PROBLEM STATEMENT

This project outlines experiments involves identifying the plant characteristics and parameters, implement a variety of control schemes, and demonstrate many important control principles. The motive of the project involves comparisons between modern and classical methods using PID and state feedback controllers respectively and assess which is more advantageous for the above described magnetic levitation system.

4 - DESIGN SPECIFICATIONS

In this project the system is linearized using Taylor's series expansion. In order to linearize a system, there are some initial conditions and parameters we consider to make our problem a bit simpler. The parameters help to proceed with the design and analysis of the system.

A system has to be linearized at particular points called **operating points.**

Operating Conditions are listed below:

Y10	2.00cm
Y20	-2.00cm
yc	12.00cm

Here are few more parameters listed below:

N	4
m	120g
g	9.81ms^-2
a	1.65
b	6.2
c	2.69
d	4.2

For PID Controllers,

We select some parameters into consideration as per design specifications that are listed below:

i. Settling Time <1.5 secs
ii. Percentage Overshoot <25%

For designing a PID Controller, the following table involving the effect of performance is also considered while varying the values to get the desired output.

PARAMETERS	Tr	Ts	Overshoot	SSE

Kp	Decrease	Small Change	Increase	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	No Change

5 - METHODOLOGY

To satisfy the design specifications, the input output equations of the system are obtained so that further analysis can be done. For the given plant, non-linear simplified equations of motions are provided, we make use of Taylor Series Approximation to convert them into linear equations. Thus, state and output equations of magnetic levitation system can also be obtained.

$Magnet$ 1 is given by,

$m\ddot{y}1 + c1\dot{y}1 + Fm12 = Fu11 - Fu21 - mg$-------- **(1)**

$Magnet$ 2 is given by,

$m\ddot{y}2 + c1\dot{y}2 + Fm12 = Fu22 - Fu12 - mg$------- **(2)**

In the above equations **(1)** and **(2),** we can eliminate the forces Fu12 & Fu21 are small and negligible as compared to the other forces in the system.

$m\ddot{y}1 + Fm12 = Fu11 - mg$ --- **(3)**

$m\ddot{y}2 + Fm12 = Fu22 - mg$ ----**(4)**

Where,

$Fu11 = \frac{u1}{a(y1+b)^4}$ --------**(5)**

$Fu22 = \frac{u1}{a(-y2+b)^4}$ ------**(6)**

$Fm12 = \frac{c}{(y12+d)^4}$ ------**(7)**

$y12 = yc + y2 - y1$----**(8)**

Since the equations (3) & (4) are non-linear, we make use of taylor series expansion to linearize them.

State variable is defined by,

$x1 = y1$

So, $\dot{x}1 = x2$ ---- **(9)**

X3 = y2

So, $\dot{x}3 = x4$ ---- **(10)**

From Equation(3),

$m\ddot{y}1 = Fu11 - mg - Fm12$

$\dot{x}2 = \frac{u1}{ma(y1+b)^4} \quad \frac{c}{m(yc-y1+y2+d)^4} - g$------ **(11)**

-

From equation(4),

$m\ddot{y}2 = Fm12 + Fu22 - mg$

$\dot{x}4 = \frac{u2}{ma(-y2+b)^4} + \frac{c}{m(yc-y1+y2+d)^4} - g$ ------**(12)**

Applying Linearization to state equation,

$\dot{x}2 = -[\frac{4c}{(\quad)^5} + \frac{4u1}{(\quad)^5}]\,\Delta x1 + \frac{4c}{(\quad)^5}\,\Delta x2 + \frac{1}{(\quad)^4}\,\Delta u1$--------(13)

($m\,yc-x1+x2+d$ $ma\ x1+b$ $m\,yc+x2-x1+d$ $ma\ x1+b$

$\dot{x}4 = [-\frac{4c}{m(yc-x1+x2+d)^4} + \frac{4u2}{ma(-x2+b)^5}]\,\Delta x2 + \frac{4c}{m(yc+x2-x1+d)^5}\Delta x1 + \frac{1}{ma(-x2+b)^4}\Delta u2$ -------(14)

$\dot{x}1 = x2$

$\dot{x}2 = -(k1+k2)\,\Delta x1 + (k12)\,\Delta x2 + (ku1)\,\Delta u1$

$\dot{x}3 = x4$

$\dot{x}4 = (k12)\,\Delta x1 + (k2-k12)\,\Delta x2 + (ku2)$

$\Delta u2$Where,

$k1 = \frac{4u1}{ma(x1+b)^5}$; $k2 = \frac{4u2}{ma(-x2+b)^5}$

$k12 = \frac{}{m(yc+x2-x1+d)^5}$

$ku1 = \frac{1}{ma(x1+b)^4}$; $ku2 = \frac{1}{ma(-x2+b)^4}$

We need to calculate the initial operating points of the system u1 and u2. Assuming that the second order derivative in equation (3) & (4) is zero, $y1'' = y2'' = 0$, the inputs u1 and u2 is given by,

$u1 = a(y1 + b)^4[\frac{c}{(yc+y2-y1+d)^4}$ $u2 = a(-y2 + b)^4[-$ c

$+ mg] = 8772970.892$ $\frac{}{(yc+y2-y1+d)^4} + mg] = 603793.5098$

6 - ANALYSIS

We make use of state and output equations to obtain transfer functions of the system. So that the plant behaviour can be analysed by implementing the step response, root locus and bode plot of the system.

6.1 - State and Output Equations

The parameters A,B,C and D can be written by comparing state and output equations with state space representation given by $\dot{X} = AX + BU$ and $Y = CX + DU$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -(k1 + k12) & 0 & k12 & 0 \\ 0 & 0 & 0 & 1 \\ k12 & 0 & k2 - k12 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ ku1 & 0 \\ 0 & 0 \\ 0 & ku2 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$xdot = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -4.78 & 0 & 3.31 * 10\text{^} - 7 & 0 \\ 0 & 0 & 0 & 1 \\ 3.31 * 10\text{^} - 7 & 0 & -0.3293 & 0 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1.117 * 10^{-6} & 0 \\ 0 & 0 \\ 0 & 1.117 * 10^{-6} \end{bmatrix} \begin{bmatrix} u1 \\ u2 \end{bmatrix}$$

$$y = \begin{bmatrix} y1 \\ y2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix}$$

6.2 - Transfer Function of Open Loop system

Transfer function is given by TF = $C(SI - A)^{-1} B + D$

```
>> sys_tf

sys_tf =

  From input 1 to output...
       1.117e-06 s^2 + 3.679e-07
   1:  -------------------------
        s^4 + 5.115 s^2 + 1.576

              3.706e-13
   2:  ----------------------
       s^4 + 5.115 s^2 + 1.576

  From input 2 to output...
              3.706e-13
   1:  ----------------------
       s^4 + 5.115 s^2 + 1.576

       1.117e-06 s^2 + 5.346e-06
   2:  -------------------------
        s^4 + 5.115 s^2 + 1.576
```

We neglect Y2/U1 and Y1/U2 since they have very small gains. Now, we take the remaining 2 transfer functions Y1/U1 and Y2/U2 into consideration and define them as Tf_1 and Tf_2.

6.3 - PID CONTROLLER

A proportional Integral Derivative controller(PID) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms. PID automatically applies an accurate and responsive correction to a control function.

The output of a PID controller, which is equal to the control input to the plant, is calculated in the time domain from the feedback error as follows:

$$u(t) = Kp\, e(t) + Ki \int e(t)dt + Kd \frac{de}{dt}$$

To get desired transient and steady state response of the closed loop system we designed the PID controller in MATLAB by using the function called “PID Tuner” for both open loop system that we have taken into consideration. The parameters obtained can be seen as below.

Parameters obtained for System Tf_1 are given below:

Controller Parameters

	Tuned
Kp	1216253636.8791
Ki	7545942733.7155
Kd	49008883.3636
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.0221 seconds
Settling time	0.251 seconds
Overshoot	24 %
Peak	1.24
Gain margin	-19.2 dB @ 12.4 rad/s
Phase margin	65.6 deg @ 57.4 rad/s
Closed-loop stability	Stable

Parameters obtained for System Tf_2 are given below:

Controller Parameters

	Tuned
Kp	4190401.3504
Ki	1291791.3729
Kd	3398277.7417
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.359 seconds
Settling time	5.8 seconds
Overshoot	15.2 %
Peak	1.15
Gain margin	-39.3 dB @ 0.616 rad/s
Phase margin	72.4 deg @ 3.97 rad/s
Closed-loop stability	Stable

Type equation here.

6.4 - Full State Feedback Control

We utilized the classical methods for control like PID to make our system stable however those had a few disadvantages as it couldn't do a lot of complex problems involving eigen values which were uncontrollable.

Modern Control theory recommends some imaginative ways of managing such issues where we can choose the position of our poles according to our requirements and that changes the system as per our needs to make it controllable and stable.

The main usage of state feedback controller is to regulate the output of the system. State feedback controller involves the use of the states to control the system.

We used the control law as:

$$u = rkr - k\mathrm{x}$$

The resulting state equation for control closed loop is given by:

$$\dot{x} = Ax + B(r.\ kr - k.\ x)$$

$$\dot{x} = (A - Bk)x + BrKr$$

We analyse the desired feedback control design in order to fulfil the design specifications.

We make use of percentage overshoot to find the damping ratio:

$$\xi = -\frac{\ln(\frac{\mathrm{Mp}}{100})}{\sqrt{(3.14)^2 + \ln{}^{\wedge}2(\frac{Mp}{100})}} = 0.41$$

Where,

Mp = 24 ; Ts = 0.251

$$Ts = \frac{4}{\xi * \mathrm{Wn}}$$

$$\mathrm{Wn} = 38.86$$

The system will be approximated with second order system characteristic polynomial as per the specifications:

$$S^2 + 2\xi\mathrm{WnS} + \mathrm{Wn}^2$$

$$S^2 + 31.86S + 1510.09$$

We find the poles to be: $-15.9363 \pm 35.0955i$

Since the system is of fourth order, it will have 4 poles and pole placement of system will be givenby:

Poles = [$-15.9363 + 35.0955i$ $-15.9363 - 35.0955i$ -47.80 -63.74]

The resulting gain K is given by,

```
K =

    1.0e+09 *

   -0.2349    0.0367   -2.6876   -0.0409
    3.1593    0.0647    2.3521    0.0938
```

Transfer functions of closed loop with gain feedback is given below:

```
Tf_fdbk =

  From input 1 to output...
            1.117e-06 s^2 + 0.000117 s + 0.002935
   1:  ---------------------------------------------------
       s^4 + 145.8 s^3 + 9973 s^2 + 4.592e05 s + 9.918e06

                  -8.079e-05 s - 0.003942
   2:  ---------------------------------------------------
       s^4 + 145.8 s^3 + 9973 s^2 + 4.592e05 s + 9.918e06

  From input 2 to output...
                   5.103e-05 s + 0.003354
   1:  ---------------------------------------------------
       s^4 + 145.8 s^3 + 9973 s^2 + 4.592e05 s + 9.918e06

            1.117e-06 s^2 + 4.584e-05 s - 0.0002877
   2:  ---------------------------------------------------
       s^4 + 145.8 s^3 + 9973 s^2 + 4.592e05 s + 9.918e06
```

6.5 - Full Order Observer Design

Considering the parameters, the desired 2nd order characteristic polynomial for the system will be:

$$S^2 + 129.4S + 4188.7$$

The desired observer's pole location is given by [-64.7, -64.7, -194.1, -258.8]

The command 'place' is used to provide the estimator gain matrix G, where the eigenvalues of $A - GC$ match the entries of the desired poles.

```
G =

   1.0e+04 *

    0.0259   -0.0000
    1.2554   -0.0016
   -0.0000    0.0323
   -0.0016    1.6744
```

Transfer functions of the open loop system with full order observer is given below:

```
tf_full_obs =

  From input 1 to output...
        -4.017e11 s^3 - 2.431e14 s^2 - 5.084e16 s - 5.441e17
   1:  ------------------------------------------------------
       s^4 + 728.1 s^3 + 2.079e05 s^2 + 2.809e07 s + 1.616e09

        -1.628e12 s^3 - 6.046e14 s^2 - 5.236e16 s - 1.298e18
   2:  ------------------------------------------------------
       s^4 + 728.1 s^3 + 2.079e05 s^2 + 2.809e07 s + 1.616e09

  From input 2 to output...
        1.555e12 s^3 + 5.037e14 s^2 + 4.576e16 s + 1.273e18
   1:  ------------------------------------------------------
       s^4 + 728.1 s^3 + 2.079e05 s^2 + 2.809e07 s + 1.616e09

        -2.329e12 s^3 - 8.506e14 s^2 - 1.031e17 s - 1.903e18
   2:  ------------------------------------------------------
       s^4 + 728.1 s^3 + 2.079e05 s^2 + 2.809e07 s + 1.616e09
```

Figure 3 - Step and Sinusoidal response of full order observer system

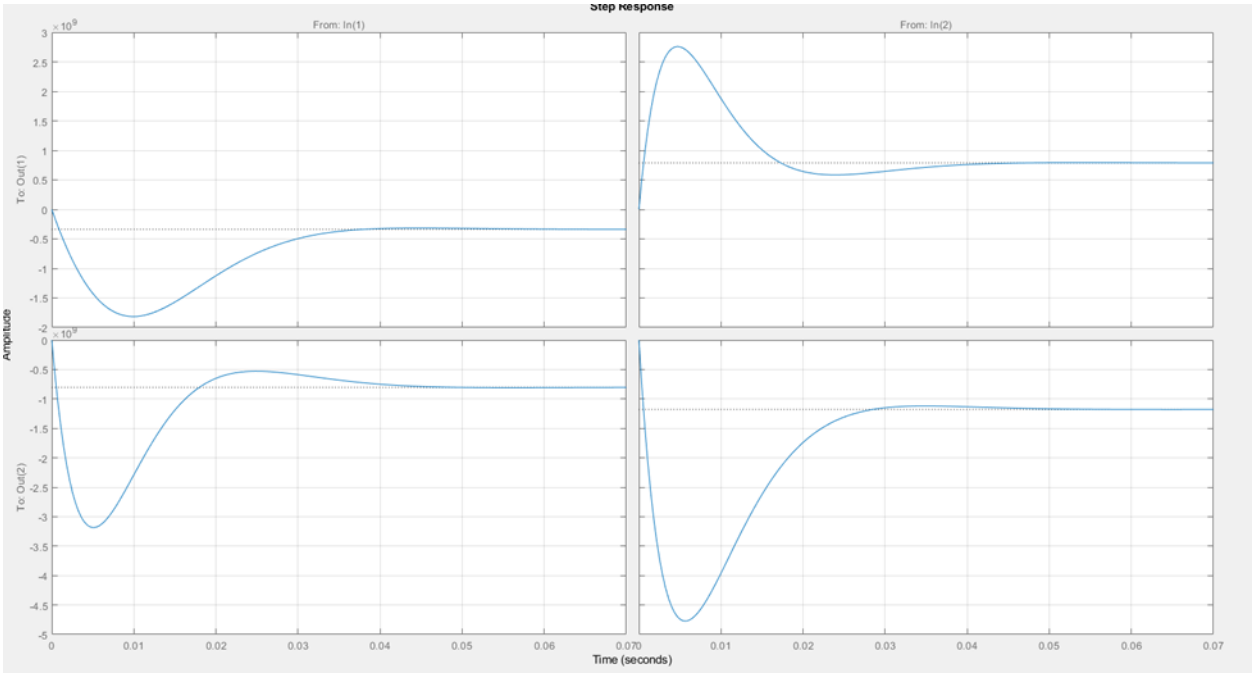


Figure 4 - Sinusoidal response of full order observer system Tf_o1

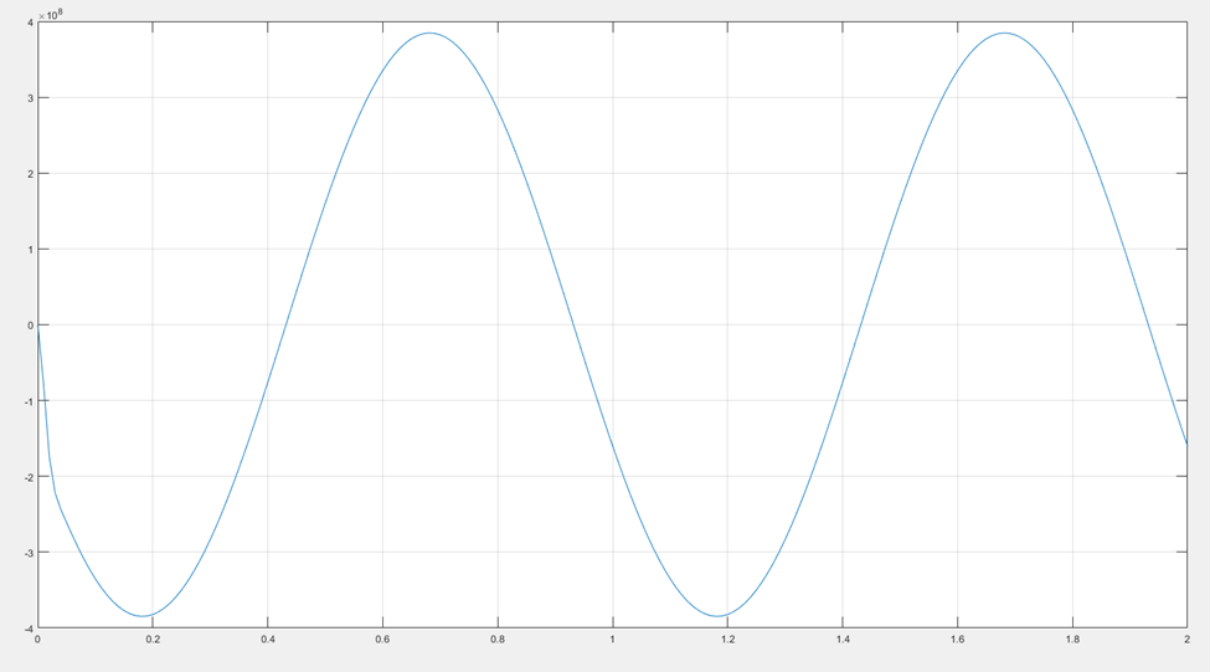
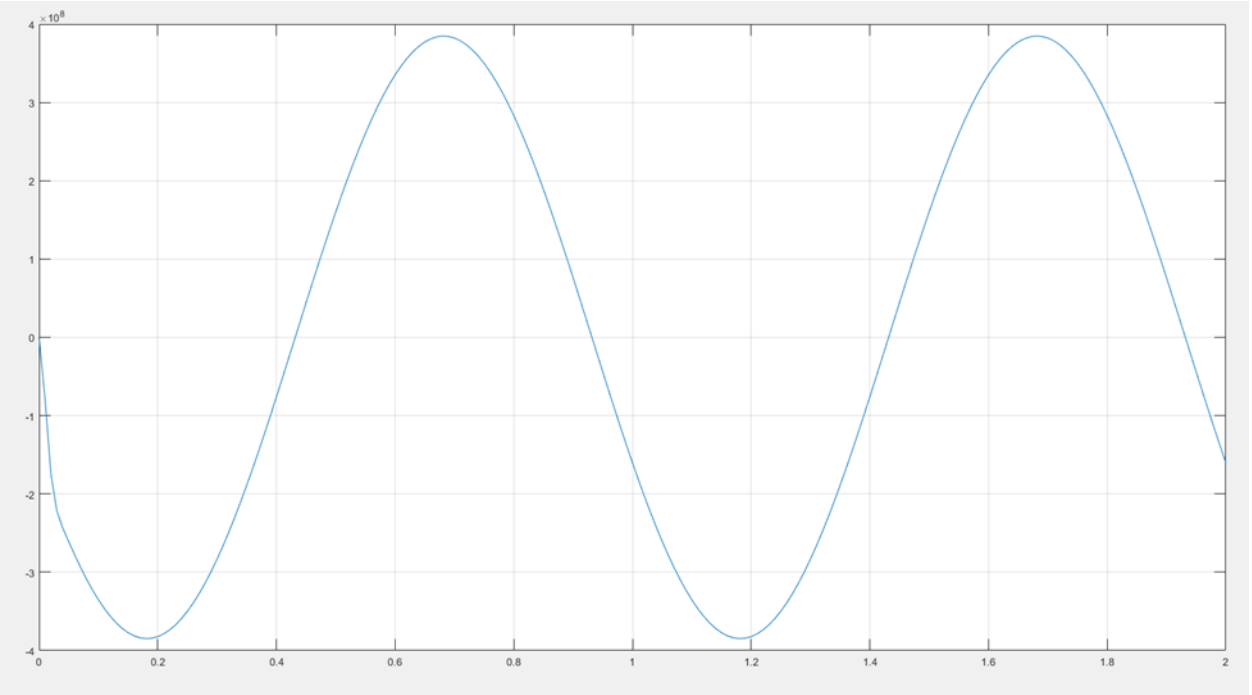


Figure 5- Sinusoidal response of full order observer system Tf_o2



6.6 - Reduced Order Observer

The order of the state observer can be decreased if there is no need to estimate a particular state as it is available in the output.

The desired second order characteristic polynomial of the system is given by:

$$S^2 + 129.4S + 4188.7 = (S + 64.72)^2$$

(S+64.72) is the reduced order of the system.

We implement Ackermann's formula to satisfy the above equation where it is chosen to give the estimator desired dynamics.

Transfer function of the reduced order observer is given below:

```
tf_red_ord_obs =

  From input 1 to output...
       -2.143e09 s^2 - 6.834e11 s - 6.719e12
   1:  ------------------------------------
             s^2 + 275.2 s + 2.123e04

       -7.349e09 s^2 - 8.264e11 s - 2.27e13
   2:  ------------------------------------
             s^2 + 275.2 s + 2.123e04

  From input 2 to output...
       5.334e09 s^2 + 6.93e11 s + 2.251e13
   1:  ------------------------------------
            s^2 + 275.2 s + 2.123e04

       -8.422e09 s^2 - 1.428e12 s - 2.86e13
   2:  ------------------------------------
             s^2 + 275.2 s + 2.123e04
```

We compare the poles and zeros of each transfer function we can determine the type of filter. For output 1 of each input, we have a high pass filter since the poles are behind the zeros. For output 2 of each input, we have a low pass filter since the zeros are behind the poles.

Figure 6 - Step and sinusoidal response of reduced order observer

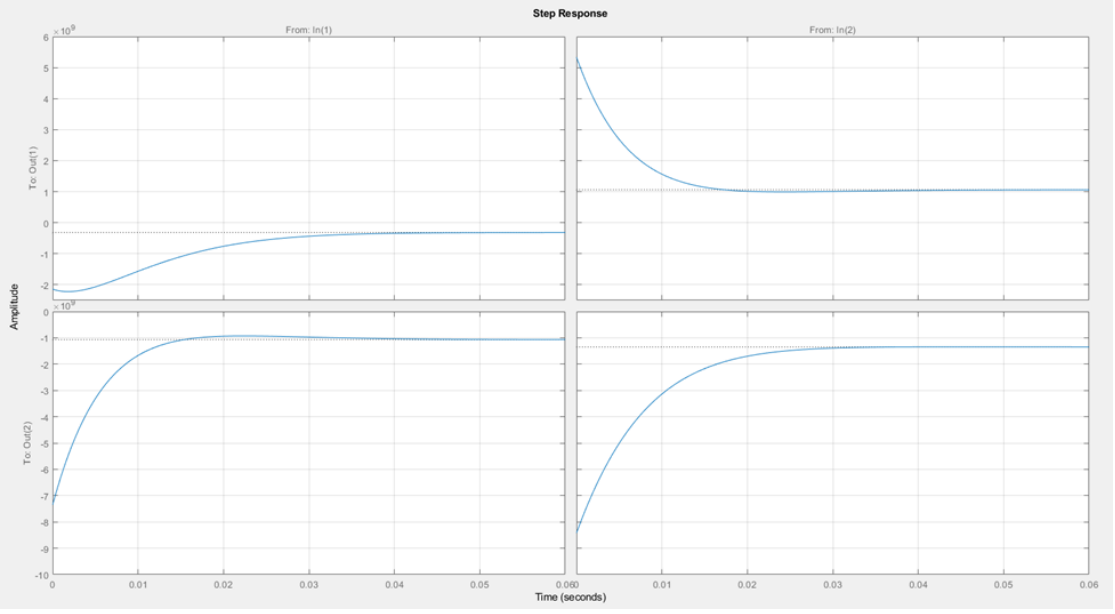


Figure 7 - Sinusoidal response of reduced order observer with system U1 and Y1

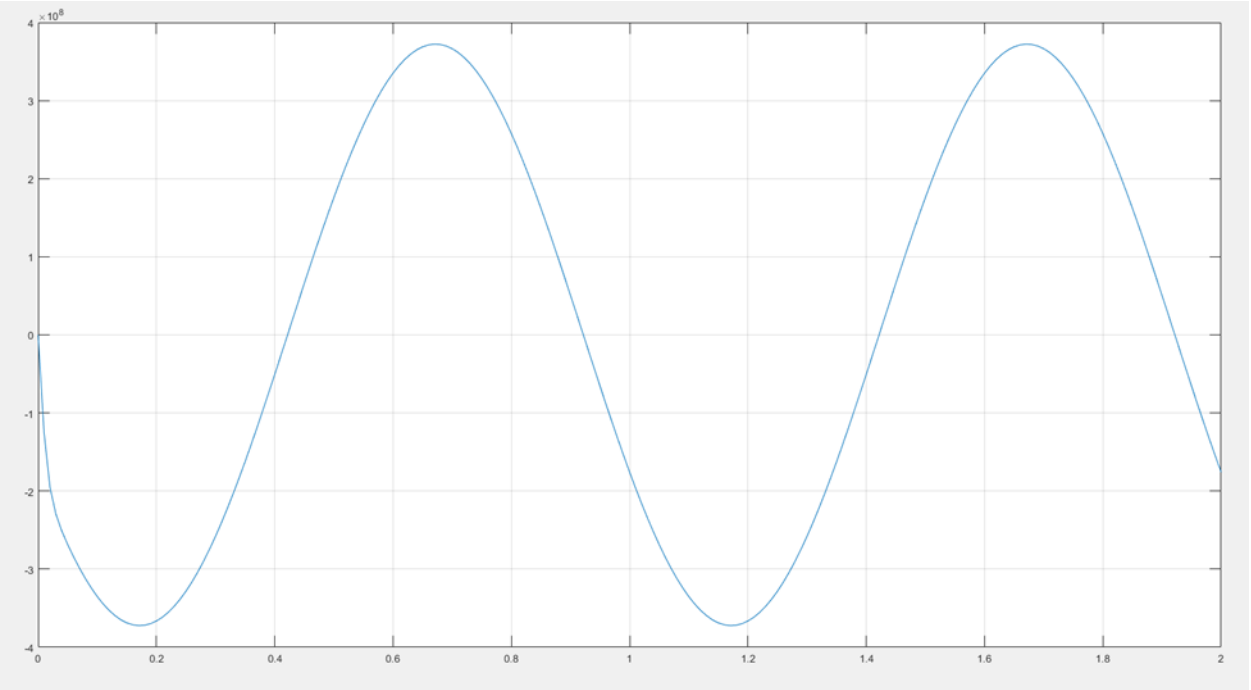
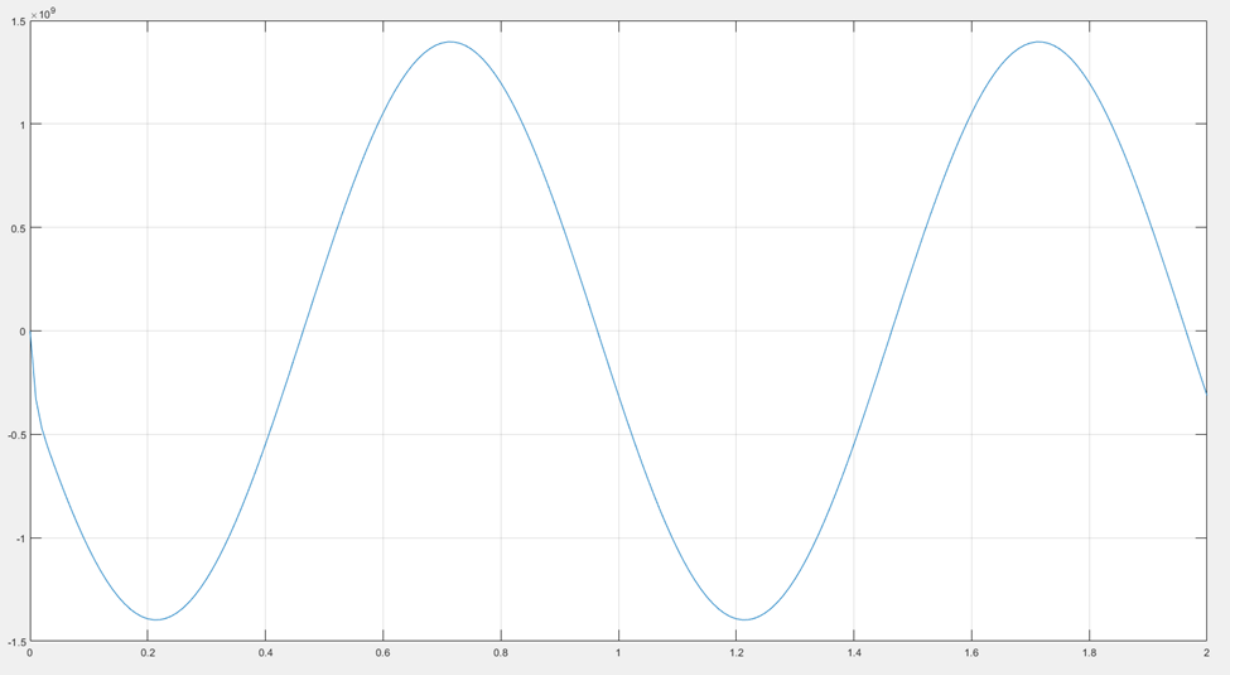


Figure 8 - Sinusoidal response of reduced order observer with system U2 and Y2



7-RESULTS

This section involves the figures, discussion, explanation, results and comparison of various topics involved in the project.

7.1 - Impulse and Step Response of Open Loop System

The impulse and step response of the system are obtained by assuming zero initial conditions.

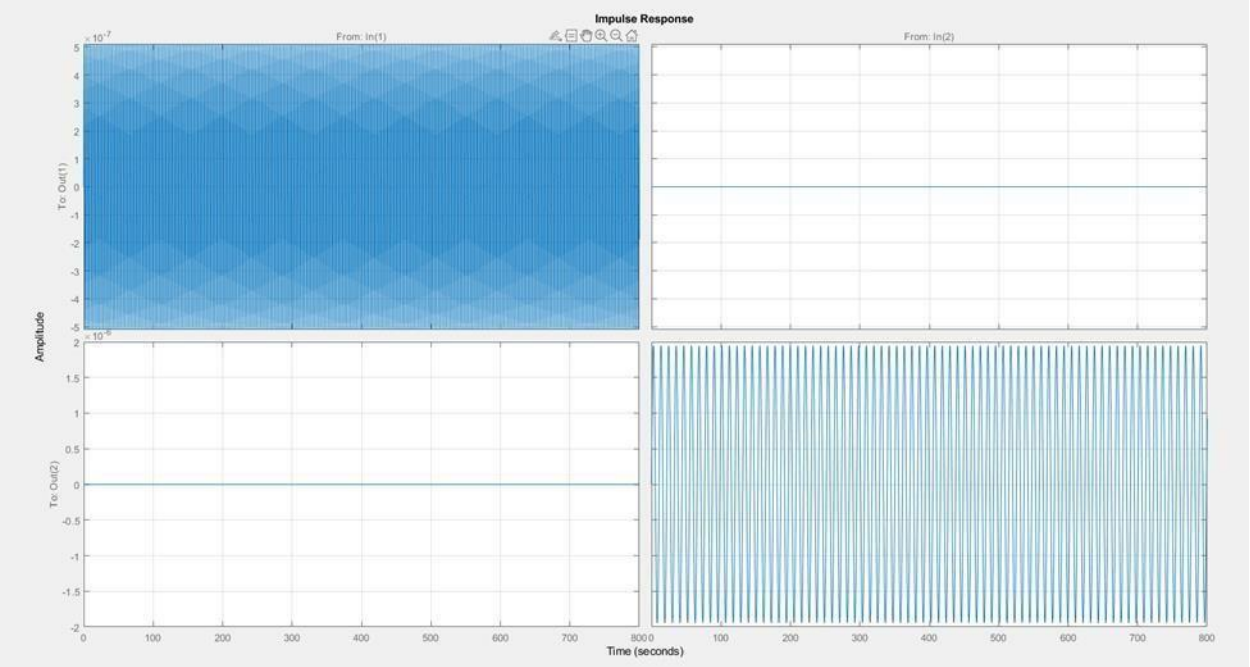


Figure 9 - Impulse response of open loop system

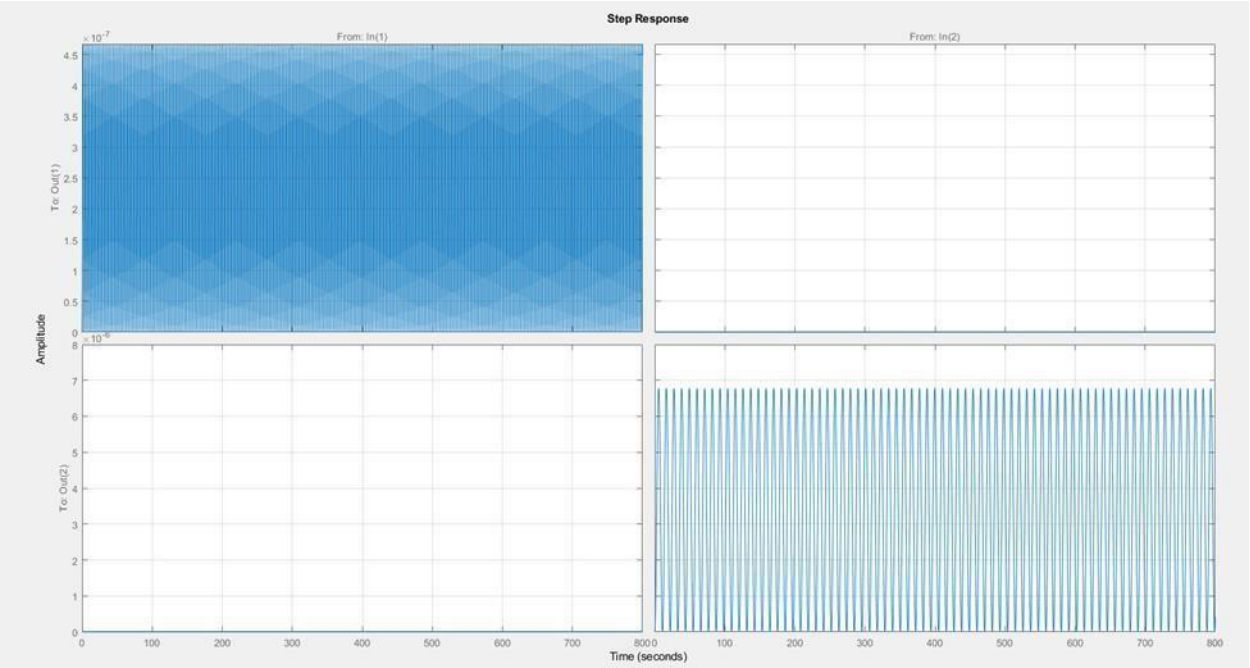


Figure 10 - Step response of open loop system

The gains of system Y2/U1 and Y1/U2 are really small which explains the straight line response. It can be seen from the above figure that the uncompensated system is unstable.

7.2 - Bode Plot and Root Locus of Open Loop System

Bode plot for the systems Tf_1 and Tf_2 are given below:

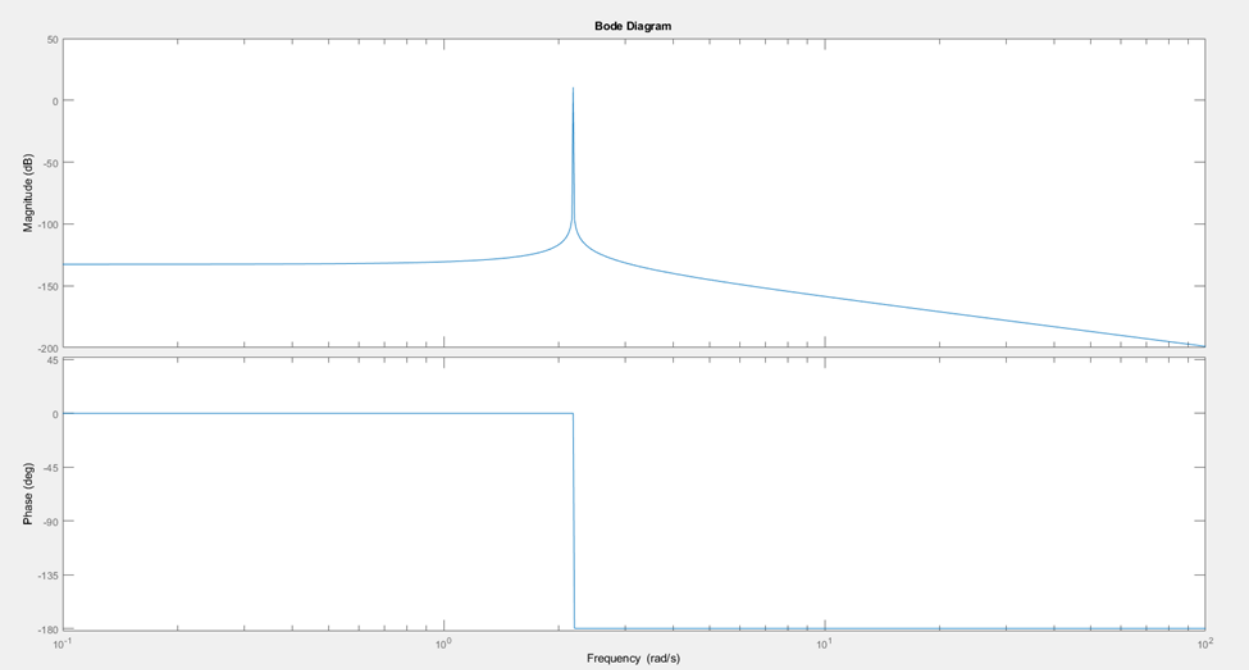


Figure 11 - Bode plot(SISO system) with Tf_1=Y1/U1

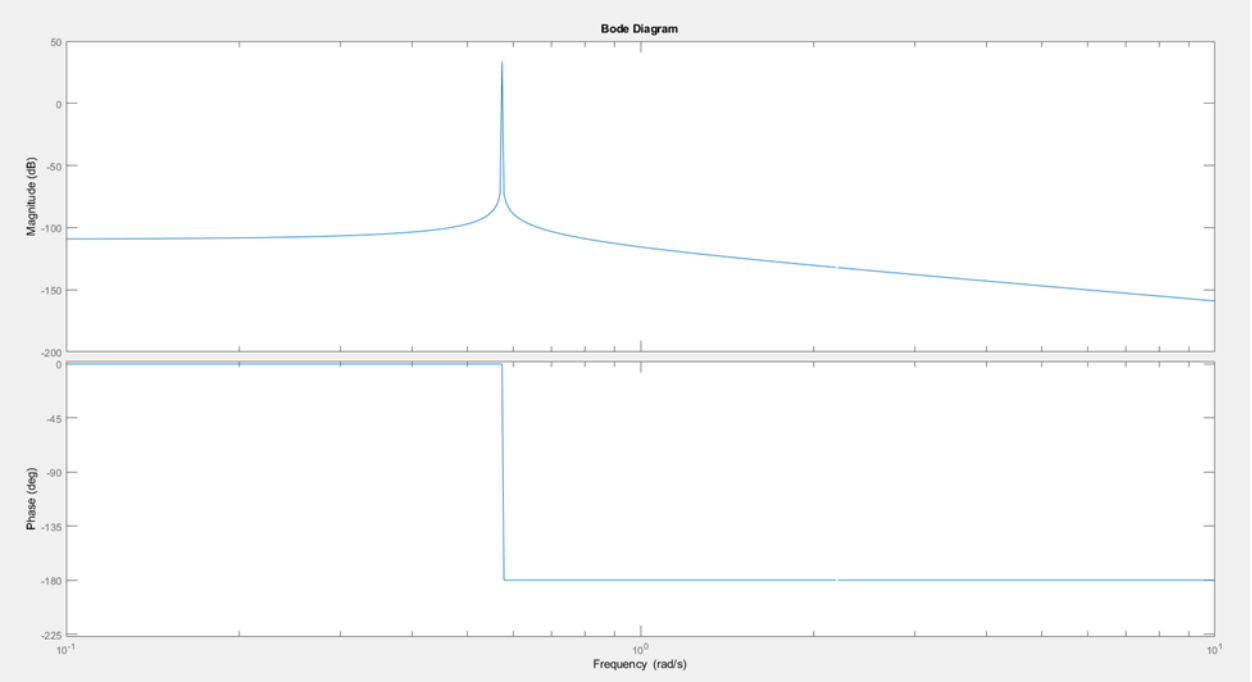


Figure 12 - Bode plot (SISO system) with Tf_2 = Y2/U2

7.3 - Root Locus for the systems Tf_1 and Tf_2

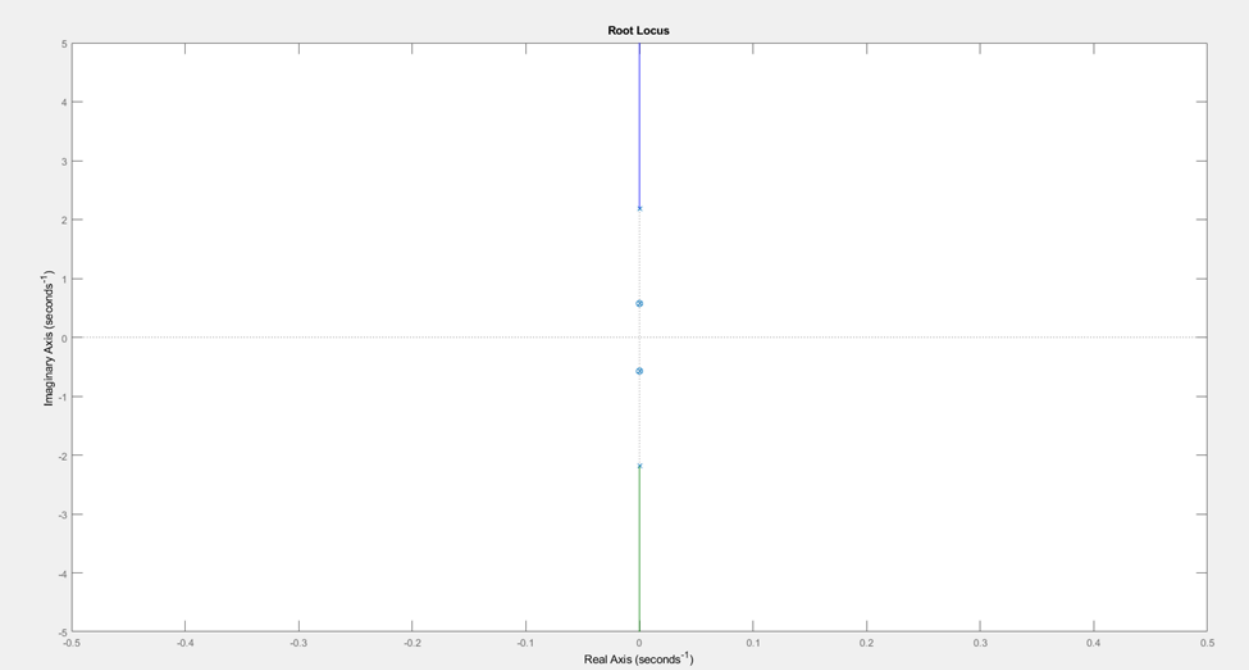


Figure 13 - Root locus(SISO system) with Tf_1

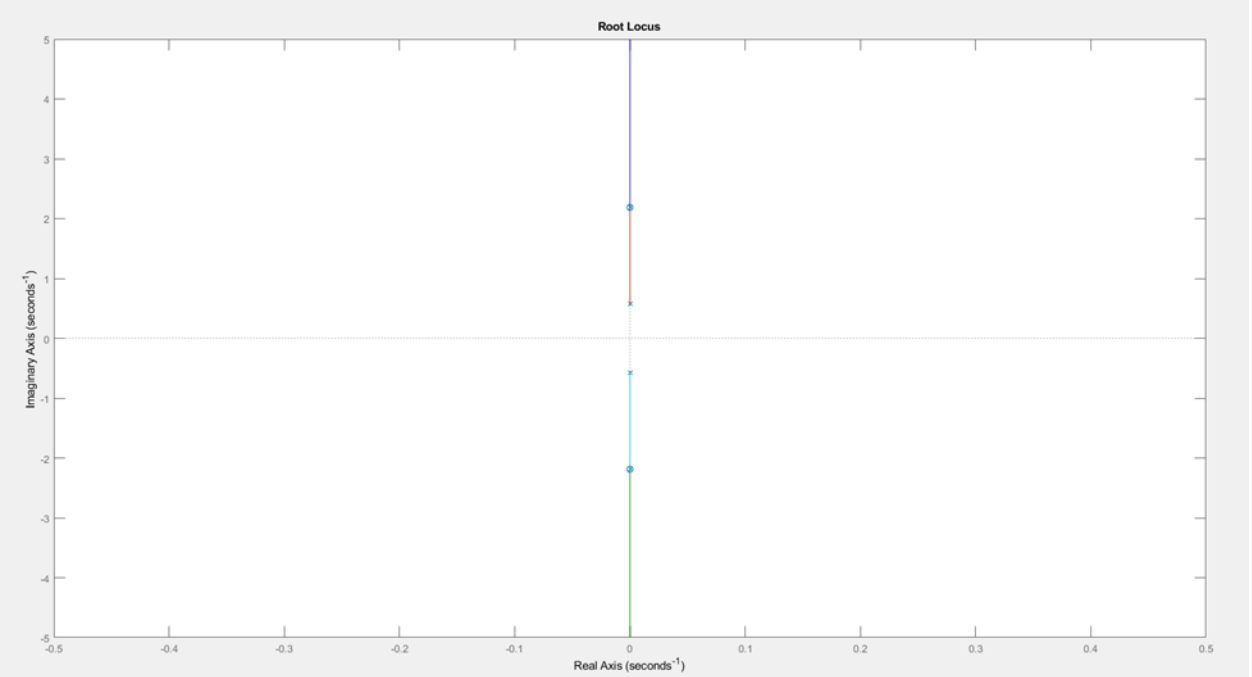


Figure 14 - Root locus(SISO system) with Tf_2

The root locus and bode plot shows further instability of the system.

7.4 - Controllable, Observable and Jordan Canonical forms

We checked the observability and controllability of open loop system by obtaining Ox and Cx matrices.

```
Cx =

   1.0e-05 *

         0         0    0.1117         0         0         0   -0.5346    0.0000
    0.1117         0         0         0   -0.5346    0.0000         0         0
         0         0         0    0.1117         0         0    0.0000   -0.0368
         0    0.1117         0         0    0.0000   -0.0368         0         0
```

Rank (Cx) = 4

```
Ox =

    1.0000         0         0         0
         0         0    1.0000         0
         0    1.0000         0         0
         0         0         0    1.0000
   -4.7854         0    0.0000         0
    0.0000         0   -0.3293         0
         0   -4.7854         0    0.0000
         0    0.0000         0   -0.3293
```

Rank(Ox) = 4

The obtained open loop system is observable and controllable since the rank of the matrix which are full rank matrices.

Observable canonical form of open loop system

```
osys =

  A =
         x1      x2      x3      x4
   x1     0       0       0  -1.576
   x2     1       0       0       0
   x3     0       1       0  -5.115
   x4     0       0       1       0

  B =
             u1         u2
   x1         1  1.442e+07
   x2         0          0
   x3         0  3.014e+06
   x4         0          0

  C =
              x1          x2          x3          x4
   y1          0   1.117e-06           0  -5.346e-06
   y2          0           0           0   3.706e-13

  D =
       u1  u2
   y1   0   0
   y2   0   0
```

Controllable canonical form of open loop system

```
csys =

  A =
          x1       x2       x3       x4
   x1       0        1        0        0
   x2       0        0        1        0
   x3       0        0        0        1
   x4  -1.576        0   -5.115        0

  B =
                u1           u2
   x1            0            0
   x2   1.117e-06            0
   x3            0            0
   x4  -5.346e-06   3.706e-13

  C =
              x1          x2          x3          x4
   y1          1           0           0           0
   y2  1.442e+07           0  3.014e+06           0

  D =
       u1  u2
   y1   0   0
   y2   0   0
```

Jordan Canonical form of matrix

```
J =

   0.0000 + 2.1875i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.5739i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 2.1875i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.5739i
```

7.5 - Step response, square wave and sinusoidal response of the closed loop systems with PID

Step responses of the closed loop system with plant Transfer functions Tf_1 and Tf_2 is shown below.

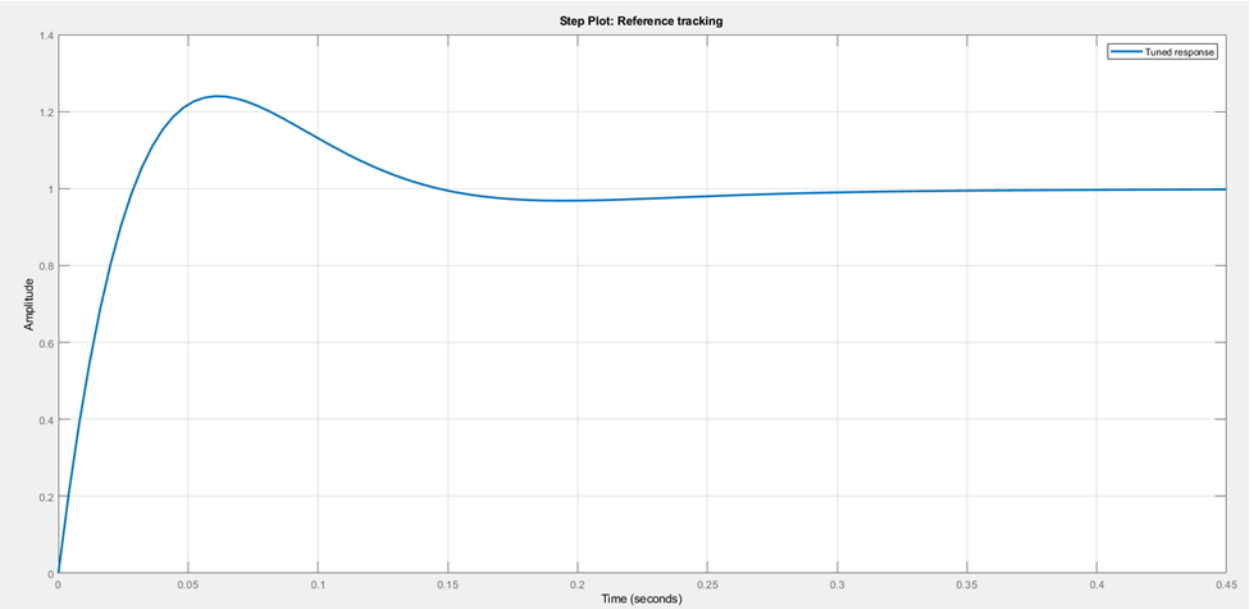


Figure 15 - Step response of closed loop system with Tf_1

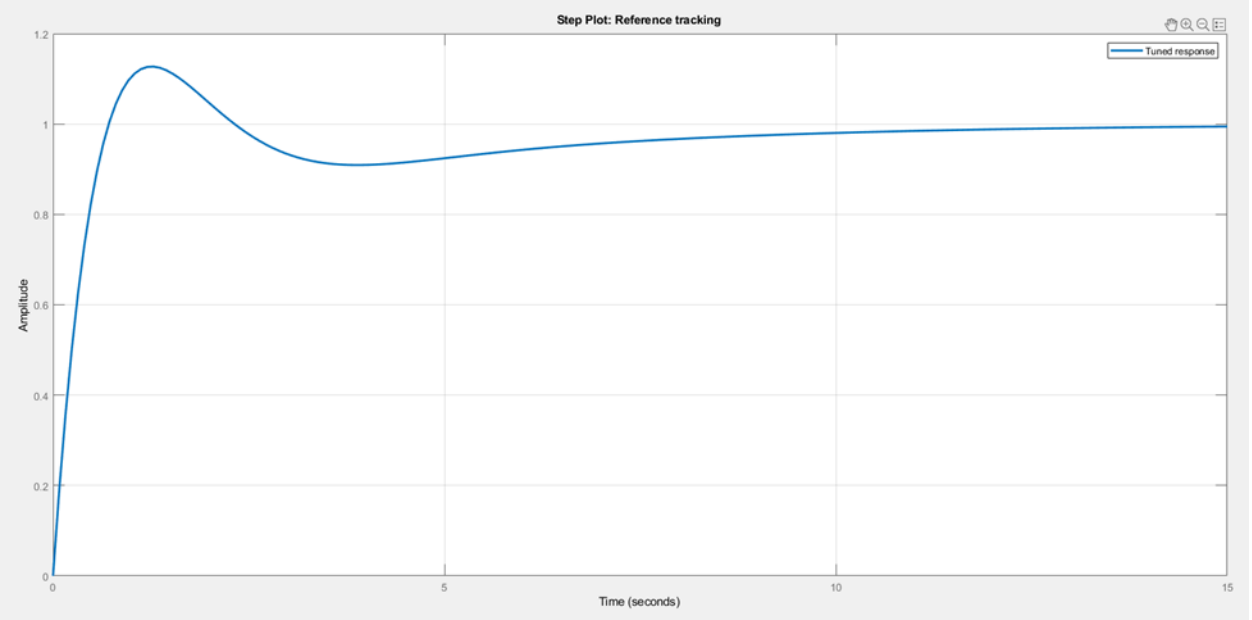


Figure 16 - Step response of closed loop system with Tf_2

Square wave responses for the closed loop system with plant Transfer functions Tf_1 and Tf_2

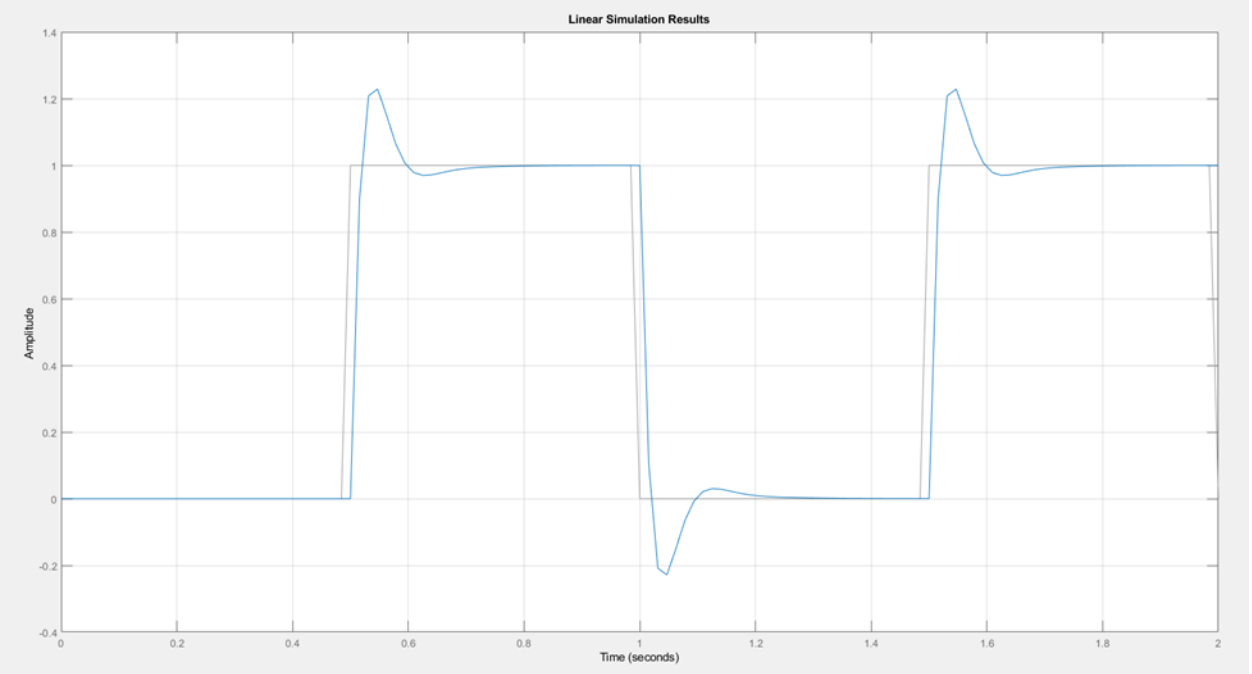


Figure 17 - Square response of closed loop system with Tf_1

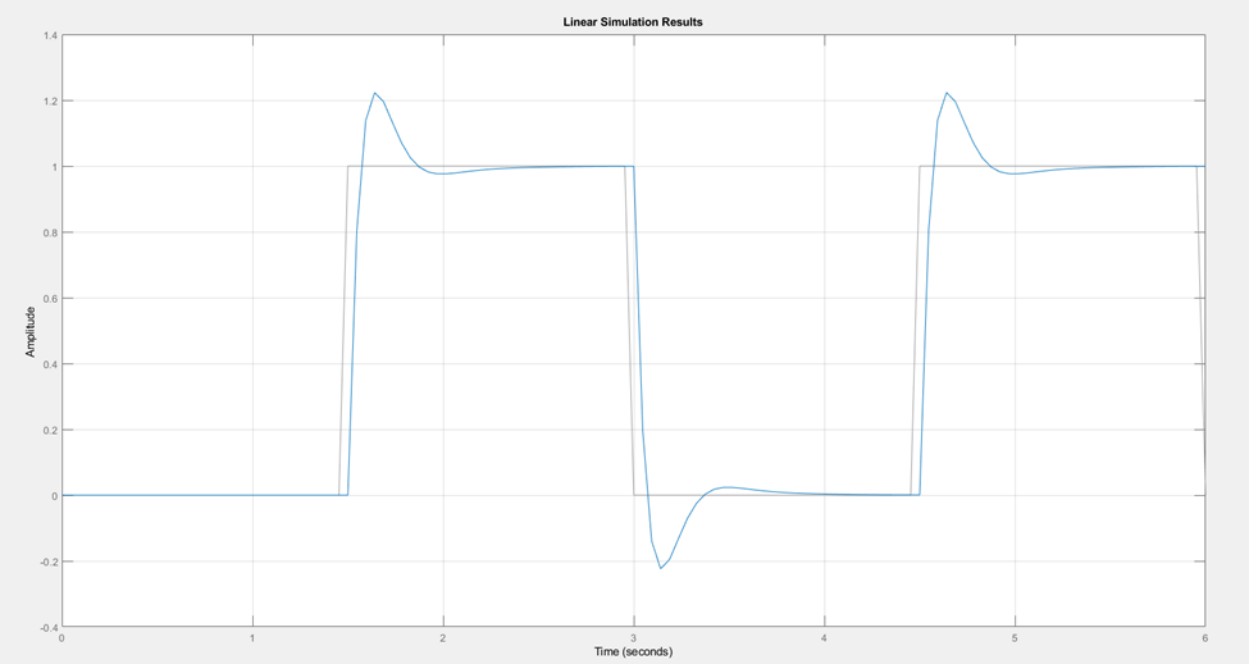


Figure 18 - Square response of closed loop system with Tf_2

Sinusoidal responses for the closed loop system with plant Transfer functions Tf_1 and Tf_2

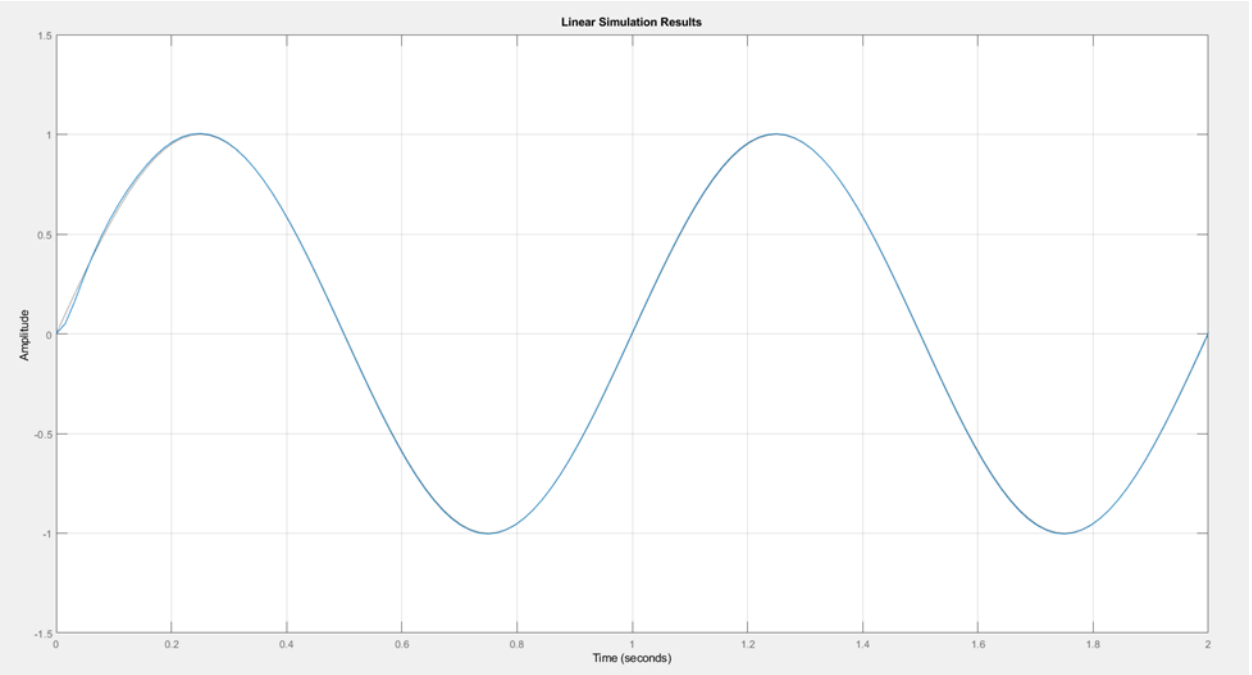


Figure 19 - Sinusoidal response of closed loop system with Tf_1

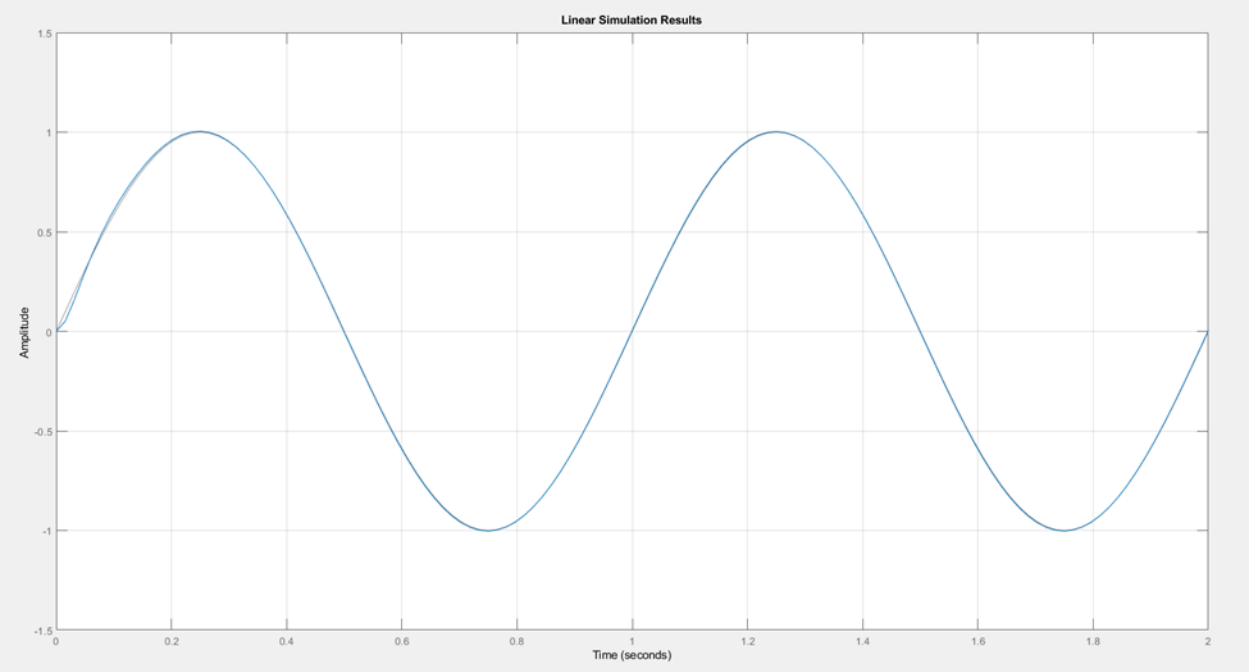


Figure 20 - Sinusoidal response of closed loop system with Tf_2

7.6 - Control Input Signals

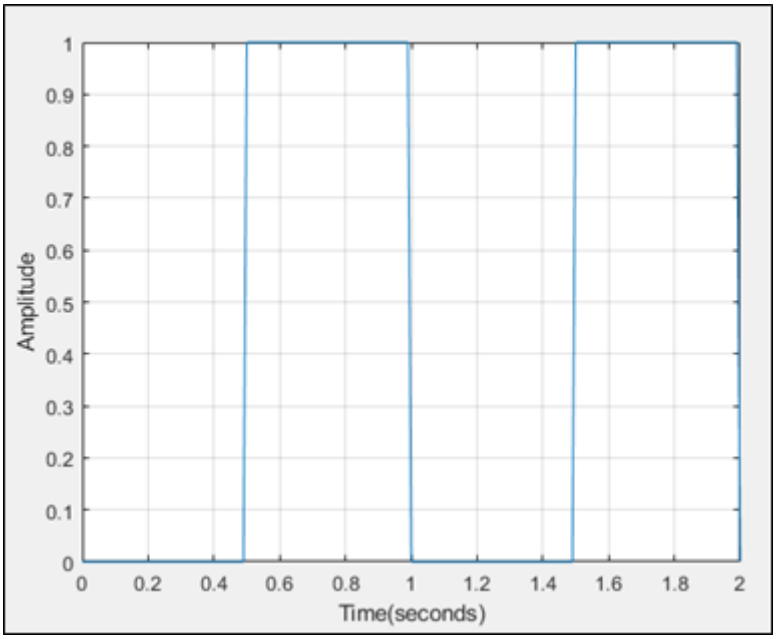


Figure 21 - Control input signal for square input of both systems (Tf_1,Tf_2)

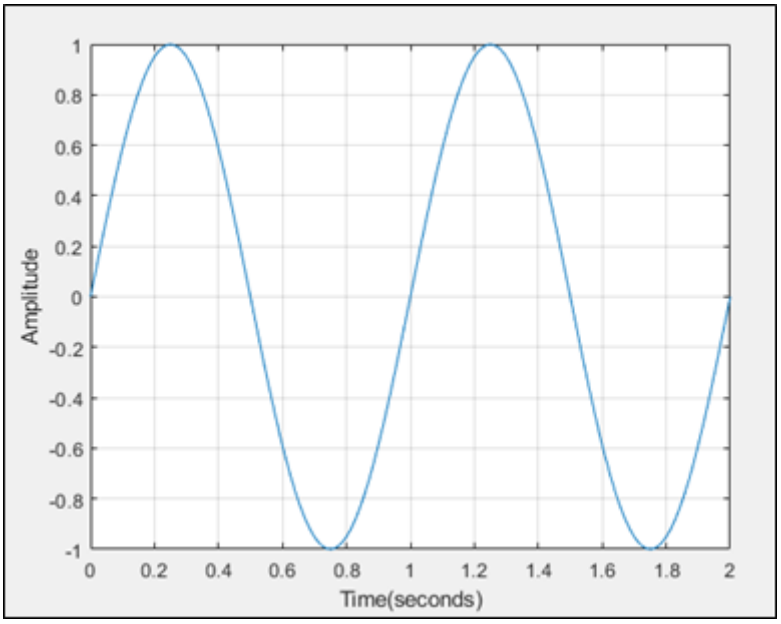


Figure 22 - Control input signal for Sinusoidal input of both systems (Tf_1,Tf_2)

7.7 - Robustness of the Design

By providing disturbance to both SISO system and by observing its root locus and response, thesensitivity to changes and system variation is checked.

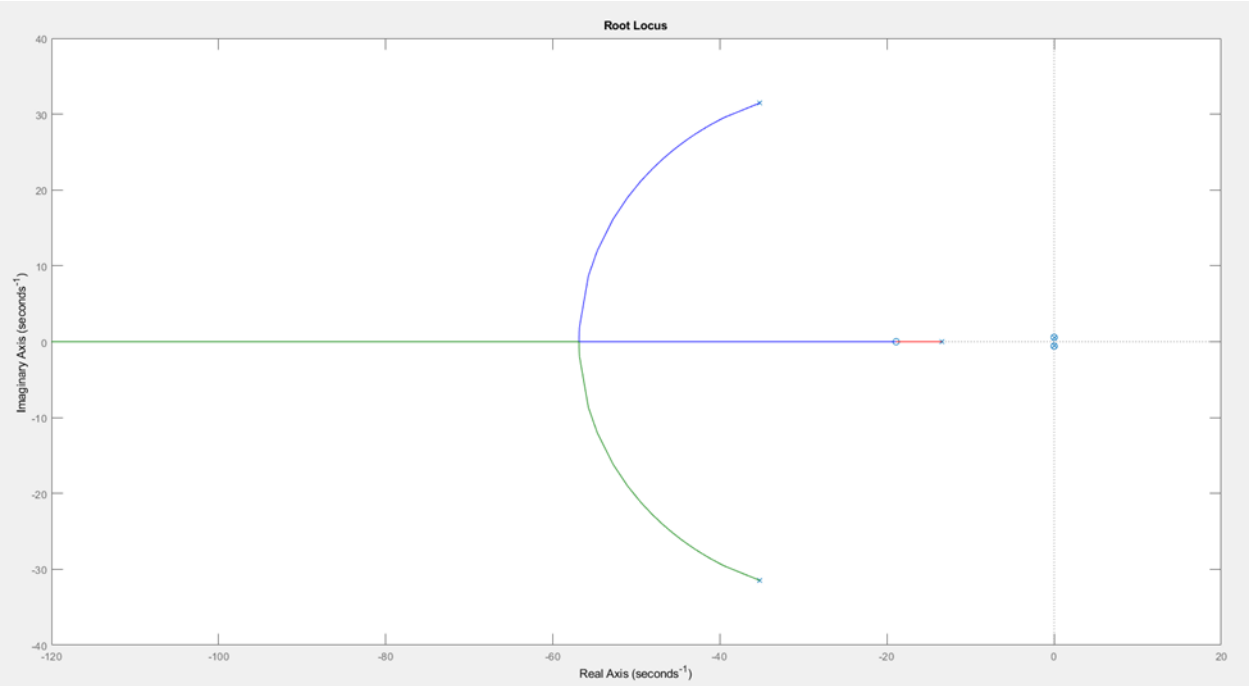


Figure 23 - Root locus of system Tf_1

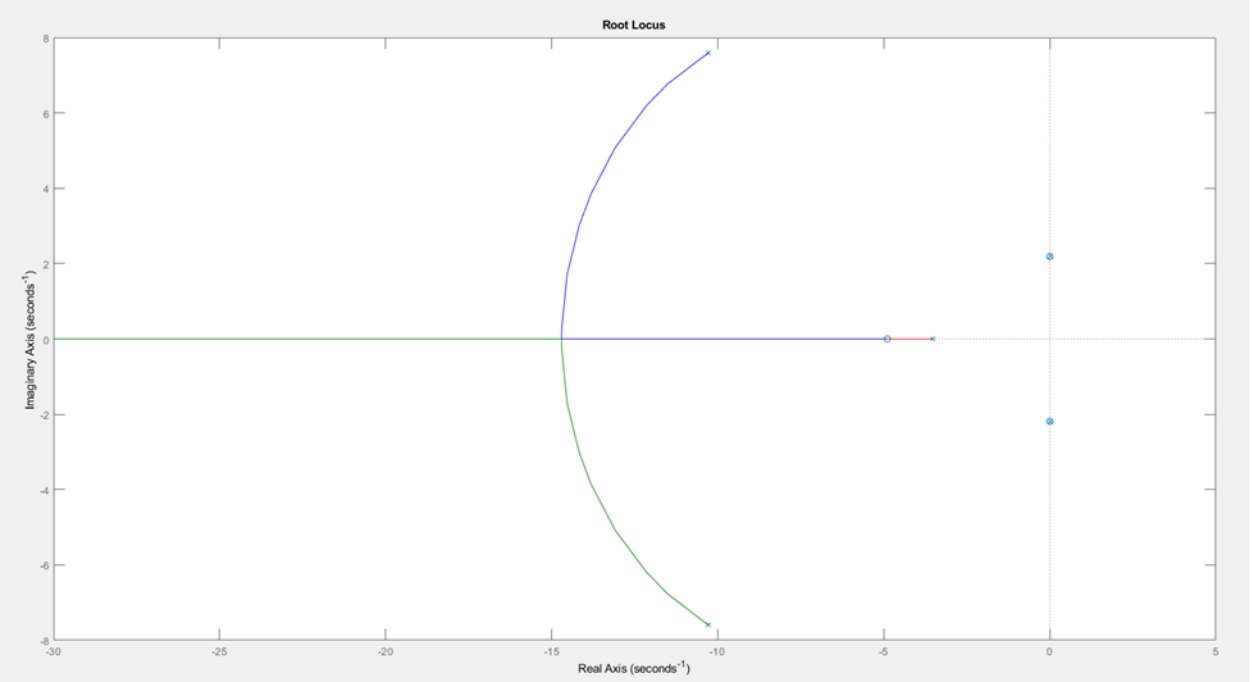


Figure 24 - Root locus of system Tf_2

7.8 - Step Response of system Tf_1 and Tf_2 with Disturbance

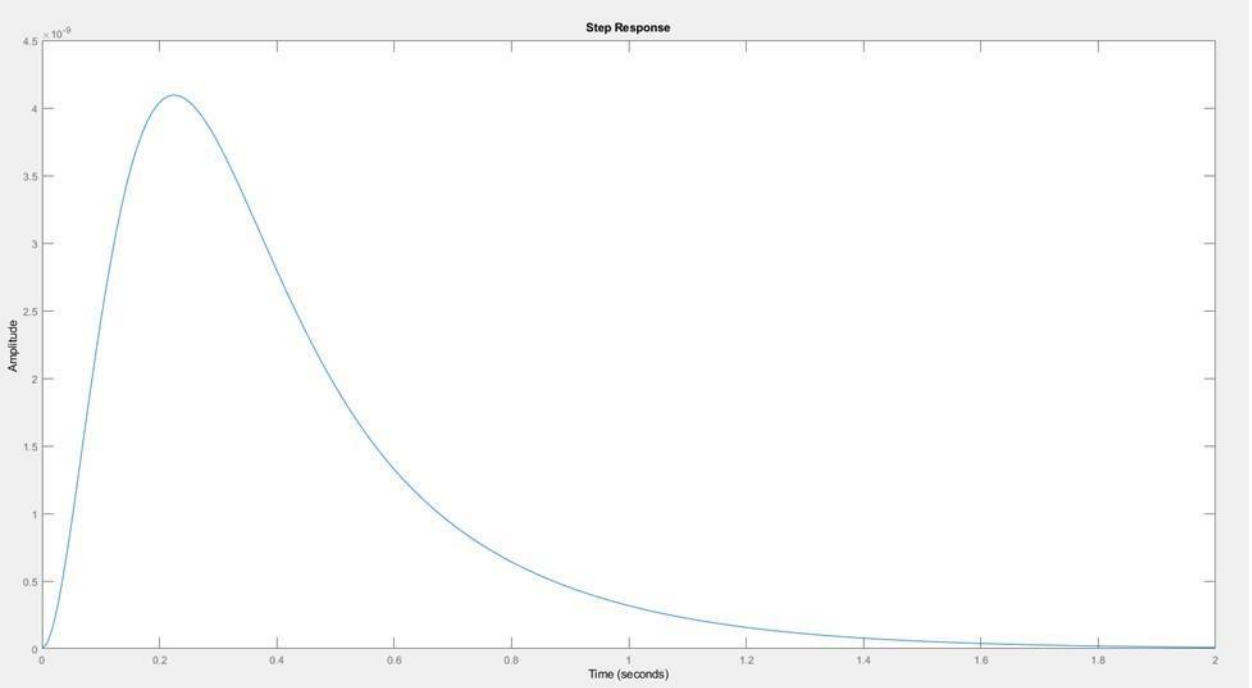


Figure 25 - step response of the system (Tf_1) with disturbance

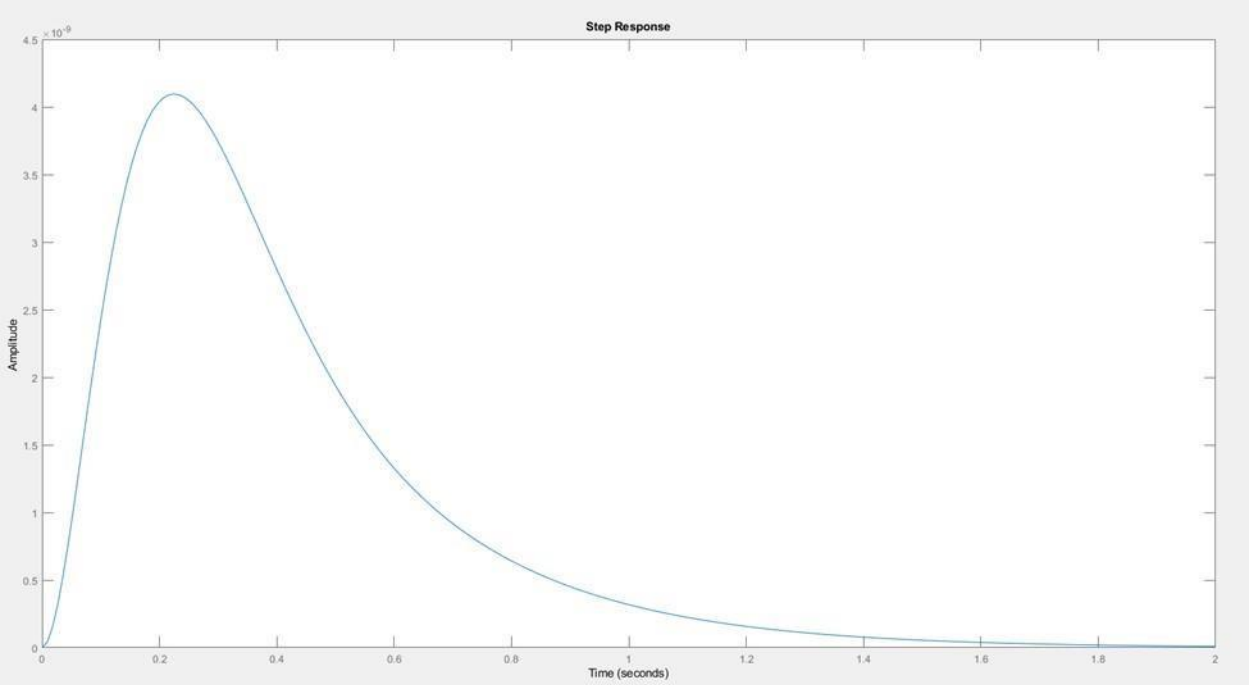


Figure 26 - Step response of system Tf_2 with disturbance

7.9 - Step Response, Square wave and Sinusoidal response of the feedback controller

Step responses of the two SISO system are shown below.

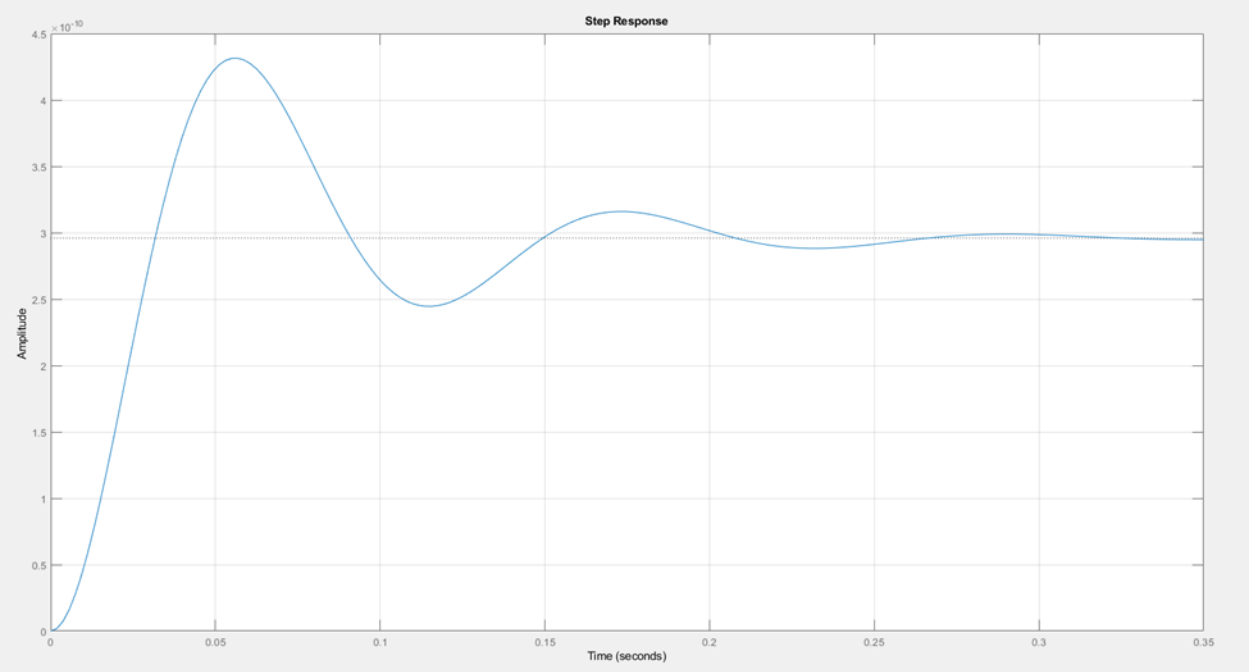


Figure 27 - Step response of SISO system Tf_f1 with state feedback

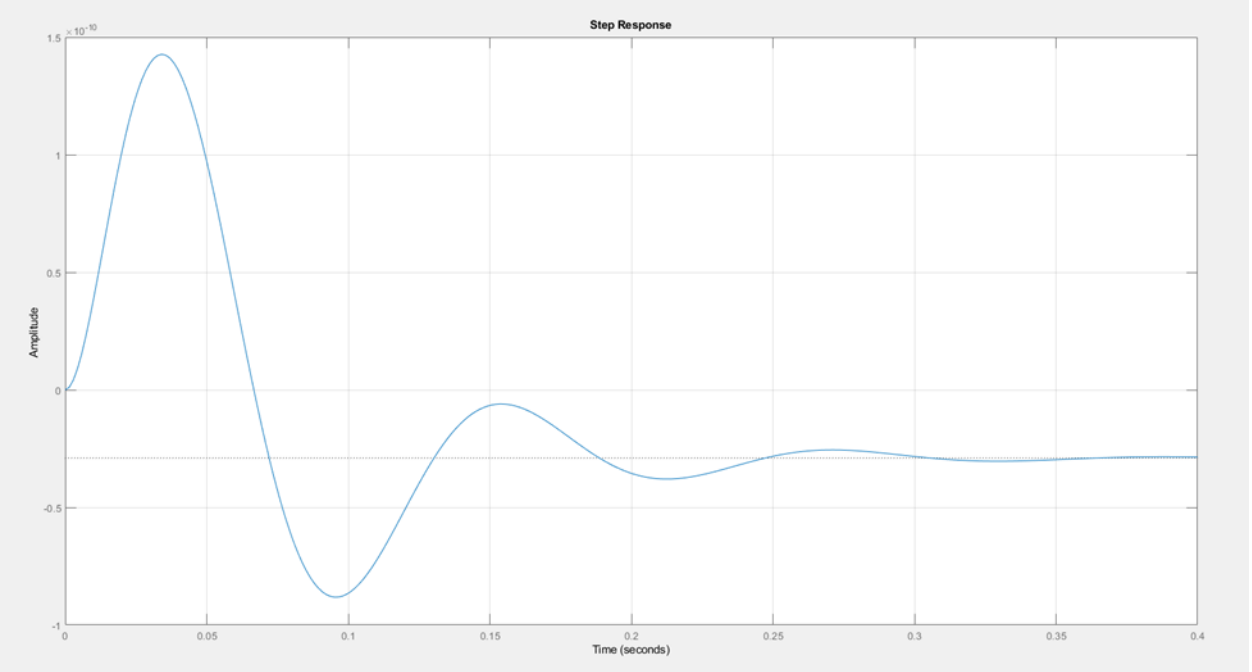


Figure 28 - Step response of SISO system Tf_f2 with state feedback

Square wave responses of two SISO systems Tf_f1 and Tf_f2

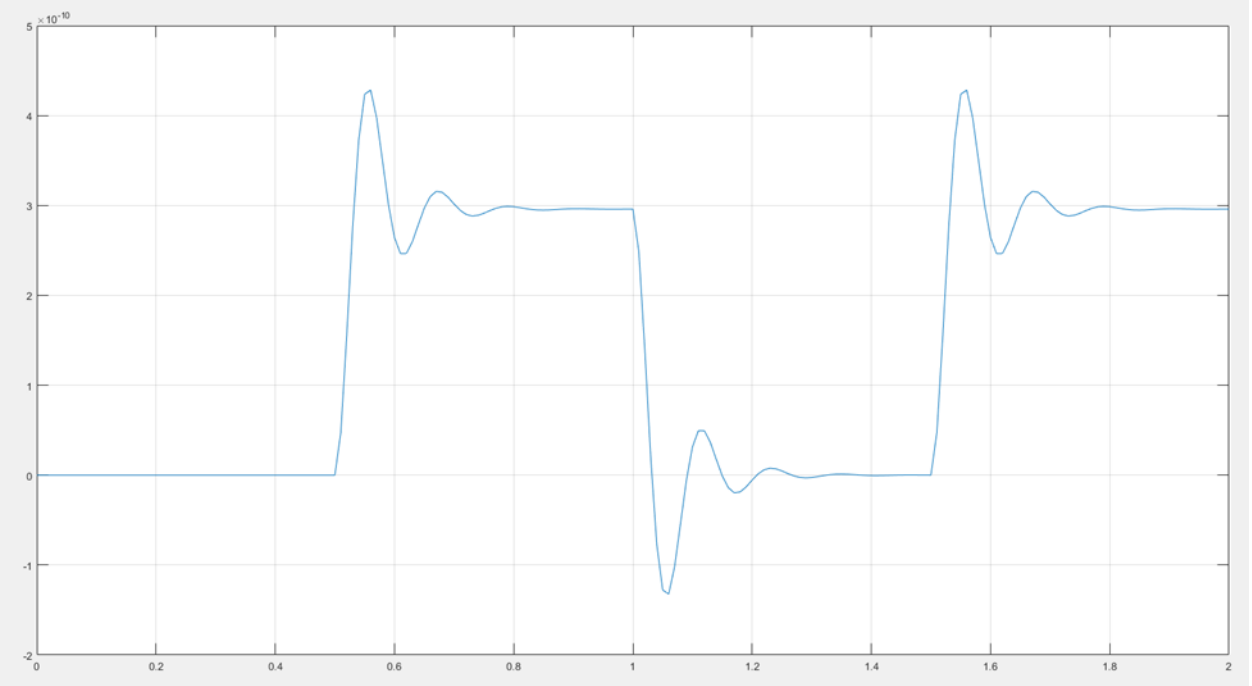


Figure 29 - Square wave responses of SISO system Tf_f1 with state feedback

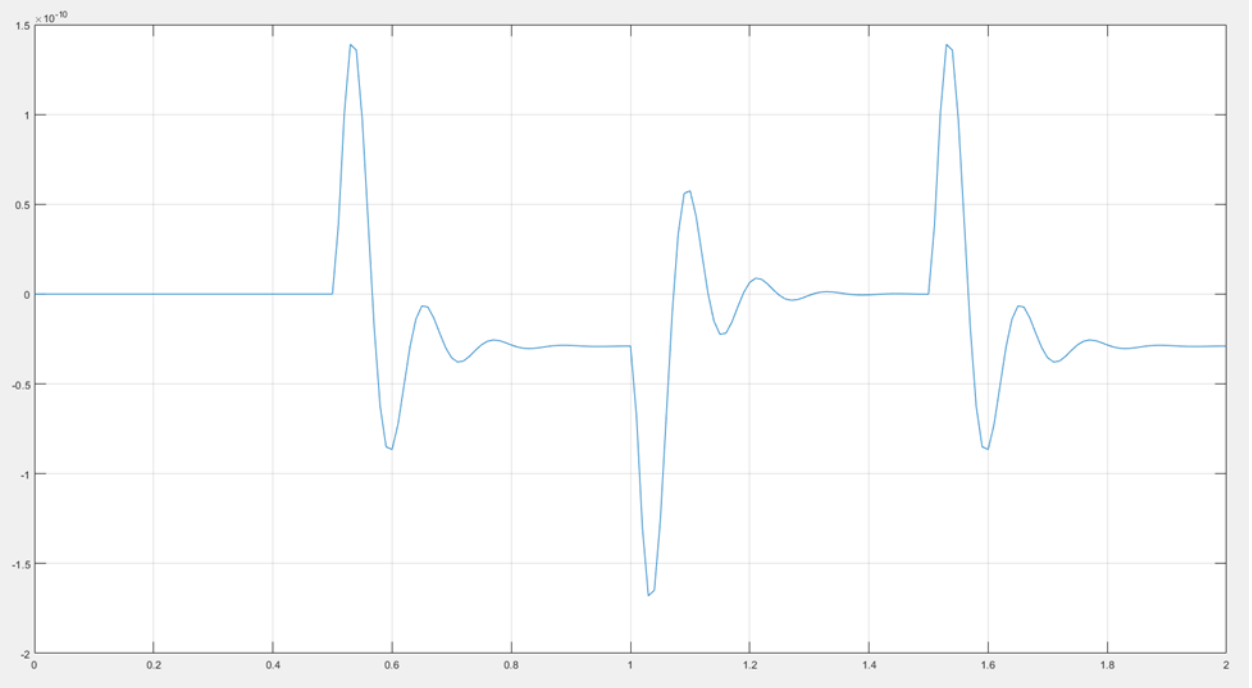


Figure 30 - Square wave responses of SISO system Tf_f2 with state feedback

Sinusoidal responses for the two SISO system Tf_f1 and tf_f2

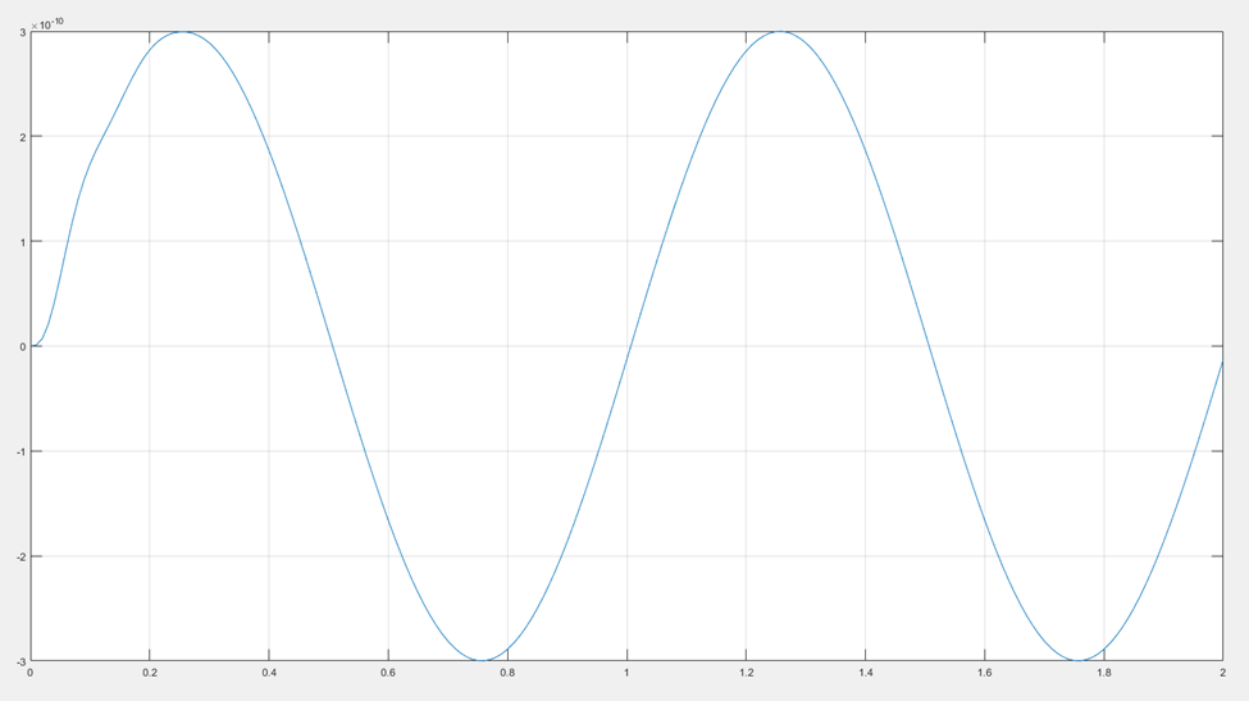


Figure 31 - Sinusoidal response of SISO system Tf_f1 with state feedback

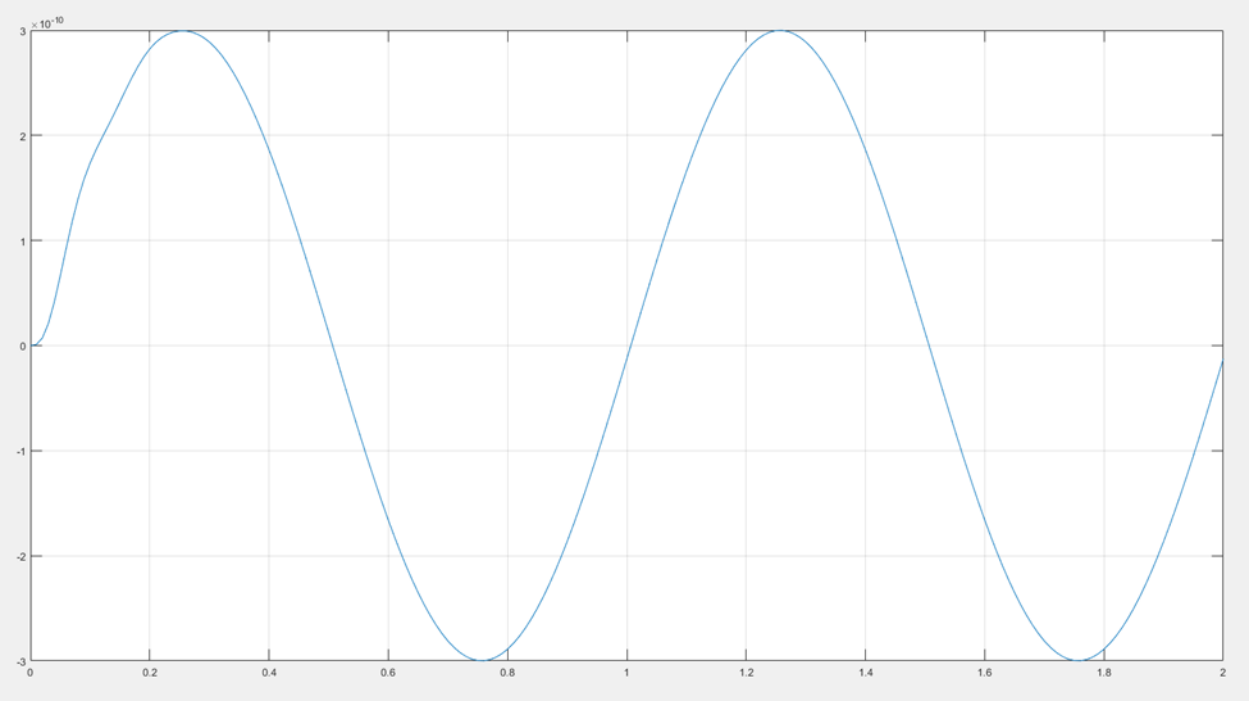


Figure 32 - Sinusoidal response of SISO system Tf_f2 with state feedback

7.10 - Control Input signal for square and sinusoidal responses

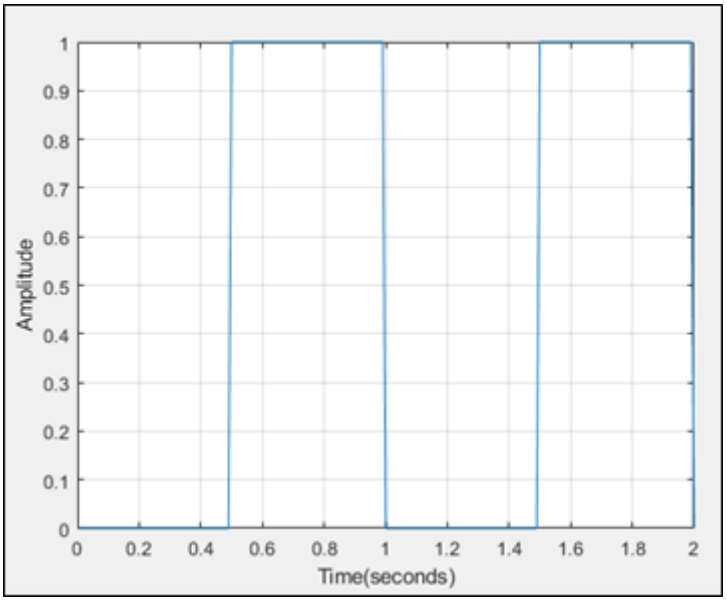


Figure 33 - Control input step signal for the two SISO system with Tf_f1 and Tf_f2

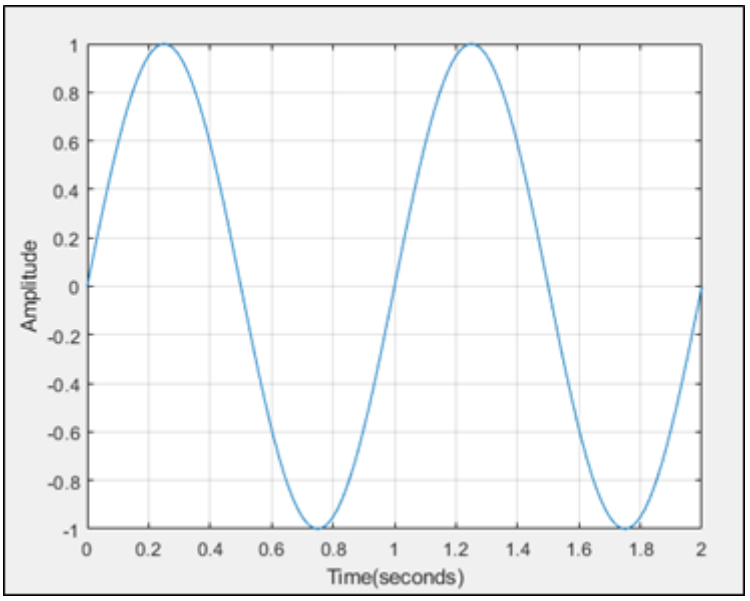


Figure 34 - Control input Sinusoidal signal for the two SISO system with Tf_f1 and Tf_f2

8-DISCUSSION

The fundamental idea of this entire project was to see how the modern and classical techniques for control hypothesis works and attempt to comprehend which technique has the most benefits on account of Magnetic levitation. Also by benefits, I mean, we contrast their responses with various inputs and perceive how well the system acted, how vigorous they were when minor mistakes or changes were made to it, and most critically how simple or troublesome was it to execute them hypothetically and will be hard to execute them practically on a continuous case.

Considering the behavior of both sort of systems, we have PID from classical control methods and then we have state feedback control from the modern world. Both methods have been executed in detailed over the previous sections and have been tested with multiple types of varying inputs, namely step input, sinusoidal input, and square wave input.

The classical control theory is based on the transfer function that is only defined under zero initial conditions and only applicable to linear time-invariant systems. It is generally restricted to single-input single-output systems as this approach becomes highly cumbersome for use in multi- input multi-output systems. It reveals only the system output for a given input and provides no information regarding the internal state of the system.

Modern control theory is essentially a time-domain approach where initial conditions may be included in the design. The organization of the state variable approach is such that it is easily amenable to a solution through digital computers.

Having clarified the fundamental contrasts between the two methodologies, let us examine how they are meant to our system. As referenced, the design of the PID controller was an option exclusively for each SISO system(Y1/U1) and (Y2/U2) where we needed to change the gains to meet the ideal determinations. After experimentation, we had the option to accomplish the responses given in the outcomes segment. Concerning the full state feedback controller and the observers, with a decent comprehension of the numerical model of the system, we had the option to put the poles at their ideal location without the tedious iterations.

In terms of step response, the amplitude of the feedback controller is much smaller, and the settling time is almost3 times faster than the PID. On the other hand, it is a bit more oscillatory than the PID, and that is because we used PID tuner in MATLAB which essentially removed the steady state error within the system. Anyhow, with a maximum amplitude of 4.5*10^-10 and a settling amplitude of 3*10^-10,the feedback controller response is much better than the PID.

Comparing the full order observer and reduced order observer we realize that the peak time increases significantly but settles immediately afterwards. The reduced order observer has an overallbetter response than the full order with lower rising time and amplitudes and faster settling times.
Overall, the full state feedback controller combined with the reduced order observer gives the moststable response in an efficient and rapid way.

Modern methods though are a bit difficult theoretically but have a pretty straight forward way if implemented correctly. They auto tune themselves, and do not require any trial and error. The problem of poles on the right hand side is perfectly dealt here, as one can select the poles as per thedesign and requirements of the plant and give them to the feedback

controller and it will adjust itself accordingly rather than redesigning the whole system again.

9 - CONCLUSION

The motive of the project was to foster thought of the changing modern and classical control hypothesis, executing them on the Magnetic levitation and conclude which one may be more successful. We had the option to design stable closed loop systems with both modern and traditional techniques and the outcomes were comparative and fulfilling. It can be seen that however PID controllers were more straightforward to carry out they required testing and trials in request to track down the ideal point. But, it is difficult to implement them involving complex eigen values. Whereas, Full state feedback can be costly to implement, but it surely gives less headache over time.

APPENDIX

```
clc
clear all
%% Operating conditions given
y10 = 2;
y20 = -2;
yc = 12;

%% Parameters
a = 1.65;
b = 6.2;
c = 2.69;
d = 4.2;
m = 120;
g = 9.81;
N = 4;
y120 = yc + y20 - y10;

%% Variables Calculations
u10 = ( a*(y10 +b)^4 ) * ((c /(y120 + d)^4) + (m*g));
u20 = ( a*(y20 + b)^4 ) * ((-c /(y120 + d)^4) + (m*g));
ju1= 1/(m * a* (y10 + b)^4);
ju2= 1/(m * a* (-y20 + b)^4);
j1= ((4 * u10)/(m * a * (y10 + b)^5));
j2= ((4 * u20)/(m * a * (-y20 + b)^5));
j12= ((4 * c)/(m * (y120 + d)^5));
%% Matrix Formation
A = [0, 1, 0, 0;
-(j1 + j12), 0, j12, 0;
0,0,0,1;
j12, 0, -(j2+j12), 0];
B = [0, 0;
ju1, 0;
0,0;
0, ju2];
C = [1, 0, 0, 0;
0, 0, 1, 0];
D=[0,0;0,0];
% MIMO State-Space Model
sys= ss(A,B,C,D);
 % Transfer Function Matrix
sys_tf = tf(sys)
% Transfer function of open loop system %
H = tf(sys);
Tf_1 = H(1,1);
Tf_2 = H(2,2);
```

```
% controllability %
Cx = ctrb(sys);
rank(Cx);

% observability %
Ox = obsv(sys);
rank(Ox);

% observable canonical form %
osys = canon(sys, 'companion')
% controllable canonical form %
csys = canon(sys, 'companion').'
% Jordan canonical form %
J = jordan(A)

% impulse and step response of open loop system %
figure
impulse(sys);grid
figure
step(sys);grid

% bode plot and root locus %
figure
bode(Tf_1);
figure
rlocus(Tf_1);
figure
bode(Tf_2);
figure
rlocus(Tf_2);

% Designing of PID controller %
 % system with Tf_1
pidTuner(Tf_1,'p');
Kp1=2841145763;
Ki1 = 26872325271;
Kd1 = 75096862;
Ctrl1=tf([Kd1 Kp1 Ki1],[1 0]); CL_1 = Ctrl1* Tf_1;

%transfer function of closed loop system with pid controller %
pid_1= feedback(CL_1,1);

% for system with Tf_2 %
pidTuner(Tf_2,'p');
Kp2 = 210894011;
Ki2 = 515731761;
Kd2 = 21559795;


Ctrl2=tf([Kd2 Kp2 Ki2],[1 0]);
CL_2 = Ctrl2* Tf_2;

% transfer function of closed loop system with pid controller %
pid_2= feedback(CL_2,1) ;

% square and sinusoidal response %
```

```
%square wave%
[u1,t1] = gensig("square",1,2);
figure
lsim(pid_1,u1,t1);
grid on

[u2,t2] = gensig("square",3,6);
figure
lsim(pid_2,u2,t2);
grid on

%sinusoidal response%
[u_1,t_1] = gensig("sin",1,2);
figure
lsim(pid_1,u_1,t_1)
grid on

[u_2,t_2] = gensig("sin",3,6);
figure
lsim(pid_2,u_2,t_2)
grid on

%Robustness%
rlocus(pid_1);
CL_distr1 = 1 + CL_1;
Dist1 = Tf_1 / CL_distr1;
figure
step(Dist1);

rlocus(pid_2);
CL_distr2 = 1 + CL_2;
Dist2 = Tf_2 / CL_distr2;
figure
step(Dist2);

% state feedback control %
 % pole placement %
Poles = [-16.2+53.73i -16.2-53.73i -48.6 -64.8];
K = place(A,B,Poles);
%closed loop matrix%
Af = A-B*K;
sys_fdbk = ss(Af,B,C,D);
Tf_fdbk =tf(sys_fdbk);

Tf_f1 = Tf_fdbk(1,1); % input1,output1
Tf_f2= Tf_fdbk(2,2); % input2, output2

% step response %
figure
step(Tf_f1);grid

figure
step(Tf_f2);grid

% square and sinusoidal response %
 %square wave response%
[Asq,tsq] = gensig('square',1,2,0.01);

[y,t,x]=lsim(Tf_f1,Asq,tsq);
```

```
figure
plot(t,y);grid

[y,t,x]=lsim(Tf_f2,Asq,tsq);
figure
plot(t,y);grid

% sinusoidal response %
[Asn,tsn] = gensig('sin',1,2,0.01);

[y,t,x]=lsim(Tf_f1,Asn,tsn);
figure
plot(t,y);grid

[y,t,x]=lsim(Tf_f2,Asn,tsn);
figure
plot(t,y);grid

% observer controller %
% full order observer %
Pole_of_obs=[-64.7 -64.7 -194.1 -258.8];
G = place(A',C',Pole_of_obs)';
Ae =A-G*C-B*K;
Be =G;
Ce =-K;
De= zeros(2,2);
sys_full_obs = ss(Ae,Be,Ce,De);
tf_full_obs = tf(sys_full_obs);
Tf_o1 = tf_full_obs(1,1);

Tf_o2 = tf_full_obs(2,2);
step(sys_full_obs);
grid on
%sinusoidal response%
Tf_o1 = tf_full_obs(1,1);
[y,t,x]=lsim(Tf_o1,Asn,tsn);
figure
plot(t,y);grid

Tf_o2 = tf_full_obs(2,2);
[y,t,x]=lsim(Tf_o1,Asn,tsn);
figure
plot(t,y);grid
%reduced order observer%
Ge = [ 64.72 0; 0 64.72];
a11 = [0 0;0 0];
A1e = [1 0;0 1];
Ae1 = [-4.78 3.31*10^-7; 3.31*10^-7 -6.16];
Aee = [0 0; 0 0];
b1 =[0 0; 0 0];
Be =[1.117*10^-6 0; 0 1.117*10^-6];
K1 =[K(1,1) K(1,3); K(2,1) K(2,3)];
Ke =[K(1,2) K(1,4); K(2,2) K(2,4)];

%observability%
Or = obsv(Aee,A1e);
rank(Or);

Arce = Aee - (Ge*A1e) - (Be*Ke) + (Ge*b1*Ke);
```

```
Brce = Ae1 - (Ge*a11) + (Aee*Ge) - (Ge*A1e*Ge) - (Be*K1) + (Ge*b1*K1) -
(Be*Ke*Ge) + (Ge*b1*Ke*Ge);
Crce = -Ke;
Drce = -K1 - (Ke*Ge);
red_ord_obs = ss(Arce,Brce,Crce,Drce);
tf_red_ord_obs = tf(red_ord_obs);
Tf_r1 = tf_red_ord_obs(1,1);
Tf_r2 = tf_red_ord_obs(2,2);
step(red_ord_obs);grid
%sinusoidal response%
[y,t,x]=lsim(Tf_r1,Asn,tsn);
figure
plot(t,y);grid

[y,t,x]=lsim(Tf_r2,Asn,tsn);
figure
plot(t,y);grid
```

REFERENCES

1) Brogan, W. L. (1991). Modern control theory. Englewood Cliffs, NJ: Prentice-Hall.
2) https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID
3) https://advancedcontrolsystems.wordpress.com/2009/11/26/classical-vs-modern-control-theory/#:~:text=The%20scope%20of%20classical%20control,output%20(SISO)%20system%20design.&text=In%20contrast%2C%20modern%20control%20theory,%2Doutput%20(MIMO)%20systems
4) Khashayar Khorasani. ENGR 6131 Linear System lecture notes, Fall 2021.