CONTROL ENGINEERING LAB

WINTER SEMESTER 2022

GROUP – 18

Aman Saini (2020EEB1155)

Anirudh Sharma(2020EEB1158)

Ansaf Ahmad (2020EEB1160)

OBJECTIVE

Design state feedback control for a given digital state-space system, so as to realise performance specifications for different applications.

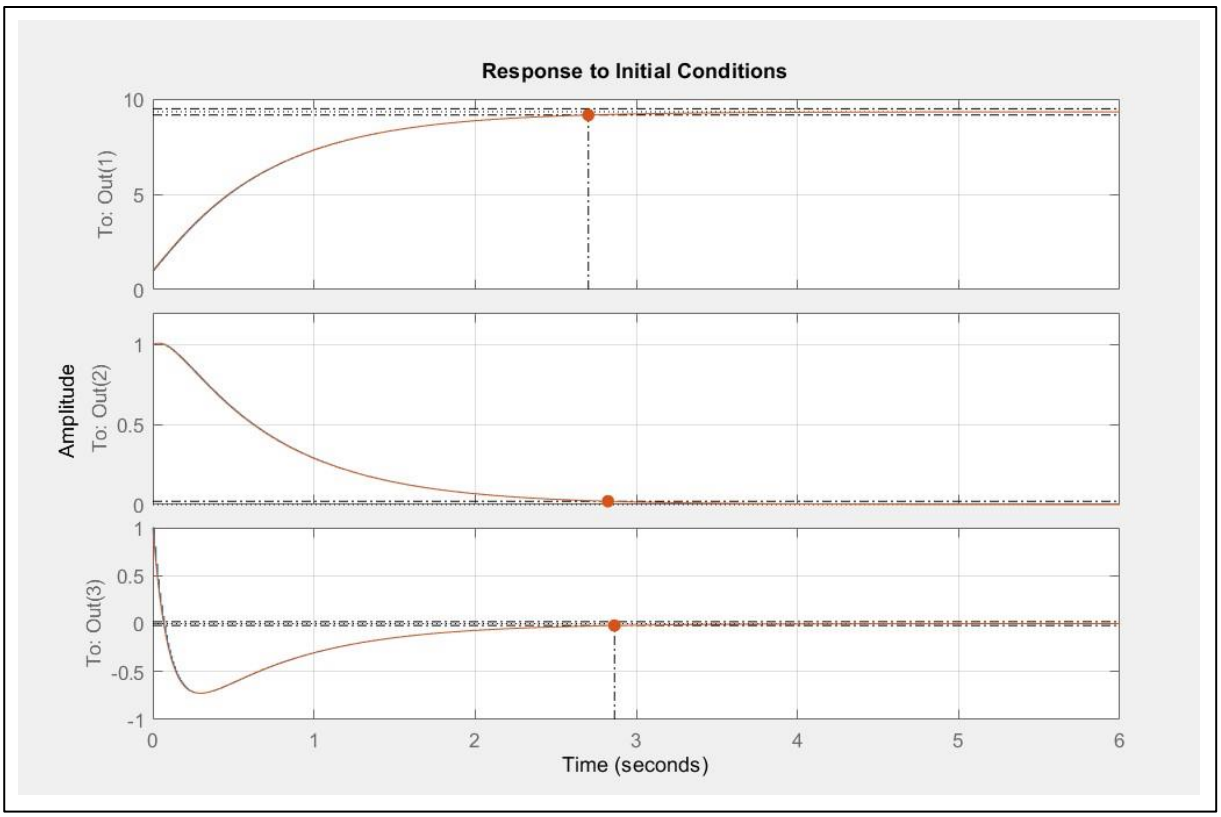
INTRODUCTION

We are given the discretised state space model for an armature-controlled DC motor, which is described by the following matrices.

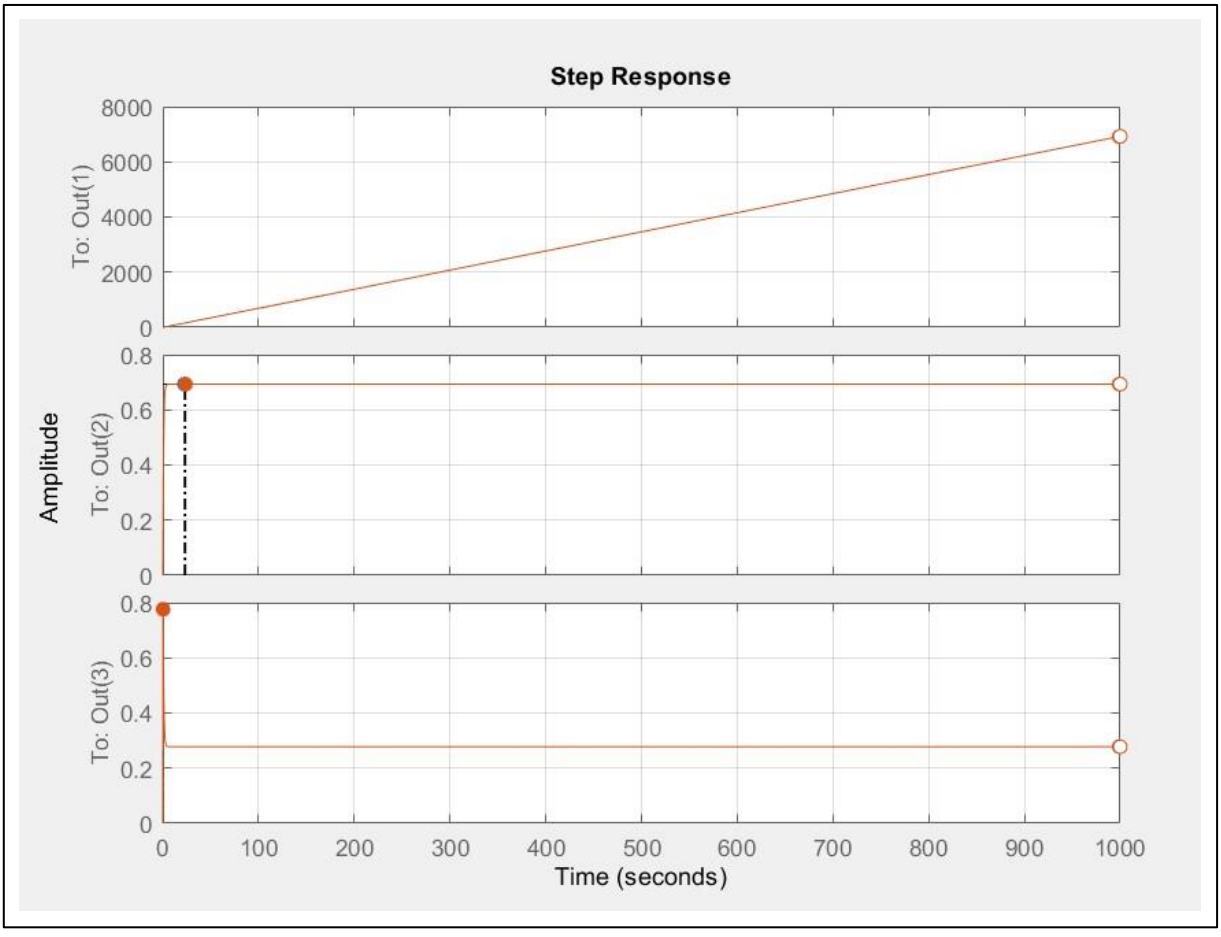
$$F = \begin{bmatrix} 1.0 & 0.1 & 0.0 \\ 0.0 & 0.9995 & 0.0095 \\ 0.0 & -0.0947 & 0.8954 \end{bmatrix} \quad G = \begin{bmatrix} 1.622 \times 10^{-6} \\ 4.821 \times 10^{-4} \\ 9.468 \times 10^{-2} \end{bmatrix}$$

The matrix C and D have not been provided but since this problem requires us to analyse the states, therefore we can take **C = 3x3 Identity matrix** (which declares every state of the system as an output, and does not alter the system in any other way) and **D = 3x1 Null matrix**. The model has been discretised using the sampling time of 0.01s. There are 3 distinct applications for which the state feedback matrices are to be designed, each corresponding to a different set of eigenvalues of the system. The eigen value of the open loop system are: **{1.0, 0.9855, 0.9054}.** With the given set of eigen values, it is clear that the open loop system is marginally stable, owing to the presence of pole at z = 1.0.

We start with the given problem by inspecting how the open loop system responds to the given initial conditions $\mathbf{[1\ 1\ 1]^T}$ with no input.



We see that with time states 2 & 3 decay to zero value and state 1 attains a steady state value of approximately 9.32 units. We further look at the step response of the open loop system and obtain the following data.



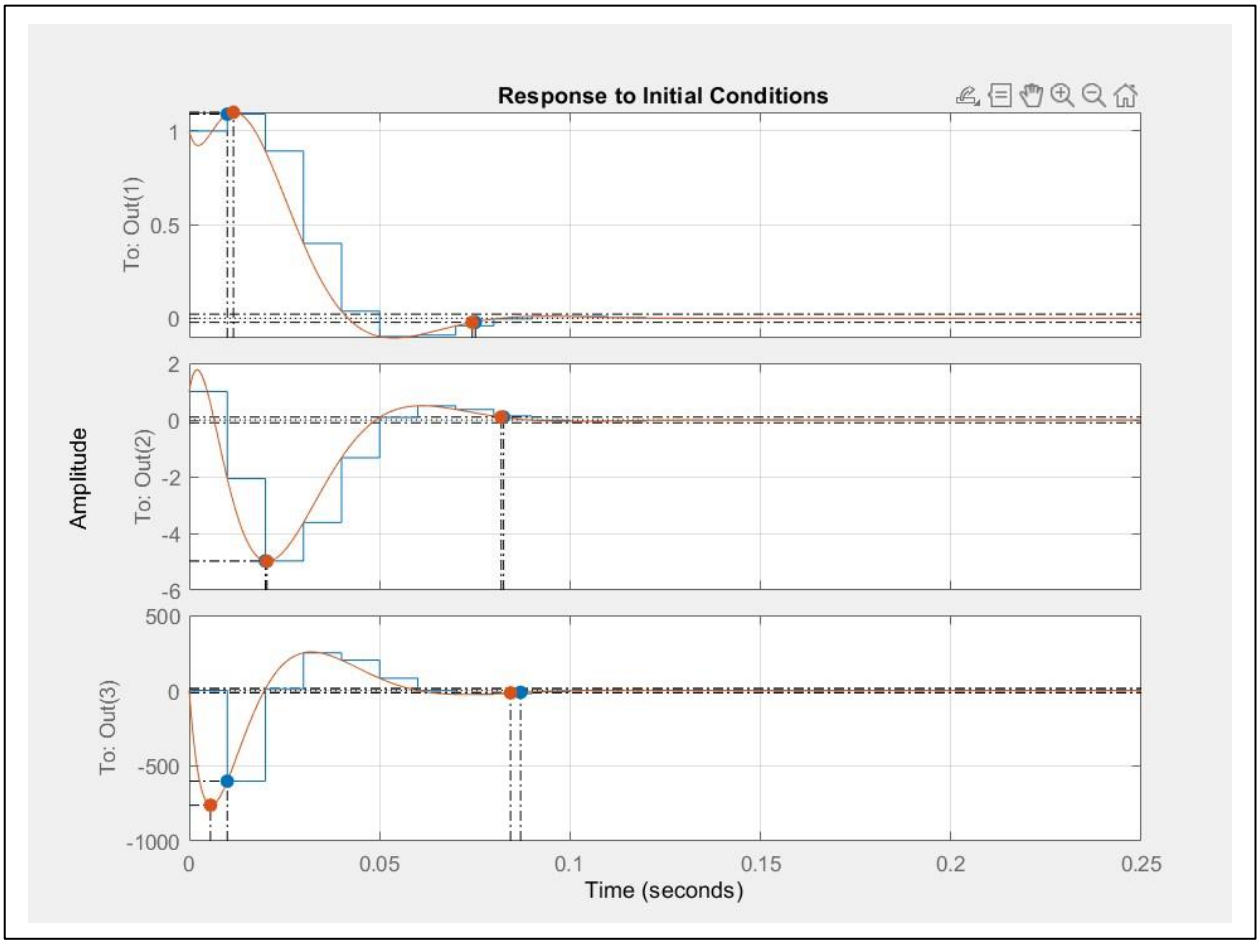
We observe that state 1 increase linearly and indefinitely with time and states 2 and 3 settle to a constant value. Having looked at how the open loop system behaves we can proceed to design the state feedback matrix for 3 applications listed.

APPLICATION NO. 1

This application requires the eigen values to be at: **{0.1, 0.4 + 0.4*j, 0.4 – 0.4*j}.** this can be achieved using the place function in MATLAB, which gives us the state feedback matrix, which we can use to design the closed loop system. The state feedback matrix that we get is as follows: **K = [4926.8 1426.5 13.7],** adding a state feedback matrix changes the structure of F matrix as well and it becomes:

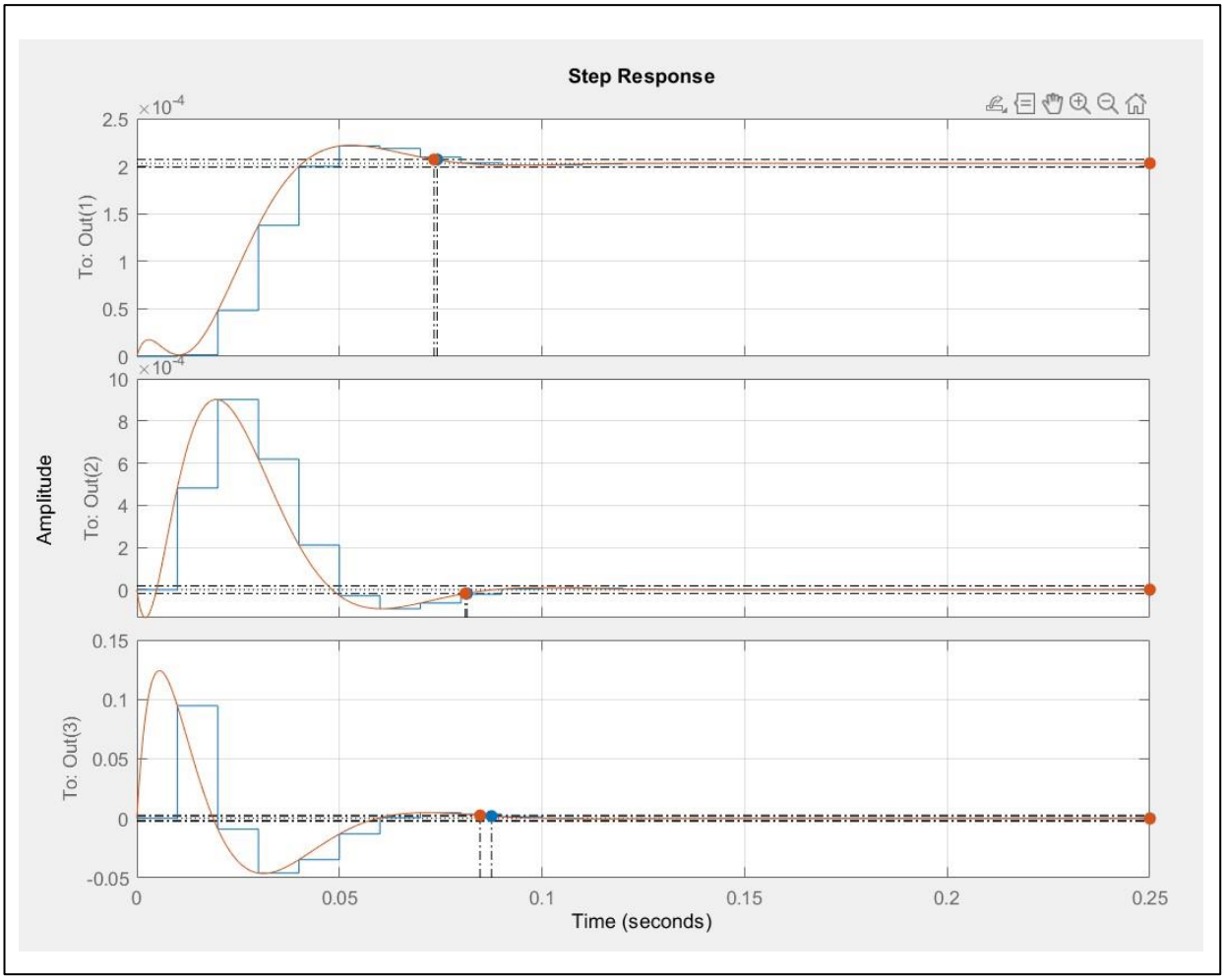
$$F = \begin{bmatrix} 0.9920 & 0.0977 & 0.0000 \\ -2.3752 & 0.3078 & 0.0029 \\ -466.47 & -135.16 & -0.3998 \end{bmatrix}$$

We see how system responds to initial conditions $\mathbf{[1\ 1\ 1]^T}$ with 0 input.



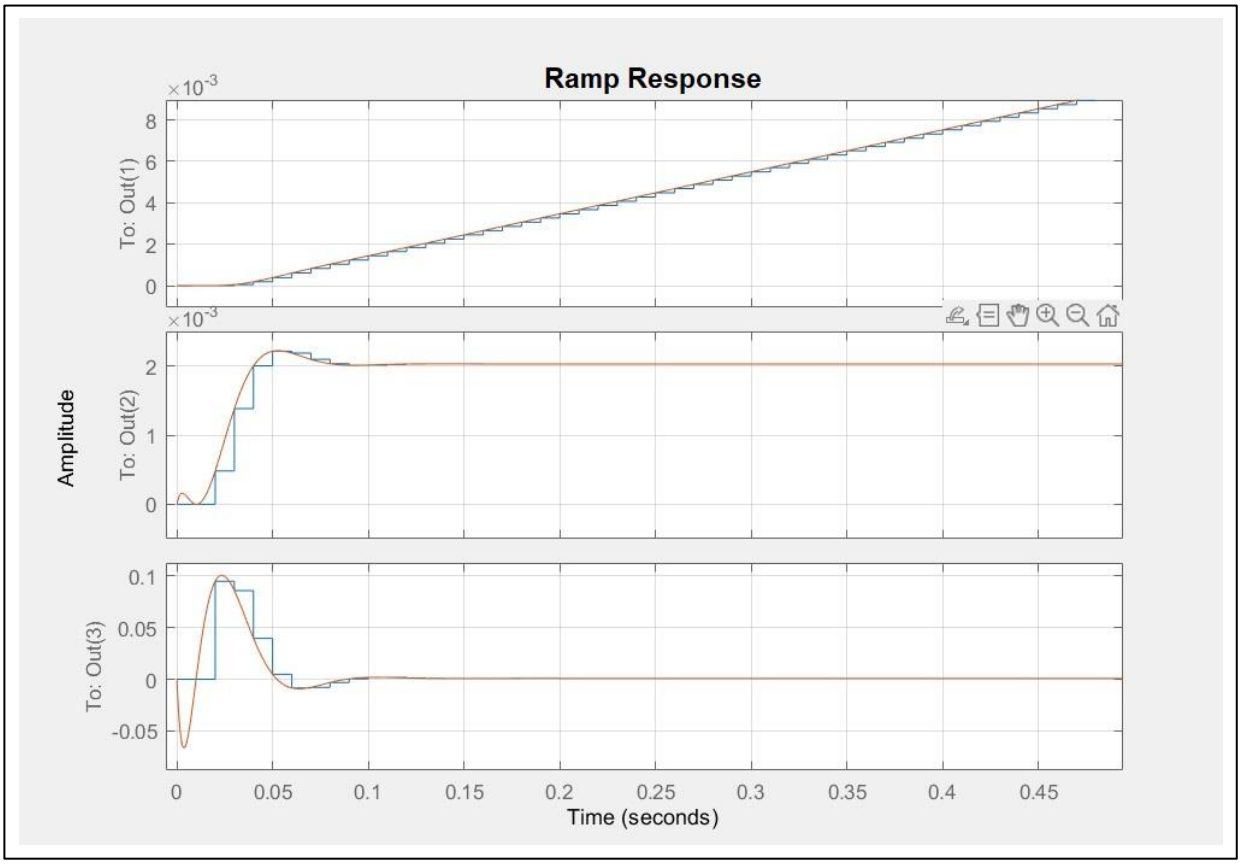
In the above graph we plot the response of the system in continuous time domain and discrete time domain. It is clear from the plots that we lose information about system dynamics in discrete domain with the given sampling time of 0.01 sec, continuous time plots help us to visualise the true nature of system response. We see that each state settles to zero value with

time, furthermore the transient time of the system of the order of 10^{-1} sec (meaning a fast response). We can further plot the step response of the system to understand the about the dynamics and stability of the system.



It is important to note that here discrete time domain plots represents the **Zero-Order Hold** of the continuous time domain plots, this means that both the plots represent the response of a digital system with a sampling time of 0.01s. It's just that we are able to see a complete picture of how the system behaves using continuous time domain plots.

We finally study the ramp response of the system for application 1 with slope = 1unit/sec.



ramp response of the system for Application 1 (discrete time and continuous time plots)

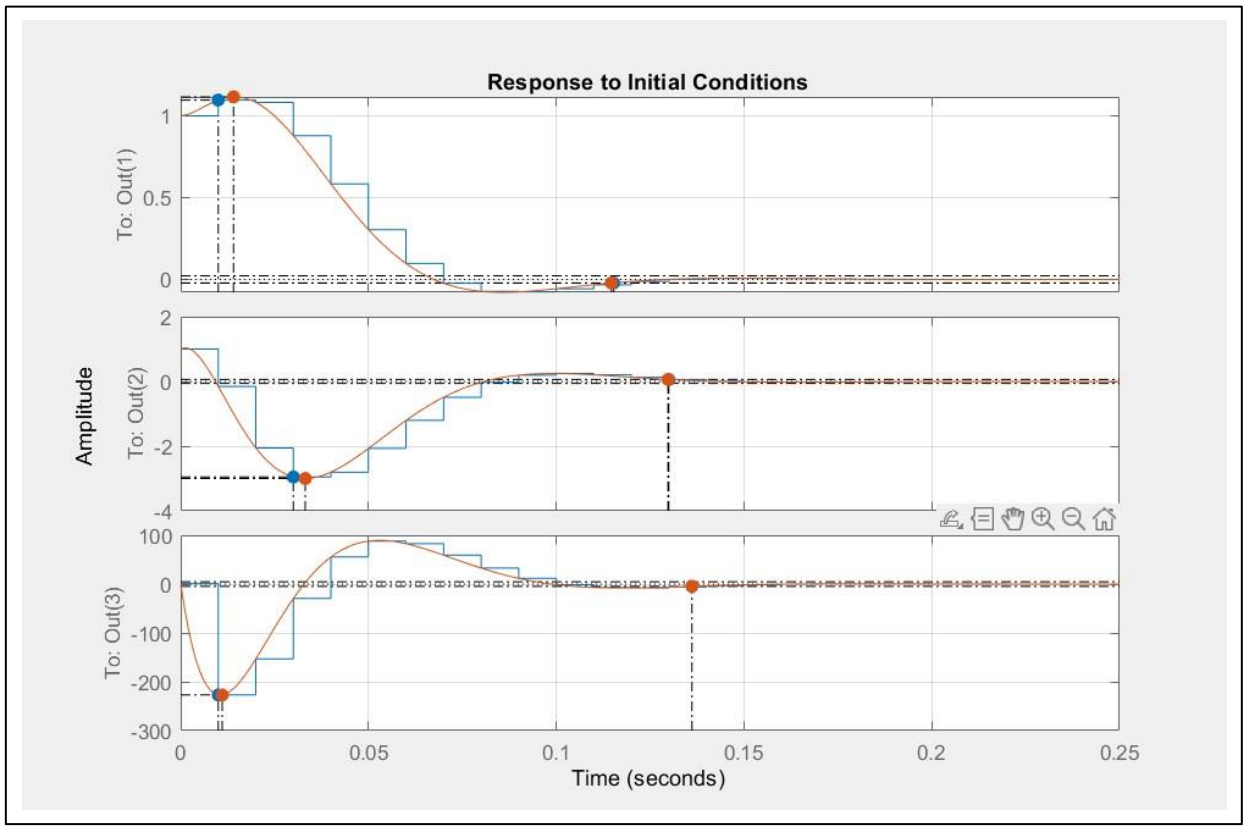
Both state 2 and 3 settle to a finite value, we observe a very high peak overshoot in case of state 3, while it is comparatively less for state 2. Apart from that we see that state 1 increases linearly and indefinitely.

APPLICATION NO. 2

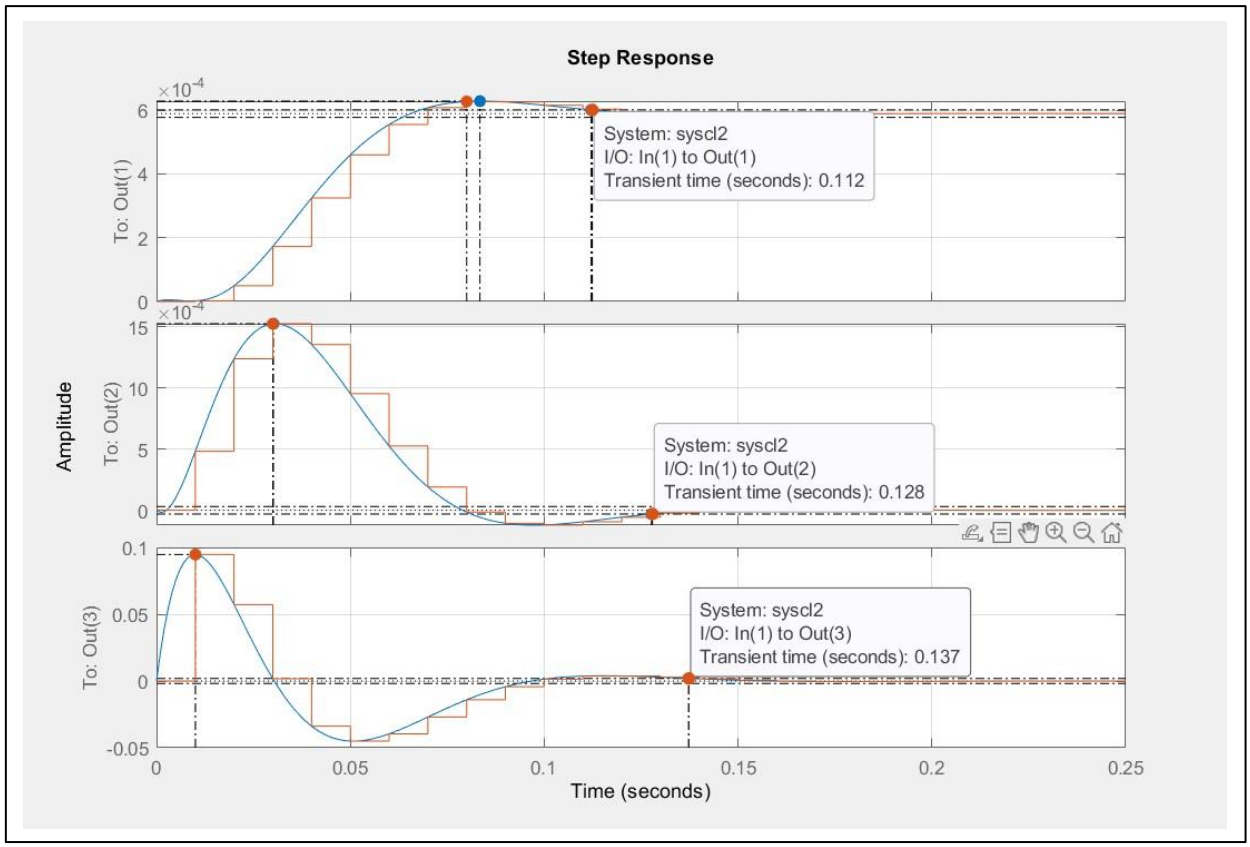
This application requires eigen values to be placed at **[0.4, 0.6 + j0.33, 0.6 – j0.33].** The feedback gain matrix that we get using place function is given as: **$K = [1698.5\ 696.4\ 10.1]^T$.** Adding the given state feedback to the system changes F to:

$$F = \begin{bmatrix} 0.9972 & 0.0989 & 0.0000 \\ -0.8188 & 0.6598 & 0.0047 \\ -160.82 & -66.0316 & -0.0570 \end{bmatrix}$$

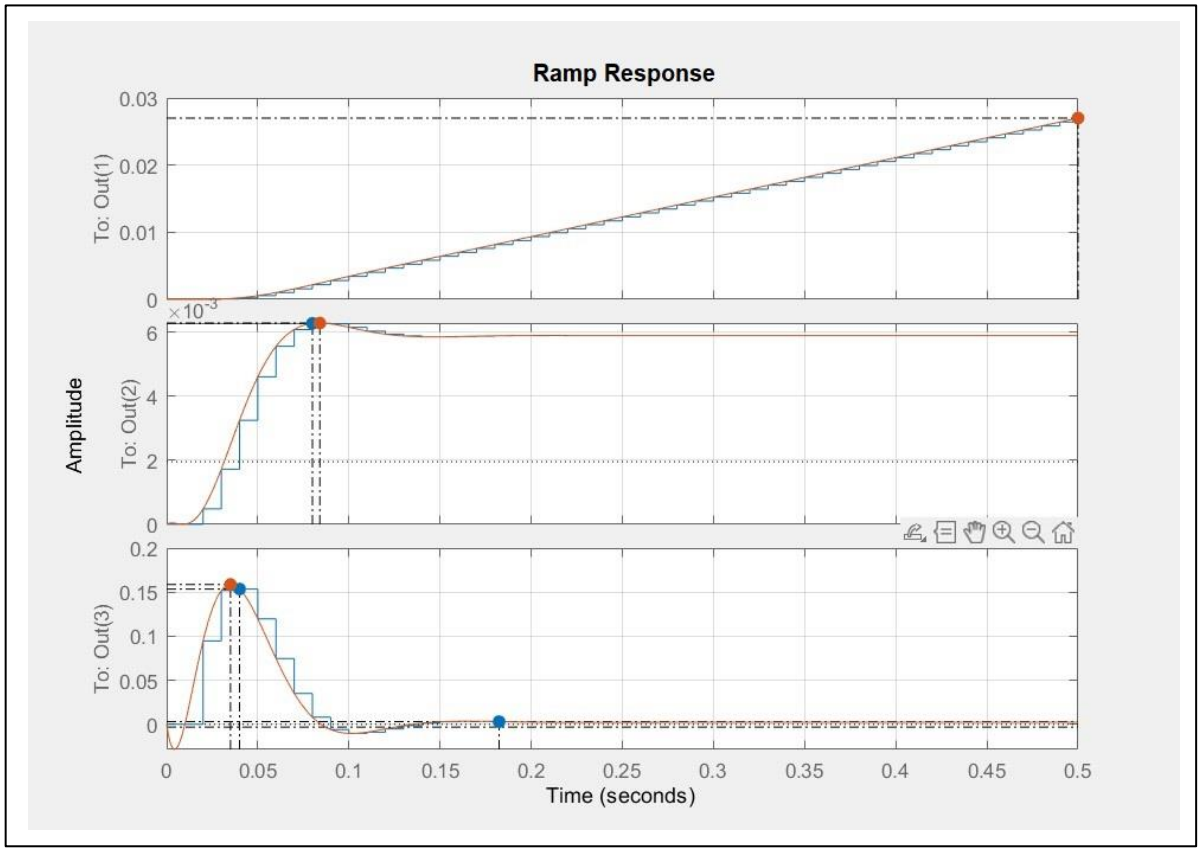
Following plots shows the response of the system configured for application 2 to initial conditions:



We observe that each state settles to zero value for zero input and given initial conditions. We also observe that the transient time for the given application is more as compared to application 1, apparent from the difference b/w magnitude of eigen-values corresponding to the 2 applications. The following plot represents the step response of the system.



Finally, we plot the ramp response of the system for application 2.



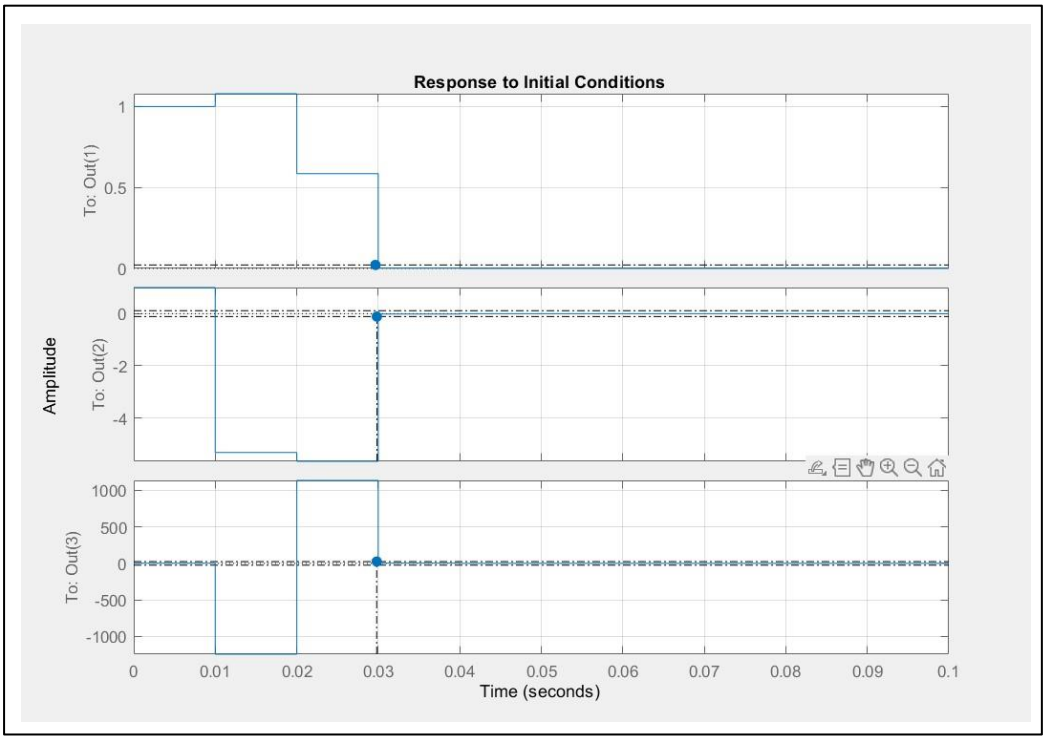
APPLICATION NO. 3

This application requires all 3 poles to be placed at z = 0, which corresponds to -∞ in the s-plane. Here multiplicity of the eigen values is 3, so the conventional pole placement approach does not work, we therefore use acker function in MATLAB for this application, however, this too at best provides an approximation, i.e., eigen values approaches 0 but never really reach it. On trying to place the poles at the desired location, inherently leads to error, and incorporation of negative real z axis which leads to errors while plotting responses, so we use approximation in pole placement itself. The feedback gain matrix obtained is given as: **K = 1e4* [1.0496 0.2609 0.0017].** And the matrix F changes to the following state:

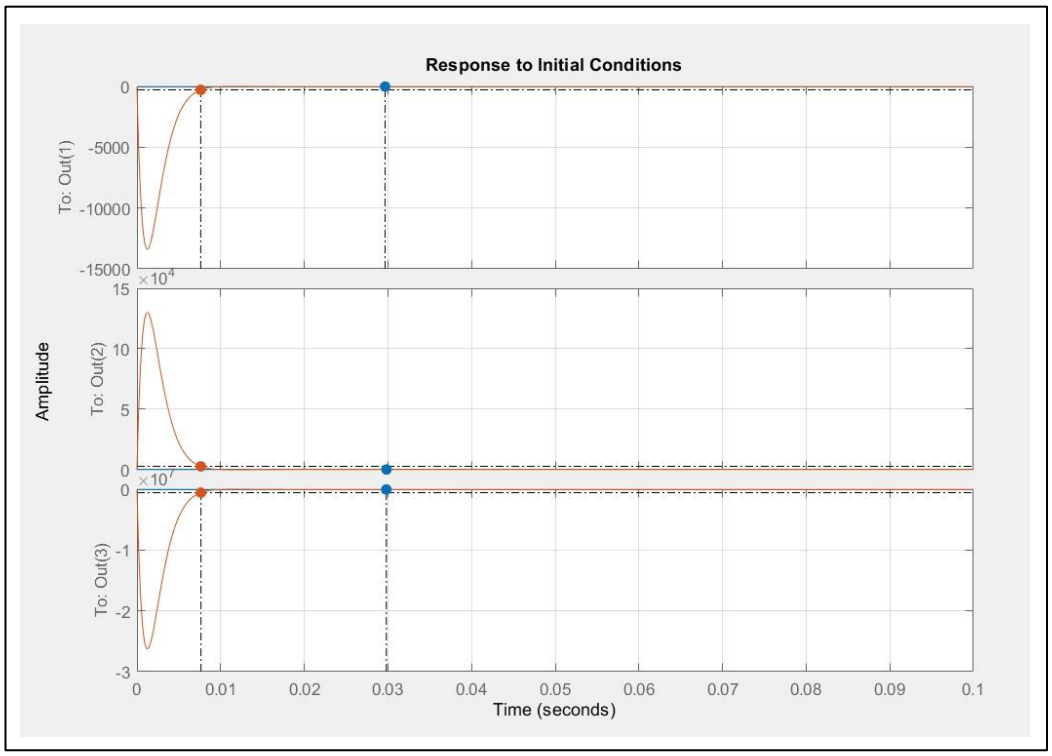
$$F = \begin{bmatrix} 0.9830 & 0.0958 & 0.0000 \\ -5.0600 & -0.2623 & 0.0013 \\ -993.7389 & -247.1185 & -0.7177 \end{bmatrix}$$

For this case we approximate all 3 eigen values to 10^{-3}, which is fairly close to zero and doesn't lead to errors in software.

The response of the system to the initial conditions is:

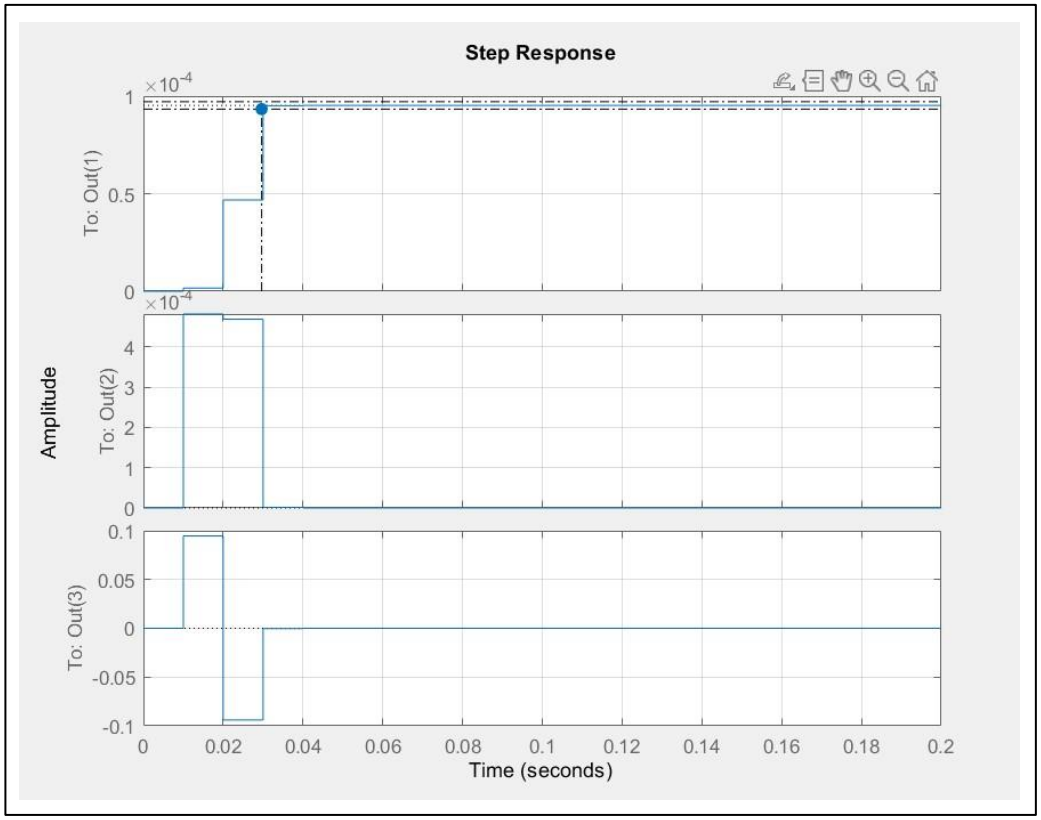


First thing we notice is that the settling time of the system in this case is very less which is understandable from the way eigen values are placed, secondly when we plot the continuous plot of the states along with the discrete plot, we obtain the following graph.

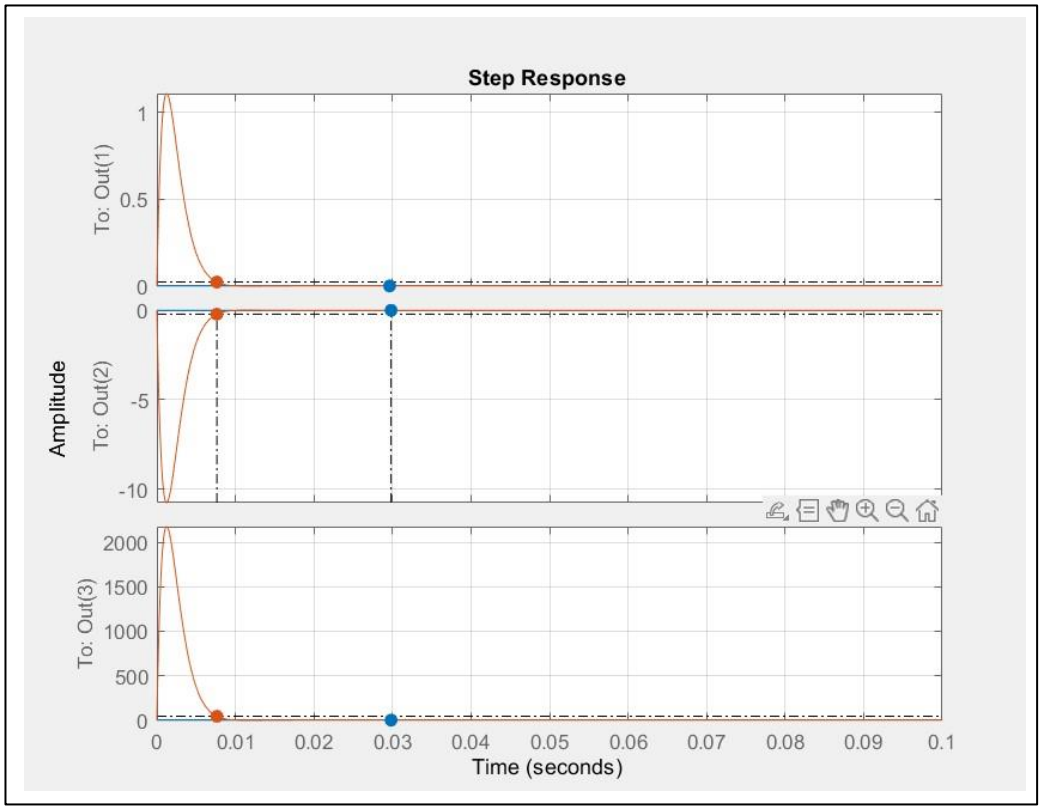


The output sampled at 0.01 sec does not provide the complete picture, the states reach extreme values before the next sample is taken, hence our digital controller never gets the information about how system behaved in the time duration of 0 – 0.01 sec.

The step response of the system is given by following plots.

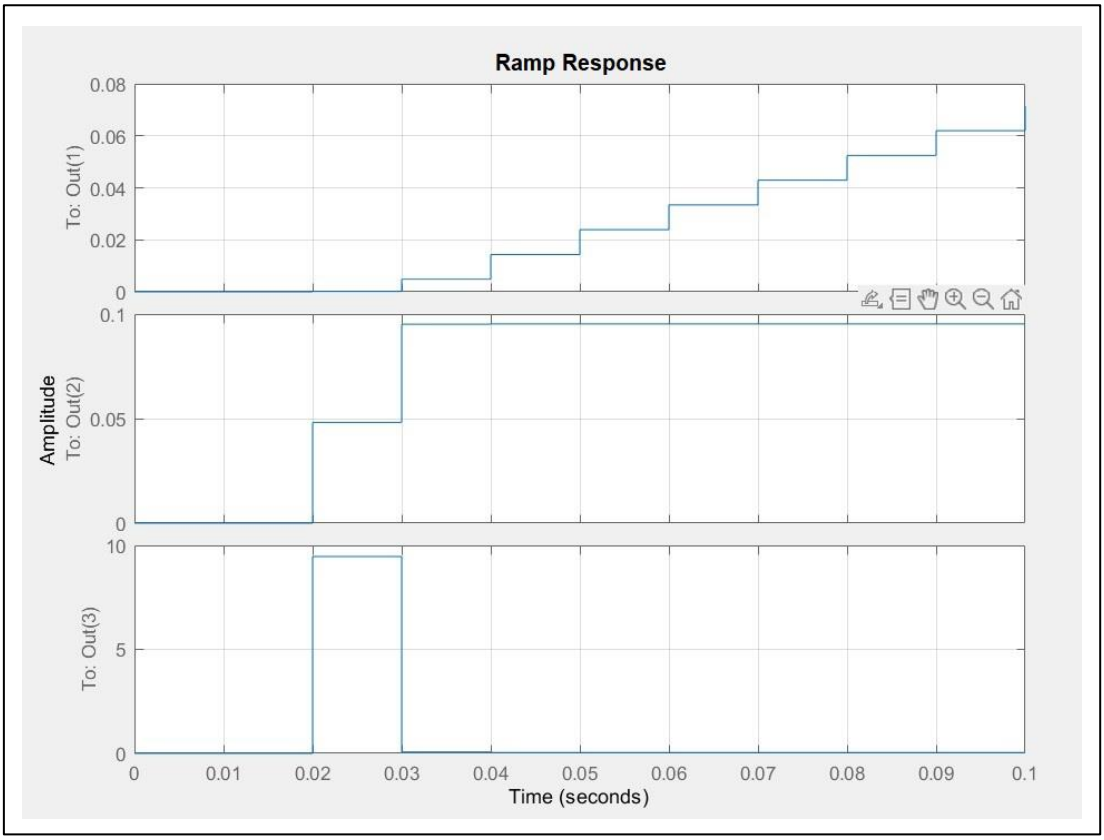


Again, to see how states of the system behave between 0 – 0.01 sec we plot the continuous time response on top of discrete response and obtain following plots:

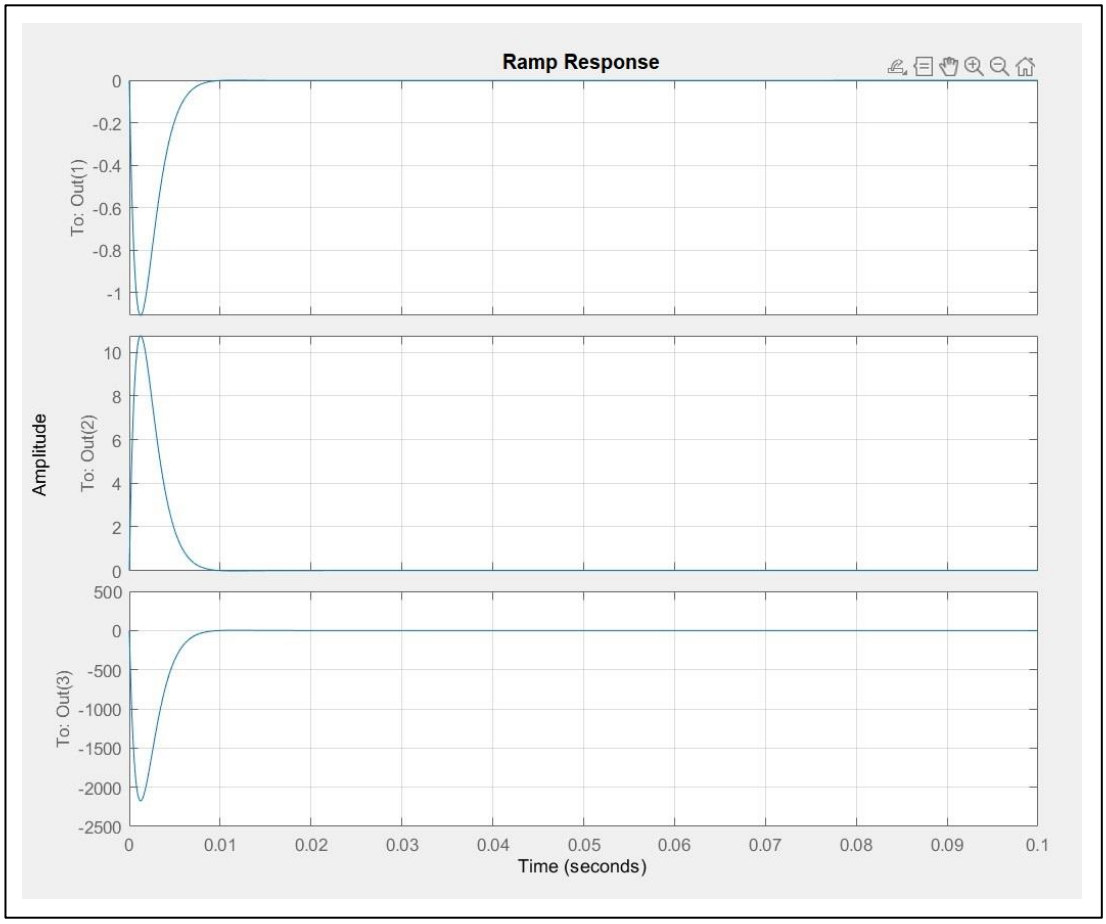


Similar to initial response, the states reach extreme values b/w **t = 0 – 0.01 sec**, which is not recorded by the controller at all.

Finally, we plot the ramp response of the system for application no. 3.



The above plots represent the states sampled by the controller; therefore, we use continuous time domain plots to study the behaviour of states more accurately.



OBSERVATIONS

The difference in the system dynamics (for different applications) directly stems from the location of eigen values of the system. For the application 1, the magnitude of eigen values is: **[0.1, 0.566, 0.566]**, the magnitude of eigen values for application 2 is: **[0.4, 0.685, 0.685].** Since the magnitude in z domain, directly corresponds to the real part in s domain, which tells us how quickly the system settles to steady state value, so it is clear that states settle faster in application 1 than application 2. Also, we see that the eigen value of the system for application 3 is **[0, 0, 0],** this means that in s domain, the poles of system approaches infinity. At best the plots for this application are an approximation, but it is clear that in this application the states settle in a comparatively less time with a very high peak overshoot.

Step-info of system for application 1 is:

State 1:

```
      RiseTime: 0.0202
  TransientTime: 0.0733
   SettlingTime: 0.0733
    SettlingMin: 1.8462e-04
    SettlingMax: 2.2200e-04
      Overshoot: 9.3726
     Undershoot: 0
           Peak: 2.2200e-04
       PeakTime: 0.0528
```

State 2:

```
      RiseTime: 3.0358e-17
  TransientTime: 0.0811
   SettlingTime: NaN
    SettlingMin: -9.0217e-05
    SettlingMax: 9.0231e-04
      Overshoot: 2.9220e+16
     Undershoot: 4.2887e+15
           Peak: 9.0231e-04
       PeakTime: 0.0196
```

State 3:

```
      RiseTime: 5.5511e-17
  TransientTime: 0.0847
   SettlingTime: NaN
    SettlingMin: -0.0464
    SettlingMax: 0.0047
      Overshoot: 8.4110e+15
     Undershoot: 2.2525e+16
           Peak: 0.1242
       PeakTime: 0.0056
```

The settling Time of states 2 and 3 comes out to be not defined but that is because states 2 and 3 settles at zero (similar reason can be given for peak overshoot), hence standard formulas does not give a correct result however it is clearly visible from plots that system settles at a constant value for all the states.

Step-info for application 2 is as follows:

State 1:

```
struct with fields:

      RiseTime: 0.0359
  TransientTime: 0.1121
   SettlingTime: 0.1121
    SettlingMin: 5.3350e-04
    SettlingMax: 6.2794e-04
      Overshoot: 6.6544
     Undershoot: 0
           Peak: 6.2794e-04
       PeakTime: 0.0834
```

State 2:

```
struct with fields:

      RiseTime: 1.8648e-16
  TransientTime: 0.1276
   SettlingTime: NaN
    SettlingMin: -1.2187e-04
    SettlingMax: 0.0015
      Overshoot: 2.6982e+16
     Undershoot: 2.1555e+15
           Peak: 0.0015
       PeakTime: 0.0302
```

State 3:

```
struct with fields:

      RiseTime: 6.9389e-18
  TransientTime: 0.1372
   SettlingTime: NaN
    SettlingMin: -0.0450
    SettlingMax: 0.0947
      Overshoot: 5.0933e+16
     Undershoot: 2.4229e+16
           Peak: 0.0947
       PeakTime: 0.0101
```

The reason why the **settling time** is undefined and **Overshoot & Undershoot** is very high (for states 2 and 3) is same as application 1 and the transient time of the system is more as compared to application 1 which can be explained from magnitude of discrete domain eigen values.

The Step-info for the application 3 is as follows:

State 1:

```
struct with fields:

       RiseTime: 0.0044
  TransientTime: 0.0297
   SettlingTime: 0.0297
    SettlingMin: 4.5789e-05
    SettlingMax: 1.7300e-04
      Overshoot: 81.5773
     Undershoot: 0
           Peak: 1.7300e-04
       PeakTime: 0.0100
```

State 2:

```
struct with fields:

       RiseTime: 2.3074e-10
  TransientTime: 0.0295
   SettlingTime: NaN
    SettlingMin: -0.0012
    SettlingMax: 4.7873e-04
      Overshoot: 3.4671e+09
     Undershoot: 1.4014e+09
           Peak: 0.0012
       PeakTime: 0.0100
```

State 3:

```
struct with fields:

       RiseTime: 1.2784e-10
  TransientTime: 0.0291
   SettlingTime: NaN
    SettlingMin: -0.0959
    SettlingMax: 0.4324
      Overshoot: 6.2579e+09
     Undershoot: 1.3880e+09
           Peak: 0.4324
       PeakTime: 0.0100
```

The above information represents just an approximation of placement of the poles. However, as eigenvalue reach z = 0 for all the states, the step response of state 1 reaches a step signal, and state 3 and state 4 reaches impulse signals. The transient time is much less that above 2 applications, which is fairly obvious.

CONCLUSIONS

We designed the state feedback matrices for all the three applications, and studied the dynamic behaviour of the system, on application of different type of inputs. Since what different states represents in the system is unknown to us, we cannot really comment upon the physical significance of different type of outputs that we get for the states, however in case of application 3, for step input, we get the responses approaching impulse and step functions, thus in case any of the outputs represents a mechanical quantity (or their linear combination) like angular speed, angle of the motor, etc., in case of application 3 we get a strong jerk on application of step input, initial conditions or ramp function as input.

MATLAB SCRIPT USED:

```
z = tf('z', 0.01);
F = [1.0 0.1 0.0; 0.0 0.9955 0.0095; 0.0 -0.0947 0.8954];
G = [1.622e-6; 4.821e-4; 9.468e-2];


C = [1 0 0; 0 1 0; 0 0 1];
D = [0; 0; 0];
sys = ss(F, G, C, D, 0.01);
eig(sys)
isstable(sys)
Tf = tf(sys);
sys_cont = d2c(sys);
figure; initial(sys, [1;1;1]);
hold on; initial(sys_cont, [1;1;1]);


% pole placement method
%% Application number 1
neweig = [0.1; 0.4 + 0.4j; 0.4 - 0.4j];
K = place(F, G, neweig);
syscl = ss(F - G*K, G, C, D, 0.01);
eig(syscl)
Tf2 = tf(syscl);
isstable(syscl)
figure; step(Tf2);
Tfcl = d2c(Tf2);
hold on; step(Tfcl);
hold off;


%% Application Number 2
neweig = [0.4; 0.6 + 0.33j; 0.6 - 0.33j];
K1 = place(F, G, neweig);
syscl2 = ss(F - G*K1, G, C, D, 0.01);
eig(syscl2)
Tf3 = tf(syscl2);
isstable(syscl2)
figure; step(Tf3);
Tfcl1 = d2c(Tf3);
hold on; step(Tfcl1);
hold off;


%% Application Number 3
neweig = [0; 0; 0];
K2 = acker(F, G, neweig);
syscl3 = ss(F - G*K2, G, C, D, 0.01);
eig(syscl3)
Tf4 = tf(syscl3);
isstable(syscl3)
figure; step(Tf4);
Tfcl2 = d2c(Tf4);
```