Control ELCN304 Project

By

Andrew Amir Ramses Ayad 1180086

Nour Eldin Khaled Said 1180443

Omar Tarek Ahmed Mohamed Ali Amer 1180004

Yousef Alaa Mustafa Ahmed Qandil 1180092

Question 1

Requirement 1

Due to the amplifier, we have

$$V_a = 50\,V_i \qquad (0)$$

And by writing the KVL equation on the motor circuit we get

$$V_a = \mathrm{Ri}_a + e_b \qquad (1)$$

Where e_b is the back emf.

We know that the back emf is proportional to the angular velocity of the motor, this yields equation no. 2

$$V_a = \mathrm{Ri}_a + K_b\dot{\theta} \qquad (2)$$

And from Newton's second law we can obtain equation no.3

$$(J_m + J_l)\ddot{\theta} = K_T i_a$$

Where J_m is the motor inertia, and J_l is the load inertia, we know $J_m = J_l$

$$2J\ddot{\theta} = K_T i_a \qquad (3)$$

And from the given equations:

$$q_o = 51\,h(t) \qquad (4)$$
$$q_1 = 82\,\theta \qquad (5)$$

And if we apply the tank equations, we get equations no. 6 and 7

$$q_1 - q_o = A\frac{d}{\mathrm{d}t}h \qquad (6)$$
$$h = q_0\,R_v \qquad (7)$$

Where R_v is the resistance value of the valve.

Plugging equation no.5 into equation no.6 gives us

$$82\,\theta - q_o = A\frac{d}{\mathrm{d}t}h \quad (8)$$

The SSR equations take the form

$$A\underline{x} + B\underline{u} = \dot{\underline{x}} \quad (9)$$
$$C\underline{x} + D\underline{u} = \underline{y} \quad (10)$$

We choose the position of the shaft as our first state, its angular velocity as the second, and the level of the tank as the third state. The input of the system is V_i and the output is $h(t)$.

$$x_1 = \theta$$
$$x_2 = \dot{\theta}$$
$$x_3 = h(t)$$

We now get the state equations:

$$\dot{x}_1 = x_2 \quad (11)$$

From equation no.3

$$2J\ddot{\theta} = K_T i_a \rightarrow \dot{x}_2 = \frac{K_T}{2J} i_a \quad (12)$$

We need to get i_a in terms of our chosen states, from equations 0 and 2:

$$50\,u = \mathrm{Ri}_a + K_b x_2$$
$$50\,u - K_b x_2 = \mathrm{Ri}_a$$

$$i_a = \frac{-K_b}{R}x_2 + \frac{50}{R}u \quad (13)$$

We can now go back to equation 12:

$$\dot{x}_2 = \frac{-K_T K_b}{2\mathrm{RJ}}x_2 + \frac{50\,K_T}{2\mathrm{RJ}}u \quad (14)$$

From equations 4, 5, and 6:

$$\dot{x}_3 = \frac{82}{A}x_1 - \frac{51}{A}x_3 \quad (15)$$

And finally, the output equation is simply

$$y = x_3 \quad (16)$$

We then substitute equations (11, 14, 15, 16) into equation 9 and 10, the SSR equations:

$$\dot{\underline{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-k_T k_b}{2\mathrm{RJ}} & 0 \\ \frac{82}{A} & 0 & -\frac{51}{A} \end{bmatrix} \underline{x} + \begin{bmatrix} 0 \\ \frac{50k_T}{2\mathrm{RJ}} \\ 0 \end{bmatrix} \underline{u}$$

$$y = [0 \quad 0 \quad 1]\, \underline{x}$$

```
clc
clear % To avoid any potential problems we clear our workspace each time we run the script.
```

We then define the system in code in order to use it later

```
% Definition of constants
kt = 9.5;
kb = 0.0704;
j = 0.0058;
area = 51;
R = 10;

% Definition of SSR matrices A,B,C, and D.
A = [0 1 0; 0 (-kt*kb)/(2*R*j) 0; (82/area) 0 (-51/area)];
B = [0; (50*kt)/(2*R*j); 0];
C = [0 0 1];
D = 0;

% Definition of the main system
sys = ss(A, B, C, D);
```

Requirement 2

We need to check the controllability and observability of our system. The formula for controllability and observability matricies are as follows:

$$\mathrm{Co} = [B \quad \mathrm{AB} \quad A^2B \quad \dots \quad A^{n-1}B]$$

$$\mathrm{Ob} = \begin{bmatrix} C \\ \mathrm{CA} \\ \dots \\ \mathrm{CA}^{n-1} \end{bmatrix}$$

$$\text{Where } A \text{ is } n \times n, \ B \text{ is } n \times r, \text{ and } C \text{ is } m \times n$$

Both Co and Ob need to be of rank n for the system to be controllable and observable respectively. There are built in functions in the *Control System Toolbox* that calculate said matricies for us

```
Co = ctrb(A, B); % Get the controllability matrix for the system.
rankCo = rank(Co); % Get its rank.

Ob = obsv(A, C); % Get the observability matrix for the system.
rankOb = rank(Ob); % Get its rank.
```

```
n = size(A, 1); % A is an n x n matrix. this line gets the value of n.

% Check if the system is controllable or not
if rankCo == n
    fprintf("System is controllable as n = rank(Co) = %d", n)
else
    fprintf("System is not controllable as n doesn't equal rank(Co),\nn = %d\nrank(Co) = %d", r
end
```

```
System is controllable as n = rank(Co) = 3
```

```

% Check if the system is observable or not
if rankOb == n
    fprintf("System is observable as n = rank(Ob) = %d", n)
else
    fprintf("System is not observable as n doesn't equal rank(Ob),\nn = %d\nrank(Ob) = %d", n,
end
```

```
System is observable as n = rank(Ob) = 3
```

Requirement 3

We need to get the transfer function of our system. This can be easily obtained using the one-liner shown below:

```
TF = tf(sys)
```

```
TF =

            6584
  -------------------------
  s^3 + 6.766 s^2 + 5.766 s

Continuous-time transfer function.
```

Requirement 4

We know the poles of any system are the eigen values of its ***A*** matrix. We can use the formula shown below and solve for *lambda* to get the eigen values (poles) of the system.

$$|\lambda I - A| = 0$$

From *requirement 1* we know that:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-k_T k_b}{2\mathrm{RJ}} & 0 \\ \frac{82}{A} & 0 & -\frac{51}{A} \end{bmatrix}$$

And after plugging in the values of all constants, we get

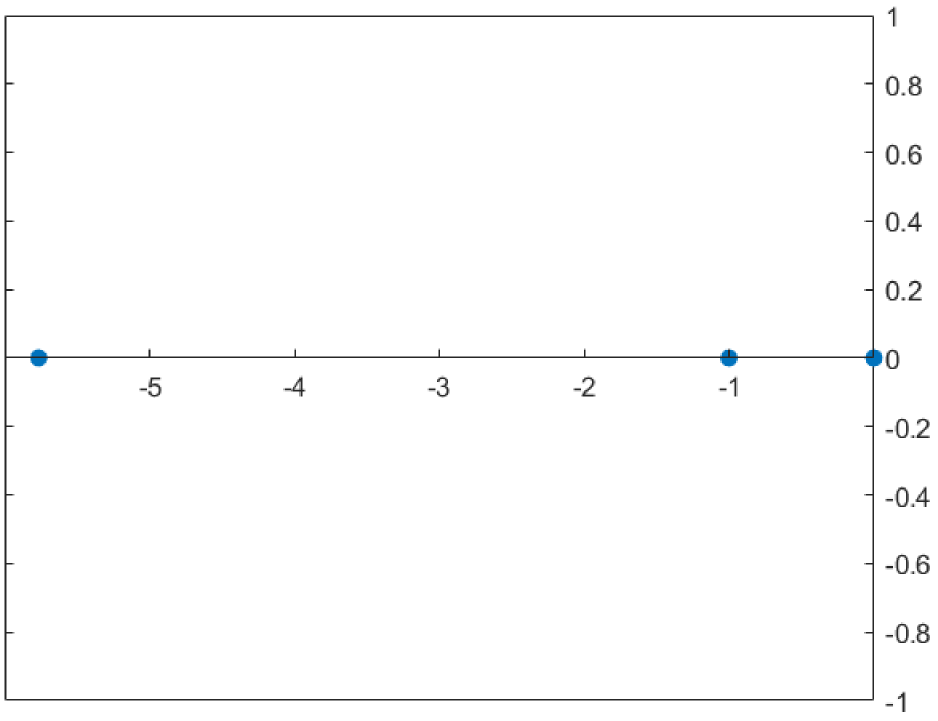
$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-836}{145} & 0 \\ \frac{82}{51} & 0 & -1 \end{bmatrix}$$

We now get the eigen values of the A matrix:

```
poles = eig(A)
```

```
poles = 3×1
   -1.0000
         0
   -5.7655
```

```
% Plot the poles
stem(real(poles),imag(poles), 'filled')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
```



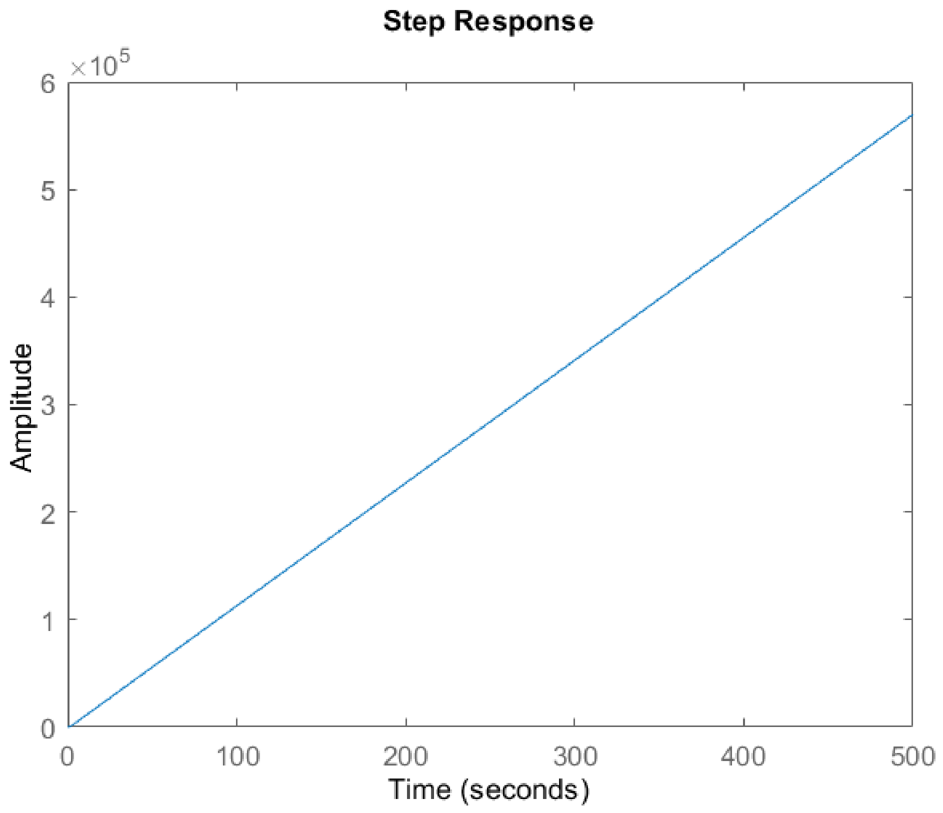
Denoted by the blue dots are the poles of our system.

Having a pole at the origin makes the system critically stable.

Requirement 5

When we operate the system with $V_i = 1\ V$ we get the following response:

```
step(sys)
```



Here we can see that the system is unstable. Applying the stability equality on the characteristic equation $a_2a_1 > a_3a_0$ further proves our point.

This means that $e_{ss} = \infty$

Requirement 6

From req. 5 we can see that a steady state response value for a step input doesnt exist. this can be fixed by adding a DC gain that is less than a certain factor K and a feedback path.

In the next code section we attempt to calculate this factor,.

We know that the stability condition for a third order system

$$a_2a_1 > a_3a_0$$

$$\text{where } a_i \text{ is the coeffecient of } x^i \text{ in the characteristic equation}.$$

Before stabilizing the system, the characteristic equation is

$$s^3 + 6.766s^2 + 5.766s + 6584 = 0$$

When we apply the condition we can see that

$$6.766 \times 5.766 \not> 1 \times 6584$$

To find the required gain we solve this simple inequality

$$6.766 \times 5.766 > 6584K$$

$$\text{where } K \text{ is the stabilizing gain value.}$$

```
k = (6.766*5.766) / 6584;
fprintf("Solution: K < %f", k)
```

```
Solution: K < 0.005925
```

We need $K < 0.005925$. Let $K = 0.002$

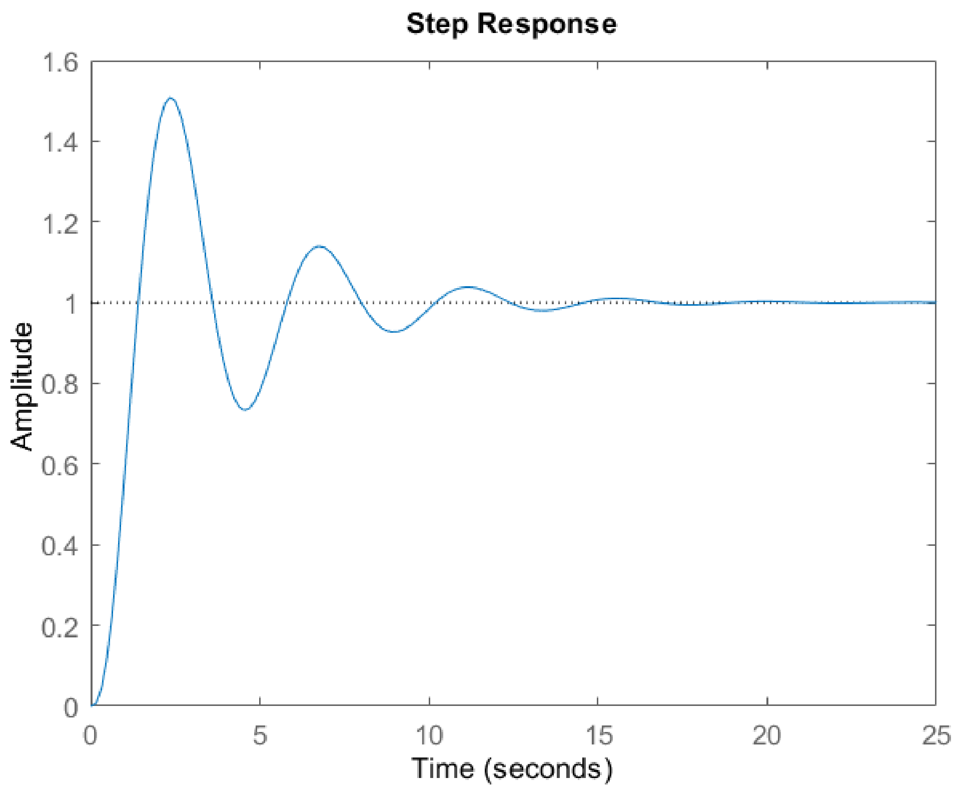
```
stabilizedOpenLoop = series(TF, 0.002)
```

```
stabilizedOpenLoop =
 
            13.17
  -------------------------
  s^3 + 6.766 s^2 + 5.766 s
 
Continuous-time transfer function.
```

```
stabilizedSystem = feedback(stabilizedOpenLoop,1) % Form the system by adding feedback.
```

```
stabilizedSystem =
 
                13.17
  ---------------------------------
  s^3 + 6.766 s^2 + 5.766 s + 13.17
 
Continuous-time transfer function.
```

```
step(stabilizedSystem)
```



```
stepinfo(stabilizedSystem)
```

```
ans = struct with fields:
        RiseTime: 0.8690
    SettlingTime: 13.3636
     SettlingMin: 0.7333
     SettlingMax: 1.5090
       Overshoot: 50.8975
      Undershoot: 0
            Peak: 1.5090
        PeakTime: 2.3485
```

Here we can see that the system is now stable under the effect of a step input. However the rising time is very large.

Requirement 7

We now study the effect of controlling our system using the controller

$$\frac{G_c(s)}{s + 1000} \text{ where } G_c(s) = 1$$

```
controller = tf([0 1], [1 1000]) % Define the controller
```

```
controller =
 
     1
  --------
  s + 1000
 
Continuous-time transfer function.
```

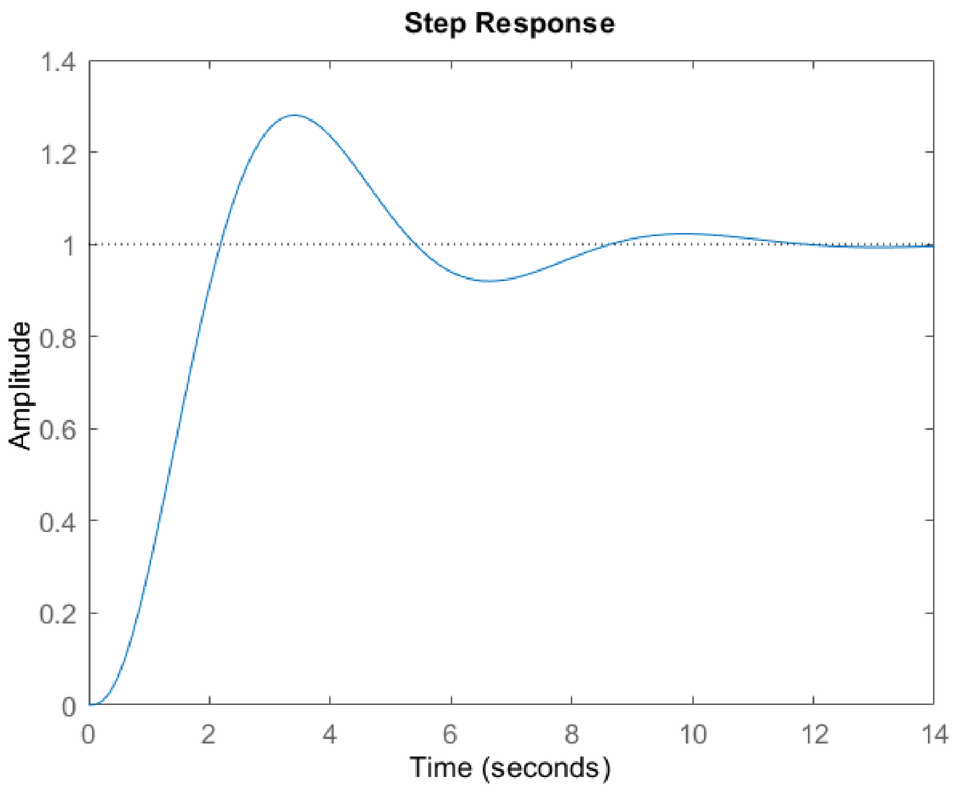
```
piCtrlSysFF = controller* TF  % PI controlled system (after stabilization) - Feedforward.
```

```
piCtrlSysFF =
 
                 6584
  ----------------------------------
  s^4 + 1007 s^3 + 6771 s^2 + 5766 s
 
Continuous-time transfer function.
```

```
piSys = feedback(piCtrlSysFF, 1) % PI controlled system - Full system.
```

```
piSys =
 
                    6584
  -----------------------------------------
  s^4 + 1007 s^3 + 6771 s^2 + 5766 s + 6584
 
Continuous-time transfer function.
```

```
step(piSys)
```



```
[y,t]=step(piSys);    % Get the response of our system.
sserror=abs(1-y(end)); % Get the steady state error.
fprintf("Steady state Error = %f", sserror)
```

```
Steady state Error = 0.005956
```

```
stepinfo(piSys)
```

```
ans = struct with fields:
        RiseTime: 1.3874
    SettlingTime: 10.3608
     SettlingMin: 0.9148
     SettlingMax: 1.2802
       Overshoot: 28.0195
      Undershoot: 0
            Peak: 1.2802
        PeakTime: 3.4323
```

Adding the controller with the feedback minimized the steady state error and stabilized the system.

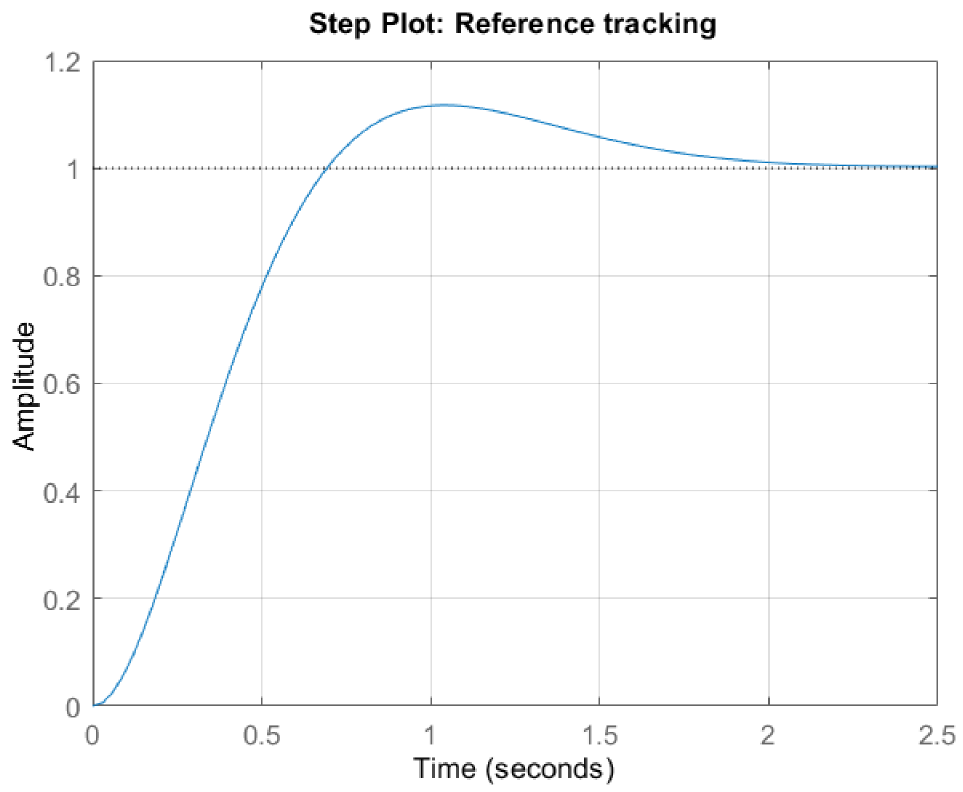
Requirement 8

We need the settling time to be less than 3 secs, and the maximum overshoot to be less than 15%. We now design the controller.

We choose the **PD** controller as we need to enhance the settling time and the overshoot only.

We need the settling time to be less than 3 secs, and the maximum overshoot to be less than 15%. We now start tuning our system. We mainly tune using trial and error.

```
% Convert Response Time to Bandwidth
% Bandwidth is equivalent to 2 divided by the Response Time
wc = 2/0.729191;

% Convert Transient Behavior to Phase Margin
% Phase Margin is equivalent to the Transient Behavior multiplied by 100
PM = 100*0.584263;

% Define options for pidtune command
opts = pidtuneOptions('PhaseMargin',PM);

% PID tuning algorithm for linear plant model
[G_c,pidInfo] = pidtune(piCtrlSysFF,'PD',wc,opts);

% Clear Temporary Variables
clear wc PM opts

% Get desired loop response
Response = getPIDLoopResponse(G_c,piCtrlSysFF,'closed-loop');

% Plot the result
stepplot(Response)
title('Step Plot: Reference tracking')
grid on
```



```matlab
% Display system response characteristics
disp(stepinfo(Response))
```

```
        RiseTime: 0.4679
    SettlingTime: 1.8317
     SettlingMin: 0.9018
     SettlingMax: 1.1170
       Overshoot: 11.7031
      Undershoot: 0
            Peak: 1.1170
        PeakTime: 1.0369
```

```matlab
% Clear Temporary Variables
clear Response
```

This gives us the follwing the controller:

```matlab
disp(G_c)
```

```
  pid with properties:

             Kp: 3.4048
             Ki: 0
             Kd: 2.5444
             Tf: 0
       IFormula: ''
       DFormula: ''
     InputDelay: 0
    OutputDelay: 0
```

```
             Ts: 0
       TimeUnit: 'seconds'
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1×1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1×1 struct]
          Notes: [0×1 string]
       UserData: []
           Name: ''
   SamplingGrid: [1×1 struct]
```

Question 2

$$F(s) = \frac{10(s+10)}{(s+1)(s+100)} = \frac{10s+100}{s^2+101s+100}$$

$$\frac{\theta_0(s)}{\theta_i(s)} = \frac{K_a * F(s) * \frac{K}{s}}{1 + K_a * F(s) * \frac{K}{s}} = \frac{K_a * \frac{10s+100}{s^2+101s+100} * \frac{K}{s}}{1 + K_a * \frac{10s+100}{s^2+101s+100} * \frac{K}{s}} = \frac{K_a * K * (10s+100)}{s(s^2+101s+100) + K_a * K * (10s+100)}$$

$$= \frac{10 * K_a * \mathrm{Ks} + 100 * K_a * K}{s^3 + 101s^2 + s(100 + 10K_a * K) + 100 * K_a * K}$$

3rd order system compare characteristic function with

$(s+\alpha)\left(s^2 + 2\zeta\omega_n s + \omega_n{}^2\right) = 0$

$s^3 + s^2(2\zeta\omega_n + \alpha) + s\left(\omega_n{}^2 + 2\zeta\omega_n\alpha\right) + \alpha\omega_n{}^2 = 0$

$s^3 + 101s^2 + s(100 + 10 * K_a * K) + 100 * K_a * K = 0$

for stability $a_2 a_1 > a_3 a_0$

$(101)(100 + 10 * K_a * K) > (1)(100 * K_a * K)$

$10100 + 1010 * K_a * K > 100 * K_a * K$

$1010 * K_a * K - 100 * K_a * K > -10100$

$910 * K_a * K > -10100$

$K_a * K > \frac{-10100}{910}$, $K_a * K > -\frac{1010}{91}$, $K_a * K > -11.0989$

All Coefficients must be larger than zero

$100 + 100 + 10 * K_a * K > 0$, $K_a * K > -10$

& $100 * K_a * K > 0$, $K_a * K > 0$

For the system to be stable $K_v > 0$

$$e_{\text{ss unit step}} = \frac{1}{1 + K_p},\ K_p = \lim_{s\to 0} G(s)H(s)$$

$$K_p = \lim_{s\to 0} K_a * \frac{10s+100}{s^2+101s+100} * \frac{K}{s}$$

$K_p = 0$

$e_{\text{ss unit step}} = \infty$

$e_{\text{ss unit ramp}} = \frac{1}{K_v}, \; K_v = \lim_{s \to 0} sG(s)H(s)$

$$K_v = \lim_{s \to 0} s * K_a * \frac{10s + 100}{s^2 + 101s + 100} * \frac{K}{s}$$

$$K_v = \lim_{s \to 0} \frac{10 * K_a * K * s + 100 * K_a * K}{s^2 + 101s + 100}$$
$$= K_a * K$$

From the comparison with characteristic Equation, we get three equations

$2\zeta\omega_n + \alpha = 101$

${\omega_n}^2 + 2\zeta\omega_n\alpha = 100 + 10 * K_a * K$

$\alpha{\omega_n}^2 = 100 * K_a * K$

```
clear  % Clear : This is a different problem and we dont need the variables from the previous 
clc

%Constants Definition
Ka = 1;                   %Amplifier Constant
K = 1;

Kv = K * Ka;




%Filter F(s)
F_num = [10 100];
F_den = conv([1 1], [1 100]); %(s+1)(s+100)
F = tf(F_num,F_den)
```

```
F =

     10 s + 100
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

```

%open loop transfer function:We have 3 series blocks, and we multiply them all
Oltf_num = conv(Kv, F_num);
Oltf_den = conv([1 0], F_den);
Oltf = tf(Oltf_num,Oltf_den)
```

```
Oltf =
       10 s + 100
  ---------------------
  s^3 + 101 s^2 + 100 s

Continuous-time transfer function.
```

```
%closed loop function with unity feedback
[Cltf_num,Cltf_den] = cloop(Oltf_num,Oltf_den);
Cltf = tf(Cltf_num,Cltf_den)
```

```
Cltf =

          10 s + 100
  ---------------------------
  s^3 + 101 s^2 + 110 s + 100

Continuous-time transfer function.
```

```
%Steady State error for unit Step
kp = dcgain(Oltf)            %dcgain function is the lim of s tends to zero
```

```
kp = Inf
```

```
ess = 1/1+kp
```

```
ess = Inf
```

```
%Steady State error for unit Ramp
G = tf(conv(Oltf_num, [1 0]),Oltf_den) %G = S*Oltf
```

```
G =

     10 s^2 + 100 s
  ---------------------
  s^3 + 101 s^2 + 100 s

Continuous-time transfer function.
```

```
kv = dcgain(G)
```

```
kv = 1
```

```
ess2 = 1/kv
```

```
ess2 = 1
```

```
%Step Response
%hold on;
subplot(1,2,1);
step(Cltf);
title('Step Response');
grid on;

% the maximum overshoot, the rise time, and the settling time.
TransientData = stepinfo(Cltf)
TransientData = struct with fields:
        RiseTime: 1.7185
    SettlingTime: 5.7229
     SettlingMin: 0.9228
     SettlingMax: 1.1303
       Overshoot: 13.0252
      Undershoot: 0
            Peak: 1.1303
        PeakTime: 3.6302
```

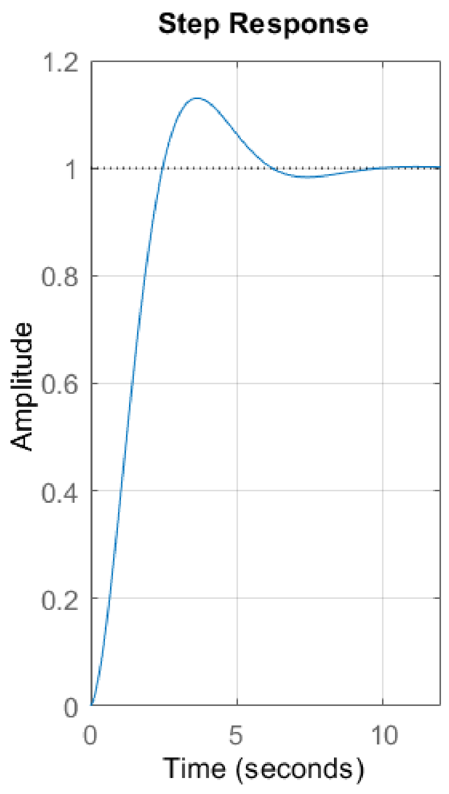
```
[poles1,dampingRatio1,naturalFreq1] = damp(Cltf)
```

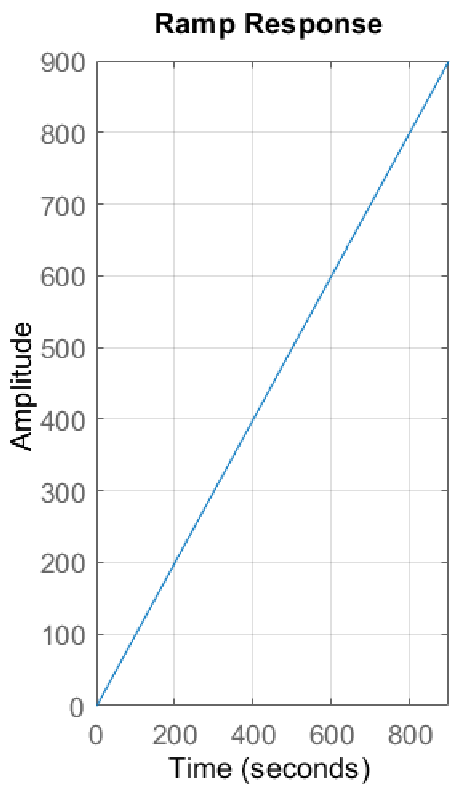
```
poles1 = 3×1
    1.0005
    1.0005
   99.9090
dampingRatio1 = 3×1
    0.5452
    0.5452
    1.0000
naturalFreq1 = 3×1 complex
  -0.5455 + 0.8387i
  -0.5455 - 0.8387i
 -99.9090 + 0.0000i
```

```


%Ramp Response
Gr = tf(Cltf_num,conv(Cltf_den, [1 0])) %We multiply the denominator of CLTF by s and then get
```

```
Gr =
 
            10 s + 100
  ------------------------------
  s^4 + 101 s^3 + 110 s^2 + 100 s
 
Continuous-time transfer function.
```

```
subplot(1,2,2);
step(Gr);                                   %therefore here (1/s^2) *CLTF
title('Ramp Response');
grid on;
```


Step Response
Amplitude
Time (seconds)
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10


Ramp Response
Amplitude
Time (seconds)
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800