Design and Control of DC Motor

Spring - 2023/2024

MKT3122 – Control System Theory

MATLAB/Simulink Code and Technical Report

Submitted By: Göktuğ Can Şimay, Ali Doğan

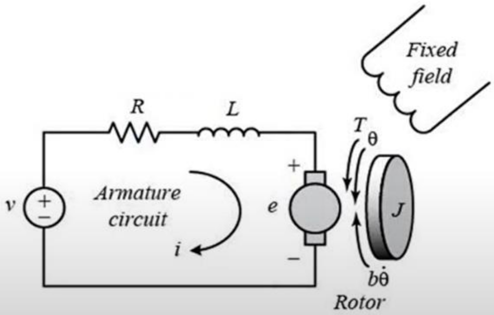
Student ID: 22067606, 22067605

Date: 11.05.2024

CONTENT

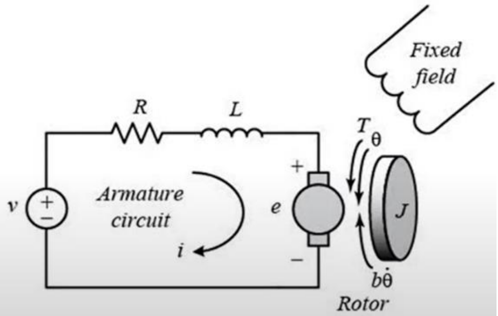
DC Motor

- The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.



DC Motor

- The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.
- The input of the system is the voltage source (v) that is applied to the motor's armature, while the output is the rotational speed of the shaft (ω)

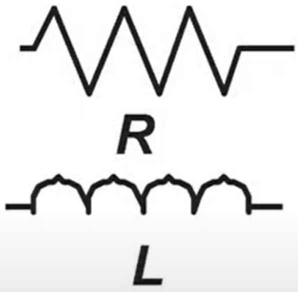


DC Motor

- System Parameters
  - ✓ Electrical Part
  - ✓ Mechanical Rotating Part

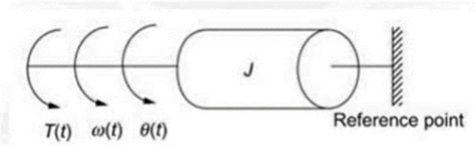
DC Motor

- Basic Element of Electrical System
  - ✓ Resistance (R)
    - ➢ Voltage: $v = Ri$
    - ➢ Current: $i = \frac{v}{R}$
  - ✓ Inductance (L)
    - ➢ Voltage: $v = L\frac{di}{dt}$
    - ➢ Current: $i = \frac{1}{L}\int_{t_o}^{t} v dt + v(t_o)$



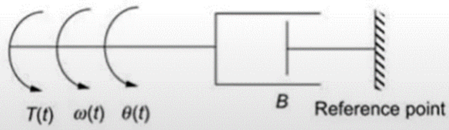
- Basis Element of Mechanical Rotating System Parameters
  - ✓ Inertia

$$T = J\alpha = J\dot{\omega}$$



  - ✓ Damper

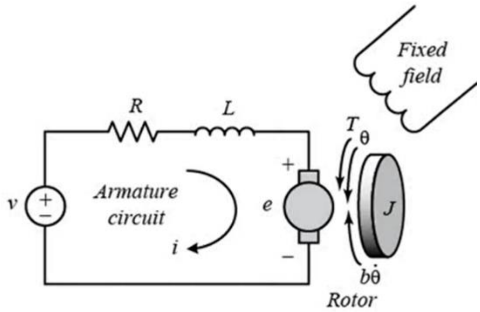
$$T = b\omega$$



- Electrical Part
  - ✓ Applying Kirchhoff's Voltage Law (KVL)

$$v - Ri - L\frac{di}{dt} - v_{emf} = 0$$

$$v - Ri - L\frac{di}{dt} - K_e\omega = 0$$



$$v_{emf} = K_e\omega$$

$$T = K_t i$$

- Mechanical Part
  - ✓ Applying Newton's 2nd law

$$T - J\dot{\omega} - b\omega = 0$$

$$K_t i - J\dot{\omega} - b\omega = 0$$

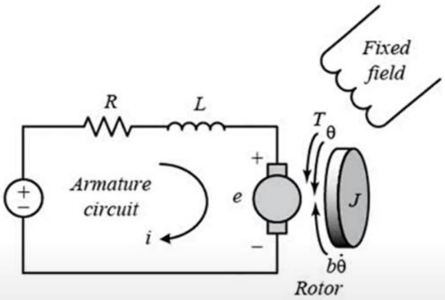
Transfer Function

- Electrical Part
  - ✓ Applying Kirchhoff's Voltage Law (KVL)

$$v - Ri - L\frac{di}{dt} - K_e\omega = 0$$

  - ✓ Applying the Laplace transform

$$V(s) - RI(s) - LsI(s) - K_e\Omega(s) = 0$$

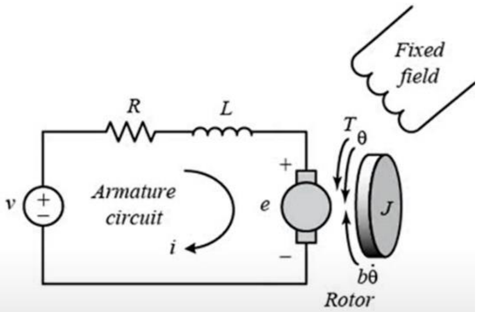


- Mechanical Part
  - ✓ Applying Newton's 2nd law $K_t i - J\dot{\omega} - b\omega = 0$
  - ✓ Applying the Laplace transform $K_t I(s) - Js\Omega(s) - b\Omega(s) = 0$

- Rearrange Equations

$$\frac{V(s) - K_e\Omega(s)}{Ls + R} = \frac{\Omega(s)(Js + b)}{K_t}$$

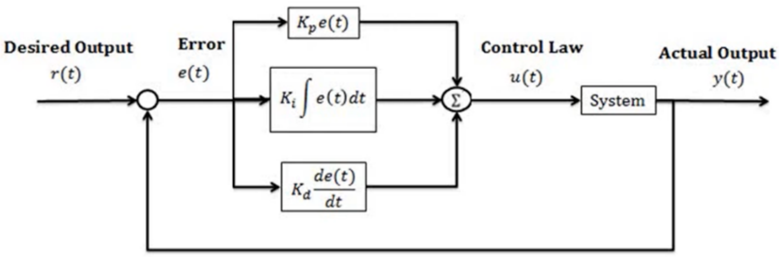
$$K_t V(s) = \Omega(s)\big((Js + b)(Ls + R) + K_e K_t\big)$$



- The Transfer Function between rotational speed $\Omega(s)$ as the output and the armature voltage $V(s)$ as the input is:

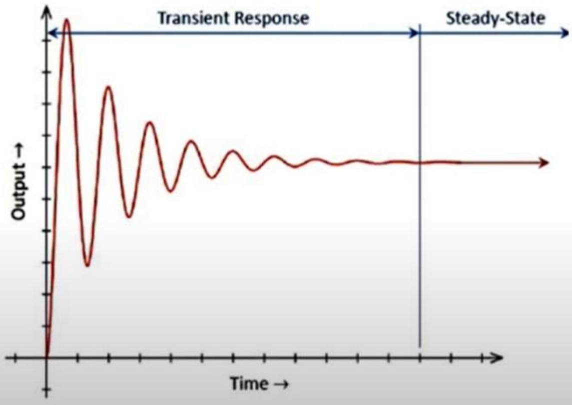
$$\frac{\Omega(s)}{V(s)} = \frac{K_t}{\big((Js + b)(Ls + R) + K_e K_t\big)} = \frac{K_t}{JLs^2 + (JR + bL)s + (bR + K_e K_t)}$$

- Proportional-Integral-Derivative (PID)
  - ✓ There are three terms in the PID controller
    - ➢ Proportional
    - ➢ Integral
    - ➢ Derivative



Controller Design

- **Regulator**
  - ✓ Improve the transit response of the system (guaranteeing the stability of the closed-loop system).
- **Tracking**
  - ✓ Follow reference input with zeros steady state error.



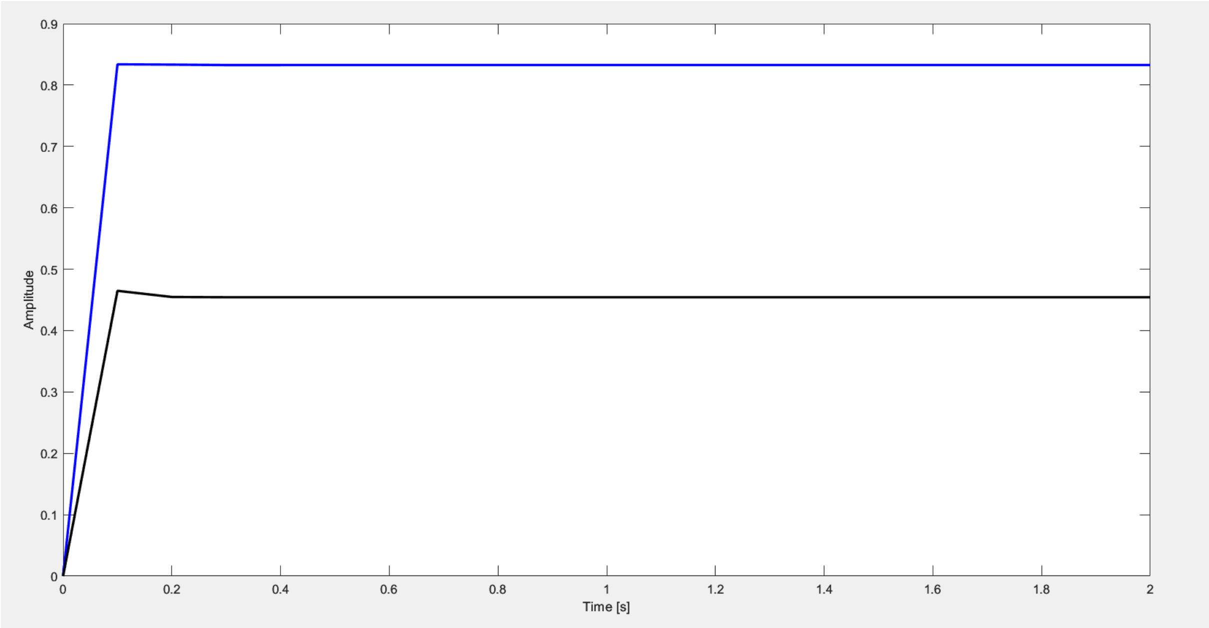
✓ Characteristics of PID Coefficients

Parameters	Overshoot	Settling Time	Steady State Error
K_p	Increase	Minor Change	Decrease
K_i	Increase	Increase	Eliminate
K_d	Decrease	Decrease	Minor Change

- The system parameters

Parameters	Symbol	Values/ Units
Moment of Inertia of the Rotor	J	$0.022\ K_g m^2$
Motor Viscous Friction Constant	b	0.5×10^{-3} N.m/($\frac{\text{rad}}{\text{sec}}$)
Electromotive Force Constant	K_e	1.2 v/($\frac{\text{rad}}{\text{sec}}$)
Motor Torque Constant	K_t	1.2 N.m/Amp
Electric Resistance	R	2.45Ω
Electric Inductance	L	0.035 mH

```
% Smülasyon Zamanı
t_sim = 2;

% Sistem Parametreleri
J = 0.022;
b = 0.5e-3;
ke = 1.2;
kt = 1.2;
R = 2.45;
L = 0.035;

% Sistem Parametrelerinin Hesaplanması
n = kt;
d = [J*L (J*R+b*L) (b*R+ke*kt)];
G = tf(n, d);

% Sistem Simülasyonu ve Çıktıyı Çizdirme
tsim = 0:0.1:t_sim;
u = ones(size(tsim));
y = lsim(G, u, tsim);

figure(1)
plot(tsim, y, 'b', 'LineWidth', 2);
xlabel('Time [s]');
ylabel('Amplitude');

%PID Kontrolcüsü Tanımı ve Geri Besleme Döngüsü
kp = 1;
ki = 0;
kd = 0;
Gc = pid(kp, ki, kd);
Gcl = feedback(Gc*G, 1);

%Kapalı Döngü Yanıtı Simülasyonu ve Çıktıyı Çizdirme
tsim = 0:0.1:t_sim;
u = ones(size(tsim));
y = lsim(Gcl, u, tsim);

hold on;
plot(tsim, y, 'k', 'LineWidth', 2);
xlabel('Time [s]');
ylabel('Amplitude');
```



Amplitude
Time [s]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

State Space

- Electrical Part
  - ✓ Applying Kirchhoff's Voltage Law (KVL)

$$v - Ri - L\frac{di}{dt} - K_e\omega = 0$$

  - ✓ Rewrite the Equation as:

$$\frac{di}{dt} = \frac{1}{L}v - \frac{R}{L}i - \frac{K_e}{L}\omega$$

- Mechanical Part
  - ✓ Applying Newton's 2nd law $K_t i - J\dot{\omega} - b\omega = 0$
  - ✓ Rewrite the Equation as: $\frac{d\omega}{dt} = \frac{K_t}{J}i - \frac{b}{J}\omega$

- **State Space**

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

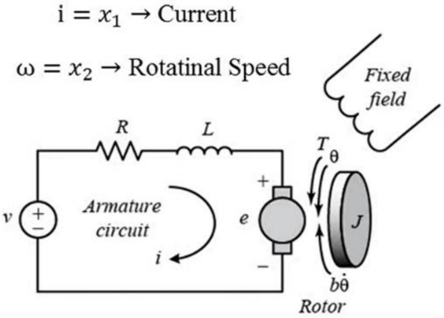
$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-\frac{R}{L} & -\frac{K_e}{L}\\ \frac{K_t}{J} & -\frac{b}{J}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}\frac{1}{L}\\ 0\end{bmatrix}v$$

$y = [0\ 1]\begin{bmatrix}x_1\\ x_2\end{bmatrix}$ Controlling the speed of the motor

$$A = \begin{bmatrix}-\frac{R}{L} & -\frac{K_e}{L}\\ \frac{K_t}{J} & -\frac{b}{J}\end{bmatrix} \qquad B = \begin{bmatrix}\frac{1}{L}\\ 0\end{bmatrix} \qquad C = [0\ 1] \qquad D = 0$$

$i = x_1 \rightarrow$ Current

$\omega = x_2 \rightarrow$ Rotatinal Speed



$$\frac{di}{dt} = \frac{1}{L}v - \frac{R}{L}i - \frac{K_e}{L}\omega$$

$$\frac{d\omega}{dt} = \frac{K_t}{J}i - \frac{b}{J}\omega$$

Controller Design

- **Design Control Law**
  - ✓ Control action compute based on the states of the system and reference input and selected on the form of:

$$u(t) = k_r r(t) - k_x x(t)$$

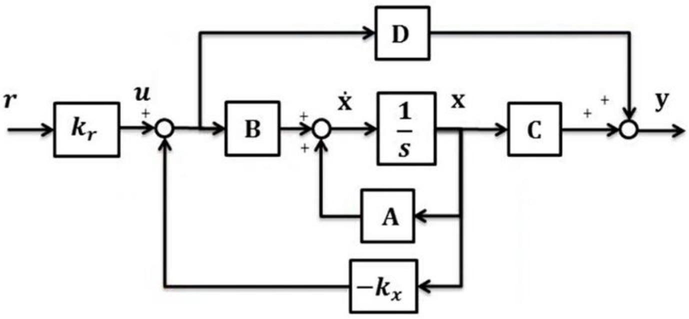
$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$\dot{x}(t) = Ax(t) + B(k_r r(t) - k_x x(t))$$

$$\dot{x}(t) = Ax(t) + Bk_r r(t) - Bk_x x(t)$$

$$\dot{x}(t) = [\mathbf{A} - \mathbf{Bk_x}]x(t) + \mathbf{Bk_r} r(t)$$

State Feedback Controller + Forward Gain



Pole Placement → k_x

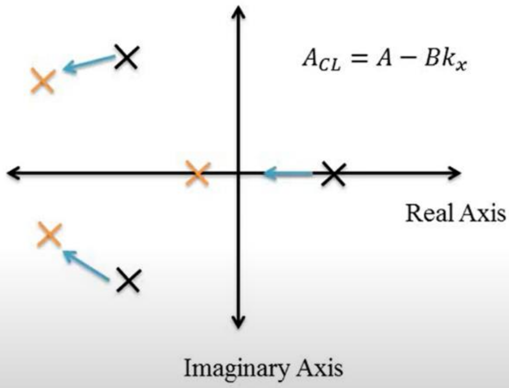
$$\boxed{\begin{aligned} A_{OL} &= A \\ A_{CL} &= A - Bk_x \end{aligned}}$$

DC gain → k_r

$$\boxed{\begin{aligned} B_{OL} &= B \\ B_{CL} &= Bk_r \end{aligned}}$$

Controller Design

- **Pole Placement**
  - ✓ The concept of pole placement is to locate the closed loop poles of the system at $p_1, p_2, ..$ which are their 'desired locations.
  - ✓ The gain matrix k_x is designed to get the desired poles location $p_1, p_2, ..$

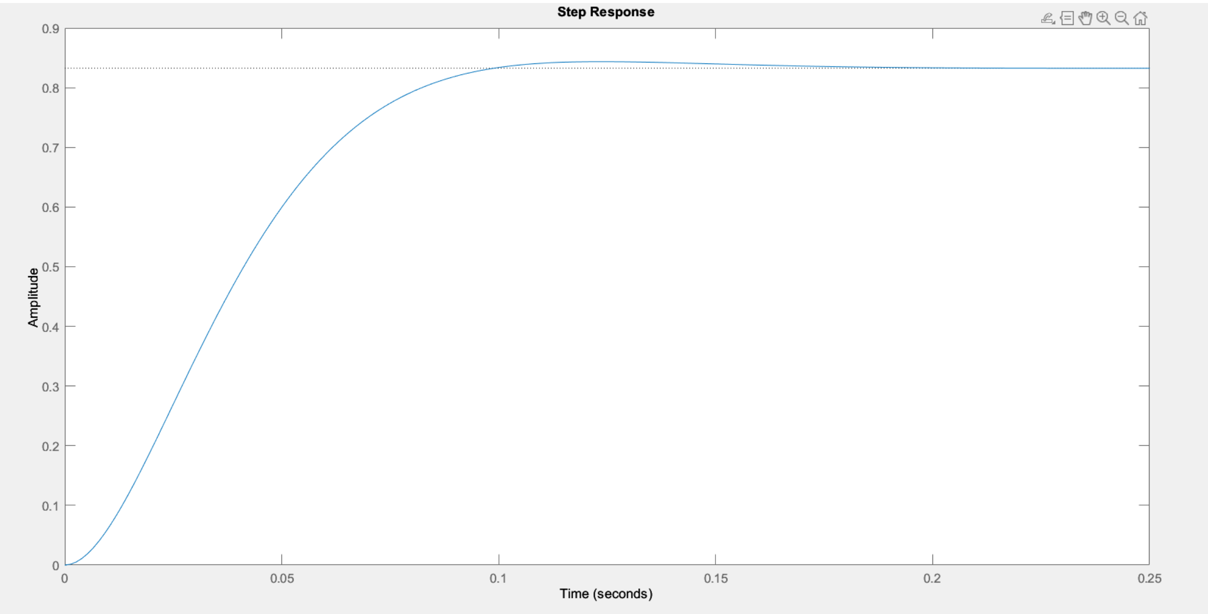


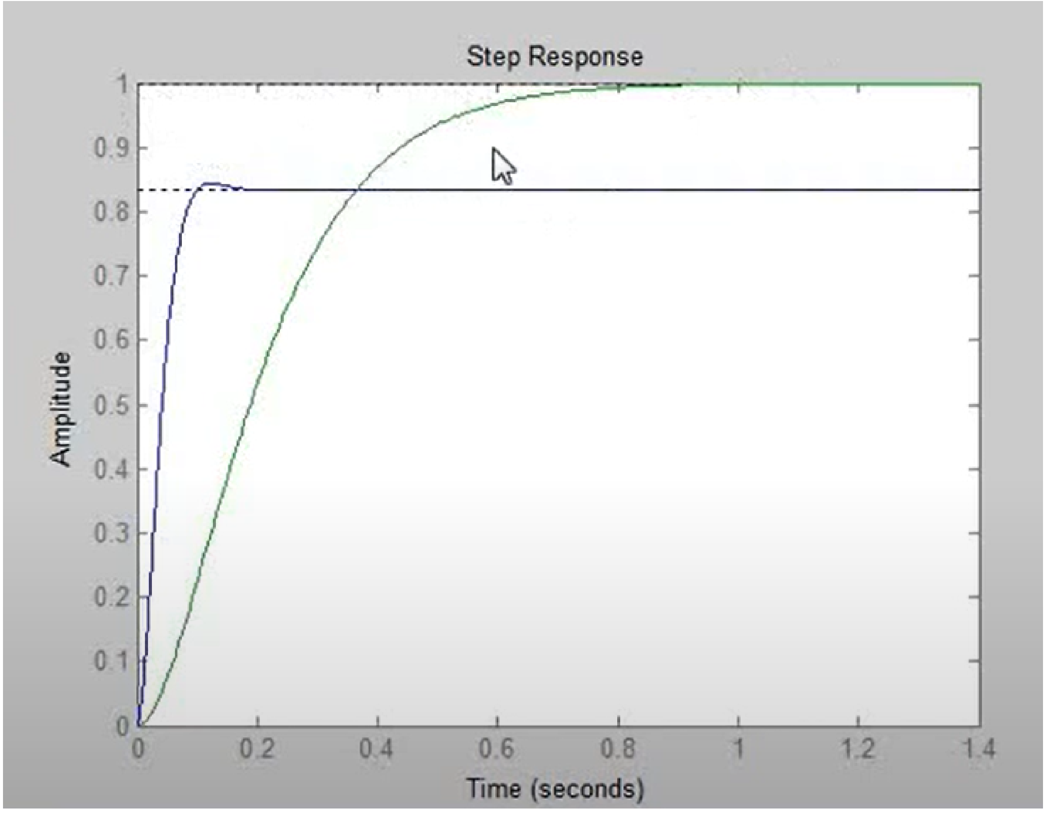
- **Linear Quadratic Regulator**
  - ✓ The concept of LQR control is to find the optimal control action $u^*(t)$ that makes the system, reach the steady state and guarantee the performance index J takes minimum value
  - ✓ The index J is given by

$$J = \int_0^\infty (X^T QX + u^T Ru)dt$$

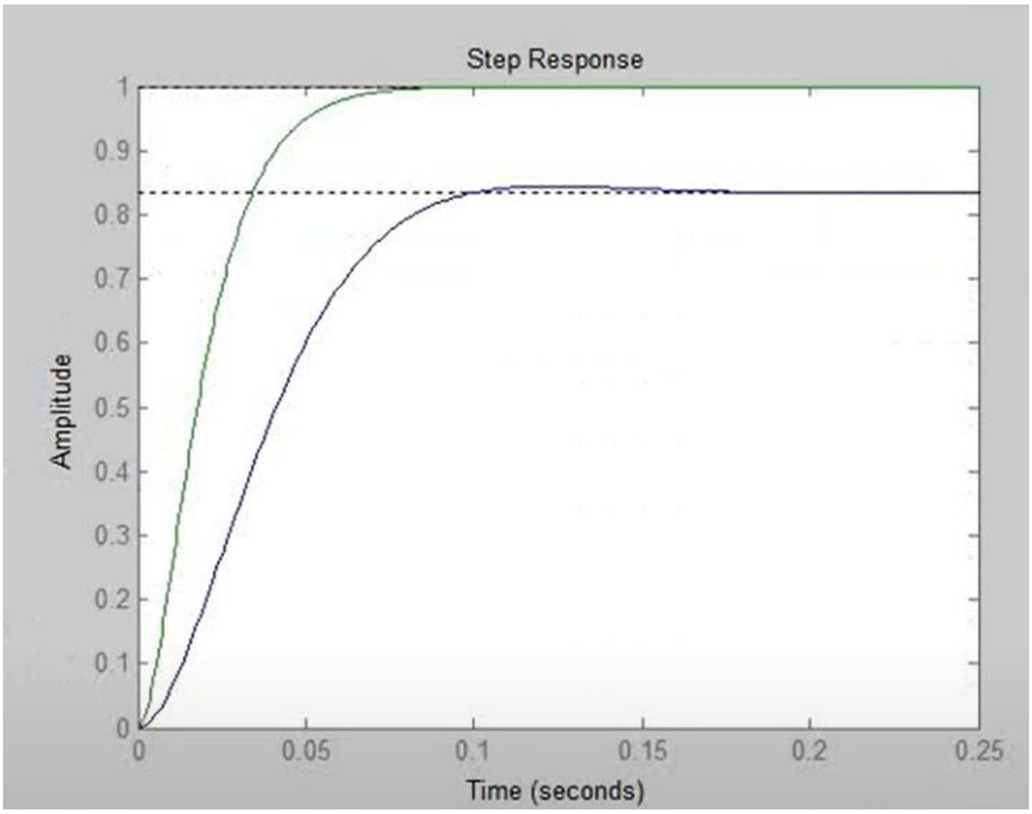
- The matrices Q and R are adjustable matrices.
  - ✓ Q square matrix with rows equal the number of the state
  - ✓ R square matrix with rows equal to the number of the input
- The gain matrix k_x is designed to get the desired trade off:
  - ✓ Q → Response fast, u → Large
  - ✓ R → Response slow, u → Small

```matlab
% System Parameters
J = 0.022;
b = 0.5e-3;
ke = 1.2;
kt = 1.2;
R = 2.45;
L = 0.035;

% State Space Model
A = [-R/L -ke/L; kt/J -b/J];
B = [1/L; 0];
C = [0 1];
D = 0;

sys_ol = ss(A, B, C, D);

p_cl = [-8 -10];
kx = place(A, B, p_cl);
Ac = A - B * kx;

sysx = ss(Ac, B, C, D);
kr = 1/dcgain(sysx);
B_cl = B * kr;

figure;
step(sys_ol)

Q = [10 0; 0 100];
R = 0.01;
kx = lqr(A, B, Q, R);
Ac = A - B * kx;
```





(pcl -8, -10)



(pcl -90, -100)

- **Controller Design**
  - **State Feedback Control with Integral Action**

Plant: $\dot{x} = Ax + Bu$

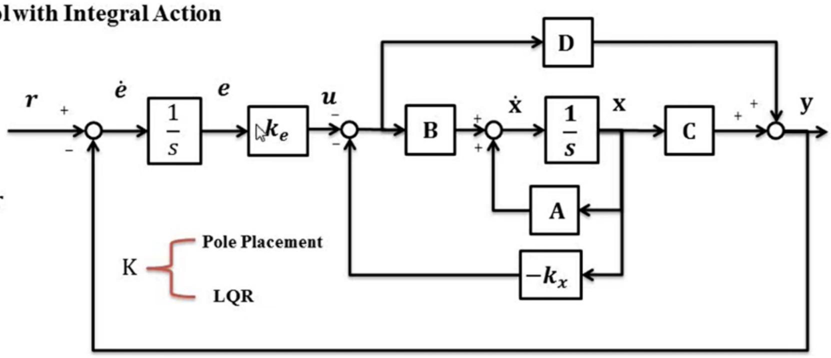
$\dot{e} = r - y = r - Cx$

$$\underbrace{\begin{bmatrix}\dot{x}\\ \dot{e}\end{bmatrix}}_{\dot{x}_e} = \underbrace{\begin{bmatrix}A & 0\\ -C & 0\end{bmatrix}}_{A_e}\underbrace{\begin{bmatrix}x\\ e\end{bmatrix}}_{x_e} + \underbrace{\begin{bmatrix}B\\ 0\end{bmatrix}}_{B_e}u + \underbrace{\begin{bmatrix}0_x\\ 1\end{bmatrix}}_{B_r}r$$

$$\dot{x}_e = A_e x_e + B_e u + B_r r$$

$$u = r - Kx_e$$

$$u = r - \begin{bmatrix}k_x\\ k_e\end{bmatrix}[x\ e]$$

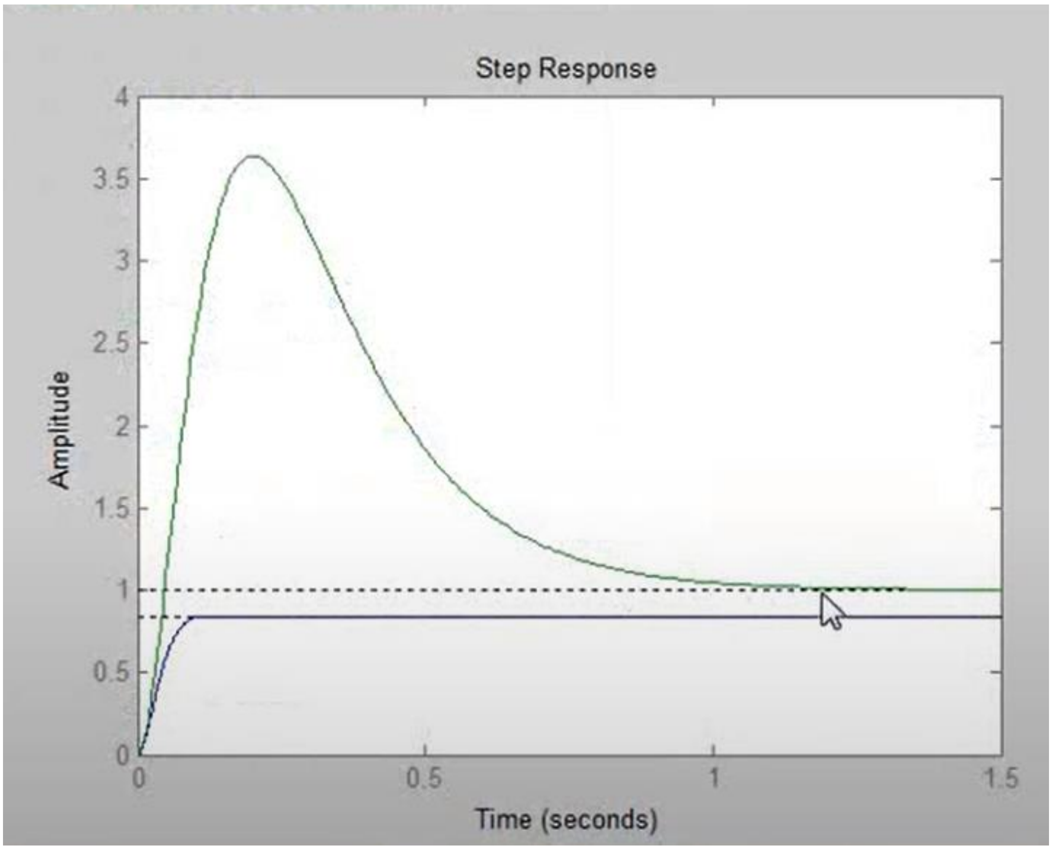


$$\dot{x}_e = A_e x_e + B_e(r - Kx_e) + B_r r \qquad \boxed{A_{cl} = A_e - KB_e} \qquad \boxed{B_{cl} = B_e + B_r}$$

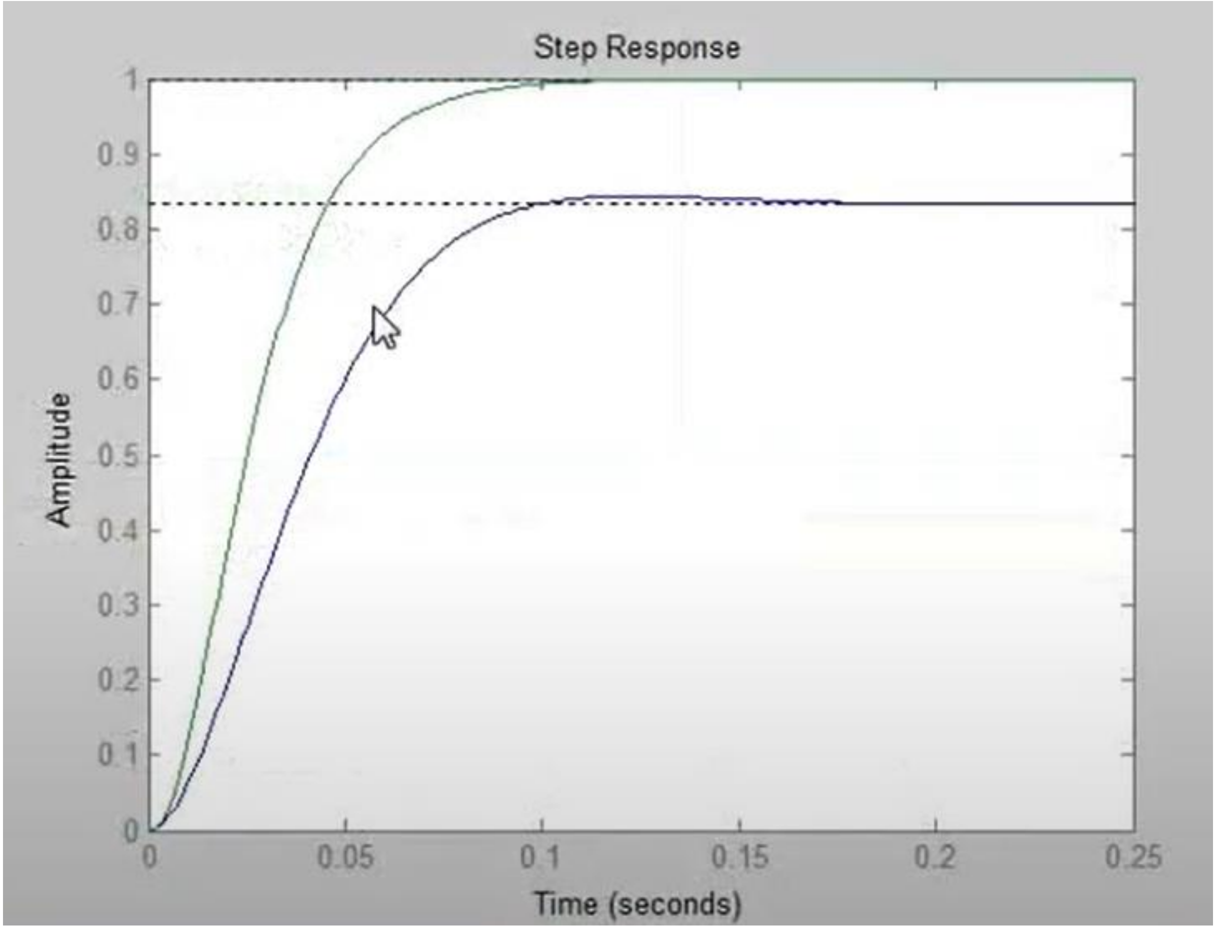
$$\dot{x}_e = A_e x_e + B_e r - B_e K x_e + B_r r \rightarrow \dot{x}_e = (A_e - KB_e)x_e + (B_e + B_r)r$$

```
J = 0.022;
b = 0.5e-3;
ke = 1.2;
kt = 1.2;
R = 2.45;
L = 0.035;

A = [-R/L -ke/L; kt/J -b/J];
B = [1/L; 0];
C = [0 1];
D = 0;

sys_ol = ss(A, B, C, D);
figure;
step(sys_ol)

Ai = [A zeros(2,1); -C 0];
Bi = [B; 0];
Ci = [C 0];

% Pole Placement
% Select Desired Poles Location
p_cl = [-150 -140 -60];
kx = place(Ai, Bi, p_cl);
Ac = Ai - Bi * kx;
Br = [0; 0; 1];
Bc = Bi + Br;

% Closed Loop System
sys_cl = ss(Ac, Bc, Ci, D);
hold on;
step(sys_cl);

% LQR
% Select Desired Q & R
Q = [100 0 0; 0 1000 0; 0 0 1000000];
```

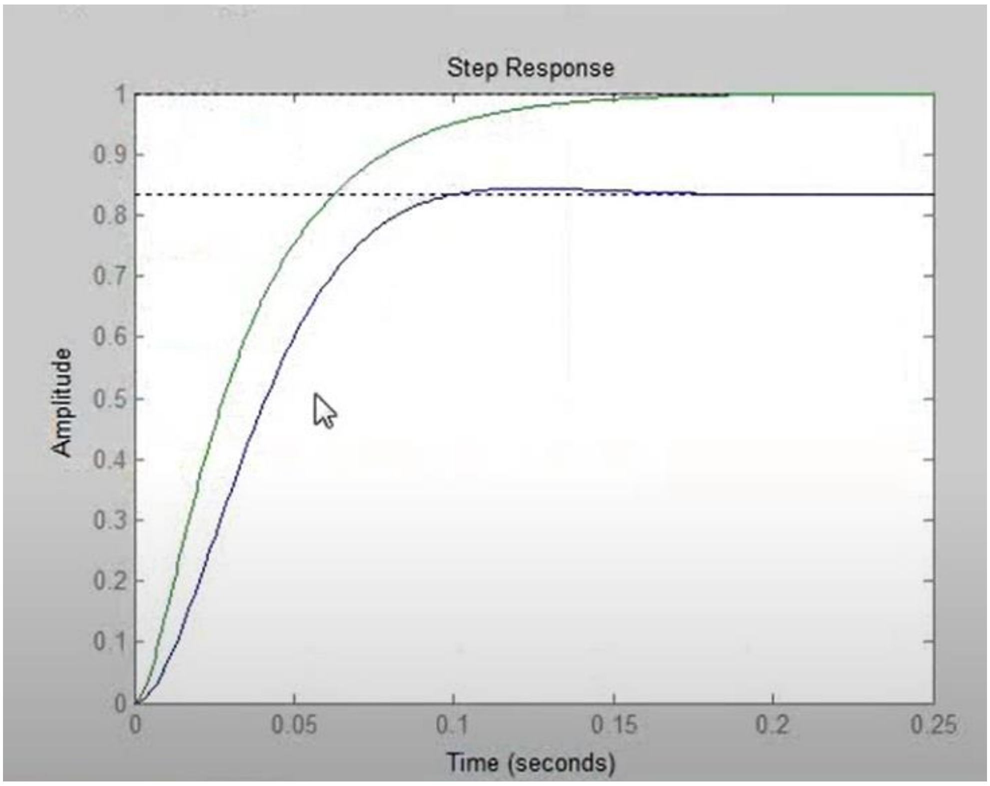
```
R = 0.0001;
kx = lqr(Ai, Bi, Q, R);
Ac = Ai - Bi * kx;
Br = [0; 0; 1];
Bc = Bi + Br;

% Closed Loop System
sys_cl = ss(Ac, Bc, Ci, D);
hold on;
step(sys_cl);
```



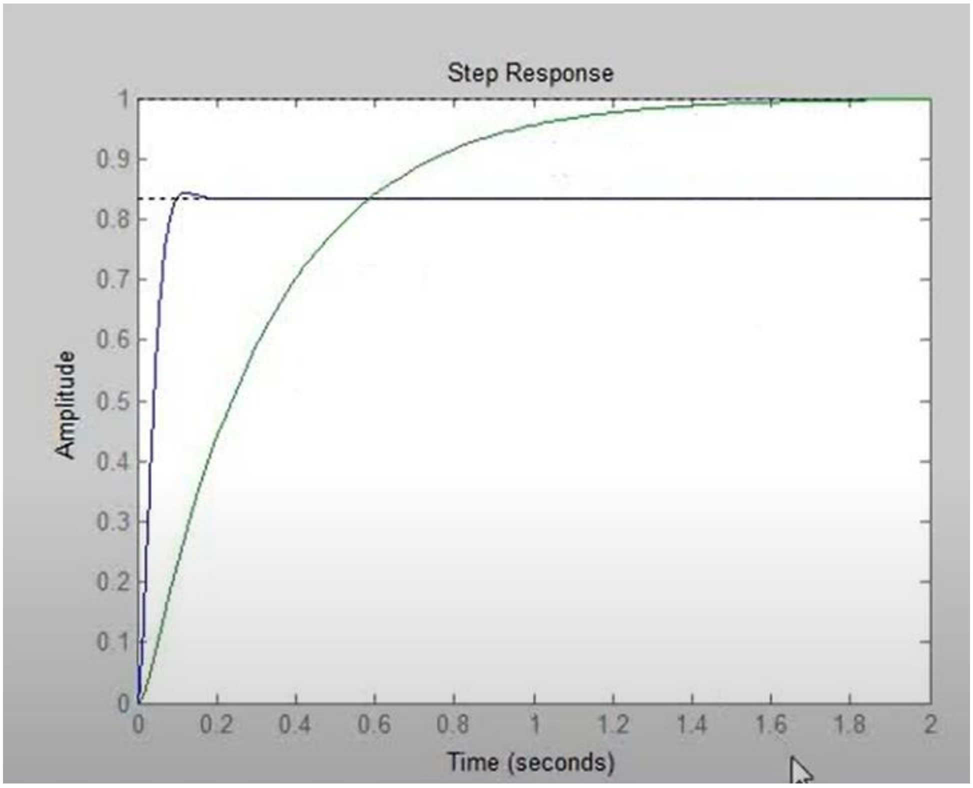
(pcl -15, -14, -6)



(pcl -150, -140, -60)



(q is same as in the code)

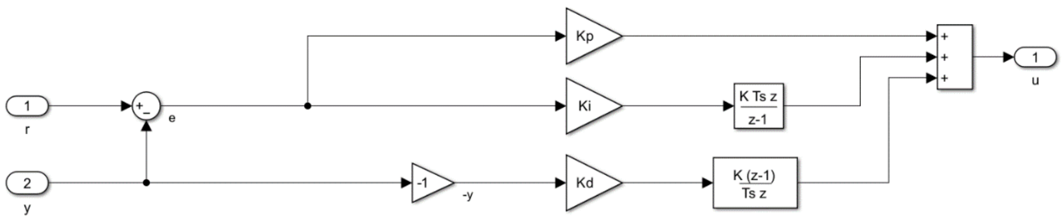


(decreasing q)

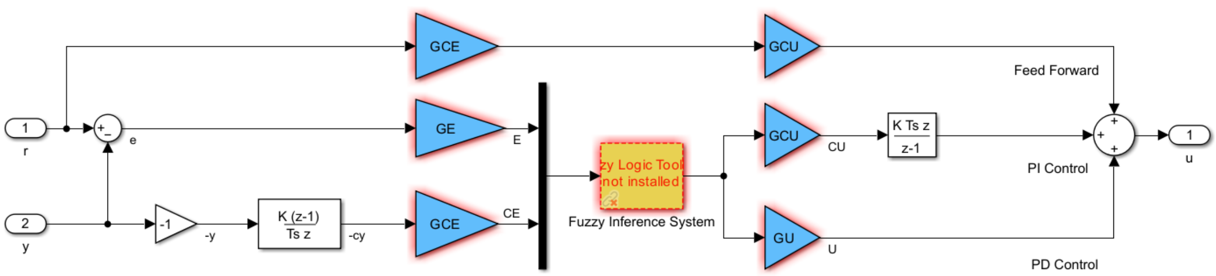
```
clc; clear; close all;

% Fungsi transfer Plant
Ts = 0.01;
J = 0.01;
b = 0.1;
Ke = 0.01;
Kt = 0.01;
R = 1;
L = 0.5;
syms s;
K = Ke;
num = K;
den = sym2poly((J*s+b)*(L*s+R)+K^2);
sys = tf(num,den);
Plant = c2d(sys,Ts,'zoh');
figure;
step(Plant);
title('Step Response');

open_system('SimFuzzyPID');
open_system('SimFuzzyPID/Fuzzy PID');

% Mendesain kendali PID konvensional
open_system('SimFuzzyPID/PID');

C0 = pid(1,1,1,'Ts',Ts,'IF','B','DF','B'); % PID structure
C = pidtune(Plant,C0); % design PID
[Kp, Ki, Kd] = piddata(C); % Parameter PID

% Asumsikan sinyal referensi bernilai 1, sehingga max. error |e|=1
% Rentang input |E| adalah [-10 10], sehingga atur |GE| = 10.

GE = 100;
GCE = GE*(Kp-sqrt(Kp^2-4*Ki*Kd))/2/Ki; % Kp = GCU * GCE + GU * GE
GCU = Ki/GE; % Ki = GCU * GE
GU = Kd/GCE; % Kd = GU * GCE

% Fuzzy inference system Sugeno:
FIS = sugfis;  % Yeni Sugeno tipi fuzzy inference sistemi oluştur

% Fungsi keanggotaan input error |E|:
FIS = addInput(FIS,[-100 100],'Name','E');
FIS = addMF(FIS,'E','gaussmf',[70 -100],'Name','Negative');
FIS = addMF(FIS,'E','gaussmf',[70 100],'Name','Positive');

% Fungsi keanggotaan input perubahan error |CE|:
FIS = addInput(FIS,[-100 100],'Name','CE');
FIS = addMF(FIS,'CE','gaussmf',[70 -100],'Name','Negative');
FIS = addMF(FIS,'CE','gaussmf',[70 100],'Name','Positive');

% Fungsi keanggotaan output |u|:
FIS = addOutput(FIS,[-200 200],'Name','u');
FIS = addMF(FIS,'u','constant',-200,'Name','Min');
```

```
FIS = addMF(FIS,'u','constant',0,'Name','Zero');
FIS = addMF(FIS,'u','constant',200,'Name','Max');

% Aturan Fuzzy
ruleList = [1 1 1 1 1;  % If |E| is Negative and |CE| is Negative then |u|
is -200 (MIN)
            1 2 2 1 1;  % If |E| is Negative and |CE| is Positive then |u|
is 0 (ZERO)
            2 1 2 1 1;  % If |E| is Positive and |CE| is Negative then |u|
is 0 (ZERO)
            2 2 3 1 1]; % If |E| is Positive and |CE| is Positive then |u|
is 200 (MAX)
FIS = addRule(FIS, ruleList);

sim('SimFuzzyPID');
load('StepPID');
load('StepFP');
figure;
if length(StepPID) > 400 && length(StepFP) > 500
    plot(StepPID(1, 1:401), StepPID(2, 101:501));
    hold on;
    plot(StepFP(1, 1:401), StepFP(2, 101:501));
else
    plot(StepPID(1, :), StepPID(2, :));  % Tüm verileri çizdir
    hold on;
    plot(StepFP(1, :), StepFP(2, :));  % Tüm verileri çizdir
end
hold off;
title('System Response');
legend('PID', 'Fuzzy-PID');

% Load simulation data
load('StepPID');
load('StepFP');

% Ensure that the data vectors are not exceeding their bounds and are of
equal length
lenPID = size(StepPID, 2);
lenFP = size(StepFP, 2);
minLength = min(lenPID, lenFP);  % Find the minimum length to safely index
both arrays

% Adjust indices to avoid out of bounds error and ensure vector length
equality
startIndexPID = max(101, minLength - 400);  % Ensure we have enough data
points to display
endIndexPID = min(startIndexPID + 400, minLength);

startIndexFP = max(101, minLength - 400);
endIndexFP = min(startIndexFP + 400, minLength);

% Plotting
figure;
```

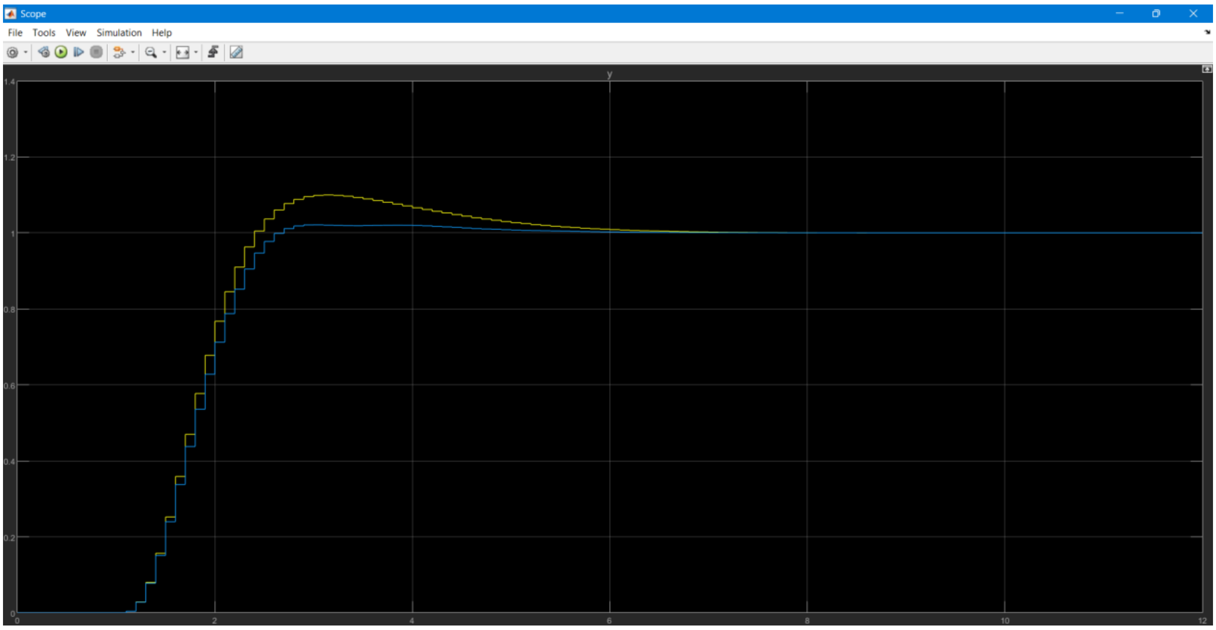
```
plot(StepPID(1, startIndexPID:endIndexPID), StepPID(2,
startIndexPID:endIndexPID));
hold on;
plot(StepFP(1, startIndexFP:endIndexFP), StepFP(2,
startIndexFP:endIndexFP));
hold off;

title('Response System');
legend('PID', 'Fuzzy-PID');
xlabel('Time');
ylabel('Response');

% Display for debugging
disp(['StepPID vector length: ', num2str(lenPID)]);
disp(['StepFP vector length: ', num2str(lenFP)]);
disp(['Using indices: ', num2str(startIndexPID), ' to ',
num2str(endIndexPID)]);
```



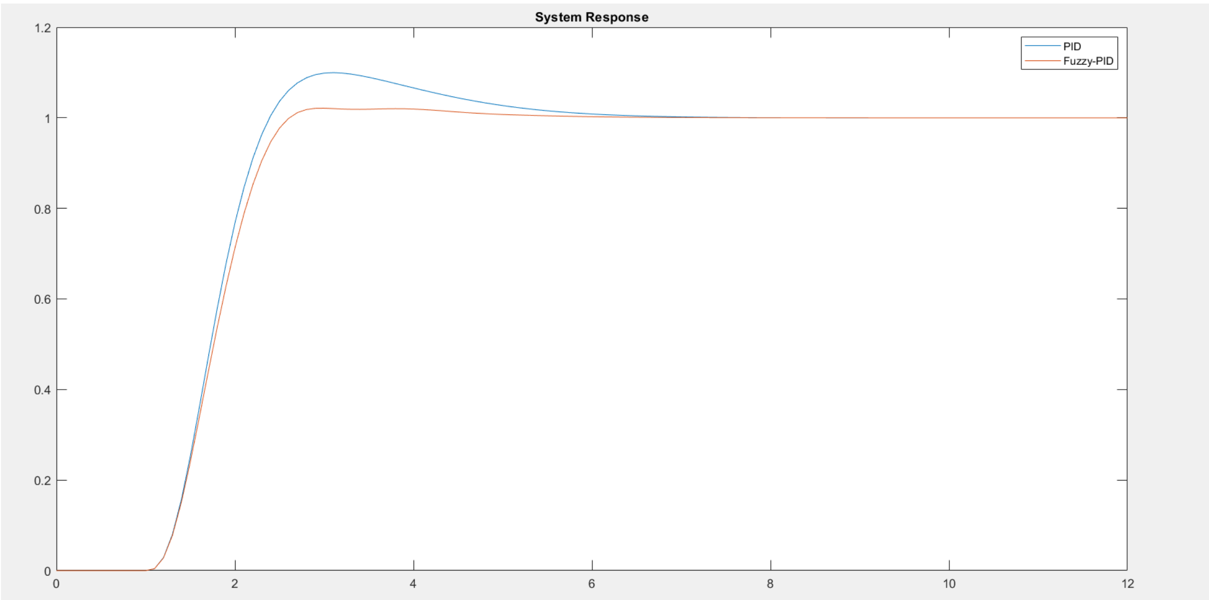
Simulink Design



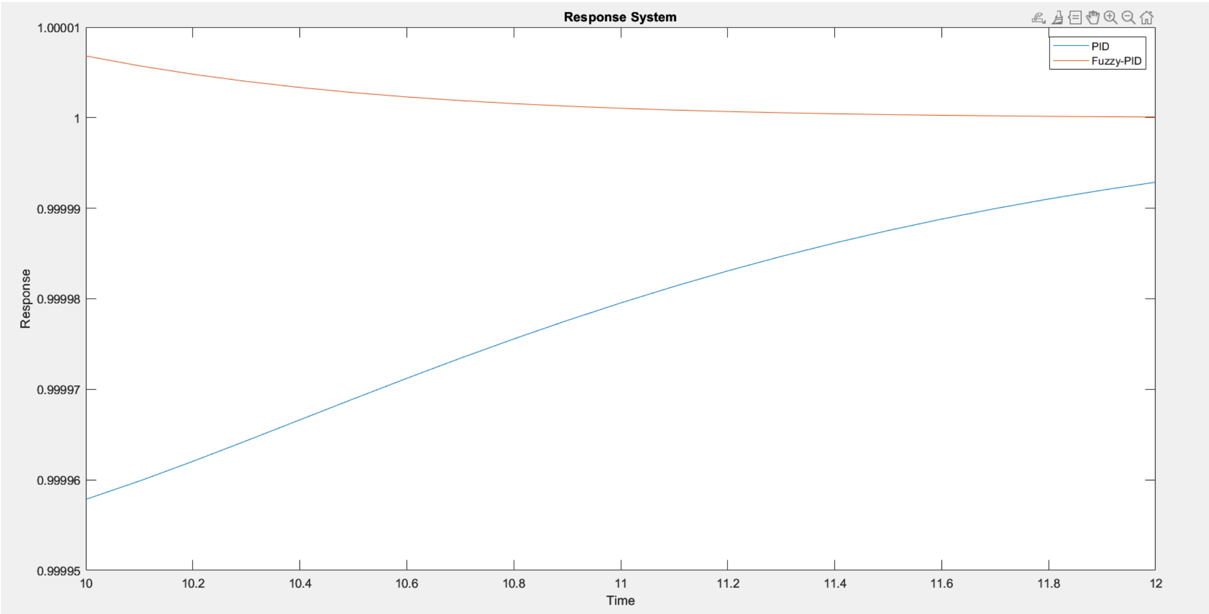
Running Simulation



Step Response



Step Response of Normal PID vs Fuzzy PID



Response Time of Normal PID vs Fuzzy PID

5. References

YTU MKT3122 Course Presentations of Assoc. Prof. Dr. Mehmet Iscan

AL-Khazraji Academy