

**PES UNIVERSITY**

**(Established under Karnataka Act No. 16 of 2013) 100 Feet Ring Road, BSK III Stage,**

**Bengaluru-560 085**

**Department of Electronics and Communication Engineering**

**Special Topic:**

**Neural Network Control of robot manipulator**

**Project Title:**

**Design of controllers for a two-link robot manipulator**

**Group member Details:**

**Name: A Anirudh Simha**

**SRN: PES1UG21EC001**

**Name: Aditya R G**

**SRN: PES1UG21EC017**

**Name: Akash Ravi Bhat**

**SRN: PES1UG21EC025**

**Faculty Name:**

**Dr. Rashmi N U**

**MATLAB CODES:**

1. **Main File**

clc;close all; clear all;

t0= 0 ; tf=10 ; T=10;

%x0= [0.1 0 0 0 ]' ; %input for PD & gravity PD controller

x0=[10,0,0,0]; %Classical PD controller input

%x0 =[0.1 0 0 0 0 0]'; %PID controller input

%x0=[0.1 ,1, 0 ,0, 0.1 ,0.1];%Adaptive controller input

%[t,x]= ode23(@fblinct,[t0 tf],x0) ;

[t,x]=ode23(@robctl,[t0 tf],x0);

%[t,x]=ode23(@robadapt,[t0 tf],x0); % Adaptive Controller

yd= sin(2\*pi\*t/T) ;

e= yd - x;

[qd,e]= robout(t,x);

plot(t,[yd,x(:,1)])

plot(t,x(:,2))

plot(t,e) %Normal plot

figure;

%plot(t,x(:,1),'-c',t,qd(:,1),'--k') %Plot for adaptive controller

legend("e1","e2")

title("Tracking error v/s Time ");

ylabel("Tracking error");

xlabel("Time");

figure;

plot(t,qd);

legend("e1","e2")

title("Joint angle v/s Time ");

ylabel("Tracking error");

xlabel("Time");

1. **Robctl code:**

function xdot= robctl(t,x) ;

% ----------------------------------------------------------

% COMPUTE CONTROL INPUT FOR ROBOT ARM

% compute desired trajectory

period= 2 ;

amp1= 0.1 ;

amp2= 0.1 ;

fact= 2\*pi/period ;

sinf= sin(fact\*t) ;

cosf= cos(fact\*t) ;

qd= [amp1\*sinf amp2\*cosf]' ;

qdp= fact\*[amp1\*cosf -amp2\*sinf]' ;

qdpp= -fact^2\*qd ;

% PD Computed-Torque control input

m1= 1 ; m2= 1 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters

kp= 100 ; kv= 20 ; % controller parameters

ki=20; %PID

% tracking errors

e= qd - [x(1) x(2)]' ;

ep= qdp - [x(3) x(4)]' ;

% computed inertia M(q) and nonlinear terms N(q,qdot)

M11= (m1 + m2)\*a1^2 + m2\*a2^2 + 2\*m2\*a1\*a2\*cos(x(2)) ;

M12= m2\*a2^2 + m2\*a1\*a2\*cos(x(2)) ;

M22= m2\*a2^2 ;

N1= -m2\*a1\*a2\*(2\*x(3)\*x(4) + x(4)^2)\*sin(x(2)) ;

N1= N1 + (m1 + m2)\*g\*a1\*cos(x(1)) + m2\*g\*a2\*cos(x(1) + x(2));

N2= m2\*a1\*a2\*x(3)^2\*sin(x(2)) + m2\*g\*a2\*cos(x(1) + x(2)) ;

% PD CT control torques

s1= qdpp(1) + kv\*ep(1) + kp\*e(1) ;

s2= qdpp(2) + kv\*ep(2) + kp\*e(2) ;

tau1= M11\*s1 + M12\*s2 + N1 ;

tau2= M12\*s1 + M22\*s2 + N1 ;

% PID CT control torques

%s1= qdpp(1) + kv\*ep(1) + kp\*e(1) + ki\*x(5) ;

%s2= qdpp(2) + kv\*ep(2) + kp\*e(2) + ki\*x(6) ;

%tau1= M11\*s1 + M12\*s2 + N1 ;

%tau2= M12\*s1 + M22\*s2 + N2 ;

%xdot(5)= e(1);

%xdot(6)= e(2);

% Gravity PD Controller

% % computed gravity terms

G1= (m1 + m2)\*g\*a1\*cos(x(1)) + m2\*g\*a2\*cos(x(1) + x(2));

G2= m2\*g\*a2\*cos(x(1) + x(2)) ;

% PD CT control torques

s1= kv\*ep(1) + kp\*e(1) ;

s2= kv\*ep(2) + kp\*e(2) ;

tau1= s1 + G1;

tau2= s2 + G2;

%Classical Joint Controller PD

tau1=kv\*ep(1)+kp\*e(1);

tau2=kv\*ep(2)+kp\*e(2);

tau=[tau1;tau2];

% ----------------------------------------------------------

% ROBOT ARM DYNAMICS

m1= 1 ; m2= 1 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters

% inertia M(q) and nonlinear terms N(q,qdot)

M11= (m1 + m2)\*a1^2 + m2\*a2^2 + 2\*m2\*a1\*a2\*cos(x(2)) ;

M12= m2\*a2^2 + m2\*a1\*a2\*cos(x(2)) ;

M22= m2\*a2^2 ;

N1= -m2\*a1\*a2\*(2\*x(3)\*x(4) + x(4)^2)\*sin(x(2)) ;

N1= N1 + (m1 + m2)\*g\*a1\*cos(x(1)) + m2\*g\*a2\*cos(x(1) + x(2));

N2= m2\*a1\*a2\*x(3)^2\*sin(x(2)) + m2\*g\*a2\*cos(x(1) + x(2)) ;

% Inversion of M(q) (for large values of n, use least-squares)

det= M11\*M22 - M12\*M12 ;

MI11= M22/det ;

MI12= -M12/det ;

MI22= M11/det ;

% state equations

xdot(1)= x(3) ;

xdot(2)= x(4) ;

xdot(3)= MI11\*(-N1 + tau1) + MI12\*(-N2 + tau2) ;

xdot(4)= MI12\*(-N1 + tau1) + MI22\*(-N2 + tau2) ;

xdot=xdot';

end

1. **Robout code:**

% file robout.m

function [qd,e]= robout(t,x)

% compute desired trajectory

period= 2 ; amp1= 0.1 ; amp2= 0.1 ;

fact= 2\*pi/period ;

sinf= sin(fact\*t) ;

cosf= cos(fact\*t) ;

qd= [amp1\*sinf amp2\*cosf] ;

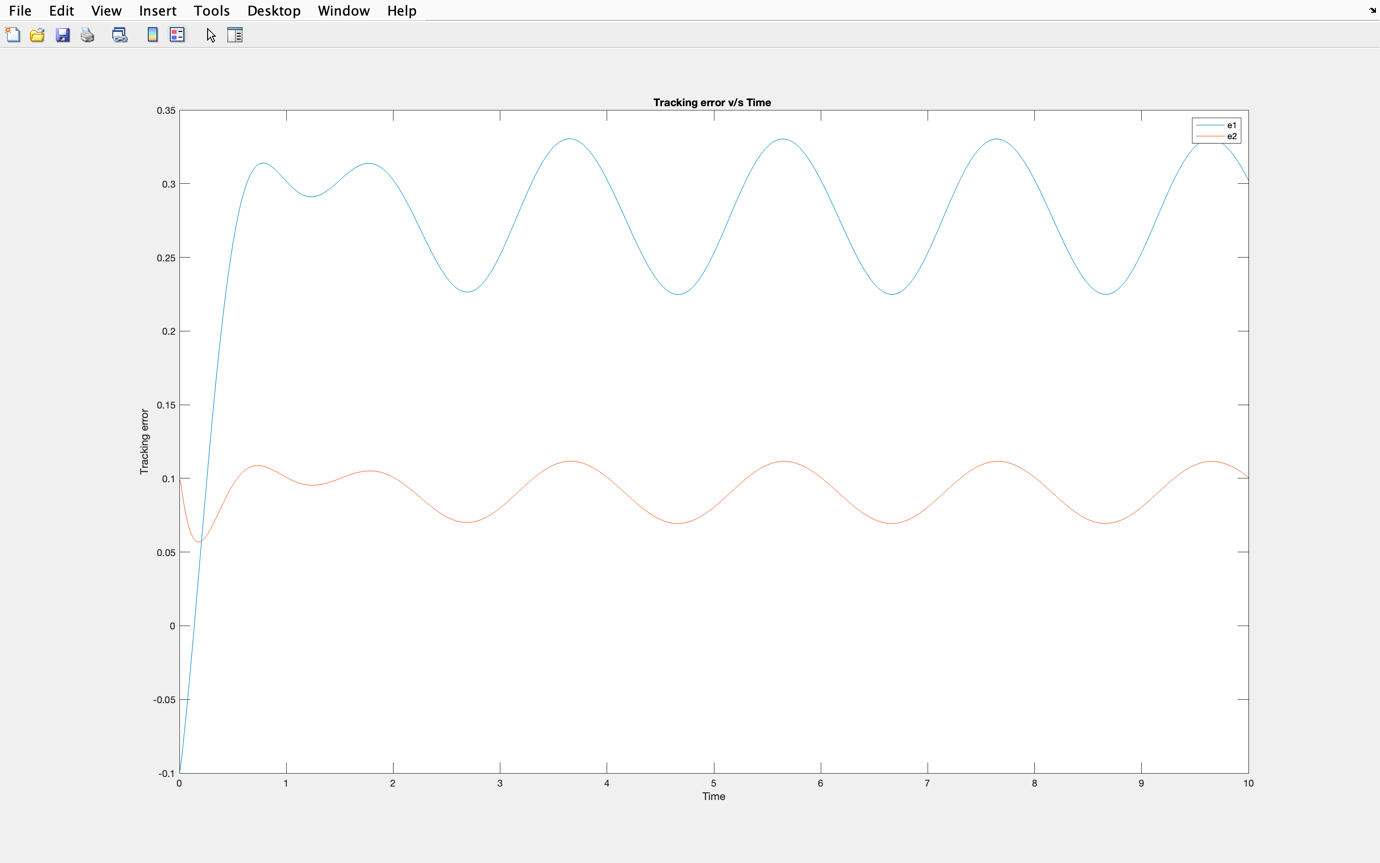
% tracking errors

e= qd - x(:,1:2) ;

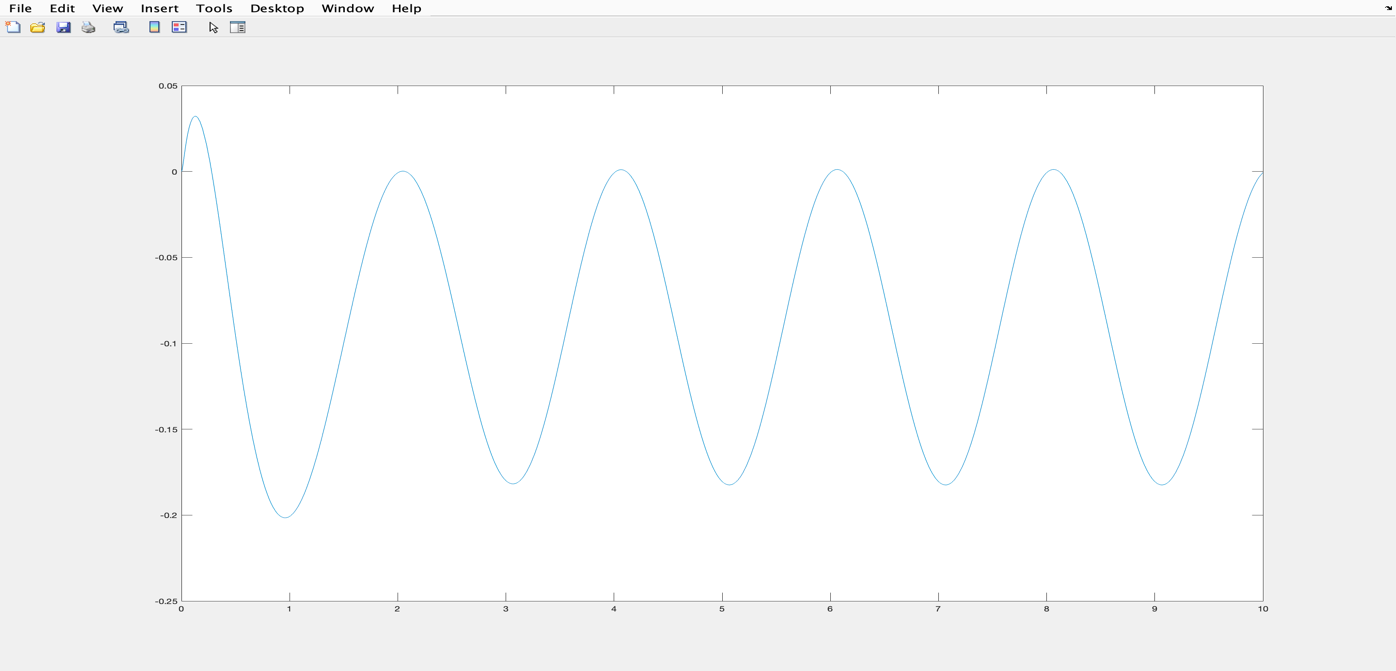
end

**Implementation of PID controller**

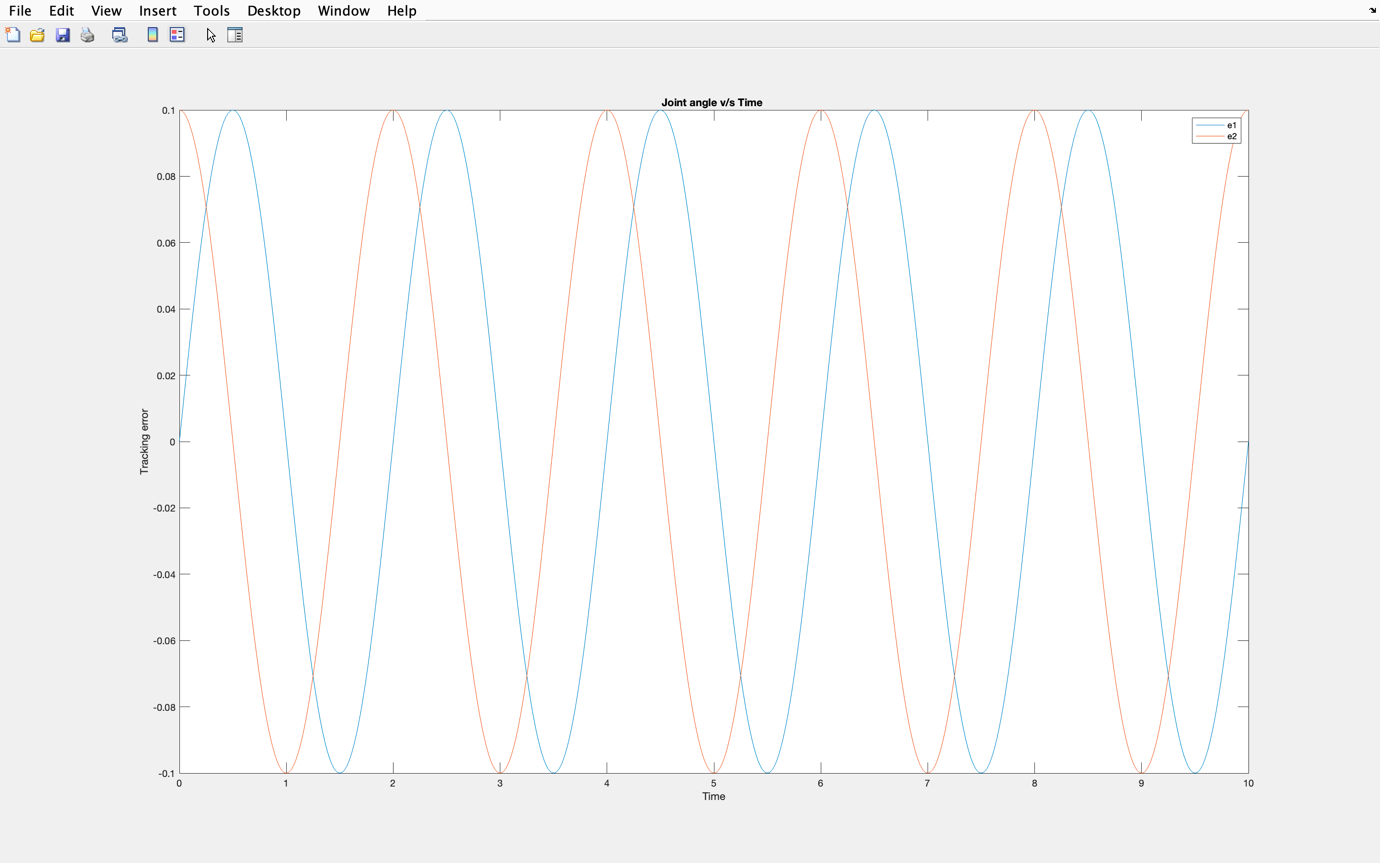
Tracking error graph:

****

Output graph:

****

Joint angle graph:

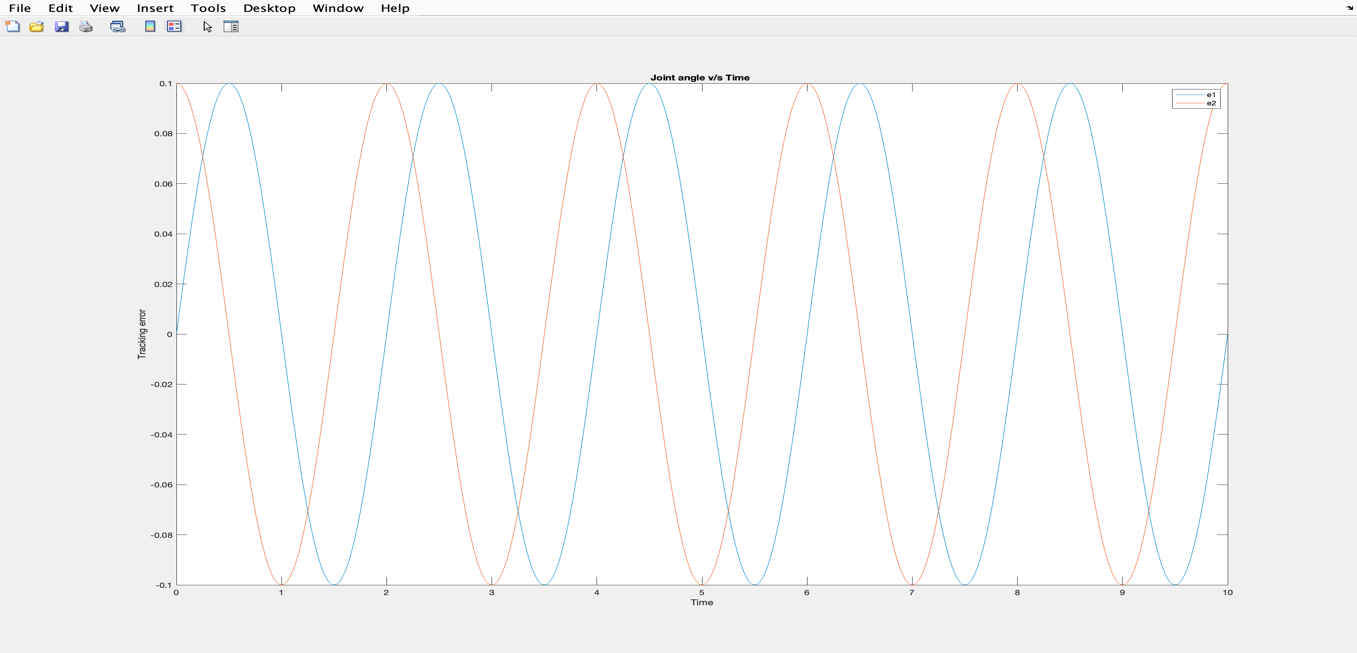
****

Conclusion:

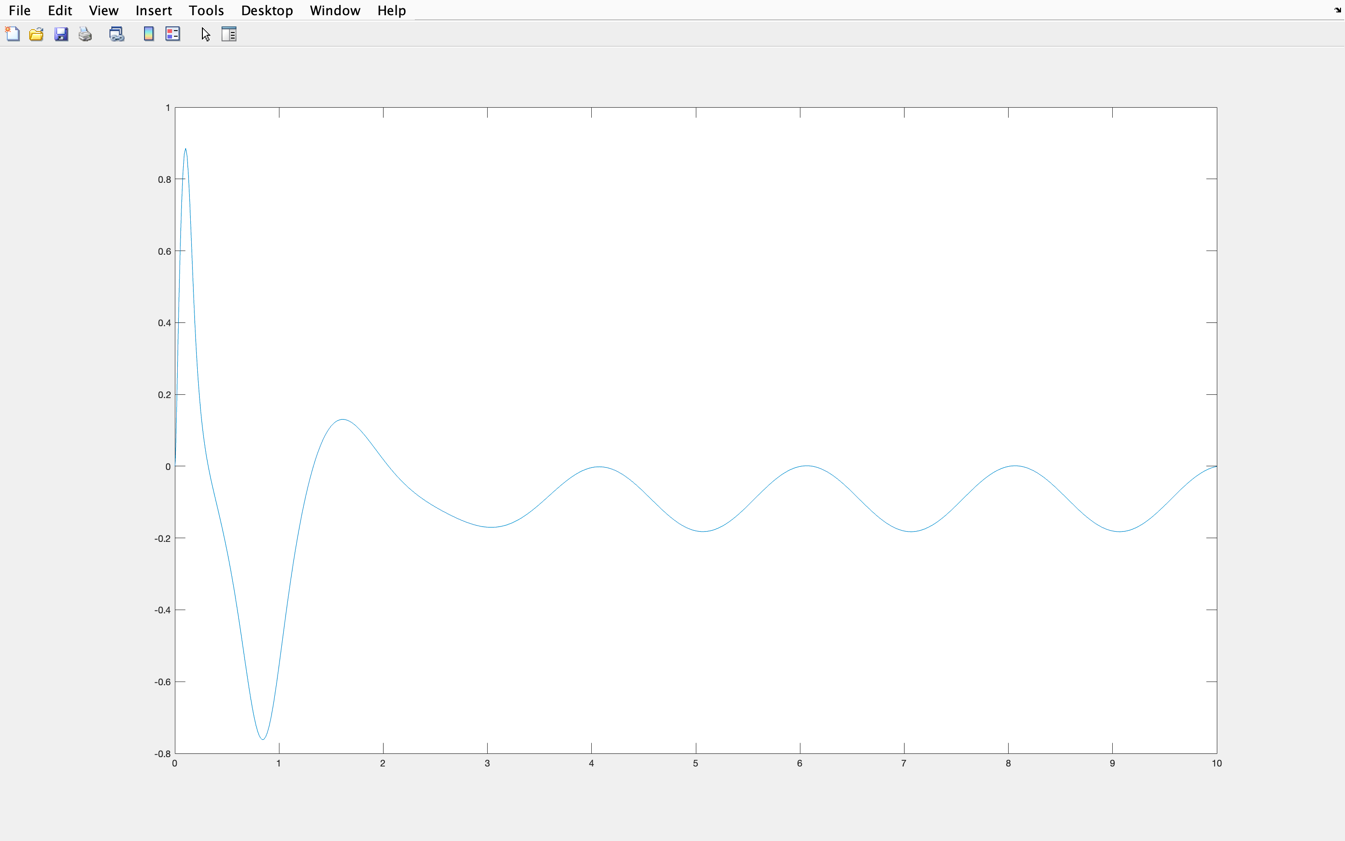
The PID controller was successfully implemented and outputs obtained were verified

**Implementation of PD controller**

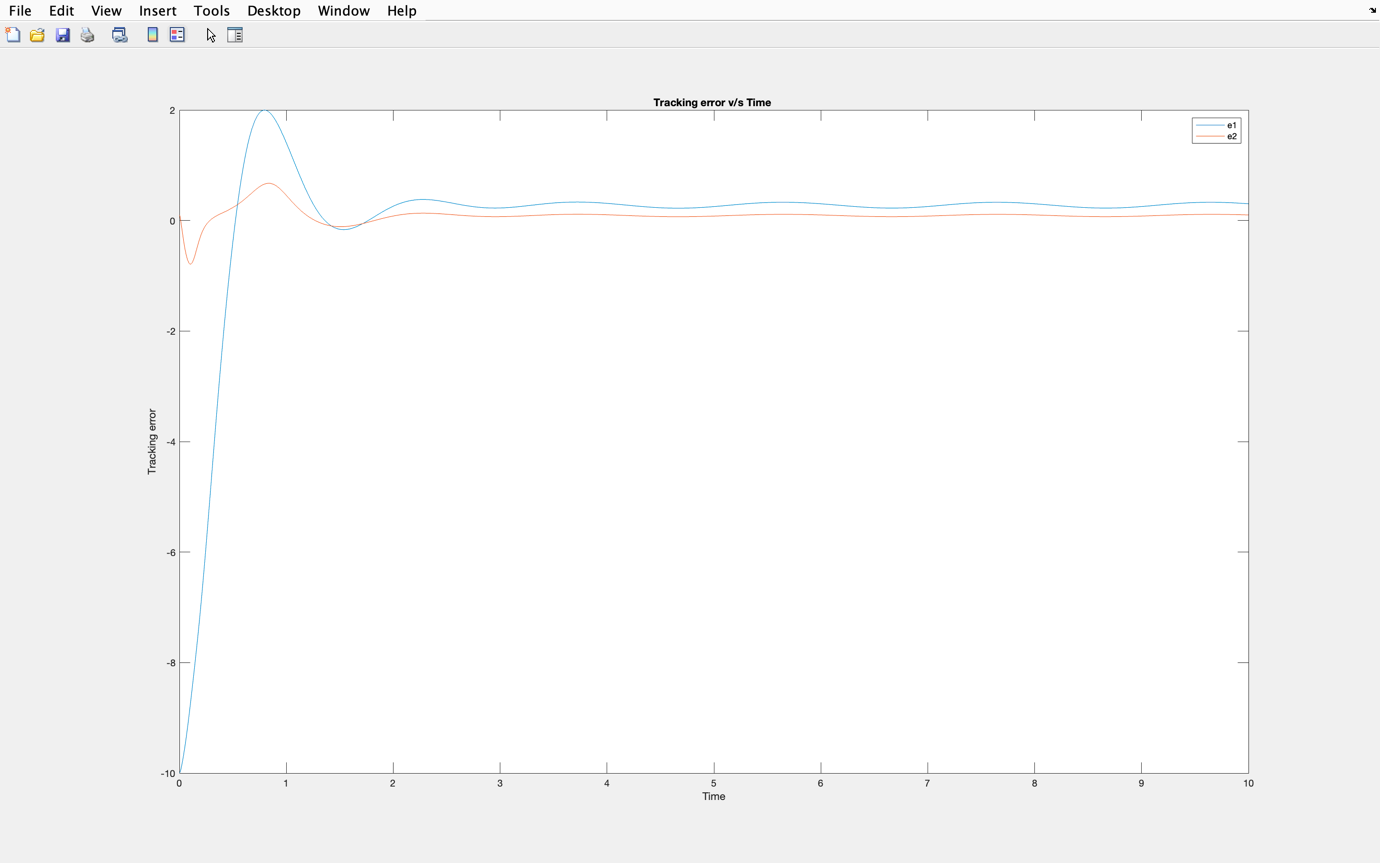
Joint angle graph:

****

Output graph:

****

Tracking error graph:

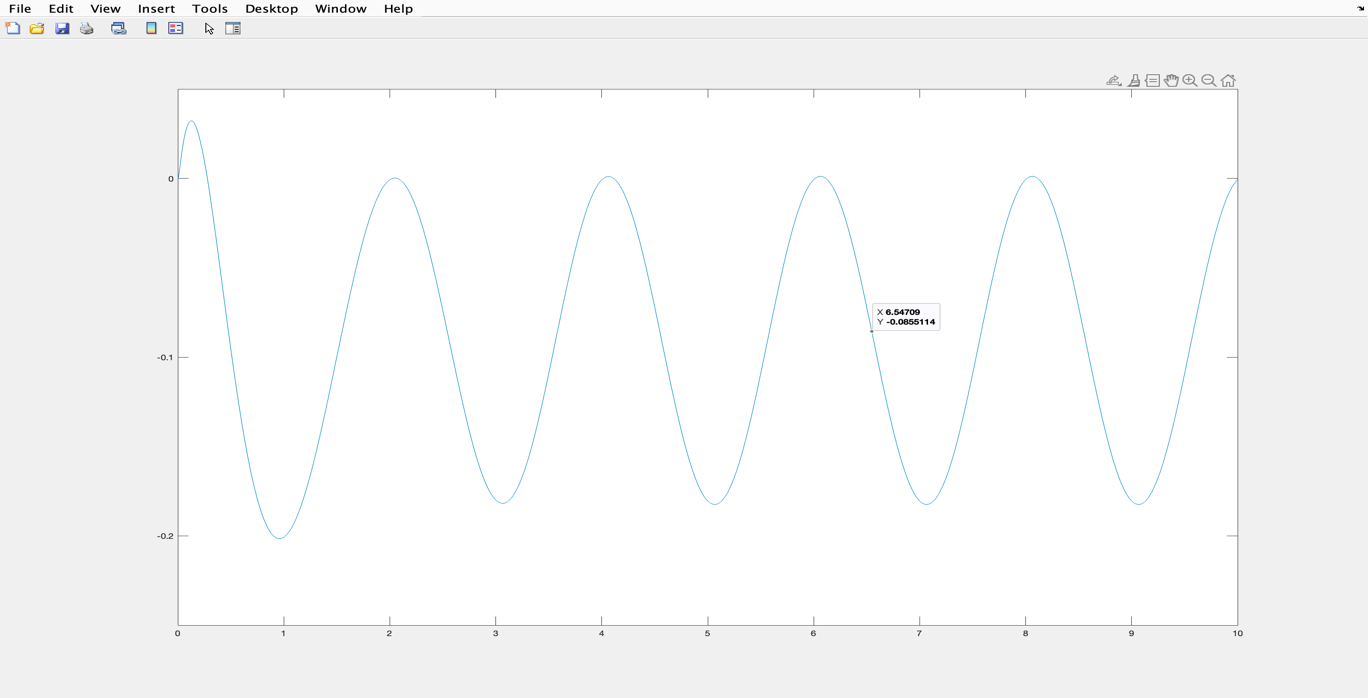
****

Conclusion:

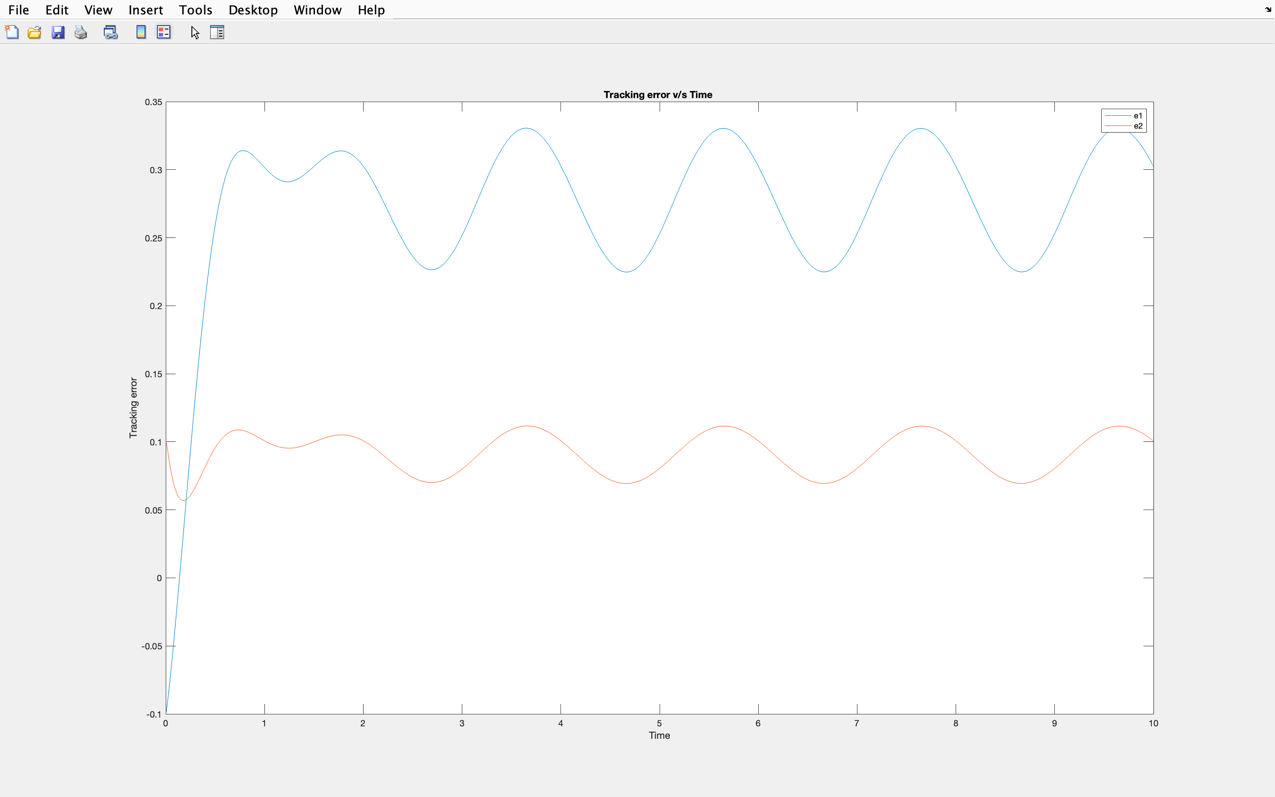
The PD controller was successfully implemented and outputs obtained were verified

**Implementation of PD Gravity controller**

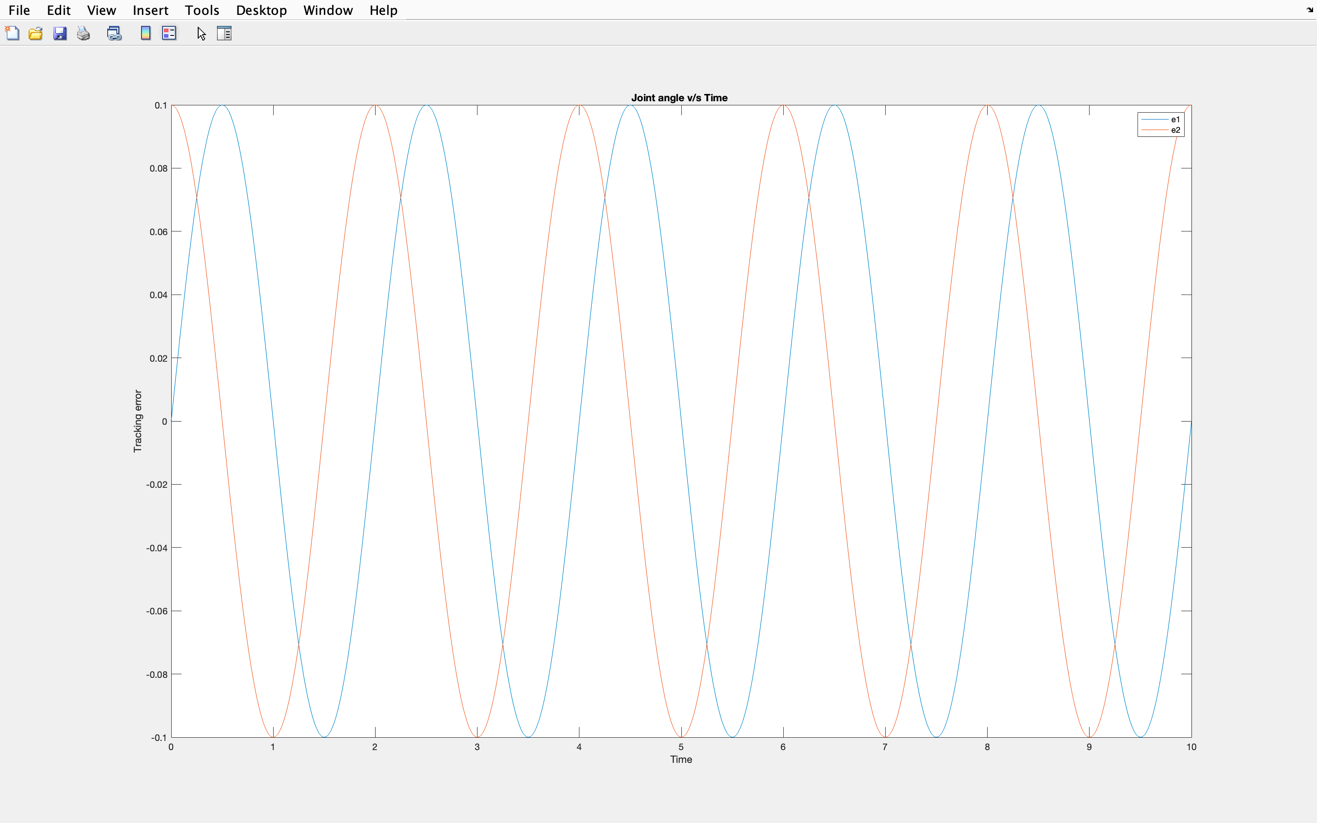
Output graph:

****

Tracking error graph:

****

Joint angle graph:

****

Conclusion:

The PD GRAVITY controller was successfully implemented and outputs obtained were verified

1. **Robadapt code(for adaptive controller)**

function xdot= robadapt(t,x) ;

% compute desired trajectory as in Fig. 3.3.2

% Use period= 2\*pi ; amp1= 1 ; amp2= 1 ;

% Adaptive control input

m1= 0.8 ; m2= 2.3 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters

Kv= 20\*eye(2) ; lam= 5\*eye(2); gam= 10\*eye(2) ; % controller parameters

period= 2 ;

fact= 2\*pi/period ;

sinf= sin(fact\*t) ;

cosf= cos(fact\*t) ;

qd= [a1\*sinf a2\*cosf]' ;

qdp= fact\*[a1\*cosf -a2\*sinf]' ;

qdpp= -fact^2\*qd ;

% tracking errors

e= qd - [x(1) x(2)]' ;

ep= qdp - [x(3) x(4)]' ;

r= ep + lam\*e ;

% compute regression matrix

f= qdpp + lam\*ep ;

W(1,1) = a1^2\*f(1) + a1\*g\*cos(x(1)) ;

W(1,2) = (a2^2 + 2\*a1\*a2\*cos(x(2)) + a1^2)\*f(1)

+ (a2^2 + a1\*a2\*cos(x(2)))\*f(2)

- a1\*a2\*x(4)\*(qdp(1)\*sin(x(2)) + lam(1,1)\*e(1))

- a1\*a2\*(x(3)+ x(4))\*(qdp(2)+sin(x(2)) + lam(2,2)\*e(2))

+ a2\*g\*cos(x(1) + x(2)) + a1\*g\*cos(x(1)) ;

W(2,1) = 0 ;

W(2,2) = (a2^2 + a1\*a2\*cos(x(2)))\*f(1) + a2^2\*f(2)

+ a1\*a2\*(qdp(1) + lam(1,1)\*e(1))\*sin(x(2))

+ a2\*g\*cos(x(1) + x(2)) ;

% control torques. Parameter estimates are [x(5) x(6)]’

tau= Kv\*r + W\*[x(5) x(6)]';

tau1= tau(1) ; tau2= tau(2) ;

% parameter updates

phidot= gam\*W'\*r ;

xdot(5)= phidot(1) ;

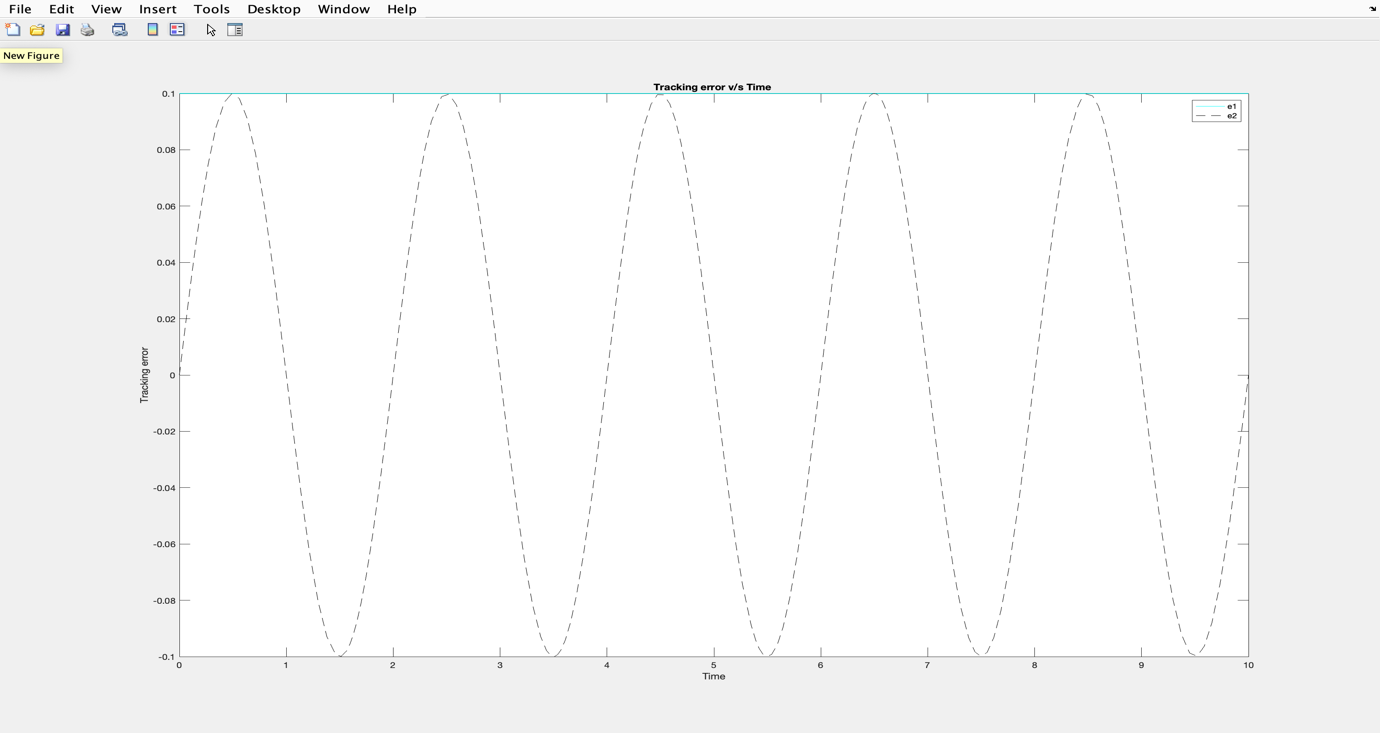
xdot(6)= phidot(2) ;

xdot=xdot';

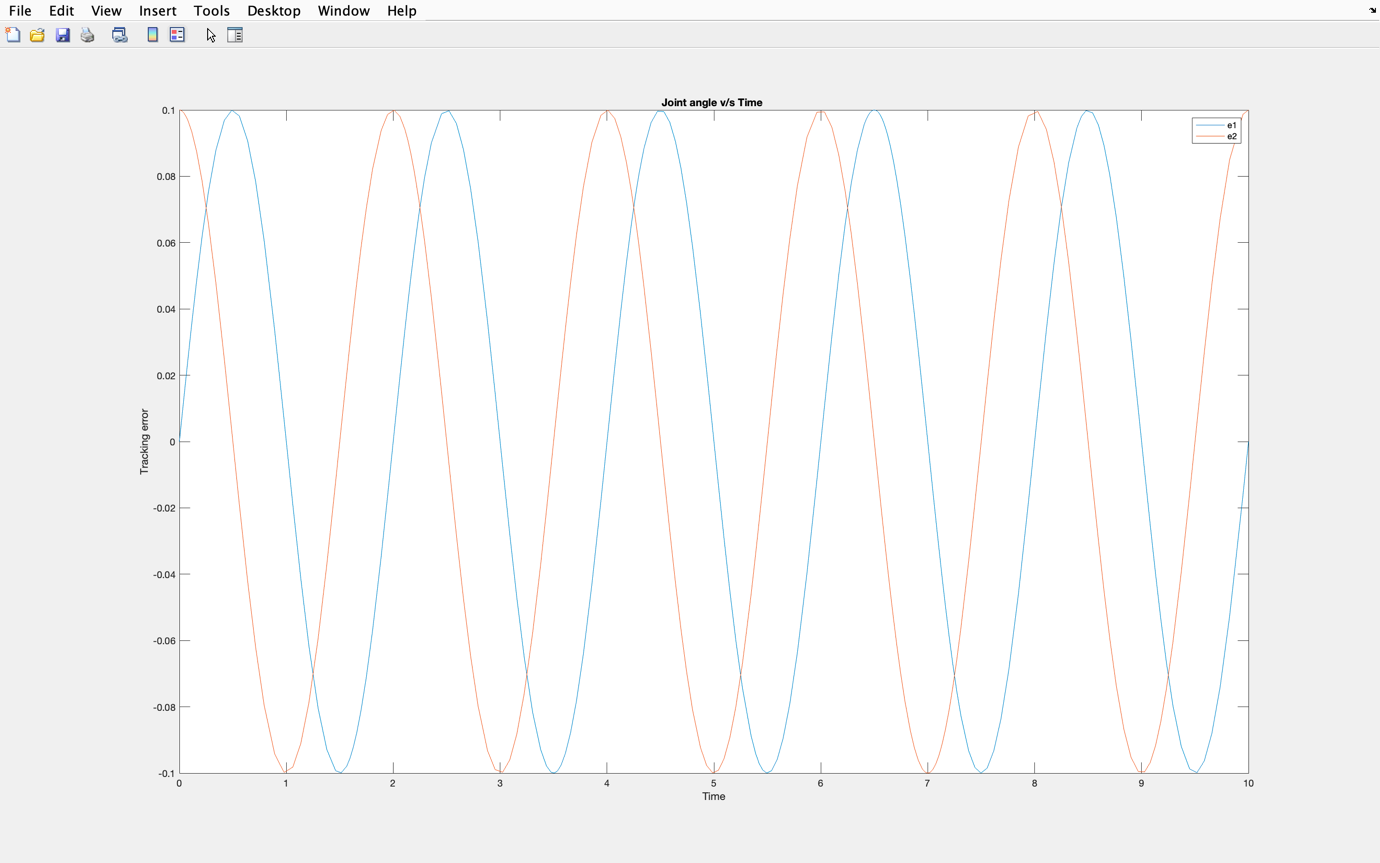
end

**Implementation of adaptive controller**

Tracking error graph:



Joint angle graph:



Conclusion:

The ADAPTIVE controller was successfully implemented and outputs obtained were verified

1. **Robctlflnn code(for FLNN controller)**

function xdot = robctlflnn(t, x)

% COMPUTE CONTROL INPUT FOR ROBOT ARM

% compute desired trajectory

period = 2; amp1 = 1; amp2 = 1;

fact = 2 \* pi / period;

sinf = sin(fact \* t);

cosf = cos(fact \* t);

qd = [amp1 \* sinf amp2 \* cosf]';

qdp = fact \* [amp1 \* cosf -amp2 \* sinf]';

qdpp = -fact^2 \* qd;

% Tracking errors

e = qd - [x(1) x(2)]';

ep = qdp - [x(3) x(4)]';

% FLNN Controller parameters

w1 = randn(); % Weight for error e1

w2 = randn(); % Weight for error e2

b = randn(); % Bias

% FLNN Controller output

flnn\_output = w1 \* e(1) + w2 \* e(2) + b;

% computed gravity terms

m1 = 0.8; m2 = 2.3; a1 = 1; a2 = 1; g = 9.8; % arm parameters

G1 = (m1 + m2) \* g \* a1 \* cos(x(1)) + m2 \* g \* a2 \* cos(x(1) + x(2));

G2 = m2 \* g \* a2 \* cos(x(1) + x(2));

% FLNN control torques

tau1 = flnn\_output + G1;

tau2 = flnn\_output + G2;

% ROBOT ARM DYNAMICS

% inertia M(q) and nonlinear terms N(q,qdot)

M11 = (m1 + m2) \* a1^2 + m2 \* a2^2 + 2 \* m2 \* a1 \* a2 \* cos(x(2));

M12 = m2 \* a2^2 + m2 \* a1 \* a2 \* cos(x(2));

M22 = m2 \* a2^2;

N1 = -m2 \* a1 \* a2 \* (2 \* x(3) \* x(4) + x(4)^2) \* sin(x(2));

N1 = N1 + (m1 + m2) \* g \* a1 \* cos(x(1)) + m2 \* g \* a2 \* cos(x(1) + x(2));

N2 = m2 \* a1 \* a2 \* x(3)^2 \* sin(x(2)) + m2 \* g \* a2 \* cos(x(1) + x(2));

% Inversion of M(q) (for large values of n, use least-squares)

det = M11 \* M22 - M12 \* M12;

MI11 = M22 / det;

MI12 = -M12 / det;

MI22 = M11 / det;

% Update the state equations

xdot(1) = x(3);

xdot(2) = x(4);

xdot(3) = MI11 \* (-N1 + tau1) + MI12 \* (-N2 + tau2);

xdot(4) = MI12 \* (-N1 + tau1) + MI22 \* (-N2 + tau2);

xdot(5) = e(1);

xdot(6) = e(2);

xdot = xdot'; end

1. **Main code for FLNN controller**

clc;

clear all;

close all;

t0= 0; tf= 10;

x0= [.1 0 0 0 0 0]';

[t,x]= ode23('robctlflnn',[t0 tf],x0);

period= 2 ; amp1= 1 ; amp2= 1 ;

fact= 2\*pi/period ;

sinf= sin(fact\*t) ;

cosf= cos(fact\*t) ;

qd= [amp1\*sinf amp2\*cosf] ;

% tracking errors

e= qd - x(:,1:2) ;

plot(t,e)

legend("e1","e2")

title("Tracking error v/s Time ");

ylabel("Tracking error");

xlabel("Time");

figure;

plot(t,qd)

legend("e1","e2")

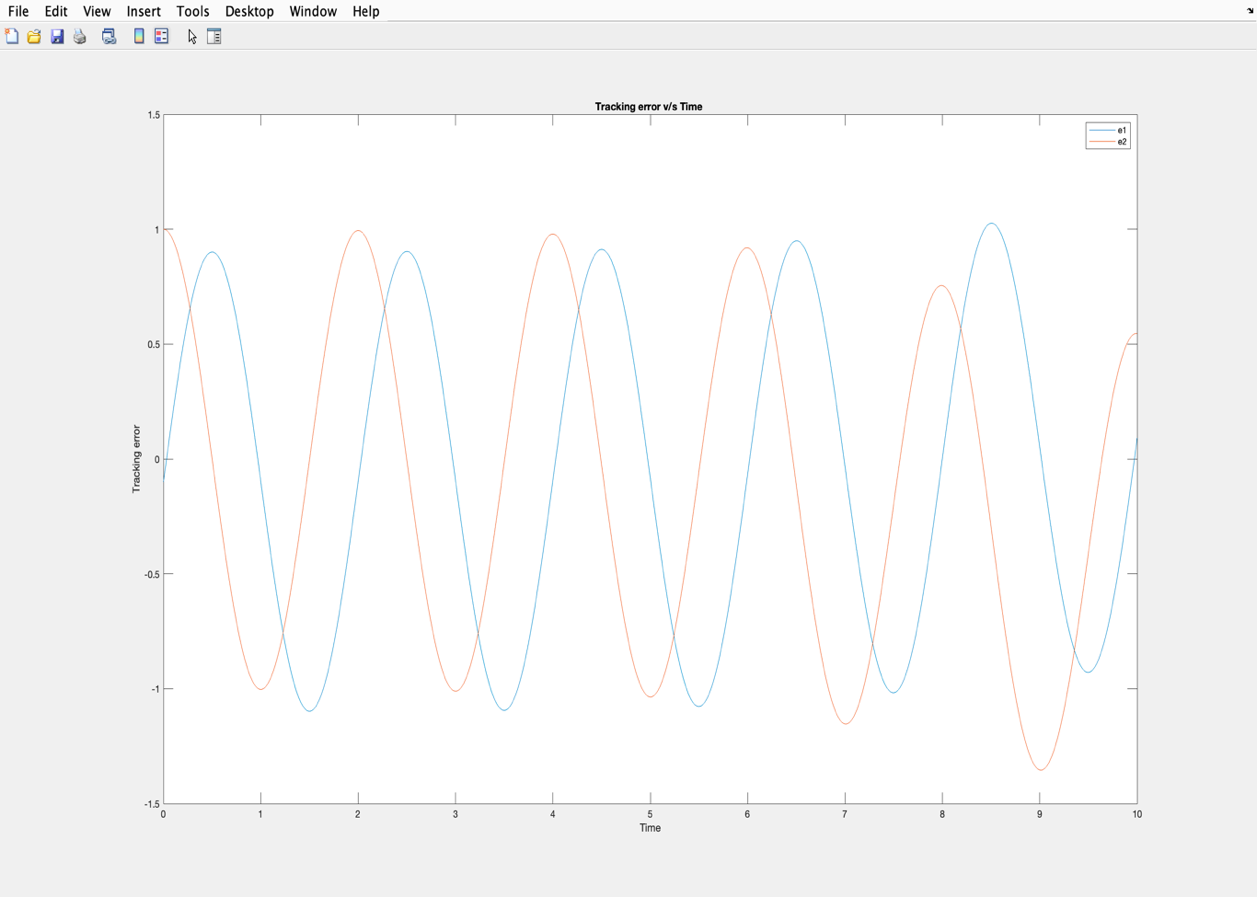
title("Joint angle v/s Time ");

ylabel("Joint angle");

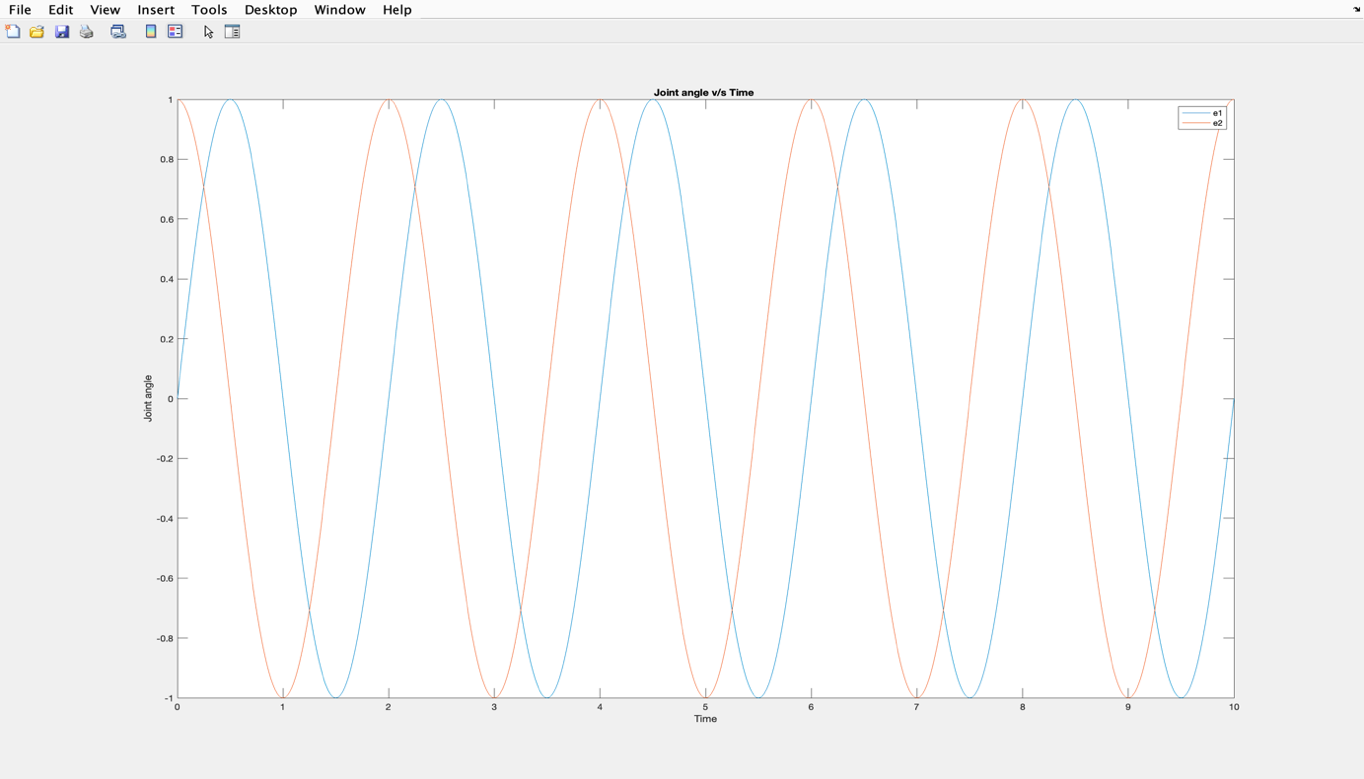
xlabel("Time");

**Implementation of FLNN controller**

Tracking error graph:

****

Joint angle graph:

****

Conclusion:

The FLNN controller was successfully implemented and outputs obtained were verified

1. **Fblinct code**

% MATLAB file for closed-loop system simulation

function xdot= fblinct(t,x)

T=10;

% Computation of the desired trajectory

yD= sin(2\*pi\*t/T) ;

yDdot= (2\*pi/T) \* cos(2\*pi\*t/T) ;

yDddot=-(2\*pi/T)^2 \* sin(2\*pi\*t/T) ;

% Computation of the control input

kp= 100 ;

kd= 14.14 ;

f = sin(x(1)) + x(2)\*x(3) + x(1)\*x(2)^2 ;

g=1+ x(1)^2 ;

y = x(1) ;

ydot= x(1)\*x(2) + x(3) ;

e=yD-y;

edot= yDdot- ydot ;

u = (-f + yDddot + kd\*edot + kp\*e)/g;

% Plant dynamics

xdot(1) = x(1)\*x(2) + x(3) ;

xdot(2) =-2\*x(2) + x(1)\*u ;

xdot(3) = sin(x(1)) + 2\*x(1)\*x(2)+u;

xdot=xdot';

end

**8.Fblinct script file**

clear all; clc;close all;

t0= 0 ; tf=10 ;

x0= [0.1 0 0 0 0 0]' ;

T=10;

[t,x] = ode23('fblinct', [t0 tf], x0);

y=x;

yd= sin(2\*pi\*t/T) ;

e= yd - y ;

figure;

plot(t,[yd,x(:,1)])

figure;

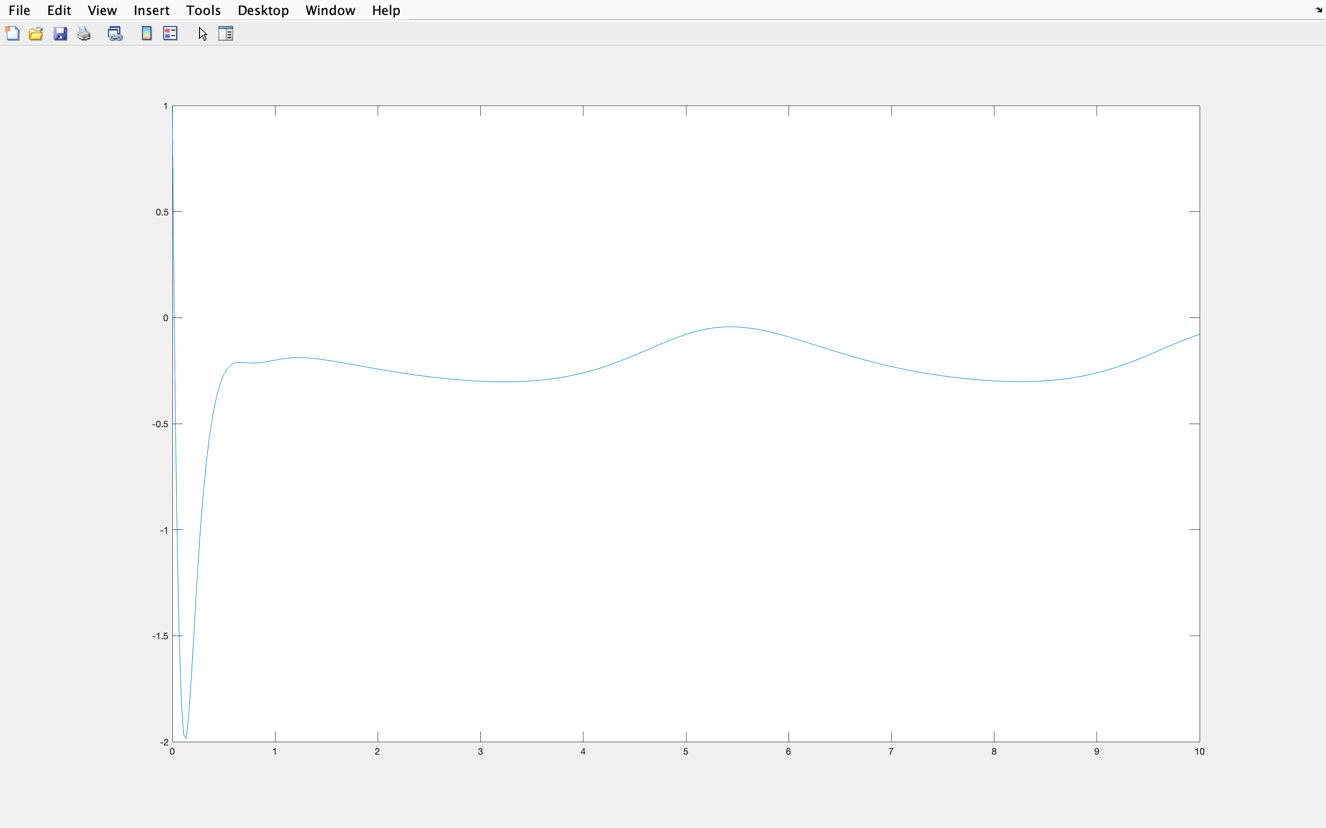
plot(t,e)

figure;

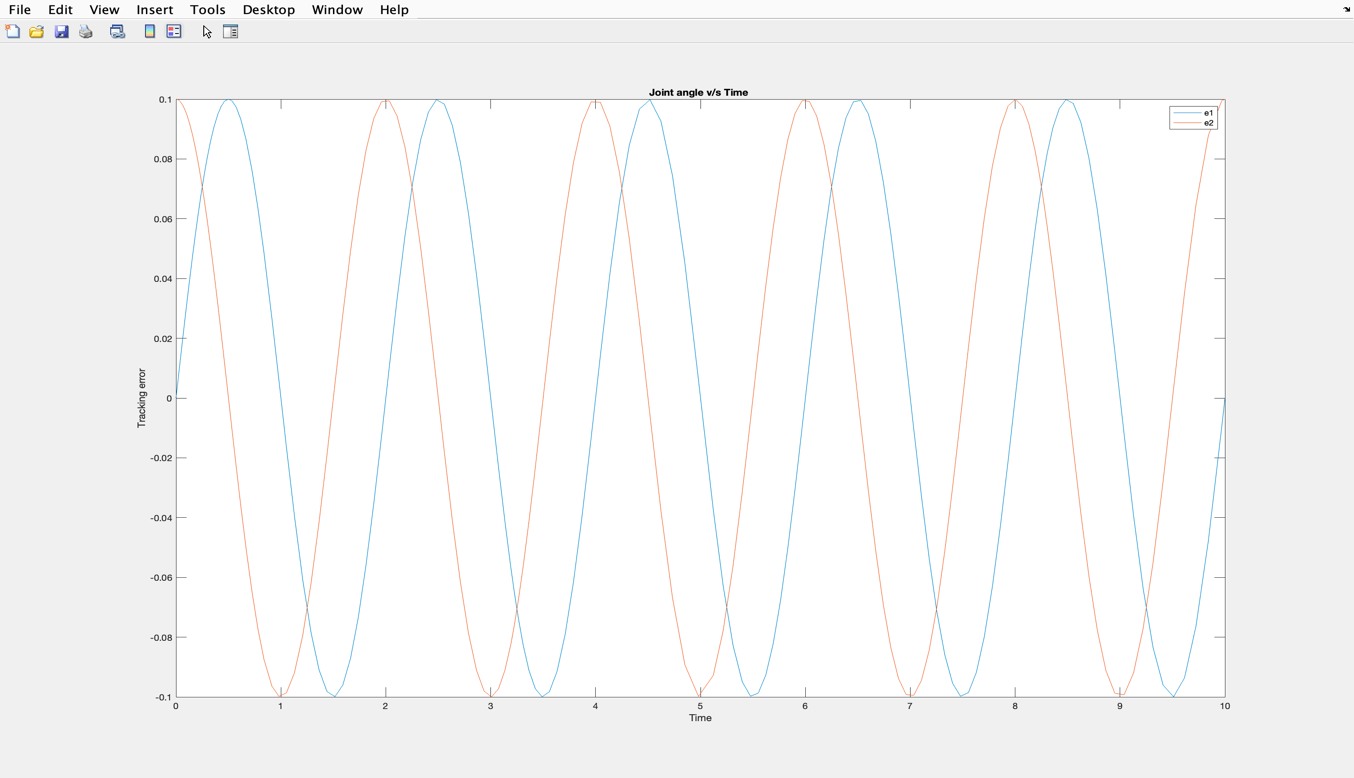
plot(t,x(:,2))

**Implementation of FBLINCT controller:**

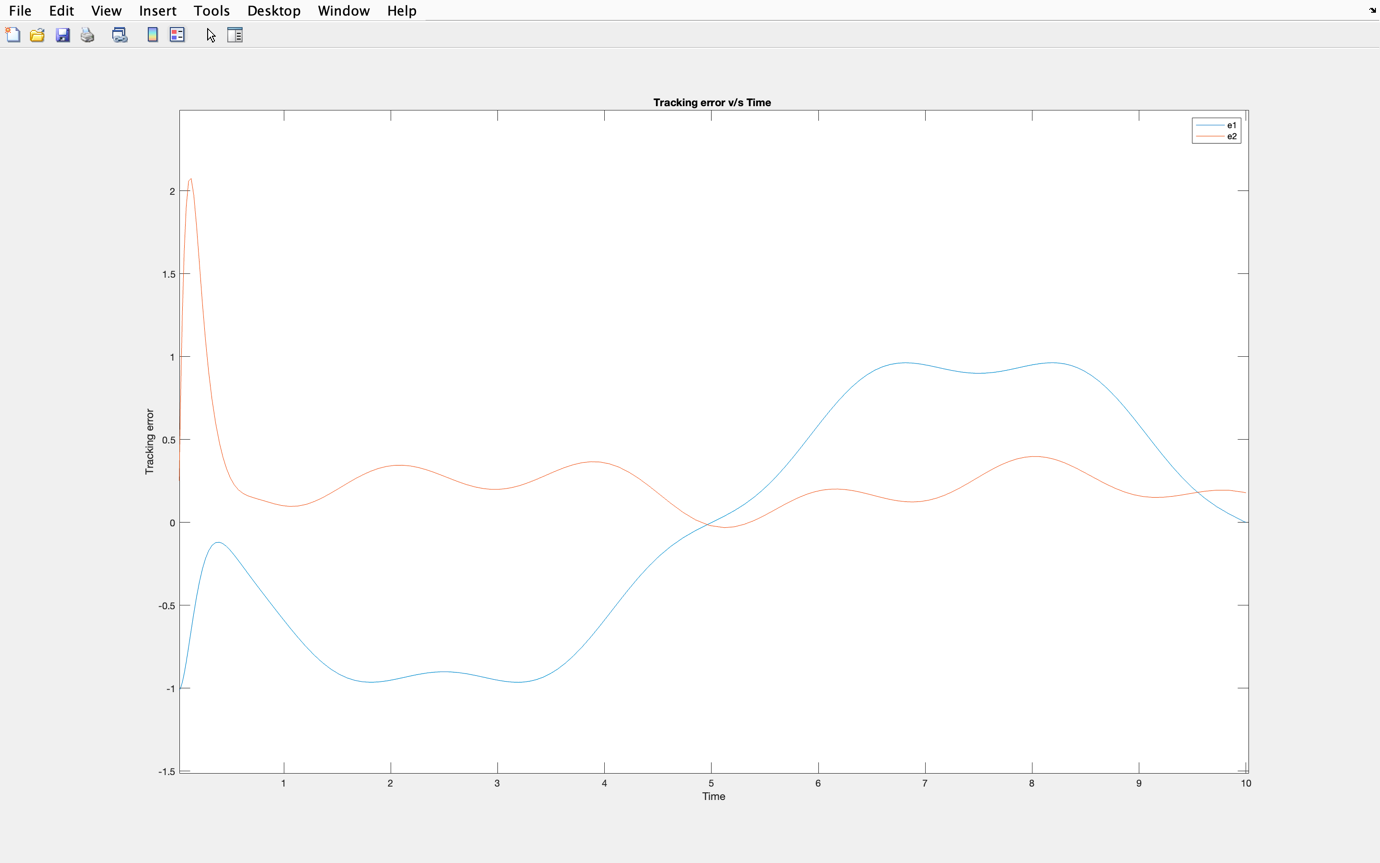
Output graph:

****

Joint angle graph:

****

Tracking error graph:

****

Conclusion:

The FBLINCT controller was successfully implemented and outputs obtained were verified

**THANK YOU ALL!**