The following code has been developed based on the planar Arm code provided for HW4 in addition to adding an animation part for animating the robotic arm responding to the PD controller:

The PlanarArm implementation code:

%This code has been developed by: Mostafa Atalla%

%HW:4 for RBE502 Robotics Control Course by: Prof. Jie Fu%

clc

clear all;

close all;

%% Initialization %%

x0=[1 0.4 0.3 0.1]; %Setting initial conditions for the state vector

xf = [0,0, 0, 0]; %Final state desired

kp=[250 0;0 250]; %Proportional gain vector

kd=[160 0;0 160]; %Dervative gain vector

tf=10; %Final time

%% Implement the PD control for set point tracking %%

options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);

[T,X] = ode45(@(t,x) odefcn(t,x,xf,kp,kd),[0 tf],x0, options);

%% Plotting theta1,theta2 under the PD controller %%

figure('Name', 'Theta1, Theta2 under PD SetPoint Control')

movegui(1,'northeast')

plot(T, (X(:,1)\*(180/pi)),'r-');

hold on

plot(T, (X(:,2)\*(180/pi)),'r--');

hold on

title('Theta1, Theta2 under PD SetPoint Control');

xlabel('TIME')

ylabel('THETA (Degrees)')

legend('Theta1','Theta2')

grid on

%% Animation of the TWO Link Manipulator %%

%Manipulator Parameters%

I1=10; I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;

%Initialization of Tip point of every Link%

[k j]=size(X);

pointl1=zeros(k,2); %End Point of Link1

pointl2=zeros(k,2); %End Point of Link2

%Create Figure for the Animation%

figure('Name', 'Animation of the 2 Link Manpiulator under PD Control')

movegui(2,'northwest')

%Computing the Tip Point positions corresponding to theta1, theta2%

for ii=1:k;

%Adjusting Figure for every iteration%

clf

xlabel('X - POSITION')

ylabel('Y - POSITION')

grid on

title('TWO LINK MANIPULATOR ANIMATION UNDER PD CONTROL');

axis([floor(min(pointl2(:,1))) 2.2 floor(min(pointl2(:,2))) 3]);

axis square

%Computing the points%

pointl1(ii,1) = l1\*cos(X(ii,1)) ; %Computing x positions of end point of link 1

pointl1(ii,2) = l1\*sin(X(ii,1)); %Computing y positions of end point of link 1

pointl2(ii,1) = pointl1(ii,1) + (l2\*cos(X(ii,1)+X(ii,2))); %Computing x positions of end point of link 2

pointl2(ii,2) = pointl1(ii,2)+(l2\*sin(X(ii,1)+X(ii,2))); %Computing y positions of end point of link 2

%Plotting the links%

line([0,pointl1(ii,1)],[0,pointl1(ii,2)],'linewidth',2,'color','black');

line([pointl1(ii,1),pointl2(ii,1)],[pointl1(ii,2),pointl2(ii,2)],'linewidth',2,'color','blue');

hold on

plot(0,0,'o','markersize',7)

plot(pointl1(ii,1),pointl1(ii,2),'o','markersize',7)

plot(pointl2(ii,1),pointl2(ii,2),'o','markersize',7)

plot(pointl2(:,1),pointl2(:,2),'-')

pause(.07)

end

The ODE Function code:

%This code has been developed by: Mostafa Atalla%

%HW:4 for RBE502 Robotics Control Course by: Prof. Jie Fu%

%% Defining the 1st order ode to be solved by the ode45%%

function dxdt=odefcn(t,x,xf,kp,kd)

%Manipulator Parameters%

I1=10; I2 = 10; m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;

a = I1+I2+m1\*r1^2+ m2\*(l1^2+ r2^2);

b = m2\*l1\*r2;

d = I2+ m2\*r2^2;

%Inertia and Coriolos Matrices Computation%

M = [a+2\*b\*cos(x(2)), d+b\*cos(x(2)); d+b\*cos(x(2)), d] %Inertia Matrix

C = [-b\*sin(x(2))\*x(4), -b\*sin(x(2))\*(x(3)+x(4)); b\*sin(x(2))\*x(3),0] %Coriolos Materix

invM = inv(M);

invMC= inv(M)\*C;

%Setting PD Controller for each Joint input Torque - Set Point Tracking%

%General PD Controller u=-Kp(e)-Kd\*e\_dot

e=[x(1)-xf(1);x(2)-xf(2)]; %position error vector

e\_dot=[x(3)-xf(3);x(4)-xf(4)]; %Velocity error vector

u=-kp\*e-kd\*e\_dot; %Controller

%State Space Representation%

%4 first order ode%

dxdt=zeros(4,1); %initialization of first order ode vector%

dxdt(1)=x(3);

dxdt(2)=x(4);

dxdt(3)= invMC(1,:)\*[x(3);x(4)]+invM(1,:)\*u;

dxdt(4)= invMC(2,:)\*[x(3);x(4)]+invM(2,:)\*u;

end

Results:

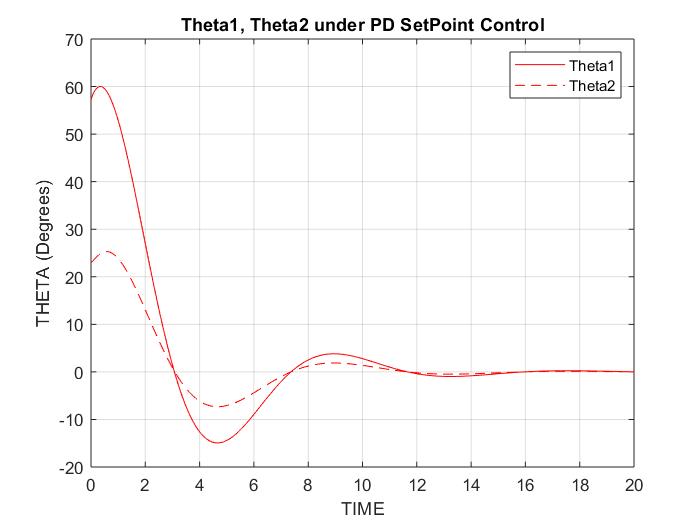
Given this initialization values:

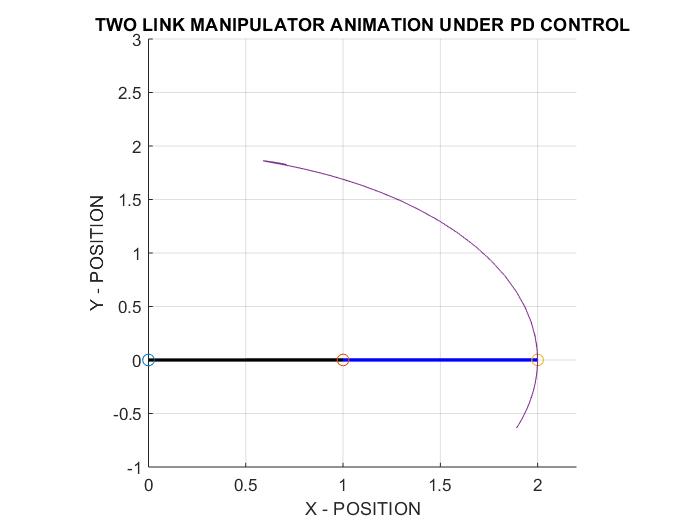
x0=[1 0.4 0.3 0.1]; %Setting initial conditions for the state vector

xf = [0,0, 0, 0]; %Final state desired

kp=[300 0;0 300]; %Proportional gain vector

kd=[150 0;0 150]; %Derivative gain vector



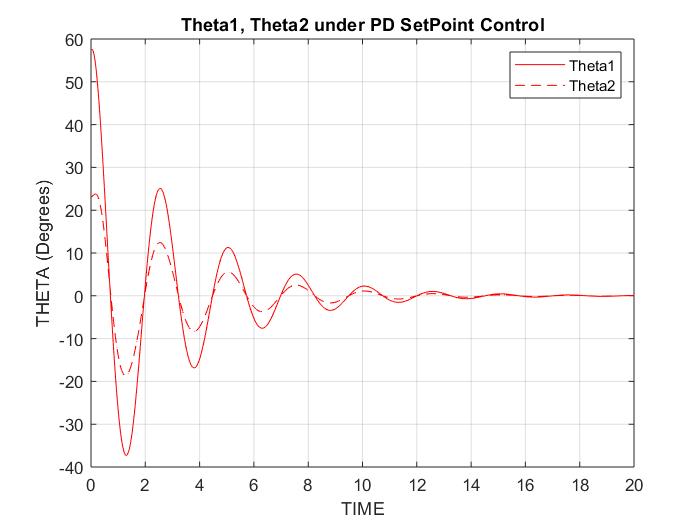


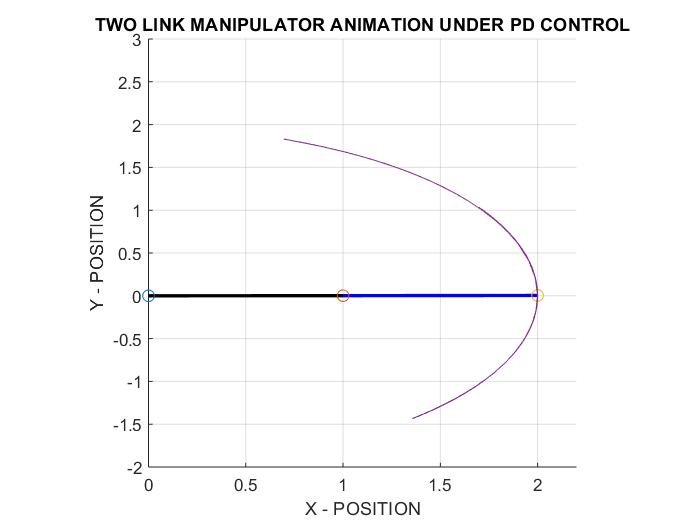
Increasing the proportional gain values to be 250 to make the response faster:

kp=[250 0;0 250]; %Proportional gain vector

kd=[25 0;0 25]; %Derivative gain vector

The response becomes faster but with a lot of oscillations > Increase Kd

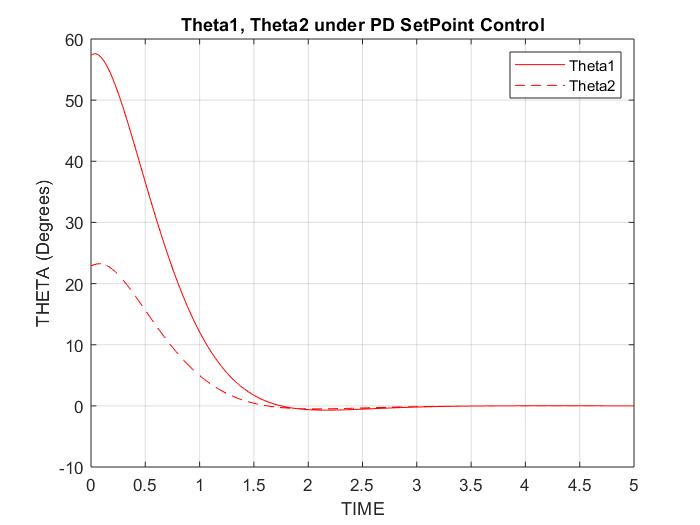




Increasing the derivative gain values to be 160 to make the response faster:

kp=[250 0;0 250]; %Proportional gain vector

kd=[160 0;0 160]; %Derivative gain vector

The oscillations have been eliminated.

