EENG 428 Introduction to Robotics Laboratory

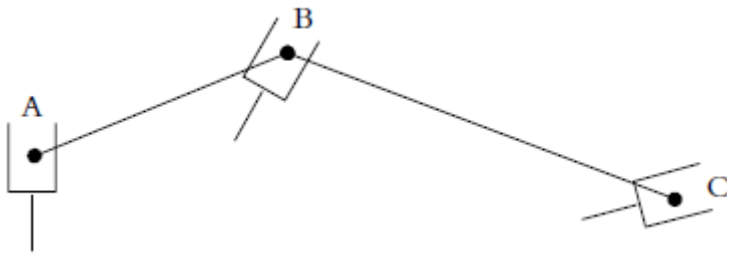
Lab Session 7

Objective:

Path and trajectory planning relates to the way a robot is moved from one location to another in a controlled manner. In this Lab session the concept of path and trajectory will be discussed in task-space and joint-space. Matlab will be used in this session in order to provide a practical example to serve the purpose of this session.

Path and Trajectory:

A path is defined as the positional description of robot configurations in a particular order. However, a path has no timing restrictions other than the order of the configurations. For example, a robot arm may be desired to go through a path described by the endpoint positions A, B and C as shown in the figure

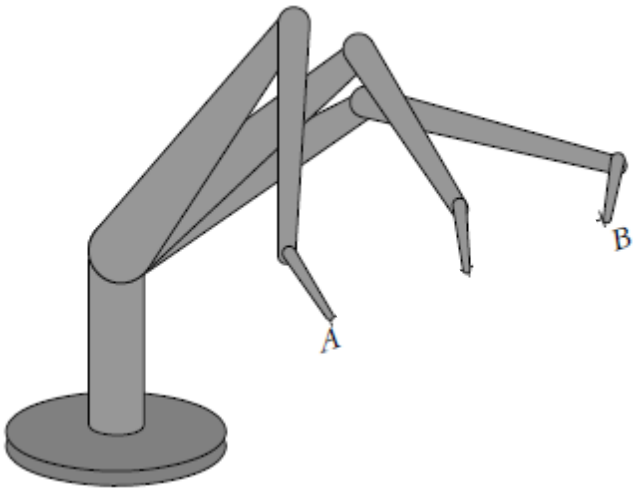


A trajectory is a path with all necessary timing specifications to calculate the required position, and velocity of the robot configuration. For example, the set of endpoint positions A, B, and C in the previous example together with the specification of velocity at each of these points forms a well-defined trajectory, since the time to reach each of the points A, B, C and also to the intermediate points between them can be interpolated from this information.

1- Background: (Cartesian-Space and Joint-Space Path Description)

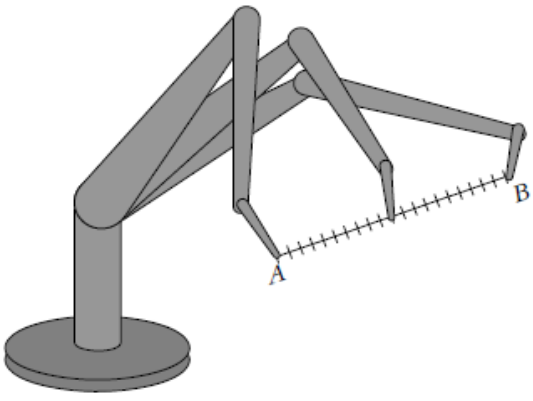
The robot configurations can be expressed either in end-positions relative to the reference frame, or by joint displacement vector, a vector in the joint-space. Consequently, a desired path is conveniently described in any of these two vector spaces. However, the interpolation of the intermediate points in different spaces results in different intermediate paths.

Consider the robot in the following figure, this robot is desired to move from point A to point B in space.



The inverse kinematic equations can be used in order to calculate the joint displacements the robot needs to get to the new location. The joint values thus calculated can be used to drive the robot joints to their new values and, consequently, move the robot arm to its new position. The description of the motion to be made by the robot by its joint values is called joint-space. In this case, although the robot will reach the desired position, the motion between the two points is unpredictable.

Now assume that a straight line is drawn between points A and B, and it is desirable to have the robot move from point A to point B in a straight line.



To do this we need to divide the line into small portions, and to move the robot through all these intermediate points. To do this task, at each intermediate location, the robot's inverse kinematic equations are solved to get joint variables which is used to drive the robot joints to the new calculated values. When all segments are completed, the robot will be at point B

In this case the sequence of movements the robot makes is described in Cartesian-space and is converted to joint-space at each segment.

Cartesian-space description is much more computationally intensive than the joint-space description, but yields a controlled and known path.

Example1:

consider a simple 2-DOF robot. We desire to move the robot from point A to point B. The configuration of the robot at point A is $\alpha = 20°$ and $\beta = 30°$ in order for the robot to be at point B, the configuration calculated to be as $\alpha = 40°$ and $\beta = 80°$. Also suppose that both joints of the robot can move at the maximum rate of 10 degrees/sec. Plan this motion in Joint-Space.

One way to move the robot from point A to B is to run both joints at their maximum angular velocities.

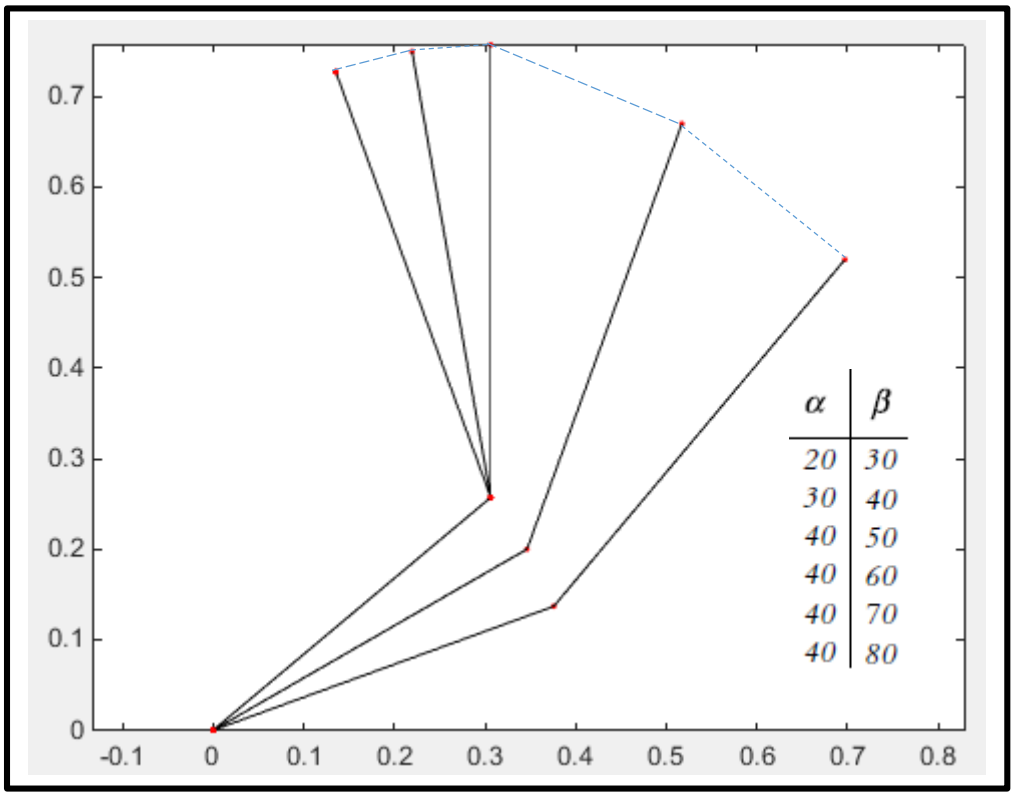
```
clear all; clc;clf; close all;format compact;
r.b1=0.4; r.b2=0.5; % these are robot link lengths.
n=10;
q0 = [20/57.2952 30/57.2952]
qn = [40/57.2952 80/57.2952]
dq= 10/57.2952;
qq1=q0(1);
qq2=q0(2);
q=[qq1 qq2];
for i=2:n

 qq1(i)=qq1(i-1) + dq;
  if (qq1(i)> qn(1))
      qq1(i)=qq1(i-1);
  end
 qq2(i)= qq2(i-1) + dq;
 if (qq2(i)>= qn(2))
     qq2(i)=qq2(i-1);
 end
 q(i,:) =[qq1(i) qq2(i)];
 r = setjoint(r, q(i,1), q(i,2));
 q(i,:)=[r.q1(1) r.q2(1)];
 p(i,:) = [r.p(1) r.p(2)];
end
% Planar Plot of the Motion
x=[0 0 0]; y=[0 0 0];
for i=1:length(q);
xo=x; yo=y;
x(1)=0; y(1)=0; % origin
x(2)= r.b1*cos(q(i,1)); y(2)=r.b1*sin(q(i,1));
x(3)= x(2)+r.b2*cos(q(i,1)+q(i,2));
y(3)= y(2)+r.b2*sin(q(i,1)+q(i,2));
plot(xo,yo,'k-'); hold on;
plot(x,y,'k-');
plot(x,y,'r.');
axis equal;
pause(0.01);
end
```

```
function rr = setjoint(r,q1,q2);
rr=r;
rr.q1=q1 ; rr.q2=q2 ; rr.q=[q1 q2];
rr.S1=sin(q1); rr.C1=cos(q1);
rr.S2=sin(q2); rr.C2=cos(q2);
rr.S12=sin(q1+q2); rr.C12=cos(q1+q2);
rr.x0 = rr.b2*rr.C12+rr.b1*rr.C1;
rr.y0 = rr.b2*rr.S12+rr.b1*rr.S1;
rr.p=[rr.x0 rr.y0];
end
```

```
function rr= setcoord(r,x0,y0);
rr=r; c=(x0^2+y0^2+r.b1^2-r.b2^2)/2.0/r.b1;
rr.q1 = [atan2(y0,x0)+atan2(sqrt(x0^2+y0^2-c^2), c)
 atan2(y0,x0)-atan2(sqrt(x0^2+y0^2-c^2), c)];
rr.S1 = sin(rr.q1); rr.C1=cos(rr.q1);
rr.q2 =atan2( y0-r.b1*rr.S1, x0-r.b1*rr.C1)-rr.q1;
rr.S2 = sin(rr.q2); rr.C2=cos(rr.q2);
rr.S12 = sin(rr.q1+rr.q2); rr.C12=cos(rr.q1+rr.q2);
end
```

The Output will be as following



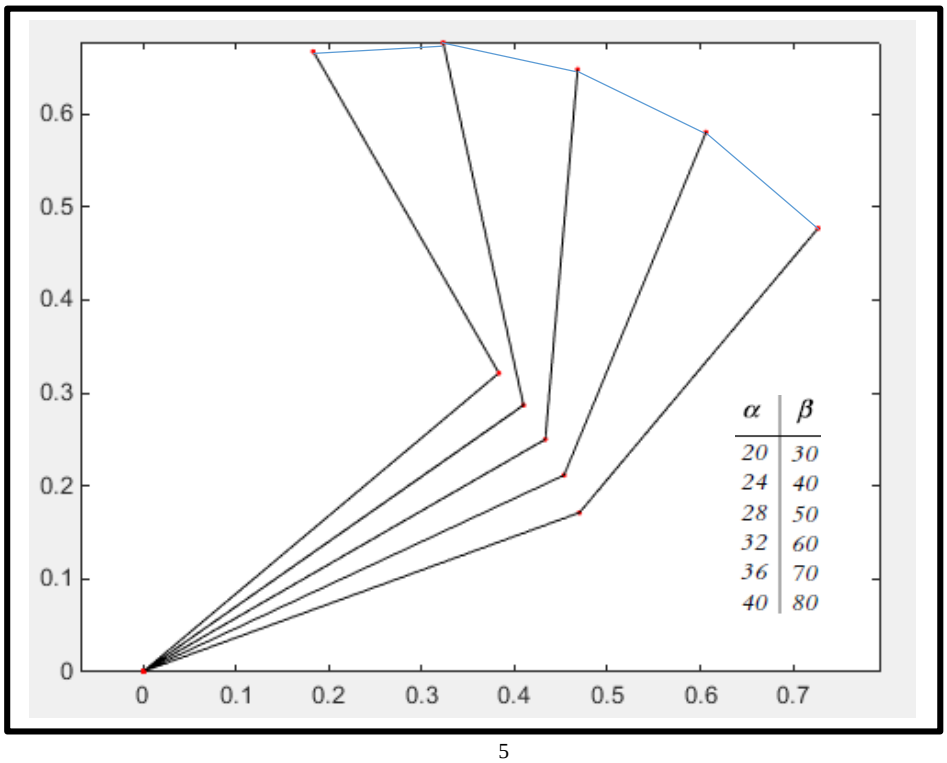
It is clear that, at the end of the second time interval, the lower link of the robot will have finished its motion, while the upper link continues for another three-time interval. Moreover, the trajectory of the end-effector of the robot is irregular, and the distances traveled by the robot's end are not uniform.

Example2:

Consider the previous example and suppose that the motions of both joints of the robot are normalized such that the joint with smaller motion will move proportionally slower so that both joints will start and stop their motion simultaneously.

```
clear all; clc;clf; close all;format compact;
r.b1=0.5; r.b2=0.4; % these are robot link lengths.
n=5;
q0 = [20/57.2952 30/57.2952]
qn = [40/57.2952 80/57.2952]
dq= (qn-q0)/(n-1);
for i=1:n
 q(i,:) = q0 + dq*(i-1)
 r = setjoint(r, q(i,1), q(i,2)); % set the joint disp to soln=1
 q(i,:)=[r.q1(1) r.q2(1)];
 p(i,:) = [r.p(1) r.p(2)];
end
% Planar Plot of the Motion
x=[0 0 0]; y=[0 0 0];
for i=1:length(q);
xo=x; yo=y;
x(1)=0; y(1)=0; % origin
x(2)= r.b1*cos(q(i,1)); y(2)=r.b1*sin(q(i,1));
x(3)= x(2)+r.b2*cos(q(i,1)+q(i,2));
y(3)= y(2)+r.b2*sin(q(i,1)+q(i,2));
plot(xo,yo,'k-'); hold on;
plot(x,y,'k-');
plot(x,y,'r.');
axis equal;
pause(0.01);
end
```

The Output will be as following

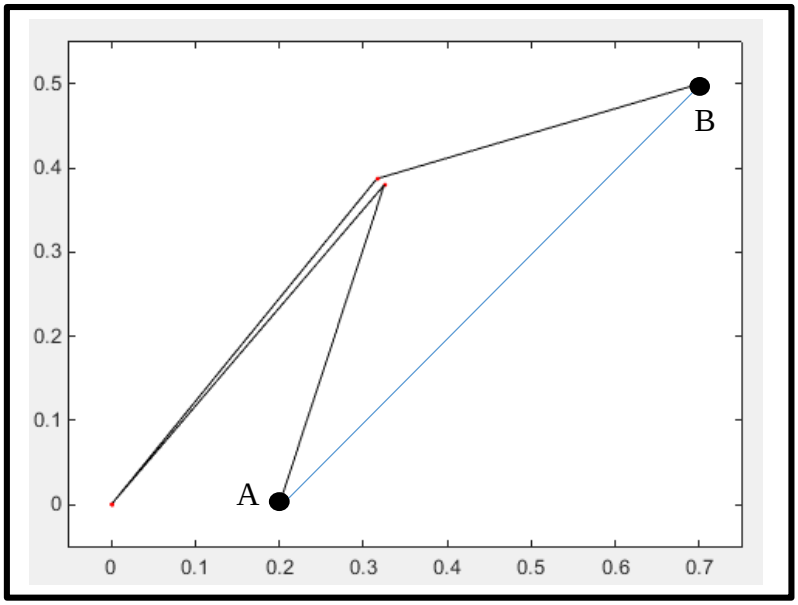


Notice that the segments of the movement are much more similar to each other than before, but the path is still irregular and different from the previous case.

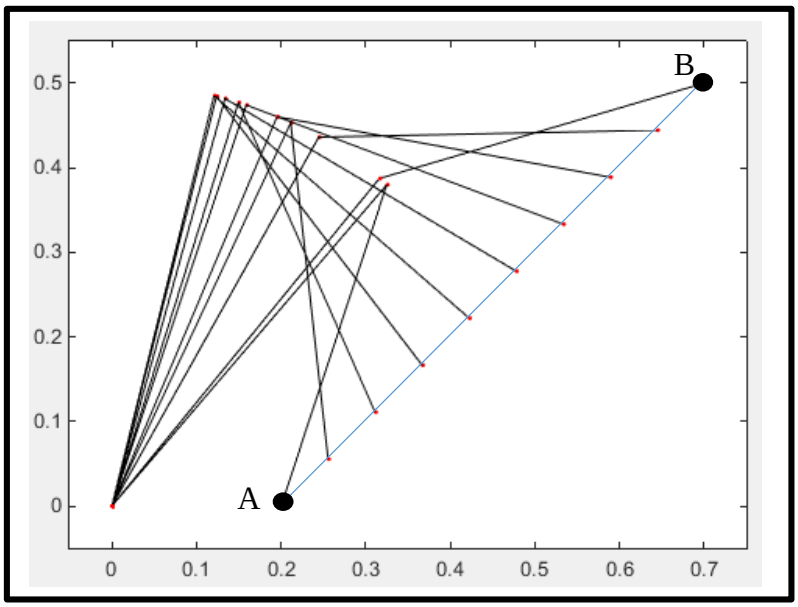
Example 1 and Example 2 planned in joint-space with the values of the joints, not the location of the end of the mechanism. The only calculation needed was the joint values for the destination and, in the second Example, normalization of the joint velocities.

Example 3

suppose we want the robot's hand to follow a known path (straight line) between points A $(0.2\ , 0)$ and B $(0.7, 0.5)$.



The simplest solution would be divide the line into segments, and solve for necessary angles α and β at each point.



This technique is called interpolation between points A and B. The resultant trajectory is in Cartesian-space since all segments of the motion must be calculated based on the information expressed in a Cartesian frame

```
clear all; clc;clf; close all;format compact;
r.b1=0.5; r.b2=0.4; % these are robot link lengths.
n=10;
p0 = [ 0.2 0]; pn = [ 0.7 0.5];
dp= (pn-p0)/(n-1);
for i=1:n
 p(i,:) = p0 + dp*(i-1)
 r = setcoord( r, p(i,1),p(i,2) ); % set the coordinates in base
frame
 r = setjoint(r, r.q1(1), r.q2(1)); % set the joint disp to soln=1
 q(i,:)=[ r.q1(1) r.q2(1) ];
 p(i,:) = [r.p(1) r.p(2)];
end
% Planar Plot of the Motion
x=[0 0 0]; y=[0 0 0];
for i=1:length(q);
xo=x; yo=y;
x(1)=0; y(1)=0; % origin
x(2)= r.b1*cos(q(i,1)); y(2)=r.b1*sin(q(i,1));
x(3)= x(2)+r.b2*cos(q(i,1)+q(i,2));
y(3)= y(2)+r.b2*sin(q(i,1)+q(i,2));
plot(xo,yo,'k-'); hold on;
plot(x,y,'k-');
plot(x+0.001,y,'r.');
axis equal; axis([-0.05 0.75 -0.05 0.55]);
pause(0.01);
end
```

2- Joint-Space Third-Order Polynomial Trajectory Planning

In this application, the initial location and orientation of the robot are known and using the inverse kinematic equations, the final joint angles for the desired position and orientation should be found. However, the motions of each joint of the robot must be planned individually. consider one of the joints, which at the beginning of the motion segment at time t_i is at θ_i, and which we want to move to a new value of θ_f at time t_f. One way to do this is to use a polynomial to plan the trajectory such that the initial and final boundary conditions (the Joints angle and velocities at the beginning and the end of the motion) become satisfy

Third-order polynomial function has the form

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

the derivative of the Third-order polynomial function has the form

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2$$

The four boundary conditions

$$\theta(t_i) = \theta_i \ , \qquad \theta(t_f) = \theta_f \ , \qquad \dot{\theta}(t_i) = \omega_i \ , \qquad \dot{\theta}(t_f) = \omega_f$$

These four pieces of information (the four boundary conditions) allow us to solve for four unknowns (c_0, c_1, c_2 and c_3)

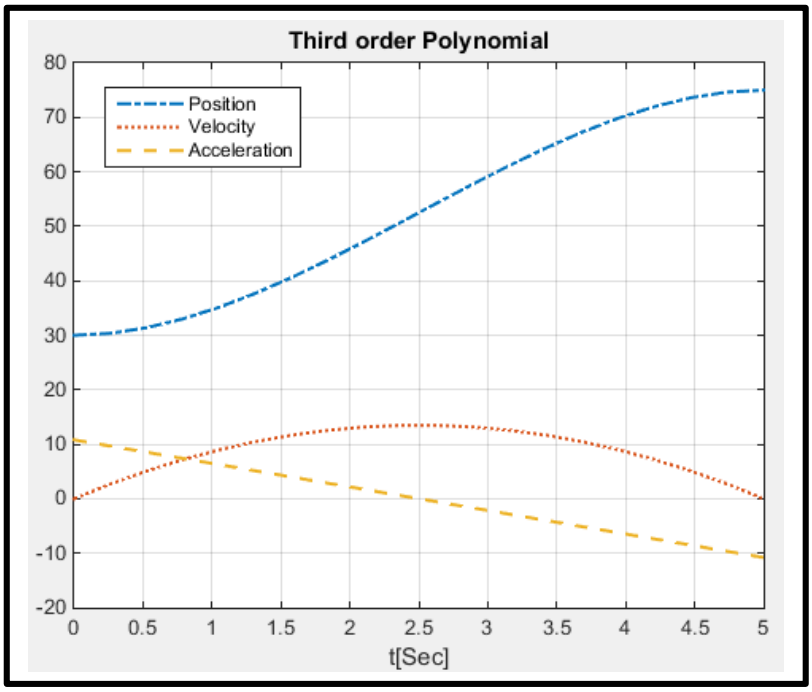
```
function [c3,c2,c1,c0] =
createTraj3(theta0,thetaf,thetad0,thetadf,tstart,tfinal)
    T = tfinal - tstart;
    c0 = theta0;
    c1 = thetad0;
    c2 = (-3 * (theta0 - thetaf) - (2 * thetad0+thetadf )*T)/ T ^ 2;
    c3 = (2 * (theta0 - thetaf) + (thetad0+thetadf )*T)/ T ^ 3;
end
```

Example 4

It is desired to have the first joint of a 6-axis robot go from initial angle of 30° to a final angle of 75° in 5 seconds. Using a third-order polynomial.

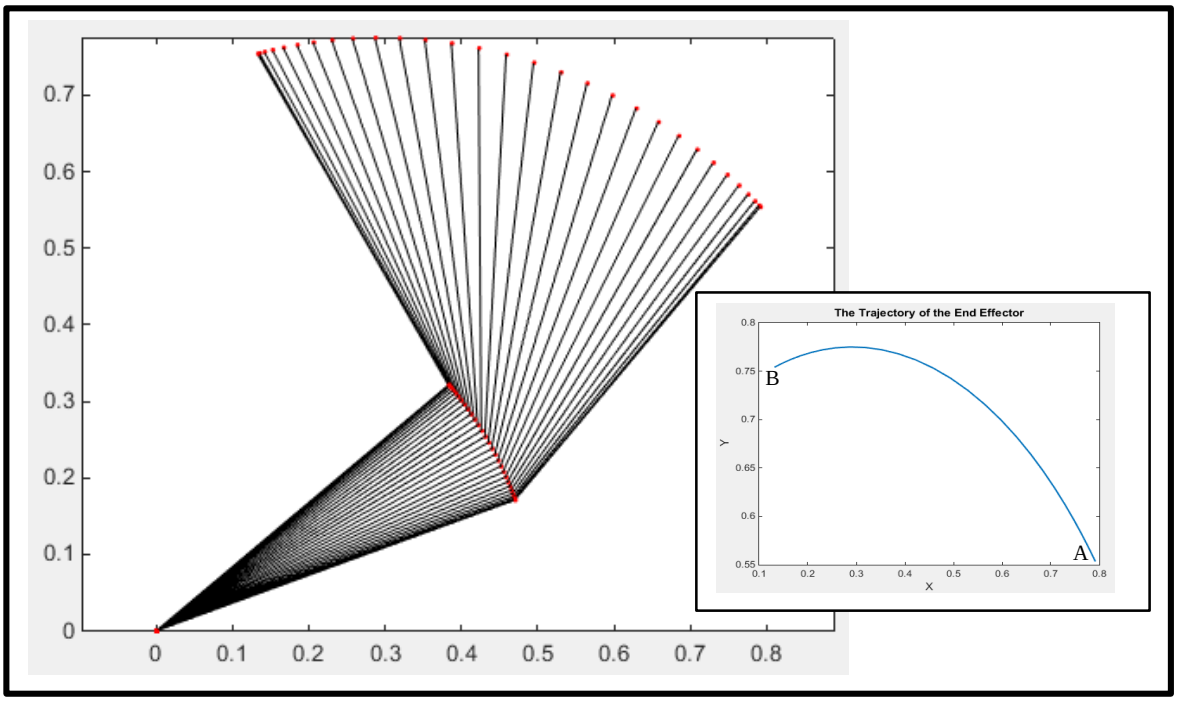
```
clc
clear all
close all
theta0 = 30; % Deg
thetaf = 75;
thetad0 = 0; % Deg/Sec
thetadf = 0;
tstart = 0; % seconds
tfinal = 5;
% find the coefficient of the 3rd order polynomial
trajectory
[a3,a2,a1,a0] =
createTraj3(theta0,thetaf,thetad0,thetadf,tstart,tfinal);
% make a polynomial
p = [a3,a2,a1,a0];
% Create time vector
t = linspace(tstart,tfinal,20);
% Evaluate the polynomial : Position
pos = polyval(p,t);
% calculate the first derivative : Velocity
pd = polyder(p);
% Evaluate the velocity
vel = polyval(pd,t);
% calculate the second derivative : Acceleration
pdd = polyder(pd);
% Evaluate the acceleration
acc = polyval(pdd,t);
% plot
figure;
P=plot(t,pos,'-.'); hold on
V=plot(t,vel,':'); hold on
A=plot(t,acc,'--'); hold on
title('Third order Polynomial')
xlabel('t[Sec]')
legend('Position','Velocity','Acceleration')
P(1).LineWidth = 1.5;
V(1).LineWidth = 1.5;
A(1).LineWidth = 1.5;
grid
```

The output shows the joint angles, velocities, and accelerations of the joint from initial angle of 30° to a final angle of 75° in 5 seconds using third-order polynomial function.



Example 5:

consider a simple 2-DOF robot. We desire to move the robot from point A to point B in 3 second. The configuration of the robot at point A is $\alpha = 20°, \beta = 30°$ and at the point B is $\alpha = 40°, \beta = 80°$. Plan this motion in the Joint-Space using the Third order polynomial function.



```
clc
clear all
close all

tstart = 0;
tfinal = 3;
dt=0.01;
L1=0.5;
L2=0.5;

theta_1_0 = 20/57.2958;
theta_1_f = 40/57.2958;
velocity_1_0 = 0;
velocity_1_f = 0;

theta_2_0 = 30/57.2958;
theta_2_f = 80/57.2958;
velocity_2_0 = 0;
velocity_2_f = 0;

% find the coefficient of the 3rd order polynomial
trajectory
[a3,a2,a1,a0] =
createTraj3(theta_1_0,theta_1_f,0,0,tstart,tfinal);
% make a polynomial
p1 = [a3,a2,a1,a0];
% Create time vector
t = linspace(tstart,tfinal,30); % time step is 0.1
% Evaluate the polynomial : Position
pos1 = polyval(p1,t)

% find the coefficient of the 3rd order polynomial
trajectory
[a33,a22,a11,a00] =
createTraj3(theta_2_0,theta_2_f,0,0,tstart,tfinal);
% make a polynomial
p2 = [a33,a22,a11,a00];
% Evaluate the polynomial : Position
pos2 = polyval(p2,t)
for i=1:30
x0(i) = L2*cos(pos1(i)+pos2(i))+L1*cos(pos1(i))
y0(i) = L2*sin(pos1(i)+pos2(i))+L1*sin(pos1(i));
ppp(i,:)=[x0(i) y0(i)]
end
figure;
PLOT1=plot(ppp(:,1),ppp(:,2))
title('The Trajectory of the End Effector')
xlabel('X')
ylabel('Y')
PLOT1(1).LineWidth = 1.5;
figure;
x=[0 0 0]; y=[0 0 0];
for i=1:length(ppp);
xoo=x; yoo=y;
x(1)=0; y(1)=0; % origin
x(2)= L1*cos(pos1(i)); y(2)=L1*sin(pos1(i));
x(3)= x(2)+L2*cos(pos1(i)+pos2(i))
y(3)= y(2)+L2*sin(pos1(i)+pos2(i))
plot(xoo,yoo,'k-'); hold on;
plot(x,y,'k-');hold on;
plot(x+0.001,y,'r.');
axis equal;
pause(0.5);
end
```

3- Cartesian-Space Trajectories

Cartesian-space trajectories relate to the motions of a robot relative to the Cartesian reference frame. For Cartesian-space the joint values must be repeatedly calculated through the inverse kinematic equations of the robot. This means that unlike the joint-space schemes in which the generated values relate directly to joint values, in Cartesian-space planning, the calculated values from the functions are positions (and orientations) of the hand, and they must still be converted to joint values through the inverse Kinematic equations.

This can be simplified into a loop as follows:

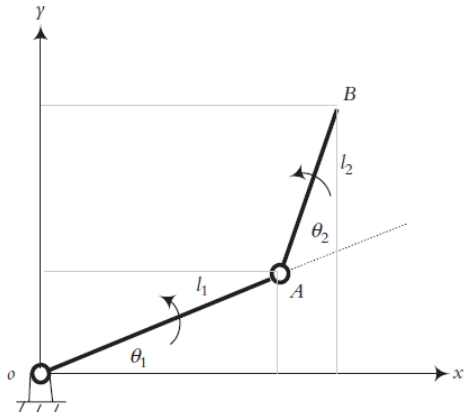
- Increment the time by $t = t + \Delta t$
- Calculate the position and orientation of the hand based on the selected function for the trajectory.
- Calculate the joint values for the position and orientation through the inverse kinematic equations of the robot.
- Send the joint information to the controller.
- Go to the beginning of the loop.

Straight-line motions between points are the most practical trajectories for industrial applications. To accomplish a straight-line trajectory, the transformation between the initial and final positions and orientations must be calculated and divided into small segments. Since we want to have a smooth straight-line transformation between the initial and final locations, we need a large number of very small segments. This, in reality, creates a large number of differential motions. Jacobian Matrix can relate the position and orientation of the hand frame at each new segment to the differential motions, and joint velocities

$$\begin{bmatrix} d\theta_1 \\ d\theta_2 \\ d\theta_3 \\ d\theta_4 \\ d\theta_5 \\ d\theta_6 \end{bmatrix} = [\ J\ ]^{-1} \begin{bmatrix} dx \\ dy \\ dz \\ \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

Example 6

consider a simple 2-DOF mechanism where each link can independently rotate. The rotation of the first link θ_1 is measured relative to the reference frame, whereas the rotation of the second link θ_2 is measured relative to the first link.



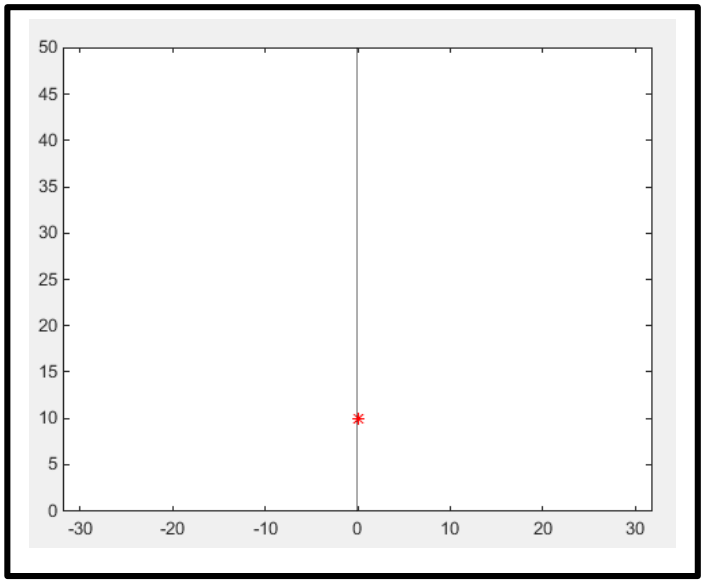
- The equations that describe the position of point B

$$x_B = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y_B = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

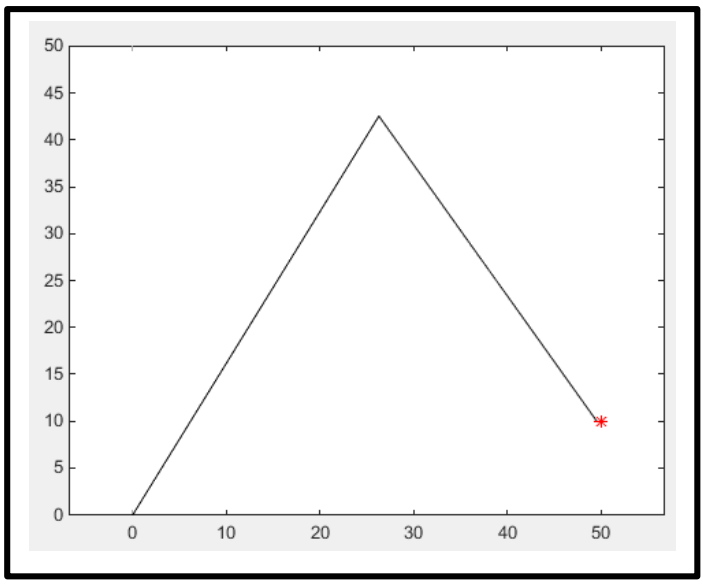
- Let the manipulator link lengths as following

$l1 = 50$	$l2 = 40\ cm$

Let the point B move from initial point (0, 10) to the final point (50,10) in 4 seconds. Assuming the joint-position for initial point is (pi/2, -pi) radians, calculate the joint displacements along the trajectory using the inverse Jacobian and implement the solution in Matlab.



Initial State



Final State

Let

$$p1 = (0,10) \quad \text{and} \quad pn = (50,10)$$

$$q1 = (pi/2, -pi)$$

$$i = 1$$

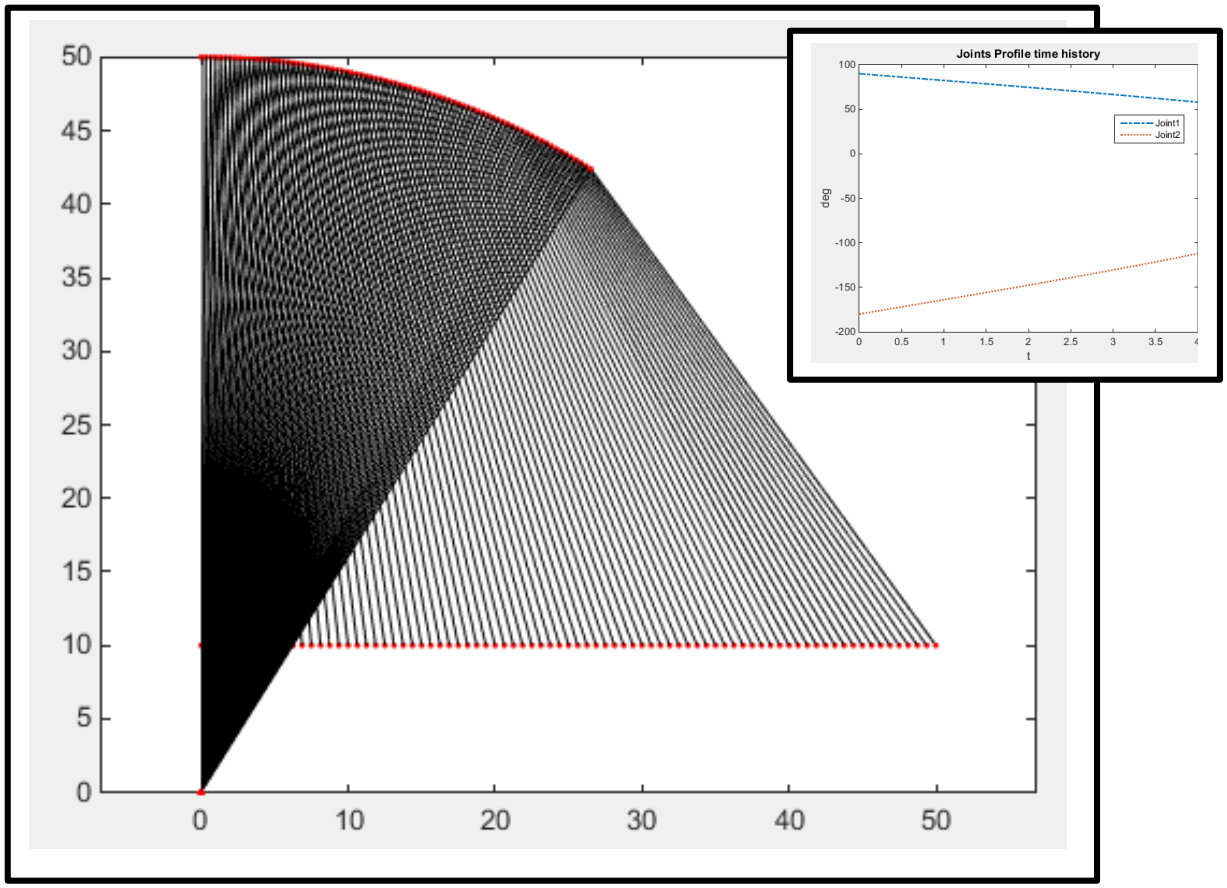
$$t = 4 \quad and \quad dt = 0.05 \quad then \quad n = t/dt$$

For $k = 1{:}\,n$

$$i = i + 1$$

$$pi = p1 + \left(\frac{k}{n}\right) * (\, pn - p0\,)$$

$$q_i = q_{i-1} + (\, J^{-1} * (p_i - p_{i-1})\,)$$



```
clear all
clc; clf; close all;
r.a1 = 50 ; r.a2 = 40 ;
p0=[0 10] ;
q0= [pi/2 -pi];
pn=[50 10];
t=4;
dt=0.05;
n=t/dt;
i=1;
pp(i,:)=p0;
qq(i,:)=q0
for k=1:n
i=i+1;
pp(i,:)= p0 + (k/n)*(pn-p0);
q=qq(i-1,:);
r.C1=cos(q(1)); r.S1=sin(q(1));
r.C2=cos(q(2)); r.S2=sin(q(2));
r.C12=cos(q(1)+q(2)); r.S12=sin(q(1)+q(2));
qq(i,:)= q + transpose(IJ(r)* transpose( pp(i,:)-pp(i-1,:)))
end
x=[0 0 0]; y=[0 0 0];
for i=1:length(qq);
xo=x; yo=y;
x(1)=0; y(1)=0; % origin
x(2)= r.a1*cos(qq(i,1)); y(2)=r.a1*sin(qq(i,1));
x(3)= x(2)+r.a2*cos(qq(i,1)+qq(i,2));
y(3)= y(2)+r.a2*sin(qq(i,1)+qq(i,2));
plot(xo,yo,'k-'); hold on;
plot(x,y,'k-');   hold on;
  plot(x+0.001,y,'r.'); hold on;
 axis equal;
pause(0.05);
end
figure;
P1=plot(0:dt:t,qq(:,1)*57.2958,'-.');hold on;
P2=plot(0:dt:t,qq(:,2)*57.2958,':');hold on;
title('Joints Profile')
legend('Joint1','Joint2')
P1(1).LineWidth = 1.5;
P2(1).LineWidth = 1.5;
```

```
% function for finding the inverse jacobian for the given 2 links manipulator
function Jinv = IJ(r);
a1 = 50 ; a2 = 40 ;
J = [-a1*r.S1-a2*r.S12 -a2*r.S12
a1*r.C1+a2*r.C12 a2*r.C12 ];
D = a1*a2*r.S2 ;
Jinv = [ J(2,2)/D -J(1,2)/D ;
-J(2,1)/D J(1,1)/D ];
end
```