HW4 ME 739 Introduction to robotics

Spring 2015
Department of Mechanical Engineering
University of Wisconsin, Madison

Instructor: Professor Michael Zinn

By

Nasser M. Abbasi

May 3, 2022

Contents

List of Tables

0.1 Problem 1

Problem 1 [20 points]

► Write a Matlab function which constructs a quintic polynomial, for the purposes of trajectory generation, given the following input parameters

t_0:	Initial time	t_f:	Final time
y_0:	Initial position	y_f:	Final position
$\dot{y}_0$:	Initial velocity	$\dot{y}_f$:	Final velocity
$\ddot{y}_0$:	Initial acceleration	$\ddot{y}_f$:	Final acceleration

An example function prototype is shown below

`[t, y, ẏ, ÿ] = QuinticPolynomial(t0, tf, y0, yf, ẏ0, ẏf, ÿ0, ÿf, n)`

where *n* is the number of elements in the time vector and output arrays (`[y, ẏ, ÿ]`)

t	:	*n*-dimensional array of trajectory time
y	:	*n*-dimensional array of trajectory position
$\dot{y}$	:	*n*-dimensional array of trajectory velocity
$\ddot{y}$	:	*n*-dimensional array of trajectory acceleration

► To check your function, create and plot the trajectory for the input parameters given below:

t_0:	0	t_f:	5
y_0:	-10	y_f:	10
$\dot{y}_0$:	-50	$\dot{y}_f$:	-50
$\ddot{y}_0$:	0	$\ddot{y}_f$:	0

e.g.
```
[t, y, dy, ddy] = QuinticPolynomial(0,5,-10,10,-50,50,0,0,1000);
figure; plot(t,y);
figure; plot(t, dy);
figure; plot(t, ddy);
```

Figure 1: Problem 1 description

A Matlab function called **QuinticPolynomial** was implemented. The following shows the call made and the three plots generated. The output matched the required output in the problem statement.

```
[t,y,dy,dyy] = QuinticPolynomial(0,5,-10,10,-50,-50,0,0,1000);
figure;
plot(t,y);
title('HW4, problem1, y(t) solution')

figure;
plot(t,dy);
title('HW4, problem1, dy(t) solution')

figure;
plot(t,dyy);
title('HW4, problem1, ddy(t) solution')
```

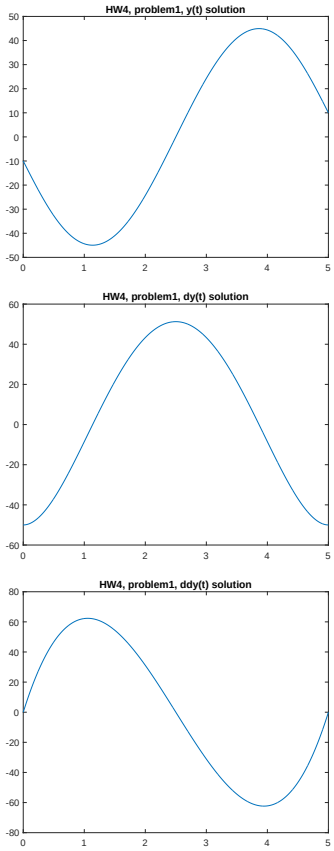


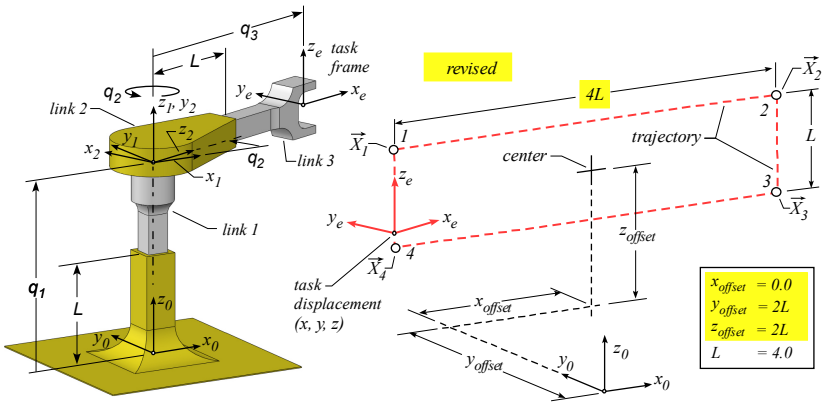
Figure 2: Problem 1 plots

The following is the Matlab source code listings of the above function.

```
function [t,y,dy,ddy] = QuinticPolynomial(t0,tf,y0,yf,dy0,dyf,ddy0,ddyf,n)
%function QuinticPolynomial to solve trajectory generation using quintic
%polynomial method
%ME 739, UW Madison, Spring 2015
%by Nasser M. Abbasi

%INPUT:
%t0   : initial time
%tf   : end time
%y0   : initial position
%yf   : final position
%dy0  : initial speed
%dyf  : final speed
%ddy0 : initial acceleration
%ddyf : final acceleration
%n    : number of samples
%
%OUTPUT
%t    : time vector
%y    : position vector
%dy   : speed vector
%ddy  : acceleration vector

syms t a0 a1 a2 a3 a4 a5;
%set up the polynomial
y   = a0+a1*t+a2*t^2+a3*t^3+a4*t^4+a5*t^5;
dy  = diff(y,t);
ddy = diff(dy,t);

%setup the 6 contraints
eq1 = subs(y,t,t0)==y0;
eq2 = subs(y,t,tf)==yf;
eq3 = subs(dy,t,t0)==dy0;
eq4 = subs(dy,t,tf)==dyf;
eq5 = subs(ddy,t,t0)==ddy0;
eq6 = subs(ddy,t,tf)==ddyf;

%solve for the unknowns
[a0,a1,a2,a3,a4,a5]=solve(eq1,eq2,eq3,eq4,eq5,eq6);

%set up time vector
t = linspace(t0,tf,n);

%use subs to replace all unknowns and time in the polynomials
y   = double(subs(y));
dy  = double(subs(dy));
ddy = double(subs(ddy));
end
```

0.2 Problem 2

Problem 2. [40 points]



Overview of 3-DOF manipulator **Task-space trajectory (rectangular)**

You are to generate the motion trajectory for the three degree of freedom manipulator shown in the above figure such that the manipulator's task frame (origin) moves between the four points that describe a square in space (see Figure). The forward and inverse kinematics are given below.

Forward kinematics:	*Inverse kinematics:*
$x_e = q_3 \cos(q_2)$	$q_1 = z_e$
$y_e = q_3 \sin(q_2)$	$q_2 = \tan^{-1}(y_e / x_e)$
$z_e = q_1$	$q_3 = \sqrt{x_e^2 + y_e^2}$

For reference, the homogeneous transformations between the successive link frames (defined in the figure above) are given below. These may be useful when animating your results in Problem 3.

$$T_1^0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_2^1 = \left[\begin{array}{ccc|c} -s_2 & 0 & c_2 & 0 \\ c_2 & 0 & s_2 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_e^2 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & q_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Problem 2. *continued*

Scenario 1: Joint-space trajectory generation

- Calculate the joint space displacements that correspond to the given task-space waypoints $\left[\vec{X}_1, \vec{X}_2, \vec{X}_3, \text{and } \vec{X}_4\right]$ (using the inverse kinematics). The task space Cartesian coordinates for each waypoint can be determined from the adjacent figure.
- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in joint-space, using the joint-space displacements calculated above. The trajectories should follow a quintic polynomial. At each waypoint, the joint-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time.
  - Task space displacements as a function of time (using the forward kinematics to calculate the task space displacements).
  - Task space displacements in 3-D space (using the Matlab `plot3` command).

Scenario 2: Task-space trajectory generation

- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in task-space, using the given task-space waypoints. The trajectories should follow a quintic polynomial. At each waypoint, the task-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time (using the inverse kinematics to calculate the joint-space displacements).
  - Task space displacements as a function of time.
  - Task space displacements in 3-D space (using the Matlab `plot3` command).
- Comment on the differences between the two approaches (advantages and disadvantages).

Figure 3: Problem 2 description

The following 2D diagram illustrates the task space trajectory which is the path that the end effector will travel over.

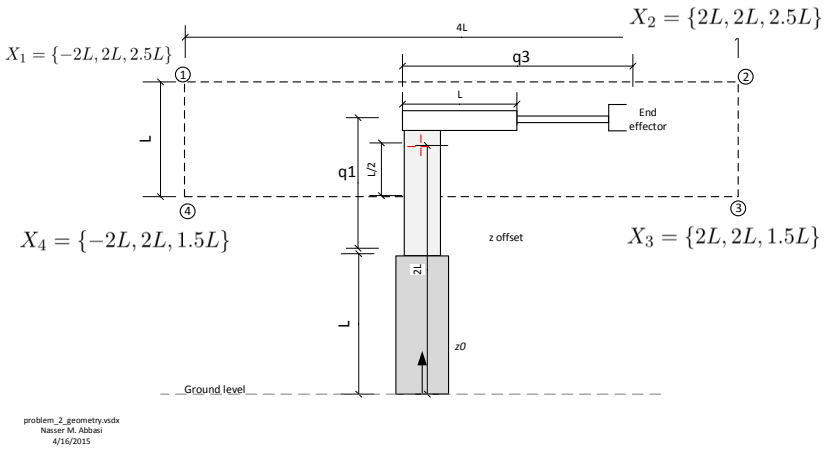


Figure 4: Problem 2 task path

0.2.1 First scenario

PART ONE:

The waypoints have the following coordinate values by inspection from the above diagram

$$\begin{aligned} X_1 &= \{-2L, 2L, 2.5L\} \\ X_2 &= \{2L, 2L, 2.5L\} \\ X_3 &= \{2L, 2L, 1.5L\} \\ X_4 &= \{-2L, 2L, 1.5L\} \end{aligned}$$

Inverse kinematics was used to determine the joint space displacements that corresponds to the above task space. For X_1 this results in

$$\begin{aligned} q_1 &= z_e = 2.5L \\ q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{-2L}\right) = 135^0 \\ q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2} \end{aligned}$$

For X_2

$$\begin{aligned} q_1 &= z_e = 2.5L \\ q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{2L}\right) = 45^0 \\ q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2} \end{aligned}$$

For X_3

$$
\begin{aligned}
q_1 &= z_e = 1.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{2L}\right) = 45^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

And for X_4

$$
\begin{aligned}
q_1 &= z_e = 1.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{-2L}\right) = 135^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

PART TWO

The trajectory in joint space is now generated. The joint space coordinates was found above and illustrated in the following diagram

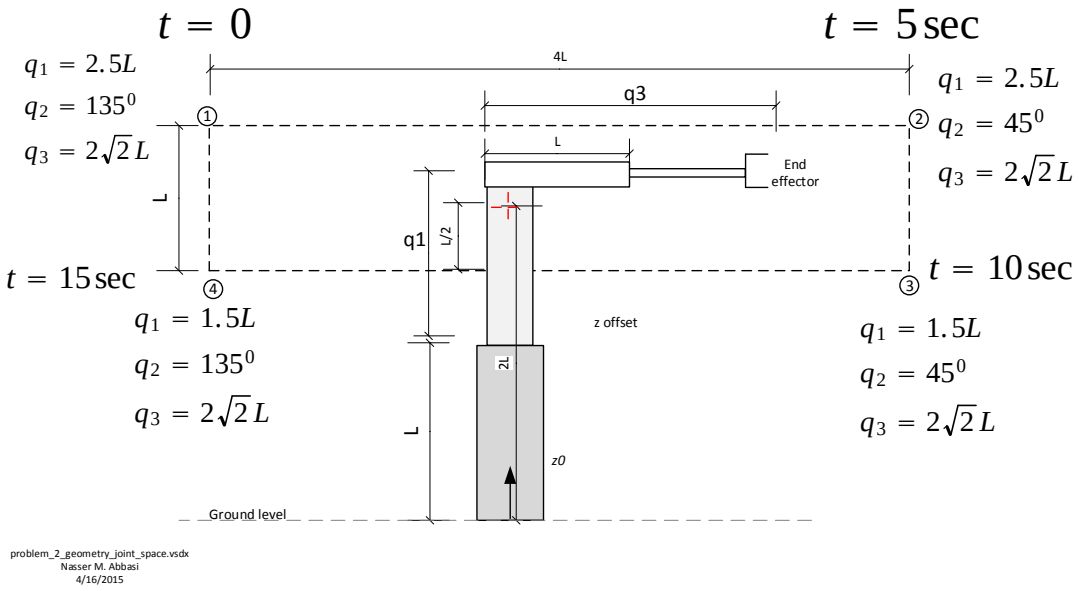


Figure 5: Problem 2 task path expressed in joint coordinates

A quintic polynomial was used with the restriction that at each waypoint $\dot{q} = 0$ and also $\ddot{q} = 0$.

The above was done for each joint space path between two points. There are 3 joints. For each joint the full path was generated. For example, for joint q_1, the polynomial between X_1 and X_2 was found by solving 6 equations in 6 unknowns. The polynomial is defined as

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

The 6 equations are

$$
\begin{aligned}
q(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{q}(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{q}(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
q(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 2.5L \\
\dot{q}(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{q}(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}
$$

Where $t_0 = 0$ and $t_f = 5$. The above 6 equations are solved for $a_0, a_1, a_2, a_3, a_4, a_5$ which now gives the polynomial $q(t)$ in order to obtain the joint coordinate at any time $0 < t < 5$.

$$
\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\
0 & 1 & 2t_f & 3t_f^2 & 4t_0^3 & 5t_f^4 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}
=
\begin{bmatrix} q(t_0) \\ \dot{q}(t_0) \\ \ddot{q}(t_0) \\ q(t_f) \\ \dot{q}(t_f) \\ \ddot{q}(t_f) \end{bmatrix}
=
\begin{bmatrix} 2.5L \\ 0 \\ 0 \\ 2.5L \\ 0 \\ 0 \end{bmatrix}
$$

Similarly, these 6 equations are solved for the next segment between X_2, X_3

$$
\begin{aligned}
q(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{q}(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{q}(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
q(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 1.5L \\
\dot{q}(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{q}(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}
$$

Where in the above $t_0 = 5$ and $t_f = 10$.

The above was done for all the segments. The same process was repeated for joints q_2 and q_3.

A Matlab program listed below was written to implement the above. The result generated is given below the source code listing.

```
function  nma_HW4_problem_2_part1()
%nma_HW4_problem_2_part1()
%This function implements problem 2, HW 4, ME 739, scenario ONE
%by Nasser M. Abbasi
%
close all;

L=4;
%generate the joint space (q) displacements, velocity and acceleration
%the value of q_1  at each X1,X2,X3,X4 have been determined from the
%inverse kinmatics as shown in the HW report above. These are now used
```

```
%to generate the polynomial

%generate polynomial for q1 trajectory and position, speed and acc. plots
figure;
[t1,q1]=process_q(2.5*L,2.5*L,1.5*L,1.5*L,'q1',1.4*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate polynomial for q2 trajectory and position, speed and acc. plots
figure;
[t2,q2]=process_q(135*pi/180,45*pi/180,45*pi/180,135*pi/180,...
    'q2',40*pi/180,155*pi/180,'angle(rad)','rad/sec','rad/sec^2');

%generate polynomial for q3 trajectory and position, speed and acc. plots
figure;
z=2*L*sqrt(2);
[t3,q3]=process_q(z,z,z,z,'q3',0,1.2*2*L*sqrt(2),'meter','m/s','m/s^2');

%generate the task space X displacement using the forward kinematics
xe = q3.*cos(q2);
ye = q3.*sin(q2);
ze = q1;

%generate task space displacements
figure;
h1 = subplot(3,1,1);
plot(t1,xe);
title(h1,{'Task space displacements','Xe'});
xlabel(h1,'time (sec)');
ylabel(h1,'meter');
axis(h1,[0 20 -2.2*L 2.2*L]);

h2 = subplot(3,1,2);
plot(t2,ye);
title(h2,'Ye');
xlabel(h2,'time (sec)');
ylabel(h2,'meter');
axis(h2,[0 20 1.8*L 3*L]);

h3 = subplot(3,1,3);
plot(t3,ze);
title(h3,'Ze');
xlabel(h3,'time (sec)');
ylabel(h3,'meter');
axis(h3,[0 20 1.4*L 2.6*L]);

%generate 3D plot of the task space trajectory
figure;
plot3(xe,ye,ze);
hold on;
plot3(-2*L,2*L,2.5*L,'o');
text(-2*L,2*L,2.5*L,'X1');

plot3(2*L,2*L,2.5*L,'o');
text(2*L,2*L,2.5*L,'X2');
```

```
plot3(2*L,2*L,1.5*L,'o');
text(2*L,2*L,1.5*L,'X3');

plot3(-2*L,2*L,1.5*L,'o');
text(-2*L,2*L,1.5*L,'X4');

plot3(0,2*L,2*L,'+');
text(.1*L,2*L,2.1*L,'center');

title('3D task space displacement');
xlabel('X'); ylabel('Y'); zlabel('Z');
zlim([0 3*L]);

end
%-----------------------
function [t,q]=process_q(x1,x2,x3,x4,the_name,lim0,lim1,y1,y2,y3)
%
%Function to generate joint space trajectory for problem 2 for specific
%joint q
%x1: first point coordinate in this joint  space
%x2: second point coordinate in this joint  space
%x3: third point coordinate in this joint  space
%x4: fourth point coordinate in this joint  space
%the_name: title to put on the plot, for example 'q1' or 'q2' etc..
%lim0 and lim1 are the y-axis limits to use for the plot
%y1: position y label
%y2: speed y label
%y3: acceleration y label

%RETURNS
%q: which is vector of q joint coordinates in joint space over
%the full time space to be used later to generate the task space
%trajectory using forward kinemetics
%
%t: the corresponding time vector

%call gen_path to obtain the polynomial coefficients
a              = gen_path(x1,x2,0,5);
[q1,dq,ddq,t1] = find_q(0,5,a); %use the coefficients to make polynomials
[h1,h2,h3]     = make_first_plot(t1,q1,dq,ddq);

title(h1,{the_name,'position'});
title(h2,'velocity');
title(h3,'acceleration');
xlabel(h1,'time (sec)'); ylabel(h1,y1);
xlabel(h2,'time (sec)'); ylabel(h2,y2);
xlabel(h3,'time (sec)'); ylabel(h3,y3);

a              = gen_path(x2,x3,5,10);
[q2,dq,ddq,t2] = find_q(5,10,a);
make_other_plot(t2,q2,dq,ddq,h1,h2,h3);
```

```
a              = gen_path(x3,x4,10,15);
[q3,dq,ddq,t3] = find_q(10,15,a);
make_other_plot(t3,q3,dq,ddq,h1,h2,h3);

a              = gen_path(x4,x1,15,20);
[q4,dq,ddq,t4] = find_q(15,20,a);
make_other_plot(t4,q4,dq,ddq,h1,h2,h3);

axis(h1,[0 20 lim0 lim1]);
q=[q1 q2 q3 q4];
t=[t1 t2 t3 t4];
end

%-----------------------------------------------
function a = gen_path(q0,qf,t0,tf)
%this function generates the polynomial coefficients by solving for the
%constraints given

dq0  = 0;
dqf  = 0;
ddq0 = 0;
ddqf = 0;

C = [ 1  t0  t0^2  t0^3    t0^4     t0^5;
      0  1   2*t0  3*t0^2  4*t0^3   5*t0^4;
      0  0   2     6*t0    12*t0^2  20*t0^3;
      1  tf  tf^2  tf^3    tf^4     tf^5;
      0  1   2*tf  3*tf^2  4*tf^3   5*tf^4;
      0  0   2     6*tf    12*tf^2  20*tf^3];

q  = [q0 dq0 ddq0 qf dqf ddqf]';
a  = C\q;

end

%-------------------------------------------------
function [q,dq,ddq,t]=find_q(t0,tf,a)
%This function takes the coefficients of the polynomial and
%return back the polynomials for position, speed and acceleration
a0  = a(1); a1 = a(2); a2 = a(3); a3 = a(4); a4 = a(5); a5 = a(6);
t   = linspace(t0,tf,1000);
q   = a0   + a1*t   + a2*t.^2    + a3*t.^3    + a4*t.^4     + a5*t.^5;
dq  = a1   + 2*a2*t + 3*a3*t.^2  + 4*a4*t.^3  + 5*a5*t.^4;
ddq = 2*a2 + 6*a3*t + 12*a4*t.^2 + 20*a5*t.^3;

end

%-----------------------------------------------------
function [h1,h2,h3]=make_first_plot(t,q,dq,ddq)
%this function plots the position in joint space
h1 = subplot(3,1,1);
plot(t,q)

h2 = subplot(3,1,2);
```

```
plot(t,dq);

h3 = subplot(3,1,3);
plot(t,ddq);
end

%-----------------------------------------------
function make_other_plot(t,q,dq,ddq,h1,h2,h3)
%this function plots the speed and acceleration trajetory in joint space
hold(h1,'on');
subplot(3,1,1);
plot(t,q);
plot(t(1),q(1),'o');

hold(h2,'on');
subplot(3,1,2);
plot(t,dq);
plot(t(1),dq(1),'o');

hold(h3,'on');
subplot(3,1,3);
plot(t,ddq);
plot(t(1),ddq(1),'o');
end
```

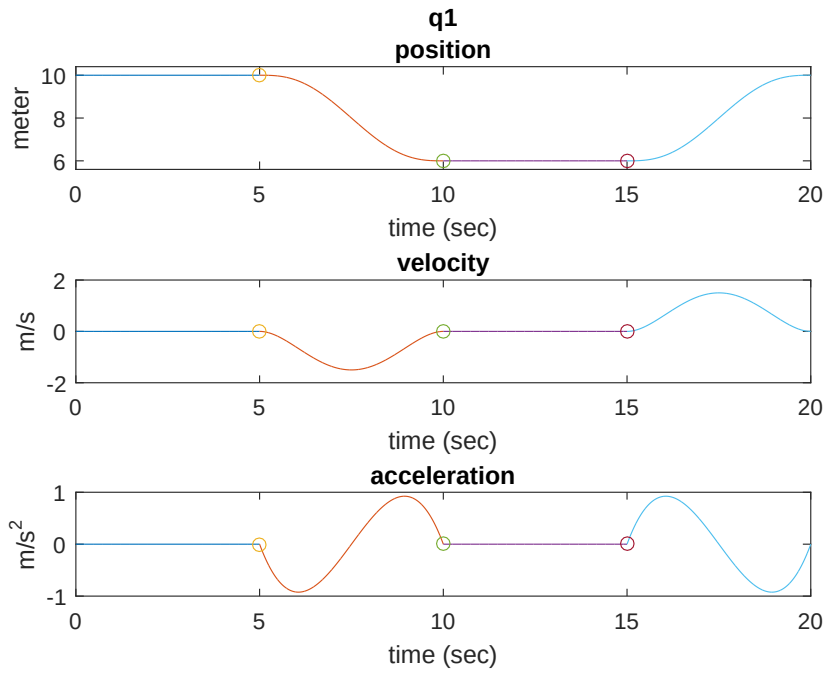


Figure 6: Problem 2, part 1, q_1 joint space trajectory

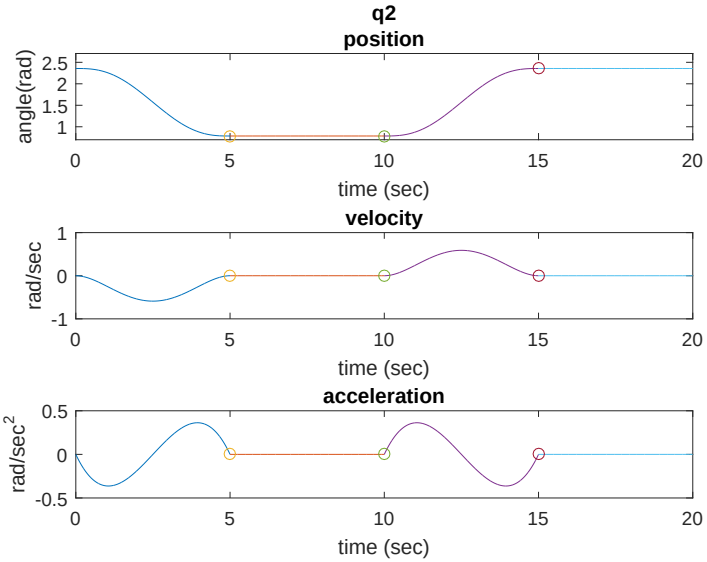


Figure 7: Problem 2, part 1, q_2 joint space trajectory

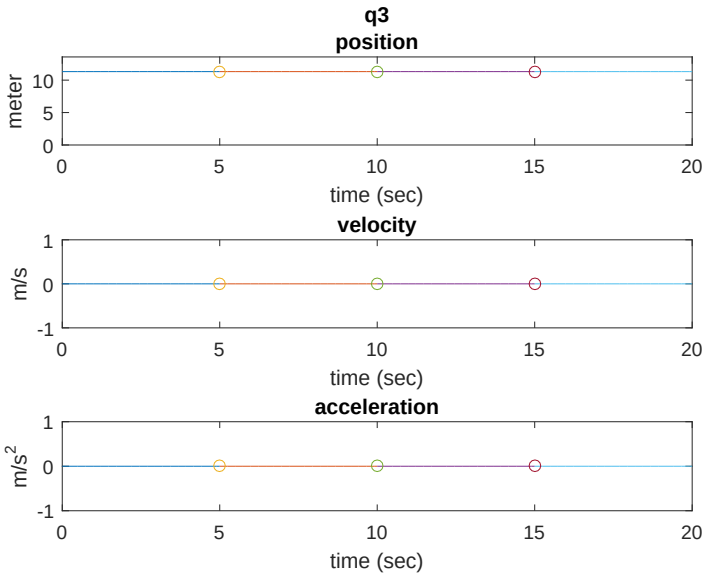


Figure 8: Problem 2, part 1, task space x_e, y_e, z_e trajectory

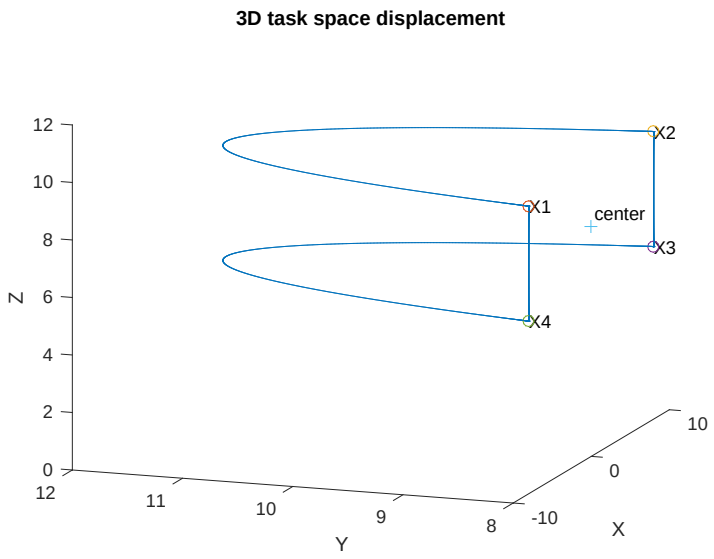


Figure 9: Problem 2, part 1, 3D plot of task space trajectory

0.2.2 Second scenario

The four part trajectory was generated again, using the same constraints as in first scenario, but now the polynomial was generated in task space as required for this part. The polynomial between waypoint X_1 and X_2 was found by solving 6 equations in 6 unknowns. The polynomial is defined for each coordinate x_e, y_e, z_e. For example, for x_e

$$x_e(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

The 6 equations are

$$\begin{aligned}
x_e(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{x}_e(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{x}_e(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
x_e(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 2.5L \\
\dot{x}_e(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{x}_e(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}$$

Where $t_0 = 0$ and $t_f = 5$. The above 6 equations are solved for $a_0, a_1, a_2, a_3, a_4, a_5$ which gives the polynomial $x_e(t)$ used obtain the joint coordinate at any time $0 < t < 5$.

$$\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\
0 & 1 & 2t_f & 3t_f^2 & 4t_0^3 & 5t_f^4 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}
=
\begin{bmatrix} x_e(t_0) \\ \dot{x}_e(t_0) \\ \ddot{x}_e(t_0) \\ x_e(t_f) \\ \dot{x}_e(t_f) \\ \ddot{x}_e(t_f) \end{bmatrix}
=
\begin{bmatrix} 2.5L \\ 0 \\ 0 \\ 2.5L \\ 0 \\ 0 \end{bmatrix}$$

Similarly, these 6 equations were solved for the next segment between waypoint X_2, X_3. The same process was repeated for y_e and z_e.

The following gives the new Matlab implementation with the plots generated. Discussion on differences between the two approaches is given at the end.

```
function  nma_HW4_problem_2_part2()
%nma_HW4_problem_2_part2()
%This function implements problem 2, HW 4, ME 739, scenario TWO
%by Nasser M. Abbasi
%
close all;

L=4;
%generate the task space (X,Y,Z) displacements, velocity and acceleration
%the value of x,y,z  at each X1,X2,X3,X4 are given in the problem from
%the diagram shown.

%generate polynomial for x-coodinate of each X waypoint trajectory
figure;
[t1,x_coordinate]=process_X(-2*L,2*L,2*L,-2*L,'x',-2.6*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate polynomial for y-coordinate of each X waypoint trajectory
figure;
[t2,y_coordinate]=process_X(2*L,2*L,2*L,2*L,...
    'y',1.9*L,2.1*L,'meter','m/sec','m/sec^2');

%generate polynomial for z-coordinate of each X waypoint trajectory
figure;
[t3,z_coordinate]=process_X(2.5*L,2.5*L,1.5*L,1.5*L,'z',1.4*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate the corresponding joint space q displacement, speed and
%acceleration from the above using forward kinematics
q1 = z_coordinate;
q2 = atan2(y_coordinate,x_coordinate);
q3 = sqrt(x_coordinate.^2+y_coordinate.^2);

%generate joint space displacements
figure;
h1 = subplot(3,1,1);
plot(t1,q1);
title(h1,{'Joint space displacements','q1'});
xlabel(h1,'time (sec)');
ylabel(h1,'meter');
axis(h1,[0 20 0.9*L 2.8*L]);

h2 = subplot(3,1,2);
plot(t2,q2);
title(h2,'q2');
xlabel(h2,'time (sec)');
ylabel(h2,'radian');
axis(h2,[0 20 30*pi/180 140*pi/180]);
```

```
h3 = subplot(3,1,3);
plot(t3,q3);
title(h3,'q3');
xlabel(h3,'time (sec)');
ylabel(h3,'meter');
axis(h3,[0 20 1.9*L 2.9*L]);

%generate 3D plot of the task space trajectory
figure;
plot3(x_coordinate,y_coordinate,z_coordinate);
hold on;
plot3(-2*L,2*L,2.5*L,'o');
text(-2*L,2*L,2.5*L,'X1');

plot3(2*L,2*L,2.5*L,'o');
text(2*L,2*L,2.5*L,'X2');

plot3(2*L,2*L,1.5*L,'o');
text(2*L,2*L,1.5*L,'X3');

plot3(-2*L,2*L,1.5*L,'o');
text(-2*L,2*L,1.5*L,'X4');

plot3(0,2*L,2*L,'+');
text(.1*L,2*L,1.6*L,'center');

title('3D task space displacement');
xlabel('X'); ylabel('Y'); zlabel('Z');
zlim([0 3*L]);

end
%------------------------
function [t,x]=process_X(x1,x2,x3,x4,the_name,lim0,lim1,y1,y2,y3)
%
%Function to generate task space trajectory for problem 2
%x1: first point coordinate in this task  space
%x2: second point coordinate in this task  space
%x3: third point coordinate in this task  space
%x4: fourth point coordinate in this task  space
%the_name: title to put on the plot, for example 'x' or 'y' or 'z'
%lim0 and lim1 are the y-axis limits to use for the plot
%y1: position y label
%y2: speed y label
%y3: acceleration y label

%RETURNS
%q: which is vector of q joint coordinates in joint space over
%the full time space to be used later to generate the task space
%trajectory using forward kinemetics
%
%t: the corresponding time vector

%call gen_path to obtain the polynomial coefficients
```

```
a              = gen_path(x1,x2,0,5);
[x_segment_1,dx,ddx,t1] = find_x(0,5,a); %use the coefficients to make polynomials
[h1,h2,h3]     = make_first_plot(t1,x_segment_1,dx,ddx);

title(h1,{the_name,'position'});
title(h2,'velocity');
title(h3,'acceleration');
xlabel(h1,'time (sec)'); ylabel(h1,y1);
xlabel(h2,'time (sec)'); ylabel(h2,y2);
xlabel(h3,'time (sec)'); ylabel(h3,y3);

a              = gen_path(x2,x3,5,10);
[x_segment_2,dx,ddx,t2] = find_x(5,10,a);
make_other_plot(t2,x_segment_2,dx,ddx,h1,h2,h3);


a              = gen_path(x3,x4,10,15);
[x_segment_3,dx,ddx,t3] = find_x(10,15,a);
make_other_plot(t3,x_segment_3,dx,ddx,h1,h2,h3);

a              = gen_path(x4,x1,15,20);
[x_segment_4,dx,ddx,t4] = find_x(15,20,a);
make_other_plot(t4,x_segment_4,dx,ddx,h1,h2,h3);

axis(h1,[0 20 lim0 lim1]);
x=[x_segment_1 x_segment_2 x_segment_3 x_segment_4];
t=[t1 t2 t3 t4];
end

%-----------------------------------------------
function a = gen_path(x0,xf,t0,tf)
%this function generates the polynomial coefficients by solving for the
%constraints given

dx0  = 0;
dxf  = 0;
ddx0 = 0;
ddxf = 0;

C = [ 1  t0  t0^2  t0^3    t0^4     t0^5;
      0  1   2*t0  3*t0^2  4*t0^3   5*t0^4;
      0  0   2     6*t0    12*t0^2  20*t0^3;
      1  tf  tf^2  tf^3    tf^4     tf^5;
      0  1   2*tf  3*tf^2  4*tf^3   5*tf^4;
      0  0   2     6*tf    12*tf^2  20*tf^3];

x  = [x0 dx0 ddx0 xf dxf ddxf]';
a  = C\x;

end

%-------------------------------------------------
function [x,dx,ddx,t]=find_x(t0,tf,a)
%This function takes the coefficients of the polynomial and
```

```
%return back the polynomials for position, speed and acceleration
a0  = a(1); a1 = a(2); a2 = a(3); a3 = a(4); a4 = a(5); a5 = a(6);
t   = linspace(t0,tf,1000);
x   = a0   + a1*t   + a2*t.^2    + a3*t.^3    + a4*t.^4     + a5*t.^5;
dx  = a1   + 2*a2*t + 3*a3*t.^2  + 4*a4*t.^3  + 5*a5*t.^4;
ddx = 2*a2 + 6*a3*t + 12*a4*t.^2 + 20*a5*t.^3;

end

%-----------------------------------------------------
function [h1,h2,h3]=make_first_plot(t,x,dx,ddx)
%this function plots the position in task space
h1 = subplot(3,1,1);
plot(t,x)

h2 = subplot(3,1,2);
plot(t,dx);

h3 = subplot(3,1,3);
plot(t,ddx);
end

%-----------------------------------------------
function make_other_plot(t,x,dx,ddx,h1,h2,h3)
%this function plots the speed and acceleration trajetory in task space
hold(h1,'on');
subplot(3,1,1);
plot(t,x);
plot(t(1),x(1),'o');

hold(h2,'on');
subplot(3,1,2);
plot(t,dx);
plot(t(1),dx(1),'o');

hold(h3,'on');
subplot(3,1,3);
plot(t,ddx);
plot(t(1),ddx(1),'o');
end
```

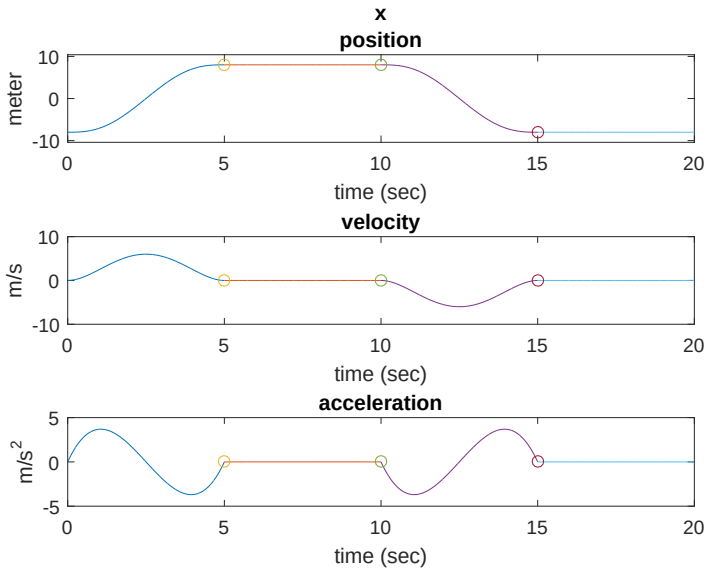


Figure 10: Problem 2, part 2, x_e task space trajectory

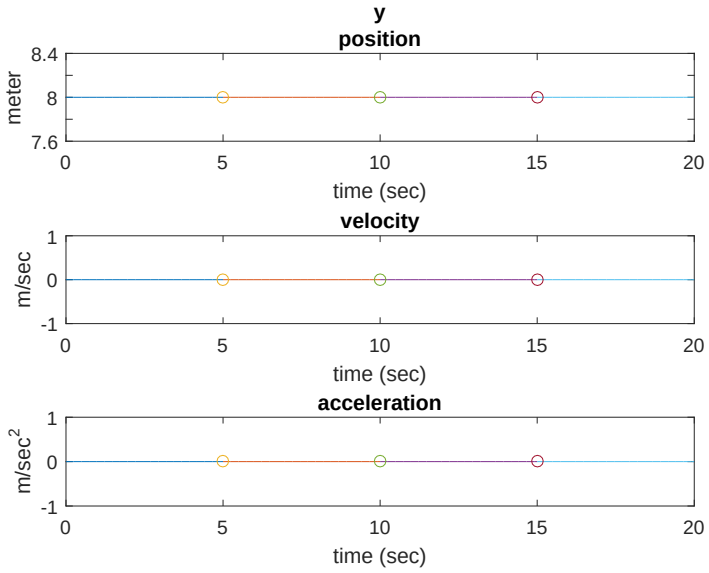


Figure 11: Problem 2, part 2, y_e task space trajectory

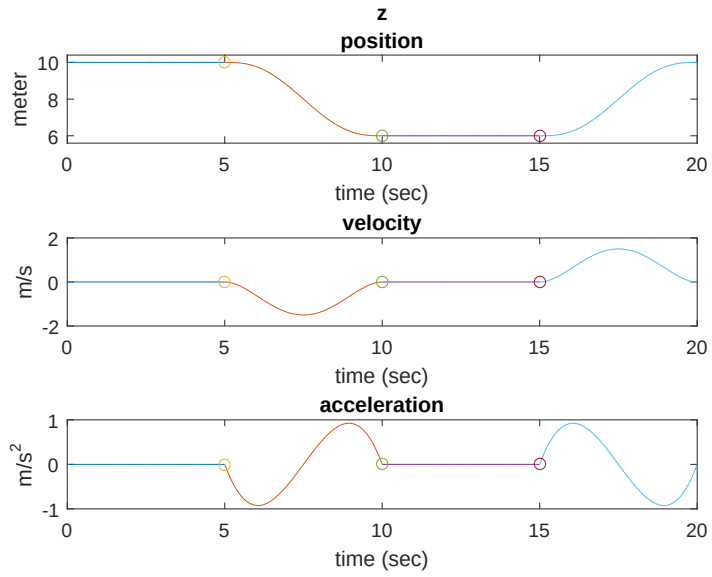


Figure 12: Problem 2, part 2, z_e task space trajectory

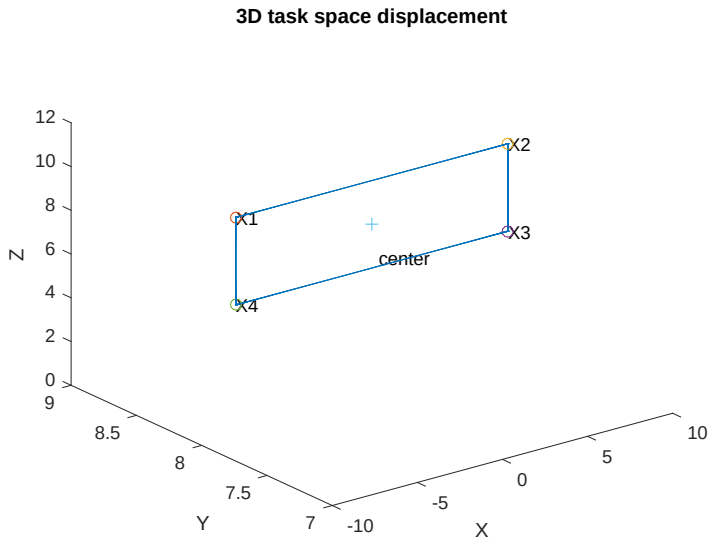


Figure 13: Problem 2, part 2, 3D plot of task space trajectory

0.2.3 discussion

The following diagram gives an overview of the difference of the algorithm used for first and second scenarios.

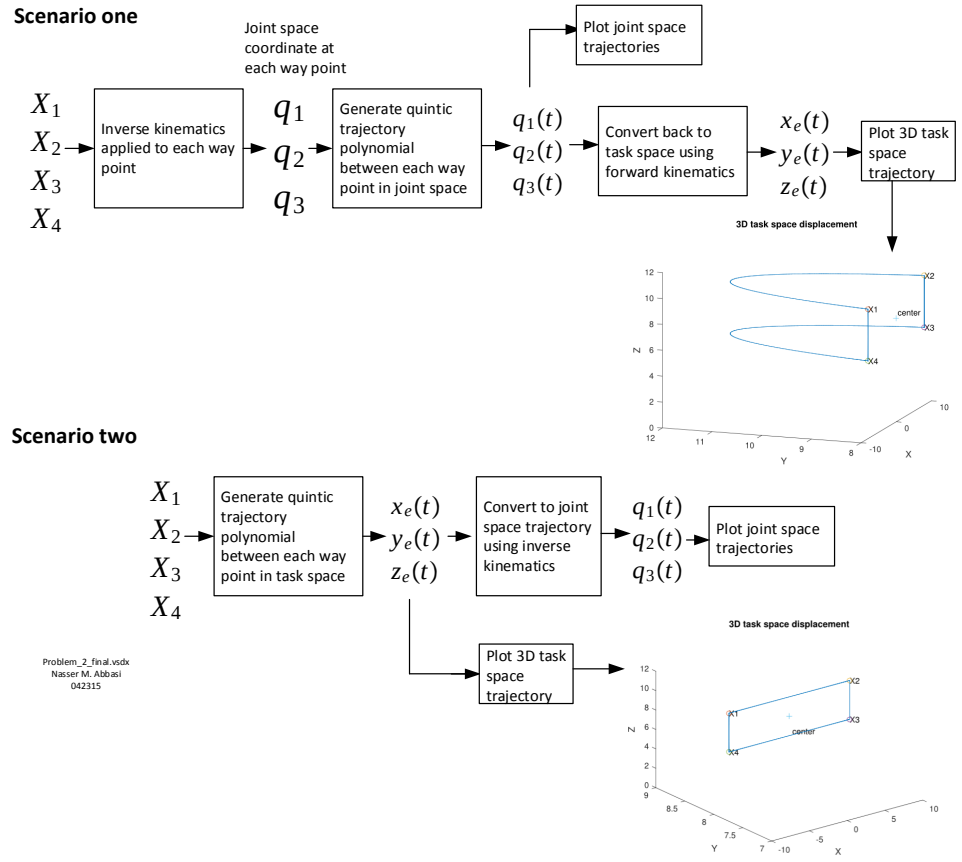


Figure 14: Problem 2 algorithm difference scenario one and two

There are two main differences observed between the two methods.

1. In the first scenario, joint space trajectories $q_1(t)$ and $q_2(t)$ were the same as in the second scenario, but $q_3(t)$ was not the same.

   In first scenario, $q_3(t)$ was constant with value $2\sqrt{2}\,L$ for the whole path. This is becuase at each way point q_3 had the same value $2\sqrt{2}\,L$, therefore the polynomial generated was straight line connecting all the way points. In second scenario, $q_3(t)$ was not constant, since $q_3(t) = \sqrt{x_e(t)^2 + y_e(t)^2}$ and $x_e(t)$ polynomial was not constant (even though $y_e(t)$ was constant over the whole path).

2. In the first scenario, $y_e(t)$ was generated by inverse kinematics using $y_e(t) = q_3(t)\sin(q_2(t))$ and even though $q_3(t)$ was constant, $q_2(t)$ was not. Hence the path along the y direction is task space was changing as can be seen from the above plot. In the second scenario, since the y was fixed at each way point, the trajectory generated in task space shows that $y(t)$ is not changing. This is why the 3D plot for the second scenario do not show the same curved path in the y dimension as in the first scenario.

The result from first scenario seems to be more realistic 3D task space path that the robot arm end effector would take. Computationally, there is little difference between the two scenarios.

0.3 Problem 3

Problem 3. [40 points]

For the manipulator described in Problem 2

- Use a parabolic well potential to define an attractive field applied to the origin of frame {e}. Use Matlab to implement a gradient descent algorithm to find a path from the specified initial configuration to the specified final configuration given below.

  Initial configuration: $q_0 = \begin{bmatrix} L & 0 & L \end{bmatrix}^T$

  Final configuration: $q_f = \begin{bmatrix} 2L & \frac{1}{2}\pi & 2L \end{bmatrix}^T$

- Animate your results using the Matlab rendering functions posted to the Learn@UW course page.

In general, you can vary ζ_i (the scaling factor applied to the attractive force, f_i) to modify the resulting path. In this case, the forces are applied at only one point so there is only one value of ζ. You can also vary the gradient descent scaling factor, α. Larger values of α can speed up the gradient descent algorithm, but can also cause numerical instabilities in the solution. In addition, you can use different values of α (α_i for joint i) to scale the joint torques and thus modify the path solution.

Gradient descent:

$$\begin{bmatrix} \textit{while } \left\| q^k - q_f \right\| > \varepsilon \\ \qquad q^{k+1} = q^k + \alpha \nabla U_\tau \quad \text{where } \nabla U_\tau = \dfrac{\tau\left(q^k\right)}{\left\| \tau\left(q^k\right) \right\|} \text{ and } \alpha = \begin{bmatrix} \alpha_1 & & 0 \\ & \ddots & \\ 0 & & \alpha_n \end{bmatrix} \\ \qquad k = k+1 \end{bmatrix}$$

- Vary α_i and comment on the resulting solution behavior (both in regards to the shape of the path and the numerical stability of the solution.

Figure 15: Problem 3 description

In this problem we need to determine only the attractive virtual force on origin of end effector frame {e}. The repulsive forces are not involved. The attractive virtual forces are approximated with a parabolic well potential. We have only one location where the force is applied to (which is the origin of frame {e}, which is the end effector). The first step is to determine the virtual force from the gradient of the parabolic potential field given by

$$\begin{aligned} \mathbf{F}_e &= -\nabla U_e \\ &= -\zeta \left(\mathbf{o}_e \left(q_{current} \right) - \mathbf{o}_e \left(q_f \right) \right) \end{aligned}$$

Where in the above $\mathbf{o}_e(q)$ is the position vector of the origin of frame {e} at current configuration q and $\mathbf{o}_e(q_f)$ is position vector of the origin of frame {e} at final configuration q_f. Hence $\mathbf{o}_e(q_f)$ is constant position vector and only $\mathbf{o}_e(q)$ will change as the robot arm moves.

We are given the final configuration as $q_f = \left[2L, \frac{\pi}{2}, 2L\right]^T$, and we now use this to obtain $\mathbf{o}_e(q_f)$. We first need to obtain T_e^0 which we obtained forward kinematics given in problem 2. Hence

$$
\begin{aligned}
T_e^0 &= T_1^0 T_2^1 T_e^2 \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\sin q_2 & 0 & \cos q_2 & 0 \\ \cos q_2 & 0 & \sin q_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & q_3 \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & q_3 \sin q_2 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Therefore, $\mathbf{o}_e(q)$ from the above is the last column. Hence

$$
\mathbf{o}_e(q_{current}) = \begin{bmatrix} q_3 \cos q_2 \\ q_3 \sin q_2 \\ q_1 \end{bmatrix}
$$

We now replace the values of q_1, q_2, q_3 in the above, with the final configuration values given

$$
\mathbf{o}_e(q_{final}) = \begin{bmatrix} 2L \cos \frac{\pi}{2} \\ 2L \sin \frac{\pi}{2} \\ 2L \end{bmatrix} = \begin{bmatrix} 0 \\ 2L \\ 2L \end{bmatrix}
$$

And the initial configuration (at time $t = 0$) is

$$
\mathbf{o}_e(q_{initial}) = \begin{bmatrix} L \cos 0 \\ L \sin 0 \\ L \end{bmatrix} = \begin{bmatrix} L \\ 0 \\ L \end{bmatrix}
$$

Using the above, we need determine the virtual force F_e

$$
\begin{aligned}
\mathbf{F}_e &= -\nabla U_e \\
&= -\zeta_e \left(\mathbf{o}_e(q_{current}) - \mathbf{o}_e(q_f) \right) \\
&= -\zeta_e \left(\begin{bmatrix} q_3 \cos q_2 \\ q_3 \sin q_2 \\ q_1 \end{bmatrix} - \begin{bmatrix} 0 \\ 2L \\ 2L \end{bmatrix} \right) \\
&= -\zeta_e \begin{bmatrix} q_3 \cos(q_2) \\ q_3 \sin q_2 - 2L \\ q_1 - 2L \end{bmatrix}
\end{aligned}
$$

In the above, all the q_i variables are the current joint variables at the current time. In simulation, there will change with time. Now that we found the virtual force, we convert it to joint toque using the linear velocity Jacobian. We obtain J_{v_e} by direct differentiation method

$$J_{v_e} = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} \end{bmatrix}$$

From problem 2, we are given that

$$\begin{aligned} x_e &= q_3 \cos q_2 \\ y_e &= q_3 \sin q_2 \\ z_e &= q_1 \end{aligned}$$

Hence J_{v_e} becomes

$$J_{v_e} = \begin{bmatrix} \partial \frac{q_3 \cos q_2}{\partial q_1} & \partial \frac{q_3 \cos q_2}{\partial q_2} & \partial \frac{q_3 \cos q_2}{\partial q_3} \\ \partial \frac{q_3 \sin q_2}{\partial q_1} & \partial \frac{q_3 \sin q_2}{\partial q_2} & \partial \frac{q_3 \sin q_2}{\partial q_3} \\ \frac{\partial q_1}{\partial q_1} & \frac{\partial q_1}{\partial q_2} & \frac{\partial q_1}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & -q_3 \sin q_2 & \cos q_2 \\ 0 & q_3 \cos q_2 & \sin q_2 \\ 1 & 0 & 0 \end{bmatrix}$$

Therefore, using the duality relation that $\tau = \sum J_{v_i}^T F_i$ and since we have forces on one joint only, this simplifies to $\tau = J_{v_e}^T F_e$

$$\begin{aligned} \boldsymbol{\phi} &= -\zeta_e \begin{bmatrix} 0 & -q_3 \sin q_2 & \cos q_2 \\ 0 & q_3 \cos q_2 & \sin q_2 \\ 1 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} q_3 \cos(q_2) \\ q_3 \sin q_2 - 2L \\ q_1 - 2L \end{bmatrix} \\ &= -\zeta_e \begin{bmatrix} q_1 - 2L \\ q_3 (\cos q_2)(q_3 \sin q_2 - 2L) - q_3^2 \cos q_2 \sin q_2 \\ q_3 \cos^2 q_2 + (\sin q_2)(q_3 \sin q_2 - 2L) \end{bmatrix} \end{aligned} \tag{1}$$

The above is the actual torque/force applied at the joints 1,2 and 3. Therefore, the force applied to joint 1 is $q_1 - 2L$ and the torque to apply to joint 2 is $q_3 (\cos q_2)(q_3 \sin q_2 - 2L) - q_3^2 \cos q_2 \sin q_2$ and the force to apply to end effector joint is $q_3 \cos^2 q_2 + (\sin q_2)(q_3 \sin q_2 - 2L)$.

For example, at initial configuration, where $q_{initial} = [L, 0, L]^T$ we find

$$\boldsymbol{\phi}_{initial} = -\zeta_e \begin{bmatrix} -L \\ -2Lq_3 \\ L \end{bmatrix}$$

And at final configuration where $q_{final} = q_f = \left[2L, \frac{\pi}{2}, 2L\right]^T$we find

$$\boldsymbol{\phi}_{initial} = -\zeta_e \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Which is what we would expect. At the final configuration, the applied forces at the joints should vanish since we have arrived at the final configuration.

Now that we have equation (1), we can use it to implement the gradient descent algorithm in order to obtain the sequence of q_i positions to use for the animation.

0.3.1 Part one

In this part, a single α value was used for all the joints. This value was changed in order to observe the affect. In second part below, different α for each joint will be used.

In this part, and in all runs, ζ was set to 100. This is the attractive force scaling value.

Below is the source code written to implement this part of the problem. The following diagram shows the initial configuration for one example run.

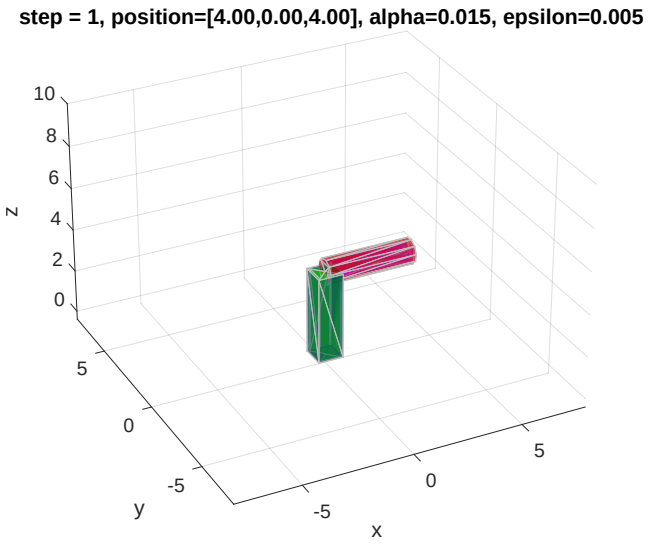


Figure 16: Initial configuration, problem 3

And the following diagram shows the final configuration reached. It shows that it took 535 steps to reach the final configuration using $\alpha = 0.015$ and $\epsilon = 0.005$

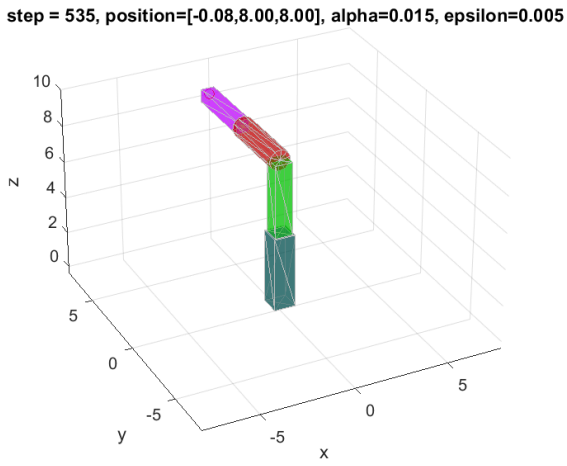


Figure 17: Final configuration, problem 3

As α was made smaller the convergence became slower, but the animation became more accurate and ran more smoothly. As the end effector came very close to the target, more vibration around that region started to show as the end effector oscillated around the target point as it converged to the exact target and the error became smaller and smaller.

To run the program, the command is `nma_HW4_problem_3`

```
function  nma_HW4_problem_3()
%function nma_HW4_problem_3()
%This function implements path planning for the 3 links robot arm in
%HW4, ME 739, problem 3, spring 2015, Univ. Wiscosin, Madison.
%
%This version is for part 1, which uses one alpha for all joints
%
%Parabolic attractive field is used to model the attractive virtual
%force on the end effector. The linear velocity Jacobian is used to
%map this force to torque forces at the three joints.
%Then gradient descent is used to obtain the joint coordinates sequence
%which is then used to simulate the motion.
%
%By Nasser M. Abbasi

close all; clear all;
L        = 4;  %length of link 1 and 3
%epsilon  = 0.005; %error tolerance
%alpha    = 0.015; %smaller value, slows down convergence but more accurate

epsilon  = 0.05; %error tolerance
zeta     = 100;  %attractive force scaling
alpha    = 0.05; %smaller value, slows down convergence but more accurate

q        = [L;0;L];  %initial joint configuration
qf       = [2*L;pi/2;2*L]; %find joint configuration
```

```
maxIter  = 1200;  %max iterations allowed

Q            = zeros(3,maxIter);  %where to save the sequence of joint q's
k            = 1;
keep_running = true;

%start of gradient descemt

while keep_running
    Q(:,k) = q;
    tau = -zeta*[q(1)-2*L;
        q(3)*cos(q(2))*(q(3)*sin(q(2))-2*L)-q(3)^2*cos(q(2))*sin(q(2));
        q(3)*(cos(q(2)))^2+sin(q(2))*(q(3)*sin(q(2))-2*L)
        ];
    q      = q + alpha*tau/norm(tau);
    if k+1>maxIter || norm( q-qf ) <= epsilon
        keep_running = false;
    else
        k = k +1;
    end
end

DO_MOVIE = false; %make true to generate movie frames
frameNumber = 0;

f_handle    = 1;
axis_limits = L*[-2 2 -2 2 -0.1 2.5];
render_view = [-.4 -.8 .5]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective % turns on perspective

% link 0 rendering initialization
r0 = L/5; sides0 = 4; axis0 = 3; norm_L0 = -1.0;
linkColor0 = [0 .3 .3]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L,r0,sides0,axis0,norm_L0,linkColor0,...
    plotFrame0,f_handle);


% link 1 rendering initialization
r1 = L/6; sides1 = 4; axis1 = 3; norm_L1 = 1.0;
linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2 = L/7; sides2 = 10; axis2 = 3; norm_L2 = -1.0;
linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3 = L/8; sides3 = 4; axis3 = 1; norm_L3 = 1.0;
linkColor2 = [0.75 0 1]; plotFrame2 = 0;
d3 =  CreateLinkRendering(L,r3,sides3,axis3,norm_L3,linkColor2,...
```

```
    plotFrame2,f_handle);

T00 = [1  0  0  0
    0  1  0  0
    0  0  1  0
    0  0   0  1];
UpdateLink(d1,T00);

for i = 1:k
    % Update frames
    q1=Q(1,i); q2=Q(2,i); q3=Q(3,i);
    T01 = [1  0  0  0
           0  1  0  0
           0  0  1  q1
           0  0   0  1];

    T12 = [-sin(q2)  0   cos(q2)   0
           cos(q2)   0   sin(q2)   0
           0         1   0         0
           0         0   0         1];

    T2e = [0   1  0  0
           0   0  1  0
           1   0  0  q3
           0   0  0  1];

    T02 = T01*T12;
    T0e = T02*T2e;

    UpdateLink(d1,T01);
    UpdateLink(d2,T02);
    UpdateLink(d3,T0e);

    title(sprintf('step = %d, position=[%3.2f,%3.2f,%3.2f], alpha=%3.3f, epsilon=%3.3f',i,...
        q3*cos(q2),q3*sin(q2),q1,alpha,epsilon));
    %if i == 1;  %pause at start of simulation rendering
    %    pause;
    %end

    if DO_MOVIE
        frameNumber = frameNumber+1;
        I = getframe(gcf);
        imwrite(I.cdata, sprintf('frame%d.png',frameNumber));
    end
    hold on;
    plot3(0,2*L,2*L,'marker','o','color','r');
    drawnow;
    hold off;
    pause(0.01);
end
end
```

0.3.2 Part two

In this part, different α_i values were used for each joint. The code to implement this part is similar to the above with the only change is in using a diagonal matrix for α instead of justone scalar value. Different α_i was used for each joint to see the effect on the path of the end effector.

The following vector of values gave the best solution

```
alpha    = diag([0.05,0.01,0.1]);
```

The above produced the least amount of oscillation as the end effector came close to the target. Making α_i smaller caused the joint i to move the slowest during the animation.

The following value of vector α caused large oscillation as the end effector came close to the target.

```
alpha    = diag([0.05,0.1,0.01]);
```

These values of α are kept in the code below to allow one to try them. The source code for this part is listed below. Using different α_i value for each joint is more flexible and allowed finding better solution than using the same *alpha* for all joints. The command to run the Matlab script is `nma_HW4_problem_3_part2`

```
function  nma_HW4_problem_3_part2()
%function nma_HW4_problem_3_part2()
%This function implements path planning for the 3 links robot arm in
%HW4, ME 739, problem 3, spring 2015, Univ. Wiscosin, Madison.
%
%This version is for part 2, which uses different alpha for each joint
%
%Parabolic attractive field is used to model the attractive virtual
%force on the end effector. The linear velocity Jacobian is used to
%map this force to torque forces at the three joints.
%Then gradient descent is used to obtain the joint coordinates sequence
%which is then used to simulate the motion.
%
%By Nasser M. Abbasi

close all; clear all;
L        = 4;  %length of link 1 and 3
%epsilon  = 0.005; %error tolerance
%alpha    = 0.015; %smaller value, slows down convergence but more accurate

epsilon  = 0.01; %error tolerance
zeta     = 100;  %attractive force scaling
%alpha    = diag([0.05,0.1,0.01]); %Causes large errors in solution
alpha    = diag([0.05,0.01,0.1]);  %Produces best solution

q        = [L;0;L];  %initial joint configuration
qf       = [2*L;pi/2;2*L]; %find joint configuration
maxIter  = 1000;  %max iterations allowed

Q           = zeros(3,maxIter);  %where to save the sequence of joint q's
k           = 1;
```

```
keep_running = true;

%start of gradient descemt

while keep_running
    Q(:,k) = q;
    tau = -zeta*[q(1)-2*L;
        q(3)*cos(q(2))*(q(3)*sin(q(2))-2*L)-q(3)^2*cos(q(2))*sin(q(2));
        q(3)*(cos(q(2)))^2+sin(q(2))*(q(3)*sin(q(2))-2*L)
        ];
    q      = q + alpha*tau/norm(tau);
    if k+1>maxIter || norm( q-qf ) <= epsilon
        keep_running = false;
    else
        k = k +1;
    end
end

DO_MOVIE = false; %make true to generate movie frames
frameNumber = 0;

f_handle    = 1;
axis_limits = L*[-2 2 -2 2 -0.1 2.5];
render_view = [-.4 -.8 .5]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective % turns on perspective

% link 0 rendering initialization
r0 = L/5; sides0 = 4; axis0 = 3; norm_L0 = -1.0;
linkColor0 = [0 .3 .3]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L,r0,sides0,axis0,norm_L0,linkColor0,...
    plotFrame0,f_handle);


% link 1 rendering initialization
r1 = L/6; sides1 = 4; axis1 = 3; norm_L1 = 1.0;
linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2 = L/7; sides2 = 10; axis2 = 3; norm_L2 = -1.0;
linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3 = L/8; sides3 = 4; axis3 = 1; norm_L3 = 1.0;
linkColor2 = [0.75 0 1]; plotFrame2 = 0;
d3 =  CreateLinkRendering(L,r3,sides3,axis3,norm_L3,linkColor2,...
    plotFrame2,f_handle);

T00 = [1  0  0  0
    0  1  0  0
```

```
    0  0  1  0
    0  0  0  1];
UpdateLink(d1,T00);

for i = 1:k
    % Update frames
    q1=Q(1,i); q2=Q(2,i); q3=Q(3,i);
    T01 = [1  0  0  0
           0  1  0  0
           0  0  1  q1
           0  0  0  1];

    T12 = [-sin(q2)  0   cos(q2)   0
           cos(q2)   0   sin(q2)   0
           0         1   0         0
           0         0   0         1];

    T2e = [0   1  0  0
           0   0  1  0
           1   0  0  q3
           0   0  0  1];

    T02 = T01*T12;
    T0e = T02*T2e;

    UpdateLink(d1,T01);
    UpdateLink(d2,T02);
    UpdateLink(d3,T0e);

    title(sprintf('step = %d, position=[%3.2f,%3.2f,%3.2f], epsilon=%3.3f',i,...
        q3*cos(q2),q3*sin(q2),q1,epsilon));
    %if i == 1;  %pause at start of simulation rendering
    %    pause;
    %end

    if DO_MOVIE
        frameNumber = frameNumber+1;
        I = getframe(gcf);
        imwrite(I.cdata, sprintf('frame%d.png',frameNumber));
    end
    hold on;
    plot3(0,2*L,2*L,'marker','o','color','r');
    drawnow;
    hold off;
    pause(0.01);
end
end
```