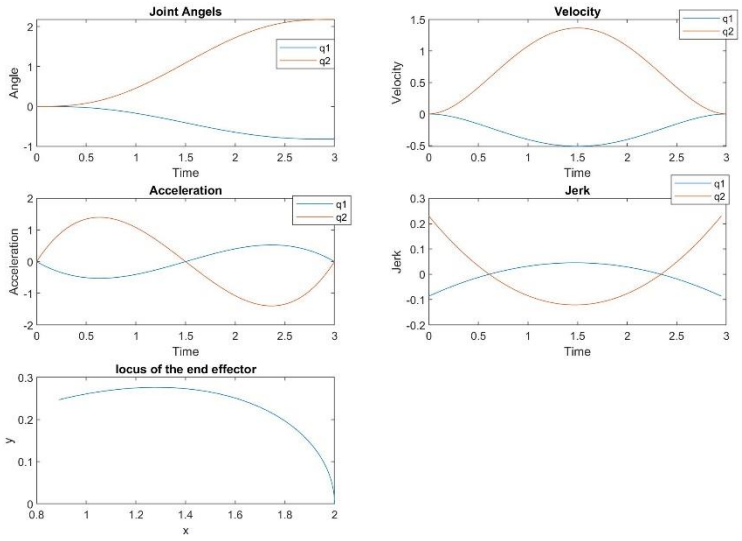
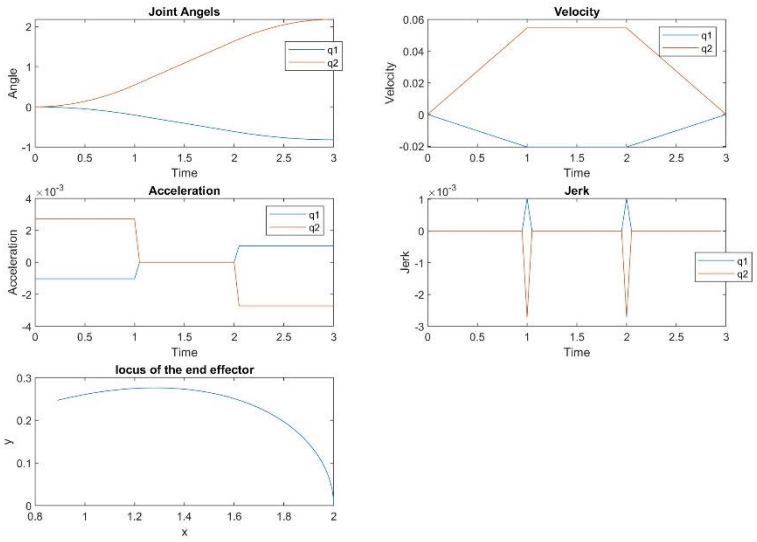
MTRN4230
Assignment 4
Runkai Zhao, z5146927
(Notice: The MATLAB programming of question 1, 2, and 4 has been uploaded as following of this report on Microsoft Team.)

1 Question 1

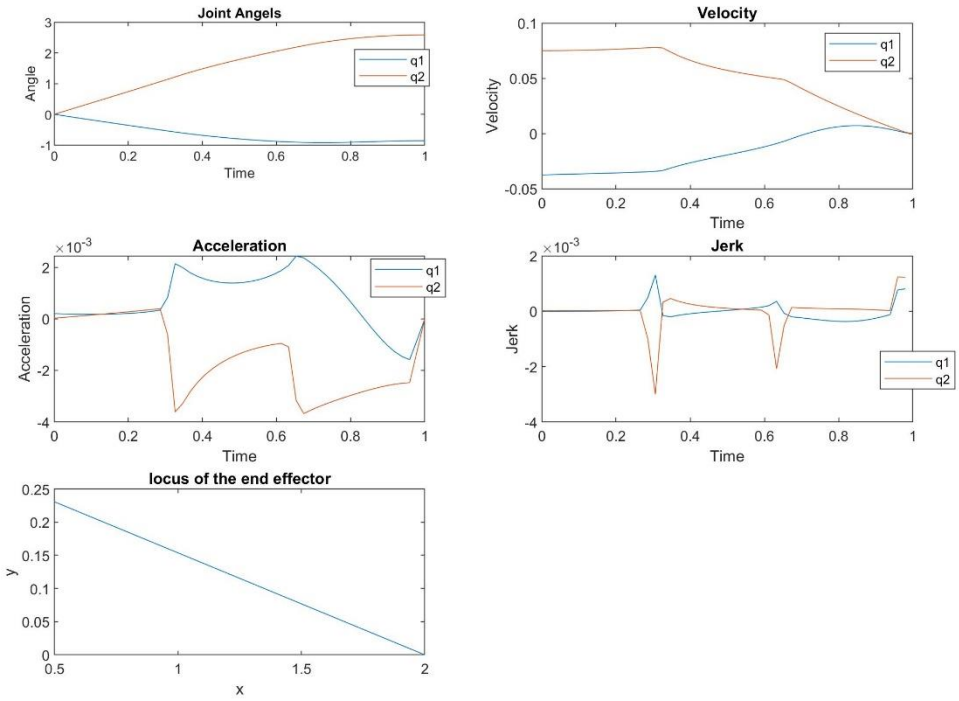
1.a Specify a joint space trajectory formed by quintic polynomials using the 'jtraj' function and a period of 3 seconds between q1 and q2.



1.b Specify a joint space trajectory formed by lspb velocity profiles using the 'mtraj' function and a period of 3 seconds between q1 and q2



1.c Specify a straight-line trajectory using the 'ctraj' function and a period of 3 seconds between q1 and q2



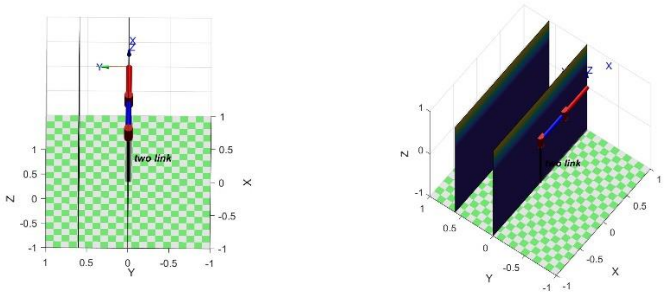
1.d Discuss the difference in performance between the three trajectories above in terms of the joint angles, velocities, accelerations, and jerks

Answer:

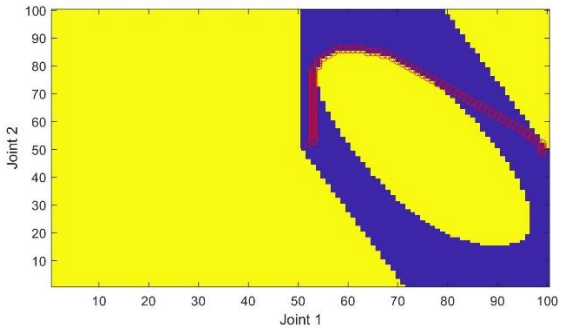
"jtraj" computing a joint space trajectory is to utilize the quintic polynomial for calculating the values of velocities and accelerations, so the graphs produced by this method is relatively smoothy and continuous. "mtraj" multi-segment multi-axis trajectory applies the scalar trajectory function, "lspb" option, for a trapezoidal trajectory, which includes linear motion and polynomial blends. It is clear that the plots of velocity, acceleration and jerk are discontinuous, but the overall trends of the function plots over time is much similar to graphs produced by "jtraj". Both two methods could build an arc trajectory between two given points, and the locus plots of end effector are same. "ctraj" could generate Cartesian trajectory of a straight line between two poses. The velocities of two joints are decreased over time, rather than first increase then decrease, so the acceleration and jerk are also different from "jtraj" and "mtraj". Obviously, the locus of end effector is a linear line.

2 Question 2

2.a Plot the obstacles above and below the robot arm



2.b Generate the shortest path using D* and plot it in the discrete joint space overlaid on the cost map.

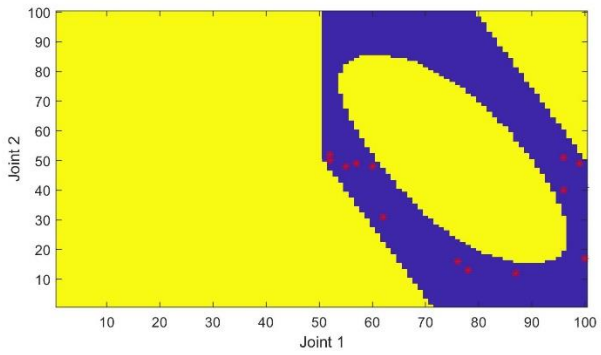



```
% initialization setup
ds = Dstar(config_space_binary');
target = [175,-5]/180*pi;
start = [5,5]/180*pi;
n1 = 100;
min1 = -pi;
max1 = pi;
n2 = 100;
min2 = -150/180*pi;
max2 = 150/180*pi;
% convert angles to cell coordinates, then appl D* trajectory planning
% method
target(1) = angle2cell(target(1), min1, max1, n1);
target(2) = angle2cell(target(2), min2, max2, n2);
start(1) = angle2cell(start(1), min1, max1, n1);
start(2) = angle2cell(start(2), min2, max2, n2);
ds.start = start; ds.plan(target); traj = ds.query(start);
plot(traj(:,1),traj(:,2),'ro')
```

2.c Convert this path back to joint angles and animate the robot to follow the path

```
% convert back to joint angles and animate it
for i = 1:size(traj)
    traj(i,1) = cell2angle(traj(i,1), min1, max1, n1);
    traj(i,2) = cell2angle(traj(i,2), min2, max2, n2);
end
figure(1); hold on;
two_link.plot(traj)
```

2.d Generate a path using PRM and plot it in the discrete joint space overlaid on the road map.




```
f2 = figure(3);
set(gcf, 'OuterPosition',[100 100 scrsz(3)/2-100 scrsz(4)/2]);
imagesc(config_space_binary'); % Note the transpose of the joint space for convenience
axis xy; % Sets axis with x and y positive directions as expected, not as image coordinates
xlabel('Joint 1')
ylabel('Joint 2')
hold on;
% convert angles to cell coordinates, then appl PRM trajectory planning
% method
target = [175,-5]/180*pi;
start = [5,5]/180*pi;
target(1) = angle2cell(target(1), min1, max1, n1);
target(2) = angle2cell(target(2), min2, max2, n2);
start(1) = angle2cell(start(1), min1, max1, n1);
start(2) = angle2cell(start(2), min2, max2, n2);
prm = PRM(config_space_binary');
prm.plan();
traj = prm.query(start, target);
plot(traj(:,1),traj(:,2),'r*');
```

2.e Convert this path back to joint angles and animate the robot to follow the path

```
% convert back to joint angles and animate it
for i = 1:size(traj)
    traj(i,1) = cell2angle(traj(i,1), min1, max1, n1);
    traj(i,2) = cell2angle(traj(i,2), min2, max2, n2);
end
figure(1)
hold on;
two_link.plot(path)
```

2.f Describe and discuss the difference between the paths generated in terms of the algorithms used

These two algorithms can produce feasible trajectories from the start point to the target point without violating the motion boundaries. Compared with the PRM algorithm, it can be seen that the D* algorithm could generate more dense and continuous cell points of joint space on the map, which leads the animation of the robot movement smoothy.

3 Question 3

3.a Two-link manipulator with the below Jacobian, what are the torques at the joints needed to apply a force vector °F = 10 X_0?

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}-l_1s_1 - l_2s_{12} & -l_2s_{12}\\ l_1c_1 + l_2c_{12} & l_2c_{12}\end{bmatrix}\begin{bmatrix}10\\ 0\end{bmatrix} = \begin{bmatrix}-10l_1s_1 - 10l_2s_{12}\\ 10l_1c_1 + 10l_2c_{12}\end{bmatrix}$$

4 Question 4

4.a Inertia Matrix

4.b D Matrix (Acceleration-related inertia matrix term)

```
syms theta1 alpha1 a1 d1
T_0_1 = [cos(theta1), -sin(theta1), 0, a1*cos(theta1);
        sin(theta1), cos(theta1), 0, a1*sin(theta1);
        0, 0, 1, 0;
        0, 0, 0, 1];

syms theta2 alpha2 a2 d2
T_1_2 = [cos(theta2), sin(theta2), 0, a2*cos(theta2);
        sin(theta2), -cos(theta2), 0, a2*sin(theta2);
        0, 0, -1, 0;
        0, 0, 0, 1];


syms theta3 alpha3 a3 d3
T_2_3 = [1, 0, 0, 0;
        0, 1, 0, 0;
        0, 0, 1, d3;
        0, 0, 0, 1];

syms theta4 alpha4 a4 d4
T_3_4 = [cos(theta4), -sin(theta4), 0, 0;
        sin(theta4), cos(theta4), 0, 0;
        0, 0, 1, d4;
        0, 0, 0, 1];
```

```
T_0_4 = T_0_1*T_1_2*T_2_3*T_3_4;
T_0_3 = T_0_1*T_1_2*T_2_3;
T_0_2 = T_0_1*T_1_2;
T_0_4 = simplify(T_0_4);
T_0_3 = simplify(T_0_3);
T_0_2 = simplify(T_0_2);
T_0_1 = simplify(T_0_1);

k = [0;0;1];
O_0_0 = [0;0;0];
O_0_4 = [T_0_4(1:3, 4)];
O_0_3 = [T_0_3(1:3, 4)];
O_0_1 = [T_0_1(1:3, 4)];

Jw1 = [[0;0;1], zeros(3,1), zeros(3,1), zeros(3,1)];
Jw2 = [[0;0;1], T_0_1(1:3, 3), zeros(3,1), zeros(3,1)];
Jw3 = [[0;0;1], T_0_1(1:3, 3), zeros(3,1), zeros(3,1)];
Jw4 = [[0;0;1], T_0_1(1:3, 3), zeros(3,1), T_0_3(1:3, 3)];

Jv1 = [cross(k, (O_0_4-O_0_0)), zeros(3,1), zeros(3,1), zeros(3,1)];
Jv2 = [cross(k, (O_0_4-O_0_0)), cross(T_0_1(1:3, 3), (O_0_4-O_0_1)), zeros(3,1), zeros(3,1)];
Jv3 = [cross(k, (O_0_4-O_0_0)), cross(T_0_1(1:3, 3), (O_0_4-O_0_1)), T_0_2(1:3, 3), zeros(3,1)];
Jv4 = [cross(k, (O_0_4-O_0_0)), cross(T_0_1(1:3, 3), (O_0_4-O_0_1)), T_0_2(1:3, 3), cross(T_0_3(1:3, 3), (O_0_4-O_0_3))];

q = [theta1; theta2; d3; theta4];
```

```
q = [theta1; theta2; d3; theta4];

syms dtheta1 dtheta2 dd3 dtheta4
dfq = [dtheta1; dtheta2; dd3; dtheta4];

% linear velocity kinetic
syms m1 m2 m3 m4
temp1 = m1*Jv1'*Jv1 + m2*Jv2'*Jv2 + m3*Jv3'*Jv3 + m4*Jv4'*Jv4;
LinearKinetic = (1/2)*dfq'*temp1*dfq;

% angualr velocity kinetic
syms I1 I2 I3 I4
R2 = T_0_1(1:3, 1:3);
R3 = T_0_2(1:3, 1:3);
R4 = T_0_3(1:3, 1:3);

rotationK1 = Jw1'*I1*Jw1;
rotationK2 = Jw2'*R2*I2*R2'*Jw2;
rotationK3 = Jw3'*R3*I3*R3'*Jw3;
rotationK4 = Jw4'*R4*I4*R4'*Jw4;
temp2 = rotationK1 + rotationK2 + rotationK3 + rotationK4;
RotationalKinetic = (1/2)*dfq'*temp2*dfq;

% D matrix 4x4
D = temp1 + temp2;
```

4.c C matrix

```
% C matrix
C = sym('a', [4 4 4]);
for k = 1:4
    for j = 1:4
        for i = 1:4
            C(i,j,k) = (1/2)*(diff(D(k,j), q(i)) + diff(D(k,i), q(j)) - diff(D(i,j), q(k)));
        end
    end
end
```

4.d Kinetic Energy

```
% the overall kinetic
K = LinearKinetic + RotationalKinetic;
```

4.e Potential Energy

```
% potential energy
P = (m3 + m4)*d3*9.81;
```

4.f g Matrix

```
% g matrix
g1 = diff(P, theta1);
g2 = diff(P, theta2);
g3 = diff(P, d3);
g4 = diff(P, theta4);
g = [g1; g2; g3; g4];
```