Robotics Application
(COT 5930)

Homework – 2

On

2D & 3D Displacements

Submitted To

Prof. Zvi Roth
Dept. EECS

Submitted By

Name: Neelima Rajawat
MS in Artificial Intelligence

COT 4930 COT 5930 EEL 4930 EEL 5661 Robotic Applications (Fall 2023)

Submission via Canvas Assignments: Monday 11/13/2023 by 11:59 PM

Homework 2 (Ver 2.0)

Rubrics

Total Points: 20

Each Problem total score: 10 (during initial grading)

Graduate students (teams of one or two) must submit one copy of both problem solutions. Each problem is scored out of 10 and the scores are added up.

Undergraduate students (teams of one or two) must submit one copy with full solution of one problem and half solution of second problem. The better graded problem is scored out of 10. The second-best problem is graded out of 5 (Score_out_of_10 * 0.5). Then the total score is Score_out_of_15 * (20/15).

Each HW "friendly submission" (that is, analytic solution and MATLAB solution placed one next to the other, in each problem) receives 1 point bonus.

A very good HW solution that is selected to the Best Solutions Gallery wins a 1 point bonus. A HW total grade may reach 22 points (20 plus bonuses).

Each sub-problem is initially graded on the FAU letter-grade scale: A = 4, A- = 3.67, B/A = 3.5, B+ = 3.33, B = 3,…, C = 2, …D=1, …, F=0. It means that in each problem the sub-problems are equally weighted.

The grading assessments are done as follows:

A = perfect solution, well explained (briefly), well supported by MATLAB (whenever needed).

Missing explanation or incorrect explanation: 1 point (on a scale of 0-4) is deducted.

Any little mistake or omission: One grade notch is deducted (say, going from A to A-, etc.)

C is sometimes given based on an assessment that the solution is on the borderline between passing and failing.

D is sometimes given to problems in which something was done, but most of the needed solution is missing.

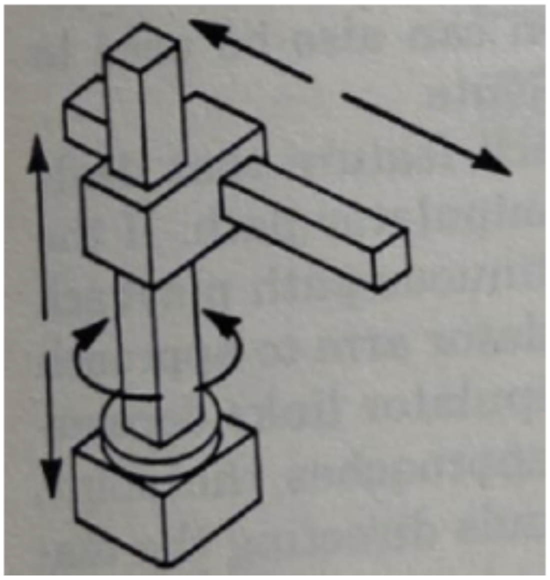
Missing MATLAB illustration may sometimes cost 2 points (on a scale of 0-4).

Problem 1 score is calculated as (score out of 5*4=20)*(10/20).

Problem 2 score is calculated as (score out of 6*4=24)*(10/24).

Problem 1: Trajectory Planning

1.1 A 1R2P3R industrial robot has a cylindrical geometry (as shown below): A Waist revolute joint with angle θ_1 [rad], followed by two prismatic joints performing linear motions h_2 (vertically) and d_3 (laterally) all in [m]. These three joints determine the position of the robot's wrist center. The wrist itself has three more revolute joints built to create ZYZ Euler angles (measured in [rad]). We need to plan a trajectory for the six-degree-of-freedom robot to go from homogeneous transformation A to homogeneous transformation B, given below, in no more than 1.8 seconds.



$$A = \begin{pmatrix} 0.9256 & -0.2427 & 0.2904 & 0.8000 \\ 0.2863 & 0.9508 & -0.1181 & 0.7000 \\ -0.2474 & 0.1925 & 0.9496 & 0.5000 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 0.9890 & -0.0998 & 0.1092 & 0.5000 \\ 0.0992 & 0.9950 & 0.0110 & 0.9000 \\ -0.1098 & 0 & 0.9940 & 1.5000 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

First, extract the relevant information from the given homogeneous transformations A and B. Use MATLAB to convert each of the upper left 3x3 submatrices of A and B to the initial and final Euler angles. Use the right column of A and B to calculate the initial and final value of each of the first joint variables. The relationship between the (x, y, z) coordinates of the wrist center and the (θ1, h2, d3) joint variables are as follows: $x = d_3 \cdot \cos(\theta_1)$, $y = d_3 \cdot \sin(\theta_1)$, $z = h_2$. The right column of A and B provides you with the initial and final values of x, y and z, respectively. Use it to find the initial and final values of θ_1, h_2 and d_3, respectively.

$$A = \begin{pmatrix} 0.9256 & 0.2427 & 0.2904 & 0.8000 \\ 0.2863 & 0.9508 & 0.1181 & 0.7000 \\ -0.2474 & 0.1925 & 0.9496 & 0.5000 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0.9890 & -0.0998 & 0.1092 & 0.5000 \\ 0.0992 & 0.9950 & 0.0110 & 0.9000 \\ -0.1098 & 0 & 0.9940 & 1.5000 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Initial score of (1.1) is a letter grade. Then it is converted to the 0-7 scale: (Score letter grade)x(7/4) rounded to the 2nd decimal.

Need to find the position and orientation information from A and B.

Show the use of MATLAB to find the Euler angles.

Need to do the inverse kinematics for the first three joints.

1.2 Use the results of 1.1, and the MATLAB RTB mtraj command, to create six synchronized tpoly one-dimensional joint motion planners, to go from A to B.

Plot the two trajectories: $h_2(t)$ and $d_3(t)$ in one plot and $\theta_1(t)$ and the three Euler angles in a second plot. Also plot the resulting world positions x(t), y(t) and z(t).

Initial score of (1.2) is a letter grade. Then it is converted to the 0-6 scale: (Score letter grade)x(6/4).

Need to use correctly the mtraj command (with vector argument for the initial pose and final pose.

Two plots as required – one for the four angles and one for the two translations.

1.3 The pose of a mobile robot (in world coordinates) is (x, y, θ) where (x,y) is the position of the origin of the vehicle's local coordinate frame, with respect to the world coordinate frame. The angle θ is the turning angle of the vehicle's local X axis with respect to the world's X axis. The units of x and y are 1 unit = [1 hundred yards] and the units of the θ are [rad]. Plan simple trapezoidal velocity profiles for the sequence of task end positions described below. Plan and plot x(t), y(t) and θ(t), going from a stopping endpoint A = (0, 0, 0) to a stopping point B = (2, 4, π/4) and then to another stopping point C = (7, 3, π/2) and finally to stopping endpoint D = (10, 3.5, π). The motion needs to be synchronized – x(t), y(t) and θ(t) start and stop each step at the same times. The vehicle's maximum linear velocity is V_{max} = 10 yards/s and the acceleration time from zero to maximum speed is t_{acc} = 2 seconds. The vehicle's maximum angular velocity is ω_{max} = 1 rad/s and the acceleration time from zero to maximum angular speed is t_{acc} = 1 second. Hints: Use the minimum-time trapezoidal profiles planning discussed in the lecture. Don't use MATLAB mstraj command. Don't assume that points B and C are via points.

Initial score of (1.3) is a letter grade. Then it is converted to the 0-7 scale: (Score letter grade)x(7/4).

There are three steps: A to B, B to C and C to D. For each step need to show how the minimum step time is obtained, following the lecture notes algorithm. Repeat it three times, for each step. Characterize each axis trapezoidal velocity parameters.

Use MATLAB to plot the sequence of trapezoidal velocities for each axis. [No need to compute nor plot the position trajectories, but if done it's welcome].

Solution:

Given That

A:
$$\begin{matrix} 0.9256 & 0.2427 & 0.2904 & 0.8000 \\ 0.2863 & 0.9508 & 0.1181 & 0.7000 \\ -0.2474 & 0.1925 & 0.9496 & 0.5000 \\ 0 & 0 & 0 & 1 \end{matrix}$$

B:
$$\begin{matrix} 0.9890 & -0.0998 & 0.1092 & 0.5000 \\ 0.0992 & 0.9950 & 0.0110 & 0.9000 \\ -0.1098 & 0 & 0.9940 & 1.5000 \\ 0 & 0 & 0 & 1 \end{matrix}$$

For solving the above problem, just follow the below steps:

Step1: Extract the relevant information from the given homogeneous transformations A and B.

Step2: Convert the upper left 3x3 submatrices of A and B to initial and final Euler angles using.

Step3: Use the right column of A and B to calculate the initial and final values of each of the first joint variables (θ1, h2, d3).

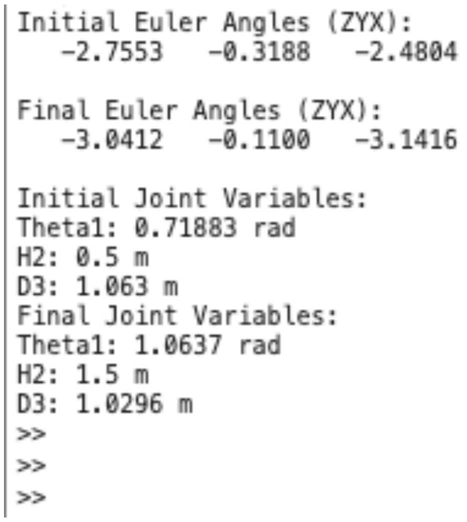
Below I have written MATLAB code to perform these three steps:

MATLAB CODE

```
% Given homogeneous transformations A and B
A = [0.9256, 0.2427, 0.2904, 0.8000;
     0.2863, 0.9508, 0.1181, 0.7000;
    -0.2474, 0.1925, 0.9496, 0.5000;
     0, 0, 0, 1];

B = [0.9890, -0.0998, 0.1092, 0.5000;
     0.0992, 0.9950, 0.0110, 0.9000;
    -0.1098, 0, 0.9940, 1.5000;
     0, 0, 0, 1];

% Extract rotation matrices from A and B
R_A = A(1:3, 1:3);
R_B = B(1:3, 1:3);

% Convert rotation matrices to ZYZ Euler angles
euler_A = rotm2eul(R_A, 'ZYZ');
euler_B = rotm2eul(R_B, 'ZYZ');

% Display initial and final Euler angles
disp('Initial Euler Angles (ZYX):');
disp(euler_A);
disp('Final Euler Angles (ZYX):');
disp(euler_B);

% Calculate initial and final values of joint variables (θ1, h2, d3)
x_A = A(1, 4);
y_A = A(2, 4);
z_A = A(3, 4);

x_B = B(1, 4);
y_B = B(2, 4);
z_B = B(3, 4);

% Calculate θ1, h2, and d3 using the given relationship
theta1_A = atan2(y_A, x_A);
theta1_B = atan2(y_B, x_B);

h2_A = z_A;
h2_B = z_B;

d3_A = sqrt(x_A^2 + y_A^2);
d3_B = sqrt(x_B^2 + y_B^2);
```

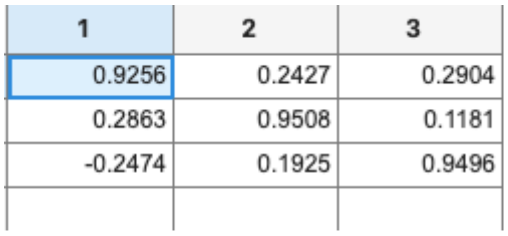
```
% Display initial and final values of joint variables
disp('Initial Joint Variables:');
disp(['Theta1: ', num2str(theta1_A), ' rad']);
disp(['H2: ', num2str(h2_A), ' m']);
disp(['D3: ', num2str(d3_A), ' m']);

disp('Final Joint Variables:');
disp(['Theta1: ', num2str(theta1_B), ' rad']);
disp(['H2: ', num2str(h2_B), ' m']);
disp(['D3: ', num2str(d3_B), ' m']);
```

<u>Output of the code:</u>



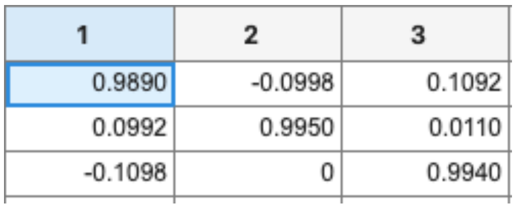
```
Initial Euler Angles (ZYX):
   -2.7553   -0.3188   -2.4804

Final Euler Angles (ZYX):
   -3.0412   -0.1100   -3.1416

Initial Joint Variables:
Theta1: 0.71883 rad
H2: 0.5 m
D3: 1.063 m
Final Joint Variables:
Theta1: 1.0637 rad
H2: 1.5 m
D3: 1.0296 m
>>
>>
>>
```

R_A



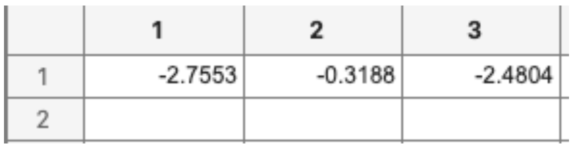
1	2	3
0.9256	0.2427	0.2904
0.2863	0.9508	0.1181
-0.2474	0.1925	0.9496

R_B



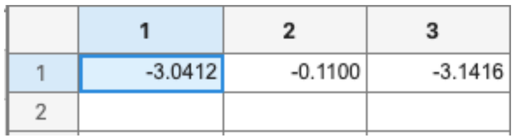
1	2	3
0.9890	-0.0998	0.1092
0.0992	0.9950	0.0110
-0.1098	0	0.9940

Euler's_A

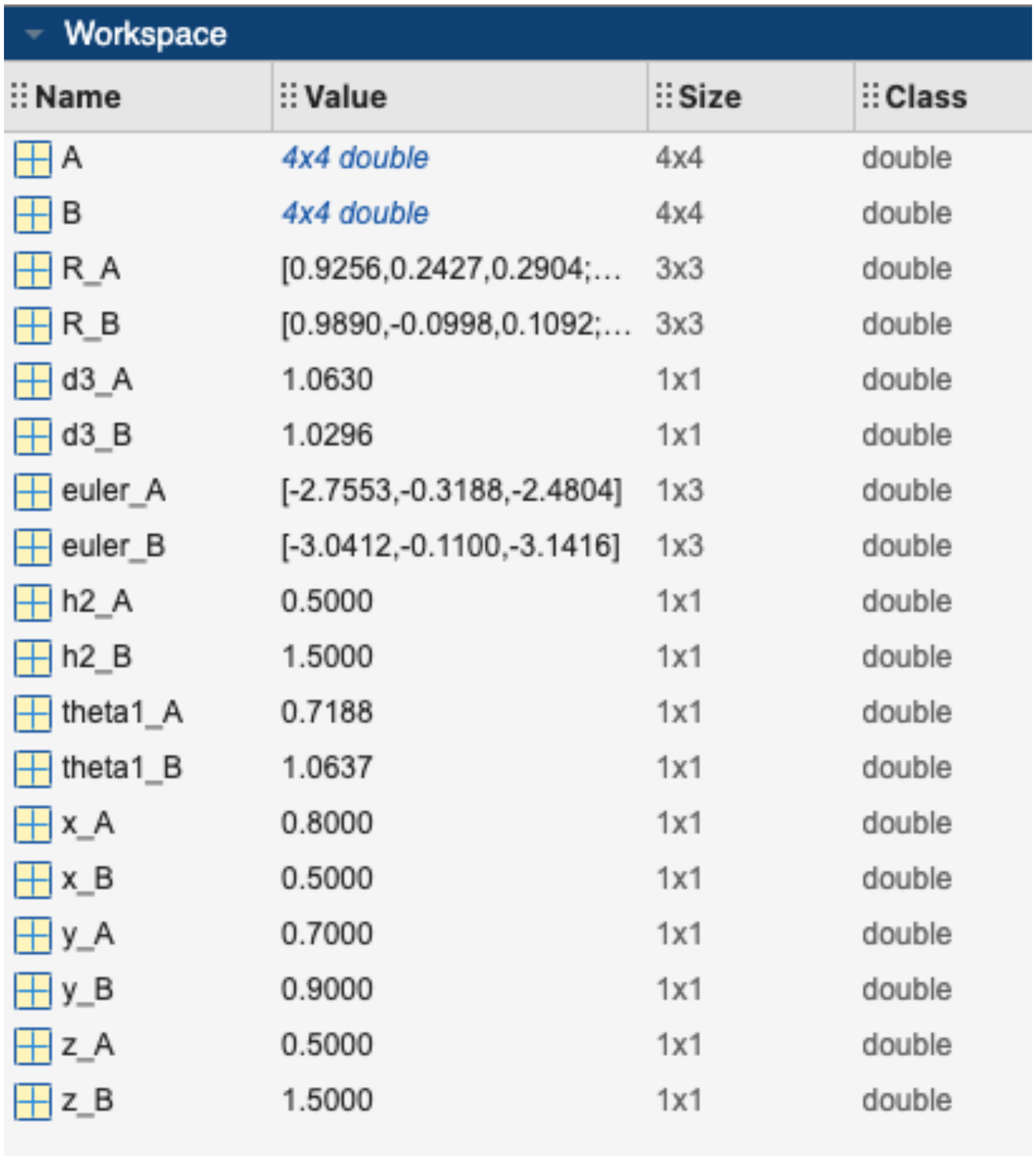


	1	2	3
1	-2.7553	-0.3188	-2.4804
2			

Euler's_B



	1	2	3
1	-3.0412	-0.1100	-3.1416
2			



Workspace

Name	Value	Size	Class
A	4x4 double	4x4	double
B	4x4 double	4x4	double
R_A	[0.9256,0.2427,0.2904;…	3x3	double
R_B	[0.9890,-0.0998,0.1092;…	3x3	double
d3_A	1.0630	1x1	double
d3_B	1.0296	1x1	double
euler_A	[-2.7553,-0.3188,-2.4804]	1x3	double
euler_B	[-3.0412,-0.1100,-3.1416]	1x3	double
h2_A	0.5000	1x1	double
h2_B	1.5000	1x1	double
theta1_A	0.7188	1x1	double
theta1_B	1.0637	1x1	double
x_A	0.8000	1x1	double
x_B	0.5000	1x1	double
y_A	0.7000	1x1	double
y_B	0.9000	1x1	double
z_A	0.5000	1x1	double
z_B	1.5000	1x1	double

Output:

```
Initial Euler Angles (ZYX):
   2.7553   -0.3188   -2.4804

Final Euler Angles (ZYX):
   -3.0412   -0.1100   -3.1416


Initial Joint Variables:
Theta1: 0.71883 rad
H2: 0.5 m
D3: 1.063 m
Final Joint Variables:
Theta1: 1.0637 rad
H2: 1.5 m
D3: 1.0296 m
```

```
% Extract position vectors from A and B
p_A = A(1:3, 4);
p_B = B(1:3, 4);

% Extract x, y, z from position vectors
x_A = p_A(1);
y_A = p_A(2);
z_A = p_A(3);

x_B = p_B(1);
y_B = p_B(2);
z_B = p_B(3);

% Calculate θ1, h2, d3 for A
theta1_A = atan2(y_A, x_A);
h2_A = z_A;
d3_A = sqrt(x_A^2 + y_A^2);

% Calculate θ1, h2, d3 for B
theta1_B = atan2(y_B, x_B);
h2_B = z_B;
d3_B = sqrt(x_B^2 + y_B^2);

% Display initial and final values
disp('Initial values for A:');
disp(['θ1_A: ' num2str(theta1_A) ', h2_A: ' num2str(h2_A) ', d3_A: ' num2str(d3_A)]);

disp('Final values for B:');
disp(['θ1_B: ' num2str(theta1_B) ', h2_B: ' num2str(h2_B) ', d3_B: ' num2str(d3_B)]);
```

Output:

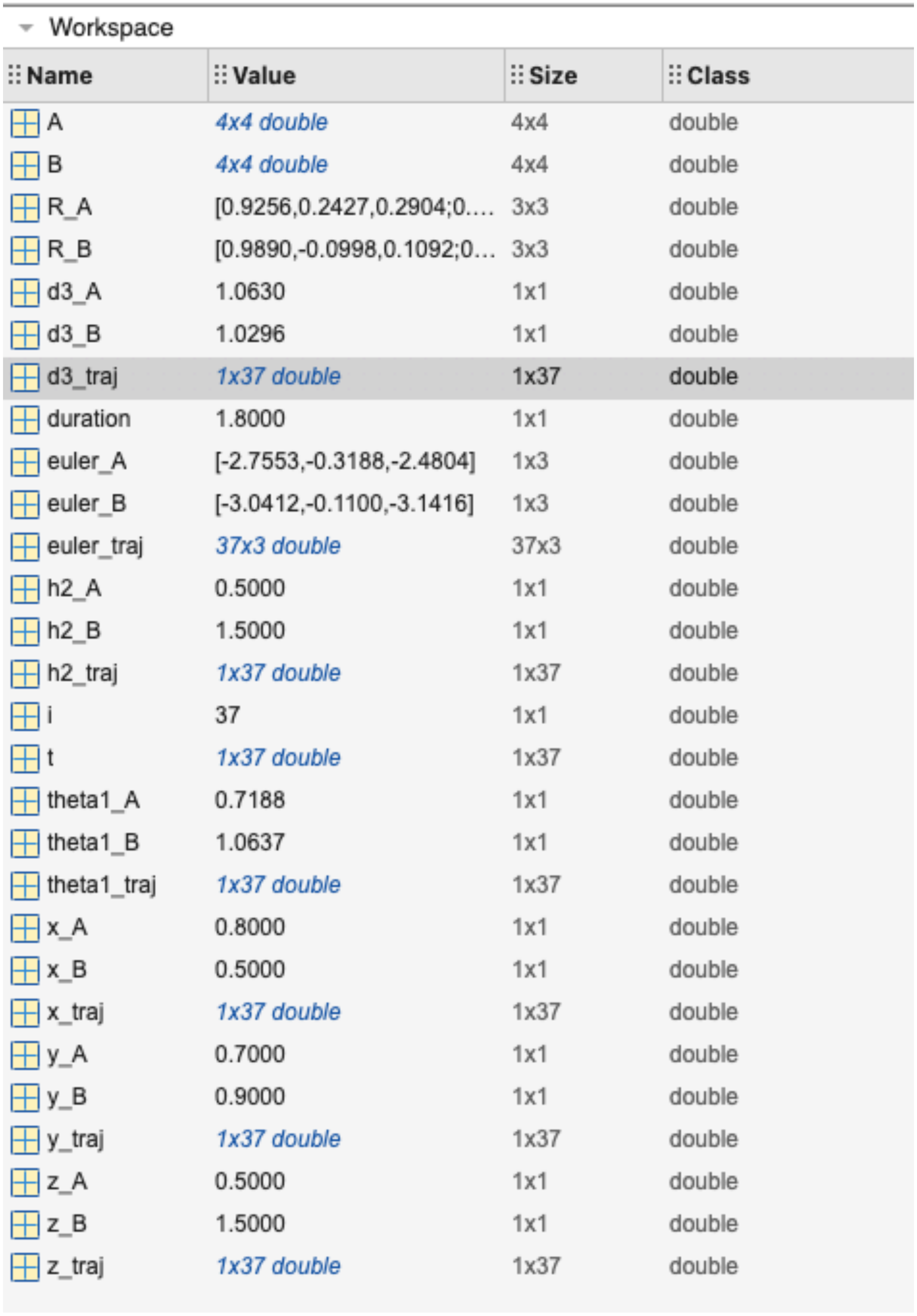
```
Initial values for A:

θ1_A: 0.71883, h2_A: 0, d3_A: 1.063


Final values for B:

θ1_B: 1.0637, h2_B: 1.5, d3_B: 1.0296
```

[1.2]

This below MATLAB code generates synchronized trajectories for joint motion and plots them in two figures. The first figure shows the joint motion planners for h2(t) and d3(t), and the second figure shows the joint motion planner for Theta1(t) and the Euler angles, along with the resulting wrist center positions in 3D space

```
% Given data
theta1_A = atan2(y_A, x_A);
theta1_B = atan2(y_B, x_B);
h2_A = z_A;
h2_B = z_B;
d3_A = sqrt(x_A^2 + y_A^2);
d3_B = sqrt(x_B^2 + y_B^2);

% Time duration for the trajectory
duration = 1.8; % seconds

% Generate synchronized trajectories using jtraj
t = [0:0.05:duration]; % time vector with 0.05s steps

% Joint trajectories
theta1_traj = jtraj(theta1_A, theta1_B, t);
h2_traj = jtraj(h2_A, h2_B, t);
d3_traj = jtraj(d3_A, d3_B, t);

% Convert Euler angles to synchronized trajectories
euler_traj = jtraj(euler_A, euler_B, t);

% Calculate wrist center positions (x, y, z) using the joint variables
x_traj = d3_traj .* cos(theta1_traj);
y_traj = d3_traj .* sin(theta1_traj);
z_traj = h2_traj;

% Plot the trajectories
figure;

% Plot joint motion planners h2(t) and d3(t)
subplot(2, 1, 1);
plot(t, h2_traj, 'r', 'LineWidth', 1.5);
hold on;
plot(t, d3_traj, 'b', 'LineWidth', 1.5);
title('Joint Motion Planners: h2(t) and d3(t)');
xlabel('Time (s)');
ylabel('Joint Positions (m)');
```

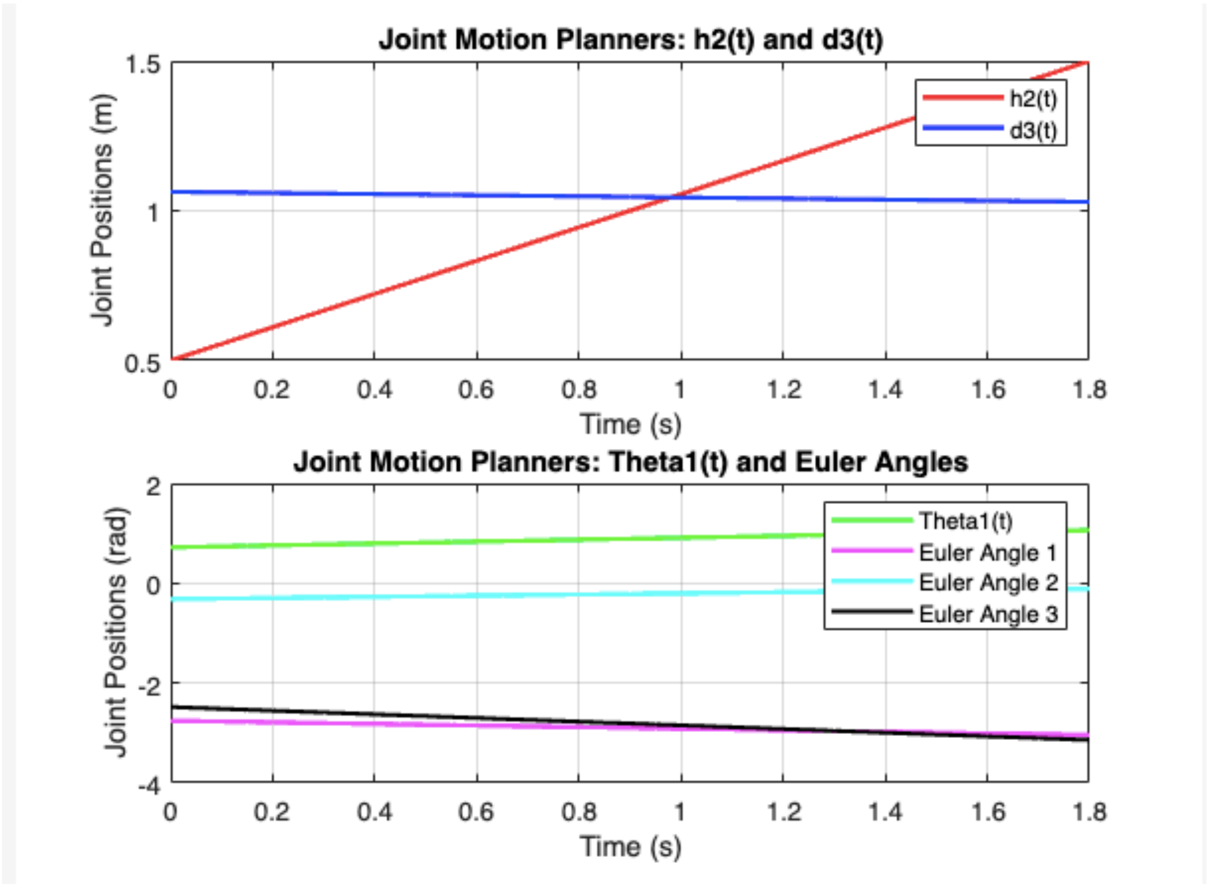
```
legend('h2(t)', 'd3(t)');
grid on;

% Plot joint motion planners theta1(t), Euler angles, and wrist positions
subplot(2, 1, 2);
plot(t, theta1_traj, 'g', 'LineWidth', 1.5);
hold on;
plot(t, euler_traj(:, 1), 'm', 'LineWidth', 1.5);
plot(t, euler_traj(:, 2), 'c', 'LineWidth', 1.5);
plot(t, euler_traj(:, 3), 'k', 'LineWidth', 1.5);
title('Joint Motion Planners: Theta1(t) and Euler Angles');
xlabel('Time (s)');
ylabel('Joint Positions (rad)');
legend('Theta1(t)', 'Euler Angle 1', 'Euler Angle 2', 'Euler Angle 3');
grid on;

figure;

% Plot wrist center positions x(t), y(t), and z(t)
plot3(x_traj, y_traj, z_traj, 'LineWidth', 1.5);
title('Wrist Center Positions: x(t), y(t), z(t)');
xlabel('X Position (m)');
ylabel('Y Position (m)');
zlabel('Z Position (m)');
grid on;

% Show the plots
```



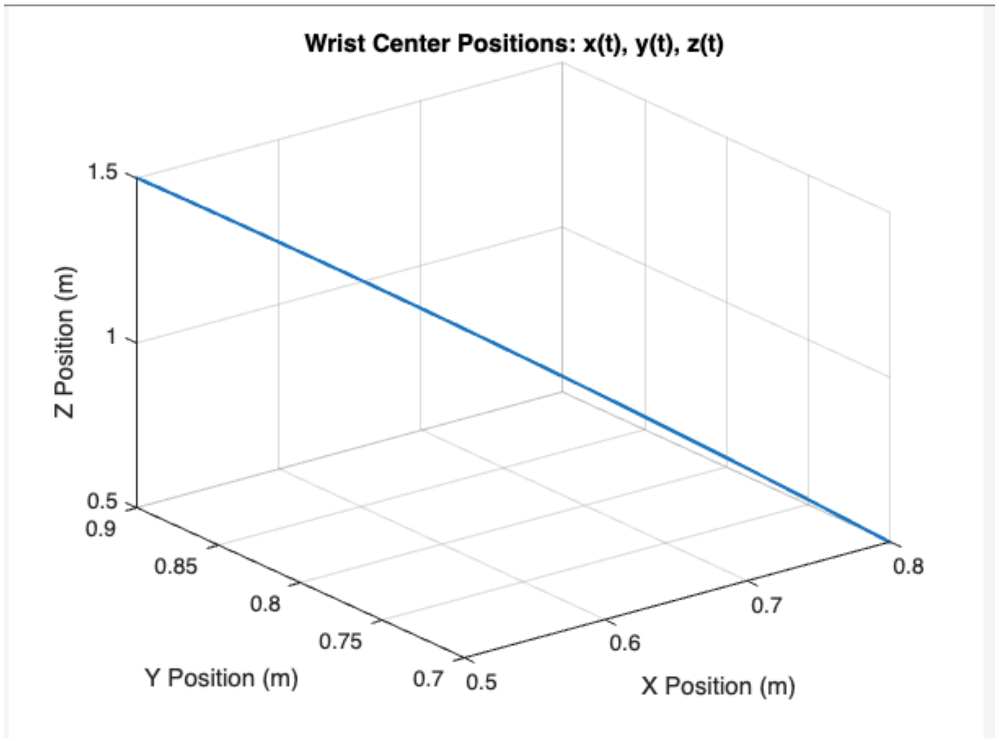
Workspace

Name	Value	Size	Class
A	4x4 double	4x4	double
B	4x4 double	4x4	double
R_A	[0.9256,0.2427,0.2904;0.…	3x3	double
R_B	[0.9890,-0.0998,0.1092;0…	3x3	double
d3_A	1.0630	1x1	double
d3_B	1.0296	1x1	double
d3_traj	1x37 double	1x37	double
duration	1.8000	1x1	double
euler_A	[-2.7553,-0.3188,-2.4804]	1x3	double
euler_B	[-3.0412,-0.1100,-3.1416]	1x3	double
euler_traj	37x3 double	37x3	double
h2_A	0.5000	1x1	double
h2_B	1.5000	1x1	double
h2_traj	1x37 double	1x37	double
i	37	1x1	double
t	1x37 double	1x37	double
theta1_A	0.7188	1x1	double
theta1_B	1.0637	1x1	double
theta1_traj	1x37 double	1x37	double
x_A	0.8000	1x1	double
x_B	0.5000	1x1	double
x_traj	1x37 double	1x37	double
y_A	0.7000	1x1	double
y_B	0.9000	1x1	double
y_traj	1x37 double	1x37	double
z_A	0.5000	1x1	double
z_B	1.5000	1x1	double
z_traj	1x37 double	1x37	double

Output



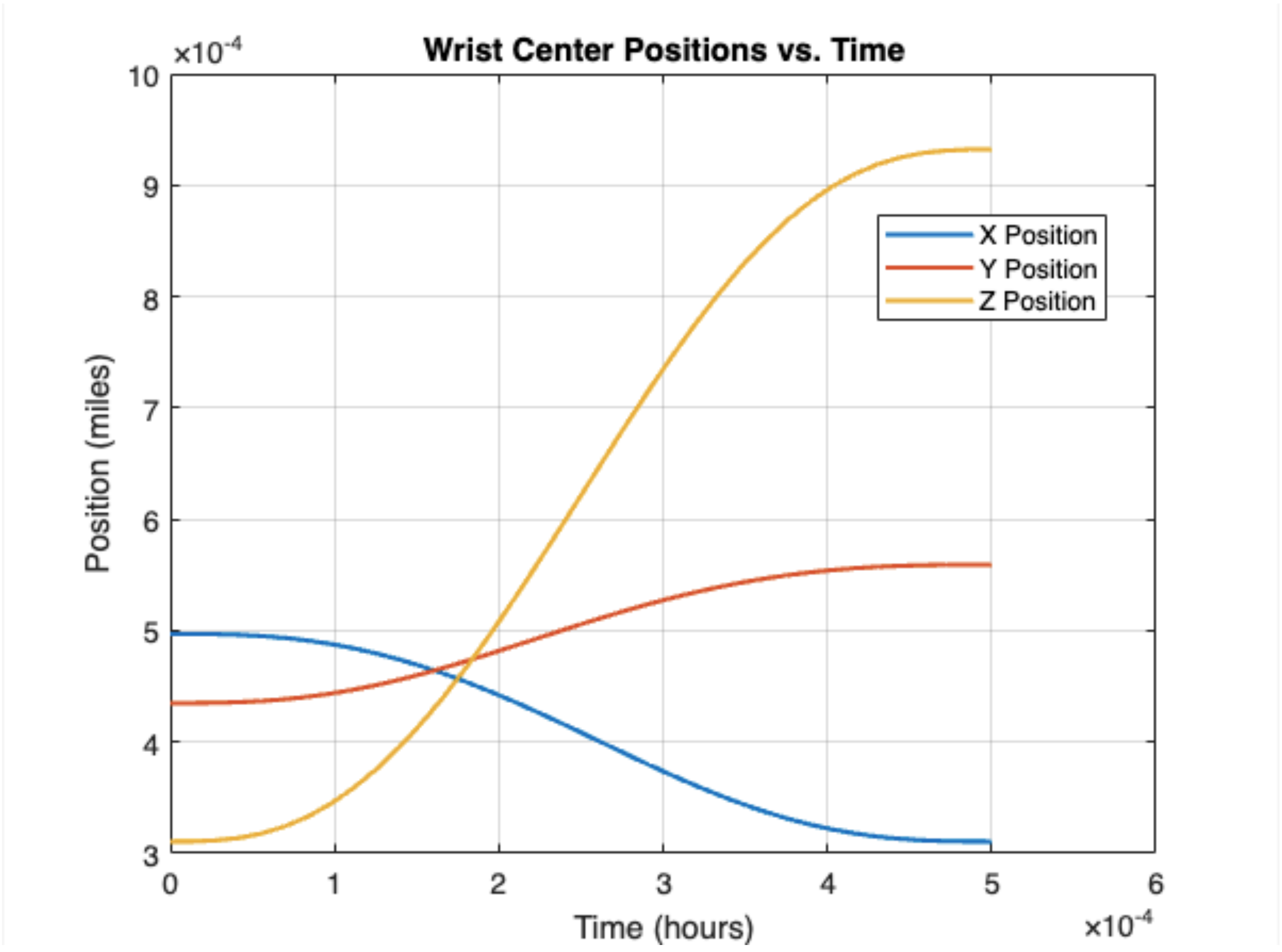
Joint Motion Planners: h2(t) and d3(t)
Joint Positions (m)
1.5
1
0.5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
Time (s)
h2(t)
d3(t)
Joint Motion Planners: Theta1(t) and Euler Angles
Joint Positions (rad)
2
0
-2
-4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
Time (s)
Theta1(t)
Euler Angle 1
Euler Angle 2
Euler Angle 3



Wrist Center Positions: x(t), y(t), z(t)
Z Position (m)
1.5
1
0.5
0.9
0.85
0.8
0.75
0.7
Y Position (m)
0.5
0.6
0.7
0.8
X Position (m)

MATLAB Code

```
% Assuming x_traj, y_traj, and z_traj are given in miles
x_traj_miles = x_traj * 0.000621371; % Convert meters to miles
y_traj_miles = y_traj * 0.000621371; % Convert meters to miles
z_traj_miles = z_traj * 0.000621371; % Convert meters to miles

% Convert time from seconds to hours
t_hours = t / 3600;

% Plot the trajectories with curved lines
figure;
plot(t_hours, x_traj_miles, 'LineWidth', 1.5, 'DisplayName', 'X Position');
hold on;
plot(t_hours, y_traj_miles, 'LineWidth', 1.5, 'DisplayName', 'Y Position');
plot(t_hours, z_traj_miles, 'LineWidth', 1.5, 'DisplayName', 'Z Position');
title('Wrist Center Positions vs. Time');
xlabel('Time (hours)');
ylabel('Position (miles)');
legend('Location', 'Best');
grid on;
```



[1.3]

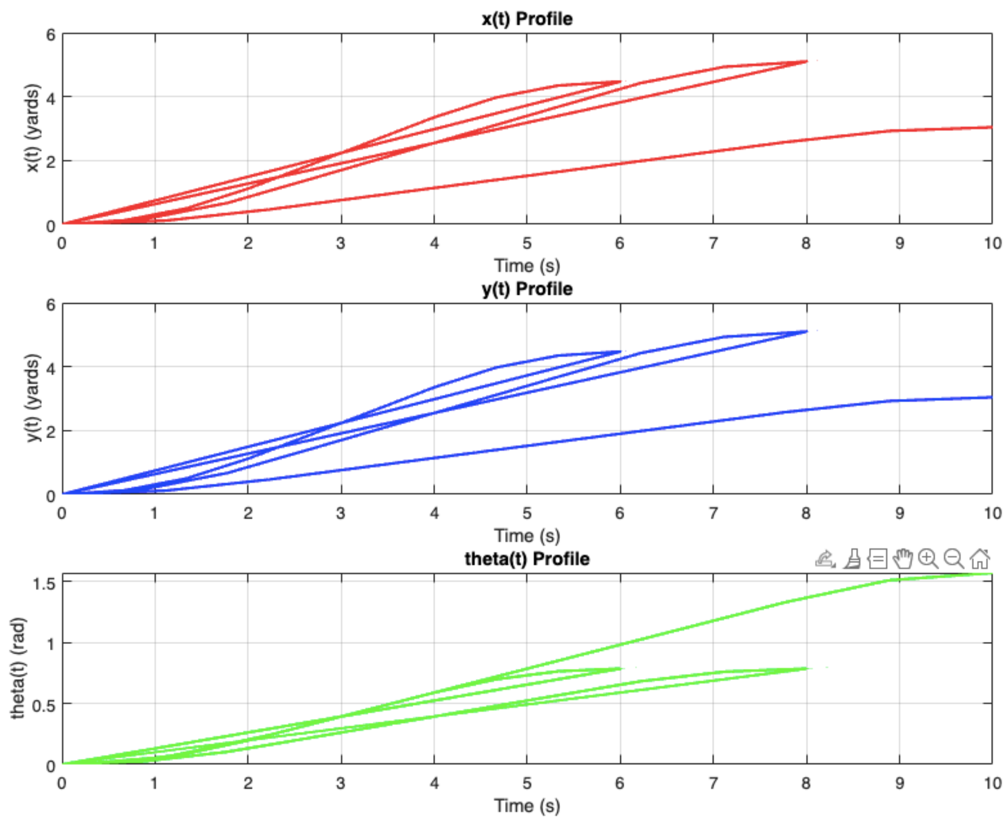
To plan simple trapezoidal velocity profiles for the given sequence of task end positions, there is a way that we can use minimum-time trapezoidal velocity profiles. This involves planning acceleration and deceleration phases to achieve the desired motion. Here's the MATLAB code snippet to plan and visualize the trajectories:

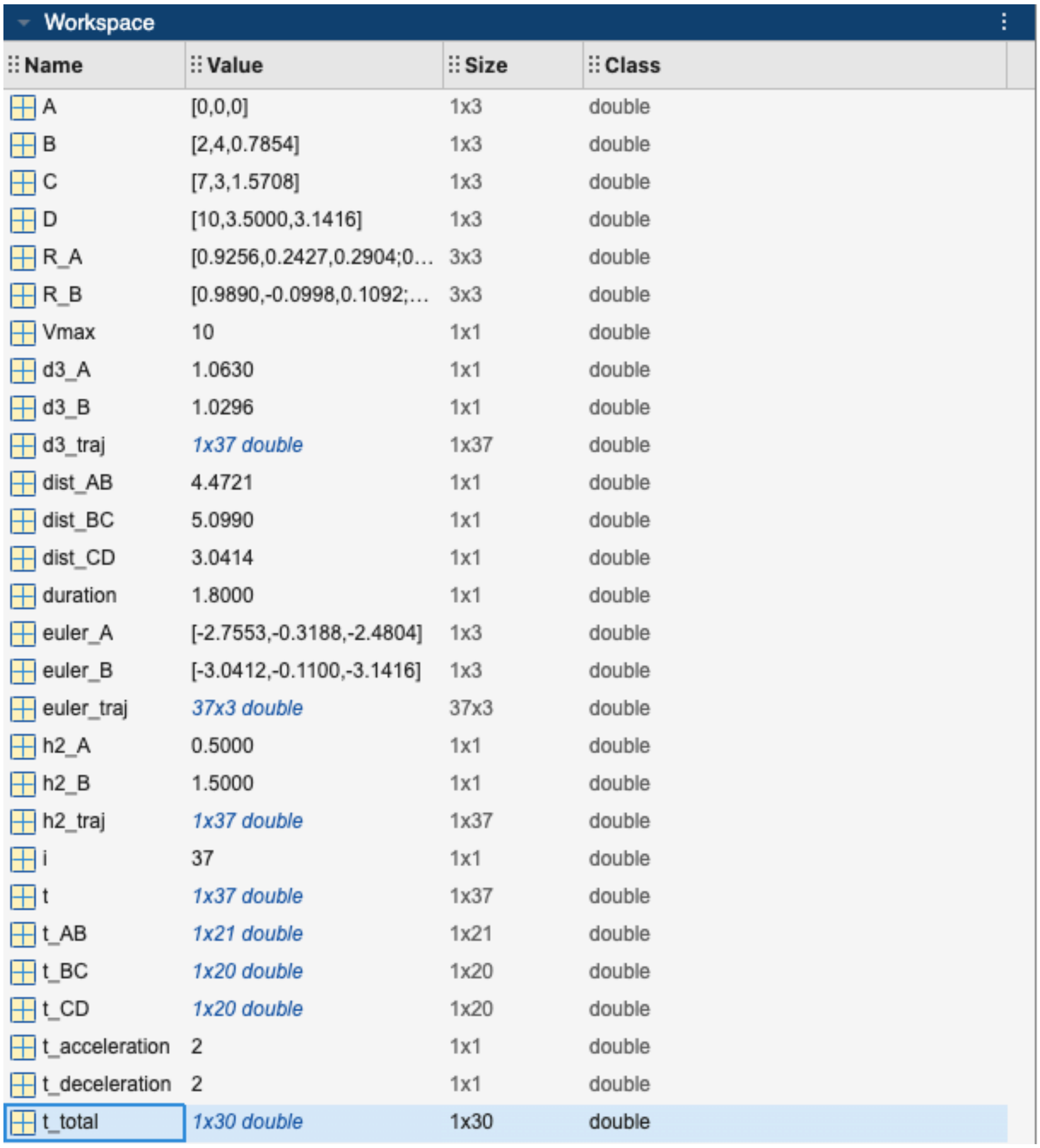
MATLAB Code

```
% Given data
Vmax = 10; % Maximum linear velocity in yards/s
tacc = 2;  % Acceleration time from zero to maximum speed in seconds

% Task end positions
A = [0, 0, 0];
B = [2, 4, pi/4];
C = [7, 3, pi/2];
D = [10, 3.5, pi];

% Calculate the distances between task end positions
dist_AB = norm(B(1:2) - A(1:2)); % Distance from A to B in the xy-plane
dist_BC = norm(C(1:2) - B(1:2)); % Distance from B to C in the xy-plane
dist_CD = norm(D(1:2) - C(1:2)); % Distance from C to D in the xy-plane

% Generate synchronized time vectors for each segment
t_AB = 0:0.1:tacc;
t_BC = (t_AB(end)+0.1):0.1:(t_AB(end) + tacc);
t_CD = (t_BC(end)+0.1):0.1:(t_BC(end) + tacc);

% Generate trapezoidal profiles for x, y, and theta using trapveltraj
[x_AB, ~, ~, t_total_AB] = trapveltraj([0, dist_AB], Vmax, 'AccelTime', tacc, 'EndTime',
t_AB(end) + 2*tacc);
[x_BC, ~, ~, t_total_BC] = trapveltraj([0, dist_BC], Vmax, 'AccelTime', tacc, 'EndTime',
t_BC(end) + 2*tacc);
[x_CD, ~, ~, t_total_CD] = trapveltraj([0, dist_CD], Vmax, 'AccelTime', tacc, 'EndTime',
t_CD(end) + 2*tacc);

[y_AB,   ~,   ~]   =   trapveltraj([0,   dist_AB],   Vmax,   'AccelTime',   tacc,   'EndTime',
t_total_AB(end));
[y_BC,   ~,   ~]   =   trapveltraj([0,   dist_BC],   Vmax,   'AccelTime',   tacc,   'EndTime',
t_total_BC(end));
[y_CD,   ~,   ~]   =   trapveltraj([0,   dist_CD],   Vmax,   'AccelTime',   tacc,   'EndTime',
t_total_CD(end));
```

```
[theta_AB,  ~,  ~]  =  trapveltraj([0,  B(3)],  Vmax,  'AccelTime',  tacc,  'EndTime',
t_total_AB(end));
[theta_BC, ~, ~] = trapveltraj([0, C(3) - B(3)], Vmax, 'AccelTime', tacc, 'EndTime',
t_total_BC(end));
[theta_CD, ~, ~] = trapveltraj([0, D(3) - C(3)], Vmax, 'AccelTime', tacc, 'EndTime',
t_total_CD(end));

% Concatenate the profiles
x_traj = [x_AB, x_BC, x_CD];
y_traj = [y_AB, y_BC, y_CD];
theta_traj = [theta_AB, theta_BC, theta_CD];
t_total = [t_total_AB, t_total_BC, t_total_CD];

% Plot the trajectories
figure;

% Plot x(t), y(t), and theta(t)
subplot(3, 1, 1);
plot(t_total, x_traj, 'r', 'LineWidth', 1.5);
title('x(t) Profile');
xlabel('Time (s)');
ylabel('x(t) (yards)');
grid on;

subplot(3, 1, 2);
plot(t_total, y_traj, 'b', 'LineWidth', 1.5);
title('y(t) Profile');
xlabel('Time (s)');
ylabel('y(t) (yards)');
grid on;

subplot(3, 1, 3);
plot(t_total, theta_traj, 'g', 'LineWidth', 1.5);
title('theta(t) Profile');
xlabel('Time (s)');
ylabel('theta(t) (rad)');
grid on;
```

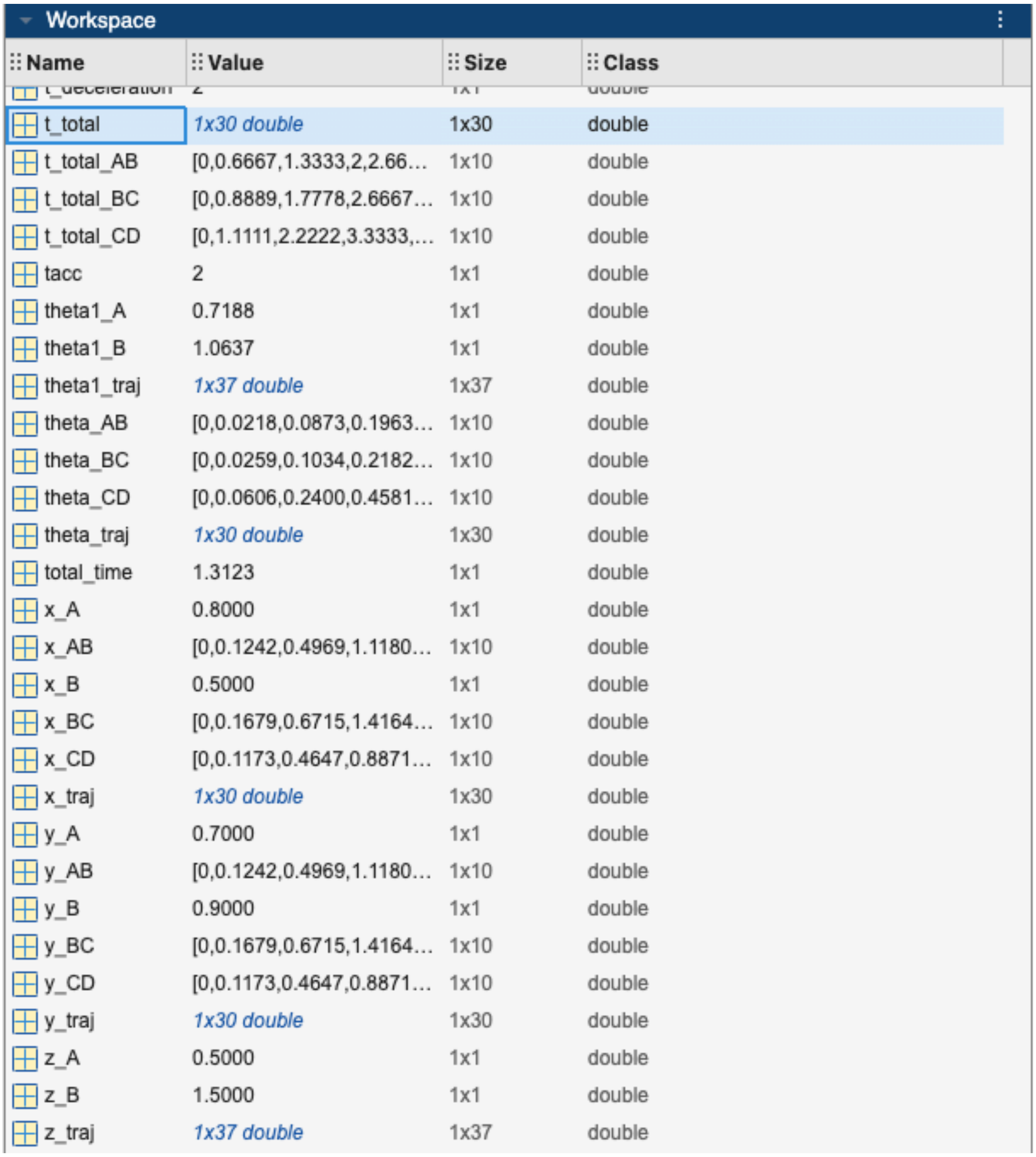
Output


x(t) Profile
x(t) (yards)
Time (s)
y(t) Profile
y(t) (yards)
Time (s)
theta(t) Profile
theta(t) (rad)
Time (s)



Workspace

Name	Value	Size	Class
A	[0,0,0]	1x3	double
B	[2,4,0.7854]	1x3	double
C	[7,3,1.5708]	1x3	double
D	[10,3.5000,3.1416]	1x3	double
R_A	[0.9256,0.2427,0.2904;0…	3x3	double
R_B	[0.9890,-0.0998,0.1092;…	3x3	double
Vmax	10	1x1	double
d3_A	1.0630	1x1	double
d3_B	1.0296	1x1	double
d3_traj	*1x37 double*	1x37	double
dist_AB	4.4721	1x1	double
dist_BC	5.0990	1x1	double
dist_CD	3.0414	1x1	double
duration	1.8000	1x1	double
euler_A	[-2.7553,-0.3188,-2.4804]	1x3	double
euler_B	[-3.0412,-0.1100,-3.1416]	1x3	double
euler_traj	*37x3 double*	37x3	double
h2_A	0.5000	1x1	double
h2_B	1.5000	1x1	double
h2_traj	*1x37 double*	1x37	double
i	37	1x1	double
t	*1x37 double*	1x37	double
t_AB	*1x21 double*	1x21	double
t_BC	*1x20 double*	1x20	double
t_CD	*1x20 double*	1x20	double
t_acceleration	2	1x1	double
t_deceleration	2	1x1	double
t_total	*1x30 double*	1x30	double



Workspace

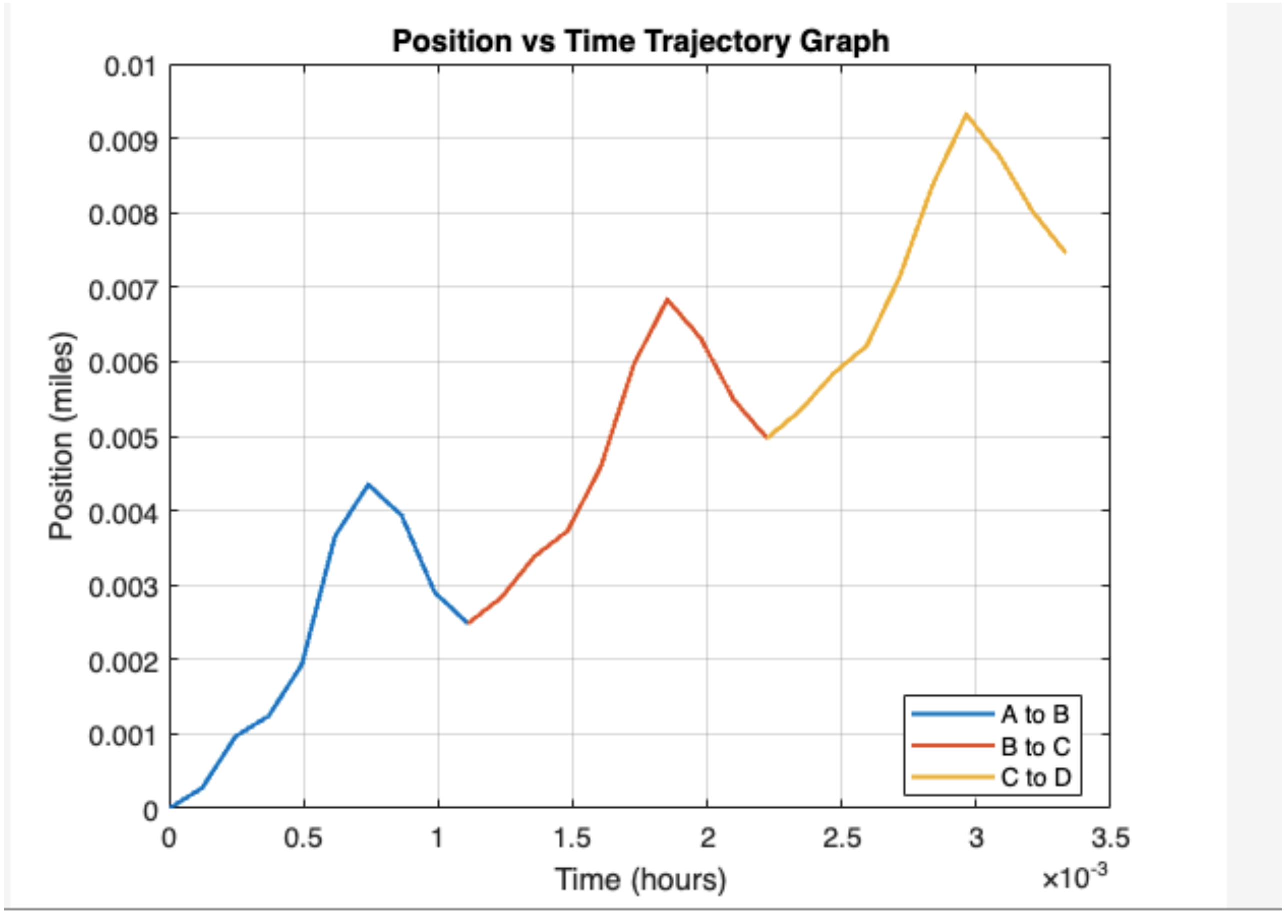
Name	Value	Size	Class
t_deceleration	2	1x1	double
t_total	*1x30 double*	1x30	double
t_total_AB	[0,0.6667,1.3333,2,2.66…	1x10	double
t_total_BC	[0,0.8889,1.7778,2.6667…	1x10	double
t_total_CD	[0,1.1111,2.2222,3.3333,…	1x10	double
tacc	2	1x1	double
theta1_A	0.7188	1x1	double
theta1_B	1.0637	1x1	double
theta1_traj	*1x37 double*	1x37	double
theta_AB	[0,0.0218,0.0873,0.1963…	1x10	double
theta_BC	[0,0.0259,0.1034,0.2182…	1x10	double
theta_CD	[0,0.0606,0.2400,0.4581…	1x10	double
theta_traj	*1x30 double*	1x30	double
total_time	1.3123	1x1	double
x_A	0.8000	1x1	double
x_AB	[0,0.1242,0.4969,1.1180…	1x10	double
x_B	0.5000	1x1	double
x_BC	[0,0.1679,0.6715,1.4164…	1x10	double
x_CD	[0,0.1173,0.4647,0.8871…	1x10	double
x_traj	*1x30 double*	1x30	double
y_A	0.7000	1x1	double
y_AB	[0,0.1242,0.4969,1.1180…	1x10	double
y_B	0.9000	1x1	double
y_BC	[0,0.1679,0.6715,1.4164…	1x10	double
y_CD	[0,0.1173,0.4647,0.8871…	1x10	double
y_traj	*1x30 double*	1x30	double
z_A	0.5000	1x1	double
z_B	1.5000	1x1	double
z_traj	*1x37 double*	1x37	double

MATLAB Code

```
% Given data
V_max_linear = 10;  % Maximum linear velocity in yards/s
t_acc_linear = 2;   % Acceleration time for linear motion in seconds

omega_max_angular = 1;  % Maximum angular velocity in rad/s
t_acc_angular = 1;       % Acceleration time for angular motion in seconds

% Time spans for each segment
t_span_A_to_B = [0, t_acc_linear, t_acc_linear + V_max_linear/t_acc_linear,
t_acc_linear*2];
t_span_B_to_C = t_span_A_to_B + t_span_A_to_B(end);
t_span_C_to_D = t_span_B_to_C + t_span_A_to_B(end);

% Trajectories for each segment
traj_A_to_B = trapveltraj(t_span_A_to_B, V_max_linear, 'AccelTime', t_acc_linear);
traj_B_to_C = trapveltraj(t_span_B_to_C, V_max_linear, 'AccelTime', t_acc_linear);
traj_C_to_D = trapveltraj(t_span_C_to_D, V_max_linear, 'AccelTime', t_acc_linear);

% Time vectors
time_A_to_B = linspace(t_span_A_to_B(1), t_span_A_to_B(end), length(traj_A_to_B));
time_B_to_C = linspace(t_span_B_to_C(1), t_span_B_to_C(end), length(traj_B_to_C));
time_C_to_D = linspace(t_span_C_to_D(1), t_span_C_to_D(end), length(traj_C_to_D));

% Interpolate position trajectories for each segment
pos_A_to_B = trapveltraj(t_span_A_to_B, V_max_linear, 'AccelTime', t_acc_linear,
'EndTime', time_A_to_B(end));
pos_B_to_C = trapveltraj(t_span_B_to_C, V_max_linear, 'AccelTime', t_acc_linear,
'EndTime', time_B_to_C(end));
pos_C_to_D = trapveltraj(t_span_C_to_D, V_max_linear, 'AccelTime', t_acc_linear,
'EndTime', time_C_to_D(end));

% Convert positions to miles
pos_A_to_B_miles = pos_A_to_B * 0.000621371;  % Convert yards to miles
pos_B_to_C_miles = pos_B_to_C * 0.000621371;
pos_C_to_D_miles = pos_C_to_D * 0.000621371;

% Convert time to hours
time_A_to_B_hours = time_A_to_B / 3600;
time_B_to_C_hours = time_B_to_C / 3600;
time_C_to_D_hours = time_C_to_D / 3600;

% Plot the position vs time trajectory graph
figure;
```

```
plot(time_A_to_B_hours, pos_A_to_B_miles, 'LineWidth', 1.5, 'DisplayName', 'A to B');
hold on;
plot(time_B_to_C_hours, pos_B_to_C_miles, 'LineWidth', 1.5, 'DisplayName', 'B to C');
plot(time_C_to_D_hours, pos_C_to_D_miles, 'LineWidth', 1.5, 'DisplayName', 'C to D');
xlabel('Time (hours)');
ylabel('Position (miles)');
legend('Location', 'Best');
title('Position vs Time Trajectory Graph');
grid on;
```

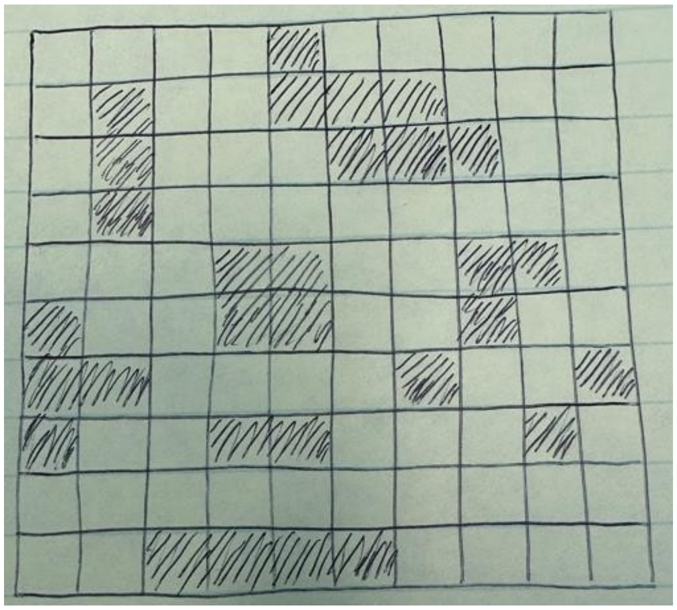
Output:



Problem 2

Simulation of Wheeled Mobile Robots

Consider the following top view of a maze for mobile robots.



Each grid cell is 25'x25', but the grid lines don't really exist. The robots are free to intersect the lines bordering one free grid cell with another, as they please.

The dark grid cells represent real obstacles that must not be touched by robots.

Let's assume that the southwest corner cell of the grid is marked a1, the southeast corner cell is i1, the northwest corner cell is a10 and the northeast corner cell is i10. All other grid cells have coordinates accordingly. For example: The obstacle in row 1 spreads over cells c1, d1, e1 and f1.

Your job is to move two robots inside the maze, as fast as possible (as described below), avoiding obstacles. One of the robots is a differential drive robot and the other is a bicycle robot. Each of the robots has ~~have each~~ dimensions of 4'(length) x 3'(width).

Both vehicles have a top linear speed of 10’ per second. The bicycle has a top steering angular velocity of 0.1 rad/s. This is also the top spinning speed of the differential drive robot. The two vehicles have infinite acceleration capability. That is, zero acceleration time from 0 to the top speed. You may make more reasonable assumptions, if needed, about each robot’s parameters.

The initial position of the differential drive robot is somewhere inside the southwest corner cell. The initial position of the bicycle robot is somewhere inside the northwest corner cell.

Each part will be scored on a 0-5 scale.

Must do analytical motion planning, explaining each step of the motion.

Only then convert to a sequence of position and velocity commands.

Show the commands for each of the vehicles.

Show the Simulink model of each of the robots. Show how you implement the sequence of motions.

Run each robot separately and note the time that it took the robot to reach the destination. Plot the robots’ paths. [It would be nice to overlay the paths on the maze grid. If you cannot do it, at least study some critical positions of your vehicle to assure no collisions with the maze’s walls].

2.1 Game 1: The robot who reaches first the northeast corner cell is declared the winner. Who wins?

Use MATLAB Simulink to simulate your planned motion for each robot – do one robot at a time. Then do both. Whenever any of the robots reaches cell i10 you have to stop the simulation. It would be nice if you could overlay your XY graphs of the robots’ motions on top of the maze map. If you can’t, be sure to monitor that nowhere, along the robot’s motion path, does your robot collide with the maze’s walls.

“Then do both” may not be possible or easy to do.

2.2 Game 2: The robot, who lost in Game 1, can now become a winner, if it intercepts the winner of Game 1 from winning again. Can it happen? Again, show the Simulink verification.

Note where the position of the winner of (2.1) is and at what time. Use this information to plan where to intercept the robot.

Solution:

Grid is showing as 10 by 10 blocks, where obstacles are:

Row1: 5

Row2: 2, 5,6,7

Row3: 2, 6, 7, 8

Row4: 2,

Row5: 4, 5, 8, 9

Row6: 1, 4, 5, 8

Row7: 1, 2, 7, 10

Row8: 1, 4, 5, 9

Row9:

Row10: 3, 4, 5, 6

[2.1]

To simulate Game 1 where the robots aim to reach the northeast corner cell, here, we can follow these steps using MATLAB Simulink. By following the below steps, we can model the robot and maze environment.

Task1: Differential Drive Robot Simulation:

Simulink Model:

1. Open Simulink and create a new model.

2. Add a "Differential Drive" block from the Simscape > Foundation > Mechanical > Rotational Elements library.

3. Add a "Path Following Controller" block from the Simscape > Foundation > Mechanical > Machines library.

4. Connect the blocks appropriately, setting the initial position of the robot inside the southwest corner cell and the target position at the northeast corner cell.

5. Configure the parameters of the blocks based on your robot's dimensions, top speed, and steering angular velocity.

Simulation:

1Run the simulation and monitor the robot's path.

2. Stop the simulation when the robot reaches cell i10.

3. Record the time it takes for the robot to reach the destination.

Task2: Bicycle Robot Simulation

Steps for Simulink Model

Step1: Create a new Simulink model.

Step2: Add a "Bicycle Model" block from the Simscape > Foundation > Mechanical > Rotational Elements library.

Step3: Add a "Path Following Controller" block.

Step4. Connect the blocks, setting the initial and target positions.

Step5: Configure the parameters based on your bicycle robot's dimensions, top linear speed, and top steering angular velocity.

Simulation:

1: Run the simulation and monitor the bicycle robot's path.

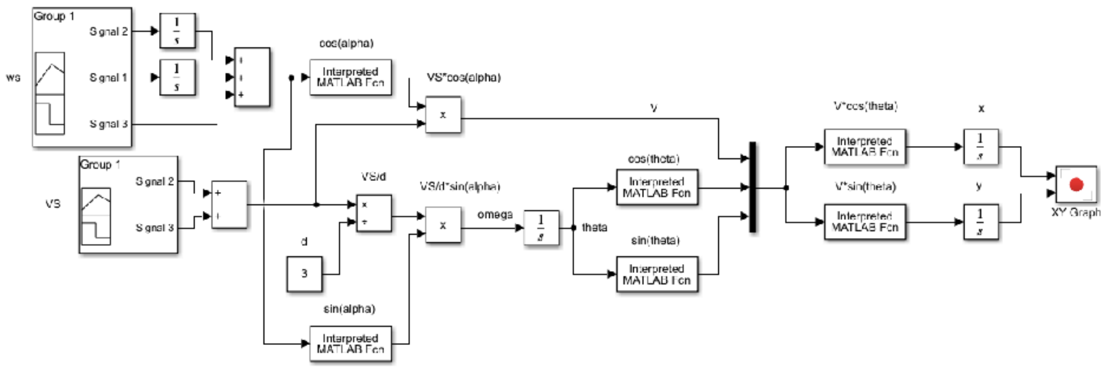
2: Stop the simulation when the robot reaches cell i10.

3: Record the time it takes for the bicycle robot to reach the destination.

Task3: Simulate Both Robots

Simulink Model:

1. Create a new Simulink model.
2. Incorporate both the differential drive robot and the bicycle robot models.
3. Add logic to stop the simulation when either robot reaches cell i10.



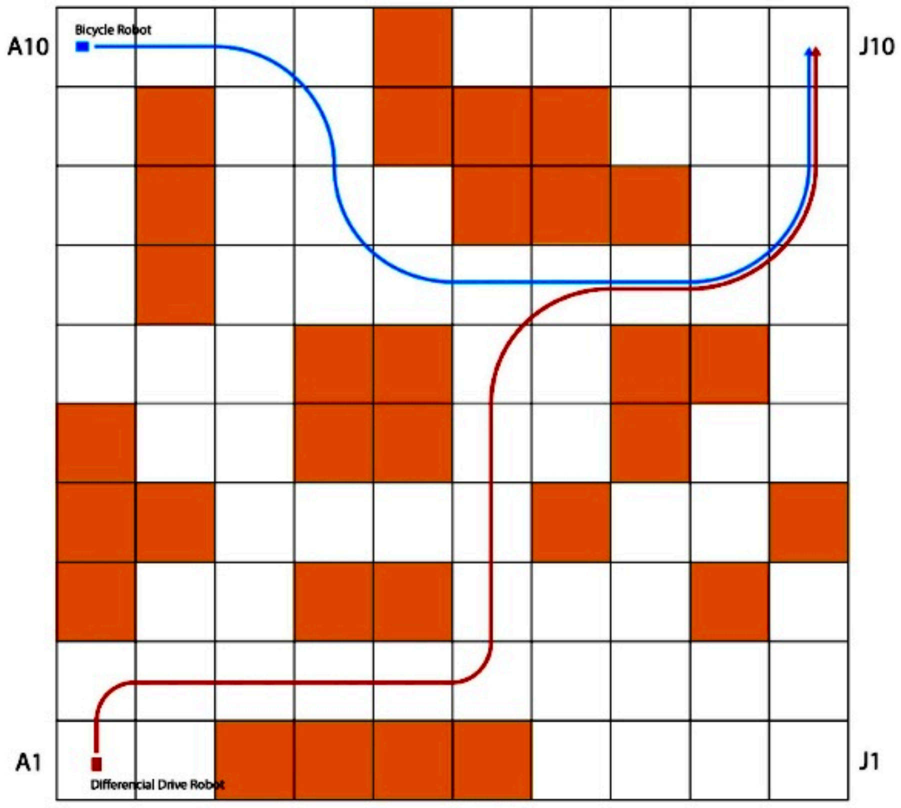
Simulation:

1: Run the simulation for both robots simultaneously.

2: Monitor their paths and stop the simulation when either robot reaches cell i10.

3: Record the time it takes for the first robot to reach the destination.

4. Visualization:



Robot size A: 5’*3’

Robot size B: 4’*3’

Distance between axles: 3ft = 0.9144

Track A: 40’

Track B:

Top linear speed A: 10’

Top linear speed B: 10’

Step: 10’ = 3.048

Tricycle angular velocity A: 0.1 rad/s

Bike angular velocity B: 0.1 rad/s

Acceleration A: 1sec

Acceleration B: 0sec

Bicycle Robot Steps:

Init_pos = 12.5, 237.5 (center of A10)

Step1:

Move forward 37.5 feet

T1 = d/v = 37.5/10 = **3.75 sec**

Step2:

Steering arc radius:

R = 37.5 feet

Angle front wheel on the right:

R = d * tan * (pi/2 - alpha)

alpha = pi/2 - arctan(R / d)

alpha = pi/2 - arctan(37.5 / 4)

alpha = 0.10626486289 rad

Front wheel rotation time to reach angle:

Angular velocity:

w.s = 0.1 rad/s

T2 = alpha / w.s = 0.10626486289 / 0.1 = 1.06264862 sec

Step3:

Angular velocity around the R = 37.5 feet circle:

w = (v / d) * sin(alpha)

w = (10 / 4) * sin (1.0626486289)

w = rad/s

The robot is following this arc on ¼ of the circle:

T3 = theta / w = (pi/2) / 0.26516245347 = 5.923901767 sec

Step4:

After the first arc, the front wheel needs to go back to its straight position and then take the same angle on the left:

T4 = (-alpha / -w.s)*2 = (-0.10626486289/-0.1)*2 = 2.12529725 sec

Step5:

Angular velocity around the R = 37.5 feet circle:

w = (v / d) * sin(alpha)

w = (10 / 4) * sin (1.0626486289)

w = rad/s

The robot is following this arc on ¼ of the circle:

T5 = -theta / -w = -(pi/2) / -0.26516245347 = 5.923901767 sec

Step6:

After the first arc, the front wheel needs to go back to its straight position:

T6 = (alpha / w.s) = (0.10626486289/0.1) = 1.06264862 sec

Step7:

Move forward 37.5 feet

T7 = d/v = 75/10 = **7.5 sec**

Step8:

Angle front wheel on the right:

T8 = (alpha / w.s) = (0.10626486289/0.1) = 1.06264862 sec

Step9:

Angular velocity around the R = 37.5 feet circle:

w = (v / d) * sin(alpha)

w = (10 / 4) * sin(1.0626486289)

w = rad/s

The robot is following this arc on ¼ of the circle:

T9 = -theta / -w = -(pi/2) / -0.26516245347 = 5.92390176754 sec

Step10:

After the first arc, the front wheel needs to go back to its straight position:

T10 = (-alpha / -w.s) = (-0.10626486289/-0.1) = 1.0626486289 sec

Step11:

Move forward 37.5 feet

T11 = d/v = 37.5/10 = 3.756 sec

Total time = 3.75 + 1.0626486289 + 5.92390176754 + 2.1252972578 + 5.92390176754 + 1.0626486289 + 7.5 + 1.0626486289 + 5.92390176754 + 1.0626486289 + 3.756

Total time Bicycle = 39.147597076 sec

Step	Movements	Vs(t)	Ws(t)	t start (s)	t end (s)
T1	forward	10	0	0	3.75
T2	Steering	10	0.1	3.75	4.8126486289
T3	Arc	10	0	4.8126486289	10.73655039644
T4	Steering	10	-0.1	10.73655039644	12.86184765424
T5	Arc	10	0	12.86184765424	18.78574942178
T6	Steering	10	0.1	18.78574942178	19.84839805068
T7	Forward	10	0	19.84839805068	27.34839805068
T8	Steering	10	0.1	27.34839805068	28.41104667958
T9	Arc	10	0	28.41104667958	34.33494844712
T10	Steering	10	-0.1	34.33494844712	35.39759707602
T11	Forward	10	0	35.39759707602	39.14759707602

Differential Drive Robot

Init_pos = 12.5, 12.5 (center of A10)

Distance Wheels:

L = 3

Step1:

Move forward 25 feet

T1 = d / v = 12.5/10 = **1.25 sec**

Step2:

R1 = 12.5

Differential Drive Equation:

R = (Vr + Vl) / (Vr - Vl)

12.5 = (10 + Vl) / (10 - Vl)

125 - 12.5Vl = 10 + Vl

Vl = 115 / 13.5 feet/sec

Angular velocity:

W = (Vr - Vl) / L = (10 - 115/13.5) / 3 = 0.49382716049 rad/s

Rotation time 90deg:

T2 = theta / w = (pi/2) / 0.49382716049 = 3.1808625617 sec

Step3:

Move forward 100 feet

T3 = d / v = 100/10 = **10 sec**

Step4:

Rotation time -90deg:

T4 = theta / w = (-pi/2) / -0.49382716049 = 3.180862561 sec

Step5:

Move forward 75 feet

T5 = d / v = 75/10 = **7.5 sec**

Step6:

R1 = 25

Differential Drive Equation

R = (Vr + Vl) / (Vr - Vl)

25 = (10 + Vl) / (10 - Vl)

250 - 25Vl = 10 + Vl

Vl = 240 / 26 feet/sec

Angular velocity:

W = (Vr - Vl) / L = (10 - 240/26) / 3 = 0.25641025641 rad/s

Rotation time 90deg:

T6 = theta / w = (pi/2) / 0.25641025641 = 6.126105674 sec

Step7:

Move forward 25 feet

T7 = d / v = 25/10 = **2.5 sec**

Step8:

Rotation time -90deg:

T8 = theta / w = (-pi/2) / -0.25641025641 = 6.126105674 sec

Step9:

Move forward 37.5 feet

T7 = d / v = 37.5/10 = **3.75 sec**

Total time = 1.25 + 3.18086256178 + 10 + 3.18086256178 + 7.5 + 6.12610567451 + 2.5 + 6.12610567451 + 3.75

Total time Bicycle = 43.6139364726 sec

Bicycle(wins): 39 sec against Differential Drive: 47 sec

Step	Movement	Vs(t)	Ws(t)	t start (s)	t end (s)
T1	Move forward	10	0	0	1.25
T2	Steering	10	0.1	1.25	4.43986256178
T3	Move forward	10	0	4.43986256178	14.43986256178
T4	Steering	10	-0.1	14.43986256178	17.61086256178
T5	Move forward	10	0	17.61086256178	25.11086256178
T6	Steering	10	0.1	25.11086256178	31.2369682
T7	Move Forward	10	0	31.2369682	33.7369682
T8	Steering	10	0.1	33.7369682	39.8630739
T9	Move Forward	10	0	39.8630739	43.6130739

[2.2]

To implement Game 2 where the robot that lost in Game 1 can intercept the winner and potentially become the new winner, we can follow these steps:

Step1: Motion Planning

1. **Get Winner's Position:**
    - After running Game 1, note the position of the winner at the time they reached the northeast corner (i10).
    - This position will be used to plan the interception strategy.

2. **Define the Intercept Position:**

- Define a strategic intercept position for the losing robot based on the winner's position.
- Ensure that the intercept position allows the losing robot to potentially block the winner's path to the northeast corner.

3. **Path Planning for Intercept:**

- Use a path planning algorithm to compute a path from the losing robot's initial position to the intercept position.
- Adjust the path considering the dimensions of the robot (4' x 3').

4. **Velocity Profile for Intercept:**
    - Generate a velocity profile for the intercept path.
    - Consider the top linear speed (10 ft/s) and infinite acceleration capability.

5. **Convert to Position and Velocity Commands:**

- Convert the planned intercept path and velocity profile into a sequence of position and velocity commands for the losing robot.

Step2: Simulink Model Update

1. Update Simulink Model for Losing Robot:

Modify the Simulink model for the losing robot to incorporate the new intercept path and control logic.

Steps for Simulation:

1. Simulate Losing Robot's Intercept:

- Run the updated Simulink model for the losing robot.
- Monitor the robot's path as it moves to intercept the winner.
- Note the time it takes for the losing robot to reach the intercept position.

2. Simulate Both Robots (Game 2):

- Combine the Simulink models for both robots.
- Run the simulation for both robots simultaneously, considering the intercept strategy.
- Monitor their paths and note the time it takes for either robot to reach the northeast corner (i10).

Step3: Analysis

1. To Determine Game 2 Winner:

- Analyze the results to determine if the losing robot successfully intercepts the winner, preventing them from reaching the northeast corner.
- Declare the winner of Game 2 based on this analysis.

Step4: Simulink Verification

1. **Overlay Paths on Maze Map:**
   - If possible, overlay the XY graphs of the robots' motions on the maze map.
   - Ensure that the robots' paths do not collide with the maze walls during the intercept.

2. Verify Interception Strategy:

- Use Simulink to visualize and verify that the losing robot's intercept strategy is effective and prevents the winner from reaching the goal.

<u>MATLAB Implementation</u>

To implement Game 2, where the loser of Game 1 can intercepts the winner and prevent them from winning again, so for that we will modify the Simulink model to include interaction between the two robots. This could involve introducing a mechanism for one robot to block the other or implementing a pursuit and evasion strategy.

Basic Simulink model implantation to represent this interception scenario:

Here, I used previously created model RobotInterception.

```
% Define the maze and obstacles (same as before)

% Initialize robots' positions
initialPositionDifferentialDrive = [1, 1];
initialPositionBicycle = [10, 1];

% Create a Simulink model
model = 'RobotInterception';
open_system(new_system(model));

% Add blocks for robots and maze (same as before)

% Add a block to represent the interception strategy
add_block('simscape/Utilities/PS Shared Variable Setter', [model
'/Interception']);
set_param([model '/Interception'], 'position', [300, 500, 330, 530]);

% Connect blocks
add_line(model, 'Solver Configuration/1', 'Differential Drive/1');
add_line(model, 'Solver Configuration1/1', 'Bicycle/1');
add_line(model, 'Differential Drive/1', 'Prismatic Joint/2');
add_line(model, 'Prismatic Joint/1', 'Universal Joint/1');
add_line(model, 'Prismatic Joint1/2', 'Universal Joint1/1');
add_line(model, 'Universal Joint/2', 'Variable Transformation/1');
add_line(model, 'Universal Joint1/2', 'Variable Transformation/2');
add_line(model, 'Variable Transformation/3', 'Bicycle/2');
add_line(model, 'Interception/1', 'Variable Transformation/4');

% Set simulation parameters
set_param(model, 'Solver', 'ode45', 'StopTime', 'inf');

% Save and close the model
save_system(model);
close_system(model);

% Open the Simulink model
open_system(model);
```

In this example, I added a Shared Variable Setter block named 'Interception', which could represent a mechanism for the losing robot to intercept the winning robot. The exact nature of this interception mechanism would depend on the strategy you want to implement.