ROBOTICS SYSTEMS

ASSIGNMENT 3

Student Names:	Yuk-Cheung Jonathan Wong
	Samuel Wong
	Xuteng Lin
Student ID:	1021374
	832280
	813146

MCEN90028
Semester One 2020
University of Melbourne

1st May 2020

1. Introduction

1.1 Brief Description of Assignment

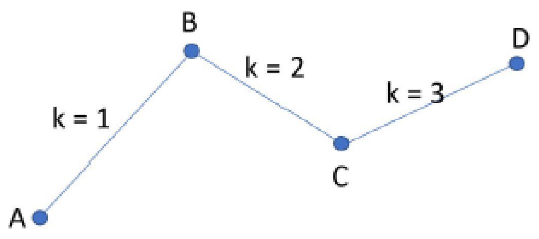

Figure 1: Given Example of Trajectory Course

As shown in figure 1, each via-point ($Vpoints$) is denoted with a letter (from A to D) where letter A is the start and D is the finish. Each segment between the via-points is a segment (k), where the maximum number of segments (k_{max}) is equivalent to the following equation:

$$k_{max} = \sum_{n} Vpoints - 1 \qquad (1)$$

Where n denotes the total number of via points (Vpoints).

A polynomial shown in figure 2 will be implemented to generate trajectory between points (in each segment):

$$x(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_P t^P \qquad (2)$$

Where $a_0, a_1 \ldots a_P$ are the coefficients of the polynomial, and where P denotes the maximum order of the polynomial (where a polynomial with $P = 3$ is cubic and $P = 5$ is quantic).

2. Assignment Tasks

2.1 Task 1: Trajectory Generation

2.1.1 Task Description

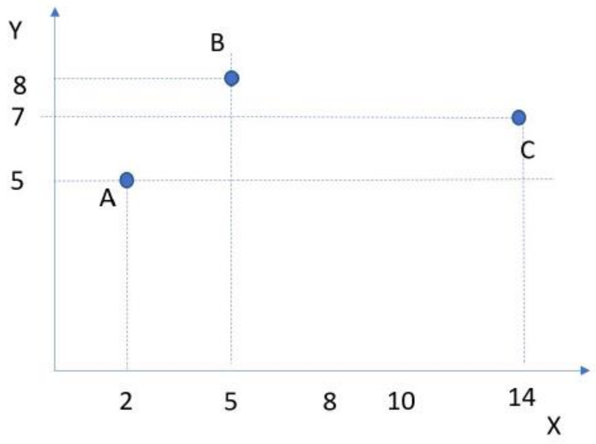

Figure 2: Given Coordinates of 3 Via-Points

As seen from figure 2, the coordinates of the via-points ($A(2,5), B(5,8), C(14,7)$) are displayed on a 2D plane (with an x and y axis). The following table depicts all the constraints that will be followed for this task:

Constraint Number:	**Constraint:**
1	Velocity at via-point B is 0.25m/s in the X-direction
2	The time duration for segment 1 (between A and B) is 3 seconds, for segment 2 (between B and C) is 5 seconds.

Where the robotic arm is to start at rest from point A and to finish at rest at point C, moving through the points chronologically (from A to B to C).

2.1.2 Approach

For this task, cubic splines were used to create the trajectory as the only variables of interest to calculate trajectory are the position and velocity of the end-effector (where quantic also involves acceleration). Here, the equation from one point to another (example from A to B as seen in equation 2) would be the following:

$$x_{AB} = a_{AB,3}t^3 + a_{AB,2}t^2 + a_{AB,1}t + a_{AB,0} \tag{3}$$
$$\dot{x}_{AB} = 3a_{AB,3}t^2 + 2a_{AB,2}t + a_{AB,1} \tag{4}$$

By breaking this down further into matrix form (where the coefficients and the time variables are split), a matrix equation can be formed:

$$\begin{bmatrix} x_{AB}(t_{AB,i}) \\ \dot{x}_{AB}(t_{AB,i}) \\ x_{AB}(t_{AB,f}) \\ \dot{x}_{AB}(t_{AB,f}) \end{bmatrix} = \begin{bmatrix} {t_{AB,i}}^3 & {t_{AB,i}}^2 & {t_{AB,i}}^1 & 1 \\ 3{t_{AB,i}}^2 & 2t_{AB,i} & 1 & 0 \\ {t_{AB,f}}^3 & {t_{AB,f}}^2 & {t_{AB,f}}^1 & 1 \\ 3{t_{AB,f}}^2 & 2t_{AB,f} & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{AB,3} \\ a_{AB,2} \\ a_{AB,1} \\ a_{AB,0} \end{bmatrix} \tag{5}$$

Where the variables $t_{AB,i}$ is the initial time of trajectory from A to B, and $t_{AB,f}$ is the final time between the two points. By rearranging equation 5, the coefficients can be found easily:

$$\begin{bmatrix} a_{AB,3} \\ a_{AB,2} \\ a_{AB,1} \\ a_{AB,0} \end{bmatrix} = \begin{bmatrix} {t_{AB,i}}^3 & {t_{AB,i}}^2 & {t_{AB,i}}^1 & 1 \\ 3{t_{AB,i}}^2 & 2t_{AB,i} & 1 & 0 \\ {t_{AB,f}}^3 & {t_{AB,f}}^2 & {t_{AB,f}}^1 & 1 \\ 3{t_{AB,f}}^2 & 2t_{AB,f} & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} x_{AB}(t_{AB,i}) \\ \dot{x}_{AB}(t_{AB,i}) \\ x_{AB}(t_{AB,f}) \\ \dot{x}_{AB}(t_{AB,f}) \end{bmatrix} \tag{6}$$

This equation also applies to find the y-axis trajectory as well.

$$\begin{bmatrix} b_{AB,3} \\ b_{AB,2} \\ b_{AB,1} \\ b_{AB,0} \end{bmatrix} = \begin{bmatrix} {t_{AB,i}}^3 & {t_{AB,i}}^2 & {t_{AB,i}}^1 & 1 \\ 3{t_{AB,i}}^2 & 2t_{AB,i} & 1 & 0 \\ {t_{AB,f}}^3 & {t_{AB,f}}^2 & {t_{AB,f}}^1 & 1 \\ 3{t_{AB,f}}^2 & 2t_{AB,f} & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} y_{AB}(t_{AB,i}) \\ \dot{y}_{AB}(t_{AB,i}) \\ y_{AB}(t_{AB,f}) \\ \dot{y}_{AB}(t_{AB,f}) \end{bmatrix} \tag{7}$$

To compute the trajectory between points A to B and B to C, the following inputs were used to calculate the necessary coefficients.

Trajectory	Point A to B (Segment AB)	Point B to C (Segment BC)
Initial time ($t_{AB,i}/s$)	0	3
Final time ($t_{AB,f}/s$)	3	8
Initial Position $(x, y)\ /m$	(2, 5)	(5,8)
Final Position $(x, y)\ /m$	(5, 8)	(14, 7)
Initial Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0	0.25
Final Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0.25	0

From the above table, it is to note that the following relationship is assumed:

$$\dot{x}_{AB}\left(t_{AB,f}\right) = \dot{x}_{BC}(t_{BC,i}) \tag{8}$$

$$x_{AB}\left(t_{AB,f}\right) = x_{BC}(t_{BC,i}) \tag{9}$$

Where the final position and velocity time and displacement of a trajectory end point is the initial position and velocity time and displacement of the following trajectory start point (as seen in equations 8 and 9, for a trajectory from A to B, the final values of such trajectory are the initial values for the following trajectory from B to C).

2.1.3 Polynomial Results

The coefficients for the trajectories obtained for task 1 for the x axis is the following:

$$\begin{bmatrix} a_{AB,3} \\ a_{AB,2} \\ a_{AB,1} \\ a_{AB,0} \\ a_{BC,3} \\ a_{BC,2} \\ a_{BC,1} \\ a_{BC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 2 \\ -0.134 \\ 2.186 \\ -9.248 \\ 16.688 \end{bmatrix} \tag{10}$$

The coefficients for the trajectories obtained for task 1 for the y axis is the following:

$$\begin{bmatrix} b_{AB,3} \\ b_{AB,2} \\ b_{AB,1} \\ b_{AB,0} \\ b_{BC,3} \\ b_{BC,2} \\ b_{BC,1} \\ b_{BC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 5 \\ 0.026 \\ -0.454 \\ 2.272 \\ 4.568 \end{bmatrix} \tag{11}$$

With these computed coefficients, the following polynomial equations are formed to generate the trajectories for task 1 (positions for x and y):

$$x_{AB} = -0.1944t^3 + 0.9167t^2 + 2 \tag{12}$$

$$x_{BC} = -0.134t^3 + 2.186t^2 - 9.248t + 16.688 \tag{13}$$

$$y_{AB} = -0.1944t^3 + 0.9167t^2 + 5 \tag{14}$$

$$y_{BC} = 0.026t^3 - 0.4540t^2 + 2.272t + 4.568 \tag{15}$$

The velocity equations are also computed as the following polynomials:

$$\dot{x}_{AB} = -0.5833t^2 + 1.833t \tag{16}$$
$$\dot{x}_{BC} = -0.402t^2 + 4.372t - 9.248 \tag{17}$$
$$\dot{y}_{AB} = -0.5833t^2 + 1.833t \tag{18}$$
$$\dot{y}_{BC} = 0.078t^2 - 0.908t + 2.272 \tag{19}$$

2.1.4 Graphical Results

With the above polynomials, the generated trajectory is visualised in the following graph (figure 3).

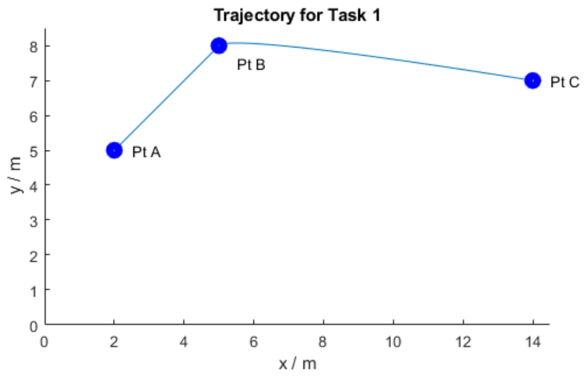

Figure 3: Computed Trajectory of Task 1

Where the displacement and velocity graphs for both x and y are also plotted as seen from figures 4 to 7.

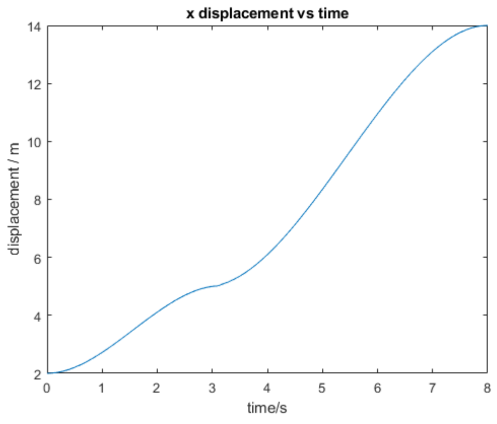

Figure 4: x Displacement Over Time of Trajectory

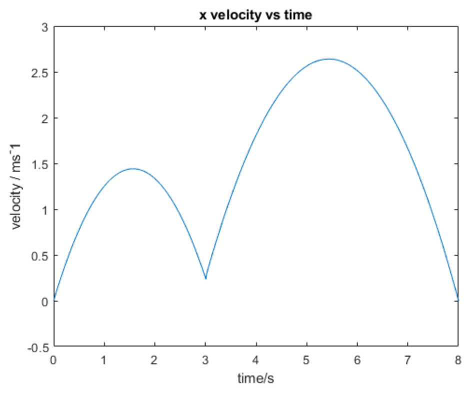

Figure 5: x Velocity Over Time of Trajectory

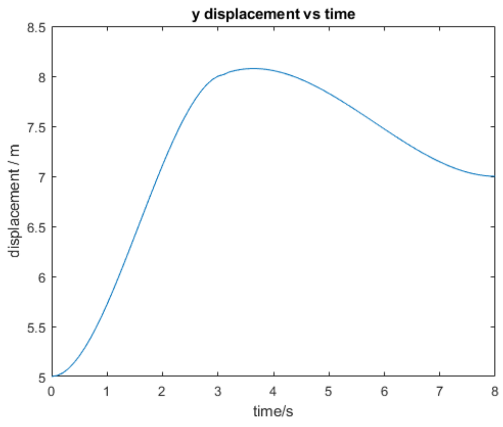

Figure 6: y Displacement Over Time of Trajectory

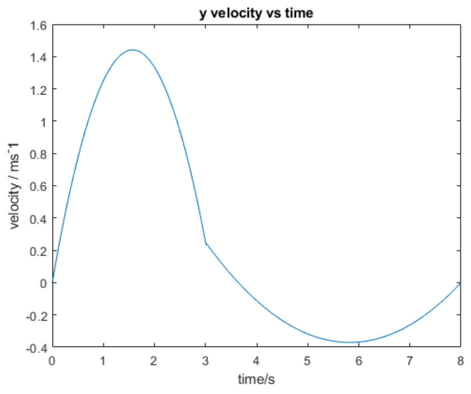

Figure 7: y Velocity Over Time of Trajectory

2.2 Task 2: Trajectory Generation with Obstacle

2.2.1 Task Description

Similar to task 1, however, the trajectory in this task must take in account of an obstacle (Obs1) situated between points B and C. A new via-point (D) is to be added such that a new trajectory may be created to avoid the obstacle. Additionally, conditions such as ensure segments BD and DC are straight lines which do not collide with Obs1 (Condition 2A), and minimized total distance travelled (Condition 2B) must be considered.

2.2.2 Approach to Find Point D

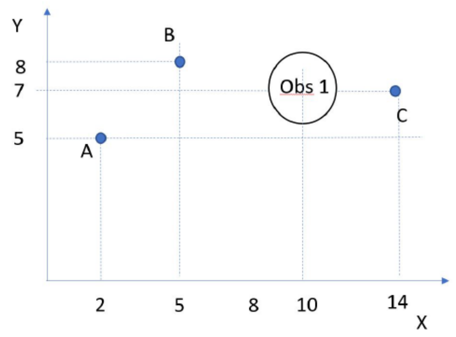

Figure 8: The Given Coordinates of the Obstacle and Via Points

With reference to figure 8, the via-point D must be located in order for the trajectory to be generated. To find this point, the following steps of logic was implemented in a separate function called "*Assignment3_searchD*":

i. Generate the circle (Obs1) and points B and C on a 2D Plane as seen in figure 8.
ii. A grid search will be initiated where the grid is determined from the radius of the obstacle (as shown in figure 9):

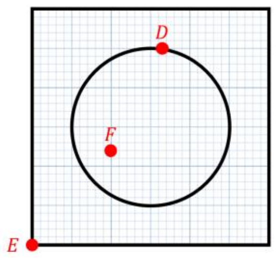

Figure 9: Plotted Grid and Obstacle

iii. The grid search will begin at point E and search through all points (with 0.1 as the resolution) within the grid, and subsequently compares the paths BD and DC (straight lines) of the points such that:
    a. If the point or the path generated is within the circle, it is discarded (Condition 2A)
    b. If the point has a longer computed length than the shortest recorded length, it is discarded
    c. If the point has the shortest computed length, the point is recorded. (Condition 2B)

    Where the values are computed between a set parameter multiplied by the radius about the x and y position of the circle (e.g point E x and y is $(x_{center\ of\ obstacle} - 1.5 * radius, y_{center\ of\ obstacle} - 1.5 * radius)$).

iv. Once the shortest path is found (between BD and DC), the point D is recorded and output to the main function where it will be used to calculate trajectory.
v. Furthermore, this "*searchD*" function is run again in the main function about the located point D such that a higher resolution point D may be found (grid searches around the located point D for the shortest length with 0.01 as the new resolution).

From the steps above, point D is located at a high resolution (within 0.01) such that the shortest path that does not overlap the obstacle is computed and found: $Point\ D = (10.20, 8.54)$ as seen in figure 10. Refer to the Appendix for more detailed pseudocode of this process.

2.2.3 Generate Trajectory Based on Point D

Using the same function that was created in task 1, the new path that is generated will need to go from A to B, B to D, D to C. The new parameters are shown below:

Trajectory	**Point A to B (Segment AB)**	**Point B to D (Segment BD)**	**Point D to C (Segment DC)**
Initial time ($t_{AB,i}/s$)	0	3	5.5
Final time ($t_{AB,f}/s$)	3	5.5	8
Initial Position $(x, y)\ /m$	(2, 5)	(5,8)	(10.2, 8.54)
Final Position $(x, y)\ /m$	(5, 8)	(10.20, 8.54)	(14, 7)
Initial Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0	0.25	0
Final Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0.25	0	0

2.2.4 Polynomial Results

The resulting coefficients for the trajectory for task 2 in the x axis are shown below:

$$\begin{bmatrix} a_{AB,3} \\ a_{AB,2} \\ a_{AB,1} \\ a_{AB,0} \\ a_{BD,3} \\ a_{BD,2} \\ a_{BD,1} \\ a_{BD,0} \\ a_{DC,3} \\ a_{DC,2} \\ a_{DC,1} \\ a_{DC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 2 \\ -0.6256 \\ 7.9264 \\ -30.4172 \\ 41.8052 \\ -0.4864 \\ 9.8496 \\ -64.2048 \\ 144.56 \end{bmatrix} \tag{20}$$

The resulting coefficients for the trajectory for task 2 in the y axis are shown below:

$$\begin{bmatrix} b_{AB,3} \\ b_{AB,2} \\ b_{AB,1} \\ b_{AB,0} \\ b_{BD,3} \\ b_{BD,2} \\ b_{BD,1} \\ b_{BD,0} \\ b_{DC,3} \\ b_{DC,2} \\ b_{DC,1} \\ b_{DC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 2 \\ -0.0291 \\ 0.3213 \\ -0.8914 \\ 8.5690 \\ 0.1971 \\ -3.9917 \\ 26.0198 \\ -46.6166 \end{bmatrix} \tag{21}$$

With the above coefficients, the following polynomial equations are formed to generate the trajectories for task 2 (positions for x and y):

$$x_{AB} = -0.1944t^3 + 0.9167t^2 + 2 \tag{22}$$
$$x_{BD} = -0.6256t^3 + 7.9264t^2 - 30.4172t + 41.8052 \tag{23}$$
$$x_{DC} = -0.4864t^3 + 9.8496t^2 - 64.2048t + 144.56 \tag{24}$$
$$y_{AB} = -0.1944t^3 + 0.9167t^2 + 5 \tag{25}$$
$$y_{BD} = -0.0291t^3 + 0.3213t^2 - 0.8914t + 8.5690 \tag{26}$$
$$y_{DC} = 0.1971t^3 - 3.9917t^2 + 26.0198t - 46.6166 \tag{27}$$

The velocity equations are also computed as the following polynomials:

$$\dot{x}_{AB} = -0.5833t^2 + 1.833t \tag{28}$$
$$\dot{x}_{BD} = -1.8768t^2 + 15.8528t - 30.4172 \tag{29}$$
$$\dot{x}_{DC} = -1.4592t^2 + 19.6992t - 64.2048 \tag{30}$$
$$\dot{y}_{AB} = -0.5833t^2 + 1.833t \tag{31}$$
$$\dot{y}_{BD} = -0.0874t^2 + 0.6426t - 0.8914 \tag{32}$$
$$\dot{y}_{DC} = 0.5914t^2 - 7.9834 + 26.0198 \tag{33}$$

2.2.5 Graphical Results

The trajectory that is generated based on the above polynomial equations is visualised in figure 10 with point D being (10.20, 8.54).

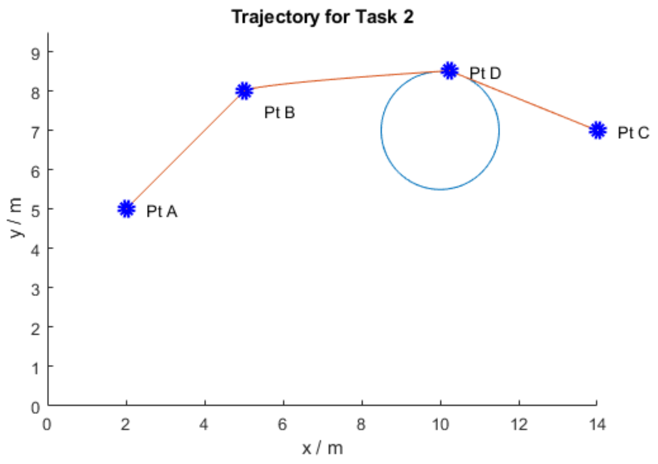

Figure 10: Trajectory of Task 2 with the Obstacle

The plot of the resulting motion from the trajectory generated are shown below in figures 11 to 14:

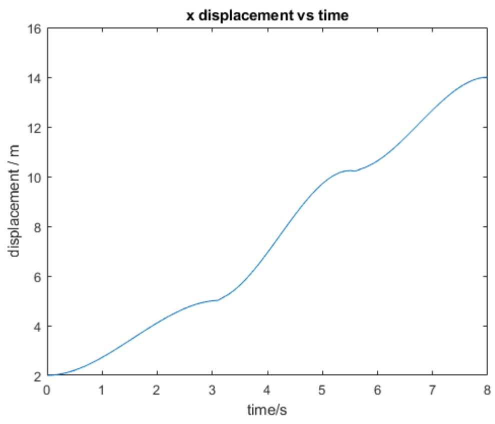

Figure 11: x Displacement Over Time of the Trajectory (From A to C)

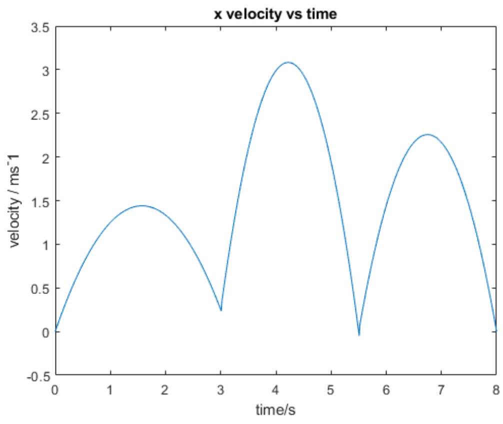

Figure 12: x Velocity Over Time of the Trajectory (From A to C)

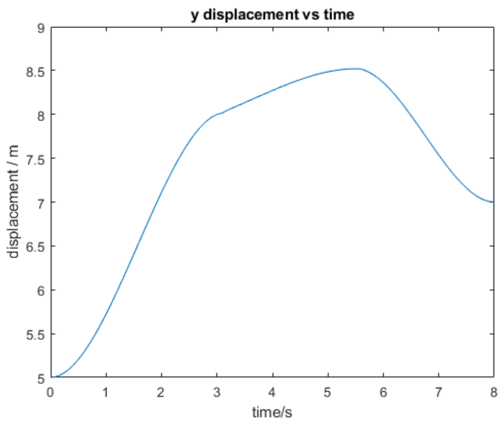

Figure 13: y Displacement Over Time of the Trajectory (From A to C)

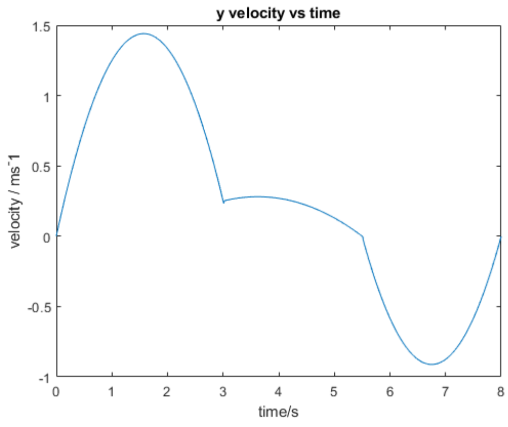

Figure 14: y Velocity Over Time of the Trajectory (From A to C)

2.3 Task 3: Trajectory Generation with Robotic Arm and Obstacle

2.3.1 Task Description

Similar to task 2, however, a planar robot (with 2 links) will be introduced. With reference to figure 15, the robot has the displayed parameters of link 1 and link 2 ($L_1 = L_2 = 9m$), where it can be seen to have both revolute joints (F and G) aligned with the Z axis. The trajectory that is designed for this task must allow the robot end-effector (T) traverse from point A to B to C. Furthermore, an additional condition (on top of 2A and 2B) is required to be followed where the straight line formed by segments FG and GT must not collide with the obstacle Obs1 (Condition 3).

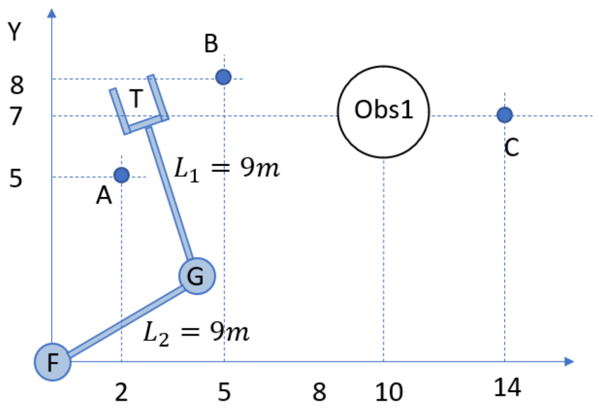

Figure 15: Robotic Arm with the Via-Points and Obstacle

2.3.2 *Inverse-Kinematic for the Planar Robot*

Inverse-kinematics was utilised to determine the angles of rotation (joint space) for the planar robot joints with respect to a certain coordinate in task space. As the robot only has two joints, the equations to find the robot's joint rotational angles (joints F and G rotational angles Q_1 and Q_2 respectively) is novel. This is necessary for section 2.3.3 where the new point D will be recalculated.
For elbow up position, the equation to convert task space coordinates (x, y) to joint space variables (Q_1, Q_2) is the following:

$$Q_{1,up} = \tan^{-1}\left(\frac{y}{x}\right) + \sin^{-1}\left(\frac{L_2 * -\sin Q_2}{\sqrt{x^2+y^2}}\right) \tag{34}$$

$$Q_{2,up} = -\cos^{-1}\left(\frac{x^2+y^2-(L_1^2+L_2^2)}{2 * L_1 * L_2}\right) \tag{35}$$

For elbow down position, the equations are the following:

$$Q_{1,down} = \tan^{-1}\left(\frac{y}{x}\right) - \sin^{-1}\left(\frac{L_2 * \sin Q_2}{\sqrt{x^2+y^2}}\right) \tag{36}$$

$$Q_{2,down} = \cos^{-1}\left(\frac{x^2+y^2-(L_1^2+L_2^2)}{2 * L_1 * L_2}\right) \tag{37}$$

With the known lengths of the robot arm ($L_1 = L_2 = 9m$) and the known rotation of each joint ($Q_{1,up}, Q_{1,down}, Q_{2,up}, Q_{2,down}$) it is possible to plot and recreate the arms on the plot along with the other via-points and obstacle. This can be seen in figure 16 where the x and y coordinates of point C was used as the input argument to find the joint rotations of F and G.

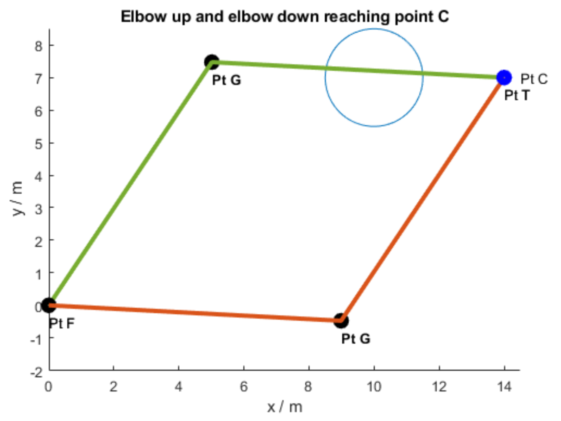

Figure 16: Elbow Up and Down Configuration of the Robotic Arm

From the above figure, it can be clearly seen that the elbow up configuration violates condition 3 where the linkage GT (green line) can be seen to collide with the obstacle (blue circle). Therefore, to evaluate for a valid solution to task 3, only the elbow down configuration should be considered.

2.3.3 *Approach to Find New Point D*

By observation, it can be concluded that the current point D (from task 2) will not satisfy Condition 3 (as the segment GT will collide with the obstacle). As such, it is necessary to compute another way to find the ideal point D such that it will retain conditions 2A and 2B along with the new condition 3. To implement this, the function "*Assignment3_searchD*" was edited such that it incorporated the option to consider condition 3. The function takes an input argument "*considerArm*" which dictates the inclusion of condition 3 for finding point D. The implemented logic is discussed in steps below:

I. The set up to find the new D point will be the same as task 2 where a grid search is implemented (Follows the same steps i) and ii) from task 2).

II. Along with the conditional statements in step iii) for task 2 (a, b and c), the inclusion of a new conditional statement is utilised:
   - A. If the input argument "*considerArm*" is true, do the following for the proposed point D coordinates:
     1. As before from step iii) in task 2, create the trajectory of BD and DC for the proposed point D.
     2. Create the planar robot arm (with elbow down configuration as discussed in section 2.3.2) where the x and y coordinates are the computed points along the trajectory for the path BD and DC.
     3. At each configuration of the robot arm (for each x and y coordinate of the trajectory), break down the arm links into several points that run evenly along the arms.
     4. For each point along the arm link, inspect if the points are in collision with the obstacle.
     5. If any point of the arm at any moment of the trajectory is colliding with the obstacle, the generated point D is discarded (Condition 3).
     6. Otherwise, if all the points of the arm at all moments of the trajectory is not colliding with the obstacle, the point D is not discarded.
   - B. The point D that is not discarded will then be saved as the new point D.

III. This process repeats until an ideal point D is located.

The final computed point D that satisfies conditions 2A, 2B and 3 is: $Point\ D = (9.86{,}5.32)$ as shown in figure 17. For more detailed pseudocode, refer to the appendix section.

2.3.4 Generate Trajectory Based on New Point D

With the same method as task 2 and same function for trajectory generation as task 1, the new path that is generated will go from A to B, B to D and D to C. The new parameters are shown below (where only point D is updated):

Trajectory	**Point A to B (Segment AB)**	**Point B to D (Segment BD)**	**Point D to C (Segment DC)**
Initial time ($t_{AB,i}/s$)	0	3	5.5
Final time ($t_{AB,f}/s$)	3	5.5	8
Initial Position $(x, y)\ /m$	(2, 5)	(5,8)	(9.86,5.32)
Final Position $(x, y)\ /m$	(5, 8)	(9.86,5.32)	(14, 7)
Initial Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0	0.25	0
Final Velocity $(\dot{x}, \dot{y})\ /ms^{-1}$	0.25	0	0

2.3.5 Polynomial Results

The resulting coefficients for the trajectory for task 3 in the x axis are shown below:

$$\begin{bmatrix} a_{AB,3} \\ a_{AB,2} \\ a_{AB,1} \\ a_{AB,0} \\ a_{BD,3} \\ a_{BD,2} \\ a_{BD,1} \\ a_{BD,0} \\ a_{DC,3} \\ a_{DC,2} \\ a_{DC,1} \\ a_{DC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 2 \\ -0.5821 \\ 7.3715 \\ -28.2630 \\ 39.1614 \\ -0.5299 \\ 10.7309 \\ -69.9494 \\ 158.1382 \end{bmatrix} \tag{38}$$

The resulting coefficients for the trajectory for task 3 in the y axis are shown below:

$$\begin{bmatrix} b_{AB,3} \\ b_{AB,2} \\ b_{AB,1} \\ b_{AB,0} \\ b_{BD,3} \\ b_{BD,2} \\ b_{BD,1} \\ b_{BD,0} \\ b_{DC,3} \\ b_{DC,2} \\ b_{DC,1} \\ b_{DC,0} \end{bmatrix} = \begin{bmatrix} -0.1944 \\ 0.9167 \\ 0 \\ 2 \\ 0.3830 \\ -4.9338 \\ 19.5105 \\ -16.4697 \\ -0.2150 \\ 4.3546 \\ -28.3853 \\ 65.4909 \end{bmatrix} \tag{39}$$

With the above coefficients, the following polynomial equations are formed to generate the trajectories for task 3 (positions for x and y):

$$x_{AB} = -0.1944t^3 + 0.9167t^2 + 2 \tag{40}$$
$$x_{BD} = -0.5821t^3 + 7.3715t^2 - 28.2630t + 39.1614 \tag{41}$$
$$x_{DC} = -0.5299t^3 + 10.7309t^2 - 69.9494t + 158.1382 \tag{42}$$
$$y_{AB} = -0.1944t^3 + 0.9167t^2 + 5 \tag{43}$$
$$y_{BD} = 0.3830t^3 - 4.9338t^2 + 19.5105t - 16.4697 \tag{44}$$
$$y_{DC} = -0.2150t^3 + 4.3546t^2 - 28.3853t + 65.4909 \tag{45}$$

The velocity equations are also computed as the following polynomials:

$$\dot{x}_{AB} = -0.5833t^2 + 1.833t \tag{46}$$
$$\dot{x}_{BD} = -1.7462t^2 + 14.7430t - 28.2630 \tag{47}$$
$$\dot{x}_{DC} = -1.5898t^2 + 21.4618t - 69.9494 \tag{48}$$
$$\dot{y}_{AB} = -0.5833t^2 + 1.833t \tag{49}$$
$$\dot{y}_{BD} = -1.1491t^2 - 9.8675t + 19.5105 \tag{50}$$
$$\dot{y}_{DC} = -0.6451t^2 + 8.7091 - 28.3853 \tag{51}$$

2.3.6 *Graphical Results*

The trajectory that is generated based on the above polynomial equations is visualised in figure 17 with point D being (9.86,5.32).

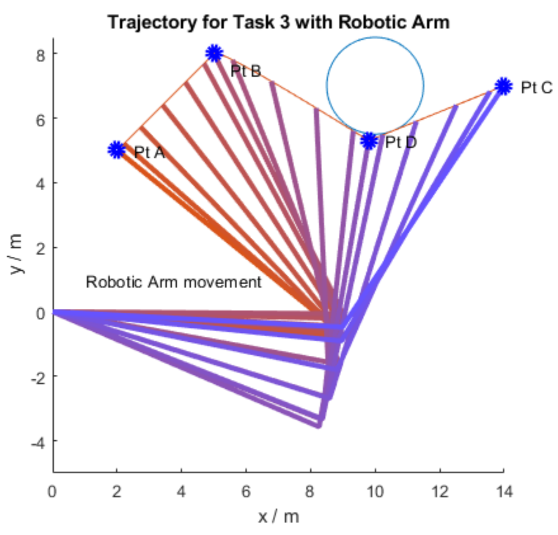

Figure 17: Trajectory of Task 3 with New Point D and Robot Arm Configurations

The plot of the resulting motion from the trajectory generated above (for task 3) are shown below in figures 18 to 21:

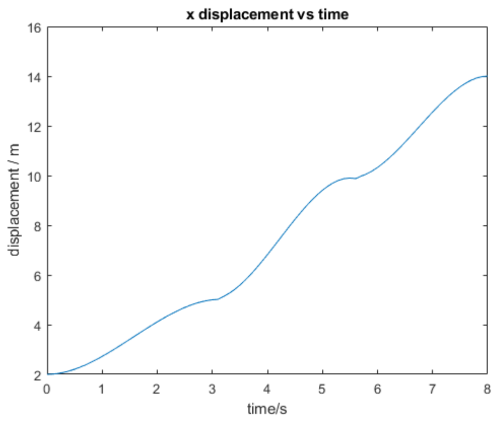

Figure 18: x Displacement Over Time of Trajectory

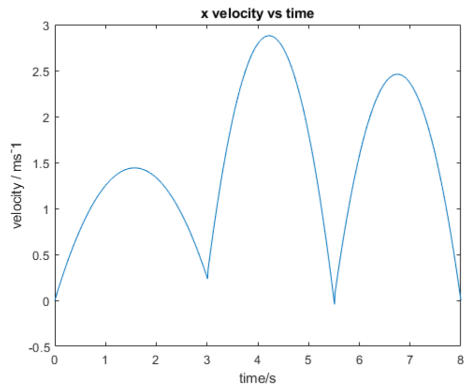

Figure 19: x Velocity Over Time of Trajectory

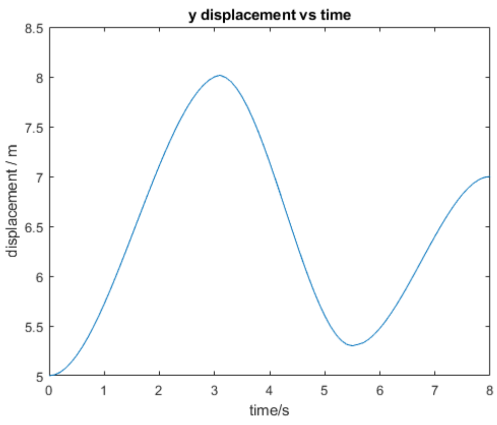

Figure 20: y Displacement Over Time of Trajectory

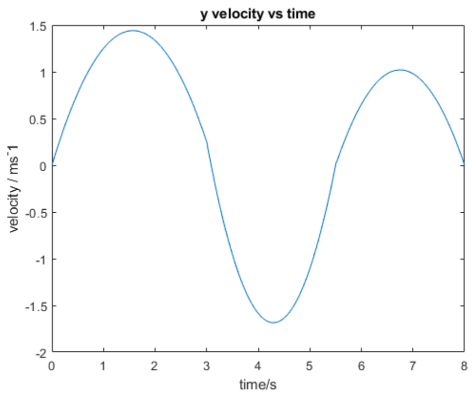

Figure 21: y Velocity Over Time of Trajectory

APPENDIX

Green text is the comment of the pseudocode for easy reading.
Red text is specifically for when the function is called to refine point D (for higher resolution in task 2), so the total distance is as minimum as possible

Function Name: Assignemnt3_searchD (input argument = considerArm)

```
% This function will search for the ideal point D where it takes in account of conditions 2A
and 2B. If the input argument considerArm is true, this function will consider condition 3
additionally.
% Define the circle (obstacle) radius and position
radius = 1.5
position (x,y) = (10,7)
% Define starting point of trajectory and ending point of trajectory
Starting point = (5,8)
Ending point = (14,7)
% Define Grid Search variables
search boundary= $1.5 * r$ % This changes to 0.1 for refinement of D
resolution = 0.1 % This changes to 0.01 for refinement of D
shortest distant = infinity
current point = (infinity, infinity)

% Search between the positive and negative boundary variables with a set resolution for both
x and y coordinates
for dx ∈ (-search boundary : resolution : search boundary)
  for dy ∈ (- search boundary : resolution :search boundary)
        % The current Point D that will be analysed
    ptD = (x-coordinates of obstacle + dx, y-coordinates of obstacle + dy)
    % ptD = (x-coordinates of previous point D + dx, y-coordinates of previous point
    D + dy) for refinement of point D
    % Condition 2B checks for the shortest path (Euclidean Distance)
    if straight line paths BD + DC is shorter than shortest saved
      compute the trajectory BD and DC
      find all points on trajectory BD and DC
      % Condition 2A ensures the straight-line segments BD and DC do not collide with
      obstacle (Obs1)
      if all points on trajectory BD and DC is outside of the circle (Obs1)
        % Save the point D if it satisfies 2A and 2B
        shortest distant = straight line path BD +DC distance
        current point = ptD

% Additional conditions for robotic arm collision detection (Task 3)
    if considerArm = true
        % Define variables for links for robotic arms
        L1, L2 = length of the robotic link 1 and link 2 (9m)
        % Generate angle arrays (joint space) for the trajectory with current point D
        for all points on the trajectory
          Q1,Q2 = angles of the robotic arms from inverse kinematics
          arm_pos = section the robotic arms into a list of coordinate points based
          on L1, L2, Q1 and Q2
          % Condition 3 checks if the robotic arm is colliding with obstacle
          for all arm_pos
              if arm_pos is inside the obstacle
                  do not save the current point to ptD

% If point D passes all conditions 2A, 2B and 3, the point is saved
coordinates of point D = current point
```