MELBOURNE
SCHOOL OF
ENGINEERING

MCEN90028
Robotic Systems

Assignment 3:
Trajectory Generation and Obstacle Avoidance Using Via Points

Assignment Group 1

Jiawei Liao	756560
Harfi Dharma Santoso	772503
Omar Sakkal	757372

Department of Mechanical Engineering

April 30, 2020

1 Description of Assignment

The purpose of this assignment is to derive and implement an algorithm for trajectory planning for a 2-D 2-Link robot arm end-effector. The trajectory is constructed to pass through a set of via points from a given initial pose and initial velocity to a final pose and final velocity. The assignment is comprised of 3 main tasks:

1. **Task 1**:
   Construction of a trajectory generation algorithm given the coordinates of the via points, the constraints at each via point and the time duration for each segment. For this assignment, the constraints include only displacement and velocity and therefore it is sufficient to construct the segments using a cubic polynomial.
2. **Task 2**:
   A circular obstacle with a known radius and spatial location is added between point B and point C. A new via point D between B and C is found using a newly constructed algorithm to avoid colliding with the obstacle while ensuring a heuristic minimum total distance is travelled.
3. **Task 3**:
   Repeat Task 2 but with an additional consideration of the 2-Link robot kinematics, where the two links of the robot should remain clear of the obstacle throughout the entire trajectory.

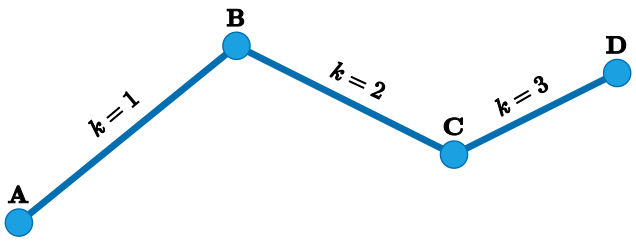

Figure 1: Example of a trajectory with multiple segments

The trajectory to be generated can have k_{max} segments, labelled as segments $k = 1, ...k_{max}$. Each segment is constructed of polynomial of order P, where P is an odd number. The order of the polynomial, once selected, will be the same for all segments. In Figure 1, a trajectory will start at point A and will end at point D, with points B and C defined as "Via Points". A trajectory with k_{max} segments will have $k_{max} - 1$ via points.
A polynomial construction of each segment will be of the form:

$$x(t) = a_0 + a_1 t + a_2 t^2 + ... + a_P t^P \tag{1}$$

For a cubic polynomial, $P = 3$, while for quintic polynomial, $P = 5$. The coefficients $a_0, a_1, ..., a_P$ are the coefficients of the polynomial and are the output of a trajectory generation function.

2 Assessment Tasks

2.1 Task1

In this task, a trajectory generation function is coded in MATLAB:

$$SegmentCoeff = TrajGen01(init, final, tfinal)$$

where

- $SegmentCoeff$ is the output, which is a $(P+1) \times k_{max}$ matrix containing the coefficients of the polynomials for all segments of the trajectory.
- $init$ is the $((P+1)/2) \times k_{max}$ matrix containing the initial conditions for all the segments. For cubic polynomials ($P = 3$), $init$ will contain the initial position, initial velocity. Hence $init$ will be a 2 by k_{max} matrix.
- Similarly, $final$ is the $((P+1)/2) \times k_{max}$ matrix containing the final conditions for all the segments.
- $tfinal$ is an array of size $k_{max} \times 1$, containing the time duration of each segment.

2.1.1 MATLAB code algorithm

The $TrajgGen01$ function is configured to accept any value of P.
Consider X as an odd polynomial (order P) function of t starting at $x = x_0$ when $t = 0$ and finishes at $x = x_f$ when $t = t_f$:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + ... + a_P t^P \tag{2}$$

The (0)th to the $((P+1)/2)$th derivatives at $t = 0$ are then:

$$\begin{aligned} &x(t_0) = x(0) = a_0 = x_0, \\ &\dot{x}(t_0) = \dot{x}(0) = a_1 = \dot{x}_0, \\ &....., \\ &x^{((P-1)/2)}(t_0) = \frac{d^{((P-1)/2)}x}{dx^{((P-1)/2)}}(t_0) = ((p-1)/2)! a_{((P-1)/2))} \end{aligned} \tag{3}$$

The (0)th to the $((P+1)/2)$th derivatives at $t = t_f$ are then:

$$\begin{aligned} &x(t_f) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + ... + a_P t^P, \\ &\dot{x}(t_f) = \dot{x}(t_f) = a_1 + 2a_2 t + 3a_3 t^2 + ... + P a_P t^{P-1}, \\ &....., \\ &x^{((P-1)/2)}(t_f) = \frac{d^{((P-1)/2)}x}{dx^{((P-1)/2)}}(t_f) = ((p-1)/2)! a_{((P-1)/2))} + ... + \frac{P!}{((P+1)/2)!} a_{(P+1)/2)}^{((P+1)/2)+1)} \end{aligned} \tag{4}$$

To solve for $a_0, a_1 ... a_P$, we can rearrange equations 2,3,4 to matrix form:

$$\underbrace{\begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & ((p-1)/2)! a_{((P-1)/2))} & ... \\ 1 & t_f & ... & t_f^P \\ 0 & 1 & ... & P t_f^{P-1} \\ ... & ... & ... & ... \\ 0 & 0 & ... & \frac{P!}{((P+1)/2)!} a_{(P+1)/2)}^{((P+1)/2)+1} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ ... \\ ... \\ ... \\ ... \\ ... \\ a_P \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} x_0 \\ \dot{x}_1 \\ ... \\ x^{((P-1)/2)}(t_0) \\ x_f \\ \dot{x}_f \\ ... \\ x^{((P-1)/2)}(t_f) \end{bmatrix}}_{X} \tag{5}$$

A is a square matrix of dimension $(P+1)$, B and X are both column matrix with $(P+1)$ rows. Solving for B would give the coefficients of the polynomial. For the case $t \neq 0$, then the generated polynomial function is given for $x(t - t_0)$.

2.1.2 Task 1 Parameters

The parameters for Task 1 are:

$$X_i = \begin{bmatrix} 2 & 5 \\ 0 & 0.25 \end{bmatrix}, X_f = \begin{bmatrix} 5 & 14 \\ 0.25 & 0 \end{bmatrix}, Y_i = \begin{bmatrix} 5 & 8 \\ 0 & 0.25 \end{bmatrix}, Y_f = \begin{bmatrix} 8 & 7 \\ 0.25 & 0 \end{bmatrix}, tfinal = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

The number of constraints in the input indicates that $P = 3$ and $k_{max} = 2$. So two functions of cubic polynomials will be used to construct the X and Y trajectory respectively.
For $P = 3$, the matrix equation 5 breaks down to:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \text{ft} & \text{ft}^2 & \text{ft}^3 \\ 0 & 1 & 2\,\text{ft} & 3\,\text{ft}^2 \end{pmatrix}_A \begin{pmatrix} a_0 \\ a_1 \\ a_3 \\ a_4 \end{pmatrix}_B = \begin{pmatrix} x_0 \\ \dot{x}_0 \\ x_f \\ \dot{x}_f \end{pmatrix}_X \tag{6}$$

2.1.3 Task 1 Results

The MATLAB script mcen90029_ass3_task1 computes the polynomial coefficients of $x(t)$ and $y(t)$ trajectory and generates three plots: displacement, velocity and trajectory.

$$xtCoeff = \begin{pmatrix} 2.0000 & 5.0000 \\ 0 & 0.2500 \\ 0.9167 & 0.9800 \\ -0.1944 & -0.1340 \end{pmatrix}, \quad ytCoeff = \begin{pmatrix} 5.0000 & 8.0000 \\ 0 & 0.2500 \\ 0.9167 & -0.2200 \\ -0.1944 & -0.0260 \end{pmatrix} \tag{7}$$

$$\begin{aligned} x(t) &= -0.1944t^3 + 0.9167t^2 + 2, \quad 0 \le t < 3, \\ x(t) &= -0.1340(t-3)^3 + 0.9800(t-3)^2 + 0.25t + 5, \quad 3 \le t \le 8 \\ y(t) &= -0.1944t^3 + 0.9167t^2 + 5, \quad 0 \le t < 3, \\ y(t) &= -0.026(t-3)^3 - 0.22(t-3)^2 + 0.25(t-3) + 8, \quad 3 \le t \le 8 \end{aligned} \tag{8}$$

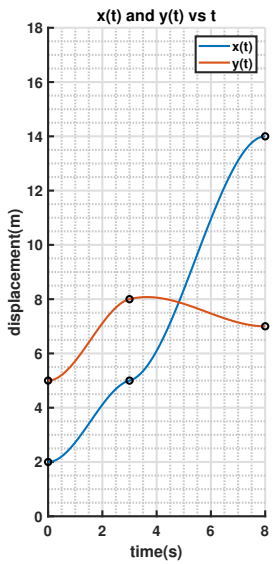

Figure 2: $x(t), y(t)$ vs t

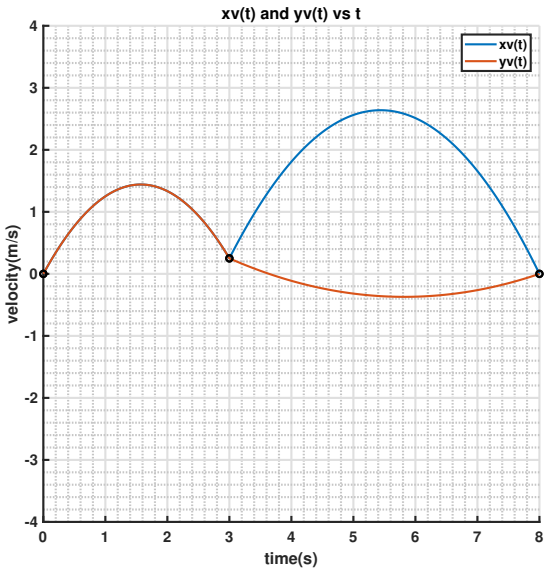

Figure 3: $\dot{x}(t), \dot{y}(t)$ vs t

3 Task 2

In Task 2, a circular obstacle of radius $1.5m$ is added at the spatial position $(10, 7)$ as depicted in Figure 4. This obstacle implies segment BC of the original trajectory designed in Task 1 is no longer valid and needs to be redesigned to ensure we are satisfying the constraints imposed (as stated below).
A new via point D is to be decided such that:

- Condition 2A: Ensure that straight-line segments BD and DC do not collide with Obs1.
- Condition 2B: We would like to try for the total distance travelled to be minimum.

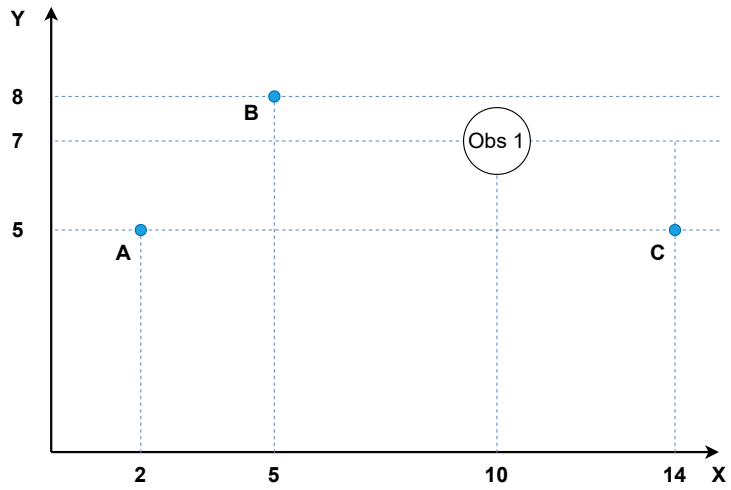

Figure 4: Circular obstacle Obs1 of radius $1.5m$ is added at $(10, 7)$

3.1 Algorithm for finding point D

3.1.1 The tangent intersection Approach

A commonly used algorithm to determine the shortest trajectory for mobile robots and circular modelled obstacles is to search for the shortest path over the collision-free tangent graph as explained by (Alexopoulos and Griffin 1992).
An example of such a collision free path is presented in Figure 5.

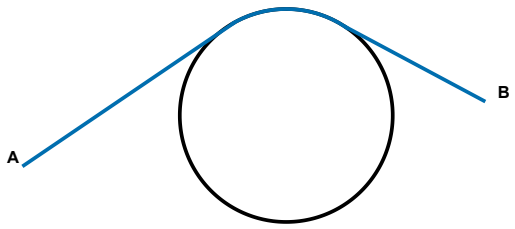

Figure 5: Example of shortest path between two points around a circular obstacle

To calculate the equation of the tangent, we first need to find the distance between the external point to the centre of the circle. Then we form a circle equation at the external point of radius equal to the distance. The two intersection points between the two circles, once connected with

the external point, forms the two tangent lines.
i.e, circle at B has equation $(x-5)^2+(y-8)^2 = 26$ and Obs1 has equation $(x-10)^2+(7-7)^2 = 1.5^2$

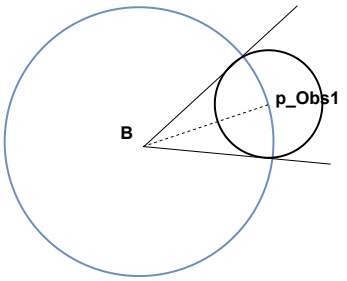

Figure 6: Tangent lines methods

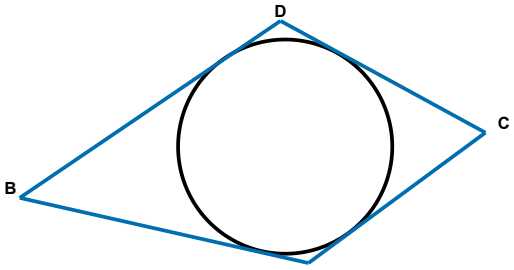

Figure 7: Example of optimal point D

Since we are considering straight line segments trajectory between points B, D and C, point D then needs to be found at the intersection of tangents lines where D is closest to the obstacle as illustrated in Figure 7.

Point B can form two tangent lines to the Obs1, and so does point D. In total there are up to 4 intersection points formed if all point B tangents and all point C tangents are not parallel. But only the two closest to the circle are relevant and the other two are much further away.
For task 2, we could just find point D such that the sum of Euclidean distance $L\_BD + L\_DC$ is a minimum, and this should theoretically achieve both Condition 2A and Condition 2B if

- Trajectory T_BD and T_DC are perfectly straight.
- No rounding errors occurred during computation.

Practically, the trajectory is only perfectly straight for the cubic polynomial trajectory if:

- The velocity constraint at two end points for both $x(t)$ and $y(t)$ are 0.

3.1.2 Shifting point D to fully ensure Condition 2A is satisfied

To fully ensure the validity of Condition 2A, numerical evaluation of the generated trajectory T_BD and T_DC is used to check the collision.
A point of (x, y) is within or on a circle of radius R at (xo, yo) if $(x - xo)^2 + (y - yo^2) <= R^2$.
If collision is detected along the trajectory, then a shift of point D is triggered by a set distance of $r\_OBS/50$ along the line from the centre of Obs1 to point D.

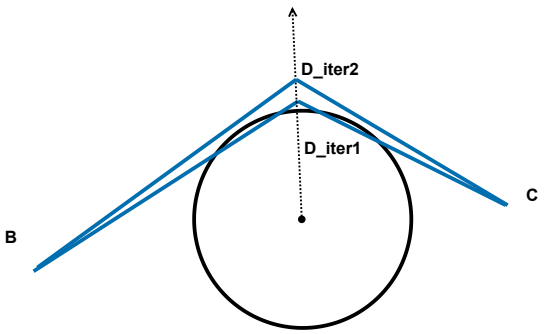

Figure 8: Example of 2 iteration of shifting point D

3.1.3 Constraint Settings Methods

The time segment is set according to the ratio of length between segment BD and segment DC, such that $tBD : tDC = L\_BD : L\_DC$.
i,e, $BD : DC = 3 : 2$, then $tBD = 3/5 \times 5 = 3s, tDC = 2/5 \times 5 = 2s$
We proposed two methods for automatically setting the velocity constraints where needed:

- "Straight" Mode: Set all adjustable velocity constraints to 0. (Point B is excluded in this case).
- "Smooth" Mode: Set all adjustable velocity constraints to the mean of the average trajectory velocity between two connecting segments.
  i.e, $x_i = 4, x_f = 10, t = 2$, then average $xv(t) = (10 - 4)/2 = 3m/s$
  This makes the trajectory more curved but ensures smoother velocity changes over the course. (Point B is excluded in this case).

3.2 General pseudo-code flowchart

Note that the code also needs to check at the start that:

- Point B and point C are clear of OBs1, otherwise point B, point C or both are inside the circle of OBs1 and this renders the problem unsolvable.
- The straight line segment BC is actually obstructed by OBs1, otheriwse there is no need to construct a via point.

The following figure is a simplified pseudo-code flowchart of our Task 2 algorithm and that Obs1 is actually in the way.

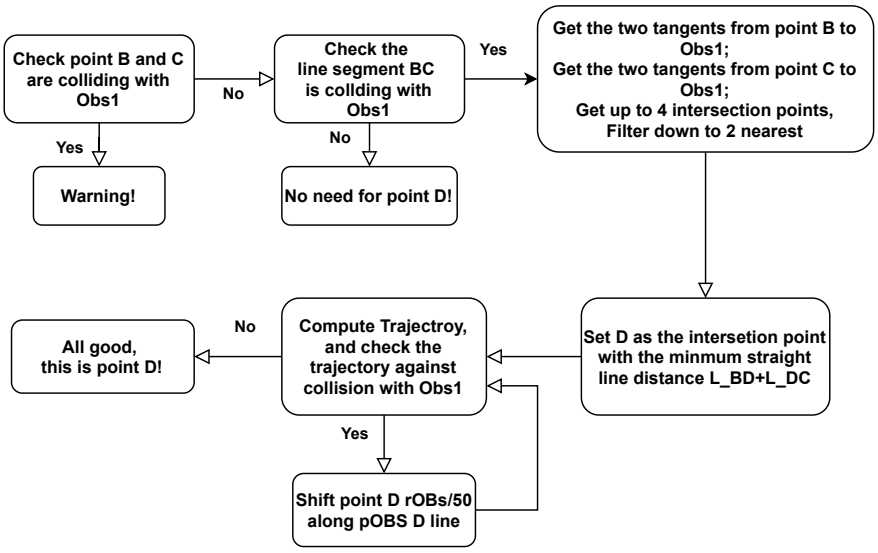

Figure 9: Task 2 algorithm flowchart

3.3 Task 2 Results

3.3.1 Mode: "Straight"

The results below are produced by running 'mcen90028_ass3_task2_m.m'.

$$X_i = \begin{bmatrix} 2 & 5 & 9.8898 \\ 0 & 0 & 0 \end{bmatrix}, X_f = \begin{bmatrix} 5 & 9.8898 & 14 \\ 0 & 0 & 0 \end{bmatrix},$$

$$Y_i = \begin{bmatrix} 5 & 8 & 5.3118 \\ 0 & 0 & 0 \end{bmatrix}, Y_f = \begin{bmatrix} 8 & 5.3118 & 7 \\ 0 & 0 & 0 \end{bmatrix}, tfinal = \begin{bmatrix} 3 \\ 2.71 \\ 2.29 \end{bmatrix}, pD = (9.8898, 5.3118)$$

$$xtCoeff = \begin{pmatrix} 2.0000 & 5.0000 & 9.8898 \\ 0 & 0 & 0 \\ 1.0000 & 1.9974 & 2.3513 \\ -0.2222 & -0.4914 & -0.6845 \end{pmatrix}, \quad ytCoeff = \begin{pmatrix} 5.0000 & 8.0000 & 5.3118 \\ 0 & 0 & 0 \\ 1.0000 & -1.0981 & 0.9658 \\ -0.2222 & 0.2701 & -0.2812 \end{pmatrix} \tag{9}$$

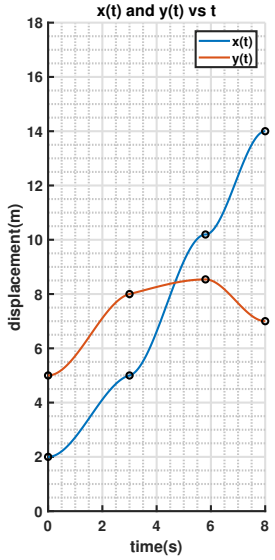

Figure 10: $x(t), y(t)$ vs t

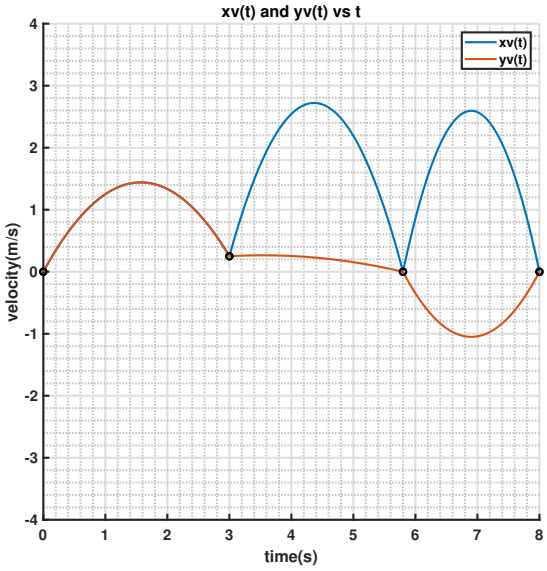

Figure 11: $\dot{x}(t), \dot{y}(t)$ vs t

3.3.2 Mode: "Smooth"

$$X_i = \begin{bmatrix} 2 & 5 & 10.1902 \\ 0 & 0.25 & 1.6071 \end{bmatrix}, X_f = \begin{bmatrix} 5 & 10.1902 & 14 \\ 0.25 & 1.6071 & 0 \end{bmatrix}$$

$$,Y_i = \begin{bmatrix} 5 & 8 & 8.5094 \\ 0 & 0.25 & -0.1786 \end{bmatrix}, Y_f = \begin{bmatrix} 8 & 8.5094 & 7.0000 \\ 0.25 & -0.1786 & 0 \end{bmatrix}, tfinal = \begin{bmatrix} 3 \\ 2.80 \\ 2.20 \end{bmatrix}, pD = (10.1902, 8.5094).$$

$$xtCoeff = \begin{pmatrix} 2.0000 & 5.0000 & 10.1902 \\ 0 & 0.2500 & 1.6071 \\ 0.9167 & 1.2335 & 0.9004 \\ -0.1944 & -0.2360 & -0.3835 \end{pmatrix}, \quad ytCoeff = \begin{pmatrix} 5.0000 & 8.0000 & 8.5094 \\ 0 & 0.2500 & -0.1786 \\ 0.9167 & 0.0801 & -0.7732 \\ -0.1944 & -0.0373 & 0.2466 \end{pmatrix} \tag{10}$$

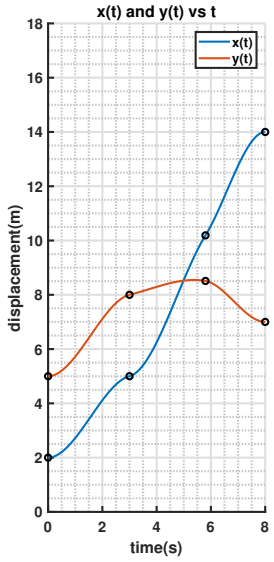

Figure 12: $x(t), y(t)$ vs t

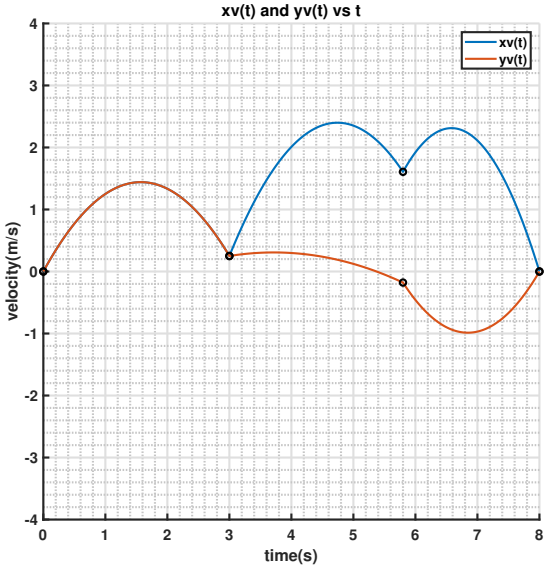

Figure 13: $\dot{x}(t), \dot{y}(t)$ vs t

4 Task 3

In Task 3, we repeat Task 2 but with the additional consideration of the structure of the robot. A planar 2 link robot (both revolute joints, with rotation axes at F and G aligned with the Z axis), as shown in Figure 14, is added to the problem. The trajectory being designed (from point A to B to C) is to be traversed by the end-effector of the robot (point T). The robot has a link length of L1 = 9m from the first to the second joint, and L2 = 9m from the second joint to the end-effector. Add the via point D, respecting Conditions 2A and 2B, with the additional condition below:

- Condition 3: The straight line formed by FG and GT must not collide the obstacle Obs1.

Condition 3 implies we can simplify the "robot" into a skeletal structure formed by two straight lines for the purpose of the assignment.

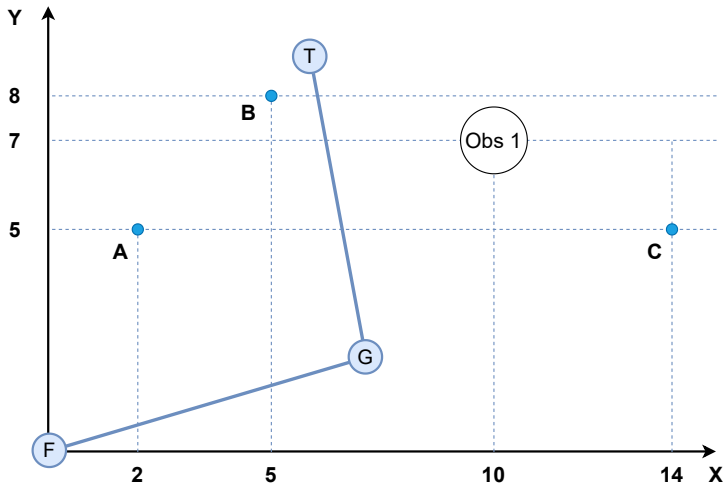

Figure 14: Structure of robot in the problem domain

4.1 Algorithm for finding point D based upon Task 2 algorithm

We will use the same principle of finding the two potential point D candidates as described in Task 2. Albeit this time we will also need to consider the kinematics of the 2-Link Robot.

4.1.1 Assumptions

The following assumptions we will make are:

- Assumption 1: The robot has to be in the same manifold throughout the trajectory.
- Assumption 2: Only one of the initial two candidates of point D produces the non-obstructed inverse kinematics solution manifold consistent with point D and C.

4.1.2 Constraint Settings Methods

The time segment is set according to the ratio of length between segment BD and segment DC, such that $tBD : tDC = L\_BD : L\_DC$. This is the same as was in Task 2
The two methods for setting the velocity constraints are the same as was in Task 2 but constraint at point B is removed.

- "Straight" Mode: Set all adjustable velocity constraints to 0.
- "Smooth" Mode: Set all adjustable velocity constraints to the mean of the average trajectory velocity between two connecting segments.

4.1.3 Inverse Kinematics and the Operation Manifold:

The inverse kinematics for a 2-D 2-Link Robot could be derived explicitly (as explained in Lectures 7-8). The diagram and two solution manifolds will be illustrated below:

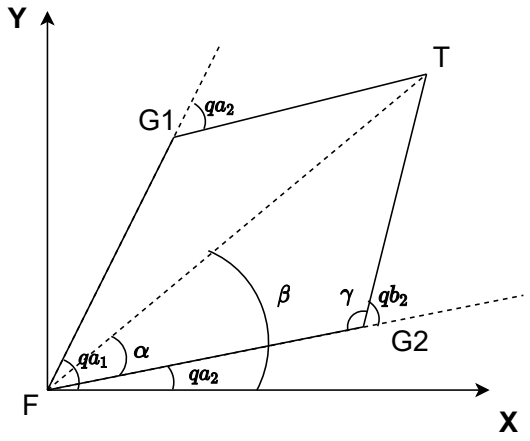

Figure 15: Inverse Kinematics solution of 2-D 2-link robot

$$q_2 = \pm \arccos\left(\frac{x^2 + y^2 - (L_1^2 + L_2^2)}{2L_1L_2}\right) = \pm D, D \geq 0 \tag{11}$$

$$\gamma = abs(\pi - q_2) \tag{12}$$

$$\alpha = \arcsin\left(\frac{\arcsin\left(L_2 \sin\left(\gamma\right)\right)}{\sqrt{(x^2 + y^2)}}\right) \tag{13}$$

$$\beta = \arctan\left(\frac{y}{x}\right) \tag{14}$$

$$q_1 = \beta - \alpha, \text{if } q_2 = D \tag{15}$$

$$q_1 = \beta + \alpha, \text{if } q_2 = -D \tag{16}$$

To determine the operating manifold of the robot, inverse kinematics solutions are first computed at point B and point C and checked to see if they collide with Obs1. There are the following cases to consider:

- 'none': No valid IK solution at both B and C.
- 'both': Both solution manifolds are possible at both B and C.
- 'up': Only elbow-up solution are possible at both B and C.
- 'down': Only elbow-down manifolds are possible at both B and C.

The two initially computed points D are also checked against the same criteria and the point D is chosen such that its non-obstructed inverse kinematic solution is consistent with point B and C. If both candidates are compatible, then the iterative process should be computed for each candidate and compare the final result.

4.1.4 Shifting point D to fully ensure Condition 2A and Condition 3

To fully ensure the validity of Condition 2A, numerical evaluation of the generated trajectory T_BD and T_DC is used to check the collision. A flag is raised if collision is detected.
To fully ensure the validity of Condition 3, numerical evaluation of the segment FG and GT is calculated at during the trajectory is used. The robot beams are made of steps of $R\_Obs1/50$ If collision of the robot beams FG or GT is detected at any instant of the trajectory, a flag is raised. A point of (x, y) is within or on a circle of radius R at (xo, yo) if $(x - xo)^2 + (y - yo^2) <= R^2$.
If a flag is raised for the current point D, then a shift of point D is triggered by a set distance of $r\_OBS/50$ along the line from the centre of Obs1 to point D.

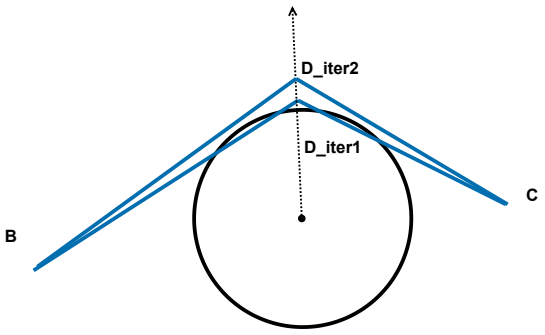

Figure 16: Example of 2 iteration of shifting point D

4.2 General pseudo-code flowchart

The following figure is a simplified pseudo-code flowchart of our Task 3 algorithm.
The same procedure of initial condition checking in Task 2 is also reserved for this implementation and that Obs1 is actually in the way.

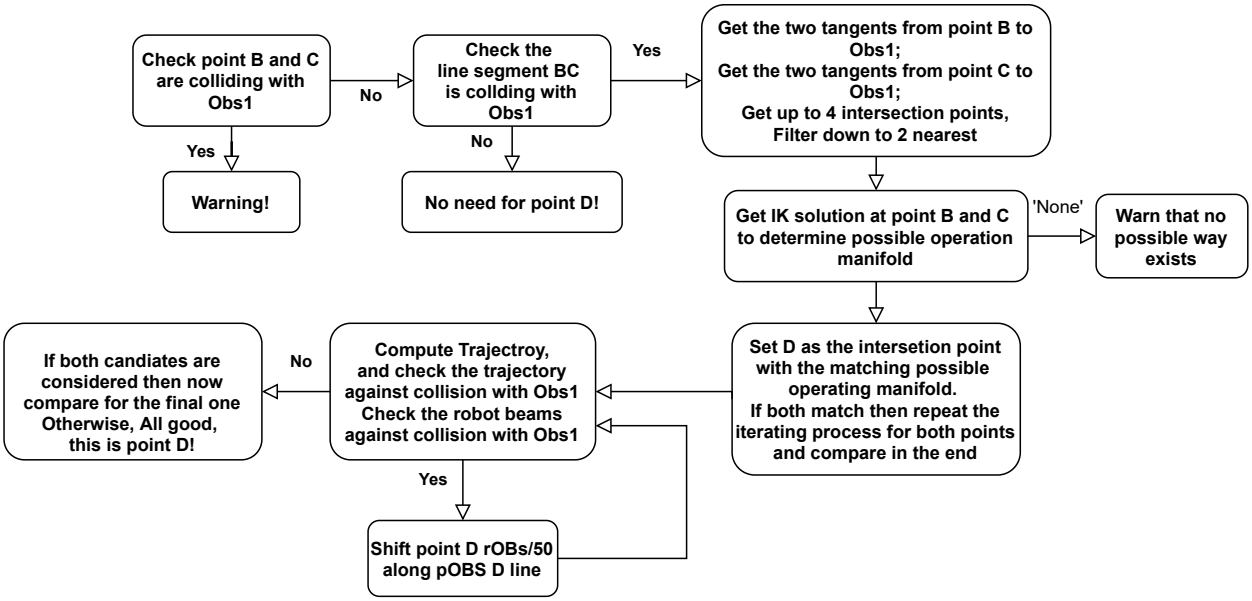

Figure 17: Task 3 algorithm flowchart

4.3 Task 3 Results

The results below are produced by running 'mcen90028_ass3_task3_m.m'.
For the given parameters, only the "elbow-up" configuration is possible at point C so "elbow-up" will be the chosen configuration of the robot. Of the two point D candidates, only the one under the circle allows "elbow-up" operation to be feasible.

4.3.1 Mode: "straight", iteration = 3

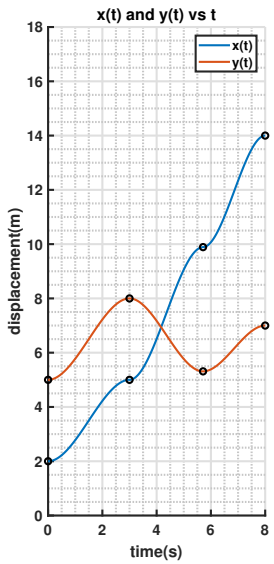

Figure 18: $x(t), y(t)$ vs t

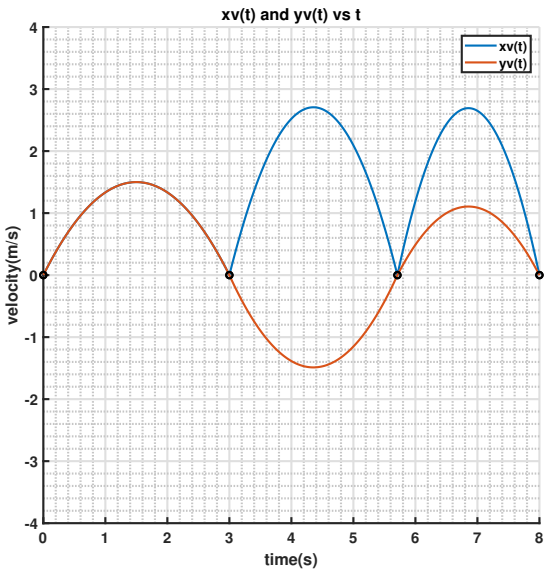

Figure 19: $\dot{x}(t), \dot{y}(t)$ vs t

$$X_i = \begin{bmatrix} 2 & 5 & 9.8898 \\ 0 & 0.25 & 0 \end{bmatrix}, X_f = \begin{bmatrix} 5 & 9.8898 & 14 \\ 0.25 & 0 & 0 \end{bmatrix}$$

$$,Y_i = \begin{bmatrix} 5 & 8 & 5.3118 \\ 0 & 0.25 & 0 \end{bmatrix}, Y_f = \begin{bmatrix} 8 & 5.3118 & 7 \\ 0.25 & 0 & 0 \end{bmatrix}, tfinal = \begin{bmatrix} 3 \\ 2.71 \\ 2.29 \end{bmatrix}, pD = (9.8898, 5.3118)$$

$$xtCoeff = \begin{pmatrix} 2.0000 & 5.0000 & 9.8898 \\ 0 & 00 & \\ 1.0000 & 1.9974 & 2.3513 \\ -0.2222 & -0.4914 & -0.6845 \end{pmatrix}, \quad ytCoeff = \begin{pmatrix} 5.0000 & 8.0000 & 5.3118 \\ 0 & 0 & 0 \\ 1.0000 & -1.0981 & 0.9658 \\ -0.2222 & 0.2701 & -0.2812 \end{pmatrix} \tag{17}$$

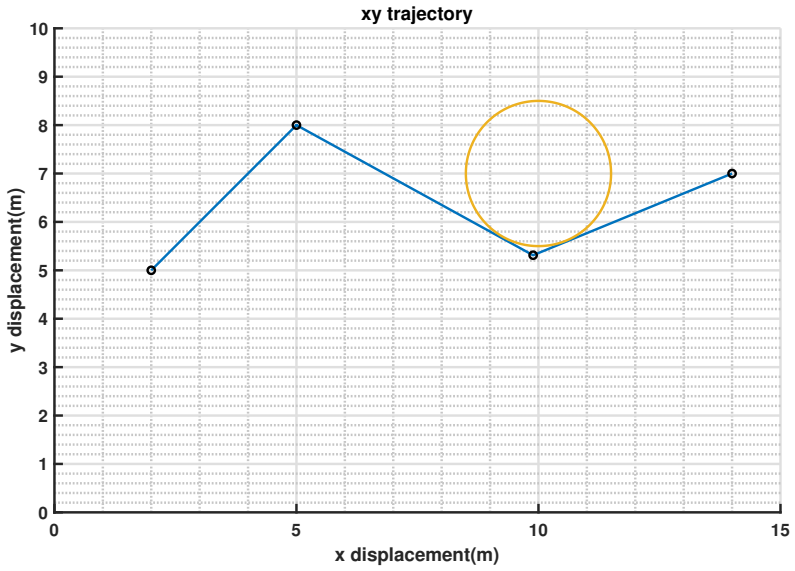

Figure 20: $x(t)$ vs $y(t)$

The "task3_straight.gif" generated by the MATLAB script provides a simple animation of the robot during the course of the trajectory.

4.3.2 Mode: "smooth", iteration = 1

$$X_i = \begin{bmatrix} 2 & 5 & 9.8938 \\ 0 & 1.4564 & 1.6605 \end{bmatrix}, X_f = \begin{bmatrix} 5 & 9.8938 & 14 \\ 1.4564 & 1.6605 & 0 \end{bmatrix}$$

$$,Y_i = \begin{bmatrix} 5 & 8 & 5.3731 \\ 0 & 0.0688 & -0.1845 \end{bmatrix}, Y_f = \begin{bmatrix} 8 & 5.3731 & 7 \\ 0.0688 & -0.1845 & 0 \end{bmatrix}, tfinal = \begin{bmatrix} 3 \\ 2.49 \\ 2.51 \end{bmatrix}, pD = (9.8938, 5.3731)$$

$$xtCoeff = \begin{pmatrix} 2.0000 & 5.0000 & 9.8938 \\ 0 & 1.4564 & 1.6605 \\ 0.5145 & 0.3115 & 0.8988 \\ -0.0604 & -0.0674 & -0.3672 \end{pmatrix}, \quad ytCoeff = \begin{pmatrix} 5.0000 & 8.0000 & 5.3731 \\ 0 & 0.0688 & -0.1845 \\ 0.9771 & -1.0558 & 1.0918 \\ -0.2146 & 0.2482 & -0.3061 \end{pmatrix} \tag{18}$$

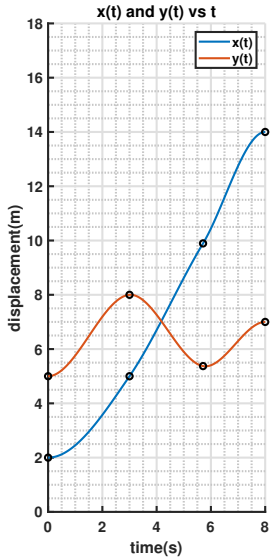

Figure 21: $x(t), y(t)$ vs t

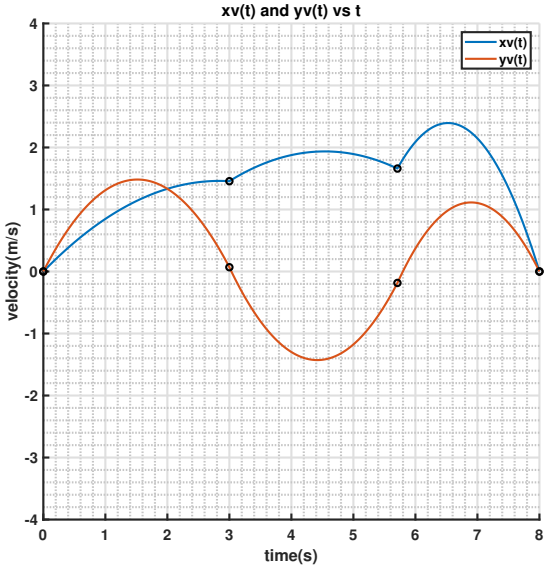

Figure 22: $\dot{x}(t), \dot{y}(t)$ vs t

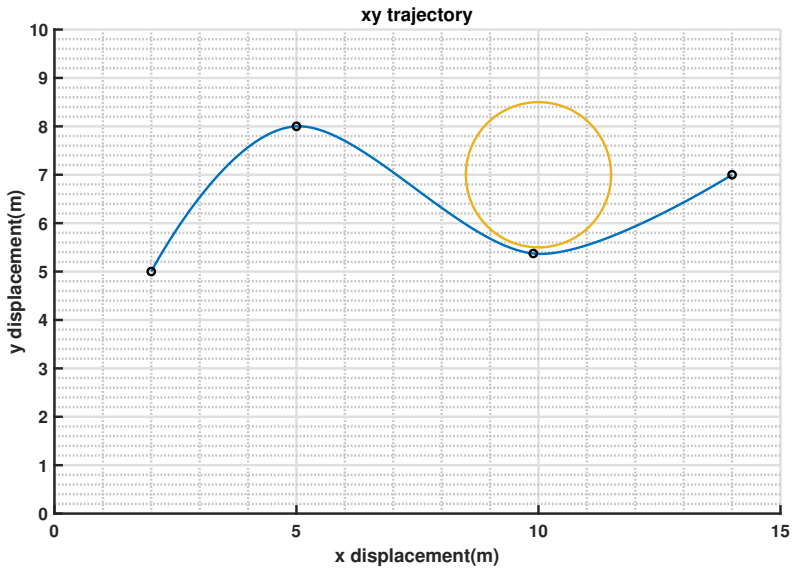

Figure 23: $x(t)$ vs $y(t)$

The "task3_straight.gif" generated by the MATLAB script provides a simple animation of the robot during the course of the trajectory.

4.4 Further Test Cases

We would also like to present you two test cases trajectory using the 'smooth' mode. Their animation files could be found under the sub-folder animation inside the MATLAB folder.

4.4.1 Case 2: $pA = (-2, -5), pB = (-5, -8), pC = (-14, -7), pObs1 = (-10, -7), rObs1 = 1.5$

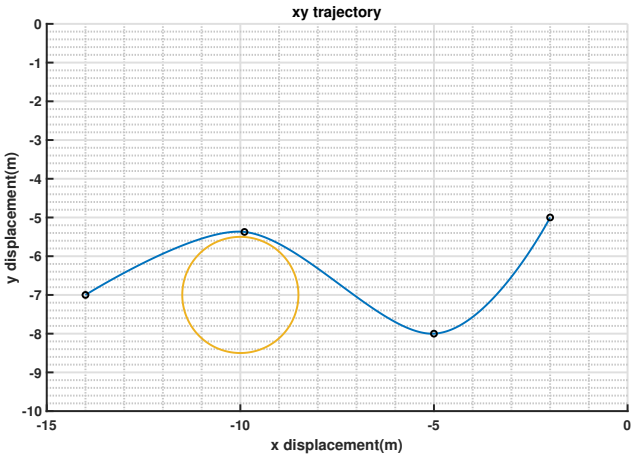

Figure 24: $x(t)$ vs $y(t)$

4.4.2 Case 2: $pA = (6, -2), pB = (6, 0), pC = (6, 8), pObs1 = (6, 3), rObs1 = 1.5$

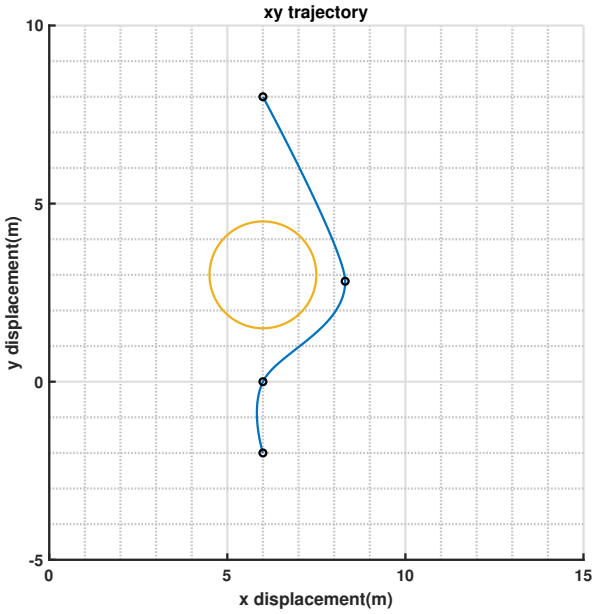

Figure 25: $x(t)$ vs $y(t)$

5 Conclusion

We have produced a generalised algorithm for trajectory generation with via point planning of a 2-D 2-Link robot to avoid circular obstacles. Our algorithm uses both analytical methods in combination with numerical depth search to find the optimal via point with a limited resolution. Although the algorithm is intended to be as general as possible, assumptions are still needed to simplify the problem and compute the adequate location of the via point.

References

[1] Christos Alexopoulos and Paul M Griffin. "Path planning for a mobile robot". In: *IEEE Transactions on systems, man, and cybernetics* 22.2 (1992), pp. 318–322.