ROBT 403 Laboratory Report 3

Daniil Filimonov
Robotics Engineering, SEDS
Nazarbayev University
Astana, Kazakhstan
daniil.filimonov@nu.edu.kz

Abdirakhman Onabek
Robotics Engineering, SEDS
Nazarbayev University
Astana, Kazakhstan
abdirakhman.onabek@nu.edu.kz

Kir Smolyarchuk
Robotics Engineering, SEDS
Nazarbayev University
Astana, Kazakhstan
kir.smolyarchuk@nu.edu.kz

***Abstract*—This report demonstrates the outcome of experiments conducted in Matlab in Lab 9 and Lab 10.**

I. Introduction

Robotic manipulators play a crucial role in industrial automation, medical surgery, space exploration, and various other applications requiring precise and repeatable movements. The ability of a manipulator to perform tasks efficiently depends heavily on its trajectory planning algorithms, which determine the paths that the manipulator's joints and end-effector follow over time [1].

Trajectory planning involves computing the desired position, velocity, and acceleration profiles for each joint of the manipulator, ensuring smooth and collision-free motion between specified points. For tasks that require high precision and smoothness, such as assembly operations or delicate material handling, it is essential to generate trajectories that not only reach the desired positions but also have continuous higher-order derivatives [2].

A common approach to achieving smooth motion is the use of polynomial functions to model the joint trajectories. Quintic polynomials, in particular, are advantageous because they provide continuity up to the acceleration level, satisfying position, velocity, and acceleration constraints at the start and end of the motion segments [3].

In this work, we focus on the trajectory planning of a 3-degree-of-freedom (3-DOF) planar manipulator. The primary objectives are to generate smooth joint trajectories using quintic polynomials, compute the inverse kinematics for intermediate via points, and analyze the dexterous workspace of the manipulator.

A. Quintic Polynomial Trajectories

Quintic polynomials are sixth-order polynomials (degree five) used to define the joint trajectories $\theta_i(t)$ of the manipulator, where i denotes the joint number. The use of quintic polynomials allows for the specification of initial and final positions, velocities, and accelerations, ensuring smooth transitions between motion segments. This is particularly important in applications where sudden changes in acceleration (jerk) can lead to mechanical stress and reduced precision [4].

The quintic polynomial trajectory is defined as:

$$\theta_i(t) = a_{0i} + a_{1i}t + a_{2i}t^2 + a_{3i}t^3 + a_{4i}t^4 + a_{5i}t^5 \quad (1)$$

where a_{0i} to a_{5i} are coefficients determined based on boundary conditions.

B. Via Points and Trajectory Segmentation

In practical applications, a manipulator often needs to pass through specific intermediate points, known as via points, before reaching the final destination. Via points are crucial for avoiding obstacles, optimizing motion paths, or meeting task-specific requirements [1]. By segmenting the overall trajectory into smaller segments between via points, we can apply quintic polynomial planning to each segment, ensuring smoothness throughout the entire path.

C. Inverse Kinematics

Inverse kinematics is the process of determining the joint angles θ_i that achieve a desired end-effector position (x, y) and orientation θ. Solving the inverse kinematics problem is essential for mapping the Cartesian space trajectories of the end-effector to the corresponding joint space trajectories required for actuator control [3].

For a 3-DOF planar manipulator, the inverse kinematics involves solving a set of nonlinear equations derived from the manipulator's geometry. Accurate and efficient inverse kinematics solutions enable the manipulator to follow complex paths accurately.

D. Dexterous Workspace Analysis

The dexterous workspace of a manipulator is the set of all positions and orientations that the end-effector can reach with arbitrary orientations, while satisfying joint limitations [5]. Understanding the workspace is critical for task planning and ensuring that desired trajectories lie within the manipulator's capabilities.

By computing and visualizing the dexterous workspace, we can evaluate the manipulator's ability to perform tasks in various regions of its operating environment. This analysis aids in optimizing manipulator design and placement within a workspace to maximize efficiency and effectiveness.

E. Applications and Significance

The methods described in this paper have broad applications in fields requiring precise motion control, such as:

- **Industrial Automation:** Automated assembly lines, pick-and-place operations, and CNC machining require precise

trajectory planning to enhance productivity and product quality [6].

- **Medical Robotics:** Surgical robots benefit from smooth and accurate motion to perform delicate procedures with minimal invasiveness [7].
- **Space Robotics:** Robotic arms on spacecraft or planetary rovers need reliable trajectory planning to manipulate objects in unpredictable environments [8].
- **Service Robotics:** Robots assisting in domestic tasks or human-robot interaction scenarios require safe and smooth motion to operate effectively alongside humans [9].

F. Contributions

This paper presents a comprehensive approach to trajectory planning for a 3-DOF planar manipulator, including:

1) Development of quintic polynomial trajectories that ensure smooth joint motion with continuous position, velocity, and acceleration profiles.
2) Implementation of inverse kinematics solutions to compute joint angles for specified end-effector positions and orientations at via points.
3) Analysis and visualization of the manipulator's dexterous workspace to validate the reachable regions and orientations.
4) Demonstration of trajectory execution that integrates the planned joint trajectories with forward kinematics to confirm end-effector path accuracy.

G. Organization of the Paper

The remainder of the paper is organized as follows: Section II details the methods used for trajectory planning, inverse kinematics, and workspace computation. Section III presents the results, including simulations and visualizations of the manipulator's motion and workspace. Section IV discusses the implications of the findings and potential areas for future research. Finally, Section V concludes the paper with a summary of the key contributions.

H. Notation and Terminology

For clarity, the following notation and terminology are used throughout the paper:

- θ_i: Joint angle of joint i.
- l_i: Length of link i of the manipulator.
- t: Time variable.
- t_f: Final time of a trajectory segment.
- a_{ji}: Coefficient of the j^{th} term in the polynomial for joint i.
- (x, y): Cartesian coordinates of the end-effector.
- θ: Orientation of the end-effector.
- Via Points: Intermediate points that the manipulator must pass through during motion.
- Dexterous Workspace: The set of all positions and orientations that the end-effector can reach with arbitrary orientations.

By thoroughly explaining the methodology and providing detailed analyses, this paper aims to contribute to the field of robotic manipulator trajectory planning and offer insights that can be applied to various robotic systems.

II. Methods

In this section, we describe the methodology used to generate smooth trajectories for a 3-DOF planar manipulator. The approach involves generating quintic polynomial trajectories, solving inverse kinematics for via points, and computing the dexterous workspace of the manipulator.

A. Quintic Polynomial Trajectory Planning

To achieve smooth motion between initial and desired configurations, we employ quintic polynomials for each joint angle $\theta_i(t)$. The quintic polynomial for joint i is defined as:

$$\begin{aligned}\theta_i(t) = a_{0i} + a_{1i}t + a_{2i}t^2 \\ + a_{3i}t^3 + a_{4i}t^4 + a_{5i}t^5\end{aligned} \tag{2}$$

The corresponding velocity and acceleration profiles are obtained by differentiating the position equation:

$$\begin{aligned}\dot{\theta}_i(t) = a_{1i} + 2a_{2i}t + 3a_{3i}t^2 \\ + 4a_{4i}t^3 + 5a_{5i}t^4\end{aligned} \tag{3}$$

$$\begin{aligned}\ddot{\theta}_i(t) = 2a_{2i} + 6a_{3i}t + 12a_{4i}t^2 \\ + 20a_{5i}t^3\end{aligned} \tag{4}$$

B. Boundary Conditions

The coefficients a_{0i} to a_{5i} are determined using boundary conditions at the start ($t = 0$) and end ($t = t_f$) of the trajectory segment. The boundary conditions are as follows:

At $t = 0$:

$$\theta_i(0) = \theta_{i,0} \tag{5}$$

$$\dot{\theta}_i(0) = \dot{\theta}_{i,0} \tag{6}$$

$$\ddot{\theta}_i(0) = \ddot{\theta}_{i,0} \tag{7}$$

At $t = t_f$:

$$\theta_i(t_f) = \theta_{i,f} \tag{8}$$

$$\dot{\theta}_i(t_f) = \dot{\theta}_{i,f} \tag{9}$$

$$\ddot{\theta}_i(t_f) = \ddot{\theta}_{i,f} \tag{10}$$

C. Solving for Polynomial Coefficients

The initial coefficients are determined directly from the initial conditions:

$$a_{0i} = \theta_{i,0} \tag{11}$$

$$a_{1i} = \dot{\theta}_{i,0} \tag{12}$$

$$a_{2i} = \frac{1}{2}\ddot{\theta}_{i,0} \tag{13}$$

To find the remaining coefficients a_{3i}, a_{4i}, and a_{5i}, we set up a system of equations using the boundary conditions at $t = t_f$. From the position equation at $t = t_f$:

$$\begin{aligned}\theta_{i,f} = a_{0i} + a_{1i}t_f + a_{2i}t_f^2 \\ + a_{3i}t_f^3 + a_{4i}t_f^4 + a_{5i}t_f^5\end{aligned} \tag{14}$$

Subtracting known terms, we define:

$$d_{1i} = \theta_{i,f} - \left(a_{0i} + a_{1i}t_f + a_{2i}t_f^2\right) \tag{15}$$

Similarly, from the velocity and acceleration equations at $t = t_f$:

$$d_{2i} = \dot{\theta}_{i,f} - (a_{1i} + 2a_{2i}t_f) \tag{16}$$

$$d_{3i} = \ddot{\theta}_{i,f} - 2a_{2i} \tag{17}$$

The system of equations to solve for a_{3i}, a_{4i}, and a_{5i} is:

$$\begin{bmatrix} t_f^3 & t_f^4 & t_f^5 \\ 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_{3i} \\ a_{4i} \\ a_{5i} \end{bmatrix} = \begin{bmatrix} d_{1i} \\ d_{2i} \\ d_{3i} \end{bmatrix} \tag{18}$$

Solving the linear system in Eq. (18) yields the coefficients a_{3i}, a_{4i}, and a_{5i}.

D. Via Points Generation

To ensure the manipulator follows a straight-line path in Cartesian space, we generate via points between the initial and desired end-effector positions. The total number of via points is $N = n_{\text{via}} + 2$, where n_{via} is the number of specified via points.

The positions of the via points are computed using linear interpolation:

$$x_{\text{via}}(k) = x_{\text{initial}} + \frac{k}{N-1}\left(x_{\text{desired}} - x_{\text{initial}}\right) \tag{19}$$

$$y_{\text{via}}(k) = y_{\text{initial}} + \frac{k}{N-1}\left(y_{\text{desired}} - y_{\text{initial}}\right) \tag{20}$$

Similarly, the orientation at each via point is interpolated:

$$\theta_{\text{via}}(k) = \theta_{\text{initial}} + \frac{k}{N-1}\left(\theta_{\text{desired}} - \theta_{\text{initial}}\right) \tag{21}$$

where $k = 0, 1, \ldots, N-1$.

E. Inverse Kinematics for Via Points

For each via point $(x_{\text{via}}(k), y_{\text{via}}(k), \theta_{\text{via}}(k))$, we compute the joint angles using inverse kinematics. The inverse kinematics for a 3-DOF planar manipulator involves the following steps:

1) Wrist Position Computation: First, calculate the wrist position (x_w, y_w) by subtracting the contribution of the third link:

$$x_w = x_{\text{via}}(k) - l_3 \cos\theta_{\text{via}}(k) \tag{22}$$

$$y_w = y_{\text{via}}(k) - l_3 \sin\theta_{\text{via}}(k) \tag{23}$$

2) Elbow Joint Angle θ_{2k}: Compute the elbow joint angle θ_{2k} using the Law of Cosines:

$$\theta_{2k} = \cos^{-1}\left(\frac{x_w^2 + y_w^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \tag{24}$$

3) Shoulder Joint Angle θ_{1k}: Calculate the shoulder joint angle θ_{1k} using trigonometric relations:

$$\theta_{1k} = \tan^{-1}\left(\frac{y_w}{x_w}\right) - \tan^{-1}\left(\frac{l_2 \sin\theta_{2k}}{l_1 + l_2\cos\theta_{2k}}\right) \tag{25}$$

4) Wrist Joint Angle θ_{3k}: Determine the wrist joint angle θ_{3k} by:

$$\theta_{3k} = \theta_{\text{via}}(k) - \theta_{1k} - \theta_{2k} \tag{26}$$

By solving Eqs. (24)–(26) for each via point k, we obtain the joint angles that serve as boundary conditions for the polynomial trajectories.

F. Dexterous Workspace Computation

The dexterous workspace of the manipulator is the set of all points that the end-effector can reach with all orientations within the joint limits. To compute the workspace:

1) Joint Angle Ranges: Define the joint angle limitations:

$$\theta_{1,\min} \le \theta_1 \le \theta_{1,\max} \tag{27}$$

$$\theta_{2,\min} \le \theta_2 \le \theta_{2,\max} \tag{28}$$

$$\theta_{3,\min} \le \theta_3 \le \theta_{3,\max} \tag{29}$$

2) Generating the Workspace Points: Loop over the ranges of θ_1, θ_2, and θ_3 to compute the end-effector positions (x, y):

$$\begin{aligned}x = l_1\cos\theta_1 + l_2\cos(\theta_1 + \theta_2) \\ + l_3\cos(\theta_1 + \theta_2 + \theta_3)\end{aligned} \tag{30}$$

$$\begin{aligned}y = l_1\sin\theta_1 + l_2\sin(\theta_1 + \theta_2) \\ + l_3\sin(\theta_1 + \theta_2 + \theta_3)\end{aligned} \tag{31}$$

By plotting all computed (x, y) points, we visualize the dexterous workspace of the manipulator.

G. Trajectory Execution

The trajectory is executed by computing the joint angles $\theta_i(t)$ at each time step using the quintic polynomials. The end-effector positions $(x(t), y(t))$ are then calculated using forward kinematics:

$$\begin{aligned}x(t) = l_1\cos\theta_1(t) + l_2\cos(\theta_1(t) + \theta_2(t)) \\ + l_3\cos(\theta_1(t) + \theta_2(t) + \theta_3(t))\end{aligned} \tag{32}$$

$$\begin{aligned}y(t) = l_1\sin\theta_1(t) + l_2\sin(\theta_1(t) + \theta_2(t)) \\ + l_3\sin(\theta_1(t) + \theta_2(t) + \theta_3(t))\end{aligned} \tag{33}$$

This process ensures that the manipulator follows the desired path while maintaining smooth joint motion profiles.

H. Summary

By integrating quintic polynomial trajectories with via point generation, inverse kinematics, and workspace computation, we achieve smooth and continuous motion for the manipulator. This method satisfies position, velocity, and acceleration constraints at each via point, ensuring precise and efficient trajectory planning.

III. Theoretical Tasks

A. Task 1

In this task, we were asked to obtain the workspace of the planar robot with constraints set as there would be only three operatable joints, while two others would be locked. Also, we were asked to make two workspaces – first one for joints able to move in range from - $\pi/4$ to $\pi/4$ (condition 1), and second one to move in range from - $\pi/2$ to $\pi/2$ (condition 2). For this, we have measured all the link lenghts, and have written a code that with a predefined angle step calculates all possible points for the manipulator. An interested reader is welcomed to read a complete code in the *DexterousSpace.py* file in the github. Briefly, the code uses nested loops to iterate over all combinations of θ_1 and θ_2, and computes the X and Y coordinates of the robot's end-effector for each combination. For a given set of joint angles, the end-effector's position (X, Y) is calculated using the forward kinematics equations for the 3R robot:

- 1. X is the sum of the cosines of the joint angles, multiplied by the corresponding link lengths.
- 2. Y is the sum of the sines of the joint angles, also multiplied by the link lengths.

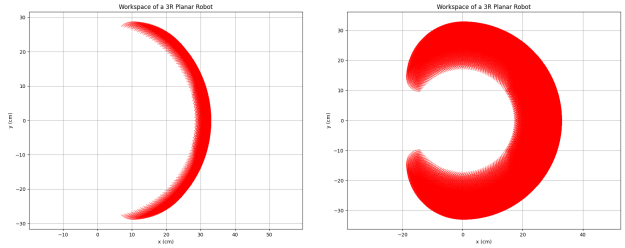

Fig. 1. Maneuverability of robot depending on 3 different footprint width, different steering command

The figure1 shows two workspaces calculated for condition 1 and condition 2 respectively.

B. Tasks 2, 3 & 4

Further, we were asked to implement the 'cubic coefficients' and 'cubic trajectory' for a smoother manipulator trajectory. To make this task more interesting, we advanced this task to implement 'quintic polynomial' which has advantage of considering acceleration in comparison to his 'qubic' brother. By adjucting start and end accelerations we make joint moves much less jerky as it would be with 'cubic' polynomial trajectory. The theoretical description of calculation was explained in equations 8-18. First three horizontal subplots of figures 2-4 show the plots of trajectories, velocities, and accelerations for each joint. As we can see, in each case the start and end accelerations are zero, which means moves without vibrations.

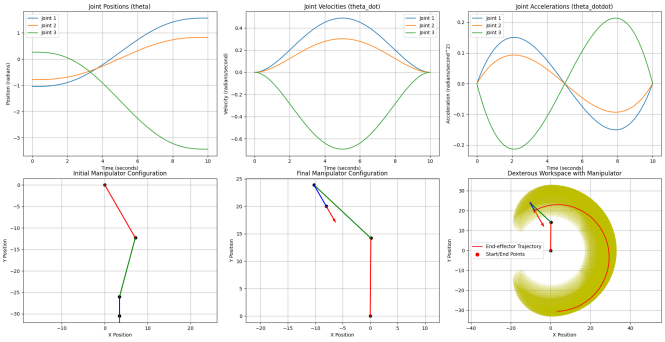

Fig. 2. Maneuverability of robot depending on 3 different footprint width, different steering command

Next, we had to show the trajectory of the end-effector form starting position to the ending position. For this, we just calculated the coordinates of the end-effector at each time step and then plotted as a red curve on the very last subplot with the workspace in figures 2-4. As could be seen, in each case the trajectory is within the workspace.

Finally, we were asked to depict the manipulator and its joints in different colors in the beginning of the operation and in the end. For this, we have developed *plot_manipulator* function that visualizes a 3-link planar robotic manipulator in 2D space, taking into account the joint angles and link lengths provided. It calculates the positions of each joint and the end-effector using forward kinematics, where the cumulative sum of joint angles is used to determine the orientation of each link. The links are plotted in different colors (red, green, and blue) for clarity, with black points marking the joints. Additionally, the function displays an arrow at the end-effector to show its orientation. This function is used in the codes *PolyTraj.py* and *PolyTrajVia.py*. Results of the initial and final state of manipulator are plotted in fourth and fifth graphs respectively of figures 2-4 .

C. Task 5

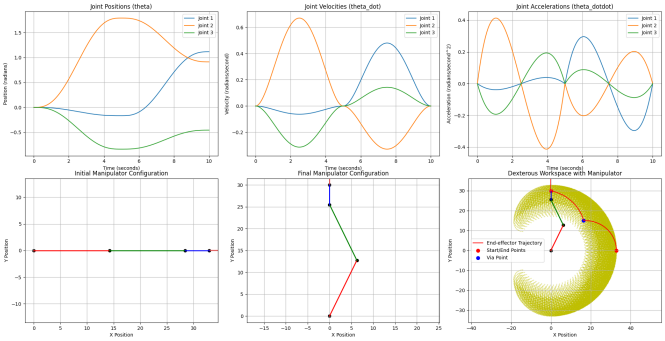

Fig. 3. Maneuverability of robot depending on 3 different footprint width, different steering command

In the final theoretical task, we were asked to make function to generate joint-space cubic trajectories between multiple via points in the Cartesian workspace. For this, we have decided to work out a separate *PolyTrajVia.py* file that would be adapted for the via points. All the math behind via points is described in equations 19-26. As could be seen in figures 3-4, we can successfuly model the movement of a manipulator via specified number of points. As such, in figure3 we have only one via-point with pretty curved trajectory, while in figure4 we have four via-points and much straighter route of the end-effector.

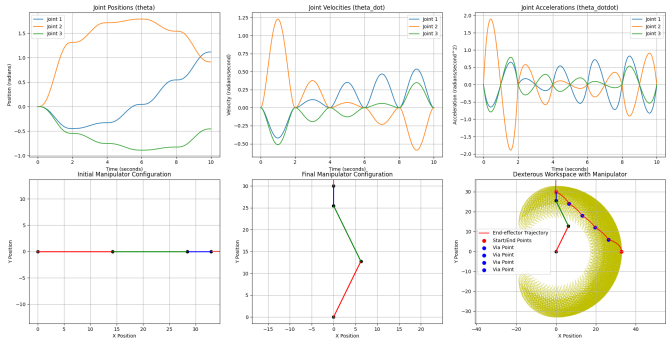

Fig. 4. Maneuverability of robot depending on 3 different footprint width, different steering command

IV. Practical Tasks

A. Single Fifth-Order Polynomial

In this task, we implemented a fifth-order polynomial trajectory from the initial home position to a final pose using inverse kinematics and polynomial trajectory planning. The robot's initial pose was set to the home position, and the final pose was at coordinates (20, 20, -10), where x axis is along the home position. The inverse kinematics function was used to compute the joint angles needed to reach the final pose. A fifth-order polynomial was then calculated for each joint to ensure a smooth trajectory for position, velocity, and acceleration. The robot followed this polynomial in real time, executing the motion from the initial to the final pose smoothly.

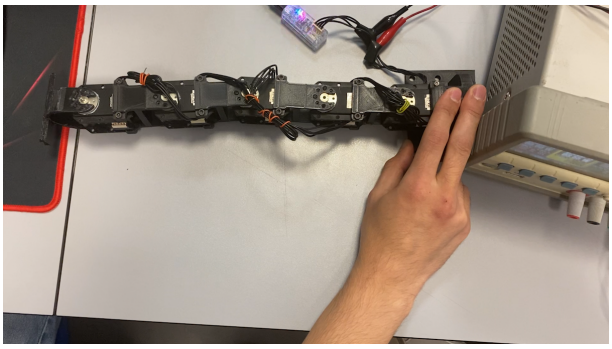

Fig. 5. Home position

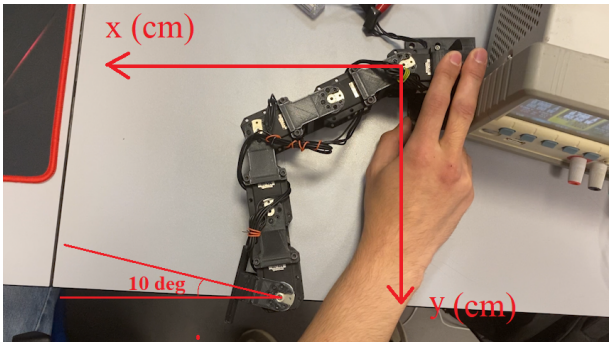

Fig. 6. Final position (x, y, theta) = (20, 20, -10)

B. One Via Point

This task involved generating a more complex trajectory where the robot moved from the initial home position to a final point, passing through a via point. The robot started from the home position, moved to the via point at (20, 20, -10), and then to the final point, which we set using inverse kinematics. For both segments (initial to via and via to final), we computed fifth-order polynomials for each joint, ensuring smooth transitions in position, velocity, and acceleration. The robot executed this trajectory in real time, first reaching the via point and then continuing smoothly to the final point, showcasing the ability to handle multiple points in the motion plan.

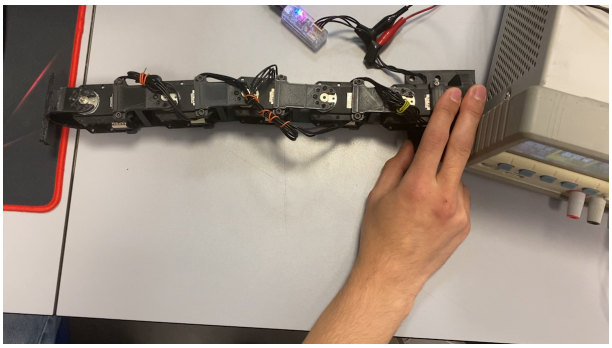

Fig. 7. Home position

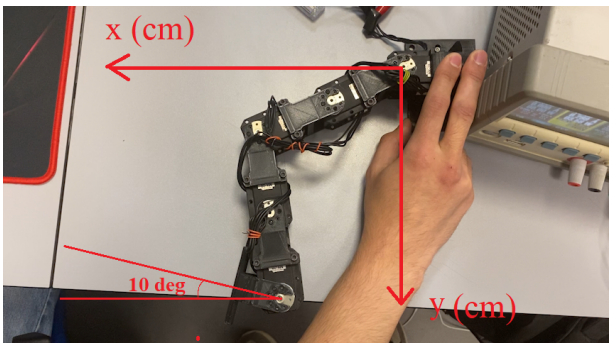

Fig. 8. Via point (x, y, theta) = (20, 20, -10)

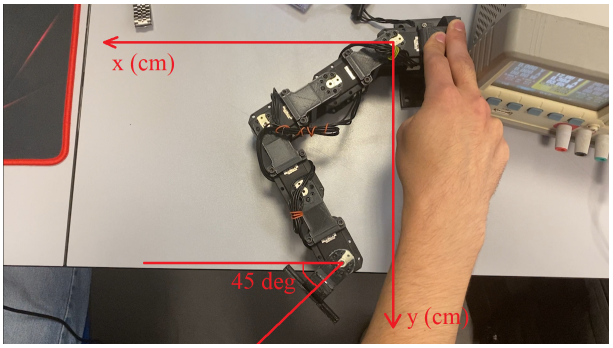

Fig. 9. Final position (x, y, theta) = (10, 20, 45)

V. Code Implementation

This section describes the code implementation used for trajectory planning and execution on a robotic arm using inverse kinematics and fifth-order polynomial interpolation. The code is structured to calculate the required joint angles for the robot to move smoothly between an initial pose and one or more target points.

A. Libraries and Dependencies

The code begins by including the necessary libraries:

- `ros/ros.h` – Provides ROS functionalities for communication with the robot.
- `std_msgs/Float64.h` – Used for sending joint angle commands to the robot.
- `eigen3/Eigen/Dense` – Eigen library is included to handle matrix operations, which are essential for solving inverse kinematics and trajectory planning problems.
- `cmath` and `math.h` – Used for trigonometric calculations and mathematical functions.

B. Inverse Kinematics Calculation

The `IK` function is responsible for calculating the joint angles of the robotic arm using inverse kinematics. Given the desired position of the end-effector `(x, y)` and its orientation `theta`, the function computes the angles of the shoulder, elbow, and wrist joints. The procedure is as follows:

- First, the position of the wrist is computed by subtracting the length of the wrist segment from the total position.
- Next, using geometric relationships and the law of cosines, the elbow angle (`theta2`) is calculated.
- The shoulder angle (`theta1`) is then computed using trigonometric relationships.
- Finally, the wrist angle (`theta3`) is determined by subtracting the sum of the shoulder and elbow angles from the overall desired angle `theta`.
- The joint angles are returned as an array of five elements, where the second and fourth elements represent unused joint angles in this setup.

C. Fifth-Order Polynomial Trajectory Planning

To ensure smooth motion, the function `calculateFifthOrderPolynomial` is used to generate a fifth-order polynomial for each joint. The polynomial provides a smooth trajectory for the joint angles, ensuring continuous position, velocity, and acceleration. The inputs to the function include:

- `t_f` – Total time of the motion.
- `timesteps_per_second` – The resolution of the trajectory in terms of time steps.
- `theta_0` and `theta_f` – The initial and final angles for the joint.
- `theta_dot_0`, `theta_dot_f` – The initial and final velocities.
- `theta_dotdot_0`, `theta_dotdot_f` – The initial and final accelerations.

This function solves for the coefficients of the fifth-order polynomial by constructing a system of equations using boundary conditions (initial and final values for position, velocity, and acceleration). The Eigen library is used to solve the linear system, resulting in the polynomial coefficients. The function then calculates the position, velocity, and acceleration at each time step, returning them in a structure.

D. Publishing Joint Commands

The `publish` function sends the calculated joint angles to ROS topics corresponding to the joints of the robot. Each joint's angle is published as a `std_msgs::Float64` message to its respective ROS topic. This ensures that the calculated angles are communicated to the robot's actuators, allowing it to perform the desired motion.

E. Main Function

The `main` function controls the overall execution of the program:

- It first initializes the ROS node and sets up publishers for each joint of the robotic arm.
- The initial inverse kinematics (IK) calculation is performed to determine the joint angles for the home position of the robot. These angles are then stored in an array.
- Afterward, the target position `(x, y, theta)` is set to `(20, 20, -10)`, and the IK function is used again to calculate the joint angles for this position.
- Using the calculated joint angles, the `calculateFifthOrderPolynomial` function is invoked to compute the trajectory for each joint. Separate polynomials are generated for the shoulder, elbow, and wrist joints.
- The program then moves the robot from the initial home position to the target via point using the calculated polynomial trajectory.
- To demonstrate a more complex motion, another target position is calculated using IK, and the robot follows a second polynomial trajectory to move from the via point to the final position.

The `ros::spinOnce()` function ensures the ROS node stays active, while `loop_rate.sleep()` maintains a 50 Hz control loop, ensuring smooth real-time operation of the robot.

F. Time Management and Trajectory Execution

The program uses ROS time functions to manage trajectory execution:

- A timer starts when the program begins executing, and the program keeps track of the elapsed time.
- During the first 5 seconds, the initial joint angles are published to move the robot to its starting position.
- Afterward, the program publishes joint commands at regular intervals (based on the calculated polynomial trajectories) to smoothly move the robot between the waypoints.

By the end of execution, the robot has followed the desired trajectory, passing through the via point and stopping at final position using fifth-order polynomial trajectories for smooth motion.

VI. Conclusion

In this report, we implemented and tested fifth-order polynomial trajectory planning on a robotic manipulator. Using inverse kinematics, we calculated joint angles to guide the robot from an initial position to a final point, passing through via points. The polynomial ensured smooth transitions in position, velocity, and acceleration. In Python, we plotted the trajectory graphs for position, velocity, and acceleration using matplotlib. These plots helped visualize the robot's movement and verify that the motion followed the expected profile. This approach effectively generated precise and smooth movements for complex motion planning tasks.

References

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Science & Business Media, 2010.

[2] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall Upper Saddle River, 2005.

[3] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 2006.

[4] S. B. Niku, *Introduction to Robotics: Analysis, Control, Applications*. John Wiley & Sons, 2010.

[5] T. Yoshikawa, *Foundations of Robotics: Analysis and Control*. MIT press, 1990.

[6] M. P. Groover, *Automation, Production Systems, and Computer-integrated Manufacturing*. Prentice Hall Press, 2007.

[7] R. H. Taylor, A. Menciassi, G. Fichtinger, P. Fiorini, and P. Dario, "Medical robotics in computer-integrated surgery," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1133–1137, 2016.

[8] A. Ellery, *Planetary Rovers: Robotic Exploration of the Solar System*. Springer, 2015.

[9] P. Kormushev and D. G. Caldwell, "Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input," *Advanced Robotics*, vol. 27, no. 12, pp. 817–828, 2013.