
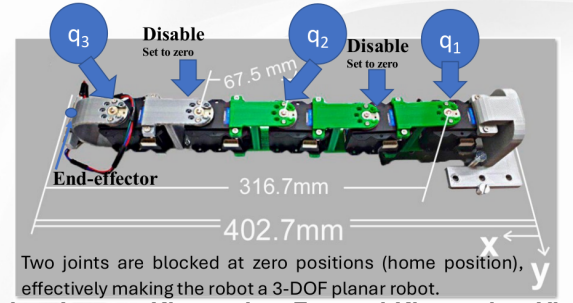


# ROBT 403 LAB 3: 3-DOF Planar Manipulator Trajectory Planning and Implementation on a Real Robot

Dinmukhamet Murat 

**Abstract**—This report describes the development and implementation of joint-space cubic trajectory planning for a 3-degree-of-freedom (3-DOF) planar manipulator. The goal was to move the robot's end-effector through multiple via points in the Cartesian workspace. We computed the workspace and used forward and inverse kinematics to determine joint angles and movement paths. Joint-space cubic polynomial trajectories ensured smooth transitions between via points. The manipulator's configuration was visualized using Python, and the trajectories were executed on a real robot via ROS Melodic. This work highlights the importance of inverse kinematics and trajectory planning for accurate, smooth robotic manipulation.

**Index Terms**—3-DOF Planar Manipulator, Joint-Space Trajectory Planning, Inverse Kinematics, Forward Kinematics, Via Points



## I. INTRODUCTION

Robotic manipulators are essential in automation, requiring precise control of movement and path planning. This report describes the development of a joint-space cubic trajectory planning system for a 3-degree-of-freedom (3-DOF) planar manipulator. The goal was to move the robot's end-effector through specific via points in the Cartesian workspace using smooth cubic polynomial trajectories.

We implemented forward and inverse kinematics to compute joint angles and developed Python visualizations to track the manipulator's movements. Additionally, we executed the planned trajectories on a real robot using ROS Melodic to validate performance. This report outlines the methods used and the successful integration of trajectory planning into the robotic system.

## II. SYSTEM SETUP

The robotic system used for the lab consists of a 5-DoF planar manipulator, where only three joints are active. We utilized ROS Melodic for communication and control, along with necessary ROS packages such as dynamixel\_motor and moveit\_arm for controlling the robot.

The manipulator was modeled using three links with lengths L1L1, L2L2, and L3L3, which were set based on the physical dimensions of the manipulator. The joint angles for the initial and final configurations were defined.

### A. Required commands

Clone the necessary files:

git clone [https://github.com/arebgun/dynamixel\\_motor.git](https://github.com/arebgun/dynamixel_motor.git)

git clone <https://github.com/KNurlanZ/snake-noetic.git>

Establish motor connections using the following ROS commands:

```
roslaunch my_dynamixel_tutorial controller_manager.launch
roslaunch my_dynamixel_tutorial
start_moveit_arm_controllers.launch
```

Once the system setup is complete, the joints of the manipulator can be controlled using ROS topics.

## III. TASK1: WORKSPACE COMPUTATION

The first task was to compute the workspace of the 3-DoF planar manipulator using forward kinematics. The robot's workspace consists of all the reachable positions of its end-effector for a range of joint angles. The robot's joint angles were varied within a range of  $[-90^\circ, +90^\circ]$ .

Kinematic Model:

The forward kinematic model for a 3-DoF planar manipulator is derived as follows:

$$x = L1\cos(q1) + L2\cos(q1+q2) + L3\cos(q1+q2+q3)$$

$$y = L1\sin(q1) + L2\sin(q1+q2) + L3\sin(q1+q2+q3)$$

Where  $q1, q2, q3$  are the joint angles, and  $L1, L2, L3$  are the link lengths. A Python script was developed to compute and plot the workspace by iterating through a range of joint angles. The resulting plot shows all possible end-effector positions the robot can achieve, confirming the robot's reachable workspace.

## IV. TASK2: JOINT-SPACE CUBIC TRAJECTORY GENERATION

Cubic polynomial trajectory generation was implemented to smoothly move the robot's joints from one configuration to another.

Cubic Polynomial Equation

The cubic trajectory is represented by:  $\theta(t) = a0 + a1t + a2t^2 + a3t^3$

Where the coefficients  $a0, a1, a2, a3$  are determined by initial and final conditions of the joint positions and velocities. The

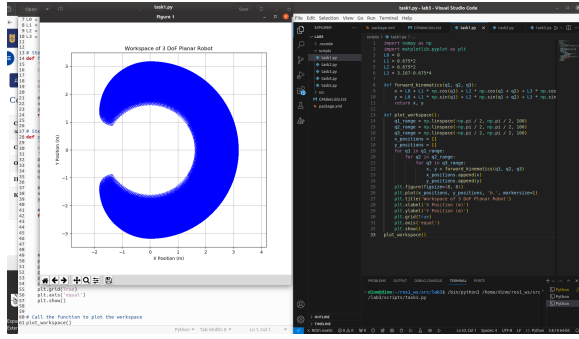


Fig. 1: Task1

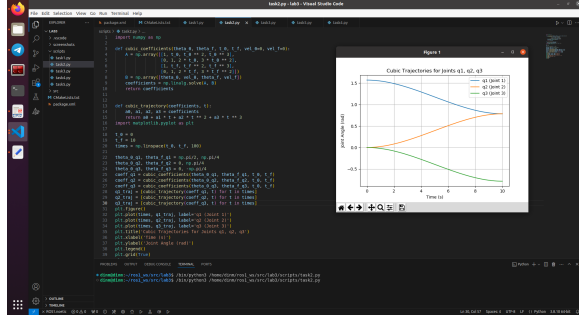


Fig. 2: Task2

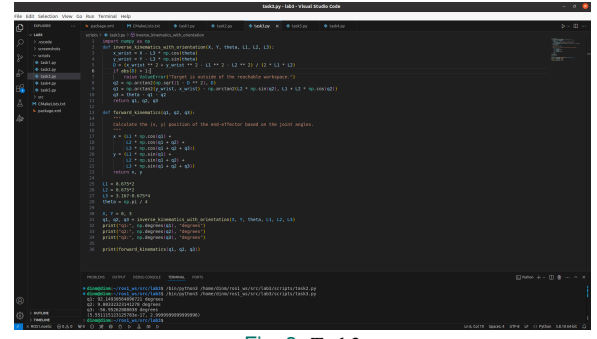


Fig. 3: Task3

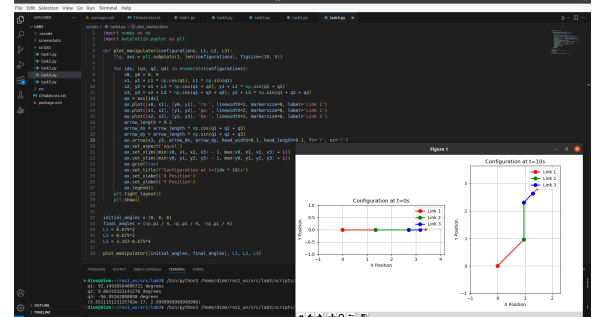


Fig. 4: Task4

following Python functions compute these coefficients and generate the trajectory:

```
def cubic_coefficients(theta_0, theta_f,
    t_0, t_f, vel_0=0, vel_f=0):
    A = np.array([[1, t_0, t_0 ** 2,
        t_0 ** 3], [0, 1, 2 * t_0,
        3 * t_0 ** 2], [1, t_f,
        t_f ** 2, t_f ** 3], [0, 1,
        2 * t_f, 3 * t_f ** 2]])
    B = np.array([theta_0, vel_0,
        theta_f, vel_f])
    coefficients = np.linalg.solve(A, B)
    return coefficients
```

The cubic trajectory was plotted for each joint angle  $q_1, q_2, q_3$  over time, showing smooth motion from initial to final configurations.

## V. TASK 3: INVERSE KINEMATICS WITH ORIENTATION

Inverse kinematics was implemented to calculate the required joint angles to achieve a given end-effector position and orientation.

### Kinematic Solution

The inverse kinematic equations were derived to compute  $q_1, q_2, q_3$  based on a given target position (X, Y) and orientation. The following Python code implements the inverse kinematics:

```
def inverse_kinematics_with_orientation(X,
    Y, theta, L1, L2, L3):
    x_wrist = X - L3 * np.cos(theta)
    y_wrist = Y - L3 * np.sin(theta)
    D = (x_wrist ** 2 + y_wrist ** 2 -
```

```
    L1 ** 2 - L2 ** 2) / (2 * L1 * L2)
    q2 = np.arctan2(np.sqrt(1 - D ** 2), D)
    q1 = np.arctan2(y_wrist, x_wrist) -
        np.arctan2(L2 * np.sin(q2), L1 +
        L2 * np.cos(q2))
    q3 = theta - q1 - q2
    return q1, q2, q3
```

The solution was validated by comparing the forward and inverse kinematic results.

## VI. TASK 4: VISUALIZATION OF MANIPULATOR CONFIGURATION OBJECTIVE

To visualize the manipulator's configuration at any point in its trajectory, including plotting both initial and final configurations. We also needed to display the joint angles and end-effector orientation during the trajectory.

We developed a plot\_manipulator function that takes in the joint configurations and visualizes the 3-DOF manipulator. The manipulator's links are displayed with distinct colors, and an arrow shows the orientation of the end-effector.

The manipulator was successfully visualized at different points in its trajectory. The initial and final configurations were clearly displayed, with arrows indicating the orientation of the end-effector.

## VII. TASK 5: JOINT-SPACE CUBIC TRAJECTORY PLANNING WITH VIA POINTS

To generate joint-space cubic trajectories between multiple via points in the Cartesian workspace and compute the corresponding end-effector trajectory. The via points were dynamically planned with specific orientations for the end-effector. The plot\_via\_points\_trajectory function was implemented to

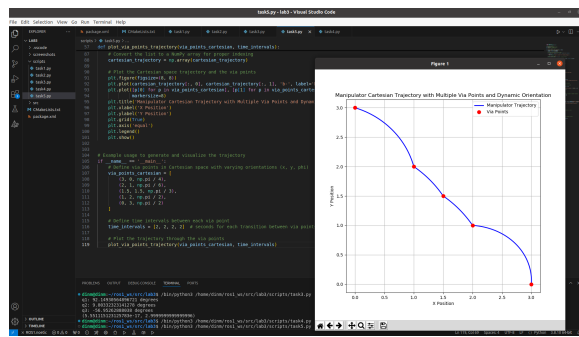


Fig. 5: Task5

generate cubic polynomials for joint-space trajectories and plot the resulting Cartesian trajectory of the manipulator. The inverse kinematics function was used to convert Cartesian via points into joint angles. A cubic polynomial was employed to smoothly interpolate between these points in joint space.

The function successfully plotted the full trajectory of the end-effector through the specified via points, demonstrating the effectiveness of the cubic trajectory planning method. The plots illustrated both the manipulator's joint-space trajectory and the Cartesian trajectory, providing clear visual feedback on its performance.

## VIII. DISCUSSION AND CONCLUSION

The lab successfully implemented joint-space cubic trajectory planning for a 3-DOF planar manipulator, showcasing the ability to visualize manipulator configurations and generate smooth trajectories. The transition from simulation to real robot control illustrated the practical applications of kinematic and trajectory planning techniques, highlighting their importance in robotic motion planning and control.