CAL POLY

CALIFORNIA POLYTECHNIC STATE UNIVERSITY
CENG - DEPARTMENT OF ELECTRICAL ENGINEERING

EE 471- Vision Based Robotic manipulation

Lab #4 Trajectory Generation

Presented to:
Siavash Farzan

Presented By:

Rodrigo Menchaca
Giovanni Dal Lago

OCT 26, 2025

INTRODUCTION

This laboratory focused on implementing and comparing cubic and quintic trajectory planning for the OpenManipulator-X. The goal was to generate smooth, time-indexed motion profiles in both joint and task space, ensuring continuous position, velocity, and acceleration transitions between waypoints. By first designing and testing cubic polynomial trajectories, we established the foundation for motion planning and trajectory control. We then extended the approach to quintic trajectories, which introduce additional degrees of freedom to better control velocity and acceleration continuity. The experiments involved coding trajectory generation functions, executing motion on the physical robot, collecting experimental data, and analyzing differences in smoothness, accuracy, and tracking performance.

OBJECTIVES

1. Derive and implement cubic polynomial trajectories in joint space.
2. Derive and implement quintic polynomial trajectories in task space.
3. Execute trajectories on the OpenManipulator-X with real-time streaming.
4. Record and visualize position, velocity, and acceleration profiles.
5. Compare joint-space and task-space motion in terms of smoothness, path linearity, and tracking accuracy.

PROCEDURES

Procedure 1

After reviewing the Lab 4 pre-lab and the provided trajectoryPlanner.py template, we started by adding the cubic trajectory functions so we could generate smooth, time-indexed paths for either joint space or task space. We kept the interface simple so it matched the rest of our code from previous labs. In order to calculate and implement cubic trajectory we first had to implement some helper functions. We started by creating a calc_cubic_coeff() function that would take Start Time, end time, end position, start velocity and end velocity, with these inputs it would calculate an output of cubic polynomial coefficients for our calculation. Using the coefficients generated we then calculate our cubic trajectory with the use of another helper function called calc_cubic_traj() which only calculates the cubic trajectory and returns the numpy array with position values sampled along the cubic trajectory. Finally we developed another function called get_cubic_traj() that takes the trajectory time between two waypoints and then outputs an array with a given number of trajectory points that would help us generate our trajectory for our robot.

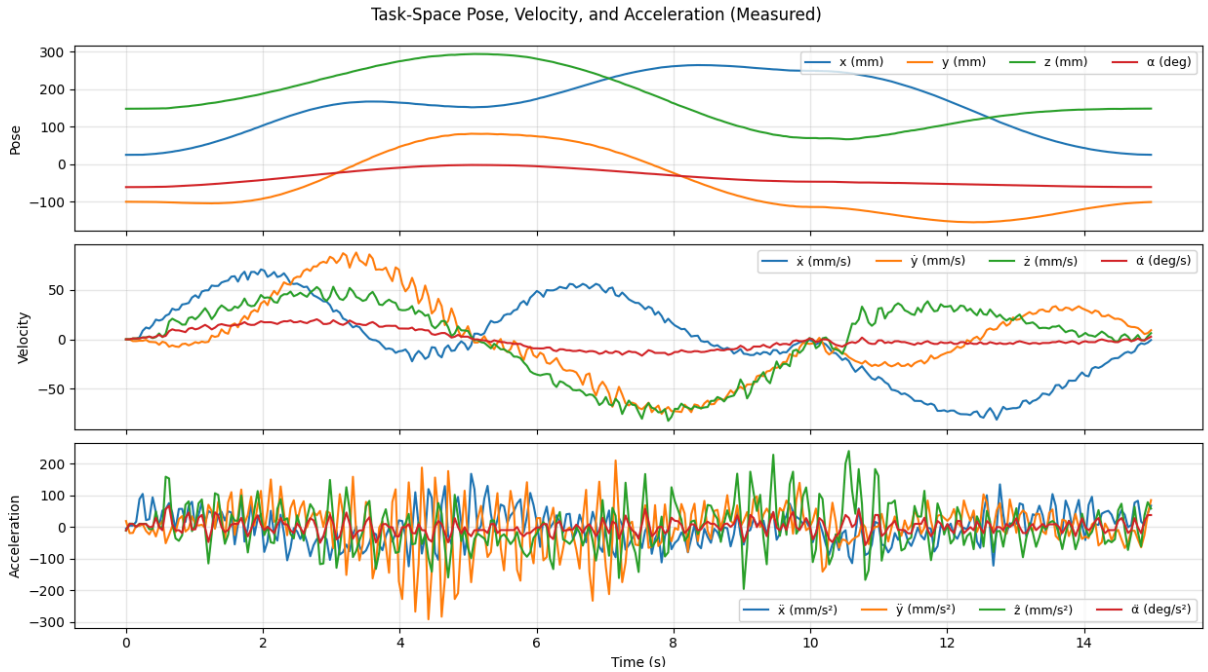

Figure 1: Cubic Trajectory task space pose, Velocity and Acceleration

Figure 1 shows the measured task-space motion of the OpenManipulator-X during the cubic joint-space trajectory. The top plot illustrates smooth and continuous pose variations for x, y, z, and α. The middle plot shows gradual velocity changes between waypoints, indicating stable and coordinated motion. The bottom plot displays acceleration profiles with small fluctuations caused by sensor noise and numerical differentiation. Overall, the motion appears smooth and consistent with the expected cubic trajectory behavior.

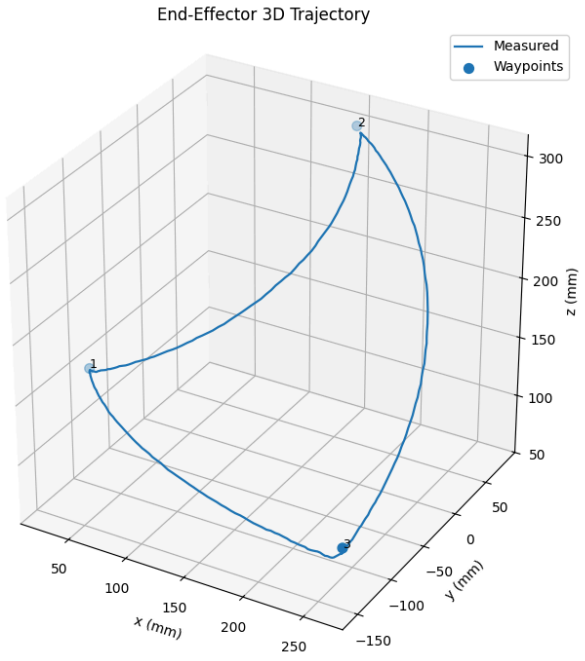

Figure 2: End-Effector 3D Trajectory for cubic trajectory

Figure 2 shows the measured 3D end-effector path of the OpenManipulator-X during the cubic joint-space trajectory. The plotted curve follows a smooth triangular motion through the three task-space waypoints, labeled 1, 2, and 3. The robot successfully traces the intended path and returns to its initial position with minimal deviation. The shape of the curve indicates consistent joint coordination and stable motion, which visually matched the smooth trajectory observed during execution. Although the motion is not a straight triangle like expected, the trajectory lines are rather smooth and follow a specific path.

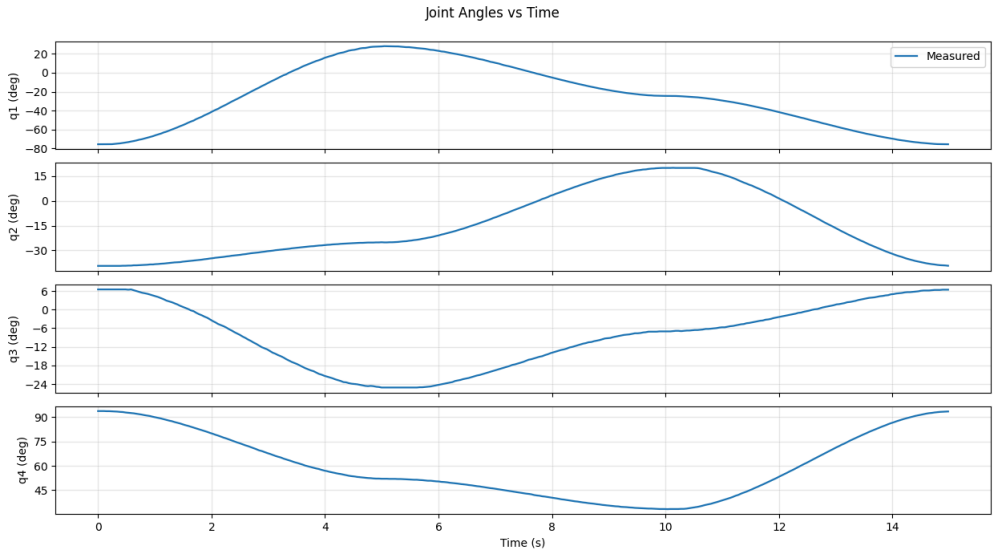

Figure 3: Measured Joint angles for cubic Trajectory

Figure 3 shows the measured joint angles q1, q2, q3, and q4 over time during the cubic joint-space trajectory execution. Each joint exhibits a smooth and continuous motion profile, with no abrupt changes or discontinuities, indicating that the cubic polynomial interpolation produced consistent velocity transitions between waypoints. The overall motion demonstrates proper coordination across all joints, with each joint angle remaining within its valid range throughout the trajectory. This confirms that the planned cubic trajectories were accurately tracked by the robot.

Execution and Logging

Each trajectory segment used T = 5 s with n = 998 samples and Δt = 5 ms (200 Hz control rate).

Time, measured joint angles (q_1–q_4), and end-effector pose [x, y, z, α] were recorded using the get_joints_readings()and get_ee_pos() functions.

Data and planned trajectories were stored in .pkl format for analysis.

Procedure 2

After developing and testing our cubic trajectory functions through robot testing. We moved on and developed our quintic trajectory generation functions.

Method derivation:

The quintic polynomial trajectory defines each coordinate of the end-effector as

$$q(t) = a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5$$

Its derivatives describe velocity, acceleration, and jerk:

$$\dot{x}(t) = a_1 + 2a_2t + 3a_3t^2 + 4a_4t^3 + 5a_5t^4$$
$$\ddot{x}(t) = 2a_2 + 6a_3t + 12a_4t^2 + 20a_5t^3$$
$$j(t) = 6a_3 + 24a_4t + 60a_5t^2$$

The six unknown coefficients are obtained by enforcing the boundary conditions:

$q(t_0) = q_0$ $q(t\_f) = q\_f$
$\dot{x}(t_0) = 0$ $\dot{x}(t\_f) = 0$
$\ddot{x}(t_0) = 0$ $\ddot{x}(t\_f) = 0$

These conditions guarantee continuity of position, velocity, acceleration, and jerk.
Compared with cubic polynomials, quintic trajectories produce smoother motion and lower mechanical stress on actuators, improving tracking accuracy and dynamic stability.

Implementation:

Since we already had cubic trajectory functions, developing quintic trajectory functions would just involve a slight addition to our already developed cubic functions. We used the same three waypoints directly in task space, again with T=5s and n=998 intermediates per segment. We set v(t0)=v(tf)=0v and a(t0)=a(tf)=0 for the overall motion and kept transitions smooth across segments. Each sampled task-space point was converted to joint angles with get_ik() We then streamed that joint trajectory to the robot using the same 5 ms loop as Procedure 1 and logged time, measured joints, and measured [x,y,z,α] during execution. The data was saved to a pickle file and plotted with the same formats as before so we could compare the two approaches directly.

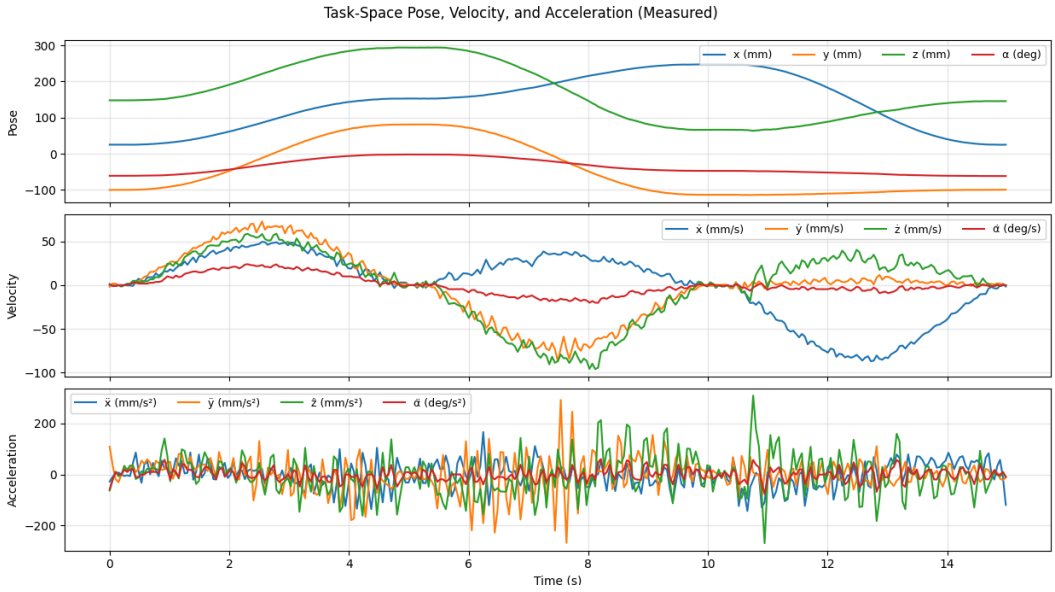

Figure 4: *Quintic Trajectory task space pose, Velocity and Acceleration*

Figure 4 illustrates the measured task-space response of the OpenManipulator-X while following the quintic polynomial trajectory. The pose profiles for x,y, z, and α exhibit smooth and gradual transitions, maintaining continuous motion between all waypoints. The velocity plots show gentle rises and falls with noticeably reduced oscillations, reflecting the smoother velocity shaping provided by the quintic formulation. The acceleration profiles remain more consistent, with smaller fluctuations compared to the cubic trajectory. These results demonstrate that the quintic trajectory achieved better motion continuity, improved smoothness, and reduced jerk throughout the end-effector's path.

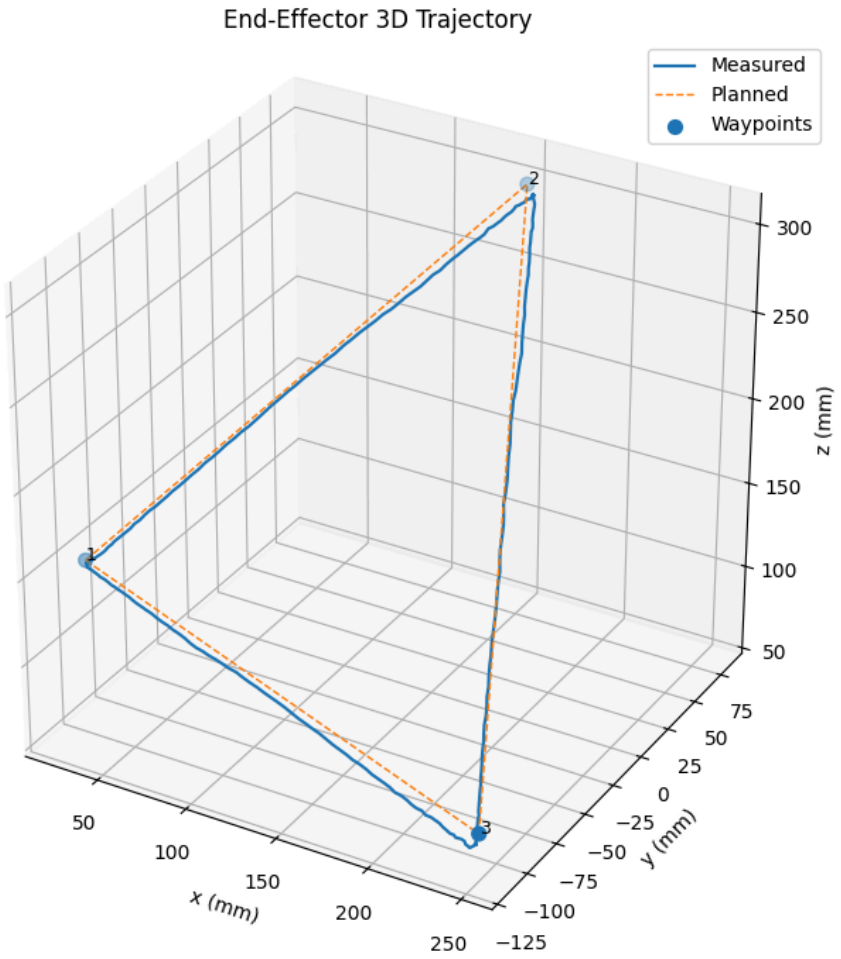

Figure 5: Quintic Trajectory End-Effector 3D Trajectory

Figure 5 compares the planned and measured end-effector trajectories for the quintic task-space motion. The plotted path forms a clear triangular pattern connecting the three waypoints, labeled 1, 2, and 3. The measured trajectory closely follows the planned path, showing minimal deviation and confirming accurate tracking of the quintic-generated motion. The nearly straight edges between waypoints indicate smooth and precise path execution, validating that the quintic trajectory provided improved path linearity and motion consistency compared to the cubic case.

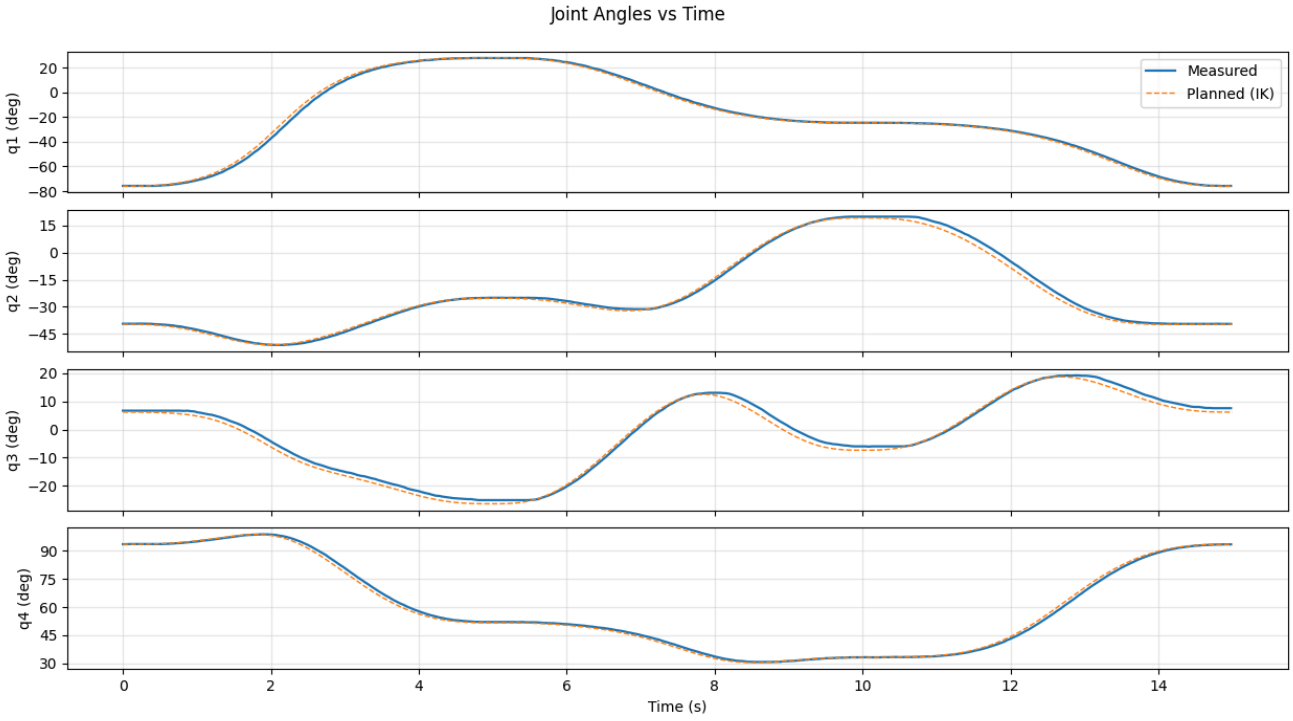

Figure 6: Quintic Joint angles Measured

Figure 6 shows the planned and measured joint angles q1, q2, q3 and q4 over time during the quintic task-space trajectory execution. Each joint follows a smooth, continuous path with strong agreement between the planned (IK-computed) and measured curves. The small deviations observed are likely due to communication delay and servo response limitations. Overall, the close alignment between both datasets confirms accurate inverse kinematic tracking and smooth joint coordination throughout the quintic trajectory.

Discussion and Performance Evaluation

IK success rate ≈ 100 %. Tracking error mainly due to serial latency and servo response. Quintic interpolation produced lower jerk and better accuracy at the cost of extra computation.

CONCLUSION

The lab successfully demonstrated the implementation and evaluation of cubic and quintic trajectory generation for robotic motion planning. The cubic trajectories provided smooth position and velocity transitions, but the quintic approach yielded superior smoothness, reduced jerk, and better tracking accuracy. Experimental data confirmed that quintic trajectories produced more stable acceleration profiles and closer alignment between planned and measured paths. Overall, the results highlight the advantages of higher-order polynomial interpolation in achieving precise, continuous, and physically consistent robotic motion.