



CAL POLY

CALIFORNIA POLYTECHNIC STATE UNIVERSITY
CENG - DEPARTMENT OF ELECTRICAL ENGINEERING

EE 471- Vision Based Robotic manipulation
Lab #3 Inverse Kinematics

Presented to:
Siavash Farzan

Presented By:

Rodrigo Menchaca
Giovanni Dal Lago

OCT 17, 2025

Introduction

Building on Labs 0–2, in Lab 3 we worked with Inverse Kinematics (IK) for the OpenManipulator-X we developed during the Pre-lab work. While forward kinematics maps joint angles to end-effector pose, IK computes the joint angles needed to reach a desired position and orientation.

The pre-lab consisted in solving the IK for the OpenManipulator-X and implementing it in Python as the function “get_ik” that takes a task-space pose and returns the corresponding joint angles (in two equivalent configurations : elbow up and elbow down). This lets us plan motion in task space using (x, y, z) coordinates and pitch instead of joint angles.

During the lab we designed a python script to verify the validity of our IK without the robot transforming back and forth coordinates and joint angles.

Finally, we used our IK to move the robot through 3 task-space waypoints, collecting data during the motion and analyzing them afterwards.

LAB PROCEDURE:

Part 1) Calculate the inverse kinematics of the OpenManipulator-X arm

The following 3D geometric diagram represent the robot in a general configuration:

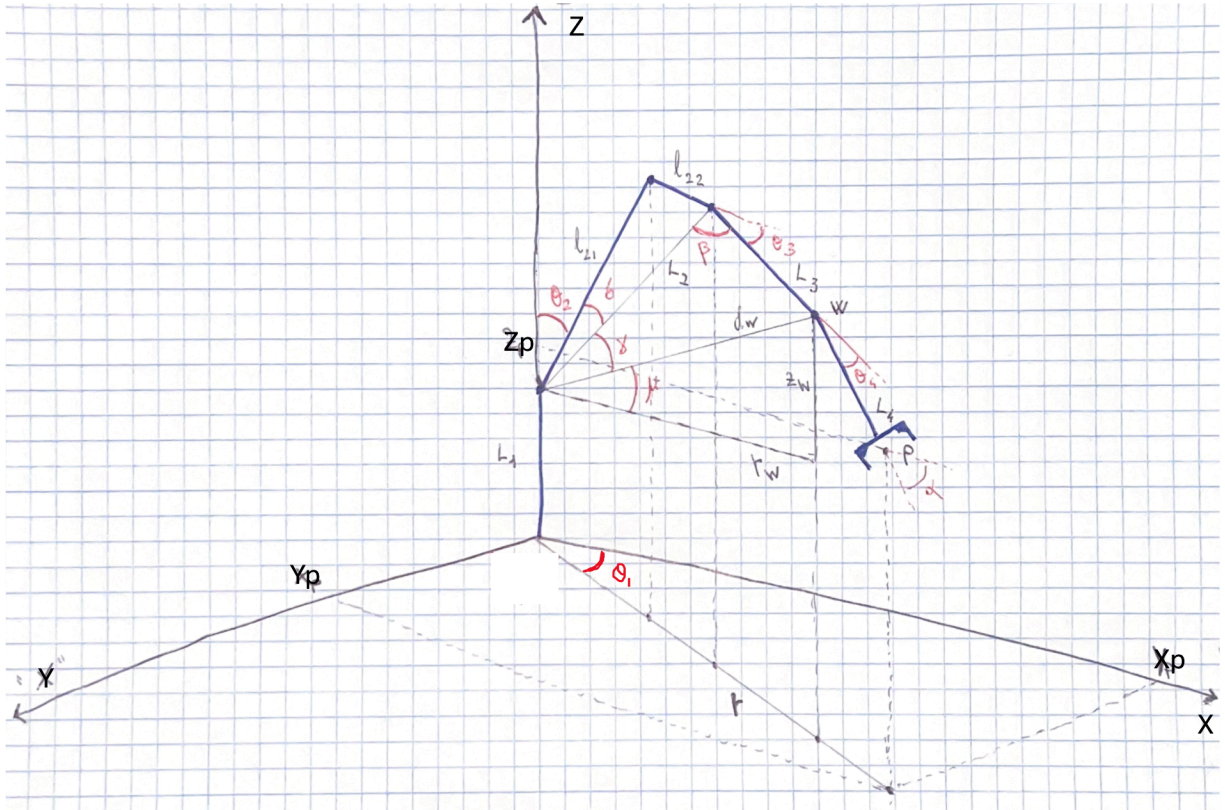


Figure 0: Sketch used for deriving Inverse Kinematics

The Task-space variables X_p , Y_p , Z_p and α can describe completely the position and orientation of the end effector.

Additionally there are some intermediate geometric variables and auxiliary angles used during the derivation of inverse kinematics.

The link lengths are as follows: p

$$L_1 = 77\text{mm}; L_2 = 130\text{mm}; L_3 = 124\text{mm}; L_4 = 126\text{mm}; l_{21} = 128\text{mm}; l_{22} = 24\text{mm}$$

The Geometric approach for IK

The goal is to get equations that explain joint angles in terms of space variables and additional intermediate constants.

The first step is defining some intermediate variables:

$$r = \sqrt{x^2 + y^2}$$

From the top view. End-effector projection onto the ground plane has coordinates **$x := X_p$; $y := Y_p$ in the geometric diagram**. r is radial distance from base axis to that projection.

$$r_w = r - L_4 \cos(\alpha)$$

Wrist position along the radial axis. The end effector extends along the arm by link L_4 oriented with pitch α . Project that last link onto the ground plane. Its radial component equals $L_4 \cos(\alpha)$. Subtracting gives the wrist radial coordinate r_w (the point between L_3 and L_4 that the 2–3 chain must reach).

$$z_w = z - L_1 - L_4 \sin(\alpha)$$

Wrist height: Base adds a vertical offset of L_1 . The end effector's vertical coordinate is z . The vertical component of L_4 is $L_4 \sin \alpha$

$$d_w = \sqrt{r_w^2 + z_w^2}$$

D_w is the distance from shoulder pivot to wrist, In the side view the shoulder-to-wrist vector has horizontal component r_w and vertical z_w . Euclidean length by Pythagoras.

Now i can easily calculate $\mu = \text{atan2}(z_w, r_w)$:

$$\tan(\mu) = \frac{z_w}{r_w}$$

Next part is to calculate the intermediate angles beta and gamma:

$$\cos(\beta) = \frac{L_2^2 + L_3^2 - d_w^2}{2L_2L_3}$$
$$\cos(\gamma) = \frac{d_w^2 + L_2^2 - L_3^2}{2d_wL_2}$$

Here it is used the law of cosines for β , in the triangle formed by links L_2 and L_3 and the line d_w .
Law of cosines: $d_w^2 = L_2^2 + L_3^2 - 2L_2L_3\cos\beta$. Rearranged for $\cos\beta$.

β is the internal angle at the elbow between L_2 and L_3 .

The same logic is followed for gamma.

Then we calculate the angles β and γ using arccos. However, this method does not uniquely define the angle value, since two different configurations, positive and negative, are both possible. This will be helpful afterwards to define elbow-up and elbow-down positions.

Finally the δ angle is easily calculated as it is a constant:

$$\tan(\delta) = \frac{\ell_{22}}{\ell_{21}}$$

Now it is possible to formulate the equations for the joint angles, depending on the previous defined constants and intermediate variables.

Firstly we calculate theta1 as it is uniquely defined:

$$\theta_1 = \tan^{-1} \left(\frac{y}{x} \right)$$

At this point, the results split into two possible configurations: elbow-up and elbow-down. In fact, for the same target position of the end-effector, the arm can reach it with the elbow pointing upward or downward. Both satisfy the kinematic equations as seen before but we choose different values of the intermediate angles; they represent two distinct physical postures of the manipulator.

Elbow up:

$$\theta_2 = \frac{\pi}{2} - \delta - \gamma - \mu$$
$$\theta_3 = \frac{\pi}{2} + \delta - \beta$$

Elbow down:

$$\theta_2 = \frac{\pi}{2} - \delta + \gamma - \mu$$
$$\theta_3 = \frac{\pi}{2} + \delta + \beta$$

The solutions are two because β and γ are not independent — they're linked by the wrist position and the triangle formed by L_2 , L_3 , and d_w . Changing the sign of one forces the other to change accordingly to keep the same end-effector position, leaving only two valid configurations: elbow-up and elbow-down.

Finally, θ_4 is calculated based on θ_2 and θ_3 , and therefore depends on whether the configuration is elbow-up or elbow-down.

$$\theta_4 = -\alpha - \theta_2 - \theta_3$$

This geometric method allowed us to derive the joint angles through a few simple steps. It is important to note that two different solutions correspond to the same end-effector position: **elbow-up** and **elbow-down**.

Elbow-up:

$$\theta_1 = \tan^{-1} \left(\frac{y}{x} \right)$$
$$\theta_2 = \frac{\pi}{2} - \delta - \gamma - \mu$$
$$\theta_3 = \frac{\pi}{2} + \delta - \beta$$
$$\theta_4 = -\alpha - \theta_2 - \theta_3$$

Elbow-down:

$$\theta_1 = \tan^{-1} \left(\frac{y}{x} \right)$$
$$\theta_2 = \frac{\pi}{2} - \delta + \gamma - \mu$$
$$\theta_3 = \frac{\pi}{2} + \delta + \beta$$
$$\theta_4 = -\alpha - \theta_2 - \theta_3$$

Procedure 2: Implement the Inverse Kinematics in Python

After calculating and developing the inverse kinematics equations during our prelab. We implemented our equations on python for faster and real time logging of our joint angles and positions for the Robot arm. In order to develop our script, we defined a function `get_ik()` on python that simply used the Robot dimensions and joint angles to figure out the inverse kinematic equations. To verify acceptable functionality of our script, we ran a quick sanity test. Using our newly developed `get_ik()` function, we tested different positions to obtain the joint angles and then used our previously defined `get_Fk()` function to find the initial positions we used for `get_ik()`. Table 1 shows our obtained results, from the table we can see that the `get_ik()` function, gives us joint angles that when used as inputs for our FK function, return the initial position results. Although there is some difference, it is very minimal to the point of it being negligible. This procedure, verifies that our functions are working correctly.

Test Case	X,Y,Z(mm), α (deg)	IK SOLUTION (Theta1,2,3,4)	FK RESULTS FROM IK
1	274,0,204,0	0,0,0,0	274.042, 0, 203.992, 0
2	16,4,336,15	14.036, -45, -58.12, 89.0243	16.034, 4.009, 336.025, 15
3	0,-270,106,0	-90, 15.213, 30.001, -45.214	0, -270.039, 105.981, 0

Table 1: Sanity check results

Procedure 3: Interface and validate your IK solution with the robot.

After verifying that our functions were working correctly, we started to implement these functions to monitor the angles and position of the robot in real time. We developed a script (`lab3_3.py`) that would move the robot through the waypoints shown in table 2. This would ultimately form a triangular path if implemented correctly. While the robot was moving through the waypoints, we gathered joint-angle data, end-effector position as well as a timestamp for each recording that was being made. The data logged was then stored in a numpy array to later save as a pickle file. Once our pickle file was saved we used python to develop certain plots for further analysis.

Waypoint	x (mm)	y (mm)	z (mm)	α (deg)
1	25	-100	150	-60
2	150	80	300	0
3	250	-115	75	-45
4	25	-100	150	-60

Table 2: Task space waypoints used (Triangular path)

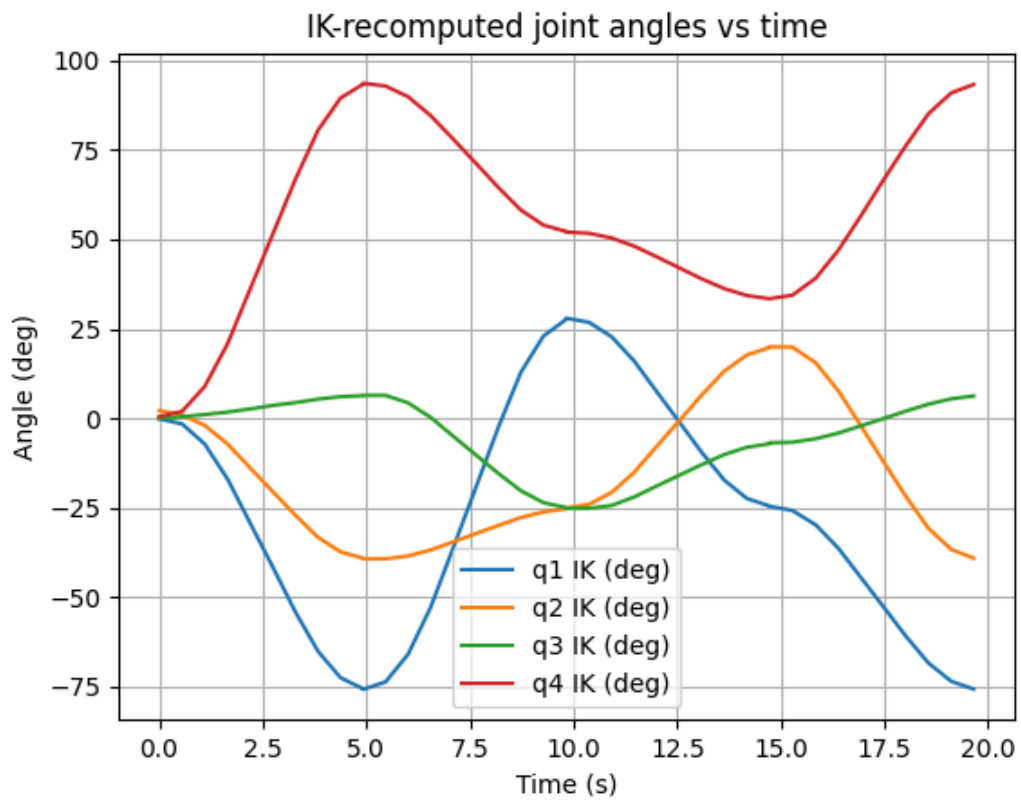


Figure 1: Inverse Kinematics computed joint angles vs time

When developing the joint angles vs time graphs with both the use of our inverse kinematics equation and our measured joint angles from the `get_joint_readings()` function.

We compared both graphs shown in Figure 1 and 2, we can clearly see that there is not much difference between both graphs.

This aligns with our previously done sanity check for our `get_ik()` function. Only this time, the values were directly read from the robot arm and not previously selected.

This implies that our functions work in a very precise manner and ensure minimal difference between the type of capturing done for the joint angle measurements.

As we can see, each of the angles falls within the reachable area of each joints, showing proper functionality of the robot and movement.

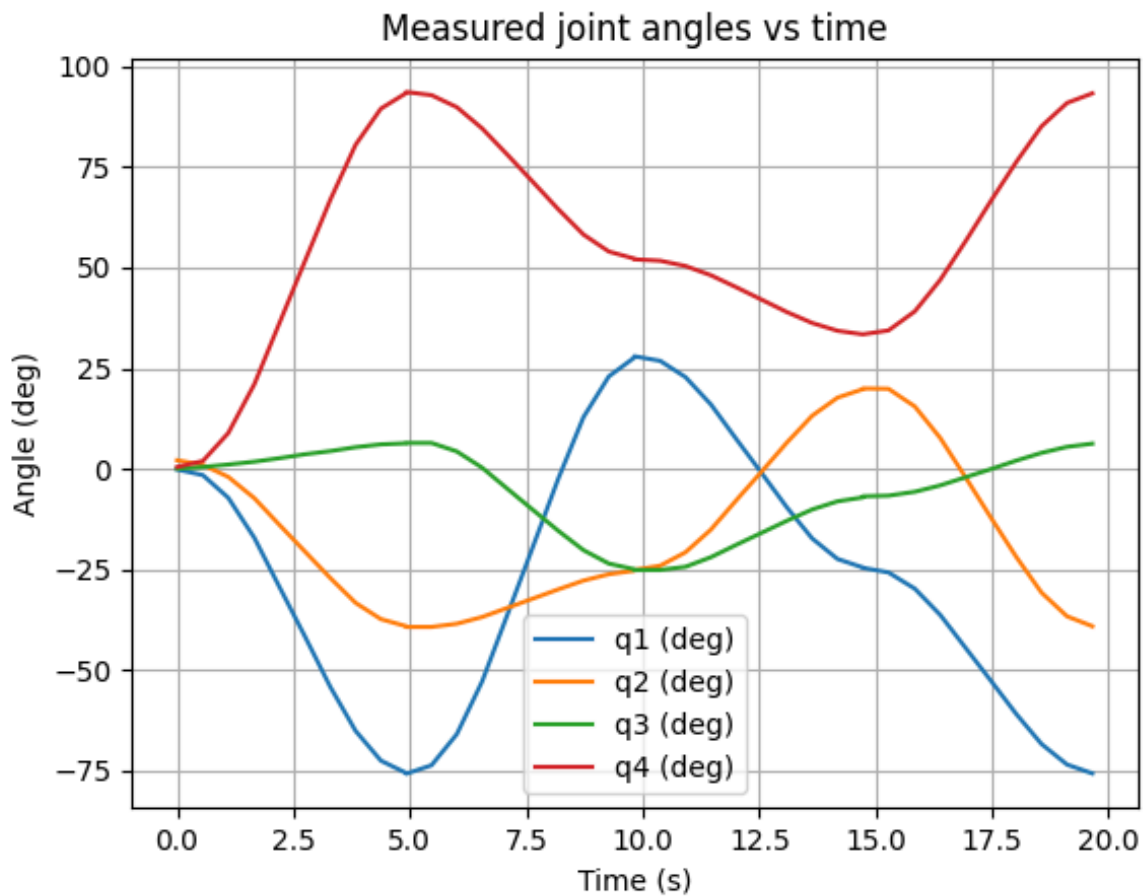


Figure 2: Measured Joint angles vs Time

3D End-effector trajectory

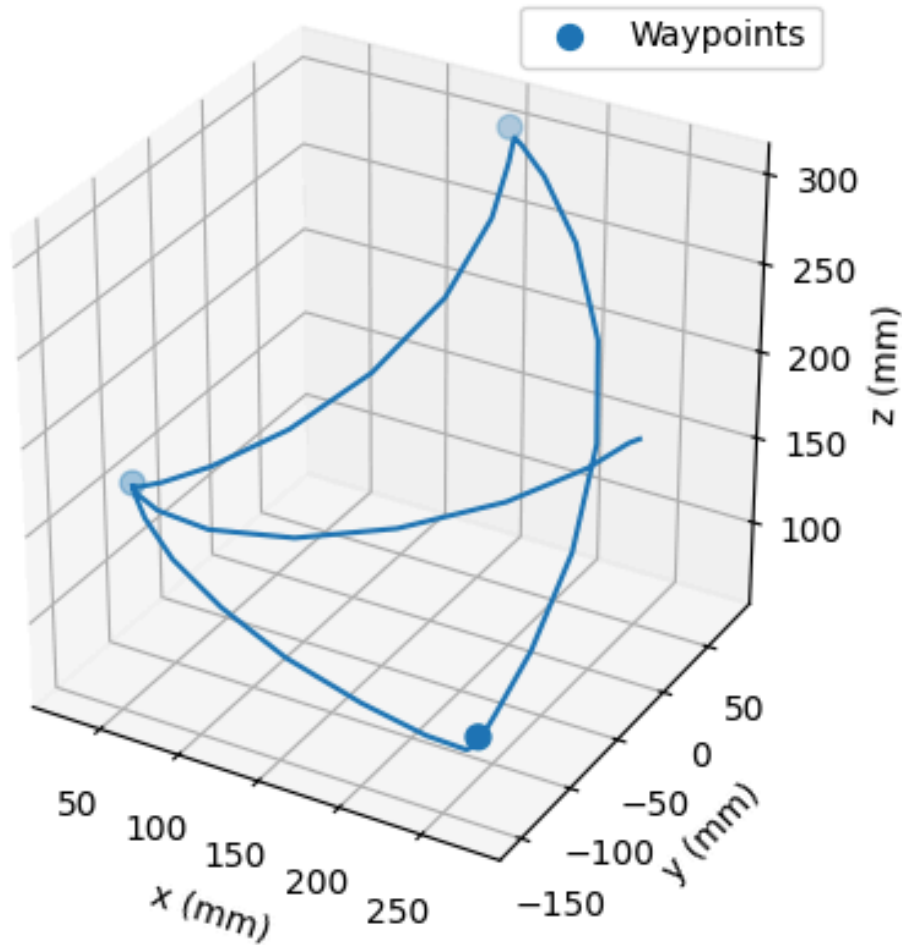


Figure 3: 3D end effector trajectory

Figure 3 shows our end effector trajectory position for the robot arm. This aligns with the expected trajectory we assigned. It also aligns with the visual movement we saw during the trajectory sequence we assigned the robot.

As we can see in the figure, the robot follows a triangle-like path to then return to its initial position that it started the trajectory at. The lines are pretty smooth which suggests minimal distortion or shaking of the robot arm. This was also verified visually when setting the robot up.

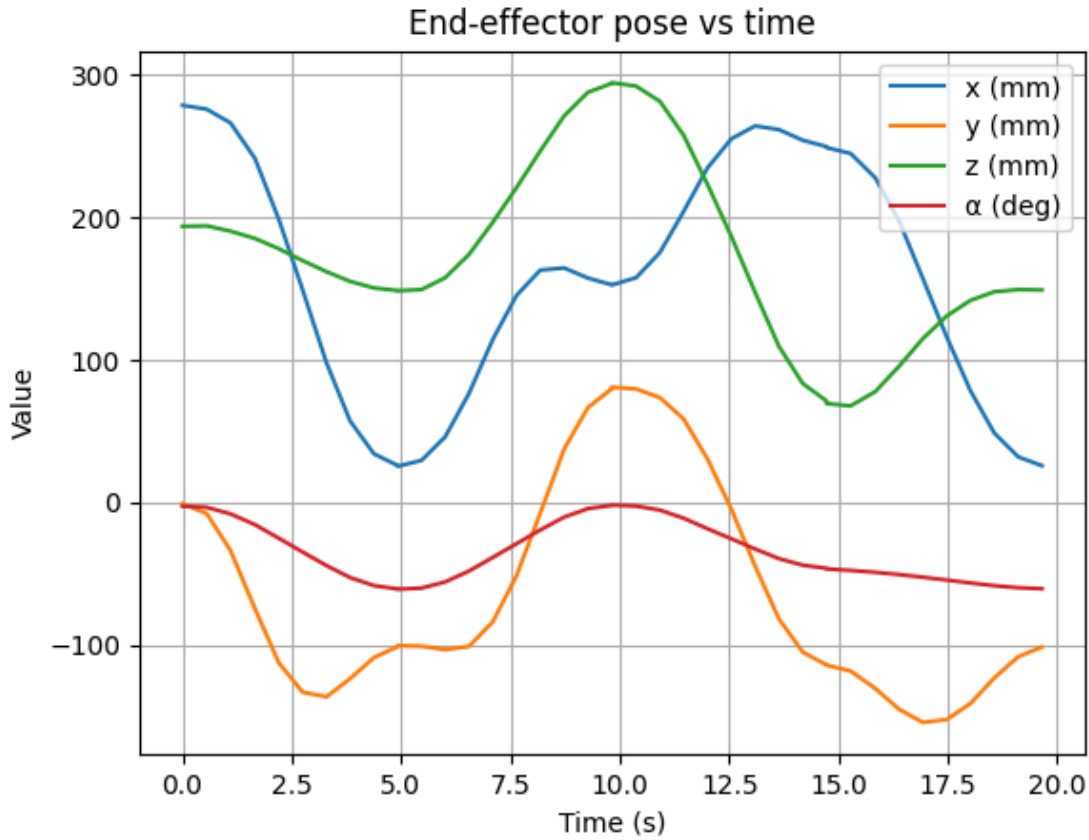


Figure 4: End Effector pose vs time

Figure 4 illustrates the variation of the end-effector pose over time as the OpenManipulator-X moved through the defined task-space waypoints. The plots show the translational components (x , y , z) in millimeters and the orientation component (α) in degrees. From the figure, it can be observed that each component follows a smooth, continuous trajectory, consistent with the robot's commanded motion profile. The x and z components exhibit periodic oscillations corresponding to the forward and vertical motion of the end-effector, while the y component demonstrates complementary variation as the manipulator transitions between the triangular waypoints.

The orientation angle α changes gradually throughout the motion, indicating consistent and stable orientation control. Overall, the smoothness of the curves suggests accurate inverse kinematic computation and coordinated multi-joint motion with minimal vibration or discontinuities during execution.

CONCLUSION:

In this lab, we successfully derived and implemented the inverse kinematics for the OpenManipulator-X and validated its performance through both simulation and experimental testing. The results from our `get_ik()` and `get_fk()` functions showed strong agreement, confirming the accuracy of our kinematic model.

When applied to real robot motion, the end-effector followed the expected triangular path with smooth, coordinated movement and minimal error between computed and measured data. Overall, the experiment demonstrated a reliable IK implementation and deepened our understanding of task-space control for multi-DOF robotic manipulators.