Lab Procedure for Simulink
Inverse Kinematics

Setup

1. It is recommended that you review Lab 2 – Application Guide before starting this lab.
2. Hardware Preparation:
    a. Ensure that the QArm Mini is securely attached to the base.
    b. Verify that the manipulator is in the rest position.
    c. Confirm that the QArm Mini is connected to the PC and turn it ON (the light in the switch should be red).
    d. Check and update the latency setting as shown in Figure 1:
        i. Navigate to Device Manager > Ports
        ii. Select the appropriate device - USB Serial Port (COMx) Make a note of the COM port Number.
        iii. Go to Port Settings > Advanced > Latency
        iv. Set the latency to 2 ms.

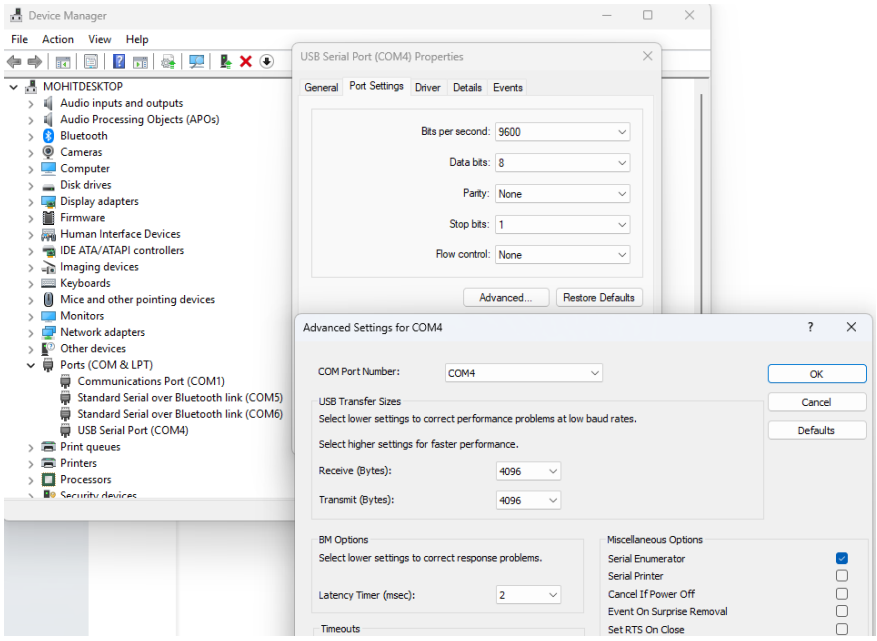

Figure 1. Latency Settings

Inverse Kinematics

1. Open teach_learn.slx
    a. In the Hardware tab, open **Hardware Settings -> Solver** and verify:
        i. Solver Type: Fixed
        ii. Solver: ODE2
        iii. Fixed Step Size: 1/30
    b. From the root level's **Application Layer**, navigate to the **Interface Layer** by opening the QArm Mini subsystem, and then the **Hardware Layer** by opening the Hardware Layer subsytem, and double-click on the **HIL Initialize** block.
    c. Update the **Board Identifier** value to match the COM port you noted during setup.
2. This section of the lab explores inverse kinematics and let you control the QArm Mini by providing coordinates in the cartesian plane.
3. The multiport switch in the model, takes in x, y, z positions (in meters, relative to the base) and γ (gamma), the end-effector angle, as inputs.
4. Ensure that the multiport switch selector constant is set to 1. Build and deploy the model using the **Monitor and Tune**  button on the Hardware or QUARC tab. Once started, the model moves the manipulator to Home Position.
5. Keeping in mind, the joint limits identified in the Workspace Identification lab, modify the delta values (in meters) and observe how the manipulator responds. For example, setting the delta values [0.05, 0, 0, 0] should move the arm 5 cm in the positive X direction. Verify that the arm correctly follows the expected movement.
6. Reset the delta values to zero to return the manipulator to its home position. Next, choose a point within the valid workspace—optionally, place a reference object at that point. Using a ruler or tape measure, measure the distance from the front of the gripper (home position) to the selected point in the x, y, and z directions (in meters). Estimate the desired end-effector angle. Enter these values into the delta constants and verify whether the manipulator accurately moves to the intended position.
7. Open the QArm Mini subsystem and to view the **Interface Layer,** check the outputs generated qArmMiniInverseKinematics function on the top left. This function computes possible joint angles and picks the optimal solution (thetaOpt), it then uses this as the input command for the QArm Mini.
8. Return to the **Application Layer**, change the delta values to a position outside the QArm Mini workspace. For example, [0.5, 0.5, 0.5, 0], which represents a **50 cm offset in x, y, and z**. What happens to the output theta values in the Interface Layer? How does the manipulator behave?

9. Stop the model, close it. Gently bring the manipulator back to rest position before shutting it down.

Teach Pendant (Learn)

1. Reset the delta values to zero to return the manipulator to its home position.
2. This model lets you to "teach" the QArm Mini a series of waypoints that simulate a robotic assembly process, emulating the learning mode of a real teach pendant.

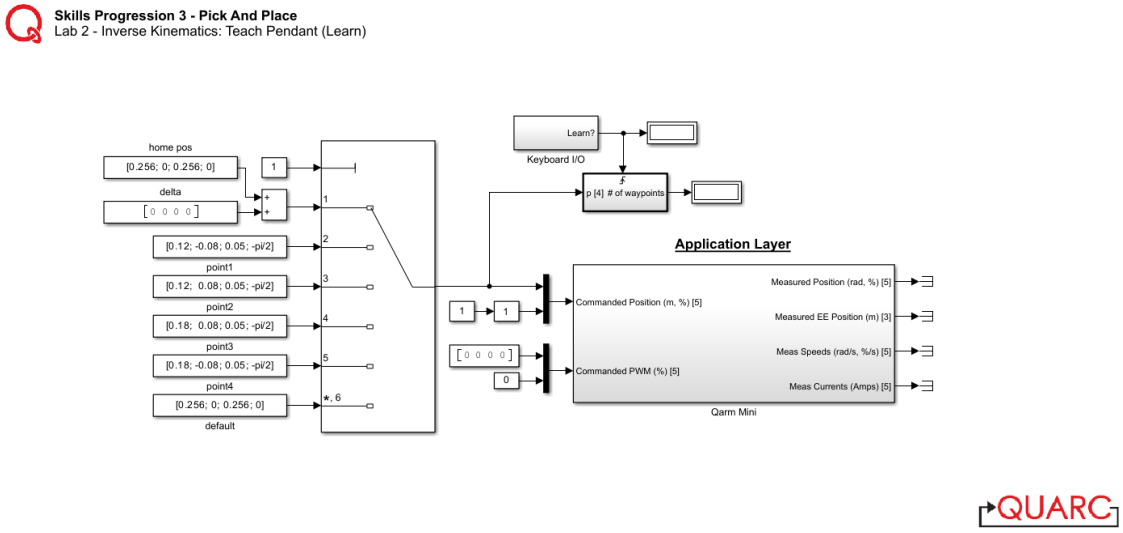

Figure 2. Teach Pendant (Learn)

3. The multiport switch in Figure 2, takes in x, y, z positions (in meters, relative to the base) and γ (gamma), the end-effector angle, as inputs.
4. There are five predefined waypoints that form the edges of the rectangle-the path the manipulator will follow:
    a. Home Position: [0.256; 0; 0.256; 0]
    b. Point 1: [0.12; -0.08; 0.05; -pi/2]
    c. Point 2: [0.12; 0.08; 0.05; -pi/2]
    d. Point 3: [0.18; 0.08; 0.05; -pi/2]
    e. Point 4: [0.18; -0.08; 0.05; -pi/2]
5. Before running the model, set the Position Select constant to 1.
6. Build and deploy the model using the **Monitor and Tune** button. Once started, the model commands the manipulator to go to Home Position.
7. **Recording the Waypoints:**

a. Validate that the end-effector reaches the expected positions.

b. Upon reaching a desired point, press the Spacebar to add the waypoint to the queue.

   Note: The Keyboard I/O and waypoints subsystem records waypoints when you press the **Spacebar**.

c. The waypoint display counter should increment upon successfully adding a point.

d. Update the Position Select to the next point in the predefined waypoints.

e. Click on the QArm Mini subsystem to open the **Interface Layer.**

f. Check the indices and numSolutions displays from the qArmMiniInverseKinematics function to verify the validity of the entered positions.

g. Go back to the root level of the model.

h. Repeat steps a – g until you have gone through all the points.

Note: If you accidentally add a waypoint or need to restart, stop and rerun the model to clear the queue. You can modify or add additional points to create a custom trajectory. If the entered position is invalid, the manipulator will hold its last valid position.

8. Once you add all waypoints to the queue, stop the model and close it. Bring the manipulator back to rest position.

Teach Pendant (Follow)

1. Open teach_follow.slx.

   a. In the Hardware tab, open **Hardware Settings -> Solver** and verify:

      i. Solver Type: Fixed

      ii. Solver: ODE2

      iii. Fixed Step Size: 1/30

   b. Navigate to the **HIL Initialize** block like step 1b in Teach Pendant (Learn).

   c. Update the **Board identifier** to match the correct COM port you noted during setup.

2. You will use this model to read the waypoints that were recorded in the Teach Pendant (Learn) section and recreate a robotic assembly process automatically.

3. Set the duration between waypoints by changing the input constant to 3 seconds. When the model is executed, the Playback subsystem will output a new waypoint every 3 seconds.

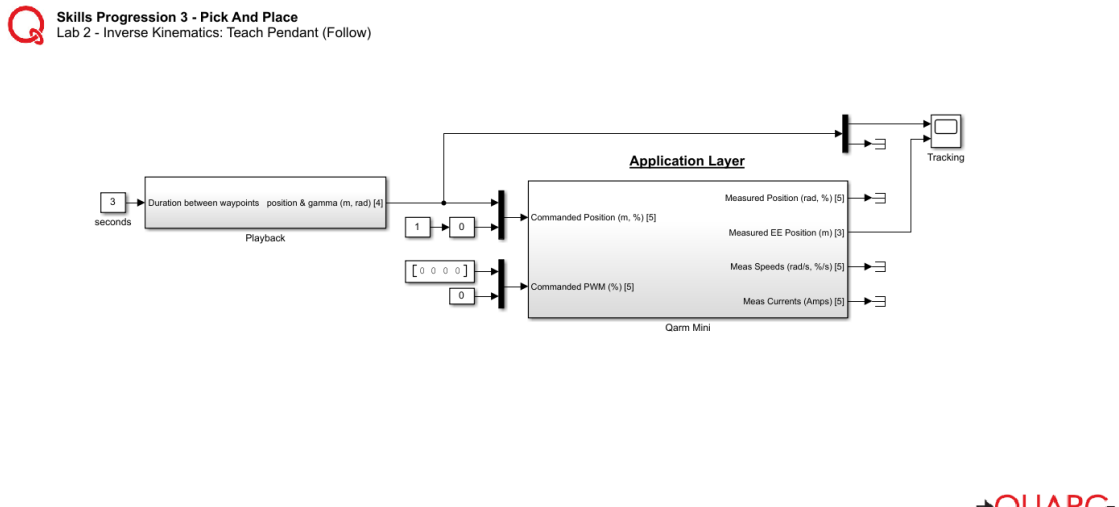

Figure 3. Teach Pendant (Follow)

4. Go back to the root level of your model (**Application Layer**). Build and deploy the model using the **Monitor and Tune** button. Once started, the model commands the first waypoint position to the manipulator.

5. Use the **Tracking scope** to monitor the desired and actual end-effector positions as the manipulator cycles between the recorded waypoints shown in Figure 4.

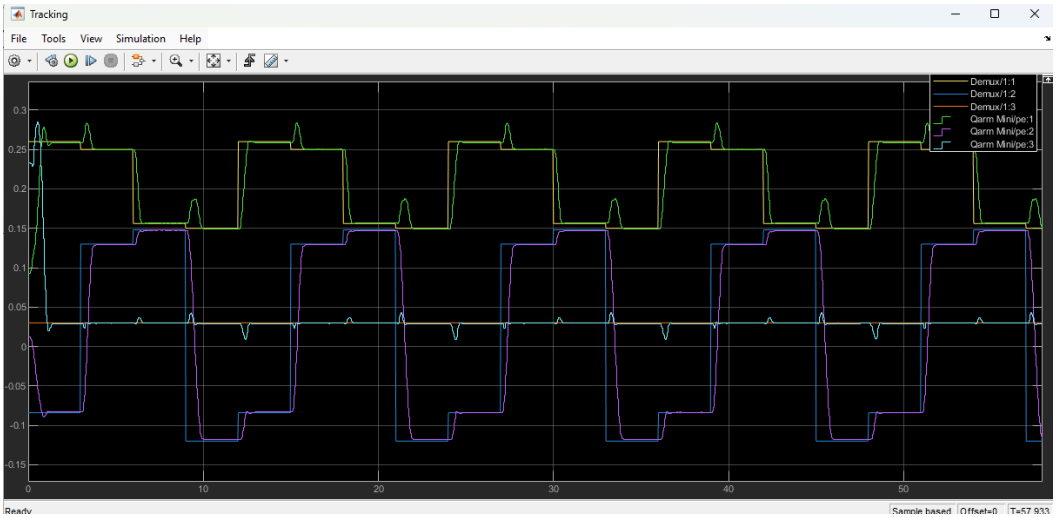

Figure 4. Desired vs Actual End Effector Positions

6. Stop the model, upon stopping you will see a 3D plot pop up highlighting the exact path the end effector traversed. Does the graph represent a rectangular trajectory? Record your observations.

7. Close the graph, turn off the arm and bring it back to rest position

Trajectory Generation

1. Open trajectory_gen.slx.
    a. In the Hardware tab, open **Hardware Settings -> Solver** and verify:
        i. Solver Type: Fixed
        ii. Solver: ODE2
        iii. Fixed Step Size: 1/30
    b. Find the **HIL Initialize** block in the Hardware Layer similar to step 1b in the first three sections.
    c. Update the **Board Identifier** to match the correct COM port you noted during setup.
2. Go back to the root level of your model (**Application Layer**). This model follows a rectangular path using predefined waypoints, like the Teach Pendant method but with automated trajectory generation.
3. Verify the waypoint definitions in Model Properties:
    a. Right-click on a blank space in your model and open Model Properties.
    b. Navigate to Callbacks > InitFcn, and confirm the waypoint values:
        i. home = [0.256; 0; 0.256; 0]
        ii. A = [0.12; -0.08; 0.05; -pi/2]
        iii. B = [0.12; 0.08; 0.05; -pi/2]
        iv. C = [0.18; 0.08; 0.05; -pi/2]
        v. D = [0.18; -0.08; 0.05; -pi/2]
4. The waypointNavigator MATLAB function in the model manages movement between waypoints by updating the initial (Pi) and final (Pf) positions. It cycles through the waypoints based on a timer and outputs the current waypoint number and time vector for trajectory generation. Motion only progresses if the startStop flag is active.
5. The cubicSpline script generates smooth trajectories between waypoints using a cubic polynomial. It returns a 3×4 matrix of coefficients for the x, y, z axes, ensuring continuous motion with zero initial and final velocities.
6. Build and deploy the model using the **Monitor and Tune**  button. Once started set the startStop switch to 1.

7. Observe whether the end effector moves as the same way as it did in the Teach Pendant experiment. Note any differences in smoothness and accuracy when moving between waypoints (e.g., A to B, B to C).

8. Use the **Tracking EE Position** scope to monitor the Commanded vs Measured end-effector positions shown in Figure 5.

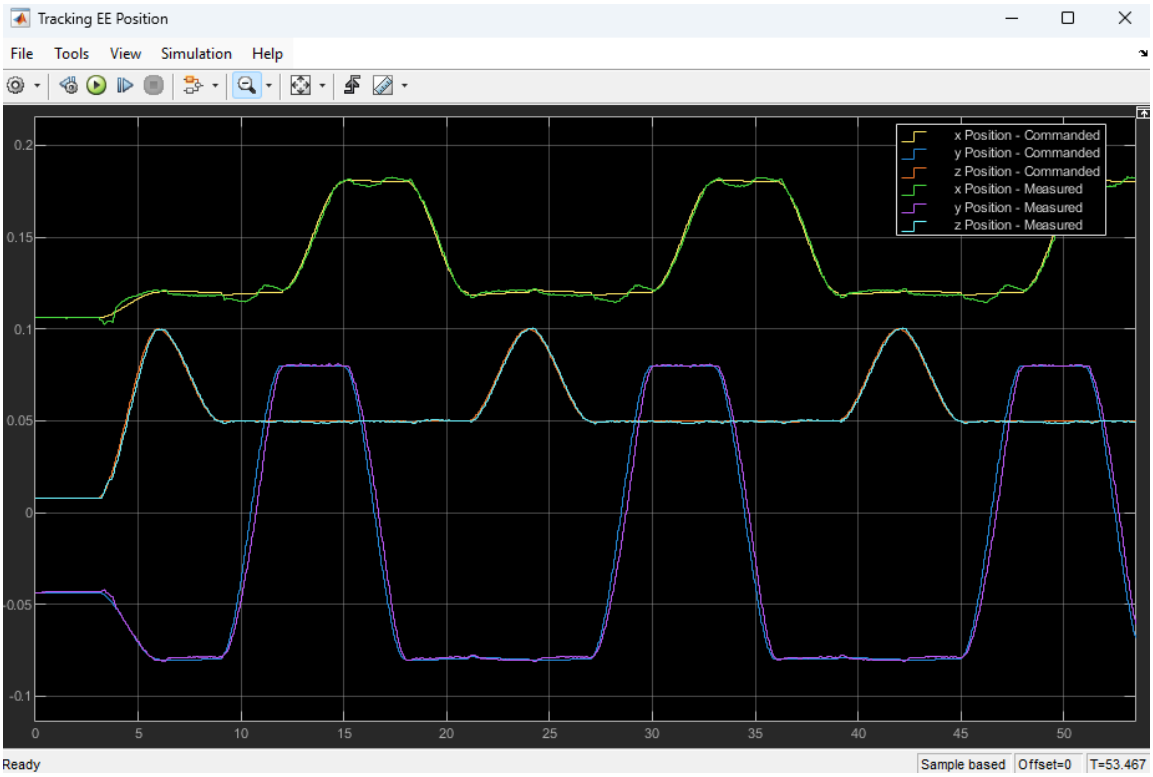

Figure 5. Commanded vs Measured EE Positions

9. Repeat the experiment but try increasing the duration parameter from 3s to 6s. how does this affect the motion between waypoints? Take screenshots of the Tracking EE Position scope.

10. Repeat the experiment but try decreasing the duration parameter from 3s to 1s. how does this affect the motion between waypoints? Take screenshots of the Tracking EE Position scope.

11. Stop the model— a 3D plot will display the end effector's trajectory. Compare this generated path with the recorded trajectory from the **Teach Pendant (Follow)**. Do you see any differences? Document your observations.

12. Close the graph, turn off the arm and bring it back to rest position.

13. Save and close your model.