Group Assignment – 1

Group members

1. Manav Mepani
2. Siddharth Shrotri
3. Prasham Soni

A Report on Forward and Inverse Kinematics Implementation for OpenManipulator-X

---

1. Forward Kinematics Implementation

Objective

The goal of this task was to implement a forward kinematics (FK) node for the OpenManipulator-X robot arm. The node:

- Subscribes to the /joint_states topic to receive joint positions in real-time.
- Calculates the end-effector pose (position and orientation) using forward kinematics equations.
- Publishes the computed pose to a dedicated ROS topic /end_effector_pose.

Node Implementation

- **Subscription**: The FK node subscribes to the `/joint_states` topic to get the current joint angles.
- **Calculation**: The node uses the Denavit-Hartenberg (DH) parameters provided for the robot to calculate the end-effector pose based on the joint values.
- **Publishing**: The pose is published on the topic `/end_effector_pose`, so it can be echoed or used by other nodes.

#Key details of the FK node:
- Subscribes to: `/joint_states`
- Publishes to: `/end_effector_pose`
- Uses standard DH conventions for calculation.

Testing Procedure

1. I moved the robot to three different positions by sending joint position references using the following ROS 2 command:

```
ros2 service call /goal_joint_space_path
open_manipulator_msgs/srv/SetJointPosition "{...}"
```

2. After each movement, I echoed the /end_effector_pose topic to print the resulting pose:

```
ros2 topic echo /end_effector_pose
```

3. Results were displayed :
    - The robot's simulation in Rviz for each position.
    - The terminal displaying the pose from the `/end_effector_pose` topic.

Code snippet of Forward Kinematics

```
    def forward_callback(self, msg):
        #Lengths of the links
        l1 = self.L1
        l2 = self.L2
        l3 = self.L3
        l4 = self.L4
        l5 = self.L5

        #The Join angles
        q1 = msg.position[0]
        q2 = -msg.position[1]
        q3 = -msg.position[2]
        q4 = -msg.position[3]
        self.get_logger().info(f'Angles received in radians are {q1}, {q2}, {q3}, {q4} and the link lengths are {l1}, {l2}, {l3}, {l4}, {l5}')

        #the constant angle at start
        A1 = math.radians(79.38)

        #Homogeneous transformation matrices

        #--------------------------------H_0_1-------------------------------------
        H_0_1 = [[1, 0, 0, 0],
                 [0, 1, 0, 0],
                 [0, 0, 1, l1],
                 [0, 0, 0, 1]]

        H_0_1 = np.array(H_0_1)
        self.get_logger().info(f'H_0_1 = \n{H_0_1}')

        self.R_0_1 = H_0_1[0:3, 0:3]    #Extracting rotational matrix
        self.O_1 = H_0_1[0:3, 3]        #Extracing Position vector

        #--------------------------------H_0_2-------------------------------------
        H_1_2 = [[math.cos(q1), 0, math.sin(q1), 0],
                 [math.sin(q1), 0, -math.cos(q1), 0],
                 [0, 1, 0, l2],
                 [0, 0, 0, 1]]
        H_1_2 = np.array(H_1_2)
        H_0_2 = np.dot(H_0_1, H_1_2)
        self.get_logger().info(f'H_0_2 = \n{H_0_2}')

        self.R_0_2 = H_0_2[0:3, 0:3]    #Extracting rotational matrix
        self.O_2 = H_0_2[0:3, 3]        #Extracing Position vector

        #--------------------------------H_0_3-------------------------------------
        H_2_3 = [[math.cos(q2 + A1), -math.sin(q2 + A1), 0, l3 * math.cos(q2 + A1)],
                 [math.sin(q2 + A1), math.cos(q2 + A1), 0, l3 * math.sin(q2 + A1)],
                 [0, 0, 1, 0],
                 [0, 0, 0, 1]]

        H_2_3 = np.array(H_2_3)
        H_0_3 = np.dot(H_0_2, H_2_3)
        self.get_logger().info(f'H_0_3 = \n{H_0_3}')

        self.R_0_3 = H_0_3[0:3, 0:3]    #Extracting rotational matrix
        self.O_3 = H_0_3[0:3, 3]        #Extracing Position vector

        #--------------------------------H_0_4-------------------------------------
        H_3_4 = [[math.cos(q3 - A1), -math.sin(q3 - A1), 0, l4 * math.cos(q3 - A1)],
                 [math.sin(q3 - A1), math.cos(q3 - A1), 0, l4 * math.sin(q3 - A1)],
                 [0, 0, 1, 0],
                 [0, 0, 0, 1]]

        H_3_4 = np.array(H_3_4)
        H_0_4 = np.dot(H_0_3, H_3_4)
        self.get_logger().info(f'H_0_4 = \n{H_0_4}')

        self.R_0_4 = H_0_4[0:3, 0:3]    #Extracting rotational matrix
        self.O_4 = H_0_4[0:3, 3]        #Extracing Position vector

        #--------------------------------H_0_5-------------------------------------
        H_4_5 = [[math.cos(q4), -math.sin(q4), 0, l5 * math.cos(q4)],
                 [math.sin(q4), math.cos(q4), 0, l5 * math.sin(q4)],
                 [0, 0, 1, 0],
                 [0, 0, 0, 1]]

        H_4_5 = np.array(H_4_5)
        H_0_5 = np.dot(H_0_4, H_4_5)

        self.R_0_5 = H_0_5[0:3, 0:3]    #Extracting rotational matrix
        self.O_5 = H_0_5[0:3, 3]        #Extracing Position vector


        for i in range(0, H_0_5.shape[0]):
            for j in range(0, H_0_5.shape[1]):
                H_0_5[i][j] = 0 if abs(H_0_5[i][j]) < 10e-2 else H_0_5[i][j]

        self.get_logger().info(f'The end effector pose is \n{H_0_5}')
        self.get_logger().info(f'The end effector pose is \n{self.R_0_5}')


        # Convert rotation matrix to quaternion
        rotation = R.from_matrix(self.R_0_5)
        quaternion = rotation.as_quat()  # Returns [qx, qy, qz, qw]

        self.get_logger().info(f'The end effector pose is Position = {self.O_5}, Quaternions = {quaternion}')
```

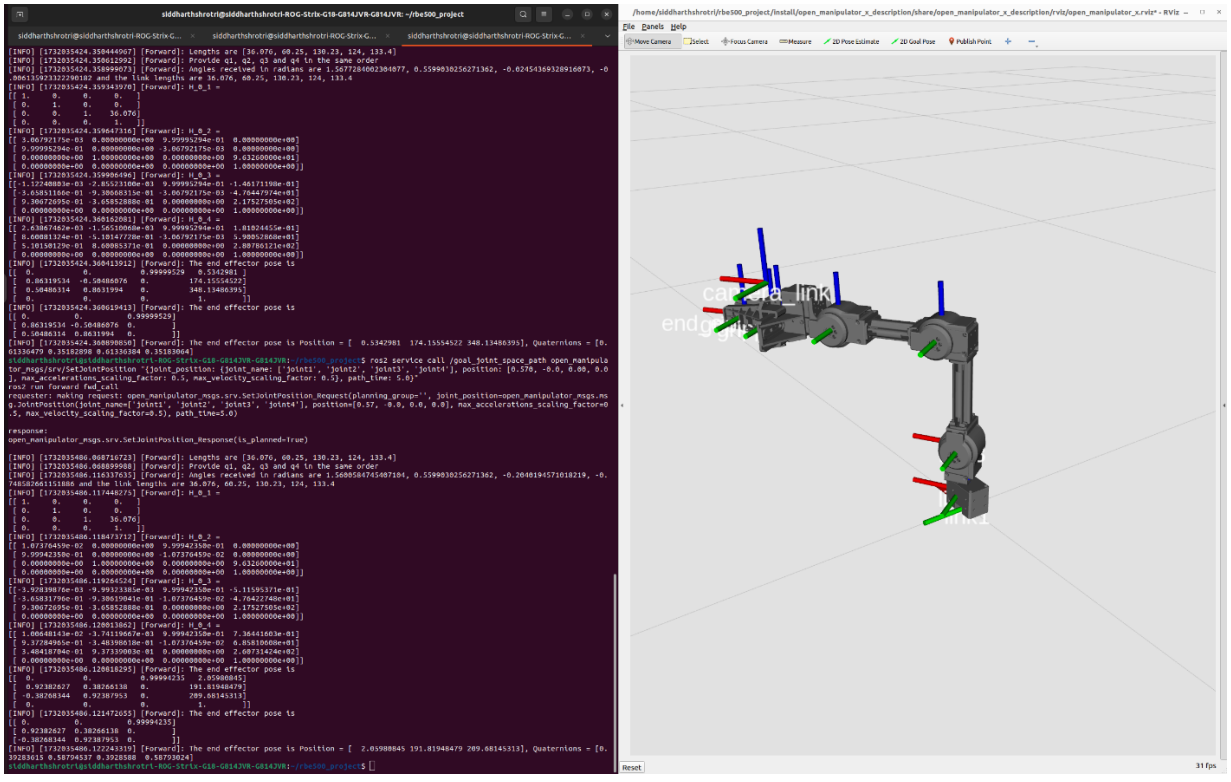
Results

- **Position 1**:

```
ros2 service call /goal_joint_space_path
open_manipulator_msgs/srv/SetJointPosition "{joint_position: {joint_name:
['joint1', 'joint2', 'joint3', 'joint4'], position: [0.570, -0.0, 0.00,
0.0], max_accelerations_scaling_factor: 0.5, max_velocity_scaling_factor:
0.5}, path_time: 5.0}"

ros2 run forward fwd_call
```

- **Robot Screenshot**:

- **Pose from Terminal**:

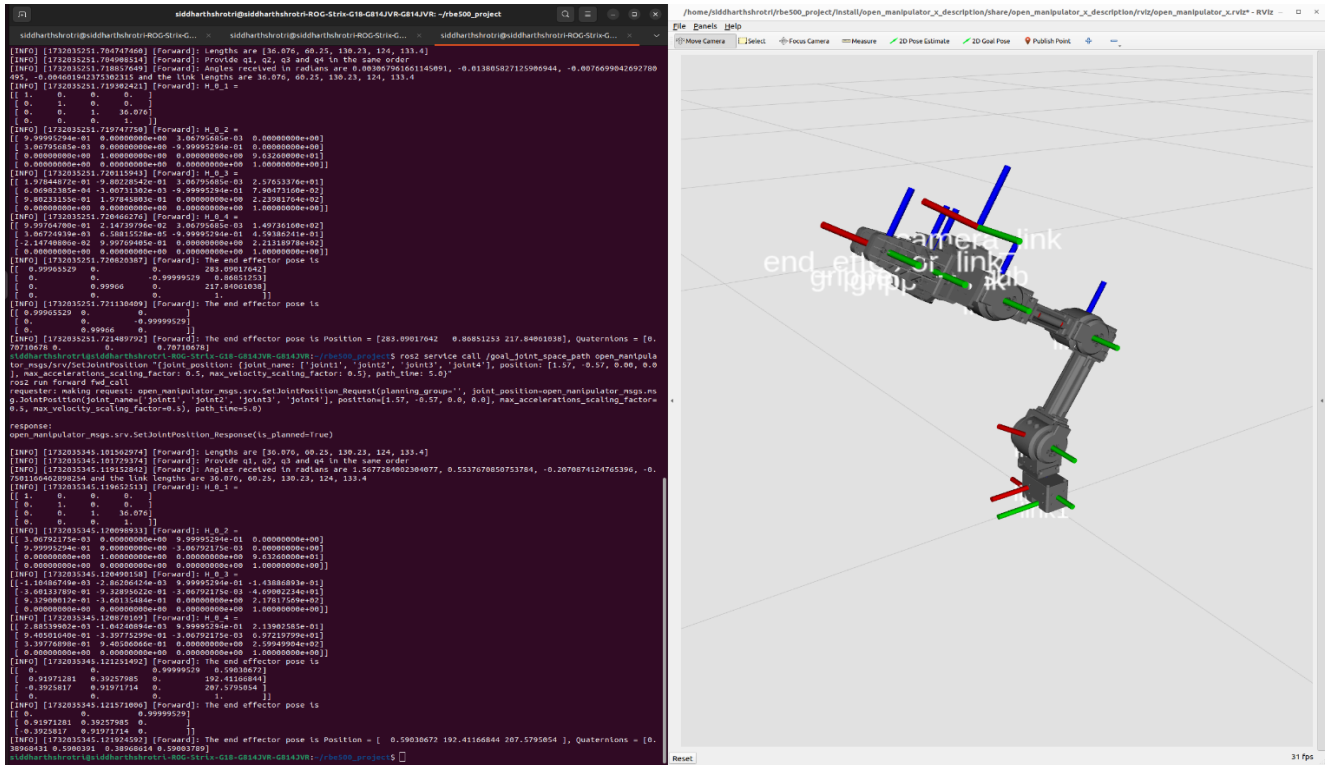


- **Position 2**:

```
ros2 service call /goal_joint_space_path
open_manipulator_msgs/srv/SetJointPosition "{joint_position: {joint_name:
['joint1', 'joint2', 'joint3', 'joint4'], position: [1.57, -0.57, 0.00,
0.0], max_accelerations_scaling_factor: 0.5, max_velocity_scaling_factor:
0.5}, path_time: 5.0}"

ros2 run forward fwd_call
```

- **Robot Screenshot**:

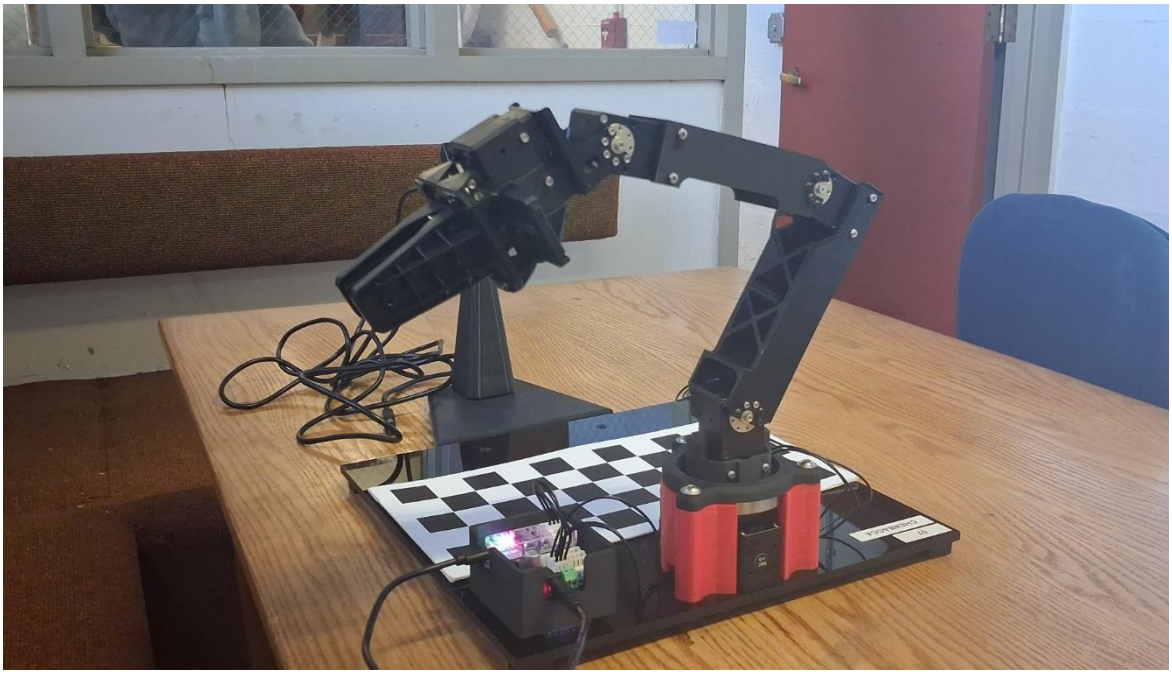
- **Pose from Terminal**:



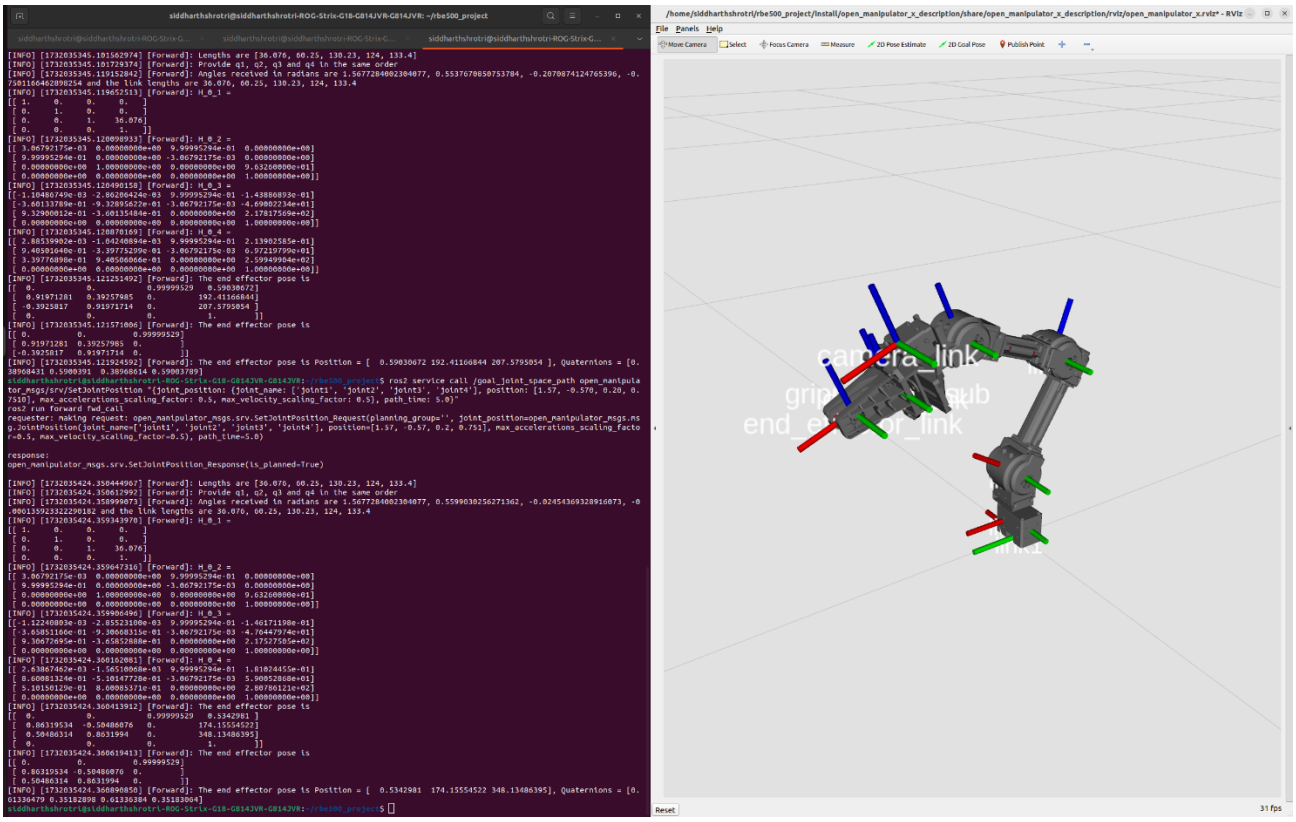
- **Position 3:**

```
ros2 service call /goal_joint_space_path
open_manipulator_msgs/srv/SetJointPosition "{joint_position: {joint_name:
['joint1', 'joint2', 'joint3', 'joint4'], position: [1.57, -0.570, 0.20,
0.7510], max_accelerations_scaling_factor: 0.5, max_velocity_scaling_factor:
0.5}, path_time: 5.0}"
ros2 run forward fwd_call
```

  - **Robot Screenshot:**



  - **Pose from Terminal:**



2. Inverse Kinematics and Bonus Question Implementation

Objective

The goal of this task was to implement an inverse kinematics (IK) node for the robot arm. The node:

- Implements a ROS 2 service that calculates joint positions for a given end-effector pose.
- Uses a service client to request joint angles for specific poses.
- Tests the implementation by moving the robot to three different poses.

Node Implementation

- **Service Server**: The IK node is implemented as a service server called `/calculate_ik`. It takes the desired end-effector pose (position and orientation) as input and calculates the corresponding joint angles.
- **Service Client**: A ROS 2 client sends the desired pose to the server using the ros2 service call command.
- **Testing**: The node's output (joint positions) is used to move the robot to the target pose, and the results are verified visually and numerically.

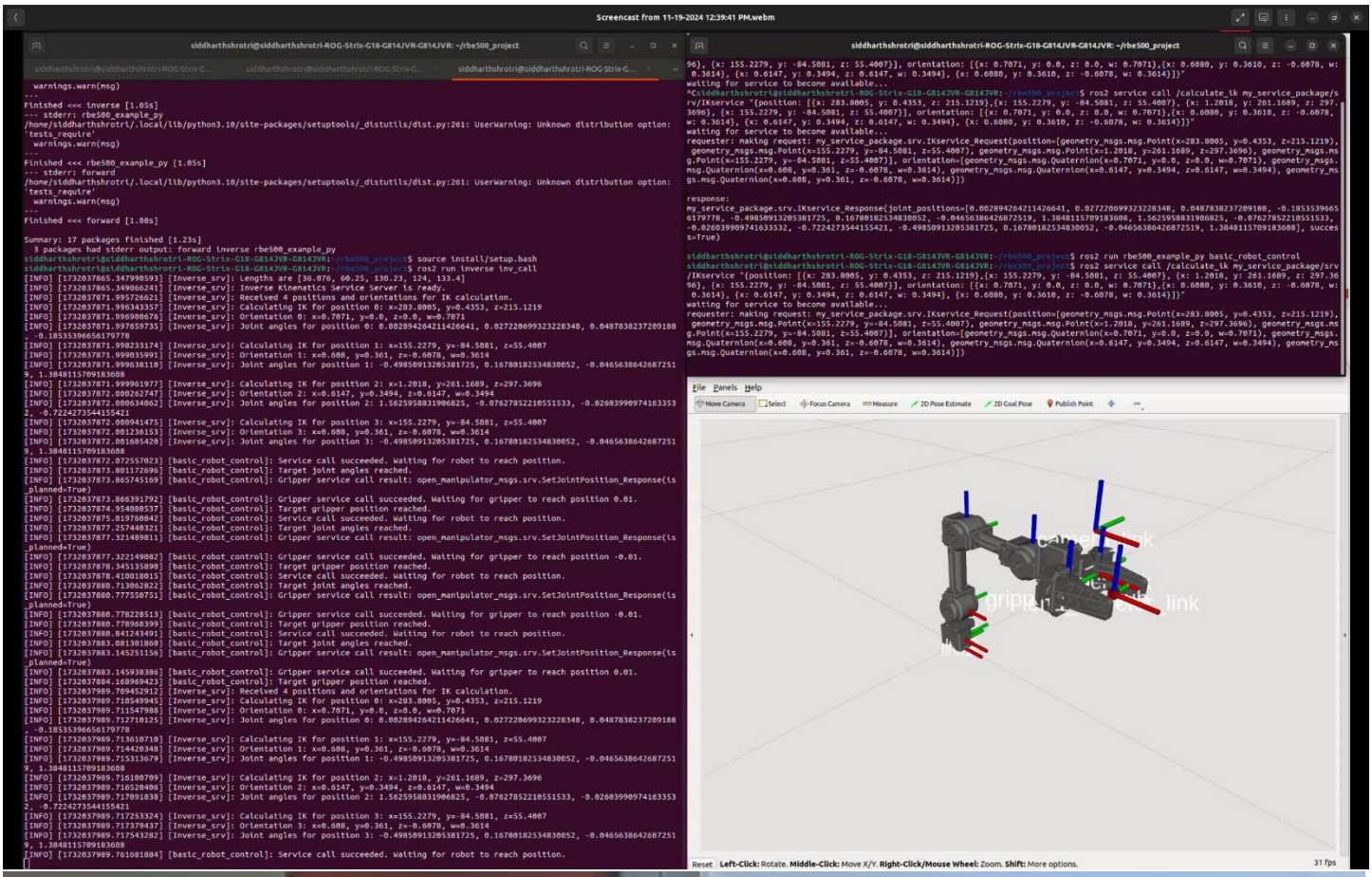
Testing Procedure

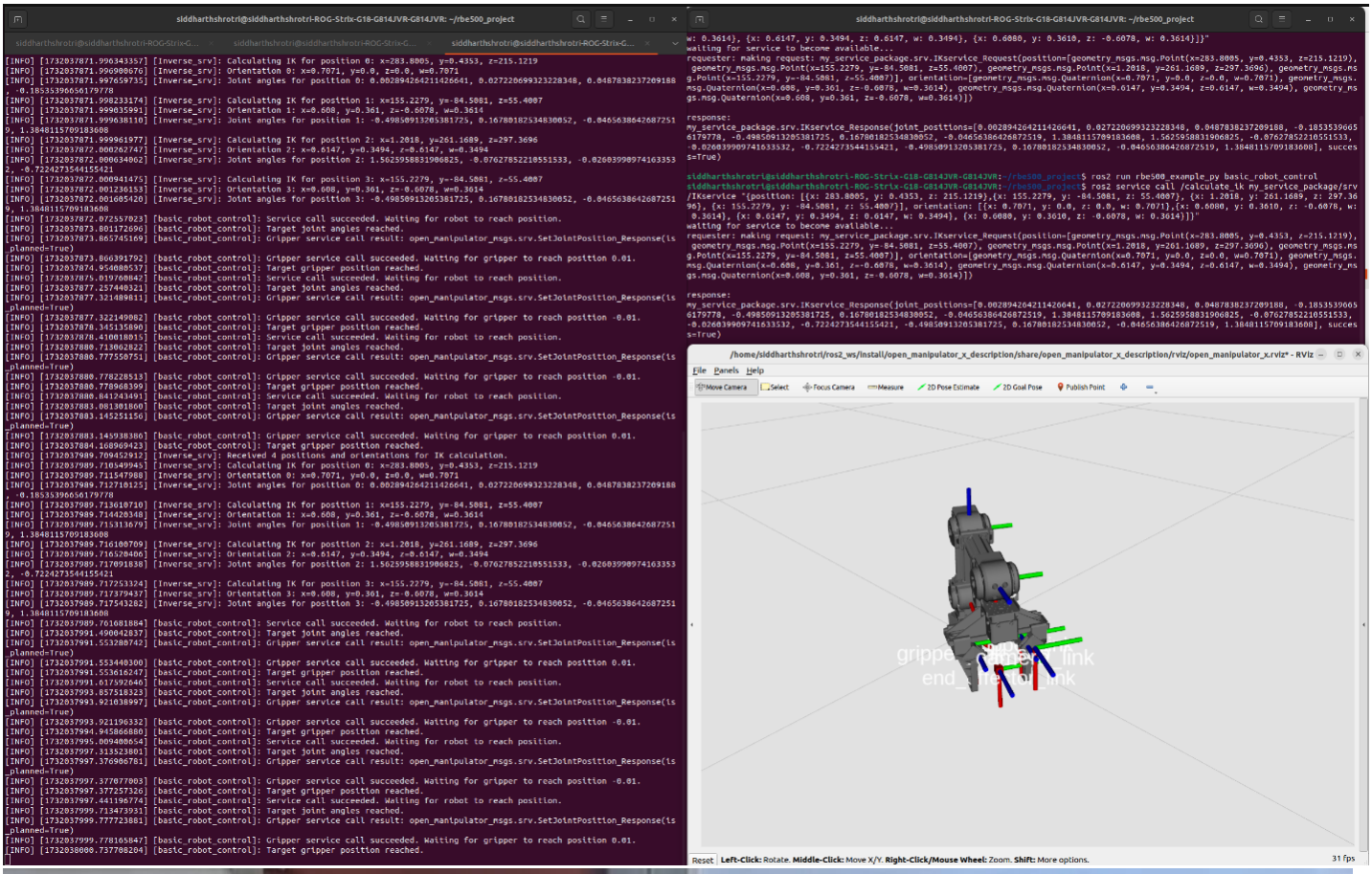
This code implements a ROS 2-based system for solving inverse kinematics (IK) and controlling a robotic manipulator with a gripper. The `Inverse` node provides an IK service (`calculate_ik`) that computes joint angles required to achieve given end-effector positions and orientations. The computation uses the robot's predefined link lengths and quaternion-based orientation handling, ensuring valid kinematic solutions with error checks. A separate `BasicRobotControl` class handles low-level control, sending joint and gripper commands via ROS services (`goal_joint_space_path` and `/goal_tool_control`) while monitoring the robot's state through `/joint_states`. The design is modular, separating the kinematics computation from hardware control, with robust logging for debugging.

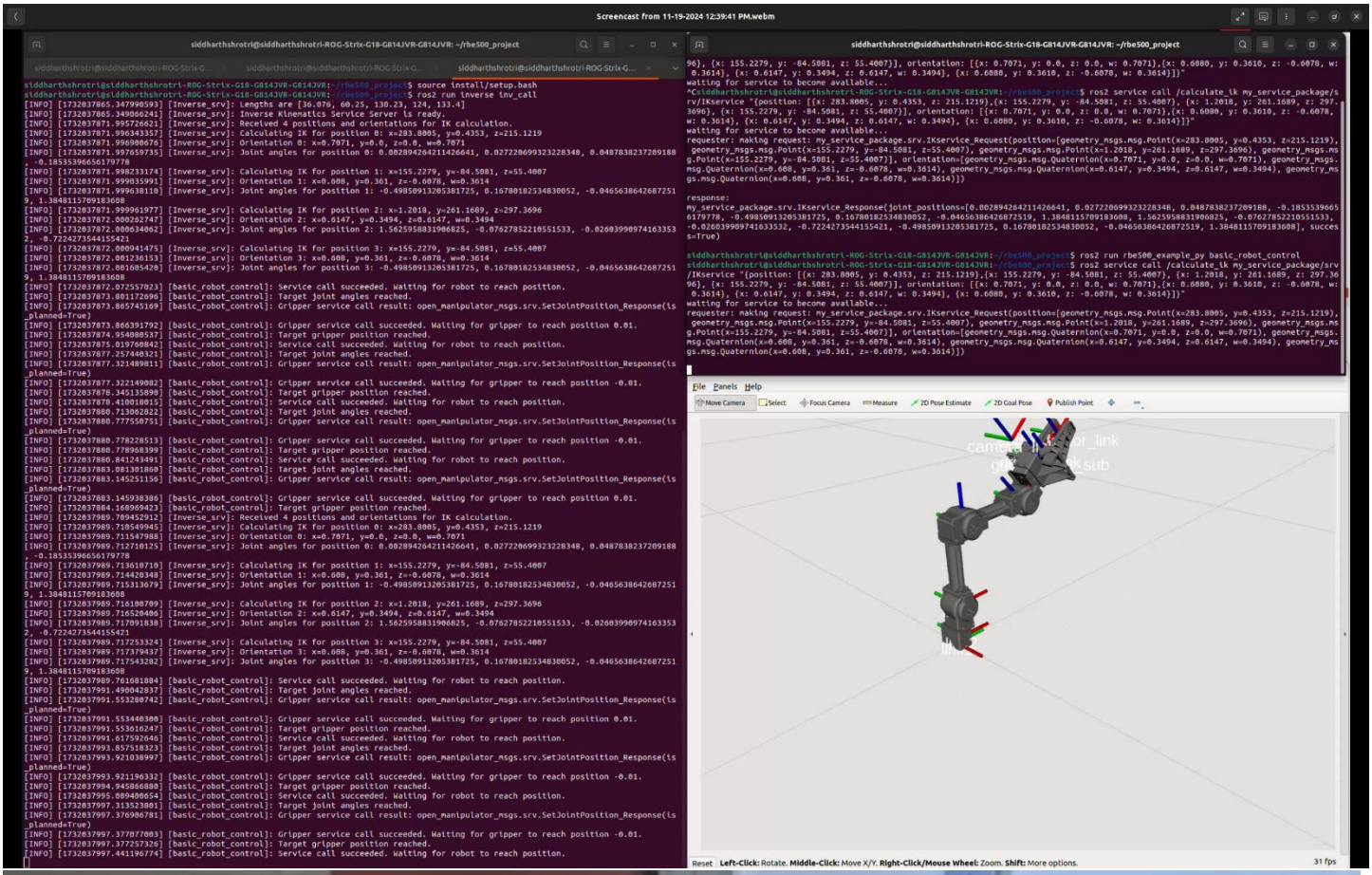
Inverse Kinematics

```python
class Inverse(Node):
    def __init__(self):
        super().__init__('Inverse_srv')

        # Link lengths (initialize these as per your manipulator's configuration)
        self.L1 = 36.076
        self.L2 = 60.25
        self.L3 = 130.23
        self.L4 = 124
        self.L5 = 133.4
        self.get_logger().info(f'Lengths are [{self.L1}, {self.L2}, {self.L3}, {self.L4}, {self.L5}]')

        # Creating the service for inverse kinematics
        self.srv = self.create_service(IKservice, 'calculate_ik', self.calculate_ik_callback)
        self.get_logger().info('Inverse Kinematics Service Server is ready.')

    def calculate_ik_callback(self, request, response):
        # Extract positions and orientations from the service request
        position = request.position
        orientation = request.orientation

        self.get_logger().info(f"Received {len(position)} positions and orientations for IK calculation.")

        try:
            # Perform IK and get joint angles along with success status
            joint_positions, success = self.perform_ik(position, orientation)
            grip_position = np.array([0.01, -0.01, -0.01, 0.01])  # Example grip positions

            if success:
                # Flatten joint positions into the response format
                response.joint_positions = [angle for joint in joint_positions for angle in joint]
                response.success = True

                # Pass the joint positions to the robot controller
                if not hasattr(self, 'basic_robot_control'):
                    self.basic_robot_control = BasicRobotControl(self)

                for idx, joint_set in enumerate(joint_positions):
                    # Send joint positions to the robot and wait for completion
                    self.basic_robot_control.send_request(joint_set)
                    self.basic_robot_control.wait_until_position_reached()

                    # Control the gripper for each position (open and close alternately)
                    grip_target = grip_position[idx]
                    self.basic_robot_control.control_gripper(grip_target)
                    self.basic_robot_control.wait_until_gripper_reached(grip_target)

            else:
                response.success = False  # Indicate failure in the response

        except Exception as e:
            self.get_logger().error(f"IK calculation failed: {e}")
            response.success = False

        return response

    def perform_ik(self, positions, orientations):
        joint_positions = []
        success = True
        A1 = math.radians(10.62)  # Adjusted angle for the manipulator

        # Loop through each position and orientation pair
        for idx, (pos, orient) in enumerate(zip(positions, orientations)):
            self.get_logger().info(f"Calculating IK for position {idx}: x={pos.x}, y={pos.y}, z={pos.z}")
            self.get_logger().info(f"Orientation {idx}: x={orient.x}, y={orient.y}, z={orient.z}, w={orient.w}")

            try:
                # Link lengths
                l1, l2, l3, l4, l5 = self.L1, self.L2, self.L3, self.L4, self.L5

                # Extract position and orientation
                xi, yi, zi = pos.x, pos.y, pos.z
                qx, qy, qz, qw = orient.x, orient.y, orient.z, orient.w

                # Normalize the quaternion
                norm = np.sqrt(qw**2 + qx**2 + qy**2 + qz**2)
                qw, qx, qy, qz = qw / norm, qx / norm, qy / norm, qz / norm

                # Calculate the rotation matrix from the quaternion
                R = np.array([
                    [1 - 2 * (qy**2 + qz**2), 2 * (qx * qy - qz * qw), 2 * (qx * qz + qy * qw)],
                    [2 * (qx * qy + qz * qw), 1 - 2 * (qx**2 + qz**2), 2 * (qy * qz - qx * qw)],
                    [2 * (qx * qz - qy * qw), 2 * (qy * qz + qx * qw), 1 - 2 * (qx**2 + qy**2)]
                ])

                # Adjust for kinematic offsets
                oi = np.array([[xi], [yi], [zi]])
                k = np.array([[1], [0], [0]])  # Unit vector along x-axis
                of = oi - np.dot(R, k) * l5

                x, y, z = of[0][0], of[1][0], of[2][0]

                # Calculate joint angles based on the manipulator's kinematics
                theta1 = math.atan2(y, x)
                D = (x**2 + y**2 + (z - (l1 + l2))**2 - l3**2 - l4**2) / (2 * l3 * l4)

                # Check for valid IK solution
                if abs(D) > 1:
                    self.get_logger().error(f"No valid IK solution for position {idx}. D={D}")
                    success = False
                    break

                theta3 = math.atan2(math.sqrt(1 - D**2), D)
                theta3_t = (theta3 - math.pi/2)

                theta2 = math.atan2((z - (l1 + l2)), math.sqrt(x**2 + y**2)) - \
                         math.atan2(l4 * math.sin(theta3), (l3 + l4 * math.cos(theta3)))
                theta4 = math.atan2((oi[2] - of[2]), (math.sqrt(oi[0]**2 + oi[1]**2) - math.sqrt(of[0]**2 + of[1]**2)))

                # Adjust joint angles for your specific manipulator's configuration
                theta2 = -theta2
                theta3_t = theta3_t + A1
                theta4 = -theta4 - A1

                self.get_logger().info(f"Joint angles for position {idx}: {theta1}, {theta2}, {theta3_t}, {theta4}")
                joint_positions.append([theta1, theta2, theta3_t, theta4])

            except Exception as e:
                self.get_logger().error(f"Error calculating IK for position {idx}: {e}")
                success = False
                break

        return joint_positions, success
```

Results:


Screencast from 11-19-2024 12:39:41 PM.webm
File Panels Help
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps


siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR: ~/rbe500_project
[INFO] [1732037871.996343357] [inverse_srv]: Calculating IK for position 0: x=283.8005, y=0.4353, z=215.1219
[INFO] [1732037871.996980676] [inverse_srv]: Orientation 0: x=0.7071, y=0.0, z=0.0, w=0.7071
[INFO] [1732037871.997659735] [inverse_srv]: Joint angles for position 0: 0.002894264211426641, 0.027220699323228348, 0.04878382372091883, -0.18535396656179778
[INFO] [1732037871.998233174] [inverse_srv]: Calculating IK for position 1: x=155.2279, y=-84.5881, z=55.4007
[INFO] [1732037871.999035991] [inverse_srv]: Orientation 1: x=0.608, y=0.361, z=-0.6078, w=0.3614
[INFO] [1732037871.999638110] [inverse_srv]: Joint angles for position 1: -0.4985891320538172 5, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608
[INFO] [1732037871.999961977] [inverse_srv]: Calculating IK for position 2: x=1.2018, y=261.1689, z=297.3696
[INFO] [1732037872.000762747] [inverse_srv]: Orientation 2: x=0.6147, y=0.3494, z=0.6147, w=0.3494
[INFO] [1732037872.000634062] [inverse_srv]: Joint angles for position 2: 1.5625958831906825, -0.07627852210551533, -0.02603990974163353 2, -0.7224273544155421
[INFO] [1732037872.000941475] [inverse_srv]: Calculating IK for position 3: x=155.2279, y=-84.5881, z=55.4007
[INFO] [1732037872.001236153] [inverse_srv]: Orientation 3: x=0.608, y=0.361, z=-0.6078, w=0.3614
[INFO] [1732037872.001605420] [inverse_srv]: Joint angles for position 3: -0.4985891320538172 5, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608
[INFO] [1732037872.072557023] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037873.801172696] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037873.865745169] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037873.866391792] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion 0.01.
[INFO] [1732037874.954080537] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037875.019760842] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037877.257440321] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037877.321489811] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037877.322149082] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion -0.01.
[INFO] [1732037878.345135890] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037878.410018015] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037880.713062822] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037880.777550751] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037880.778228513] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion -0.01.
[INFO] [1732037880.778960399] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037880.841241491] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037883.081301860] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037883.145251156] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037883.145933886] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion 0.01.
[INFO] [1732037884.168969423] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037989.709452912] [inverse_srv]: Received 4 positions and orientations for IK calculation.
[INFO] [1732037989.710549945] [inverse_srv]: Calculating IK for position 0: x=283.8005, y=0.4353, z=215.1219
[INFO] [1732037989.711547988] [inverse_srv]: Orientation 0: x=0.7071, y=0.0, z=0.0, w=0.7071
[INFO] [1732037989.712710125] [inverse_srv]: Joint angles for position 0: 0.002894264211426641, 0.027220699323228348, 0.04878382372091883, -0.18535396656179778
[INFO] [1732037989.713610710] [inverse_srv]: Calculating IK for position 1: x=155.2279, y=-84.5881, z=55.4007
[INFO] [1732037989.714420348] [inverse_srv]: Orientation 1: x=0.608, y=0.361, z=-0.6078, w=0.3614
[INFO] [1732037989.715313679] [inverse_srv]: Joint angles for position 1: -0.4985891320538172 5, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608
[INFO] [1732037989.716108709] [inverse_srv]: Calculating IK for position 2: x=1.2018, y=261.1689, z=297.3696
[INFO] [1732037989.716520406] [inverse_srv]: Orientation 2: x=0.6147, y=0.3494, z=0.6147, w=0.3494
[INFO] [1732037989.717091838] [inverse_srv]: Joint angles for position 2: 1.5625958831906825, -0.07627852210551533, -0.02603990974163353 2, -0.7224273544155421
[INFO] [1732037989.717253324] [inverse_srv]: Calculating IK for position 3: x=155.2279, y=-84.5881, z=55.4007
[INFO] [1732037989.717379437] [inverse_srv]: Orientation 3: x=0.608, y=0.361, z=-0.6078, w=0.3614
[INFO] [1732037989.717543282] [inverse_srv]: Joint angles for position 3: -0.4985891320538172 5, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608
[INFO] [1732037989.761681884] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037991.490042837] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037991.553288742] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037991.553440300] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion 0.01.
[INFO] [1732037991.553616247] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037991.617592646] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037993.857518323] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037993.921038997] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037993.921196332] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion -0.01.
[INFO] [1732037994.945866880] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037995.009400654] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037997.313523801] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037997.376986781] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037997.377077003] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion -0.01.
[INFO] [1732037997.377257326] [basic_robot_control]: Target gripper position reached.
[INFO] [1732037997.441196774] [basic_robot_control]: Service call succeeded. Waiting for robot to reach position.
[INFO] [1732037999.713473933] [basic_robot_control]: Target joint angles reached.
[INFO] [1732037999.777723881] [basic_robot_control]: Gripper service call result: open_manipulator_msgs.srv.SetJointPosition_Response(is_planned=True)
[INFO] [1732037999.778165847] [basic_robot_control]: Gripper service call succeeded. Waiting for gripper to reach posttion 0.01.
[INFO] [1732038000.737708204] [basic_robot_control]: Target gripper position reached.
siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR: ~/rbe500_project
w: 0.3614}, {x: 0.6147, y: 0.3494, z: 0.6147, w: 0.3494}, {x: 0.6088, y: 0.3610, z: -0.6078, w: 0.3614}]}"
waiting for service to become available...
requester: making request: my_service_package.srv.IKservice_Request(position=[geometry_msgs.msg.Point(x=283.8005, y=0.4353, z=215.1219), geometry_msgs.msg.Point(x=155.2279, y=-84.5881, z=55.4007), geometry_msgs.msg.Point(x=1.2018, y=261.1689, z=297.3696), geometry_msgs.msg.Point(x=155.2279, y=-84.5881, z=55.4007)], orientation=[geometry_msgs.msg.Quaternion(x=0.7071, y=0.0, z=0.0, w=0.7071), geometry_msgs.msg.Quaternion(x=0.608, y=0.361, z=-0.6078, w=0.3614), geometry_msgs.msg.Quaternion(x=0.6147, y=0.3494, z=0.6147, w=0.3494), geometry_msgs.msg.Quaternion(x=0.608, y=0.361, z=-0.6078, w=0.3614)])
response:
my_service_package.srv.IKservice_Response(joint_positions=[0.002894264211426641, 0.027220699323228348, 0.04878382372091883, -0.18535396656179778, -0.4985891320538172, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608, -0.07627852210551533, -0.02603990974163353, -0.7224273544155421, -0.4985891320538172, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608], success=True)
siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR:~/rbe500_project$ ros2 run rbe500_example_py basic_robot_control
siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR:~/rbe500_project$ ros2 service call /calculate_ik my_service_package/srv/IKservice "{position: [{x: 283.8005, y: 0.4353, z: 215.1219},{x: 155.2279, y: -84.5881, z: 55.4007}, {x: 1.2018, y: 261.1689, z: 297.3696}, {x: 155.2279, y: -84.5881, z: 55.4007}], orientation: [{x: 0.7071, y: 0.0, z: 0.0, w: 0.7071},{x: 0.6088, y: 0.3610, z: -0.6078, w: 0.3614}, {x: 0.6147, y: 0.3494, z: 0.6147, w: 0.3494}, {x: 0.6088, y: 0.3610, z: -0.6078, w: 0.3614}]}"
waiting for service to become available...
requester: making request: my_service_package.srv.IKservice_Request(position=[geometry_msgs.msg.Point(x=283.8005, y=0.4353, z=215.1219), geometry_msgs.msg.Point(x=155.2279, y=-84.5881, z=55.4007), geometry_msgs.msg.Point(x=1.2018, y=261.1689, z=297.3696), geometry_msgs.msg.Point(x=155.2279, y=-84.5881, z=55.4007)], orientation=[geometry_msgs.msg.Quaternion(x=0.7071, y=0.0, z=0.0, w=0.7071), geometry_msgs.msg.Quaternion(x=0.608, y=0.361, z=-0.6078, w=0.3614), geometry_msgs.msg.Quaternion(x=0.6147, y=0.3494, z=0.6147, w=0.3494), geometry_msgs.msg.Quaternion(x=0.608, y=0.361, z=-0.6078, w=0.3614)])
response:
my_service_package.srv.IKservice_Response(joint_positions=[0.002894264211426641, 0.027220699323228348, 0.04878382372091883, -0.18535396656179778, -0.4985891320538172, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608, -0.07627852210551533, -0.02603990974163353, -0.7224273544155421, -0.4985891320538172, 0.16780182534830052, -0.04656386426872519, 1.3848115709183608], success=True)
/home/siddharthshrotri/ros2_ws/install/open_manipulator_x_description/share/open_manipulator_x_description/rviz/open_manipulator_x.rviz* - RViz
File Panels Help
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps


Screencast from 11-19-2024 12:39:41 PM.webm
siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR: ~/rbe500_project
siddharthshrotri@siddharthshrotri-ROG-Strix-G18-G814JVR-G814JVR: ~/rbe500_project
File Panels Help
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps

3. Bonus Task: Object Pickup (20 Points)

Objective

The objective of the bonus task was to implement a sequence where the robot picks up an object placed at a known position and lifts it. The sequence required:

1. Moving the robot to an intermediate position above the object.
2. Moving the gripper down to the object's position.
3. Closing the gripper to pick up the object.
4. Returning to the intermediate position with the object lifted.

Steps

1. **Object Placement**: The object was placed at a known position in the workspace (e.g., a chessboard).
2. **Joint Angle Calculation**: The IK service was used to compute joint angles for the intermediate position and the exact pickup position.
3. **Movement Execution**: The calculated joint angles were sent to the robot using the following command:

```
ros2 service call /goal_joint_space_path
open_manipulator_msgs/srv/SetJointPosition"{...}"
```

4. **Gripper Control**: The gripper was controlled to pick up and hold the object.
5. **Object Lifting**: The robot was commanded to move back to the intermediate position with the object lifted.

Joint movement:

- Sends desired joint positions to the robot using the `/goal_joint_space_path` service.
- Monitors the robot's current joint angles to verify that the target position is reached.
- Implements a **tolerance-based check** to ensure the robot reaches the desired state.

Gripper control:

- Sends commands to control the gripper (e.g., open or close).
- `/goal_tool_control` service is used to open and close the gripper to grip the object.

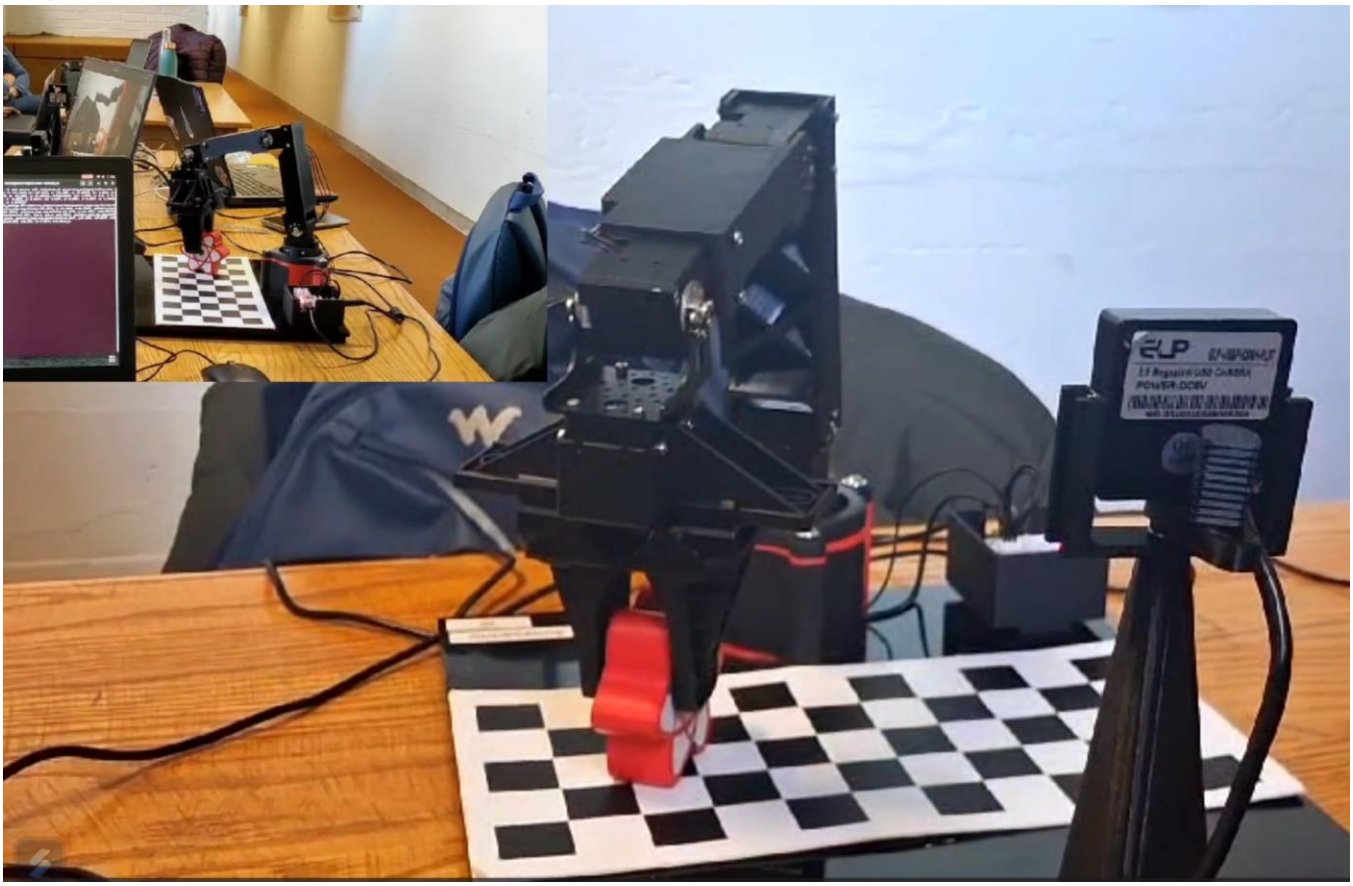
Code snippet of Robot and Gripper control

```python
class BasicRobotControl(Node):
    def __init__(self, parent_node):
        super().__init__('basic_robot_control')

        self.parent_node = parent_node
        self.current_joint_states = None
        self.joint_positions = None
        self.joint_state_received = False

        # Create service client for joint position control
        self.client = self.create_client(SetJointPosition, 'goal_joint_space_path')
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for SetJointPosition service...')

        # Create service client for gripper control
        self.gripper_client = self.create_client(SetJointPosition, '/goal_tool_control')
        while not self.gripper_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for /goal_tool_control service...')

        # Subscribe to /joint_states to monitor the current joint positions
        self.create_subscription(JointState, '/joint_states', self.joint_state_callback, 10)

    def joint_state_callback(self, msg):
        # Update the current joint states
        self.current_joint_states = msg.position
        self.joint_state_received = True

    def send_request(self, joint_positions):
        # Send the joint positions using SetJointPosition service
        self.joint_positions = joint_positions
        request = SetJointPosition.Request()
        request.planning_group = ''
        request.joint_position.joint_name = ['joint1', 'joint2', 'joint3', 'joint4']
        request.joint_position.position = joint_positions
        request.path_time = 2.5

        self.future = self.client.call_async(request)

    def wait_until_position_reached(self):
        # Wait for the robot to reach the desired joint positions
        rclpy.spin_until_future_complete(self, self.future)
        if self.future.done():
            try:
                self.future.result()
                self.get_logger().info("Service call succeeded. Waiting for robot to reach position.")
            except Exception as e:
                self.get_logger().error(f"Service call failed: {e}")
                return

        # Verify that the robot reached the target position
        tolerance = 0.05  # Allowable difference in joint angles
        while rclpy.ok():
            rclpy.spin_once(self)
            if self.joint_state_received and self.current_joint_states:
                differences = [abs(target - current) for target, current in zip(self.joint_positions, self.current_joint_states)]
                if all(diff < tolerance for diff in differences):
                    self.get_logger().info("Target joint angles reached.")
                    break

    def control_gripper(self, grip_position):
        # Send command to control the gripper
        request = SetJointPosition.Request()
        request.planning_group = ''
        request.joint_position.joint_name = ['gripper']
        request.joint_position.position = [grip_position]
        request.path_time = 2.5
        self.gripper_future = self.gripper_client.call_async(request)

    def wait_until_gripper_reached(self, grip_position):
        # Wait for the gripper to reach the target position
        tolerance = 0.05  # Allowable difference in gripper position
        while rclpy.ok():
            rclpy.spin_once(self)
            if self.joint_state_received and self.current_joint_states:
                diff = abs(grip_position - self.current_joint_states[-1])
                if diff < tolerance:
                    self.get_logger().info("Target gripper position reached.")
                    break
```

Results

The video for Robot Pick and Place is uploaded separately

Step 1:



Step 2 :