Lab 5 – Exercise

Meet Kansara – 220929270 Roll no. 54

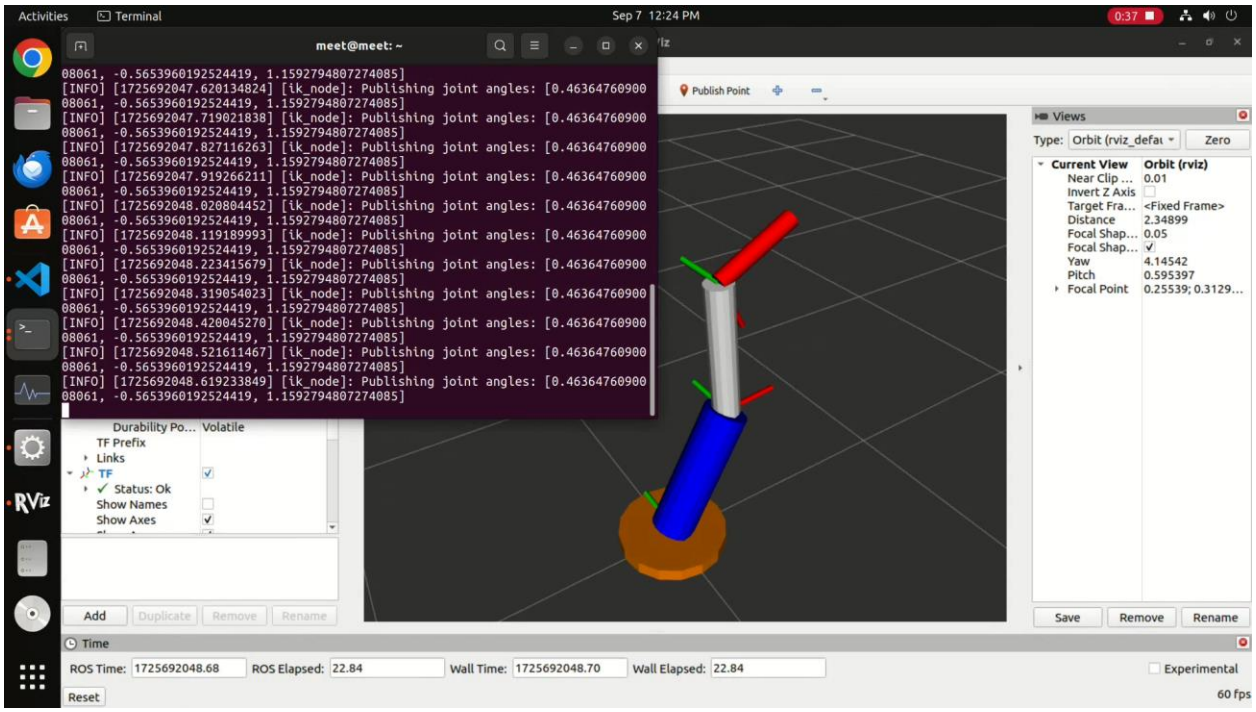
Exercise: To simulate a manipulator using inverse kinematics controllers, and to simulate a UR5 model in Rviz and Gazebo.

Code Execution and analysis:

1. Inverse kinematics controller for manipulator in RViz:

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
from rclpy.clock import Clock
import sys
import math

class InverseKinematicsPublisher(Node):
    def __init__(self):
        super().__init__('ik_node')
        topic_ = "/joint_states"
        self.joints = ['base_arm1_joint', 'arm1_arm2_joint', 'arm2_arm3_joint']

        # Robot arm link lengths (modify these according to your robot's specifications)
        self.L1 = 0.5  # Length of the first link
        self.L2 = 0.5  # Length of the second link
        self.L3 = 0.3  # Length of the third link

        # Handle command-line arguments
        if len(sys.argv) < 4:
            self.get_logger().error("Not enough arguments provided. Using default values.")
            self.goal_ = [0.5, 0.5, 0.5]  # Default end-effector position
        else:
            try:
                self.goal_ = [float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3])]
            except ValueError:
                self.get_logger().error("Invalid argument(s) provided. Using default values.")
                self.goal_ = [0.5, 0.5, 0.5]  # Default end-effector position

        self.publisher_ = self.create_publisher(JointState, topic_, 10)
        self.timer_ = self.create_timer(0.1, self.timer_callback)

    def inverse_kinematics(self, x, y, z):
        # Calculate theta1 (base rotation)
        theta1 = math.atan2(y, x)

        # Calculate the distance from the base to the end-effector in the x-y plane
        r = math.sqrt(x**2 + y**2)

        # Calculate the distance from the second joint to the end-effector
        s = z - self.L1

        # Calculate the distance from the second joint to the end-effector
        D = math.sqrt(r**2 + s**2)

        # Calculate theta3 using the cosine law
        cos_theta3 = (D**2 - self.L2**2 - self.L3**2) / (2 * self.L2 * self.L3)
```

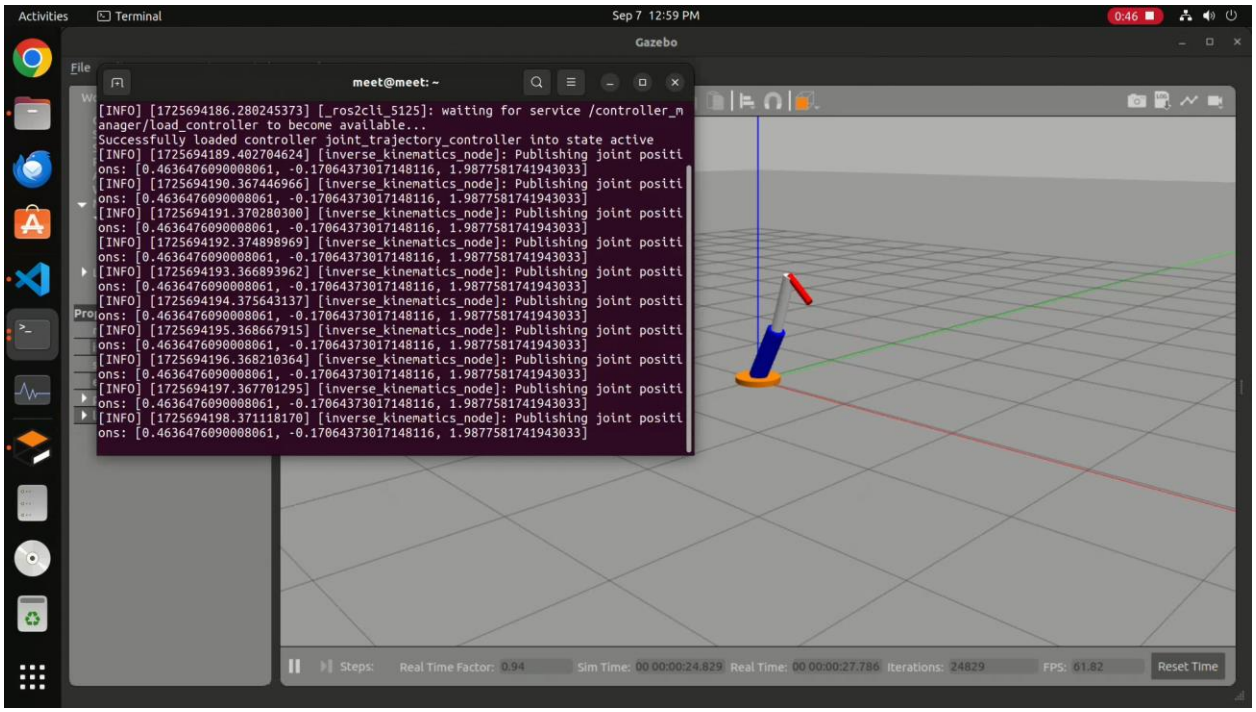
```
        if cos_theta3 > 1 or cos_theta3 < -1:
            self.get_logger().error("Target position is out of reach!")
            return None
        theta3 = math.acos(cos_theta3)

        # Calculate theta2
        theta2 = math.atan2(s, r) - math.atan2(self.L3 * math.sin(theta3), self.L2 + self.L3 *
math.cos(theta3))

        return [theta1, theta2, theta3]

    def timer_callback(self):
        joint_angles = self.inverse_kinematics(self.goal_[0], self.goal_[1], self.goal_[2])
        if joint_angles is None:
            return
        msg = JointState()
        current_time = self.get_clock().now().to_msg()
        msg.header.stamp = current_time
        msg.name = self.joints
        msg.position = joint_angles
        self.publisher_.publish(msg)
        self.get_logger().info(f"Publishing joint angles: {joint_angles}")

def main(args=None):
    rclpy.init(args=args)
    node = InverseKinematicsPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



2. Inverse kinematics controller for manipulator in Gazebo:

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from builtin_interfaces.msg import Duration
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from geometry_msgs.msg import Pose
import numpy as np

class InverseKinematicsPublisher(Node):

    def __init__(self):
        super().__init__('inverse_kinematics_node')
        topic_ = "/joint_trajectory_controller/joint_trajectory"
        self.joints = ['base_arm1_joint', 'arm1_arm2_joint', 'arm2_arm3_joint']
        self.link_lengths = [0.5, 0.5, 0.3]  # Length of arm1, arm2, and arm3
        self.joint_limits = [
            (-2.14, 2.14),  # base_arm1_joint limits
            (-2.14, 2.14),  # arm1_arm2_joint limits
            (-2.14, 2.14)   # arm2_arm3_joint limits
        ]
        self.declare_parameter("end_effector_pose", [0.5, 0.5, 0.5])
        self.goal_pose = self.get_parameter("end_effector_pose").value
        self.publisher_ = self.create_publisher(JointTrajectory, topic_, 10)
        self.timer_ = self.create_timer(1, self.timer_callback)

    def inverse_kinematics(self, x, y, z):
        L1, L2, L3 = self.link_lengths

        # Calculate the distance to the target
        target_distance = np.sqrt(x**2 + y**2 + z**2)

        # Check if the target is reachable
        if target_distance > sum(self.link_lengths):
            self.get_logger().warn("Target position is out of reach. Moving to closest possible
position.")
            scale = sum(self.link_lengths) / target_distance
            x *= scale
            y *= scale
            z *= scale

        # Calculate theta1 (base rotation)
        theta1 = np.arctan2(y, x)

        # Calculate the position of the wrist center
        wc_x = x - L3 * np.cos(theta1)
        wc_y = y - L3 * np.sin(theta1)
        wc_z = z

        # Calculate distance to wrist center
        D = np.sqrt(wc_x**2 + wc_y**2 + wc_z**2)

        # Calculate theta3
        cos_theta3 = (D**2 - L1**2 - L2**2) / (2 * L1 * L2)
        cos_theta3 = np.clip(cos_theta3, -1.0, 1.0)  # Ensure the value is in the valid range
        theta3 = np.arccos(cos_theta3)

        # Calculate theta2
```

```
        theta2 = np.arctan2(wc_z, np.sqrt(wc_x**2 + wc_y**2)) - np.arctan2(L2 * np.sin(theta3), L1
+ L2 * np.cos(theta3))

        # Apply joint limits
        theta1 = np.clip(theta1, self.joint_limits[0][0], self.joint_limits[0][1])
        theta2 = np.clip(theta2, self.joint_limits[1][0], self.joint_limits[1][1])
        theta3 = np.clip(theta3, self.joint_limits[2][0], self.joint_limits[2][1])

        return [theta1, theta2, theta3]

    def timer_callback(self):
        msg = JointTrajectory()
        msg.joint_names = self.joints
        point = JointTrajectoryPoint()
        joint_positions = self.inverse_kinematics(*self.goal_pose)
        point.positions = joint_positions
        point.time_from_start = Duration(sec=2)
        msg.points.append(point)
        self.publisher_.publish(msg)
        self.get_logger().info(f"Publishing joint positions: {joint_positions}")

def main(args=None):
    rclpy.init(args=args)
    node = InverseKinematicsPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



UR5 urdf file:

```
<?xml version="1.0"?>
<robot name="ur5">
  <link name="base_link">
    <inertial>
      <origin xyz="-0.00036702 -0.00036758 0.016561" rpy="0 0 0" />
      <mass value="4.0" />
      <inertia ixx="0.00060258" ixy="1.7502E-05" ixz="1.5277E-06" iyy="0.00060261" iyz="1.5241E-06"
izz="0.0010709" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/base_link.STL" />
      </geometry>
      <material name="White">
        <color rgba="0 0 0 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/base_link.STL" />
      </geometry>
    </collision>
  </link>

  <link name="shoulder_link">
    <inertial>
      <origin xyz="-0.078966 0.016439 -6.3565E-06" rpy="0 0 0" />
      <mass value="3.7" />
      <inertia ixx="0.0028298" ixy="0.00031932" ixz="-1.3132E-07" iyy="0.002538" iyz="4.4907E-07"
izz="0.0026207" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/shoulder_link.STL" />
      </geometry>
      <material name="Black">
        <color rgba="0 0 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/shoulder_link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="shoulder_pan_joint" type="revolute">
    <origin xyz="0 0 0.0099916" rpy="1.5708 1.5708 0" />
    <parent link="base_link" />
    <child link="shoulder_link" />
    <axis xyz="1 0 0" />
    <limit lower="-6.2832" upper="6.2832" effort="300" velocity="3" />
    <dynamics damping="10" friction="1" />
  </joint>

  <link name="upper_arm_link">
    <inertial>
      <origin xyz="-0.21652 0.079293 8.661E-08" rpy="0 0 0" />
      <mass value="8.393" />
      <inertia ixx="0.0063671" ixy="-9.2154E-12" ixz="9.0018E-07" iyy="0.021824" iyz="-2.2572E-07"
izz="0.021327" />
    </inertial>
```

```
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/upper_arm_link.STL" />
      </geometry>
      <material name="Blue">
        <color rgba="0 1 0 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/upper_arm_link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="shoulder_lift_joint" type="revolute">
    <origin xyz="-0.080008 0.053821 0" rpy="0 0 0" />
    <parent link="shoulder_link" />
    <child link="upper_arm_link" />
    <axis xyz="0 -1 0" />
    <limit lower="-1.57" upper="1.57" effort="200" velocity="3" />
    <dynamics damping="10" friction="1" />
  </joint>

  <link name="forearm_link">
    <inertial>
      <origin xyz="-0.1739 -0.033049 -2.6025E-06" rpy="0 0 0" />
      <mass value="2.275" />
      <inertia ixx="0.002241" ixy="-0.00011285" ixz="4.2229E-07" iyy="0.011808" iyz="-2.6928E-07"
izz="0.011655" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/forearm_link.STL" />
      </geometry>
      <material name="Green">
        <color rgba="0.5 0 0.5 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/forearm_link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="elbow_joint" type="revolute">
    <origin xyz="-0.425 0 0" rpy="0 0 0" />
    <parent link="upper_arm_link" />
    <child link="forearm_link" />
    <axis xyz="0 1 0" />
    <limit lower="-6.2832" upper="6.2832" effort="200" velocity="3" />
    <dynamics damping="10" friction="1" />
  </joint>

  <link name="wrist_1_link">
    <inertial>
      <origin xyz="0.003753 0.044978 1.2618E-05" rpy="0 0 0" />
      <mass value="1.219" />
      <inertia ixx="0.00041141" ixy="4.2784E-05" ixz="-2.8598E-07" iyy="0.00036744" iyz="4.089E-07"
izz="0.000387" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
```

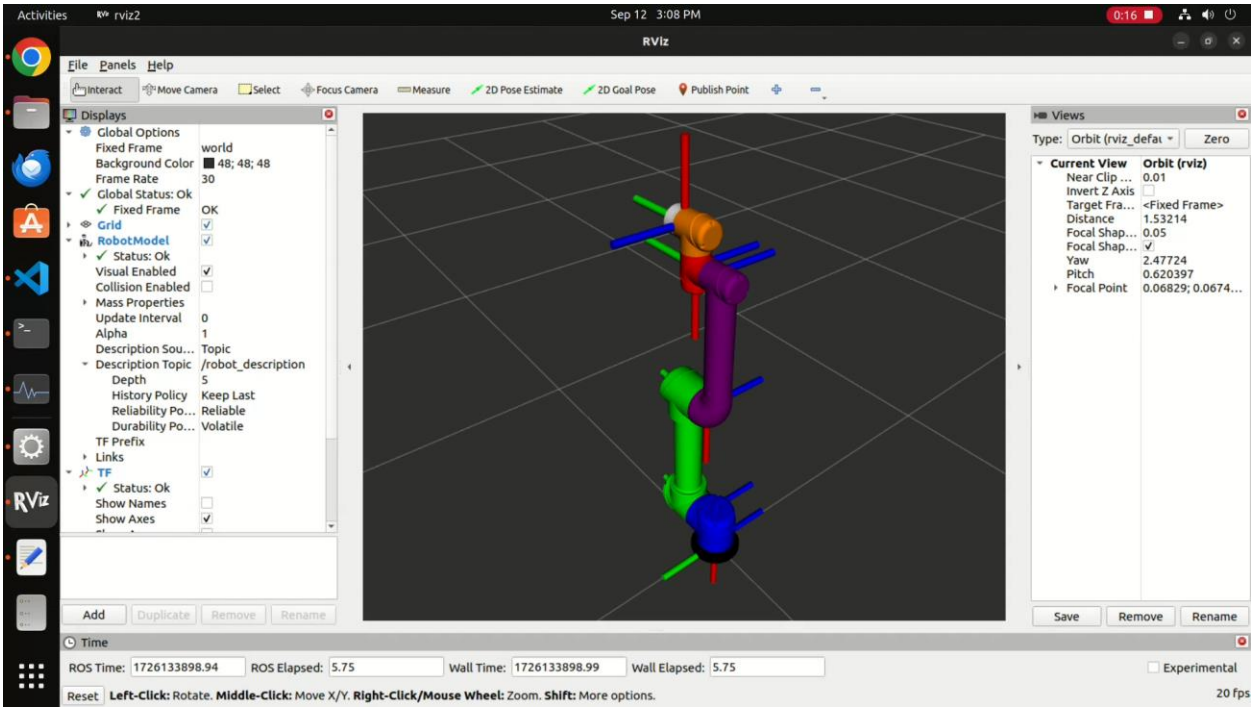
```
      <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_1_link.STL" />
    </geometry>
    <material name="Purple">
      <color rgba="1 0 0 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_1_link.STL" />
    </geometry>
  </collision>
</link>

<joint name="wrist_1_joint" type="revolute">
  <origin xyz="-0.39225 0 0" rpy="0 0 0" />
  <parent link="forearm_link" />
  <child link="wrist_1_link" />
  <axis xyz="0 -1 0" />
  <limit lower="-6.2832" upper="6.2832" effort="200" velocity="3" />
  <dynamics damping="10" friction="1" />
</joint>

<link name="wrist_2_link">
  <inertial>
    <origin xyz="-0.083749 -0.0037532 -1.2446E-05" rpy="0 0 0" />
    <mass value="1.219" />
    <inertia
      ixx="0.00036744" ixy="4.2782E-05" ixz="4.1013E-07"
      iyy="0.0004114" iyz="-2.8709E-07" izz="0.000387" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_2_link.STL" />
    </geometry>
    <material name="Red">
      <color rgba="1 0.5 0 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_2_link.STL" />
    </geometry>
  </collision>
</link>

<joint name="wrist_2_joint" type="revolute">
  <origin xyz="0 0.056149 0" rpy="0 0 0" />
  <parent link="wrist_1_link" />
  <child link="wrist_2_link" />
  <axis xyz="1 0 0" />
  <limit lower="-6.2832" upper="6.2832" effort="200" velocity="3" />
  <dynamics damping="10" friction="1" />
</joint>

<link name="wrist_3_link">
  <inertial>
    <origin xyz="-7.8282E-05 0.018179 0.00011982" rpy="0 0 0" />
    <mass value="0.1879" />
    <inertia
      ixx="6.8269E-05" ixy="5.7754E-08" ixz="-3.2009E-07"
      iyy="0.00011604" iyz="-8.84E-08" izz="6.8551E-05" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_3_link.STL" />
```

```
      </geometry>
      <material name="Orange">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="/home/meet/ros2_ws/src/ur5_tutorial/meshes/wrist_3_link.STL" />
      </geometry>
    </collision>
  </link>

  <joint name="wrist_3_joint" type="revolute">
    <origin xyz="-0.09492 0.041547 0" rpy="3.1416 0 3.1416" />
    <parent link="wrist_2_link" />
    <child link="wrist_3_link" />
    <axis xyz="0 -1 0" />
    <limit lower="-6.2832" upper="6.2832" effort="100" velocity="3" />
    <dynamics damping="10" friction="1" />
  </joint>

  <link name="world">
    <!-- No inertial properties required -->
  </link>

  <joint name="world_to_base" type="fixed">
    <parent link="world"/>
    <child link="base_link"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </joint>

  <gazebo reference="base_link">
      <material>Gazebo/Black</material>
  </gazebo>

  <gazebo reference="shoulder_link">
      <material>Gazebo/Blue</material>
  </gazebo>

  <gazebo reference="upper_arm_link">
      <material>Gazebo/Green</material>
  </gazebo>

  <gazebo reference="forearm_link">
      <material>Gazebo/Purple</material>
  </gazebo>

  <gazebo reference="wrist_1_link">
    <material>Gazebo/Red</material>
  </gazebo>

  <gazebo reference="wrist_2_link">
    <material>Gazebo/Orange</material>
  </gazebo>

  <gazebo reference="wrist_3_link">
    <material>Gazebo/White</material>
  </gazebo>

  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <robot_sim_type>gazebo_ros2_control/GazeboSystem</robot_sim_type>
      <parameters>/home/meet/ros2_ws/src/ur5_tutorial/config/ur5control.yaml</parameters>
    </plugin>
  </gazebo>

  <ros2_control name="GazeboSystem" type="system">
    <hardware>
```

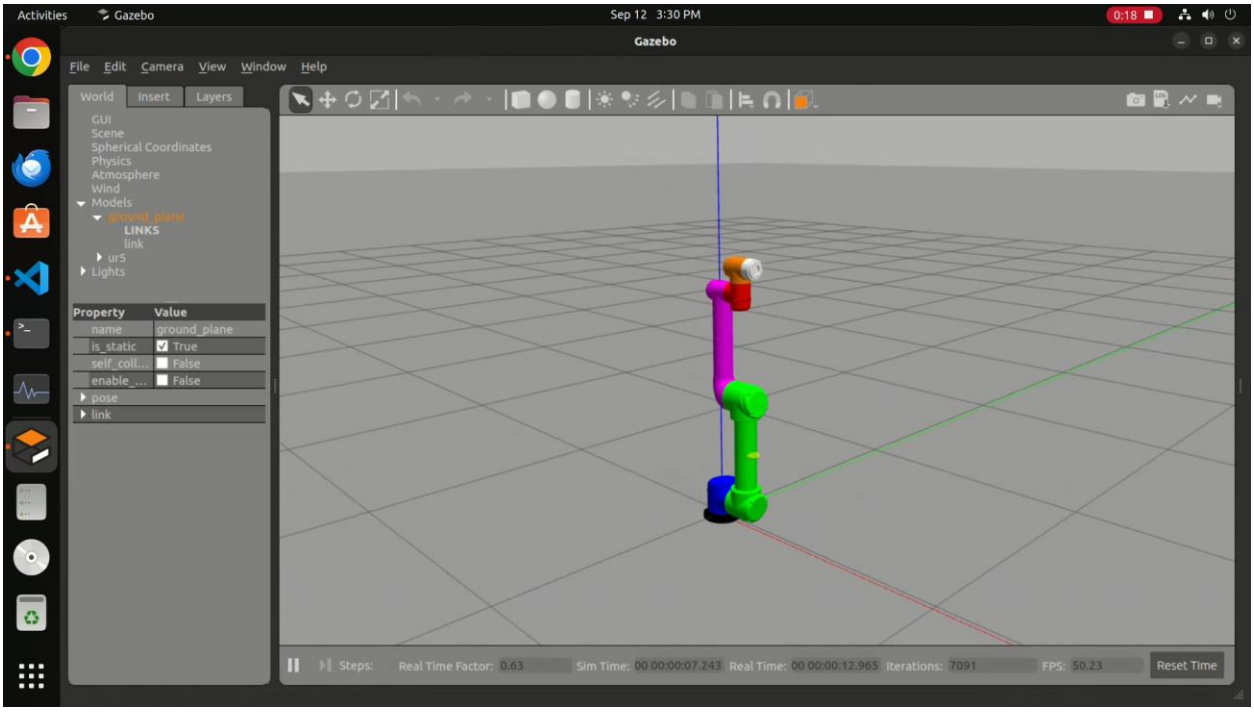
```
      <plugin>gazebo_ros2_control/GazeboSystem</plugin>
    </hardware>

    <joint name="shoulder_pan_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>

    <joint name="shoulder_lift_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>

    <joint name="elbow_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>

    <joint name="wrist_1_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>

    <joint name="wrist_2_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>

    <joint name="wrist_3_joint">
      <command_interface name="position"/>
      <state_interface name="position">
        <param name="initial_position">0.0</param>
      </state_interface>
      <state_interface name="velocity"/>
    </joint>
  </ros2_control>

  <transmission name="base_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="shoulder_pan_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="base_link_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="shoulder_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="shoulder_lift_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="shoulder_link_motor">
```

```
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="upper_arm_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="elbow_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="uppr_arm_link_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="forearm_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="wrist_1_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="forearm_link_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="wrist_1_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="wrist_2_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="wrist_1_link_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="wrist_2_link_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="wrist_3_joint">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="wrist_2_link_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <gazebo reference="shoulder_link">
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="upper_arm_link">
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="forearm_link">
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="wrist_1_link">
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="wrist_2_link">
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="wrist_3_link">
    <selfCollide>true</selfCollide>
  </gazebo>

</robot>
```

UR5 model in RViz:



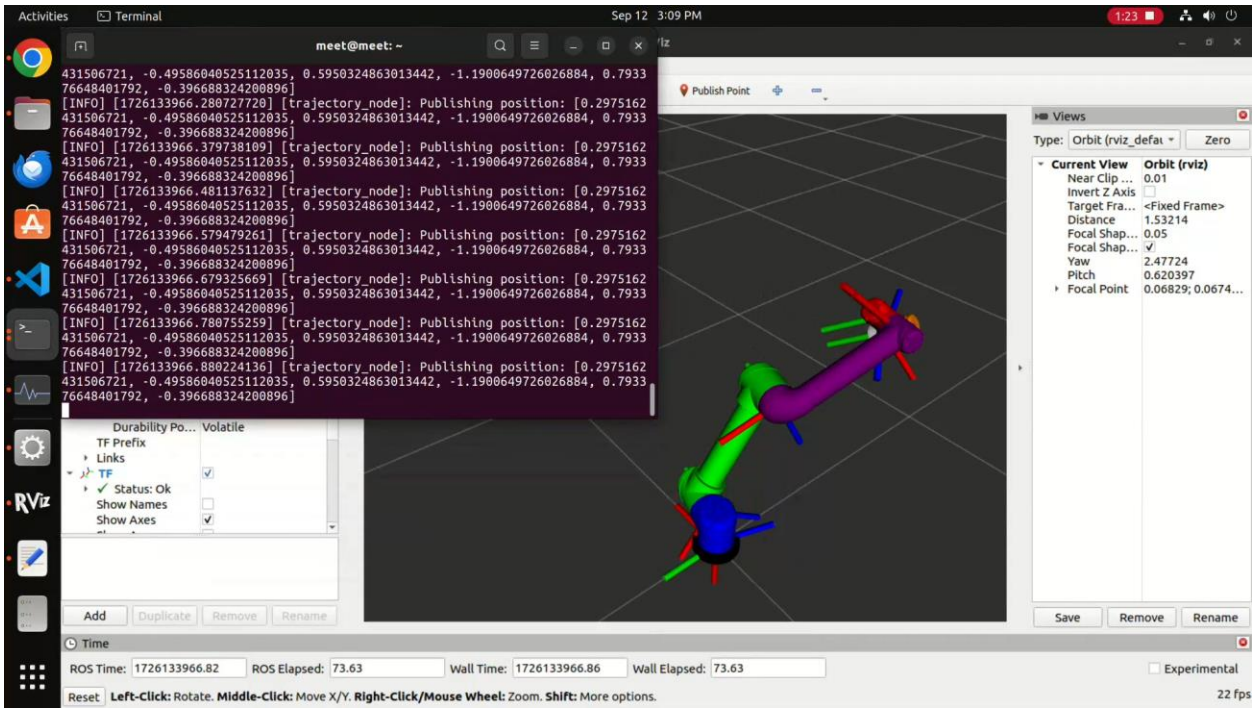
UR5 model in Gazebo:



3. Simulation of UR5 model In RViz:

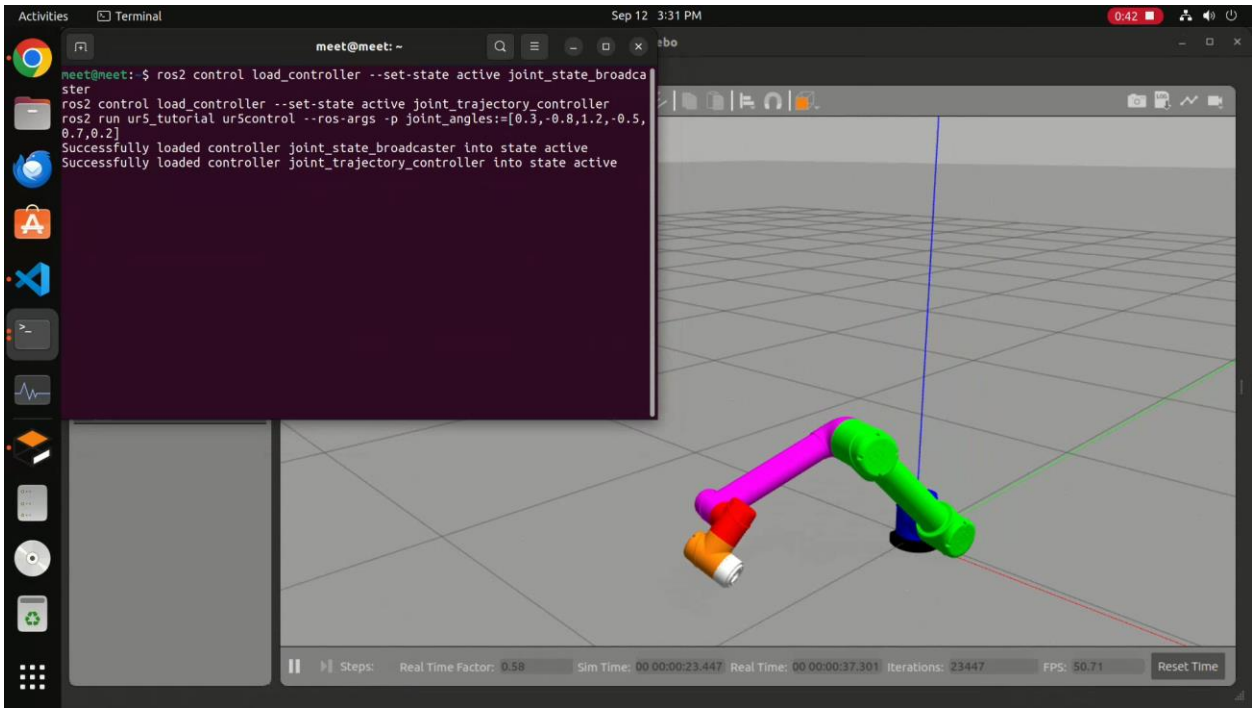
```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
from rclpy.clock import Clock
import sys
import atexit

class TrajectoryPublisher(Node):
    def __init__(self):
        super().__init__('trajectory_node')

        topic_ = "/joint_states"
        self.joints = [
            'shoulder_pan_joint', 'shoulder_lift_joint',
            'elbow_joint', 'wrist_1_joint',
            'wrist_2_joint', 'wrist_3_joint'
        ]

        # Handle command-line arguments
        if len(sys.argv) < 7:
            self.get_logger().error("Not enough arguments provided. Using default values.")
            self.goal_ = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  # Default values
        else:
            try:
                self.goal_ = [float(sys.argv[i]) for i in range(1, 7)]
            except ValueError:
                self.get_logger().error("Invalid argument(s) provided. Using default values.")
                self.goal_ = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  # Default values

        self.publisher_ = self.create_publisher(JointState, topic_, 10)
        self.timer_ = self.create_timer(0.1, self.timer_callback)

        # For interpolation
        self.current_position = [0.0] * len(self.joints)
        self.step_size = 0.01
        self.total_steps = int(1 / self.step_size)
        self.goal_reached = False

        # Register exit handler
        atexit.register(self.hold_final_position)

    def timer_callback(self):
        msg = JointState()
        current_time = Clock().now().to_msg()

        msg.header.stamp.sec = current_time.sec
        msg.header.stamp.nanosec = current_time.nanosec
        msg.name = self.joints

        if not self.goal_reached:
            # Interpolate each joint position towards the goal
            for i in range(len(self.joints)):
                self.current_position[i] += (self.goal_[i] - self.current_position[i]) *
self.step_size

            # Check if the goal is reached
```

```
            if all(abs(self.goal_[i] - self.current_position[i]) < 0.01 for i in
range(len(self.joints))):
                self.goal_reached = True
                self.get_logger().info("Goal reached. Holding position...")

        msg.position = self.current_position
        self.publisher_.publish(msg)
        self.get_logger().info("Publishing position: {}".format(self.current_position))

    def hold_final_position(self):
        msg = JointState()
        current_time = Clock().now().to_msg()

        msg.header.stamp.sec = current_time.sec
        msg.header.stamp.nanosec = current_time.nanosec
        msg.name = self.joints
        msg.position = self.current_position

        # Publish the final position one more time to ensure it's held
        self.publisher_.publish(msg)
        self.get_logger().info("Final position held: {}".format(self.current_position))

def main(args=None):
    rclpy.init(args=args)
    node = TrajectoryPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



4. Simulation of UR5 model in Gazebo:

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from builtin_interfaces.msg import Duration
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint

class UR5TrajectoryPublisher(Node):

    def __init__(self):
        super().__init__('ur5_trajectory_node')
        topic_ = "/joint_trajectory_controller/joint_trajectory"
        self.joints = ['shoulder_pan_joint', 'shoulder_lift_joint', 'elbow_joint', 'wrist_1_joint',
'wrist_2_joint', 'wrist_3_joint']
        self.declare_parameter("joint_angles", [0.0, -1.57, 0.0, -1.57, 0.0, 0.0])
        self.goal_ = self.get_parameter("joint_angles").value
        self.publisher_ = self.create_publisher(JointTrajectory, topic_, 10)
        self.timer_ = self.create_timer(1, self.timer_callback)

    def timer_callback(self):
        msg = JointTrajectory()
        msg.joint_names = self.joints
        point = JointTrajectoryPoint()
        point.positions = self.goal_
        point.time_from_start = Duration(sec=2)
        msg.points.append(point)
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = UR5TrajectoryPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



```yaml
controller_manager:
  ros__parameters:
    update_rate: 100  # Hz

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

    joint_trajectory_controller:
      type: joint_trajectory_controller/JointTrajectoryController

joint_trajectory_controller:
  ros__parameters:
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint
    interface_name: position
    command_interfaces:
      - position
    state_interfaces:
      - position
      - velocity
    state_publish_rate: 50.0
    action_monitor_rate: 20.0
    allow_partial_joints_goal: false
    constraints:
      stopped_velocity_tolerance: 0.01
      goal_time: 0.0
      shoulder_pan_joint: {trajectory: 0.05, goal: 0.03}
      shoulder_lift_joint: {trajectory: 0.05, goal: 0.03}
      elbow_joint: {trajectory: 0.05, goal: 0.03}
      wrist_1_joint: {trajectory: 0.05, goal: 0.03}
      wrist_2_joint: {trajectory: 0.05, goal: 0.03}
      wrist_3_joint: {trajectory: 0.05, goal: 0.03}
```

Conclusion: Simulations of invers kinematics controllers for manipulator, and control of UR5 model have been successfully conducted in RViz and Gazebo.