

AI 407 – Introduction to Robotics

Lab 7 Manual

OpenManipulator-X Programming and Control - Part 2

❖ Lab objectives

- ✓ Learn to use MoveIt for motion planning and robot control.
- ✓ Visualizing robot movements and sensor data in RViz.
- ✓ Learn how to subscribe to the `/joint_states` topic.

❖ Lab Requirements

- ✓ **Software:** Ubuntu 22.04 LTS, ROS 2 Humble
 - ✓ **Hardware:** Students should work on Lab Devices
-

▪ Before You Start

Kindly read the manual, review the references if any, before beginning implementation.

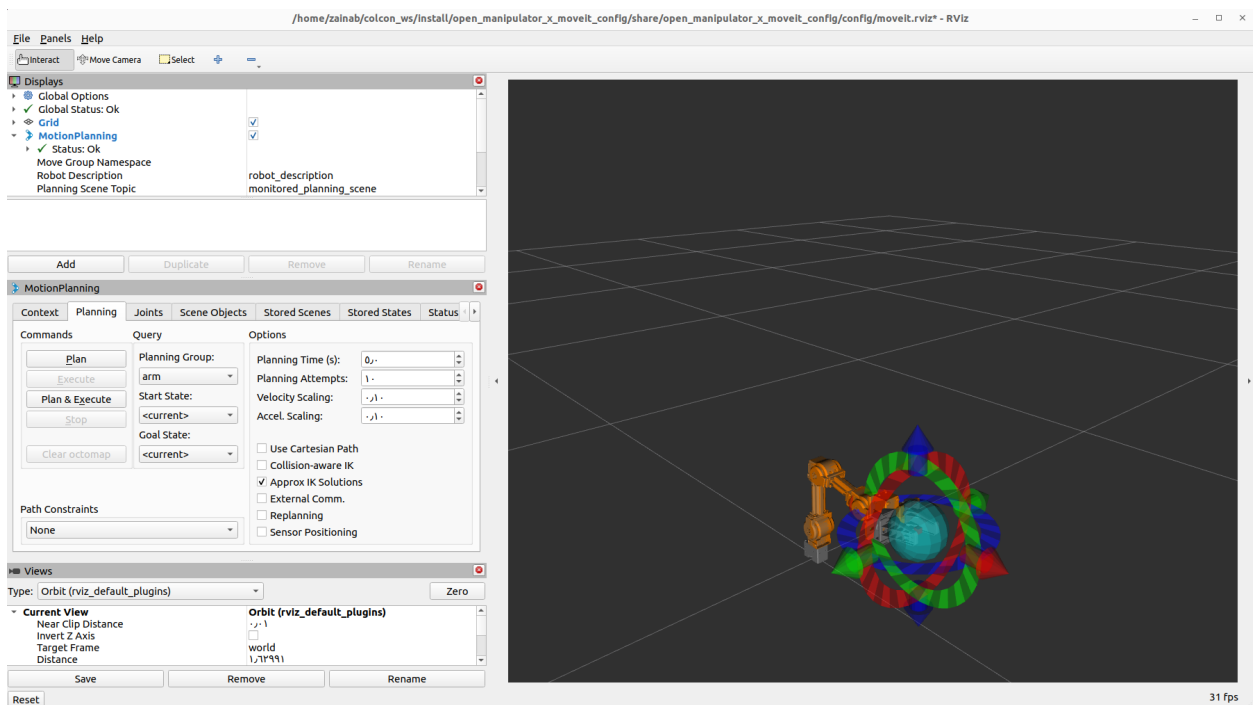
❖ Introduction to MoveIt & RViz

MoveIt is a powerful motion planning framework in ROS that allows you to control robotic arms and other robotic manipulators. It provides the following key features:

- **Motion Planning:** Calculates a sequence of movements from the robot's current to a goal position.
- **Collision Checking:** Ensures that the robot does not collide with its environment during movement
- **Inverse Kinematics (IK):** Solves for the joint configurations required to achieve a desired pose.
- **Execution:** Executes the motion plan by controlling the robot's actuators.

MoveIt is responsible for motion planning, controlling the manipulator, and handling tasks such as path planning, inverse kinematics (IK), and trajectory execution.

RViz is a visualization tool that provides a 3D interface where you can interact with the robot, visualize its movements, and plan trajectories using the MoveIt plugin for RViz.



In the following table we can see the key differences between Gazebo, Moveit, and RViz tools:

Feature	Gazebo	Moveit	RViz
Purpose	Robot simulation (physics and environment)	Motion planning and manipulation	Robot data visualization and interaction
Core Focus	Physics, environment, robot simulation	Path planning, motion execution, kinematics	Visualization of data, robot models, sensor outputs
3D Simulation	Yes (full 3D environment)	No (only motion paths and execution)	Yes (visualizes robot model and data in 3D)
Integration	Integrates with ROS for testing and simulation	Integrates with ROS for motion planning	Integrates with ROS for visualizing robot states
Physics Engine	Yes (includes collision, gravity, friction)	No (not a physics simulator)	No (visualization tool only)

❖ Exercise 1: Setting Up MoveIt for OpenManipulator-X

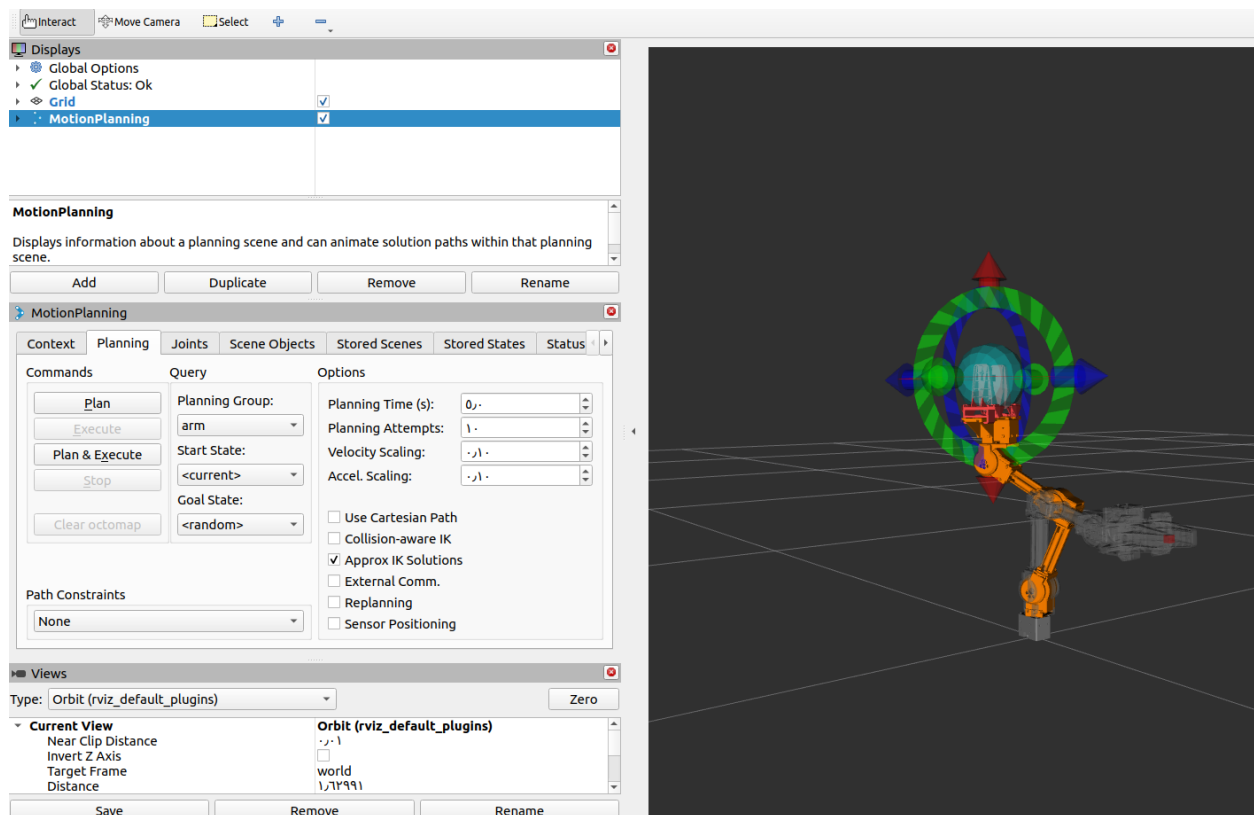
Step 1: Launch OpenManipulator-X on the Gazebo simulator (**Terminal #1**)

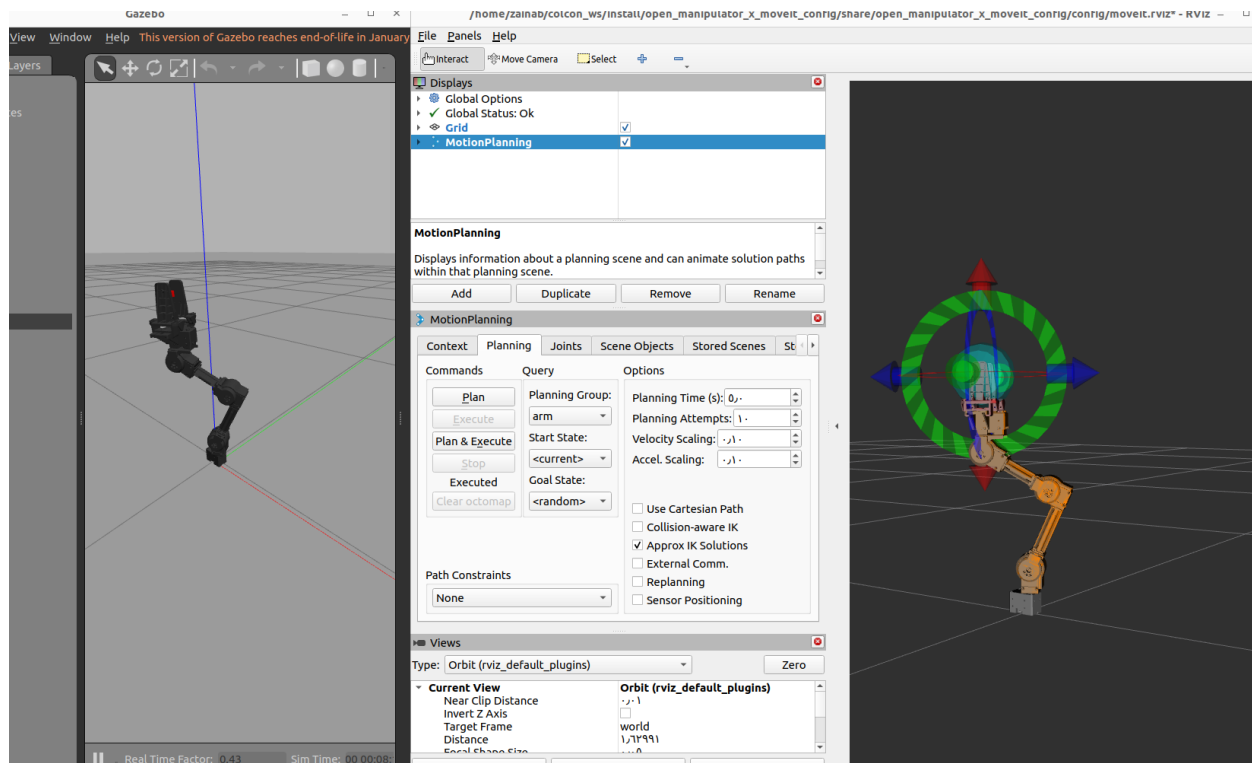
ros2 launch open_manipulator_x_bringup gazebo.launch.py

Step 2: Launch the open_manipulator_controller in Moveit & Rviz (**Terminal #2**)

ros2 launch open_manipulator_x_moveit_config moveit_core.launch.py

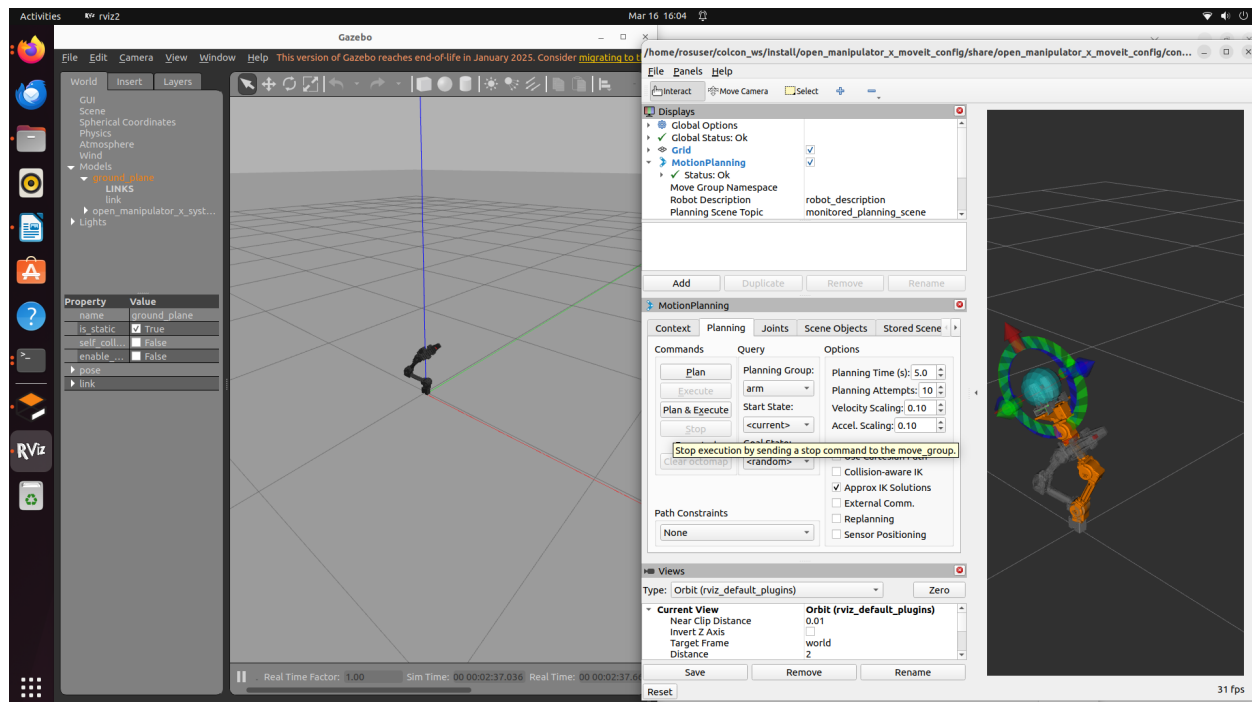
Step 3: Set the goal state to random and click on the Plan button in RViz. The MoveIt framework will compute a motion plan. Then, click Execute to perform the movement on the robot, and observe the results.



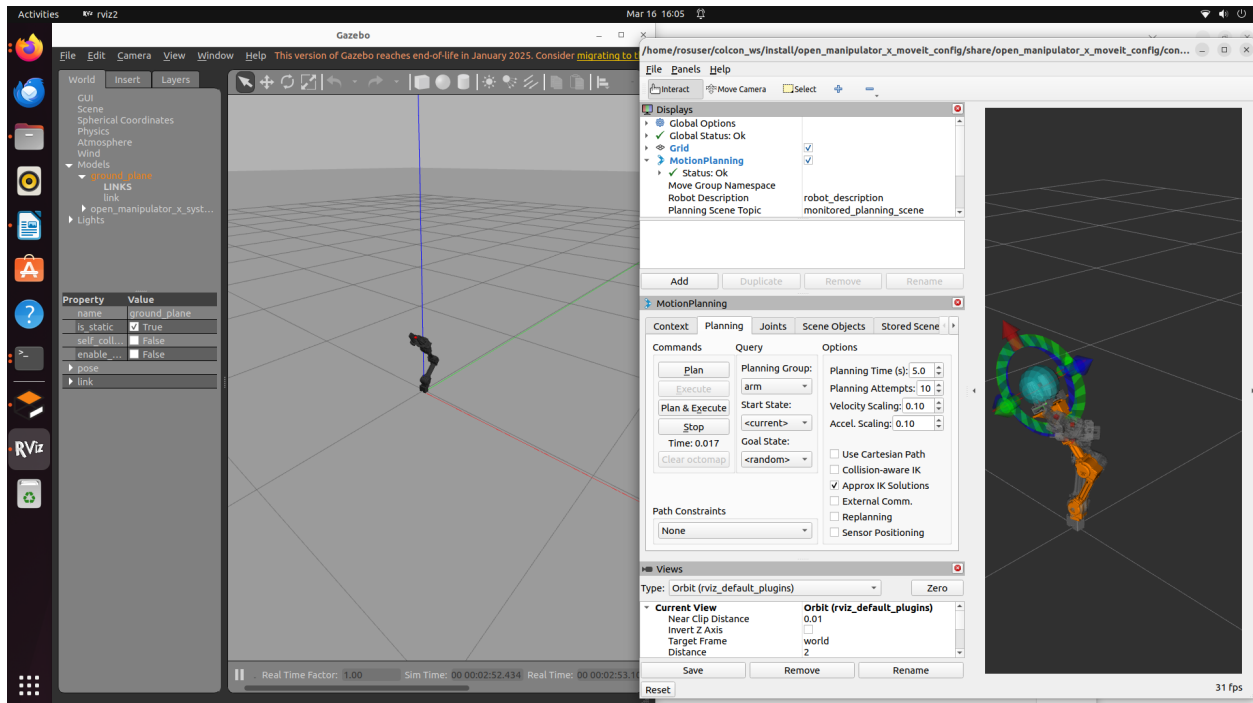


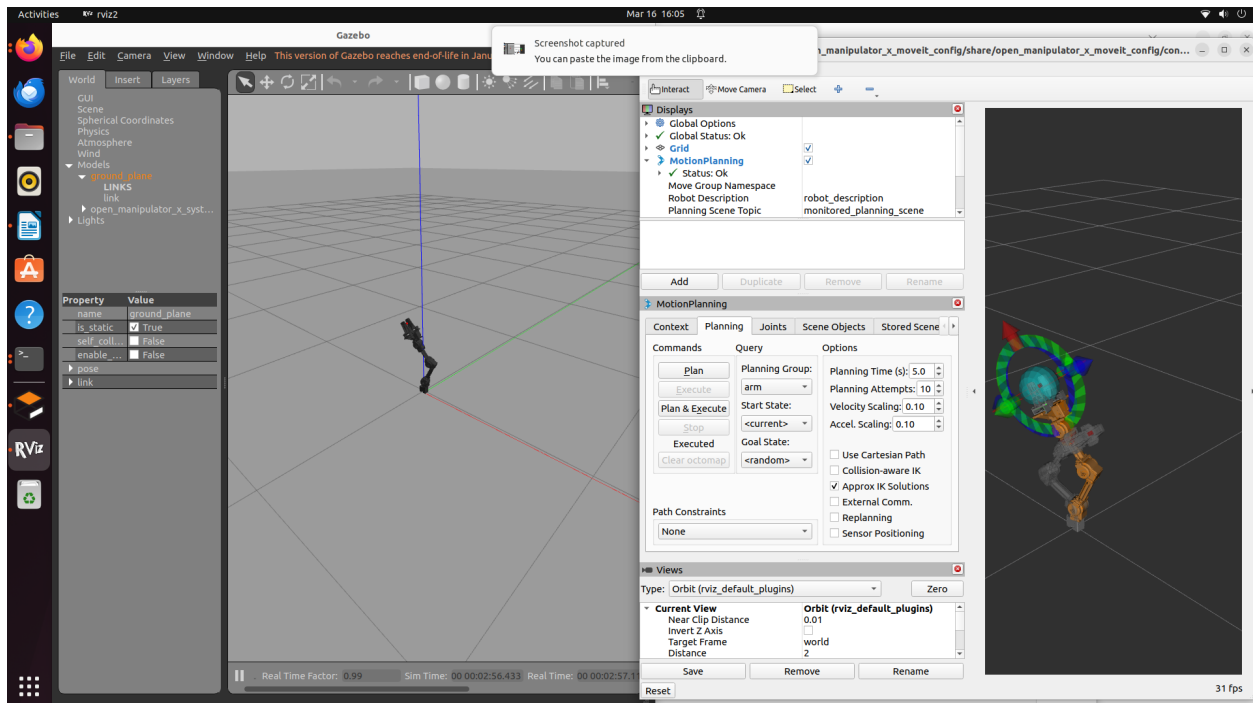
Step 4: try two random positions and observe the results

Put screenshots of your work here (for ALL the steps):



University of Prince Mugrin
College of Computer and Cyber Sciences
Artificial Intelligence Department
Abubakar waziri-4220056





❖ Exercise 2: Subscribing to Joint States

Step 1: Understanding the /joint_states Topic

The **/joint_states** topic provides real-time information about the robot's joint positions, velocities, and efforts. The data is published as messages of type **sensor_msgs/JointState**.

Each JointState message contains:

- position: A list of joint angles (in radians) that represent the current position of each joint.
- velocity: The rate of change of the joint positions (in radians per second).
- effort: The force or torque applied to the joints (in Newton-meters).

```
zainab@zainab-Lenovo-IdeaPad-S340-15IIL:~$ ros2 topic info /joint_states
Type: sensor_msgs/msg/JointState
Publisher count: 1
Subscription count: 2
zainab@zainab-Lenovo-IdeaPad-S340-15IIL:~$ ros2 interface show sensor_msgs/msg/JointState
# This is a message that holds data to describe the state of a set of torque controlled joints.
#
# The state of each joint (revolute or prismatic) is defined by:
# * the position of the joint (rad or m),
# * the velocity of the joint (rad/s or m/s) and
# * the effort that is applied in the joint (Nm or N).
#
# Each joint is uniquely identified by its name
# The header specifies the time at which the joint states were recorded. All the joint states
# in one message have to be recorded at the same time.
#
# This message consists of a multiple arrays, one for each part of the joint state.
# The goal is to make each of the fields optional. When e.g. your joints have no
# effort associated with them, you can leave the effort array empty.
#
# All arrays in this message should have the same size, or be empty.
# This is the only way to uniquely associate the joint name with the correct
# states.

std_msgs/Header header
  builtin_interfaces/Time stamp
    int32 sec
    uint32 nanosec
  string frame_id

string[] name
float64[] position
float64[] velocity
float64[] effort
```

In this exercise, you will write a simple ROS 2 node to subscribe to the `/joint_states` topic and log the joint positions in real-time.

Task: You'll create a new workspace with your name (e.g., `name_lab7_ws`), create a package, and then write the following python script, then run it using `ros2` command (please follow the same steps from previous lab to execute the code)

❖ **Start Gazebo and MoveIt:**

Launch the simulation environment and MoveIt to enable motion planning and control.

❖ **Run the Subscriber Node:**

In a separate terminal, execute your subscriber node to start listening to the /joint_states topic.

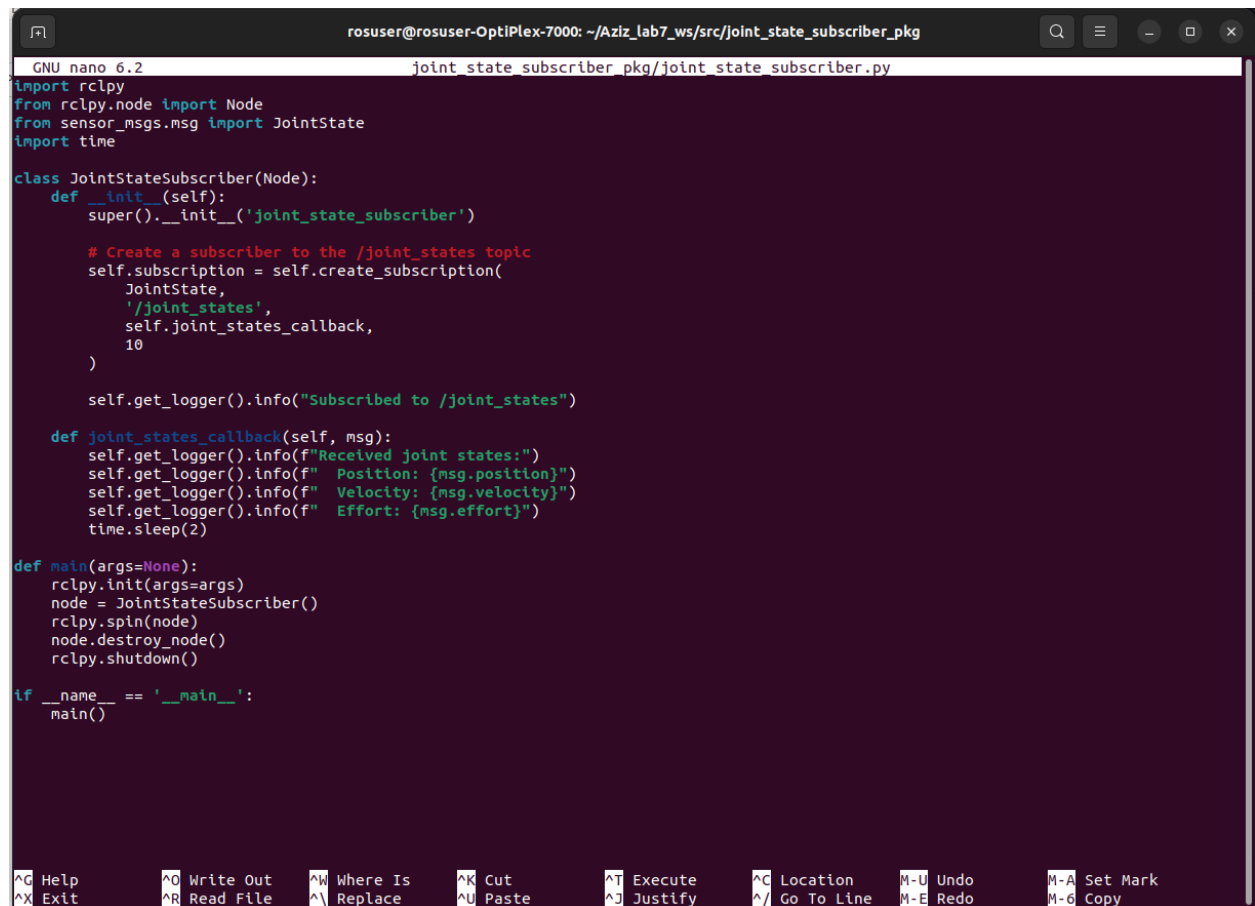
❖ **Move the Robot using MoveIt:**

Use the **Plan & Execute** feature in MoveIt to command the robot to a new position.

❖ **Observe the Subscriber Output:**

Monitor the terminal running the subscriber node to see the received joint states.

Put screenshots of your work here (for ALL the steps)::



```
rosuser@rosuser-OptiPlex-7000: ~/Aziz_lab7_ws/src/joint_state_subscriber_pkg
GNU nano 6.2 joint_state_subscriber_pkg/joint_state_subscriber.py
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
import time

class JointStateSubscriber(Node):
    def __init__(self):
        super().__init__('joint_state_subscriber')

        # Create a subscriber to the /joint_states topic
        self.subscription = self.create_subscription(
            JointState,
            '/joint_states',
            self.joint_states_callback,
            10
        )

        self.get_logger().info("Subscribed to /joint_states")

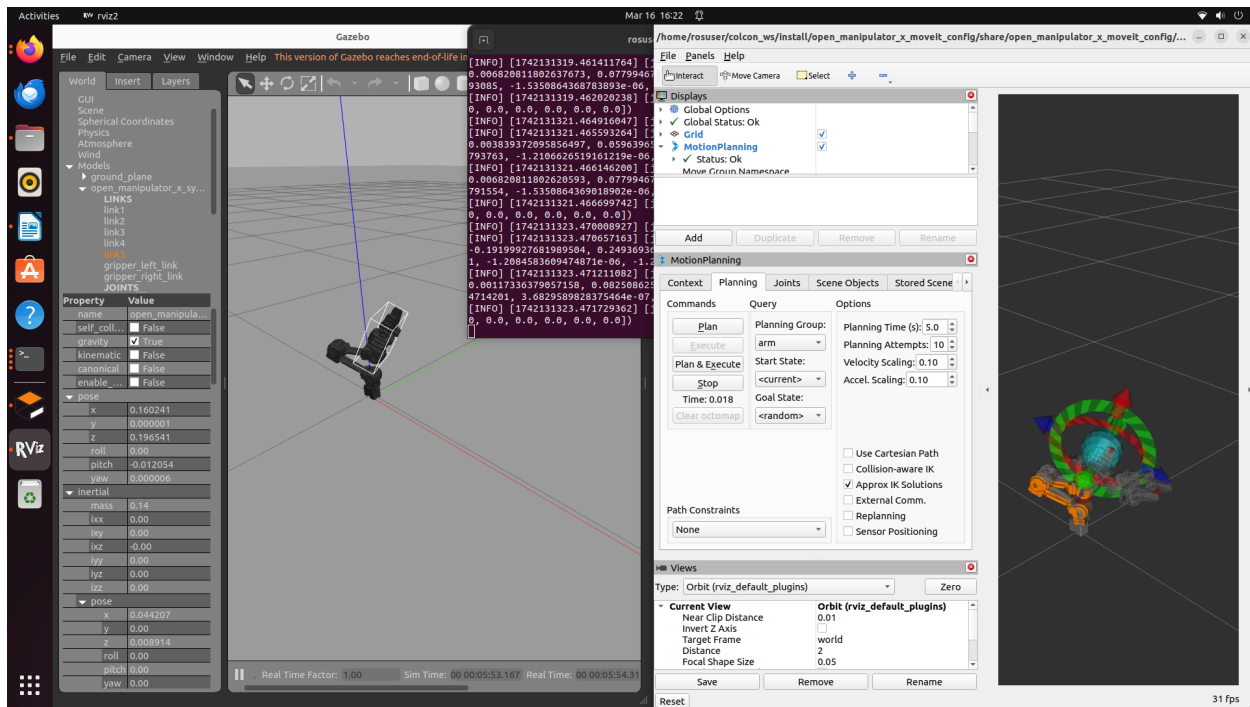
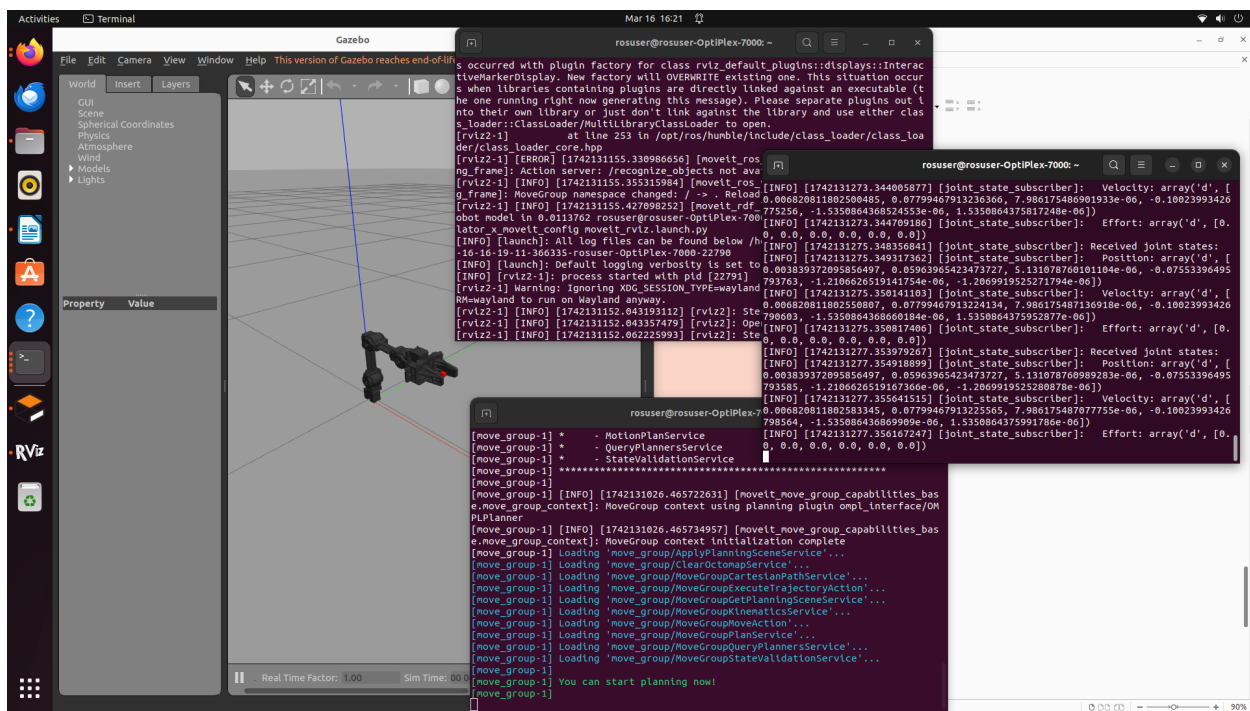
    def joint_states_callback(self, msg):
        self.get_logger().info(f"Received joint states:")
        self.get_logger().info(f"  Position: {msg.position}")
        self.get_logger().info(f"  Velocity: {msg.velocity}")
        self.get_logger().info(f"  Effort: {msg.effort}")
        time.sleep(2)

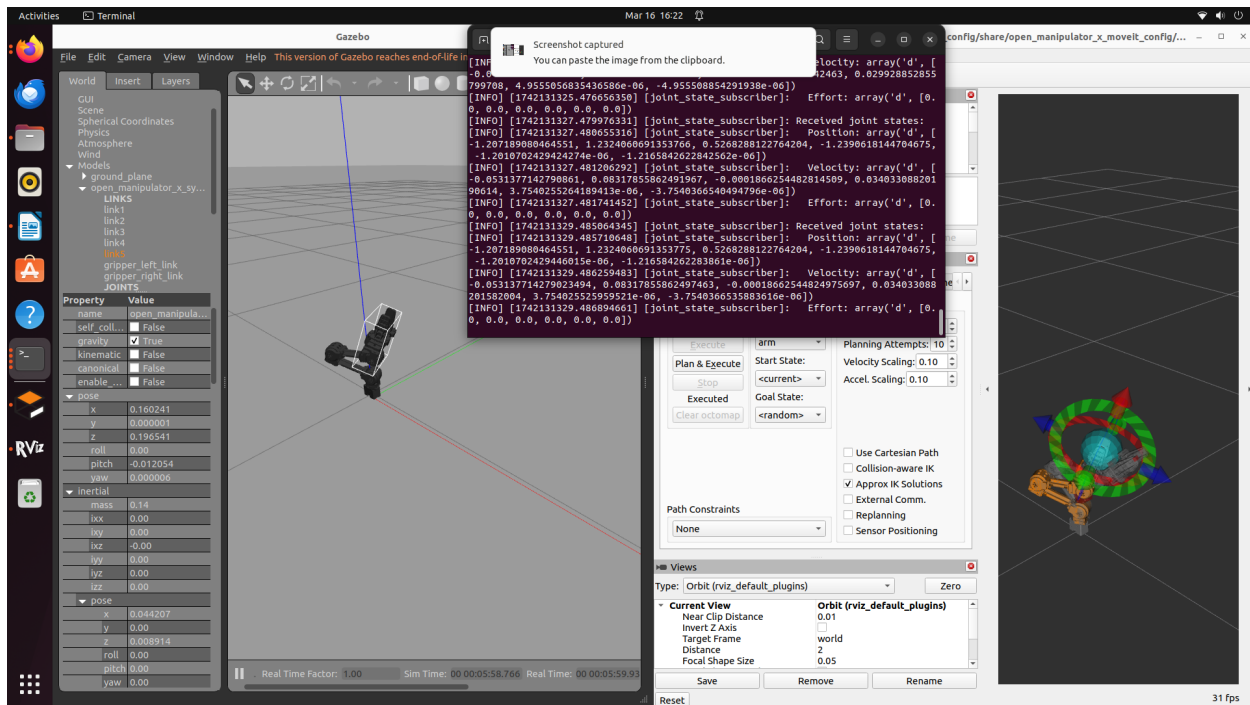
def main(args=None):
    rclpy.init(args=args)
    node = JointStateSubscriber()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^_ Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo M-G Copy

University of Prince Mugrin
College of Computer and Cyber Sciences
Artificial Intelligence Department
Abubakar waziri-4220056





❖ Lab activity:

Task1: Modify the Subscriber to Log Velocities and Efforts:

- Modify the code to log additional joint information like velocity and effort by accessing msg.velocity and msg.effort.
- Log both position, velocity, and effort in the callback function.

1. Provide the new added lines in the code:

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
import time

class JointStateSubscriber(Node):
    def __init__(self):
        super().__init__('joint_state_subscriber')

    # Create a subscriber to the /joint_states topic
    self.subscription = self.create_subscription(
        JointState,
        '/joint_states',
```

```
        self.joint_states_callback,  
        10  
    )  
  
    self.get_logger().info("Subscribed to /joint_states")  
  
def joint_states_callback(self, msg):  
    self.get_logger().info(f"Received joint states:")  
    self.get_logger().info(f" Position: {msg.position}")  
    self.get_logger().info(f" Velocity: {msg.velocity}")  
    self.get_logger().info(f" Effort: {msg.effort}")  
    time.sleep(2)  
  
def main(args=None):  
    rclpy.init(args=args)  
    node = JointStateSubscriber()  
    rclpy.spin(node)  
    node.destroy_node()  
    rclpy.shutdown()  
  
if __name__ == '__main__':  
    main()
```

2. Explain the role of each added line.

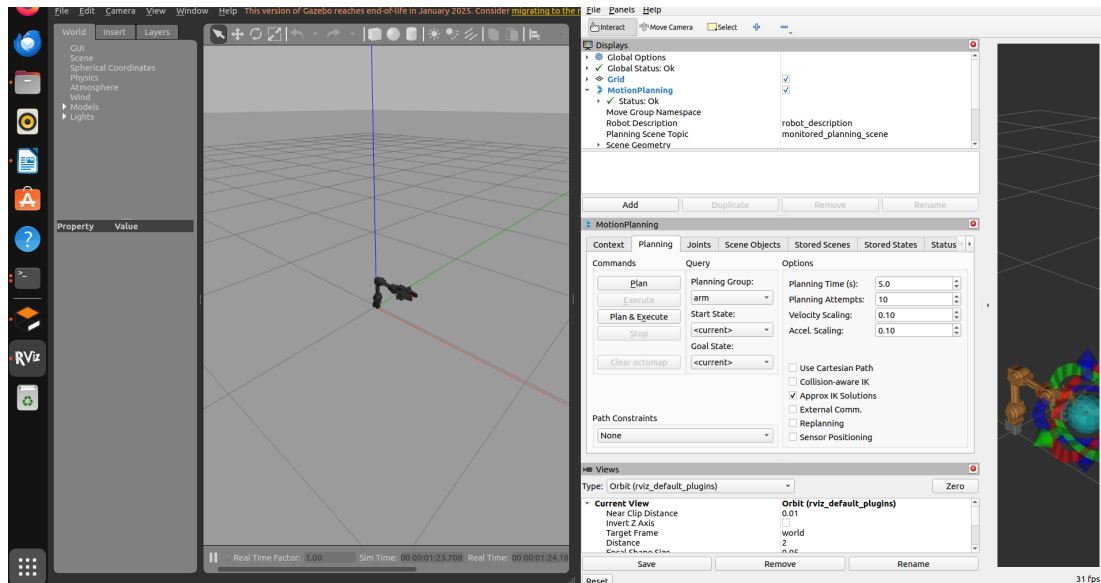
self.get_logger().info(f" Velocity: {msg.velocity}")

-
- This line logs the velocity of each joint.
 - The velocity is measured in radians per second (rad/s) for rotational joints and meters per second (m/s) for prismatic joints.
 - It helps in monitoring how fast each joint is moving in real-time.
-

self.get_logger().info(f" Effort: {msg.effort}")

-
- This line logs the effort (torque or force) applied to each joint.
 - The effort is measured in Newton-meters (Nm) for rotational joints and Newtons (N) for prismatic joints.
 - It is useful for analyzing how much force is being exerted on each joint, which can be important for torque-based control or detecting mechanical issues.
-

3. Provide screenshot of the terminal that's running the code, and the OpenManipulator-X gazebo window and moveit:



References

https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/