

*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.

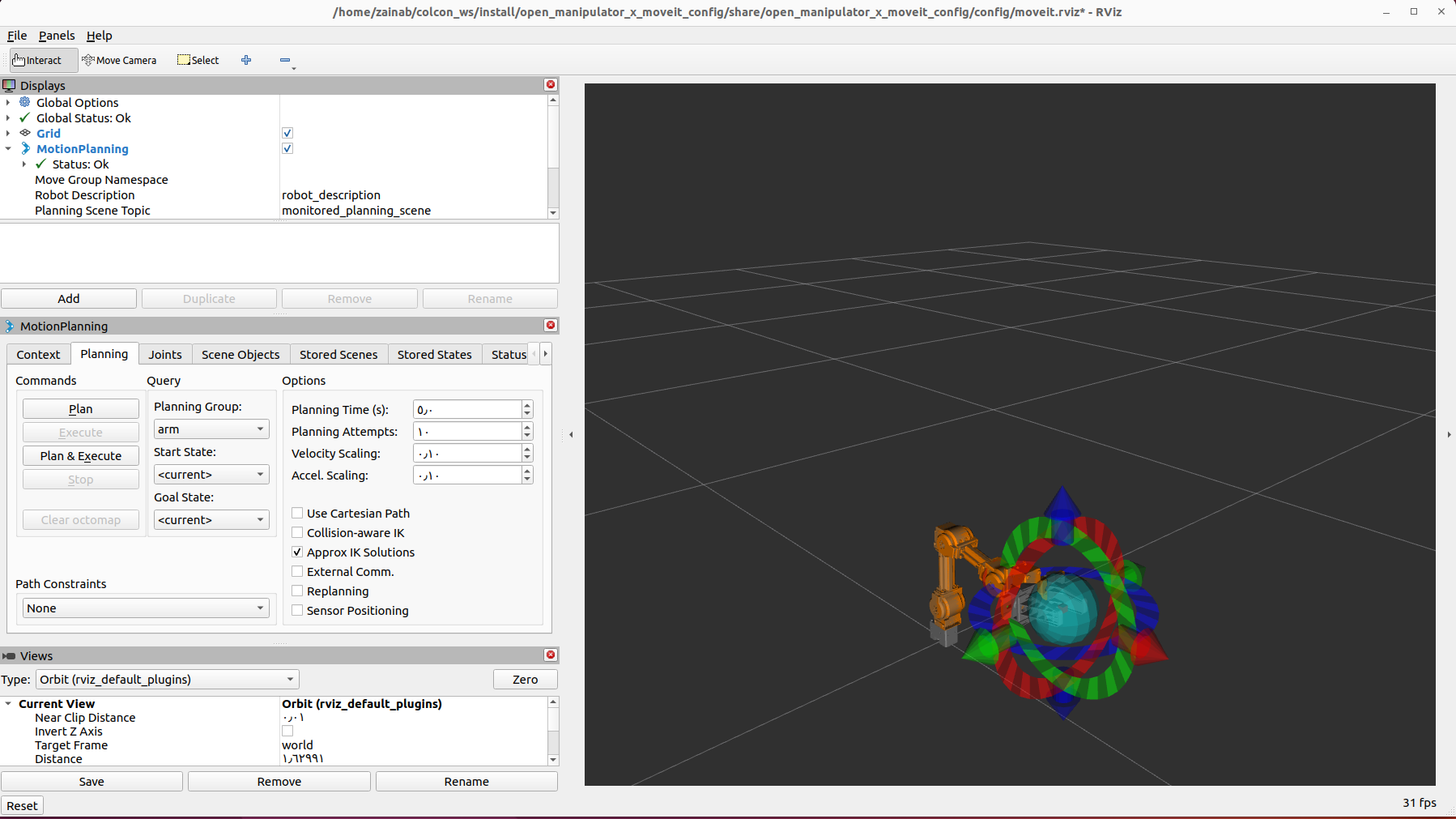
* **Introduction to Moveit & RViz**

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**

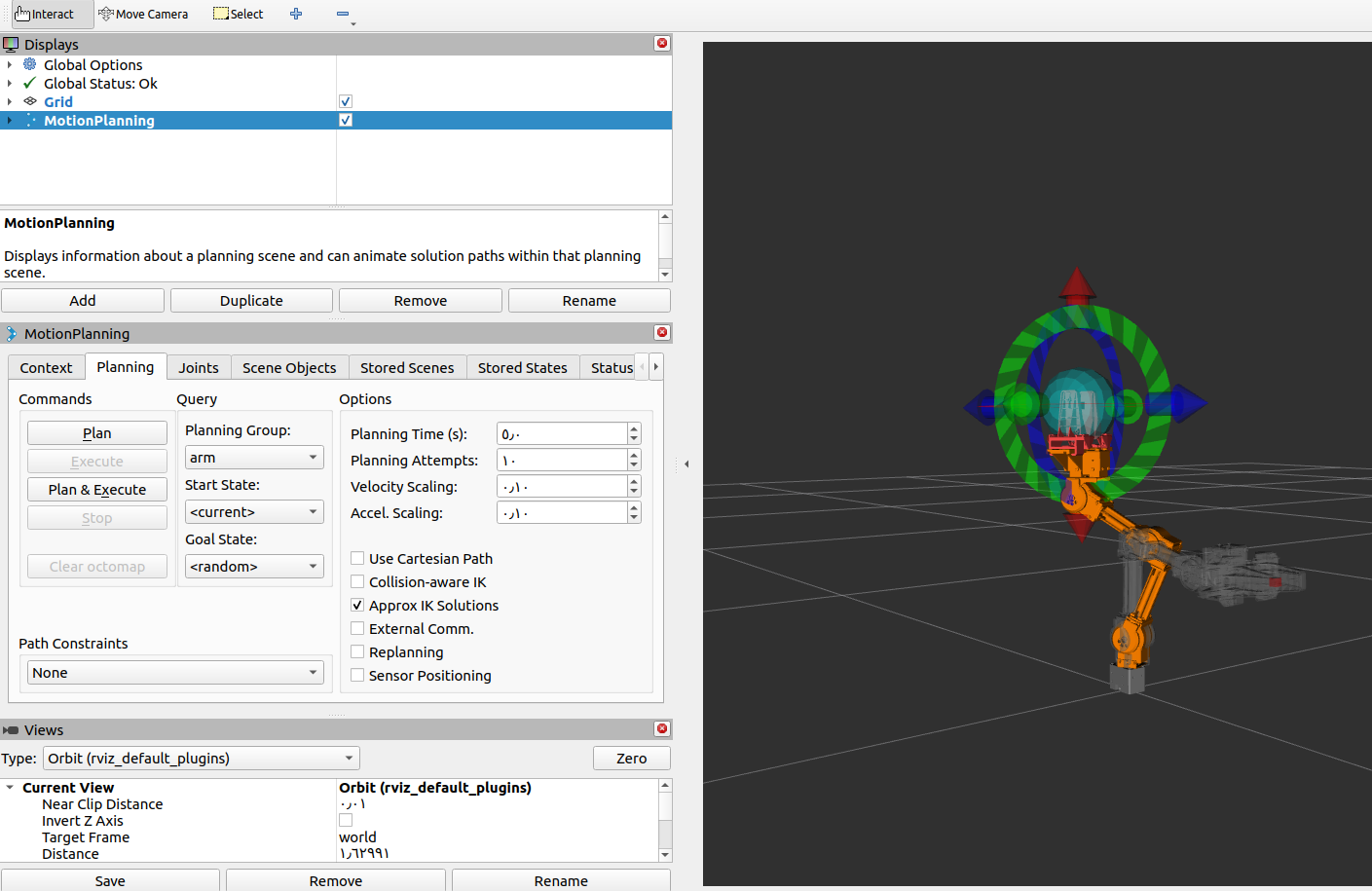
**Step 1: Launch OpenManipulator-X on the Gazebo simulator (Terminal #1)**

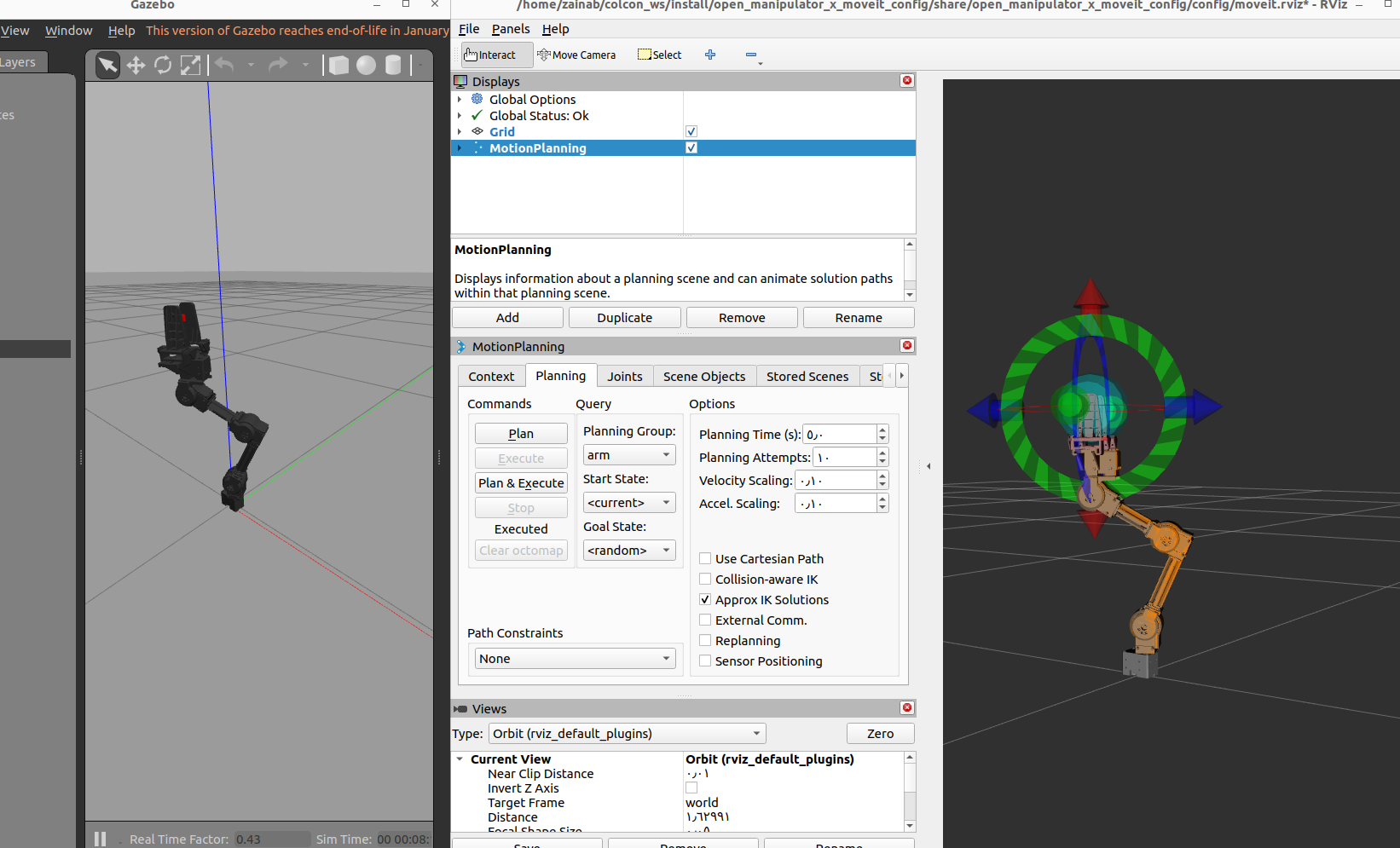
***ros2 launch open\_manipulator\_x\_bringup gazebo.launch.py***

**Step 2: Launch the open\_manipulator\_controller in Moveit & Rviz** **(Terminal #2)**

***ros2 launch open\_manipulator\_x\_moveit\_config moveit\_core.launch.py***

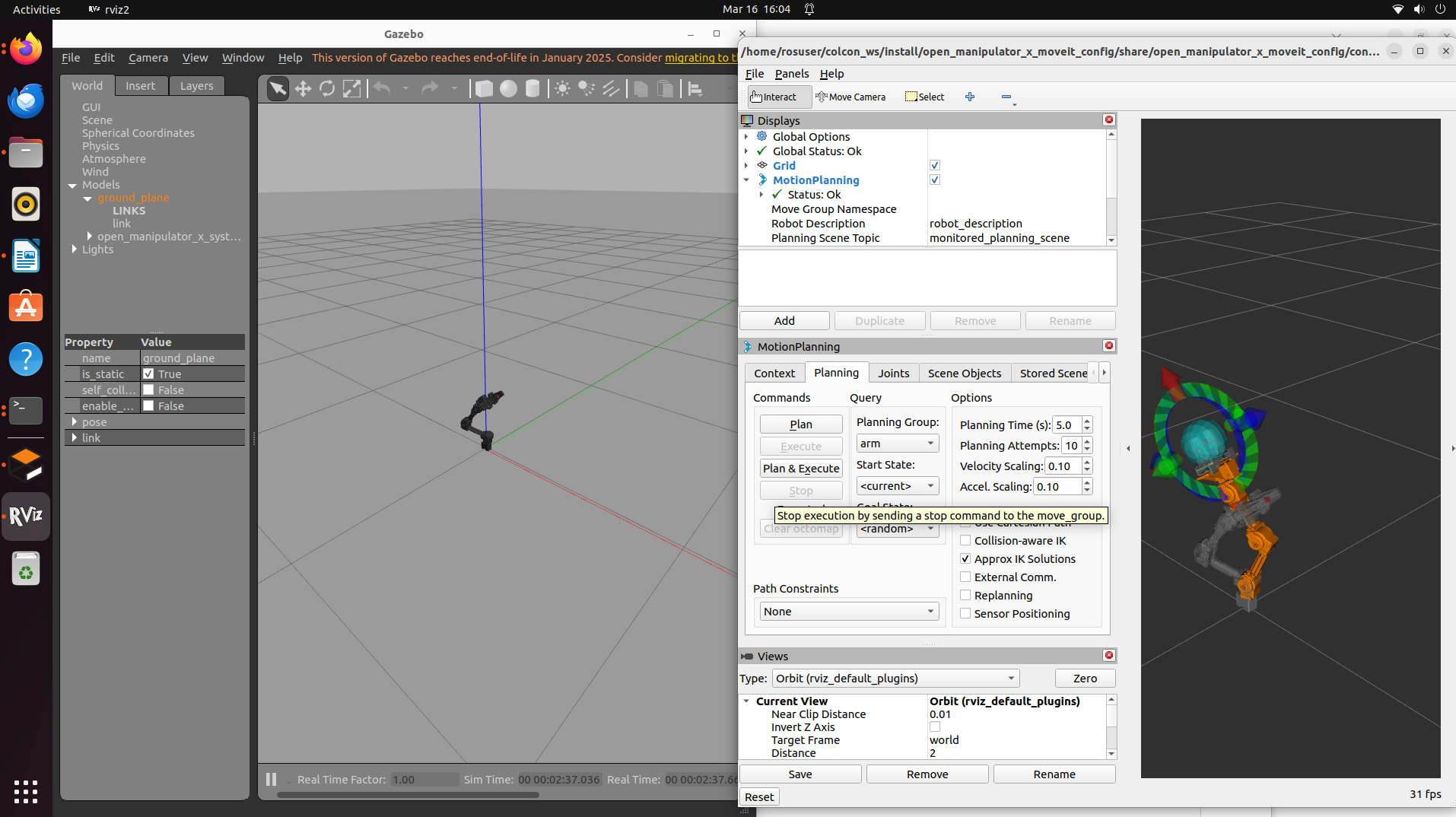
**Step 3:** **Set the goal state to random and click on the Plan button in RViz. The MoveIt framework will compute a motion plan. Then, click Execute to perform the movement on the robot, and observe the results.**

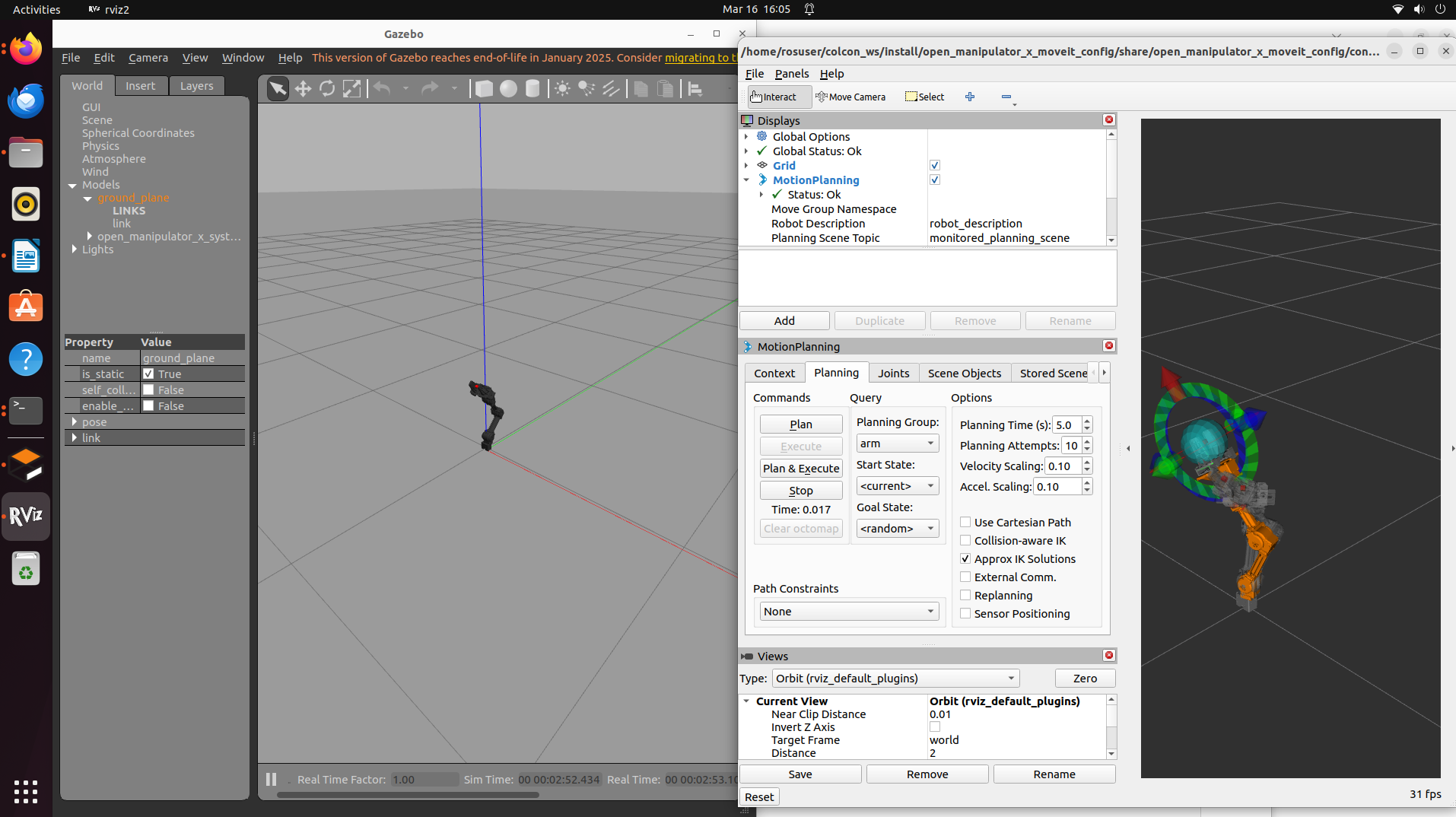


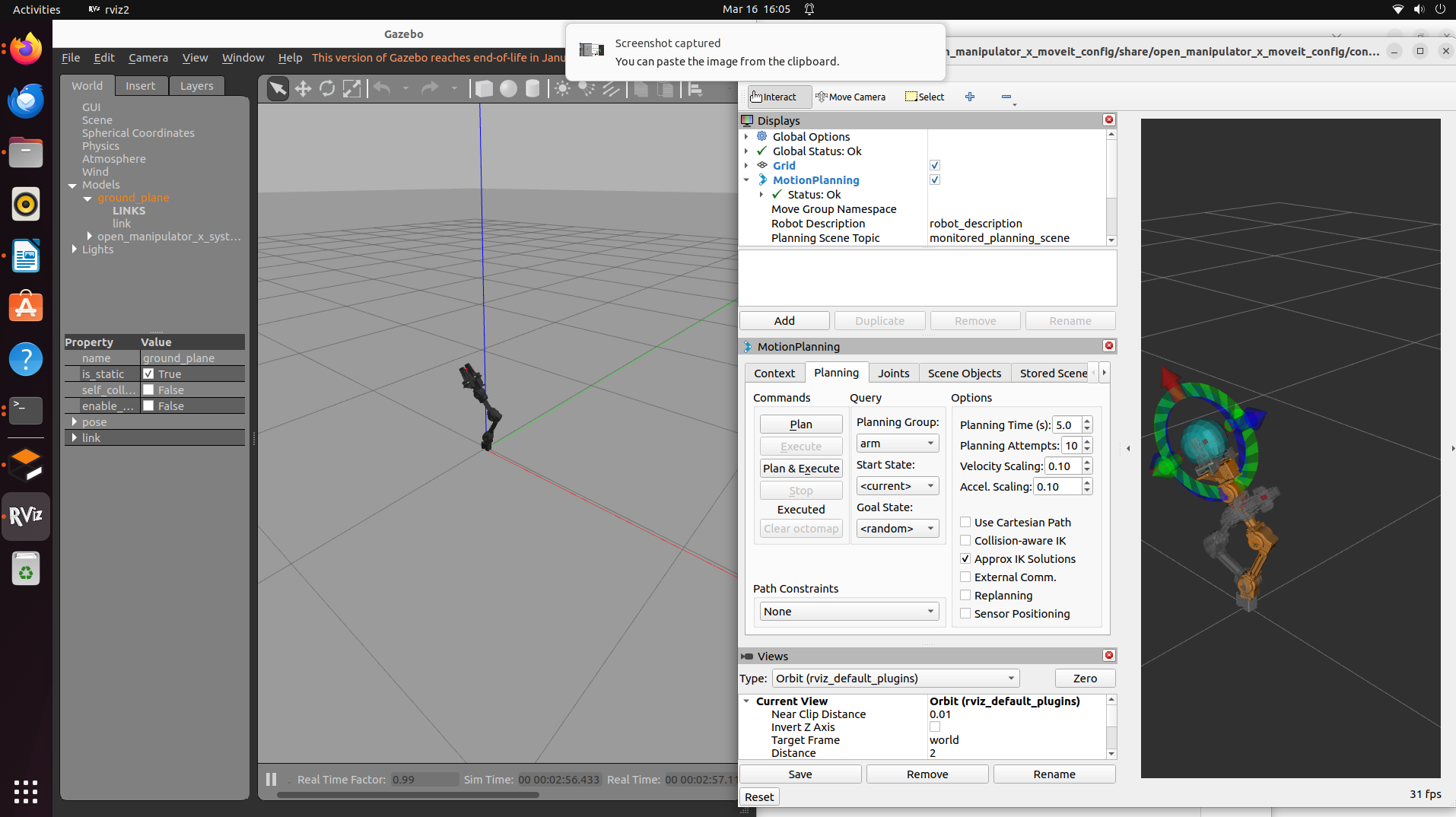


**Step 4:** **try two random positions and observe the results**

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







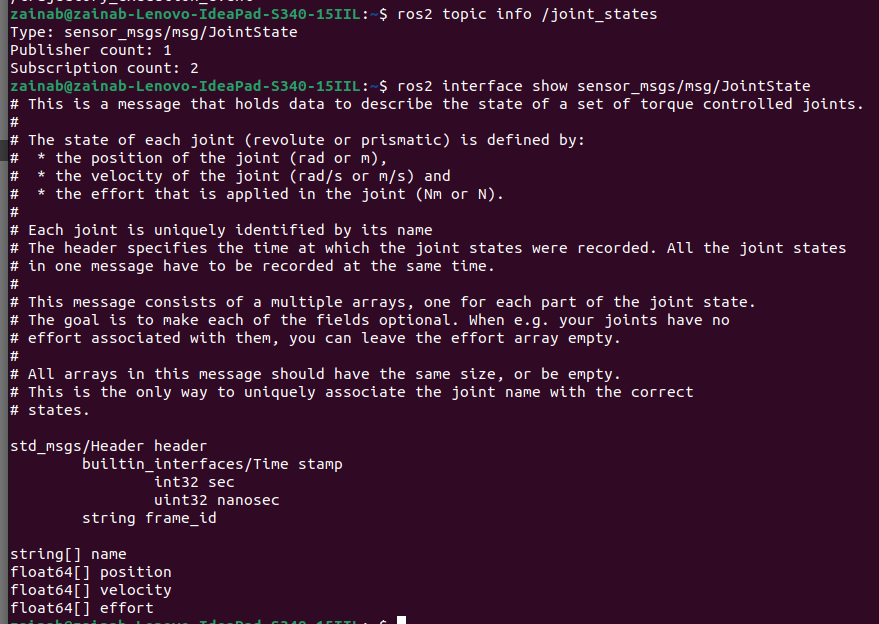
* **Exercise 2: Subscribing to Joint States**

**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).



**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 the 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.

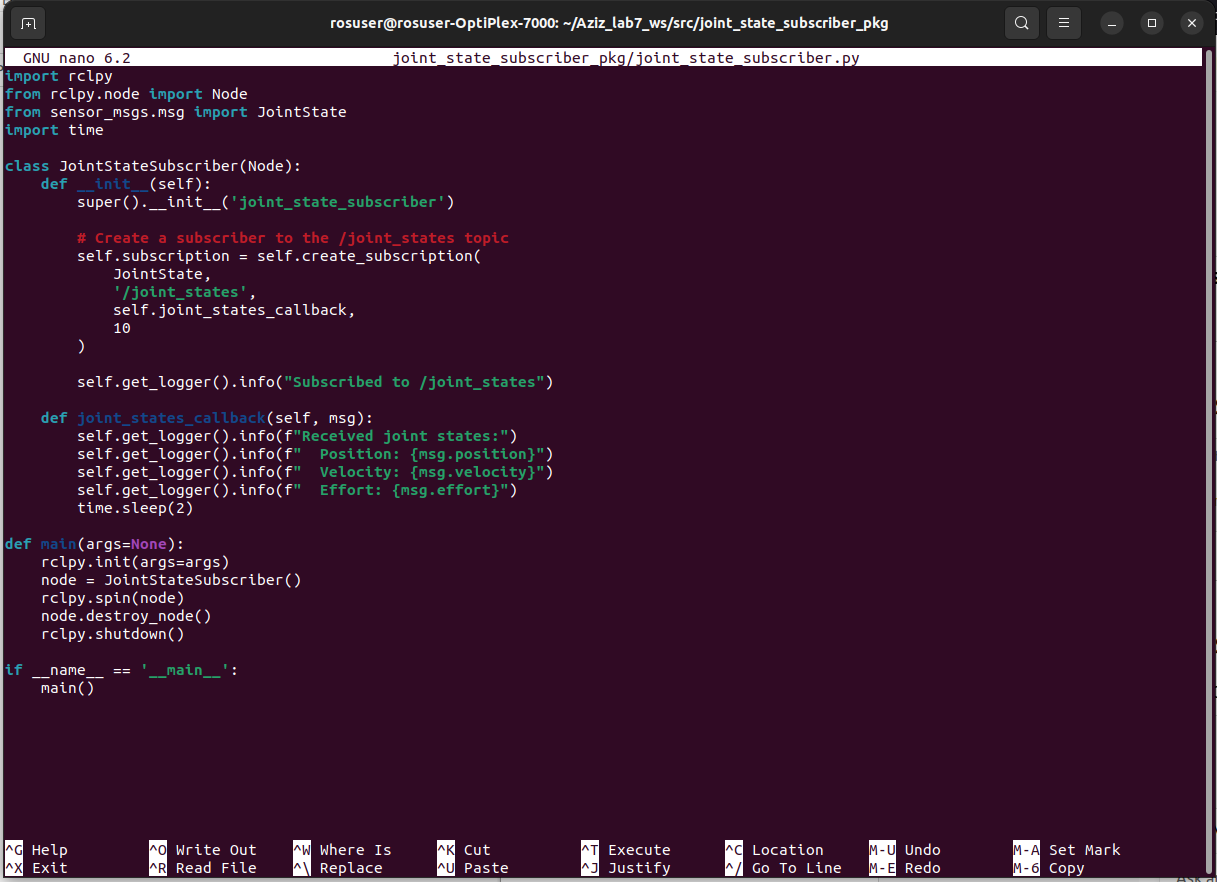
* **Move the Robot using MoveIt**:

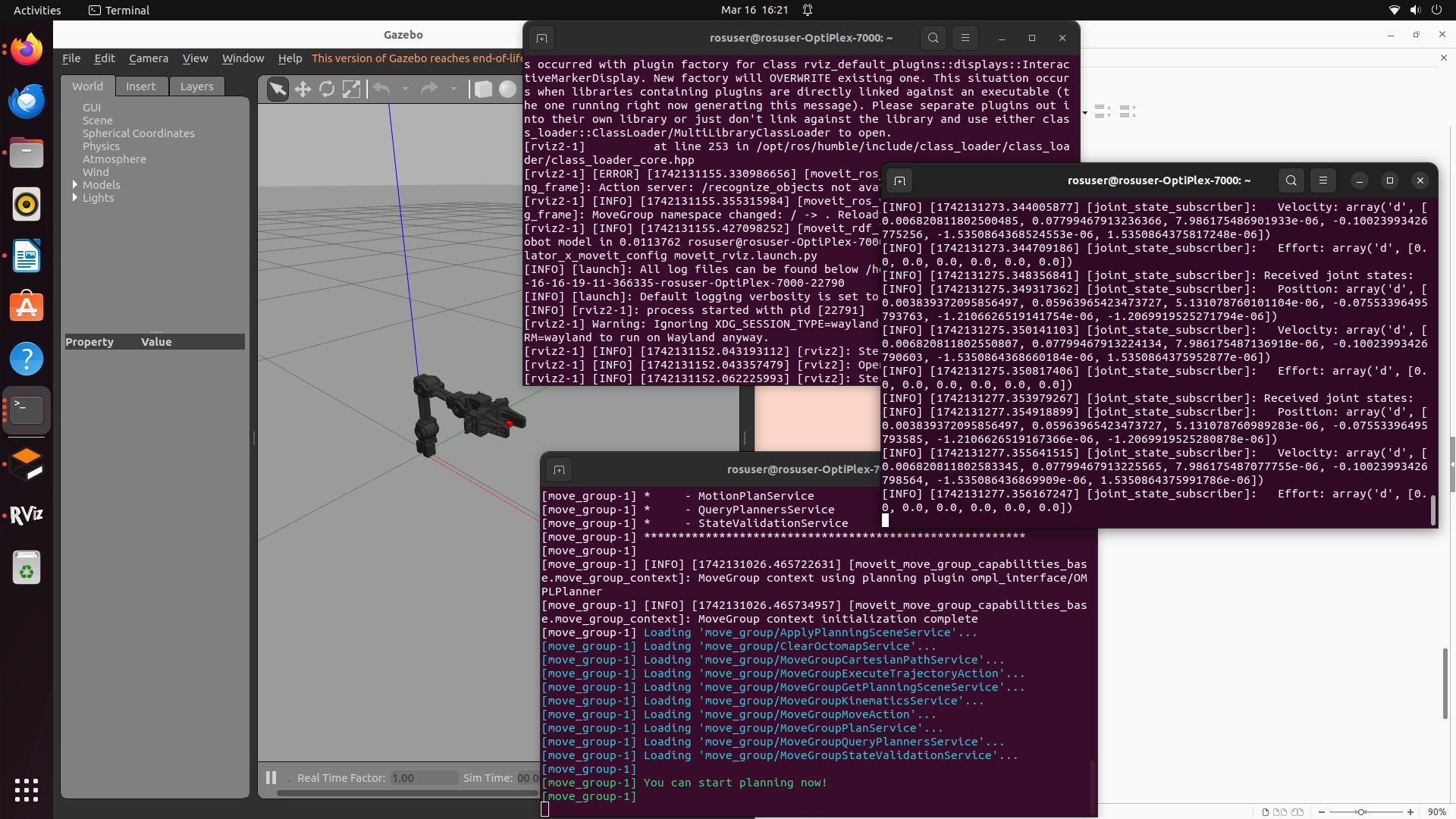
Use the **Plan & Execute** feature in MoveIt to command the robot to a new position.

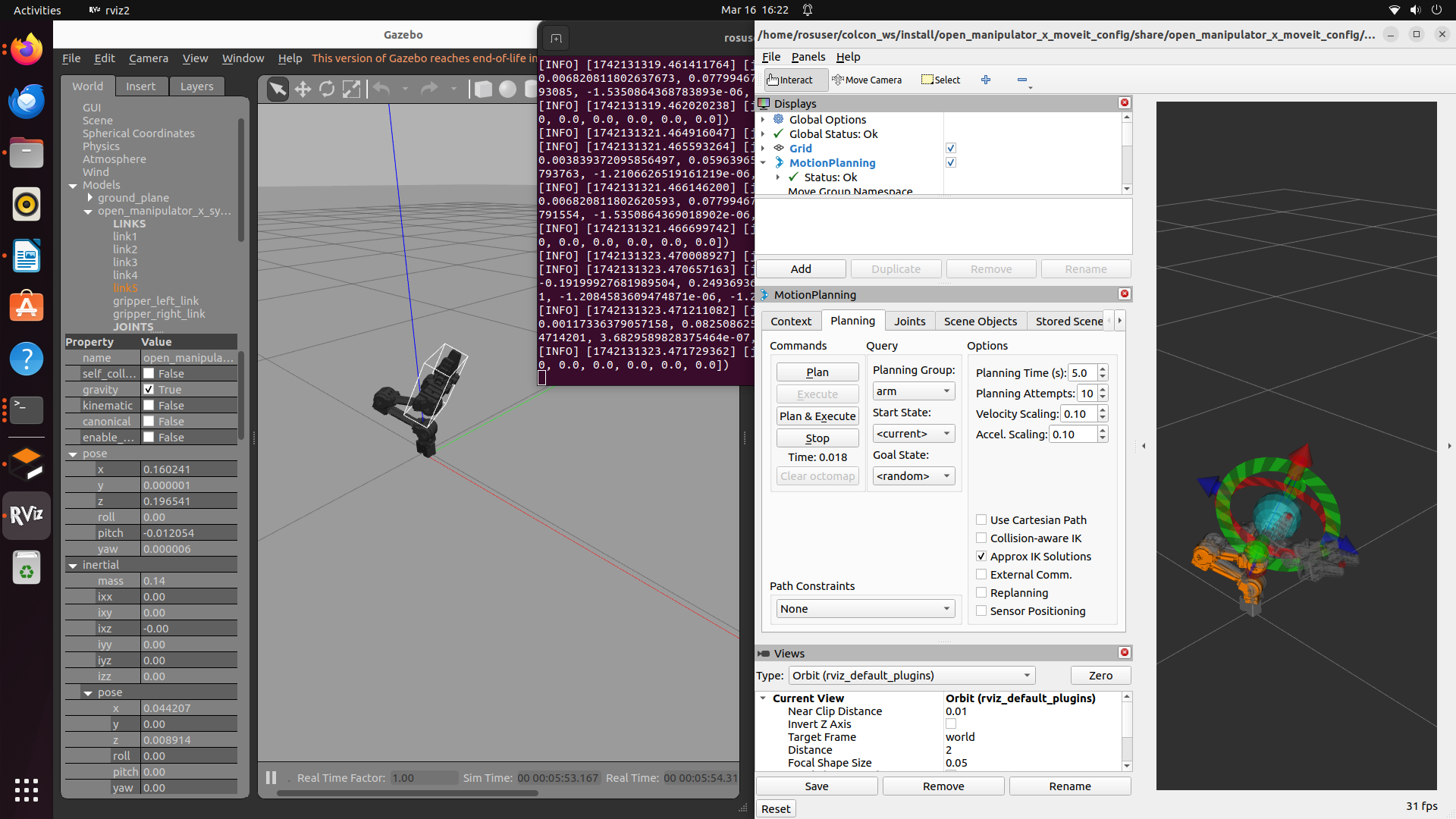
* **Observe the Subscriber Output**:

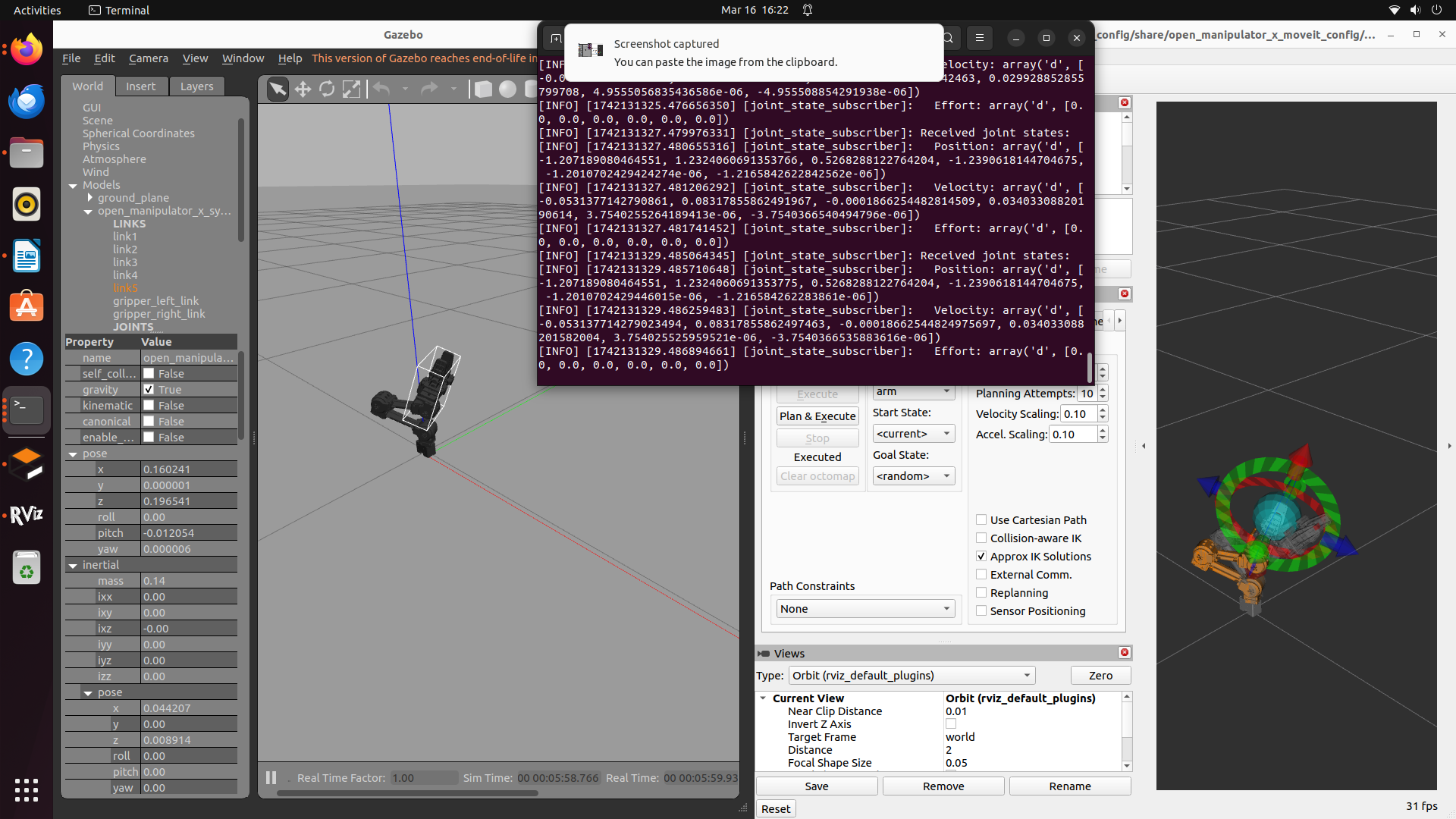
Monitor the terminal running the subscriber node to see the received joint states.

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

****

****

****

****

* **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()

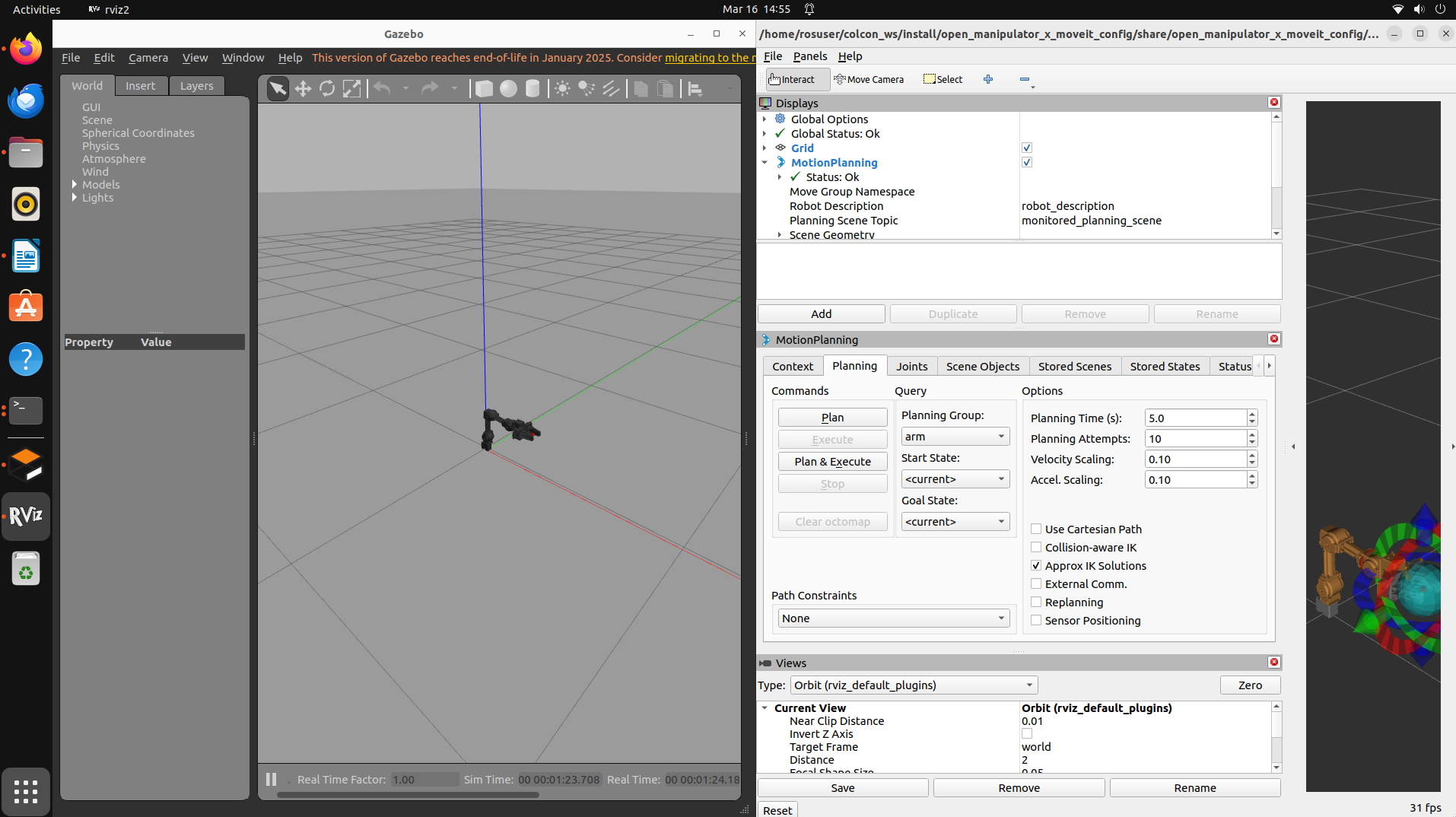
1. Explain the role of each added line.

**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/**](https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/)