**Department of Electrical Engineering**

**Faculty Member:** Dr. Hafsa Iqbal **Date:** 12/03/24

**Semester:** 8th  **Group:** 01

# Lab Engineer: Munadi Ahmad Sial

# EE381 Robotics

Lab 6: Custom Teleoperation, Launch Files and Plotting

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 – CLO4** | **PLO5 - CLO5** | | **PLO8-CLO6** | **PLO9-CLO7** |
| **Name** | **Reg. No** | **Viva/Quiz/Demo** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Hassan Rizwan | 335753 |  |  |  |  |  |
| Muhammad Abdullah Sohail | 343642 |  |  |  |  |  |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
| Muhammad Saad | 333414 |  |  |  |  |  |

# Table of Contents

[**1. Table of Contents 2**](#_Toc161624756)

[**2. Custom Teleoperation, Launch Files and Plotting 3**](#_Toc161624757)

[2.1 Introduction 3](#_Toc161624758)

[2.2 Objectives 3](#_Toc161624759)

[2.3 Lab Conduct 3](#_Toc161624760)

[2.4 Theory 3](#_Toc161624761)

[**3. Lab Tasks 4**](#_Toc161624762)

[3.1 Lab Task 1 - Publish and Subscribe Simultaneously 4](#_Toc161624763)

[3.2 Lab Task 2 – Custom Teleoperation 7](#_Toc161624764)

[3.3 Lab Task 3 – Launch Files 11](#_Toc161624765)

[3.4 Lab Task 4 – RQt Plot 12](#_Toc161624766)

[4. **Conclusion** 13](#_Toc161624767)

# Custom Teleoperation, Launch Files and Plotting

## Introduction

This laboratory exercise is focused on creating a custom teleoperation program for a robot simulation. This lab will extend the concepts of node communication to a more practical situation. Creating a custom teleoperation program will provide useful insights into ROS programming. Additionally, this laboratory exercise is focused on creating launch files which are used to start ROS programs simultaneously. The final portion of this lab will involve real-time plotting with the RQt interface in ROS.

## Objectives

The following are the main objectives of this lab:

* Create a node that can publish and subscribe simultaneously
* Create a custom teleoperation program for a simulated robot
* Create and execute a Launch File
* Plot graphs of messages on a topic

## Lab Conduct

* Respect faculty and peers through speech and actions
* The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
* In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code between the #START and #END parts of these commented lines. Do NOT remove the commented lines.
* Use the tab key to provide the indentation in python.
* When you provide the code in the report, keep the font size at 12

## Theory

Teleoperation is the movement of a robot from one place to another via a user input. Teleoperation is useful even in autonomous robots to control them should they malfunction during tests. A teleoperation program works by detecting keystrokes and publishing twist messages. In a practical robot, ROS graphs can consist hundreds of nodes. Executing such nodes one-by-one is not convenient so instead these nodes are *launched* at the same time with the help of launch files. A launch file contains the nodes, parameters and models etc. that can be executed all at once.

A brief summary of the terminal commands needed for working in ROS are provided below:

* **colcon build**

build the packages whenever a node is created or modified

* **colcon build --packages-select <package\_name>**

build a specific package whenever a node is created or modified

* **. install/setup.bash**

make the terminal “aware” of the workspace (notice the dot and the space)

* **ros2 pkg create --build-type ament\_python <package\_name>**

create a new package (must be done in src directory)

* **ros2 pkg create --build-type ament\_python <package\_name> --dependencies rclpy std\_msgs geometry\_msgs**

create a new package with dependencies

* **ros2 run <package\_name> <node\_name>**

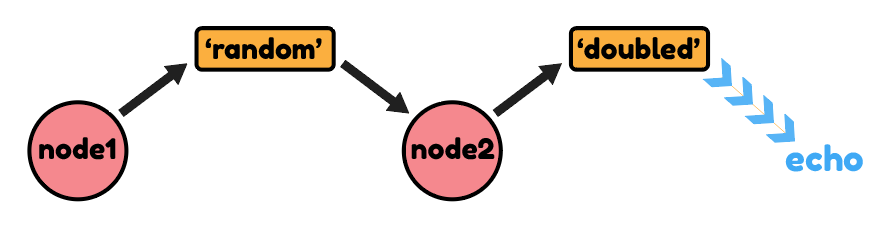
execute a node

In the tasks where the terminal is shown (screenshot or video), ensure that the student names are shown on the terminal otherwise marks may be deducted.

# Lab Tasks

## Lab Task 1 - Publish and Subscribe Simultaneously

This task will focus on creating the following node graph.



In the above graph, there are 2 nodes and 2 topics. Node 1 generates random numbers from 1 to 30 which are published to a ‘random’ topic. Node 2 subscribes to this ‘random’ topic to get the random numbers. It then doubles the received numbers and publishes them to a ‘doubled’ topic. The values on the ‘doubled’ topic are echoed to a new terminal.

Node 2 is notable in that it both subscribes and publishes. To make such a node, you can use both **create\_subscription** and **create\_publisher** functions in the node constructor. The timer callback is not needed in this node as we want to publish only when the random number is received (and not in regular intervals). The subscriber (listener) callback function is retained in the node and the doubled values are published *from within* the subscriber callback function.

Create a package (name of the package must contain a name of one group member) in the robot workspace. Then, create node1 and node2 in this package. Use Int32 messages from std\_messages. For task submission, you need to run 3 terminals. The first terminal runs node1 (showing random numbers), the second terminal runs node 2 (showing doubled values) and the third terminal echoes the values in the ‘doubled’ topic. To echo, use the given command:

**ros2 topic echo <topic\_name>**

Provide the class codes for both nodes and a SINGLE screenshot which shows the three terminals working together.

**### NODE 1 CLASS CODE STARTS HERE ###**

import random

import rclpy

from rclpy.node import Node

from std\_msgs.msg import Int32

# Class

class Publisher(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("node\_1")

        self.publisher = self.create\_publisher(Int32, "random", 10)

        timer\_period = 1

        self.timer = self.create\_timer(timer\_period, self.timer\_callback)

    def timer\_callback(self):

        msg = Int32()

        rand\_num = random.randint(1, 30)

        msg.data = rand\_num

        self.publisher.publish(msg)

        self.get\_logger().info("Node 1 published " + str(msg.data))

def main(args=None):

    rclpy.init(args=args)

    simple\_publisher = Publisher()

    rclpy.spin(simple\_publisher)

    simple\_publisher.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**### NODE 1 CLASS CODE ENDS HERE ###**

**### NODE 2 CLASS CODE STARTS HERE ###**

import rclpy

from rclpy.node import Node

from std\_msgs.msg import Int32

# Class

class Subscriber(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("node\_2")

        self.subscription = self.create\_subscription(

            Int32, "random", self.listener\_callback, 10

        )

        self.publisher = self.create\_publisher(Int32, "doubled", 10)

    def listener\_callback(self, msg):

        val = msg.data \* 2

        self.get\_logger().info(f"Node 2 received " + str(val))

        msg = Int32()

        msg.data = val

        self.publisher.publish(msg)

def main(args=None):

    rclpy.init(args=args)

    simple\_subscriber = Subscriber()

    rclpy.spin(simple\_subscriber)

    simple\_subscriber.destroy\_node()

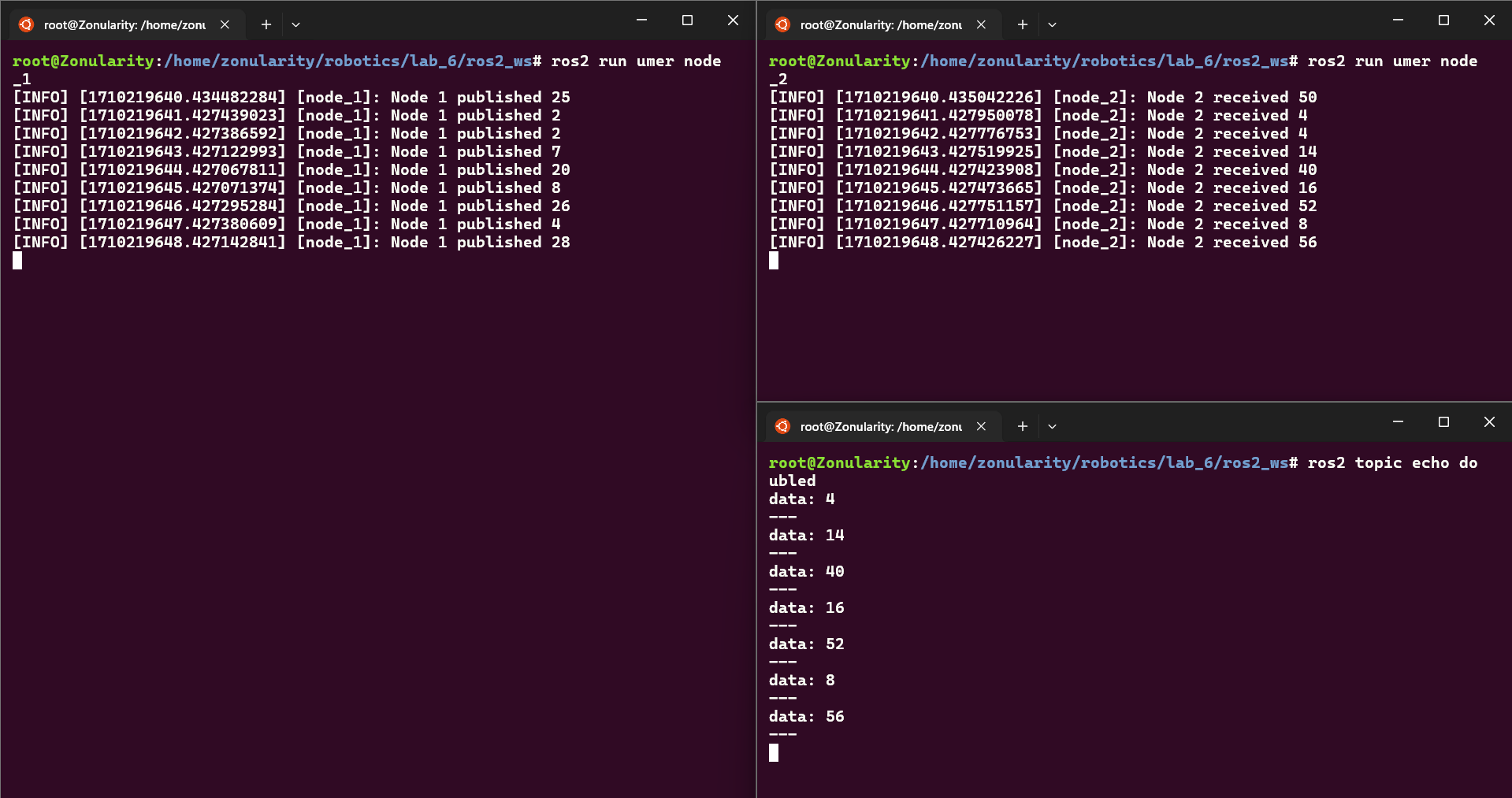
    rclpy.shutdown()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**### NODE 2 CLASS CODE ENDS HERE ###**

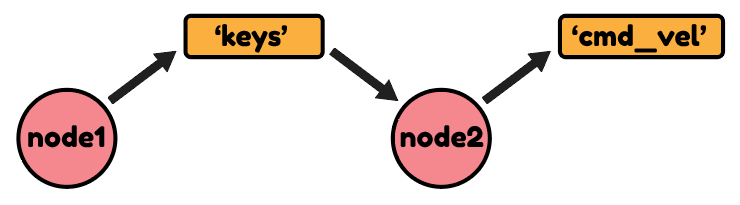
**### TASK 1 SCREENSHOT STARTS HERE ###**



**### TASK 1 SCREENSHOT ENDS HERE ###**

## Lab Task 2 – Custom Teleoperation

This task will focus on creating and implementing the following node graph.



In this task, you will use your understanding from task 1 to build a custom teleoperation program for the robot simulation. Create 2 nodes called key\_pub.py and keys\_to\_twist.py in your package (Do not alter the nodes from the previous task as you will need them again in task 3).

The first node (key\_pub.py) will detect keystrokes from user and publish the string of the pressed key to a ‘keys’ topic. To detect the keystrokes, use the following libraries in the node:

**import sys, termios, tty, select**

Next, you need to make some changes to the main function. In the main function, add the following statements after the object initialization and before the node is set to spin:

**old\_attr = termios.tcgetattr(sys.stdin)**

**tty.setcbreak(sys.stdin.fileno())**

Add another statement after the node spin (and before the destroy node statement):

**termios.tcsetattr(sys.stdin,termios.TCSADRAIN,old\_attr)**

In the node class for key\_pub, add a timer callback function (to regularly check for keystrokes). In the timer, use the following statements to record the pressed key:

**if select.select([sys.stdin],[],[],0)[0] == [sys.stdin]:**

**msg.data = sys.stdin.read(1)**

The message can then be published to the ‘keys’ topic which expects a String data (from std\_msgs.msg).

The second node (keys\_to\_twist.py) subscribes to the ‘keys’ topic and depending on which key is pressed, it will publish a twist message to the robot to change its motion. The second node is both a subscriber and a publisher (recall task 1). Use the following mapping for the key presses:

w --> move forward at speed 1.0

s --> move backward at speed 1.0

a --> turn left at speed 1.0

d --> turn right at speed 1.0

q --> stop moving or turning

For this task, you need provide the codes for both of the nodes. You also need to make a video called lab6\_custom\_teleop. The video must show both the nodes running (in their own terminals) and the simulation showing the robot responding to the pressed keys. You will need to keep the key\_pub terminal active so that it can receive the keystrokes. The key\_pub terminal must also log (display) the key you are pressing.

**### KEY\_PUB CODE STARTS HERE ###**

import select

import sys

import termios

import tty

import rclpy

from rclpy.node import Node

from std\_msgs.msg import String

class KeyPublisher(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("key\_publisher")

        self.publisher = self.create\_publisher(String, "keys", 10)

        self.timer = self.create\_timer(0.1, self.timer\_callback)

    def timer\_callback(self):

        if select.select([sys.stdin], [], [], 0)[0] == [sys.stdin]:

            key = sys.stdin.read(1)

            msg = String()

            msg.data = key

            self.publisher.publish(msg)

def main(args=None):

    rclpy.init(args=args)

    key\_pub = KeyPublisher()

    old\_attr = termios.tcgetattr(sys.stdin)

    tty.setcbreak(sys.stdin.fileno())

    rclpy.spin(key\_pub)

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old\_attr)

    key\_pub.destroy\_node()

    rclpy.shutdown()

**### KEY\_PUB CODE ENDS HERE ###**

**### KEYS\_TO\_TWIST CODE STARTS HERE ###**

import select

import sys

import termios

import tty

import rclpy

from geometry\_msgs.msg import Twist

from rclpy.node import Node

from std\_msgs.msg import Int32, String

class KeysToTwist(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("keys\_to\_twist")

        self.subscription = self.create\_subscription(

            String, "keys", self.listener\_callback, 10

        )

        self.publisher = self.create\_publisher(Twist, "cmd\_vel", 10)

    def listener\_callback(self, msg):

        key = msg.data

        self.get\_logger().info(f"Received: {key}")

        twist = Twist()

        if key == "w":

            twist.linear.x = 1.0

        elif key == "s":

            twist.linear.x = -1.0

        elif key == "a":

            twist.angular.z = 1.0

        elif key == "d":

            twist.angular.z = -1.0

        elif key == "q":

            twist.linear.x = 0.0

            twist.angular.z = 0.0

        else:

            self.get\_logger().info(f"Unknown key: {key}")

        self.publisher.publish(twist)

def main(args=None):

    rclpy.init(args=args)

    key\_twist = KeysToTwist()

    rclpy.spin(key\_twist)

    key\_twist.destroy\_node()

    rclpy.shutdown()

**### KEYS\_TO\_TWIST CODE ENDS HERE ###**

## Lab Task 3 – Launch Files

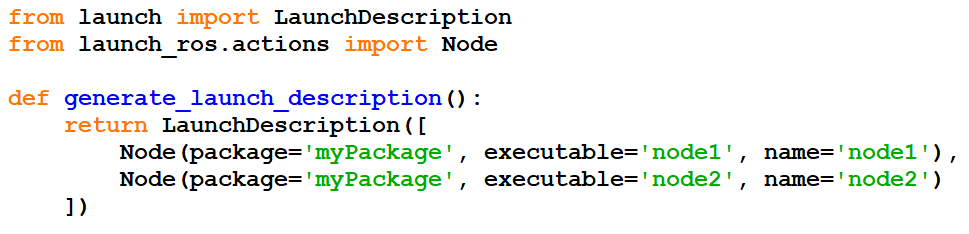
In this task, you will create a launch file that will execute a few nodes at the same time:

**Node 1:** A node that generates random numbers from 1 to 30 and publishes them to ‘random’ topic

**Node 2**: A node that subscribes to the ‘random’ topic and doubles the received numbers

**Nodes 3, 4:** The teleoperation nodes from Task 2 (key\_pub and keys\_to\_twist)

For nodes 1 and 2, ensure the get\_logger() function is used to display their results on the terminal. Create a folder named “launch” in your package and inside it, place a text file named “first\_launch.py”. This is the launch file which will execute all of the above nodes. The syntax for a launch file that executes two nodes is given as follows:

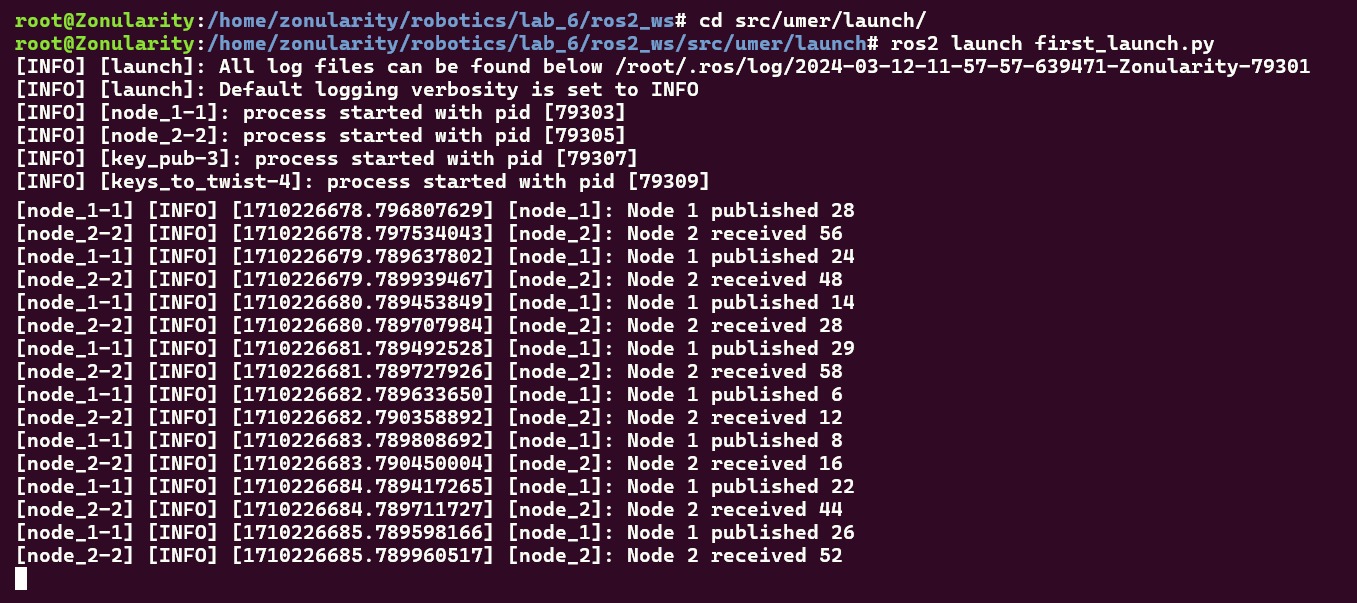


Study the above code and make a launch file that will run all four of the mentioned nodes. After creating the launch file as required, build your package. Then go to the launch folder from the terminal and execute the following command to execute the launch file:

**ros2 launch <launch\_file\_name>**

All of your nodes must start simultaneously. For the task submission, provide the launch file code:

**### LAUNCH FILE CODE STARTS HERE ###**



**### LAUNCH FILE CODE ENDS HERE ###**

## Lab Task 4 – RQt Plot

In this task, you will move your robot around in the simulation and display plots of the linear and angular velocities from the cmd\_vel topic. To make a plot, open a terminal and launch RQt GUI:

**rqt**

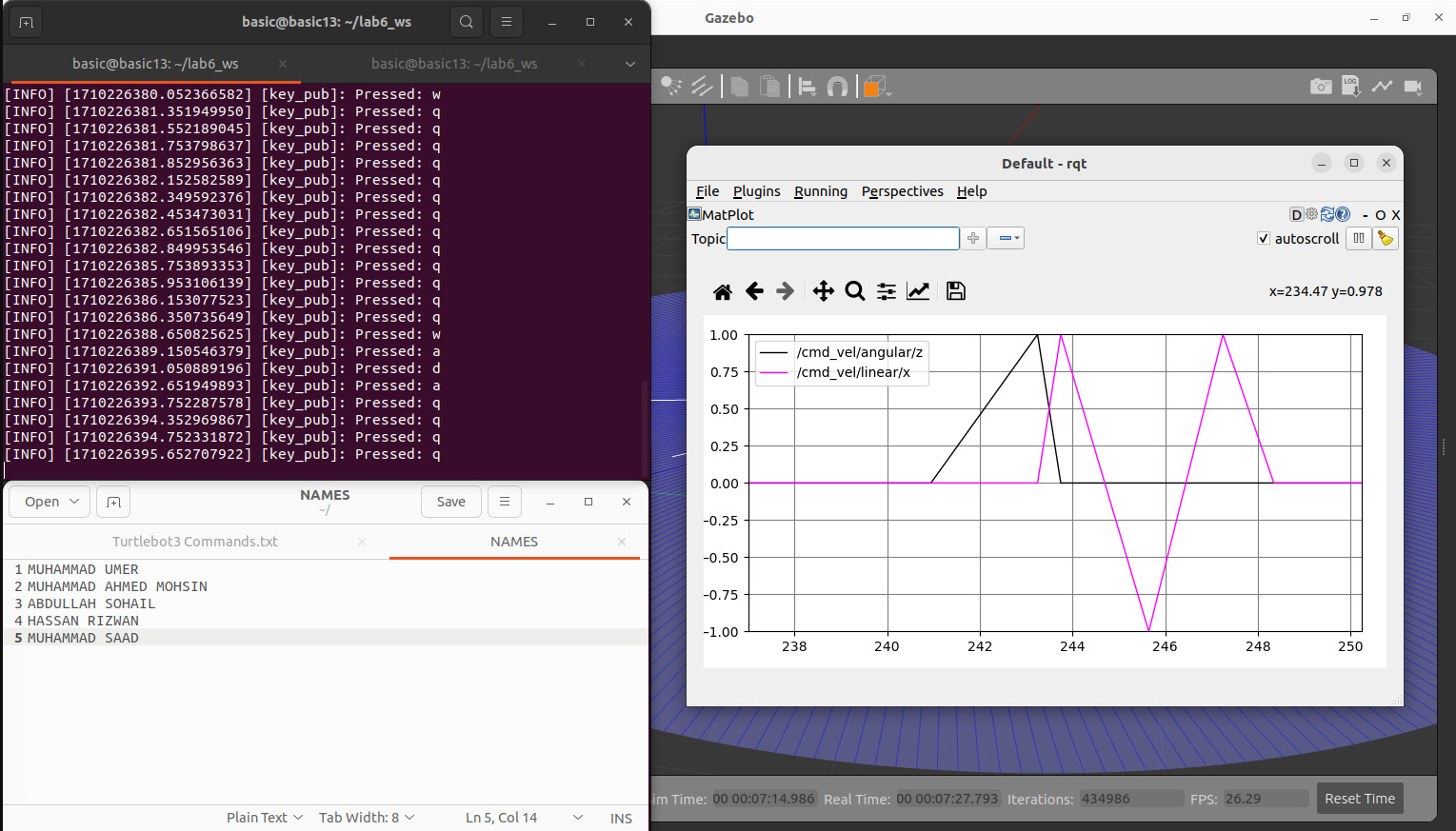
The RQT GUI will open. Close any windows in the GUI. Then, go to Plugins > Visualization > Plot. An empty plot will appear. From the topic textbox, add the following topics:

**cmd\_vel/linear/x**

**cmd\_vel/angular/z**

The plots will now appear as the robot is moved. For task submission, take a screenshot/video showing the plot. Ensure your names are appearing on a terminal in the screenshot.

**### PLOT SCREENSHOT STARTS HERE ###**



**### PLOT SCREENSHOT ENDS HERE ###**

# Conclusion

In conclusion, this lab provided a comprehensive exploration of key concepts in robotics and ROS programming. By successfully creating a custom teleoperation program, implementing launch files, and mastering real-time plotting with the RQt interface, we gained practical insights into node communication and simultaneous publishing and subscribing. These skills are essential for developing robust and efficient robotic systems. Moving forward, the knowledge gained from this lab will serve as a solid foundation for tackling more complex challenges in robotics and autonomous systems.