Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 2

Prelab Tasks:

- **Lab 1** needs to be completed and fully understood before this lab as we will be using the same **publisher.py** and **subscriber.py** files.
- Review the concept of **launch files** in ROS beforehand, using this link: https://www.theconstruct.ai/ros-5-minutes-006-ros-launch-file/ (do watch the video).

Task 1: Creating a Launch File for Complex Number Publisher and Subscriber [15 MARKS]

In this task, we will create a ROS **launch file** to streamline the running of multiple nodes. Start by reviewing the launch file **syntax** and **structure** using the links below:

- https://www.theconstruct.ai/ros-5-minutes-006-ros-launch-file/.
- https://automaticaddison.com/how-to-create-a-launch-file-in-ros-noetic/.

1. Create a launch file that accepts **user-defined** arguments for a complex number's real and imaginary parts, which can be specified when launching the file. [5 MARKS]

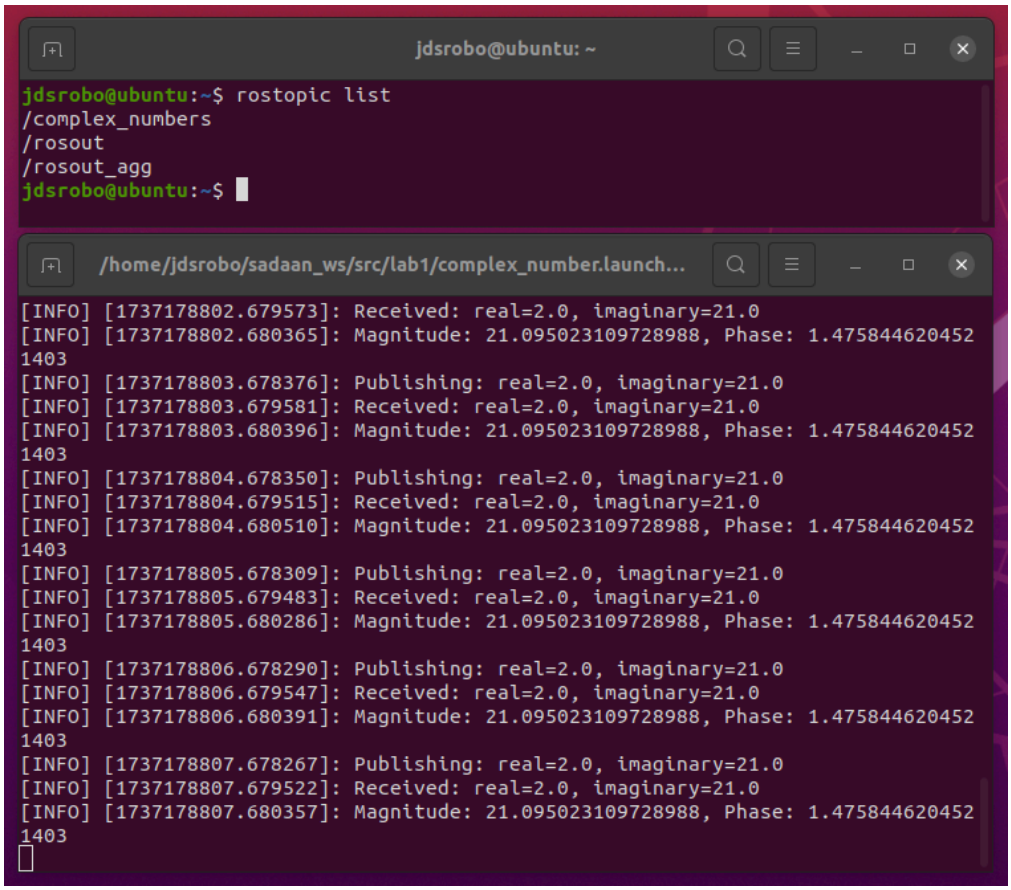
2. Use these arguments (**real** and **imaginary**) to update the publisher node (**publisher.py**) from the previous lab to publish the user-defined complex number. [5 MARKS]

3. Initiate the subscriber node (**subscriber.py**) from the **previous lab** in the **same** launch file to compute and display the **magnitude** and **phase** of the published complex number. [5 MARKS]

HINTS:

- If a **launch** folder **doesn't exist** under your **lab1 package**, then input `mkdir launch` in terminal to create it. Also, to create the launch file via terminal, use this command: `touch complex_number.launch`.
- To **edit** the launch file, simply type `gedit complex_number.launch` in **terminal**
- Remember to always do `catkin_make` to **rebuild your package** and **source your workspace** using `source devel/setup.bash` whenever you make **any changes** to your workspace.
- As a reference, for launching my **complex_number.launch** file, I used this command: `roslaunch lab1 complex_number.launch real:=2.0 imaginary:=21.0`.
- To **verify communication** between the nodes, you can input this command: `rostopic echo /complex_numbers` in **another** terminal.

Your final output for this task should look similar to this:



Task 2: Working with the Turtlesim package and Rosbags [20 MARKS]

2.1 Go to Goal and Straight Line Tutorial using Turtlesim:

In this task, you will explore and practice basic turtle movements using ROS and the **Turtlesim package**. Follow along **tutorial # 10** "**Practicing Python with Turtlesim**" located here: https://wiki.ros.org/turtlesim/Tutorials.

There is **one small adjustment** you need to make to **move.py** to **make it work without errors**:
speed = float(input("Input your speed:"))
distance = float(input("Type your distance:"))

You need to attempt "Moving in a Straight Line" and "Moving to goal" to **complete this task**. This should be fairly easy as everything, including the python scripts, is provided in the tutorials.
[5 MARKS]

2.2 Square and Circle Movement of Turtles:

In this task, you will practice controlling **multiple turtles** with **independent** movement patterns and managing nodes using a **single launch file** called **circle_plus_square.launch**.

Steps:

1. Write new Python scripts (**move_circle.py** and **move_square.py**) to:
    1.1. Control **Turtle 1** to move in a **square** pattern.
    1.2. Control **Turtle 2** to move in a **circle** pattern.
2. Create a **launch file** to:
    2.1. Launch the **Turtlesim environment**.
    2.2. Spawn **Turtle 2** at any specific position **other than** that of Turtle 1.
    2.3. Execute the **movement scripts** for both turtles.

[10 MARKS]

HINTS:

- Launch the first turtle with the turtlesim window using this command: rosrun turtlesim turtlesim_node.
- To launch the second turtle inside the same window, use this command: rosservice call /spawn [choose your own spawn location here] turtle2. The spawn location syntax is like this: 5 5 0.0
- The launch file should launch the first turtle, the second turtle can be launched via terminal commands.

Note: If you figure out how to launch the second turtle using the same launch file, you get 3 extra marks for this lab.

2.3 Recording a Rosbag:

In this task, you will learn how to record ROS data using rosbags.**Tutorial 9.1** "**rosbag Tutorials**" linked over here: https://wiki.ros.org/turtlesim/Tutorials has everything you need to understand for this task.

Steps:

1. **Start** the Turtlesim **simulation** and the turtle movement **nodes**.
2. Use the rosbag record command to **capture specified topics** (e.g., **cmd_vel, turtle1/pose**).
3. Stop the recording after some time once the **required data** has been captured.

[5 MARKS]

Task 3: Turtle Chasing Behavior (baraf paani) [15 MARKS]

In this task you will implement dynamic turtle behavior using keyboard control and node to node interaction.

Steps:

1. Initiate the **turtlesim environment** (rosrun turtlesim turtlesim_node) and **spawn Turtle 2** at **1 1 0.0** using this command: rosservice call /spawn 1 1 0.0 turtle2.
2. Launch the **teleop_turtle** node for **keyboard control of Turtle 1** using this command: rosrun turtlesim turtle_teleop_key.
3. Write a python script for **Turtle 2** called **chase.py**, implementing the following functionality:
    - 3.1. **Subscribe** to Turtle 1's pose topic (**/turtle1/pose**), which provides the **current position and orientation** of **Turtle 1**
    - 3.2. Compute the movement required for **Turtle 2** to **chase Turtle 1**
        - 3.2.1. Continuously adjust Turtle 2's velocity to move towards Turtle 1's position. The script should calculate the **relative distance and direction** between the two turtles and adjust Turtle 2's movement accordingly
    - 3.3. Make a launch file called **chase_turtle.launch** that launches the following:
        - 3.3.1. Turtlesim environment node (turtlesim)
        - 3.3.2. turtle_teleop_key (keyboard controller package)
        - 3.3.3. chase.py

[15 MARKS]

HINTS:

- Try to use this python library import: from turtlesim.msg import Pose.
- Search **callback functions** in ROS Noetic.

Note: If you are able to make Turtle 2 stop when it catches Turtle 1, you get 1 extra mark.

Submission Requirements:

1. Include a **zip file** containing the following items:
    - 1.1. **Launch file** + **Updated publisher.py** script for **Task 1**.
    - 1.2. **Screenshots** for **all sub-tasks** performed in **Task 2**.
    - 1.3. Python scripts (**move_circle.py** and **move_square.py**) and the launch file **circle_plus_square.launch** for **Task 2, sub-task 2.2**.
    - 1.4. **Rosbag file** for **Task 2, sub-task 2.3**.
    - 1.5. The python script **chase.py** and the launch file **chase_turtle.launch** for **Task 3.**

2. Submit a **LAB report** in PDF format (please do not submit word files). This report should provide explanations for all tasks performed along with screenshots.