Laboratory Report 2

Section 1
Assem Akbayeva
Aidana Zhumagaliyeva
Alma Toleubekova
Zhuldyz Altayeva

1. Introduction

The development of robotic systems is mainly done within the framework of the Robot Operating System (ROS). In this report we studied the 5 Degree of Freedom (DoF) Planar Manipulator, a versatile robotic arm designed for various applications in research and industry. By leveraging the extensive resources available within the ROS community, we compiled and utilized several packages from Git repositories to enhance our understanding and control of the manipulator.

Our primary objectives included initiating the robot's movements, implementing single joint control, and analyzing the performance of a PID controller in response to a step input. Visualization of these responses was facilitated through the "rqt" tool, providing valuable insights into the system's dynamics. Additionally, we explored the complexities of coordinating multiple joints to achieve more intricate motions, such as those resembling the fluid movement of a snake.

Through this report, we aim to present our methodologies, findings, and the implications of our work, contributing to the broader understanding of robotic manipulation and control within the ROS ecosystem.

The code and files can be found at https://github.com/asemqr/RoboticsLabs/tree/LabReport2

2. Task 1

Before proceeding with the main task, several preparatory steps were necessary to ensure proper functioning of both the simulated and real robot environments.

2.1. Simulation Setup The robot simulation was initiated with the command:

$ roslaunch gazebo_robot gazebo.launch

A new terminal was used to publish a new angle to the last joint: /robot/joint5/command

2.2. Real Robot Setup The USB adapter was connected, and access was granted with: sudo chmod -R 777 /dev/ttyUSB0

Configuration files were modified to switch from simulation to the real robot by updating the file path in the hand_tutorial_moveit_controller_manager.launch.xml file.

2.3. A ROS node named **joint_controller** was developed within the **controller** package to manage the planar robot's joints. This node listens for std_msgs/Float64 data and only publishes a command to move a joint if the incoming value is higher than the previous one, ensuring efficient operation by preventing unnecessary movements.

The core functionality is implemented in the jointCommandCallback function, which compares each incoming message to the previous value. If the new value is higher, it publishes the command to move the joint. Otherwise, no command is sent, conserving the robot's movement and enhancing operational efficiency.

The node is initialized with ros::init(argc, argv, "rotate"), and it subscribes to a topic (/joint5/input) to receive the data. The node then spins, processing incoming messages and invoking the callback function as necessary. The node's implementation includes a comparison of incoming values with a previously stored value. If the new value exceeds the previous one, it publishes the command to the joint's position controller. The node subscribes to the /joint5/input topic and publishes commands to /robot/joint5_position_controller/command.

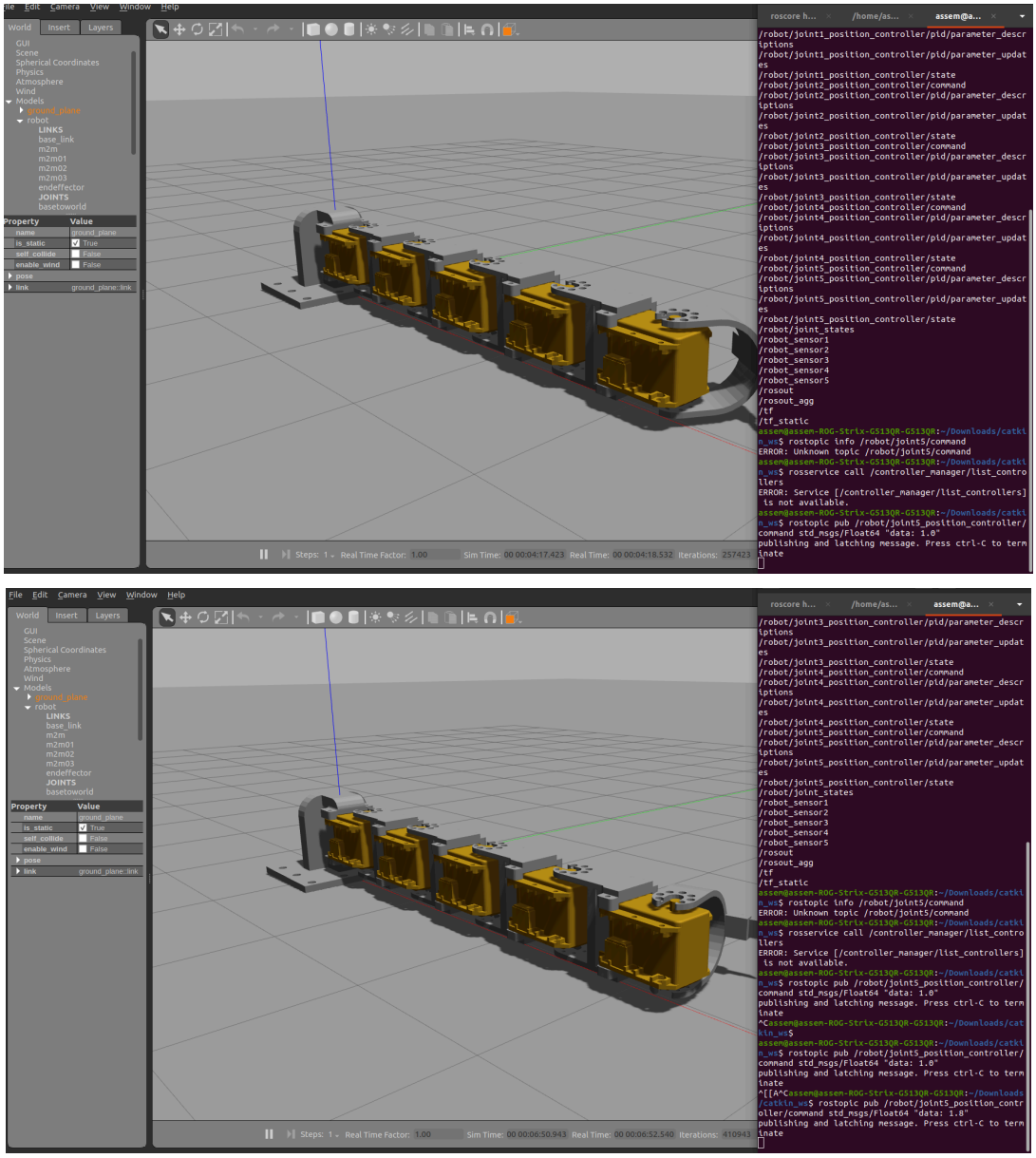

Figure 1. Publisher and listener node in beginner tutorials package

3. Task 2

In Task 2, we generated the step response for both the base and end-effector joints of the robot by sending a square-wave function through a ROS node, named square-node controller. We compiled catkin_ws space using catkinmake, ran the node and observed the plot in rqt, choosing the topics like joint's current pos and goal pos.

For joint 1, we observed a quick rise time and some overshoot before the joint stabilized with little oscillation. The response was relatively fast. In contrast, the fifth joint exhibited a slower rise time, with a noticeable delay before the joint started moving. There was less overshoot, and the joint settled more gradually. Overall, the base joint responded faster and with more overshoot, while the end-effector joint was more stable but slower.

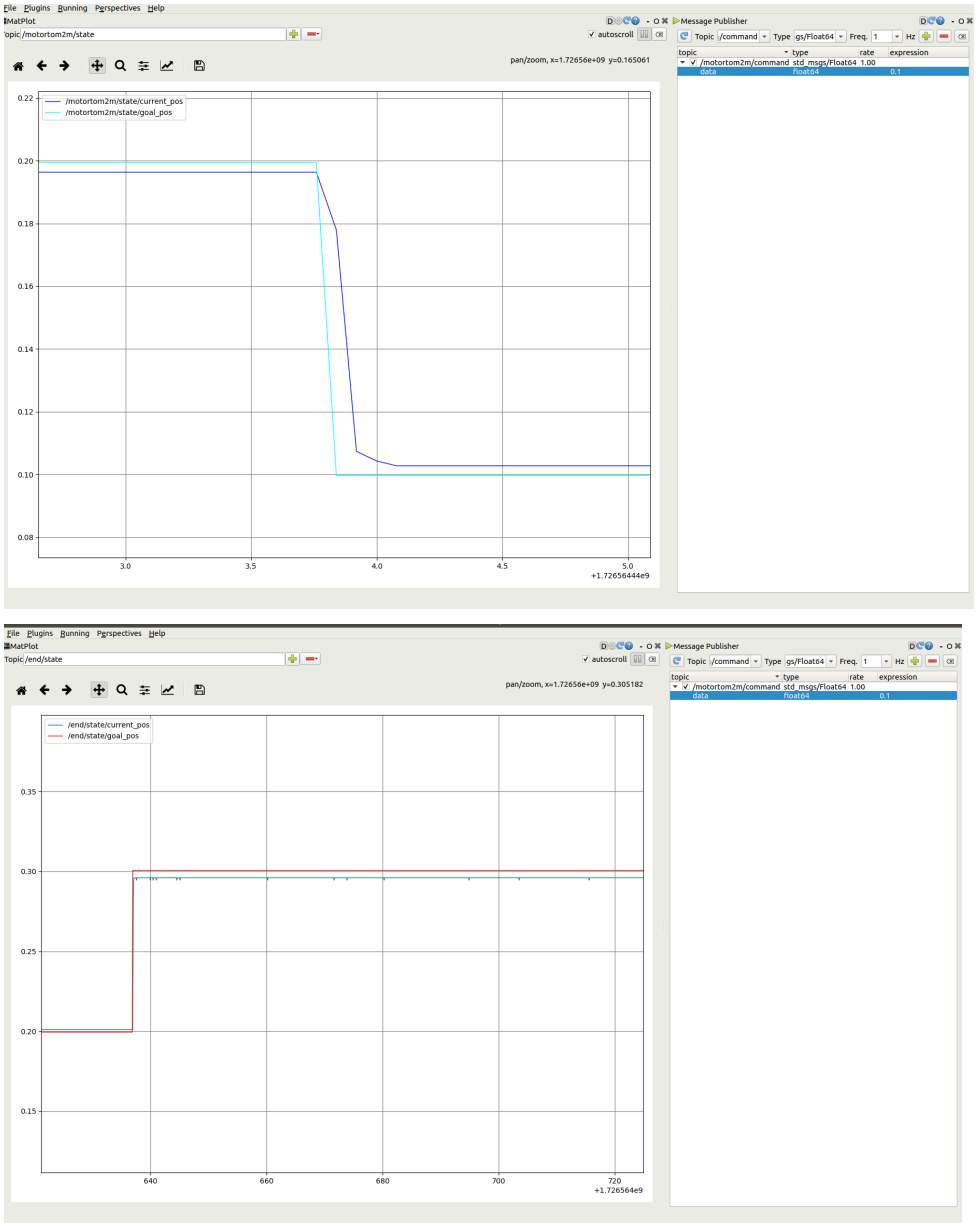

Figure 2. The step wave outputs of joint 1 and 5

4. Task 3

We modified the existing ROS node to generate and publish a sine-wave function as the velocity command to the joints. This involved creating a publisher node that sends a std_msgs/Float64 type message, where the sine function's output was assigned to the angular velocities of both joints.

The sine-wave signal was continuously updated and published to the /robot/joint_command topic at regular intervals. This commanded the robot to oscillate the joints smoothly, following the sine-wave pattern.

After making the necessary modifications to the node, we recompiled the package using catkin_make. Once the node was successfully built, it was executed, and we observed the robot's response in real-time.

Using rqt, we visualized the response of the joints to the sine-wave command. The rqt plot displayed the desired sine-wave signal in blue (command value) and the actual movement of the joints in red (real value). Any deviation between the two curves indicated the lag or errors in the joint response. This process helped us evaluate the robot's dynamic performance under varying conditions.

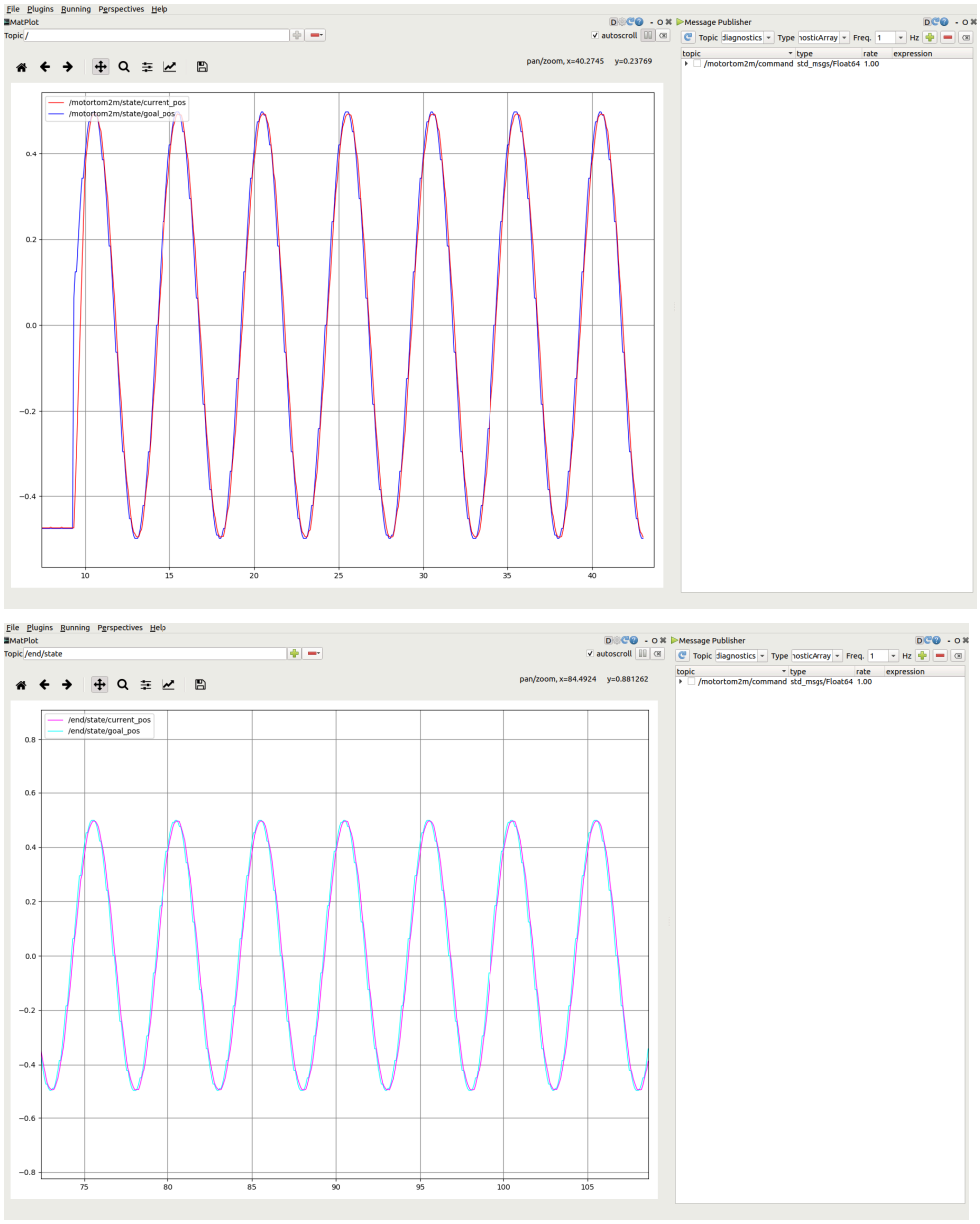

Figure 3. The sine wave outputs of joint 1 and 5

5. Task 4

During Task 4, we experimented with adjusting the Proportional gain (P) of the PID controller for the robot's joints and observed the effects on both the step and sine responses.

When we increased the Proportional gain P, we observed that the joints, both at first and the fifth joint, reached the desired position more quickly. However, this also led to some overshoot and for joint 1. When we decreased P, the response became slower and more controlled, with reduced overshoot. Still, we had a slight steady-state error, where the joint did not settle exactly at the desired position.

For the sine-wave response, increasing P improved led to bigger oscillations. On the other hand, decreasing P caused fewer oscillations. Overall, our adjustments show that if we increase P, we improve precision but lose stability, while decreasing P improves stability but slows down the system, making it less reactive and accurate.

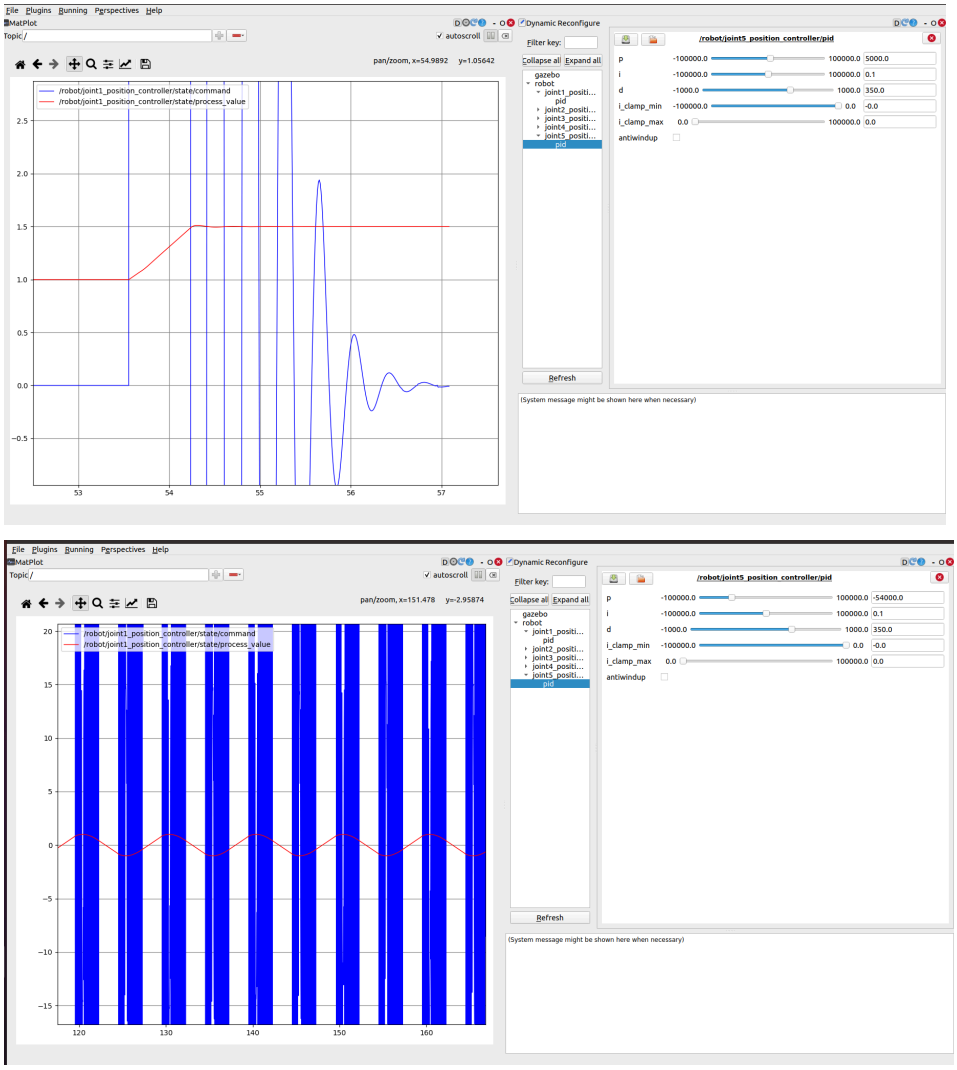

Figure 4. PID Tuning outputs of base and end-effector joints (included only main two graphs)

6. Task 5

The task is to make the robot move like a snake by creating a wave-like motion across its joints. Each joint follows a sinusoidal trajectory, with a phase shift between them to simulate the slithering effect. The code uses ROS to control five joints, publishing their positions in real-time based on sine wave calculations.

For this task, our code calculates the position of each joint using a sine wave, with phase shifts applied between joints to ensure coordinated wave motion. Five ROS publishers are set up to send these positions to the robot's joints in real-time, updating at a rate of 10 Hz. The program continuously runs in a loop, calculating new positions for each joint based on elapsed time and publishing these values to the corresponding topics. This creates the desired snake-like movement, with the robot's joints moving in a smooth, wave-like pattern, mimicking a slithering motion.

Link to the video of the task: snake_robot.MOV

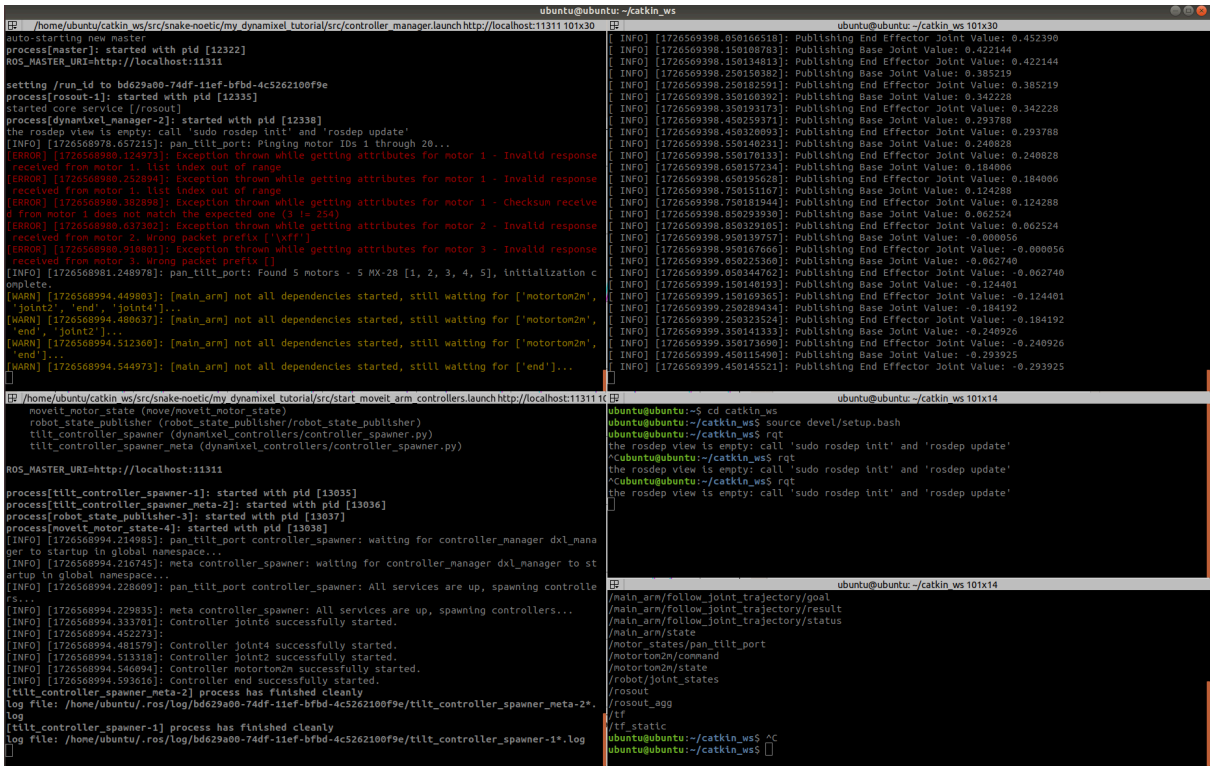

Figure 5. The commands to run snake robot in Ros Melodic