Robotics and XR

Aswin Vattapparambathu Jayaprakash

December 15, 2024

Contents

0 Setup

Initially I used WSL (Windows Subsystem for Linux) with docker to setup ROS but due to my system limitations, I was not able to get a good frame rate and gazebo was not running smoothly. The same problem persisted when I tried it with virtual machine. So, now I am using the UEF server. I was able to run the commands and open Rviz and gazebo and it is running smoothly.

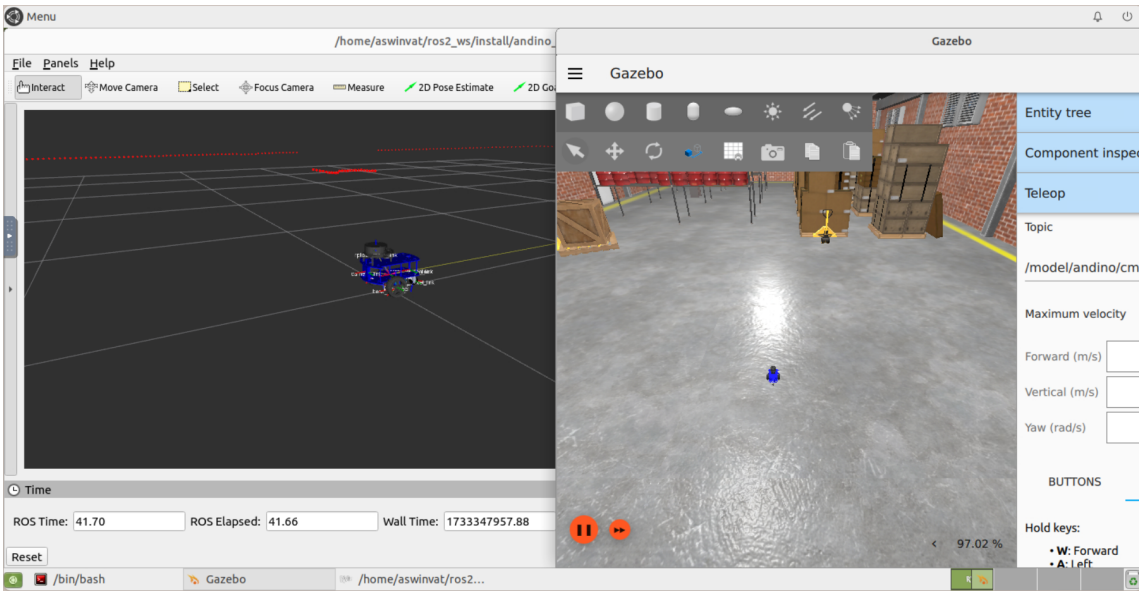

Figure 1: Rviz and gazebo running int the UEF server

1 ROS 2 Introduction

The available topics were successfully listed. The topic works on the publisher subscriber model. We output the data from the Lidar scanner (/scan) using *echo*. *inf* indicates that there are no obstacles near the robot.

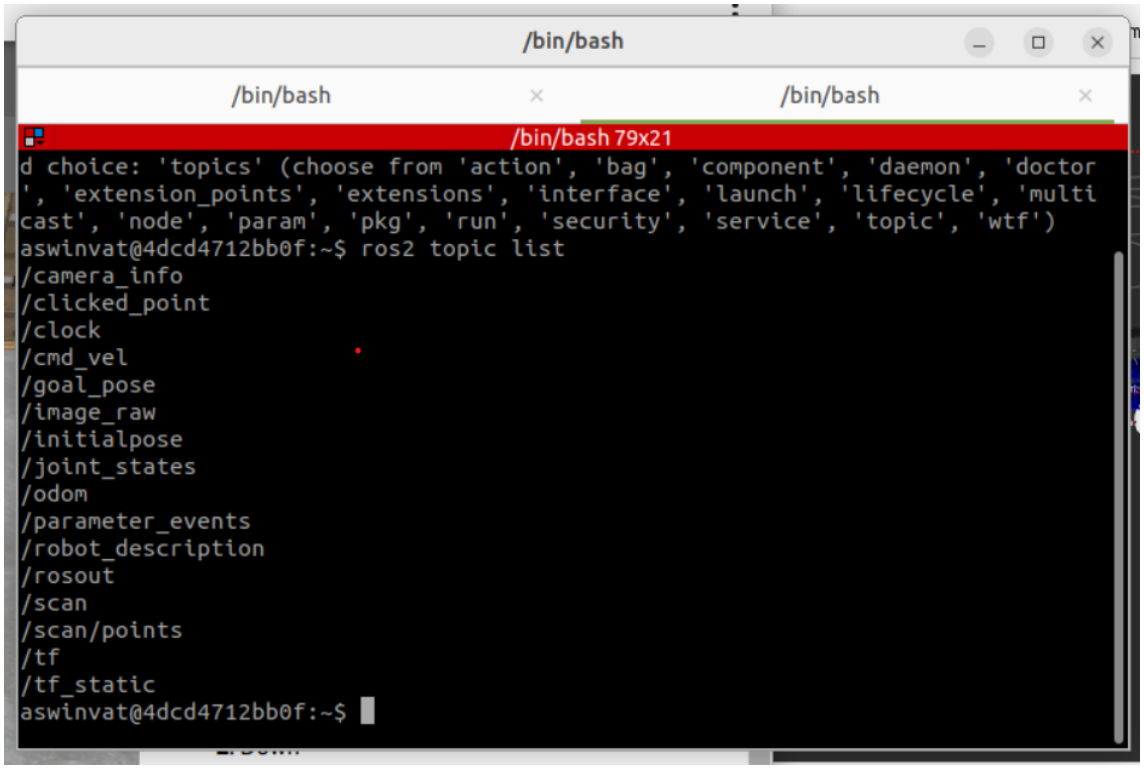

Figure 2: Listing the available topics

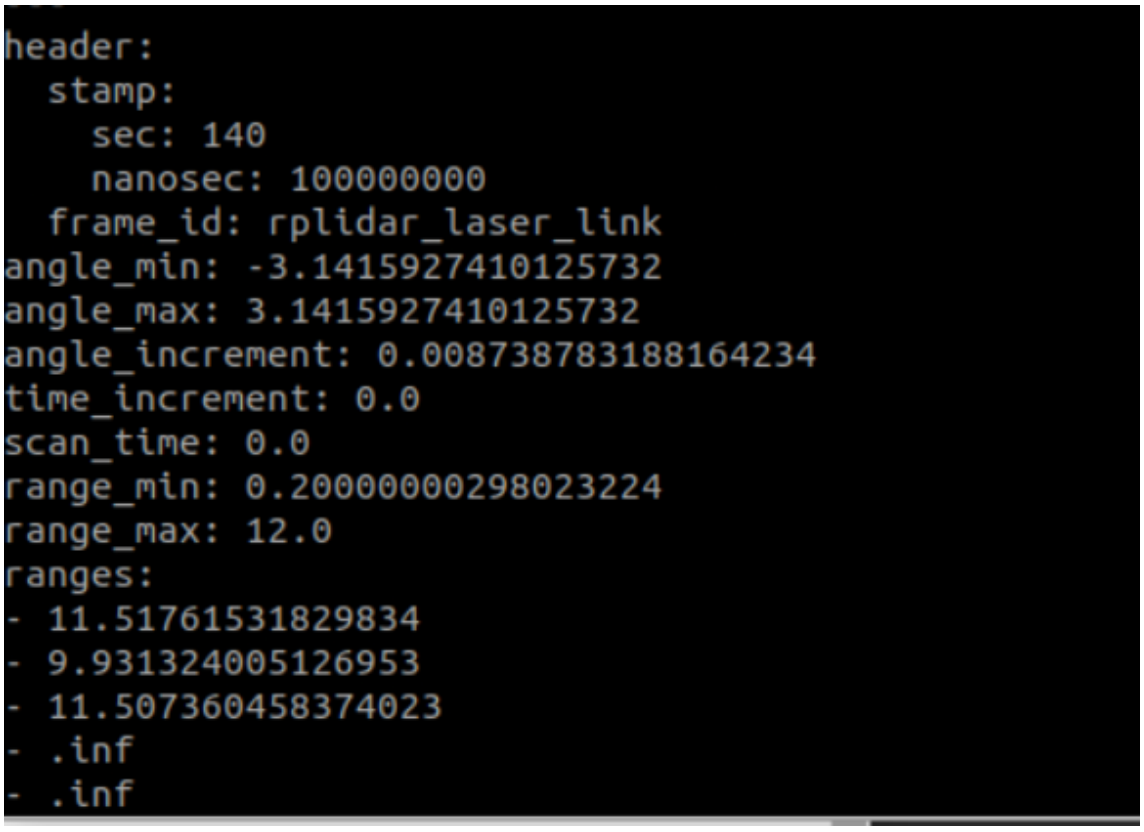

Figure 3: /scan output

2 SLAM and Navigation Demo

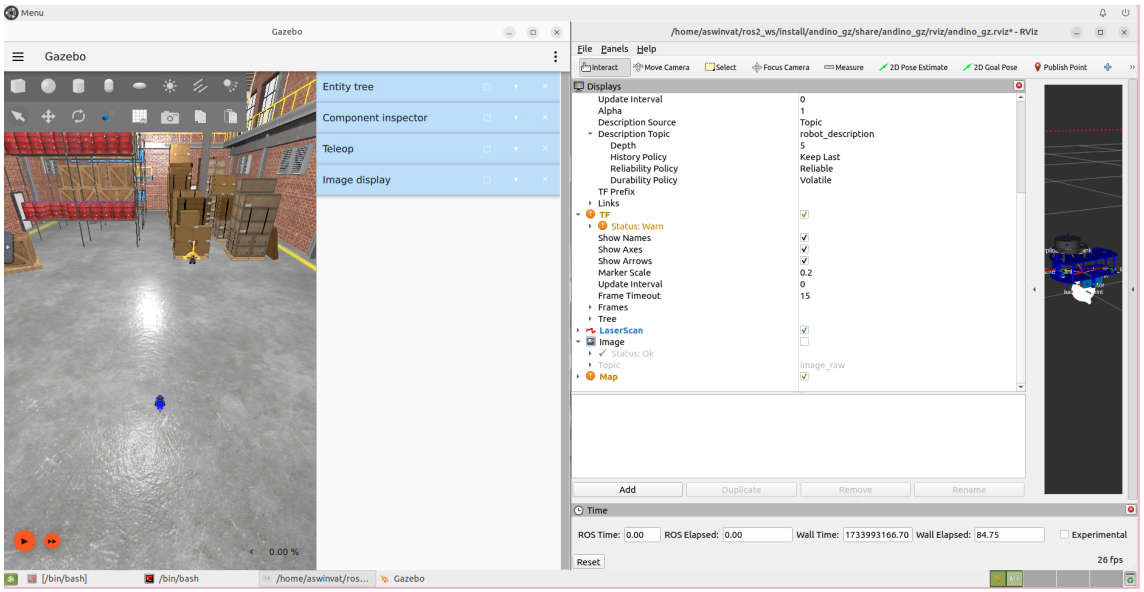

Figure 4: subscribing to /map topic to view the map that is being built.

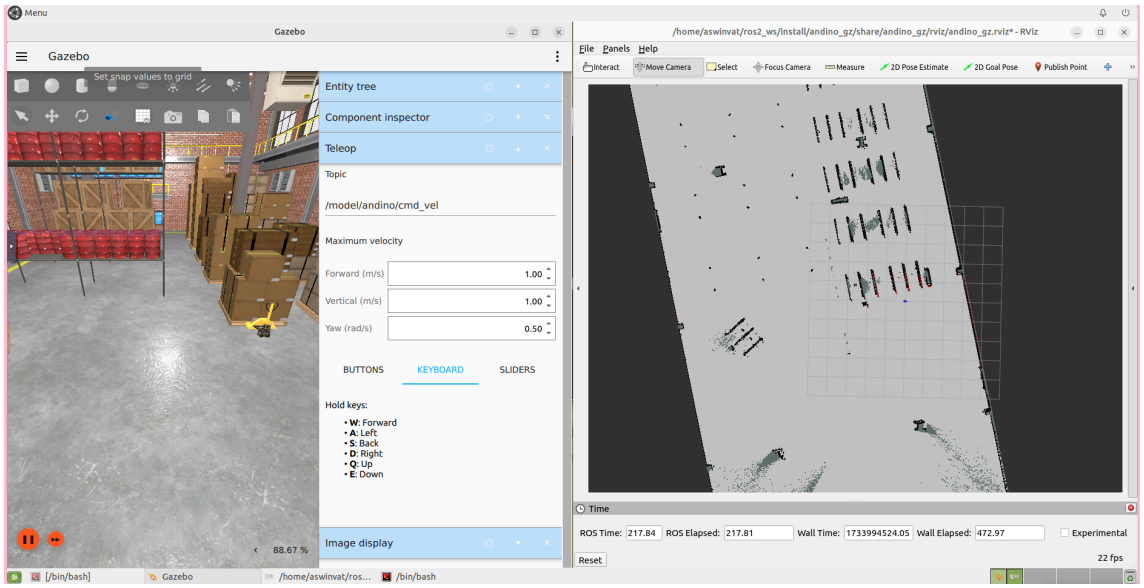

Figure 5: Using Gazebo teleop to drive the robot around in the simulation and map the area.

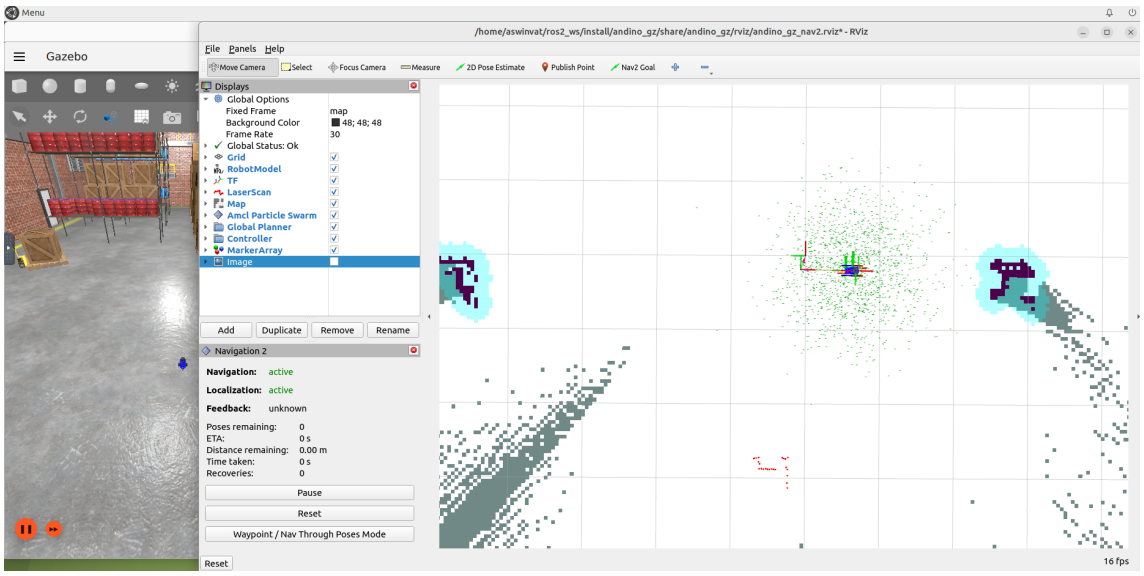

Figure 6: Robot's location on the map

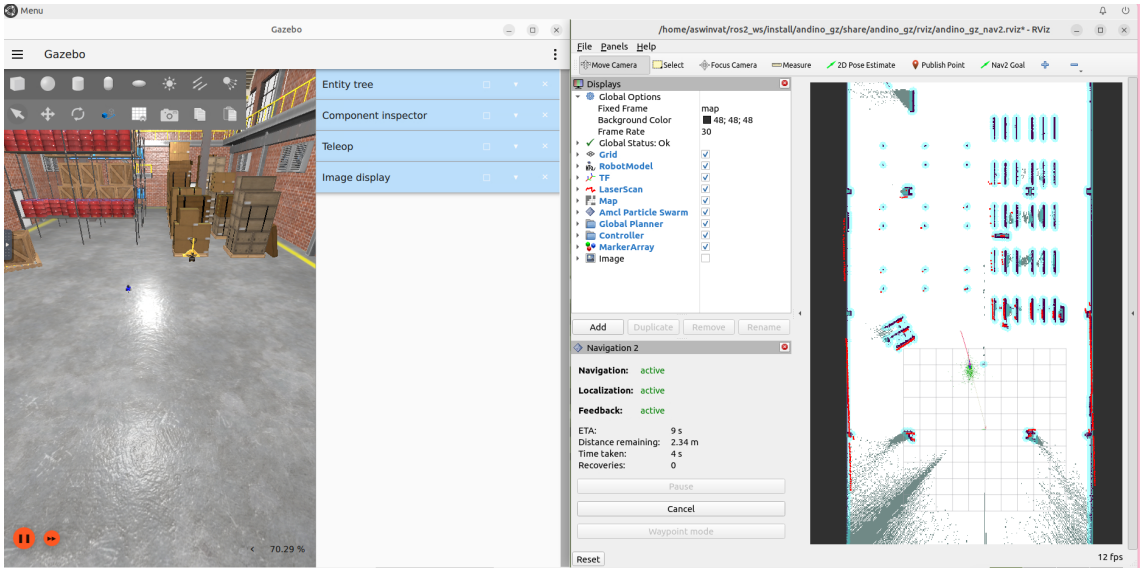

Figure 7: Giving a "Nav2 Goal" from RViz to start the autonomous navigation to a desired location on map

Even though the robot tries to follow the initially planned path towards the goal, it often makes adjustments to the path at times. Also, it does not always reach the goal as it gets stuck and aborts the process.

```
aswinvat@fceab79e8ffc:/$ ros2 service list -t
/amcl/change_state [lifecycle_msgs/srv/ChangeState]
/amcl/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/amcl/get_available_states [lifecycle_msgs/srv/GetAvailableStates]
/amcl/get_available_transitions [lifecycle_msgs/srv/GetAvailableTransitions]
/amcl/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/amcl/get_parameters [rcl_interfaces/srv/GetParameters]
/amcl/get_state [lifecycle_msgs/srv/GetState]
/amcl/get_transition_graph [lifecycle_msgs/srv/GetAvailableTransitions]
/amcl/list_parameters [rcl_interfaces/srv/ListParameters]
/amcl/set_parameters [rcl_interfaces/srv/SetParameters]
/amcl/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/andino_gz_sim/ros_gz_bridge/describe_parameters [rcl_interfaces/srv/DescribePar
ameters]
/andino_gz_sim/ros_gz_bridge/get_parameter_types [rcl_interfaces/srv/GetParamete
rTypes]
/andino_gz_sim/ros_gz_bridge/get_parameters [rcl_interfaces/srv/GetParameters]
/andino_gz_sim/ros_gz_bridge/list_parameters [rcl_interfaces/srv/ListParameters]
/andino_gz_sim/ros_gz_bridge/set_parameters [rcl_interfaces/srv/SetParameters]
/andino_gz_sim/ros_gz_bridge/set_parameters_atomically [rcl_interfaces/srv/SetPa
rametersAtomically]
/behavior_server/change_state [lifecycle_msgs/srv/ChangeState]
```

Figure 8: Displaying the ROS2 service list with their message types

3 Create your first ROS 2 package

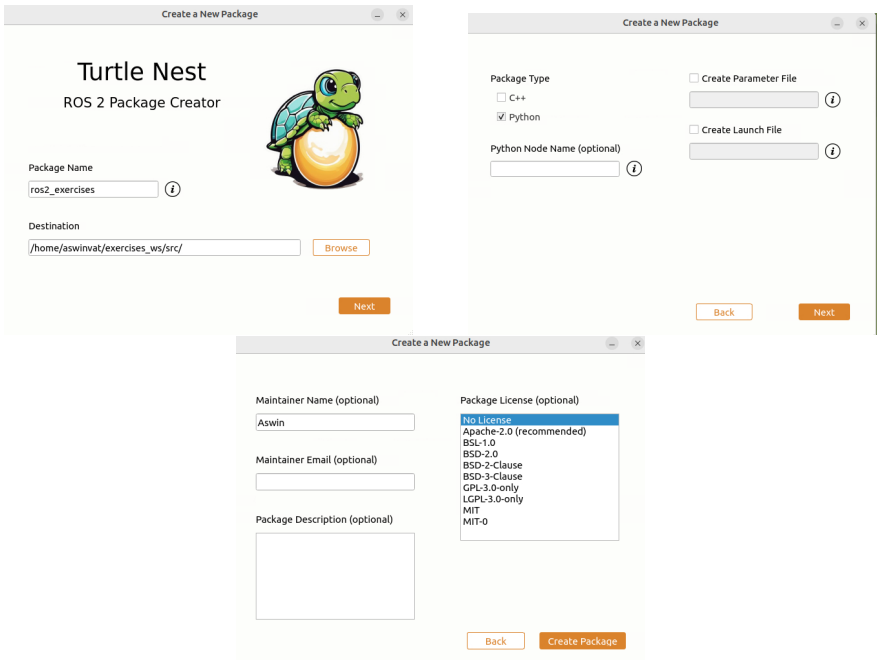

Figure 9: Making packages using Turtle nest tool

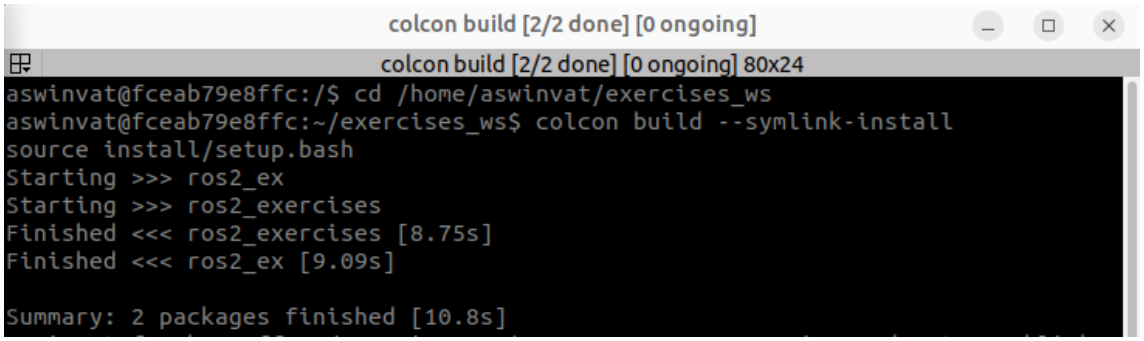

Figure 10: Build and source all the packages inside your exercises_ws -workspace.

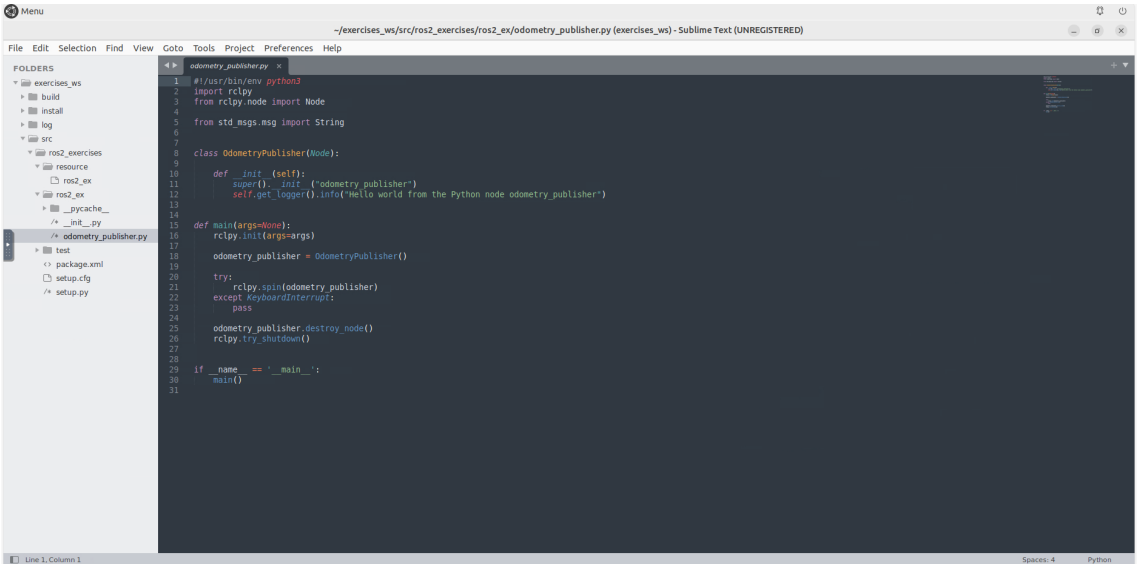

Figure 11: Viewing the exercises_ws folder in Sublime text editor

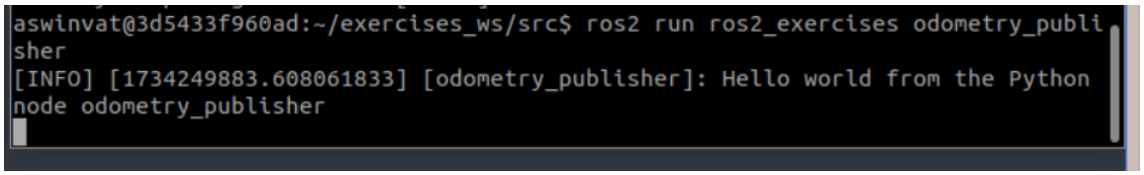

Figure 12: Printing message from my new OdometryPublisher Node

Initially I faced an error saying "package not found" while I was trying to print the message. I found that it was because I was not giving the correct directory for the command. I fixed it and was able to print the message successfully.

4 Calculate and publish your robot's odometry using wheel velocities

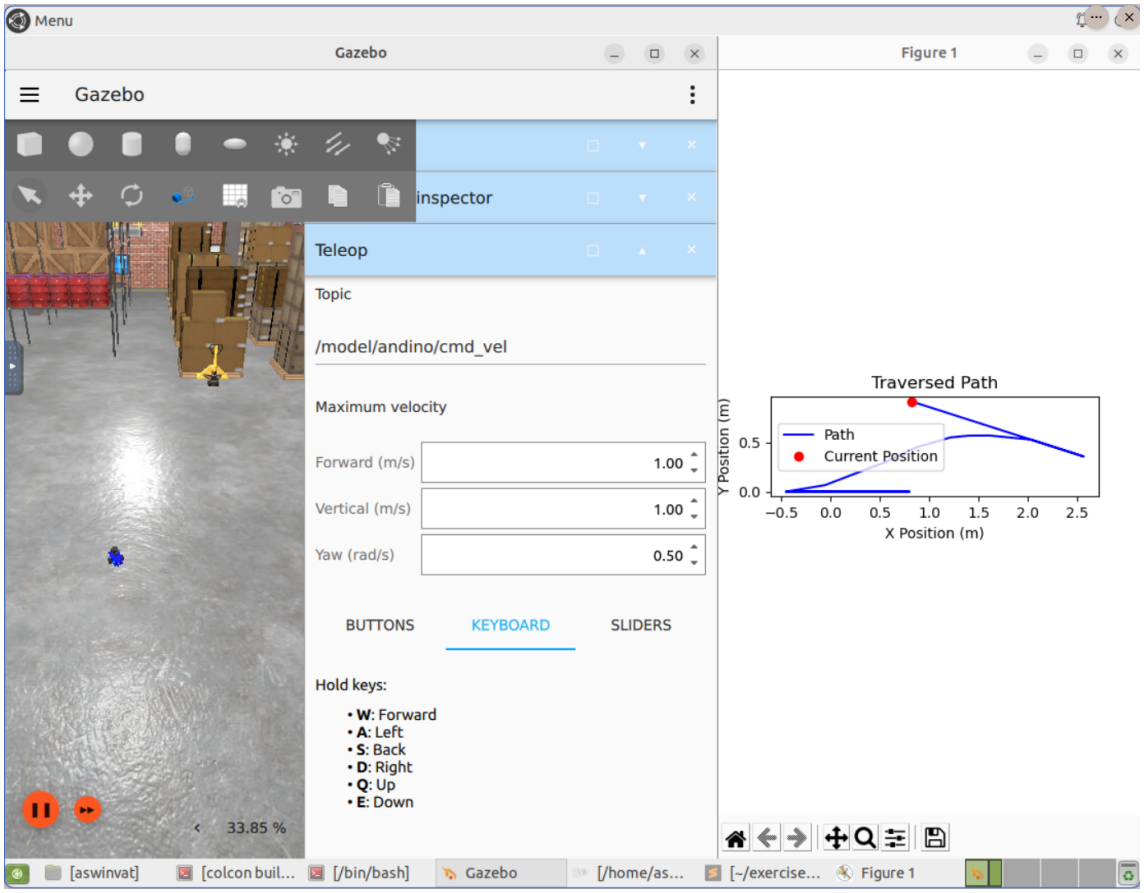

Figure 13: Visualization of odometry data while driving the robot i the simulation

The odometry data does correspond to the driven path in the simulation. But it can be less precise in the real world due to various errors and environmental factors. Also, there are situations where the robot is moving according to odometry data but the robot is actually stuck at some obstacle or fallen upside down. This is because we are using only the wheel odometry and can be eliminated by combining odmetry data from various sensors of the robot.

5 Custom path planning using Nav2

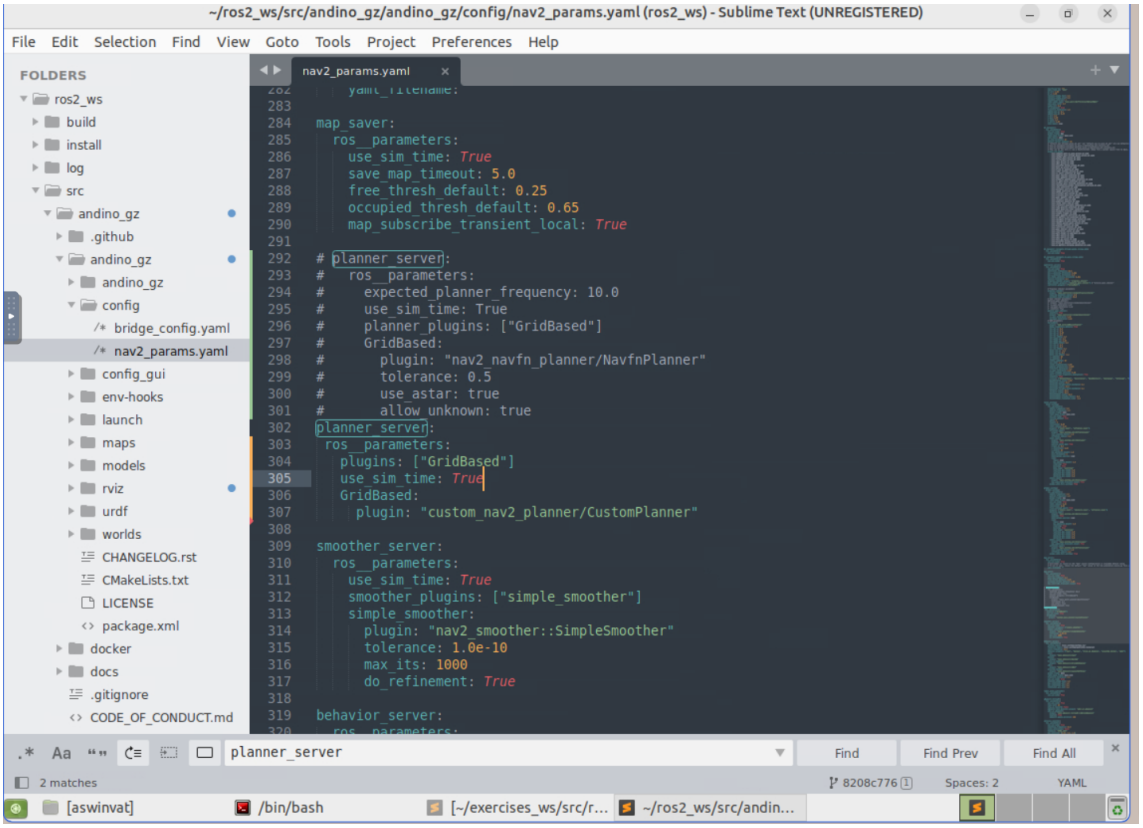

Figure 14: Updating Andino's parameter file with coustom imlimentation of path planning using Sublime Text

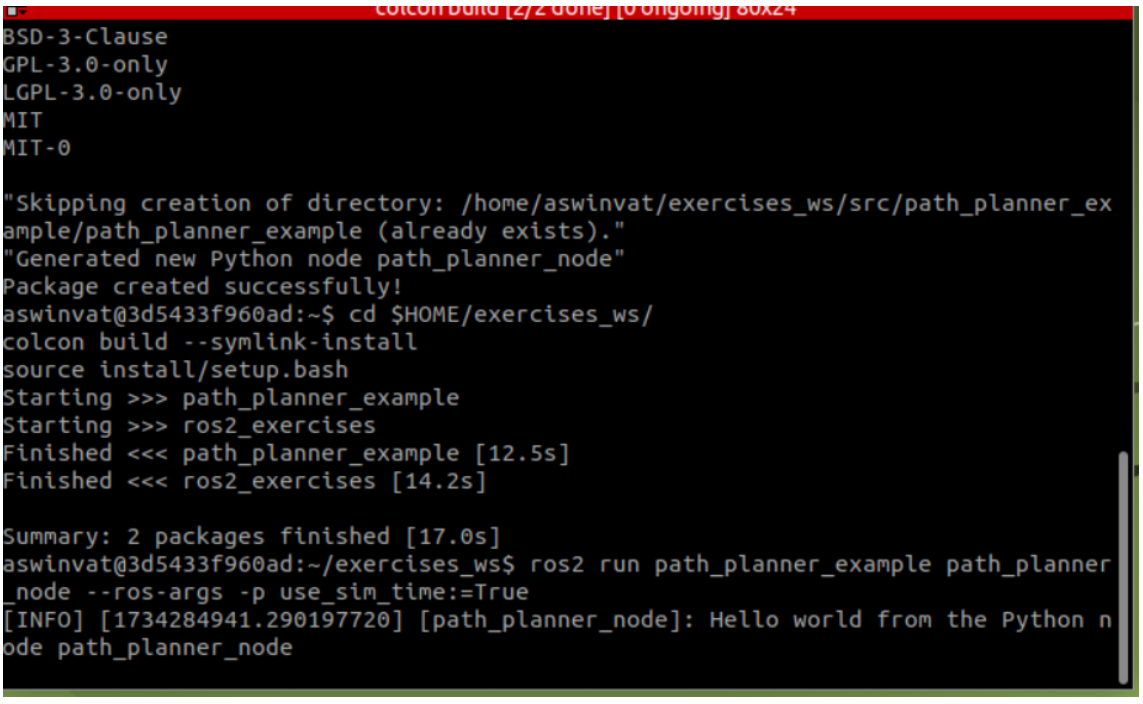

Figure 15: New ros2 package with node named "path_planner_node" was built and sourced

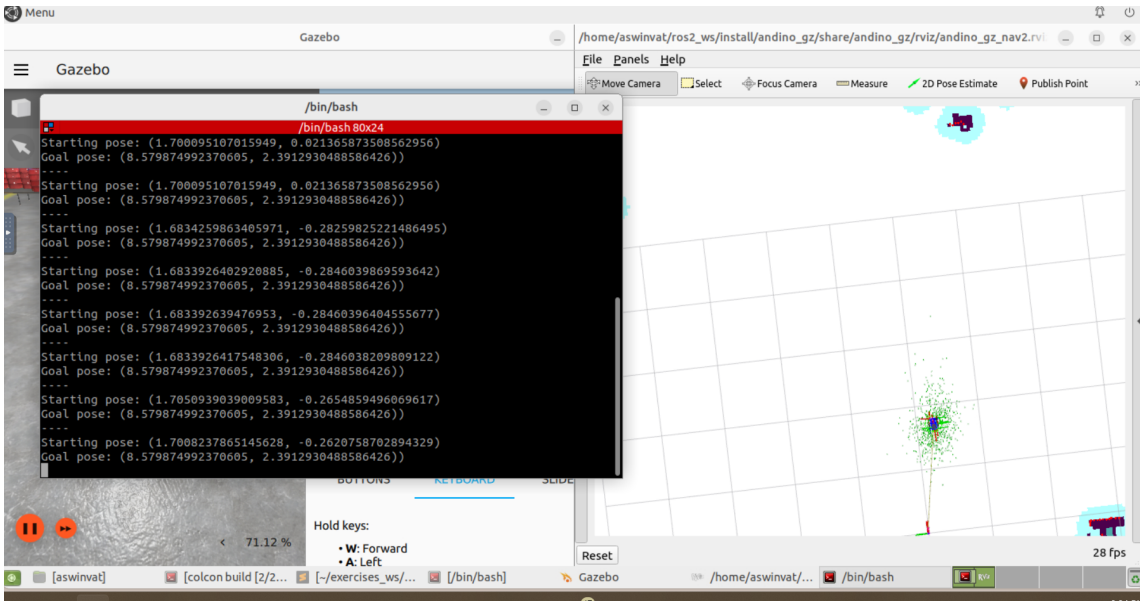

Figure 16: When given a pose estimation and a new Nav2 goal, the node now prints start and goal positions with a certain interval. Nav2 keeps requesting replanning of the path periodically by default.

A few seconds after sending the goal, the robot will start moving a little because Nav2 has a behavior server running to have recovery behaviors if a plan could not be calculated. (This is useful for example if the robot is stuck in a way that it is not possible to calculate the path, or the goal is not reachable.)

After editing the *create_straight_plan()* function in the node, the robot follows a straight path towards the goal.

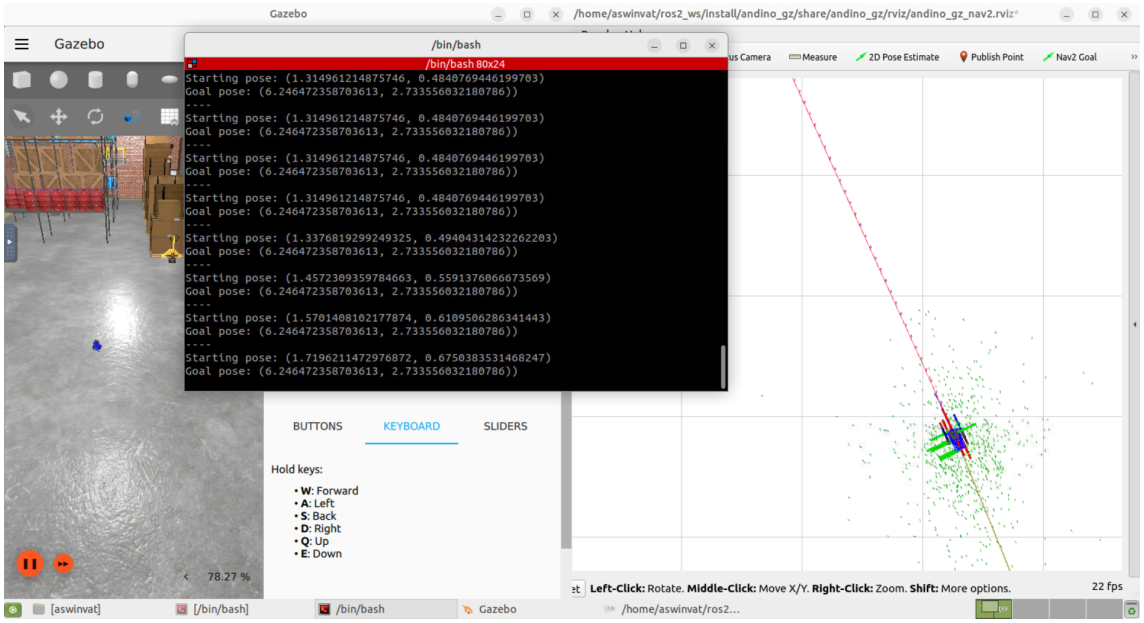

Figure 17: Robot finding straight path towards assigned goal

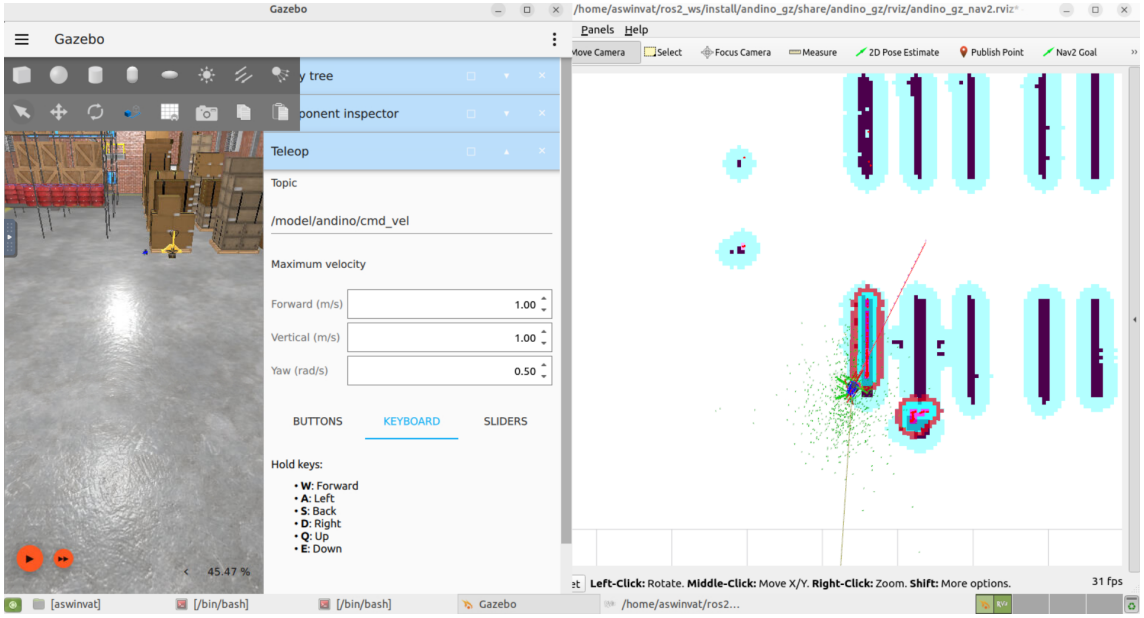

Figure 18: Robot getting stuck at obstacles

But the problem with this algorithm of finding the straight path is that the robot can get stuck at obstacles easily and is less likely to reach the goal if it is behind many obstacles. So, we need more nuanced and efficient algorithms like A*, D*, etc.