Assessment task completion documentation for the Position of ROS Developer in GOAT Robotics.

1. The task is implemented in the ROS2-Humble environment.

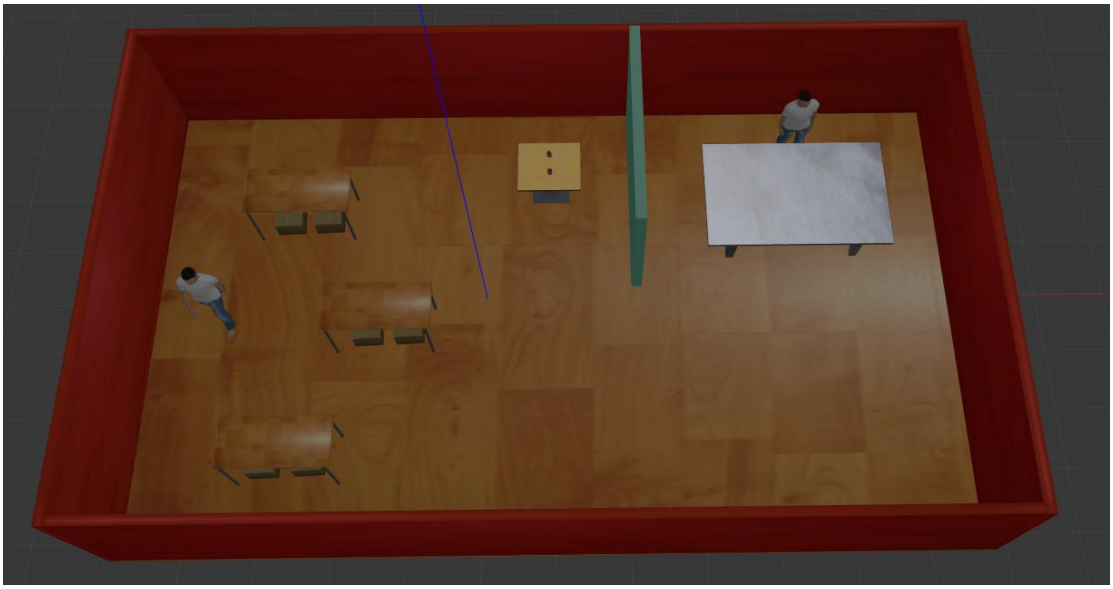
2. Concepts used to implement the task using ROS2

    1. Package and dependencies and node include
        - ➔ Created a package named **delivery_robot** using the ament_cmake build system because the code is implemented using C++. And Apache-2.0 license is used.
        - ➔ rclcpp, std_msgs, nav2_msgs, geometry_msgs, rclcpp_action and rosidl_default_generators dependency packages are needed for this task to meet the objective and it is included in **CMakeLists.txt** and **package.xml** files.
        - ➔ The order_management node is implemented to handle the orders placed and canceled, included in the **CMakeLists.txt** file with above dependencies.

    2. URDF and Building Editor for Simulation.
        - ➔ The whole concept is implemented in the simulation, which makes us easy to observe what's happening in the simulated environment.
        - ➔ The robot is built up using the URDF fileformat named **delivery_bot.urdf.xacro** and **intertial_xacros.xacro** file included to reduce the code for adding the property and inertial values for the shapes of the links, under the description folder in the package. The urdf file has some gazebo plugins such as
            - **libgazebo_ros_camera.so** for the camera that gives us the camera functionalities and publishes the image to a ROS message.

- **libgazebo_ros_ray_sensor.so** plugin for the lidar that gives us the laser_scan functionalities and publishes the scan to a ROS message.
- **libgazebo_ros_diff_drive.so** plugin for the diff_drive_controller from the ros2_controllers to make the robot move by subscribing the /cmd_vel topic.

➔ The Restaurant environment is created using **Building Editor** in gazebo and added models such as furnitures, walls, floor_panels etc., and created a world file named **restaurant.world** under worlds folder in the package.



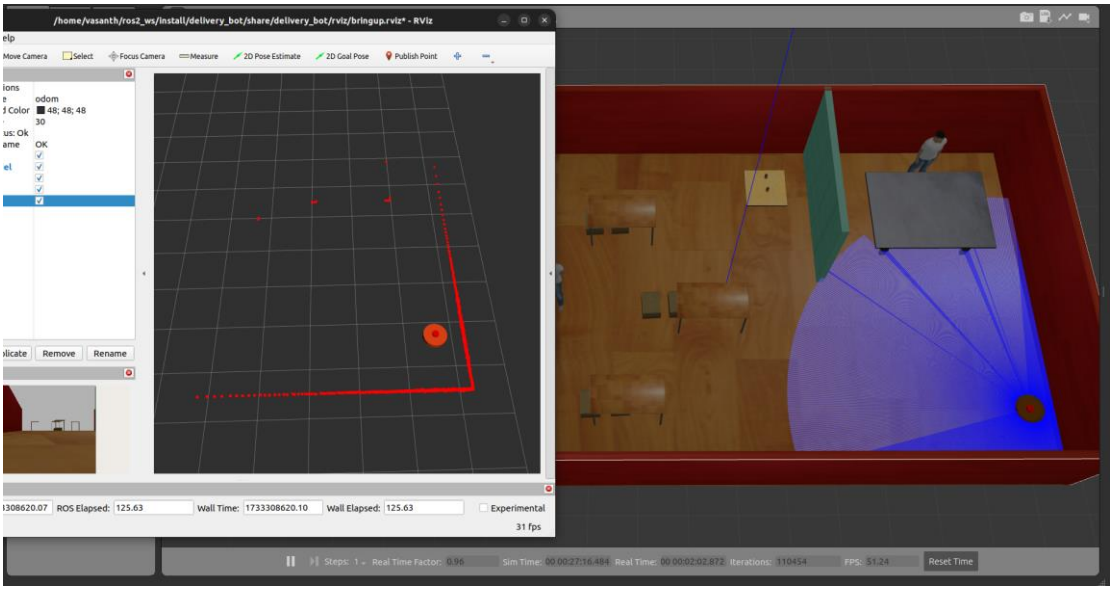
3. Service files for placing and canceling the order.

➔ Custom service files(.srv) created and named **PlaceOrder.srv** for placing the order, and **CancelOrder.srv** for canceling the order under the srv folder, and included these service files in CMAKELISTS.TXT file under rosidl_genrate_interfaces dependency.

4. Bringup and move the robot in gazebo and rviz

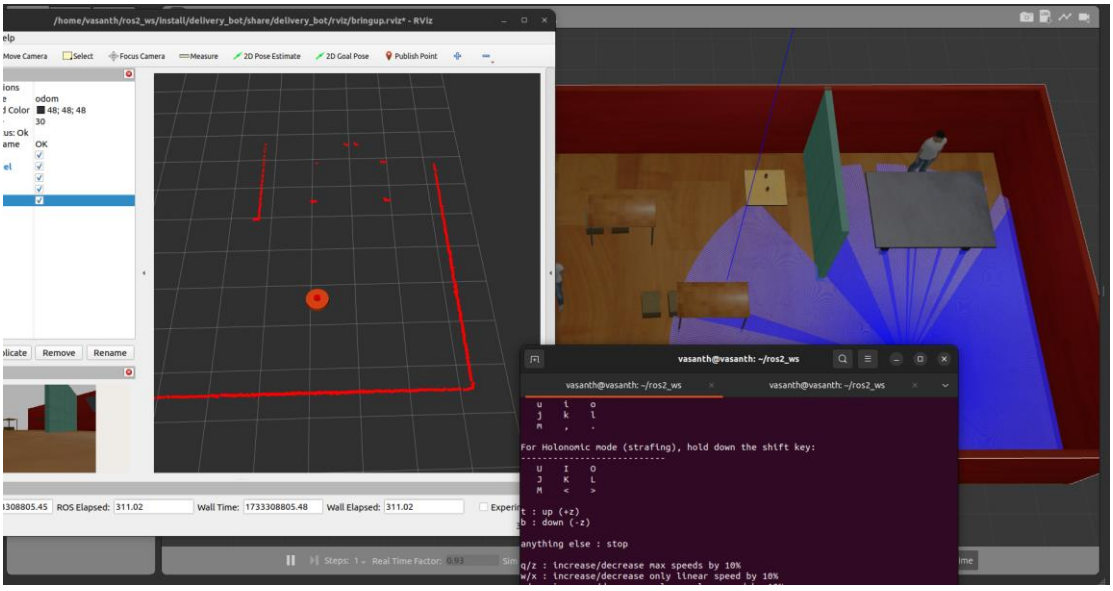
➔ the brinup_launch.py file under the launch folder in the package, has rviz2, robot_state_publisher, Joint_state_publisher, gazebo_ros packages as nodes, by launching this command

ros2 launch delivery_bot bringup_launch.py



the robot appears in the rviz and gazebo with the restaurant environment. Because of we added the diff_drive_controller plugin, we can subscribe the topic /cmd_vel using this command

ros2 run teleop_twist_keyboard teleop_twist_keyboard



and make the robot move as we want using the keyboard.

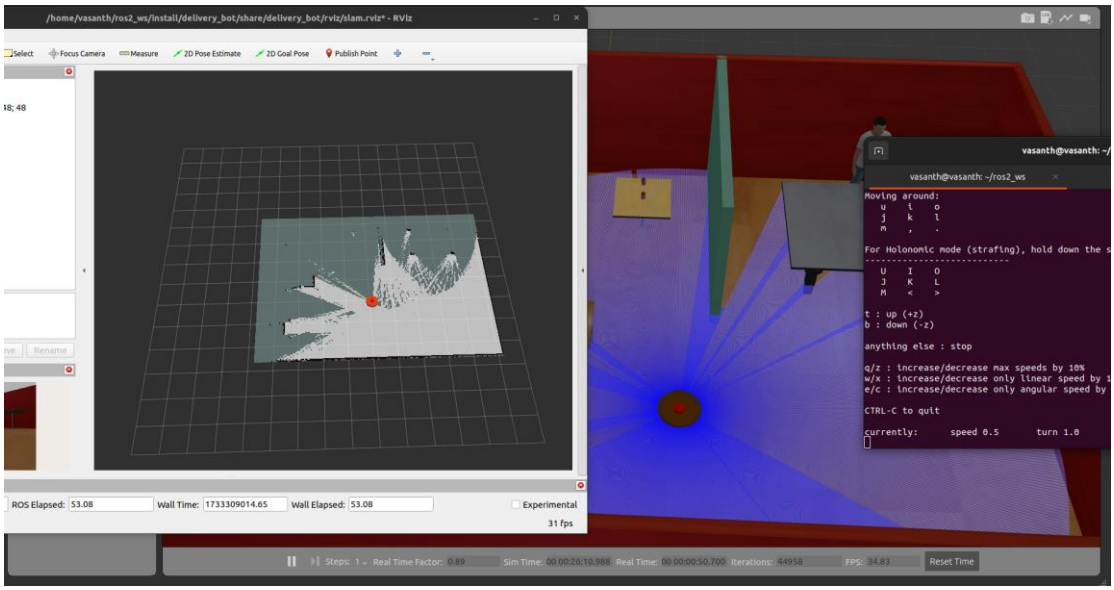
5. Mapping, localizing and nav2

➔ the slam_launch.py file under the launch folder in the package, has rviz2, slam_toolbox packages as nodes along with bringup node, slam.yaml file under the params folder, has the ros parameters and plugin for the slam_toolbox package. along with the bringup node, and rviz2 node. Launching this command

ros2 launch delivery_bot slam_launch.py

enables the robot to scan and map the environment along with the execution of this node

ros2 run teleop_twist_keyboard teleop_twist_keyboard



once the map is created save the map file using this command

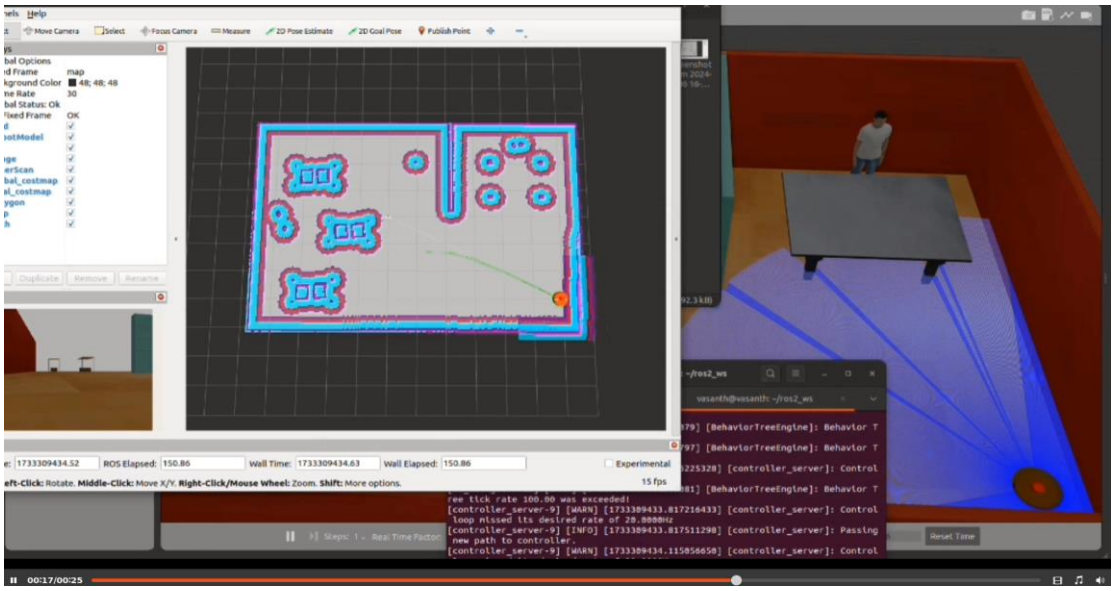
ros2 run nav2_map_server map_saver_cli -f maps/restaurant

➔ The navigation_launch.py file under the launch folder in the package, has rviz2, nav2_amcl, nav2_map_server nodes for localizing the robot, amcl.yaml file in the params folder has the parameters, laser_model_type and robot_model_type for the amcl

and map_server package. nav2_lifecycle_manager package controls the execution of the two packages while running.

➔ The same navigation_launch.py file also has nav2_controller, nav2_smoother, nav2_planner, nav2_bahaviors, nav2_bt_navigator, nav2_waypoint_follower, nav2_velocity_smoother nodes to take care of the navigation of the robot while approaching to the goal_pose when given. The nav2_params.yaml file has parameters and plugins required for the above nodes along with global_costmap and local_costmap. Launching this command

ros2 launch delivery_bot navigation_launch.py



starts to execute the rviz and gazbeo, first the robot has to localize the robot in the rviz using 2d_pose_estimate and we can set the goal_pose in the environment using 2d_goal_pose. The robot will reach the goal_pose.

6. Order Management Node

➔ The order_management.cpp node has its header file order_management.hpp file under the include/delivery_bot folder.

➔ The public and private member functions and variable delcaration is defined in the order_management.hpp file.

The declaration of a publisher

- **/ordered_tables** – publishes the number of orders ordered in the restaurant.

The declaration of a subscriber

- **/kitchen_confirmation_response** – establish and receives the confirmation data as a string from kitchen, whenever the food is ready for the table to be served next. If the confirmation from the kitchen received as 'No', the robot returns to the home position.
- **/table_confirmation_response** – establish and receives the confirmation data as a string from the ordered tables served. After the confirmation received from the table, the robot returns to the kitchen to take the food for another next table ordered.

The declaration of a service

- **/place_order_service** – establish and receive the data from the table number to place the order.
- **/cancel_order_service** – establish the service and receive the data from the table number to cancel the order.

The declaration of a client for rclcpp_action

- **/navigate_to_pose** – handles and send the goal for the goal_msg to the action_server.

The variables declaration

- Because of we used the robot which does not have any z axis move or variations. We used only x and y axis coordinates. std::pair is used for that for home and kitchen positions.
- Since there are three tables, std::unordered_map is used to store the table_numbers and their x and y coordinates

➔ The declaration and defintion of member functions

- **place_order_check()** - receives the request of the table number and store the order in the queue, and update the

size of the queue. If there are more than 3 orders received, the queue will not accept.

- **cancel_order_check()** - receives the request of the table number to cancel the order of the table number which is already ordered, update the queue and size of the queue.
- **publish_ordered_tables()** - publishes the size of the queue whenever it is updated.
- **move_to_kitchen()** - checks whether the action_server is available at first, then it has the geometry_msgs as a goal_pose of the kitchen coordinates and send the goal_pose to the goal of the NavigateToPose.
- **kitchen_confirmation_response_callback()** - subscribe the topic from /kitchen_confirmation_response and receives the confirmation data from the kitchen, then move_to_table() is called to serve the table which is in the order of the queue. If the confirmation data received as no, also if there is no orders in the queue, the return_home() function called to
- **move_table(int table_number)** – checks if there are orders in the queue at first, if there are, then the goal pose of the ordered table number coordinates is send, by calling the send_goal_to_nav2(). If there are no orders found in the queue, then prints out “no more orders to serve.”
- **table_confirmation_response_callback()** - subscribe the topic from /table_confirmation_response and receives the confirmation data from the table which is served just now, delete the table numbered served,

update the order of the queue and size of the queue, calls the print_queue function to print the orders in the queue, calls print_remaining_orders() to print how many orders are remaining and then calls move_to_kitchen() to take the food for the next ordered table in the queue.

- **return_home()** - the goal_pose of the map coordinates send to the action_server by calling send_goal_to_nav2().
- **send_goal_to_nav2()** - handles the goal to send the action_server, by calling the feedback_callback function which prints the robot is reached its goal or not.
- **print_remaining_orders()** - prints out the order_size of the queue.
- **print_order_queue()** - prints out the orders in the queue.

7. Execution workflow

➔ Launch the navigation_lanch.py file

ros2 launch delivery_bot navigation_launch.py

➔ Localize the bot using 2d_pose_estimate in rviz, the robot always in home position at starts.

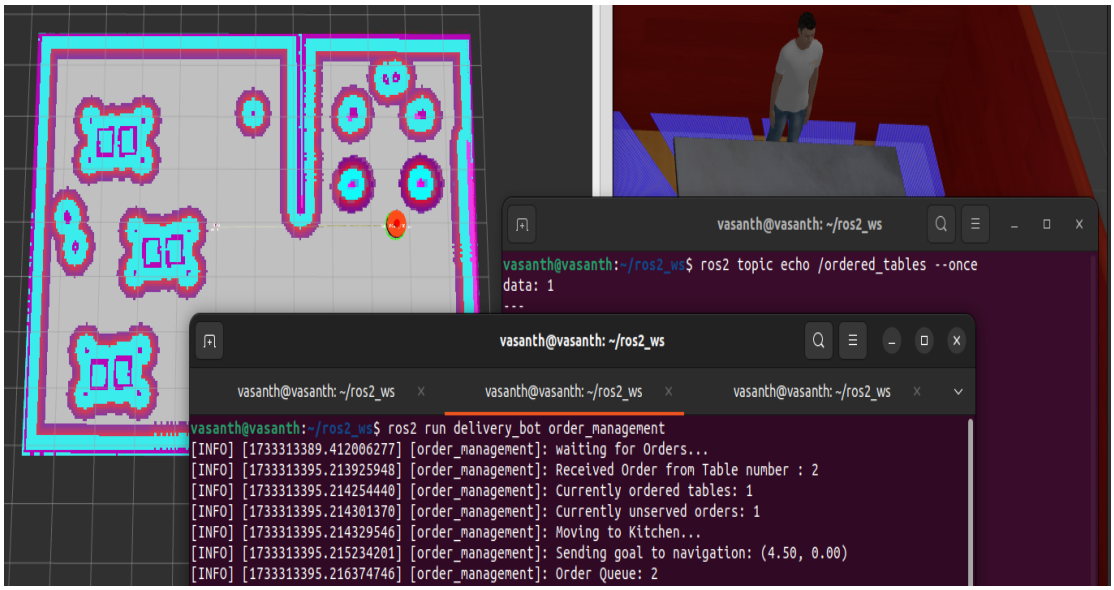
➔ Launch the order_management node using this command in another terminal

ros2 run delivery_bot order_management

➔ Place an order for a table number

ros2 service call /place_order delivery_bot/srv/PlaceOrder "{table_number: 2}"

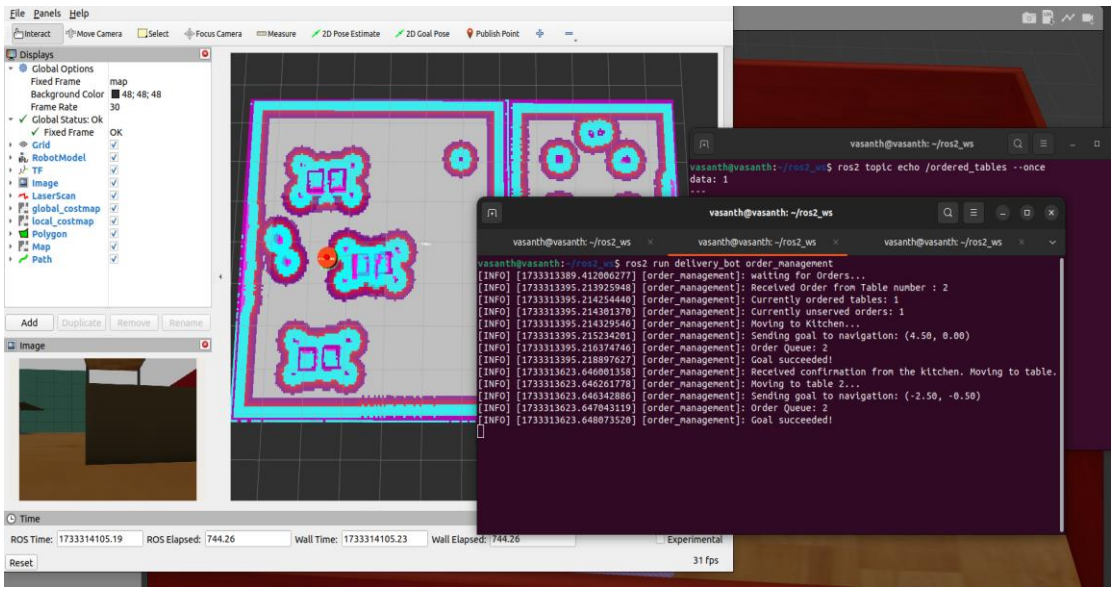
➔ The queue is updated you can see in the order_management node terminal output, also when echo the /ordered_tables topic



➔ Now the robot reached the kitchen coordinates, it is waiting for the confirmation from the kitchen. We can give the confirmation as

ros2 topic pub /kitchen_confirmation_response std_msgs/msg/String "{data: 'yes'}" –once

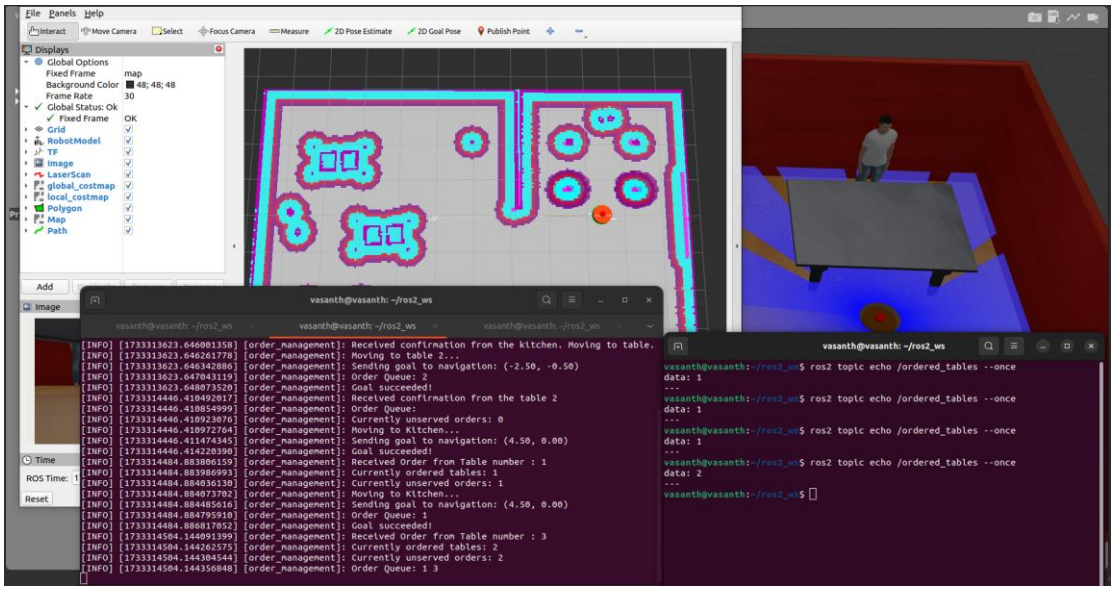
the robot starts to move to the table 2, which is ordered.



➔ Now the robot is reached the table 2, and waiting for the confirmation. We can give the confirmation as

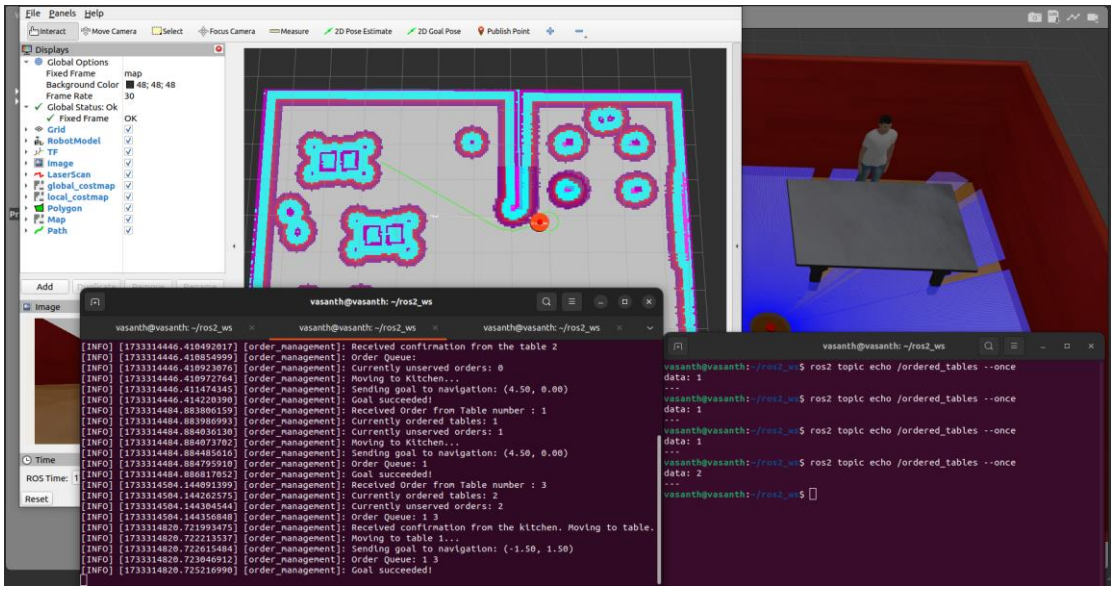
ros2 topic pub /table_confirmation_response std_msgs/msg/String "{data: 'yes'}" –once

then the robot starts move to kitchen to take the food for another next order in the queue.

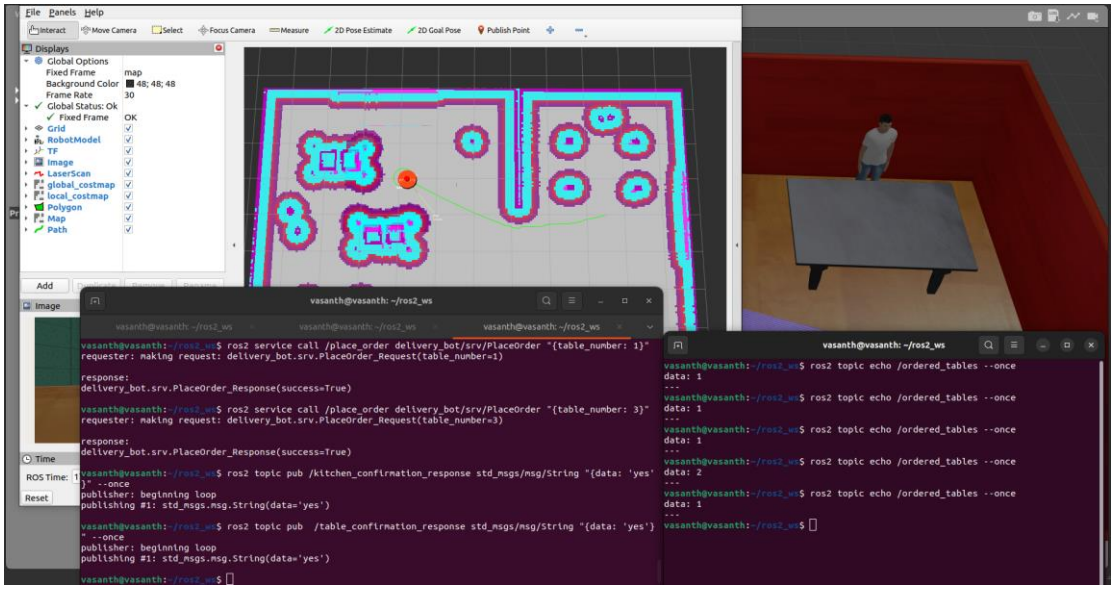


➔ As you can see the above image, While going to the kitchen, there are two more orders placed, table number 1 and table number 3. The queue is updated and the queue_size is also updated. Now there are two tables to serve. Waiting for the confirmation from the kitchen.

➔ When the confirmation is received, the robot starts to move to table 1 to serve and will wait for the confirmation

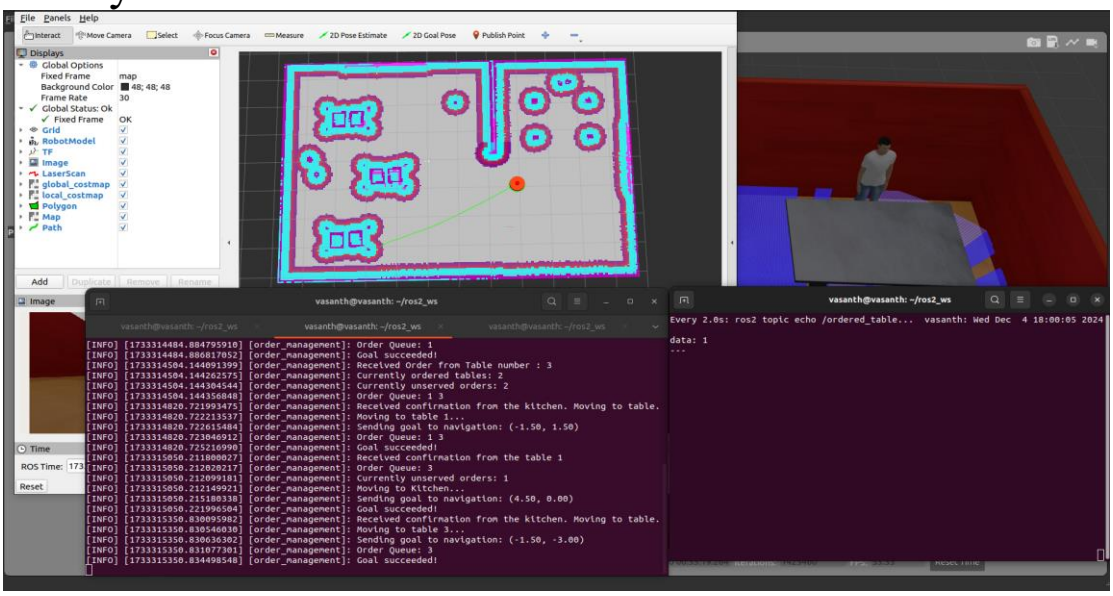


➔ Confirmation from the table 1 is received, the queue and queue size is upated now because one of the two table order is served. starts moving to the kitchen now. And wait for the confirmation to serve another unserved table which is table 3.



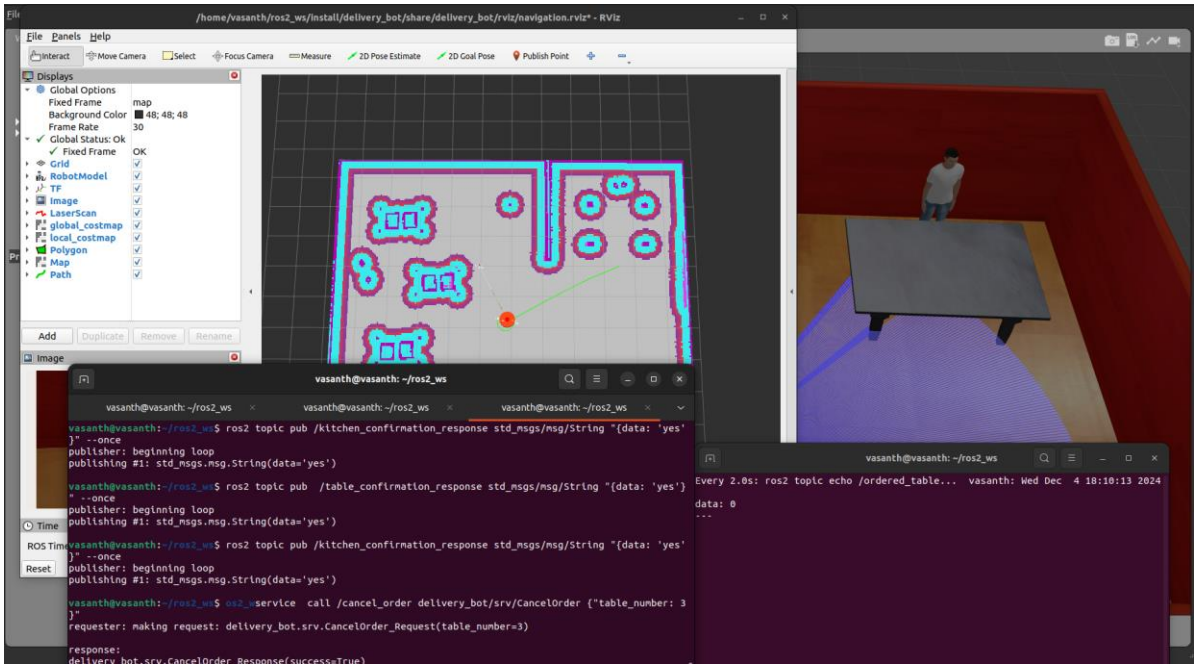
➔ Now what if we cancel the table number 3 order on the way to the table after the confirmation received from the kitchen. Lets see what happens,

at first the robot is en-route to the table 3 to serve, and the queue size is also 1 since the robot is not delivered the food yet.



We cancel the table number 3 order now by

ros2 service call /cancel_order delivery_bot/srv/CancelOrder "{table_number: 3}"



➔ Now the robot turn around and make a goal_pose to the kitchen coordinates. Also the queue and queue size is updated. Now there are no orders to serve. We can give the confirmation from the kitchen as ‘No’ like this

ros2 topic pub /kitchen_confirmation_response std_msgs/msg/String "{data: 'no'}" –once

So the robot can go to Home position and wait for another order to receive and do the task repeatedly whenever the order received or canceled.