Abu Hanif Muhammad Syarubany
20258259

I. INTRODUCTION

This project focuses on developing and evaluating motion planning techniques for a four-wheeled mecanum robot in a simulated ROS–Gazebo environment. The study explores both custom and open-source planners, including a control-space motion primitive planner and the TEB (Timed Elastic Band) local planner integrated with the move_base package. By combining global mapping, local obstacle detection, and dynamic path optimization, the experiments aim to achieve robust navigation and obstacle avoidance in both static and dynamic environments.

II. IMPLEMENTATION

II.1 ***Summary of Learning, Challenges, and Result***

II.1.1 Problem 1

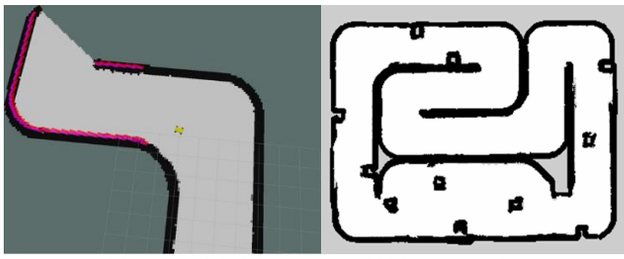

Figure 1 Global Ocuppany Grid Map Generation

In this problem, I built a global occupancy grid map by using depth camera. To build this, I made a new node, occupancy grid mapper, which subscribes to the local costmap node such that I can make persistent mapping. After that, I controlled the robot using teleoperation node to move the robot such that the robot can explore the whole map. Every area that has been explored is saved in the occupancy grid mapper node, such that the whole map can be generated after the robot finishing wandering around. The global occupancy grid map that has been generated is then used on the later problem, especially problem 3 and problem 4 as the global cost map.

II.1.2 Problem 2

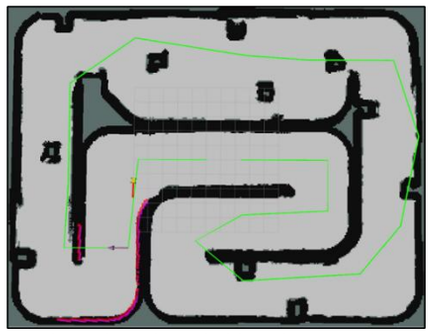

Figure 2 Motion primitives demonstration

Through this experiment, I developed motion planning using motion primitives. In here, the graph planner will send navigation goal one by one with a certain timing out duration to the robot such that robot can approach the navigation goal. As shown in the simulation video, the robot can finish the whole track given by the graph planner. But, there are two collisions happened because the robot could not have an aggressive steering delta if the obstacle was just too close in front of the robot. The challenges throughout this experiment is this problem requires a lot of parameter tuning, such as weight and reward parameter in the cost term calculation. Because of that, I need to do a lot of simulations to find the right value and weights for the parameters. Although there is still a little collision occurred, the robot can still continue completing the task by approaching the navigation goal until finished.

II.1.3 Problem 3

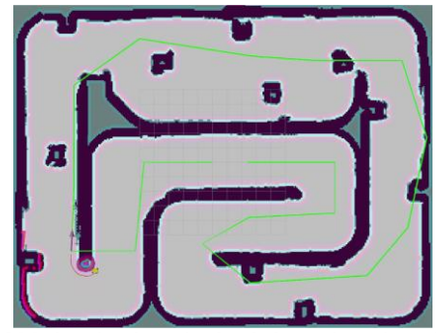

Figure 3 Motion planning using local planner in static environment

Through this experiment, I tried to use other motion planning using opensource local planner, specifically TEB (Timed Elastic Band) planner. In this experiment, the robot successfully finished the trajectory path given by the graph planner without any collision. The challenges throughout the experiment is to tune the TEB parameters such that it can avoid obstacles while doing the task.

II.1.4 Problem 4

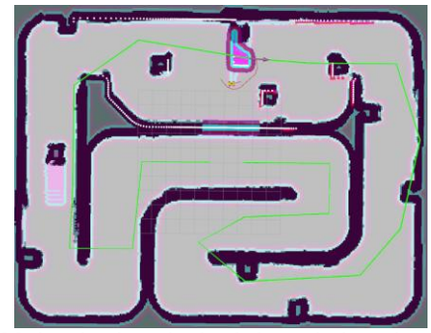

Figure 4 Motion planning using local planner in dynamic environment

Through this experiment, I used the same planner as the previous problem. Instead of having a local cost map layer derived by the depth camera, I used the 2d-lidar scan to have an obstacle layer (voxel layer) with a lethal value such that this layer can track dynamic movement from the moving object with lethal cost and inflation cost value. The challenges here are I need to tune many parameters, calculate many tolerance parameters (TF, planner, and controller), as well as the frequency on each layer. But, throughout the task, the robot sometimes got hit by the moving obstacles such that it ruins the odometry sensors. If that case happened, the robot will be confused and can not continue following the trajectory path. However, most of the time, the robot can accomplish the goal following the whole trajectory without any collision with the probability of 60%, which means that 6 out of 10 simulations that I tried are successful. I address the challenges encountered only by tuning parameters.

II.2 ***Planner Design***

II.2.1 Motion Primitives

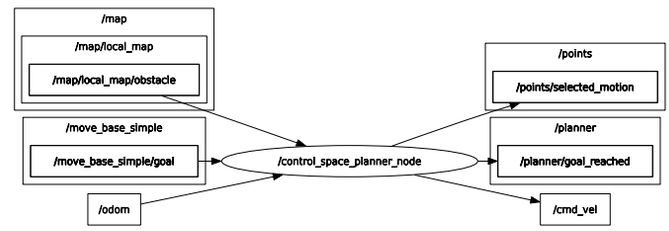

Figure 5 Motion Primitives rqt graph

In motion primitive based planner, the control space planner node is using local map obstacle derived from the depth camera cost map. It can publish `/points/selected_motion` and `cmd_vel` to directly control the robot movement in reaching the navigation goal. The navigation goal is published through `/move_base_simple/goal` which is controlled by the graph planner node. To seamlessly connect with the

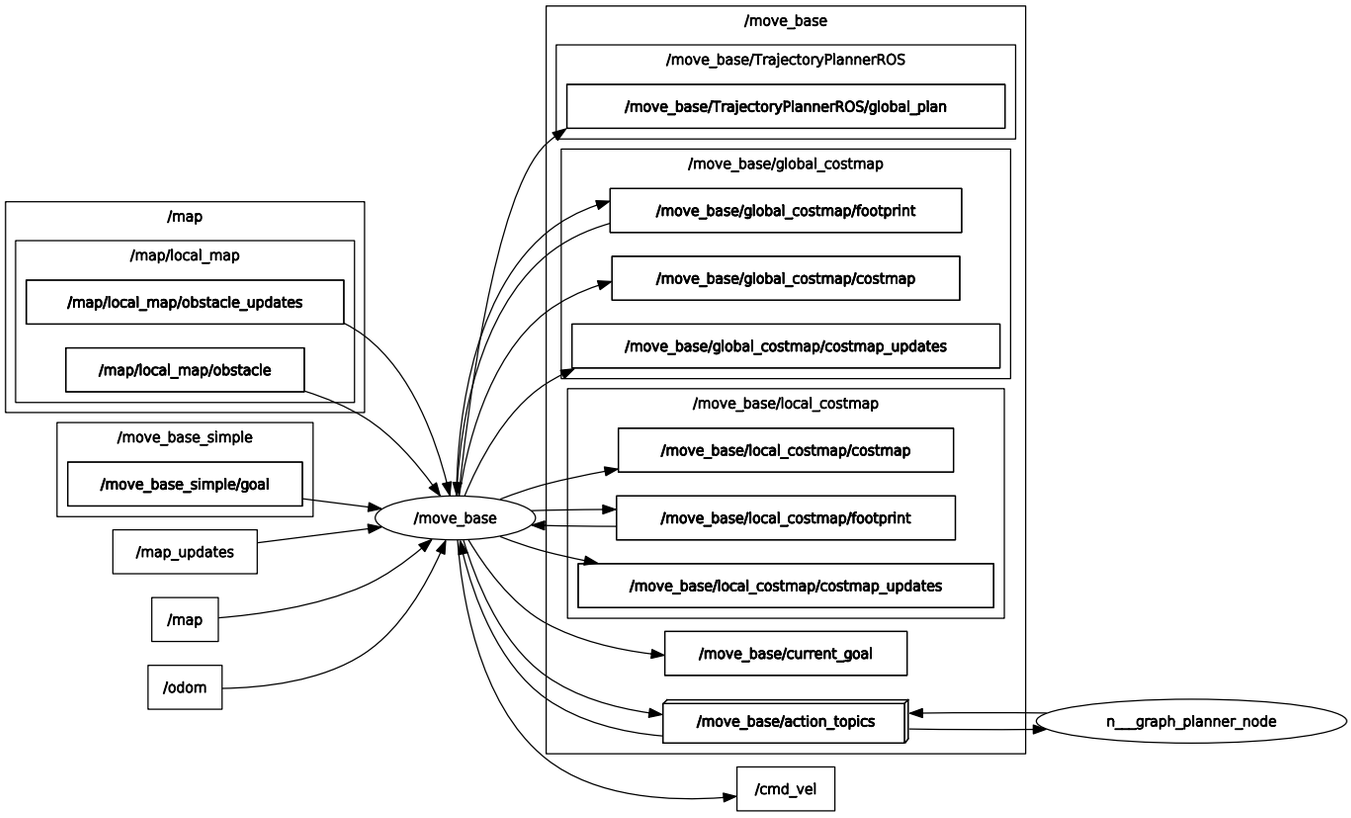

Figure 6 TEB planner (move_base) rqt graph

graph planner node, the planner also publish `/planner/goal_reached` to notify whether the navigation goal has been reached or not. If the goal has been reached, the graph planner will reveal the next navigation goal.

The key methodology to derive motion primitives based planner is cost term calculation. Below is the total cost calculation that I used in the control space planner node:

```
cost_total =
        W_GOAL * J_goal +
        W_HEAD * J_head +
        W_STEER * J_steer +
        W_TRAV * J_trav -
        W_PROG * progress_m;
```

In addition to that cost, there is a reward term if the robot takes an early turning due to a sudden potential collision as follows:

```
cost_total -= W_EARLY_TURN * J_early_turn;
```

This cost calculation is essential because the decision making derived by the motion planner is using these cost terms as reference to select whether the robot needs to go forward or take a turn (steering delta). By using the traversability weight (W_TRAV), every potential collision will be penalized such that the robot will try to avoid that occurrence, and also every progress will be served as a reward with the help of the progress weight (W_PROG).

II.2.2 TEB Planner

Throughout the experiments, especially problem 3 and 4, I used TEB planner utilizing *move_base* opensource packages. As shown in the Figure 6, the planner use 2 different kinds of cost map. The first one is the global cost map which is global occupancy grid map generated in the first problem. This occupancy grid map is served using the map server such that it can publish while streaming the `/map` ROS topic. In the problem 3, the local cost map is derived from the depth camera while the problem 4 is replaced by the obstacle layer derived from the 2d-lidar scan (`/scan` ROS topic). The right-hand side of the diagram shows the internal modules of the move_base node. It illustrates how move_base manages both the global and local costmaps (for global planning and obstacle avoidance) and connects them with the Trajectory Planner (TEB Local Planner) to generate smooth trajectories. These components work together to continuously update the robot's path, compute control commands (/cmd_vel), and communicate with higher-level nodes.

III. Conclusion

Throughout the experiments, the implemented planners successfully demonstrated autonomous navigation and obstacle avoidance using depth camera and LiDAR data. The motion primitive planner offered flexible control but required extensive parameter tuning, while the TEB planner provided smoother, more reliable trajectory generation. Although occasional failures occurred in dynamic conditions, overall results show that integrating global mapping and local planning enables the robot to navigate complex environments efficiently and autonomously.

IV. References

Necessary dependencies:

```
sudo apt-get install ros-noetic-teb-local-planner
sudo apt-get install ros-noetic-navigation
```

Launching simulation references:

```
-> /catkin_ws $ roslaunch main_simulator
P1_Occupancy_Grid_Creation.launch
-> /catkin_ws $ roslaunch main_simulator
P2_Motion_Primitives.launch
-> /catkin_ws $ roslaunch main_simulator
P3_Other_Planner_Static_Env.launch
-> /catkin_ws $ roslaunch main_simulator
P4_Other_Planner_Dynamic_Env.launch
```