Final Project

ME5413 Autonomous Mobile Robotics

Wu Lei	A0263257B
Zhao Rui	A0263567U
Zhu Yueqing	A0263357Y
Wang Zimo	A0263620M

DATE: April 2, 2023

1 Introduction

This report presents the design and evaluation of a robot navigation software stack for a mini-factory environment with three accessible areas and one inaccessible area. The aim of the project is to enable the robot to move from the starting point to a given pose in each area in sequence, including assembly lines, packaging areas, and delivery vehicles. The project is divided into two main tasks: mapping and navigation. In the mapping task, the environment is mapped using our chosen algorithm, and the performance of the SLAM algorithm is evaluated. The navigation task involves navigating the robot through a given sequence of goal poses and evaluating the accuracy of the robot's final pose. This report describes the mapping and navigation pipelines in detail, analyzes their performance using various qualitative and quantitative metrics, and discusses the challenges encountered during the project with proposed solutions.

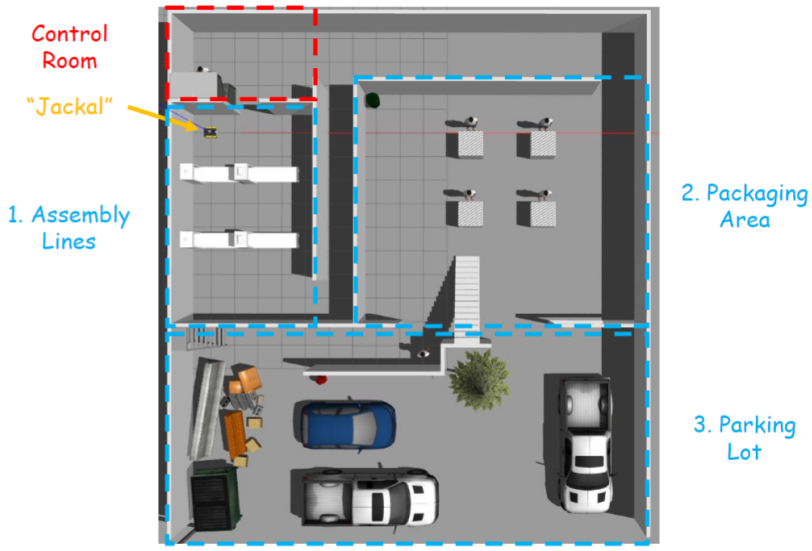

Figure 1: Mini-factory

2 Mapping

In order to generate a map for navigation, it is necessary to build a map through the mapping algorithm In the mapping part of this work, a total of three methods have been tried: 2d-lidar, 3D-lidar and 3D-lidar+IMU. 2D-lidar and 3D-lidar use the ROS built-in gmapping function and A-LOAM algorithm, respectibely. 3dlidar + IMU uses FastLIO, Lego-Loam and LIO-SAM algorithm.

2.1 2D-lidar Algorithm

Firstly, we try to utilize the gmapping algorithm to generate the map. The gmapping algorithm is an essential tool for mapping unknown environments using a mobile robot. This is a probabilistic algorithm, utilizes 2D lidar sensor data to estimate the robot's position and orientation in the environment, allowing it to create an accurate map over time. The map generated by the gmapping is shown in the Figure 12, as is shown in the image, the generated map can not match well with the current point cloud, which means that it is hard to navigation in this generated map.

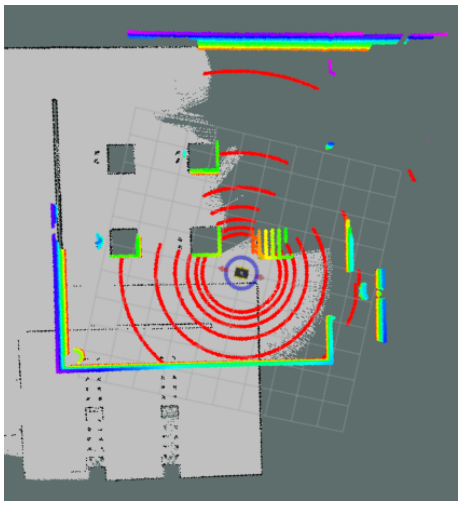

Figure 2: Map of Gmapping Method

After complete the mapping, it is necessary to evaluate the performance of the algorithm, the Figure 3 shows the mapping error of the Gmapping.

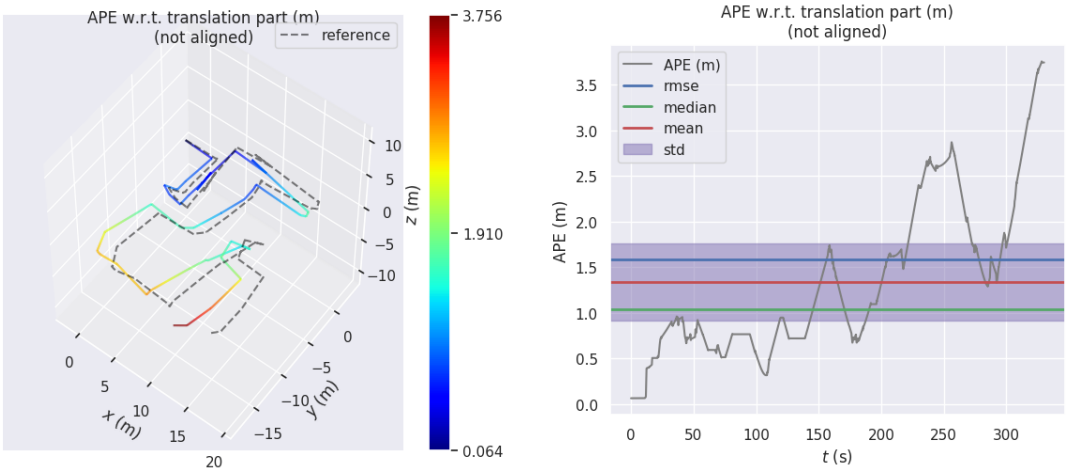

Figure 3: Error of Gmapping

the error of this method is so high that the estimated path has deviated from the ground truth, which means that the map can not be used for path planning. Furthermore, in

the Gazebo world there is a window and a stare case, but the 2D lidar cannot recognize them very well, the map will let some pixels that the robot cannot enter be the free area.

2.2 3D-lidar Algorithm

After using 2D lidar mapping method, we try to utilize the 3D-lidar of the robot to reconstruct the world in point cloud and turn it into 2D navigation map. In this work we utilize the A LiDAR Odometry and Mapping Algorithm(A-LOAM) algorithm, this algorithm uses a combination of point-to-plane and point-to-point optimization methods to estimate the pose and map the environment. It also incorporates an adaptive motion compensation mechanism to account for vehicle motion and LiDAR sensor errors. The Figure 4 shows the mapping effect of the A-LOAM algorithm, the A-LOAM can reconstruct the terrain very well by receiving the data from 3D lidar, but there is a problem that the A-LOAM the ground has slop in the corridor, which makes the ground upwarping (As is shown in the right), which makes the ground be recognized as obstacles during transform the 3D point cloud to the 2D map and make navigation become so hard. So it is necessary to find a better method to do the mapping.

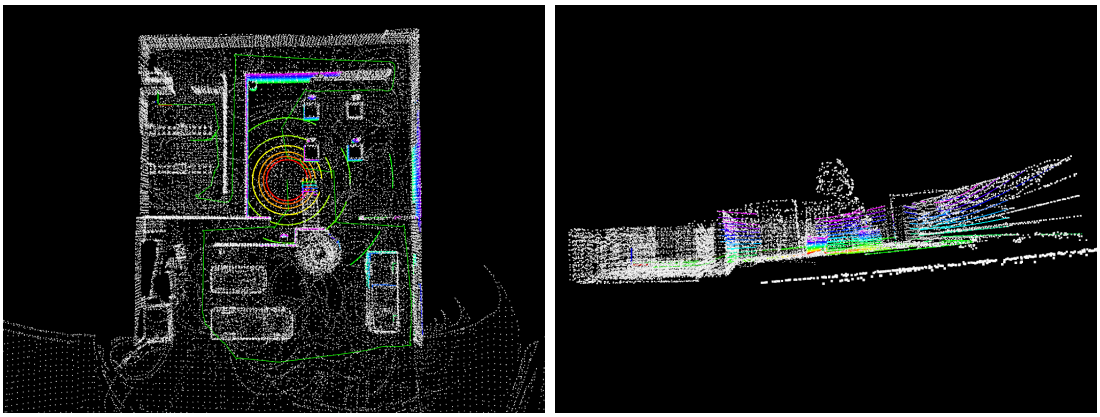

Figure 4: Map of A-LOAM Method

After conduct mapping we evaluate the performance of the algorithm, as is shown in the Figure 5, the error almost comes from the slop of the ground. It is necessary to reduce the upwarping of the ground.

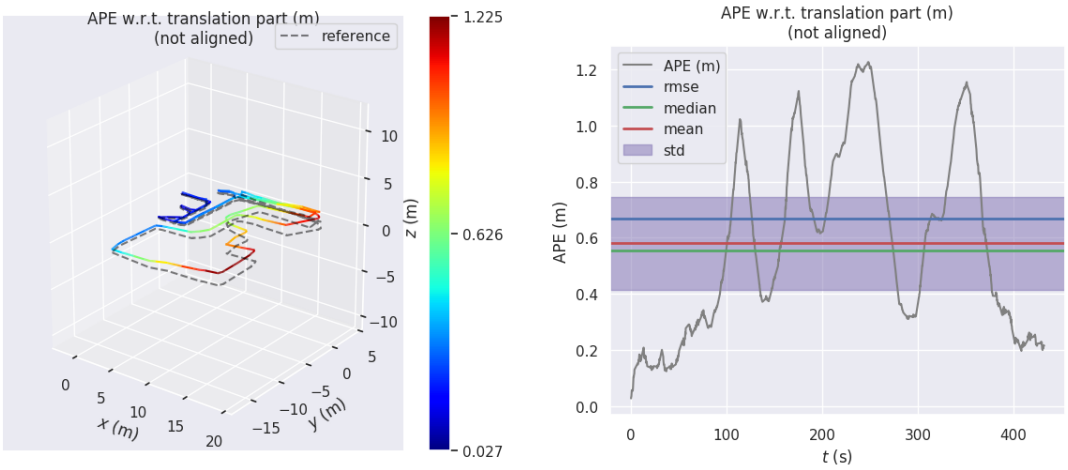

Figure 5: A-LOAM

2.3 Multi-sensor Algorithm

Then we turn to take advantage of the multi-sensor algorithm, in this work, there is an IMU sensor in the robot, so we try to search the mapping algorithm utilize 3D lidar and IMU sensor, since IMU contains the physical data during the robot's moving, which can help to build the ground. After trying the Fast-LIO, Lego-LOAM and LIO-SAM algorithm, we finally choose the Fast and Lightweight LiDAR Odometry(Fast-LIO) algorithm, which perform most steady in our CPU(CPU cannot calculate fast will cause drift of mapping). FastLIO is a real-time visual-inertial odometry system that fuses data from a 3D LiDAR sensor, an inertial measurement unit (IMU). This algorithm is designed to be computationally efficient, allowing for real-time performance on low-power processors.

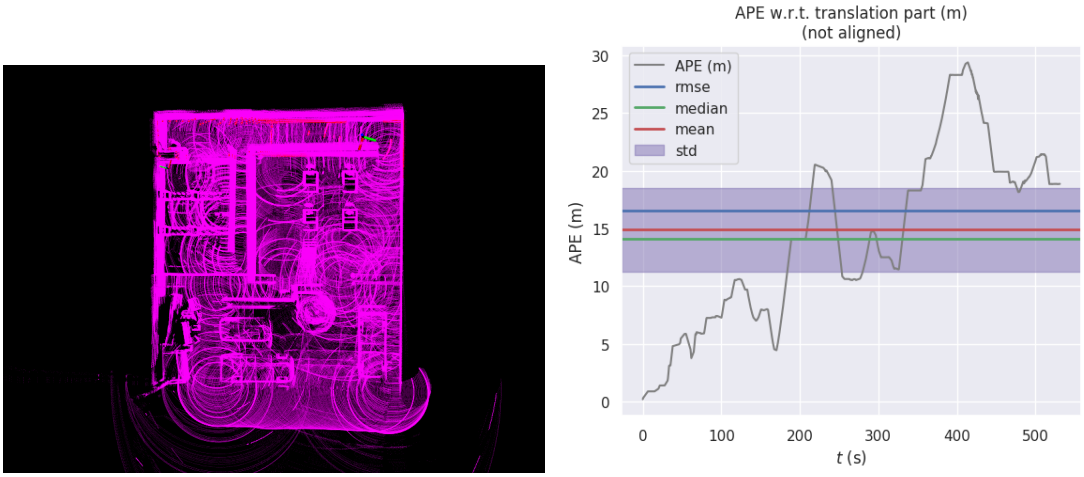

Figure 6: Map and Error of Fast-LIO

As is shown in the Figure 6, The Fast-LIO method can reconstruct the terrain feature very well, although the map is still upwarping, it is much better than A-LOAM. However,

there is a problem that the constructed map has a rotation about 75°, the error almost comes from this rotation, after trying to check the code of Fast-LIO, we still cannot find the problem is.

2.4 Generation of 2D Map

After conduct the mapping task, it is necessary to turn it to the 2D map for navigation, we select the point cloud map generated by the Fast-LIO algorithm. But if just apply the point cloud map as 2D map directly, it will makes the navigation very hard, so we let the point cloud rotate around the origin point about 75° to match the ground truth. Then the point cloud map is down sampled in order to reduce the points' density, Figure 7 and Figure 8 shows the effect of different down sample factors. In this work we choose $downsamplefactor = 0.2$

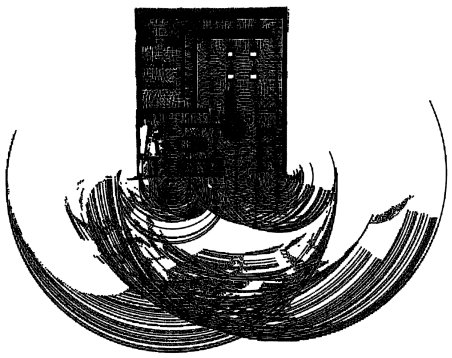

Figure 7: Down Sample Factor= 0.2

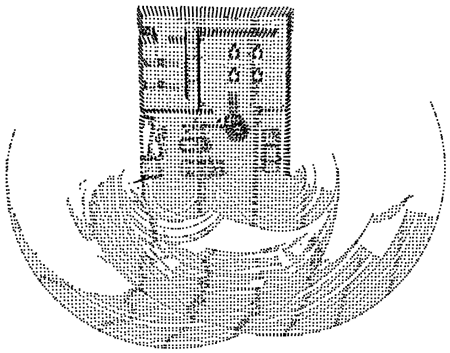

Figure 8: Down Sample Factor= 0.5

After get the down sampled point cloud map, in order to reduce the influence of the upwarping of the ground, we utilize the Random Sample Consensus(RANSAC) algorithm, which is an algorithm for removing the ground plain. As is shown in the Figure 15 and Figure 16, the map using RANSAC to remove the ground plain will reduce the ground be recognized as the occupied area. Furthermore, in order to remove some noise points, the radius filtering is used, this method recognizes the noise points by checking the number of its neighbors in the fixed radius. As is shown in the map, there is still some problems. Although the RANSAC can remove ground plane, the algorithm will also remove some barriers that not so high such as the windows, it order to solve this problem, we just apply all the point cloud to generate the map. So the whole tree will be recognized as the obstacle, despite the robot can move under the tree.

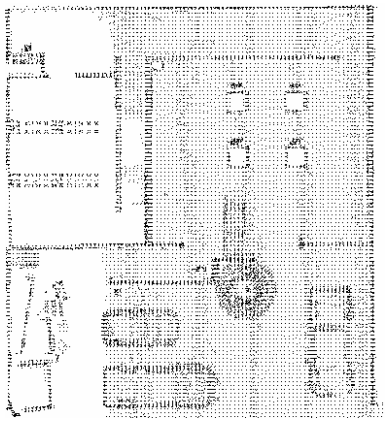

Figure 9: Generated Map without RANSAC

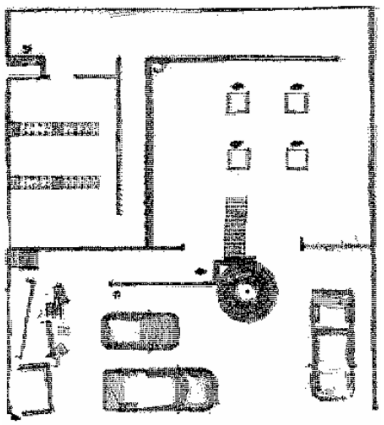

Figure 10: Generated Map with RANSAC

3 Navigation

Localization and navigation are essential components of mobile robot systems. Localization allows the robot to determine its position and orientation in the environment, while navigation enables the robot to plan a path from its current location to a goal location. In this part, we will discuss the framework of localization and navigation in mobile robot systems, which consists of odometry and AMCL for localization, and global and local planners for navigation. Moreover, we will introduce the algorithms used in global and local planners, including Dijkstra, A* (A star), DWS, and TEB.

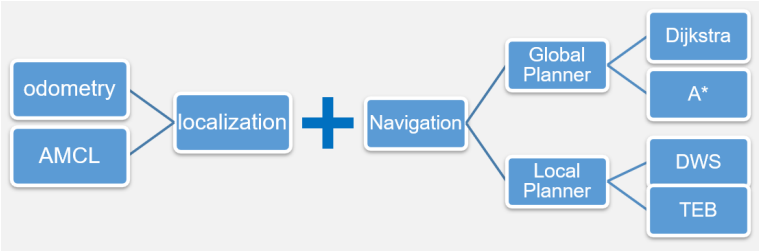

Figure 11: Mindmap of navigation

3.1 Localization

Localization is the process of estimating the robot's position and orientation in the environment. Odometry is a method commonly used for estimating the robot's position and orientation based on the measurement of its wheels' motion. However, odometry will accumulate errors over time due to wheel slippage or uneven surfaces. To overcome this problem, we can use AMCL (Adaptive Monte Carlo Localization), which is a particle filter-based algorithm that estimates the robot's position and orientation by comparing sensor measurements with a map of the environment. It can provide accurate and robust localization even in dynamic environments.

3.2 Navigation

Navigation is the process of planning a path for the robot from its current location to a goal location. It can be divided into global and local planning. Global planning involves finding a long-term path from the start location to the goal location. Besides, local planning focuses on finding a short-term path that avoids obstacles and keeps the robot on the global path. The description of global and local planner are shown below :

GlobalPlanning Global planning can be performed using various algorithms, such as Dijkstra and A*. Dijkstra's algorithm is a graph search algorithm that finds the shortest path between two nodes in a graph. A* is a heuristic search algorithm that expands nodes in a graph based on their estimated distance to the goal. A* is more efficient than Dijkstra's algorithm when there are obstacles in the environment.

LocalPlanning Local planning can be performed using algorithms such as DWS (Dynamic Window Approach) and TEB (Timed Elastic Band). DWS is a method that considers the robot's kinematics and environment constraints to generate a set of candidate trajectories. It evaluates the trajectories based on their feasibility and proximity to the goal and selects the best one. TEB is a method that generates a continuous trajectory by balancing between the robot's dynamic constraints and the environment constraints. It provides smooth and safe paths for the robot.

3.3 MOVE_BASE Package

The MOVE_BASE package is a software package widely used in robotics for autonomous navigation. It is composed of several key components, including localization, planners, costmap, navigation stack setup, and controller message. In this report, we will introduce the framework of the MOVE_BASE package and focus on the costmap component.

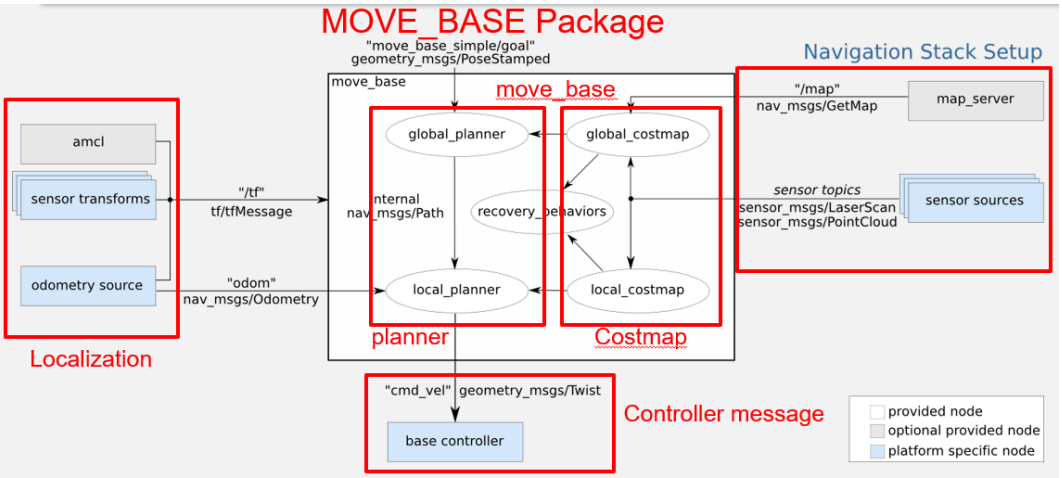

Figure 12: Move base package

Planners Planners are responsible for generating a path for robot to follow and reach its destination. In the MOVE_BASE package, planners are composed of two parts: global_planner and local_planner. Base_local_planner is a package we used for local planning that supports constraints in x and theta. While it provides some support for y velocities, users are limited to a pre-specified list of valid y velocity commands. It is a good choice for our robot system as our robot has no speed in the y direction. The package includes the DWA (Dynamic Window Approach) option, which is also present in the dwa_local_planner package. However, the latter is essentially a re-write of the DWA option in the base_local_planner and is considered cleaner and easier to understand. As is shown in Figure 13, as the A* planner will plan sharp bends at the corner, which is not suitable for robot's movement. In this task, Djikstra planner is chosed as global planner.

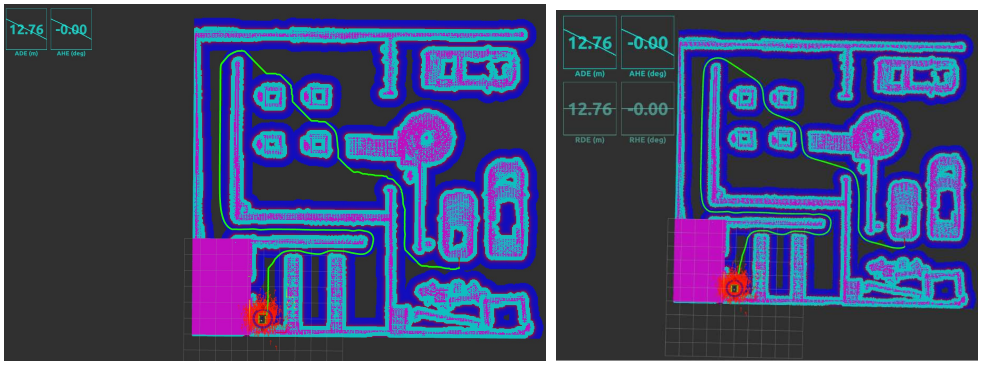

Figure 13: A* path and Djikstra path

Costmap The costmap is a significant component of the MOVE_BASE package that provides a representation of the environment that the robot can use for navigation. It is composed of two parts: global_costmap and local_costmap. The global_costmap is a map that represents the robot's entire environment and is used by the global planner to generate a long-term path. The global_costmap assigns a cost to each cell in the map based on the obstacles and other environmental features present in that cell. The local_costmap is a smaller map that represents the robot's immediate surroundings and is used by the local planner to generate a short-term path. The local_costmap is created by combining sensor data with the global_costmap. The sensor data is used to update the local_costmap with the current environment, and the global_costmap is used to provide a long-term context for the robot's navigation.

Localization and Navigation In the MOVE_BASE package, localization is achieved using AMCL (Adaptive Monte Carlo Localization), sensor transforms, and odometry source, where sensor transforms are used to convert the sensor data from its local coordinate frame to the robot's coordinate frame, odometry source is used to estimate the

robot's motion based on its wheel encoders. The navigation stack setup is responsible for configuring the robot's sensors and other components to enable autonomous navigation. It is composed of two parts: map_server and sensor sources. The map_server component loads the map of environment, while sensor sources provide sensor data to the system.

3.4 Result

After researching and attempting multiple options, we ultimately chose the Dijkstra algorithm as the global planner and the default base_local Planner (which is similar to DWA) from the move_base package as the local planner. By adjusting and selecting various parameters such as move_base.launch, local_cost_params, and global_cost_params, we were able to obtain an ideal solution.The Figure 14 shows the position and heading error of a sequence goal pose during the planning.

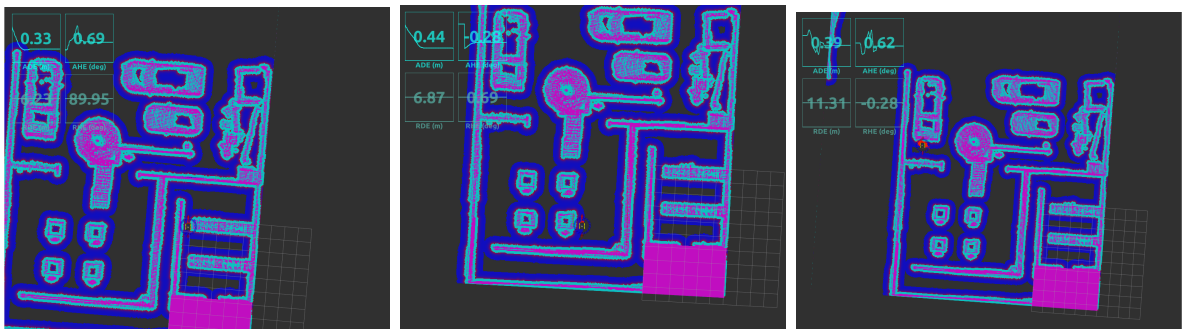

Figure 14: Position and heading error of a sequence goal pose

3.5 Problems and Solutions

To make the jackal robot finish the navigation task well, we have enconunted many problems such as the robot hit the objects and so on. After check out pretty much of sources and trial, we find all the problems related to four aspects: costmap, relationship between global and local planner, calculation efficiency, and Sensor accuracy. Specific problems encountered will be analyzed based on these aspects and provide detailed solutions.

Hitting an object resulted in navigation failure At first, we find out the robot always hit the wall or other objects. Since we cannot make the map 100 pecents accuracy and there is some error about the localization, we have to make the robot path keep a suitable distance to the objects. As a result, we increased the inflation radius of the global costmap and deceased the weight of local planner, which make sure the robot run in a safer path and reduced the probability of collision.

Rotate or stop When the passage is too narrow This question is a little same as the question above, when the robot cannot find a least cost path or cannot localize

itself, it will stop, rotate or backWord.we can increase the inflation radius of the global costmap and decrease the inflation radius of the local costmap.

Slow turning speed After the robot is able to complete all tasks, we found that its turning speed is very slow. After consulting the literature, we found that this is mainly related to two factors: first, the need to increase the sampling frequency of the angular velocity, and second, the detection range of the local planner is too large, which requires a long time for calculation. So we need to decrease the detection area of the local planner.

Set up prohibition areas on the map In order to achieve the task of prohibiting the robot from entering the control room, we came up with multiple solutions. At first, we planned to directly modify the file by setting the costmap value of the control room area to 255. However, this would affect the AMCL localization. Therefore, I used the prohibition_layer plugin and successfully solved this problem.

Figure 15: Set up prohibition areas

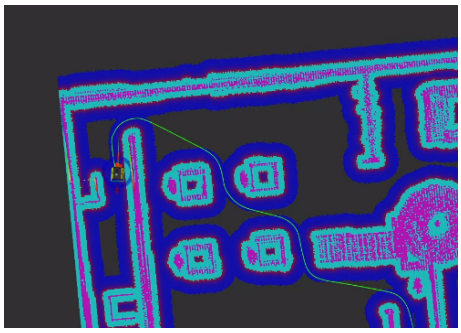

Figure 16: Identify new obstacle

Improve location accuracy To improve the location accuracy, we tried to modify the parameters of AMCL in amcl. launch. But it does not work at all. And we also tried using lidar data as the input of amcl algorithm. But restricted to our computer performance, it is hard to run the whole proceed smoothly, so we give up finally.

Identify new obstacles and navigate around them At the beginning, our robot cannot find out the new obstacle. After we check out the code, we find out our obstacle plugin subscribe a wrong topi. And after we change the topic name back to "tim551", it works well.

When the car is stationary, it rotates and the costmap recognition errors When the car is stationary in the simulation environment, we found that it rotates slowly in rviz. After consulting the literature, we found that this is likely due to errors and interference in the IMU. By configuring the IMU in the gazebo settings and resetting it, we successfully solved the problem and confirmed that the rotation of the car was caused by IMU errors. However, after using imu_utils to calibrate it, there was no improvement in the result.