ME5413-Final Project

Group number:19

Group member:

LI BO: A0269522Y

XUE HAOYAN: A0268454U

TIAN YUHONG: A0268423B

LI ZONGRUI: A0268535U

Github:

https://github.com/ChaoXi845/ME5413.git

Content

Task 1: Mapping

First of all, according to the requirements of the topic, it is necessary to create a static map for the actual map in the gazebo.

1.1 Confirm the way to build the map: online and offline.

1.1.1 Online map building

1. Real-time: Online mapping can build maps in real-time, so it can quickly adapt to environmental changes and make robots more robust in dynamic environments.

2. Small footprint: Online mapping only needs to store the map at the current moment, so the requirement for storage space is relatively low.

1.1.2 Offline mapping

1. Low computing load: Offline mapping does not require real-time data processing, so less computing resources are required.

2. It takes up a lot of space: Offline mapping needs to store all the collected data and maps, so the requirements for storage space are relatively high.

3. Inability to quickly adapt to environmental changes: Since offline mapping needs to be completed before processing, it cannot quickly adapt to environmental changes, making the robot not robust enough in a dynamic environment.

To sum up, in order to collect more details of the map, we chose the method of manual online mapping. When manually building a map, we drive along as many map features and the edges of different terrains as possible, which can better capture the environment. Details and features, improve the accuracy and completeness of mapping.

1.2 Mapping route planning

When manually constructing a map online, the selection of the route has an important impact on the final map result. The selection of the route will directly affect important indicators such as the accuracy, completeness, and coverage of the map.

1.2.1 Map accuracy

When choosing a route, priority should be given to the area to be established and the accuracy requirements of the target map. For smaller maps and maps that require high precision, a more detailed path should be selected as much as possible so that the robot can detect the surrounding environment more accurately.

1.2.2 Map completeness

When choosing a path, it should try to cover the entire mapping area as completely as possible, so that the robot can complete the map construction without missing any areas.

1.2.3 Coverage

The coverage of a path refers to the proportion of the area that the robot perceives when driving through the path. When choosing a path, try to choose a path that maximizes the coverage area so that the robot can perceive the surrounding environment more fully.

1.2.4 Movement efficiency

When choosing a path, the movement efficiency of the robot should be considered so that the map construction task can be completed as quickly as possible while ensuring map accuracy, integrity, and coverage.

Based on the above considerations, we determined the route as follows:
（Where the yellow route is the return road (the repeated road taken)）

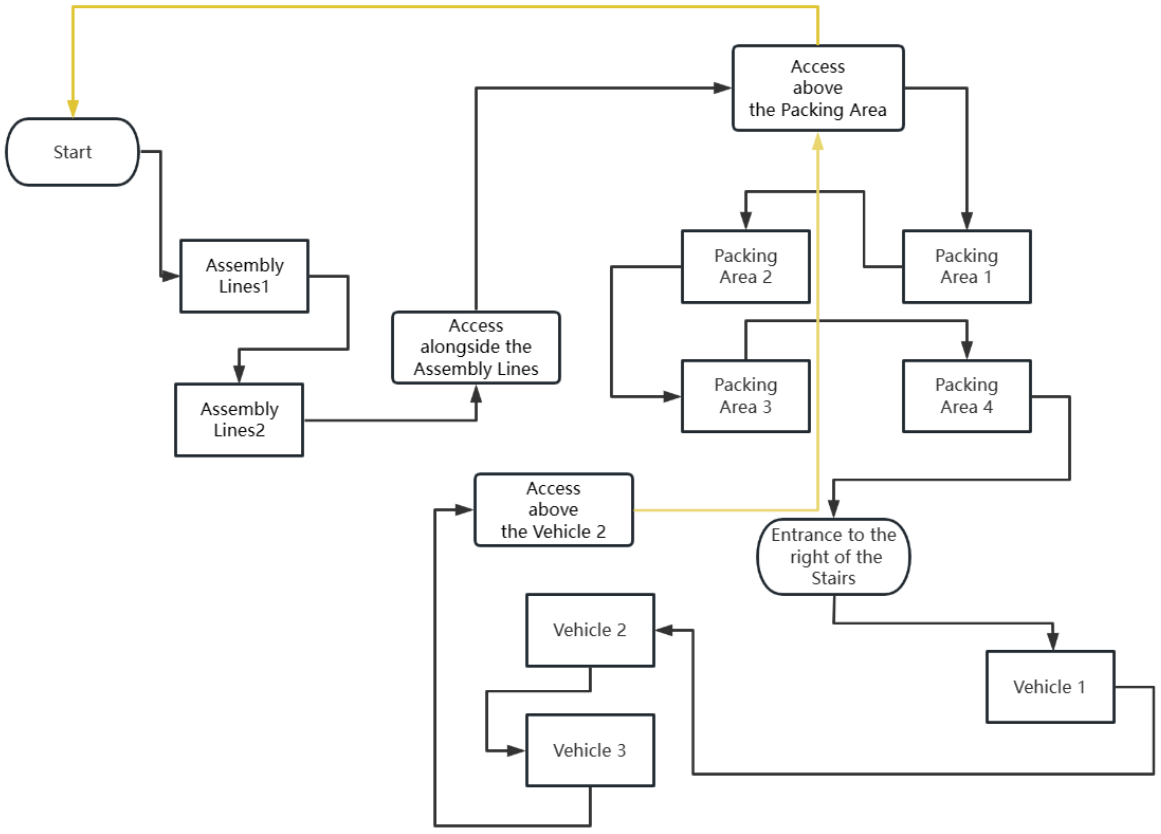

Figure1. Mapping route chart

1.3 Confirm the method of mapping, improved Gmapping, and improved cartographer.

1.3.1 Gmapping's pipeline:

1. Collect data from robots through sensors, which will be used to build maps.
2. Send the data to the gmapping node through the ROS node.
3. The gmapping node receives sensor data and converts it into a data type built into ROS.
4. The gmapping node processes the data using the SLAM algorithm and builds a 2D map around the robot.
5. Adjust the parameters of Gmapping according to the effect and re-contrast the mapping, and select the one with the best mapping effect as the result and save it.

1.3.2 Cartographer's Pipeline:

1. Use online mapping to collect environmental data with sensors.
2. Send the data to the cartographer node through the ROS node.
3. The cartographer node receives the laser sensor data and uses the SLAM algorithm to build a map around the robot.
4. Optimize the parameters of Cartographer, re-construct and compare, and select the best result to save.

1.4 EVO evaluation

According to the title, Groundtruth is stored in /gazebo/ground_truth/state topic

1. View all topics and data formats, using the command:

```
Rostopic list -v
```

2. Find the topic in *nav_msgs/Odometry* format, and find that Estimated odometry is stored in */odometry/filtered* topic,

3. Record bag file, use the command:

```
rosbag record /gazebo/ground_truth/state /odometry/filtered -o bag_filename.bag
```

4. Use EVO to evaluate SLAM accuracy, choose to evaluate absolute pose error, and use the command:

```
evo_ape bag bagfile_name.bag /gazebo/ground_truth/state /odometry/filtered -a ---plot --plot_mode=xy
```

1.4.1 Map

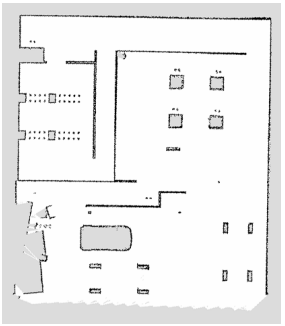

Figure.2 Map from Gmapping

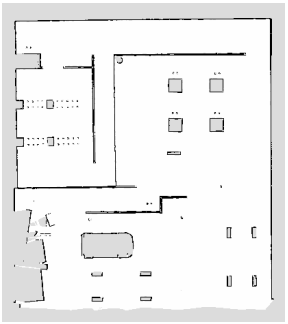

Figure.3 Map from Cartographer

As shown in the figure, there is no significant difference between the maps generated by the two SLAM algorithms. It can be seen that the density of the landmark feature in the map built by cartographer is slightly large. Moreover, the performance gap is relatively large when later using the evo tool to evaluate SLAM performance. This may be because gmapping uses a particle filter algorithm, therefore it can achieve very high map accuracy in a small area, which is suitable for high-precision map construction in a small area, while cartographer uses an optimization-based graph optimization algorithm, which means it might be more suitable for map building in a large area.

1.4.2 EVO result

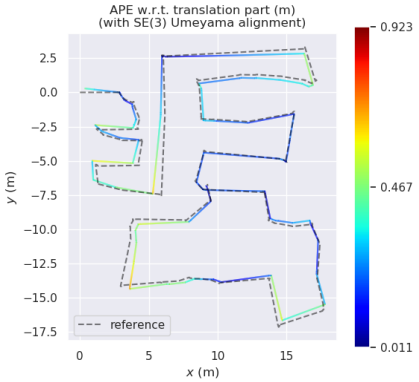

Figure.4 EVO Result for Cartographer

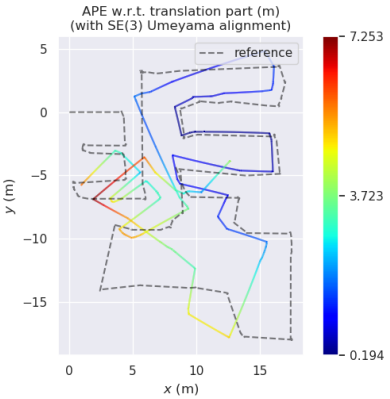

Figure.5 EVO Result for Gmapping

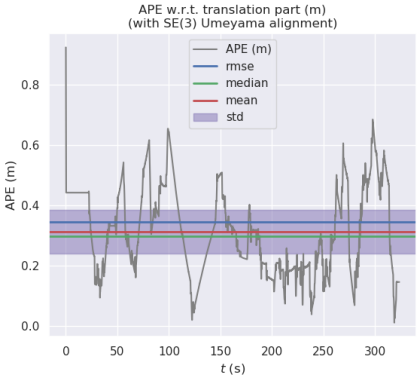

Figure.6 EVO Result for Cartographer

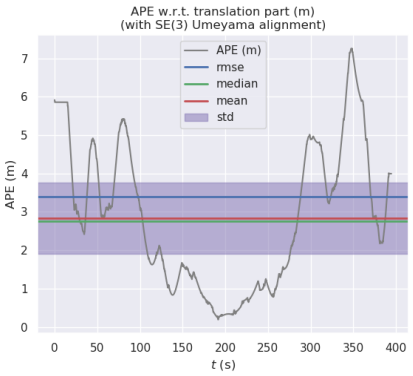

Figure.7 EVO Result for Gmapping

As shown in the figure, the absolute trajectory deviations of the two SLAM algorithms are quite different. In the Cartographer evaluation diagram, the estimated trajectory is almost consistent with the actual trajectory. While for the gmapping algorithm, there are varying degrees of angular deviation between the estimated trajectory and the actual trajectory.

Table1. EVO Evaluation Data Comparison Tables

SLAM Algorithm	Cartographer	Gmapping
max /m	0.922793	7.253055
min /m	0.312658	2.840217
rmse /m	0.343919	3.395774

As shown in the table, from the three important parameters of maximum absolute trajectory deviation, minimum absolute trajectory deviation and root mean square error, the accuracy of SLAM using cartographer in this usage scenario is relatively high.

1.5 Problems and solutions

1.5.1 The mapping range is too short

When starting to build the map, the robot cannot locate the front wall and the back wall at the same time in one space, which leads to an increase in positioning error and imprecise subsequent robot path planning.

Solution: Enter the .lua file settings to raise the lidar scanning range parameter to increase the maximum scanning range.

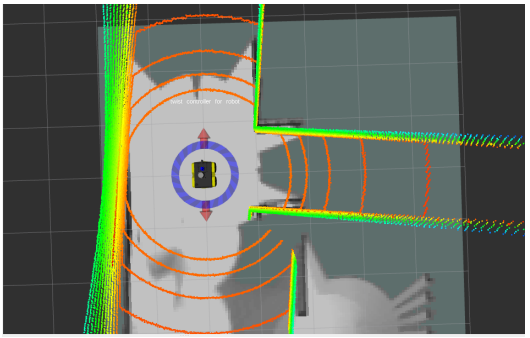

Figure.8 minor lidar max range

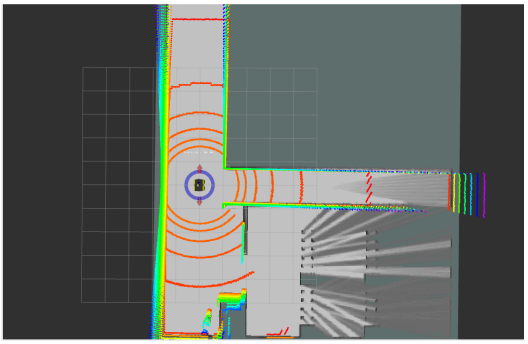

Figure.9 large lidar max range

1.5.2 Noise and distortion in the process of mapping

When composing a map, there is too much noise in the map process, which affects the accuracy of the map. In addition, due to the matching error between submaps, the map is slightly distorted and the map is locally inconsistent.

Solution: Raise the "submaps.num_range_data" parameter to make submaps distinct enough to improve loop closure.

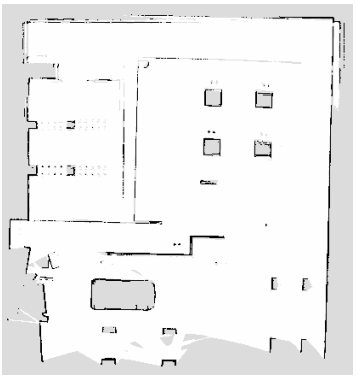

Figure.10 Map with small submap range

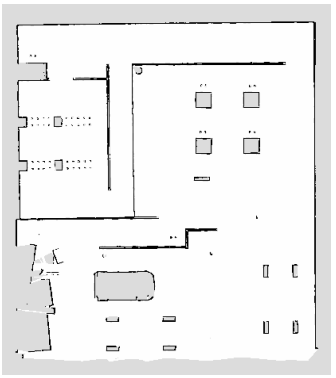

Figure.11 Map with large submap range

Task 2: Navigation

After getting the ideal map created by cartographer, we use NAVFN and Time-Elastic-Band(TEB) for global path planner and local path planner respectively to navigate. To get a better result, we modify some parameters of the AMCL algorithm.

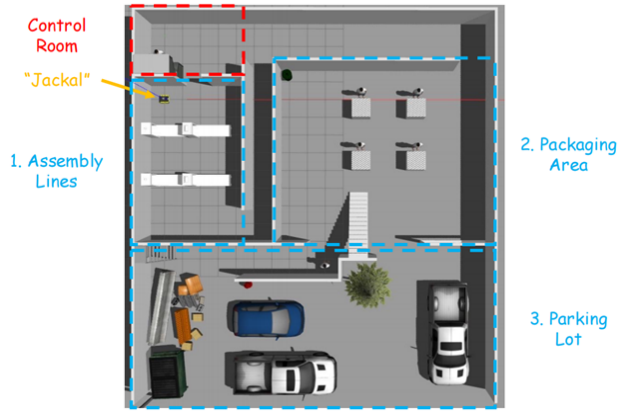

Figure.12 gazebo world

2.1 Local cost map and Global cost map

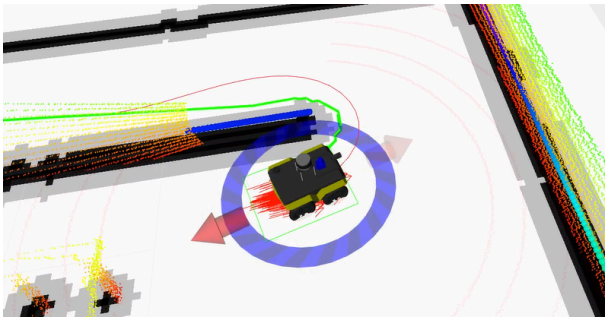

Figure.13 Local cost map route(red) and Global cost map route(green)

In the navigation algorithm, a local cost map and a global cost map work together to achieve control and path planning for the robot. The local cost map is usually a small area around the robot's current location and is used primarily for local path planning and obstacle refuge. The global cost map contains obstacle information for the entire working area of the robot and is usually generated from a pre-acquired map. This map is mainly used for global path planning and global awareness of the robot's surroundings.

The route plotted by the global cost map (the green line in the additional video) is constantly updated as the robot's current position changes, i.e. for each unit of robot movement, the plotted route is replotted as the relative positions of the current point and the target point change, in real-time.

The local cost map is a map that moves with the robot's position, and the intersection of the global cost map and the local cost map is used as the current target point of the local cost map to plan the path for the local area where the robot is located until the target point is reached (red line in the additional video).

In navigation, the global cost map is typically used first to plan the robot's approximate path, while the local cost map is used to adjust the path to avoid obstacles as the robot moves. The local cost map is updated in real-time to reflect changes in the robot's surroundings when the robot encounters an obstacle or when the path needs to be re-routed.

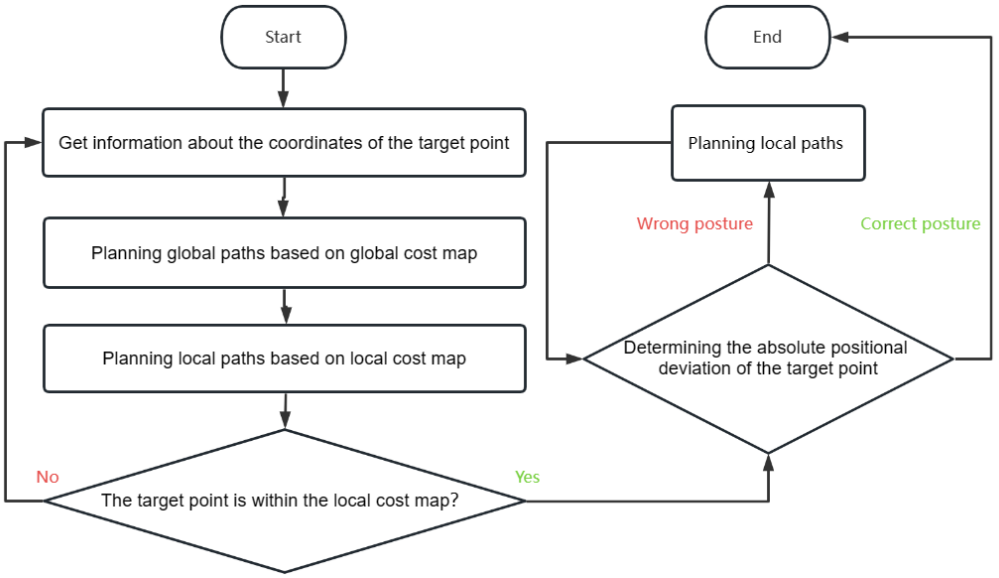

Figure.14 The flow chart of navigation

2.2 Path plan algorithm

2.2.1Global path plan (Navfn) :

During global planning, the Navfn algorithm can quickly calculate the global optimal path and can update the path in real-time to adapt to changes in the environment and can be used in conjunction with other algorithms in the ROS navigation stack (e.g. local cost maps) to form a complete path planning system.

2.2.2 Local path plan (TEB) :

TEB is suitable for differential driving and even car-like robots. It incorporates the time dimension into trajectory planning and can achieve smoother trajectory planning through time optimization and is

also able to rationally adjust the robot's speed and direction of movement based on the predicted motion of dynamic obstacles. It can judge dynamic obstacles effectively.

The following pictures(Figure.15) show the constitution of our map in ROS Rviz.

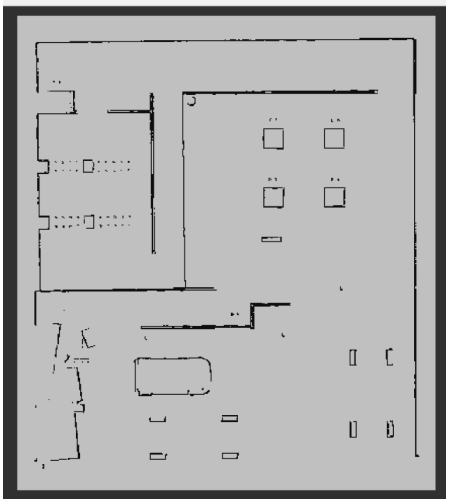

Figure.15(a) Initial Map

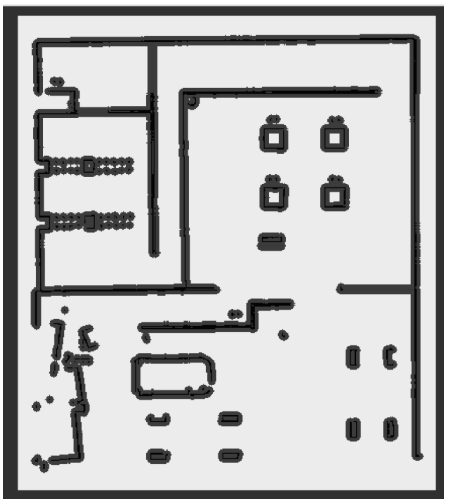

Figure.15(b) Global Costmap

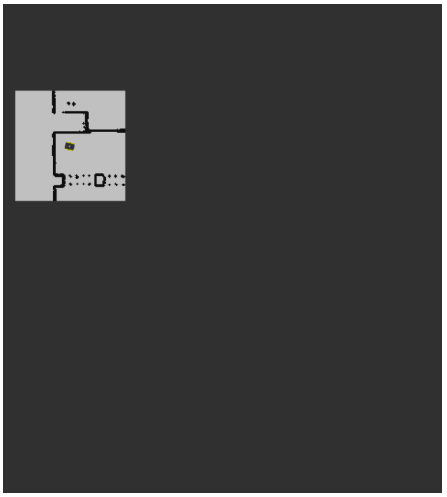

Figure.15(c) Local Costmap

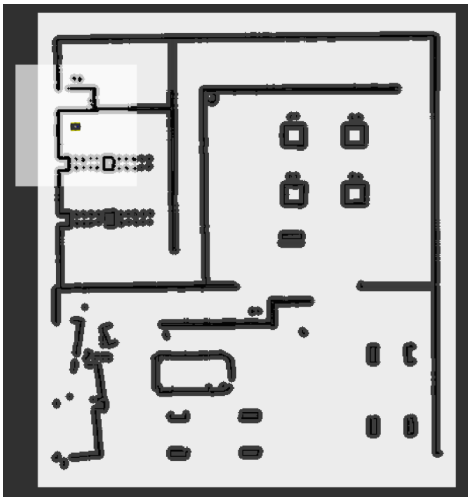

Figure.15(d) Final Map

2.3 Problems and solutions

2.3.1 Localization: Pointclouds do not match the Map

When we first tested the navigation step, we found the pointclouds did not match with the obstacles shown in the map. After the robot running, the coincidence became better but also not ideal enough, this situation impacted our goal position.

Solution: modify the params of AMCL algorithm.

There are some params we modified:

param	default	our value
max_particles	2000	5000

odom_alpha3	0.2	0.8
update_min_a	0.1	0.2
transform_tolerance	1.0	0.1
recovery_alpha_slow	0.0	0.001

Followed picture shows the results before and after the modification, it is obvious to see the initial AMCL result that the pointclouds of the car is not coincident with the obstacle in the map(Figure.16). After modifying the params, the pointclouds coincide with the obstacle ideally(Figure.17):

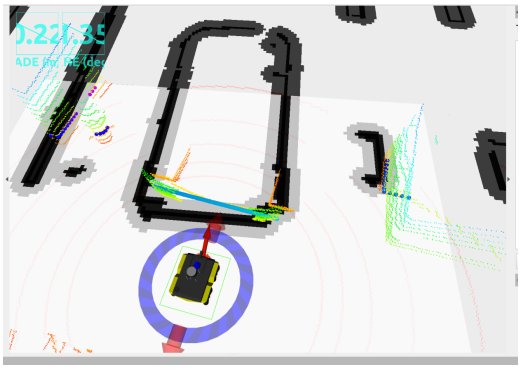

Figure.16 Initial AMCL result

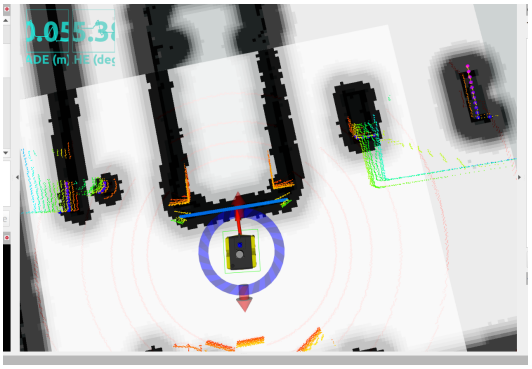

Figure.17 Modified AMCL result

2.3.2 Inaccessible or undetected areas

In this task, there are some areas that cannot be accessible or be detected by the optics sensor: the operation room and the glass wall(transparent). For these areas, we create an additional costmap to help the global planner during navigation.

The ROS system provides a plugin tool for users to plug their own algorithm. We use this tool to plug an additional costmap layer, named cost_prohibition_layer(CPL). This layer does not appear in the 'real' map. It is a kind of costmap, when the robot uses the navigation algorithm to plan the path, it will consider the prohibition areas as obstacles to avoid.

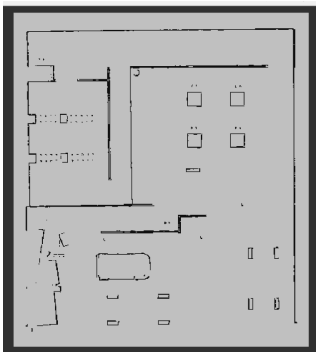

Figure.18 Real Map

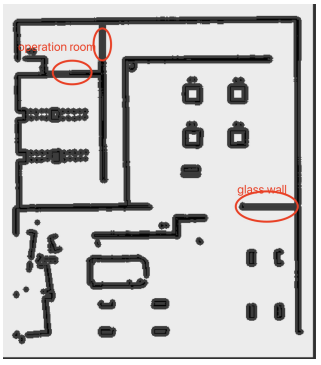

Figure.19 Map with CPL

2.3.3 Drop into costmap trap

Sometimes, when the car-like robot moves or turns too quickly, it will drop into the costmap. In the gazebo world, it does not impact with the obstacle, but it does not plan new path or move continuably.

Solution: For this issue, our solution is to make the inflation_radius of the global costmap larger than the one of the local costmap(Figure.20). With this solution,the global path will keep more away from the real obstacle than the local path. It will help the robot detach from the trap.

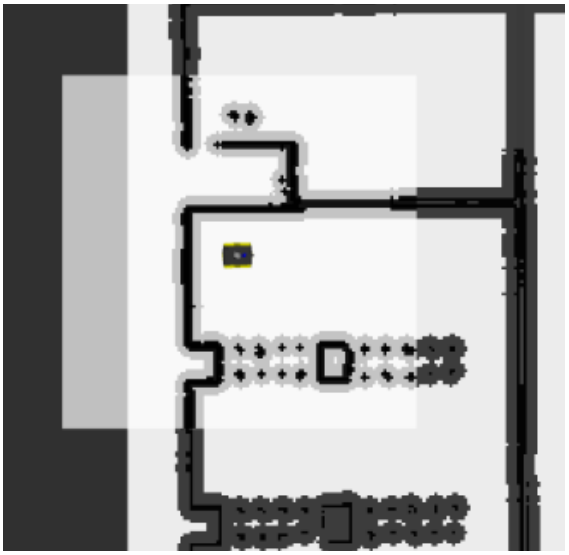

Figure.20 Different inflation_radius of two costmap

Potential future work

1. For the obstacles which can not be detected by the laser, like glass wall, we can add the ultrasonic sensors.In this way, the robot scans the map so that the parts that are not scanned by LiDAR or are not scanned well are presented more clearly on the resulting map.

2. Use hybrid A* algorithm: It is more efficient than the A* algorithm in that it takes up less memory and uses more heuristic information to construct the heuristic function in this aspect of heuristic function selection, thus allowing for a faster search for the target point.

3. Cartographer 3D: For 3D obstacles such as stairs that have space for the car-like robot to pass through, we can use 3D point clouds data to map, like cartographer 3D, ALoam and so on.