# Final Project Report

ME5413: Autonomous Mobile Robotics

**GROUP 14**

**Zhao Yimin A0285282X**

**Huang Zhenbiao A0285316B**

**Huang Yanqiao A0285209Y**

**Mo Yifei A0285328W**

**Chen Ruiyang A0285289J**

**Luo Yuanchun A0285275U**


April 26, 2024

Link to our Project: https://github.com/huangyqs123/ME5413_Final_Project

# 1 Task1: Mapping

In this task, we are required to try out different slam algorithms to map the environment and evaluate their performance. We have tried the default Gmapping, Cartographer 2D, Cartographer 3D, and fast-lio, and got the best result when using fast-lio for mapping.

## 1.1 Gmapping

In the starter code, the default SLAM algorithm provided is Gmapping. Through experimentation, we discovered that the performance of Gmapping is acceptable in environments with rich features and low complexity, such as the initial room. However, upon entering the corridor area, the robot struggles to determine its position accurately. This difference leads to map overlapping issues, and the Lidar output fails to align with the constructed map, as illustrated in fig 9. Since Gmapping does not provide many parameters to adjust, we decided to attempt other SLAM algorithms.

## 1.2 Cartographer 2D

For the .lua configuration, we create `5413_Task1.lua`, which is modified from the revo_lds.lua included in the configuration files. Firstly, we have modified the following contents:

```
tracking_frame = "base_link",
published_frame = "odom",
provide_odom_frame = false,
use_odometry = True,
use_nav_sat = true,
```

Then, we remap all the topics provided in cartographer_2d.launch to let cartographer receive the correct topic. However, we encountered some problems when using the default parameters to do the slam progress, the result is as figure 1.

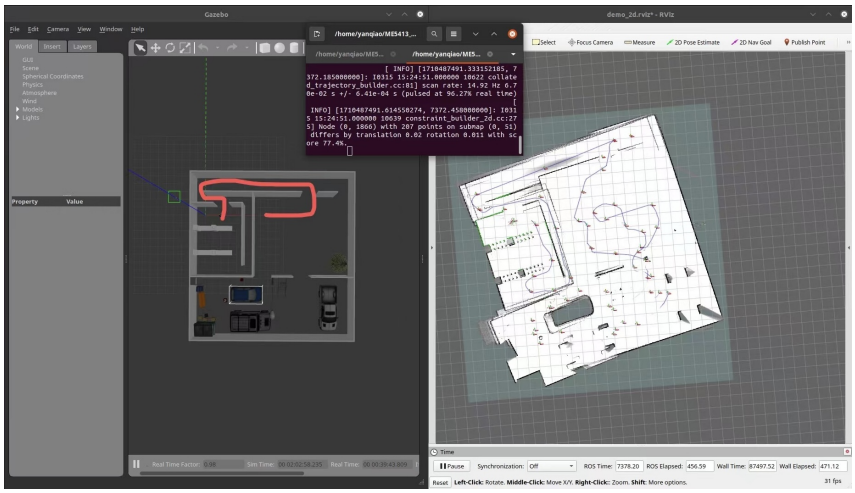
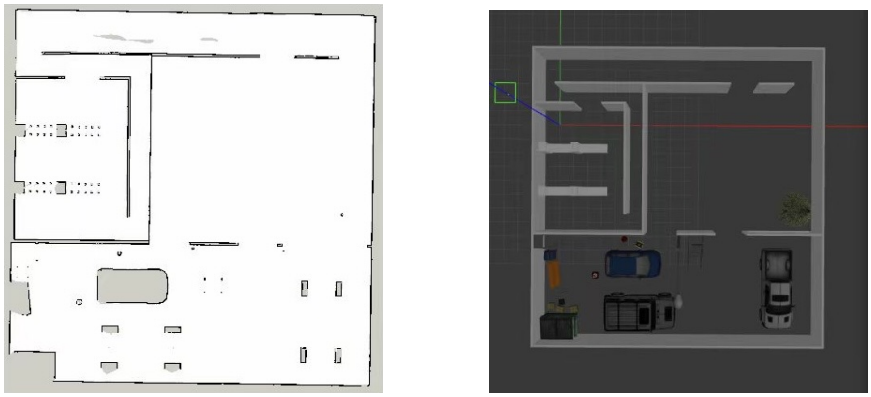


Figure 1: Default Parameters Mapping

The main problem is when the robot travels from the Assembly lines area to the random boxes area, it will pass through a very long corridor, and when the robot encounters the same wall shared by the assembly lines area and random boxed area, it cannot perform loop closure correctly, then the whole map will begin to get messy. To solve this issue, we have made some extra modifications to the .lua file, as follows:

```
TRAJECTORY_BUILDER_2D.submaps.num_range_data = 170
TRAJECTORY_BUILDER_2D.min_range = 0.3
TRAJECTORY_BUILDER_2D.max_range = 55.
```

The purpose of this is to increase the maximum range of the LiDAR data and to allow the cartographer to receive more data before starting to create new sub-maps. We have also set `use_imu_data = False` as we discover that we will get a better result when not using the imu data. The resulting map and the original world in gazebo are as Figure 2.



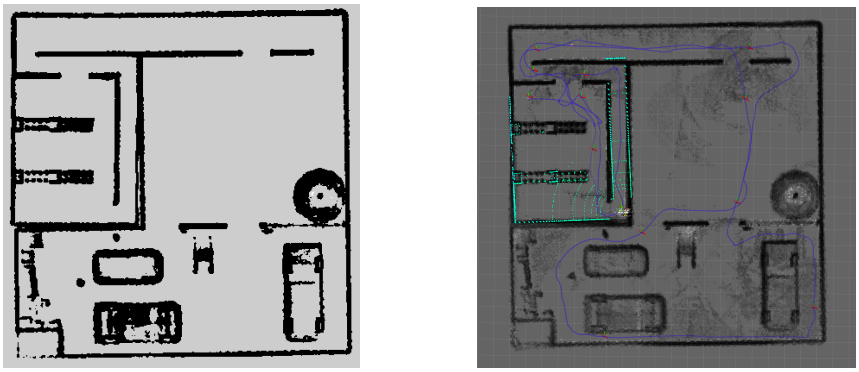
(a) Cartographer Result (b) Map Result

Figure 2: Compare between Cartographer Map and World Map

The evaluation result of cartographer 2d is as Figure 10 and Figure 11. Compared the cartographer 2d result with the original map, We can find a few problems with 2d slam. One is that the tree in the lower right corner behind the car and the hollow wall can't be detected, second is that the ladder located in the Parking Lot area can't be detected, and there are two cars that can only be detected by their wheels as well. This is because the 2d slam algorithm is limited by the height of the radar and cannot scan the complete environment. Therefore, we decided to use 3d slam algorithm instead.

## 1.3 Cartographer 3D

We also tested cartographer 3D. The main advantage compared with cartographer 2d is that 3D lidar can scan the complete environment, so we modified the .lua file from the official demo to have a try. As shown in figure 3, the map created by cartographer 3D included many items cannot be detected by 2d lidar. However, comparing with the result of cartographer 2d, the map we get from cartographer 3d contained too much noises, and the resolution is also not high enough, so we decided to try other 3D slam algorithms for better performance.



(a) Cartographer 3D Result (b) Cartographer 3D in rviz

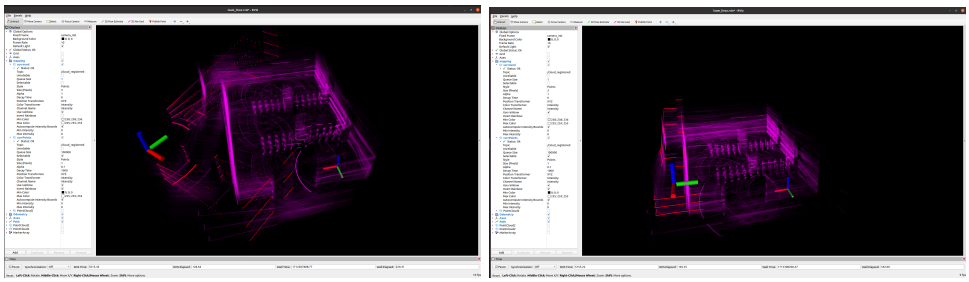
Figure 3: Compare between Cartographer 3D result Map and in rviz

## 1.4 Fast-lio

To enable Fast-lio to work properly in the given environment, we need to modify the names of topics. The configuration file of Fast lio is called `velodyne.yaml` and the following two lines are revised to match the topic name of the point cloud data and imu:

```
lid_topic:  "/mid/points"
imu_topic:  "/imu/data"
```

Following initial attempts using Fast-lio's default parameters, it was observed that the alignment between point clouds and ground truth data occasionally failed. After checking parameter guidance and explanations, it was determined that the issue was from the 3D LiDAR itself: its provided frequency was insufficient for the algorithm to localize the unknown environment effectively. Consequently, the "scan/rate" is increased from 10 to 50. Figure 4. shows the effects of changing the scan rate.



(a) 10Hz result (b) 50Hz result

Figure 4: Comparison of changing the scan rate

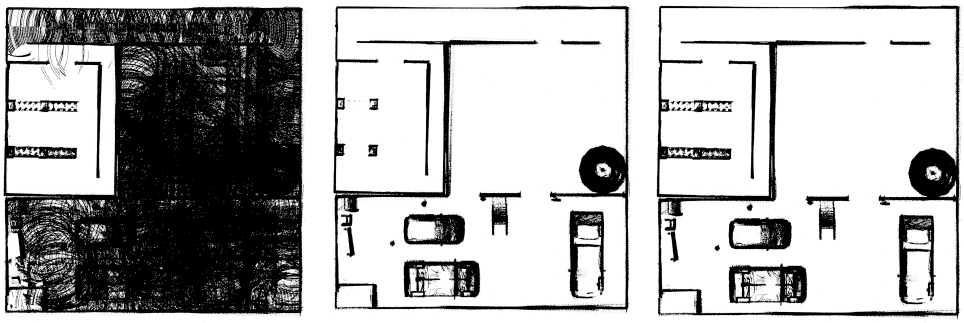
The overall performance of the SLAM result using Fast-lio could reproduce scenes from Gazebo as shown in figure 12 and figure 13. Also, we applied evo tool to evaluate the performance of Fast-lio as shown in 14 .The final pcd file is uploaded to Google Drive, and you can download the file through this link: https://drive.google.com/file/d/1h8SsmxdyiPjZ7QzeqL6QoLtx3f03PphK/view?usp=sharing.

### 1.5 Converting 3D map to 2D

After acquiring the 3D pcd format map generated by fast-lio, we need to convert it to 2D map as we are using 2d navigation method. To achieve this, we wrote an additional script `fast-lio_2d.py`. We first calculate the max and min `XYZ` coordinates of the point cloud. Then, we converted the 3d point cloud map to 2d, at a top-down view, using the following code and applied it to all the points in the point cloud:

```
grid_x = int(np.floor((x - min_x) / grid_size))
grid_y = int(np.floor((y - min_y) / grid_size))
```

We then apply morphological operations to remove small isolated points, using a 3x3 kernel and `cv2.morphologyEx` method. After performing all these operations, we identify a major issue. The resulting map has large amounts of ground noise points. It is easy to find out that those ground noise points are mostly near the ground plane, so we apply a pass-through filter on Z axis to filter out these ground noise points. From the previous steps we can get the maximum z is 5.46, while the minimum z is -0.32. So we set the Z axis filter limits to 0.45-infinite and 1-infinite, respectively, and got the result in figure 5.



(a) Filter limit 0.45-inf (b) Filter limit 1-inf (c) Combined map

Figure 5: Z axis filter result

As we can see from Figure 5(b), if we set the filter limit from 1-infinite, the resulting image is very clean without any ground noises. However, the bottom of the assembly line will be filtered too, which will cause potential problems when we perform navigation. As Figure 5(a), if we set the filter limit from 0.45-infinite, we can see the bottom of the assembly line clearly, but the other areas are full of noise. This could be because the Z position of the assembly line rooms is higher than other rooms. To get a complete and clean map, we use the mask function in Photoshop to combine those two maps we obtained above. The final resulting map is as Figure 5(c). And we will use this map in the following navigation module.

## 2 Task2: Navigation

In this task, we are required to navigate our robot with the help of the map we obtained in the previous section. The basic navigation task (navigate to a goal point) can be divided into 2 parts: `Localization` and `Planning`. The advanced navigation task (navigating to

a specific random box) also involves some vision-based technology to help us identify the position of the box before we can perform navigation.

## 2.1 Localization

### 2.1.1 Localization with AMCL

Adaptive Monte Carlo Localization method (AMCL) is a very suitable method to perform 2D localization. To make it work better in our task, we have modified the following parameters:

```
<param name="laser_max_beams" value="2000"/>
<param name="laser_min_range" value="-1"/>
<param name="laser_max_range" value="-1"/>
<param name="min_particles" value="2000"/>
<param name="max_particles" value="5000"/>
```

The main idea for this modification is to enlarge the maximum laser beams to increase the localization accuracy, set the maximum and minimum LiDAR distances to theoretical limit values (-1), and increase the maximum particles number to enlarge the particle sample.

It's also worth mentioning that the start position of the robot in the map depends on `my_map.yaml`, which is provided by some Slam methods like cartographer. However, as we convert our map from 3D to 2D manually, we need to create our own yaml file and calculate the correct origin point to let our robot start at the correct position.

### 2.1.2 Frame Drifting Problem

In actual operation, we found that if the robot remained stationary for an extended period, it would experience a frame drift issue, leading to subsequent improper localization, as shown in Figure 15(a). By subscribing to the /jackal_velocity_controller/odom topic, we noticed that the output pose value kept changing even when the robot was not moving, causing the robot to rotate on the spot. We suspect this was due to IMU noise or inherent issues with the simulated odometry itself.

To resolve this problem, we first tried modifying the update_min_a parameter in the amcl.launch file. This parameter represents the minimum angular offset required for the AMCL algorithm to update the localization. By decreasing this parameter value to accelerate the AMCL localization update frequency, which will make some improvements. However, it doesn't get to the root of the problem, and setting the parameter too small will increase computational complexity, and there is a probability that the robot will experience a sudden significant localization shift.

The other method we tried is modified the parameters in `robot_localization.yaml` file. We noticed that in `spawn_jackal.launch`, it includes `control.launch`, which will enable the EKF filter, subscribing the original /jackal_velocity_controller/odom topic, and publishing the /odometry/filtered topic. The EKF filter node uses configuration in `robot_localization.yaml`. We modify it as in Figure 6

```
odom0: /jackal_velocity_controller/odom
odom0_config: [false, false, false,
               true, true, true,
               true, true, true,
               false, false, false,
               false, false, false]
odom0_differential: true
```

(a) Odom config

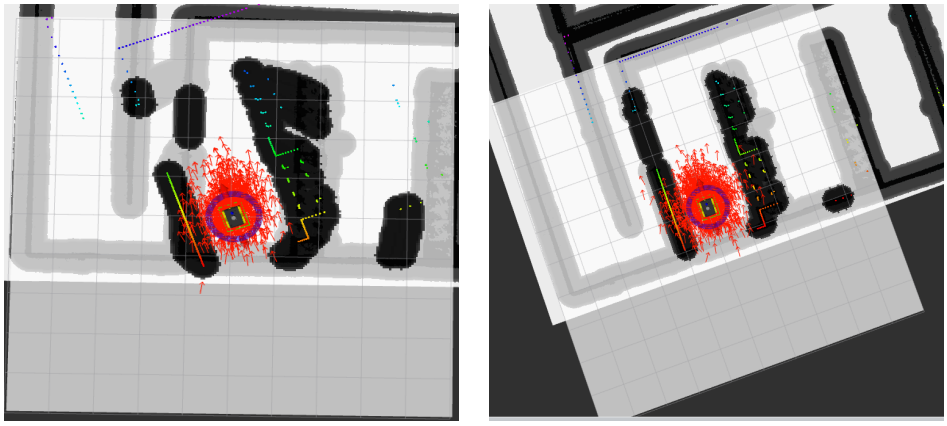
```
imu0: /imu/data
imu0_config: [false, false, false,
              false, false, true,
              false, false, false,
              false, false, true,
              true, true, false]
imu0_differential: true
```

(b) IMU config

Figure 6: Configuration for robot odometry EKF

Our main modification is set `odom0_differential` and `imu0_differential` to true to process the current odom and IMU data as a difference from the previous data. This helps the filter to better process the odom and IMU data, reducing the impact of cumulative errors. In addition, since we know that the angular velocity information from the odometer include large noises, we chose not to use the Z-axis angular velocity information provided by the odometer during the EKF process, but to use the Z-axis angular velocity information and rotation information provided by the imu instead.

After our modification, we have solved the frame drifting problem from the root. As we can see from Figure 15(b), using our modified EKF parameters, the robot won't have a frame drifting problem even if it stays still for 1 minute.



(a) Default EKF params (b) Our modified EKF params

Figure 7: Comparing different filter params after 1 minute in start pose

## 2.2 Planning

`move_base` is a ROS package which facilitates navigation by allowing a robot to move from its current location to a specified goal point. It required a global planner and local planner to work, and both planner required configurations on cost map. We choose `global_planner/GlobalPlanner` as the global planner and we tested different local planner, like `base_local_planner` and `teb_local_planner`. We chose `teb_local_planner` since it has better performance than other planners we tested.

### 2.2.1 Costmap Configuration

The costmap contains 3 main parts: static layer, inflation layer and obstacle layer.

The static layer came from the `\map` topic, it represents the map we get from slam.

The main purpose of the inflation layer is to add an 'inflation' region around obstacles in static layer. This will convert the map from work-space to configuration-space, and allow the robot to avoid obstacles as much as possible when planning its path, in order to prevent collisions. We set the inflation radius as 0.33, which means 330mm, since the size of jackal is 508 x 430 x 250 mm.

Obstacle layer represents the obstacles found by lidar but not contained in the basic map, like some dynamic items not shown in the slam step. The inflation radius for obstacle layer is the same with inflation layer, and the source topic is `/front/scan`

The costmap for local planner and global planner both contains these layers, as shown in Figure 8.

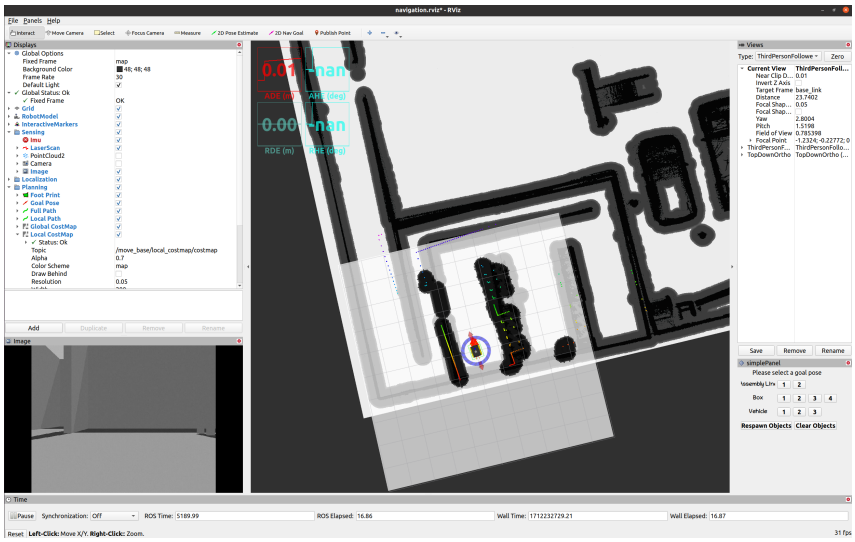


Figure 8: Costmap Layers

### 2.2.2 Virtual walls

Due to the task requirements, there would be restricted areas on the map. To prevent the robot from entering these areas, we introduced the virtual wall parameters in the costmap. By downloading the relevant package, we added the prohibition_layer parameter to the plugins section of the costmap-related yaml files. We then created a prohibition_areas.yaml file containing the ranges of prohibited areas and inserted it into the move_base launch file for the costmap to use. Based on the task, we set up a long strip-shaped prohibited area at the Assembly Lines exit to force the robot to take the outer corridor. In the Random Blockade area, one of the two entrances would randomly generate an obstacle barrel, representing that entrance being blocked. Therefore, we set prohibited points in the remaining space of the two entrances, which, through experimentation, would block that entrance together with the generated obstacle barrel, forcing the robot to enter the box area through the other

entrance. As shown in Figure **??**, the black areas without physical boundaries represent the constructed virtual walls, and the planned green path meets the requirement of avoiding the restricted areas.

# 3 Task3: Perception

The perception part can be divided into three parts. Firstly, move to the area with boxes and patrol randomly. Secondly, try to detect the target box. After finding the target, estimate its position and navigate there.

## 3.1 Random Patrols

Set the randomly generated goal inside the box area. After reaching the goal, generate another random one to make sure the vehicle patrols in this area completely and continuously. However, the goal would be set on the generated objects cause they were not included in the map at the SLAM stage. To be more specific, the system updates the cost map when the vehicle searches object in the box area, and the move base will find the goals on objects unfeasible. To handle this situation, the move base status is listened, and a new random goal is set if the ABORT status is received. Moreover, the same action is executed if the SUCCEEDED status is received. This is for achieving continuous random patrol by setting a new random goal when the former one has been accomplished. In addition to these, a goal is checked to see if it is on or nearby the cost map before it is sent. The cost map topic is listened and checked in the range of the goal point and a circle with a radius of 0.3 metres around it. Same to the previous situation, a new random point is generated if any value in cost map checking area is higher than the threshold of 50.

## 3.2 Object Detection

By applying template match on the raw image revived from the corresponding topic, object detection is easily accomplished. The template is set to none when button that is not relevant to "box" is clicked. It helps avoid detection all the time and contributes to saving resources. There are some improvements to the default template match. The most important one is setting a threshold to match the value. In this project, only the result with more than 0.75 similarity score can be accepted. Another improvement is multiple scale template match, which means the template is resized with different scales and then used to match. The result with the highest score is adopted. This method ensures that targets at different distances can be detected.

## 3.3 Target Position Estimation

### 3.3.1 Calculation Theory

After finding the target, the position of it to map needs to be calculated and published. This position matrix ${}^{target}T_{map}$ is presented as:

$$^{target}T_{map} = {}^{target}T_{optical} \cdot {}^{optical}T_{map} \tag{1}$$

where $^{optical}T_{map}$ is the position from camera optical to original point. Use tf listener to acquire this matrix easily. And $^{target}T_{optical}$ is the position from target to camera optical. To get this matrix, depth information is required to involve calculation of:

$$X = \frac{(u - c_x) \cdot Z}{f_x} \tag{2}$$

$$Y = \frac{(v - c_y) \cdot Z}{f_y} \tag{3}$$

$$Z = Z \tag{4}$$

where $X$ and $Y$ are the horizontal and vertical coordinates in the camera frame, and $Z$ is the depth from the camera to the target. Moreover, $(u, v)$ is the coordinate of central pixel in region of interest (ROI), $u$ is the horizontal coordinate and $v$ is the vertical coordinate. $(c_x, c_y)$ is the optical center of the camera, and $f_x, f_y$ are the focal lengths of the camera, expressed in terms of pixels. $f_x$ is the focal length in the horizontal direction, and $f_y$ is the focal length in the vertical direction.

### 3.3.2 Add Kinect Camera for Depth Information

As we concluded from the former part, depth image is required. However, the default camera is only a common mono camera which can not be used to gain depth image. We checked the urdf files in `jackal_description` package first. In the default accessories.urdf.xacro file, we found some options for camera, like `bumblebee2` or a pair of `flea3` cameras. However, we think it would be easier to implement a perception algorithm on depth camera than stereo cameras, so we decided to add an Kinect depth camera in this urdf.

By referring to the official tutorial provided by gazebo and the urdf of turtlebot3 waffle, we successfully add Kinect on the jackal robot as shown in Figure 16.

## 3.4 Communication Problems

There is one difficulty caused by topic communication between move base tool and random goal tool. Move base always sends the status topic many times, which also leads to generating random goal many times. In that case, the current received move base status id is compared with the last one to ensure a random goal is generated when a new status comes in, and the time interval between two consecutively generated goals was set to a minimum of 0.3 seconds. These measures ensure random goal is only generated once when a new status comes in.

Furthermore, the inputs of the rgb image, depth image, and ego pose are not synchronized, and the detection process takes a lot of time. In that case, ApproximateTimeSynchronizer from message filter is used to acquire rgb and depth images with time intervals

up to 0.01 second. Then, use the time stamp of the depth image to get the corresponding transformation matrix through tf listener, so that sync these three data.

### 3.5 Detection result

The detection result is visualized by the Table 1. The Euclidean distance is set as an evaluation criterion. We tried 3 experiments, each time restarting the entire experimental environment and conducting 5 times of detection tasks in it. The following table indicates the displacement result. The average error for each experiment is under or equal to 1.3 meter which is very accurate.

| Experiment | Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4 | Iteration 5 | Average |
|---|---|---|---|---|---|---|
| First | 1.13 | 1.43 | 1.47 | 1.27 | 1.21 | 1.30 |
| Second | 1.12 | 1.10 | 1.60 | 1.41 | 0.94 | 1.23 |
| Third | 0.85 | 1.32 | 1.00 | 1.24 | 1.40 | 1.16 |

Table 1: Error table for perception task experiments. All data in meters.

### 3.6 Evaluation

To evaluate the performance of the navigation task, we recorded screenshots when we completed our goals. According to the task description, we needed to move the robot to three goal poses which are Assembly Line 1, Random Box 3 and Delivery Vehicle 3. The results are shown in Figure 17, Figure 18 and Figure 19.

Based on the provided results and the four metrics of ADE, RDE, AHE, and RHE, we assess the error between the robot's final pose and the goal pose in terms of position error and heading error. We can see that in Assignment 1, the robot successfully reached the goal of Assembly Line 1 while maintaining a low position error. The heading error spiked due to inaccurate predictions during initialization and sudden turns. In Assignment 2, the robot searched within the designated area and eventually found and docked near the goal of Box 3, exhibiting low values for ADE, AHE, and RDE, while RHE spiked due to the final momentary turn near the box. In Assignment 3, the robot successfully reached the goal of Vehicle 3, with all four metrics maintaining low error values.

From the results and evaluation parameters, we can conclude that our robot can effectively complete the tasks as required and reach the specified goal positions.

# 4 Appendix

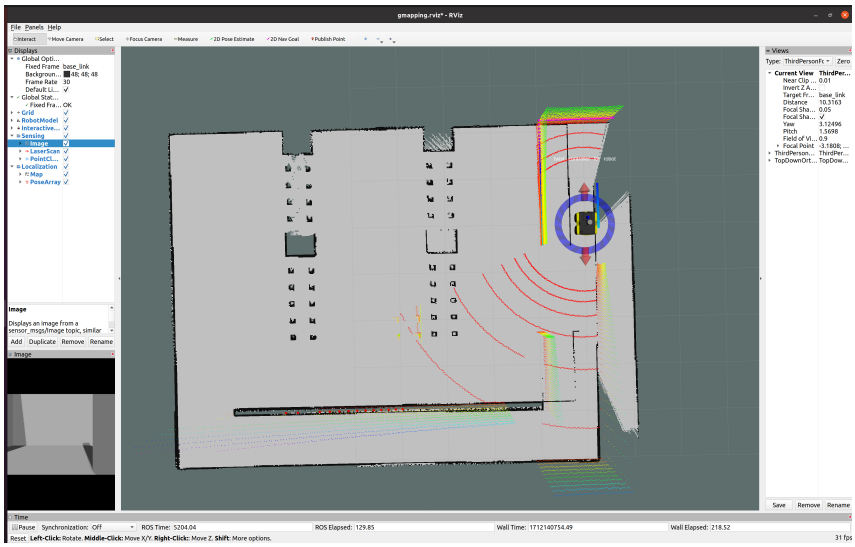


Figure 9: Default Parameters Gmapping Result

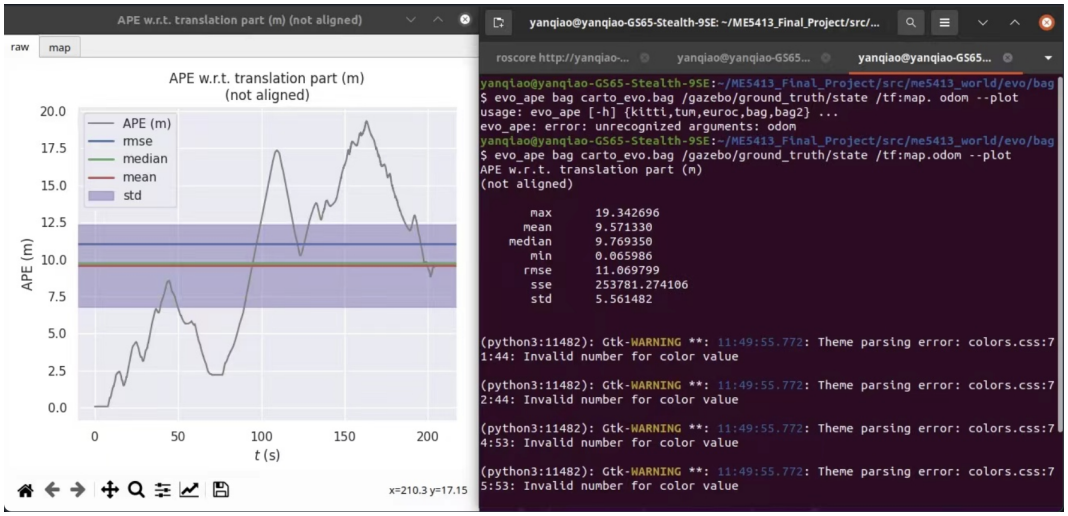



Figure 10: Cartographer 2d evaluation

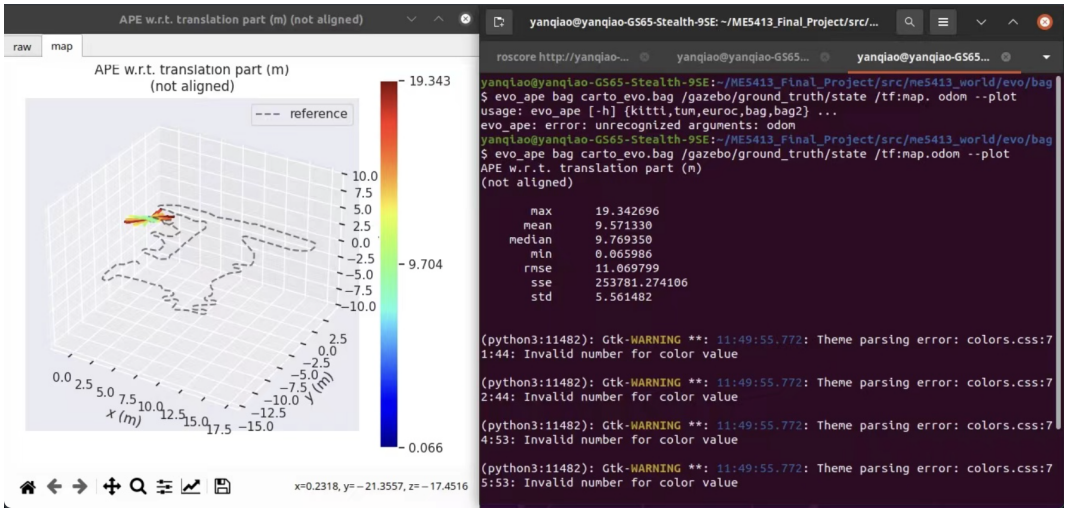



Figure 11: Cartographer 2d evaluation

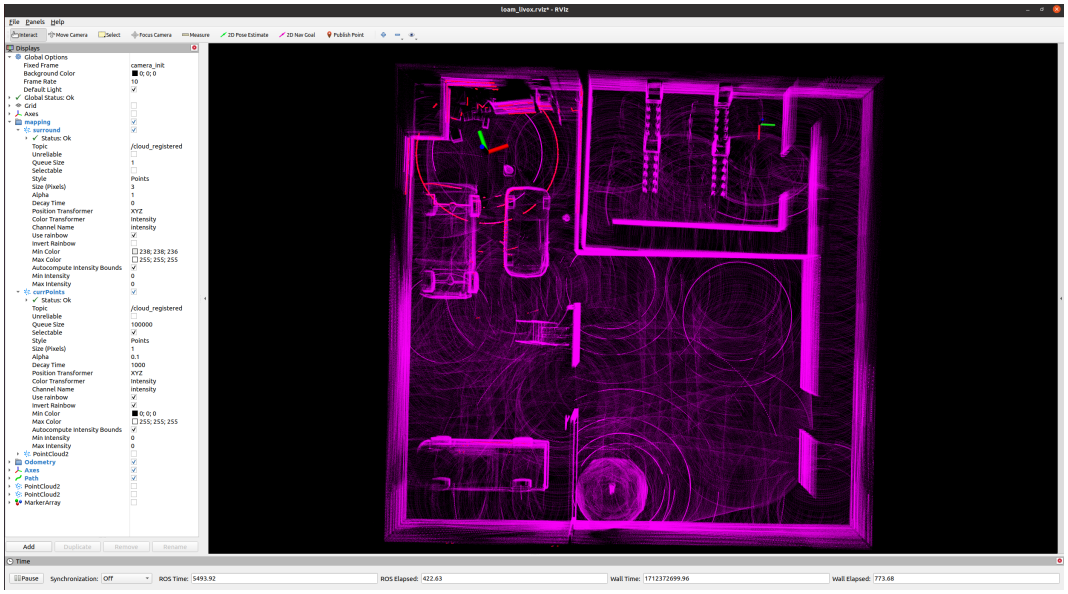


Figure 12: Fast-lio Rviz result

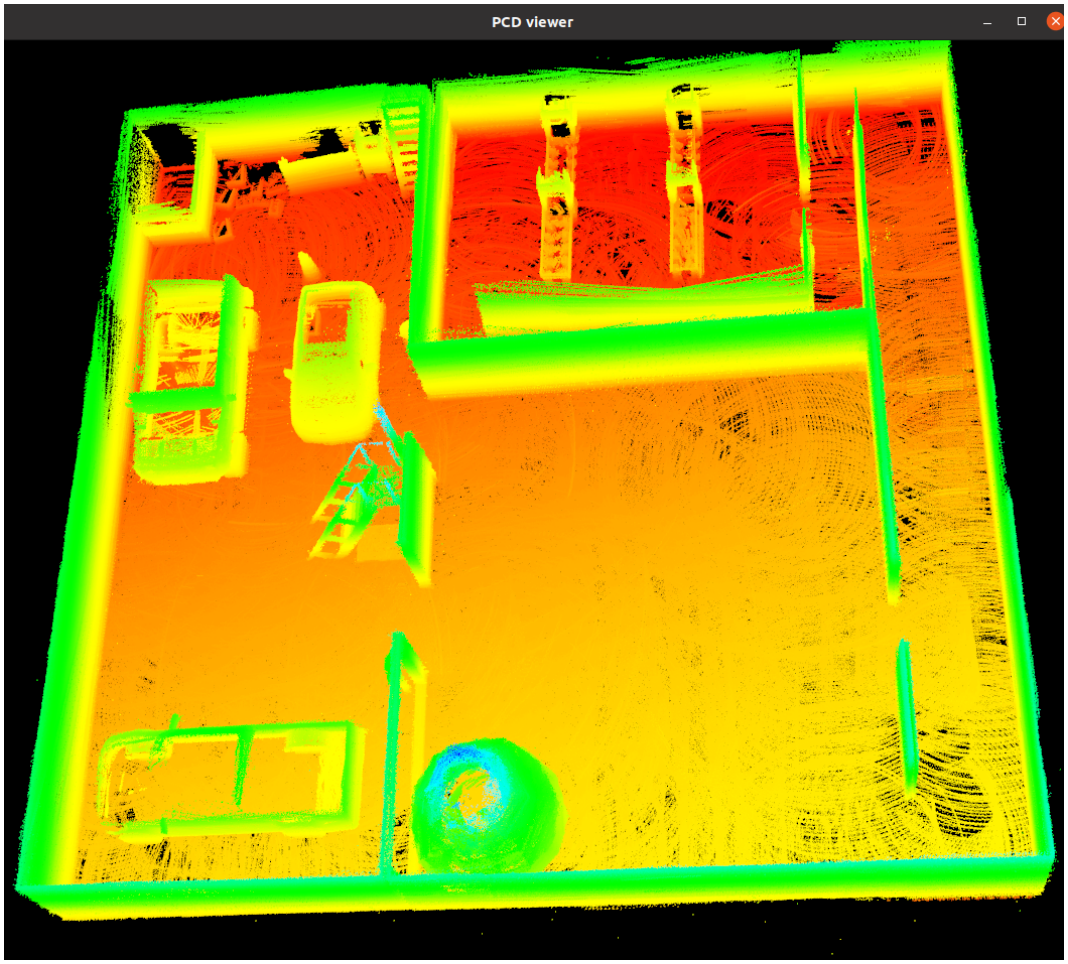



Figure 13: Fast-lio PCD viewer result

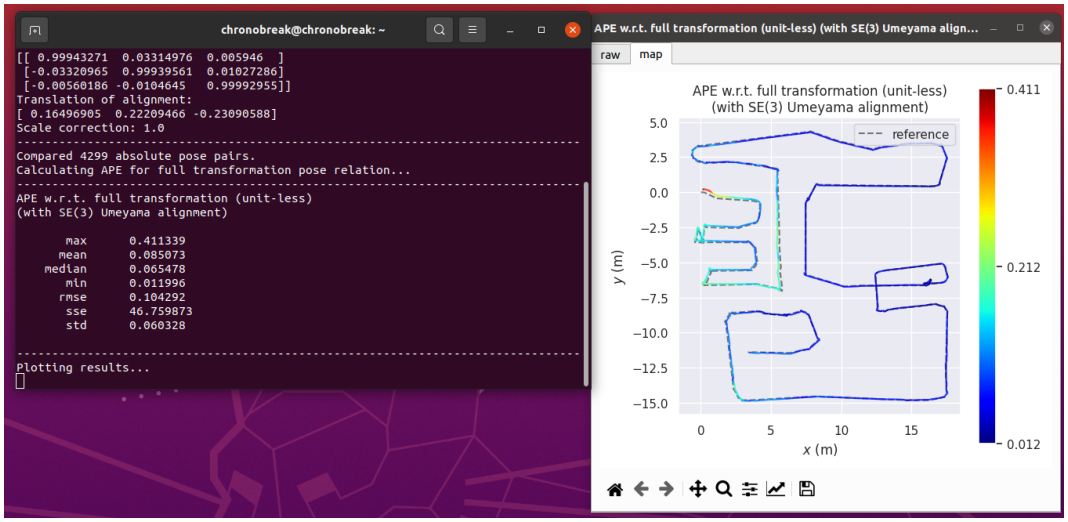
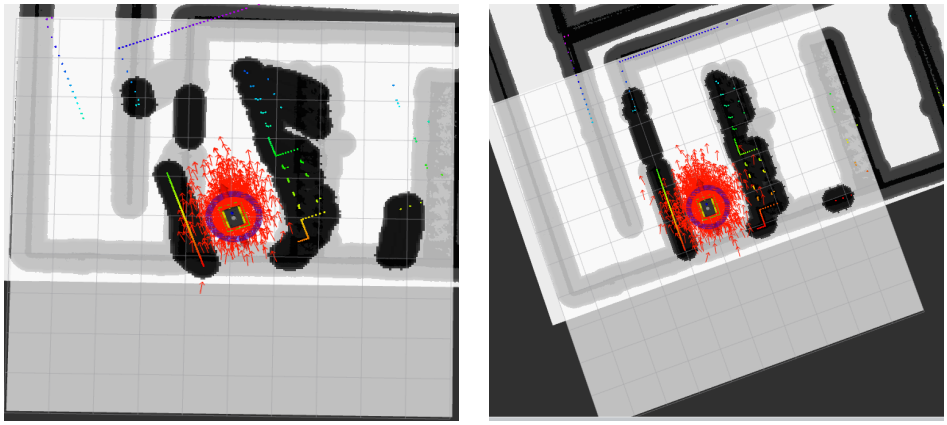
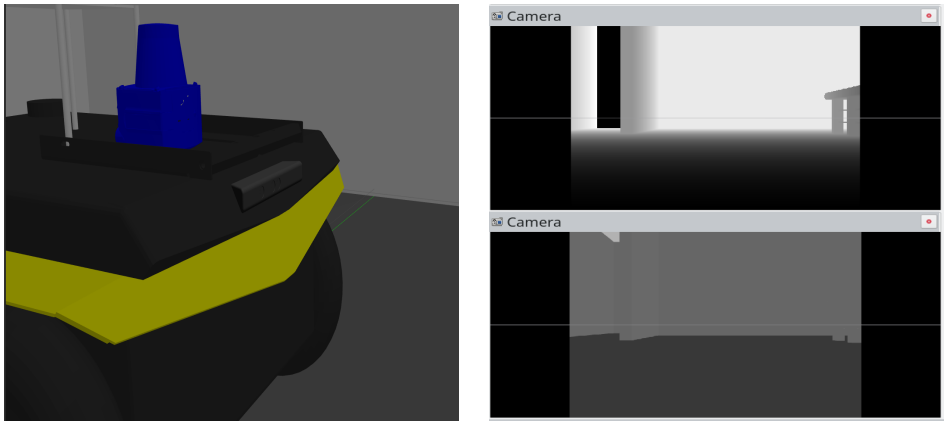



Figure 14: Fast-lio evaluation



(a) Default EKF params (b) Our modified EKF params

Figure 15: Comparing different filter params after 1 minute in start pose




(a) Kinect position (b) Kinect visions

Figure 16: Kinect depth camera on robot

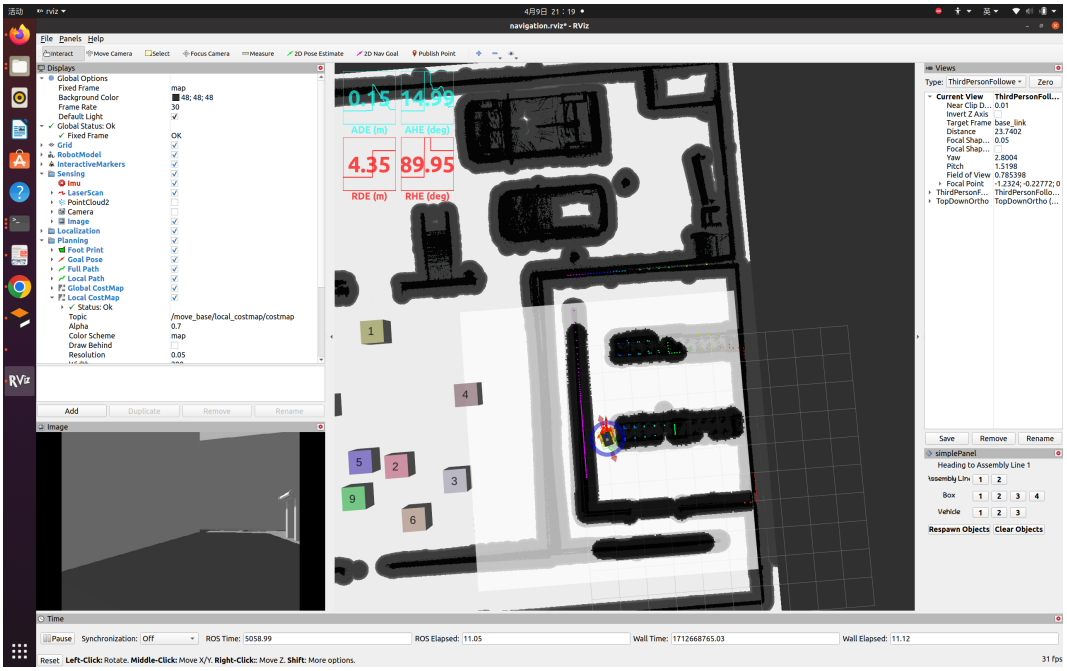



Figure 17: Assignement 1 screenshot of result

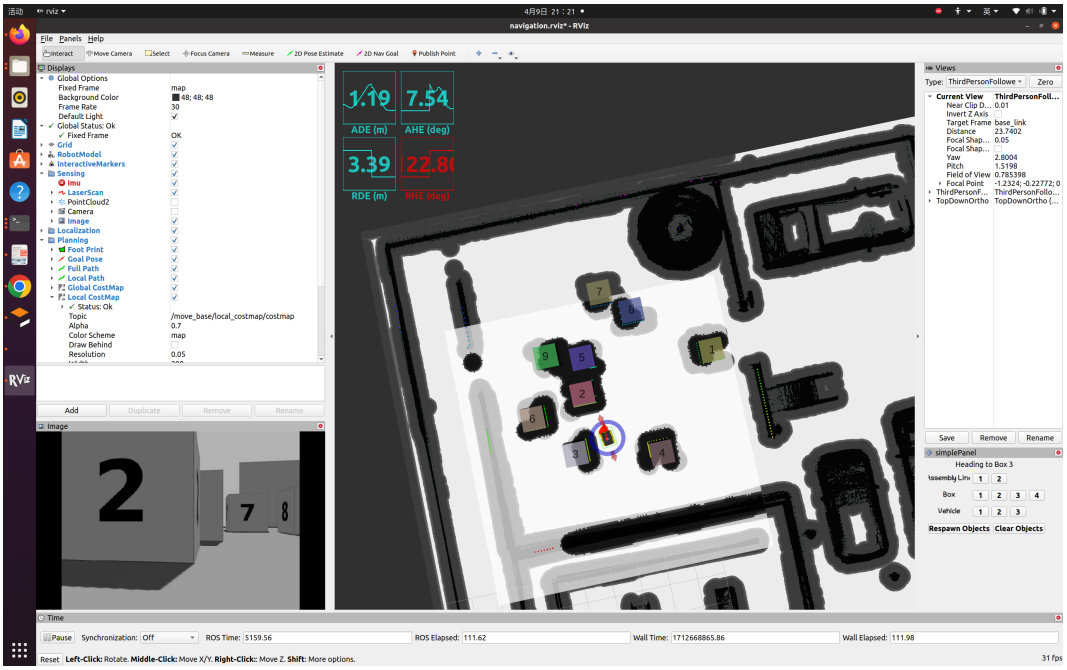



Figure 18: Assignement 2 screenshot of result

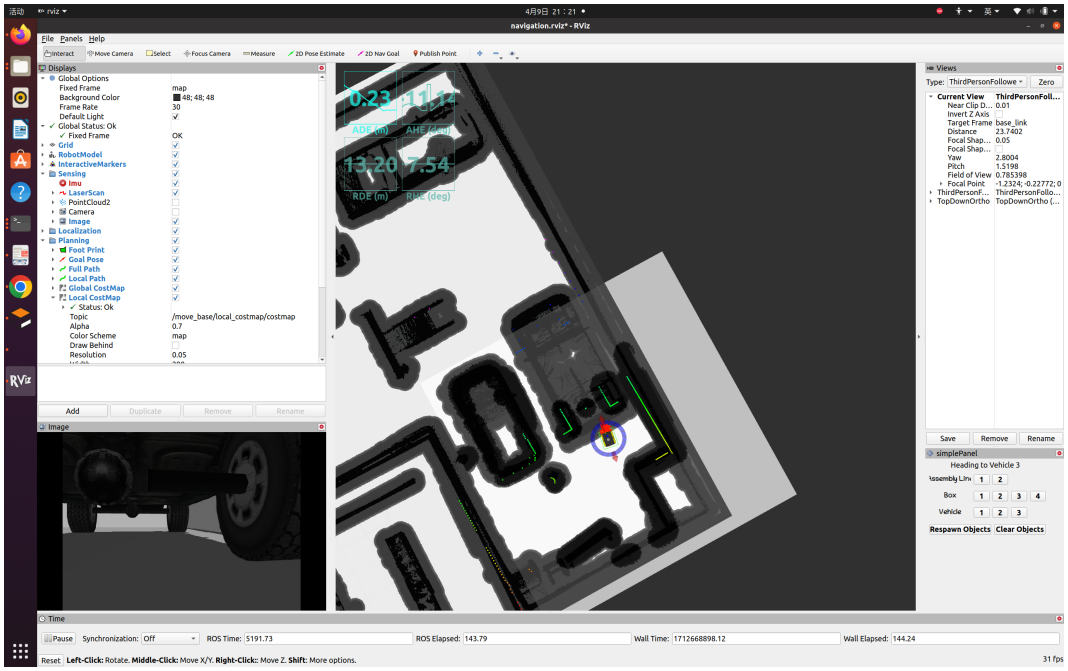



Figure 19: Assignement 3 screenshot of result