Final Project Report

Assigned by ME5413

Submitted by **Group 4**

Fan Xiuqi A0285060J

Fang Chongyu A0285314E

Wang Sijie A0284859E

Chen Ziao A0285086U

Zhou Tingguang A0285076W

Lyu Zhengwen A0285247W

Supervisor： Prof. Marcelo H ANG Jr

Session 2023/2024

09 Apr 2024

Github:
[LandoFan/ME5413_Final_Project: NUS ME5413 Autonomous Mobile Robotics Final Project (github.com)]

1. Project Description

The field of mobile robotics is rapidly evolving, particularly in its ability to perceive the surrounding environment and make decisions. Modern mobile robots integrate advanced sensors, including but not limited to LiDAR, mono and stereo cameras, as well as data processing algorithms and machine learning techniques. These enhancements enable them to operate independently and perform complex tasks across diverse environments. Applications range from industrial automation to the service sector and extend to high-risk exploration and rescue operations.

In this project, a mini-factory environment is presented, comprising three target areas and one restricted area, as depicted in *Figure 1*. The objective of the project is to employ a self-designed robotic navigation software stack, enabling the "Jackal" robot to sequentially move from the starting point to the specified poses within each aera: 1. ‘Assembly Line 1’, 2. ‘Packaging Area 1’, and 3. ‘Delivery Vehicle 2’.

2. Mapping

In robot navigation, map construction is a fundamental element involving the creation of a digital representation of the environment, which robots can utilize for more effective navigation and task execution. In this project, map construction will enable the "Jackal" robot to comprehend and interact with its operational environment, including identifying paths to the assembly line, packaging area, and delivery vehicles. Accurate maps allow the robot to navigate to designated areas without human intervention and enable more efficient route planning, reducing travel time and enhancing productivity. This project will employ a comparative approach using three mapping algorithms: Gmapping, A-Loam, and FAST-LIO, to implement the mapping task.

2.1 Gmapping

The default mapping algorithm in the original package, gmapping, estimates the robot's pose and maps the surroundings using a particle filter.

Gmapping is a 2D SLAM algorithm, which mainly focuses on the small scenes. In a big scene, it will need more calculations and memory.Based on the default algorithm, the odometry topic will rotation to a certain degree with a relatively big RMSE, showing in *Figure 2*. This rotation will influence path planning in the future.

As a result, the 2D SLAM method is incapable to map the features in this scene. Because 3D visual SLAM algorithms will collect too much information and result in high processing costs, the FAST-LIO and A-Loam algorithm are used to map the surroundings.

2.2 A-Loam

A-Loam (Adaptive Lidar Odometry and Mapping) is an enhanced version of Loam (Lidar Odometry and Mapping), which is a system for real-time three-dimensional mapping and localization. A-Loam introduces several improvements that enhance the performance and stability of Loam. These enhancements include adaptive feature extraction and matching strategies, which help maintain accuracy and efficiency in localization and map construction across varied environments. Additionally, it may incorporate algorithmic optimizations for complex settings, such as dynamically adjusting parameters to adapt to environmental changes or increasing robustness against environmental disturbances. These features make A-Loam particularly effective in dynamically challenging and varied real-world applications.

Operate process:

1. After building the 'ME5413_Final_Project' and A-Loam, the A-Loam is moved into 'ME5413_Final_Project/src' folder as a package in the repo whose name is 'A-LOAM'. Then build the Octomap as an auxiliary means and move it into 'ME5413_Final_Project/src' in same way. Compile the entire project package simultaneously using the 'catkin_make' command.
2. Write the 'aloam.launch' file and take care to rewrite the value to 32 and change the value in 'accessories.urdf.xacro', which is in the jackal-description file, set it to HDL32E.Means A-Loam will use HDL-32E for higher accuracy scanning.
3. Run two different roslaunch commands from two terminals: 'roslaunch me5413_Final_Project me5413_project.world' and 'roslaunch A-LOAM octomap.Launch', which is used to simulate the startup world and the auxiliary tool Octomap. After generating the map, select 'Octomap' in the options to observe the auxiliary 2D map.
4. Run a roslaunch command from the terminal under the 'A-LOAM/launch' folder: 'roslaunch aloam.launch', which is used to start the A-Loam location & Mapping task. Respectively using keyboard to control the forward, backward, and direction of the trolley, and sample the point cloud of the entire map.
5. The generated .pcd file and .bag file are located in the home directory folder.

2.3 Fast Lio

Fast-Lio, a fast and robust Lidar-Inertial Odometry (LIO) package proposed by the MARS Laboratory at the University of Hong Kong, is designed to achieve high-precision, low-latency real-time 3D positioning and mapping.

By leveraging precise sensor fusion techniques, the Fast-Lio algorithm attains exceptional accuracy in both positioning and map construction. Furthermore, it integrates various strategies to handle dynamic environments and sensor noise, emphasizing on reducing computational delays for real-time navigation. The algorithm's adaptability in supporting various Lidar and IMU devices significantly boosts its compatibility and versatility. Consequently, Fast-Lio emerges as a comprehensive solution for numerous applications that demand precise localization, offering broad utility across a diverse array of fields. The error of this algorithm evaluated by EVO is shown in *Figure 6* and *7*.

2.4 Point Cloud to 2D-Map

The transition from point cloud to 2d-map, the format of .pgm, highly relies on the PCL (Point Cloud Library). The main idea is to use the PassThrough Filter and Radius Outlier Removal to filter unnecessary point clouds and save the remaining part, regarded as the 2D map after transition. The 'SetMapTopicMsg' function converts the processed point cloud into a grid map format. This involves calculating the minimum and maximum X, Y coordinates of all points in the cloud, and then determining the size and origin of the grid map based on these coordinates and the given map resolution. Each point is converted to an occupied or free state on the grid map based on its position on the map.

2.4.1 The PassThrough Filter

The principle of the pass-through filter depends on the values of axis z. After setting the threshold of z, whether the inner points or the outer points should be saved based on personal choice. The threshold can be set in the launch file.

2.4.2 The Radius Outlier Removal

The core concept is that use the local neighborhood information of each point in the point cloud to decide whether the point should be retained. For every point in the point cloud, the radius filter considers a spherical region centered at that point with a specified radius. Within this spherical region, it calculates the number of neighbor points around the center point.

If this number is less than a certain threshold, the center point is considered an outlier and is removed; otherwise, it is retained.

2.5 Results and Discussion

In our algorithm, the point cloud outputs of the Aloam and the Fast-lio are treated by this method other than octomap, by different thresholds.
The threshold of both pass-through filter and radius filter depends on the certain situation, which cannot always achieve good performance. In A-Loam point cloud map, showing in *Figure 3*, some of the noises are in the same plane as the wall or cars, so it is hard to separated them, showing in *Figure 4*. The 2D map generated by A-Loam is also not as good enough as we expect, showing in *Figure 5*. In Fast-Lio point cloud

map as *Figure 8*, the noise can be separated clearly by selecting the outer thresholds, showing in *Figure 9*.
Thus, the map used in navigation part comes from the Fast Lio outputs.

3. Navigation

Navigation is a complex and dynamic process that involves more than merely following a predetermined path, it also includes adjustments based on new information and obstacles that arise. Robotic navigation can be distinctly divided into two main components: localization and path planning. These processes are interdependent and collectively provide decision support for effective movement of robots in both known and unknown environments.

3.1 Localization—AMCL

3.1.1 Algorithm principle and parameter configuration

AMCL is a sophisticated robot localization algorithm based on particle filtering, primarily used to help robots determine their positions within a known map, particularly in environments where the robot's location is uncertain. Its core advantage lies in dynamically adjusting the number of particles based on motion data received from the odometry. Each particle undergoes state prediction using a motion model that updates its position and orientation, thereby simulating the actual movement of the robot. Concurrently, the algorithm performs resampling based on the importance weights of the particles, where particles with low weights are eliminated, and those with high weights are replicated. This process ensures that particles which more closely match the sensor data occupy a larger proportion of the particle set. Through this method, the most probable location of the robot is estimated.

3.1.2 Frame drifting

Frame drifting involves a gradual loss of accurate tracking of a robot's precise location during its navigation process. Initially, there may be no apparent drift, but during operation at Assembly Line 1, significant frame drifting can occur which is shown in *Figure 10*. This phenomenon may be attributed to two main factors:

- **Accumulative Error (Primary):** Each movement measurement can introduce minor errors, which accumulate over time, ultimately leading to significant positional deviations. This type of error is cumulative and can significantly affect the overall accuracy of the robot's localization over extended periods or distances.

- **Particle Filter Issues:** Degeneracy: When most of the particle weights concentrate on incorrect estimations, it results in a reduction of effective particles. This degeneracy can skew the particle distribution and degrade the localization performance. Resampling Impoverishment: Improper handling during the resampling process can lead to a loss of particle diversity, preventing the particle set from effectively covering all possible states in the state space. This impoverishment can further exacerbate localization errors and reduce the robustness of the particle filter.

During the test process, it is found that by tuning the update frequency and publish frequency of global costmap can alleviate this phenomenon.

3.2 Path Planning

3.2.1 Costmap

In robot navigation, the static map obtained by SLAM cannot be utilized directly. It is unable to react to changes in obstacles and is also incapable of preventing collisions. To fulfill navigation requirements, it is essential to supplement the static map with additional information such as the obstacle data obtained from time to time, the inflation area to create a navigable map, which is a costmap. The robot uses the costmap to determine which areas are safe and easy to traverse and which areas are obstacles or present some level of difficulty. In addition, by representing obstacles and other elements in the environment as higher-cost cells in the grid, the robot can plan paths that avoid collisions.

Each cell in the costmap can have one of 255 different cost values, the underlying structure that it uses is capable of representing only three. Specifically, each cell in this structure can be either free, occupied, or unknown. Each status has a special cost value assigned to it upon projection into the costmap.

In ROS navigation, there are 2 costmaps: local cost map and global costmap. The former is used for global path planning, while the latter is used for local path planning. Both costmaps are composed of multiple layers. Generally, those layers can be:

- Static Map Layer: static map constructed by SLAM.
- Obstacle Map Layer: obstacle map detected by sensors, for example lidar.
- Inflation Layer: outward expansion area from each other layers, collision prevention.
- Other Layers: customized shaped costmap

In this project, a prohibition layer was defined to forbid jackal robot from passing through the prohibition area. An opensource package for customizing prohibition area in costmap was used to achieve the function. Finally, local costmap was composed of obstacle map layer, inflation layer and a prohibition layer while global costmap was composed of obstacle map layer, inflation layer and a prohibition layer and static layer.

During the tuning process, we found that the parameters mainly affecting the navigation performance are the update and publish frequency of the costmaps and the radius of the inflation layer. The update frequency of the costmap directly affects the accumulation error of the odometry. If the frequency were too high the error accumulation would be so fast that the localization becomes terrible. To enable the local planner to find a path, the inflation radius of obstacles could not be too big, otherwise the local planner would think it is too narrow to pass through and don't find a optimal path, which is shown in *Figure 11* Detour. This problem sometimes leaded to failing to park beside the target when the target box is close to neighbor boxes.

3.2.2 Local and Global planner

- DWA (Local Planner)
  At the beginning, DWA planner was used as the local planner to dynamically generate local path along the global path to avoid obstacles. However, various parameter configurations have been attempted without achieving stable performance with the DWA planner. During navigation, the DWA planner continuously generates new local paths, resulting in the Jackal robot moving slowly. Moreover, it exhibits poor performance when navigating corners, despite setting a large inflation layer for obstacles, the Jackal robot remains unable to pass through.
- TEB (Local Planner)
  The core concept of the TEB (Timed Elastic Band) local planner is based on the theory of elastic bands, which involves creating a flexible, band-like area around the planned path. This band can expand, contract, and bend as needed to dynamically adjust the path in response to changes in environmental conditions. The navigation strategy of TEB allows the robot to adjust its orientation during movement to face the target position. This strategy reduces the need for additional rotational adjustments at the target point, thereby enhancing the overall efficiency of navigation. Worth mentioned, the footprint is required to set same as the robot as "Polygonal" [[-0.31, -0.255], [-0.31, 0.255], [0.31, 0.255], [0.31, -0.255]] to achieve a better identified shape in turning and obstacle avoidance.
- Global Planner (Global Planner)
  The core characteristics of the Global Planner algorithm primarily revolve around graph-based search, utilizing a standard A* graph search algorithm (as used in HW3) to plan routes. Additionally, it employs a costmap, as previously described. The Global Planner provides macroscopic navigational capabilities within modern mobile robotic systems.

The Global Planner initially generates an optimal path from the current position to the target location based on the global map. This path takes into consideration the avoidance of large obstacles and satisfies other long-distance navigation requirements. The TEB Local Planner receives this path from the Global Planner and begins navigation along it. During this process, TEB dynamically adjusts and optimizes the path to address dynamic obstacles and unforeseen events encountered. This includes bending, extending, or contracting the path to maintain a safe distance from obstacles. As the robot progresses, the TEB continuously evaluates and adjusts the robot's speed and direction to optimize travel efficiency and safety. If the robot encounters large-scale obstacles or blockages that cannot be resolved through simple local adjustments, the Global Planner can be reactivated to update the path to circumvent the obstacle.

Through this collaborative effort, the Global planner and TEB together ensure effective navigation of the robot in complex and variable environments. The Global Planner provides macro-level navigational guidance, while the TEB is responsible for micro-level path adjustments and optimizations.

3.2.3 Navigate to ‘1’ in random boxes (Visual orientation)

It is challenging for the jackal robot to navigate itself to a randomly spawned object whose coordinates unrevealed. The key to parking beside the target No.1 box lies in how to find the target box and obtain its coordinates. Early tests showed that TEB local planner the jackal robot was able to avoid those random boxes and travel through boxes area. On this basis, the plan was design as:

1) Randomly set a goal position in the box area and move to that point.
2) Implementing template matching to lock on target box during the process to the random target.
3) If the target box was found, then terminate current action and reset the goal as heading toward the coordinates of the box calculated from the camera depth and internal parameters.
4) If the target box was not found until arriving the random target, then set another random goal position and move to it.
5) Loop until finally navigate to the box No.1.

As the coordinates of the random boxes are unavailable, a vision-based navigation method is necessary. Given that the Jackal robot was initially equipped with only a monocular camera, acquiring depth information or locating the boxes becomes unfeasible. Hence, the initial step involves adjusting the Jackal model's configuration file to select alternative cameras.

According to the tutorials online, there was a Kinect camera model supported for gazebo. First download the model from the official source and then modify the configuration files following tutorials, the depth information could be obtained from a new created topic “front/depth/image_raw”.

Template Matching

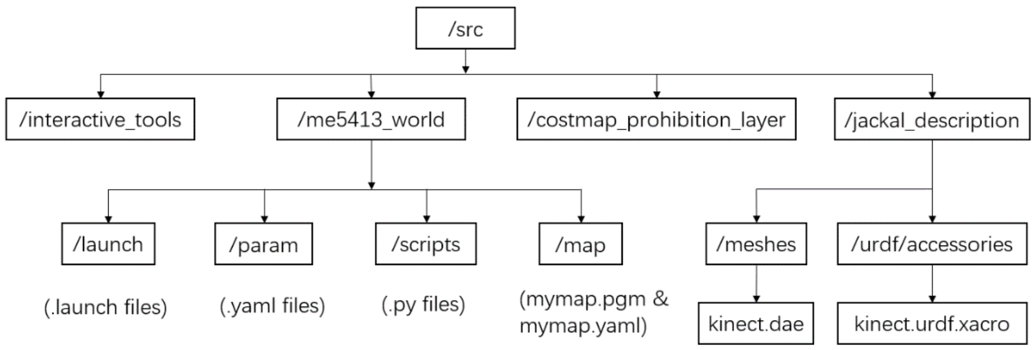
Based on the camera information, we firstly subscribe some topics from the gazebo environment. We subscribe the "/front/rgb/image_raw" to acquire the raw image, acquire the depth information from the “/front/depth/image_raw” and get the goal name from the "/rviz_panel/goal_name". The template matching process only works when the agent is searching for the ‘number 1’ box. Thus, template matching will only be activated when the goal name includes the sentence ‘box’.

The template matching method depends on a template figure. The camera pixels are 512x640. We set the template figure size as 124x160. However, due to the movement of the agent, the explicit size cannot be determined. A scaling method is used to detect the target number. Two different interpolar methods are used to fit the bigger size and smaller size figure. INTER_AREA interpolation method is suitable for image shrinking. It uses area-based interpolation, where the new pixel value is estimated based on the average of neighboring pixels' values. INTER_CUBIC interpolation method is suitable for image enlarging. It employs cubic interpolation polynomial to estimate the new pixel value. This method considers a larger neighborhood of pixels, hence better handling the details when enlarging images. We use the gray level image to match other than the binary image, because the environment in the gazebo is gray and changing with the light. It is really hard to distinguish the threshold.
We use ‘cv2.TM_CCOEFF_NORMED’ to realize the templating, anticipating a recall number bigger than 0.75 regarded as correct matching. After succeeding in matching, the system publishes detection 2D messages and the depth information. The Detection2D message type is employed to describe the information of objects or targets detected within an image.

Camera Depth Process
After the target type is set to a box, the visioin part starts working, but only when the template matching score reaches the threshold we set, the template matching will be considered successful and the coordinate of the box will be published as goal_pose. This goal is calculated by the internal parameters of the camera and the current depth. Specifically, we obtain the internal parameters of the camera by subscribing to the topic /front/rgb/camera_info, as shown in the *Figure 18*. As for depth, it is obtained by subscribing to the topic “/front/depth/image_raw”. After the current depth information and the camera's internal parameter information are available, the coordinates of the box in the camera frame are calculated. And then the coordinates is transformed to the map frame. When the template matching is successful, the coordinates will be published to the topic “/move_base_simple/goal”.

3.2.4 Specific operation process

Workspace structure:
In the project, the structure of files added and modified were reorganized as the figure below:



All ".launch" files and ".yaml" files modified or called were placed under respective folder. The python scripts for random boxes task were placed under "/scripts" folder. The map generated by SLAM was placed under "/map" folder. The configuration files for Kinect camera were added to respective directory.

In the starter code, the "navigation.launch" indicated the files to be modified which were launch files for localizer and move base. Noticing that those two files belong to a ROS jackal-navigation package, the initial step involves copying these two files and pasting them in the customized directory and subsequently making the necessary modifications. The "amcl.launch" is all about parameters of amcl so it was more like tuning than modifying. Here are some important parameters that were tuned:

odom_alpha 1~5: The noises of the odometry data

min_particles: The minimum number of particles allowed in the particle filter

max_particles The maximum number of particles allowed in the particle filter

laser_max_beam: The maximum number of laser beams used to process LiDAR data

With most other parameters being default values, the AMCL yielded good performance.

The "move_base.launch" file was also from "jackal_navigation" package provided in ROS, which configures the various components including local planner, global planner, local costmap, global costmap and loads respective parameters for them.

To achieve better performance, the parameters of those system components were well tuned. In "prohibition_areas.yaml", the prohibition area was defined. In "teb_planner_param.yaml", robot velocity range parameters were tuned as jackal robot capacities, which were obtained by manually controlling the robot to move whilst listening to odom topic information. For the Global Planner, there were 3 built-in planners: carrot planner, navfn and global_planner. After testing, the global_planner was chosen as it yielded best performance with both A-star and Dijkstra algorithms. *Figure 19* shows the result of navfn planner, which failed to generate a full path. As mentioned in the introduction of costmap, the update frequency and publish frequency and inflation radius were well tuned with other parameters slightly tuned.

With all tuning and file configurations done, all tasks can be successfully accomplished.

3.3 Results and Discussion

The results of the navigation are shown in the *Figure 17*. Since the heading error is related to the target goal heading. The heading error just shows the rotation of the agents in the direction to the target point. The reason why there is a period of 0 in the middle of the runtime is that when the agent moves towards the box, no certain position is published. The absolute position error is 0 at last, meaning that the agent arrives at the final goal.

Random box problem:

When the agent goes to the box 1, the explicit location of the box is unknown. Thus, a virtual goal must be set to enable the agent to go to this goal. On the way to this goal, if the agent can see the box 1, the template matching works and publish a new goal, which is the location of the box acquired by the vision, driving the agent reaches the box 1. However, if the agent cannot see the target box, the vision cannot work. We can just randomize the location until the agent can see the box 1 on the path.

Thus, a few virtual locations are selected at first. After the box locations are set, the button 'box 1' is clicked to enable the agent to go to the virtual location, which is located at the same time when clicking. A global planning path can be set. The virtual location can be renewed by clicking the button again if the agent cannot see the target box on the global planning path. By looping this process, the agent can get access to the target box definitely.

Future Improvement:

Drift problem:

To solve the drift problem, simultaneous utilization of AMCL and EKF is an effective strategy. EKF (Extended Kalman Filter) is a method based on Kalman filtering suitable for dealing with nonlinear systems. It extends the linear assumptions of Kalman filtering using the first-order Taylor series expansion to handle nonlinear functions, thereby estimating the robot's state. Running both EKF and AMCL concurrently allows the two filters to independently update states. The weights of the outputs from EKF and AMCL are dynamically adjusted based on environmental features and the reliability of sensors. This leverages the continuous localization advantage of EKF to handle high-frequency update demands, while using the environmental matching strength of AMCL for regular corrections to counteract the cumulative errors in EKF.

Not totally correct location:

After the agent finds the target box, a new target goal is set based on the depth calculation in the file 'camera_calculation.py'. However, this calculation is not quite correct. The position of the final goal sometimes cannot quite near the box.
These remaining problems remain to be improved.

Appendix

	Gmapping	A-Loam	Fast-Lio
Mean	6.17	0.97	0.36
Rmse	7.33	1.09	0.43
std	3.96	0.49	0.23

Table 1

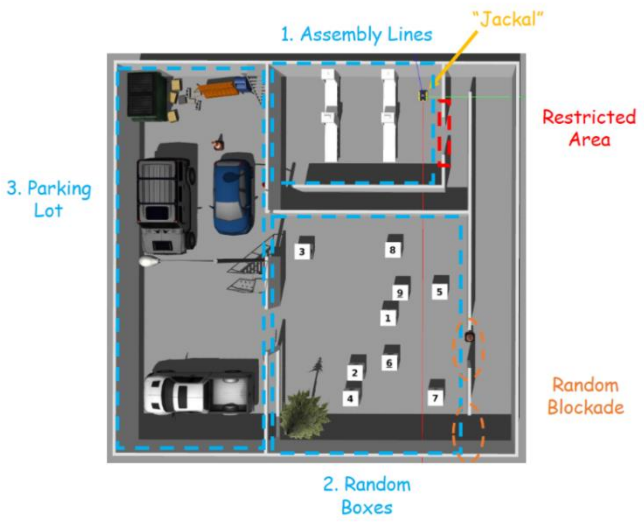
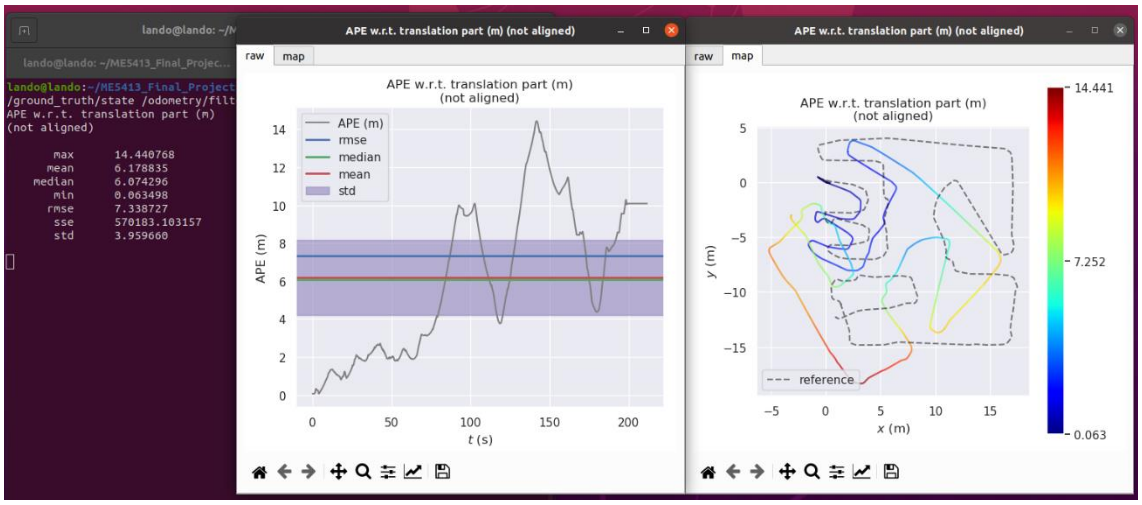


Figure 1 Schematic of the given mini-factory



```
lando@lando:~/ME5413_Final_Project
/ground_truth/state /odometry/filt
APE w.r.t. translation part (m)
(not aligned)

       max      14.440768
      mean      6.178835
    median      6.074296
       min      0.063498
      rmse      7.338727
       sse      570183.103157
       std      3.959660
```

lando@lando: ~/M
lando@lando: ~/ME5413_Final_Projec...
APE w.r.t. translation part (m) (not aligned)
raw
map
APE w.r.t. translation part (m)
(not aligned)
APE (m)
rmse
median
mean
std
t (s)
APE w.r.t. translation part (m) (not aligned)
raw
map
APE w.r.t. translation part (m)
(not aligned)
y (m)
x (m)
reference
14.441
7.252
0.063

Figure 2 The Error of the map by Gmapping

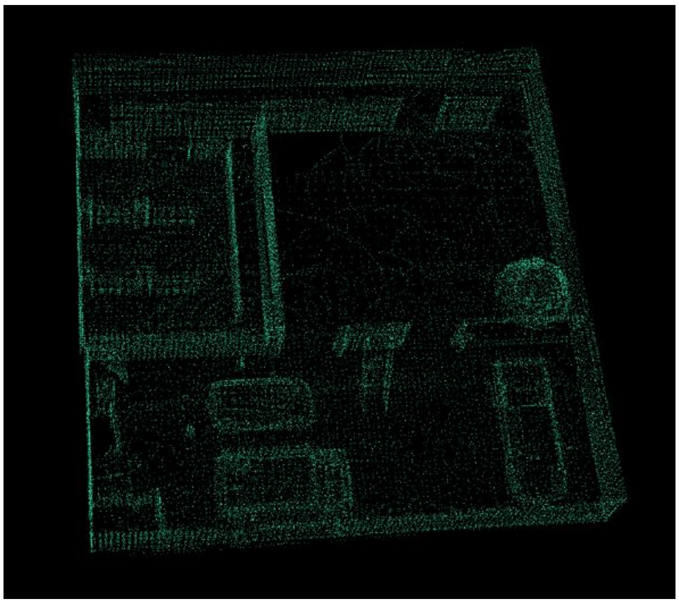
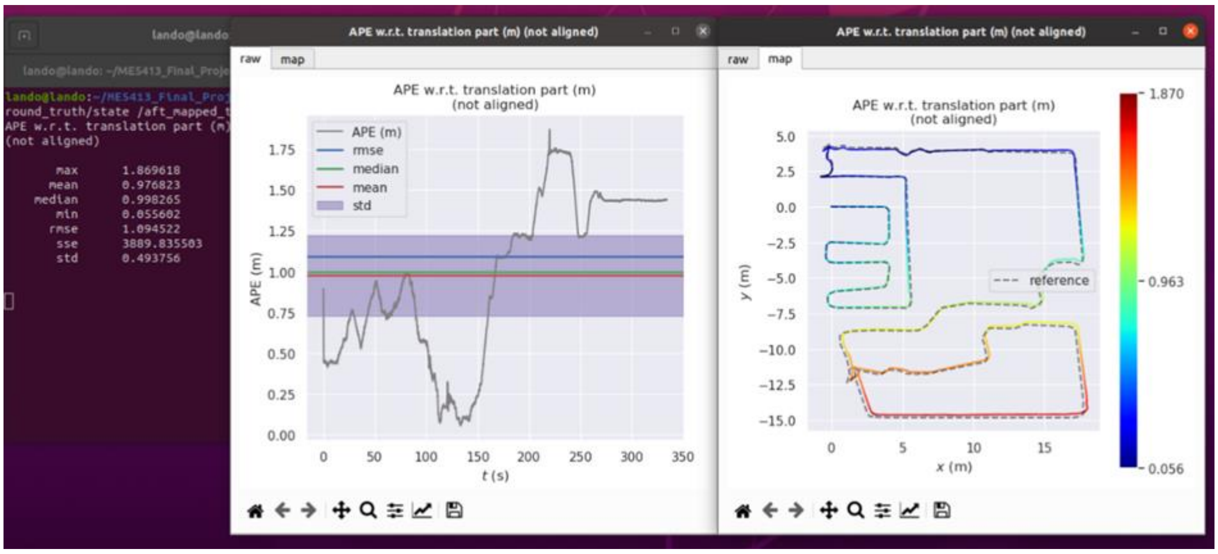


Figure 3 Point cloud of A-Loam



```
lando@lando:~/MES413_Final_Proj
round_truth/state /aft_mapped_t
APE w.r.t. translation part (m)
(not aligned)

       max	1.869618
      mean	0.976823
    median	0.998265
       min	0.055602
      rmse	1.094522
       sse	3889.835503
       std	0.493756
```

lando@lando
lando@lando: ~/MES413_Final_Proje
APE w.r.t. translation part (m) (not aligned)
raw
map
APE w.r.t. translation part (m)
(not aligned)
APE (m)
rmse
median
mean
std
t (s)
APE (m)
y (m)
x (m)
reference
1.870
0.963
0.056

Figure 4 The Error of the map by A-Loam

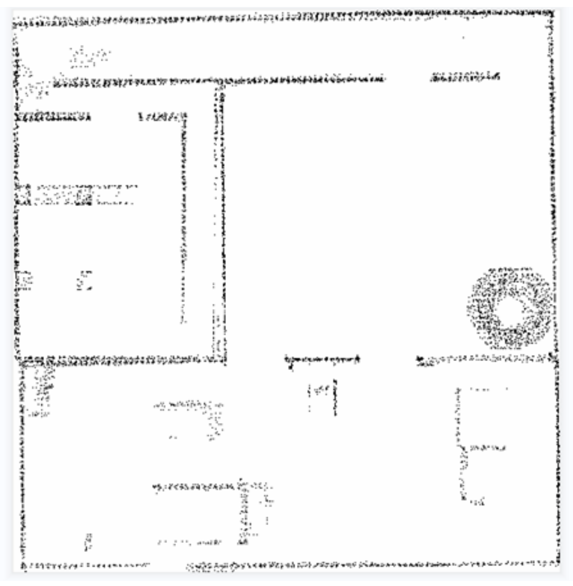


Figure 5 The 2D map generated from A-Loam

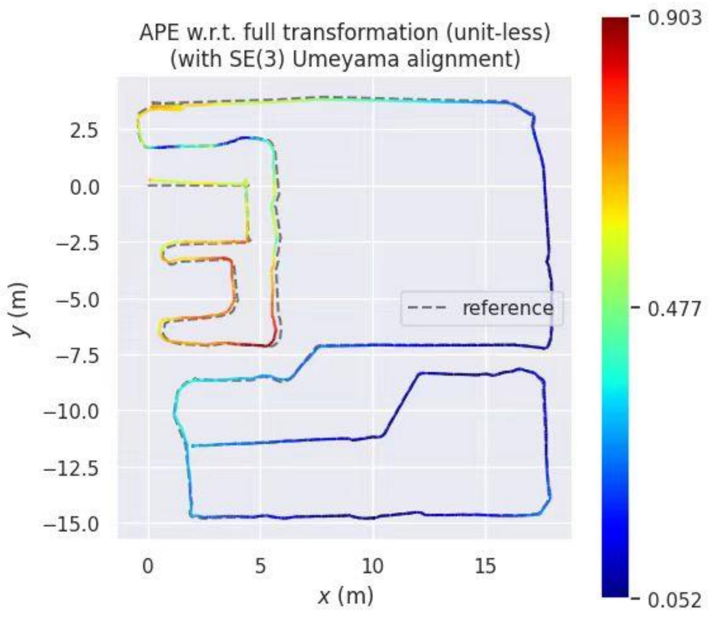


Figure 6 The Error of the map by Fast-Lio

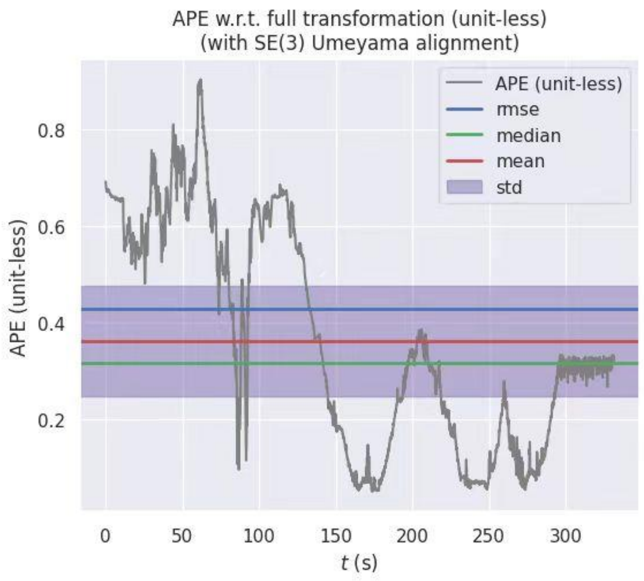


Figure 7 The Error of the map by Fast-Lio

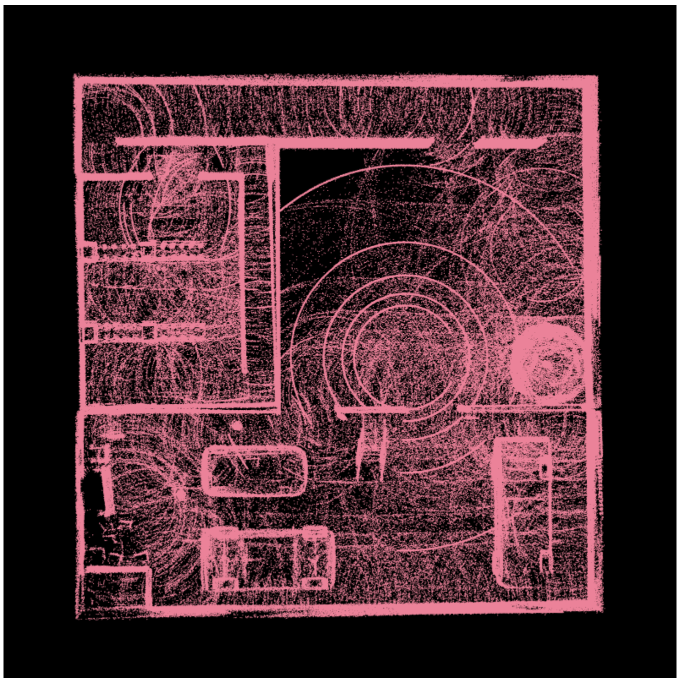


Figure 8 The Point Cloud generated by Fast-Lio

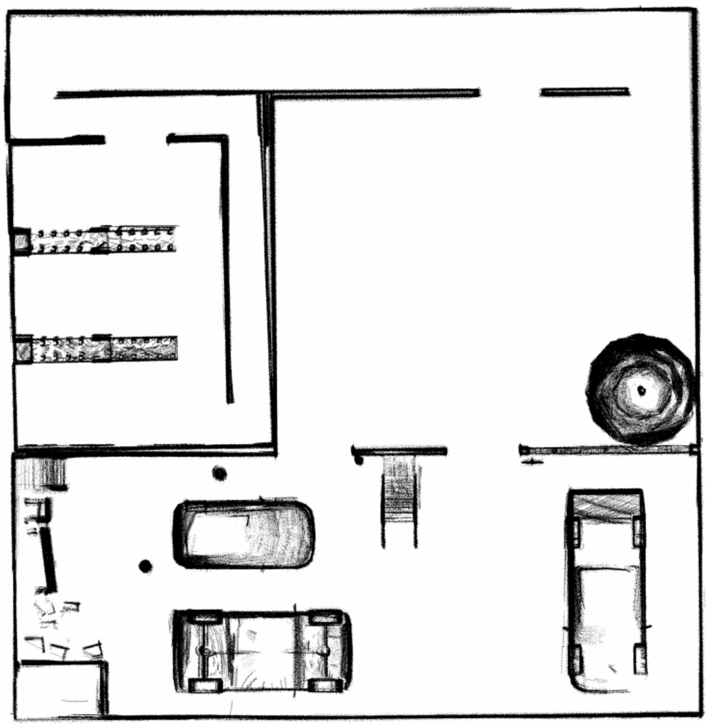


Figure 9 The Converted Map

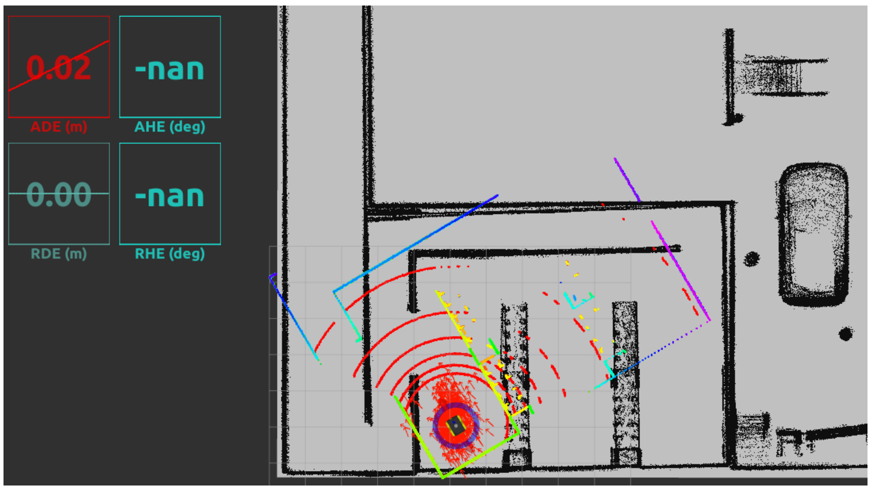


Figure 10 Localization Drift

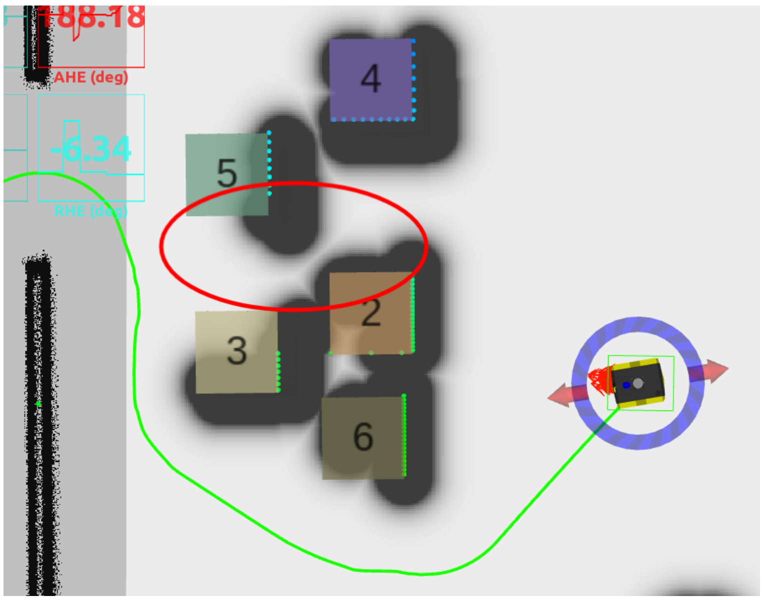


Figure 11 Detour

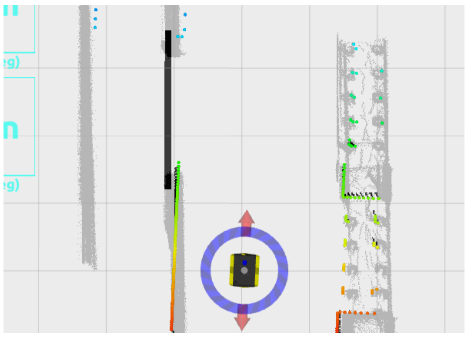
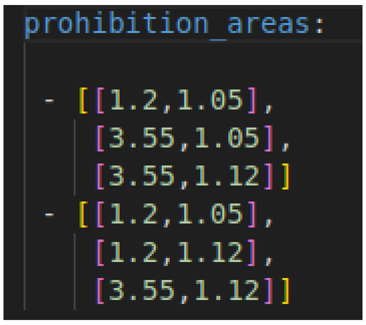


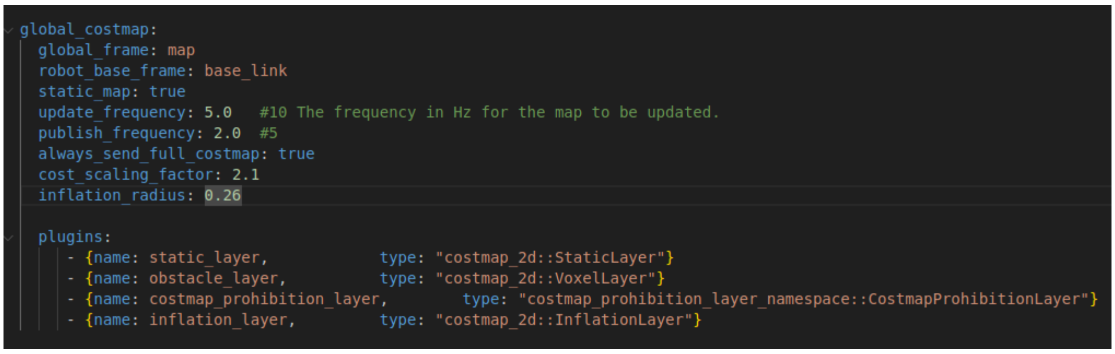
Figure 12 Prohibition Layer



```yaml
prohibition_areas:

 - [[1.2,1.05],
    [3.55,1.05],
    [3.55,1.12]]
 - [[1.2,1.05],
    [1.2,1.12],
    [3.55,1.12]]
```

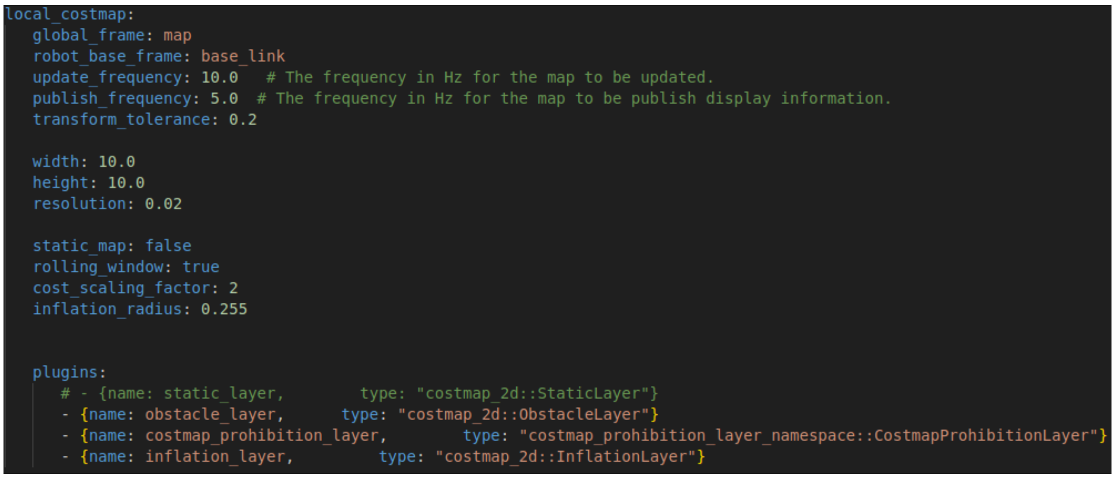
Figure 13 prohibition_areas.yaml



```yaml
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  static_map: true
  update_frequency: 5.0   #10 The frequency in Hz for the map to be updated.
  publish_frequency: 2.0  #5
  always_send_full_costmap: true
  cost_scaling_factor: 2.1
  inflation_radius: 0.26

  plugins:
    - {name: static_layer,            type: "costmap_2d::StaticLayer"}
    - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
    - {name: costmap_prohibition_layer,        type: "costmap_prohibition_layer_namespace::CostmapProhibitionLayer"}
    - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

Figure 14 global_costmap_params.yaml



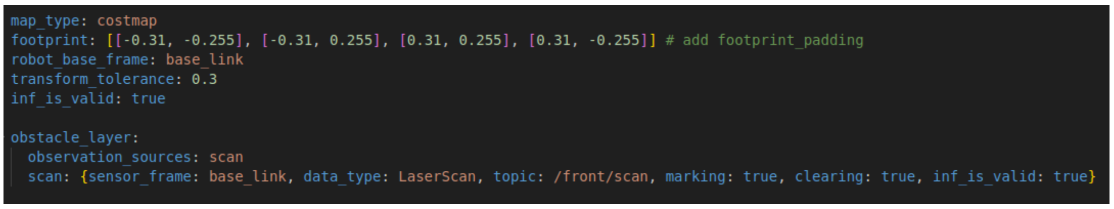
```yaml
local_costmap:
   global_frame: map
   robot_base_frame: base_link
   update_frequency: 10.0   # The frequency in Hz for the map to be updated.
   publish_frequency: 5.0  # The frequency in Hz for the map to be publish display information.
   transform_tolerance: 0.2

   width: 10.0
   height: 10.0
   resolution: 0.02

   static_map: false
   rolling_window: true
   cost_scaling_factor: 2
   inflation_radius: 0.255


   plugins:
      # - {name: static_layer,        type: "costmap_2d::StaticLayer"}
      - {name: obstacle_layer,      type: "costmap_2d::ObstacleLayer"}
      - {name: costmap_prohibition_layer,        type: "costmap_prohibition_layer_namespace::CostmapProhibitionLayer"}
      - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

Figure 15 local_costmap_params.yaml



```yaml
map_type: costmap
footprint: [[-0.31, -0.255], [-0.31, 0.255], [0.31, 0.255], [0.31, -0.255]] # add footprint_padding
robot_base_frame: base_link
transform_tolerance: 0.3
inf_is_valid: true

obstacle_layer:
  observation_sources: scan
  scan: {sensor_frame: base_link, data_type: LaserScan, topic: /front/scan, marking: true, clearing: true, inf_is_valid: true}
```

Figure 16 costmap_common_params.yaml

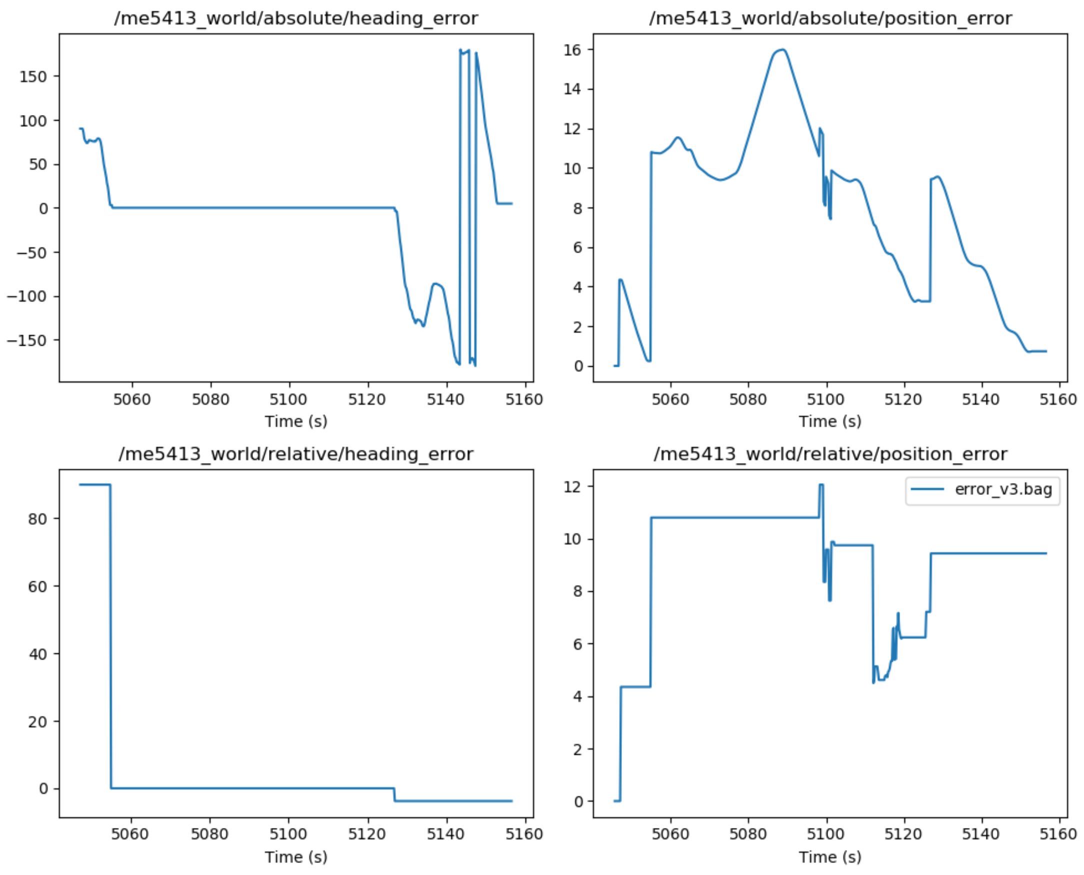
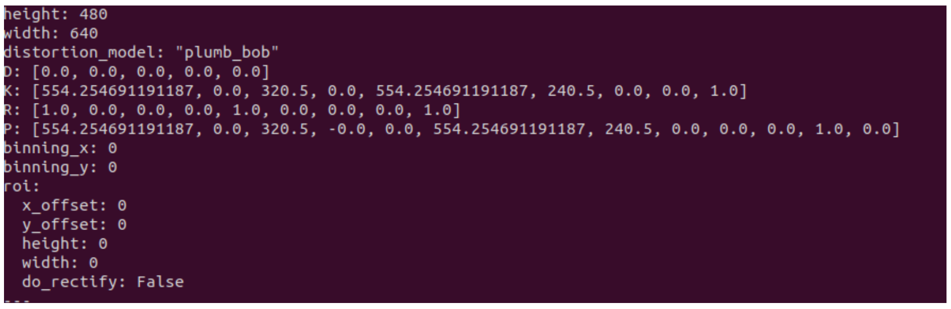


Figure 17 The Error of Navigation



```
height: 480
width: 640
distortion_model: "plumb_bob"
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [554.254691191187, 0.0, 320.5, 0.0, 554.254691191187, 240.5, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [554.254691191187, 0.0, 320.5, -0.0, 0.0, 554.254691191187, 240.5, 0.0, 0.0, 0.0, 1.0, 0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
  do_rectify: False
```

Figure 18 Camera Internal Parameters

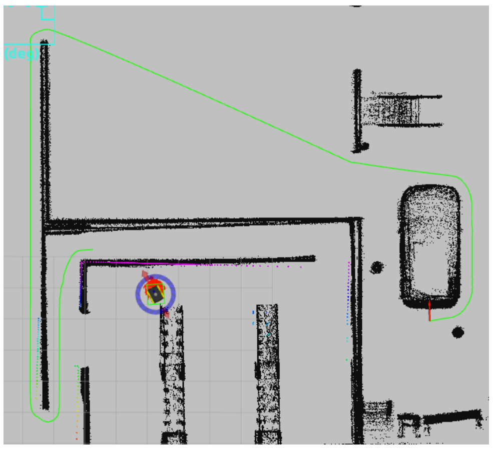


Figure 19 NAVFN Planner Result

Reference

[1] https://github.com/hku-mars/FAST_LIO

[2] https://github.com/nuslde/aloam_lidar_odom_result_generate

[3] http://wiki.ros.org/costmap_2d#Occupied.2C_Free.2C_and_Unknown_Space

[4] https://blog.csdn.net/weixin_43928944/article/details/115905084

[5] https://blog.csdn.net/m0_56588389/article/details/121043489

[6] Zheng K. Ros navigation tuning guide[J]. Robot Operating System (ROS) The Complete Reference (Volume 6), 2021: 197-226.

[7] https://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide