# Group 1 ME5413 Final Project

Jia Yansong (A0263119H), Li Jiawei (A0263169X), Deng Chengruyi (A0263158B), and Gao Ziteng (A0263505J)

March 30, 2023

Code is available: https://github.com/JYS997760473/ME5413_Final_Project

## 1 Task 1: Mapping

For the mapping part, we use three different SLAM algorithms (Gmapping, Cartographer, and ALOAM) to build the map for the simulation environment, and the mapping results are evaluated by EVO[1], a python package for evaluating odometry and SLAM.

**Evaluation Metrics** The main evaluation metrics for the mapping task are Absolute Pose Error (APE) and Root Mean Square Error (RMSE):

$$APE_i = |P_{est,i} - P_{ref,i}| \tag{1}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} APE_i^2} \tag{2}$$

where $P_{est,i}$ is the estimated pose, and $P_{ref,i}$ is the groundtruth pose.

### 1.1 GMapping

#### 1.1.1 Introduction

GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from raw laser range data and odometry. ROS gmapping contains a ROS wrapper for OpenSlam's Gmapping implemented by [2] and [3], and it provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called *slam_gmapping*.

#### 1.1.2 Pipeline

The pipeline of GMapping is shown in Figure 1. The input data is laser and odometry, then the laser is first pre-processed to remove outliers and spurious readings, and the odometry data is used to estimate the robot's motion since the last pose estimation. Next, the pre-processed data is used to match the current scan to the previous scan using a technique called scan matching. This involves finding the pose that aligns the current scan with the previous scan as closely as possible. Then, the matched scan is used to estimate the robot's pose in the environment. This involves finding the pose that best explains the observed range data and odometry data. In the map update part, the estimated pose is used to update the map of the environment. The map is represented as a grid of occupancy probabilities, where each cell in the grid represents the probability of the corresponding area being occupied by an obstacle. Then, the algorithm checks for loop closures, which occur when the robot revisits a previously visited location. This involves comparing the current scan with previous scans to detect similarities and estimate the pose of the robot when it previously visited that location. Next, the algorithm uses loop closure information to optimize the map and pose estimates. This involves adjusting the poses of the robot and the map to minimize the errors in the loop closures. Finally, the final output of the GMapping algorithm is a map of the environment.

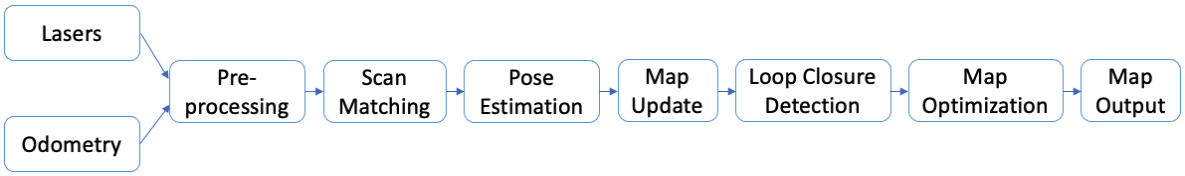



Figure 1: GMapping Pipeline

The pipeline of the whole mapping process using GMapping is shown in Figure 2. First launch the gazebo world environment, then launch the GMapping node and record */gazebo/ground_truth/state* and */odometry/filtered* topics at the same time, after mapping, save the map and use EVO[1] to evaluate the slam results stored in the rosbag recorded.

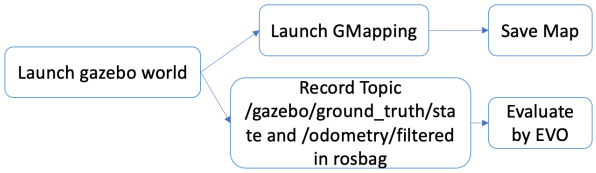



Figure 2: Pipeline of the whole mapping process using GMapping

The node graph of *slam_gmapping* is shown in Figure 3, where */gazebo* is the gazebo environment node, */slam_gmapping* and */ekf_localization* nodes are used to map the environment, and finally, the map is in */map* topic.

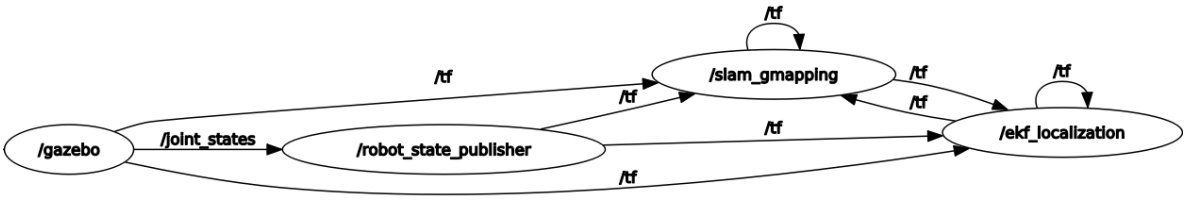



Figure 3: Node Graph

### 1.1.3 Map Result

The final map is shown in Figure 4.

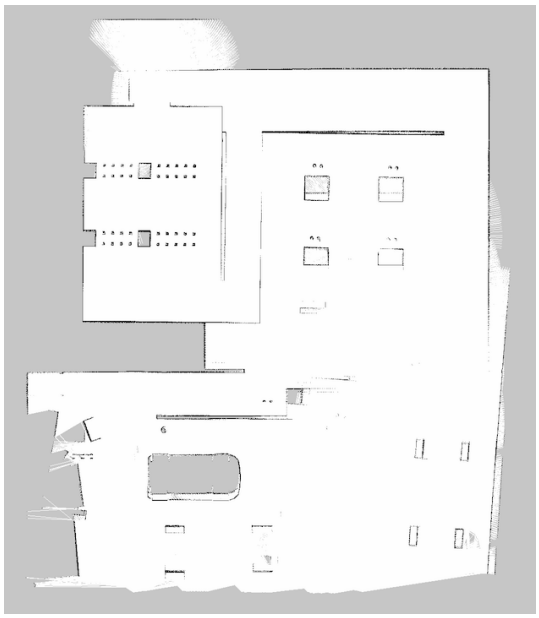
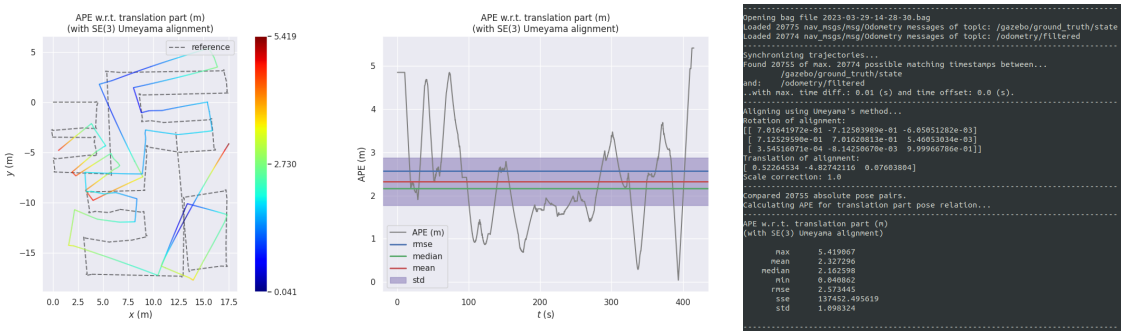


Figure 4: Final Map by GMapping

### 1.1.4 Evaluation

The mapping result is evaluated by EVO[1], and the evaluation map, evaluation metrics $APE$ and $RMSE$ versus time, and evaluation results are shown in Figure 5a, Figure 5b and Figure 5c respectively.




(a) Evaluation Map (b) Evaluation metrics versus time (c) Evaluation results in terminal

Figure 5: Mapping evaluation results by EVO[1]

From Figure 5a we can see that there is a big difference between the groundtruth trajectory and the actual trajectory. From Figure 5c, the max and mean Absolute Pose Error (APE) are $5.419067m$ and $2.327296m$ respectively. The Root Mean Square Error (RMSE) is $2.573445m$.

### 1.1.5 Summary

Although the map generated by GMapping generally matches the groundtruth map, there are still differences in some details. Also, since it only uses 2D lasers, it can only do 2D mapping, so from Figure 4, we can see that it only draws the tires of two cars on the map, and misses a hollow wall on the right side of the map.

## 1.2 Cartographer

### 1.2.1 Introduction

Cartographer is a system that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations, and it has been integrated into ROS.

### 1.2.2 Pipeline

The pipeline of the whole mapping process using Cartographer is shown in Figure 6. First, we modify the *backpack_2d.lua* of Cartographer and create a new *mapping.launch* file. The modified *backpack_2d.lua* file and new *mapping.launch* file are shown in Listing 1 and Listing 2. Then, launch the gazebo world environment. Next, launch the Cartographer mapping node and at the same time, record */gazebo/ground_truth* and */odometry/filtered* topics. Finally, after mapping, save the final map and evaluate the slam result using EVO[1].


Figure 6: Pipeline of the whole mapping process using Cartographer

The node graph of mapping by the Cartographer is shown in Figure 7.

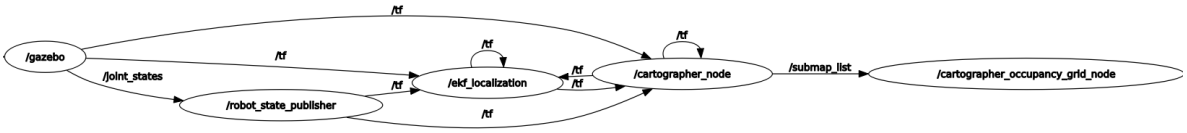



Figure 7: Node Graph

### 1.2.3 Map Result

The final map generated is shown in Figure 8.

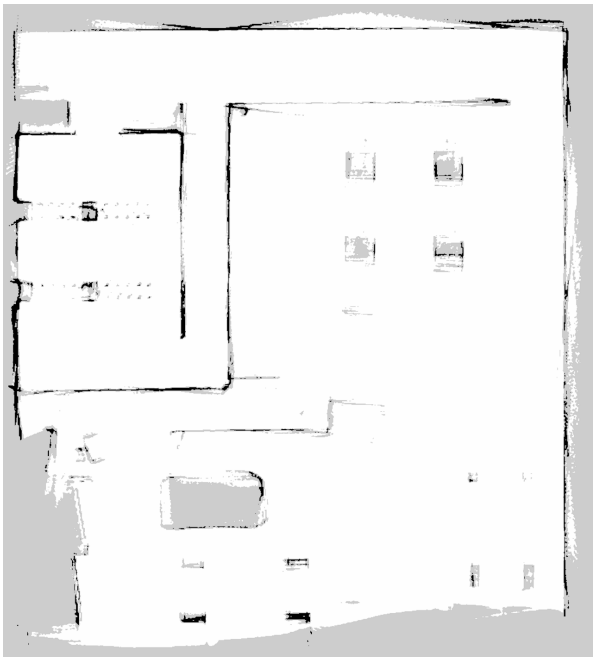
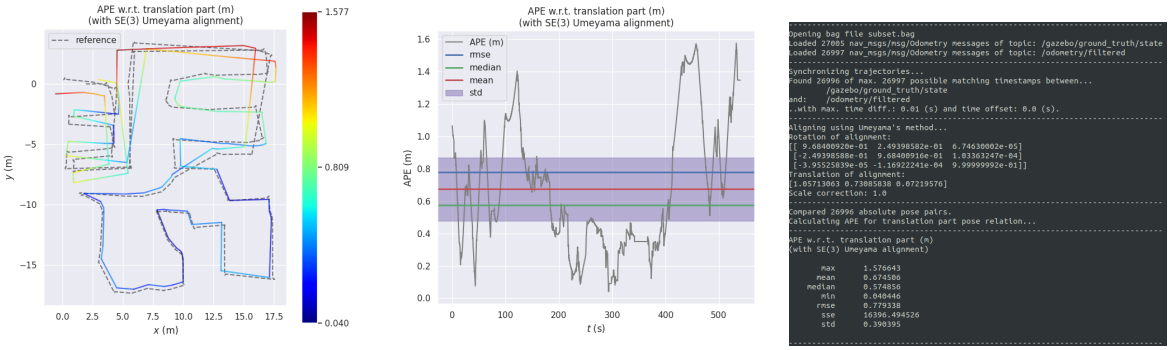


Figure 8: Final Map by Cartographer

### 1.2.4 Evaluation

The evaluation map, evaluation metrics $APE$ and $RMSE$ versus time, and evaluation results are shown in Figure 9a, Figure 9b, and Figure 9c respectively.




(a) Evaluation Map (b) Evaluation metrics versus time (c) Evaluation results in terminal

Figure 9: Mapping evaluation results by EVO[1]

From Figure 9a we can see that compared with Figure 5a, the cartographer's map is more accurate, though there is still some error between the groundtruth trajectory. From Figure 9c, the max and

mean Absolute Pose Error (APE) are 1.576643m and 0.674506m respectively. The Root Mean Square Error (RMSE) is 0.779338m.

#### 1.2.5 Summary

Compared with the evaluation results with GMapping's, Cartographer's *APE* and *RMSE* are much smaller, and the cartographer's result is much more accurate. However, from the generated map shown in Figure 8, it is obvious that the boundaries of walls and obstacles are very blurred and distorted, and also, because we only use 2D cartographer, we still cannot map the full outlines of the vehicles and the hollow wall on the right side of the map, which are the biggest problem of the cartographer.

### 1.3 ALOAM

#### 1.3.1 Introduction

A-LOAM is an Advanced implementation of LOAM[4], which uses Eigen and Ceres Solver to simplify code structure.

#### 1.3.2 Pipeline

The pipeline of the whole process using ALOAM is shown in Figure 10. First, we modify the *aloam_velodyne_VLP_16.launch* file of ALOAM to remap the topic */velodyne_points* to */mid/points*, and the modified file is Listing 3. Then launch the gazebo environment world, and then launch the ALOAM mapping node and record all topics in a rosbag. After walking all over the environment, use *pcl_ros* ros package to convert the 3D point cloud mapping result in topic */laser_cloud_map* shown in Figure 11a to *.pcd* files, and then choose the final *.pcd* file and use pcd2pgm (https://github.com/Hinson-A/pcd2pgm_package/tree/develop) to convert it to the 2D map shown in Figure 11b. Because the ALOAM uses 3D lidar point cloud to do slam, and there are a lot of noise points on the ground in Figure 11b, so we use GIMP (https://www.gimp.org/) to manually remove those noise black points mapped on the ground, and finally, the 2D map is shown in Figure 11c, whose resolution is 0.02m.

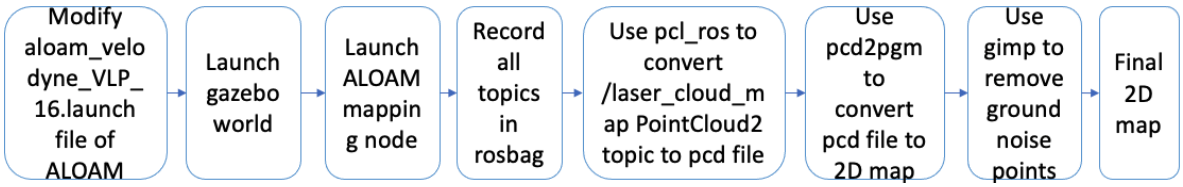
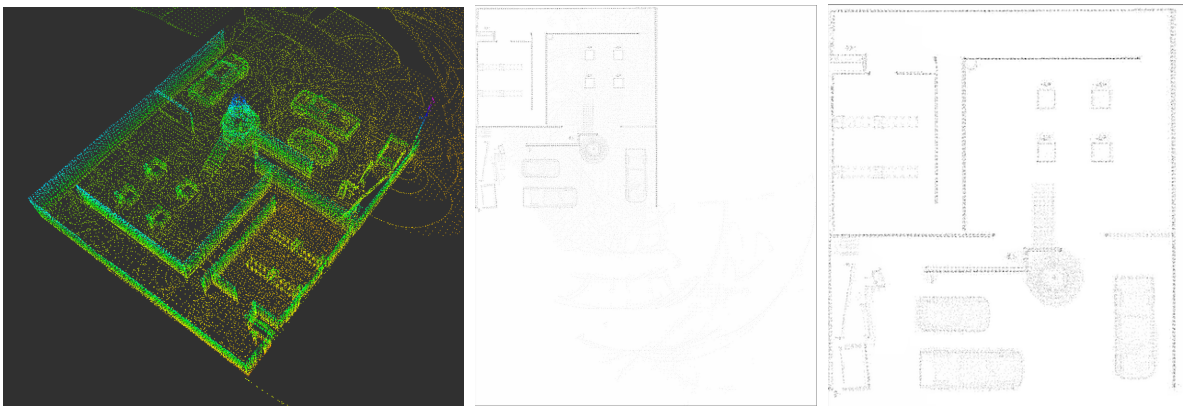



Figure 10: Pipeline of the whole mapping process using ALOAM

#### 1.3.3 Map Result

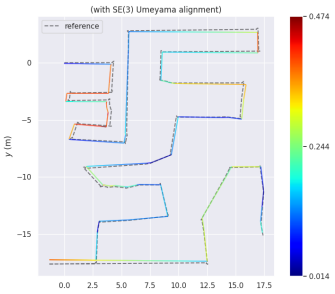


(a) 3D map (b) 2D map (c) 2D map after ground removal

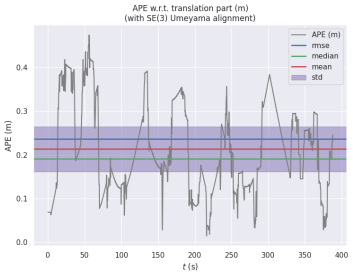
Figure 11: Map Results

### 1.3.4 Evaluation

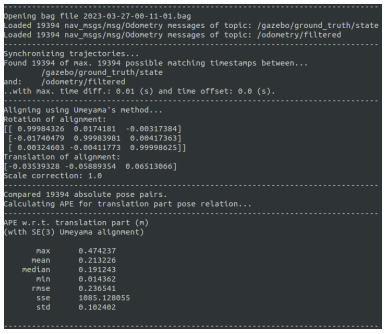
The evaluation map, evaluation metrics $APE$ and $RMSE$ versus time, and evaluation results are shown in Figure 12a, Figure 12b and Figure 12 respectively




(a) Evaluation Map




(b) Evaluation metrics versus time




(c) Evaluation results in terminal

Figure 12: Mapping evaluation results by EVO[1]

From Figure 12a we can see that the estimated trajectory is almost the same as the groundtruth trajectory, which is much better than those of GMapping and Cartographer. From Figure 12c, the max and mean Absolute Pose Error (APE) are 0.474237m and 0.213226m respectively. The Root Mean Square Error (RMSE) is 0.236541m.

### 1.3.5 Summary

Compared with the evaluation results of GMapping and Cartographer, it is obvious that ALOAM is the best, what is more, from Figure 11a and Figure 11c we can see that in ALOAM's map, the outlines of three vehicles and the hollow wall on the side of the environment are clearly drawn on the 2D map, which is quite important for the next navigation task.

## 1.4 Summary

In this mapping task part, we use GMapping, Cartographer, and ALOAM to do SLAM and generate three different maps of the environment. The evaluation results by EVO[1] are shown in Table 1, and it is obvious that ALOAM is the best, and it maps all the obstacles and walls in the environment to the 2D map in great detail. We use the map generated by ALOAM to do navigation in next navigation task part.

Table 1: Evaluation results of three SLAM algorithms

| Algorithm / Evaluation Metric (m) | GMapping | Cartographer | ALOAM |
|---|---|---|---|
| Max APE | 5.419067 | 1.576643 | 0.474237 |
| Mean APE | 2.327296 | 0.674506 | 0.213226 |
| Median APE | 2.162598 | 0.574856 | 0.191243 |
| Min APE | 0.040862 | 0.040446 | 0.014362 |
| RMSE | 2.573445 | 0.779338 | 0.236541 |

# 2 Task 2: Navigation

## 2.1 Pipeline

The pipeline of navigation is shown in Figure 13. First, we tune the parameters for *move_base* node and prepare *.launch* and *.yaml* files. Next, because there is an inaccessible area, and there is no door in this area, we use *GIMP* (https://www.gimp.org/) manually draw doors for the 2D map,

and the doors are shown in red circles in Figure 14. Then launch the gazebo world environment and launch the navigation node. Finally, choose the goal pose and position to navigate the robot.

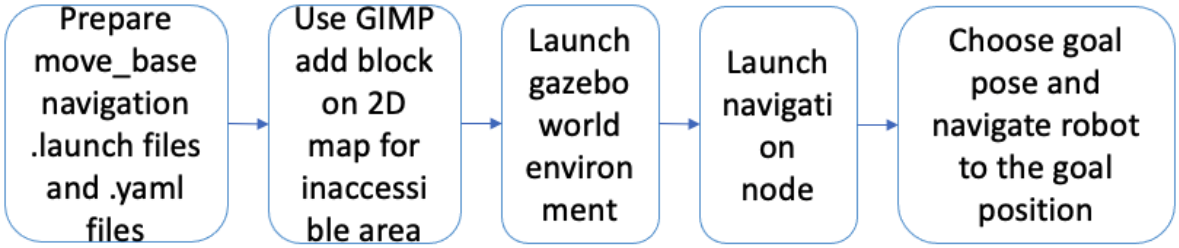



Figure 13: Pipeline of navigation

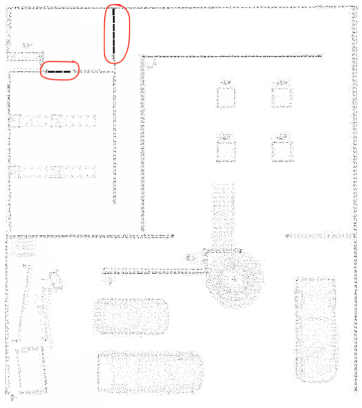


Figure 14: Doors drawn on the 2D map

## 2.2 Move_base

The *move_base* package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The *move_base* node links together a global and local planner to accomplish its global navigation task. The *move_base* node maintains two cost maps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks. We tune the parameters of the *move_base* package and prepare *.launch* and *.yaml* files to make the robot follow the planned global path strictly. The *move_base.launch* file is shown in Listing 4, *global_costmap_params.yaml* file is shown in Listing 5, *local_costmap_params.yaml* file is shown in Listing 6, *base_local_planner_params.yaml* file is shown in Listing 7 and *costmap_common_params.yaml* file is shown in Listing 8. We mainly tune the size of local cost map smaller, so that it will follow the planned path from the global cost map.

## 2.3 Localization

The localization algorithm used is AMCL, which is a probabilistic localization system for a robot moving in 2D. It implements the KLD-sampling Monte Carlo localization approach[5] which uses a particle filter to track the pose of a robot against a known map. AMCL takes in a laser-based map, and its map localization pipeline is shown in Figure 15.

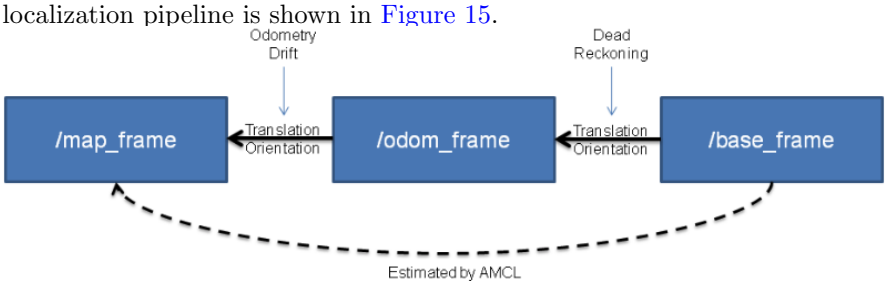



Figure 15: Pipeline of AMCL

## 2.4 Planning

### 2.4.1 Global Planning: Dijkstra

We use the $navfn$ package implemented in ROS to do global planning, and the $navfn$ package uses the Dijkstra algorithm[6] to plan the shortest path for the robot to move from the current position to the goal position and pose.

Dijkstra's algorithm finds the shortest path between two nodes in a graph by starting at a given source node and always selecting the closest unvisited node to the source using a greedy strategy. The algorithm maintains two sets of vertices: one for known vertices and another for unknown vertices. Initially, all vertices are unknown and the distance between all pairs of vertices is set to infinity. It updates the distances of neighboring vertices and adds the closest one to the set of known vertices until all vertices are known. The shortest distances of the vertices in the known set are guaranteed to be determined, so no further updates are needed. Finally, the algorithm returns the shortest path from the source vertex to all other vertices in the graph.

### 2.4.2 Local Planning: Trajectory Rollout

The local planner in ROS searches for the best path by creating multiple trajectories and selecting the best one based on collision check and performance score. The Trajectory Rollout algorithm is used for path planning in unknown environments, allowing the robot to avoid obstacles and navigate autonomously. The algorithm samples discrete points in robot control space, performs a forward simulation to predict the robot's movement, evaluates each trajectory based on proximity to the obstacle, target, global path, and velocity, and selects the highest score trajectory to send to the base. The process is repeated continuously.

The difference between DWA and Trajectory Rollout lies in how they sample the robot's control space. Trajectory Rollout samples all available velocities during the entire simulation phase, while DWA samples only during one simulation step. DWA is more efficient as it uses less sampling space, but Trajectory Rollout may be better for low-acceleration robots that require constant acceleration during forward simulation.

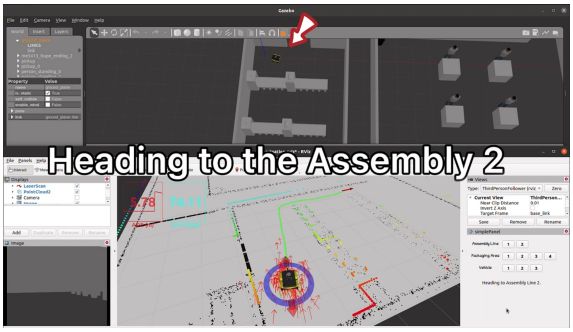
## 2.5 Result and Evaluation

We are group 1, so our navigation task is first to move to Assembly Line 2, then move to Package Area 2, and finally move to Vehicle 2.

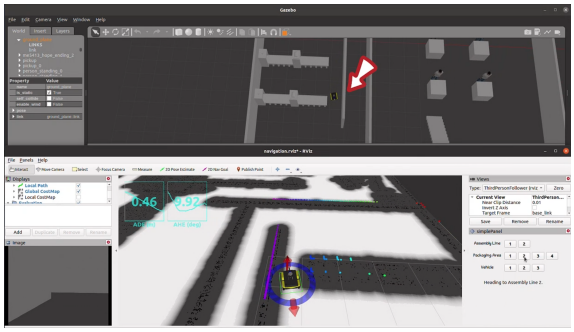
The results are evaluated by position error and heading error.

### 2.5.1 To Assembly Line 2

The planned path to assembly line 2 is shown in Figure 16a, and the arriving pose is shown in Figure 16b.




(a) Heading to Assembly Line 2



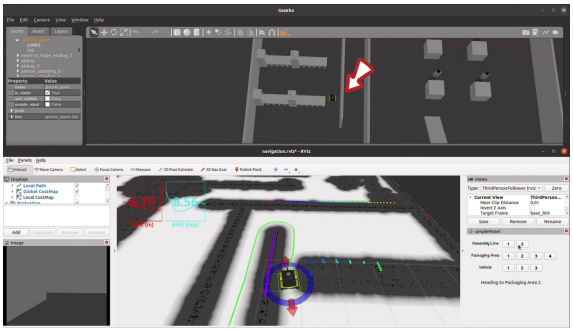
(b) Arrive Assembly Line 2

Figure 16: To Assembly Line 2

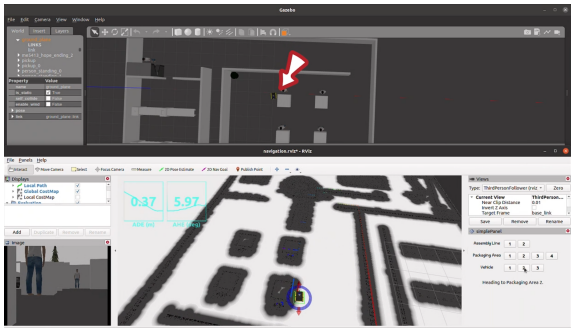
From Figure 16b, the position error is 0.46m and the heading error is 9.92 degrees.

### 2.5.2 To Package Area 2

The planned path to Package Area 2 is shown in Figure 17a, and the arriving pose is shown in Figure 17b.



(a) Heading to Package Area 2



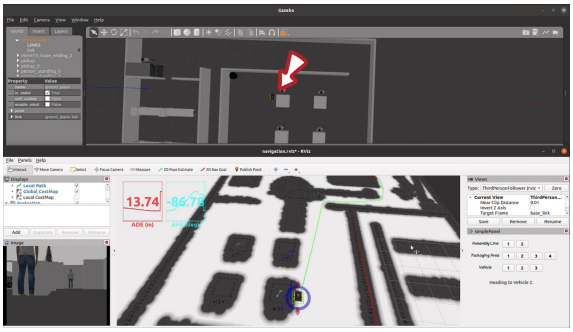
(b) Arrive Package Area 2

Figure 17: To Package Area 2

From Figure 17b, the position error is 0.37m and the heading error is 5.97 degrees.

### 2.5.3 To Vehicle 2

The planned path to Vehicle 2 is shown in Figure 18a, and the arriving pose is shown in Figure 18b.



(a) Heading to Vehicle 2


(b) Arrive Vehicle 2

Figure 18: To Vehicle 2

From Figure 18b, the position error is 0.17m and the heading error is 0.16 degrees.

## 2.6 Summary

From these three sub-navigation tasks, the average position error is 0.33m, and the average heading error is 5.35 degrees. Meanwhile, we find that as the navigation task progresses, the errors of both position and heading is gradually decreasing.

# 3 Summary

In this final project, we successfully design a robot navigation software stack. We first use three different SLAM algorithms (GMapping, Cartographer, and ALOAM) to do mapping for the whole simulation environment, and finally choose the map by ALOAM, then we use the map to navigate the robot to move to where we expected.

# References


[1] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[2] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," 01 2005, pp. 2432–2437.

[3] ——, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[4] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," 07 2014.

[5] D. Fox, "Kld-sampling: Adaptive particle filters." 01 2001, pp. 713–720.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

# 4 Appendix

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  -- published_frame = "base_link",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_pose_extrapolator = true,
  use_odometry = true,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

MAP_BUILDER.use_trajectory_builder_2d = true
TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1
TRAJECTORY_BUILDER_2D.use_imu_data = false

return options
```

Listing 1: backpack_2d.lua

```
<launch>

    <!-- Connect the robot to a keyboard teleop controller -->
    <node name="teleop_twist_keyboard" pkg="teleop_twist_keyboard" type="
    teleop_twist_keyboard.py" output="screen" respawn="true"/>

    <node name="cartographer_node" pkg="cartographer_ros"
        type="cartographer_node" args="
            -configuration_directory $(find cartographer_ros)/configuration_files
            -configuration_basename backpack_my_2d.lua"
        output="screen">
        <remap from="echoes" to="horizontal_laser_2d" />
        <remap from="scan" to="front/scan" />
        <remap from="odom" to="odometry/filtered" />
    </node>


    <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
        type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

    <!-- Launch Rviz with our settings -->
    <node type="rviz" name="rviz" pkg="rviz" args="-d $(find me5413_world)/rviz/
    gmapping.rviz" output="log" respawn="true"/>

</launch>
```

Listing 2: cartographerMapping.launch

```
<launch>

    <param name="scan_line" type="int" value="16" />

    <!-- if 1, do mapping 10 Hz, if 2, do mapping 5 Hz. Suggest to use 1, it will
    adjust frequence automaticlly -->
    <param name="mapping_skip_frame" type="int" value="1" />

    <!-- remove too closed points -->
    <param name="minimum_range" type="double" value="0.3"/>

    <remap from="/velodyne_points" to="/mid/points" />

    <param name="mapping_line_resolution" type="double" value="0.2"/>
    <param name="mapping_plane_resolution" type="double" value="0.4"/>

    <node pkg="aloam_velodyne" type="ascanRegistration" name="ascanRegistration"
    output="screen" />

    <node pkg="aloam_velodyne" type="alaserOdometry" name="alaserOdometry" output="
    screen" />

    <node pkg="aloam_velodyne" type="alaserMapping" name="alaserMapping" output="
    screen" />

</launch>
```

Listing 3: aloam_velodyne_VLP_16.launch

```
<launch>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="
    screen" clear_params="true">

    <rosparam file="$(find me5413_world)/my_navigation/params/costmap_common_params.
    yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find me5413_world)/my_navigation/params/costmap_common_params.
    yaml" command="load" ns="local_costmap" />

    <rosparam file="$(find me5413_world)/my_navigation/params/map_nav_params/
    local_costmap_params.yaml" command="load" />
    <rosparam file="$(find me5413_world)/my_navigation/params/map_nav_params/
    global_costmap_params.yaml" command="load" />

    <rosparam file="$(find me5413_world)/my_navigation/params/
    base_local_planner_params.yaml" command="load" />
    <rosparam file="$(find me5413_world)/my_navigation/params/move_base_params.yaml"
    command="load" />

    <param name="base_global_planner" type="string" value="navfn/NavfnROS" />
    <param name="base_local_planner" value="base_local_planner/TrajectoryPlannerROS"/>

    <remap from="odom" to="odometry/filtered" />
  </node>

</launch>
```

Listing 4: move_base.launch

```
global_costmap:
   global_frame: map
   robot_base_frame: base_link
   update_frequency: 20.0
   publish_frequency: 5.0
   width: 40.0
   height: 40.0
   resolution: 0.02
   origin_x: -20.0
   origin_y: -20.0
   static_map: true
   rolling_window: false
```

```
13
14    inflation_radius: 0.5
15    cost_scaling_factor: 10.0
16
17    plugins:
18    - {name: static_layer, type: "costmap_2d::StaticLayer"}
19    - {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
20    - {name: inflater_layer, type: "costmap_2d::InflationLayer"}
```

Listing 5: global_costmap_params.yaml

```
1  local_costmap:
2     global_frame: map
3     robot_base_frame: base_link
4     update_frequency: 20.0
5     publish_frequency: 5.0
6     width: 1.5
7     height: 1.5
8     resolution: 0.02
9     static_map: false
10    rolling_window: true
11
12    inflation_radius: 0.2
13    cost_scaling_factor: 3.0
```

Listing 6: local_costmap_params.yaml

```
1  TrajectoryPlannerROS:
2
3    # Robot Configuration Parameters
4    acc_lim_x: 10.0
5    acc_lim_theta:  5.0
6
7    max_vel_x: 0.5
8    min_vel_x: 0.1
9
10   max_vel_theta: 1.0
11   min_vel_theta: -1.0
12   min_in_place_vel_theta: 0.314
13
14   holonomic_robot: false
15   escape_vel: -0.5
16
17   # Goal Tolerance Parameters
18   yaw_goal_tolerance: 0.157
19   xy_goal_tolerance: 0.25
20   latch_xy_goal_tolerance: false
21
22   # Forward Simulation Parameters
23   sim_time: 3.5
24   sim_granularity: 0.02
25   angular_sim_granularity: 0.02
26   vx_samples: 6
27   vtheta_samples: 20
28   controller_frequency: 20.0
```

Listing 7: base_local_planner_params.yaml

```
1  map_type: costmap
2  origin_z: 0.0
3  z_resolution: 1
4  z_voxels: 2
5
6  obstacle_range: 2.5
7  raytrace_range: 3.0
8
9  publish_voxel_map: false
10 transform_tolerance: 0.5
11 meter_scoring: true
12
13 footprint: [[-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]]
```

```
footprint_padding: 0.1

plugins:
- {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
- {name: inflater_layer, type: "costmap_2d::InflationLayer"}

obstacles_layer:
  observation_sources: scan
  scan: {sensor_frame: front_laser, data_type: LaserScan, topic: front/scan, marking:
    true, clearing: true, min_obstacle_height: -2.0, max_obstacle_height: 2.0,
    obstacle_range: 2.5, raytrace_range: 3.0}

inflater_layer:
 inflation_radius: 0.20
 cost_scaling_factor: 3.0
```

Listing 8: costmap_common_params.yaml