HOMEWORK REPORT

ME5413 AUTONOMOUS MOBILE ROBOTICS

Final GROUP 3

A REPORT SUBMITTED FOR THE ASSIGNMENT OF ME5413
DEPARTMENT OF MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

Supervisor:

Prof. Marcelo H Ang Jr

9/4/2024

1 Introduction

This project aims to apply advanced robotics techniques such as mapping, and planning to enable autonomous navigation in a virtual mini-factory. The goal is to use SLAM and navigation algorithms to guide a robot from Assembly Line 2 to Packaging Area 4, then to Delivery Vehicle 1, dynamically avoiding obstacles along the way.(Figure 1).

Initially, we used gmapping for mapping, but its lack of IMU and odometry data integration led to poor results. Switching to Lego-LOAM, which integrates IMU data, we faced customization challenges that hindered performance. We then adopted FAST-LIO, creating a point cloud map to identify paths and obstacles effectively. Enhancing navigation, we applied a radius filter to reduce noise and improve accuracy, converting the map into a 2D grid for ROS navigation. The mapping algorithm's precision was verified with EVO by measuring the Absolute Pose Error (APE) against a predefined path.

Task 2 focuses on navigating the Jackal robot via an interactive GUI in Rviz. We start by establishing basic navigation capabilities, where the robot autonomously plans its route with a global planner and avoids obstacles using a local planner based on received coordinates. We then enhance navigation by marking certain zones as inaccessible and adding visual recognition to improve the robot's autonomous movement.

2 Tasks 1: Mapping

Task 1 evaluated different mapping algorithms in a simulated factory environment, starting with gmapping and progressing through LeGO-LOAM to FAST-LIO due to challenges like sensor data integration and calibration issues. FAST-LIO emerged as the most effective, combining efficient computation and data fusion. Accuracy assessments were conducted using the EVO tool, showcasing each algorithm's performance in the complex setup.

The simulation, set in a Gazebo-based mini-factory, featured a car park with vehicles and clutter, a workshop with assembly lines, and an area with randomly placed boxes, designated into target and restricted zones for the experiment (Figure 1).

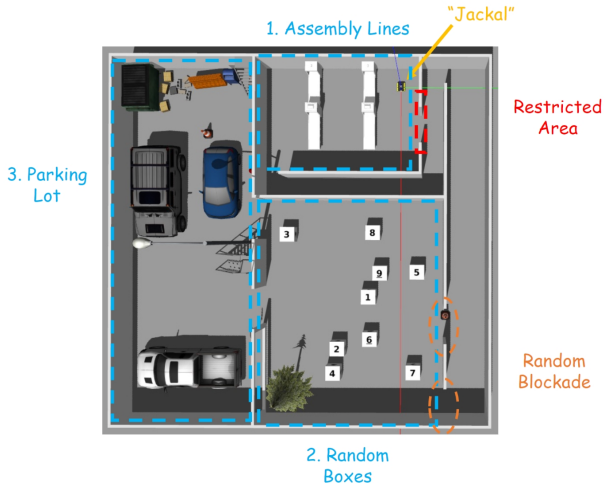


Figure 1: Gazebo Simulation World for Mapping

2.1 Analysis of Initial Mapping Attempts

Initial mapping trials were conducted using the standard template code provided for the task. This code utilizes the gmapping algorithm. This algorithm is an established method for creating 2D maps from LiDAR inputs. It became evident that the algorithm's default configuration did not integrate data from the IMU and wheel odometry during the first three tests. This process is essential for refining the pose estimation. The deficiencies in the map quality are evident in Figure 2, which delineates two significant shortcomings in the mapping process:

1. **Uniform Corridor Challenge:** The LiDAR sensor consistently perceived identical scenes within the corridor sections, resulting in notable mapping inaccuracies. This challenge is attributed to the

algorithm's dependence on environmental features to discern variations in space, which are inherently scarce in uniformly structured zones such as corridors. These discrepancies are graphically represented in Figure 2a.

2. **Dimensional Limitation of Sensor Data:** The gmapping's reliance on 2D LiDAR data inherently constrains the scope of its environmental perception to a single plane. Consequently, this limitation is manifested as an absence of multi-level structural details in the resulting maps, rendering any element outside the horizontal scanning range of the LiDAR undetectable. This limitation is particularly potent for the 3-D environment, leading to maps that do not fully capture the spatial dynamics, as observed in Figure 2b.

(a) gmapping Result Directly

(b) gmapping by Adding Artificial Landmarks

Figure 2: gmapping Results

2.2 LeGO-LOAM Mapping

LeGO-LOAM [1] is an advanced method for real-time 3D SLAM that uses LiDAR data. It is particularly designed for ground vehicles, optimizing for both efficiency and performance in such environments. The core innovation of LeGO-LOAM lies in its ability to perform robust odometry and mapping by exploiting the structure of ground environments. It separates ground from non-ground points to optimize the processing flow, reducing computational load while maintaining high accuracy in pose estimation and map construction. Additionally, LEGO-LOAM algorithm provides optional Imu data input to correct or enhance the point cloud data, which may seem to be a solution to the corridor problem.

2.2.1 LeGO-LOAM Setup

To set up LeGO-LOAM on ROS Melodic, several modifications are required after cloning the repository from GitHub. The installation of the **'gtsam'** library is the first step, laying the groundwork for further optimization algorithms. This is followed by adjustments to the C++ standard and OpenCV version dependencies in the **'utility.h'** and **'CMakeLists.txt'** files, respectively. This was vital for the successful compilation of the workspace using the `catkin_make` command.

Following the initial setup, the LiDAR topic was configured to **"/mid.points"** and the IMU topic to **"/imu/data"**. Prior to activating LeGO-LOAM, it was necessary to preprocess IMU data to ensure its compatibility. A crucial step in this preprocessing phase was the synchronization of timestamps between IMU and LiDAR data, as discrepancies in timing could significantly affect the accuracy of data integration. Additionally, a calibrated external reference rotation matrix was incorporated within the `nh("~")` function. An external parameter conversion function, `imuConverter(const sensor_msgs::Imu& imu_in)`, was developed and subsequently invoked within the `imuHandler(const sensor_msgs::Imu::ConstPtr& imuIn)` callback function, facilitating the adaptation of LeGO-LOAM to process IMU data effectively.

(a) LeGO-LOAM Running Screenshot

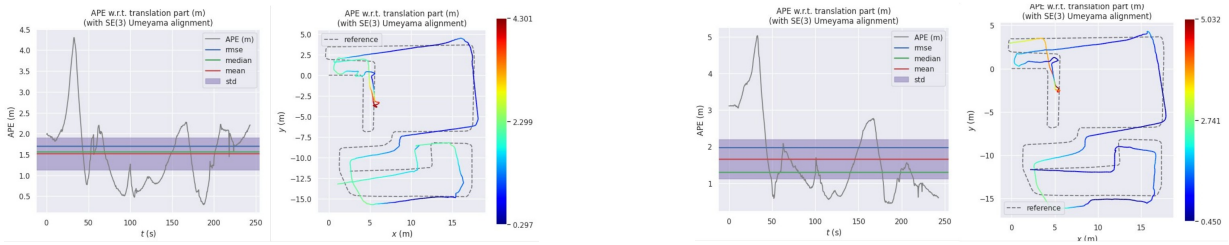
(b) Point Cloud Result by LeGO-LOAM

Figure 3: LeGO-LOAM Results

The mapping process was initiated by launching the **"me5413.world"** simulation environment along with the **"lego_loam"** node. To facilitate the control of the robot within this environment, the **"teleop_twist_keyboard"** node was executed. The outcomes of the LeGO-LOAM mapping are illustrated in Figure 3, showcasing the effectiveness of the setup in generating detailed environmental maps.

2.2.2 Problem Analysis

The ground truth odometry and the actual odometry data generated by LeGO-LOAM were documented and analyzed using the EVO tool. This analysis facilitated a direct comparison between the results obtained when IMU data were integrated and when only LiDAR point cloud data were used. The comparison is depicted in Figure 4, with Figure 4a illustrating the Absolute Pose Error (APE) when mapping with IMU data, and Figure 4b showing the APE for mappings conducted without utilizing IMU data.



(a) APE of using imu data

(b) APE of not using imu data

Figure 4: EVO of LeGO-LOAM Results

From the analysis of both the point cloud map and Absolute Pose Error (APE), it is evident that the mapping results obtained from LeGO-LOAM are significantly unsatisfactory. Notably, the robot lost its pose upon entering the first corridor, as illustrated in Figure 3a. The generated map diverges considerably from the ground truth, highlighting this discrepancy. Although the performance appears to improve after exiting the corridors, the initial misalignment casts a shadow over the overall mapping accuracy. However, when the APE was plotted with alignment adjustments, the error rates in the latter half of the route were found to be commendably lower, suggesting some degree of recovery in mapping accuracy post-corridor navigation (Figure 4a, Figure 4b)

The observed mapping inaccuracies can be attributed to two primary factors:

1. **Sensor Calibration:** A lack of proper calibration between the IMU and the LiDAR sensor contributes to inaccuracies in the fused data. This issue stems from incorrect extrinsic calibration, specifically the erroneous calculation and definition of the parameters for the extrinsic rotation matrix, which affects the relative pose between the IMU and LiDAR.

2. **Data Synchronization:** Precise temporal synchronization between IMU and LiDAR data is crucial. Discrepancies in timestamps or sensor latency result in inaccuracies within the fused data.

3. **Data Fusion Algorithm:** The chosen method for fusing IMU and LiDAR data may not be optimal, potentially due to an incorrect implementation of the fusion algorithm or the algorithm's inherent lack of robustness.

These elements critically influence the system's ability to accurately map the environment. It also highlights the need for refined calibration, synchronization, and data fusion methodologies.

Given the challenges with calibration and the difficulty in adjusting the current system, it became clear that a simpler and more effective solution was needed. The decision to switch to Fast-LIO, an algorithm known for its user-friendliness and advanced capabilities, was made to simplify development and improve mapping accuracy. This move is expected to enhance the mapping process, making it more straightforward and efficient.

2.3 FAST-LIO Mapping

FAST-LIO [2] is a LiDAR-inertial odometry package designed to enhance the navigation capabilities of small-scale mobile robots like UAVs, particularly in challenging environments. It utilizes a tightly-coupled iterated Kalman filter to merge LiDAR feature points with IMU measurements effectively, addressing motion distortion and reducing computational load through a novel formula for computing the Kalman gain.

A key innovation of FAST-LIO is its efficient computation approach, which relies on the state dimension rather than the measurement dimension for calculating the Kalman gain. This significantly reduces computational demands, enabling FAST-LIO to perform real-time processing even on systems with limited computing power, such as UAV onboard computers. The framework has demonstrated its robustness and efficiency across various testing environments, making it a valuable tool for applications requiring precise and reliable navigation under challenging conditions.

2.3.1 FAST-LIO Setup

Since the FAST-LIO must support Livox serials LiDAR firstly, so the livox_ros_driver must be installed and sourced before run any FAST-LIO luanch file. After cloning the github repo, we need to install the livox-SDK and the livox-ros driver first and then install the driver before building the FAST-LIO project.

When building the FAST-LIO project, we will face a "ikd-tree" error. Ikd-Tree is an incremental k-d tree designed for robotic applications. The ikd-Tree incrementally updates a k-d tree with new coming points only, leading to much lower computation time than existing static k-d trees. Besides point-wise operations, the ikd-Tree supports several features such as box-wise operations and down-sampling that are practically useful in robotic applications [3].

Once the project is ready to run, we need to modify FAST-LIO's configuration file so that it can receive messages generated in our virtual Gazebo environment. We create a "me5413_FL.yaml" file to configure FAST-LIO. We set the lidar topic to "/mid/points" and the imu topic to "/imu/data" and increase the scan rate to 40Hz to make the point cloud more dense.

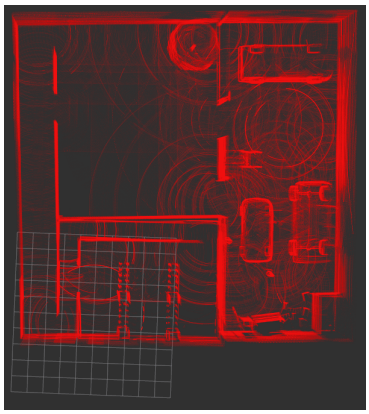


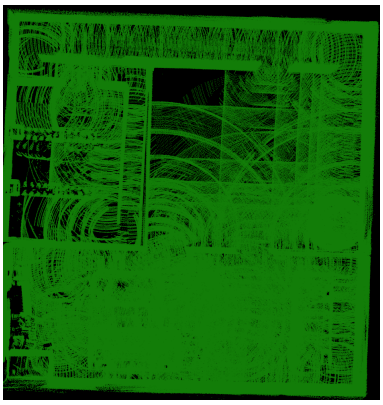
Figure 5: Point Cloud Map in Rviz

Finally, we modified the subscription theme of rviz and rewrote a launch file. In rviz we subscribed the **"/cloud_registered"** topic to show the point cloud map from FAST-LIO and to make the point

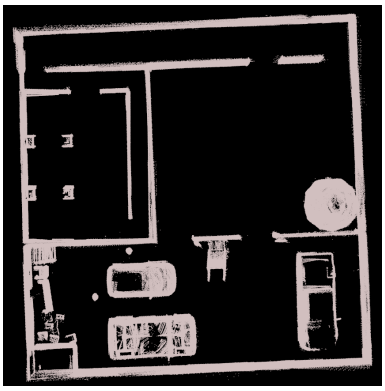
cloud shown in rviz more dense and stable we changed the Decay Time and Queue Size parameters in a very large number. The launch file is basically **"mapping_velodyne.launch"** in FAST-LIO and "manual.launch" in **"me5413_world"** together and add **"rqt_robot_steering"** node to control the robot. The point cloud in rviz was shown in Fig 5.

2.3.2 Point Cloud Image Filtering

The raw point cloud generated by FAST-LIO is shown in Fig.6a. But obviously it contain a significant amount of noise due to various factors such as the complexity of the environment, sensor inaccuracies and the inherent limitations of high speed motion processing. Among the different types of noise, radiated noise is the most obvious. Radiation noise patterns in point clouds generated by FAST-LIO are typically caused by range measurement errors and scanning mechanisms of LiDAR sensors. As the sensor scans the environment, errors in measuring the distance to an object or surface can cause points to falsely appear to emanate or radiate from a location, creating noise that distorts the true shape and structure of the scanned environment.



(a) Point Cloud Image Before Filtering



(b) Point Cloud Image After Radius Filtering

Figure 6: Point Cloud Image

We use the radius filter for noise reduction, the image after filtering is shown in Fig. 6b. The radius filter aims to remove noise by analyzing the density of points within a specified radius around each point. The underlying idea is to identify and discard points that do not have enough neighboring points within a certain distance, under the assumption that such isolated points are likely to be noise rather than meaningful parts of the surface.

The mathematical approach to applying a radius filter can be described as follows:
1. For each point P_i in the point cloud, calculate the distance d to every other point P_j in the cloud.
2. Count the number of points N within a predefined radius r from point P_i.
3. Compare N to a threshold T. If $N < T$, the point P_i is considered noise and is removed from the dataset.

2.3.3 Convert Point Clouds to 2D Maps

Since grid map is required for ros navigation, the filtered map needs to be converted to grid map (shown in Fig.7). We use an algorithm to convert our filtered map to 2D grid map. It first determining the minimum and maximum coordinates (x_min, x_max, y_min, y_max) to define the grid's extent. It then sets the origin of the grid map at the minimum x and y values and calculates the width and height of the map based on these extents and a predefined resolution (map_resolution). Each point in the point cloud is then iterated over, and its position is used to update the corresponding cell in the grid map with an occupancy value.

2.3.4 EVO Evaluation

We used evo to plot FAST-LIO-produced odometers against ground truth (Fig.8a, and calculated Absolute Pose Error (APE) between theoretical and actual values to measure FAST-LIO's accuracy (Fig.8b). The Fast LIO's navigation performance is commendable, with its ability to track a path displaying reasonable accuracy as evidenced by the estimated trajectory's proximity to the reference in

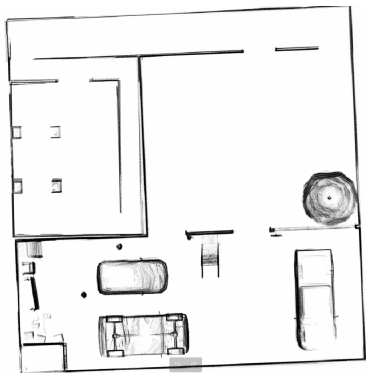
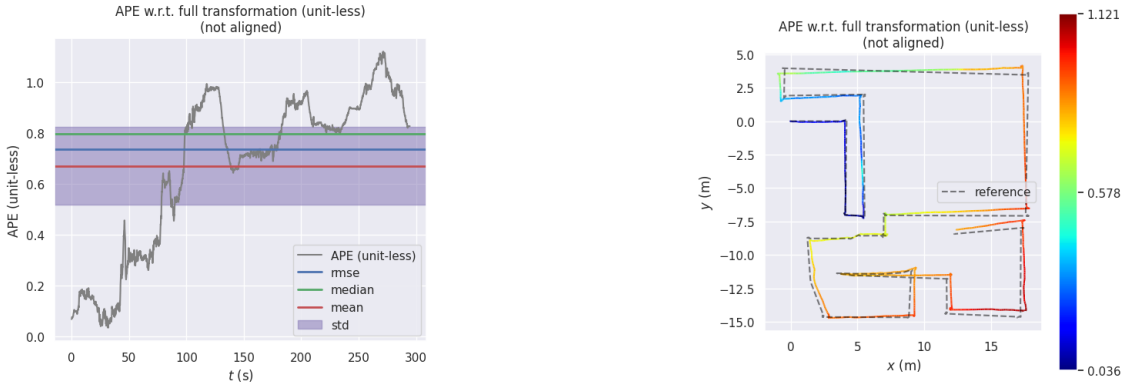


Figure 7: 2D Grid Map

the trajectory plot. Despite this, the presence of deviations and fluctuations in the APE over time, as shown in the temporal stability evaluation, indicates that the system's performance is subject to variations possibly due to dynamic environmental factors, sensor noise, or inherent limitations in the algorithm's mapping and optimization processes. Nonetheless, the statistical measures of the APE—root mean square error, median, mean, and standard deviation—remain within a moderate range, suggesting that the system's performance, while exhibiting occasional outliers, is generally stable across the dataset.



(a) Absolute Pose Error of Fast-LIO

(b) Fast-LIO's Estimated vs. Reference Path

Figure 8: EVO Evaluation

2.4 Challenges and Improvements

In Task 1, mapping a simulated factory environment revealed several challenges with different SLAM algorithms, from gmapping to LeGO-LOAM, and finally to FAST-LIO. Initial efforts with gmapping struggled with sensor data integration and accuracy, particularly in environments with repetitive structures like corridors. Transitioning to LeGO-LOAM introduced issues with sensor calibration and data synchronization, underscoring the need for precise alignment between IMU and LiDAR sensors and coherent data integration.

The switch to FAST-LIO represented a significant improvement, offering better computational efficiency, real-time processing, and enhanced noise reduction. FAST-LIO's ability to tightly integrate LiDAR and IMU data, along with its efficient computational framework, addressed many of the initial challenges. However, challenges such as noise in the point cloud data and the need for accurate 2D map conversions persisted. Solutions included advanced filtering techniques to improve data quality and algorithms for efficiently transforming point clouds into 2D grid maps for navigation. This journey from gmapping to FAST-LIO not only highlighted the complexities of robotic mapping in diverse environments but also showcased the potential of advanced SLAM algorithms and data processing techniques to overcome these challenges, paving the way for more accurate and reliable robotic navigation systems.

3 Tasks 2: Navigation

In this section, the ultimate goal is to navigate the Jackal using the provided GUI within Rviz. Our task is primarily divided into several key parts. Initially, the task is to implement fundamental navigation capabilities. Specifically, once the controller issues a set of coordinates, the Jackal should be able to utilize the global planner to map out a route and concurrently work with the local planner to avoid obstacles. Building upon this foundation, we aim to introduce additional functionalities. These include designating certain areas as no-go zones, and ensuring that the Jackal steers clear of these regions during its movement. Furthermore, we plan to incorporate visual recognition techniques into the navigation process.

3.1 Fundamental Navigation

In this part, we will achieve navigation through the integration of a global planner and a local planner. The global planner is responsible for mapping out the complete path. However, it may overlook the collision volumes at specific locations. This is where the local planner comes into play. It is designed to prevent the Jackal from colliding with obstacles during its journey.

3.1.1 Global Planner

After analyzing the requirements of the task, we have decided to utilize the Dijkstra algorithm as our global path planning algorithm. First and foremost, the Dijkstra algorithm is a complete algorithm, which ensures certainty in finding the shortest path from a single source. Secondly, due to the absence of heuristic approaches, it offers better stability and robustness. Furthermore, for our task, which does not necessitate the estimation of the distance to the destination, and given that all edge weights are the same or have minimal variation, heuristic algorithms are not essential. Therefore, we have opted for the Dijkstra algorithm.

To apply the Dijkstra algorithm, we first add global_planner_params.yaml file in params folder and set lethal_cost = 253, neutral_cost = 50 and cost_factor = 3.0. Then we set use_dijkstra to true which means to use Dijkstra algorithm.

As shown in Fig. 9, an overview of the planned route in a mapped environment is drawn, where the green line represents the path charted by the global planner. The outlined trajectory ensures navigational directives for optimal movement within the space, avoiding known obstacles and efficiently directing the route from the starting point to the intended destination.

Figure 9: Overview of Planned Route(Global Planner)

3.1.2 Local Planner

As for the local planner, the default one is base_local_planner. During experiments, it has been determined that the performance of the local path planning algorithm is unsatisfactory. The generated paths are notably brief, resulting in diminished vehicular speed and frequent deviations from the intended

trajectory. And we found another local planner 'teb_local_planner'. The forward-looking algorithm of teb_local_planner is relatively long so when encountering obstacles, the local path will automatically plan to avoid obstacles. However, it also has its shortcomings; in more complex environments, the local path is more prone to confusion.

The Fig. 10 illustrates the trajectory generated by the local planner, indicated by the blue line. It is designed to guide the Jackal along the path planned by the global planner while simultaneously avoiding potential obstacles detected in the immediate vicinity.

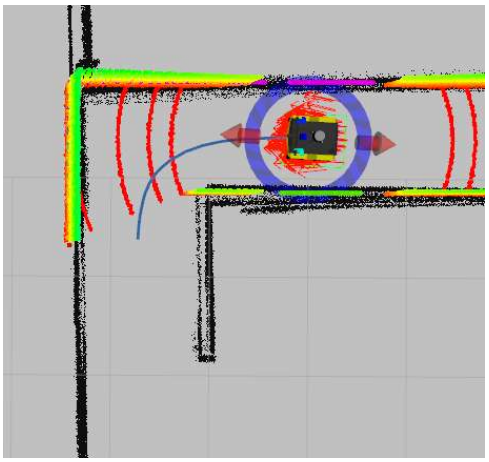


Figure 10: Overview of Planned Route(Local Planner)

3.2 Restricted Area

After completing the fundamental tasks, we turned our attention to more detailed issues, starting with the restricted area. According to the task requirements, the Jackal is prohibited from passing through the Restricted Area depicted in Fig. 1. Due to the constraints of the controller, which we were unable to adjust, we ultimately resolved to utilize the expansion pack which is called costmap_prohibition_layer, then we add the .yaml file which defines the prohibited area, then add this node to move_base and global_costmap parameter file. Results as shown below:

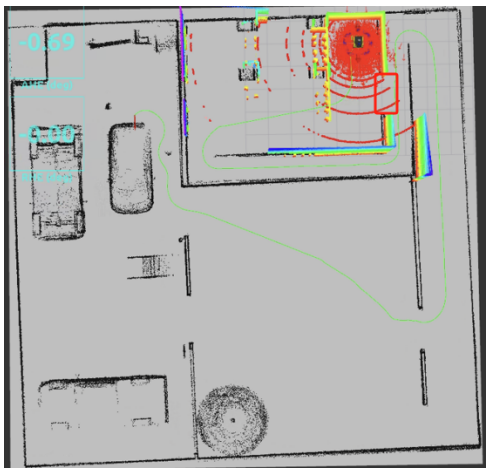


Figure 11: Overview of Planned Route(Restricted Area)

From Fig. 11, it is observable that despite the more proximate route being a direct left turn, the Jackal has opted to pass from the right side. This indicates the efficacy of our configuration; the Jackal has successfully avoided traversing through the restricted area.

3.3 Navigate to Boxes

This part of the project posed the greatest challenge, as it did not involve simple positioning through direct pixel coordinates. Instead, it required the use of visual recognition to identify the location of boxes and subsequently navigate the Jackal to these locations. Another difficulty lay in determining the best approach to guide the Jackal to the boxes. To address this, we explored two solutions: one involved the integration of stereo vision cameras with visual recognition, and the other combined LIDAR data with visual recognition.

Initially, when the objective is to navigate the Jackal to a designated box, commands are issued to guide the Jackal toward the upper right corner of the Random Boxes area at coordinates (8,0). Subsequently, the vehicle is directed to circumnavigate this region. During the navigation process, we employ the template matching algorithm from the OpenCV library, as used in Homework1_Perception. This algorithm compares the live camera feed with a preselected set of numerical images. Upon recognizing the number corresponding to the box number issued by the GUI, the Jackal is then steered to face the box directly. The next task is to guide the Jackal to the side of the box, for which we attempted the following two approaches:

3.3.1 Stereo Camera with Visual Recognition

Given that the Jackal is now positioned directly facing the target box, we endeavored to determine the distance between them. The initial thought was to employ a depth camera or stereo vision to acquire depth information. Several cameras had already been configured in the .urdf configuration file, so we decided to use the first stereo camera by enabling the JACKAL_stereo_FLEA3, which entailed setting the: if value="$(optenv JACKAL_STEREO_FLEA3 0)" to 1. Subsequently, to calculate the depth, we needed to use the following formula to transform the pixel coordinates into three-dimensional coordinates in the camera coordinate system.

$$X = \frac{(u - c_x) \times Z}{f_x} \tag{1}$$

$$Y = \frac{(v - c_y) \times Z}{f_y} \tag{2}$$

$$Z = Z \tag{3}$$

In this model, X, Y, and Z represent the spatial coordinates in the camera's coordinate system. The u and v are the pixel coordinates in the image plane, f_x and f_y represent the camera's focal length (in terms of pixels), and c_x, c_y are the optical center (principal point) of the camera's lens. However, according to the configuration file, we cannot directly obtain the camera's focal length. Yet, we noticed that the configuration file includes the camera's offset distance(0.16) the horizontal field of view(1.047), and the horizontal resolution(640). Thus, utilizing Equation (4), we can derive the focal length

$$f = \frac{w}{2\tan\left(\frac{\mathrm{HFOV}}{2}\right)} \tag{4}$$

where: f is the camera's focal length, usually expressed in pixels. w is the image's horizontal resolution, i.e., the width of the image in pixels. $HFOV$ is the camera's horizontal field of view, usually expressed in radians. tan represents the tangent function.

We also obtain the position and attitude of the camera with respect to the cart, these parameters are called the camera's external parameters, through which the captured target can be converted from the camera coordinate system to the cart coordinate system, and the formula for the coordinate system conversion is as follows:

$$P_{car} = R \cdot P_{camera} + t \tag{5}$$

However, the result seems limited due to the error caused by the camera itself. So we used another Lidar with Visual Recognition method.

3.3.2 Lidar with Visual Recognition

Regarding the LiDAR, since the Jackal is already facing the box, we directly obtain the radar data from directly ahead as the distance the Jackal moves forward. After reaching the target, this approach yields results that are much more accurate compared to those obtained with the stereo camera as shown in the video.

3.4 Challenges and Solutions

3.4.1 Point Cloud Offset

Problem: We have found that the point cloud tends to drift significantly even when the robot is stationary. Although it will re-align the point cloud positions when the robot starts moving, this can still lead to issues such as being unable to plan routes due to endpoint deviations in the area of obstacles.

Solution: In tackling this challenge, our first step was to adjust the global cost map by fine-tuning parameters such as inflation radius and cost scaling factor. We then turned our attention to configuring the costmap_converter. After this adaptation and the results were remarkably positive, resulting in a navigation system that was more efficient

3.4.2 Deadlock

Problem: When we navigate the robot at the start, we find that the robot does not follows the global planning path, and it will hit the wall when facing the 180-degree corner.

Solution: The basic reason is due to the parameter of the local planner algorithm, so we change the default algorithm and tune the parameter of the new algorithm, another important element is the width and height of the local cost-map, finally, we determine using width and height equal 5. Meanwhile, we assign a high value to the inflation parameters of the global cost-map, while setting a lower value for the inflation parameters of the local cost-map.

3.5 Overall

For the navigation performance test, we score our algorithm from the following aspects: error of the final pose and the goal pose, navigation with randomly selected start and end pose. The given sequence of goal poses of our group is "Assembly Line 2", "Box 4" and "Vehicle 1", the following table show the error between the final pose and the goal pose and the result is also shown in the appendix.

Table 1: Position Error and Heading Error of three poses

Pose	Assembly Line 2	Box 4	Vehicle 1
ADE (m)	0.44	1.13	0.44
AHE (deg)	-2.77	0	2.88
RDE (m)	6.17	6.16	6.01
RHE (deg)	89.95	0	95.54

As the above table shows, the error is acceptable, some reasons like the global map not matching the true map excellently will result in the error. And the AHE and RHE of Box 4 is 0, it is because that That's because we corrected the angle in advance. Then we can select a better mapping algorithm or tune the local planner algorithm's parameters, but note that the local planner algorithm might calculate many local paths when the robot is close to the goal position and the robot will oscillating near the target point if the value of these parameters is too small.

Overall, our implementation is successful, not only in navigating the jackal to a given location and combining it with visual recognition for location search and navigation, but also in optimising the original implementation to minimise deadlocks and wall crashes.

3.6 Github Link

Github Repo

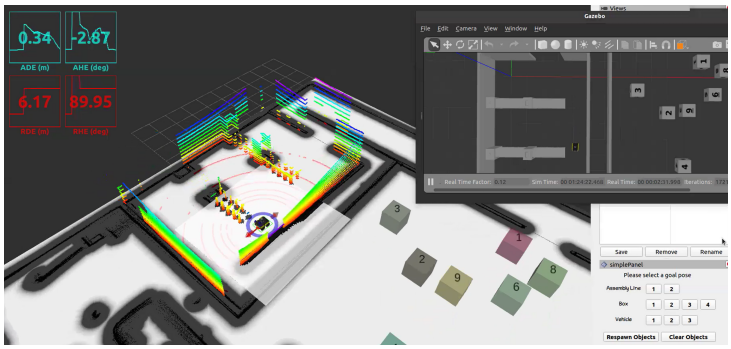
References

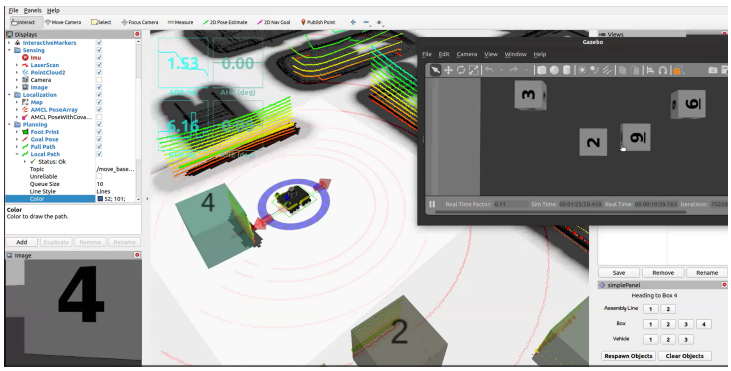
[1] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. pages 4758–4765, 2018.

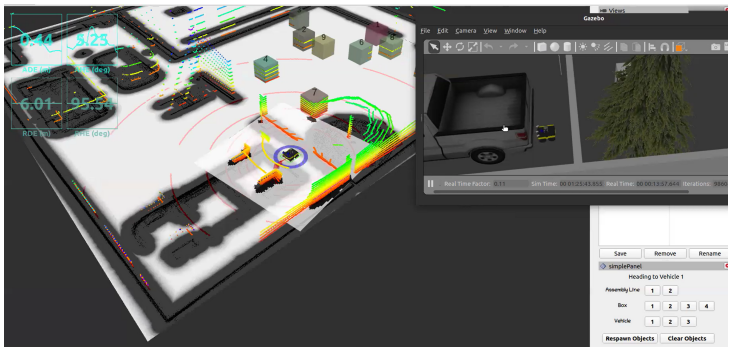
[2] Wei Xu and Fu Zhang. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter. *IEEE Robotics and Automation Letters*, 6(2):3317–3324, 2021.

[3] Yixi Cai, Wei Xu, and Fu Zhang. ikd-tree: An incremental kd tree for robotic applications. *arXiv preprint arXiv:2102.10808*, 2021.

APPENDICES


0.34
-2.87
ADE (m)
AHE (deg)
6.17
89.95
RDE (m)
RHE (deg)
Gazebo
Save
Remove
Rename
simplePanel
Please select a goal pose
Assembly Line
Box
Vehicle
Respawn Objects
Clear Objects


1.53
0.00
6.16
4
2
Gazebo
Save
Remove
Rename
simplePanel
Heading to Box 4
Respawn Objects
Clear Objects


0.40
8.23
6.01
Gazebo
Save
Remove
Rename
simplePanel
Heading to Vehicle 1
Respawn Objects
Clear Objects