Our GitHub repo link: https://github.com/Wang-Theo/ME5413\_Final\_Project

# Project Description

This is the final project of ME5413 Autonomous Mobile Robotics AY22-23 Semester 2.

We are given a mini-factory environment which is shown in Figure 1 in Gazebo. This environment has 3 accessible areas and 1 inaccessible area.

|  |
| --- |
|  |
| Figure 1 Mini-factory environment |

The aim of the project is to design a robot navigation software stack that can: From the starting point, move to the given pose within each area in sequence:

* Assembly Line 1, 2
* Packaging Area 1, 2, 3, 4
* Delivery Vehicle 1, 2, 3

Thus, the tasks consist of mapping and navigation.

# Task 1 Mapping

In this task, we need to choose an algorithm to map the environment and then also evaluate its accuracy.

Here as we can see from Figure 1, there are some complex and special objects like glass wall, stair, debris and tree. For example, the glass cannot be scanned by lidar except the outline border, the stair is a three-dimensional structure and debris are piled up in different sizes etc.

We realize that 2D SLAM algorithm is not capable to map the features of these special objects. And also considering that visual SLAM obtains too much information, thus the amount of calculations and loads are large. As the environment here is not too complicated and only simple navigation task is required, the use of lidar is the most suitable.

Thus, we decide to choose FAST-LIO to do the mapping which use the fusion of 3D LiDAR and IMU.

## Map the Environment with FAST-LIO

FAST-LIO is a computationally efficient and robust Lidar-Imu Odometry method proposed by the MARS Laboratory of the University of Hong Kong. It utilizes tightly coupled, iteratively extended Kalman filters to fuse lidar landmarks with IMU data for robust navigation in fast-motion, noisy or clutterous environments.

In this task, we refer the repo [1] of the FAST-LIO original authors. Our operation process to build the repo can be concluded as:

* Git clone the *“ME5413\_Final\_Project”* repo given and follow the readme to build it.
* Git clone and build FAST-LIO in our own workspace.
* Move FAST-LIO into *“ME5413\_Final\_Project/src”* folder as a package in the repo whose name is *“FAST\_LIO\_”*.
* Use command *“roslaunch me5413\_world world.launch”* to launch the world and use *“rostopic list”* to find the two topics we need: pointcloud topic *“/mid/points”* for 3D LiDAR and sensor\_msgs/Imu topic *“/imu/data”* for IMU.
* Set the LiDAR pointcloud topic name and IMU topic name in *“velodyne.yaml”* in config folder of *“FAST\_LIO\_”* which is shown in *velodyne.yaml* in Appendix.
* Create a new launch file *“fast\_lio.launch”* in the launch folder of *“me5413\_world”* package which is shown in *fast\_lio.launch* in Appendix
* Remake the repo using command *“catkin\_make”*.

Thus, now we can successfully do the mapping with command *“roslaunch me5413\_world world.launch”* in first terminal and command *“roslaunch me5413\_world fast\_lio.launch”* in second terminal.

However, we found that there are two problems occurring.

First one is we need to cut off the ground pointclouds which is shown in Figure 2.

|  |
| --- |
|  |
| Figure 2 Problem: not cut off ground pointclouds |

Second one is when we turn the Jackal, the pointclouds sometimes fail to coincide with original one, like there will be two sets of pointclouds for one wall in different direction which is shown in Figure 3.

|  |
| --- |
|  |
| Figure 3 Problem: fail to coincide with original pointclouds |

To solve the first problem, we add a limit height in axis for pointclouds in the callback function in *“LaserMapping.cpp”* in *“FAST\_LIO\_”* which is shown in *pointcloud callback function in LaserMapping.cpp* in Appendix. This could allow to not publish the ground pointclouds in RViz. And we also filter the ground pointclouds in the result *scans.pcd* file which will be introduced later.

To solve the second problem, we try couple of methods and finally we find that if we increase the 3D LiDAR’s rate from 10Hz to 50 Hz in *“velodyne.yaml”*, the problem will be solved.

Then now if we follow again the process steps introduced, the pointcloud *scans.pcd* file will be saved. But this pointcloud file still have some noise points and ground points need to be filtered. We also need to convert the *pcd* file to a planar grid map which will help do the navigation task.

Thus, we write another package *“pcdtomap”* into *“ME5413\_Final\_Project/src”.* The remaining process are:

* Create a package named *“pcdtomap”* and write the algorithm with two filter: pass-through filter and radius filter in *“pcdtomap.cpp”*.

Notice that pass-through filter here is for filtering the ground pointclouds in axis and pointclouds too far away in and axis, radius filter is for filtering the noise points.

* Build package structure and write makefile.
* Add a launch file *“pcdtomap.launch”* to set the parameters for filters and launch to convert pcd to grid map.

Finally, the whole process is done. We can do the mapping task with command:

* *“roslaunch me5413\_world world.launch”* in first terminal.
* *“roslaunch me5413\_world fast\_lio.launch”* in second terminal.
* *“roslaunch pcdtomap pcdtomap.launch”* in third terminal.
* *“rosrun map\_server map\_saver”* in the forth terminal while in map folder.

The RViz, pcd file and grid map result of mapping is shown in Figure 4.

|  |
| --- |
|  |
| Figure 4 Mapping results |

## Evaluate FAST-LIO Algorithm Performance

To quantitatively evaluate the SLAM algorithm performance, here we use the method of EVO.

Here we verify the accuracy by comparing my own odometry */Odometry* topic with the published */gazebo/ground\_truth/state* topic (nav\_msgs::Odometry). The process is as follows:

* While doing mapping, open one more terminal using command *“rosbag record /gazebo/ground\_truth/state /Odometry -o EVO\_perform.bag”* to record a rosbag.
* Use EVO command *“evo\_ape bag EVO\_perform.bag /gazebo/ground\_truth/state /Odometry -r full -va --plot --plot\_mode xy”* to evaluate and show the accuracy result.

The evaluation performance result is shown in Figure 5 and Figure 6.

|  |
| --- |
|  |
| Figure 5 FAST-LIO accuracy performance using EVO (figure and table) |

As we can see from the Figure 5, the mean alignment is and the max alignment is . The results show that the accuracy of algorithm performs very good.

# Reference

[1] Cris.Wei (2021) FAST-LIO [Source Code] https://github.com/hku-mars/FAST\_LIO

# Appendix

*fast\_lio.launch*

文本

描述已自动生成

*velodyne.yaml*

文本

描述已自动生成

pointcloud callback function in *LaserMapping.cpp*

文本

描述已自动生成

*pcdtomap.launch*

文本

描述已自动生成