National University of Singapore

College of Design and Engineering

ME5413 Autonomous Mobile Robotics: Final Project

Group 9:
Chen Yihui A0263115N
Wang Renjie A0263387U
Shen Yixiu A0263954U
Wang Longfei A

Supervisor:
Prof. Marcelo H Ang Jr

Github repo link: https://github.com/Wang-Theo/ME5413_Final_Project

Email Address: e1010473@u.nus.edu
e1010745@u.nus.edu
e@u.nus.edu
e@u.nus.edu

March 25, 2023

1 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 Fig. 1 in Gazebo. This environment has 3 accessible areas and 1 inaccessible area.

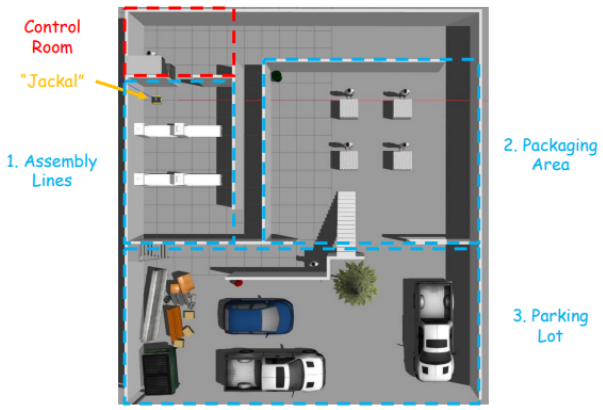


Figure 1: Mini-factory environment

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.

2 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 Fig. 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.

2.1 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 Fig. 2.

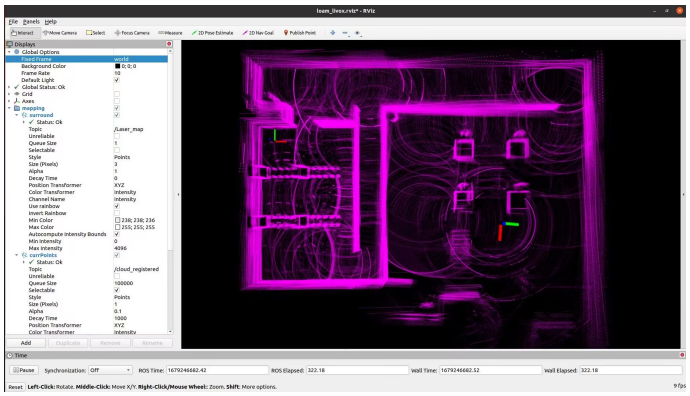


Figure 2: Problem 1: not cut off ground pointclouds

The 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 Fig. 3.

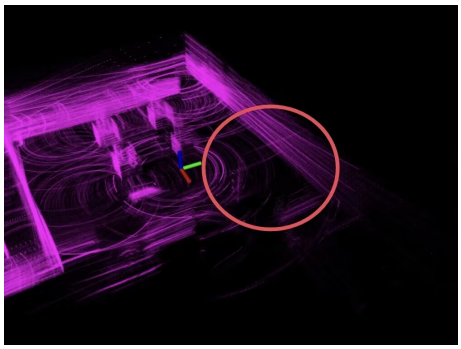


Figure 3: Problem 2: fail to coincide with original pointclouds

To solve the first problem, we add a limit height in z 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"*.

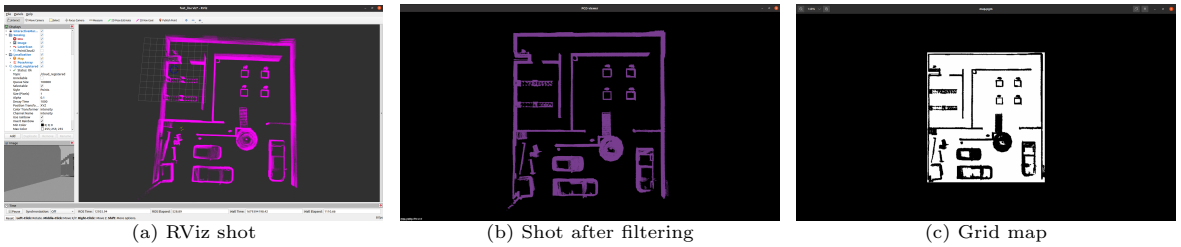
Notice that pass-through filter here is for filtering the ground pointclouds in z axis and pointclouds too far away in x and y axis, radius filter is for filtering the noise points.

- 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 Fig. 4.



(a) RViz shot (b) Shot after filtering (c) Grid map

Figure 4: Mapping results

2.2 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 Fig. 5.

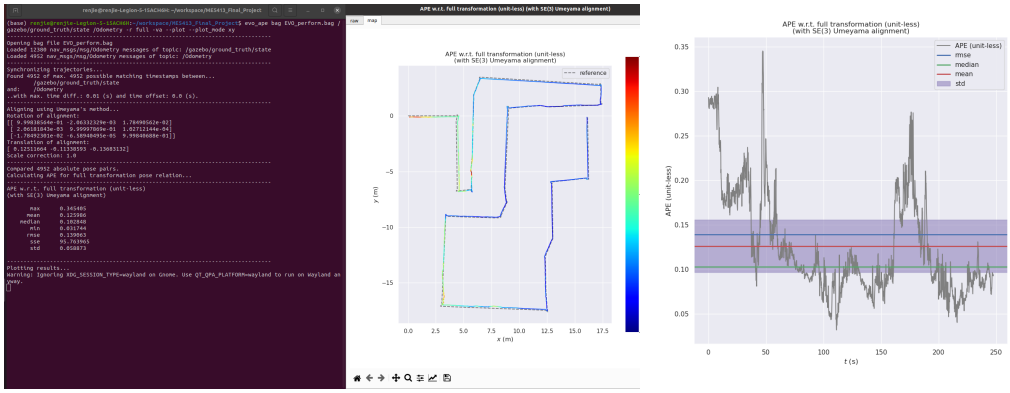


Figure 5: FAST-LIO accuracy performance using EVO (figure and table)

As we can see from the Fig. 5, the mean alignment is 0.125986 and the max alignment is 0.345405. The results show that the accuracy of algorithm performs very good.

2.3 Discuss challenges faced and solutions

The problems that we met during doing mapping have been introduced above, which can be concluded as:

1. Need to cut off ground cloudpoints.
2. Fail to coincide with original cloudpoints when turning Jackal.
3. Need to filter noise cloudpoints.
4. Need to convert pcd file to grid map for navigation.

So, the solutions we used are:

1. Add height limit for cloudpoints in callback function and add pass-through filter to filter ground points of pcd file.
2. Increase LiDAR rate from 10 Hz to 50 Hz.
3. Add radius filter to remove noise points.
4. Write algorithm package by ourselves to project pointclouds to 2D grid map.

The details could be found above in first part of Mapping Task.

References

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

A Appendix

fast_lio.launch

```
<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"/>

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

  <!-- Launch file for velodyne16 VLP-16 LiDAR -->
  <rosparam command="load" file="$(find fast_lio)/config/velodyne.yaml" />

  <param name="feature_extract_enable" type="bool" value="0"/>
  <param name="point_filter_num" type="int" value="4"/>
  <param name="max_iteration" type="int" value="3" />
  <param name="filter_size_surf" type="double" value="0.5" />
  <param name="filter_size_map" type="double" value="0.5" />
  <param name="cube_side_length" type="double" value="1000" />
  <param name="runtime_pos_log_enable" type="bool" value="0" />
  <node pkg="fast_lio" type="fastlio_mapping" name="laserMapping" output="screen" />

</launch>
```

velodyne.yawl

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

    time_sync_en: false         # ONLY turn on when external time synchronization is really not possible
    time_offset_lidar_to_imu: 0.0 # Time offset between lidar and IMU calibrated by other algorithms, e.g. LI-Init (can be found in README).
                                  # This param will take effect no matter what time_sync_en is. So if the time offset is not known exactly, please set as 0.0

preprocess:
    lidar_type: 2               # 1 for Livox serials LiDAR, 2 for Velodyne LiDAR, 3 for ouster LiDAR,
    scan_line: 64
    scan_rate: 50               # only need to be set for velodyne, unit: Hz,
    timestamp_unit: 2           # the unit of time/t field in the PointCloud2 rostopic: 0-second, 1-milisecond, 2-microsecond, 3-nanosecond.
    blind: 2

mapping:
    acc_cov: 0.1
    gyr_cov: 0.1
    b_acc_cov: 0.0001
    b_gyr_cov: 0.0001
    fov_degree:    180
    det_range:     100.0
    extrinsic_est_en:  false      # true: enable the online estimation of IMU-LiDAR extrinsic,
    extrinsic_T: [ 0, 0, 0.28]
    extrinsic_R: [ 1, 0, 0,
                   0, 1, 0,
                   0, 0, 1]

publish:
    path_en:  false
    scan_publish_en:  true       # false: close all the point cloud output
    dense_publish_en: true       # false: low down the points number in a global-frame point clouds scan.
    scan_bodyframe_pub_en: true  # true: output the point cloud scans in IMU-body-frame

pcd_save:
    pcd_save_en: true
    interval: -1                # how many LiDAR frames saved in each pcd file;
                                # -1 : all frames will be saved in ONE pcd file, may lead to memory crash when having too much frames.
```

pointcloud callback function in *LaserMapping.cpp*

```
PointCloudXYZI::Ptr pcl_wait_pub(new PointCloudXYZI(500000, 1));
PointCloudXYZI::Ptr pcl_wait_save(new PointCloudXYZI());
void publish_frame_world(const ros::Publisher & pubLaserCloudFull)
{
    float clip_ground = 0.0;
    if(scan_pub_en)
    {
        PointCloudXYZI::Ptr laserCloudFullRes(dense_pub_en ? feats_undistort : feats_down_body);
        int size = laserCloudFullRes->points.size();
        PointCloudXYZI::Ptr laserCloudWorld( \
                        new PointCloudXYZI(size, 1));

        for (int i = 0; i < size; i++)
        {
            if (laserCloudFullRes->points[i].z > clip_ground)
            {
                RGBpointBodyToWorld(&laserCloudFullRes->points[i], \
                                    &laserCloudWorld->points[i]);
            }
        }

        sensor_msgs::PointCloud2 laserCloudmsg;
        pcl::toROSMsg(*laserCloudWorld, laserCloudmsg);
        laserCloudmsg.header.stamp = ros::Time().fromSec(lidar_end_time);
        laserCloudmsg.header.frame_id = "base_link";
        pubLaserCloudFull.publish(laserCloudmsg);
        publish_count -= PUBFRAME_PERIOD;
    }
```

pcdtomap.launch

```
<!-- -->
<launch>
    <node pkg="pcdtomap" name="pcdtomap" type="pcdtomap" output="screen">
    <!-- filepath to save .pcd file -->
    <param name="file_directory" value=
"/home/renjie/workspace/ME5413_Final_Project/src/FAST_LIO_/PCD/" />
    <!-- .pcd file name-->
    <param name="file_name" value= "scans" />
    <!-- minimum height-->
    <param name="thre_z_min" value= "-2.0" />
    <!-- maximum height-->
    <param name="thre_z_max" value= "0.7" />
    <!--0 select points in height range, 1 select points out of height range -->
    <param name="flag_pass_through" value= "1" />
    <!-- radius filter's radius-->
    <param name="thre_radius" value= "0.3" />
    <!-- radius filter's points required-->
    <param name="thres_point_count" value= "10" />
    <!-- resolution of gripmap-->
    <param name="map_resolution" value= "0.05" />
    <!-- topic of gripmap, use map_server to save -->
    <param name="map_topic_name" value= "map" />
    </node>

</launch>
```