ME5413 Autonomous Mobile Robotics

Homework II SLAM

Submitted by Homework Group 6

LI Peizhuo	Matrix Number
YU Zhuoyuan	Matrix Number
ZHONG Ningze	Matrix Number

AY2023/2024 (SEMESTER 2)

12th, Mar.,2024

Content

TASK 1 2D SLAM

In this Section, we initially perform SLAM modelling with Cartographer on the provided ROS bag and save the map in .pgm format constructed by Cartographer. Subsequently, we utilise EVO to evaluate the performance of Cartographer. In the following sections, we will gradually describe how we achieved our objectives.

1.1 Cartographer

Cartographer SLAM is a probabilistic algorithm that uses a combination of data from various sensors, such as LiDAR, IMU, and cameras, to create a map of the environment while simultaneously determining the robot's location within that map. It employs a scan matching technique to align consecutive LiDAR scans and filter out noise and outliers and then uses an optimisation algorithm to estimate the robot's trajectory and map the environment. For more detail you can follow the link:

https://google-cartographer-ros.readthedocs.io/en/latest/compilation.html

1.2 Configuration of Cartographer

Upon analyzing the /scan topic within the 2dlidar.bag file, we discovered a set of data typed as sensor_msgs/LaserScan. This indicates that the ROS bag file contains a series of scans of the surrounding environment conducted by a LiDAR. Based on this discovery, we decided to utilize Cartographer's revo_lds.lua configuration file to process this data. This decision is based on the fact that revo_lds.lua is pre-configured for specific LiDAR sensors, optimizing crucial SLAM parameters like scan frame names and data preprocessing. In order to make Cartographer run better, we have made the following adjustments to the key parameters in revo_lds.lua:

Table 1 Adaption of Cartographer

Parameters	From	To
tracking_frame	horizontal_laser_link	base_footprint
published_frame	horizontal_laser_link	odom
provide_odom_frame	true	false
use_odometry	false	true
TRAJECTORY_BUILDER_2D.max_range	8	16

And command the <remap from="scan" to="horizontal_laser_2d" /> from demo_revo_lds.launch.

1.3 Result

As a result, there is only some drift in the positioning, and the maps built are more reasonable than before, which proves that the appropriate parameter settings can indeed greatly improve the performance of the algorithm.

To run the code, please view the README.md file in the appendix, we have provide the

command that can help you easily run the code.

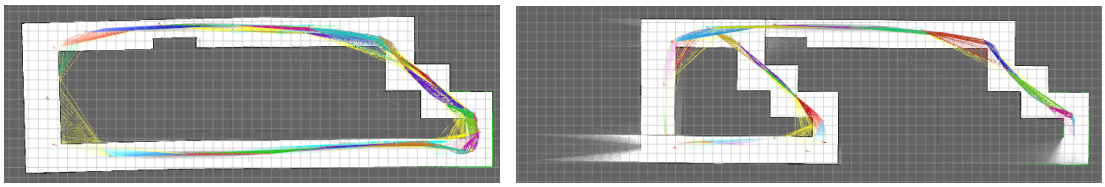


Fig 1 Result of odometry and pure LiDAR

The mapping results are shown in Fig 1. We found that Cartographer struggled to complete the mapping task effectively. This may be due to the algorithm's reliance on wheel odometry to supplement the localization process. Without odometry, Cartographer must rely solely on LiDAR data, which can be challenging in environments with fewer distinctive features or in scenarios where the sensor coverage is sporadic. Additionally, it is worth to know that at the end of the mapping process, especially in the lower left corner of the map, multiple large offsets were observed, which may be mainly due to cumulative errors and the SLAM closed-loop detection mechanism.

As mapping proceeds, errors will gradually accumulate, and when the robot re-explores the starting point or a known area, loop closure detection is activated to identify and correct these errors. During this process, the graph undergoes adjustments to eliminate accumulated deviations along the path and ensure overall accuracy and consistency of the map. This optimization can cause the map to visually deform significantly, especially after loop closures are detected and pose map optimization is performed. Nonetheless, these adjustments are necessary to maintain the accuracy and reliability of the final map. Correctly configuring SLAM parameters can help minimize this shift, ensuring a more efficient and accurate map building process.

Bonus EVO of Cartographer

In the evo test, we utilized the position of the base_footprint within the map coordinate frame from the /tf topic in the 2dlidar.bag as a reference. We compared this against the position of the base_link relative to the map generated by Cartographer. This comparison yielded the results are shown in Fig 2.

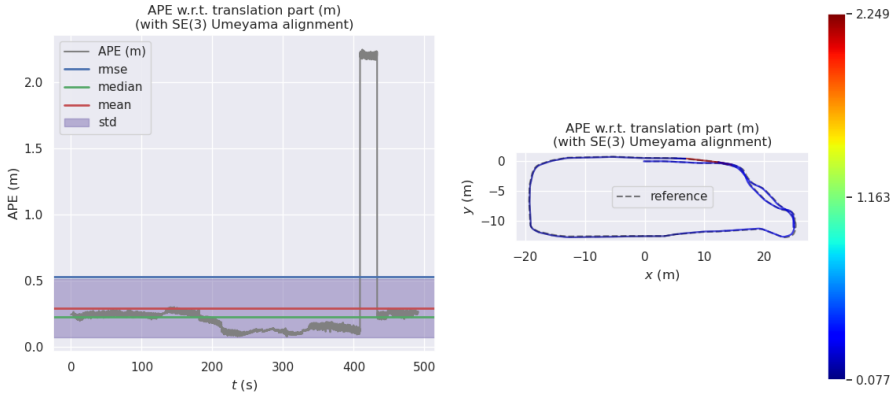


Fig 2 EVO test result

Task 2

In this section, we're delving into the performance of select SLAM algorithms within ROS, specifically focusing on Fast-LIO and A-LOAM, through Pure LiDAR and multi-sensor fusion methods. Utilizing EVO for precise evaluation, we'll assess these algorithms—Fast-LIO for its swift LiDAR-inertial odometry and A-LOAM for its sophisticated LiDAR odometry and mapping capabilities. Conducted in real-time, our comparison will spotlight differences in trajectory accuracy, map quality, and computational demands. The study leverages a Velodyne LiDAR-captured urban dataset from the provided ROS bag, aiming to offer insights that inform future developments in robotic mapping and navigation.

2.1 A-LOAM (Pure LIDAR SLAM Algorithm)

A-LOAM (Advanced LiDAR Odometry and Mapping) is an enhanced SLAM algorithm that capitalizes on LiDAR sensors for autonomous navigation and robotic mapping. This algorithm adeptly creates a 3D environmental map by emitting laser pulses and capturing the reflected beams to gauge distances. Concurrently, it integrates odometry data from the robot's motion sensors, enabling precise estimations of the robot's shifts between successive sensor scans. For more detail you can follow the link: https://github.com/HKUST-Aerial-Robotics/A-LOAM

2.1.1Adaptation of A-LOAM

Given that our system utilizes Ubuntu 20.04 with ROS Noetic, and considering our goal to generate a txt file in TUM format upon completion of the run for subsequent EVO evaluation, we have set up the following configuration:

- Modify “set(CMAKE_CXX_FLAGS "-std=c++11")” to “set(CMAKE_CXX_FLAGS "-std=c++14")” in the CMakeLists.txt.

Modifying the four '.cpp' files in the 'src' file:

- Modify “/camera_init” in the four '.cpp' files to “camera_init”.
- Modify “#include <opencv/cv.h>” in the scanRegistration.cpp to “#include <opencv2/imgproc.hpp>”.
- Modify “CV_LOAD_IMAGE_GRAYSCALE” in the kittiHelper.cpp to “cv::IMREAD_GRAYSCALE”.
- Add code in laserMapping.cpp to generate TUM.txt.

For this particular task, as the point cloud data is published under the topic /kitti/velo/pointcloud, it is necessary to modify the corresponding launch file. On the second line of the launch file, we add the following remap tag: “<remap from="velodyne_points" to="/kitti/velo/pointcloud"/>”. This ensures that the data is correctly read and utilized.

2.1.2A-Loam Result

First, compel A-LOAM by “catkin_make” and then create a new file named "txt", and generate aloam.txt. After that use the commands “cd A-LOAM_ws”, “source devel/setup.bash” and “roslaunch aloam_velodyne aloam_velodyne_HDL_32.launch” to run A-LOAM and play the ROS bag to generate the result as shown in Figx. At the same time, we employ the pointcloud_to_pcd method to save the point cloud images. We then select the last image

captured as the result shown as Fig 3.

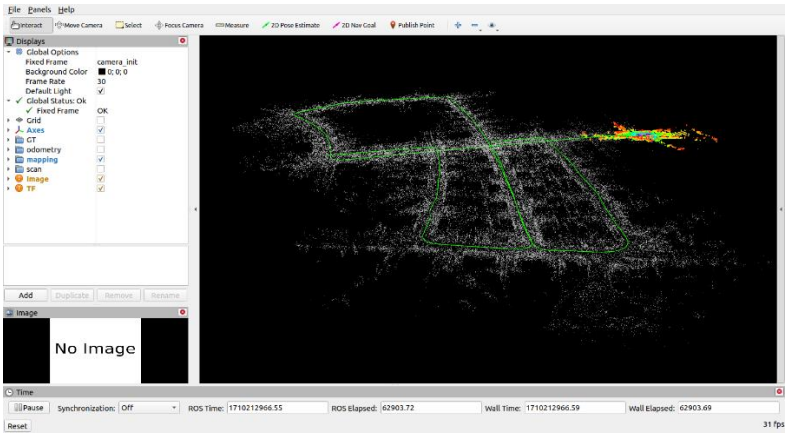


Fig 3 Result generated by A-LOAM

According to the position of odometry, camera and imu, calculate the coordinate transformation matrix in coordinate_trans.py, and output the measured coordinates "aloam_tum_2.txt" in the camera coordinate system. Run the command "evo_ape have_fun_tum.txt aloam_tum_2.txt -r full --plot --plot_mode xz" to get the ape of ALOAM SLAM result in TUM format shown in Fig 4.

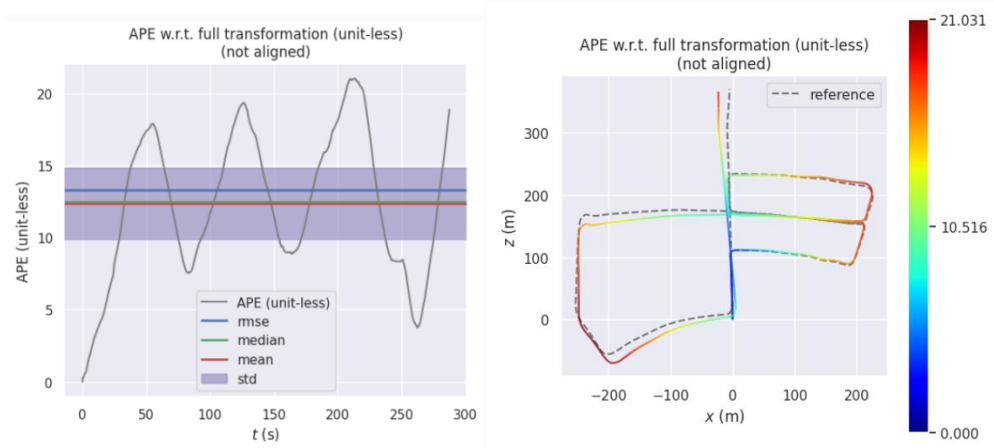


Fig 4 A-LOAM EVOst result

2.2 FAST-LIO (Multi-sensor Fusion SLAM Algorithm)

Fast-LIO happens in two major steps: firstly, inertial data is used for pre-integrating motion estimates, providing a fast prediction of the system's state. Secondly, these predictions are refined with LiDAR data through iterative optimization, aligning point clouds to a global map with high accuracy. The algorithm employs an efficient point cloud management strategy, reducing computational load by selectively processing only the most informative points. For more detail you can follow the link: https://github.com/hku-mars/FAST_LIO

2.2.1Adaptation of FAST-LIO

Due to Fast-LIO's strong adaptability, we made only minor adjustments related to ROS

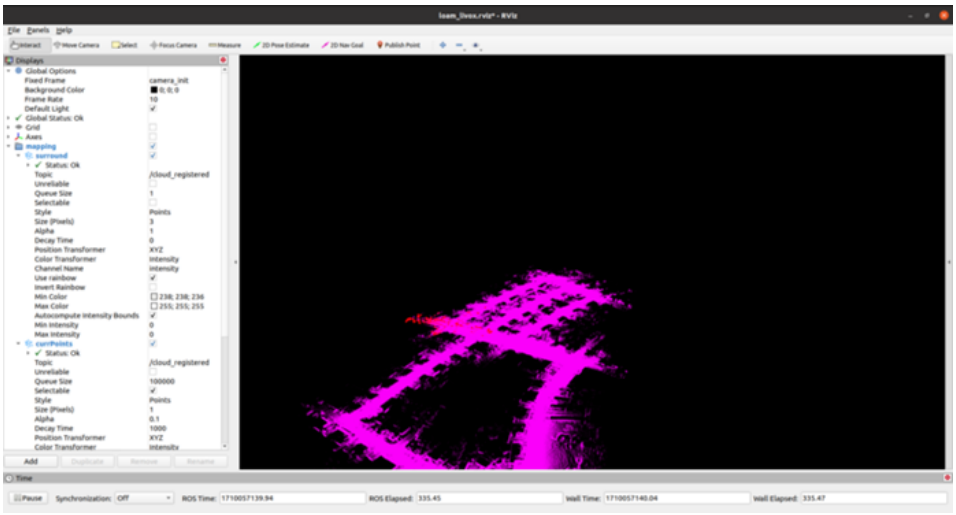
topics and the relative positioning of the LiDAR and IMU. The specific adjustments are as Table 2:

Table 2 Adaptation of Fast-Lio

Parameters	From	To
lid_topic	"/velodyne_points"	"/kitti/velo/pointcloud"
imu_topic	"/imu/data"	"/kitti/oxts/imu"
extrinsic_T	[0, 0, 0.28]	[0.81, -0.32, 0.8]

2.2.2Fast-Lio Result

To run the code, please view the README.md file in the appendix, we have provide the command that can help you easily run the code. The rviz result of Fast-Lio is shown as Fig 5.



As shown in the Fig 5, we observed that Fast-LIO performs well for most of the time but exhibits significant errors at the beginning and end stages. We suspect that the initial error may be attributed to a slow IMU initialization and insufficient data fusion at the start, while the ending errors could be due to the accumulation of drift over time. Which means improvement, such as enhancing IMU warm-up procedures and improving the data fusion algorithms early in the process, as well as implementing more robust drift correction techniques to maintain accuracy throughout the operation, will be useful.

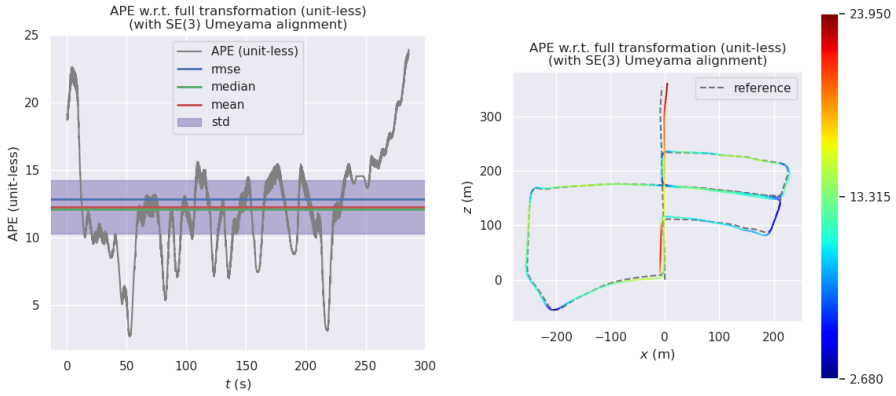


Fig 5 Fast-Lio EVO result