HOMEWORK REPORT

ME5413 AUTONOMOUS MOBILE ROBOTICS

GROUP 28

Anse Min (A0285307B)
Huang Yiming(A0285028B)
Niu Dixiao(A0284913X)
(Master of Science (Robotics), NUS)
(Master of Science (Mechanical Engineering), NUS)

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

Supervisor:

Prof. Marcelo H Ang Jr

14/3/2024

1 Task 1: Running Cartographer on the given ROS bag

1.1 Introduction

1.1.1 2dlidar.bag

The `2dlidar.bag` file is a ROS bag file containing sensor data and associated information necessary for SLAM (Simultaneous Localization and Mapping) using a package like Cartographer. This dataset includes:

- **Odometry:** Represents the ground truth trajectory for reference purposes.
- **Scan:** Contains the raw sensor data published by the 2D lidar sensor in the laser frame, which is essential for running mapping algorithms like Cartographer.
- **TF:** Includes the transformations that provide the positional relationship between different coordinate frames used in the dataset.

1.1.2 Cartographer

Cartographer is a system that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations. Developed by Google, it is open-source and designed to be highly versatile and easily deployable in various environments. Cartographer excels in creating a rapid and accurate map of an unknown environment while also determining the location within it, using data from various sensors like Lidar, radar, and cameras. Its algorithms are robust to variations in environment, sensor performance, and are used in applications ranging from autonomous vehicles to robotics.

1.2 Detailed Process

1.2.1 Modifications on Several Configurations

Before running Cartographer, several configurations need to be modified to build a link between Cartographer ROS and the given 2dlidar.bag.Basically following the official guide from *https://google-cartographer-ros.readthedocs.io/en/latest/your_bag.html*, we wrote our own `.lua` and `.launch` file to achieve it. There are several notable changes list bellow:
Firstly, we create `hw2_task1.lua` under `catographer_ros/configuration_files` then modify:

1. `tracking_frame = "laser_link"`
2. `published_frame = "odom"`
3. `provide_odom_frame = false`

`"tracking_frame"` is the name of the coordinate system tracked by the SLAM algorithm. Here we generally choose the highest frequency sensor that in the given rosbag, the sensor data is in laser_link and we choose it as the tracking frame.
`"published_frame"` is the name of the sub-coordinate system used to publish the position. Noticed that the rosbag has already provided odom frame, thus we select odom frame as the publish frame and set the provide value to false.
Secondly modify `hw2_task1.launch`:

1. Delete lines for finding robot urdf, because the rosbag has already provided /tf messages
2. Modify configuration into `-configuration_basename hw2_task1.lua`
3. Remap to /scan topic by `<remap from="scan" to="/scan" />`

1.2.2 Algorithm Running

The process of SLAM (Simultaneous Localization and Mapping) using Cartographer involves several key stages. This can be obseved in Fig4 after running the command:
`roslaunch cartographer_ros hw2_task1.launch bag_filename:=/<path>/2dlidar.bag`

1. **Initial Ego Trajectory (Frontend)**: This phase corresponds to the 'Frontend' in the SLAM framework. Cartographer first takes sensor data acquisition, particularly from a LiDAR sensor, to create an initial estimation of the robot's trajectory or ego-motion in relation to the environment.

2. **Initial Map Building**: At this early stage of 'Map Building', Cartographer uses the initial ego-trajectory to start forming a preliminary map. It's a rough draft of the environment based on the immediate data from the LiDAR.

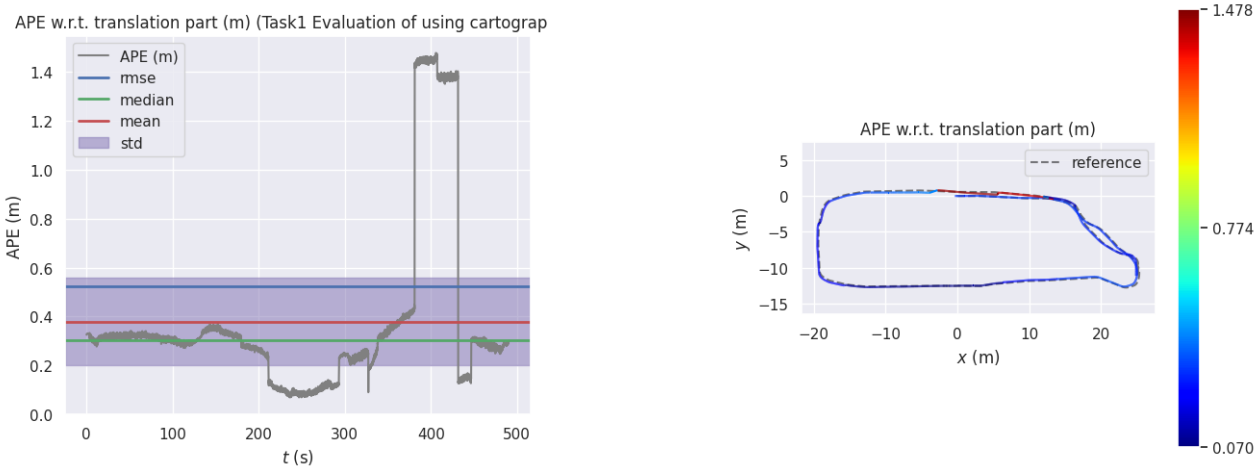
3. **Backend Optimizing Ego Trajectory**: In the 'Backend' phase, Cartographer refines the initial estimates. The backend optimization algorithms adjust the trajectory for accuracy, which often involves loop closure techniques to correct any drift and ensure consistency in the map.

4. **Final Map**: The final output after comprehensive 'Map Building', where all data and corrections are consolidated into a coherent and accurate map of the environment. This map is the result of iterative backend optimizations and loop closure adjustments that reconcile the robot's journey with the features of the physical space it has traversed (*Fig4 B, C*).

After rosbag finished, run `rosservice call /write_state` in a new terminal to save the `pbstream` file. Then run `rosrun cartographer_ros cartographer_pbstream_to_ros_map` to convert it into `.pgm` map, which is shown in Fig4 D.

1.3 BONUS: Evaluation

We noticed that Ground truth of the robot's trajectory is stored in the '¡2dlidar.bag¿' itself in topic '/odom', and the result trajectory is store inside map.base_footprint under /tf topic.
In order to use EVO tools for this rosbag, we recorded the two topic above (named as 'cartographer.bag') when we running the cartographer on the rosbag. Then after the bag finished can we get the recorded rosbag. Fig1 shows the results of evaluation of cartographer on 2dlidar.bag by running this command:
`evo_ape bag cartographer.bag /odom /tf:map.base_footprint -p --plot_mode xy -a`



(a) APE of Cartographer

(b) Estimated trajectories

Figure 1: Absolute Pose Error Result

2 Task 2: Running 3D SLAM algorithms on the given ROS bag

2.1 VINS-Mono

2.1.1 Brief Introduction of VINS

VINS-Mono is a real-time SLAM framework for Monocular Visual-Inertial Systems. It uses an optimization-based sliding window formulation for providing high-accuracy visual-inertial odometry.

A single vision sensor works well in most texture-rich scenes, but it basically doesn't work if it encounters a scene with fewer features such as glass, white walls, etc. Therefore, ideal localisation can be achieved by fusion of multiple sensors. The complementary nature of vision and inertial measurements makes them particularly suitable for fusion to improve accuracy. Although IMU have very large cumulative errors over long periods of time, their relative displacement data is highly accurate over short periods of time, so fusing IMU data can improve their positioning accuracy when vision sensors fail.

The VINS framework is divided into five sections. The first part, Measurement Preprocessing, involves the preprocessing of observation data, tracking of image data, and pre-integration of IMU data. The second part, Initialisation, is primarily used for pure visual initialisation and visual-inertial co-initialisation. The third and fourth parts, not mentioned in your summary, would naturally involve steps that bridge initialisation and the final optimization processes. The fourth part, Global Pose Graph Optimization, focuses on optimizing the global graph. Finally, the fifth part, Loop Detection, deals with the identification of loops. Block diagram illustrating the full pipeline of VINS was shown in Fig. 5a

2.1.2 Fine-Tuning of VINS

Since VINS is a vision-based odometry algorithm. The parameters of the camera have a great influence on the accuracy of odometry. The camera internal and external parameters included in the source code are calculated based on the EuRoC Dataset and need to be modified for the KITTI dataset. The transfer matrix from the camera to the body system needs to be changed to Eq.1

$$T = \begin{bmatrix} -0.02218873 & -0.01354233 & 0.99989895 & 1.1031531 \\ -0.99989259 & 0.00435299 & -0.02270281 & -0.85632959 \\ 0.03227481 & -1.00072563 & 0.00467743 & 0.76942567 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The camera internal reference matrix represents the projected relationship between the coordinates of an object in the world coordinate system and the coordinates of the image in the camera, usually denoted by f_x, f_y, c_x, c_y. According to the camera parameters of KITTI, we changed them to 7.070912e+02, 7.070912e+02, 6.018873e+02 and 1.831104e+02. The screenshot of VINS in operation was shown as Fig 5b.

2.1.3 Evaluation

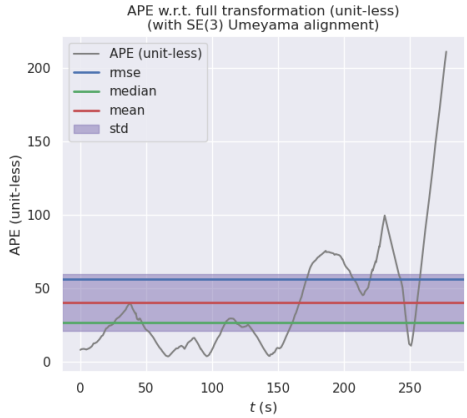
As shown in Fig. 2a,the RMSE value of VINS is about 50. due to the inherent defects of monocular vision odometry, the RMSE value of VINS is high compared to radar based odometry. The RMSE is also greatly increased due to the localization failure after 200s (shown in Fig. 2b). Drift of odometry

after 200s occurs when a vehicle stops and waits at an intersection and misrecognises a vehicle in the other direction, and therefore incorrectly assumes that the other vehicle's direction of movement is the direction of its own movement, leading to drift in the odometer. Although the error was corrected to some extent in the later loopback detection process, the odometer failure due to the error in the direction of motion was fatal.

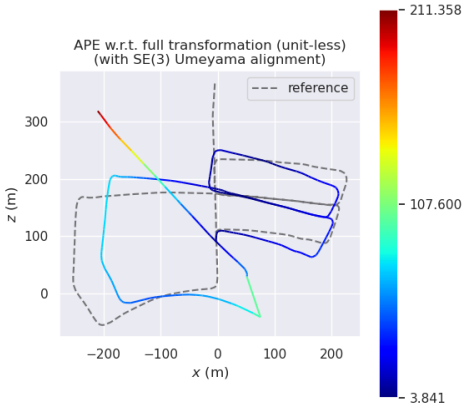
2.2 LOAM

2.2.1 A-LOAM

A-LOAM is an advanced version of the Lidar Odometry and Mapping (LOAM) algorithm, which is designed for real-time 3D mapping using lidar. The general LOAM algorithm divides the problem into two parts: odometry, which estimates the movement of the lidar at a high frequency but with lower fidelity, and mapping, which runs at a lower frequency but with higher accuracy for precise point cloud



(a) Absolute Pose Error(APE) of VINS



(b) Estimated trajectories

Figure 2: Absolute Pose Error Result

registration. A-LOAM stands out because it integrates enhancements over the original LOAM for better accuracy and efficiency in creating real-time maps. It generally maintains the same architecture with a front-end module for feature extraction and matching to estimate relative transformations, and a back-end module for map optimization.

2.2.2 LeGO-LOAM

LeGO-LOAM is a system tailored for robotic ground vehicles (UGVs). It differentiates itself from other LOAM algorithms by being lightweight and optimized for detecting the ground plane in variable terrain conditions. This optimization is particularly beneficial for UGVs that operate on the ground as opposed to aerial vehicles or applications that may not always have a clear ground plane.The system is capable of real-time 6D pose estimation. A notable feature of LeGO-LOAM is that it performs segmentation before feature extraction, ensuring that the ground is consistently detected for the mapping process. The odometry part of the system employs a two-step optimization process to compute the 6D transformation accurately.

For LeGO-LOAM to adapt to new sensors, proper projection of the point cloud into a range image is crucial, and ground detection must be precise. The system has been implemented with certain parameters for the VLP-16, but adjusting these parameters to match the specifications of a new sensor is key to ensuring successful mapping and odometry. It should be noted that while an IMU can greatly enhance the accuracy of the estimation, proper alignment with the Lidar sensor is crucial.

2.2.3 Algorithm Running

Both A-LOAM and LeGO-LOAM are pure lidar SLAM algorithm that we need to modify the file to subscribe lidar data of the rosbag (which is under /kitti/velo/pointcloud topic in have_fun.bag)

- `"scanRegistration.cpp"` under A-LOAM: CModify topic into "/kitti/velo/pointcloud" (line 478)

For A-LOAM:

- Run `roslaunch aloam_velodyne aloam_velodyne_HDL_64.launch`
- Run `rosbag play have_fun.bag` in a new terminal

For LeGO-LOAM:

- Run `roslaunch lego_loam run.launch`
- Run `rosbag play have_fun.bag --clock --topic /kitti/velo/pointcloud` in a new terminal

Fig6 a, b shows the running progress of the two algorithm, while Fig6 c, d shows the final point cloud maps respectively.

2.2.4 Evaluation

Both two LOAMs algorithm do not include tum of kitti format output directly, thus we need to record the results' odometry data into rosbag first and then convert them into tum format for transformation as well as using EVO tools.The Absolute Pose Error (APE) metric is used in SLAM systems to gauge the accuracy of a robot's trajectory against a known baseline. It measures how far off the estimated positions are from the actual path.

For A-LOAM, heightened APE at the trajectory's start and conclusion could suggest initialization issues or complications during the final state estimations. This situation might arise from the algorithm grappling with an unfamiliar setting or attempting to leverage environmental features that lack consistency or are sparse (Fig7, A). LeGO-LOAM, conversely, shows a more uniform APE, indicative of its ground plane segmentation from 3D LiDAR data, an approach that likely affords it stability by focusing on consistent features for localization, thereby rendering it less prone to environmental dynamism. The relative flatness of its APE curve through both the commencement and wrap-up of the mapping process points to LeGO-LOAM's ability to offer reliable trajectory estimates consistently. This steadiness is particularly advantageous in environments where other algorithms might falter, suggesting that LeGO-LOAM may outperform A-LOAM in terms of robustness across a variety of settings (Fig7, B).

Achieving a low and steady APE is vital for the dependable operation of robotic navigation and the precision of the generated maps. Both A-LOAM and LeGO-LOAM exhibit resilience; however, LeGO-LOAM's specialized optimizations might provide it with an edge in maintaining trajectory consistency. This thorough comparison underlines the importance of selecting the right SLAM algorithm tailored to specific environmental conditions and the robotic application's requirements.

Fig8 reveals insights into each SLAM algorithm's trajectory estimation accuracy over time. A-LOAM (Fig7, A) exhibits significant error peaks at the beginning and end, suggesting challenges with initial calibration or abrupt trajectory changes, while LeGO-LOAM (Fig7, B) shows a more stable error pattern, indicating consistent performance likely due to efficient ground plane optimization.LeGO-LOAM segments and processes LiDAR data from horizontally-placed sensors like the Velodyne VLP-16, ensuring ground plane detection. This focus on ground segmentation contributes to its stable performance, offering an edge over other LOAM systems that may not specifically target ground optimization. Also its robustness in different terrains is achieved through the correct projection of the point cloud into a range image and the accurate detection of the ground, which is vital for the algorithm's mapping and localization processes.

The mean error for A-LOAM is around 4.4, compared to LeGO-LOAM's 3.5, highlighting LeGO-LOAM's superior trajectory alignment with the ground truth. This difference suggests that LeGO-LOAM may process sensor data more effectively or maintain algorithmic stability across diverse environmental conditions, reflecting how each algorithm manages environmental complexities and sensor noise.

APPENDICES

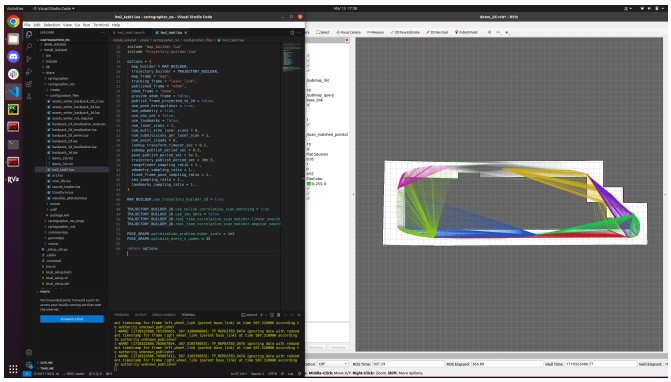


Figure 3: Screenshot of the Cartographer SLAM process.

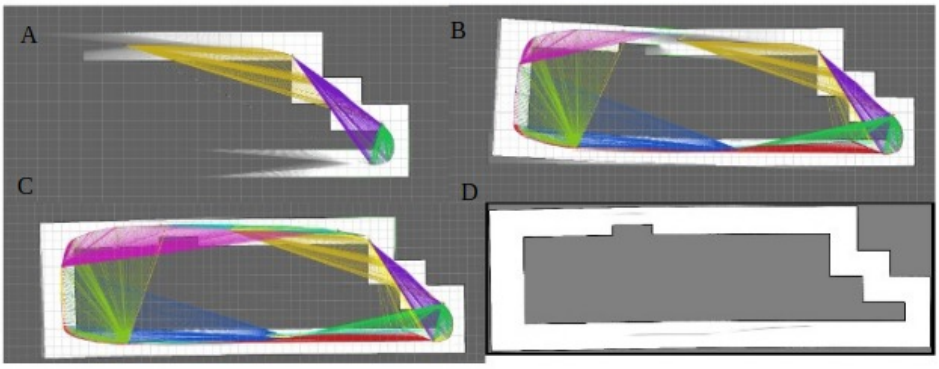
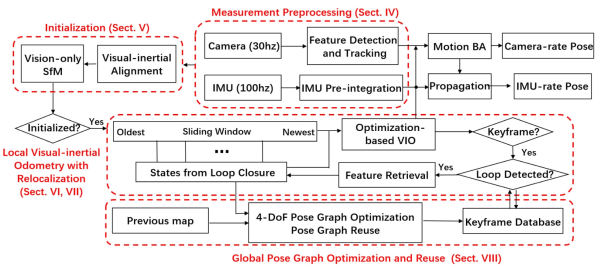
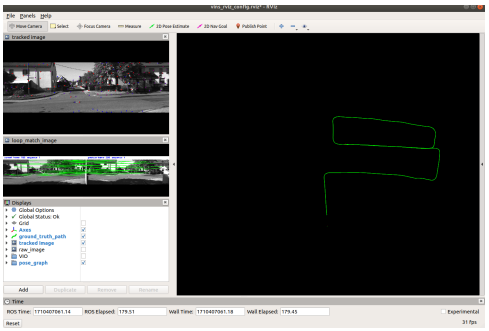


Figure 4: The sequential stages of the Cartographer SLAM process.

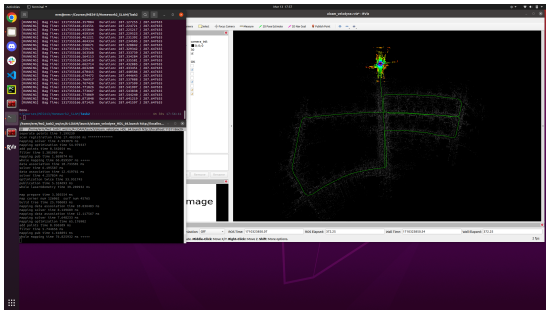


(a) Work Flow of VINS

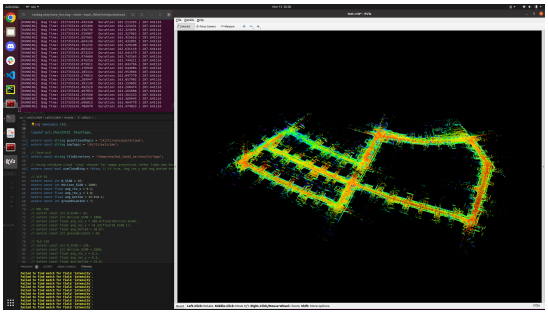


(b) VINS in Operation

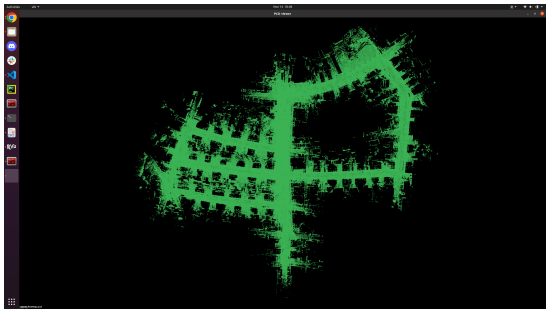
Figure 5: VINS



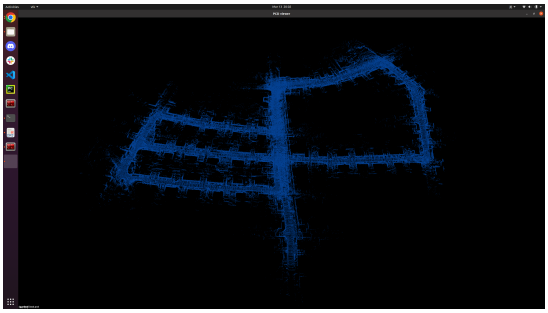
(a) A-LOAM in progress



(b) LeGO-LOAM in progress



(c) A-LOAM point cloud map



(d) LeGO-LOAM point cloud map

Figure 6: Screenshots of LOAM running

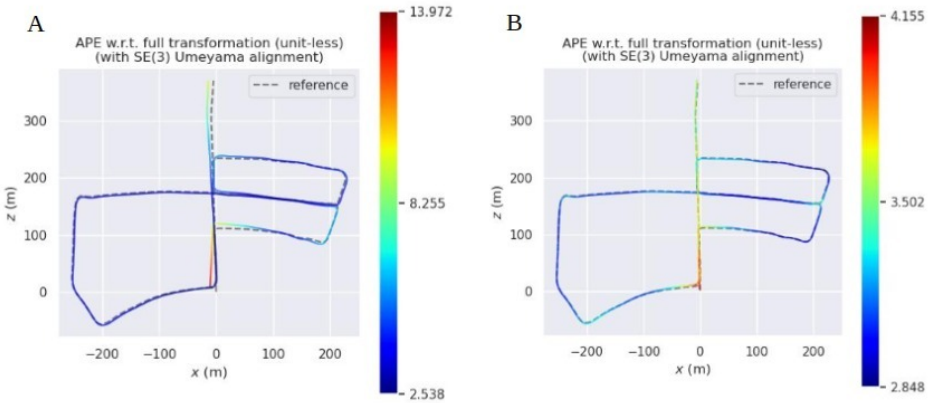


Figure 7: ALOAM (A) and LEGO-LOAM (B) APE Ego-Trajectory

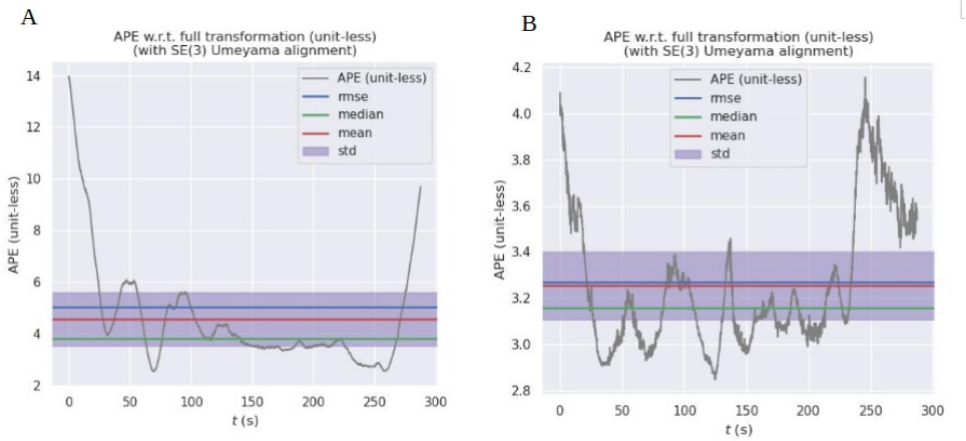


Figure 8: ALOAM (A) and LEGO-LOAM (B) APE vs Time graph