Homework 2: SLAM

Assignment For

ME5413 Autonomous Mobile Robotics

Submitted by

Group 15

Fan Xiuqi A0285060J
Wang Suji A0284859E
Chen Ziao A0285086U

2024.03.14

Faculty of
Mechanical Engineering
National University of Singapore

Session 2023/2024

1. Task one: Running Cartographer

1.1 An overview of the principles

a. Input Sensor Data:

LiDAR data undergoes voxel filtering and adaptive voxel filtering before scan matching, while chassis odometry information is processed through attitude extrapolation and scan matching. IMU data, including linear acceleration and angular velocity, is preprocessed and utilized for attitude extrapolation before scan matching.

b. PoseExtrapolator:

Chassis odometry and previous odometry compute incremental (x, y, θ) changes to estimate velocity. IMU pre-integration calculates velocity, displacement, and angular changes. The last pose and previous pose calculations derive incremental (x, y, θ) changes to estimate acceleration and angular velocity. These computations collectively contribute to estimating the initial pose for the subsequent front-end match.

c. Local SLAM:

A submap is created from several consecutive scans, which can be thought of as a two-dimensional array, and the value under each index represents the probability. Each time the latest scan is obtained, it needs to be inserted into the optimal position in the submap, so that when the pose of the point bundle in the scan is converted and falls into the submap, the reliability and reliability of each point are the highest. The optimization of Th is carried out by scan matching, where the optimization problem is to solve the least squares problem. The method is a bicubic interpolation, the output result of this function is a number within (0,1), and the number outside of this can be generated, but not taken into account, through the optimization of this smoothing function, it can provide better accuracy than raster resolution. The least-squares problem is solved in Cartogrper using Google's own Ceres library.

d. Global SLAM

Insert the scan of the submap & all the previously created sub maps - > loop detection (here branch and bound) - > compute constraints - > pose optimization

Loopback detection: All created submaps and the current laser scan will be used as matching for loopback detection. If the current scan and all created submaps are close enough, then the loop will be found through some sort of match strategy. In order to reduce the amount of computation and improve the efficiency of real-time loopback detection, Cartographer applies the branch and bound optimization method to optimize the search. The matching problem can be described as follows:

$$\xi^* = argmin \quad \sum_{k=1}^{K} M_{nearest}(T_\xi h_k) \qquad \text{(BBS)}$$

$$\xi \in W$$

If a good enough match is obtained, at this point, the loopback detection part has ended, and the loopback has been detected. Next, we need to optimize the poses in all sub maps based on the current scan pose and a pose in the submap that matches the closest one, even if the residual E is the smallest.

1.2 Code and Results

The SLAM algorithm through Cartographer is configured via *my_robot.launch* and *my_robot.lua*, while visualizing the result via the default *demo_2d.rviz*. The launch file defines a series of nodes to run the SLAM system and the lua file contains information about sensor settings and algorithm parameters. The desired topics including /tf, /odom, /map are recorded in a new rosbag for map generation in p.m. format and performance evaluation by EVO. The node graph is shown in image 10.

2. Task two: Running 3D SLAM algorithms

2.1 A-LOAM

2.1.1 An overview of the principles

a. Feature Point Extraction

The reference criterion for feature point extraction is curvature, and since the laser returns unevenly distributed points in the environment, the smoothness of such a local surface is defined as the criterion for classification. The smoothness is calculated as follows:

$$c = \frac{1}{|S| \cdot \left|\left|X^L_{(k,i)}\right|\right|} \left|\left| \sum_{j \in S, j \neq i} \left(X^L_{(k,i)} - X^L_{(k,j)}\right) \right|\right|$$

where S is the set of successive points returned by the laser in the same frame.

$X^L_{(k,i)}$ refers to L (radar) coordinate system the ith point in Point cloud P_k for the kth scan.

After the curvature is calculated, a scan is divided into 4 identical sub-regions, each of which provides 2 edge points and 4 planar points according to the size of the curvature.

b. Finding Feature Point Correspondence

The feature association algorithm uses point cloud data in two consecutive frames

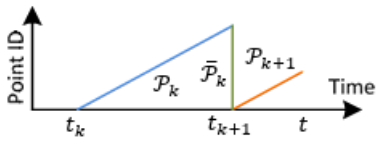

Figure 1. Reprojecting point cloud to the end of a sweep

In order to obtain the data correlation between two frames, the distance from the point to the line should be as small as possible, and the distance from the point to the surface should be as small as possible. Therefore, the distance calculation formula is given as follows:

$$d_\varepsilon = \frac{\left|\left(\tilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}\right) \times \left(\tilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,l)}\right)\right|}{\left|\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right|}$$

c. Motion Estimation

After obtaining the inter-frame relationship, how to solve it in mathematical form is the focus of our state estimation algorithm, and the paper uses the idea of nonlinear least squares to construct the residual function, optimize the distance parameters, and obtain the transition matrix is what we consider to be the optimal solution. The Leeuwenberg-Marquardt method of nonlinear optimization is used to construct and solve the problem, which is expressed as follows:

$$T_{k+1}^{L} \leftarrow T_{k+1}^{L} - (J^T J + \lambda diag(J^T J))^{-1}(J^T d)$$

d. Lidar Odometry Algorithm

The odometer part is basically obtained by integrating the aforementioned algorithm content

e. Lidar Mapping Algorithm

The mapping algorithm is calculated based on scan-to-map/submap. The blue curve in the above figure is the lidar change attitude on the map, which is generated by the mapping algorithm in the kth scan. The orange curve is scanned at k+1 moment and is obtained based on the change in T_{k+1}^{L}. The mileage calculation method publishes a relatively stable point cloud message, which is mapped to the global map by the conversion of the radar to the world coordinate system, and this process is represented as a green box matching the existing black box to update the map.

2.1.2 Code and Results

We implement A-LOAM algorithm through *aloam.launch*, which contains parameter settings and node launching for map construction. The /velodyne_points topic is remapped to /kitti/velo/pointcloud to ensure that A-LOAM can receive the correct point cloud data. The result is visualized via the default *aloam_velodyne.rviz*. The node graph is shown in image 11. The number of scanning lines of the lidar is set to 64 while the frequency of mapping is set as 10. We removes points that are too close, with a minimum set to 5 meters, helping filter out noise points that may occur when the lidar is too close to objects. The topics including /aft_mapped_to_init and /laser_cloud_map are recorded in a new rosbag for map generation and performance evaluation.

2.2 LEGO-LOAM

2.2.1 An overview of the principles

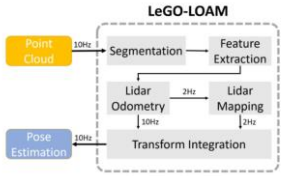

Figure 2. Monolithic framework

a. Segmentation

Add the Segmentation operation to project the point cloud as a distance image to separate the ground points from the non-ground points. Each point that remains will have three attributes: (1) the label of the point cloud; (2) the number of rows and columns in the distance image; (3) Distance value.

b. Feature Extraction

This module is similar to the classical LOAM in calculating smoothness, taking into account the uniform distribution of feature points in all directions, and divides the distance image level into several sub-images.

c. LiDAR Odometry

In the process of feature extraction $\{F_p, f_p, F_e, f_e\}$ we obtained four sets of feature points,

and optimized them as follows:

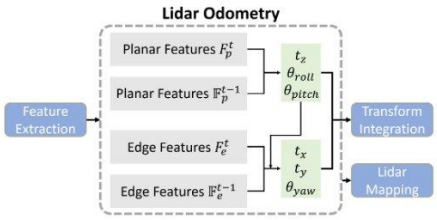

Figure 3. Lidar odometry performs two-step Levenberg Marquardt optimization to get 6D transformation.

d. LiDAR Mapping

The global point cloud map is divided into many cubes, and then a certain number of cubes are selected according to the effective detection distance of the sensor (100m for VLP-16) and combined into the point cloud map Q^{t-1} obtained at the time of $t-1$, and then to the current frame Q^t and its intersecting parts, which is basically the same as the classical LOAM algorithm.

2.2.2 Code and Results

Based on the algorithm and existing repository on GitHub, the launch file should be edited to fit your own demands. The input of the algorithm is the topic /kitti/velo/pointcloud. By launching the ros, we also put the topics necessary into a new rosbag including the odometry topic and point cloud topic and the data are read into tum format. In order to transfer the coordinate from the lidar coordiate to the camera 0 coordinate, we use a transition matrix to achieve the result. The node graph is shown in image 12. The script has been written in *trans.py*. The RMSE of the result is 12, which is relatively high in image 6. The reasons may be as follows:

The feature point extraction process may be affected by sensor noise, motion blur or environmental changes, resulting in unstable extracted feature points, which affects the accuracy of localization and map building. As there may be errors in the extraction and matching process of feature points, these errors may accumulate over time, leading to a gradual increase in the localization and mapping errors. The transition matrix may be not accurate enough, leading to some errors as well.

2.3 VINS

2.3.1 An overview of the principles

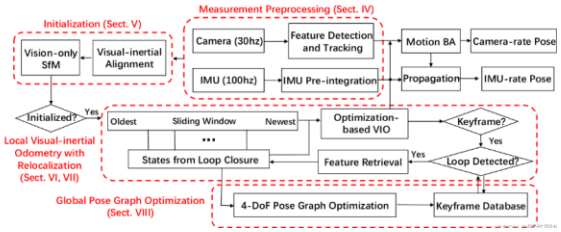

Figure 4. Overall framework

The functional module of VINS can include five parts: front-end data preprocessing, initialization, back-end nonlinear optimization, closed-loop detection, and closed-loop

optimization. VINS-FUSION is based on VINS-Mono and adds GPS and other sensors that can obtain global observation information so that VINS-FUSION can use global information to eliminate cumulative errors and reduce closed-loop dependence.

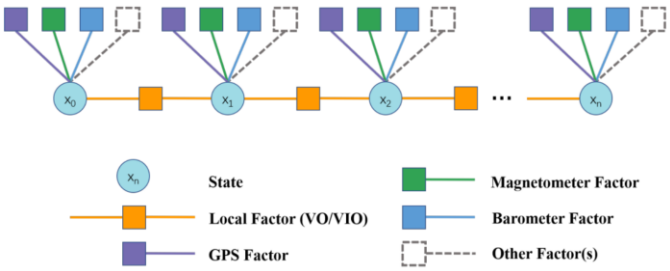

Figure 5. Constraints between observations and states

Convert GPS coordinates, i.e. latitude and longitude, to a geodetic coordinate system. It is customary to choose the right-hand coordinate system, and the positive direction of the x, y, and z axes is the northeast-earth or northeast-heaven direction respectively. Next, we can calculate the residuals of the global constraints:

$$Z_t^{GPS} - h_t^{GPS}(X) = Z_t^{GPS} - h_t^{GPS}(X_t) = P_t^{GPS} - P_t^{W}$$

where Z is the GPS measurement, X is the state prediction, and the h equation is the observation equation. X can be calculated from the state of the previous moment and the pose transformation of the current moment from the previous moment. where Z is the GPS measurement, X is the state prediction, and the h equation is the observation equation. X can be calculated from the state of the previous moment and the pose transformation of the current moment from the previous moment. After that, the optimization is handed over to the BA optimizer for iterative optimization, and vins fusion uses Ceres as the optimizer.

2.3.2 Code and Results

Compared to the algorithms above, the difference between vins and loam is that vins is a multi-sensor fusion SLAM algorithm but loam is a pure lidar algorithm. Besides lidars, this algorithm needs more sensors including two cameras and imu. In our algorithm, we just use two cameras, which topics are /kitti/color_image_left/image_raw and /kitti/color_image_left/image_raw. Thus, a configuration file should be realized, which is the .yaml file including the position and information of the camera. We get the odometry and point cloud topics by recording the rosbag and put the odometry topic into a tum format. The node graph is shown in image 13. After that, we compare two methods with and without loopback detection. The results are shown in images 8 and 9. Compared with both images, the loop optimization can significantly increase the accuracy of the algorithm. However, the RMSE is still 17.77. The reason of the bad performance may be that the cameras are not configed very well. Moreover, we do not use the IMU to help cameras do SLAM.

3. Comparison and Conclusion

Compared with the three algorithms, due to lack of IMU information, the Vins-Fusion performs worst. In comparison to both Aloam and Lego Loam, the ALoam is more accurate in outdoor SLAM due to its algorithm, which uses loop closure detection other than feature detection in Lego Loam.Thus, based on the sensors and algorithm we use, the A_Loam performs best on the given dataset.

Appendix

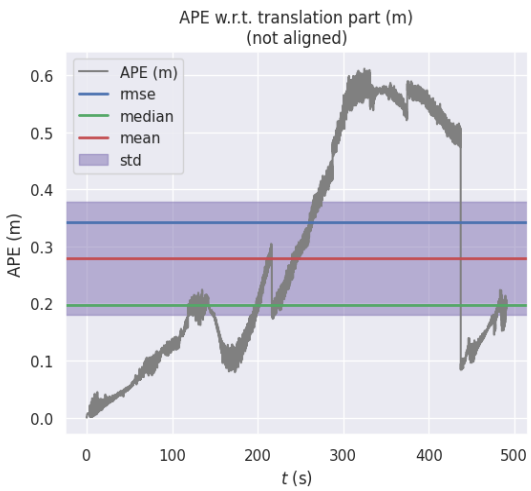

Image 1. Cartographer error

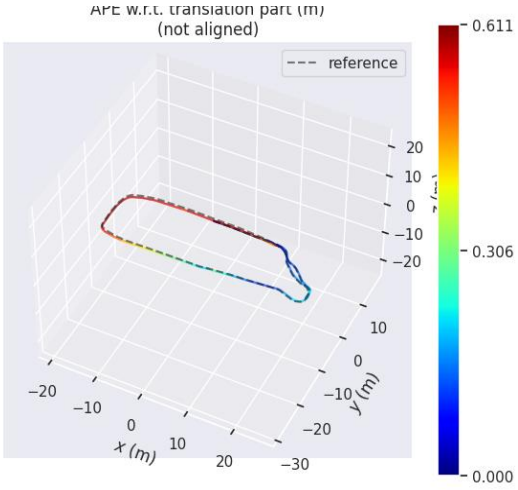

Image 2. Cartographer error in 3D

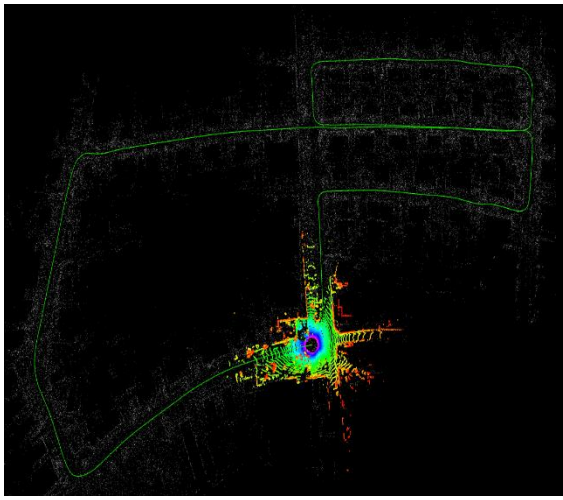

Image 3. The path by A_Loam

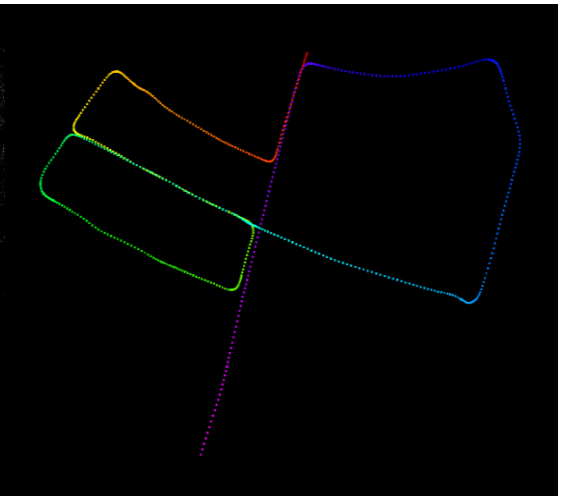

Image 4. The path by Lego_Loam

Image 5. The path by Vins-fusion

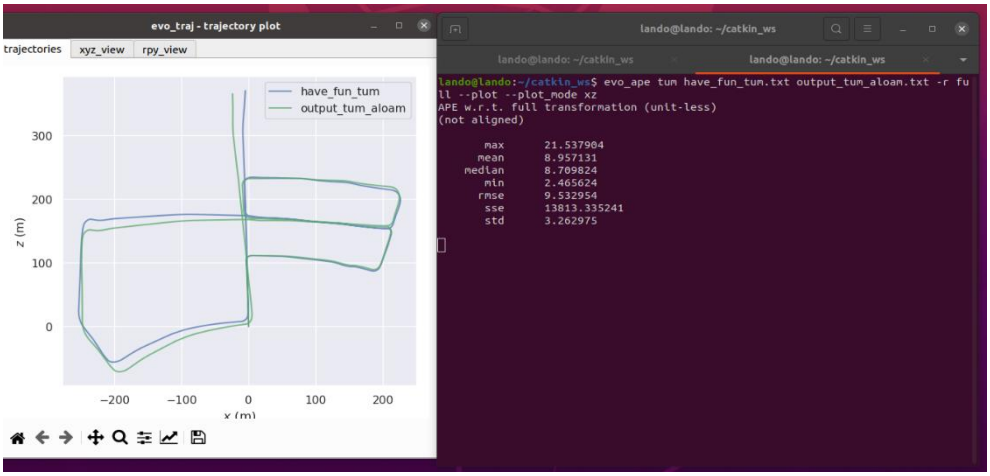

Image 6. The error of Aloam path

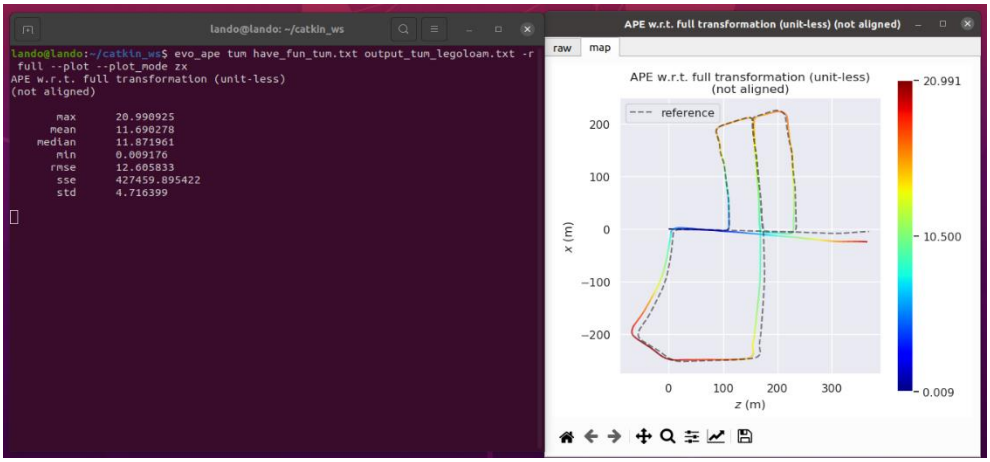

Image 7. The error of Lego Loam path

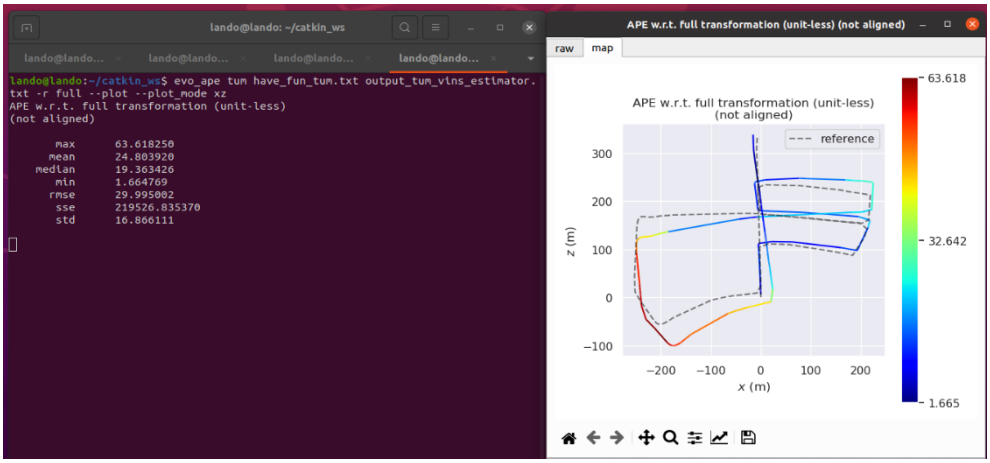

Image 8. The error of the Vins-fusion without loop

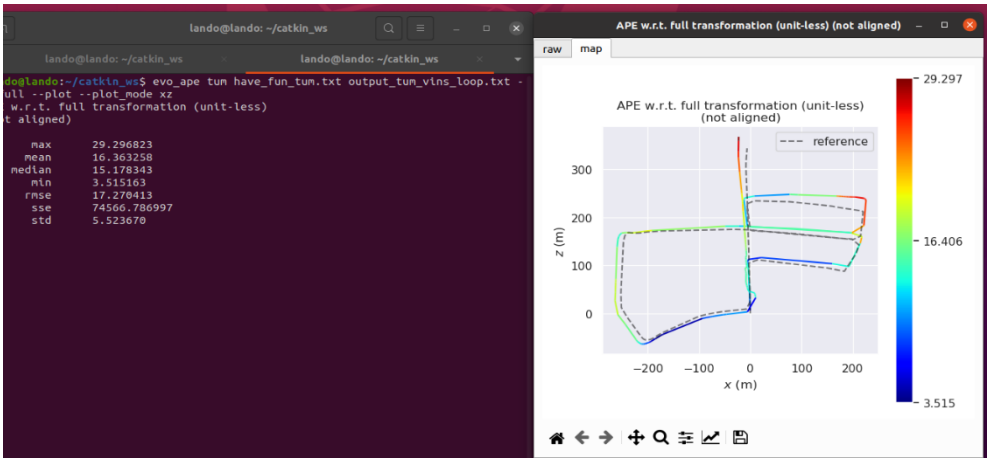

Image 9. The erroe of the Vins-fusion with loop

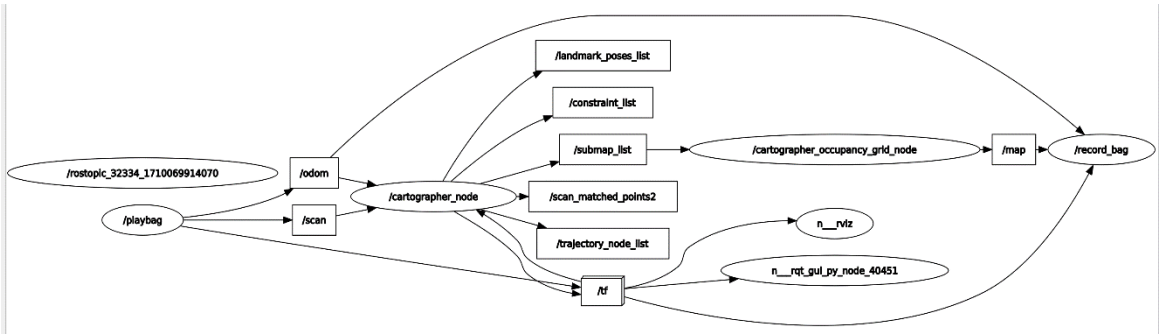

Image 10. The node graph of cartographer

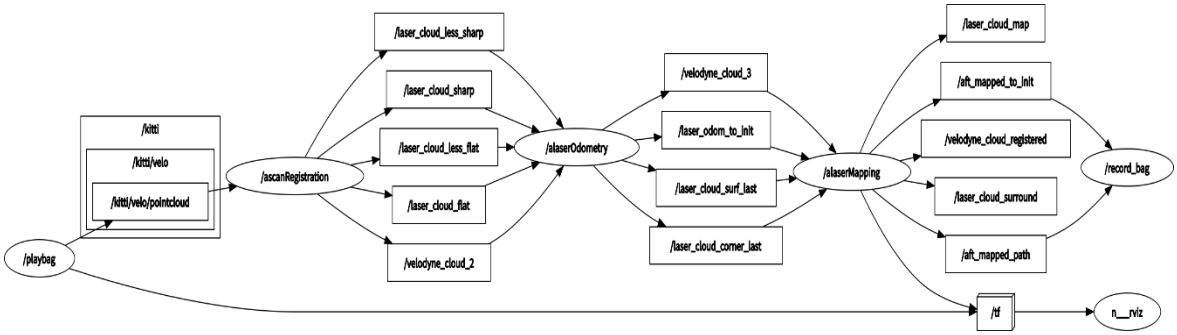

Image 11. The node graph of A_loam

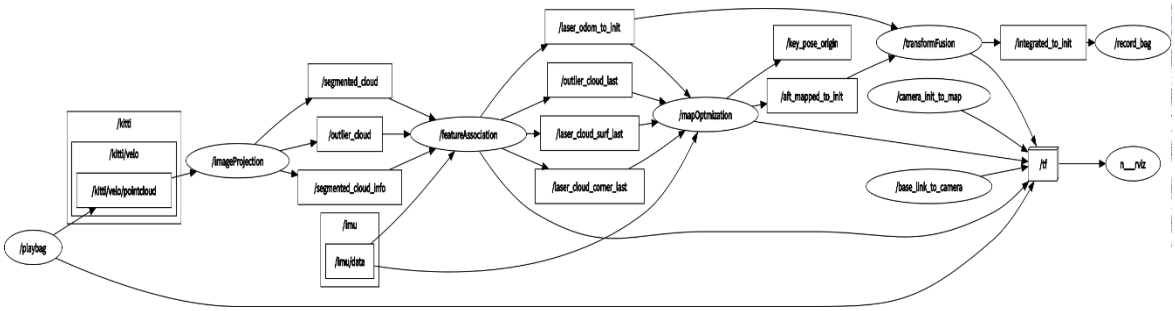

Image 12. The node graph of Lego_loam

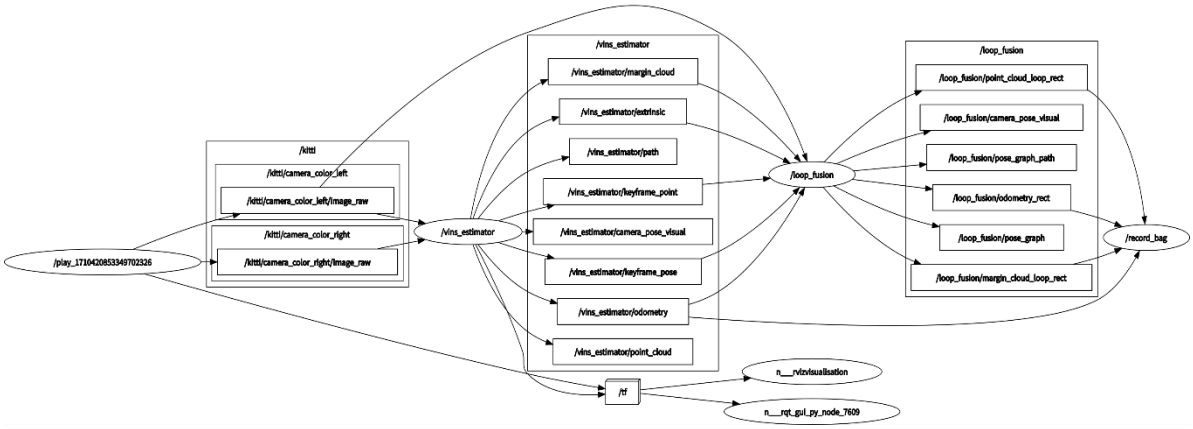

Image 13. The node graph of Vins-fusion