Scan Context 3D Lidar Inertial Odometry via Iterated ESKF and Incremental K-Dimensional Tree

by

Chang Xu

Submitted in partial fulfilment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
June 2022

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis focused on a 3D lidar inertial odometry algorithm framework that improves the Lightweight and ground optimized lidar odometry and mapping (LeGO-LOAM) by constructing a new back-end optimization algorithm. In comparison with the LeGO-LOAM, the feature extraction and image projection processes are still the same. Two step Levenberg Marquardt method was replaced with an iterated ESKF method in the lidar odometry to produce a better initial pose for the robots, and the k-dimensional(k-d) tree method in the lidar mapping is replaced with the ikd-Tree method to ensure high performance mapping process in real time. In the loop closure, a scan context search method is added to better correct the algorithm's final trajectory.

The proposed algorithm is tested and simulated by configuring the robot operating system (ROS) with the ubuntu virtual Linux system. The performance of the optimized back-end algorithm has compared with other three algorithms to show the proposed algorithm has better accuracy in the localization and mapping process.

LIST OF ABBREVIATIONS USED

SLAM	Simultaneous Localization and Mapping
NDT	Normal Distributions Transform
ICP	Iterative Closest Point
G2O	General Framework for Graph Optimization
GTSAM	Georgia Tech Smoothing and Mapping
LOAM	Lidar Odometry and Mapping in Real-time
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
LEGO	Lightweight and Ground Optimized
LIDAR	Light Detection and Ranging
CSM	Correlative Scan Matching
LM	Levenberg Marquardt
KD	K-Dimensional
IKD	Incremental K-Dimensional
ROS	Robot Operating System
PCL	Point Cloud Library
TF	Transform Fusion
APE	Absolute Pose Error
RPE	Relative Pose Error

ACKNOWLEDGMENTS

I was very grateful to everyone who helped me finish my thesis during my graduate studies.

First and foremost, I would thank my supervisor: Dr. Jason Gu, who provided me the opportunity to work on this thesis and helped me find a research direction. His professional guidance and asking experts in related research fields to answer my questions helped me solve many problems and challenges. And I was also very grateful to my peer Hanxiang Zhang. When I was researching this thesis, we helped each other and made progress in the process of learning and exploring together.

I also wanted to appreciate two professors of my committee members: Dr. Ya-Jun Pan and Dr. Kamal El-Sankary. I was grateful for their time on my defense, and the professional suggestions.

Finally, I wanted to thank my family, without their support, understanding, and care. I may not be able to achieve my goals smoothly.

CHAPTER 1 INTRODUCTION

1.1 Background

In today's society, human development is getting faster and faster, especially in the development of science and technology. As people's lives become more prosperous, many new industries emerge. Especially after entering the 21st century, robotics and artificial intelligence have become the most popular research projects, and various countries and large technology internet companies have entered this field. After decades of development, robots and artificial intelligence systems have become a part of our lives and can help human to do a lot of things. For example, robotics are mainly divided into two categories which are robotic arms, and mobile robotics (footed robotics). Many industrial robots are based on robotic arms; as shown in Fig. 1(a), robotic arms are used in factories to build cars. In many surgeries today, the direct operation of the doctor by hand may cause the big wound, and the blood loss will lead to many problems. A unique medical robotic arm has been developed to assistant the doctors to operate, as shown in Fig. 1(b), which can reduce the risk of the process and make the process easier. Wounds can be made smaller, allowing the patient to heal faster.

(a) [44]

(b) [45]

Figure 1: (a) Industrial robotic arms, (b) Medical robotic arms.

Mobile robots are now mainly used for transportation. As shown in Fig. 2(a), heavy objects can be placed on top of mobile robots to transport to the directed location. It can reduce the burden of people when handling them. As shown in Fig. 2(b), mobile robots can also be put into firefighting. The crawler-type design allows it to move in various complex environments. It can replace firefighters entering the fire scene for local firefighting to protect firefighters' safety.

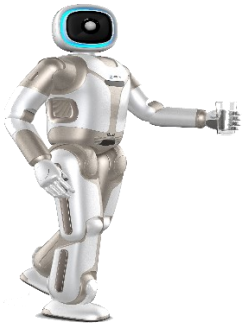
(a)[46]

(b) [47]

Figure 2: (a) Transport mobile robotics, (b) Firefighting mobile robotics.

Legged robots are part of the mobile robots, and it is mainly divided into quadruped robots (Fig. 3(a)) and bipedal humanoid robots (Fig. 3(b)). Compared with robotic arms and mobile robots, these robotics is still immature technology. Robustness and balance are significantly challenged in unstable ground or particular environments.

(a)[48]



(b) [49]

Figure 3: (a) Four-legged robotics, (b) Bipedal humanoid robotics.

Path planning, obstacle avoidance, and state estimation are already the fundamental prerequisites for mobile robots. Simultaneous localization and mapping (SLAM) is a popular method, which is now widely used in autonomous driving with mobile robotics. High-performance real-time simultaneous localization and mapping based on visual and LiDAR sensors can support six degrees of freedom state estimation [1]. Cameras and lidars are the most mainstream sensor for SLAM; they have their effects in different using environments. For example, cameras are influenced by temperature and light but lidars are not. As Fig. 4 shows, different types of cameras and lidars are commonly used in this research.

(a) [50]

(b) [51]

(c) [52]

(d) [53]

Figure 4: Different types of cameras and lidars: (a) T-265 tracking camera, (b) RGBD depth camera, (c) Ouster 1-64 channel, (d) Velodyne VLP-16

Due to the lightweight, camera has already become a common substitute for lidar. Visual-based methods usually use monocular or stereo cameras and triangulate features in continuous images to determine camera movement. Vision-based frameworks have VPS-SLAM, ORB-SLAM, feature constrained active visual SLAM, etc. [2]-[4]. Although vision-based methods are particularly suitable for location recognition, the accuracy and robustness of the camera are poor. It will have the uncertainty problem when the monocular camera faces the scale problem. In addition, external factors such as weather, light, temperature, and appearance changes can also affect the accuracy of the camera [5]. The lidar has a higher resolution and strong anti-action interference ability than the camera. It can provide more accurate and farther environmental measurement, and it is not affected by changes in illumination. Lidar is mainly used in two different environments, indoor and outdoor, and it mainly uses two different spatial methods, which are 2D and 3D for LiDAR-SLAM. 2D LiDAR SLAM comprises Gmapping [6], Fast-SLAM [7], Hector-SLAM [8], Nav SLAM [9], KartoSLAM [10] and so on. Similarly, autonomous vehicles have greater demand for outdoor scenes, 3D lidar is currently used more frequently, and then the obtained point cloud is used for mapping and positioning. Now mainstream state-of-the-art algorithms have LOAM [11], LeGO LOAM [12], and LIO-SAM (Tightly coupled Lidar Inertial Odometry via Smoothing and Mapping) [13], and Suma (Surfel-based Mapping for 3d Laser Range Data) [14], etc. These advanced frameworks and algorithms aim to achieve low-drift and high-precision self-motion state estimation. Although 3D lidar SLAM is relatively expensive, it is more accurate and stable than visual SLAM and 2D lidar SLAM [15]. Therefore, this report will focus on 3D lidar SLAM fusion with IMU; its 3D point cloud map effect shown in Fig. 5.

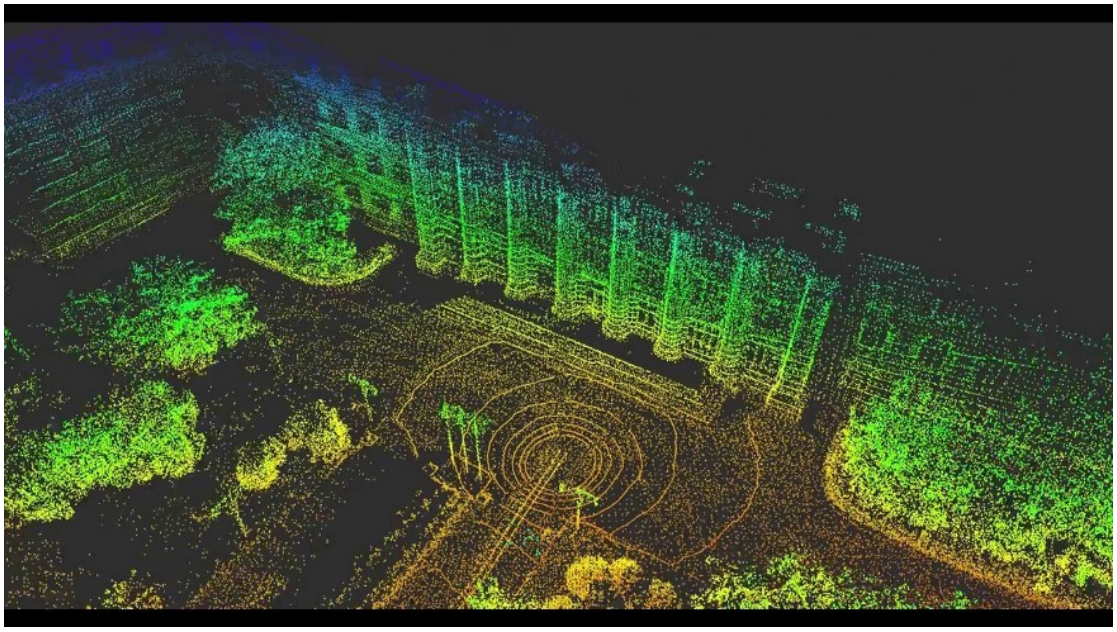


Figure 5: 3D LiDAR SLAM point cloud map [54]

As shown in Fig. 6, the first step of the general localization and mapping process of lidar SLAM is to receive sensor data and transmit these data to the front end. The front-end preprocesses the sensor data and do the point cloud feature association and scan matching, and the classic methods used in the front end are ICP [16], NDT [17] and CSM [18] which is published and used in the cartographer. After that, the point cloud data processed by the front end is transmitted to the back end for nonlinear optimization. The main methods for nonlinear optimization are graph optimization and filters. The popular methods of graph optimization are G2O [19] and GTSAM [20] mainstream libraries, and the non-linear filter optimization methods are mainly used Kalman filter (KF) [33], extended Kalman filter (EKF) [43] and particle filter (PF) [42]. At the same time as back-end optimization happened, the loop closure will work together to correct the final global map. Combining all the point data from the back-end optimization and the correction from the loop detection, the entire SLAM positioning and mapping process is realized.

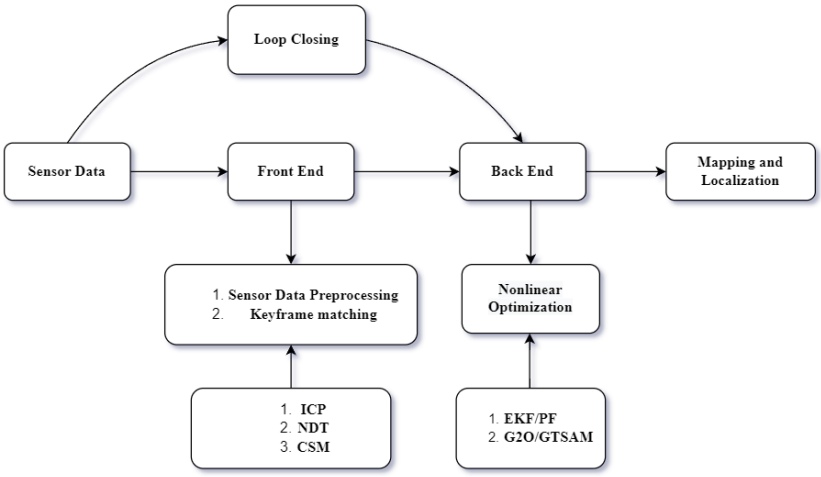


Figure 6: General localization and mapping process for LiDAR SLAM

1.2 Contributions

According to the background mentioned above of lidar SLAM, in 2D indoor lidar SLAM, the cartographer framework [41] that open-sourced from Google become the best algorithm and no other algorithms can better than it in same conditions. Therefore, everyone has begun to research and optimize lidar SLAM in large-scale outdoor scenes in 3D environments under such circumstances. This paper proposes a fast 3D lidar inertial odometry via scan context loop closure and incremental k-d tree framework. This framework improves the back-end optimization of the algorithm which is based on the front-end of the LeGO-LOAM. Three methods were changed in the back-end optimization in the proposed algorithm to achieve a better performance in large-scale outdoor scenes in real-time. The main contributions are as follows:

- The first optimization is the processing of IMU pre-integration. The IMU pre-integration processing of LeGO-LOAM will still generate drift and jitter when running IMU and lidar odometry datasets. To get a better initial pose of the

robots, LINS [25] proposed an iterated error state Kalman filter method to ensure both accuracy and efficiency.

- The second optimization part is to replace the search method of the k-d tree with the search method of the ikd-Tree [23] in the lidar mapping part, which is more efficient and ensures better real-time performance and increases the speed and accuracy of the mapping process.
- The last optimization is to add the scan context search method [24] to the final loop closure detection part. The scan context method is used to preserve the internal structure of the point cloud. At the same time, the absolute position information of the lost point can be surfaced to reduce perceptual aliasing error in the case and better correct the final trajectory of the algorithm.

The primary process of the algorithm can be shown in Fig. 7 below, and the back-end optimization made is in the red box:

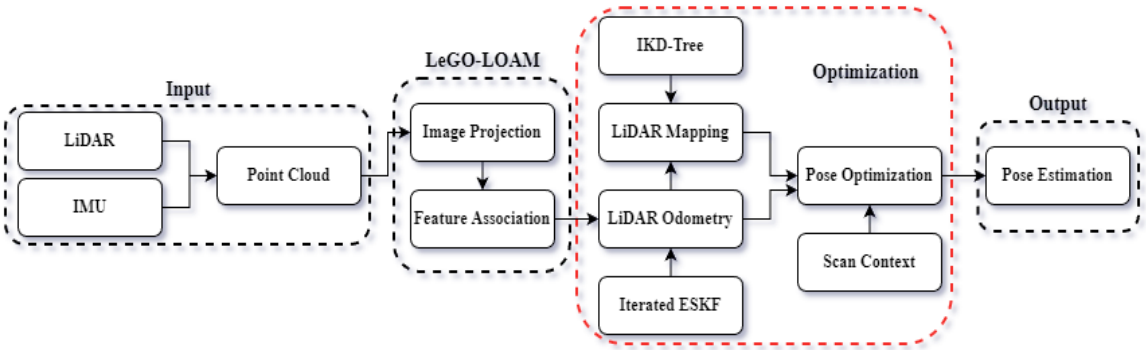


Figure 7: The main process of the proposed algorithm

1.3 Organization

This thesis report is mainly divided into the following chapters:

Chapter 1: This chapter will introduce the history of the robotics and background of the SLAM, including the usage, types of SLAM, and why I choose 3D lidar as my thesis.

Chapter 2: This chapter is mainly a literature review of three lidar SLAM algorithms used to discuss and compare with the improved slam algorithm in chapter 4.

Chapter 3: This chapter will introduce the improved parts and principles of the proposed algorithm so that readers can more intuitively know which features are improved and how it works.

Chapter 4: This chapter will compare and discuss the proposed algorithm with the lidar SLAM algorithm introduced in Chapter 2, which shows the trajectories and errors of those algorithms compared with the ground truth.

Chapter 5: The final chapter summarizes the paper and proposes future improvements and work.

CHAPTER 2 LITERATURE REVIEW

Based on my research direction which is 3D lidar SLAM, this chapter will introduce some classic and popular lidar SLAM algorithm frameworks commonly used in robotics companies in recent years, such as LeGO-LOAM, LIO-SAM and A-LOAM. In the fourth chapter, it will compare these algorithms with the improved algorithm to better reflect the performance optimization of the algorithms relative to these mainstream algorithms.

2.1 Lightweight and Ground Optimized Lidar Odometry and Mapping

LeGO-LOAM is a lightweight and ground-optimized lidar odometry and mapping method based on LOAM [11] for real-time six-degree-of-freedom (6-DOF) attitude estimation of ground vehicles. It is mainly used for the ground optimization, using the existence of ground plane in its segmentation and optimization steps. First, the point cloud segmentation is used to filter out the noise, and the plane and edge features are obtained by the feature extraction method. Then the two-step LM optimization method [40] is used to solve the transformation between the elements to eliminate the pose estimation error caused by drift. Compared to LOAM, LeGO-LOAM can achieve better accuracy at a reduced computational cost [12].

The primary process of LeGO-LOAM is shown in Fig. 8. The initial point cloud segmentation and feature extraction will be mentioned in Sections 3.1 and 3.2. In this chapter, lidar odometry and lidar mapping parts for the LeGO-LOAM will be described.

The lidar odometry part is divided into two steps, named label matching and two-step L-M optimization. This module estimates the sensor's motion in two frames during

consecutive scans and performs point-to-edge or point-to-plane scan matching to find the transformation relationship [12]. This part is briefly described [36].

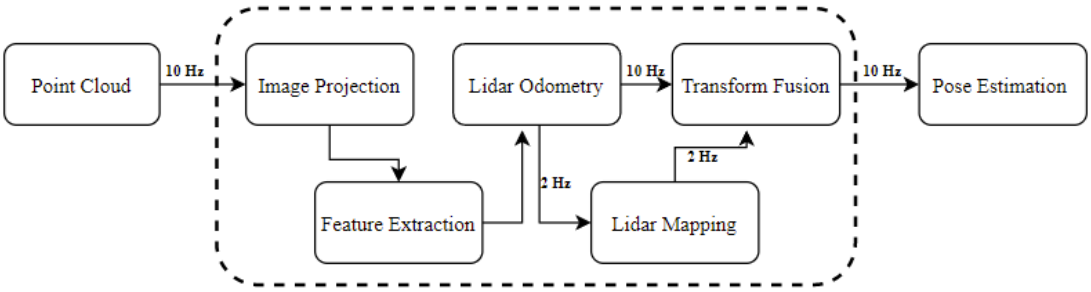


Figure 8: LeGO-LOAM system overview [12]

Label matching is equivalent to the optimization of the first step here. Compared with LOAM [11], the operation of loopback detection is added here. Its general mapping process is shown in Fig. 9.

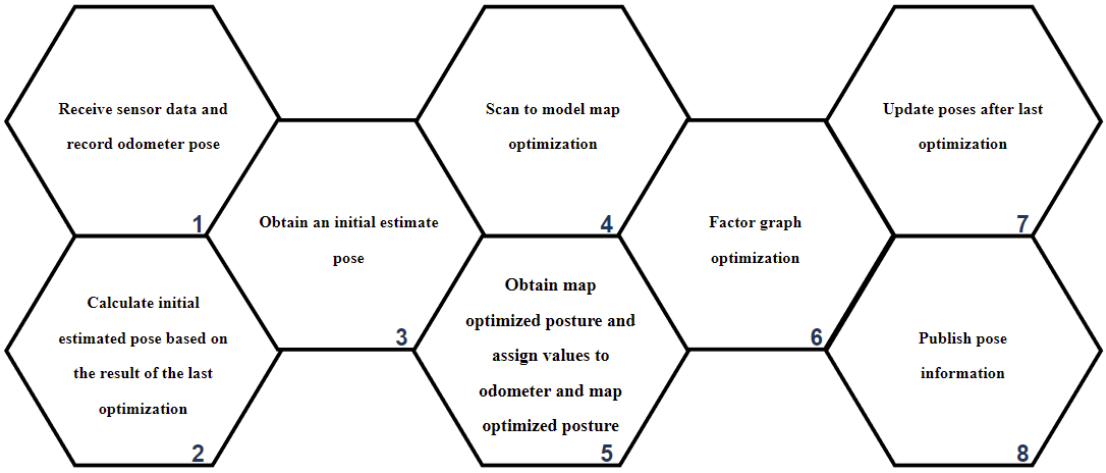


Figure 9: Mapping process of map optimization

The first function here is the callback function, which converts all the point cloud data from the format defined by ROS to the PCL format. Then the process of closed-loop detection and visualization of the map is carried out, and then the primary function of this part is the back-end optimization. In this article, the back-end optimization method uses the gtsam library. The primary better process is to convert the coordinate

system to the world coordinate system to obtain lidar coordinates used for mapping. If the closed-loop detection is performed, a new point cloud is inserted at the end and the old point cloud in the front is deleted. If the closed-loop is not performed, the search is performed in the neighborhood, and the keyframes are stored in the corresponding queue after the double loop. Whether the loop is closed or not, downsampling is performed to reduce the amount of data. The latest point cloud data obtained by downsampling is registered with the existing map to update the robot's precise pose and fusion mapping. It is mainly divided into corner point optimization and plane point optimization, registration and update [12].

The principle of corner optimization is to calculate the eigenvalues and eigenvectors of the orthogonal matrix. If it is a corner feature, its eigenvalue will be much larger than the other two features. If it is a plane feature, then the eigenvalue will be much smaller than the other two features, then decide whether to optimize. If optimized, it will define three sets of variables and calculate the area of the parallelogram obtained by the cross product between them by Eq. (1). Then the normal vector as Eq. (2) can be obtained by the cross-product again [12].

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (y_0 - y_1)(z_0 - z_2) - (y_0 - y_2)(z_0 - z_1) \\ -(x_0 - x_1)(z_0 - z_2) - (x_0 - x_2)(z_0 - z_1) \\ (x_0 - x_1)(y_0 - y_2) - (x_0 - x_2)(y_0 - y_1) \end{bmatrix}, \tag{1}$$

$$[l_a \quad l_b \quad l_c] = [X \quad Y \quad Z] \times [x_1 - x_2 \quad y_1 - y_2 \quad z_1 - z_2]/a_1/l_1, \tag{2}$$

where $a_1 = \sqrt{X^2 + Y^2 + Z^2}$, $l_1 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$. The norm of $\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$ is area of parallelogram, and $[l_a, \quad l_b, \quad l_c]$ is unit normal vector. Then the surface optimization is carried out, and the principle is basically the same as that of the corner point optimization [12].

The final optimization is a two-step LM optimization [40]. Its optimization process is to set Eq. (3) as a point on the local coordinate system and transform it into Eq. (4) in the global map coordinate system. Then define the error and combine the above equation to get the error as shown in Eq. (5). Based on Eq. (4), use the error to obtain partial derivatives of rotation and translation, respectively as shown in Eq. (6), and finally solve the content displayed in Eq. (7) [12].

$$X_{(k+1,i)}^{L} = (px, py, pz)^{T}, \tag{3}$$

$$X_{(k+1,i)}^{W} = G\left(X_{(k+1,i)}^{L}, T_{(k+1)}^{W}\right) = R \cdot X_{(k+1,i)}^{L} + t, \tag{4}$$

Where $X_{(k+1,i)}^{L}$ and $X_{(k+1,i)}^{W}$ are point i at time k+1 in the lidar and world frame, px, py, pz are the feature points, $G(.)$ is a transform function and $T_{(k+1)}^{W}$ includes rotations and shifts.

$$d = D\left(X_{(k+1,i)}^{W}, map\right) = D\left(G(X_{(k+1,i),}^{L} T_{(k+1)}^{W}), map\right) = D\left(R \cdot X_{(k+1,i)}^{L} + t, map\right), \tag{5}$$

$$\frac{\partial loss}{\partial ex} = \frac{\partial D(G(X_{(k+1,i),}^{L} T_{(k+1)}^{W}), map)}{\partial ex} = \frac{\partial D(\cdot)}{\partial G(\cdot)} \cdot \frac{\partial\left(R \cdot X_{(k+1,i)}^{L}\right)}{\partial ex}, \tag{6}$$

$$\frac{\partial D(\cdot)}{\partial G(\cdot)} = \frac{\partial d}{(\partial X_{(k+1,i)}^{W})} = \left(\frac{\partial d}{\partial x}, \frac{\partial d}{\partial y}, \frac{\partial d}{\partial z}\right) = (l_a, l_b, l_c),$$

$$\frac{\partial\left(R * X_{(k+1,i)}^{L}\right)}{\partial ex} = \frac{\partial(R)}{\partial ex} \cdot \left(p_x, p_y, p_z\right)^{T} = \begin{bmatrix} sy \cdot cx \cdot sz & cz \cdot sy \cdot cx & -sx \cdot sy \\ -sx \cdot sz & -sx \cdot cz & -cx \\ cy \cdot cx \cdot sz & cy \cdot cz \cdot cx & -cy \cdot sx \end{bmatrix} \cdot \left(p_x, p_y, p_z\right)^{T}. \tag{7}$$

Same approach can solve $\frac{\partial\left(R \times X_{(k+1,i)}^{L}\right)}{\partial ey}$ and $\frac{\partial\left(R \times X_{(k+1,i)}^{L}\right)}{\partial ez}$.

After completing the above steps and algorithm equations, we realize the functions of lidar odometry surveying and mapping. After that, the next step we have to do is the coordinate system's transformation and fusion. Its primary function is the fusion calculation of pose information. Driven by two callback functions, the first is to obtain the accurately registered pose as the after mapped transformation and obtain

the registered speed as before the mapped change to prepare for the following calculation. Another callback function is to fuse the odometry information from the coarse registration in feature association with the odometry information in the map optimization, then send the final outgoing odometry topic in the callback function. The topic of transform fusion and odometry in the callback function is the final decision. Compared with LOAM [11], LeGO-LOAM has a minor relative error and running time performance.

2.2 Tightly Coupled Lidar Inertial Odometry via Smoothing and Mapping

This framework for high precision trajectory estimation and map construction used by smoothing and mapping tightly coupled lidar inertial odometry. LIO-SAM can perform a variety of relative and absolute measurements, including loop closure detection. The IMU brings the initial pose estimation to assume a nonlinear motion model to eliminate the tilt of the point cloud to optimize the lidar odometry. The framework uses local-scale scan matching instead of global-scale scan matching, which significantly improves the real-time performance of the system. Selective introduction of keyframes and an effective sliding window store new keyframes into a previously fixed-size set of sub-keyframes. This is the main contribution that the framework brings [13].

In this system, the world frame is W, the robot body frame is B, R is the rotation matrix, p is the position vector, v is the velocity, and b is the IMU bias. Then we can get the state matrix x of the robot as in Eq. (8) [13].

$$x = [R^T, p^T, v^T, b^T]^T. \tag{8}$$

The LIO-SAM [13] system receives sensor data from 3D lidar, IMU or GPS and uses these observations to estimate the state and trajectory of the robot. This state

estimation can be formulated as a maximum a posteriori problem, modelled using a factor graph equivalent to solving a nonlinear least-squares problem under the assumption of a gaussian noise model. The system mainly consists of the IMU pre-integration factor, lidar odometry factor, GPS factor, and loop closure factor. The optimization is performed using Bayes trees [37] for incremental smoothing and new nodes after mapping to erode the previous elements. Since my project does not use the GPS factor, this part will not be introduced below.

Through the measurement definition of the angular velocity and acceleration of the IMU, the original IMU measurement value at time t can be obtained. Using the measurement value of the IMU, the motion trajectory of the robot can be inferred, and the velocity V, position P and rotation R equations of the robot at time $t + \Delta t$ can be obtained in Eq. (9) [13]. b_t is the bias and n_t is the white noise here, $\hat{\omega}_t$ and $\hat{a}_t$ are the raw IMU measurements. Assuming that the angular velocity and acceleration remain constant during the above process, the relative motion in the two times can be obtained using the IMU pre-integration method [38].

$$\begin{aligned} V_{t+\Delta t} &= V_t + g\Delta t + R_t(\hat{a}_t - b_t^a - n_t^a)\Delta t, \\ P_{t+\Delta t} &= P_t + V_t\Delta t + \frac{1}{2}g\Delta t^2 + \frac{1}{2}R_t(\hat{a}_t - b_t^a - n_t^a)\Delta t^2, \\ R_{t+\Delta t} &= R_t exp\big((\hat{\omega}_t - b_t^\omega - n_t^\omega)\Delta t\big). \end{aligned} \tag{9}$$

In the lidar odometry factor, the operation of feature extraction is performed first. The edge or plane feature is extracted by the roughness of the point in the local area, and the more significant roughness point is the edge feature. LIO-SAM [13] adopts the method of keyframe selection, which is widely used in the field of visual SLAM. All lidar frames between two keyframes will be discarded to balance map density and memory consumption, thus maintaining a relatively sparse factor graph for nonlinear optimization.

The lidar odometry is mainly composed of three parts, which are the sub-keyframes of the voxel map, scan matching and relative transformation. The sliding window method is used in the voxel map to create a fixed number of point cloud maps scanned by lidar, and keyframes are extracted for estimation. It is transformed with sub-keyframes and merged into two types of voxel maps based on the feature extraction. Scan matching uses the method in [36] because of better computational efficiency and robustness. To obtain features in the lidar frame by IMU to predict the initial transformation of robot motion. The final relative transform is mainly the principle of point-to-line and point-to-surface of the LeGO-LOAM [12]. It is very similar to Eq. (13-14) because the two papers are from the same institution. After that, the Gauss-Newton method is used to solve the optimal transformation by minimizing Eq. (10) to get $\Delta T_{i,i+1}$ which in Eq. (11), which is the lidar odometry factor of the two poses [13].

$$\min_{T_{i+1}} \left\{ \sum_{p_{i+1,k} \in {}' F_{i+1}^{e}} d_{e_k} + \sum_{p_{i+1,k} \in {}' F_{i+1}^{p}} d_{p_k} \right\}, \tag{10}$$

$$\Delta T_{i,i+1} = T_i^T T_{i+1}. \tag{11}$$

$p_{i+1,k \in {}' F_{i+1}^{e}}$ means edge features $p_{i+1,k}$ in ${}' F_{i+1}^{e}$ and $p_{i+1,k \in {}' F_{i+1}^{p}}$ means planar features $p_{i+1,k}$ in ${}' F_{i+1}^{p}$. $\Delta T_{i,i+1}$ is to transform sub keyframes in to the frame x_i.

Because the proposed algorithm does not involve the content of the GPS factor, it will not be mentioned here. The last is the loop closure factor, where the factor graph method is used, which is a loop closure detection method based on Euclidean distance. The function implementation principle is that when a new state X_{i+1} is added to the factor graph, the factor graph is searched. The initial states that are close to X_{i+1} in Euclidean space are found, and then the sub-keyframes in the lidar frame are matched to get the relative transformation. Adding it to the graph as a closed-loop factor can correct the height error [13].

LIO-SAM [13] shows better performance on various outdoor datasets in terms of error when comparing LOAM [11] and LIOM [39].

2.3 Lidar Odometry and Mapping in Real-time

Since the lidar odometry is in motion, errors will continue to accumulate in the integration process, which will lead to the drift of the odometry. LOAM [11] proposed a method using point cloud matching and feature extraction to solve the problem of odometry drift. It first uses a high-frequency but lower-accuracy odometry for matching, then uses a low-frequency but higher-precision odometry for correction, and uses a combination of high and low-frequency odometry to correct the drift and error of the odometry to ensure real-time performance and accuracy at the same time. A-LOAM is a lidar-based slam framework based on LOAM [11] and the theory is the same as LOAM [11].

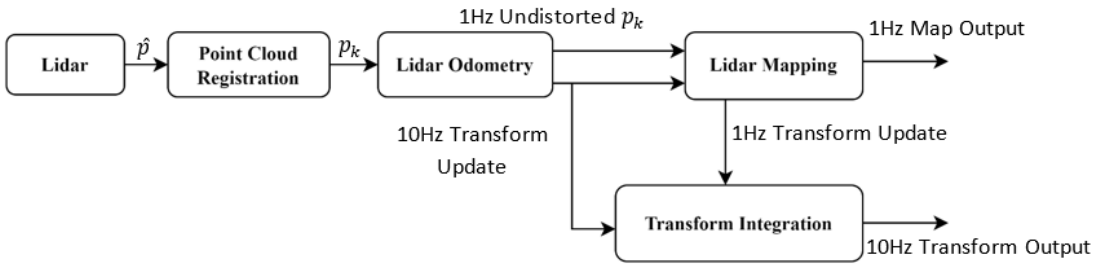


Figure 10: System overview of the LOAM [11]

The system overview of the LOAM is shown in Fig. 10, the lidar odometry is composed of four parts. First, the point cloud data $\hat{P}$ in the lidar coordinate system is obtained, and then the point cloud obtained by the kth scan is composed of frame data P_k. Finally P_k is processed in the lidar odometry and lidar mapping nodes. The lidar odometry mainly acquires the motion between two frames of point cloud data to

achieve the function of de-distortion. The operating frequency of this node is 10 Hz, and then the processed result is passed to the lidar mapping node, which is 1 Hz frequency work to match the map and register point cloud data without distortion. The last transform integration node receives the transform information of the first two nodes and performs information fusion processing at a frequency of 10 Hz [11].

First, in the part of feature point extraction, non-uniformly distributed point cloud data is generated by constructing two lidars, so feature points are extracted from each scan. The selection of feature points is shown by the curvature Eq. (12), and the surrounding points of curvature points make up a set of consecutive points used to find the curvature. The square of the five-point differences around the feature point is calculated to compare the curvature. To prevent the feature points from being too dense, the point cloud obtained by each scan is divided into four parts. In each part, two points with the most considerable curvature are selected as edge points and four with the minor curvature are selected as plane points. When selecting feature points, it should avoid to choose and select points around the points and points on a plane parallel to the laser line as shown in Fig. 11(a), and should not determine the blocked points in Fig. 11(b) [11].

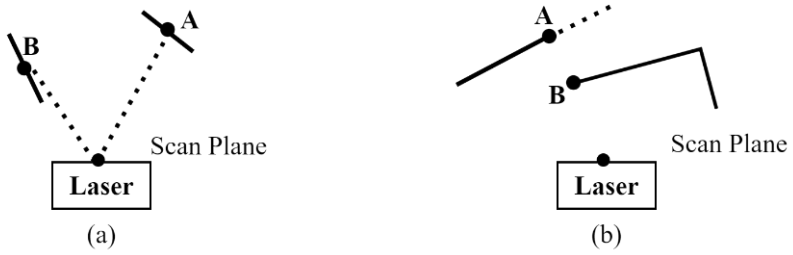


Figure 11: (a) Point B on a surface patch that is parallel to the laser beam, treat point B as an unreliable point, (b) Point A on a surface patch is blocked with point B scan plane, treat point A as unreliable point

$$C=\frac{1}{|S|\cdot\left\|X_{(k,i)}^{L}\right\|}\left\|\sum_{j\in S,j\neq 1}(X_{(k,i)}^{L}-X_{(k,j)}^{L})\right\|. \quad (12)$$

$|S|$ represents the set of continuous point of p from the same range image row. After the curvature is calculated, the operation of picking points is carried out to remove the unreliable points, and if the distance between the two points is close, the 5 points that are farther away are directly removed.

The next step is to find the correspondence of the feature points. The mileage calculation method obtains the motion in a frame of point cloud time, and the time t is used to represent the start time of the kth scan. At the end of a scan, there will be a point cloud P_k, and the projected point cloud $\overline{P_k}$ and at time $t+1$, P_{k+1} is matched to obtain the relative motion and the mutual correspondence. The relationship between the two is by first finding the two closest points in the previous frame of data corresponding to an edge point i and judging whether these two points are edge points, j and l must be pointed in different bundles to ensure that there is at most one edge point in a segment of a line, as shown in Fig. 12. Then we can find the distance from the point to the line as Eq. (13) [11].

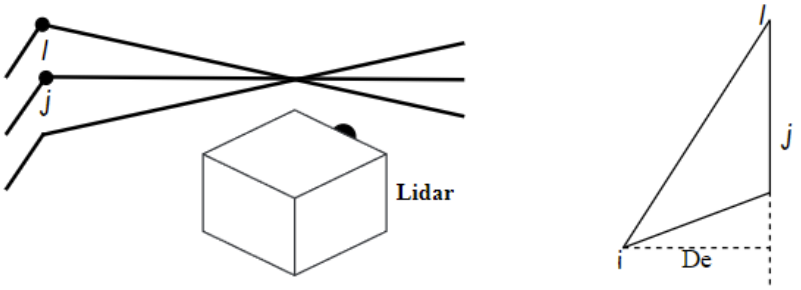


Figure 12: Corresponding corner feature

$$De=\frac{\left|(\tilde{X}_{(k+1,i)}^{L}-\tilde{X}_{(k,j)}^{L})\times(\tilde{X}_{(k+1,i)}^{L}-\tilde{X}_{(k,l)}^{L})\right|}{\left|\tilde{X}_{(k,j)}^{L}-\tilde{X}_{(k,l)}^{L}\right|}. \quad (13)$$

Here $\tilde{X}^L_{(k,i)}$, $\bar{X}^L_{(k,j)}$ and $\bar{X}^L_{(k,l)}$ form the corresponding edge line. The numerator is two loud cross products. After the modulus is calculated, it becomes the area of the triangle. As shown in Fig. 11, the denominator is the vector whose modulus is equivalent to the length of the triangle's base.

Using a similar principle, we can obtain the point-to-surface distance as shown in Fig. 13 and Eq. (14).

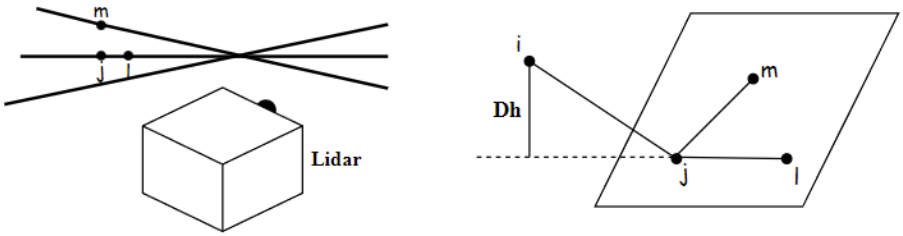


Figure 13: Corresponding surface feature

$$Dh = \frac{\left| (\bar{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}) \right|}{\left| (\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}) \times (\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,m)}) \right|}, \tag{14}$$

where i, j, l and m represent the feature index in the corresponding sets, $\tilde{X}^L_{(k,i)}$, $\bar{X}^L_{(k,j)}$, $\bar{X}^L_{(k,l)}$ and $\bar{X}^L_{(k,m)}$ form the corresponding planar patch. The lower part of the numerator represents the area of the triangle obtained by the cross-multiplication of two vectors. The upper part of the numerator represents the height of the cube, the whole numerator represents the volume of the cube. The denominator represents the triangle area of the cube's base, then divides the two. The final result is the distance from the point to the surface.

The third part is motion estimation. In this paper, it is assumed that the lidar is moving at a constant speed. Therefore, through the transformation matrix of the endpoint of a frame of data relative to the start point, any point in this data frame can be obtained by

time interpolation close to the start point. The interpolation formula is shown in Eq. (15) [11].

$$T_{(k+1,i)}^{L} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T_{k+1}^{L}. \tag{15}$$

T_{k+1}^{L} is lidar pose transform from t_{k+1} to t, and it contains rigid motion of the lidar in 6 DOF. t_i is the timestamp with a given point i.

In order to obtain the correspondence between the points in the data of this frame and the previous frame, Eq. (16) is represented by a rotation matrix R and a translation matrix T. Since the derivation of the rotation matrix is relatively complicated, the rotation matrix is expanded by the Rodrigues formula as shown in Eq. (17). After derivation of the rotation matrix, we can get the distance from a point to a line and point to the surface [11].

$$X_{(k+1,i)}^{L} = R\tilde{X}_{(k+1,i)}^{L} + T_{(k+1,i)}^{L}(1:3), \tag{16}$$

$$R = e^{\hat{\omega}\theta} = I + \hat{\omega}\sin\theta + \hat{\omega}^2(1 - \cos\theta), \tag{17}$$

where $X_{(k+1,i)}^{L}$ is coordinates of a point I in sets of edge points and planar points extracted from the point cloud received during the sweep. $\tilde{X}_{(k+1,i)}^{L}$ is the corresponding points in set of points reprojected to the beginning of the sweep. $T_{(k+1,i)}^{L}(1:3)$ is first to third entries of $T_{(k+1,i)}^{L}$ and R is a rotation matrix. θ is the magnitude of the rotation. $\hat{\omega}$ is the skew symmetric matrix of ω, and ω is a unit vector representing the rotation direction.

Through the motion of the lidar and the LM method, a non linear error function (18) for optimization can be obtained. Each row in f represents a feature point, and the Jacobian matrix is calculated as Eq. (19), and finally the nonlinear iterative method is used. Let d become 0 [11].

$$f(T_{k+1}^{L}) = d, \tag{18}$$

$$T_{k+1}^{L} \leftarrow T_{k+1}^{L} - (J^T J + \lambda diag(J^T J)^{-1} J^T d), \tag{19}$$

where $J = \partial f / \partial T_{k+1}^{L}$, λ is a Levenberg Marquardt method factor.

The last part is the lidar mapping part, as shown in Fig. 14. This is the mapping process. The curve in figure represents the lidar pose on the map. T_k^W is generated by the mapping algorithm at sweep k, and T_{k+1}^{L} indicates the lidar motion during sweep $k+1$ which is computed by the lidar odometry algorithm. The existing point cloud on the map Q_k is used to match the undistorted point cloud which is published by the odometry algorithm is projected on the map indicated as $\bar{Q}_{k+1}$ [11].

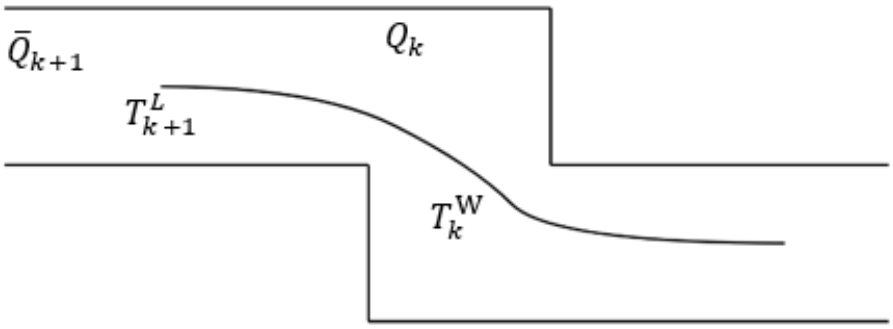


Figure 14: Lidar mapping process

To calculate the distance from the feature point to the corresponding point, the distance from a point to line and point to the surface can be obtained as shown in Equations (13-14), and then Equations (20-21) can be derived. The difference is that there are many advantages in $\bar{Q}_{k+1}$. At the same timestamp t, the nonlinear optimization is solved again by LM optimization, and $\bar{Q}_{k+1}$ is registered on the map. These points are evenly distributed by reducing the point cloud size in the map by a voxel grid filter [11].

$$f_e\left(X_{(k+1,i)}^{L}, T_{k+1}^{L}\right) = D_e, i \epsilon e_{k+1} \tag{20}$$

$$f_h\left(X_{(k+1,i)}^L, T_{k+1}^L\right) = D_h, i \epsilon h_{k+1} \quad (21)$$

e_{k+1} and h_{k+1} are the sets of edge points and planar points. $f_e(.)$ is geometric relationship between an edge point in e_{k+1} and the corresponding edge line, $f_h(.)$ is geometric relationship between a planar point in h_{k+1} and the corresponding planar patch.

As shown in Fig. 15, this is the integration of pose transforms. T_k^W represents the pose output of the lidar mapping, area T_{k+1}^L represents the conversion output of the lidar odometry when the frequency is about 10 Hz. The lidar pose is relative to the map which combines two transformations, at the same frequency as the lidar odometry [11].

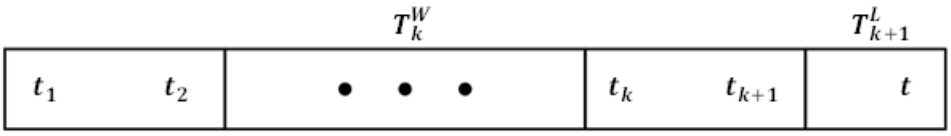


Figure 15: Integration of pose transforms

LOAM can be said to be the most classic framework in SLAM. It has driven the development of the entire lidar SLAM after the LOAM is proposed. In this situation, we are usually referred to a series of algorithm frameworks based on LOAM as LOAM-based algorithms.

CHAPTER 3 ARCHITECTURE IMPROVEMENTS

This chapter will introduce the improvement made in the proposed lidar slam framework and the background and rationale of these changes. The changes are mainly divided into five parts: image projection and feature association from front end of the LeGO-LOAM, using iterated error state Kalman filter of the IMU pre-integration in lidar odometry, ikd-Tree method to replace the k-d tree method in lidar mapping, and the addition of the scan context method to the loop closure detection in pose optimization. The proposed algorithm can generate a better trajectory from the environment without large odometry errors and drifts. A detailed analysis of the improvements will be shown below.

3.1 Image Projection

This part is mainly for point cloud segmentation. First, the starting point and the last point are angle converted, the depth of the point cloud is calculated one by one, then saved the point cloud with depth. Different scanning circles determine whether it is a ground point to confirm the ground is level. Finally, the point cloud segmentation operation is performed. If it is recognized as a feature point and a ground point, then include a point cloud every 5 points to determine whether it is a ground point and saved to the point cloud. The unprocessed point of the feature detects the four adjacent points up, down, left and right. The local feature includes the adjacent point when the plane angle is more significant than sixty degrees. When the number of neighboring points reaches 30, the registration is successful because the robot operates in a noisy environment; small objects can form small and unreliable features. After that, the registered segmentation point cloud is published [12].

3.2 Feature Association

The main body of this section is divided into two parts: feature point extraction and feature point matching. First, the down-sampling filter and the coordinate system transformation of the reference LOAM [11] are defined to obtain the world coordinate system's distortion displacement and speed. At the same time, the displacement distortion is removed. The displacement, velocity and rotation are corrected [12].

In the feature point matching, the shift distortion calculation, as Equations (22-24), is the shift distortion in the world frame [11]. α, β, γ are pitch, yaw and roll angles, and $\Delta S_{world_{sta2cur_{x/y/z}}}$ represents shift from the start to current in x, y and z axis in the world frame.

$$\Delta S_{world_{sta2cur_{x/y/z}}} = S_{cur\_x/y/z} - S_{sta\_x/y/z} - V_{x/y/z}\Delta t,$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}^T = \begin{bmatrix} \Delta S_x \\ \Delta S_y \\ \Delta S_z \end{bmatrix}^T R_y(\beta) = \begin{bmatrix} \Delta S_{world_x} \\ \Delta S_{world_y} \\ \Delta S_{world_z} \end{bmatrix}^T \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix}, \tag{22}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}^T = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}^T R_x(\alpha) = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix}, \tag{23}$$

$$\begin{bmatrix} \Delta S_{local\_x} \\ \Delta S_{local\_y} \\ \Delta S_{local\_z} \end{bmatrix}^T = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}^T R_z(\gamma) = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}^T \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{24}$$

Eq. (25) shows the shift distortion in the local frame, ΔS_{local} represents the shift in the local frame:

$$\begin{bmatrix} \Delta S_{local\_x} \\ \Delta S_{local\_y} \\ \Delta S_{local\_z} \end{bmatrix}^T = \begin{bmatrix} \Delta S_{world_{sta2cur_x}} \\ \Delta S_{world_{sta2cur_x}} \\ \Delta S_{world_{sta2cur_x}} \end{bmatrix}^T R_y(\beta)R_x(\alpha)R_z(\gamma). \tag{25}$$

Same as the shift distortion calculation, the velocity distortion in the world coordinate system is as in Eq. (26), and in the local coordinate system as Eq. (27) shows [11].

$$\Delta V_{world_{sta2cur\_x/y/z}} = V_{cur\_x/y/z} - V_{sta\_x/y/z}, \tag{26}$$

$$\begin{bmatrix} \Delta V_{local\_x} \\ \Delta V_{local\_y} \\ \Delta V_{local\_z} \end{bmatrix}^T = \begin{bmatrix} \Delta V_{world_{sta2cur_x}} \\ \Delta V_{world_{sta2cur_y}} \\ \Delta V_{world_{sta2cur_z}} \end{bmatrix}^T R_y(\beta) R_x(\alpha) R_z(\gamma). \tag{27}$$

After the shift distortion is calculated, we need to remove the shift distortion as following operations. First, the current coordinate system needs to be converted to the world coordinate system as shown in Eq. (28). The world coordinate system is converted to the initial coordinate system to subtract the amount of shift distortion as Eq. (29) shows [11].

$$\begin{bmatrix} S_{world\_x} \\ S_{world\_y} \\ S_{world\_z} \end{bmatrix}^T = \begin{bmatrix} S_{local\_cur\_x} \\ S_{local\_cur\_y} \\ S_{local\_cur\_z} \end{bmatrix}^T R_z(\gamma)^{-1} R_x(\alpha)^{-1} R_y(\beta)^{-1}, \tag{28}$$

$$\begin{bmatrix} S_{local\_sta\_x} \\ S_{local\_sta\_y} \\ S_{local\_sta\_z} \end{bmatrix}^T = \begin{bmatrix} S_{world_x} \\ S_{world_y} \\ S_{world_z} \end{bmatrix}^T R_y(\beta) R_x(\alpha) R_z(\gamma) + \begin{bmatrix} \Delta S_{world_{sta2cur_x}} \\ \Delta S_{world_{sta2cur_y}} \\ \Delta S_{world_{sta2cur_z}} \end{bmatrix}^T. \tag{29}$$

After completing the above calculations, the shift, velocity and angular shift will be corrected. Then convert the current coordinate system to the world coordinate system and subtract the shift distortion value as Eq. (30) shows. In Eq. (31), when the last frame minus the time difference of the previous frame is less than the scan period, the correction Eq. (32) will be performed, when Δt is smaller than the scan period [11].

$$\begin{bmatrix} a_{world\_x} \\ a_{world\_y} \\ a_{world\_z} \end{bmatrix}^T = \begin{bmatrix} a_{local\_x} \\ a_{local\_y} \\ a_{local\_z} \end{bmatrix}^T R_z(\gamma)^{-1} R_x(\alpha)^{-1} R_y(\beta)^{-1}, \tag{30}$$

$$\Delta t = t_{last} - t_{back}, \tag{31}$$

$$\begin{aligned} S_{last\_x/y/z} &= S_{back\_x/y/z} + V_{back\_x/y/z}\Delta t + \frac{1}{2} a_{back\_x/y/z}\Delta t^2, \\ V_{last\_x/y/z} &= V_{back\_x/y/z} + a_{back\_x/y/z}\Delta t, \\ \theta_{last\_x/y/z} &= \theta_{back\_x/y/z} + V_{back\_x/y/z}\Delta t, \end{aligned} \tag{32}$$

where θ represents the angular rotation and V represents the angular velocity. After subtracting the influence of gravity, the actual value of acceleration in the xyz direction is obtained, and the coordinate axis conversion is performed to unify it to the right-hand coordinate system with the z-axis forward and the x-axis to the right. After completion, it enters the stage of feature extraction. This stage mainly calculates the relative pose and time difference between the point clouds, calculates the relative movement speed, and converts the point clouds to the IMU coordinate system [12].

Then calculate the linear interpolation that are Rf and Rb which are Equations (33-34), using IMU to correct point cloud distortion by Eq. (35).

$$Rf = \frac{T_{scan\_cur} + T_{point} - T_{imu\_point\_back}}{T_{imu\_point\_front} - T_{imu\_point\_back}}, \tag{33}$$

$$Rb = \frac{T_{imu\_point\_front} - T_{scan_{cur}} - T_{point}}{T_{imu\_point\_front} - T_{imu\_point\_back}}, \tag{34}$$

$$\begin{aligned} \theta_{imu\_x/y/z\_cur} &= \theta_{imu\_x/y/z\_front} Rf + \theta_{imu\_x/y/z\_back} Rb, \\ S_{imu\_cur\_x/y/z} &= S_{imu\_x/y/z\_front} Rf + S_{imu\_x/y/z\_back} Rb, \\ V_{imu\_cur\_x/y/z} &= V_{imu\_x/y/z\_front} Rf + V_{imu\_x/y/z\_back} Rb. \end{aligned} \tag{35}$$

Then calculate the curvature as shown in Eq. (12) to express the smoothness of the plane. After that, the feature point extraction process is carried out. First, the current point cloud frame is divided into six parts according to the line bundle to prevent the feature extraction from being too dense, and then arranged according to the curvature from small to large. When extracting a feature point, the curvature must be greater than or less than a certain threshold to prevent it from being a ground point. Afterward, 20 points with more significant curvature are extracted as corner points, and 20 with smaller curvature are removed as plane points [12].

After the feature extraction stage, the feature points are matched and associated. There are two main matching methods here. The first is surface feature matching. As shown

in Fig. 12, the surface feature points are first projected to the starting coordinate system of the frame. Next, look for the nearest neighbor point every five iterations in the kd tree by searching through voxel filtering. Find the other two points to form a plane in the direction where the scan id increases or decreases. Then the distance Dh from the point to the surface can be obtained [11], as shown in Eq. (14).

The second part is line feature matching. The principle is similar to surface features. As shown in Fig. 11, the distance from a point to line De can be obtained by Eq. (13). After the feature extraction and matching are completed, the surface and line features are matched and optimized. An accumulated rotation amount will be shown in Eq. (36).

$$R_{cur}^{start} = R_{last}^{start} * (R_{last}^{cur})^{-1}, \tag{36}$$

Notice that $R_{last}^{start} = R_z R_x R_y$ $R_{cur}^{last} = R_y R_x R_z$.

Then correct the Euler angle of the current point, as shown in Eq. (37).

$$(R_{cur}^{start})' = R_{end} R_{start}^{-1} R_{cur}^{start}, \tag{37}$$

$R_{end}\ and\ R_{start}^{-1}$ are rotated with zxy axis.

Then the feature point extraction and feature point matching is done. It is the front end of the LeGO-LOAM which is used in the proposed algorithm.

3.3 Lidar Odometry (Iterated ESKF Method)

The IMU pre-integration process mainly occurs in the odometry module, primarily composed of propagation and update sub-modules. It performs an iterative Kalman filter [33] and outputs the initial odometry and undistorted features. Then it uses the mapping module to initialize the global map and output the new odometry to update the map with the latest features [25].

The lidar inertial odometry module has the iterated ESKF method that uses the IMU to measure and extract features from two consecutive scans to estimate the relative transformation of the vehicle. The iterated ESKF method is constructed using a robot-centric formulation, which allows for the placement of linearization errors due to the accumulation of uncertainties [34]. Then the state equation as shown in Eq. (38) can be created.

$$\mathrm{x}_{w}^{b_k} = \left[\mathrm{p}_{w}^{b_k}, \mathrm{q}_{w}^{b_k}\right],$$

$$\mathrm{x}_{b_{k+1}}^{b_k} = \left[\mathrm{p}_{b_{k+1}}^{b_k}, \mathrm{v}_{b_{k+1}}^{b_k}, \mathrm{q}_{b_{k+1}}^{b_k}, \mathrm{b}_a, \mathrm{b}_g, \mathrm{g}^{b_k}\right]. \tag{38}$$

Let F_w represent the fixed world frame and F_{b_k} represent the IMU-affixed frame at lidar time step k. From the Eq. (38), $\mathrm{x}_{w}^{b_k}$ and $\mathrm{p}_{w}^{b_k}$ are the location and position of F_w with respect to F_{b_k}, $\mathrm{x}_{b_{k+1}}^{b_k}$ is the local relative transformation state from $F_{b_{k+1}}$ to F_{b_k}. $\mathrm{q}_{w}^{b_k}$ is the unit quaternion represent the rotation from F_w to F_{b_k}. $\mathrm{p}_{b_{k+1}}^{b_k}$ and $\mathrm{q}_{b_{k+1}}^{b_k}$ are the translation and rotation from $F_{b_{k+1}}$ to F_{b_k}. $\mathrm{v}_{b_{k+1}}^{b_k}$ is velocity, b_a and b_g are acceleration and gyroscope bias. And the final component g^{b_k} is part of the local state [25].

In order to make the state estimation have good characteristics, an error state is used to solve $\mathrm{x}_{b_{k+1}}^{b_k}$, and δ is used to represent the error term as Eq. (39) shows. $\delta\theta$ is a 3 DOF error angle, based on the ESKF traditions, once $\delta\mathrm{x}$ is solved, the final $\mathrm{x}_{b_{k+1}}^{b_k}$ can be obtained by injecting $\delta\mathrm{x}$ into the state prior of $\mathrm{x}_{b_{k+1}}^{b_k}$ [25].

$$\delta\mathrm{x} = \left[\delta\mathrm{p}, \delta\mathrm{v}, \delta\theta, \delta\mathrm{b}_a, \delta\mathrm{b}_\mathrm{g}, \delta\mathrm{g}\right]. \tag{39}$$

When new IMU measurement data is received, we propagate the error state, the error state covariance matrix and the state prior. We can obtain a linearized continuous-time model of the IMU error state as shown in Eq. (40) [25].

$$\delta\dot{\mathrm{x}}(t) = \mathrm{F}_t\delta\mathrm{x}(t) + \mathrm{G}_t\mathrm{w}, \tag{40}$$

where $w = \left[n_a^T, n_g^T, n_{b_a}^T, n_{b_g}^T\right]^T$ is Gaussian vector. The error state transition matrix F_t and noise Jacobian G_t at time t is shown below:

$$F_t = \begin{bmatrix} 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & -R_t^{b_k}[\hat{a}_t]_\times & -R_t^{b_k} & 0 & 0 \\ 0 & 0 & -[\widehat{\omega}_t]_\times & 0 & -I_3 & -I_3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -R_t^{b_k} & 0 & 0 & 0 \\ 0 & -I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

I_3 is the identity matrix and $R_t^{b_k}$ is the rotation matrix from the IMU-affixed frame at time t. The acceleration and angular rate at time t are $\hat{a}_t$ and $\widehat{\omega}_t$, then we can discretizing Eq. (40) to get propagation Eq. (41). $\Delta t = t_\tau - t_{\tau-1}$, t_τ and $t_{\tau-1}$ represent IMU time steps, Q is the covariance matrix of w [25]. Refer to [35], the robocentric state that uses discrete time propagation model can predict the state prior.

$$\begin{aligned} \delta\mathrm{x}_{t_\tau} &= \left(\mathrm{I} + \mathrm{F}_{t_\tau}\Delta t\right)\delta\mathrm{x}_{t_{\tau-1}} \\ \mathrm{P}_{t_\tau} &= \left(\mathrm{I} + \mathrm{F}_{t_\tau}\Delta t\right)\mathrm{P}_{t_{\tau-1}}\left(\mathrm{I} + \mathrm{F}_{t_\tau}\Delta t\right)^T + \left(\mathrm{G}_{t_\tau}\Delta t\right)\mathrm{Q}\left(\mathrm{G}_{t_\tau}\Delta t\right)^T \end{aligned} \tag{41}$$

After getting the above equations, we optimize it through an iterative update method. In the optimization problem in iterative Kalman filter state update [34], the model Eq. (42) [25] can be derived by analyzing the prior state derivative and residual function $f(.)$.

$$\min_{\delta\mathrm{x}} \|\delta\mathrm{x}\|_{(P_k)^{-1}} + \left\| f\left(-\mathrm{x}_{b_{k+1}}^{b_k} \boxplus \delta\mathrm{x}\right) \right\|_{(\mathrm{J}_k\mathrm{M}_k\mathrm{J}_k^T)^{-1}}. \tag{42}$$

The function output in f is the stacked residual vector calculated from point-to-surface or point-to-edge. J_k is the Jacobian of $f(.)$ with respect to measurement noise and M_k is the covariance matrix of the measurement noise. Given $\mathrm{x}_{b_{k+1}}^{b_k}$, we can get the error term (43) of f in the case of the ith feature point [25].

$$f_i\left(x_{b_{k+1}}^{b_k}\right)=\begin{cases}\dfrac{\left|\left(\hat{p}_i^{l_k}-p_a^{l_k}\right)\times\left(\hat{p}_i^{l_k}-p_b^{l_k}\right)\right|}{\left|p_a^{l_k}-p_b^{l_k}\right|} & if\ p_i^{l_{k+1}}\in\mathbb{F}_e,\\ \dfrac{\left|\left(\hat{p}_i^{l_k}-p_a^{l_k}\right)^T\left(\left(p_a^{l_k}-p_b^{l_k}\right)\times\left(p_a^{l_k}-p_c^{l_k}\right)\right)\right|}{\left|\left(p_a^{l_k}-p_b^{l_k}\right)\times\left(p_a^{l_k}-p_c^{l_k}\right)\right|} & if p_i^{l_{k+1}}\in\mathbb{F}_p,\end{cases} \tag{43}$$

where $\hat{p}_i^{l_k}=R_l^{b^T}\left(R_{b_{k+1}}^{b_k}\left(R_l^b p_i^{l_{k+1}}+p_l^b\right)+p_{b_{k+1}}^{b_k}-p_l^b\right)$. It is similar to Eq. (13) and (14). $\hat{p}_i^{l_k}$ is the transformed point of $p_i^{l_{k+1}}$, p_l^b and R_l^b represent the extrinsic parameter between the IMU and lidar. Then we can use iterated updated Eq. (44) to solve Eq. (42):

$$\begin{aligned}\mathrm{K}_{k,j}&=\mathrm{P}_k\mathrm{H}_{k,j}^T\left(\mathrm{H}_{k,j}\mathrm{P}_k\mathrm{H}_{k,j}^T+\mathrm{J}_{k,j}\mathrm{M}_k\mathrm{J}_{k,j}^T\right)^{-1},\\ \Delta x_j&=\mathrm{K}_{k,j}\left(\mathrm{H}_{k,j}\delta x_j-f\left(-x_{b_{k+1}}^{b_k}\boxplus\delta\mathrm{x}_j\right)\right),\\ \Delta x_{j+1}&=\delta x_j+\Delta x_j.\end{aligned} \tag{44}$$

Δx_j is the correction vector at j_{th} iteration. H here is the Jacobian matrix of $f(.)$ with respect to δx_j. In every iteration, new matched edges and planes will be found to minimize the error metric. We can update the error state covariance matrix P_k (45) when $f\left(x_{b_{k+1}}^{b_k}\right)$ is below a certain threshold [25].

$$\mathrm{P}_{k+1}=\left(I-\mathrm{K}_{k,n}\mathrm{H}_{k,n}\right)\mathrm{P}_k\left(I-\mathrm{K}_{k,n}\mathrm{H}_{k,n}\right)^T+\mathrm{K}_{k,n}\mathrm{M}_k\mathrm{K}_{k,n}^T. \tag{45}$$

Then we can obtain the final $x_{b_{k+1}}^{b_k}$, we can initialize the next state $x_{b_{k+2}}^{b_{k+1}}$ with $\left[0_3,v_{b_{k+2}}^{b_{k+1}},q_o,b_a,b_g,g^{b_{k+1}}\right]$ after using the estimated relative transformation which make the raw features from distorted to undistorted. q_o represent the identity quaternion, $v_{b_{k+1}}^{b_{k+1}}$ and $g^{b_{k+1}}$ can be calculated by $v_{b_{k+1}}^{b_{k+1}}=R_{b_k}^{b_{k+1}}v_{b_{k+1}}^{b_k}$ and $g^{b_{k+1}}=R_{b_k}^{b_{k+1}}g^{b_k}$. Since the robot-centered reference frame itself has no uncertainty, the covariance matrix still has covariances for the velocity, bias and local gravity, and

the covariance corresponding to the relative position is set to 0. Each time the update is done, the global pose can be updated through the synthesis step as in Eq. (46) [25].

$$x_w^{b_{k+1}} = \begin{bmatrix} p_w^{b_{k+1}} \\ q_w^{b_{k+1}} \end{bmatrix} = \begin{bmatrix} R_{b_k}^{b_{k+1}}(p_w^{b_k} - p_{b_{k+1}}^{b_k}) \\ q_{b_k}^{b_{k+1}} \otimes q_w^{b_k} \end{bmatrix}. \tag{46}$$

All of the above performed calculations and formulations are primarily intended to facilitate the initialization of the filter state. The initial acceleration bias and the external parameters of the lidar-IMU can be obtained through offline calibration. The initial roll and pitch angles can be obtained from the unbiased acceleration measurement before moving, and the initial local gravity can be obtained through coordinate system transformation. This completes the initialization process for this part [25].

3.4 Lidar Mapping (Incremental KD Tree)

Building the original k-d tree is inefficient and time-consuming from scratching all the points. Compared to the static k-d tree [29], ikd-Tree only uses new future points to update the k-d tree incrementally. The redundant operation of rebuilding the whole tree can be eliminated and the computation time is shorter. And ikd-Tree will actively monitor the tree structure and partially rebalance the tree to effectively perform the nearest point search to maximize the overall efficiency [23]. The detailed substance of ikd-Tree will be introduced below.

A k-d tree is treated as a binary search tree and inherits the same increment operation. Generally, two schemes are designed for a fast rebalancing of k-d trees: hardware-based acceleration and specially designed structures to support dynamic rebalancing. Previously, single-core and multi-core CPUs [27-28] and real-time building algorithms on GPUs were firstly proposed. Then, converting a static k-d tree to a

dynamic tree increases the recent search time. ikd-Tree is an efficient and complete data structure that simultaneously supports point deletion and downsampling as incremental operations. It only builds unbalanced subtrees, which significantly saves running time. Such data structures are very suitable for robotics, such as SLAM, motion planning, etc. [23].

The data structure of the ikd-Tree has the general properties of a standard k-d tree, and its left and right nodes are represented by two pointers respectively. The point information is then stored in the point, and the rest is incrementally updated to design new properties [26]. When building an incremental k-d tree, the approach is similar to building a static k-d tree. The only difference is that additional information is required to maintain cumulative updates. The construction process is mainly constructing a point array, sorting with the most significant covariance, saving the intermediate point to the new binary tree node, and recursively constructing the left and right child nodes to satisfy all the required incremental updates [23].

Incremental update operations mainly include inserting, deleting, or re-inserting points in the k-d tree. The insertion operation is to append a new point to the k-d tree, while the delete operation is a lazy deletion, the deleted point is not immediately removed from the tree but marked as deleted and later inserted to the tree is referred as re-insertion. Incremental update support is divided into point-wise updates and box-wise updates. A point-wise update inserts, deletes, or reinserts a single point on the tree, while a box-wise update inserts, deletes, or reinserts all points in a given box aligned with the data axis. Because the lazy delete method is still very inefficient when using recursive updates, a strategy is dropped to update the lazy labels of descendant nodes [23].

In this algorithm, there are two support functions for updating the attributes on the tree node. The two functions represent deleting and copying to its child nodes and saving all the nodes on the word count to the corresponding points. The point-wise update is implemented recursively. The algorithm searches downwards from the root node and appends a new tree node after comparing the coordinates on the division axis of the new point with the points stored on the tree node. Box-wise update is achieved by inserting new points into an incremental k-d tree. Given a box of point C_O updated on the root number, first-pass its lazy label to its child nodes, then recursively search the k-d tree from the root node to check whether the current node has an intersection with C_O. If there is no intersection, recursively return directly to the update tree, delete if included in C_O. If intersected but not included in C_O, all attributes of the current node are updated [23].

After completing the above incremental update operation, ikd-Tree will further downsample. Given a point and downsampling resolution, the algorithm will divide the space into cubes with a resolution length. By storing the point and all points in the box containing the point in an array, the point closest to the center of the box is kept. Then delete the existing points in the box and insert the nearest point into the k-d tree to downsample the point cloud to reduce the number of point clouds, as shown in Fig. 16 [23].

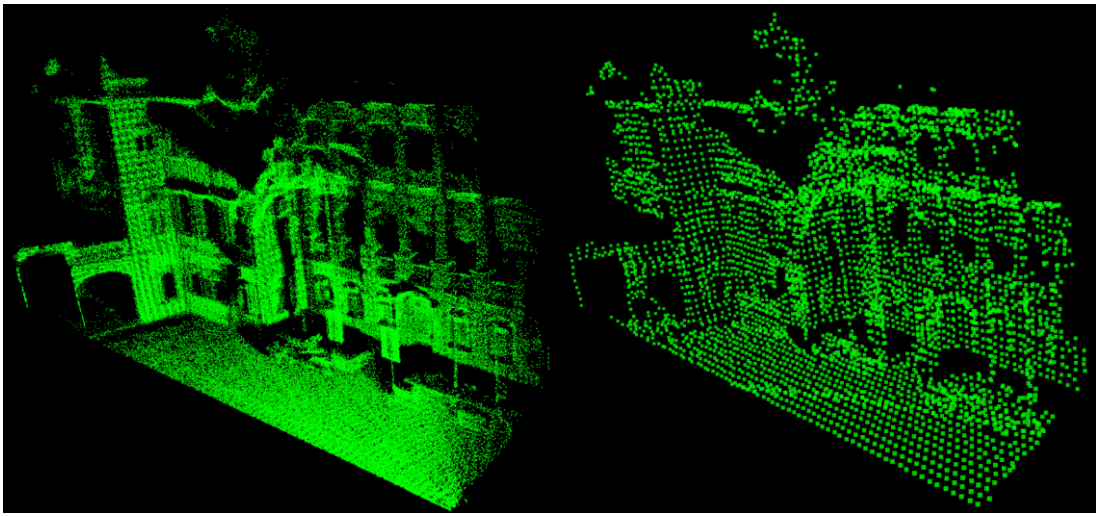


Figure 16: Point cloud downsample from left to right

Table I: Comparison of supported incremental updates

		Static K-D Tree	Dynamic K-D Tree	Scapegoat K-D Tree	ikd-Tree
Point-wise	Insert	X	√	√	√
	Delete	X	X	√	√
	Re-insert	X	X	X	√
Box-wise	Insert	X	√	√	√
	Delete	X	X	X	√
	Re-insert	X	X	X	√
Downsample		X	X	X	√

As Table I mentioned in the paper [23], compared with other types of k-d trees, the ikd-Tree has all the functions and that's reason why use the ikd-Tree method in the proposed algorithm.

Finally, there is the process of rebalancing, which needs to be rebalanced by partially rebuilding the dynamics. The first is that its equilibrium criterion consists of two sub-criteria, the α-balanced criterion and α-deleted criterion. Then it should be satisfied with the following condition [23]:

$$S(T.leftson) < \alpha_{bal}(S(T) - 1),$$

$$S(T.rightson) < \alpha_{bal}(S(T) - 1),$$

where $\alpha_{bal}\epsilon(0.5,1)$ and $S(T)$ is the tree size attribute of the node T [23].

The α-deleted criterion of the sub-tree rooted at T (47):

$$I(T) < \alpha_{del}S(T), \tag{47}$$

where $\alpha_{del}\epsilon(0,1)$ and $I(T)$ represents the number of invalid nodes on the subtree [23].

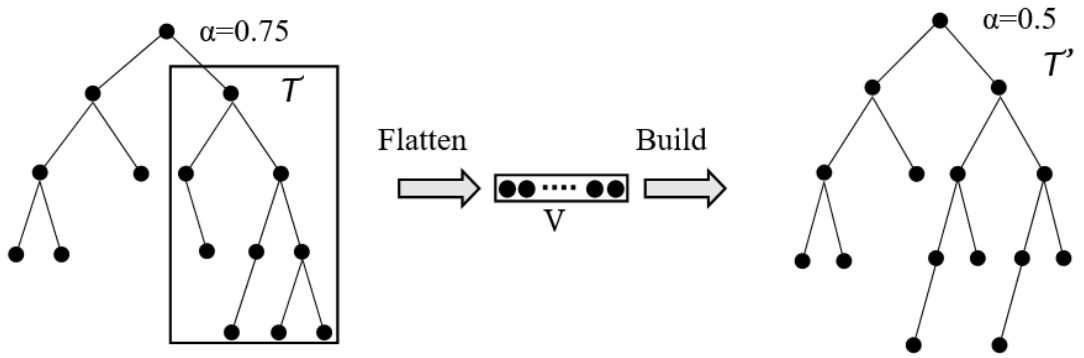


Figure 17: Rebuild unbalanced sub-tree steps

If the incremental k-d tree satisfies the above two conditions, the subtree is balanced, and the balance of all subtrees means that the entire tree is balanced. When a condition is not met, a rebuilding process is triggered to rebalance the subtree. The reconstruction process is to flatten the subtree into a point storage array, and then rebuild a new perfectly balanced k-d tree as shown in Fig. 17 [23].

When rebuilding a large subtree on an incremental k-d tree, it is observed that the computing power decreases significantly. In order to maintain the efficiency of the algorithm, a two-thread reconstruction method is proposed. The main thread rebuilds subtrees smaller than a predetermined threshold, and the second thread rebuilds the remaining subtrees. The algorithm of locking nodes can avoid information loss and memory conflicts between the main thread and the second thread [23].

Finally, the nearest neighbor search [30] is done, which is an exact nearest search instead of an approximate search on an incremental kd tree [23]. After satisfying the above algorithms and functions, the principle implementation of ikd-Tree is completed.

3.5 Loop Closure (Scan Context Method)

In SLAM, position recognition is a significant problem. Although the closed-loop is very important for robot navigation, the perception aliasing situation will result in a significant error due to some environmental influences. In the proposed algorithm, the scan context method, a non-histogram based global descriptor from 3D light detection and ranging scans is used. The method can record the 3D structure of the visible space directly from the sensor without relying on the histogram, and effectively detect loops by using the similarity score to calculate the distance between contexts [24].

Scan Context method is another novel spatial descriptor with matching algorithms specifically for outdoor locations using a single 3D scan. Its main contribution is that it has an efficient bin encoding function, which is invariant to the density and norm of the point cloud. Scan Context can preserve the internal structure of the point cloud. It can avoid the loss of absolute position information of points like using histogram, thus improving the discriminative ability, and can detect reverse cycle by using scan context. This method can also provide a rotation-invariant sub-descriptor for the first nearest neighbor search to avoid searching all databases during loop detection, which reduces efficiency [24].

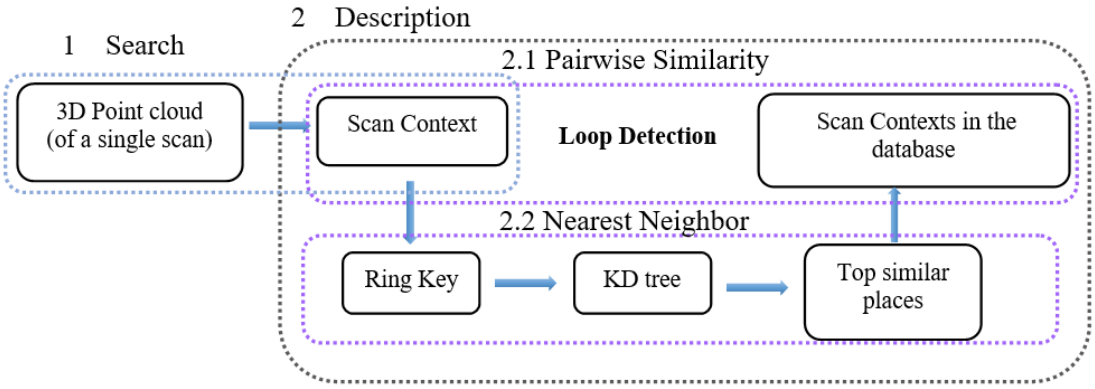


Figure 18: Scan context algorithm overview

As shown in Fig. 18, it is the finishing process for location recognition using scan context, a measure of the distance between two scan contexts is calculated by creating a scan context given a point cloud from a 3D scan. The scanning context uses the method of the maximum point height in each bin to summarize the vertical shape of the surrounding structures, which does not require extensive computation to analyze point cloud features. First, the 3D scan is divided into azimuthal and radial bins in sensor coordinates equidistant manner. The scanning center acts as a global key point, and N_S and N_r are used to represent the number of sectors and rings. If the maximum sensing range of the lidar sensor is set to L, then the radial gap between the rings is $\frac{L}{N_r}$, and the central angle of the sector is $\frac{2\pi}{N_S}$ [24].

The first process of making a scan context is to divide the entire point of the 3D scan into mutually exclusive point clouds [31]. Because the point cloud is divided at fixed intervals, bins farther from the sensor have a more comprehensive physical area than close bins, and both are equally encoded into a single pixel of the scan context. Scan context method can compensate for the lack of far-point information and treat nearby dynamic objects as sparse noise. After the point cloud is partitioned, use the point cloud in the bin to assign a real value to each bin [32], and the encoding function of

the bin is $\emptyset(P_{ij})$. The content in the z brackets is a function that returns the z coordinate value of point p [24]. We can obtain the matrix of scanning context I through the above process as shown in Eq. (48).

$$\emptyset(P_{ij}) = \max_{p \in P_{ij}} z(p),$$
$$I = (a_{ij}) \in R^{N_r \times N_s}, a_{ij} = \emptyset(P_{ij}). \tag{48}$$

Here, I^q and I^c are used to represent the scan context obtained by the query point cloud and the candidate point cloud, respectively. They are compared in column A (distance matrix), given scan context pair requires the sum of the distances between the columns at the same index and context pair, which can be used to measure the similarity of the two places. The cosine distance is used to calculate the distance between two vectors c_j^q and c_j^c at the same index, and the following Eq. (49) can be obtained by dividing the sum by the number of columns Ns for normalization [24].

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c_j^q \cdot c_j^c}{\|c_j^q\| \|c_j^c\|}\right). \tag{49}$$

With a column-wise comparison, because the viewpoint of lidar varies from place to place, the columns of candidate scan contexts in the same area may move. Since the context is a sensor position-dependent representation, the row order is always consistent, but the column order may vary due to changes in the lidar sensor relative to the global coordinates. To alleviate this problem, calculate the minimum distance of all possible column shift scan contexts to determine the distance at that time and the number of column shifts for optimal alignment [24], as shown in Eq. (50). Here we can use the iterative closest point algorithm to provide a good initial value for the additional shift information. I_n^c is n columns of scan context shifted from the original one which is I^c.

$$D(I^q, I^c) = \min_{n \in [N_s]} d(I^q, I_n^c),$$

$$n* = \underset{n \in [N_s]}{\operatorname{argmin}}\, d(I^q, I_n^c). \tag{50}$$

The last thing to mention is the two-stage search algorithm. The scan context method integrates two typical mainstream place recognition context algorithms, namely acceptance similarity score and nearest neighbor search, to achieve a faster search time. The above Eq. (50) for the current distance can provide a two-stage hierarchical search algorithm introducing a ring bond. The ring key is a rotation-invariant descriptor extracted from the scan context. Each row of the scan context r is encoded as a single real value by the ring encoding function r, then the ring encoding function used is the occupancy rate of the ring using the L_O norm as shown in Eq. (51) [24].

$$\psi(r_i) = \frac{\|r_i\|_0}{N_s}, \tag{51}$$

Since the occupancy is independent of the viewpoint, the ring bond achieves rotational invariance. Although this method is less informative than scanning the context, the ring keys allow for a fast search to find possible cycle candidates to build the keys of the k-d tree. The number of retrieved most similar keys and the scan context are compared with a constant number of candidates of the query scan context by using the distance, and the one closest to the threshold is selected as the re-entry point. As shown in the following Eq. (52), C represents a set of candidate indices extracted from the k-d tree is also given a threshold. c^* is the index determined as the loop position [24].

$$c^* = \underset{c_k \in C}{\operatorname{argmin}}\, D(I^q, I^{c_k}), s.t\ D < \tau. \tag{52}$$

CHAPTER 4 RESULTS AND DISCUSSION

This chapter will present experiments and comparisons between the proposed algorithm and the other three algorithms described in Chapter 2, which are LeGO-LOAM, LIO-SAM and A-LOAM simulated on different datasets. The hardware device used is a laptop with Intel Core i7-87500H CPU, 2.20GHz and 32GB memory. All algorithms are implemented in C++ programming and did the simulation in ROS under Ubuntu Linux environment. The sensors suited in this report are Ouster OS-1-64 channels 3D lidar and 9-axis IMU. Three different large scale outdoor datasets named KAIST, Riverside and DCC are used to compare the results. The scan numbers and trajectory length was described in Table II.

Table II: Dataset details of scans and trajectory length

Dataset	Scan (Poses)	Trajectory Length (m)
KAIST	86758	5965.198
Riverside	64944	6587.393
DCC	52402	4912.210

4.1 KAIST Dataset Comparison

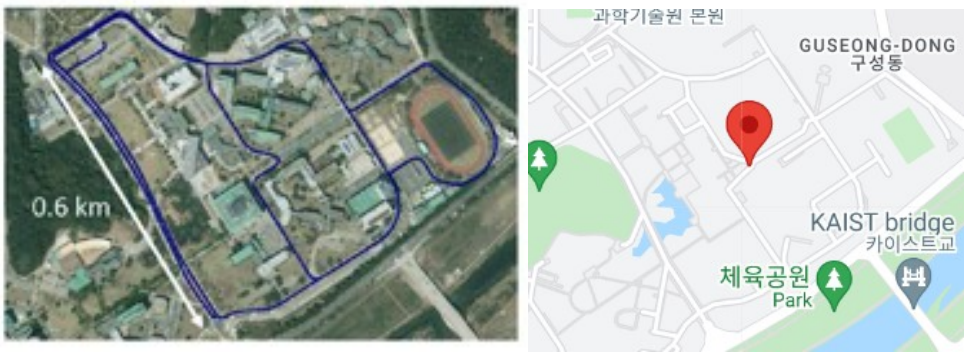


Figure 19: KAIST satellite map and google map [55]

The first dataset I used to compare is the KAIST dataset which is 5965.18 meters in total as Table II shows. Fig. 19 shows its satellite map on the left side and the ground truth trajectory drawn by blue lines, and its google map which is generated from South Korea on the right side. It shows that this dataset is the real scene and allows the comparison between the algorithms more authentic and reliable. Fig. 20 is the point cloud map of the KAIST dataset that used the proposed algorithm to simulate with the visualization tool-Rviz. We can see that the point cloud map and the real environment map in Fig. 19 almost fit, which is a good illustration of the algorithm's accuracy.

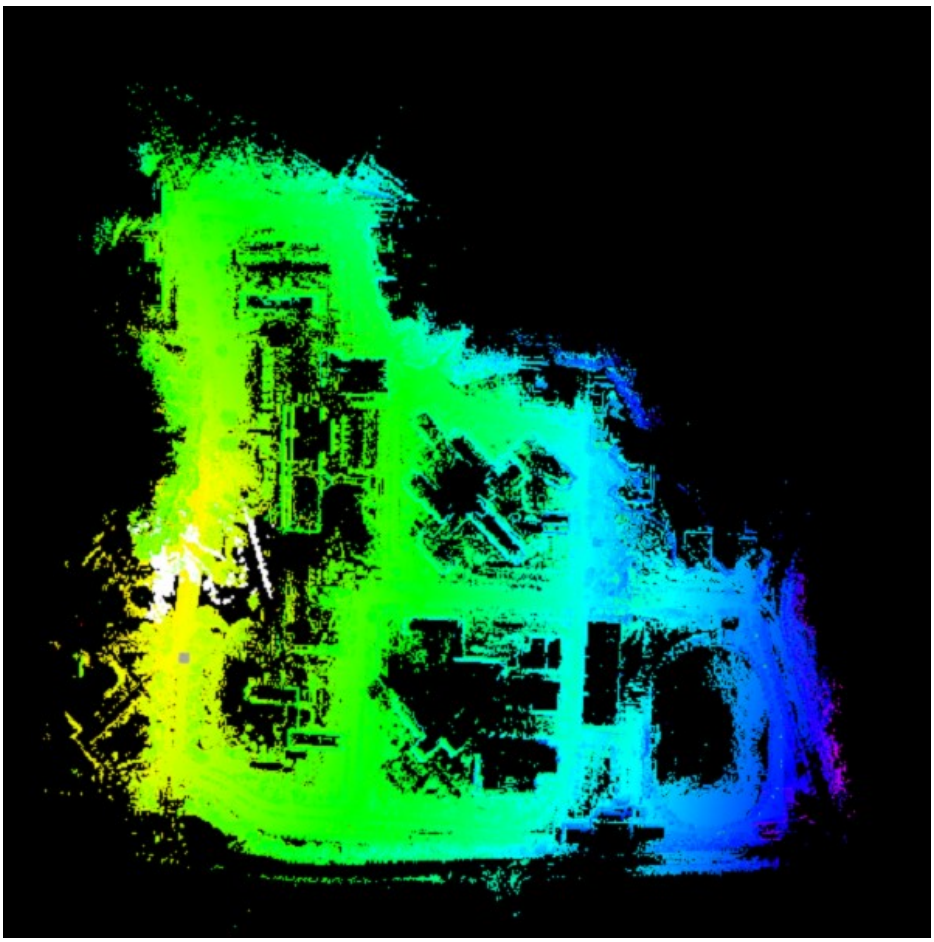
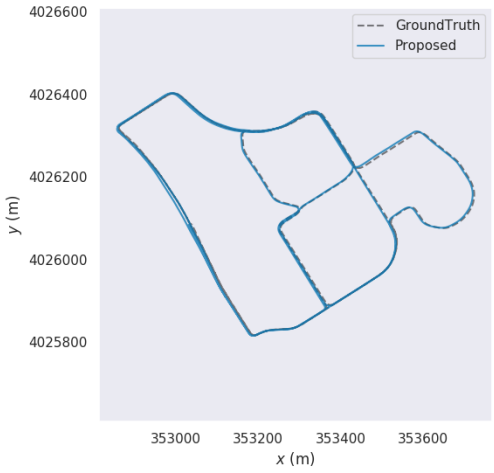
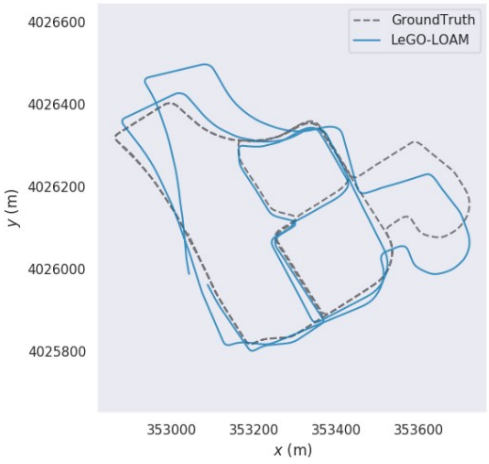


Figure 20: KAIST dataset point cloud map

Then we compare the trajectories of the proposed algorithm and the other three algorithms with the ground truth. In Fig. 21(a-d), the proposed algorithm LeGO-LOAM, LIO-SAM and A-LOAM used for comparison with the ground truth trajectory. Fig. 21(e) compares the above four algorithm trajectories and the ground truth trajectory in the same parameters and conditions. It is clearly to see that the proposed algorithm trajectory most fits the ground truth trajectory. The proposed algorithm is better than the other three algorithm trajectories. Compared with the proposed algorithm, the other three algorithms still have different degrees of drift when the KAIST dataset runs the second lap, especially for the LeGO-LOAM.

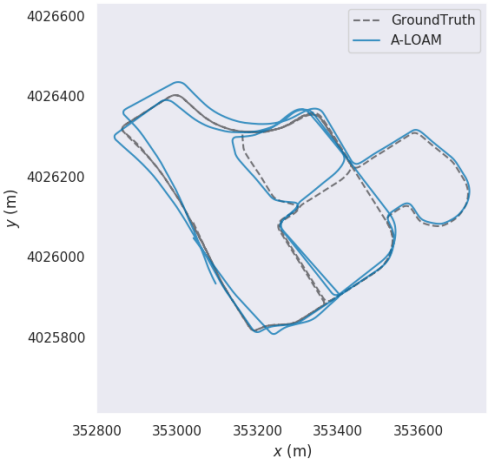


(a)

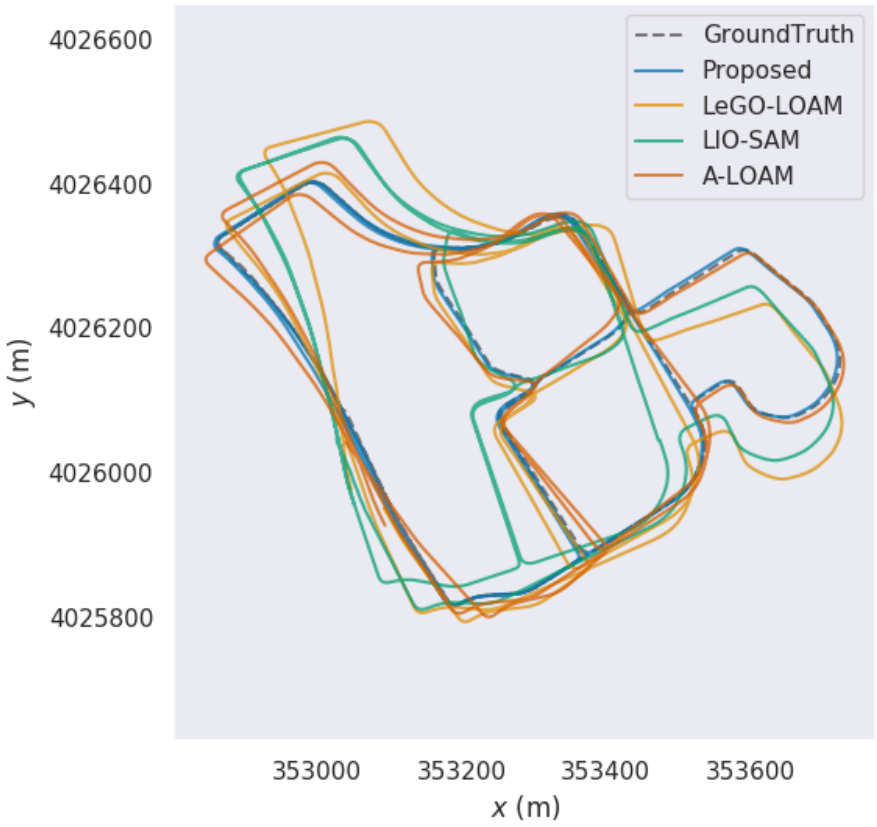


(b)

(c)



(d)



(e)

Figure 21: Each SLAM algorithm simulation trajectories of KAIST dataset compared with ground truth (a) The proposed algorithm, (b) LeGO-LOAM, (c) LIO-SAM, (d) A-LOAM,(e) All algorithms.

As Fig. 22 shows, these two plots represent the absolute pose error and the relative pose error for the proposed algorithm (the blue line segment) and other three algorithms (LeGO-LOAM, LIO-SAM and A-LOAM) used to compare through the KAIST dataset. In the absolute pose error plot, it is obvious to see that in the first lap which in the half time, the algorithm has less absolute pose error compared with the other three algorithms. When in the second lap, the absolute pose error for the proposed algorithm is increasing fast and has the second largest absolute pose error only better than the LeGO-LOAM. From the above trajectory plots, it can be proved that the absolute pose error of the proposed algorithm should not be so large. It may

due to the fact that the dataset has run two laps in total, and its initial pose point has a large error relative to the first lap when running the second lap. The loopback detection is used to correct the final trajectory, but the absolute pose error may not be well updated here. Thus, the absolute pose error in the second lap does not have the best performance compared to the first lap. The right plot in Fig. 22 is relative pose error of those four algorithms simulated with the KAIST dataset. The relative pose error should be more convincing than the absolute pose error, because it is the ratio between the absolute pose error and the true value. The proposed algorithm compared with the LeGO-LOAM, LIO-SAM and A-LOAM has the minimum relative pose error almost closed to 0. It means that when run the KAIST dataset, the proposed algorithm has a better performance than other three algorithms, and the absolute pose error of the proposed algorithm for the second lap of the dataset may not perform well.

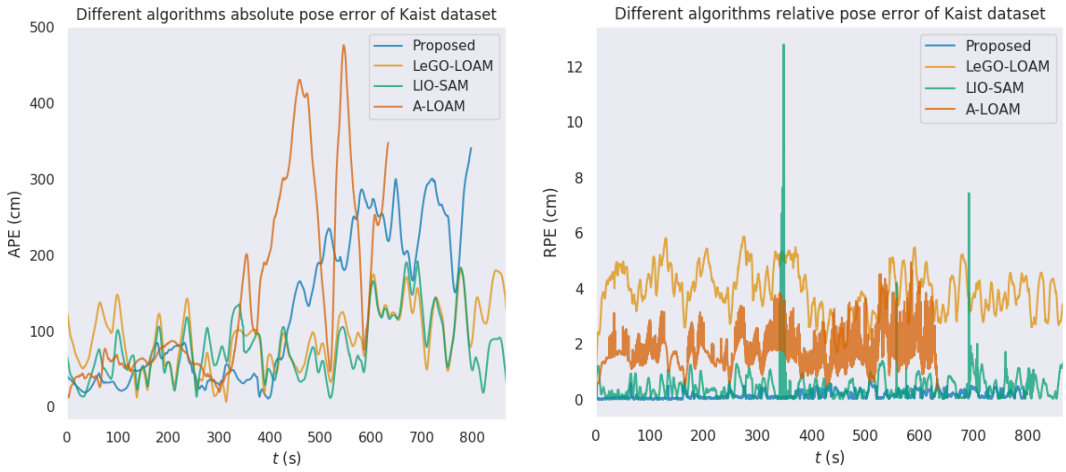


Figure 22: APE and RPE of four algorithms for KAIST dataset

Then Table III and Table IV show below indicate the maximum, minimum, mean and standard deviation of the absolute pose error and relative pose error of those four algorithms for the KAIST dataset. In Table III, because the proposed algorithm did not work well in the second lap, it only performs the third algorithm in those four

algorithms of absolute pose error. But in table IV, the proposed algorithm has the minimum relative pose error and achieves the best, compared with LeGO-LOAM, LIO-SAM and A-LOAM, it is obvious that the proposed algorithm has made better optimizations and performances.

Table III: APE for KAIST dataset of four algorithms

APE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	340.956	125.359	11.044	97.102
LeGO-LOAM	180.360	91.601	6.712	40.370
LIO-SAM	191.763	78.677	11.799	40.083
A-LOAM	476.187	146.656	12.672	123.294

Table IV: RPE for KAIST dataset of four algorithms

RPE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	0.747	0.160	0.00570	0.134
LeGO-LOAM	5.881	3.927	1.181	0.842
LIO-SAM	12.800	0.45	0.018	0.499
A-LOAM	4.947	1.652	0.277	0.540

4.2 Riverside Dataset Comparison

Figure 23: Riverside satellite map and google map [55]

The second dataset I used to compare is the Riverside dataset which is 6587.393 meters in total as table II shows. Fig. 23 shows its satellite map on the left side and the ground truth trajectory drawn by blue lines, and its location in South Korea in the google map on the right side. Fig. 24 is the point cloud map of the Riverside dataset that I used the proposed algorithm to run in the Linux virtual environment as the KAIST dataset did. It is able to see that the shape of the Riverside point cloud map is the same as the shape of the Riverside under a satellite map or google map as shown in Fig. 23.

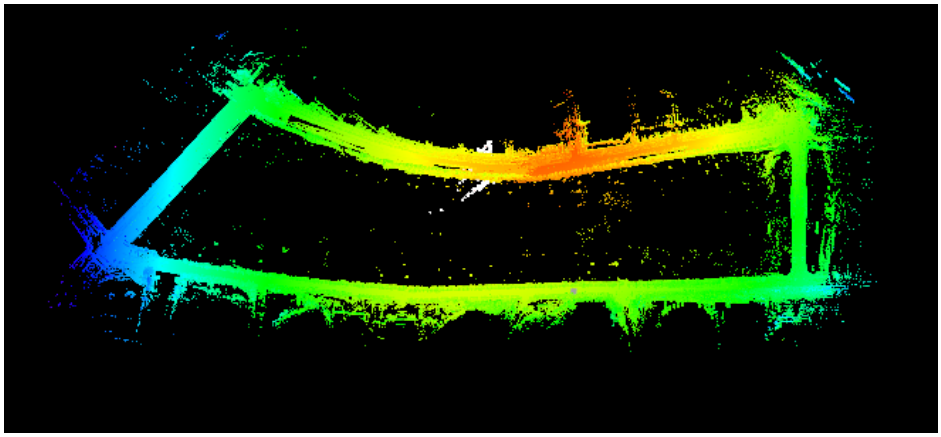
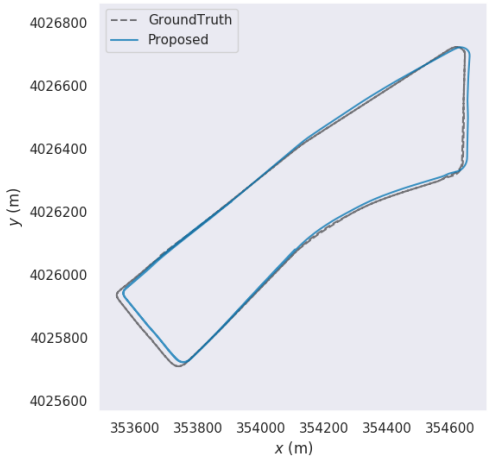
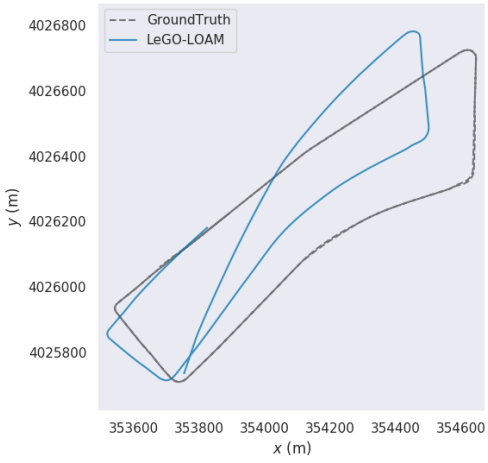


Figure 24: Riverside dataset point cloud map

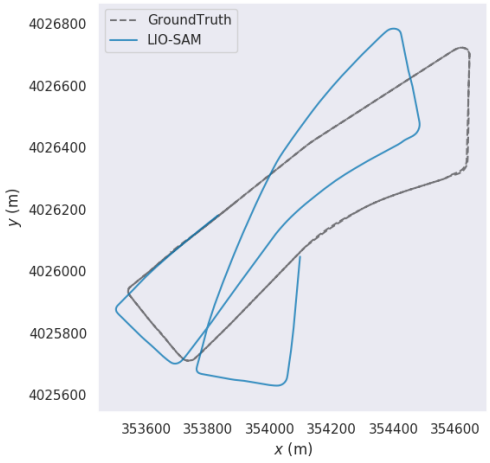
From Fig. 25(a-d), it can be shown that the trajectory generated by the proposed algorithm and the trajectories generated by the other three algorithms for comparisons with the ground truth trajectory alone. However, the trajectory generated by the proposed algorithm and the ground truth trajectory dose not fit perfectly well. The proposed algorithm still performs a better loop closure function and better corrects the final trajectory compared with the other three algorithms. It can be seen that the other three algorithms do not return to the starting point by resulting in a large offset. Fig. 25(e) compares the trajectories generated by the four algorithms and the ground truth trajectory, so that the superiority of each algorithm can be seen more intuitively.



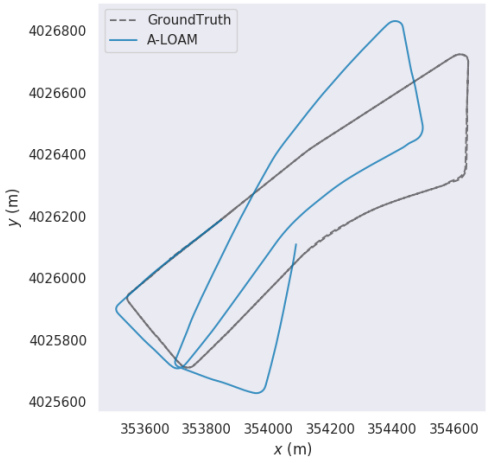
(a)



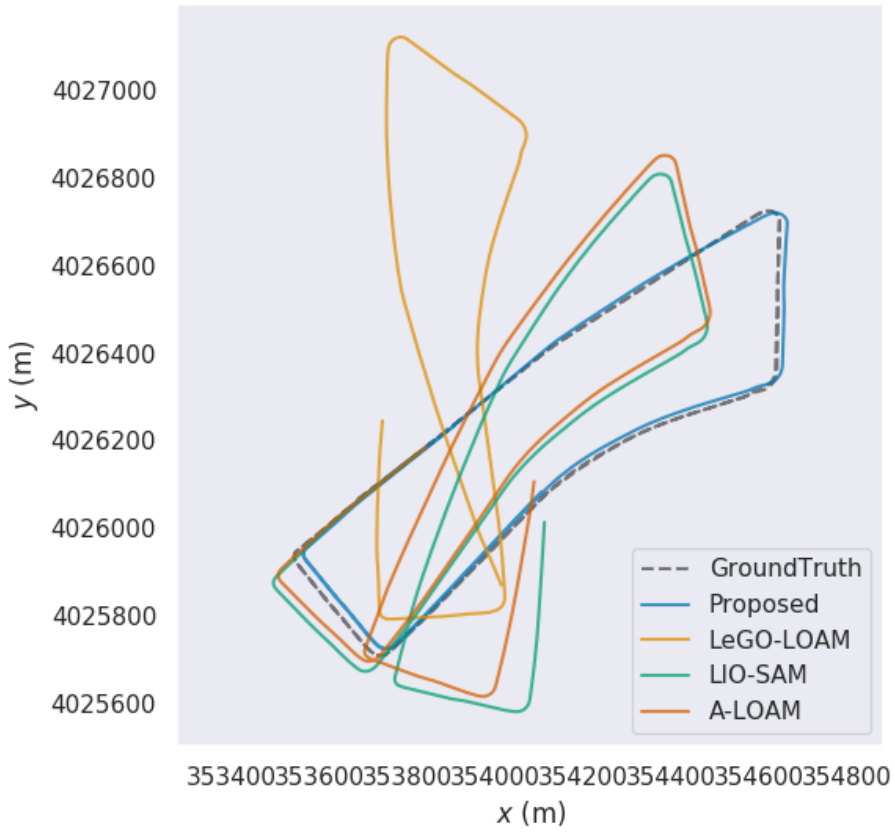
(b)



(c)



(d)



(e)

Figure 25: Each SLAM algorithm simulation trajectories of Riverside dataset compared with ground truth (a) The proposed algorithm, (b) LeGO-LOAM, (c) LIO-SAM, (d) A-LOAM, (e) All algorithms.

In Fig. 26, it shows the absolute pose error and relative pose error plots of four algorithms which are the proposed algorithm, LeGO-LOAM, LIO-SAM and A-LOAM implement with Riverside dataset. The left plot is the absolute pose error of those four algorithms, it can be seen that the proposed algorithm is the blue line segment. Even though its absolute pose error fluctuates wildly, the proposed algorithm still shows the best performance and the most minor absolute pose error compared to the other three classic popular algorithms. The right plot in Fig. 26 is the relative pose error of those four comparison algorithms run on the Riverside dataset; the proposed algorithm also has the minimum relative pose error compared with the

other three algorithms. There is some slight fluctuation may be caused by the large size of the Riverside. Based on the absolute pose error and relative pose error plots, the proposed algorithm still performs better than LeGO-LOAM, LIO-SAM and A-LOAM, which means the proposed algorithm based on the front end of the LeGO-LOAM after back end optimization can reduce more errors and increase the accuracy of final trajectory compared with the other three algorithms.

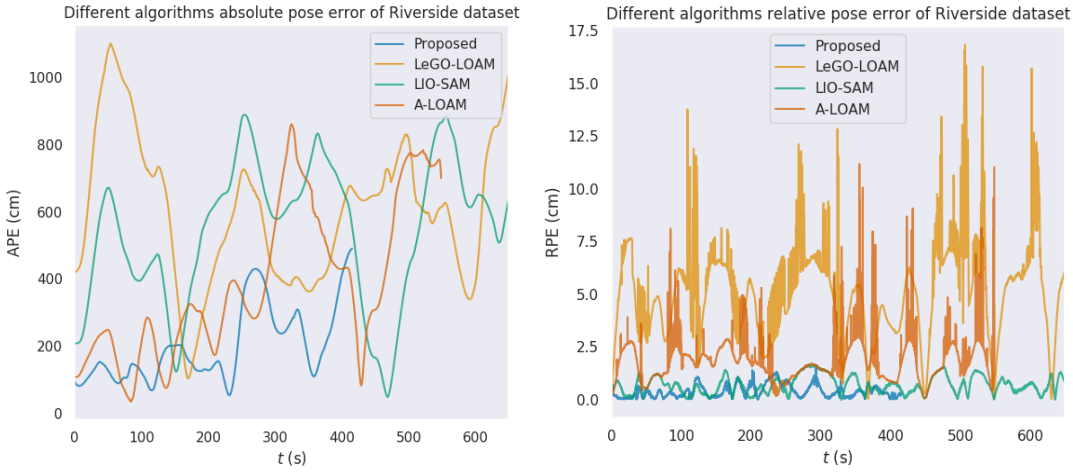


Figure 26: APE and RPE of four algorithms for Riverside dataset

Table V below shows the absolute pose error for the Riverside dataset of those four algorithms. Using the proposed algorithm compared with the other three algorithms, it is clear to see that only the minimum absolute pose error of the proposed algorithm does not have the best performance. And the other three conditions still have the minimum error and have the best performance. Table VI gives the relative pose error for the Riverside dataset of the proposed algorithm, LeGO-LOAM, LIO-SAM and A-LOAM in maximum, minimum, mean and standard deviation of pose errors. The proposed algorithm compared with the other three algorithms still achieves the best performance. Based on the absolute pose error and relative pose error shown in Table

V and Table VI, the proposed algorithm, after back end optimization has improved the origin LeGO-LOAM algorithm.

Table V: APE for Riverside dataset of four algorithms

APE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	490.851	199.961	54.078	109.485
LeGO-LOAM	1100.950	601.168	104.084	211.523
LIO-SAM	888.986	544.190	49.106	207.759
A-LOAM	860.563	387.688	35.429	225.669

Table VI: RPE for Riverside dataset of four algorithms

RPE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	1.603	0.399	0.0083	0.306
LeGO-LOAM	16.841	5.117	0.035	2.276
LIO-SAM	32.661	0.692	0.0182	0.615
A-LOAM	11.171	1.786	0.065	0.977

4.3 DCC Dataset Comparison

Figure 27: DCC satellite map and google map [55]

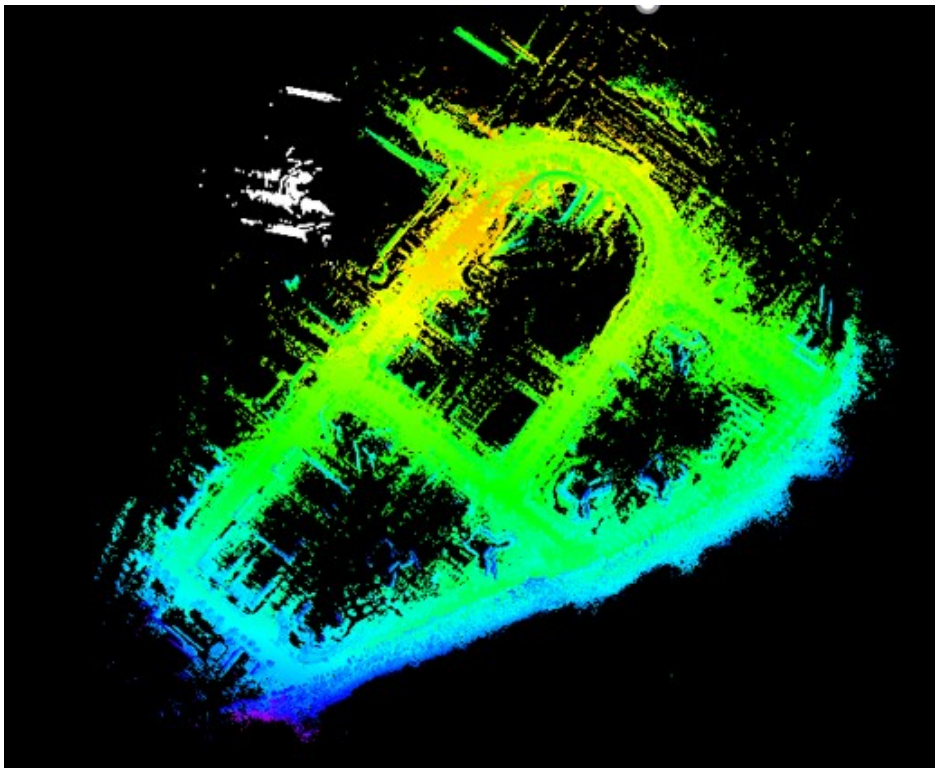
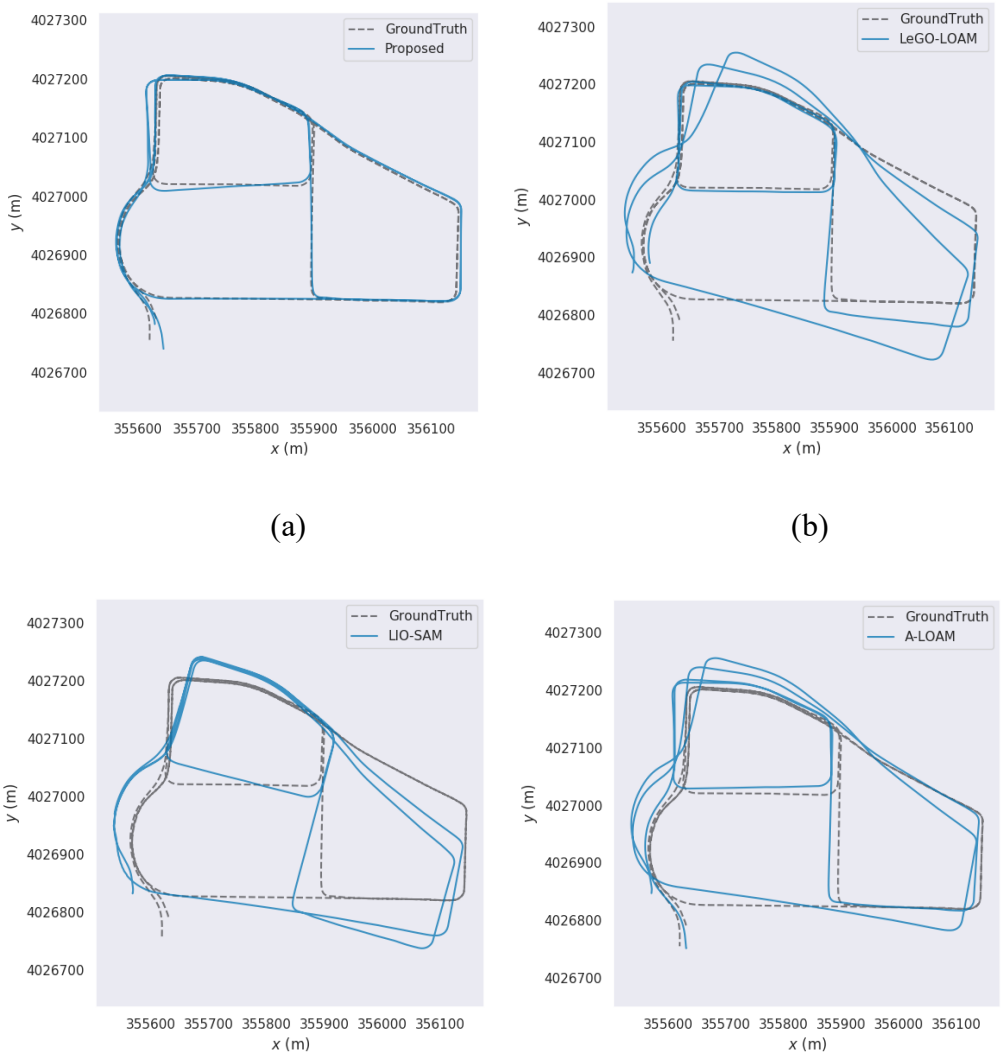


Figure 28: DCC dataset point cloud map

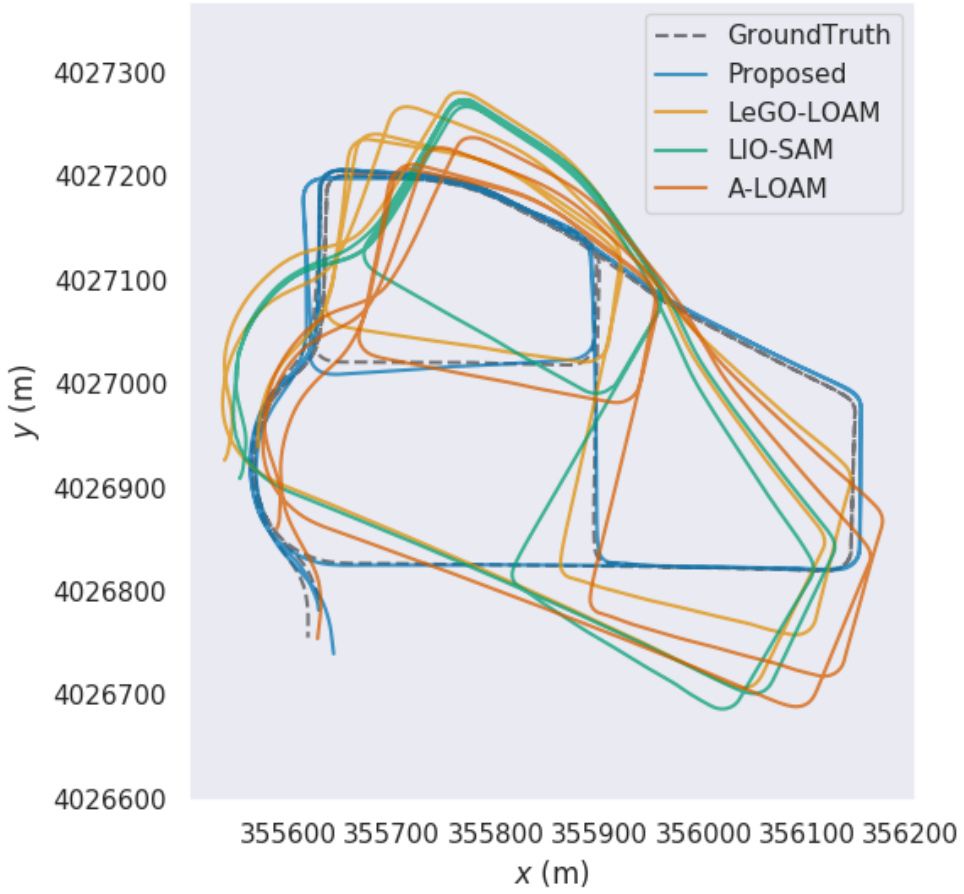
The last dataset I used to compare is the DCC dataset which is 5412.21 meters long as Table II shows. In Fig. 27, the left plot is its satellite map and the trajectory drawn by blue lines. The right plot is its real world location in South Korea on the google map.

Fig. 28 is the point cloud map for the DCC dataset under the simulation of the proposed algorithm. There are a large number of consistencies between the point cloud map and its satellite map and google map. It means that the proposed algorithm almost restores the map of the real large-scale outdoor scene in the simulation without large odometry errors and drifts.



(a) (b)

(c) (d)



(e)

Figure 29: Each SLAM algorithm simulation trajectories of DCC dataset compared with ground truth (a) The proposed algorithm, (b) LeGO-LOAM, (c) LIO-SAM, (d) A-LOAM, (e) All algorithms.

Fig. 29(a-d) shows the trajectories generated by the proposed algorithm, LeGO-LOAM, LIO-SAM and A-LOAM compared with the ground truth trajectory. It is obvious that the proposed algorithm has better performance than other three algorithms for the final trajectory. The algorithms of LeGO-LOAM, LIO-SAM and A-LOAM can return to the initial pose of the robots, and have drifted to varying degrees on the second lap, but the proposed algorithm can still correct the final trajectory. There is only a tiny error occurred around the end path compared with the ground truth, it proves that the proposed algorithm has better accuracy than the

LeGO-LOAM, LIO-SAM and A-LOAM algorithms. Fig. 29(e) gives a better comparison plot for those algorithms with ground truth trajectories. The blue line segment for the proposed algorithm trajectory shows in the figure is most fitting to the dotted line which is the ground truth trajectory.

Fig. 30 indicates the absolute pose error and relative pose error plots for the DCC dataset, the blue line segment is represented the proposed algorithm. As both plots show, whether it is absolute pose error or relative pose error, the proposed algorithm has the minimum error and the best performance compared with LeGO-LOAM, LIO-SAM and A-LOAM. Compared with the simulation of the KAIST and the Riverside datasets, when simulated the proposed algorithm for the DCC dataset, the absolute and relative pose errors are closer to 0 and the fluctuation range is not extensive. It may because the data volume of this dataset is smallest in these three datasets, and the algorithm's computing efficiency is the best which reduces the absolute and relative pose errors. For LIO-SAM algorithm, there is a spike in relative pose error at 200s, KAIST has the same problem, it may be due to the large odometry errors occurred at this time, and also large trajectory drifts happened as shown in Fig. 29(b).

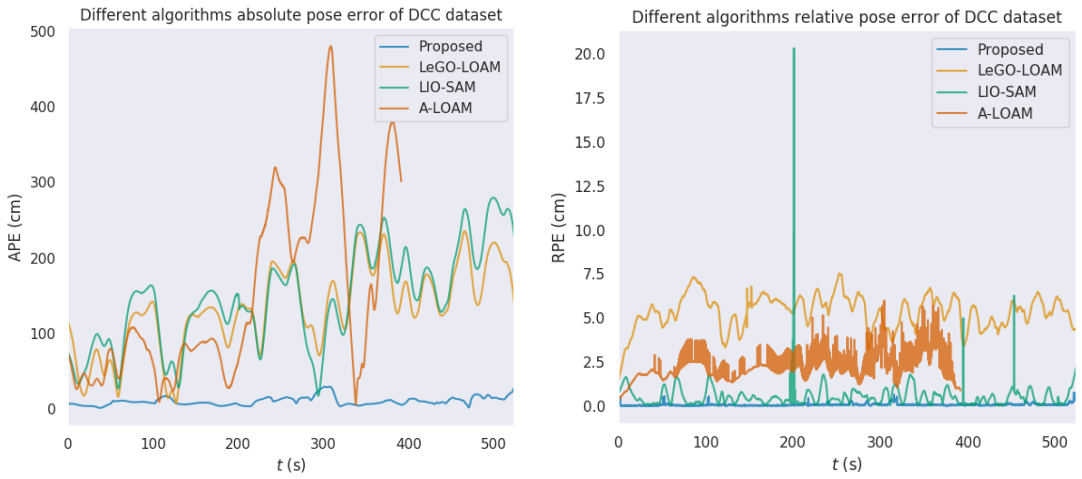


Figure 30: APE and RPE of four algorithms for DCC dataset

Table VII and Table VIII indicate the absolute pose error and relative pose error for the DCC dataset of those four algorithms that used to be compared. In standards of maximum, minimum, mean and standard deviation of the absolute pose error or relative pose error comparison, the proposed algorithm has the minimum error which indicates that the proposed algorithm has the best performance on the DCC dataset. Compared with LeGO-LOAM, LIO-SAM and A-LOAM, if the proposed algorithm runs a smaller size of large-scale outdoor scene dataset, it will give a better performance.

Table VII: APE for DCC dataset of four algorithms

APE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	30.006	10.828	2.066	5.475
LeGO-LOAM	235.753	133.358	7.779	54.836
LIO-SAM	279.879	147.288	18.034	63.780
A-LOAM	480.252	147.917	6.550	118.755

Table VIII: RPE for DCC dataset of four algorithms

RPE	Max (cm)	Mean (cm)	Min (cm)	Std (cm)
The proposed Algorithm	0.766	0.073	0.008	0.075
LeGO-LOAM	7.528	5.358	1.410	0.951
LIO-SAM	20.351	0.549	0.022	0.637
A-LOAM	5.998	2.243	0.352	0.762

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

There are enough results to discuss by comparing the proposed optimized algorithm based on the front end of the LeGO-LOAM and three other algorithms: LeGO-LOAM, LIO-SAM and A-LOAM after simulation under three different large outdoor scene datasets in real world map. When the proposed algorithm simulated in DCC dataset, the absolute and relative pose error are smaller than the other three algorithms, resulting in a more fitted trajectory and better performance. In the Riverside dataset, the proposed algorithm can still have the minimum errors in terms of relative pose error, and have better performance in most aspects of absolute pose error. In KAIST dataset, the proposed algorithm's absolute pose error and relative pose error still has the best performance compared to the LeGO-LOAM, LIO-SAM and A-LOAM in the first lap. The absolute pose error will be higher in the second lap, it may be caused by the untimely update, which is also a part that can be optimized in the future. To improve the real-time performance of the algorithm, an iterated ESKF with a robocentric formulation method was used in the lidar odometry IMU pre-integration. The k-d tree method was replaced with the incremental kd-tree method in lidar mapping, and the scan context research method was added to the final loopback detection. The results of the proposed method simulated on Mulran datasets gathered from real scenes in South Korea. The results show that the proposed algorithm can achieve better accuracy when compared with LeGO-LOAM, and achieve similar or better performance when compared with LIO-SAM and A-LOAM. Those three methods optimized the algorithm and give a better performance compared with these three mainstream algorithms which are LeGO-LOAM, LIO-SAM and A-LOAM.

5.2 Future Work

Lidar SLAM has shown a good effect on localization and mapping. It can be used in combination with other functional modules to achieve better artificial intelligence to save computing resources allocation. However, the proposed algorithm shows higher accuracy than some other classical mainstream lidar SLAM algorithms after optimization. It is still a tightly coupled framework of a single lidar and IMU, which is used in areas that require a higher safety factor such as intelligent driving. Since lidar still has a significant error in a specific environment and has low accuracy in small distance, the following two works can be improved and optimized in future work:

1. Sensor Fusion: Due to the limitations of lidar in some environments, it can be fused with more sensors to build mapping and localization, such as GPS, cameras and millimeter-wave radar, and let these sensors compensate for their high-precision parts, so that the SLAM is more efficient and more accurate.
2. Path Planning: Now the path planning algorithm is widely used in the field of autonomous driving, then the proposed algorithm can combine with the path planning. First, the combined algorithm can use the SLAM algorithm to build and draw the map in an unknown environment, and then use the path planning algorithm to allow the intelligent robot car to plan the optimal path.

BIBLIOGRAPHY

[1] Y. Liu and J. Miura, "RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods," in IEEE Access, vol. 9, pp. 23772-23785, 2021.

[2] S. Fujimoto, Z. Hu, R. Chapuis and R. Aufrère, "ORB-SLAM map initialization improvement using depth," *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.

[3] H. Bavle, P. De La Puente, J. P. How and P. Campoy, "VPS-SLAM: Visual Planar Semantic SLAM for Aerial Robotic Systems," in *IEEE Access*, vol. 8, pp. 60704-60718, 2020.

[4] X. Deng, Z. Zhang, A. Sintov, J. Huang and T. Bretl, "Feature-constrained Active Visual SLAM for Mobile Robot Navigation," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[5] H. Ye, H. Huang and M. Liu, "Monocular Direct Sparse Localization in a Prior 3D Surfel Map," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[6] Y. Abdelrasoul, A. B. S. H. Saman and P. Sebastian, "A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM," *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, 2016.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A factored solution to the simultaneous localization and mapping problem,” in *Proc. AAAI/IAAI*, 2002.

[8] S. Nagla, "2D Hector SLAM of Indoor Mobile Robot using 2D Lidar," *2020 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, 2020.

[9] N. Tongprasit, A. Kawewong and O. Hasegawa, "PIRF-Nav 2: Speeded-up online and incremental appearance-based SLAM in an indoor environment," *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, 2011.

[10] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2011.

[11] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, 2014, p. 9.

[12] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[13] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[14] T. Choi, S. Song, H. Park, S. Yoon and S. Yang, "SUMA: Software-defined Unified Monitoring Agent for SDN," *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[15] T. Li et al., "P3-LOAM: PPP/LiDAR Loosely Coupled SLAM With Accurate Covariance Estimation and Robust RAIM in Urban Canyon Environment," in IEEE Sensors Journal, vol. 21, no. 5, pp. 6660-6671, 1 March1, 2021.

[16] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239-256, Feb. 1992.

[17] Biber, Peter & Straßer, Wolfgang. (2003). The Normal Distributions Transform: A New Approach to Laser Scan Matching. *IEEE International Conference on Intelligent Robots and Systems*. 3. 2743 - 2748 vol.3. 10.1109/IROS.2003.1249285.

[18] E. B. Olson, "Real-time correlative scan matching," *2009 IEEE International Conference on Robotics and Automation*, 2009。

[19] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, "G2o: A general framework for graph optimization," *2011 IEEE International Conference on Robotics and Automation*, 2011.

[20] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, December 2006.

[21] A. Censi, "An ICP variant using a point-to-line metric," *2008 IEEE International Conference on Robotics and Automation*, 2008.

[22] J. Lin and F. Zhang, "Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[23] W. Xu, Y. Cai, and F. Zhang, “ikd-Tree: An Incremental KD Tree for Robotic Applications” 2021, *arXiv:2102.10808*.

[24] G. Kim and A. Kim, "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[25] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang and M. Liu, "LINS: A Lidar-Inertial State Estimator for Robust and Efficient Navigation," *2020 IEEE International Conference on Robotics and Automation (ICRA),* 2020.

[26] W. Xu and F. Zhang, "FAST-LIO: A Fast, Robust LiDAR-Inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter," in *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317-3324, April 2021.

[27] W. Hunt, W. R. Mark and G. Stoll, "Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic*," 2006 IEEE Symposium on Interactive Ray Tracin*g, 2006.

[28] M. Shevtsov, A. Soupikov, and A. Kapustin, "Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes," *Computer Graphics Forum*, vol. 26, no. 3, pp. 395-404, 2007.

[29] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509-517, 1975.

[30] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1),* vol. 2, no. 331-340, p. 2, 2009.

[31] S. G. Salve and K. C. Jondhale, "Shape matching and object recognition using shape contexts," *2010 3rd International Conference on Computer Science and Information Technology*, 2010.

[32] E. Morello and C. Ratti, "A digital image of the city: 3D isovists in Lynch's urban analysis," *Env. and Plan. B: Plan. and Design*, vol. 36, no. 5, pp. 837-853, 2009.

[33] C. M. Greve and K. Hara, "Real-Time Estimation of Discharge Current Oscillations Using an Iterated Extended Kalman Filter," 2021 IEEE International Conference on Plasma Science (ICOPS), 2021.

[34] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053-1072, 2017.

[35] Z. Huai and G. Huang, "Robocentric Visual-Inertial Odometry*," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[36] J. Zhang and S. Singh, "Low-drift and Real-time Lidar Odometry and Mapping," *Autonomous Robots*, vol. 41(2): 401-416, 2017.

[37] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," *2011 IEEE International Conference on Robotics and Automation*, 2011.

[38] C. Forster, L. Carlone, F. Dellaert and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual--Inertial Odometry," in *IEEE Transactions on Robotics,* vol. 33, no. 1, pp. 1-21, Feb. 2017.

[39] H. Ye, Y. Chen and M. Liu, "Tightly Coupled 3D Lidar Inertial Odometry and Mapping," *2019 International Conference on Robotics and Automation (ICRA),* 2019.

[40] R.Hartley and A.Zisserman, Multiple View Geometry in Computer Vision. New York, *Cambridge University Press*, 2004.

[41] W. Hess, D. Kohler, H. Rapp and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *2016 IEEE International Conference on Robotics and Automation (ICRA),* 2016.

[42] Li Meng and Liang Jia-hong, "A Federated particle filtering algorithm based on EKPF," *2011 International Conference on Electric Information and Control Engineering*, 2011.

[43] W. Junrui, G. Teng, W. Xinju, W. Libao, Z. Jingchao and D. Li, "SOC Estimation of Extended Kalman Filter Based on Hardware-in-the-Loop Simulation Platform," 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA), 2021, pp. 1610-1613, doi: 10.1109/ICIEA51954.2021.9516396.

[44] ASSEMBLY. (2019, March 27). *Industrial Robot Sales Broke Records in 2018*. ASSEMBLY RSS. Retrieved July 1, 2022, from https://www.assemblymag.com/articles/94819-industrial-robot-sales-broke-records-in-2018

[45] Scott, C. (2016, August 10). *Is da vinci robotic surgery a revolution or a ripoff?* Healthline. Retrieved July 1, 2022, from https://www.healthline.com/health-news/is-da-vinci-robotic-surgery-revolution-or-ripoff-021215#A-solution-in-search-of-a-problem

[46] *Mir200™: Automate Transport Tasks and focus on higher value activities*. Allied Automation, Inc. (2018, October 4). Retrieved July 1, 2022, from https://www.allied-automation.com/mir200-automate-transport-tasks-and-focus-on-higher-value-activities/

[47] Deshmane, P. (2021, December 17). *Precise Motion Systems Drive Autonomous Mobile Robots*. Mobile Robot Guide. Retrieved July 1, 2022, from https://mobilerobotguide.com/2021/12/17/precise-motion-systems-drive-autonomous-mobile-robots/

[48] Murphy, M. (2018, February 13). *This four-legged robot can open doors and we're all doomed*. MarketWatch. Retrieved July 1, 2022, from https://www.marketwatch.com/story/this-four-legged-robot-can-open-doors-and-were-all-doomed-2018-02-12

[49] Vaudel, C. (n.d.). *Walker Humanoid Service Robot*. RobotLAB Group - Robotics Solution Integrator. Retrieved July 1, 2022, from https://www.robotlab.com/higher-ed-robots/store/walker-humanoid-service-robot-for-research

[50] RealSense, I. (2022, March 7). *Tracking camera T265*. Intel® RealSense™ Depth and Tracking Cameras. Retrieved July 1, 2022, from https://www.intelrealsense.com/tracking-camera-t265/

[51] RealSense, I. (2021, November 12). *Introducing the Intel® realsense™ depth camera D455*. Intel® RealSense™ Depth and Tracking Cameras. Retrieved July 1, 2022, from https://www.intelrealsense.com/depth-camera-d455/

[52] Www.general-Laser.at. (n.d.). *Ouster OS1-64 Lidar Sensor*. General Laser. Retrieved July 1, 2022, from https://general-laser.at/en/shop-en/lidar-en/ouster-os1-64-lidar-sensor-en

[53] *Puck lidar sensor, high-value surround Lidar*. Velodyne Lidar. (2022, June 3). Retrieved July 1, 2022, from https://velodynelidar.com/products/puck/

[54] Hur, M. (2019, October 7). *Autonomous Driving, slam and 3D mapping robot*. Medium. Retrieved July 1, 2022, from https://medium.com/@hurmh92/autonomous-driving-slam-and-3d-mapping-robot-e3cca3c52e95

[55] *Mulran - dataset*. MulRan - Dataset. (n.d.). Retrieved July 1, 2022, from https://sites.google.com/view/mulran-pr/dataset