University of Nottingham
UK | CHINA | MALAYSIA

EEEE3056 Third Year Projects in Electrical and Electronical Engineering

2022/23 Session

3D scanning and model reconstruction based on laser SLAM

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

UNIVERSITY OF NOTTINGHAM NINGBO CHINA

Name: Mr. Yuanzhi Lou

Student ID : 20215854

Supervisor: Dr. Liang Huang

Moderator : Dr. David Chieng

Declaration

I hereby declare that the work described in this report has been done by myself and no portion of the work contained in this report has been submitted in support of any application for any other degree or qualification on this or any other university or institution of learning.

_____

Yuanzhi Lou

Abstract

Simultaneous Localization and Mapping (SLAM) has been an exceedingly popular area of research in recent years, which has a wide range of applications in various fields including autonomous navigation, augmented reality, and gaming. In this report, a real-time SLAM algorithm based on LiDAR sensor is proposed and applied to the 3D scanning and model reconstruction system. This algorithm can be divided into three parts, including laser odometry (front end), loop closure detection and global optimization (back end). The laser odometry is achieved by applying the Point-to-plane Iterative Closest Point (ICP) algorithm and the Levenberg–Marquardt (LM) algorithm, and it is fused with the Attitude and Heading Reference System (AHRS). Loop closure detection is based on the map-to-map ICP algorithm, and the global optimization is accomplished through the graph-based optimization algorithm. In addition, a 3D game for visualization is implemented.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

Simultaneous Localization and Mapping (SLAM) describes the problem of whether a robot moving in an unknown environment can independently construct a map of its surroundings and simultaneously estimate its own position and pose in the map, and it has been a typical problem in the field of intelligent robot in recent decades.

Originating at the International Conference on Robotics and Automation (ICRA) in 1986 [1], SLAM problem received a considerable amount of attention from researchers and the solutions to it was considered the holy grail of the robotics field once it was presented. As time went on, a large number of papers on it were published, and at the same time its knowledge body and theoretical structure became increasingly mature.

After decades of development, SLAM is already regarded to be a settled problem at the theoretical level, however, it still has limitations in practice. For instance, there is not a single SLAM algorithm that can work well in any environment, which means that one algorithm can only be applied to a certain class of similar scenarios. This limits its applications in dynamically varying environments. Other issues such as large-scale map representation, dynamic barrier consideration, and computational efficiency also limit the applications of SLAM.

1.2 Aim

This project aims to design and implement a real-time 3D laser SLAM system that enables efficient, high-quality, and robust automated 3D scanning and model reconstruction for both indoor and outdoor scenes, as well as self-localization in these scenes.

1.3 Objectives

1.3.1 Preparation Stage

i. Literature review of the state-of-the-art technologies on 3D laser SLAM and investigate its market value.
ii. Set up the Robot Operating System (ROS) environment, the LiDAR sensor, and any other hardware or software required in this project.

1.3.2 Design Stage

iii. Design the laser odometry algorithm to realize basic map construction and localization capability.
iv. Design a loop closure detection approach that has high efficiency and low error rate.
v. Design a global optimization method to eliminate cumulative error generated by the laser odometry.
vi. Integrate AHRS into the laser odometer.
vii. Realize visualization by using tools including RViz, Pangolin and Unreal Engine.

1.3.3 Test Stage

viii. Compare the performance of proposed SLAM algorithm with other works in different scenes.
ix. Analysis the time cost of SLAM algorithm to validate its real-time capability.
x. Summarize the entire development process of the project and explore its potential improvements.

1.4 Applications

With the increasing popularity of automation technology, the applications of laser SLAM are becoming widely widespread, especially in the fields of robotics, autonomous driving, and construction, where market demand has grown significantly. In the field of robotics and autonomous driving, laser SLAM is used for real-time mapping and navigation, enabling robots or vehicles to move securely and efficiently in complex environments, especially in the cases that GPS fails to work; in the construction area, it helps engineers to map and monitor the buildings and improve the efficiency of workers. In addition, with the rapid development of Augmented Reality (AR) technology in recent years, it also turns out a great demand for 3D SLAM. As a key component of AR, SLAM offers the ability of adding virtual objects based on real environments [2].

1.5 Deliverables

i. A literature review covering the 3D laser SLAM related knowledge and its market value.
ii. A ROS environment to develop and test the SLAM algorithm.
iii. A hardware environment including the LiDAR sensor and AHRS sensor.
iv. A C++ program that realizes the AHRS fusion algorithm.
v. A C++ program for the laser odometry algorithm.
vi. A C++ program that realizes the loop closure detection and global optimization approach.
vii. A laser odometry algorithm integrated with AHRS.
viii. Visualization tools including RViz and Pangolin that are used to show the testing results.
ix. A game by unreal engine that can be used to display the results of the 3D reconstruction.

Chapter 2 Literature Review

2.1 Point Cloud Registration

The point cloud registration algorithm describes a method that feeds in two point cloud sets, and outputs a transformation (including rotation and translation) that makes the two point cloud sets overlap as much as possible. The output transformation is the relative position relationship between these two point cloud sets. In laser SLAM, this algorithm is used to predict the change in position and pose of the detected point cloud between two adjacent time frames, hence determine the position and pose of the laser camera itself. This algorithm plays an extremely important role in laser SLAM system and its performance determines the magnitude of the cumulative error caused by the laser odometer. A highly accurate point cloud registration algorithm leads to smaller cumulative error, hence a high-performance laser odometer. As long as the laser odometer's cumulative error is low enough, the loop closure detection and global optimization parts can be neglected, which can greatly increase the real-time performance of the SLAM system.

The iterative closest point (ICP) algorithm is broadly applied in laser-based point cloud registration methods. Depending on the matching object, the ICP algorithm can be divided into three types, including point-to-point ICP, point-to-line ICP and point-to-plane ICP [3].

2.1.1 Point-to-point ICP

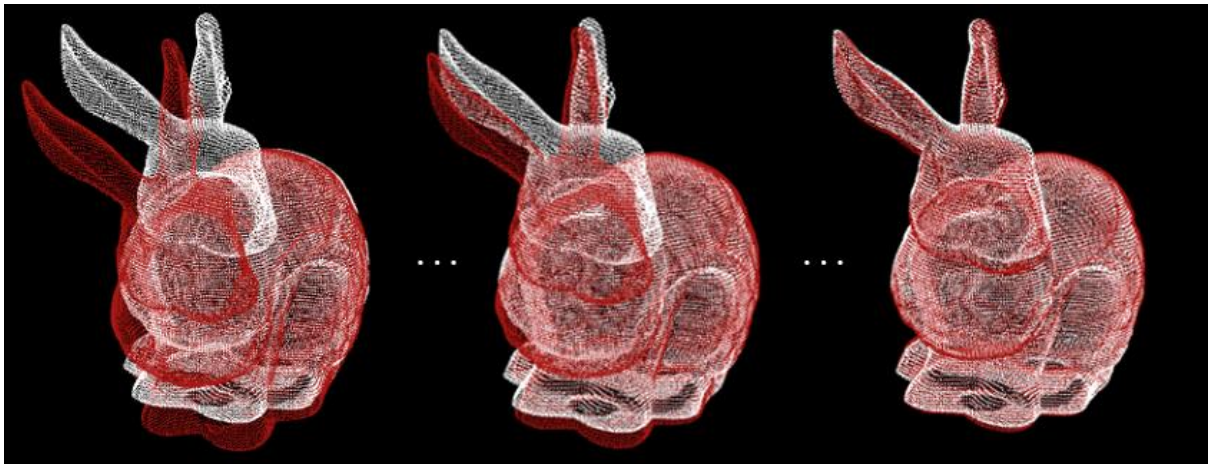


Figure 2-1 The Iteration Process of Point-to-point ICP (Red: Source, White: Target)

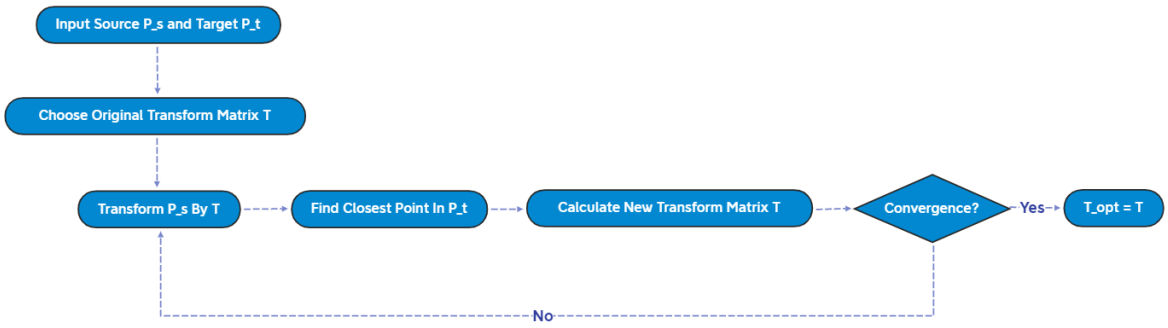


Figure 2-2 The Principle of Point-to-point ICP

Point-to-point iterative closest point algorithm, originally proposed by Besl and Mckay in 1992 [4], was designed to perform computationally efficient and accurate registration of 3D point cloud, and it and its variants are still in use today.

The process and principle of this algorithm is shown in Figure 2-1 and Figure 2-2 respectively. Assuming that there is a source point cloud set P_s and a target point cloud set P_t, the optimal transformation T, in which each point in P_s after the transformation coincides as closely as possible with the corresponding point in P_t, is obtained through iterations. For each iteration, all the points in P_s will be transformed by the optimal transformation obtained from last iteration, and then for each transformed point, the closest point in P_t will be found and treated as the corresponding point P_t^i. After that, the new optimal transformation of current iteration T_{cur} will be calculated by the Equation 2.1, and T_{cur} will enter the next iteration [5]. This process will loop until convergence.

$$T_{cur} = argmin(\frac{1}{N}\sum_{i=1}^{N}\left\|P_t^i - (R \cdot P_s^i + t)\right\|^2) \tag{2.1}$$

Where R and t represent rotation and translation of T respectively, and N represents the number of points in P_s.

This algorithm is simple and does not require feature extraction of the point cloud. However, it has a significant computational cost in searching for the corresponding points, and it only considers the point-to-point distances, ignoring information about the structure of the point cloud.

2.1.2 Point-to-line ICP

Point-to-line ICP (PLICP) is a variant of ICP algorithm proposed in 2008, and a point-to-line metric is used in this algorithm. Compared with the point-to-point ICP, PLICP has higher precision, faster convergence rate, and requires fewer iteration steps. However, it shows lower robustness when the initial displacement errors are large [6]. Figure 2-3 shows the difference in algorithm principle between point-to-point ICP and PLICP. The main difference is that the point-to-point ICP finds the nearest point and treats the distance between the two points as the loss function, while the PLICP finds the nearest two points and connects them as a line, using the distance from the point to the line as the loss function.

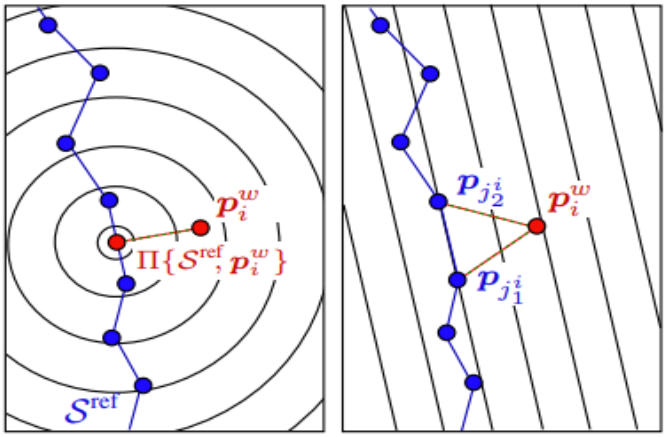


Figure 2-3 The Principle Difference Between Point-to-point ICP and PLICP [6]

The loss function of PLICP is shown by the Equation 2.2 below. By minimizing this loss function (LF) in each iteration process which is the same as the point-to-point ICP, the optimal transform matrix T can be found.

$$LF = \frac{1}{N}\sum_{i=1}^{N}\left\|n_i^T \cdot \left[P_t^i - \left(R \cdot P_s^i + t\right)\right]\right\|^2 \tag{2.2}$$

Where P_s^i represents the coordinate of the point in the source point cloud set P_s, P_t^i represents the coordinate of either of the two nearest points, n_i represents the normal vector of the line formed by the two nearest points, R and t represent rotation and translation of T respectively, and N represents the number of points in P_s.

2.1.3 Point-to-plane ICP

Point-to-plane ICP is an advanced version of PLICP, and the only difference between them is that PLICP requires to find the nearest two points to form a line and minimize the point-

to-line distance, while the point-to-plane ICP needs to find the nearest three or more points to form a plane and minimize the point-to-plane distance. The loss function of it is almost the same as PLICP, which is shown by the Equation 2.2 above, and the only difference is the definition of n_i. In point-to-plane ICP, n_i is the normal vector of the plane formed by these nearest points, instead of a line.

Different from point-to-point metric and Point-to-line metric which have closed-form expression, the point-to-plane metric is often resolved by non-linear least square (NLS) methods, such as Gauss-Newton method and Levenberg-Marquardt method [7]. Through using NLS methods, the computational speed of each iteration is reduced, but the convergence rate is increased [8]. In addition, compared with the point-to-point ICP and PLICP, point-to-plane ICP considers more information about the structure of the point cloud, which contributes to a better accuracy.

2.2 Attitude and Heading Reference System

An attitude and heading reference system (AHRS) consists of a MEMS based 3-axis gyroscope, 3-axis accelerometer and 3-axis magnetometer with a reference system derived from the Earth's gravity and magnetic fields. By analyzing the data from these sensors, AHRS is able to provide attitude information on three axes, including roll, pitch, and yaw. The AHRS used in this project is shown in Figure 2-4 below.

Figure 2-4 AHRS Used in the Project

The AHRS has similar concept and function with the Inertial Measurement Unit (IMU), but there are differences between them. The first one is that the AHRS has an additional 3-axis magnetometer than the IMU, which makes it necessary for the AHRS to work in the Earth's

magnetic field; the second one is that the AHRS has an integrated on-board processing system, which can output attitude information directly. What's more, AHRS is cheaper than IMU for the same accuracy demand.

AHRS is reliable and widely used in multiple applications, such as vehicles, satellites, and spacecrafts [9], and plays an important role in many SLAM systems. In the SLAM system, attitude estimation using LiDAR sensor solely is susceptible to external factors such as motion interference and occlusion, which will cause cumulative error. To overcome these problems, an AHRS can be introduced as a compensation for the laser odometry. In this way, the accuracy of the SLAM system can be significantly improved.

2.3 **Loop Closure Detection**

Loop Closure Detection (LCD) refers to the ability of a robot to recognize a scene that has been visited before. It plays the role of a bridge in the SLAM system, linking the laser odometer at the front end and the global optimization at the back end. Only when the loop closure is detected, the global optimization will be activated to eliminate the cumulative error produced by the laser odometer and generate consistent and accurate map. Therefore, an efficient and low error rate LCD algorithm is a key component of the SLAM system.

Currently, LCD methods can be divided into three categories, including point feature based, object based, and learning based methods. The point feature-based methods use a descriptor to encode the structure around the key point and treat the descriptor as the feature to be matched; the object-based methods use objects, such as a desk or a computer, as the feature; and the learning-based methods use deep learning to learn the feature from raw 3D data [10].

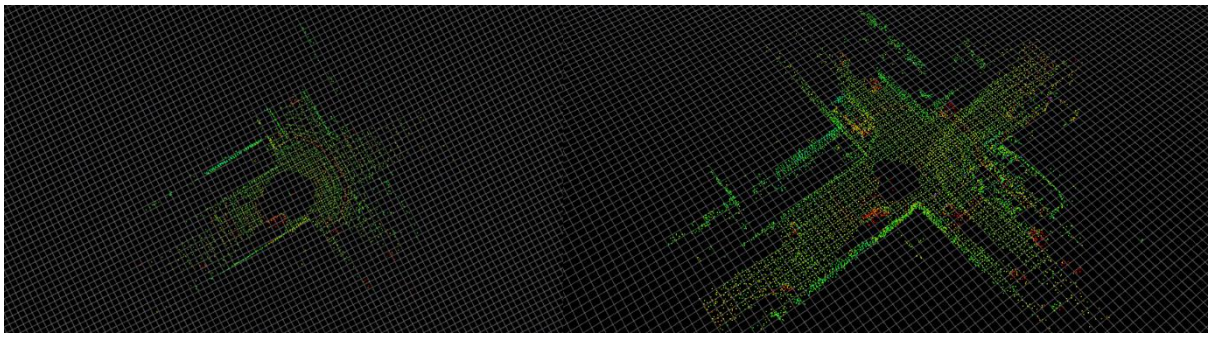


Figure 2-5 Comparison of Scan (Left) and Map (Right)

In this project, an LCD approach based on point-to-point ICP is used. This approach is conceptually a point feature-based method, but no descriptor is used. According to the amount

of point cloud used for matching, this approach can be divided into three types, scan to scan, scan to map, and map to map. Note that scan represents a LiDAR frame, while map represents a local map that contains several LiDAR frames, which is shown in Figure 2-5.

2.3.1 Scan to Scan

The scan-to-scan LCD approach uses the point-to-point ICP method to match the point cloud of the current frame with the point cloud of all history frames, screening out the history frames with the highest matching degree. As long as its matching degree is above a threshold, this history frame can be considered to be in the same position and pose as the current frame, which means a loop closure is detected. The advantage of this approach is that the computation is fast, but it is of low accuracy and prone to incorrect judgements.

2.3.2 Scan to Map

The scan-to-map LCD approach matches the point cloud of the current frame with the point cloud of all history local maps. Compared with the scan-to-scan approach, this approach is much more accurate and has a lower risk of incorrect judgements. Although the computing speed has decreased due to the increase in the size of the point cloud, its efficiency is still within acceptable tolerances. One typical algorithm that applies the scan-to-map method is Cartographer [11], which proved that scan-to-map matching is more efficient and robust than the scan-to-scan matching method.

2.3.3 Map to Map

The map-to-map LCD method is an advanced version of scan-to-map method. It collects the latest several dozens of frames, and combine them as one local map, which is used to compare with the history local maps. Although its accuracy has been further improved, its huge computational cost makes it impossible to implement in most real-time SLAM systems. Therefore, optimization must be done to reduce its computational cost, which is a problem that many researchers are focusing on. For instance, [12] proposed an improved Cartographer algorithm based on the map-to-map LCD method. Optimization approaches such as preliminary matching and lazy decision are applied to improve the real-time performance and robustness of the algorithm.

2.4 Global Optimization

Global optimization is extremely important in a SLAM system. This is because no matter how accurate a laser odometer is, due to its nature of only analyzing data at the latest moment in time, it will inevitably cause cumulative error in the constructed map after working on a large scale over a long period of time. Global optimization, on the other hand, analyzes data from all historical moments in time with a global view, thus correcting the entire map to make it globally consistent. Global optimization algorithms can be divided into two categories, filtering methods represented by the Extended Kalman Filter (EKF), and non-linear optimization methods represented by the Graph-based Optimization.

2.4.1 Graph-based Optimization

The graph in Graph-based Optimization is an important data structure that consists of a number of vertexes, and edges that connect these vertexes, where the vertexes represent objects, and the edges represent the relationships between objects. The graph can be divided into directed graph and undirected graph. In a directed graph, edges are directed and represent a one-way relationship from one vertex to another, while in an undirected graph, edges are undirected and represent a two-way relationship between vertexes. In the graph-based optimization algorithm for SLAM, an undirected graph is usually used to represent the state of the robot, where vertexes represent the pose of the robot at each moment, and edges represent the relative relationship of poses between adjacent moments [13]. This graph is known as the Pose Graph, which is shown in Figure 2-6.

In the Figure 2-6, the direction in which the numbers increase represents the direction of the robot's movement, the circles (vertexes) represent the robot’s poses, and the solid lines (edges) represent the changes in pose during robot movement. In addition, the solid circles represent the poses when the loop closure is detected, and the dotted lines represent the constraints between the corresponding loop closure poses. Ideally, if the cumulative error of the laser odometer is ignored, the corresponding poses when the loopback is detected should coincide. However, the actual situation should be the same as that shown in Figure 2-6, where there will be difference between the corresponding loop closure poses due to the cumulative error. The difference is represented by the constraints $\{m_1 \quad m_2 \quad m_3\}$. The core of the graph-based optimization is to minimize the sum of all constraints by updating the whole Pose Graph,

so that the corresponding loop closure poses can coincide as much as possible. As the map is built based on poses, the correction of the poses also allows the map to be optimized.

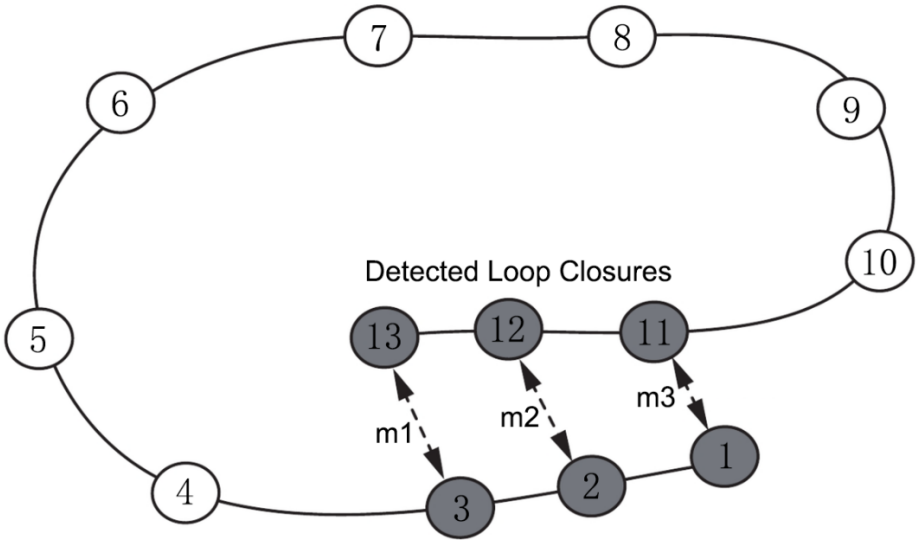


Figure 2-6 The Pose Graph

Since firstly proposed in 1997 [14], the graph-based optimization method has been developed for more than two decades, and it is becoming the mainstream SLAM back-end global optimization algorithm due to its impressive performance.

2.4.2 EKF-based Optimization

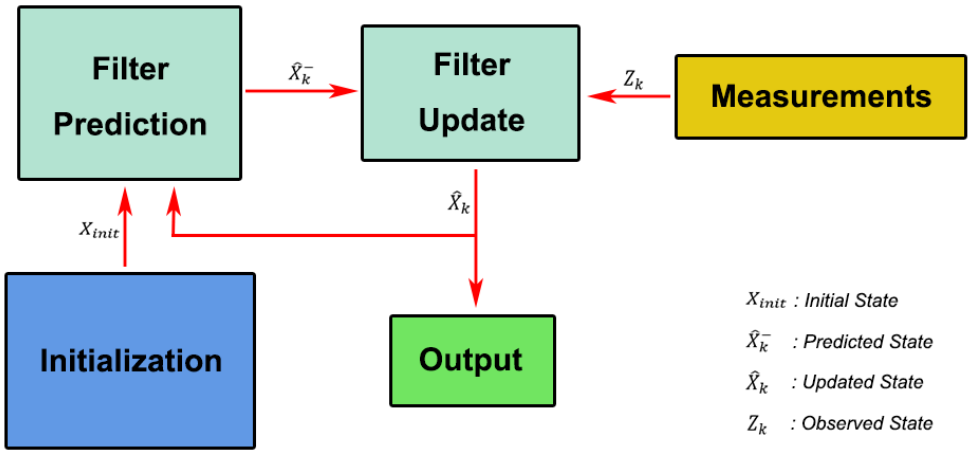


Figure 2-7 The Principle of EKF-based Optimization Method

The EKF-based optimization method uses a Kalman filter to estimate the pose of the robot and the local map. It uses two state vectors representing pose and local map respectively, and then updates these two vectors based on the sensor measurements, so that the accumulated

error can be filtered out. In principle, EKF-based optimization can be divided into two parts, including prediction and update, which is shown in Figure 2-7. In the prediction, the pose and local map at the current moment will be predicted through the motion model and the states of the last moment, and the state transition matrix and covariance matrix will be calculated. In the update part, the covariance matrix will be updated by the error between the observed and predicted values, and then the states, including pose and local map, will be corrected by the updated covariance matrix.

EKF-based optimization was the main optimization method at the end of the last century when the SLAM problem emerged [15]. However, since the Jacobian coefficients are evaluated on the estimated state values, this optimization method may produce inconsistent estimates. This inconsistency is tightly linked to the local observability of the SLAM problem. It is because of its local nature that inconsistency cannot be completely avoided [16]. Therefore, this kind of approaches has been gradually obsoleted by researchers and replaced by graph-based optimization methods.

2.5 3D Map Model Types

There are many types of 3D model that can be used to construct a map, such as the point cloud, occupancy grid, mesh, etc. In this project, the first two types will be applied to the SLAM system.

2.5.1 Point Cloud

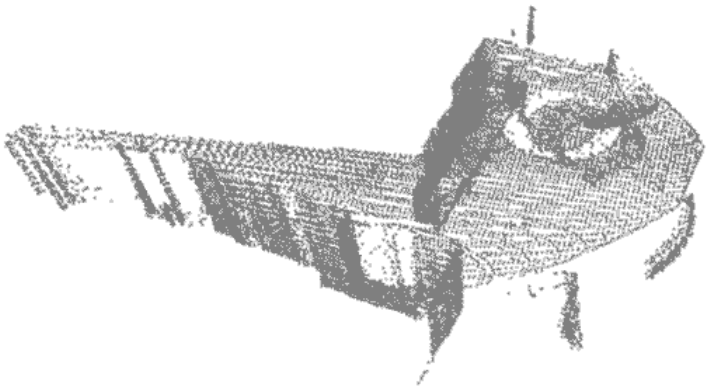


Figure 2-8 The Map in the Form of Point Cloud

A 3D point cloud is a set of points in a 3D coordinate system. These points are usually defined by (x, y, z), and they can be used to represent the surface of an object or a map, as shown in Figure 2-8.

Point cloud is used in a wide range of applications. In the field of architecture, 3D point cloud data are mainly applied for 3D model reconstruction and geometry quality inspection [17]. In the area of autonomous driving, it is used in SLAM and object detection technology. Additionally, in the Virtual and Augmented Reality area, it helps to create realistic virtual environments and superimpose digital information onto the real world.

2.5.2 Occupancy Grid

Occupancy grid is a 2D or 3D random field model that divides the space into multiple small cells and maintains the occupancy status of each cell [18]. The occupancy status is usually represented by a binary value (0 or 1) or a probability value between 0 and 1. Figure 2-9 below shows a map in the form of occupancy grid which shares the same environment with the one in Figure 2-8. In contrast to the point cloud in which the points have no volume, each cell in the occupancy grid is a cube. This characteristic makes it more popular than point cloud in the field of obstacle avoidance and path planning.

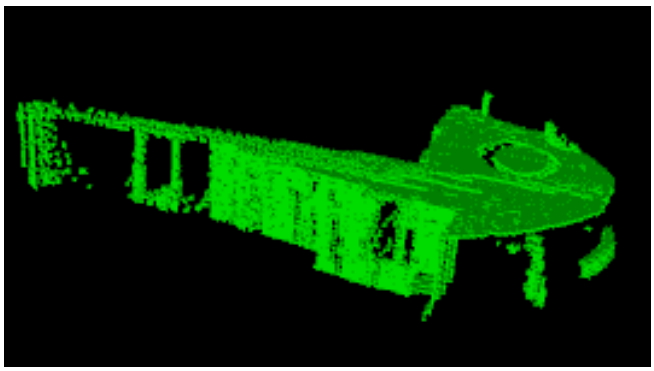


Figure 2-9 The Map in the Form of Occupancy Grid

Occupancy grid is able to provide a dense and effective representation of the environment that can be easily updated in real time. However, the accuracy of the occupancy grid depends on the resolution of the grid, and higher resolution requires more computational resources.

2.6 Robot Operating System

Robot Operating System (ROS) is a Linux-based open-source framework that contains abundant tools for developing robot systems. To understand this project, several concepts of ROS need to be introduced. Firstly, a node is an executable that carries out a certain mission. secondly, communication between nodes can be achieved by subscribing and publishing a topic, which is a specific information bus. For example, a node can publish messages to a topic, and other nodes can subscribe to that topic to receive those messages.

2.7 Empirical Studies

Over the last few decades, a lot of research focusing on laser SLAM has been carried out. The goal is to enable robots to self-explore in a variety of different environments, including the ocean and space, and to do jobs that humans cannot do.

In the early history of laser SLAM research, SLAM algorithms based on filtering methods were dominant, among which FastSLAM was a classic algorithm in that period. This algorithm decomposes the posterior of the SLAM problem into the product of the pose posterior and the landmark posterior, and this decomposed posterior can be efficiently approximated using a particle filter. Compared with the EKF-SLAM, FastSLAM has lower complexity and higher efficiency in large scale scenarios, as well as better performance in data association [19]. In 2003, an advanced variant of FastSLAM, FastSLAM 2.0 [20], was proposed. In contrast to FastSLAM, this approach incorporates the latest measurement in the pose prediction process to utilize particles more effectively. In addition, it uses multiple particles to better deal with cases with ambiguous data association.

With the development of laser SLAM, researchers have found that SLAM algorithms based on filtering methods have limitations, so a more promising approach, graph-based optimization, is gradually coming into vogue. GraphSLAM [21], proposed in 2006, is a unified algorithm for the offline SLAM problem, which applies the graph-based optimization techniques to SLAM. This algorithm converts the SLAM posterior into a graph and then reduces it through variable elimination techniques so that the original SLAM problem can be simplified as a lower-dimensional problem that can be solved by conventional optimization methods. Based on GraphSLAM, an alternative algorithm, LAGO-SLAM, has been implemented, which improves the optimization process so that no initial estimation is needed

to find the correct graph configuration [22]. However, this improvement increases the computational cost of the algorithm.

In 2014, a powerful laser SLAM structure was created with the name, LOAM-SLAM [23]. The algorithm uses a 10Hz laser odometer to estimate the velocity of the LiDAR, and a 1Hz laser odometer for fine matching of the point cloud. By combining these two odometers together, a real-time SLAM system is achieved. The block diagram of this algorithm is shown in Figure 2-10 below.

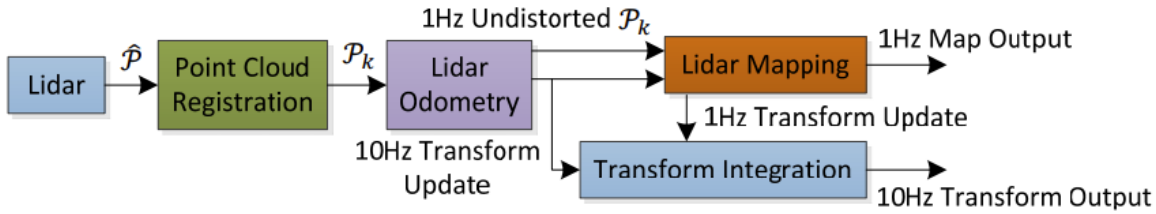


Figure 2-10 The Block Diagram of LOAM-SLAM [23]

LOAM-SLAM has the advantages of low drift and low computing complexity. However, due to the lack of the loop closure detection and global optimization part, it will face a large cumulative error in large scale scenarios. To overcome this issue, LeGO-LOAM [24], an improved version of LOAM-SLAM, was proposed in 2018. Compared to LOAM, this algorithm makes use of the ground when performing the segmentation and optimization steps, and the back-end optimization algorithm is incorporated to reduce the cumulative error. In addition, it can run in real-time in low-power embedded devices while ensuring its accuracy. Another variant of the LOAM is LIO-SAM [25], which is proposed in 2020. In this algorithm, a tightly coupled lidar inertial odometry which integrates an inertial measurement unit (IMU) and a LiDAR sensor is used to realize highly accurate pose estimation and map construction in real time. The addition of the IMU eliminates the error from motion distortion and provides a good initial pose for the laser odometer, all of which makes the lidar inertial odometry more accurate and produce less cumulative error. Additionally, in order to improve the real-time performance, this algorithm uses a sliding window method to marginalize old frames for scan matching.

Chapter 3 System Model

3.1 Development Environment Setup

3.1.1 Robot Operating System (ROS) Installation

The system used in this project is Ubuntu 18.04, so the corresponding version of ROS, Melodic Morenia, is chosen. To install it, a more convenient tool, fishros, which integrates all installation codes into a single script file, is used rather than the tedious installation process on the ros official website.

3.1.2 Livox Mid-40 LiDAR Sensor Configuration

In this project, point cloud data from the real environment collected by the Livox Mid-40 LiDAR sensor is used in addition to the dataset downloaded online. To use this LiDAR sensor in Ubuntu, three steps should be done. The first step is to download the Livox-SDK from GitHub and install it on Ubuntu. Livox SDK is a software development kit for all Livox products. Based on C/C++, it provides user-friendly C-style interfaces for driving the LiDAR sensor. The second step is to install the Livox ROS Driver, which enables the LiDAR sensor in the ROS environment. By the way, it is because the Livox ROS Driver can only run under Ubuntu 14.04/16.04/18.04 that Ubuntu 18.04 is chosen as the operating system for this project. The final step is to connect the Mid-40 to the computer through the Livox Converter using Ethernet cables and to change the IP address of the computer so that it is under the same network segment as Mid-40, thus enabling communication between them. After completing these three steps, the Mid-40 is capable of providing real-world point cloud data on ROS.

3.1.3 Unreal Engine 4 and Its Communication With ROS

Unreal Engine 4 (UE4) is a tool used to design the visualization software of this project. There is no installation package for it on the Linux operating system, so it can only be installed by compiling its source code. Once installed, two more things needed to be done to enable communication between the UE4 and ROS. The first thing is to download the Rosbridge, which provides an API to ROS for non-ROS programs, enabling them to communicate with ROS. Another thing is to add the ROSIntegration plugin into UE4. This plugin provides the ability to communicate with a running roscore for UE4.

3.2 LiDAR-based SLAM Algorithm

The proposed LiDAR-based SLAM Algorithm can be divided into two parts, Laser Odometry (tracking) at the front end and Loop Closure Detection and Global Optimization (mapping) at the back end, which will be explained separately below.

3.2.1 Laser Odometry – Tracking

The Laser Odometer (tracking process) can be divided according to its functionality into several modules: initialization, valid point cloud acquisition (filtering), inter-frame point cloud matching, status (pose and map) update, and topic publication. The logical relationship between them is shown in Figure 3-1 below.

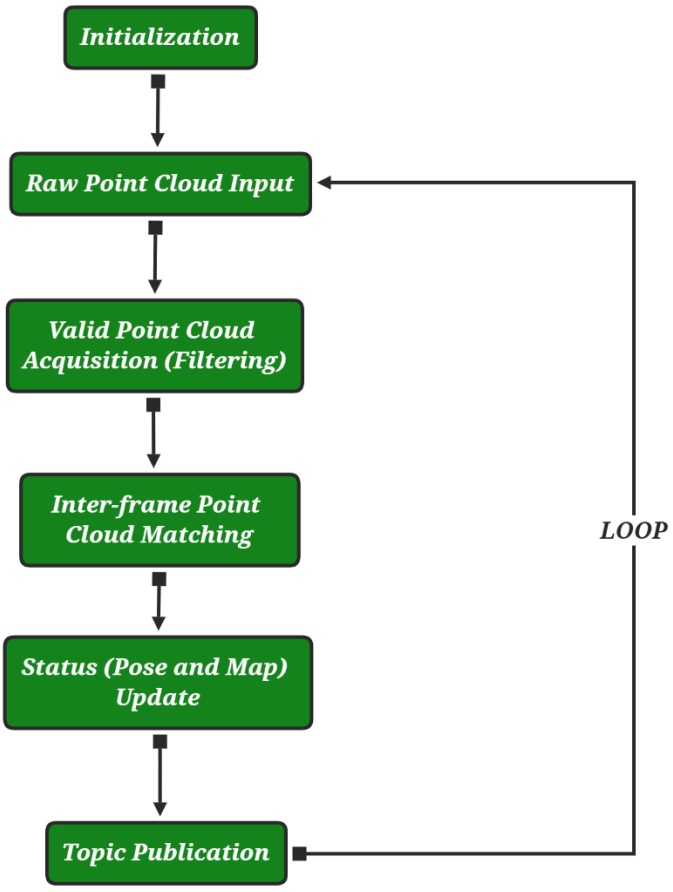


Figure 3-1 The Logical Relationship Between Modules in Laser Odometer

During the initialization stage, three steps are completed. The first step is to create and initialize a ros node, which contains the entire tracking process, by using the *init* function of ros. The second step is to build the entire communication structure of the tracking process, in

other words, to determine which topics need to be subscribed to and which need to be published. The communication structure is shown in Table 3-1.

Table 3-1 Communication Structure of Tracking Process

Topic Name	Communication Type	Function
$/livox/lidar$	Subscribe	Receive raw point cloud data from the LiDAR Sensor
$/pose$	Publish	Publish poses computed by the laser odometry
$/cloud$	Publish	Publish local maps computed by the laser odometry
$/feature$	Publish	Publish feature points of the latest scan that will be used in the global optimization process

The third step is to initialize the hyperparameters of the tracking process. To make it more convenient to modify hyperparameters, the yaml-cpp library is used, thus all hyperparameters can be obtained by loading the $.yaml$ configuration file. In this way, the hyperparameters can be changed in the configuration file directly, instead of modifying in the program which needs to be compiled again. Table 3-2 shows some important hyperparameters that play an essential role in the tracking process. All these hyperparameters will be repeatedly mentioned later.

Table 3-2 Important Hyperparameters in Tracking Process

Hyperparameters Name	Description
$resolution$	Minimum distance between neighboring points in the point cloud
$iterCount\_num$	Maximum iteration number of the point-to-plane ICP algorithm
$nearestKSearch\_num$	Number of points used to fit a plane in Inter-frame Point Cloud Matching part
max_d	Maximum normalized distance from the fitting points to the fitted plane, which determines whether the fitted plane qualifies
cal_curva_rate	Step used to search for feature points
lim	It determines the size of the local map

The next stage is valid point cloud acquisition. In this part, raw point cloud data for each frame is received, but this data cannot be used directly because the LiDAR sensor suffers from a detection failure problem. When it fails to scan a point, it sets the coordinates of that point to $(0,0,0)$. Since for the LiDAR the origin is on itself and it can never scan itself, it can be concluded that $(0,0,0)$ is an invalid location. Therefore, in the program, points that are located at the origin are filtered out and the rest are considered as valid points.

Then comes the inter-frame point cloud matching stage and status update stage, both of which will be presented together. Figure 3-2 shows the detailed algorithmic logic of these two sections.

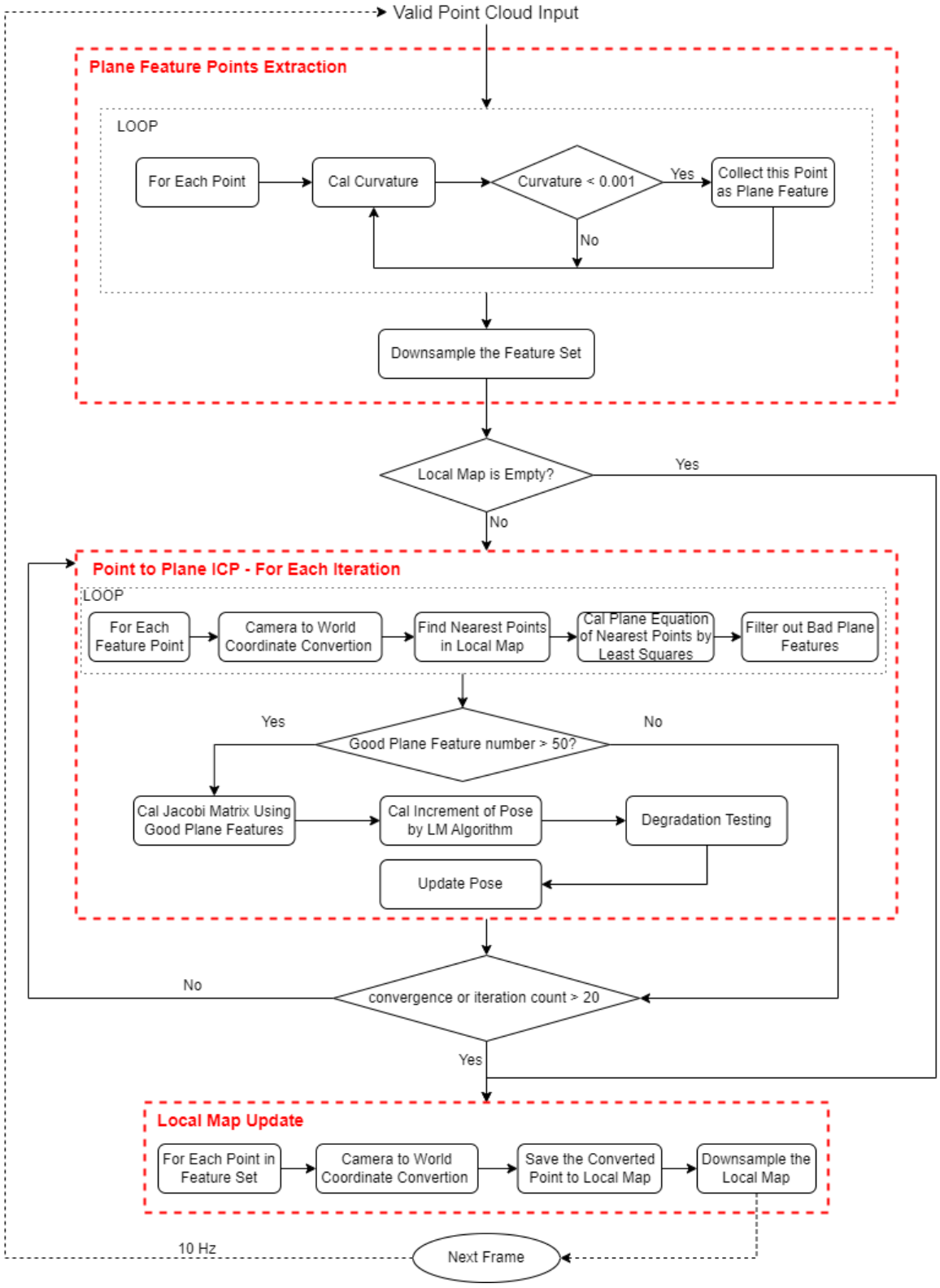


Figure 3-2 Algorithmic Logic of Inter-frame Point Cloud Matching and Status Update

From Figure 3-2, it can be seen that when the valid point cloud is fed in, a collection of planar feature points is first carried out. A planar feature point refers to a point that lies in a plane, which means the curvature at this point is relatively small. Figure 3-3 shows the arrangement of the points in the valid point cloud. According to this arrangement, the curvature at P_i can be represented by the sum of the differences in coordinates between this point and its neighboring points, as shown in Equation 3.1 below. As long as the curvature of a point is small enough (<0.001), it can be considered as a planar feature point.

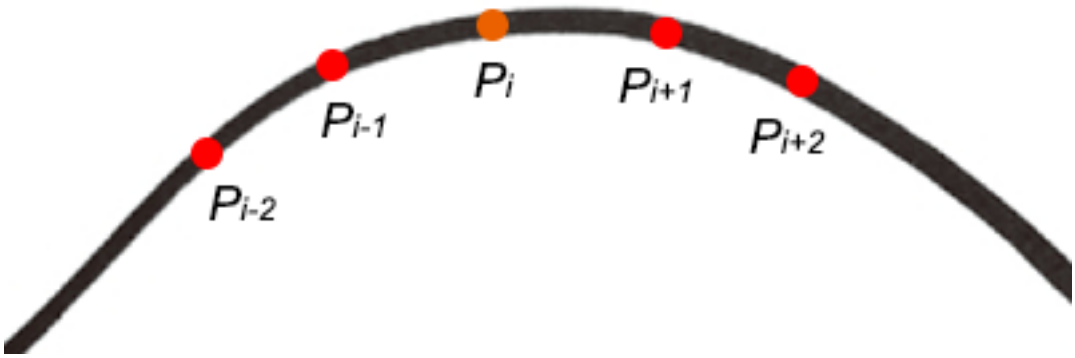


Figure 3-3 Arrangement of Points in Valid Point Cloud

$$crv(i) = \sum_{i-2}^{i+2}\left(x_i - x_j\right)^2 + \sum_{i-2}^{i+2}\left(y_i - y_j\right)^2 + \sum_{i-2}^{i+2}\left(z_i - z_j\right)^2 \tag{3.1}$$

Where $crv(i)$ represents the curvature at P_i, (x, y, z) represents the coordinate of P.

Considering real-time performance, two additional steps are made in the process of planar feature collection. Firstly, the step size for iterating through the valid point cloud when searching for planar feature points is controlled by the hyperparameter cal_curva_rate in Table 3-2. By giving this hyperparameter a large value, the computational cost can be reduced. Secondly, the feature set containing all planar feature points is down-sampled, which reduces the number of points while making them more evenly distributed.

The planar feature collection is followed by the Point-to-plane ICP process. In this process, for each iteration, firstly, each feature point will be converted from the LiDAR camera coordinate system to the world coordinate system by using the estimated pose T, as shown in Equation 3.2 – 3.5 below.

$$T = (\theta \quad \gamma \quad \varphi \quad t_x \quad t_y \quad t_z)^T \tag{3.2}$$

$$R = \begin{pmatrix} cos\gamma & 0 & sin\gamma \\ 0 & 1 & 0 \\ -sin\gamma & 0 & cos\gamma \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$t = (t_x \quad t_y \quad t_z)^T \tag{3.4}$$

$$P_{world}(x, y, z) = RP_{cam}(x, y, z) + t \tag{3.5}$$

The transformed feature points are in the same coordinate system as the local map. So, the closest points in the local map to the transformed feature points can be found. The hyperparameter $nearestKSearch\_num$ is used to determine the number of closest points selected. By applying the Least Squares method, these closest points are used to fit a plane as shown in Equation 3.6 below.

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \dots & \dots & \dots \end{pmatrix}^{-1} \begin{pmatrix} -1 \\ -1 \\ -1 \\ \dots \end{pmatrix} \tag{3.6}$$

Where A, B, C are the coefficients of fitted plane.

After fitting the plane, the distances to the fitted plane are calculated for all the nearest points involved in the plane fit process by using Equation 3.7 below. When all the distances are smaller than the threshold which is controlled by the hyperparameter max_d, this plane is considered a good plane, and this planar feature point is considered a good feature point.

$$d = \frac{|Ax+By+Cz+D|}{\sqrt{A^2+B^2+C^2}} \tag{3.7}$$

Where A, B, C are the coefficients calculated in Equation 3.6, and D is equal to 1.

The next step is to minimize the sum of distances between the transformed good feature points and their corresponding fitted planes by applying the Levenberg–Marquardt (LM) algorithm. The loss function is shown in Equation 3.8 below.

$$LF = \sum(Af_x(x_{cam}) + Bf_y(y_{cam}) + Cf_z(z_{cam}) + D)^2 \tag{3.8}$$

Where $f_x(x_{cam}) = x_{world}$, $f_y(y_{cam}) = y_{world}$, and $f_z(z_{cam}) = z_{world}$. f_x, f_y, f_z are the camera-to-world transform function shown in Equation 3.2 – 3.5 above.

Through the LM algorithm, the increment of pose, ΔT can be obtained by Equation 3.9 below.

$$\Delta T = -(J^T J + \lambda I)^{-1} J^T F \tag{3.9}$$

Where J represents the Jacobian matrix of the loss function, λ represents the damping factor, I represents the identity matrix, and F represents the distances between the transformed good feature points and their corresponding fitted planes calculated by current pose.

The Jacobian matrix of the loss function is calculated using Equation 3.10 below.

$$J = \begin{pmatrix} A_0 \frac{\partial f_x}{\partial \theta} + B_0 \frac{\partial f_y}{\partial \theta} + C_0 \frac{\partial f_z}{\partial \theta} & A_0 \frac{\partial f_x}{\partial \gamma} + B_0 \frac{\partial f_y}{\partial \gamma} + C_0 \frac{\partial f_z}{\partial \gamma} & A_0 \frac{\partial f_x}{\partial \varphi} + B_0 \frac{\partial f_y}{\partial \varphi} + C_0 \frac{\partial f_z}{\partial \varphi} & A_0 & B_0 & C_0 \\ A_1 \frac{\partial f_x}{\partial \theta} + B_1 \frac{\partial f_y}{\partial \theta} + C_1 \frac{\partial f_z}{\partial \theta} & A_1 \frac{\partial f_x}{\partial \gamma} + B_1 \frac{\partial f_y}{\partial \gamma} + C_1 \frac{\partial f_z}{\partial \gamma} & A_1 \frac{\partial f_x}{\partial \varphi} + B_1 \frac{\partial f_y}{\partial \varphi} + C_1 \frac{\partial f_z}{\partial \varphi} & A_1 & B_1 & C_1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_i \frac{\partial f_x}{\partial \theta} + B_i \frac{\partial f_y}{\partial \theta} + C_i \frac{\partial f_z}{\partial \theta} & A_i \frac{\partial f_x}{\partial \gamma} + B_i \frac{\partial f_y}{\partial \gamma} + C_i \frac{\partial f_z}{\partial \gamma} & A_i \frac{\partial f_x}{\partial \varphi} + B_i \frac{\partial f_y}{\partial \varphi} + C_i \frac{\partial f_z}{\partial \varphi} & A_i & B_i & C_i \end{pmatrix} \tag{3.10}$$

The damping factor is calculated using Equation 3.11 below.

$$\lambda = 1 - 0.9 \times \frac{|Ax+By+Cz+D|}{\sqrt{\sqrt{x^2+y^2+z^2}}} \tag{3.11}$$

Where (x, y, z) is the coordinate of the transformed good feature points.

Next is the degradation detection section. In this part, the eigenvalues and eigenvectors of $J^T J$ are calculated, and the fraction of degradation direction of ΔT is removed. Empirically, the degradation detection only needs to be run in the first iteration. After degradation detection, the pose is updated by Equation 3.12 below.

$$T = T + \Delta T \tag{3.12}$$

The iteration of the inter-frame point cloud matching and pose update stage will continue until the convergence occurs which means ΔT is very small, or the iteration count is larger than threshold controlled by the hyperparameter $iterCount\_num$. This hyperparameter needs to be chosen carefully, as it has a relatively large impact on the computational cost.

After the iteration is finished, the local map is updated. In this section, all the feature points are transformed from camera coordinate system to world coordinate system by the optimized pose, and then added to the local map. To avoid overlapping points and control the

size of local map, the local map is down sampled with the $resolution$, which is a hyperparameter shown in Table 3-2. In addition, the hyperparameter lim is used to control the positional range of the local map.

Finally, topics that contain the optimized pose, updated local map, and feature points are published, and the tracking program is waiting for the next frame.

3.2.2 Loop Closure Detection and Global Optimization – Mapping

The mapping process can be divided into initialization, history information acquisition, loop closure detection, graph-based optimization, and topic publication. The logical relationship between them is shown in Figure 3-4 below. When initialization finishes, two threads start at the same time, one for history information acquisition and the other for the rest of modules, with data exchange between them.

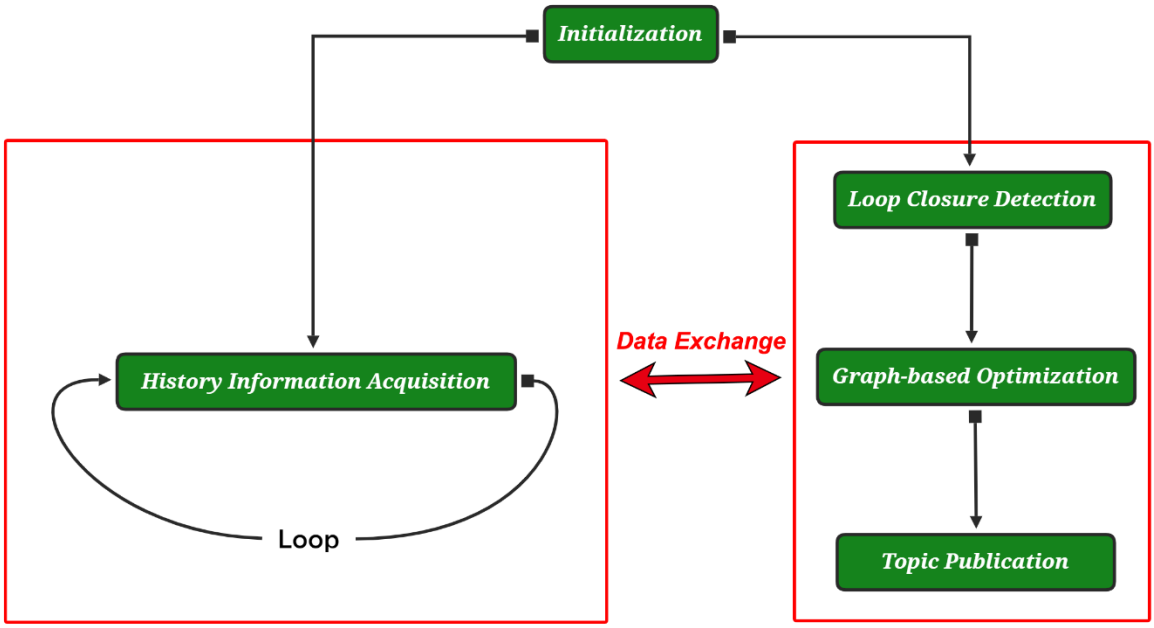


Figure 3-4 Logic of Mapping Process

The initialization of the mapping process is similar to the tracking process as it also involves the creation of a ros node, the construction of a communication structure and the definition of hyperparameters. However, the content of the communication structure and hyperparameters is different from that of tracking process. The communication structure and important hyperparameters used in this process are shown in Table 3-3 and Table 3-4 respectively.

Table 3-3 Communication Structure of Mapping Process

Topic Name	Communication Type	Function
$/pose$	Subscribe	Receive unoptimized poses from tracking process
$/cloud$	Subscribe	Receive local maps from tracking process
$/feature$	Subscribe	Receive feature points from tracking process
$/optpose$	Publish	Publish optimized poses
$/globalmap$	Publish	Publish optimized global map

Table 3-4 Important Hyperparameters in Mapping Process

Hyperparameters Name	Description
$closure\_search\_radius$	Search radius for historical loopback frame in loop closure detection part
$sec\_difference$	Minimum time difference between the latest frame and its historical loopback frame
$resolutionICP$	Resolution of local maps used in the ICP process
$score\_thre$	Maximum mean squared error between local maps of the latest frame and its historical loopback frame during the ICP process
$resolution$	Resolution of optimized global map

After initialization, two threads, including a data receiving thread and a main thread, begin. The data receiving thread is responsible for receiving and storing the unoptimized pose, local map, and feature points for each frame published by the tracking node and corresponding them with a timestamp so that the data from the same frame will have the same timestamp. This thread will not stop as long as the tracking node is working. In the main thread, all data collected in the data receiving thread will be processed in order to detect loop closures and improve the poses and global map. In addition, the main thread makes use of a number of libraries. The Octomap library is utilized for global map storage, the PCL library for point cloud processing, and the GTSAM library for graph-based optimization.

Figure 3-5 depicts the logic in the main thread. First, the latest unoptimized pose in the data pool is used to construct a pose graph, with its pose as a node in the graph and its relative relationship to the prior pose as a constraint. Following that, a loopback coarse detection is conducted. In this stage, the closest poses to the latest pose are selected, where the selection radius is determined by the hyperparameter, $closure\_search\_radius$. To eliminate the influence of adjacent previous frames, the time differences between the latest pose and its nearest poses are calculated, and only if the time difference exceeds a threshold set by the

hyperparameter $sec\_difference$, is this nearest pose considered to be a possible loopback for the latest pose.

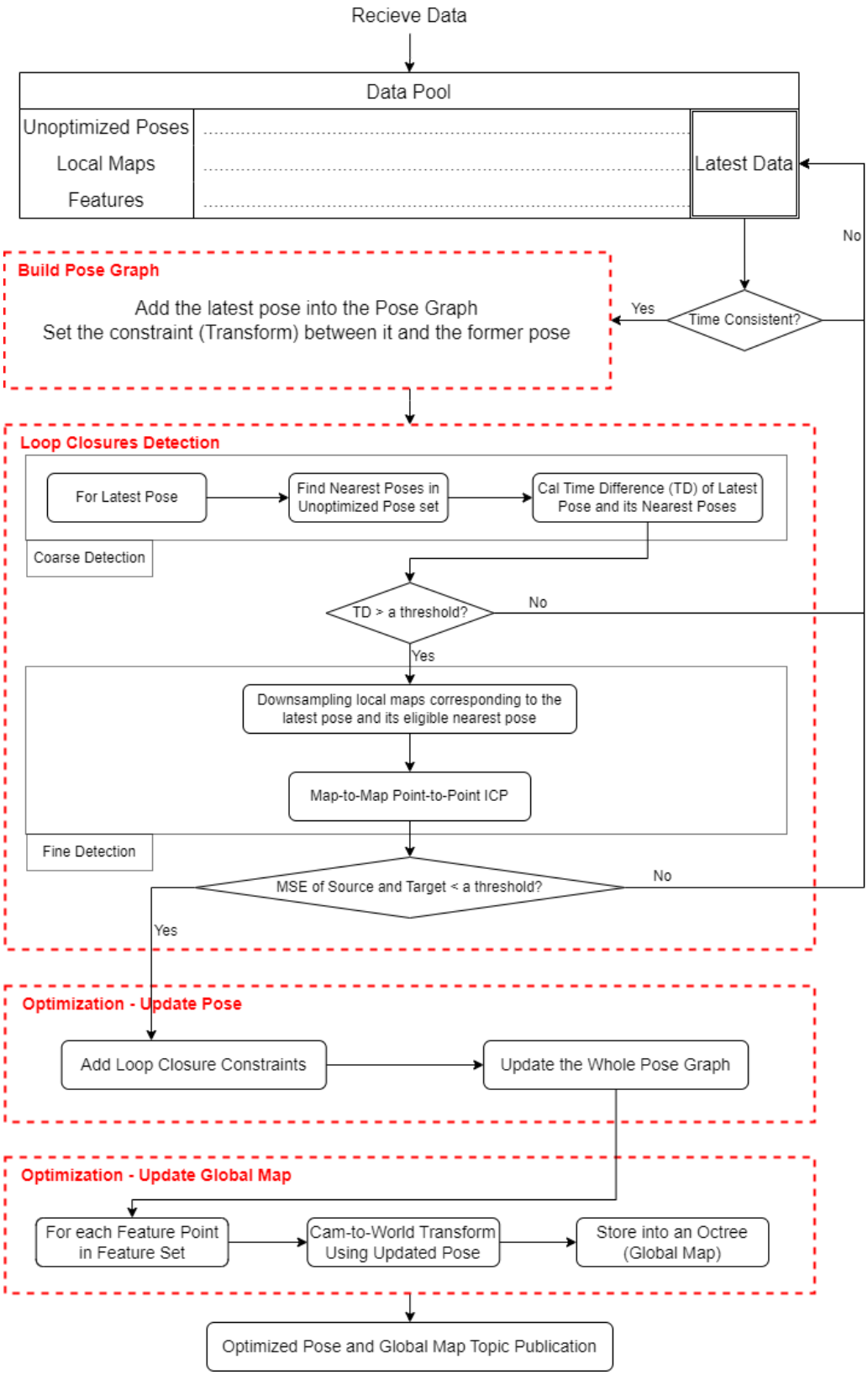


Figure 3-5 Logic of Main Thread in Mapping Process

The loopback fine detection step comes next. The map-to-map point-to-point ICP technique is used in this section to assess if a historical pose is a loopback of the latest posture. However, the map-to-map approach applies ICP with a huge number of points, which has an incredibly high computational cost that our real-time system cannot afford. To address this issue, a down sampling operation is performed on both the source and target local maps before the ICP operation, allowing the overall number of points to be lowered without affecting their underlying structure. Note that the resolution of down sampling operation is controlled by the hyperparameter $resolutionICP$.

When a loopback is found, the loop closure constraint between the latest pose and the loopback pose will be added into the pose graph, and the whole pose graph is then optimized based on this constraint. After all the poses are optimized, the global map can be reconstructed by transforming the feature points from camera coordinate system to world coordinate system using their corresponding poses. The global map is stored in an octree to increase real-time performance and decrease storage memory costs. Its accuracy is determined on the hyperparameter $resolution$.

3.3 Attitude and Heading Reference System Integration

An off-the-shelf AHRS module is integrated into the SLAM system to significantly decrease the accumulated error generated by the laser odometer. This module employs an extended Kalman filter method to provide attitude information represented as a quaternion at frequencies up to 200Hz, and it is capable of transmitting the data over the ROS. Because the AHRS estimate of rotation is far more precise than the ICP method, the original algorithm's rotation information in the laser odometer is overridden by the AHRS output, but the translation information stays unaffected. This results in a significant reduction in the cumulative error in rotation. Furthermore, the addition of AHRS solves the problem that the original algorithm fails to match between frames when the laser sensor moves too fast, resulting in abrupt changes in pose. This greatly improves the SLAM system's robustness when moving at high speed. Figure 3-6 depicts the integration logic for AHRS and laser odometry in the program. The rotation information from the AHRS module is received and stored in a flow window buffer which can always contain rotation information from the last five seconds. The algorithm in laser odometry is unchanged, which still estimates both rotation information and translation

information. The sole difference is that the rotation information estimated by the original algorithm is replaced with the one in the flow window buffer that has a closest timestamp.

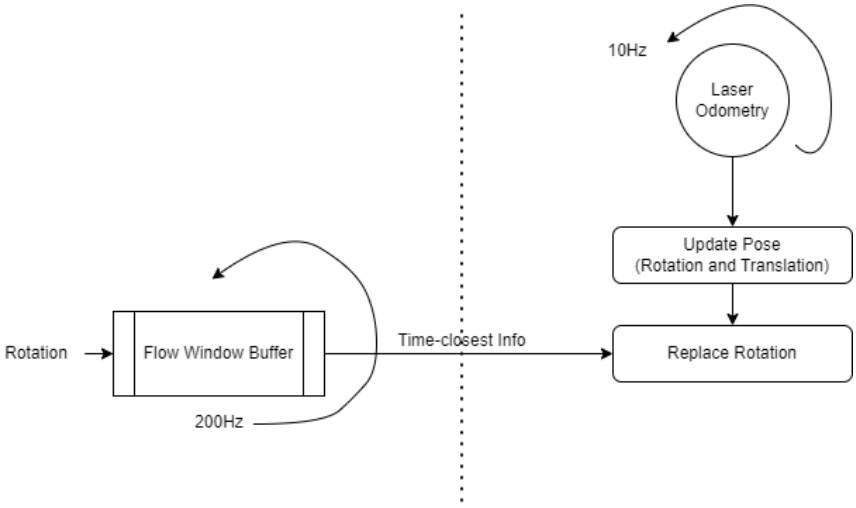


Figure 3-6 Integration Logic for AHRS and Laser Odometry

3.4 Visualization Tools

Three different visualization tools are used at different stages of the project to show and compare the performance (including maps and poses) of the SLAM system from multiple perspectives and to aid debugging.

3.4.1 RViz

RViz is a visualization tool included with ROS. It is able to be manually configured to receive topics published by other ROS nodes and has been integrated with a wide range of data types in ROS. As long as RViz supports the data types of the topics, the data can be presented immediately in the display area in the form provided by RViz. Furthermore, the presentation of these data can be altered within certain limits, such as the size of the points in the point cloud and the display style of the poses. In this project, RViz is the primary visualization tool, and it is used to display a wide range of data, including poses and maps, for debugging and presentation purposes.

3.4.2 Pangolin

Pangolin is a popular C++ 3D visual drawing toolkit that can be used to effortlessly construct point clouds. In the early stages of this project's algorithm development, this library

is frequently used to debug code on point cloud data processing. Unlike RViz, it does not require a topic subscription to obtain data, and the data needed for presentation can be retrieved straight from the program. It is also more flexible and offers a broader range of display methods. This project, however, favors RViz for large-scale presentation of multiple data types since Pangolin is considerably less integrated and much more difficult to use.

3.4.3 3D game by Unreal Engine

In addition to RViz and Pangolin, a 3D game built using Unreal Engine technology was developed to improve the presentation of maps and poses and to provide users with a more participatory experience. This game, unlike the previous two developer-oriented visualization tools, is fully user-oriented.

To enrich this 3D game, several different functions have been developed. The first one is the camera perspective switch function. For this function, the user can switch the observing perspective by pressing the Tab key. There are now two perspectives available: a God perspective and a third-person perspective with the bear as the subject. In the God view, the user can see all of the details displayed in the game's 3D space, from a single point to the entire map; while in the third-person perspective, the user can control a bear to move across the map and enjoy the map from the bear's view. The second one is the pose display function. To provide a more comprehensible representation of the rotation information in the poses provided by the SLAM system, 3DUI technology is employed to construct a massive screen in the game environment, which is used to display a massive Cartesian coordinate axis that changes in real time. This axis's origin remains constant, and its rotation direction represents the rotation information output by the SLAM system. Users can observe the rotation more visually by gazing at the screen. For the translational information in poses, a real-time updated roadmap is displayed in the 3D space of the game. The user can observe this roadmap to determine whether loop closures occur and to visualize the effects of back-end global optimization. The third one is the ability of real-time point cloud display and update at runtime. This functionality makes the display of local and global maps a reality. Users can get a clearer view of the impact of global optimization on the global map.

Chapter 4 Tests, Results and Discussions

4.1 Comparison of the Proposed SLAM Algorithm with Other Works

This section will compare the performance of the SLAM algorithm proposed in this research with some of the other popular Laser SLAM methods. In this paper, three scenarios will be compared: an open-source indoor environment dataset from the web, an indoor environment dataset scanned by myself using LiDAR sensor, and an open-source outdoor environment datasets from the web. The proposed SLAM algorithm will be compared to the existing mainstream algorithm, LOAM-SLAM, in each of these circumstances.

4.1.1 Indoor 3D Scenario Dataset from Web

(a) Configurations

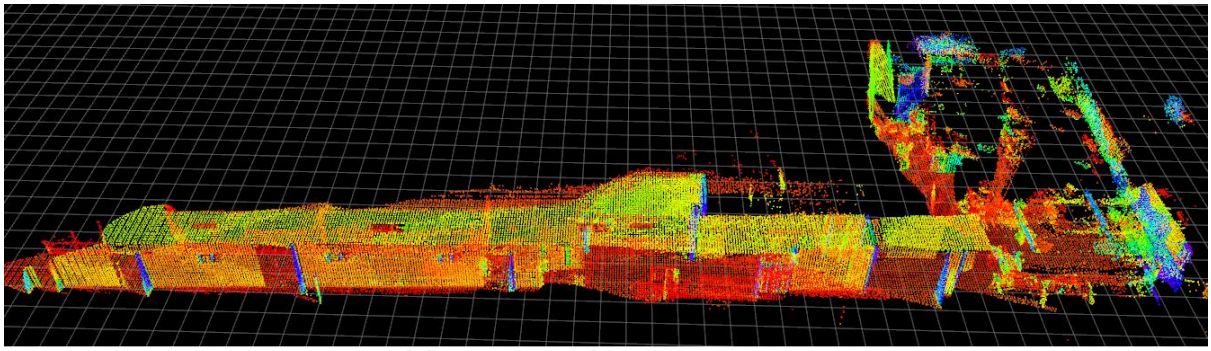
In this section, an indoor laser dataset was downloaded from the Internet. The scene of this dataset is a corridor in an office building. Because this dataset lacks IMU data, the AHRS module of the proposed algorithm is not activated in this case. In addition, there is no loop closure in this scene, so the global optimization part is also not activated. Therefore, this section is a comparison between LOAM-SLAM and the laser odometry part of the proposed algorithm. The hyperparameter configuration for the proposed algorithm is displayed in Table 4-1.

Table 4-1 Hyperparameter Configuration of Proposed Algorithm for Indoor Internet Dataset

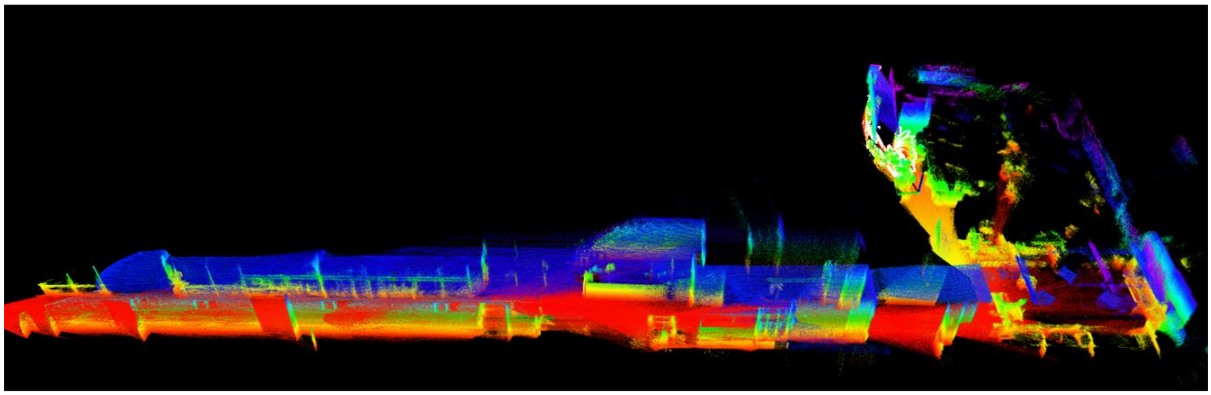
Tracking	$resolution$	0.15	Mapping	$closure\_search\_radius$	5
	$iterCount\_num$	15		$sec\_difference$	30
	$nearestKSearch\_num$	8		$resolutionICP$	0.8
	max_d	0.2		$score\_thre$	0.3
	cal_curva_rate	1		$resolution$	0.1
	lim	30			

(b) Results

Figure 4-1 depicts the global map of the office building's corridor produced by both the proposed method and LOAM-SLAM, and Figure 4-2 depicts their poses.

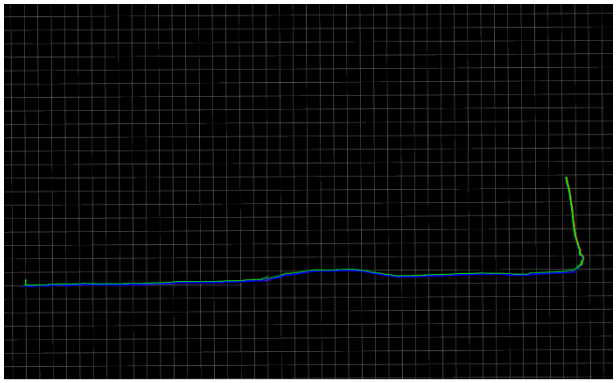


(a) Proposed SLAM Algorithm

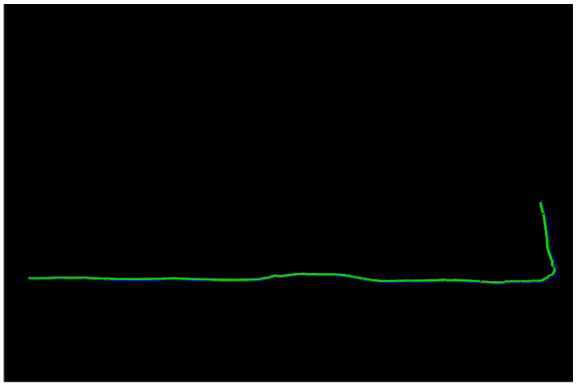


(b) LOAM-SLAM Algorithm

Figure 4-1 Comparison of Global Maps for Indoor Internet Dataset



(a) Proposed SLAM Algorithm



(b) LOAM-SLAM Algorithm

Figure 4-2 Comparison of Poses for Indoor Internet Dataset

As shown in Figures 4-1 and 4-2, the algorithm proposed in this project can achieve similar results to LOAM-SLAM in terms of pose estimation, when the AHRS and back-end optimization modules of it are not utilized. However, in terms of map building, the proposed algorithm's global map is rather crude in the details and does not look as good as the map built by LOAM-SLAM. But in most SLAM application scenarios, whether the map looks good or not is unimportant; what matters is its accuracy.

4.1.2 Indoor 3D Scenario Dataset from LiDAR Sensor

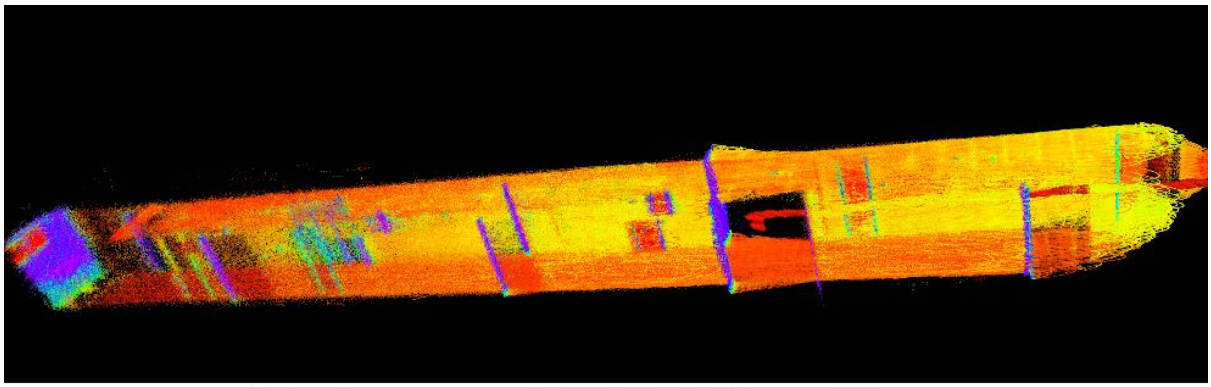
(a) Configurations

In this section, an indoor laser dataset is collected by me using the Mid-40 LiDAR sensor. The scene of this dataset is the hallway of the student dormitory, which is shown in Appendix. The hyperparameter configuration for this scene is displayed in Table 4-2 as below.

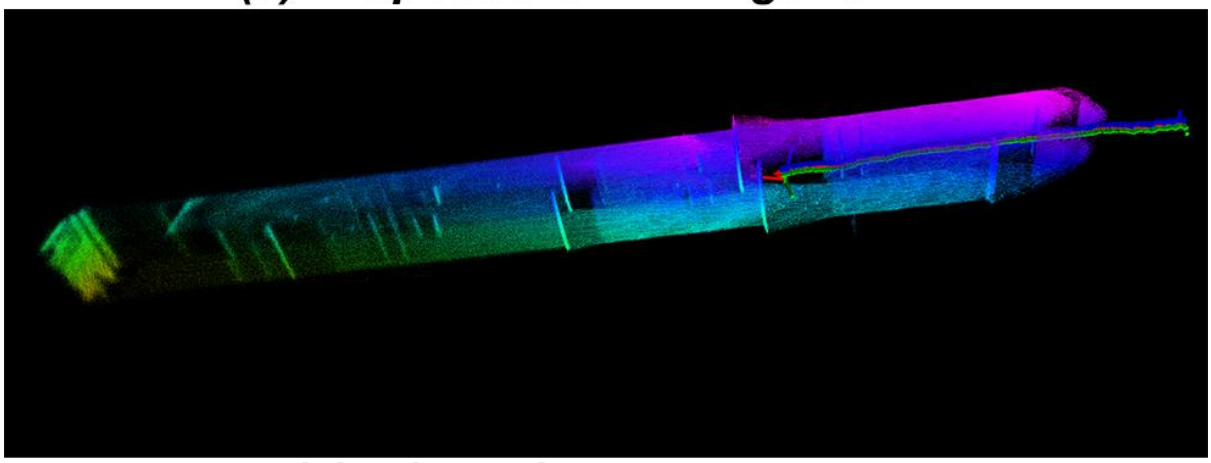
Table 4-2 Hyperparameter Configuration of Proposed Algorithm for Indoor Self-Collected Dataset

Tracking	$resolution$	0.15	Mapping	$closure\_search\_radius$	5
	$iterCount\_num$	15		$sec\_difference$	30
	$nearestKSearch\_num$	8		$resolutionICP$	0.8
	max_d	0.1		$score\_thre$	0.3
	cal_curva_rate	1		$resolution$	0.4
	lim	30			

(b) Results

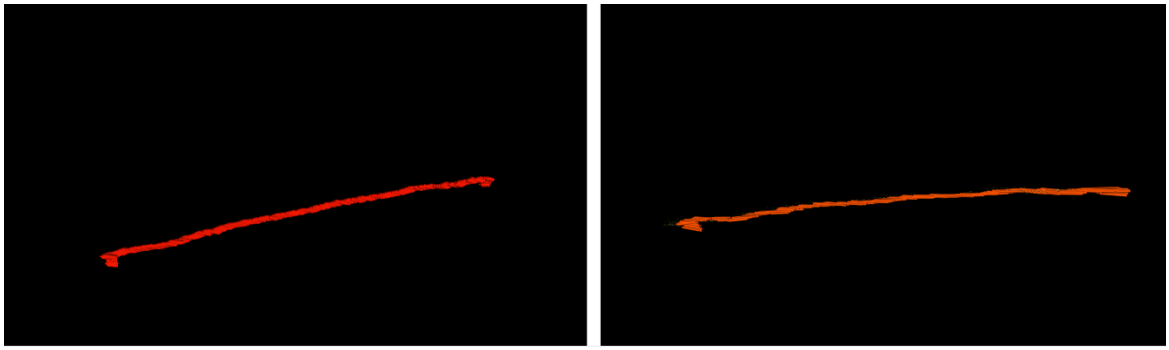


(a) Proposed SLAM Algorithm



(b) LOAM-SLAM Algorithm

Figure 4-3 Comparison of Global Maps for Self-Collected Indoor Dataset



(a) Proposed SLAM Algorithm (b) LOAM-SLAM Algorithm

Figure 4-4 Comparison of Poses for Self-Collected Indoor Dataset

Figure 4-3 and 4-4 shows the global maps and poses estimated by both the proposed SLAM algorithm and the LOAM-SLAM algorithm. It can be seen that the results of using this self-collected dataset are same as the one using the dataset from Web in 4.1.1 above.

4.1.3 Outdoor 3D Scenario Dataset from Web

(a) Configurations

The scenario in the KITTI Odometry Dataset (Velodyne laser data) with sequence number 07, which is a road scene, is used in this section. The Ground Truth (GT) of the LiDAR's pose in the scene 07 is shown in Figure 4-5 as below.

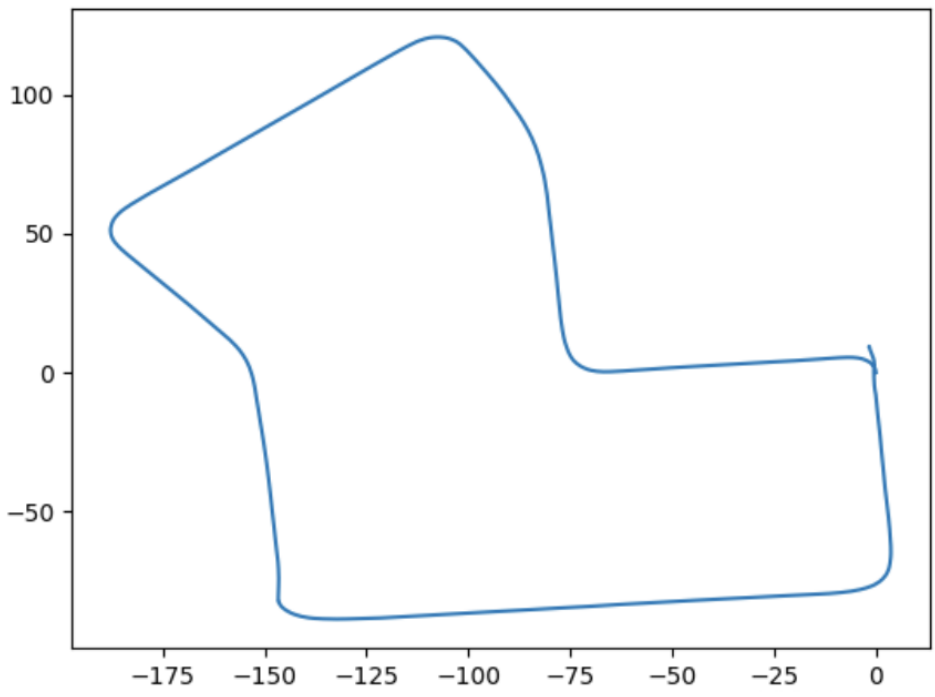


Figure 4-5 GT of LiDAR's Pose in 07 Sequence

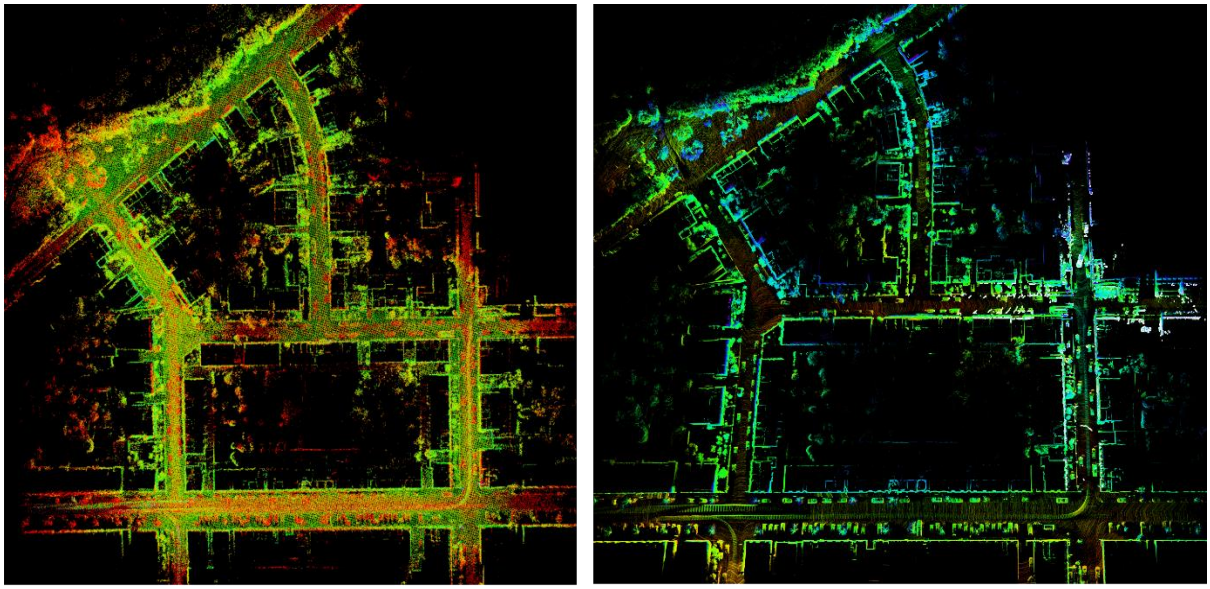
Besides, the hyperparameter configuration of the proposed SLAM algorithm is shown in Table 4-3 below, while the default configuration is used for the LOAM-SLAM algorithm.

Table 4-3 Hyperparameter Configuration of Proposed Algorithm for Outdoor Internet Dataset

Tracking	$resolution$	0.4	Mapping	$closure\_search\_radius$	5
	$iterCount\_num$	15		$sec\_difference$	30
	$nearestKSearch\_num$	8		$resolutionICP$	0.8
	max_d	0.1		$score\_thre$	0.3
	cal_curva_rate	4		$resolution$	0.25
	lim	30			

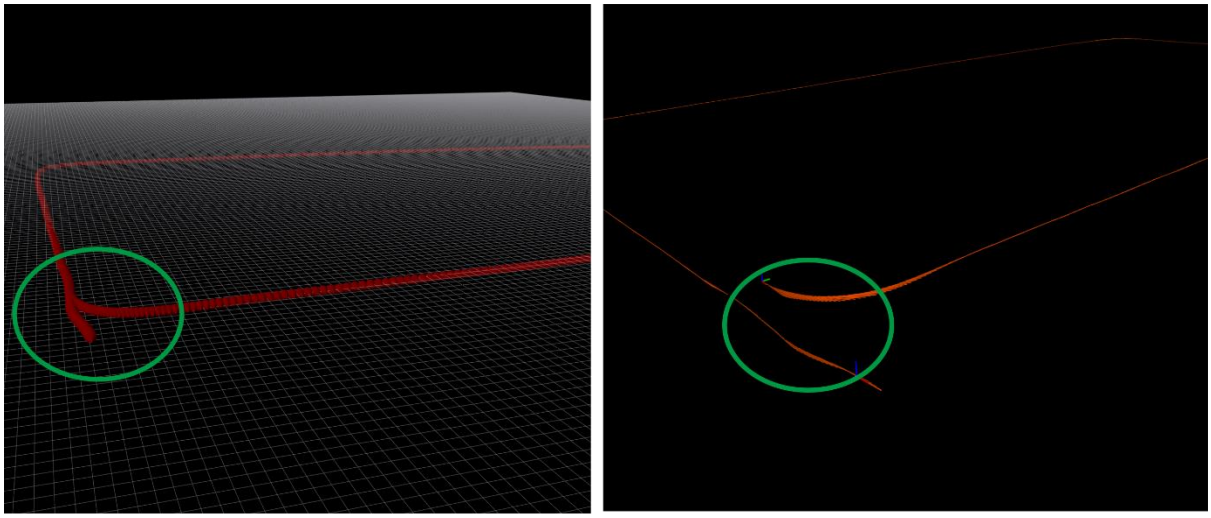
(b) Results

Figure 4-6 shows the global map of the 07 Sequence scene generated by both the proposed SLAM algorithm and the LOAM-SLAM algorithm, and Figure 4-7 shows the poses of both algorithm near the loopback.



(a) Proposed SLAM Algorithm *(b) LOAM-SLAM Algorithm*

Figure 4-6 Global Map Comparison of Proposed SLAM Algorithm and LOAM-SLAM Algorithm



(a) Proposed SLAM Algorithm (b) LOAM-SLAM Algorithm

Figure 4-7 Poses Comparison of Proposed SLAM Algorithm and LOAM-SLAM Algorithm

The global maps created by the SLAM algorithm proposed in this study and the LOAM-SLAM method for this KITTI dataset are quite comparable in terms of overall structure, as shown in Figure 4-6. The LOAM-SLAM technique omits the ground when constructing the map, making it appear briefer, however the proposed approach keeps as much detail as possible and is more informative.

Figure 4-7 shows the superiority of the proposed SLAM algorithm compared to LOAM-SLAM. Because the LOAM-SLAM algorithm lacks a back-end optimization module, a large number of cumulative errors would be generated when running for a long time, resulting in the phenomenon depicted in Figure 4-7 (b). However, as shown in Figure 4-7 (a), the proposed algorithm performs back-end optimization as it finally detects the loopback, thereby eliminating the cumulative error.

4.2 Unreal Engine 3D Game Display

Figures 4-8, 4-9, 4-10 and 4-11 show the display of this 3D game.

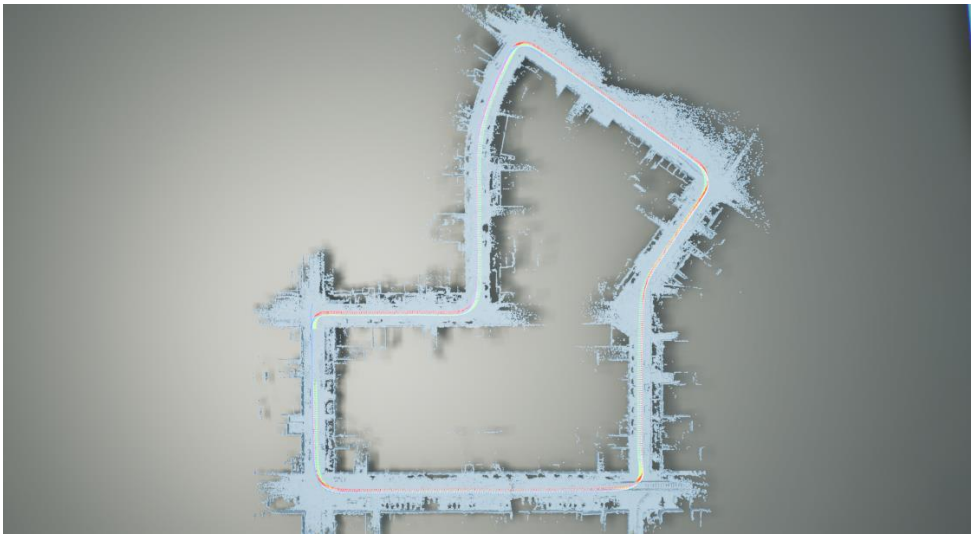


Figure 4-8 3D Game Display 1

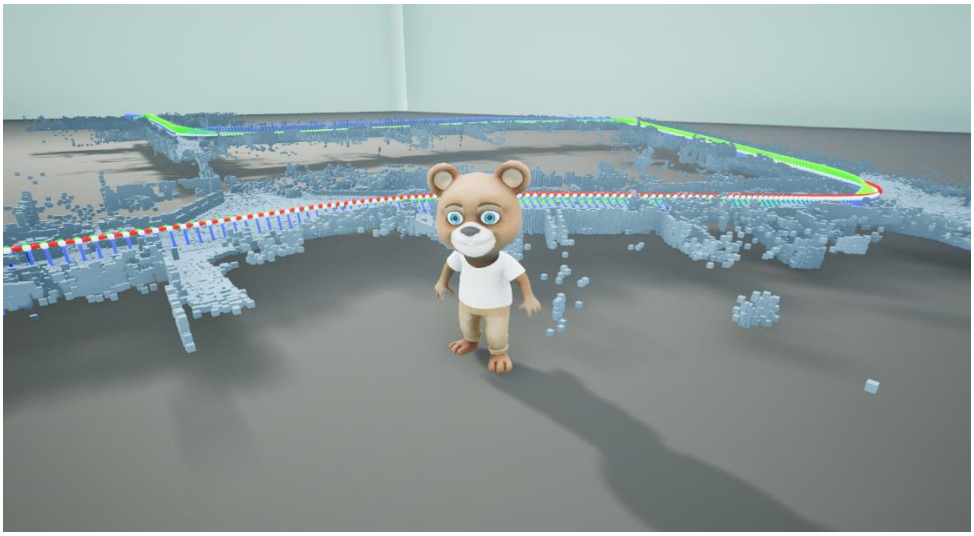


Figure 4-9 3D Game Display 2

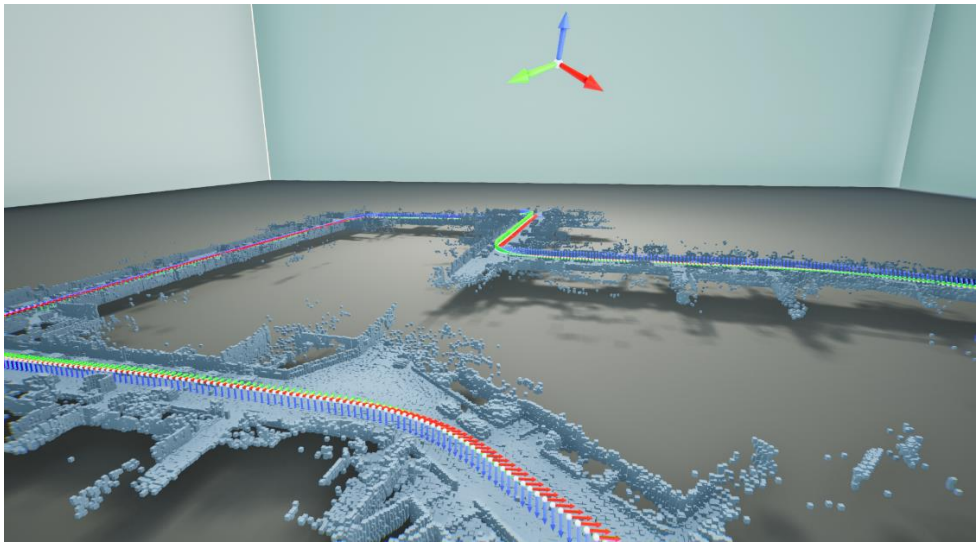


Figure 4-10 3D Game Display 3

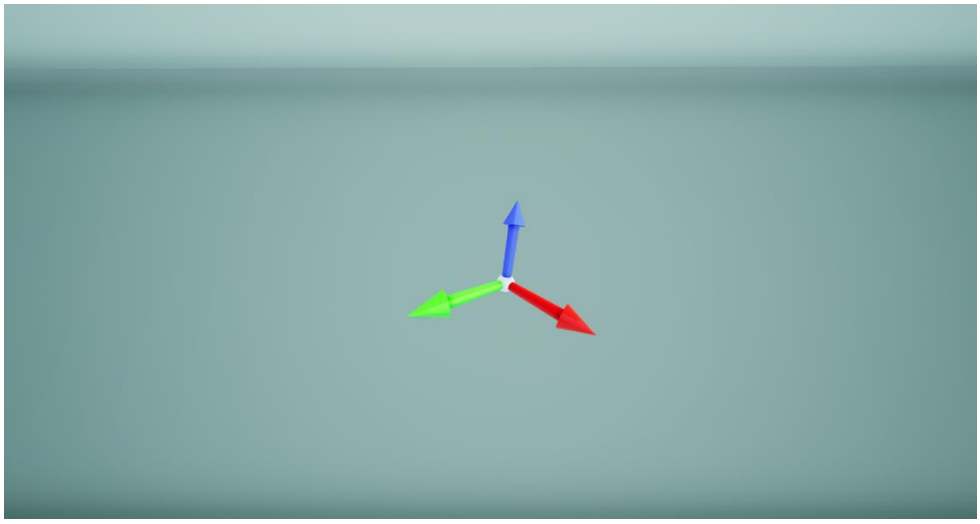


Figure 4-11 3D Game Display 4

4.3 Time Cost of Algorithms

In this section, the real-time performance of the proposed SLAM system is examined. During the examination the whole algorithm is divided into three parts, including Laser Odometry, Loop Closure Detection, and Global Optimization, and the time costs of each part are measured and discussed.

4.3.1 Laser Odometry

As illustrated in Figure 3-1, the Laser Odometer is separated into several submodules, including initialization, valid point cloud acquisition, inter-frame point cloud matching, status update, and topic publication. The time cost of each of these submodules (except initialization) and the overall time cost are shown in Table 4-4 below.

Table 4-4 Time Costs of Laser Odometry

Submodule	Time Cost (s)
Valid Point Cloud Acquisition	0.016
Inter-frame Point Cloud Matching	0.028
Status Update	0.001
Topic Publication	0.001
Overall	0.046

As shown in Tables 4-4, the Valid Point Cloud Acquisition and Inter-frame Point Cloud Matching modules consume the bulk of time in Laser Odometry, whereas the Status Update

and Topic Publication modules consume very little. Besides, the overall time cost of Laser Odometry is less than half of the input rate (0.1s), which means that the Laser Odometry part has good real-time performance.

4.3.2 Loop Closure Detection

As shown in Figure 3-5, the Loop Closure Detection algorithm can be divided into two submodules, including coarse detection and fine detection. These two submodules' time cost are shown in Table 4-5 as below.

Table 4-5 Time Costs of Loop Closure Detection

Submodule	Time Cost (s)
Coarse Detection	0.0001
Fine Detection	0.1
Overall	0.1001

Table 4-5 shows that the time cost of Coarse Detection is negligible, while the Fine Detection spends the majority of the time. In terms of the overall time cost, 0.1001 seconds is within tolerance of a loop closure detection algorithm.

4.3.3 Global Optimization

Figure 3-5 also shows that the Global Optimization is consist of four parts, including Pose Graph construction, Pose Optimization, Global Map Optimization, and Topic Publication. The time costs of them are displayed in Table 4-6.

Table 4-6 Time Costs of Global Optimization

Submodule	Time Cost (s)
Pose Graph construction	0.0002
Pose Optimization	0.02
Global Map Optimization	1.5
Topic Publication	0.27
Overall	1.7902

As shown in Table 4-6, Global Map Optimization and Topic Publication use most of the time in the Global Optimization method, particularly Global Map Optimization, which took

1.5 seconds. This is owing to the vastness of the Global Map and the enormous quantity of point clouds. However, this 1.5 second is already the time cost of the improved algorithm version, and the initial version consumes 44 seconds. The total time cost of Global Optimization method is 1.79 seconds, which has a negative impact on the real-time performance of the entire SLAM system.

Chapter 5 Conclusion

5.1 Summary

In conclusion, a real time Laser SLAM system, which is capable of 3D model reconstruction and self-localization, is successfully designed, implemented, and validated in this project. The whole system is divided into three stages. The first one is the laser odometry algorithm, which estimates the pose of the robot by a frame-to-frame matching algorithm and draws a local map based on these poses. In the second stage, the loop closure detection algorithm, which detects the loopbacks by comparing the similarity of two different local maps, is applied. The last stage is global optimization algorithm. In this stage the graph-based optimization method is used to update the poses of the robot and the global map is built based on the updated poses. In addition, a fully user-oriented 3D game is developed to improve the presentation of maps and poses and to provide users with a more participatory experience.

The proposed SLAM algorithm is compared with the LOAM-SLAM algorithm in three different scenarios, including two indoor scenes and one outdoor scene. These comparisons reveal that in terms of pose estimation, the proposed algorithm is more accurate than LOAM-SLAM due to its global optimization algorithm; in terms of map construction, the proposed algorithm creates a map that is coarser than LOAM-SLAM but contains more information. Furthermore, because of its capacity to reduce cumulative errors, which LOAM-SLAM lacks, the proposed technique can tackle the problem of global map inconsistencies. In terms of the algorithm's real-time performance, the laser odometry part and loop closure detection part Meet real-time requirements. However, Global Optimization part has a relatively lager time cost which has a negative impact on the entire SLAM system.

5.2 Future Works

Future works will focus on the improvement of real-time performance, robustness, accuracy of the proposed SLAM system. Furthermore, a system for intelligent selection of hyperparameters will be developed to adapt to different complex environments and to cope with changes in the environment.

References

[1] H. Taheri and Z. C. Xia, "SLAM; definition and evolution," Engineering Applications of Artificial Intelligence, vol. 97, p. 104032, 2021/01/01/ 2021, doi: https://doi.org/10.1016/j.engappai.2020.104032.

[2] C. Theodorou, V. Velisavljevic, V. Dyo, and F. Nonyelu, "Visual SLAM algorithms and their application for AR, mapping, localization and wayfinding," Array, vol. 15, p. 100222, 2022/09/01/ 2022, doi: https://doi.org/10.1016/j.array.2022.100222.

[3] P. Li, R. Wang, Y. Wang, and W. Tao, "Evaluation of the ICP Algorithm in 3D Point Cloud Registration," IEEE Access, vol. 8, pp. 68030-68048, 2020, doi: 10.1109/ACCESS.2020.2986470.

[4] J. B. Paul and D. M. Neil, "Method for registration of 3-D shapes," in Proc.SPIE, 1992, vol. 1611, pp. 586-606, doi: 10.1117/12.57955. [Online]. Available: https://doi.org/10.1117/12.57955

[5] F. Wang and Z. Zhao, "A survey of iterative closest point algorithm," in 2017 Chinese Automation Congress (CAC), 20-22 Oct. 2017 2017, pp. 4395-4399, doi: 10.1109/CAC.2017.8243553.

[6] A. Censi, "An ICP variant using a point-to-line metric," in 2008 IEEE International Conference on Robotics and Automation, 19-23 May 2008 2008, pp. 19-25, doi: 10.1109/ROBOT.2008.4543181.

[7] KL. Low, "Linear least-squares optimization for point-to-plane icp surface registration," Chapel Hill, University of North Carolina, vol. 4, no. 10, pp. 1-3, 2004.

[8] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in Proceedings Third International Conference on 3-D Digital Imaging and Modeling, 28 May-1 June 2001 2001, pp. 145-152, doi: 10.1109/IM.2001.924423.

[9] J. F. Guerrero-Castellanos, H. Madrigal-Sastre, S. Durand, N. Marchand, W. F. Guerrero-Sánchez, and B. B. Salmerón, "Design and implementation of an Attitude and Heading Reference System (AHRS)," in 2011 8th International Conference on Electrical Engineering, Computing Science and Automatic Control, 26-28 Oct. 2011 2011, pp. 1-5, doi: 10.1109/ICEEE.2011.6106610.

[10] Z. Wang, Y. Shen, B. Cai, and M. T. Saleem, "A Brief Review on Loop Closure Detection with 3D Point Cloud," in 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR), 4-9 Aug. 2019 2019, pp. 929-934, doi: 10.1109/RCAR47638.2019.9044021.

[11] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in 2016 IEEE International Conference on Robotics and Automation (ICRA), 16-21 May 2016 2016, pp. 1271-1278, doi: 10.1109/ICRA.2016.7487258.

[12] S. Yang, C. Geng, M. Li, J. Liu, and F. Cui, "Improved Cartographer Algorithm Based on Map-to-Map Loopback Detection," in 2022 6th CAA International Conference on

Vehicular Control and Intelligence (CVCI), 28-30 Oct. 2022 2022, pp. 1-5, doi: 10.1109/CVCI56766.2022.9964502.

[13] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," IEEE Intelligent Transportation Systems Magazine, vol. 2, no. 4, pp. 31-43, 2010, doi: 10.1109/MITS.2010.939925.

[14] F. Lu and E. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," Autonomous Robots, vol. 4, no. 4, pp. 333-349, 1997/10/01 1997, doi: 10.1023/A:1008854305733.

[15] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," IEEE Transactions on Robotics and Automation, vol. 17, no. 3, pp. 229-241, 2001, doi: 10.1109/70.938381.

[16] Y. Zhang, T. Zhang, and S. Huang, "Comparison of EKF based SLAM and optimization based SLAM algorithms," in 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), 31 May-2 June 2018 2018, pp. 1308-1313, doi: 10.1109/ICIEA.2018.8397911.

[17] Q. Wang and M.-K. Kim, "Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018," Advanced Engineering Informatics, vol. 39, pp. 306-319, 2019/01/01/ 2019, doi: https://doi.org/10.1016/j.aei.2019.02.007.

[18] A. Elfes, "Occupancy grids: A stochastic spatial representation for active robot perception," in Proceedings of the Sixth Conference on Uncertainty in AI, 1990, vol. 2929: Morgan Kaufmann San Mateo, CA, p. 6.

[19] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," Aaai/iaai, vol. 593598, 2002.

[20] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in IJCAI, 2003, vol. 3, no. 2003, pp. 1151-1156.

[21] S. Thrun and M. Montemerlo, "The graph SLAM algorithm with applications to large-scale mapping of urban structures," The International Journal of Robotics Research, vol. 25, no. 5-6, pp. 403-429, 2006.

[22] F. Duchoň, J. Hažík, J. Rodina, M. Tölgyessy, M. Dekan, and A. Sojka, "Verification of slam methods implemented in ros," Journal of Multidisciplinary Engineering Science and Technology (JMEST), vol. 6, no. 9, pp. 2458-9403, 2019.

[23] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in Robotics: Science and Systems, 2014, vol. 2, no. 9: Berkeley, CA, pp. 1-9.

[24] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1-5 Oct. 2018 2018, pp. 4758-4765, doi: 10.1109/IROS.2018.8594299.

[25] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 24 Oct.-24 Jan. 2021 2020, pp. 5135-5142, doi: 10.1109/IROS45743.2020.9341176.

Appendix