SLAM-RAMU: 3D LiDAR-IMU Lifelong SLAM with Relocalization and Autonomous Map Updating for Accurate and Reliable Navigation

Bushi Chen, Xunyu Zhong, Han Xie, Pengfei Peng, Huosheng Hu, Xungao Zhong, Qiang Liu

Abstract

Purpose: Autonomous Mobile Robots (AMRs) play a crucial role in industrial and service fields. However, the LiDAR-based Simultaneous Localization and Mapping (SLAM) used by AMRs encounters challenges in dynamic and changing environments.

Design/methodology/approach: This research introduces SLAM-RAMU, a Lifelong SLAM system that addresses these challenges by providing precise and consistent relocalization and autonomous map updating. During the mapping process, local odometry is obtained using iterative error state Kalman filtering, while back-end loop detection and global pose graph optimization are utilized for accurate trajectory correction. Additionally, a fast point cloud segmentation module is incorporated to robustly distinguish between floor, walls, and roof in the environment. The segmented point clouds are then used to generate a 2.5D grid map, with particular emphasis on floor detection to filter the prior map and eliminate dynamic artifacts. In the positioning process, an initial pose alignment method is designed, which combines 2D branch-and-bound search with 3D iterative closest point (ICP) registration. This method ensures high accuracy even in scenes with similar characteristics. Subsequently, scan-to-map registration is performed using the segmented point cloud on the prior map. The system also includes a map updating module that takes into account historical point cloud segmentation results. It selectively incorporates or excludes new point cloud data to ensure consistent reflection of the real environment in the map.

Findings: The performance of the SLAM-RAMU system was evaluated in real-world environments and compared against state-of-the-art methods. The results demonstrate that SLAM-RAMU achieves higher mapping quality and relocalization accuracy, and exhibits robustness against dynamic obstacles and environmental changes.

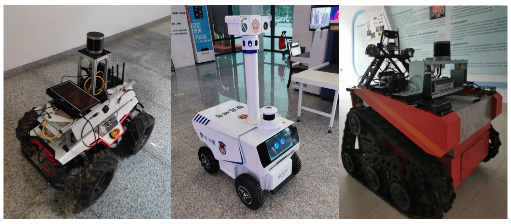
Originality/value: Compared to other state-of-the-art methods in simulation and real environments, SLAM-RAMU showed higher mapping quality, faster initial aligning speed and higher repeated localization accuracy.

1 Introduction

With the development of Lidar technology (especially the increasing popularity of 3D LiDAR), autonomous mobile robots (AMRs) based on SLAM have been increasingly used in industrial and service fields, as shown in the Figure 1. Where, in applications involving complex environments with diverse pedestrian, dynamic obstacles or constantly changing scene, it is crucial for AMRs to maintain accurate and reliable navigation during the execution of the task. Thus, the LiDAR-based SLAM navigation of AMR must meet the following three challenges, i.e. require i) high-precision prior map capable to handle dynamic obstacles and environmental changes, ii) robust initial alignment operation enabling fast and accurate registration with the prior map, and iii) reliable relocalization performance ensuring real-time and accurate navigation while automatically updating the prior map.

However, existing Lidar-based SLAM methods more or less have limitations in AMR applications to adapt to the complex dynamic environments. Most SLAM methods (Shan & Englot 2018, Shan et al. 2020) are developed based on LOAM (Zhang & Singh 2014), where the extraction of features leads to the loss of certain constraint information. Even with the utilization of graph optimization in the back-end, accuracy cannot

The authors gratefully acknowledge the financial support for this work from the 1912 project, and the open fundation of the Xiamen Key Laboratory of Frontier Electric Power Equipment and Intelligent Control (NO. XKLFEIC-201301).



Notes: AMRs with 3D LiDAR SLAM navigation.
Source: Authors' own work.

Figure 1: AMRs

be guaranteed. (Xu et al. 2022) processes all point clouds, but it lacks loop closure detection and global optimization. Therefore, in long-term SLAM navigation, it is inevitable to encounter drift. Other than that, most SLAM methods ignore dynamic obstacles which leads to ghosting in the prior map. At the same time, the existing dynamic removal methods (Kim & Kim 2020, Lim et al. 2021) simply remove the dynamic obstacles without distinguishing static point clouds (e.g., walls, pillars, floors). And SOTA relocalization methods (Koide et al. 2019) based on prior maps ignore changes in the environment, which cannot guarantee repeated localization accuracy. Furthermore, the lifelong SLAM systems with map updates (Zhao et al. 2021, Einhorn & Gross 2015, Kim & Kim 2022) suffer from high complexity, multiple map switching, and inability to ensure positioning continuity.

In this work, we propose a systematic framework to achieve lifelong SLAM based on 3D LiDAR and IMU, which will not be affected by dynamic obstacles and environmental changes. Our method employs a decoupled front-end local odometry (Xu et al. 2022) and a back-end optimization that optimizes global poses through loop closure detection. And we introduce a fast point cloud segmentation algorithm to generate a weighted 2.5D grid map (Fankhauser & Hutter 2016) only using static floor / wall points. Then, combined with the optimized poses, a high-precision 3D prior map is generated. Moreover, our lifelong SLAM has a relocalization sub-system for initial alignment and navigational positioning. Its initial alignment module combines the segmentation results of the point cloud and utilizes a 2D branch-and-bound approach (Hess et al. 2016) for enhanced efficiency, followed by ICP registration (Besl & McKay 1992) for accuracy. Its navigational positioning module employs iterative error state Kalman filter for pose prediction and update (Xu et al. 2022). Simultaneously, the 3D prior map is updated with the 2.5D grid map filtering, which enabling the removal of dynamic points from the current frame and timely updating of the prior map to adapt to environmental changes.

We conducted experiments in challenging dynamic environments to demonstrate the performance of the proposed SLAM framework. The results indicate that our methods outperform the state-of-the-art works (Shan et al. 2020, Xu et al. 2022, Koide et al. 2019, Lee et al. 2022). The contributions of this work can be summarized as follows:

- Less is more: Efficient point cloud segmentation method that facilitates the mapping of static environments, which only including fixed reference information such as floors, walls, and pillars.
- High-precision mapping: Local odometry is coupled with loop closure detection and pose graph optimization for accurate prior map containing floors, walls from the cloud segmentation results.
- Robust global alignment: Combining segmented point clouds with resolution pyramid maps, and employing branch-and-bound search and ICP registration to achieve robust relocalization with unknown initial values.
- Reliable relocalization for navigation: Fast relocalization is designed to minimize the impact of dynamic obstacles while the prior map is automatically updated to reduce interference caused by environmental changes. This ensures to get real-time lifelong SLAM for accurate and reliable navigation.

2 Related Works

2.1 LiDAR(-Inertial) SLAM

SLAM is an essential component of a complete robotic system, and with the widespread use of multi-line LiDAR, various localization methods have been proposed. Zhang & Singh (2014) introduced LOAM, which extracts line and plane features from point clouds. It utilizes frame-to-frame registration for odometry with low fidelity and submap-to-map registration for mapping and odometry with high fidelity. Shan & Englot (2018), Shan et al. (2020) presented LeGO-LOAM and LIO-SAM. LeGO-LOAM builds upon LOAM by incorporating a ground segmentation module to reduce computation and includes loop closure detection to mitigate long-term errors. LIO-SAM incorporates IMU measurements, uses preintegration for initial registration values, and employs direct frame-to-submap registration along with ISAM2 (Kaess et al. 2012) for system-wide optimization.

Jiang et al. (2023) proposed a method which tightly couples point cloud features with ground constraints and gravity constraintss for graph optimization in the back-end. Xu & Zhang (2021), Xu et al. (2022) presented FAST-LIO and its improved version, FAST-LIO2. These methods utilize IMU measurements for pose prediction, apply data compensation to the point cloud, and employ iterative error-based Kalman filtering based on the assumption of a strong Markov property.

In the works (Shan & Englot 2018, Shan et al. 2020), feature segmentation is applied to the point cloud during mapping, which leads to the loss of certain point cloud constraints and introduces significant errors in the z-axis. Additionally, the complexity of the structure makes it challenging to ensure real-time performance. On the other hand, (Xu et al. 2022) utilizes all the point cloud data for calculations, ensuring odometry accuracy through Kalman filtering and real-time performance. However, it does not include global optimization, which can result in significant errors in large-scale mapping scenarios.

The proposed SLAM module in this paper combines the strengths of both methods (Xu et al. 2022, Shan et al. 2020). The front-end local odometry is developed based on the method (Xu et al. 2022), ensuring accuracy and real-time performance. The back-end incorporates loop closure detection and pose graph optimization, resulting in higher localization and mapping accuracy compared to other methods.

2.2 Initial Aligning

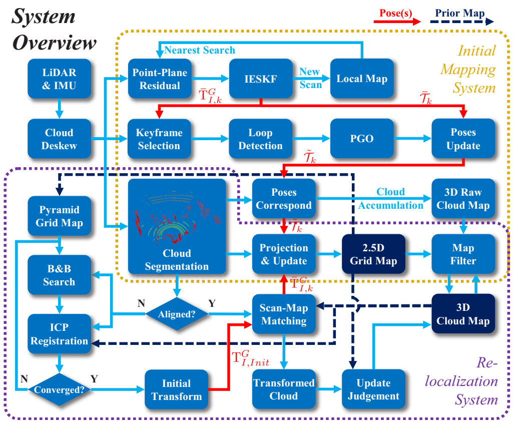
The traditional methods for initial alignment in laser-based systems can be categorized into two types. The first category is the branch-and-bound approach proposed by Hess et al. (2016). They developed a multi-resolution map and calculated scores of overlapping pixels at regular intervals. By iteratively searching for the optimal solution using depth-first search, they achieved initial alignment. Koide et al. (2019) extended this approach to 3D by projecting point clouds within a specific height range onto a xy-plane and using branch-and-bound to search for the optimal solution.

The second category is feature descriptor-based registration methods. For instance, Rusu et al. (2009) introduced the FPFH descriptor, which encodes the spatial geometric relationships between a point and its neighboring points. In (Koide et al. 2019), they constructed a KD-tree of FPFH descriptors and performed nearest neighbor searches to find the most similar results.

The method proposed by (Hess et al. 2016) ensures real-time performance but sacrifices 3D information, making it susceptible to misalignment in repetitive scenes. The Rusu et al. (2009) method performs well when the scene has distinct features, but it struggles to handle sparse feature scenarios, leading to ambiguous results. Additionally, the computation of descriptors in this system is time-consuming, resulting in low efficiency. Our method combines 2D and 3D methods, ensuring both robustness and computational efficiency in initial alignment.

2.3 Lifelong SLAM

In indoor applications, the environment undergoes changes over time, necessitating the real-time updating of the prior map, known as Lifelong SLAM. Zhao et al. (2021) proposed a fundamental and general framework for this purpose. Their approach collects dynamic information during positioning, detects differences between the old map and the real-time updated map, and prunes the old map based on this information. Einhorn & Gross (2015) presented a sensor-independent graph-based SLAM system. They performed probabilistic updates on the occupancy map using a normal distribution and optimized each node in the pose graph, which stored information about the current submap. By trimming the pose graph vertices, they avoided data inflation



Notes: Overview of the proposed method SLAM-RAMU. It consists of an initial mapping system, which outputs grid map M and point cloud map $\mathcal{G}_{build}$, and a relocalization system, which outputs odometry with map updating.
Source: Authors' own work.

Figure 2: Overview of the system

resulting from lifelong operations. Kim & Kim (2022) introduced LT-mapper, which addressed the lifelong problem by decoupling it into different periods. They performed independent alignment, dynamic cleanup, and generated a new map while pruning the original map.

However, the aforementioned methods maintain multiple sets of maps simultaneously, which may compromise real-time performance. The map updates involve switching and trimming processes between maps, which can introduce inconsistencies. Therefore, in our approach, we propose a continuous map updating system that achieves dynamic updates within a single point cloud map. This ensures consistency and real-time performance in localization.

3 Lifelong SLAM System Overview

The SLAM-RAMU system (Figure 2), designed in this paper, consists of two main components: initial mapping with global optimization and relocalization with map updates.

The initial mapping system is divided into three parts. The local odometry module obtains continuous pose estimates using iterative error state Kalman filtering. The loop detection and global optimization module updates the historical pose information by detecting loop closures and optimizing the global pose graph. The cloud segmentation and mapping module process the point clouds, segmenting them based on smoothness to generate a 2.5D grid map. It also produces a 3D point cloud map that has been segmented and filtered, with specific attention given to floor detection for removing dynamic artifacts.

The relocalization system first performs initial alignment based on the segmented point cloud, obtaining an initial pose estimate. Then, it performs scan-map registration using the segmented point cloud on the prior map to achieve real-time pose estimation. Simultaneously, based on the segmented point cloud and grid map, the original point cloud map is updated to reflect the real environment by selectively adding or subtracting new point cloud information.

Overall, the SLAM-RAMU system combines initial mapping and relocalization, providing a comprehensive and efficient solution to the lifelong SLAM problem in an indoor AMR environment.

4 Initial Mapping with Global Optimization

4.1 LiDAR-inertial Odometry

The local odometry module is adapted from (Xu et al. 2022). The LiDAR coordinate system is denoted as $(\cdot)^L$, the IMU coordinate system is denoted as $(\cdot)^I$, and the first frame's IMU position in the mapping process is selected as the global map coordinate system denoted as $(\cdot)^G$. The extrinsic calibration between LiDAR and IMU is determined to be fixed as $\mathrm{T}_L^I = (\mathrm{R}_L^I, \mathrm{p}_L^I)$ through a calibration method (Zhu et al. 2022). $\hat{\mathrm{T}}$ represents the predicted pose from the Kalman filter, and $\bar{\mathrm{T}}$ represents the updated pose. The raw point cloud information obtained at time t_k for the kth frame is denoted as $\mathcal{P}_k^L$. The IMU pose at this time can be represented as $\mathrm{T}_{I,k}^G = (\mathrm{R}_{I,k}^G, \mathrm{p}_{I,k}^G)$, and the LiDAR pose can be represented as $\mathrm{T}_{L,k}^G$. The relationship can be expressed as:

$$\mathrm{T}_{L,k}^G = \mathrm{T}_{I,k}^G \mathrm{T}_L^I \tag{1}$$

Neglecting the zero drift and system noise of the IMU system, the kinematic model of the IMU system at any arbitrary moment in continuous time can be described as:

$$\dot{\mathrm{R}}_I^G = \mathrm{R}_I^G \omega^\wedge, \ \dot{\mathrm{p}}_I^G = \mathrm{v}_I^G, \ \dot{\mathrm{v}}_I^G = \mathrm{R}_I^G \mathrm{a} + \mathrm{g}^G, \ \dot{\mathrm{g}}^G = 0 \tag{2}$$

where R_I^G and p_I^G represent the orientation and position information of the IMU in the global coordinate system, g^G is the gravity vector in the global coordinate system, ω and a are the measured values of the IMU in ideal conditions.

4.1.1 Cloud Deskew

The point in $\mathcal{P}_k^L$ obtained at time t_k is actually collected between t_{k-1} and t_k. When the system is in motion, points collected at different moments correspond to different robot poses. Assuming that there is a point p_i^L in $\mathcal{P}_k^L$ collected at time t_i, where t_i is a moment between t_{k-1} and t_k, and the LiDAR pose at time t_i is described by $\mathrm{T}_{L,i}^G$, combining $\mathrm{T}_{L,k-1}^G$ and the IMU kinematic model, we can use the propagation equation in (Xu et al. 2022) for the pose prediction, obtaining the predicted poses $\hat{\mathrm{T}}_{L,i}^G$ for each point and the predicted pose $\hat{\mathrm{T}}_{L,k}^G$ at the end of the frame. Applying the following equation:

$$\hat{p}_i^L = \hat{\mathrm{T}}_{L,i}^G (\hat{\mathrm{T}}_{L,k}^G)^{-1} p_i^L \tag{3}$$

where $\hat{p}_i^L$ is the projection of p_i^L onto the cloud at time t_k. All the projected points $\hat{p}_i^L$ within the interval between t_{k-1} and t_k are accumulated to generate the de-skewed cloud $\hat{\mathcal{P}}_k^L$ for the kth frame. $\hat{\mathcal{P}}_k^L$ is transformed to $(\cdot)^I$ by $\hat{\mathcal{P}}_k^I = \mathrm{T}_L^I \hat{\mathcal{P}}_k^L$. And then the local map $\mathcal{L}_{odom}$ is generated with $\hat{\mathcal{P}}_0^I$ directly.

4.1.2 Point-Plane Residual

For each point $\hat{p}_i^I$ in $\hat{\mathcal{P}}_k^I$, the nearest neighboring plane satisfying certain conditions is identified within $\mathcal{L}_{odom}$. The measurement model is as follows:

$$0 = (\mathrm{u}_i^G)^{\mathrm{T}} (\bar{\mathrm{T}}_{I,k}^G \hat{p}_i^I - \mathrm{q}_i^G) \tag{4}$$

where u_i^G is the normal vector of the nearest neighboring plane, and q_i^G is a point on that plane. By combining the measurement model, an optimization equation is established in (Xu et al. 2022). Then the Kalman filter updates $\hat{\mathrm{T}}_{I,k}^G$ to $\bar{\mathrm{T}}_{I,k}^G$. The point cloud $\bar{\mathcal{P}}_k^G$ in $(\cdot)^G$ is obtained as follows:

$$\bar{\mathcal{P}}_k^G = \bar{\mathrm{T}}_{I,k}^G \hat{\mathcal{P}}_k^I \tag{5}$$

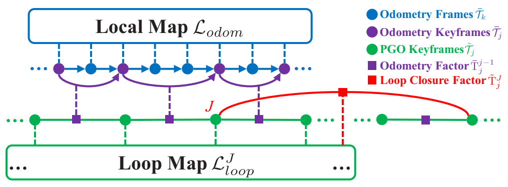
4.2 Loop Detection and Pose Optimization

4.2.1 keyframe Selection

The schematic diagram of pose graph optimization is shown in Figure 3. The set of $\bar{\mathrm{T}}_{I,k}^G$ is denoted as $\bar{\mathcal{T}}_k$, where $\bar{\mathcal{T}}_k = [\bar{\mathrm{T}}_{I,0}^G, \ldots, \bar{\mathrm{T}}_{I,k}^G]$. There also exists a set of keyframes $\bar{\mathcal{T}}_j$ such that $\bar{\mathcal{T}}_j \subset \bar{\mathcal{T}}_k$, and $\bar{\mathcal{T}}_j = [\bar{\mathrm{T}}_{I,0}^G, \ldots, \bar{\mathrm{T}}_{I,j}^G]$. The selection of keyframes is determined using the following equation, where $f(\cdot)$ indicates a judgment based on the Euclidean distance and angular difference between poses:

$$\bar{\mathrm{T}}_{I,j}^G \leftarrow f(\bar{\mathrm{T}}_{I,k}^G, \bar{\mathrm{T}}_{I,j-1}^G) \tag{6}$$

When $\bar{\mathrm{T}}_{I,k}^G$ satisfies the judgment with $\bar{\mathrm{T}}_{I,j-1}^G$, $\bar{\mathrm{T}}_{I,k}^G$ is added to $\bar{\mathcal{T}}_j$ and denoted as $\bar{\mathrm{T}}_{I,j}^G$. Similarly, $\bar{\mathrm{T}}_{I,j}^G$ represents the pose of the robot at time t_j. The pose transformation $\bar{\mathrm{T}}_j^{j-1}$ between $\bar{\mathrm{T}}_{I,j-1}^G$ and $\bar{\mathrm{T}}_{I,j}^G$ forms the odometry factor between global pose graph nodes, providing local constraints.


Local Map $\mathcal{L}_{odom}$
Odometry Frames $\bar{\mathcal{T}}_k$
Odometry Keyframes $\bar{\mathcal{T}}_j$
PGO Keyframes $\tilde{\mathcal{T}}_j$
Odometry Factor $\bar{\mathrm{T}}_j^{j-1}$
Loop Closure Factor $\tilde{\mathrm{T}}_j^J$
J
Loop Map $\mathcal{L}_{loop}^J$

Notes: Data transfer between local odometry and global pose graph optimization.$(\bar{\cdot})$ represents data before optimizing, and $(\tilde{\cdot})$ after optimizing.
Source: Authors' own work.

Figure 3: Schematic diagram of data transmission

4.2.2 Loop Detection

The loop detection module uses a radius-based search method. Taking into account the time-related constraints, in a certain radius range of $\bar{\mathrm{T}}_{I,j}^G$ witn $\bar{\mathcal{T}}_j$, the nearest pose is identified using the kd-tree. Assuming that there exists a pose $\bar{\mathrm{T}}_{I,J}^G$ at $j = J$ that satisfies the time and radius condition, the point clouds corresponding to the frames around $\bar{\mathrm{T}}_{I,J}^G$ in $\bar{\mathcal{T}}_j$, denoted as $\{\bar{\mathcal{P}}_{J-m}^G \ldots, \bar{\mathcal{P}}_J^G \ldots, \bar{\mathcal{P}}_{J+m}^G\}$, are used to generate a submap $\mathcal{L}_{loop}^J$. And then conduct ICP registration between $\bar{\mathcal{P}}_j^G$ of $\bar{\mathrm{T}}_{I,j}^G$ and $\mathcal{L}_{loop}$ to obtain the transformation $\tilde{\mathrm{T}}_j^J$ and judge validity of the loop by registration score.

4.2.3 PGO

If the loop is valid, the loop closure factor between the corresponding nodes is added using $\tilde{\mathrm{T}}_j^J$, and the global pose graph optimization is performed using the iSAM2 (Kaess et al. 2012) to obtain the updated set $\tilde{\mathcal{T}}_j$, where the pose representation is $\tilde{\mathrm{T}}_{I,j}^G$. Finally, the kd-tree is reconstructed using $\tilde{\mathcal{T}}_j$ and the loop detection operation is repeated.

4.2.4 Poses Update

The following equation is used to update $\bar{\mathcal{T}}_k$ using $\tilde{\mathcal{T}}_j$, where $\tilde{\mathrm{T}}_{I,k}^G$ represents the updated frame pose, forming the set $\tilde{\mathcal{T}}_k$.

$$\tilde{\mathrm{T}}_{I,k}^G = \bar{\mathrm{T}}_{I,k}^G (\bar{\mathrm{T}}_{I,j}^G)^{-1} \tilde{\mathrm{T}}_{I,j}^G \tag{7}$$

4.3 Cloud Segmentation and Mapping

While optimizing the poses, the proposed method also segments the point cloud and then removes dynamic objects from the map. The de-skewed cloud point $\hat{\mathcal{P}}_k^L$ of time t_k consist of N_k points, which can be known as a set $\hat{\mathcal{P}}_k^L = \{p_1^L, \ldots, p_{N_k}^L\}$ with 3D points $p_i^L = (x_i \; y_i \; z_i \; ring_i \; time_i)^{\mathrm{T}}$.

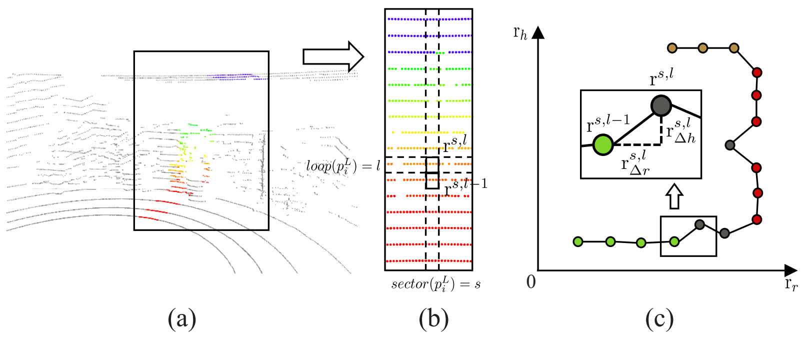
4.3.1 Cloud Segmentation

In (Himmelsbach et al. 2010), all points are projected and binned on to the xy-plane. The minimum z value of all points in the bin is used to represent the bin. Our segmentation algorithm refers to their method, and combined with the point's own parameters to optimize the algorithm. Similar to (Himmelsbach et al. 2010), the parameter $\Delta\alpha$ is set as the angle covered by each sector. Therefore, we obtain $M_{sector} = \frac{2\pi}{\Delta\alpha}$ sectors. The index of the sector corresponding to p_i^L is denoted as $sector(p_i^L)$ and can be calculated by the following equation:

$$sector(p_i^L) = \frac{\arctan(y_i, x_i)}{\Delta\alpha} \tag{8}$$

If the $time_i$ parameter of p_i^L is stable and reliable, then $sector(p_i^L)$ can be calculated by the following equation to improve the efficiency of the algorithm. Where, the value range of $time_i$ is represented as $[time_{min}, time_{max}]$.

$$sector(p_i^L) = \frac{time_i - time_{min}}{time_{max} - time_{min}} M_{sector} \tag{9}$$



Notes: (a) represents part of the point cloud in a scan. Select some sectors of point cloud to be colored as a rainbow color from -0.5m to 2.5m on the z-axis. (b) The colored points are extracted in the LiDAR view to show the relationship between points and rooms. The points between the vertical dotted lines are points with $sector(p_i^L) = s$, and $loop(p_i^L) = l$ between the horizontal dotted lines. The intersection forms the room $\mathrm{r}^{s,l}$ which is displayed as rectangles in (b). (c) Project the rooms with $sector(\mathrm{r}) = s$ onto a r_r-r_h coordinate system. The circles represent each room, and the solid line represents the slope between the rooms. The color correspondences are the floor(green), the wall(red), the roof(brown), and the grey ones do not meet the conditions which will be abandoned.
Source: Authors' own work.

Figure 4: Schematic diagram of pointcloud segmentation

Different from (Himmelsbach et al. 2010), inspired by (Zhang & Singh 2014), we explore 2D projection to 2.5D with a relation between rings. p_i^L has a reliable parameter $ring_i$, representing the order relationship between the point cloud beams. The point cloud has a total of τ_{max} rings, and we set the parameter $\Delta\tau$ as the number of rings covered by each loop. Thus, we obtain $M_{loop} = \frac{\tau_{max}}{\Delta\tau}$ loops, and the index of the loop corresponding to p_i^L is denoted as $loop(p_i^L)$, which can be calculated using the following equation:

$$loop(p_i^L) = \frac{ring_i}{\Delta\tau} \tag{10}$$

Each p_i^L has its indexes of sector and loop, we denote the set $\mathcal{R}^{s,l}$ as points with same indexes:

$$\mathcal{R}^{s,l} = \{p_i^L \in \hat{\mathcal{P}}_k^L | sector(p_i^L) = s, loop(p_i^L) = l\} \tag{11}$$

Assuming there are r_{size} points in $\mathcal{R}^{s,l}$, it can be represented as $\mathcal{R}^{s,l} = \{p_1^{s,l}, \ldots, p_{\mathrm{r}_{size}}^{s,l}\}$ with 3D points $p_i^{s,l} = (x_i\ y_i\ z_i\ ring_i\ time_i)^{\mathrm{T}}$. We adjust the format of $p_i^{s,l}$ to $(h_i\ r_i)^{\mathrm{T}}$ using the following equations:

$$h_i = \sqrt{{x_i}^2 + {y_i}^2},\ \ r_i = z_i \tag{12}$$

Set $\mathcal{R}^{s,l}$ corresponds to a room $\mathrm{r}^{s,l}$, in which there are multiple parameters. Among them, the height is represented as r_h, the distance is represented as r_r, and the height covariance is represented as r_{hcov}. The above parameters are calculated according to the following equation:

$$\mathrm{r}_h = \frac{\sum^{\mathrm{r}_{size}} h_i}{\mathrm{r}_{size}},\ \ \mathrm{r}_r = \frac{\sum^{\mathrm{r}_{size}} r_i}{\mathrm{r}_{size}},\ \ \mathrm{r}_{hcov} = \frac{\sum^{\mathrm{r}_{size}} (\mathrm{r}_h - h_i)^2}{\mathrm{r}_{size}} \tag{13}$$

To avoid the limitation of the minimum value, the information of $\mathrm{r}^{s,l}$ is determined by the average value of all points in $\mathcal{R}^{s,l}$. At the same time, the minimum and maximum loop values for each sector are recorded as l_{beg} and l_{end}. Then, the rooms in each sector are sequentially calculated and evaluated, as shown in Algorithm 1.

The segmentation method first checks the values of l_{beg} and l_{end} to skip sectors where data collection has failed. Starting from l_{beg}, as shown in Figure 4, for each room $\mathrm{r}^{s,l}$ and the previous room $\mathrm{r}^{s,l-1}$, the method calculates the height difference $\mathrm{r}^{s,l}_{\Delta h}$ and distance difference $\mathrm{r}^{s,l}_{\Delta r}$, and uses them to calculate the slope $\mathrm{r}^{s,l}_{s}$ and the slope difference $\mathrm{r}^{s,l}_{\Delta s}$ between the rooms, representing the smoothness between the loops.

Threshold check is then applied to the calculated values. For the floor and roof, assuming that the robot runs in an environment without steep slopes, it first checks whether $\mathrm{r}^{s,l}_{h}$ satisfies within a specific range. It

Algorithm 1: Cloud Segmentation Method

```
    Input: $\mathrm{r}^{s,l}, \mathrm{r}^{s,l-1}$, floor slope threshold $\sigma_{\mathrm{F}}^{s}$ and so on $\sigma_{\mathrm{F}}^{\Delta s}$, $\sigma_{\mathrm{F}}^{hcov}$, $\sigma_{\mathrm{R}}^{s}$, $\sigma_{\mathrm{R}}^{\Delta s}$, $\sigma_{\mathrm{W}}^{s}$, $\sigma_{\mathrm{W}}^{\Delta s}$, possible floor
           height interval $\mathrm{I_F}$ and so on $\mathrm{I_R}$
    Output: Segmented cloud $\mathcal{F}, \mathcal{R}, \mathcal{W}$
 1  if $l_{beg} > l_{end}$ then
 2      Skip this sector;
 3  end
 4  for $l \leftarrow l_{beg}$ to $l_{end}$ do
 5      $\mathrm{r}_{\Delta h}^{s,l} \leftarrow \mathrm{r}_{h}^{s,l} - \mathrm{r}_{h}^{s,l-1}$;  $\mathrm{r}_{\Delta r}^{s,l} \leftarrow \mathrm{r}_{r}^{s,l} - \mathrm{r}_{r}^{s,l-1}$;
 6      $\mathrm{r}_{s}^{s,l} \leftarrow \mathrm{r}_{\Delta h}^{s,l} \div \mathrm{r}_{\Delta r}^{s,l}$;  $\mathrm{r}_{\Delta s}^{s,l} \leftarrow \mathrm{r}_{s}^{s,l} - \mathrm{r}_{s}^{s,l-1}$;
 7      if $\mathrm{r}_{h}^{s,l} \in \mathrm{I_F}$ and $\mathrm{r}_{s}^{s,l} < \sigma_{\mathrm{F}}^{s}$ and $\mathrm{r}_{\Delta s}^{s,l} < \sigma_{\mathrm{F}}^{\Delta s}$ and $\mathrm{r}_{hcov}^{s,l} < \sigma_{\mathrm{F}}^{hcov}$ then
 8          Push back relevant points to floor cloud $\mathcal{F}$;
 9      else if $\mathrm{r}_{h}^{s,l} \in \mathrm{I_R}$ and $\mathrm{r}_{s}^{s,l} < \sigma_{\mathrm{R}}^{s}$ and $\mathrm{r}_{\Delta s}^{s,l} < \sigma_{\mathrm{R}}^{\Delta s}$ then
10          Push back relevant points to roof cloud $\mathcal{R}$;
11      else if $\mathrm{r}_{s}^{s,l} > \sigma_{\mathrm{W}}^{s}$ and $(\mathrm{r}_{s}^{s,l})^{-1} - (\mathrm{r}_{s}^{s,l-1})^{-1} < \sigma_{\mathrm{W}}^{\Delta s}$ then
12          Push back relevant points to wall cloud $\mathcal{W}$;
13      else
14          Skip this room;
15      end
16  end
```

then ensures that $\mathrm{r}_{s}^{s,l}$ is below a certain threshold relative to the robot's current pose. Finally, it evaluates $\mathrm{r}_{\Delta s}^{s,l}$ to judge the smoothness between rooms. In the case of the floor, to ensure robustness, $\mathrm{r}_{hcov}^{s,l}$ is also considered. For wall points, after taking the inverse transformation, the method only checks the absolute slope and relative slope difference for classification.

4.3.2 Poses correspond

After segmentation, removing the roof points and unclassified points, we select the floor point cloud $\hat{\mathcal{F}}_{k}^{L}$ and the wall point cloud $\hat{\mathcal{W}}_{k}^{L}$. These point clouds are then associated with poses in $\tilde{\mathcal{T}}_{k}$ using the following equation. Obtain $\tilde{\mathcal{F}}_{k}^{G}$ and $\tilde{\mathcal{W}}_{k}^{G}$ in $(\cdot)^{G}$.

$$(\tilde{\mathcal{F}}_{k}^{G}, \tilde{\mathcal{W}}_{k}^{G}) = \tilde{\mathrm{T}}_{I,k}^{G}\mathrm{T}_{L}^{I}(\hat{\mathcal{F}}_{k}^{L}, \hat{\mathcal{W}}_{k}^{L}) \tag{14}$$

4.3.3 Projection & Update

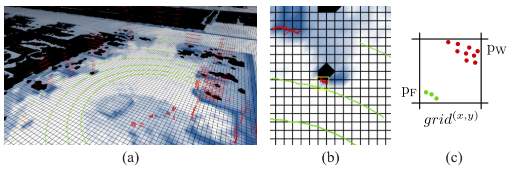
A 2.5D grid map M with a resolution of r_{M} is constructed in $(\cdot)^{G}$, utilizing (Fankhauser & Hutter 2016) with the multi-dimensional and easily addressable properties As shown in Figure 5, $grid^{(x,y)}$ represents a grid with a center point located at (x, y) and a diameter of r_{M}. The points in $\tilde{\mathcal{F}}_{k}^{G}$ and $\tilde{\mathcal{W}}_{k}^{G}$ are projected onto M. Assuming that $grid^{(x,y)}$ contains n floor points $\mathrm{p_F}$ and m wall points $\mathrm{p_W}$, θ_{F} and θ_{W} represent the weights of the floor and wall, respectively. The information contained in $grid^{(x,y)}$ is computed using the following equation:

$$grid_{judge}^{(x,y)} = \sum^{m} \theta_{\mathrm{W}} - \sum^{n} \theta_{\mathrm{F}},\ grid_{height}^{(x,y)} = \min(\mathrm{p_F}.z) \tag{15}$$

where $grid_{judge}$ represents the floor judgment information, determining whether the corresponding grid is classified as floor or wall, and $grid_{height}$ represents the lowest height of the grid.

4.3.4 Map Filter

Finally, the map filtering process is performed. Inspired by (Lim et al. 2021), all points in $\tilde{\mathcal{F}}_{k}^{G}$ and $\tilde{\mathcal{W}}_{k}^{G}$ are accumulated to obtain the original 3D point cloud map $\mathcal{G}_{raw}$. For each point p_{raw} in $\mathcal{G}_{raw}$, if its corresponding $grid_{judge}^{(x,y)}$ in M satisfies a threshold condition ($\leqslant$0), it is classified as floor, and indicating that only floor points should exist on the floor. Based on this, filtering is performed in combination with $grid_{height}^{(x,y)}$, with $r_{\mathcal{G}}$



Notes: (a) represents the segmented point clouds $\tilde{\mathcal{F}}_k^G$ (green) and $\tilde{\mathcal{W}}_k^G$ (red) from a 3D perspective. The varying shades of the floor indicate the magnitude of $grid_{judge}^{(x,y)}$. (b) shows a 2D view, and a localized zoom-in of the yellow box is depicted in (c). This particular grid contains n floor points $\mathrm{p_F}$ and m wall points $\mathrm{p_W}$.
Source: Authors' own work.

Figure 5: Schematic diagram of gridmap

representing the point cloud map resolution, as shown in the following equation:

$$\mathcal{G}_{build} = \{\mathrm{p}_{raw} | \begin{array}{c} \mathrm{p}_{raw} \in \mathcal{G}_{raw}, grid_{judge}^{(x,y)} \leqslant 0, \\ grid_{height}^{(x,y)} - \mathrm{p}_{raw}.z < r_{\mathcal{G}} \end{array} \} \tag{16}$$

The dynamic points on the floor have been filtered out, resulting in a new 3D map $\mathcal{G}_{build}$ that contains only floor and wall information, excluding dynamic obstacles.

5 Relocalization and Autonomous Map Updating

5.1 Relocalization for Initial Aligning

When a robot is turned on and its initial pose in the prior map is unknown, this requires a first global relocalization process to achieve initial aligning. We propose an efficient and robust initial aligning method. The method takes the initial segmented clouds $\hat{\mathcal{F}}_{Init}^L$ and $\hat{\mathcal{W}}_{Init}^L$ obtained from the segmentation method described earlier as input, as well as the 2.5D grid map M and 3D point cloud map $\mathcal{G}_{build}$ generated by the mapping system.

5.1.1 Pyramid Grid Map

Firstly, referring to (Hess et al. 2016), a multi-resolution pyramid map $M^{level}, \{level = 0, 1, 2, 3\}$ is created. M^0 is obtained directly from M as follows:

$$M_{(x,y)}^0 = \begin{cases} -1, & M.grid_{judge}^{(x,y)} \leqslant 0 \\ 1, & M.grid_{judge}^{(x,y)} > 0 \\ 0, & others \end{cases} \tag{17}$$

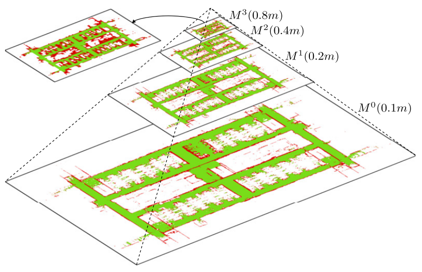
where (x, y) represents the pose of each grid center relative to the origin. $M.grid_{judge}^{(x,y)}$ represents the floor judgment at the corresponding (x, y) position in M. The other layers of the pyramid map are calculated using the following equation:

$$M_{(x/2,y/2)}^{level} = \sum M_{(x,y)}^{level-1} \tag{18}$$

Finally, for the remaining layers, the following equations are performed to differentiate between floor and wall:

$$M_{(x,y)}^{level} = \begin{cases} -1, & M_{(x,y)}^{level} \leqslant 0 \\ 1, & M_{(x,y)}^{level} > 0 \\ 0, & others \end{cases} \tag{19}$$

This generates a multi-resolution pyramid map with segmentation information, as shown in Figure 6. The original $grid_{judge}$ is used to judge the M^0 layer, and then an iterative strategy is employed using M^0 to generate subsequent layers. This strategy avoids the problem of losing features in low resolution maps that would occur if each layer used the $grid_{judge}$ information.



Notes: The actual display of each layer of the pyramid M. The upper corner zooms in on the coarsest resolution, and M^3, M^2, M^1, M^0 reflect the features from coarse to fine.
Source: Authors' own work.

Figure 6: Resolution pyramid

Algorithm 2: Initial Aligning Method

```
   Input: $\hat{\mathcal{F}}_k^L, \hat{\mathcal{W}}_k^L, M, \mathcal{G}_{build}$, score tolerance $\sigma_{score}$
   Output: $\mathrm{T}_{L,Init}^G$
 1 Build pyramid map $M^{level}$ based on $M$;
 2 Set pyramid vector set $\mathcal{V}_{\mathrm{F}}^{level}$ and $\mathcal{V}_{\mathrm{W}}^{level}$ based on $\hat{\mathcal{F}}_k^L$ and $\hat{\mathcal{W}}_k^L$;
 3 Set initial possible 2D pose set $\mathcal{S}^{2D}$;
 4 repeat
 5     Get best $\tilde{\mathrm{s}}^{\mathrm{2D}}$ of $\mathcal{S}^{\mathrm{2D}}$ using BBS ;
 6     Switch $\tilde{\mathrm{s}}_{pose}^{\mathrm{2D}}$ to $\mathrm{s}_{pose}^{\mathrm{3D}}$ with $grid_{height}$;
 7     Get $\tilde{\mathrm{s}}_{pose}^{\mathrm{3D}}$ using ICP with initial transform $\mathrm{s}_{pose}^{\mathrm{3D}}$;
 8     Calc 3D registration score $\tilde{\mathrm{s}}_{score}^{\mathrm{3D}}$ and push $\tilde{\mathrm{s}}^{\mathrm{3D}}$ to $\mathcal{S}^{3D}$;
 9     Block $\mathcal{S}^{2D}$ based on $\mathrm{s}^{\mathrm{2D}}$;
10 until $\mathrm{s}_{score}^{\mathrm{3D}} > \sigma_{score}$ or max iteration;
11 Pick pose with best score in $\mathcal{S}^{3D}$ as $\mathrm{T}_{L,Init}^G$;
```

Finally, M^{level} is divided into three parts: floor grids (-1), wall grids (1), and unknown grids (0). Then, an initial set of possible positions $\mathcal{S}^{\mathrm{2D}}$ is established, where each element s^{2D} contains score, level, and pose information. $\mathrm{s}_{pose}^{\mathrm{2D}}$ represents the 2D position information of the element, which is selected on the coarsest resolution map M^3 in the translation dimension. Since the robot is only expected to exist above the floor, only positions where $M_{(x,y)}^{level}$ is floor grid are selected to reduce the number of elements in the set. In the rotation dimension, samples are taken at fixed intervals within 360 degrees. $\mathrm{s}_{score}^{\mathrm{2D}}$ represents the score corresponding to that position, and $\mathrm{s}_{level}^{\mathrm{2D}}$ represents the level corresponding to that position. $\hat{\mathcal{F}}_{Init}^L$ and $\hat{\mathcal{W}}_{Init}^L$ are also constructed into multi-resolution sets by voxel downsampling, resulting in $\mathcal{V}_{\mathrm{F}}^{level}$ and $\mathcal{V}_{\mathrm{W}}^{level}$. Next, for each element s^{2D} in $\mathcal{S}^{\mathrm{2D}}$, obtain the translated and rotated 2D points $\bar{\mathcal{V}}_{\mathrm{F}}^{level}$ and $\bar{\mathcal{V}}_{\mathrm{W}}^{level}$ by the following equation:

$$(\bar{\mathcal{V}}_{\mathrm{F}}^{level}, \bar{\mathcal{V}}_{\mathrm{W}}^{level}) = \mathrm{s}_{pose}^{\mathrm{2D}}(\mathcal{V}_{\mathrm{F}}^{level}, \mathcal{V}_{\mathrm{W}}^{level}) \tag{20}$$

For all $\mathrm{K_F}$ points with pose of (x, y) in $\bar{\mathcal{V}}_{\mathrm{F}}^{level}$, assuming that the number of points satisfying $M_{(x,y)}^{level}$ as floor gird is $\mathrm{N_F}$, we can similarly obtain the parameters $\mathrm{K_W}$ and $\mathrm{N_W}$ for $\bar{\mathcal{V}}_{\mathrm{W}}^{level}$. Therefore, $\mathrm{s}_{score}^{\mathrm{2D}}$ is calculated as follows:

$$\mathrm{s}_{score}^{\mathrm{2D}} = (\sum^{\mathrm{N_F}} \theta_{\mathrm{F}}^{+} - \sum^{\mathrm{K_F}-\mathrm{N_F}} \theta^{-} + \sum^{\mathrm{N_W}} \theta_{\mathrm{W}}^{+} - \sum^{\mathrm{K_W}-\mathrm{N_W}} \theta^{-}) 2^{\mathrm{s}_{level}^{\mathrm{2D}}} \tag{21}$$

where θ_{F}^{+} represents the positive weight for floor points, θ_{W}^{+} represents the positive weight for wall points, and θ^{-} represents the negative weight. Additionally, considering that the density of 2D points cross different levels, a level score gain of $2^{\mathrm{s}_{level}^{\mathrm{2D}}}$ is added to ensure that the scores remain comparable.

5.1.2 B&B Search

The overall method is shown as Algorithm 2. In (Hess et al. 2016), a comprehensive description of the branch and bound search (BBS) method with depth-first search is provided. In this paper, based on that, considering the low robustness of 2D registration, an improvement is made. After obtaining the optimal result $\tilde{\mathrm{s}}^{\mathrm{2D}}$ from the BBS, a 3D pose $\mathrm{s}^{\mathrm{3D}}_{pose}$ is obtained by combining it with the corresponding $grid_{height}$ in M.

5.1.3 ICP Registration

$\hat{\mathcal{F}}^{L}_{Init}$ and $\hat{\mathcal{W}}^{L}_{Init}$ are combined into $\hat{\mathcal{C}}^{L}_{Init}$. ICP registration is then performed between $\hat{\mathcal{C}}^{L}_{Init}$ and $\mathcal{G}_{build}$ with the initial transformation $\mathrm{s}^{\mathrm{3D}}_{pose}$ to obtain the solution $\tilde{\mathrm{s}}^{\mathrm{3D}}_{pose}$ and calculate the corresponding score $\tilde{\mathrm{s}}^{\mathrm{3D}}_{score}$. If the ICP is converged, we can obtain $\tilde{\mathrm{s}}^{\mathrm{3D}}_{pose}$ and then calculate the corresponding score $\tilde{\mathrm{s}}^{\mathrm{3D}}_{score}$. Using $\tilde{\mathrm{s}}^{\mathrm{3D}}_{pose}$, $\hat{\mathcal{C}}^{G}_{Init}$ in $(\cdot)^{G}$ is obtained. $\hat{\mathcal{C}}^{G}_{Init}$ contains N_{all} points. The nearest neighbor distances r_i within $\mathcal{G}_{build}$ of each point in $\hat{\mathcal{C}}^{G}_{Init}$ are calculated. Here, i represents the index of the point, and r_i less than the map resolution is considered as an inlier. There are N_{in} inliers. The final score is expressed as follows:

$$\tilde{\mathrm{s}}^{\mathrm{3D}}_{score} = \frac{\mathrm{N}_{in}}{\mathrm{N}_{all}} \div \frac{\sum^{\mathrm{N}_{in}} r_i}{\mathrm{N}_{in}} \tag{22}$$

where the former equation can be viewed as the inlier ratio parameter, which is the ratio of the number of inliers to the total number of input points in the point cloud. The latter can be viewed as the registration error parameter, which is the mean distance between all inliers and their nearest neighbors. A higher score indicates a more accurate registration.

The result $\tilde{\mathrm{s}}^{\mathrm{3D}}$ is stored in the set $\mathcal{S}^{3D}$, and a region constraint is applied to $\mathcal{S}^{2D}$ to prevent the BBS from outputting duplicate solutions. This iteration process continues until the convergence condition is met or the maximum number of iterations is reached. Finally, the pose corresponding to the element $\mathcal{S}^{3D}$ with the highest score in is output as $\mathrm{T}^{G}_{L,Init}$.

5.2 Relocalization for Navigation

The initial aligning result $T^{G}_{L,Init}$ is transformed to $T^{G}_{I,Init}$ in $(\cdot)^{I}$, which serves as the initial pose for the relocalization system, denoted as $\bar{T}^{G}_{I,0}$. Same as the previous odometry section, for the kth frame, we can get $\hat{T}^{G}_{I,k}$ and $\hat{\mathcal{P}}^{I}_{k}$. However, in navigation secton, $\hat{\mathcal{P}}^{I}_{k}$ does not directly execute point-to-plane registration. Instead, it follows a consistency strategy between mapping and relocalization. The segmentation method is used to obtain $\hat{\mathcal{F}}^{L}_{k}$ and $\hat{\mathcal{W}}^{L}_{k}$ from the proposed cloud segmentation method and form $\hat{\mathcal{C}}^{I}_{k}$ using the following equation:

$$\hat{\mathcal{C}}^{I}_{k} = \{\mathrm{p}|\mathrm{p} \in \hat{\mathcal{F}}^{L}_{k}, \mathrm{p} \in \hat{\mathcal{W}}^{L}_{k}\} \tag{23}$$

Compared to $\hat{\mathcal{P}}^{I}_{k}$, $\hat{\mathcal{C}}^{I}_{k}$ reduces the number of point cloud points and removes most irregular points, improving the success rate of finding nearest neighbor surfaces. And then, $\hat{\mathcal{C}}^{I}_{k}$ undergoes point-to-plane registration and residual calculation within $\mathcal{G}_{build}$, followed by Kalman filtering to obtain the updated pose $\bar{\mathrm{T}}^{G}_{I,k}$.

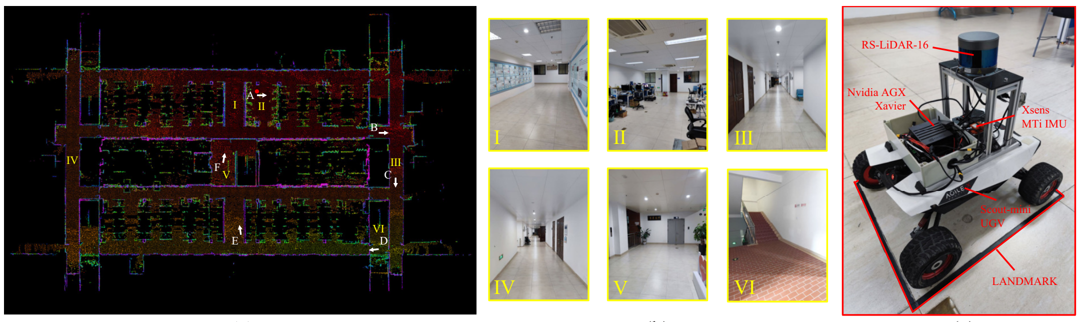
In registration process, we can create $knn(\cdot)$ which outputs number of nearest neighbor points within $\mathcal{G}_{build}$ and k_{min} which means minimum number of points forming a plane. And we can split $\hat{\mathcal{C}}^{I}_{k}$ into two parts using the following equations:

$$\begin{aligned} \mathcal{C}^{\mathrm{P}} &= \{\mathrm{p}|\mathrm{p} \in \hat{\mathcal{C}}^{I}_{k},\ knn(\mathrm{p}) \geqslant k_{min}\}, \\ \mathcal{C}^{\mathrm{NP}} &= \{\mathrm{p}|\mathrm{p} \in \hat{\mathcal{C}}^{I}_{k},\ knn(\mathrm{p}) < k_{min}\} \end{aligned} \tag{24}$$

Only $\mathcal{C}^{\mathrm{P}}$ can go to residual calculation process and effect the final pose. At the same time, the points corresponding to dynamic objects in navigation process will be classified as $\mathcal{C}^{\mathrm{NP}}$ due to the fact that only stationary wall and floor points are retained in $\mathcal{G}_{build}$. Therefore, proposed method eliminates the influence of dynamic objects on positioning during the navigation process, ensuring the stability of the relocalization system.

5.3 Autonomous Map Updating

When the environment undergoes changes, such as a group of shelves or tables being moved or shifted, the relocalization system's accuracy is inevitably influenced if it relies on the original map $\mathcal{G}_{build}$ for registration. To address this issue, we propose an autonomous map updating module.



(a) (b) (c)

Notes: (a) shows a top view of the real office environment. In (a), we selected some scenes from I to VI and photographed them as shown in B which contained offices, corridors and stairwells. (c) shows the platform for the experiment, containing the sensors, with a landmark in red dot of (a) on the floor for testing repeated localization accuracy. In (a), we also selected several poses as A to E arrows for initial aligning. The starting point of the arrow indicates the position, and the direction indicates the angle.
Source: Authors' own work.

Figure 7: Description of the dataset

In a dynamic environment, $\hat{\mathcal{C}}_k^I$ is divided into four categories:

- Original static wall point cloud $\mathcal{C}^{\mathrm{S}}$, such as load-bearing walls.
- Newly added static wall point cloud $\mathcal{C}^{+\mathrm{S}}$, such as newly installed shelves or tables.
- Dynamic obstacle point cloud $\mathcal{C}^{\mathrm{D}}$, such as moving individuals.
- Floor point cloud $\mathcal{C}^{\mathrm{F}}$.

The task of the entire update module is to partition these four categories of point clouds. The segmentation module is responsible for the differentiation of $\mathcal{C}^{\mathrm{F}}$ and the other three categories. According to the previous navigation section, we can distinguish $\mathcal{C}^{\mathrm{S}}$ using the following equation:

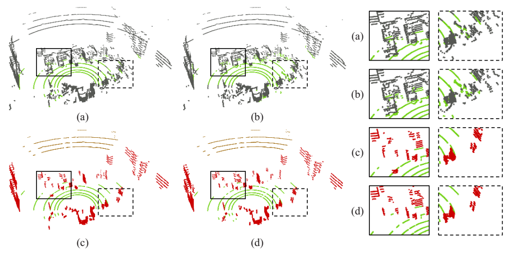
$$\mathcal{C}^{\mathrm{P}} = \mathcal{C}^{\mathrm{S}},\ \mathcal{C}^{\mathrm{NP}} = \mathcal{C}^{+\mathrm{S}} \cup \mathcal{C}^{\mathrm{D}} \tag{25}$$

Subsequently, for the differentiation of $\mathcal{C}^{+\mathrm{S}}$ and $\mathcal{C}^{\mathrm{D}}$, the 2.5D grid map method is used again. $\mathcal{C}^{+\mathrm{S}}$, $\mathcal{C}^{\mathrm{D}}$, and floor points $\mathcal{C}^{\mathrm{F}}$ are transformed into $(\cdot)^G$ using $\bar{\mathrm{T}}_{I,k}^{G}$, and then execute *Projection & Update* method. Points in $\mathcal{C}^{+\mathrm{S}}$ and $\mathcal{C}^{\mathrm{D}}$ will be classified by floor judgment of their corresponding grids in M. If a point is only a dynamic obstacle, it will not have an impact on the final floor judgment. The corresponding grid will still satisfy $grid_{judge}^{(x,y)} \leqslant 0$. On the other hand, for newly added static points, the corresponding grid will gradually accumulate, and eventually $grid_{judge}^{(x,y)} > 0$, indicating a wall grid. At the same time, the floor distribution in M is also updated based on $\mathcal{C}^{\mathrm{F}}$.

As described above, the distinction between $\mathcal{C}^{+\mathrm{S}}$ and $\mathcal{C}^{\mathrm{D}}$ is completed, $\mathcal{C}^{\mathrm{D}}$ is removed, and $\mathcal{C}^{+\mathrm{S}}$ is added to $\mathcal{G}_{build}$, completing the update of the prior map. This update process is a progressive process used to distinguish long-term stationary objects from short-term stationary objects. Additionally, at a certain frequency, *Map Filter* is performed. It takes $\mathcal{G}_{build}$ as input, combines it with the grid map M, and completes the removal and update of the prior map. Together, these two processes constitute the automatic map update module of the system.

6 Experiments

The experiment was tested in both real and simulation environments. The real environment as shown in Figure 7 contained a wide range of walls and long corridors with dynamic objects. The real experiment platform consisted of a Robosense RS-LiDAR-16 LiDAR, an Xsens MTi 30 series IMU, and an Agilex Scout-mini UGV. The simulation environment was set up as a garbage dump scenario in which the placement of trash



Notes: Segmentation results of proposed method and SOTA methods.In (a)(b), green is the segmented floor, gray is the unsegmented point cloud, in (c)(d) the green is the floor, the red is the wall, and the brown is the roof. The comparison on the right shows the misclassification of methods.
Source: Authors' own work.

Figure 8: Comparison of segmentation

cans, etc., changes. The simulated robot also contained a 16-line LIDAR and an IMU. The computing platform used for experiments was Nvidia AGX Xavier or a personal computer with an AMD R7-4800H processor.

We collected several datasets using the above sensors for our experiments. Four datasets were collected as follows, details of which were presented in later sections:

- *Office Global Dataset*: The real robot passed through most areas of the office and returned to the origin. The dataset is used for building the map.
- *Office Repeat Dataset*: In dynamic environment, the real robot passed the same place several times through different paths in order to test the repeated localization accuracy of the proposed method.
- *Garbage Dump A Dataset*: The simulated robot moved in Garbage Dump A and created a prior map for relocalization.
- *Garbage Dump B Dataset*: Change the position of most of the items in Garbage Dump A to create Garbage Dump B. The simulated robot moved through B, recording the trajectory and comparing it with the simulated odometey as the ground truth(GT).

6.1 Cloud Segmentation

The referenced method (Himmelsbach et al. 2010) and the state-of-the-art (SOTA) method patchwork++ (Lee et al. 2022) were chosen for comparison.In terms of real-time performance, we tested methods on the personal computer. The results are shown in Figure 8, where (a) (Himmelsbach et al. 2010) took 36.284 ms, (b) patchwork++ (Lee et al. 2022) took 2.232 ms, (c) the proposed method took 2.268 ms, and (d) the proposed method utilizing the time parameter took 1.559 ms. Moreover, the proposed method simultaneously segments multiple classes of points, such as floor,wall and roof.

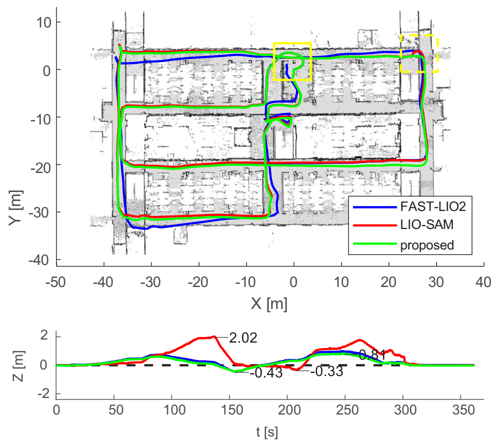
As can be seen from Figure 8, there is a significant occurrence of misclassifying non-floor points as floor points in methods (a) and (b). This misclassification phenomenon is particularly prominent at the junction between the floor and walls. However, in the proposed method, thanks to the multi-dimensional criteria for floor points, there is no misclassification in floor segmentation, demonstrating higher robustness.

6.2 Mapping Quality

We used *Office Global Dataset* for mapping quality testing and compared the proposed method with SOTA methods FAST-LIO2 (Xu et al. 2022) and LIO-SAM (Shan et al. 2020).

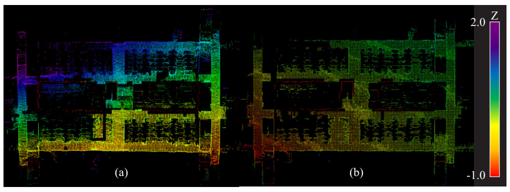
6.2.1 Trajectory

The aerial view and elevation of trajectories are shown in Figure 9. In the solid box of Figure 9, due to the lack of global optimization, the trajectory of FAST-LIO2 cannot be closed. In the dashed box of Figure 9, the



Notes: Aerial view and elevation over time on the private dataset, showing the path with different methods.
Source: Authors' own work.

Figure 9: Aerial view and elevation



Notes: Floor error of built map, (a) is the floor of LIO-SAM, (b) is the floor of proposed method. The z-axis height is displayed in rainbow colors from -1m to 2m.
Source: Authors' own work.

Figure 10: Comparison of floor error

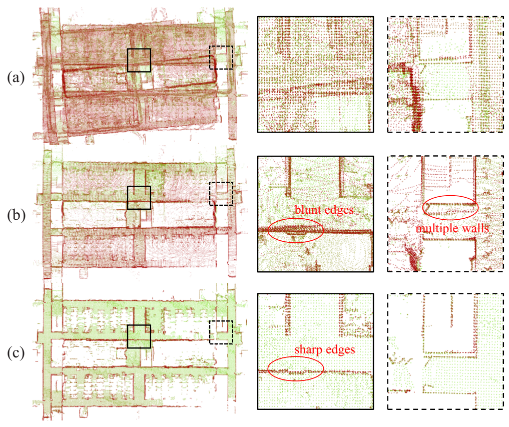
robot made a sharp turn, causing LIO-SAM to drift. As a comparison, from the aerial view, the trajectory obtained by the proposed method is closed and with no drift throughout.

In addition to the problem of drift, LIO-SAM also suffered from a large ground height estimation error, due to the feature based odometry loses several constraints. From elevation of Figure 9, the maximum height difference of LIO-SAM reached 2.35 metres, and in contrast, only 1.24 meters for the proposed method is, which is approximately half of LIO-SAM. In order to visualize the floor error, we used the Probabilistic Map Filter algorithm (Zhang et al. 2003), and extracted the floor of the map built by LIO-SAM and proposed method, as shown in Figure 10. Obviously, the proposed method has a more uniform change in height and a smaller difference.

Based on the trajectories of the respective methods, we can obtain the corresponding maps through point cloud accumulation, as shown in Figure 11. Due to the lack of global optimization, map built by FAST-LIO2 had huge errors that did not reflect the real environment. LIO-SAM broadly reflected the real environment, but there were still problems in details due to the drift in the trajectory. This was reflected in the map by the presence of blunt edges and multiple walls, as shown in the right box of Figure 11. On the other hand, the proposed method benefited from reliable loop closure detection and efficient pose graph optimization, resulting in sharp edges and a true reflection of the real environment.

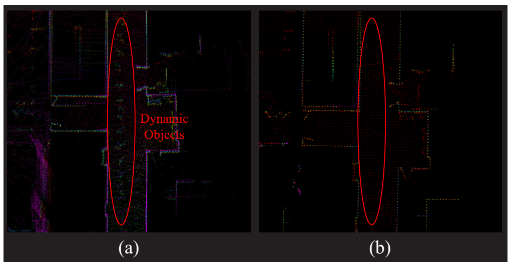
6.2.2 Dynamic Removal

Additionally, with the integration of the dynamic removal method, the proposed method successfully removed a significant portion of dynamic objects, as shown in Figure 12. In comparison to (a), with the same resolution of 0.2 m, (b) exhibited a reduction of approximately 70% in the number of point clouds, thus saving storage



Notes: (a), (b) and (c) are point cloud maps built by FAST-LIO2, LIO-SAM and the proposed method, respectively. The box on the right shows a comparison of different maps in the same place.
Source: Authors' own work.

Figure 11: Comparison of built map



Notes: Partial enlarged map of the built map, (a) is LIO-SAM, (b) is proposed method.
Source: Authors' own work.

Figure 12: Schematic diagram of dynamic filter

space.

6.3 Initial Aligning

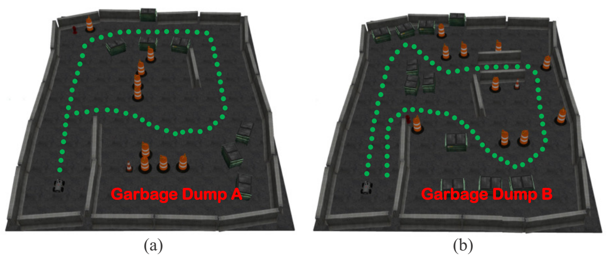
To demonstrate the performance of the proposed initial aligning method, six locations within the map were selected as test benchmarks, as shown in Figure 7.

The proposed method was compared with two other methods: the BBS (Hess et al. 2016), which is modified to suit for 3D point clouds, and the FPFH method (Rusu et al. 2009) based on descriptors. The experimental results are shown in Table 1. The BBS successfully completed the relocalization task in most cases, but failed in locations C and E. This is because these two locations have similar long corridors with other similar scenes in the map, as seen in Figure 7. Reliance solely on 2D alignment can easily result in misalignment. In contrast, the proposed method combined 2D and 3D information, enhancing robustness. In terms of the computation time, thanks to the robust floor detection of the proposed method, the initial BBS priority queue is constrained, significantly improving convergence speed. The proposed method achieved a time consumption of only 10% compared to the original BBS method. The FPFH method failed in all locations due to the high repetition of the environment.

Table 1: Align Compare (unit: second)

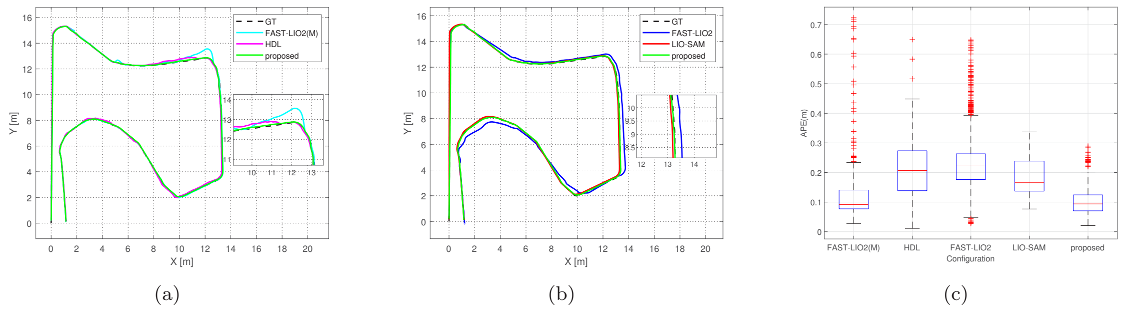
	A	B	C	D	E	F
BBS	8.351	2.894	fail	2.199	fail	2.767
FPFH	fail	fail	fail	fail	fail	fail
proposed	**0.480**	**0.172**	**0.274**	**0.149**	**0.202**	**0.297**

Source: Authors' own work.



Notes: Schematic diagram of the simulation scenario where the green points constitute the robot motion trajectory.
Source: Authors' own work.

Figure 13: Simulation scenario



Notes: Comparison of trajectories of different methods and box plots of absolute trajectory errors.
Source: Authors' own work.

Figure 14: Comparison of trajectories

6.4 Relocalization Accuracy

6.4.1 Simulation Environment

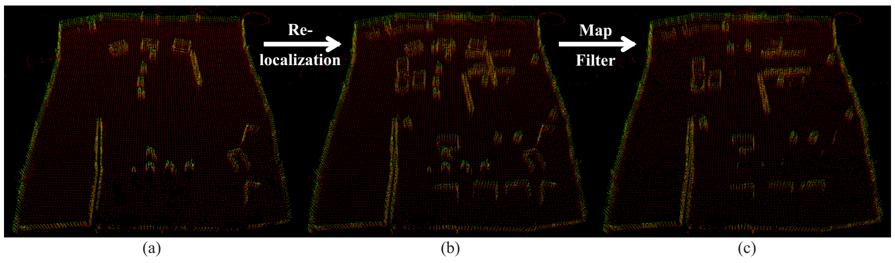
We created two simulated garbage dump environments, as illustrated in Figure 13, to evaluate the accuracy of relocalization. Initially, we utilized the *Garbage Dump A Dataset* to construct a prior map within the Garbage Dump A environment, employing the mapping method outlined previously (Figure 15(a)). Subsequently, we transitioned to a different environment, referred to as the *Garbage Dump B Dataset* to compare various relocalization methods. Specifically, we selected FAST-LIO2(M) and HDL for comparison purposes. In the case of FAST-LIO2(M), we exclusively employed frame-to-map matching within FAST-LIO2 without conducting any point cloud accumulation. On the other hand, HDL (Koide et al. 2019), a state-of-the-art relocalization method, utilized the Normal Distributions Transform (NDT) for registration and the Unscented Kalman Filter (UKF) for pose updates. HDL has demonstrated its reliability in a wide range of environments (Koide et al. 2019).

The trajectories of relocalization-based methods are depicted in Figure 14(a). A comparison of the Absolute Trajectory Error (ATE) is presented in Table 2. Notably, FAST-LIO2(M) and HDL lacked a map update

Table 2: Absolute Trajectory Error Analysis

Method	RMSE(m)	MAX(m)	<0.15m(%)	<0.2m(%)
FAST-LIO2(M)	0.153	0.723	78.157	92.491
HDL	0.222	0.649	29.730	47.635
FAST-LIO2	0.275	0.648	35.932	66.271
LIO-SAM	0.195	0.337	14.991	33.390
proposed	**0.111**	**0.289**	**85.154**	**96.587**

Source: Authors' own work.



Notes: The changes of pointcloud map during relocalization and after map filter.
Source: Authors' own work.

Figure 15: Changes of built map

module, potentially causing significant offsets when the surrounding environment deviated substantially from the prior map, as evident in the enlarged section of Figure 14(a). In contrast, our proposed method incorporates a robust map update module, yielding outstanding relocalization performance.

We conducted a similar comparison between our proposed method and state-of-the-art (SOTA) SLAM methods, with results displayed in Figure 14(b) and summarized in Table 2. Thanks to the prior map, our method consistently outperformed traditional SLAM methods.

In Figure 14(c), box plots depicting ATE for the five methods are presented. Figure 15 illustrates the evolution of the prior map throughout the entire process. During relocalization, new static points were progressively incorporated into the map, culminating in a filtration process guided by the Map Filter. A direct comparison with Figure 13 readily demonstrates the consistent alignment of the point cloud map with the actual environment.

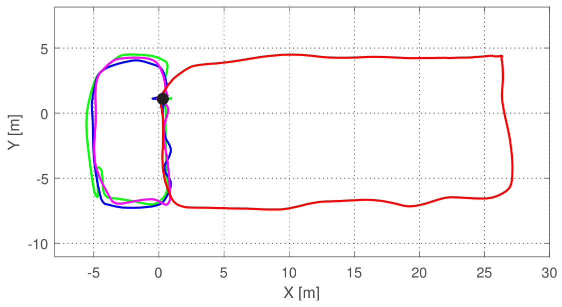
6.4.2 Real Environment

To demonstrate the method's generalizability, we conducted tests in a real-world environment. We conducted a comparative analysis, pitting our method against the relocalization methods, namely FAST-LIO2(M) and HDL, previously mentioned.

The proposed method benefits from the consistency between relocalization and mapping, which gives it an advantage in terms of repeated localization accuracy. To conduct the test, we recorded the trajectory using the *Office Repeat Dataset* as illustrated in Figure 16. A landmark was placed in the environment as shown in Figure 7(c). In addition to this, the environment was changed as shown in Figure 17 by adding cardboard boxes as new static walls. The robot was controlled to start from the landmark, pass through different paths, and then pass the same landmark four more times to obtain the robot's pose at each arrival. The standard deviation of each axis was calculated, and the repeatability of the localization was determined. As summarized in Table 3, the proposed method achieved a repeatable localization accuracy of 0.5cm, which was significantly better than others.

Figure 17 shows the updated grid map that correctly reflects the floor changes of the real environment, which benefits the robot's path planning. The updated point cloud map is shown in Figure 18. New static wall points are accurately added to the map and existing wall points are updated, making the point cloud map more relevant to the environment.

To test the stability of the relocalization system with respect to dynamic objects, we held the robot



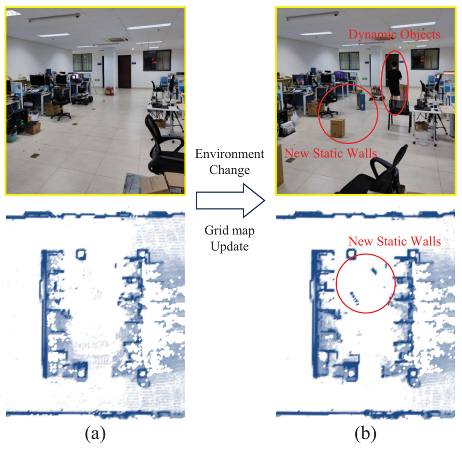
Notes: The trajectory of *Office Repeat Dataset*. The robot reaches the same landmark (black point) after different paths (different colors) in a dynamic environment.
Source: Authors' own work.

Figure 16: Trajectory of Office Repeat Dataset

Table 3: Repeatability (unit: meter)

	σ of x-axis	σ of y-axis	σ of z-axis	Error
FAST-LIO2(M)	0.00612	0.00368	0.00162	0.00732
HDL	0.00542	0.01452	0.00571	0.01652
proposed	0.00276	0.00320	0.00288	**0.00512**

Source: Authors' own work.



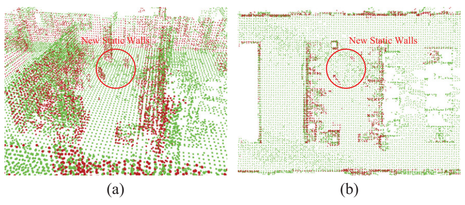
Notes: (a) are environment and grid map before updating, and (b) are after updating.
Source: Authors' own work.

Figure 17: Results of map updating in real

stationary and walked randomly around it as shown in Figure 19(a). The poses are shown as in Figure 19(b). We analyzed it quantitatively, as shown in Table 4. The proposed relocalization system is not disturbed by dynamic objects and can maintain high stability.

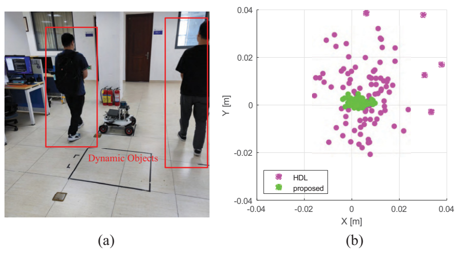
6.5 Time Analysis

Real-time performance is also one of the core advantages of the proposed method. We evaluated the time consumption of each module within per LiDAR frame of the relocalization in real environment in the proposed SLAM-RAMU for comparison with HDL on a laptop PC with AMD R7-4800H processor. The results are



Notes: (a) and (b) are plots of the updated point cloud map from different viewpoints. The green points are from the original map, and the red points are new or updated static points.
Source: Authors' own work.

Figure 18: Plots of the updated map



Notes: (a) shows that human walked randomly around the robot as dynamic objects. (b) shows the distribution of the poses from HDL and proposed method.
Source: Authors' own work.

Figure 19: Schematic diagram of dynamic environment

Table 4: Stability (unit: meter)

	σ of x-axis	σ of y-axis	σ of z-axis	Error
FAST-LIO2(M)	0.00489	0.00203	0.00108	0.00541
HDL	0.00997	0.01291	0.00468	0.01696
proposed	0.00249	0.00121	0.00124	**0.00303**

Source: Authors' own work.

Table 5: Mean Time Consumption (unit: millisecond)

		AMD R7
proposed	Deskew	3.903
	Segmention	2.483
	Odometry	7.352
	Map Update	3.767
	Total	17.505
HDL	Total	30.125

Source: Authors' own work.

shown in Table 5. The proposed method reduces the processing time by about 50% compared to HDL, meeting the real-time requirements completely.

7 Conclusion

We proposed SLAM-RAMU, a Lifelong SLAM system using LiDAR and IMU, to provide accurate and consistent relocalization and autonomous map updating for challenging dynamic environments. In the initial mapping stage, the global optimization is combined with robust local odometry and loop closure detection, in order to obtain a more accurate trajectory. At the same time, the point cloud is segmented to obtain static point cloud map without dynamic objects. In the relocalization stage, the 2D branch-and-bound search and 3D ICP registration are united for fast initial pose alignment. Then, the scan-map registration is performed using the segmented point cloud on the prior map to achieve real-time pose estimation. Simultaneously, the original point cloud map is updated to reflect the real environment based on the segmented point cloud and grid map. Finally, we compared to other state-of-the-art methods in simulation and real environments, and SLAM-RAMU showed higher mapping quality, faster initial aligning speed and higher repeated localization accuracy.

References

Besl, P. J. & McKay, N. D. (1992), Method for registration of 3-d shapes, *in* 'Sensor fusion IV: control paradigms and data structures', Vol. 1611, Spie, pp. 586–606.

Einhorn, E. & Gross, H.-M. (2015), 'Generic ndt mapping in dynamic environments and its application for lifelong slam', *Robotics and Autonomous Systems* **69**, 28–39.

Fankhauser, P. & Hutter, M. (2016), 'A universal grid map library: Implementation and use case for rough terrain navigation', *Robot Operating System (ROS) The Complete Reference (Volume 1)* pp. 99–120.

Hess, W., Kohler, D., Rapp, H. & Andor, D. (2016), Real-time loop closure in 2d lidar slam, *in* '2016 IEEE international conference on robotics and automation (ICRA)', IEEE, pp. 1271–1278.

Himmelsbach, M., Hundelshausen, F. V. & Wuensche, H.-J. (2010), Fast segmentation of 3d point clouds for ground vehicles, *in* '2010 IEEE Intelligent Vehicles Symposium', IEEE, pp. 560–565.

Jiang, Y., Wang, T., Shao, S. & Wang, L. (2023), '3d slam based on ndt matching and ground constraints for ground robots in complex environments', *Industrial Robot: the international journal of robotics research and application* **50**(1), 174–185.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J. & Dellaert, F. (2012), 'isam2: Incremental smoothing and mapping using the bayes tree', *The International Journal of Robotics Research* **31**(2), 216–235.

Kim, G. & Kim, A. (2020), Remove, then revert: Static point cloud map construction using multiresolution range images, *in* '2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 10758–10765.

Kim, G. & Kim, A. (2022), Lt-mapper: A modular framework for lidar-based lifelong mapping, *in* '2022 International Conference on Robotics and Automation (ICRA)', IEEE, pp. 7995–8002.

Koide, K., Miura, J. & Menegatti, E. (2019), 'A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement', *International Journal of Advanced Robotic Systems* **16**(2), 1729881419841532.

Lee, S., Lim, H. & Myung, H. (2022), Patchwork++: Fast and robust ground segmentation solving partial under-segmentation using 3d point cloud, *in* '2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 13276–13283.

Lim, H., Hwang, S. & Myung, H. (2021), 'Erasor: Egocentric ratio of pseudo occupancy-based dynamic object removal for static 3d point cloud map building', *IEEE Robotics and Automation Letters* **6**(2), 2272–2279.

Rusu, R. B., Blodow, N. & Beetz, M. (2009), Fast point feature histograms (fpfh) for 3d registration, *in* '2009 IEEE international conference on robotics and automation', IEEE, pp. 3212–3217.

Shan, T. & Englot, B. (2018), 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)', IEEE, pp. 4758–4765.

Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C. & Rus, D. (2020), Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping, *in* '2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)', IEEE, pp. 5135–5142.

Xu, W., Cai, Y., He, D., Lin, J. & Zhang, F. (2022), 'Fast-lio2: Fast direct lidar-inertial odometry', *IEEE Transactions on Robotics* **38**(4), 2053–2073.

Xu, W. & Zhang, F. (2021), 'Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter', *IEEE Robotics and Automation Letters* **6**(2), 3317–3324.

Zhang, J. & Singh, S. (2014), Loam: Lidar odometry and mapping in real-time., *in* 'Robotics: Science and Systems', Vol. 2, Berkeley, CA, pp. 1–9.

Zhang, K., Chen, S.-C., Whitman, D., Shyu, M.-L., Yan, J. & Zhang, C. (2003), 'A progressive morphological filter for removing nonground measurements from airborne lidar data', *IEEE transactions on geoscience and remote sensing* **41**(4), 872–882.

Zhao, M., Guo, X., Song, L., Qin, B., Shi, X., Lee, G. H. & Sun, G. (2021), A general framework for lifelong localization and mapping in changing environment, *in* '2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 3305–3312.

Zhu, F., Ren, Y. & Zhang, F. (2022), Robust real-time lidar-inertial initialization, *in* '2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 3948–3955.

8 Corresponding Author

Xunyu Zhong can be contacted at: zhongxunyu@xmu.edu.cn