LiDAR SLAM with Plane Adjustment for Indoor Environment

Lipu Zhou[1], Daniel Koppel[1] and Michael Kaess[2]

***Abstract*—Planes ubiquitously exist in the indoor environment. This paper presents a real-time and low-drift LiDAR SLAM system using planes as the landmark for the indoor environment. Our algorithm includes three components: localization, local mapping and global mapping. The localization component performs real-time and global registration, instead of the scan-to-scan registration adopted in the state-of-the-art LiDAR odometry and mapping (LOAM) framework that yields lower fidelity poses. The local mapping component optimizes poses of the keyframes within a sliding window and parameters of the planes observed by these keyframes. The global mapping component conducts global plane adjustment (GPA) that jointly refines plane parameters and keyframe poses. The GPA is triggered when planes are revisited, rather than a place is revisited. This can establish constraints among remote places, and correct the drift without having to go back to a previously visited place. We adopt the point-to-plane distance to construct the cost functions of all the three components. Although this distance results in a large-scale least-squares problem that seems not suitable for real-time applications, we propose efficient algorithms to solve the resulting minimization problems by exploiting the special structure of the point-to-plane distance. Experimental results show that our algorithm achieves real-time performance and outperforms the state-of-the-art LiDAR SLAM algorithms.**

***Index Terms*—SLAM, Range Sensing, Mapping, Localization**

I. INTRODUCTION

SIMULTANEOUS localization and mapping (SLAM) using LiDAR is essential for many indoor robotic applications, such as autonomous navigation, control and motion planning. Additionally, LiDAR SLAM can yield a dense 3D map that is important for many indoor computer vision tasks, such as augmented reality (AR), and 3D object classification [1]. Due to its importance, this paper focuses on investigating LiDAR SLAM in the indoor environment.

Planar surfaces ubiquitously exist in the indoor environment. Thus, planes are widely used in the scan registration in previous works, such as surfel-based methods [2], [3], planar variants [4], [5] of the iterative closest point (ICP) framework [6], and the LiDAR odometry and mapping (LOAM) framework [7] and its variants [8]–[10]. In these works, planes are not jointly optimized with poses. Instead, they are estimated on the fly from a small set of the points in the global point cloud which is generated by assembling previous LiDAR scans together. This may result in a vicious loop. Specifically, the pose error lowers the quality of the global point cloud, which in turn reduces the accuracy of the pose estimation. We overcome this problem by jointly optimizing plane parameters and LiDAR poses, called **plane adjustment** (**PA**) in this paper that is the counterpart of the bundle adjustment (BA) in visual SLAM. Besides, planar objects have two sides, and a LiDAR may see them at different locations, as shown in Fig. 1. As the two sides are typically close to each other, the nearest-neighbor matching scheme widely adopted in previous works may result in wrong data association that in turn may lead to a large pose estimation error, named the **double-side issue** in this paper. This issue can be solved by comparing the plane normal direction.

Manuscript received: February 24, 2021; Revised: May 13, 2021; Accepted: June 7, 2021.

This paper was recommended for publication by Editor Javier Civera upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by Magic Leap

[1]The authors are with Magic Leap, 1376 Bordeaux Dr, Sunnyvale, CA 94089, USA {lzhou,dkoppel}@magicleap.com

[2]Michael Kaess is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA kaess@cmu.edu

Digital Object Identifier (DOI): see top of this page.

This work presents a LiDAR SLAM system using planes as the landmark based on the state-of-the-art visual SLAM framework, ORB-SLAM [11]. Our algorithm has three components: **localization**, **local mapping** and **global mapping**. The localization component establishes the local-to-global data association, and then undistorts and registers a new scan to the global plane model in real time. The local and global mapping components correct the drift and improve the map through PA. The cost function is crucial for a least-squares problem [12]. In the three components, instead of choosing the widely used plane-to-plane cost [13]–[16], we adopt the more robust point-to-plane cost [17]. We show that the seemingly prohibitive computational load from the point-to-plane cost can be significantly reduced, so our algorithm can achieve robustness and real time simultaneously.

The main contributions of this paper include:

- We adopt the point-to-plane distance to construct the cost function, and verify that the resulting large-scale least-squares problems can be efficiently solved.
- We introduce a new loop closure criterion and present a corresponding detection method based on validation of the geometric consistency. We trigger the loop closure if existing planes are revisited, rather than a place is revisited, as demonstrated in Fig. 2. The traditional loop closure detection method based on appearance similarity [2], [18]–[22] is not suitable for this task.
- We introduce the forward ICP flow to enable real-time global registration. In contrast, the LOAM framework [7] and its variants [8]–[10] perform low-frequency global registration. Besides, the forward ICP flow tracks planar points from the nth scan to the $(n+1)th$ scan which generates a smaller number of constraints than the ICP process [22] where the whole $(n+1)th$ scan is used to find the nearest planar points in the nth scan.
- We solve the double-side issue that may cause wrong data association. We make the plane normal point toward the LiDAR center. Then the two sides of a plane can be easily distinguished.

II. Related Work

Scan Registration ICP [6] and its variants [4], [5] provide a general approach to align two point clouds. Surfel-based methods [2], [23] give an alternative way for scan registration. These approaches ignore the motion within a scan, which may result in a suboptimal pose estimation. The approaches [3], [24] model the in-scan motion (or called the continuous-time SLAM) and combine it with the surfel. Sparse feature-based registration is generally more efficient. A well-known feature-based framework with in-scan motion correction is LOAM [7]. LOAM uses plane and edge as features, and solves the SLAM problem by two algorithms. The first algorithm provides real-time but low-fidelity pose estimation by scan-to-scan registration. The second algorithm refines the pose by registering a scan to the global point cloud. It runs at a much lower frequency due to the higher computational complexity. LeGO-LOAM [8] reduces the number of features used in LOAM and leverages the ground plane to assist segmentation and optimization. Lin *et al.* [9] extend LOAM to the solid state LiDAR. Deschaud [25] uses implicit moving least squares (IMLS) to represent the surface, and presents a scan-to-model registration method. This algorithm shows better accuracy

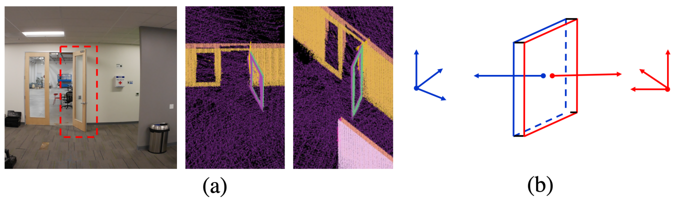


Fig. 1. Double-side issue of planar objects. Fig. (a) demonstrates an example of the double-side issue of the planar object. There are holes in the glass areas, as the laser can see through the glass. Planar objects (such as doors and walls) have two sides that are close to each other. This may result in wrong data association for the nearest-neighbor search method that are widely used in previous works. We use the plane normal to solve this problem as shown in (b). We make the normal of a plane point toward the origin of the sensor coordinate system. As demonstrated in (a), this method succeeds in distinguishing the two sides of the door and the wall.

than LOAM, but it does not achieve real-time performance. The recent work LOL [10] introduces place recognition into LOAM to correct the accumulated drift of LOAM. One major drawback of the mapping algorithm based on incremental scan registration is the lack of a counterpart of the BA in visual SLAM. The pose estimation errors end up accumulating into the global map, which in turn degrades the pose estimation accuracy. This forms a vicious circle. Besides, these algorithms adopt the nearest-neighbor search to establish data association, which is hard to handle the double-side issue mentioned above. Furthermore, the global registration is generally computationally demanding, as directly establishing local-to-global data association requires to search large global K-D trees, and except for the surfel-based methods, the global model (planes or lines) are generally calculated on the fly.

SLAM with Planes In the algorithms mentioned above, planes are not explicitly detected, matched, parameterized and optimized. In fact, planes have also been explicitly treated as landmarks in SLAM systems of various depth sensors. Pathak *et al.* [13] introduce an algorithm named minimally uncertain maximal consensus to find the plane correspondences for pose estimation. Kaess [14] introduces a new plane parameterization based on the quaternion. Besides, he presents the relative plane formulation to improve the convergence speed. This formulation is adopted in [15] for the global optimization after loop closure. Geneva *et al.* [16] introduce the closest point (CP) to parameterize planes, which outperforms the plane parameterization introduced in [14]. Planes together with other features are also used in the EKF framework for pose estimation [26]. In these works, they adopt the plane-to-plane cost. As shown in [17], the point-to-plane cost generally converges faster and leads to more accurate results. Ferrer [27] derives the closed-form gradient of an algebraic point-to-plane cost. As only the gradient is used in the optimization, it is generally less efficient than the Levenberg-Marquardt (LM) method [28]. The recent work [29] derives the first and second order derivatives of the point-to-plane cost, which is sufficient to apply the LM algorithm. However, as its computational complexity is $O(N^2)$ where N is the number of points, this method is hard to be applied to a large-scale problem. This paper adopts the point-to-plane cost. We show the resulting continuous-time scan registration, **local PA** (**LPA**) and **global PA** (**GPA**) can be solved efficiently. Although similar concepts are used in our previous work π-LSAM [22], this work has significant improvement compared to [22]. For the scan registration, we adopt the first-order Taylor expansion to simplify the computation rather than the LM algorithm used in π-LSAM. For LPA, π-LSAM adopts the marginalization strategy which is difficult to remove the false data association and difficult to recover the marginalized information. In this paper, we overcome this problem by introducing the cost function constructed by fixing old keyframe poses and parameters of unseen planes. Using this cost function, we can easily remove false data associations and add prior constraints, which are generated from previous observations, on a plane when the plane is revisited. To speed up the computation, we introduce an

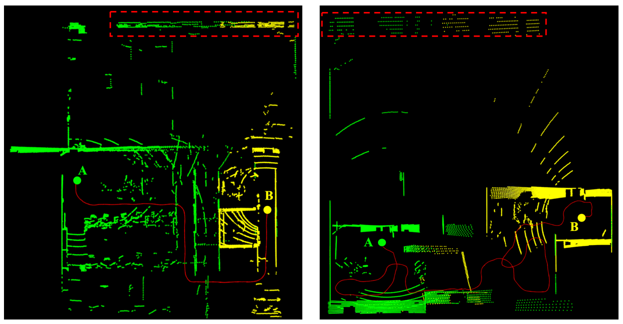


Fig. 2. Two examples of establishing loop closure between two remote locations. Walls marked in the red rectangles above are initially observed at location A, and then are revisited at location B. The revisited walls can provide constraints to correct drift. The scan A and B are almost non-overlapping. Since planes are infinite, they can be revisited without returning to the location where they were first observed. This significantly differs from visual SLAM. The 3D models of the above two datasets are shown in Fig. 8 (c) and (d).

integrated cost matrix (ICM) for each plane. Moreover, the GPA in π-LSAM is triggered when a robot returns back to a previous location. In this work, we conduct GPA when a plane is revisited.

Loop closure is important for a SLAM system. To detect a loop, previous approaches generally require the sensor to return back to a visited place [2], [18]–[22]. This makes the mapping inefficient. **Although loop closure is typically related to place recognition, essentially it is to establish the data association between current observations and previously visited landmarks.** Thus, we consider that a loop occurs when previous planes are revisited. Since a plane is an unbounded object, it can be revisited at a place far from the first place where this plane was observed, as shown in Fig. 2. Thus we can correct the drift without having to go back to a previously visited location. This can speeds up the mapping.

III. System Overview

Our system has three components: localization, local mapping and global mapping, as shown in Fig. 3.

The localization component establishes the local-to-global point-to-plane data association. Then it undistorts and registers the current scan to the global plane model, and determines whether a new keyframe should be added or not.

The local mapping is triggered once a new keyframe is added. This component detects new planes and matches them with the existing planes, then it optimizes the keyframes within a sliding window and the the planes observed by these keyframes.

The global mapping is performed when a plane is revisited. This component jointly optimizes all the keyframe poses and plane parameters.

IV. Notations and Preliminaries

In this paper, we use italic, boldfaced lowercase and boldfaced uppercase letters to represent scalars, vectors and matrices, respectively.

1) Pose: We represent a rigid transformation as a rotation matrix $\mathbf{R} \in SO(3)$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$, or more concisely as a transformation matrix $\mathbf{T} \in SE\left(3\right)$

This paper adopts the angle-axis parameterization $\boldsymbol{\omega} = [\omega_1; \omega_2; \omega_3]$ to represent $\mathbf{R}$. Let us define the skew matrix of $\boldsymbol{\omega}$ as $[\boldsymbol{\omega}]_\times$ that lies in the tangent space $\mathfrak{so}\left(3\right)$ of the manifold $SO\left(3\right)$ at the identity [14]. The exponential map $exp : \mathfrak{so}\left(3\right) \to SO\left(3\right)$ is

$$exp\left([\boldsymbol{\omega}]_\times\right) = \mathbf{I} + \frac{sin\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|}[\boldsymbol{\omega}]_\times + \frac{1 - cos\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times^2. \quad (1)$$

Accordingly, we parameterize $\mathbf{T}$ as $\boldsymbol{x} = [\boldsymbol{\omega}; \mathbf{t}]$.

Let $\mathbb{S}_k$ denote the kth scan captured within $(t_k^s, t_k^e]$. $\mathbb{S}_k$ is associated with two poses $\mathbf{T}_k^s$ and $\mathbf{T}_k^e$ which represent the LiDAR pose at t_k^s and t_k^e, respectively. **$\mathbf{T}_k^s$ is estimated by the localization component, and $\mathbf{T}_k^e$ is optimized in LPA and GPA.** The relationship between the two poses is

$$\mathbf{T}_{k-1}^e = \mathbf{T}_k^s. \quad (2)$$

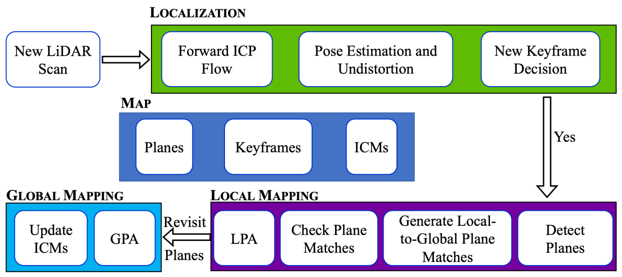


Fig. 3. Overview of our SLAM system. It has three components, *i.e.*, localization, local mapping and global mapping. The three components all interact with the map. LPA and GPA represent the local and global plane adjustment introduced in section VI-G and VII, respectively. ICM is defined in section VI-E, which is used to speed up LPA.

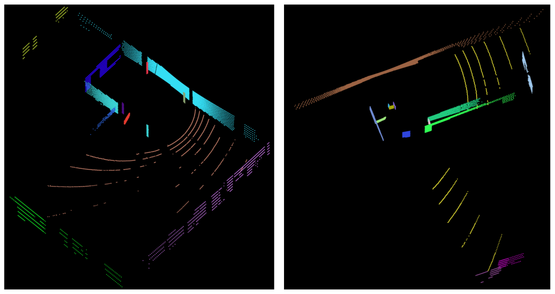


Fig. 4. Planes detected by our algorithm.

This relationship is important to correct the drift of the pose $\mathbf{T}_k^e$ after LPA and GPA. To simplify the notation in the following description, we use $\mathbf{T}_k$ to represent $\mathbf{T}_k^s$ or $\mathbf{T}_k^e$ if its meaning can be recognized by the context.

2) Plane: Planar objects have 2 sides, which are generally close to each other, as shown in Fig. 8. This may cause wrong data association. We call this issue the double-side issue. We set the plane normal looking toward the LiDAR origin. Then the two sides of a planar object can be easily distinguished.

We use $\boldsymbol{\pi} = [\mathbf{n}; d]$ to represent a plane, where $\mathbf{n}$ is the normal looking toward the LiDAR origin with $\|\mathbf{n}\|_2 = 1$, and d is the distance from the origin to the plane. The point-to-plane residual δ for a point $\mathbf{p}$ on $\boldsymbol{\pi}$ observed at pose $\mathbf{T}$ is

$$\delta = \boldsymbol{\pi}^T \mathbf{T}\bar{\mathbf{p}}, \tag{3}$$

where $\bar{\mathbf{p}}$ denotes the homogeneous coordinates of $\mathbf{p}$. According to the requirement of different tasks, $\boldsymbol{\pi}$ and $\mathbf{T}$ can be optimized jointly or either of them can be fixed. In the optimization, we adopt the closest point (CP) $\boldsymbol{\eta}$ to parameterize $\boldsymbol{\pi}$, *i.e.*, $\boldsymbol{\eta} = \mathbf{n}d$ [16]. Given $\boldsymbol{\eta}$, we have $\boldsymbol{\pi} = [\frac{\boldsymbol{\eta}}{||\boldsymbol{\eta}||_2}; ||\boldsymbol{\eta}||_2]$.

V. LOCALIZATION

A. Initial Global Planes

We assume the LiDAR starts from a stationary state. Thus the first frame does not suffer from the motion distortion. We extract planes from the LiDAR point cloud based on a region growing method similar to [30]. Specifically, we first estimate the normal of each point, and then we segment the point cloud by clustering points with similar normals. For each cluster, we apply the RANSAC algorithm to detect a plane. We keep the plane with more than $\mathcal{N}$ points ($\mathcal{N} = 30$ in our experiments).

B. Forward ICP Flow

In this paper, we introduce the forward ICP flow to track planes scan by scan, instead of detecting planes in each scan as done in previous works [13]–[16]. This can reduce the runtime and naturally propagate the local-to-global data association scan by scan. Fig. 4 shows the results of this algorithm.

Assume $\mathbb{S}_k$ and $\mathbb{S}_{k-1}$ are two consecutive scans. Suppose $\mathbb{P}_{k-1,i}$ is the set of points belonging to the ith plane detected in $\mathbb{S}_{k-1}$ with the parameters $\boldsymbol{\pi}_{k-1,i}$, and $\mathbb{P}_{k-1,i}$ is associated with the m_ith global plane $\boldsymbol{\pi}^g_{m_i}$. That is to say we have a set of local-to-global point-to-plane correspondences $\mathbb{P}_{k-1,i} \leftrightarrow \boldsymbol{\pi}^g_{m_i}$ for $\mathbb{S}_{k-1}$. We then track $\mathbb{P}_{k-1,i}$ in scan $\mathbb{S}_k$.

We build a K-D tree on $\mathbb{S}_k$. For each point $\mathbf{p}_{k-1,i,j} \in \mathbb{P}_{k-1,i}$, we find the n nearest neighbors in $\mathbb{S}_k$ ($n = 2$ in our experiments). Eliminating redundant points, we obtain a point set $\tilde{\mathbb{P}}_{k,i}$. We apply the RANSAC algorithm to fit a plane $\boldsymbol{\pi}_{k,i}$ from $\tilde{\mathbb{P}}_{k,i}$, and obtain an inlier set $\mathbb{P}_{k,i}$. We then expand $\mathbb{P}_{k,i}$ by incorporating nearby points whose distances to $\boldsymbol{\pi}_{k,i}$ are smaller than $5cm$. If the number of points in $\mathbb{P}_{k,i}$ is larger than $\mathcal{N}$ ($\mathcal{N}$ is defined in section V-A), and the angle between the normals of $\boldsymbol{\pi}_{k,i}$ and $\boldsymbol{\pi}_{k-1,i}$ is smaller than θ (we use $\theta = 15°$), we obtain a set of local-to-global point-to-plane correspondences $\mathbb{P}_{k,i} \leftrightarrow \boldsymbol{\pi}^g_{m_i}$ for scan $\mathbb{S}_k$. Here the angular threshold θ is used to solve the double-side issue.

The π-LSAM [22] adopts an opposite direction to establish the local-to-global constraints. As the number of points in $\mathbb{S}_k$ is generally larger than the number of planar points $\cup\mathbb{P}_{k-1,i}$ in $\mathbb{S}_{k-1}$, our new algorithm generally results in a smaller number of candidate data associations, which will speed up the computation.

C. State Estimation and Scan Undistortion

Relative Pose We define the rigid transformation from scan k to scan $k-1$ as $\mathbf{T}_{k-1,k}$ with the rotational and translational components $\mathbf{R}_{k-1,k}$ and $\mathbf{t}_{k-1,k}$, respectively. Suppose $\boldsymbol{\omega}_{k-1,k}$ is the angle-axis representation of $\mathbf{R}_{k-1,k}$. We use $\boldsymbol{x}_{k-1,k} = [\boldsymbol{\omega}_{k-1,k}; \mathbf{t}_{k-1,k}]$ to parameterize $\mathbf{T}_{k-1,k}$.

We denote the rigid transformation at the last point of scan $k-1$ and scan k as $\mathbf{T}_{k-1}$ and $\mathbf{T}_k$, respectively. The relationship between $\mathbf{T}_{k-1}$, $\mathbf{T}_k$ and $\mathbf{T}_{k-1,k}$ is

$$\mathbf{T}_k = \mathbf{T}_{k-1}\mathbf{T}_{k-1,k}. \tag{4}$$

We aim to calculate $\boldsymbol{x}_{k-1,k}$, after which we can in turn get $\mathbf{T}_{k-1,k}$ and finally obtain $\mathbf{T}_k$.

Linear Interpolation of Relative Pose As mentioned in section IV-1, the points in scan $\mathbb{S}_k$ are captured within $(t_k^s, t_k^e]$. When the LiDAR is moving, $\mathbb{S}_k$ will be distorted by the motion. This problem is generally addressed by linearly interpolating the pose for each point in $\mathbb{S}_k$ [7], [9]. For $t \in (t_k^s, t_k^e]$, we define $s = \frac{t - t_k^s}{t_k^e - t_k^s} \in (0, 1]$. Then $\mathbf{R}^t_{k-1,k}$ and $\mathbf{t}^t_{k-1,k}$ of the rigid transformation $\mathbf{T}^t_{k-1,k}$ can be estimated by the following linear interpolation

$$\mathbf{R}^t_{k-1,k} = exp\left(s\left[\boldsymbol{\omega}_{k-1,k}\right]_\times\right) \text{ and } \mathbf{t}^t_{k-1,k} = s\mathbf{t}_{k,k-1}. \tag{5}$$

Pose Estimation Cost Now, let us consider the set of correspondences $\mathbb{P}_{k,i} \leftrightarrow \boldsymbol{\pi}^g_{m_i}$. Assume the jth point $\mathbf{p}_{k,i,j} \in \mathbb{P}_{k,i}$ is captured at time $t_{k,i,j}$. According to (3), (4) and (5), the residual for $\mathbf{p}_{k,i,j} \leftrightarrow \boldsymbol{\pi}^g_{m_i}$ can be written as

$$\delta_{k,i,j}\left(\boldsymbol{x}_{k-1,k}\right) = \left(\boldsymbol{\pi}^g_{m_i}\right)^T \mathbf{T}_{k-1}\mathbf{T}^{t_{k,i,j}}_{k-1,k}\bar{\mathbf{p}}_{k,i,j}. \tag{6}$$

Define $\mathbf{l}^g_{m_i} = \left(\boldsymbol{\pi}^g_{m_i}\right)^T \mathbf{T}_{k-1}$. Substituting it into (6), we have

$$\delta_{k,i,j}\left(\boldsymbol{x}_{k-1,k}\right) = \mathbf{l}^g_{m_i}\mathbf{T}^{t_{k,i,j}}_{k-1,k}\bar{\mathbf{p}}_{k,i,j}. \tag{7}$$

Suppose we have N_k sets of point-to-plane correspondences $\left\{\mathbb{P}_{k,i} \leftrightarrow \boldsymbol{\pi}^g_{m_i}\right\}_{i=1}^{N_k}$, and $\mathbb{P}_{k,i}$ has $N_{k,i}$ points. We can formulate the least-squares cost function for $\boldsymbol{x}_{k-1,k}$ as follows

$$\min_{\boldsymbol{x}_{k-1,k}} \sum_{i=1}^{N_k}\sum_{j=1}^{N_{k,i}} \delta^2_{k,i,j}\left(\boldsymbol{x}_{k-1,k}\right). \tag{8}$$

As different points have different transformation matrices, solutions for the traditional point-to-plane registration problem, such as [31], are not suitable for (8). Here we employ the linearization of $\mathbf{R}^t_{k-1,k}$ to simplify this problem.

Approximation of Rotation Assuming the rotation within $\mathbb{S}_k$ is small, then we can adopt the first-order Taylor expansion to approximate the $\mathbf{R}^t_{k,k-1}$ in (5):

$$\mathbf{R}^t_{k-1,k} \approx \mathbf{I} + s\left[\boldsymbol{\omega}_{k-1,k}\right]_\times . \tag{9}$$

Let us define $\mathbf{l}^g_{m_i} = \left[a^g_{m_i}, b^g_{m_i}, c^g_{m_i}, d^g_{m_i}\right]$ and $\bar{\mathbf{p}}_{k,i,j} = \left[x_{k,i,j}, y_{k,i,j}, z_{k,i,j}, 1\right]^T$. Substituting $\mathbf{l}^g_{m_i}$, $\bar{\mathbf{p}}_{k,i,j}$ and (9) into (7) and expanding it, we obtain a linear constrain on $\boldsymbol{x}_{k-1,k}$:

$$\begin{aligned}
&\mathbf{a}_{k,i,j}\boldsymbol{x}_{k-1,k} = b_{k,i,j}, \text{where}\\
&\mathbf{a}_{k,i,j} = s_{k,i,j}\cdot\left[c^g_{m_i}y_{k,i,j} - b^g_{m_i}z_{k,i,j},\ a^g_{m_i}z_{k,i,j} - c^g_{m_i}x_{k,i,j},\right.\\
&\qquad\left. b^g_{m_i}x_{k,i,j} - a^g_{m_i}y_{k,i,j},\ a^g_{m_i},\ b^g_{m_i},\ c^g_{m_i}\right],\\
&b_{k,i,j} = -\left(a^g_{m_i}x_{k,i,j} + b^g_{m_i}y_{k,i,j} + c^g_{m_i}z_{k,i,j} + d^g_{m_i}\right).
\end{aligned} \tag{10}$$

In $\mathbf{a}_{k,i,j}$, we set $s_{k,i,j} = \frac{t_{k,i,j} - t^s_k}{t^e_k - t^s_k}$. Stacking all the constraints from $\left\{\mathbb{P}_{k,i} \leftrightarrow \boldsymbol{\pi}^g_{m_i}\right\}_{i=1}^{N_k}$, we obtain a linear system for $\boldsymbol{x}_{k-1,k}$

$$\mathbf{A}_k\boldsymbol{x}_{k-1,k} = \mathbf{b}_k. \tag{11}$$

Thus we have a closed-form solution for $\boldsymbol{x}_{k-1,k} = -\left(\mathbf{A}_k^T\mathbf{A}_k\right)^{-1}\mathbf{A}_k^T\mathbf{b}_k$. After we get $\boldsymbol{x}_{k-1,k}$, we can recover $\mathbf{R}_{k-1,k}$ using the exponential map (1).

Iterative Approximation When the rotation between $\mathbb{S}_{k-1}$ and $\mathbb{S}_k$ is large, the first-order Taylor expansion (9) cannot well approximate $\mathbf{R}^t_{k,k-1}$. In this case, we iteratively refine the solution. Specifically, we start with calculating an initial estimation $\boldsymbol{x}^0_{k-1,k}$ from (11) which generates an initial transformation matrix $\mathbf{T}^0_{k-1,k}$. If $||\boldsymbol{\omega}^0_{k-1,k}||_2 > \tau$, we update $\bar{p}_{k,i,j} = \mathbf{T}^{0,t_{k,i,j}}_{k-1,k}\bar{p}_{k,i,j}$, where $\mathbf{T}^{0,t_{k,i,j}}_{k-1,k}$ is the linear interpolation of $\mathbf{T}^0_{k,k-1}$ at time $t_{k,i,j}$ calculated by (5). We then use the new $\bar{p}_{k,i,j}$ to generate a new linear system (11). We solve it to get $\boldsymbol{x}^1_{k-1,k}$ and the corresponding $\mathbf{T}^1_{k-1,k}$. We repeat these steps at most $\mathcal{K}$ times or until the rotation angle is smaller than τ. Given m iterations, $\mathbf{T}_{k-1,k}$ can be computed by

$$\mathbf{T}_{k-1,k} = \mathbf{T}^m_{k-1,k}\mathbf{T}^{m-1}_{k-1,k}\cdots\mathbf{T}^0_{k-1,k}. \tag{12}$$

During the iteration, we adopt the bisquare weight for robustness as in [7]. In our experiments, we set $\mathcal{K} = 5$ and $\tau = 0.5°$.

D. Keyframe Decision

We use the following criteria to add a keyframe:

- The distance between the current frame and the last keyframe is larger than $0.2m$.
- The ration angle between the current frame and the last keyframe is larger than $10°$.
- 20% of the points in the current frame are not tracked.

When one of the above criteria is met, we trigger the local mapping process for this new keyframe.

VI. LOCAL MAPPING

A. Add New Keyframe

Detect Planes For a new keyframe, we first undistort the scan by applying the transformation in (5). Then we detect planes in the point cloud that has not been tracked, using the method described in section V-A. We keep planes that have more than $\mathcal{N}$ points ($\mathcal{N}$ is defined in section V-A).

Match Planes Using the estimated pose, new local planes are transformed into the global frame and then matched with the global planes. Specifically, for a new plane, we first select the global planes whose normals are nearly parallel to its normal (the angle between the normals is less than $10°$ in our experiments). Next, we compute the mean distance from the points of this new plane to each of the candidate global planes, and keep the one with the smallest mean distance. Let $\bar{d}_{min}$ denote this smallest mean distance. If $\bar{d}_{min} < \gamma$ (we set $\gamma = 5cm$ in our experiments), we keep this new correspondence. If $\bar{d}_{min} < 3\gamma$, we trigger an extra **geometric consistency check** (**GCC**) described below, otherwise we add a new global plane.

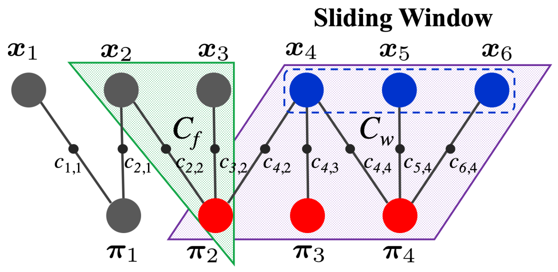


Fig. 5. A schematic of the factor graph of LPA with window size three. $\boldsymbol{x}_i$ represents the state of the ith keyframe, and $\boldsymbol{\pi}_j$ denotes the jth plane. $c_{i,j}$ is the cost from the set of points of $\boldsymbol{\pi}_j$ recorded at $\boldsymbol{x}_i$. In LPA, we optimize the poses of the keyframes within a sliding window (*i.e.*, $\boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6$) and the planes observed by them (*i.e.*, $\boldsymbol{\pi}_2, \boldsymbol{\pi}_3, \boldsymbol{\pi}_4$). Other poses (*i.e.*, $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$) and planes (*i.e.*, $\boldsymbol{\pi}_1$) are fixed. The cost function contains two parts, *i.e.*,C_f and C_w, defined in (14). Both of them can be significantly simplified.

Geometric Consistency Check We represent the linear system (11) generated from the original correspondences as $\mathbf{A}^o_k\boldsymbol{x}_{k-1,k} = \mathbf{b}^o_k$, and denote its solution as $\hat{\boldsymbol{x}}^o_{k-1,k}$. Let $\mathbf{A}^i_k\boldsymbol{x}_{k-1,k} = \mathbf{b}^i_k$ represent the linear constraints obtained from the ith new correspondence. We solve the following augmented linear system to check the ith new correspondence

$$\mathbf{A}^{o+i}_k\boldsymbol{x}_{k-1,k} = \mathbf{b}^{o+i}_k,\ \text{where}\ \mathbf{A}^{o+i}_k = \begin{bmatrix}\mathbf{A}^o_k\\ \mathbf{A}^i_k\end{bmatrix}, \mathbf{b}^{o+i}_k = \begin{bmatrix}\mathbf{b}^o_k\\ \mathbf{b}^i_k\end{bmatrix}. \tag{13}$$

Let $\hat{\boldsymbol{x}}^{o+i}_{k-1,k}$ denote the solution of (13) and $\bar{d}^{o+i}_{min}$ denote the mean point-to-plane distance for the solution $\hat{\boldsymbol{x}}^{o+i}_{k-1,k}$. We keep the ith new correspondence if the following two criteria are met:

- $\bar{d}^{o+i}_{min} < \gamma$.
- $||\mathbf{A}^o_k\hat{\boldsymbol{x}}^{o+i}_{k-1,k} - \mathbf{b}^o_k||_2 < \lambda||\mathbf{A}^o_k\hat{\boldsymbol{x}}^o_{k-1,k} - \mathbf{b}^o_k||_2$. This requires the cost of the original constraints increase only slightly. We set $\lambda = 1.05$ in the experiments.

If the ith new correspondence only meets the second criterion and $\bar{d}^{o+i}_{min} < 2\gamma$, which means the new correspondence seems consistent to the current plane model, we treat it as an **undetermined correspondence** (**UC**). This situation may happen if the drift is large or the new plane is close to an existing plane. We conduct additional GCCs for all the UCs in LPA and GPA. The UC will be tracked, but it will not be used for scan registration until it is verified as a correct correspondence.

We can efficiently solve the linear system (13). Specifically, for each new correspondence, we solve the normal equation system ${\mathbf{A}^{o+i}_k}^T\mathbf{A}^{o+i}_k\boldsymbol{x}^{o+i}_{k-1,k} = {\mathbf{A}^{o+i}_k}^T\mathbf{b}^{o+i}_k$. We find that ${\mathbf{A}^{o+i}_k}^T\mathbf{A}^{o+i}_k = {\mathbf{A}^o_k}^T\mathbf{A}^o_k + {\mathbf{A}^i_k}^T\mathbf{A}^i_k$ and ${\mathbf{A}^{o+i}_k}^T\mathbf{b}^{o+i}_k = {\mathbf{A}^o_k}^T\mathbf{b}^o_k + {\mathbf{A}^i_k}^T\mathbf{b}^i_k$. It is clear that ${\mathbf{A}^o_k}^T\mathbf{A}^o_k$ and ${\mathbf{A}^o_k}^T\mathbf{b}^o_k$, which dominate the computation, are shared among the new linear systems, and we already have them when we compute $\boldsymbol{x}_{k-1,k}$ in (11). Thus the augmented linear system (13) can be easily solved. Finally, after we find all the new correspondences, we can efficiently solve the final linear system that combines all the constraints, as we have already computed ${\mathbf{A}^o_k}^T\mathbf{A}^o_k$ and ${\mathbf{A}^o_k}^T\mathbf{b}^o_k$, and some ${\mathbf{A}^i_k}^T\mathbf{A}^i_k$ and ${\mathbf{A}^i_k}^T\mathbf{b}^i_k$ during GCC.

B. Cost Function of Local Plane Adjustment

The LPA optimizes the poses of the keyframes within the sliding window and all the parameters of the planes observed by these keyframes, as demonstrated in Fig. 5. We set the window size as N_w ($N_w = 8$ in our experiments). The latest N_w keyframes form a set $\mathbb{W}$. Assume the planes observed by the N_w keyframes form a set $\mathbb{O}$, and the keyframes out of the window which see the planes in $\mathbb{O}$ form a set $\mathbb{F}$. The poses of the keyframes in $\mathbb{F}$ are fixed. Denote the parameterization of the pose of the ith keyframe as $\boldsymbol{x}_i$. Suppose the jth plane $\boldsymbol{\pi}_j$ has the CP parameters $\boldsymbol{\eta}_j$, and the measurements of $\boldsymbol{\pi}_j$

at $\boldsymbol{x}_i$ are a set of $K_{i,j}$ points defined as $\mathbb{P}_{i,j} = \{\mathbf{p}_{i,j,k}\}_{k=1}^{K_{i,j}}$. Each $\mathbf{p}_{i,j,k} \in \mathbb{P}_{i,j}$ provides one constraint $\delta_{i,j,k}$ with the form defined in (3). In LPA, we minimize the following cost

$$\min_{\substack{\boldsymbol{x}_i,\boldsymbol{\eta}_j\\ i\in\mathbb{W}\\ j\in\mathbb{O}}} \underbrace{\sum_{i\in\mathbb{W}}\sum_{j\in\mathbb{O}_i}\underbrace{\sum_{k=1}^{K_{i,j}} \delta^2_{i,j,k}\left(\boldsymbol{x}_i,\boldsymbol{\pi}_j\right)}_{c_{i,j}(\boldsymbol{x}_i,\boldsymbol{\pi}_j)}}_{C_w} + \underbrace{\sum_{j\in\mathbb{O}}\sum_{m\in\mathbb{F}_j}\underbrace{\sum_{k=1}^{K_{m,j}}\delta^2_{m,j,k}\left(\boldsymbol{\pi}_j\right)}_{c_{m,j}(\boldsymbol{\pi}_j)}}_{C_f}, \quad (14)$$

where $\mathbb{O}_i$ is the set of planes observed by the ith keyframe $\boldsymbol{x}_i$, and $\mathbb{F}_j$ is the set of keyframes out of the sliding window which see the jth plane $\boldsymbol{\pi}_j$. Here $\delta_{m,j,k}(\boldsymbol{\pi}_j)$ only depends on $\boldsymbol{\pi}_j$, as the pose out of the window is fixed. Assume $\mathbf{J}_w$ and $\mathbf{J}_f$ are the Jacobian matrices of the residual $\boldsymbol{\delta}_w$ in C_w and the residual $\boldsymbol{\delta}_f$ in C_f, respectively. Then the Jacobian matrix $\mathbf{J}_l$ and the residual $\boldsymbol{\delta}_l$ of (14) have the form

$$\boldsymbol{\delta}_l = \left[\boldsymbol{\delta}_w^T, \boldsymbol{\delta}_f^T\right]^T \text{ and } \mathbf{J}_l = \left[\mathbf{J}_w^T, \mathbf{J}_f^T\right]^T. \quad (15)$$

Then we can compute the step vector of the LM algorithm by solving the linear system $\left(\mathbf{J}_l^T\mathbf{J}_l + \lambda\mathbf{I}\right)\boldsymbol{\xi} = -\mathbf{J}_l^T\boldsymbol{\delta}_l$. As planes generally have many observations, directly computing this linear system is time-consuming. The computation of $\boldsymbol{\delta}_w$ and $\mathbf{J}_w$ can be simplified by the algorithm introduced in [17]. Here we focus on $\boldsymbol{\delta}_f$ and $\mathbf{J}_f$.

C. Residual Vector $\boldsymbol{\delta}_f$

We first consider the residual vector $\boldsymbol{\delta}_f$ in C_f. Assume $\mathbf{q}_{m,j,k}$ is a point on $\boldsymbol{\pi}_j$ observed at pose $\mathbf{T}_m$. $\delta_{m,j,k}\left(\boldsymbol{\pi}_j\right)$ in (14) has the form

$$\delta_{m,j,k}\left(\boldsymbol{\pi}_j\right) = {\boldsymbol{\pi}_j}^T\mathbf{T}_m\bar{\mathbf{q}}_{m,j,k} = \bar{\mathbf{q}}_{m,j,k}^T{\mathbf{T}_m}^T\boldsymbol{\pi}_j. \quad (16)$$

Stacking all the $K_{m,j}$ residuals in (16), we obtain a residual vector

$$\boldsymbol{\delta}_{m,j}\left(\boldsymbol{\pi}_j\right) = \underbrace{\mathbf{Q}_{m,j}\mathbf{T}_m^T}_{\mathbf{P}_{m,j}}\boldsymbol{\pi}_j = \mathbf{P}_{m,j}\boldsymbol{\pi}_j, \quad (17)$$

where $\mathbf{Q}_{mj} = \left[\cdots,\ \bar{\mathbf{q}}_{mjk},\ \cdots\right]^T$.

Further stacking all $\boldsymbol{\delta}_{m,j}(\boldsymbol{\pi}_j)$ for $m \in \mathbb{F}_j$, we get the residual vector $\boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)$ for $\boldsymbol{\pi}_j$ in C_f:

$$\boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right) = \left[\ \cdots,\quad \mathbf{P}_{mj}^T,\quad \cdots\right]^T\boldsymbol{\pi}_j = \mathbf{P}_j\boldsymbol{\pi}_j, \quad (18)$$

We finally stack all $\boldsymbol{\delta}_j\left(\boldsymbol{\eta}_j\right)$ for $j \in \mathbb{O}$ to get the residual vector $\boldsymbol{\delta}_f$ in C_f in (14):

$$\boldsymbol{\delta}_f = \left[\ \cdots,\ \boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)^T,\ \cdots\ \right]^T. \quad (19)$$

D. Jacobian Matrix $\mathbf{J}_f$

To calculate $\mathbf{J}_f$, we first calculate the derivatives of $\delta_{m,j,k}\left(\boldsymbol{\pi}_j\right)$ in (16) with respect to $\boldsymbol{\eta}_j$. Here we define $\bar{\mathbf{p}}_{mjk} = \mathbf{T}_m\bar{\mathbf{q}}_{m,j,k}$, as $\mathbf{T}_m$ is fixed. Then (16) can be rewritten as

$$\delta_{m,j,k}\left(\boldsymbol{\pi}_j\right) = \bar{\mathbf{p}}_{m,j,k}^T\boldsymbol{\pi}_j. \quad (20)$$

Let us define $\boldsymbol{\eta}_j = \left[\eta_{j,1}, \eta_{j,2}, \eta_{j,3}\right]^T$. Then the derivatives of $\delta_{m,j,k}\left(\boldsymbol{\pi}_j\right)$ with respect to $\boldsymbol{\eta}_j$ have the form

$$\begin{aligned}\frac{\partial\delta_{m,j,k}(\boldsymbol{\pi}_j)}{\partial\boldsymbol{\eta}_j} &= \frac{\partial\delta_{m,j,k}(\boldsymbol{\pi}_j)}{\partial\boldsymbol{\pi}_j}\frac{\partial\boldsymbol{\pi}_j}{\partial\boldsymbol{\eta}_j}\\ &= \bar{\mathbf{p}}_{m,j,k}^T\underbrace{\left[\frac{\partial\boldsymbol{\pi}_j}{\partial\eta_{j,1}}, \frac{\partial\boldsymbol{\pi}_j}{\partial\eta_{j,2}}, \frac{\partial\boldsymbol{\pi}_j}{\partial\eta_{j,3}}\right]}_{\boldsymbol{V}_j} = \bar{\mathbf{p}}_{m,j,k}^T\boldsymbol{V}_j.\end{aligned} \quad (21)$$

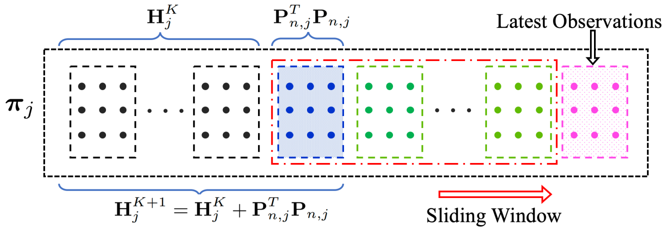
Then, using (18) and (21), we obtain the Jacobian matrix block of $\boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)$ in (19)

$$\mathbf{J}_j = \begin{bmatrix}\mathbf{0} & \cdots & \mathbf{P}_j\boldsymbol{V}_j & \cdots & \mathbf{0}\end{bmatrix}. \quad (22)$$

Stacking all $\mathbf{J}_j$ ($j \in \mathbb{O}$), we get the Jacobian matrix $\mathbf{J}_f$ of $\boldsymbol{\delta}_f$ in (19)

$$\mathbf{J}_f = \left[\ \cdots,\quad \mathbf{J}_j^T,\quad \cdots\right]^T. \quad (23)$$

As the number of points recorded by a LiDAR is large, $\boldsymbol{\delta}_f$ in (19) and $\mathbf{J}_f$ in (23) involve a large amount of computation. We introduce the reduced residual vector $\boldsymbol{\delta}_f^r$ and the reduced Jacobian matrix $\mathbf{J}_f^r$ to reduce the computational complexity.



$\mathbf{H}_j^K$
$\mathbf{P}_{n,j}^T\mathbf{P}_{n,j}$
Latest Observations
$\boldsymbol{\pi}_j$
$\mathbf{H}_j^{K+1} = \mathbf{H}_j^K + \mathbf{P}_{n,j}^T\mathbf{P}_{n,j}$
Sliding Window

Fig. 6. A schematic of updating ICM $\mathbf{H}_j$ of plane $\boldsymbol{\pi}_j$ incrementally. Assume the nth keyframe is about to be removed from the sliding window, and there are K sets of observations of $\boldsymbol{\pi}_j$ out of the sliding window. $\mathbf{P}_{n,j}$ is generated from the set of observations of $\boldsymbol{\pi}_j$ at the nth keyframe with the form defined in (17). Then we update $\mathbf{H}_j$ by $\mathbf{H}_j^{K+1} = \mathbf{H}_j^K + \mathbf{P}_{n,j}^T\mathbf{P}_{n,j}$.

E. Reduced Residual Vector and Jacobian Matrix

Using (18), the last two summation of C_f in (14) has the form

$$\begin{aligned}C_j &= \sum_{m\in\mathbb{F}_j} c_{m,j}(\boldsymbol{\pi}_j) = \boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)^T\boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)\\ &= \boldsymbol{\pi}_j^T\underbrace{\mathbf{P}_j^T\mathbf{P}_j}_{\mathbf{H}_j}\boldsymbol{\pi}_j = \boldsymbol{\pi}_j^T\mathbf{H}_j\boldsymbol{\pi}_j = \boldsymbol{\pi}_j^T\mathbf{L}_j\mathbf{L}_j^T\boldsymbol{\pi}_j,\end{aligned} \quad (24)$$

where $\mathbf{H}_j = \mathbf{L}_j\mathbf{L}_j^T$ is the Cholesky decomposition of the 4×4 matrix $\mathbf{H}_j$, and $\mathbf{L}_j$ is a 4×4 lower triangular matrix. As $\mathbf{H}_j$ is a 4×4 matrix, the runtime of the Cholesky decomposition is negligible. We call $\mathbf{H}_j$ the **integrated cost matrix** (**ICM**) for $\boldsymbol{\pi}_i$. Let us define reduced the residual vector $\boldsymbol{\delta}_f^r$ and reduced Jacobian matrix $\mathbf{J}_f^r$ as

$$\boldsymbol{\delta}_f^r = \left[\ \cdots\ \boldsymbol{\delta}_j^{rT}\ \cdots\right]^T \text{ and } \mathbf{J}_f^r = \left[\ \cdots\ \mathbf{J}_j^{rT}\ \cdots\right]^T, \quad (25)$$

where $\boldsymbol{\delta}_j^r = \mathbf{L}_j^T\boldsymbol{\pi}_j$ and $\mathbf{J}_j^r = \begin{bmatrix}\mathbf{0} & \cdots & \mathbf{L}_j^T\boldsymbol{V}_j & \cdots & \mathbf{0}\end{bmatrix}$. $\boldsymbol{\delta}_f^r$ and $\mathbf{J}_f^r$ include all the information required by the LM algorithm, and using these reduced forms significantly decreases the computational time. Specifically, we have the following two lemmas:

Lemma 1: $\mathbf{J}_f^T\mathbf{J}_f = \mathbf{J}_f^{rT}\mathbf{J}_f^r$ and $\mathbf{J}_f^T\boldsymbol{\delta}_f = \mathbf{J}_f^{rT}\boldsymbol{\delta}_f^r$.

Lemma 2: *Assume there are K_j residuals in $\boldsymbol{\delta}_j\left(\boldsymbol{\pi}_j\right)$ (defined in (18)). The runtime of computing $\mathbf{J}_j^r$, $\boldsymbol{\delta}_j^r$, $\mathbf{J}_j^{rT}\mathbf{J}_j^r$ and $\mathbf{J}_j^{rT}\boldsymbol{\delta}_j^r$ is $\frac{4}{K_j}$ of the runtime of computing the original $\mathbf{J}_j$, $\boldsymbol{\delta}_j$, $\mathbf{J}_j^T\mathbf{J}_j$ and $\mathbf{J}_j^T\boldsymbol{\delta}_j$.*

We prove Lemma 1 and 2 in the Appendix section.

F. Incremental Computation of $\mathbf{H}_j$

As we show above, $\mathbf{H}_j$ in (24) is crucial to simplify the computation. $\mathbf{H}_j$ summarizes the cost derived from the observations of plane $\boldsymbol{\pi}_j$ at keyframes out of the sliding widow. Due to the large number of observations of $\boldsymbol{\pi}_j$, it is time-consuming to compute $\mathbf{H}_j$ from scratch each time. Here we introduce an incremental method to avoid the redundant computation for $\mathbf{H}_j$, as demonstrated in Fig. 6.

Assume there are K keyframes out of the sliding window which have seen $\boldsymbol{\pi}_j$. This results in a corresponding $\mathbf{H}_j^K$ computed from (24). Suppose $\boldsymbol{\delta}_{n,j}\left(\boldsymbol{\pi}_j\right) = \mathbf{P}_{n,j}\boldsymbol{\pi}_j$ with the form defined in (17) is the residual vector derived from the observations of $\boldsymbol{\pi}_j$ at the nth keyframe $\boldsymbol{x}_n$ that is about to be removed from the sliding window, as demonstrated in Fig. 6. Using the definition of $\mathbf{H}_j$ in (24) and the definition of $\mathbf{P}_j$ in (18), we can compute $\mathbf{H}_j^{K+1}$ for the $K+1$ keyframes out of the sliding window as follows

$$\mathbf{H}_j^{K+1} = \mathbf{P}_j^T\mathbf{P}_j = \sum_{m\neq n}\mathbf{P}_{m,j}^T\mathbf{P}_{m,j} + \mathbf{P}_{n,j}^T\mathbf{P}_{n,j} = \mathbf{H}_j^K + \mathbf{P}_{n,j}^T\mathbf{P}_{n,j}. \quad (26)$$

Each global plane has an ICM. Once a keyframe is removed from the sliding window, we update the ICMs of the planes observed by this keyframe through (26).

G. Local Plane Adjustment

Before we perform the optimization, we remove the oldest keyframe if the sliding window is full. Now we consider minimizing (14). As (14) is generally a large-scale optimization problem, we first

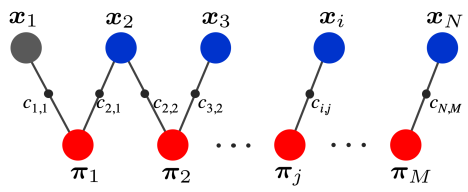


Fig. 7. Factor graph for a GPA problem with N poses and M planes. $\boldsymbol{x}_1$ is fixed during the optimization.

simplify its residual vector $\boldsymbol{\delta}_l$ and the corresponding Jacobian matrix $\mathbf{J}_l$ in (15). In [17], we proved that $\boldsymbol{\delta}_w$ and $\mathbf{J}_w$ in (15) can be replaced by a reduced residual vector $\boldsymbol{\delta}_w^r$ and a reduced Jacobian $\mathbf{J}_w^r$ in the LM algorithm. Let us define

$$\mathbf{J}_l^r = \left[\mathbf{J}_w^{r\,T}, \mathbf{J}_f^{r\,T} \right]^T \text{ and } \boldsymbol{\delta}_l^r = \left[\boldsymbol{\delta}_w^{r\,T}, \boldsymbol{\delta}_f^{r\,T} \right]^T. \tag{27}$$

Then we have the following theorem for $\mathbf{J}_l^r$ and $\boldsymbol{\delta}_l^r$:

Theorem 1: **$\mathbf{J}_l^r$ and $\boldsymbol{\delta}_l^r$ can replace $\mathbf{J}_l$ and $\boldsymbol{\delta}_l$ in the LM algorithm.**

As $\mathbf{J}_l^r$ and $\boldsymbol{\delta}_l^r$ have a much lower dimension than $\mathbf{J}_l$ and $\boldsymbol{\delta}_l$, they can significantly reduce the computational complexity. The proof of Theorem 1 can be found in the Appendix section.

During the optimization, if there exist UCs in the new keyframe, we test them. If the mean point-to-plane distance $\bar{d}_L$ of a UC is smaller than γ (γ is defined in section VI-A), we treat it as a correct correspondence. If $\bar{d}_L < 2\gamma$, we still treat it as a UC, otherwise we remove it. After LPA, if there does not exist a UC, we correct the drift of the localization component. Let $\mathbf{T}_n^s$ denote the pose of the latest keyframe. Using (2), we know its correspond tracking pose is $\mathbf{T}_{n-1}^e$. We calculate $\Delta\mathbf{T} = \mathbf{T}_n^s(\mathbf{T}_{n-1}^e)^{-1}$. Then we apply $\Delta\mathbf{T}$ to the latest tracking pose to correct its drift. If new correspondences or UCs are retained, we consider previous planes are revisited. The GPA introduced bellow is then triggered.

VII. Global Mapping

When global planes are revisited as introduced in section VI-A, GPA is triggered after LPA. GPA jointly optimizes keyframe poses and plane parameters by minimizing the point-to-plane distances.

Assume that there are M planes and N keyframes. The factor graph of GPA is demonstrated in Fig. 7. Using the notations in section VI-B, we can write the cost function of GPA as

$$\min_{\substack{\boldsymbol{x}_i, \boldsymbol{\eta}_j \\ i \neq 1}} \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j}\left(\boldsymbol{x}_i, \boldsymbol{\pi}_j\right). \tag{28}$$

We adopt the method introduced in [17] to efficiently solve (28). During the optimization, we test the remaining UCs. If the mean point-to-plane distance $\bar{d}_G$ of a UC is smaller than γ (γ is defined in section VI-A), we treat it as a correct correspondence, otherwise we remove it. After GPA, the drift of the tracking pose is corrected as done after LPA.

Update ICM After we refine the poses, we need to update the ICM $\mathbf{H}_j$ of each plane. This can be done efficiently. As shown in (26), the important elements for computing $\mathbf{H}_j$ are the summands $\mathbf{P}_{n,j}^T\mathbf{P}_{n,j}$. Using the form of $\mathbf{P}_{n,j}$ defined in (17), we have $\mathbf{P}_{n,j}^T\mathbf{P}_{n,j} = \mathbf{T}_n\mathbf{Q}_{n,j}^T\mathbf{Q}_{n,j}\mathbf{T}_n^T$. As $\mathbf{Q}_{n,j}^T\mathbf{Q}_{n,j}$ is fixed, we only need to compute it once. After GPA, we use the refined pose $\mathbf{T}_n$ to update $\mathbf{P}_{n,j}^T\mathbf{P}_{n,j}$, and then we sum them up to get the new $\mathbf{H}_j$. Since we only need to compute the most time-consuming part $\mathbf{Q}_{nj}^T\mathbf{Q}_{nj}$ once and afterwards we can reuse it, updating $\mathbf{H}_j$ is fast.

VIII. Experimental Results

A. Dataset

We used a NavVis M6 device to collect four indoor datasets with many large rotation motions, as shown in Fig. 8. The trajectory estimated by the NavVis system is treated as the ground truth. The NavVis device has a Velodyne VLP-16 LiDAR and 3 Hokuyo single-layer LiDARs. It estimates the pose through the LiDARs mentioned above and an IMU. In addition, it can also use the wifi signal and manually set ground control points to trigger loop closure to improve the accuracy of the trajectory. The four trajectories were generated by fusing all the information through hours of offline computation. We only use the data collected by the Velodyne VLP-16 LiDAR, and adopt the **absolute trajectory error** (**ATE**) to evaluate the performance of different algorithms.

TABLE I
The keyframe ATEs of LeGO-LOAM [8], BALM [29] and variants of our algorithm on four indoor datasets. LeGO-LOAM [8] fails to complete the sequences A, C and D, and BALM [8] fails to finish the sequence C, as shown in Fig. 10.

	A	B	C	D
Length (m)	261.9	294.0	391.7	139.7
LeGO-LOAM [8] (m)	-	1.33	-	-
BALM [29] (m)	0.34	0.21	-	0.23
π-LSAM [22] (m)	0.082	0.16	0.13	0.11
Ours - LPA - GPA (m)	0.29	0.25	0.24	0.46
Ours - GPA (m)	0.18	0.14	0.11	0.16
Ours (m)	**0.039**	**0.042**	**0.033**	**0.048**

B. Results

In this section, we provide the experimental results. We ran the experiments on a computer with a 3.1 GHz Intel i7 CPU and 16G memory. We first explore the impact of different design selections of our algorithm. Then we compare our algorithm with the state-of-the-art LiDAR SLAM algorithms, LeGO-LOAM [8] and BALM [29]. Finally, we provide the runtime of our algorithm. In the following experiments, we run all the compared algorithms 5 times on each dataset, and report the median of the resulting ATEs.

Ablation Study We study two variants of our algorithm:

- Ours - GPA - LPA: Our algorithm without GPA and LPA.
- Ours - GPA: Our algorithm without GPA.

Table I lists the ATEs of the above variants of our algorithm, and Fig. 9 shows the trajectories generated by them on dataset A and D. It is clear that LPA and GPA can significantly improve the accuracy of trajectory. Dataset D contains a loop. In previous work, the loop closure will be triggered at the end of the trajectory. In our algorithm, the loop closure can be triggered much earlier as shown in Fig. 2. Comparing the trajectories from our algorithm and the one without GPA in Fig. 9, we find that our loop closure strategy can correct the drift much earlier instead of correcting it at the end. Fig. 8 shows the maps generated by our algorithm. Our algorithm can distinguish and reconstruct the two sides of planar objects. This results in more accurate data association compared to the nearest-neighbor method.

Benchmarking We compare our algorithm with LeGO-LOAM [8] and BALM [29]. Table I lists the results of the compared algorithms. Our algorithm, even the variant without GPA and LPA, significantly outperforms LeGO-LOAM [8]. BALM adopts planes and lines as the landmarks. It dose not have the loop closure. We find that our algorithm without GPA (Ours - GPA) surpasses BALM. This is probably because BALM adopts the nearest-neighbor method to get the data association that may results in false data associations due to the double-side issue of the planar objects. As shown in Fig. 11, the two sides of the walls reconstructed by BALM are mixed together. Besides, BALM estimates the plane parameters from a small set of points, instead the plane parameters in our algorithm are estimated from a much larger set of points, which may result in more accurate plane parameters. Fig. 10 shows the trajectories of the compared algorithms. LeGO-LOAM and BALM are not stable for the large rotation motion. LeGO-LOAM fails to finish the sequences A, C and D, and BALM fails on the sequence C. Furthermore, the odometry trajectory of LeGO-LOAM that is estimated from scan-to-scan registration has obvious jitters, and its trajectory from scan-to-map registration is much smoother but it has a lower frequency. Our algorithm and BALM perform global registration for each scan. Thus the resulting odometry trajectory is smooth and accurate.

Runtime We use the dataset B and C that have the longest two trajectories to evaluate the runtime of different components of our algorithm. Table II shows the results. The average runtime

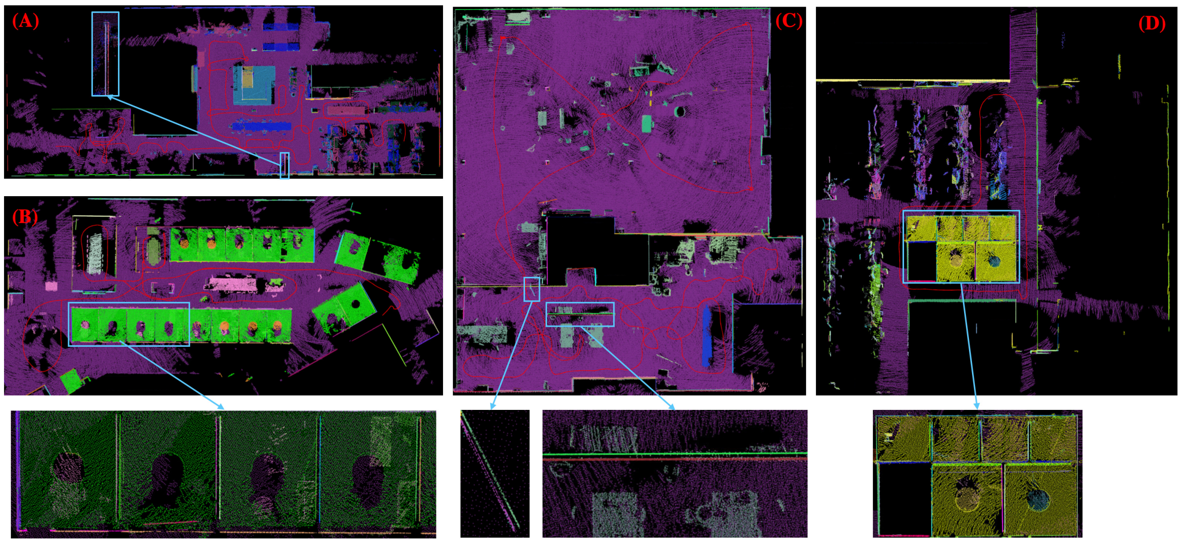


Fig. 8. Planes and trajectories generated by our algorithm for the four datasets. Our algorithm can identify and recover the double sides of planar objects (such as walls and doors), which can lead to more accurate data association compared to the nearest-neighbor search.

TABLE II
RUNTIME (MS) OF DIFFERENT COMPONENTS OF OUR ALGORITHM.

Dataset	Localization			Local Mapping				Global Mapping	
	Forward ICP Flow	Pose Estimation	Keyframe Decision	Detect Planes	Match Planes	GCC	LPA	GPA	Update ICM
B	25.6 ± 5.1	16.5 ± 4.2	0.076 ± 0.031	10.1 ± 2.9	5.7 ± 3.1	0.65 ± 0.36	8.6 ± 3.5	395.6 ± 301.9	1.5 ± 1.3
C	28.2 ± 5.8	17.3 ± 4.8	0.074 ± 0.029	9.8 ± 2.6	6.0 ± 3.2	0.67 ± 0.35	9.1 ± 2.9	420.2 ± 330.2	1.7 ± 1.6

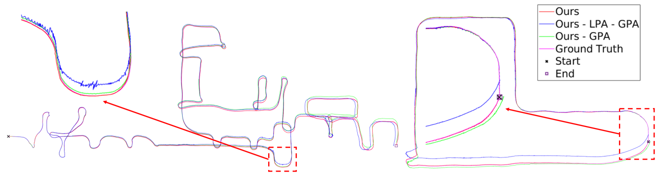


Fig. 9. The results of our algorithm and its variants on dataset A and D. The benefit of our loop closure strategy is clear by comparing the trajectories from our algorithm and “Ours - GPA” on dataset D (the 2nd image). Our method can correct the drift much earlier (as shown in the 2nd image of Fig. 2) instead of correcting it at the end where the traditional loop closure occurs.

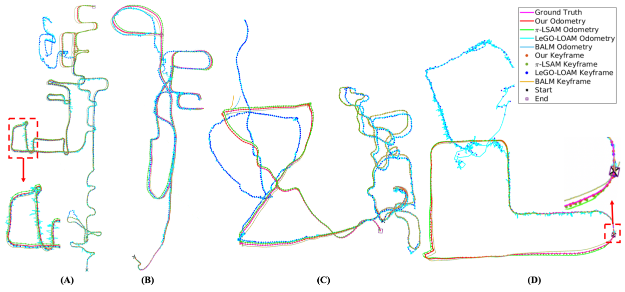


Fig. 10. The trajectories of our algorithm, LeGO-LOAM [8] and BALM [29]. The odometry trajectories of LeGO-LOAM have obvious jitters as shown in the rectangle of dataset A. BALM smooths each pose, so its keyframe and odometry trajectories have the same number of poses. For dataset D, the odometry and keyframe trajectories of π-LSAM [22] are different. This is because its loop closure is trigger at the end of the trajectory. Our method can conduct the loop closure much earlier to correct the drift as shown in the 2nd image of Fig. 2, which benefits the online applications, such as motion planning and control.

of the localization component is about $42.1ms$ and $45.5ms$ on dataset B and C, respectively. As one scan of a VLP-16 LiDAR takes 100 ms, our algorithm achieves real time. On the other hand, the localization component of π-LSAM [22] averagely takes about $62.3ms$ and $60.7ms$ on dataset B and C, respectively. Thus our algorithm is faster than π-LSAM. We also test the runtime of the plane extraction algorithm introduced in section V-A. The runtime of this algorithm is about $71.3 \pm 5.6ms$ and $73.2 \pm 6.1ms$ on dataset B and C, respectively. Thus, our plane tracking strategy is about 3 times faster. The local mapping can be finished around $25.1 \pm 5.8ms$ and $25.6 \pm 6.3ms$ for dataset B and C, respectively. Thus it can quickly update the map and the trajectory to reduce the drift. We also test the runtime for directly minimizing (14). Our algorithm can speed up LPA 32 and 37 times on dataset B and C, respectively. The standard deviation of GPA is large. This is because GPA is triggered at obviously different places.

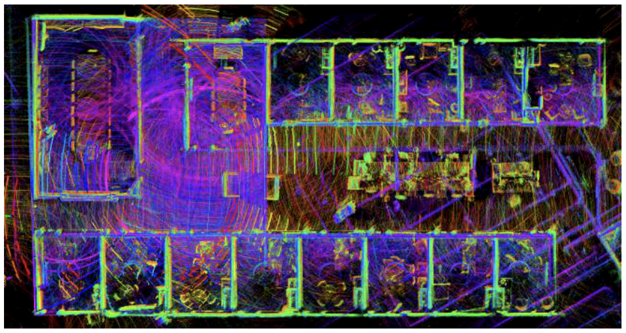


Fig. 11. The point cloud of BALM [29] on dataset B. Compared to our results in Fig. 8, it is clear that BALM mixes the double sides of the walls.

IX. CONCLUSION

In this paper, we present a novel LiDAR SLAM algorithm using planes as the landmark for the indoor environment. Our algorithm consists of localization, local mapping and global mapping. The localization component provides real-time and globally consistent poses, rather than the lower fidelity local registration pose in the LOAM framework [7]. To achieve this, we introduce the forward ICP flow to maintain the correspondences between local planes and global planes scan by scan, which avoids getting this information from scratch through the nearest-neighbor search in previous works. We show that the large-scale least-squares problem from the point-to-plane cost can be significantly simplified. We linearize the rotation matrix to simplify the LiDAR scan undistortion and pose estimation. We introduce the reduced residual vector and Jacobian matrix to speed up the PA. As planes can be revisited at a remote place, we propose to trigger the global mapping when existing planes are revisited, rather than a place is revisited. The experimental results show that our algorithm outperforms the state-of-the-art LiDAR SLAM methods [8], [29] and achieves real-time performance.

Appendix

A. Proof of Lemma 1

Proof. We first prove $\mathbf{J}_f^T\mathbf{J}_f = \mathbf{J}_f^{r\,T}\mathbf{J}_f^r$. $\mathbf{J}_f$ in (23) and $\mathbf{J}_f^r$ in (25) are block vectors. Using the block matrix multiplication rule, we have

$$\mathbf{J}_f^T\mathbf{J}_f = \sum_{j\in\mathbb{O}} \mathbf{J}_j^T\mathbf{J}_j, \text{ and } \mathbf{J}_f^{r\,T}\mathbf{J}_f^r = \sum_{j\in\mathbb{O}} \mathbf{J}_j^{r\,T}\mathbf{J}_j^r. \tag{29}$$

Using the definition of $\mathbf{J}_j$ in (22), we know that $\mathbf{J}_j^T\mathbf{J}_j$ only has one non-zero term $\mathbf{V}_j^T\mathbf{P}_j^T\mathbf{P}_j\mathbf{V}_j$. According to the definition of $\mathbf{H}_j$ in (24), we have $\mathbf{V}_j^T\mathbf{P}_j^T\mathbf{P}_j\mathbf{V}_j = \mathbf{V}_j^T\mathbf{H}_j\mathbf{V}_j$ Similarly, according to the definition of $\mathbf{J}_j^r$ in (25), $\mathbf{J}_j^{r\,T}\mathbf{J}_j^r$ also only has one non-zero term $\mathbf{V}_j^T\mathbf{L}_j\mathbf{L}_j^T\mathbf{V}_j$. Using (24), we have $\mathbf{V}_j^T\mathbf{L}_j\mathbf{L}_j^T\mathbf{V}_j = \mathbf{V}_j^T\mathbf{H}_j\mathbf{V}_j$. According to the above discussion, we know $\mathbf{J}_j^T\mathbf{J}_j = \mathbf{J}_j^{r\,T}\mathbf{J}_j^r$. As the summands of $\mathbf{J}_f^T\mathbf{J}$ and $\mathbf{J}_f^{r\,T}\mathbf{J}_f^r$ in (29) are equal, we have $\mathbf{J}_f^T\mathbf{J}_f = \mathbf{J}_f^{r\,T}\mathbf{J}_f^r$.

Then we prove $\mathbf{J}_f^T\boldsymbol{\delta}_f = \mathbf{J}_f^{r\,T}\boldsymbol{\delta}_f^r$. $\boldsymbol{\delta}_f$ in (19), $\mathbf{J}_f$ in (23), $\boldsymbol{\delta}_f^r$ and $\mathbf{J}_f^r$ in (25) are all block vectors. Using the block matrix multiplication rule, we have

$$\mathbf{J}_f^T\boldsymbol{\delta}_f = \sum_{j\in\mathbb{O}} \mathbf{J}_j^T\boldsymbol{\delta}_j, \text{and } \mathbf{J}_f^{r\,T}\boldsymbol{\delta}_f^r = \sum_{j\in\mathbb{O}} \mathbf{J}_j^{r\,T}\boldsymbol{\delta}_j^r. \tag{30}$$

According to the definition of $\mathbf{J}_j$ in (22) and the definition of $\boldsymbol{\delta}_j$ in (18), $\mathbf{J}_j{}^T\boldsymbol{\delta}_j$ only has one non-zero term $\mathbf{V}_j^T\mathbf{P}_j^T\mathbf{P}_j\boldsymbol{\pi}_j$. Using the definition of $\mathbf{H}_j$ in (24), we have $\mathbf{V}_j^T\mathbf{P}_j^T\mathbf{P}_j\boldsymbol{\pi}_j = \mathbf{V}_j^T\mathbf{H}_j\boldsymbol{\pi}_j$. Similarly, according to the definition of $\mathbf{J}_j^r$ and $\boldsymbol{\delta}_j^r$ in (25), the only non-zero term of $\mathbf{J}_j^{r\,T}\boldsymbol{\delta}_j^r$ is $\mathbf{V}_j^T\mathbf{L}_j\mathbf{L}_j^T\boldsymbol{\pi}_j = \mathbf{V}_j^T\mathbf{H}_j\boldsymbol{\pi}_j$. Thus using (30), we have $\mathbf{J}_f^T\boldsymbol{\delta}_f = \mathbf{J}_f^{r\,T}\boldsymbol{\delta}_f^r$ □

B. Proof of Lemma 2

Proof. Comparing $\mathbf{J}_j^r$ and $\boldsymbol{\delta}_j^r$ in (25) with $\mathbf{J}_j$ in (22) and $\boldsymbol{\delta}_j$ in (18), and according to the forms of $\mathbf{J}_j^{r\,T}\mathbf{J}_j^r$, $\mathbf{J}_j^{r\,T}\boldsymbol{\delta}_j^r$, $\mathbf{J}_j{}^T\mathbf{J}_j$ and $\mathbf{J}_j{}^T\boldsymbol{\delta}_j$ in the proof of Lemma 1, we know that the difference between the two sets of entities is that we use $\mathbf{L}_j$ to replace $\mathbf{P}_j$. $\mathbf{L}_j$ has 4 rows, and $\mathbf{P}_j$ has K_j rows. Thus, the runtime for calculating the reduced entities is $\frac{4}{K_j}$ relative to calculating the original ones. □

C. Proof of Theorem 1

Proof. To prove this theorem, we only need to verify $\mathbf{J}_l^T\mathbf{J}_l = \mathbf{J}_l^{r\,T}\mathbf{J}_l^r$ and $\mathbf{J}_l^T\boldsymbol{\delta}_l = \mathbf{J}_l^{r\,T}\boldsymbol{\delta}_l^r$. Using the definition of $\mathbf{J}_l$ and $\boldsymbol{\delta}_l$ in (15) and the definition of $\mathbf{J}_l^r$ and $\boldsymbol{\delta}_l^r$ in (27), we have

$$\begin{aligned}\mathbf{J}_l^T\mathbf{J}_l = \mathbf{J}_w^T\mathbf{J}_w + \mathbf{J}_f^T\mathbf{J}_f,\ \mathbf{J}_l^{r\,T}\mathbf{J}_l^r = \mathbf{J}_w^{r\,T}\mathbf{J}_w^r + \mathbf{J}_f^{r\,T}\mathbf{J}_f^r,\\ \mathbf{J}_l^T\boldsymbol{\delta}_l = \mathbf{J}_w^T\boldsymbol{\delta}_w + \mathbf{J}_f^T\boldsymbol{\delta}_f,\ \mathbf{J}_l^{r\,T}\boldsymbol{\delta}_l^r = \mathbf{J}_w^{r\,T}\boldsymbol{\delta}_w^r + \mathbf{J}_f^{r\,T}\boldsymbol{\delta}_f^r.\end{aligned} \tag{31}$$

According to [17] we know $\mathbf{J}_w^T\mathbf{J}_w = \mathbf{J}_w^{r\,T}\mathbf{J}_w^r$, and according to Lemma 1 we have $\mathbf{J}_f^T\mathbf{J}_f = \mathbf{J}_f^{r\,T}\mathbf{J}_f^r$. Thus we know the summands of $\mathbf{J}_l^T\mathbf{J}_l$ and $\mathbf{J}_l^{r\,T}\mathbf{J}_l^r$ in (31) are equal. Finally, we have $\mathbf{J}_l^T\mathbf{J}_l = \mathbf{J}_l^{r\,T}\mathbf{J}_l^r$. Similarly, as $\mathbf{J}_w^T\boldsymbol{\delta}_w = \mathbf{J}_w^{r\,T}\boldsymbol{\delta}_w^r$ and $\mathbf{J}_f^T\boldsymbol{\delta}_f = \mathbf{J}_f^{r\,T}\boldsymbol{\delta}_f^r$, we obtain $\mathbf{J}_l^T\boldsymbol{\delta}_l = \mathbf{J}_l^{r\,T}\boldsymbol{\delta}_l^r$. □

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.

[2] J. Behley and C. Stachniss, "Efficient surfel-based SLAM using 3D laser range data in urban environments." in *Robotics: Science and Systems*, vol. 2018, 2018.

[3] D. Droeschel and S. Behnke, "Efficient continuous-time SLAM for 3D lidar-based online mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5000–5007.

[4] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP." in *Robotics: science and systems*, vol. 2, no. 4. Seattle, WA, 2009, p. 435.

[5] F. Pomerleau, F. Colas, and R. Siegwart, *A Review of Point Cloud Registration Algorithms for Mobile Robotics*, 2015.

[6] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[7] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," *Autonomous Robots*, vol. 41, no. 2, pp. 401–416, 2017.

[8] 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)*. IEEE, 2018, pp. 4758–4765.

[9] J. Lin and F. Zhang, "LOAM Livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3126–3131.

[10] D. Rozenberszki and A. L. Majdik, "LOL: Lidar-only odometry and localization in 3D point cloud maps," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020, pp. 4379–4385.

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[12] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[13] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga, "Fast Registration Based on Noisy Planes with Unknown Correspondences for 3-D Mapping," *IEEE Trans. on Robotics*, vol. 26, no. 3, pp. 424–441, 2010.

[14] M. Kaess, "Simultaneous localization and mapping with infinite planes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4605–4611.

[15] M. Hsiao, E. Westman, G. Zhang, and M. Kaess, "Keyframe-based dense planar SLAM," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5110–5117.

[16] P. Geneva, K. Eckenhoff, Y. Yang, and G. Huang, "LIPS: Lidar-Inertial 3D plane SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 123–130.

[17] L. Zhou, D. Koppel, H. Ju, F. Steinbruecker, and M. Kaess, "An Efficient Planar Bundle Adjustment Algorithm," in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2020, pp. 136–145.

[18] R. Dubé, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, "Segmap: Segment-based mapping and localization using data-driven descriptors," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020.

[19] H. Wang, C. Wang, and L. Xie, "Intensity scan context: Coding intensity and geometry relations for loop closure detection," in *2020 International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[20] J. Jiang, J. Wang, P. Wang, P. Bao, and Z. Chen, "LiPMatch: LiDAR Point Cloud Plane Based Loop-Closure," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6861–6868, 2020.

[21] Y. Wang, Z. Sun, C.-Z. Xu, S. Sarma, J. Yang, and H. Kong, "LiDAR iris for loop-closure detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5769–5775.

[22] L. Zhou, S. Wang, and M. Kaess, "π-LSAM: LiDAR smoothing and mapping with planes," in *Proc. IEEE Intl. Conf. on Robotics and Automation, ICRA*, Xi'an, China, May 2021, to appear.

[23] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[24] C. Park, P. Moghadam, S. Kim, A. Elfes, C. Fookes, and S. Sridharan, "Elastic LiDAR fusion: Dense Map-Centric Continuous-Time SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1206–1213.

[25] J.-E. Deschaud, "IMLS-SLAM: scan-to-model matching based on 3D data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2480–2485.

[26] Y. Yang and G. Huang, "Observability analysis of aided ins with heterogeneous features of points, lines, and planes," *IEEE Transactions on Robotics*, vol. 35, no. 6, pp. 1399–1418, 2019.

[27] G. Ferrer, "Eigen-factors: Plane estimation for multi-frame and time-continuous point cloud alignment," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 1278–1284.

[28] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[29] Z. Liu and F. Zhang, "BALM: Bundle Adjustment for Lidar Mapping," *Robotics and Automation Letters*, vol. 6, no. 2, pp. 3184–3191, 2021.

[30] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak, "Fast plane detection and polygonalization in noisy 3D range images," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2008, pp. 3378–3383.

[31] L. Zhou, S. Wang, and M. Kaess, "A fast and accurate solution for pose estimation from 3D correspondences," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1308–1314.