π-LSAM: LiDAR Smoothing and Mapping With Planes

Lipu Zhou[1], Shengze Wang[2] and Michael Kaess[3]

***Abstract*— This paper introduces a real-time dense planar LiDAR SLAM system, named π-LSAM, for the indoor environment. The widely used LiDAR odometry and mapping (LOAM) framework [1] does not include bundle adjustment (BA) and generates a low fidelity tracking pose. This paper seeks to overcome these drawbacks for the indoor environment. Specifically, we use the plane as the landmark, and introduce plane adjustment (PA) as our back-end to jointly optimize planes and keyframe poses. We present the π-factor to significantly reduce the computational complexity of PA. In addition, we introduce an efficient loop detection algorithm based on the RANSAC framework using planes. In the front-end, our algorithm performs global registration in real time. To achieve this performance, we maintain the local-to-global point-to-plane correspondences scan by scan, so that we only need a small local KD-tree to establish the data association between a LiDAR scan and the global planes, rather than a large global KD-tree used in previous works. With this local-to-global data association, our algorithm directly identifies planes in a LiDAR scan, and yields an accurate and globally consistent pose. Experimental results show that our algorithm significantly outperforms the state-of-the-art LOAM variant, LeGO-LOAM [2], and our algorithm achieves real time.**

I. Introduction

Simultaneous localization and mapping (SLAM) is a fundamental problem in the robotics community. Thus, SLAM using various sensors has been extensively studied. LiDAR SLAM has many robotic applications, such as autonomous navigation and motion planning. This paper focuses on LiDAR SLAM in the indoor environment.

Nowadays, it is known that bundle adjustment (BA) is important for a SLAM system. However, many LiDAR SLAM systems do not include BA. Instead, the iterative closest point (ICP) [3] framework and its variants [4], [5], [6], are generally adopted in LiDAR SLAM systems. One of the most successful LiDAR SLAM frameworks is the LiDAR odometry and mapping (LOAM) framework [1], which is based on two parallel registration algorithms. In the LOAM framework, the map is a point cloud that is generated by registering LiDAR scans incrementally. As the lack of joint optimization, the pose estimation error lowers the quality of the global point cloud, which in turn reduces the accuracy of the pose estimation. This forms a vicious loop. In addition, the LOAM framework provides a low fidelity tracking pose which is calculated from local scan-to-scan registration. This paper seeks to overcome the above two problems.

Motivated by [7], we use the plane as the landmark. We present a new LiDAR SLAM framework, named π-LiSAM. The main contributions of this paper are as follows:

a) We introduce plane adjustment (PA), which jointly optimizes keyframe poses and plane parameters to minimize the point-to-plane distance, in the back-end.

b) We present the π-factor to significantly reduce the computational complexity of PA.

c) We present an efficient method to align two LiDAR scans using planes for loop closure detection.

d) We introduce an ICP process to enable global registration in real time. Unlike the traditional ICP that only considers the data association between two point clouds, our ICP process maintains the data association scan by scan, as demonstrated in Fig. 3. This allows us to directly identify planes in the LiDAR scan without having to explicitly conduct the time-consuming plane detection in each scan. In addition, the ICP process allows us to get the local-to-global data association from a small local KD-tree, rather than a large global KD-tree in previous works [1], [2], [8].

e) The map in our algorithm is represented as plane parameters, which is easy to store and update. In contrast, the map in the ICP-based methods, such as LOAM [1], is generally organized as KD-trees. Thus, updating the map requires rebuilding the KD-trees. It would be costly for a large-scale scene.

Our experimental results show that our algorithm achieves real time and significantly outperforms the state-of-the-art LOAM variant, LeGO-LOAM [2].

II. Related Work

As a LiDAR provides 3D measurements directly, the ICP framework [3] seemingly provides a straightforward way to align the point clouds. However, since a LiDAR can only provide a sparse point cloud, it is infeasible to get the exact point-to-point correspondences. Deschaud [9] presents a scan-to-model matching framework which uses the implicit moving least squares surface representation. It provides accurate pose estimation, but it does not achieve real time. On the other hand, ICP variants, such as using planes [4], [5], [6], are generally employed in the LiDAR SLAM system. LOAM [1] provides a real-time and low-drift solution based on two parallel registration algorithms using planes and lines. The first algorithm provides real-time but low fidelity poses using the scan-to-scan local registration. The second algorithm carries out the scan-to-map global registration. It provides high accurate poses with

*This work was mostly done when the first two authors were with the Robot Perception Lab at Carnegie Mellon University.

[1] Lipu Zhou is with Magic Leap, 1376 Bordeaux Dr, Sunnyvale, CA 94089, USA lzhou@magicleap.com

[2] Shengze Wang is with Department of Computer Science, University of North Carolina, 3175 Brooks, Chapel Hill, NC 27599, USA shengzew@cs.unc.edu

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

low frequency. The pose from the second algorithm is then used to correct the drift of the first one. There are two main disadvantages in the LOAM framework:

- The real-time pose from the first algorithm is of low quality [10]. This may cause errors in the motion planning and control tasks of an autonomous robot.
- There lacks BA in the LOAM framework.

Some works have been presented to improve LOAM. LeGO-LOAM [2] improves the feature extraction in LOAM, and leverages the ground plane to facilitate the segmentation and optimization steps. LOAM is originally designed for spinning LiDARs. LOAM_livox [8] extends the LOAM framework to the solid state LiDAR with small FoV. It also adopts the parallel computing to achieve real-time global registration. Parallel computing needs a powerful CPU, it may be not suitable for an embedded system which has limited computational resources. We present a more efficient way to solve this problem. LOL [11] combines LOAM and SegMap [12] which is used for loop closure detection. This algorithm requires a GPU.

The map of the ICP based method is generally a 3D point cloud organized as KD-trees. An alternative way is to organize the map as features. Extracting feature from a point cloud has been extensively studied [13], [14]. Unlike visual SLAM, where points are generally used, planes are generally used in SLAM with depth sensors. In [15], they present a surfel-based LiDAR SALM algorithm that requires a GPU. Pathak *et al.* [16] use planes for pose registration. Planes can also be used in the EKF framework [17], [18], [19]. These algorithms do not include BA. As BA is an important part of a SLAM system, planes are introduced into BA for the RGBD sensor [7], [20]. The cost function is critical for an optimization problem [21]. These works employ the plane-to-plane distance, which evaluates the difference between two plane parameters, to construct the cost function. As the recent work [22] shows that the point-to-plane distance is more robust to initial errors and converges faster than the plane-to-plane distance, this paper adopts the point-to-plane distance to construct the cost function. Since each plane can generate many observations at a pose, the point-to-plane distance generally results in a large-scale least-squares problem. Ferrer [23] introduces the Eigen-Factor to estimate a planar surface by minimizing the point-to-plane distance. The author derives the closed-form of the gradient, and adopts a gradient-based optimization method which is inefficient. The method introduced in [22] can be used in the Levenberg-Marquardt (LM) algorithm [24]. But it requires a QR decomposition to compress the multiple point-to-plane constraints of a plane into four constraints. This paper introduces the π-factor to simply integrate the multiple point-to-plane constraints of a plane into one constraint.

Loop closure is important for a SLAM system. Loop detection is generally associated with place recognition [25], [26], [27], [12], [28]. Specifically, the sensor data is encoded into a descriptor. If the current descriptor is sufficiently close to a descriptor in the database, it is supposed that a loop closure has occurred. Manually designing a descriptor for the LiDAR point cloud is challenging. Although deep learning based descriptors [12], [28] show breakthrough results, they generally require a GPU. In [15], they introduce a map-based loop closure detection method which also needs a GPU. This paper introduces an efficient loop closure detection method. As the drift of our system is low, we trigger the loop detection if the robot approaches a previously visited place. We align planes in the current scan and the target scan using the RANSAC method [29]. As the number of planes in the environment is generally small, this method is efficient.

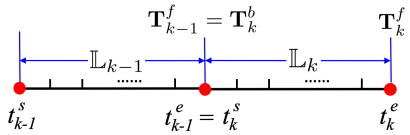


Fig. 1. A keyframe $\mathbb{L}_k$ recorded at $(t^s_k, t^e_k]$ has two kinds of pose, *i.e.*, $\mathbf{T}^f_k$ at time t^e_k generated by the front-end, and $\mathbf{T}^b_k$ at time t^s_k used in the back-end. This is because the measurements within a LiDAR scan are not recorded at the same time. Our front-end transforms the points in $\mathbb{L}_k$ to the coordinate system at t^s_k. We have $\mathbf{T}^f_{k-1} = \mathbf{T}^b_k$. This relationship is important for drift correction introduced in section V-C.

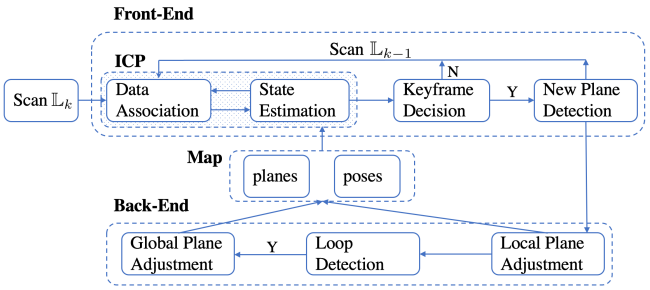


Fig. 2. π-LASM system overview. Our front-end generates globally consistent poses in real time. Our back-end maintains the map and corrects the drift of the front-end.

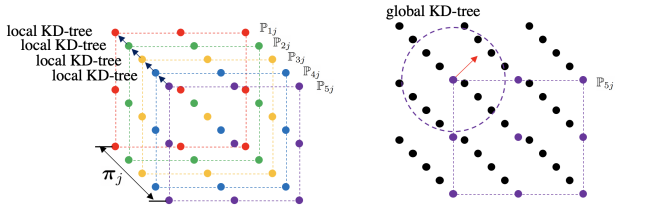
III. Notations and System Overview

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

LiDAR Pose In this work, we represent a pose by $\mathbf{T} \in SE(3)$ which consists of $\mathbf{R} \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$. We use the angle-axis representation $\boldsymbol{\omega}$ to parameterize $\mathbf{R}$. Then we can parameterize $\mathbf{T}$ as a vector $\boldsymbol{x} = [\boldsymbol{\omega}; \boldsymbol{t}] \in \mathbb{R}^6$. Let us define the skew matrix of $\boldsymbol{\omega}$ as $[\boldsymbol{\omega}]_\times \in \mathfrak{so}(3)$. Then the exponential map $exp : \mathfrak{so}(3) \rightarrow SO(3)$ has the form

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

The points within a LiDAR scan $\mathbb{L}$ are not recorded at the same time. Assume the kth scan $\mathbb{L}_k$ is recorded within $(t^s_k, t^e_k]$. If $\mathbb{L}_k$ is a keyframe, it has two kinds of pose: the real-time tracking pose $\mathbf{T}^f_k$ at time t^e_k from the front-end, and the smoothed pose $\mathbf{T}^b_k$ from the back-end at time t^s_k, as illustrated in Fig. 1. $\mathbf{T}^f_k$ is the pose when the last point of a scan is recorded. Our front-end removes the motion distortion within $\mathbb{L}_k$. After removing the distortion, the points in $\mathbb{L}_k$ are transformed to the LiDAR coordinate system at time t^s_k. Thus the pose used in our back-end is the pose at t^s_k. Let



(a) Our ICP process (b) Traditional point-to-plane ICP

Fig. 3. A schematic of our ICP process (a) and the traditional point-to-plane ICP (b). Here $\mathbb{P}_{ij}$ is the set of points on $\boldsymbol{\pi}_j$ observed at pose $\boldsymbol{x}_i, i \in [1, 5]$. In our ICP process (a), the local-to-global data association is propagated scan by scan. The benefit is that we do not need to build and query a large KD-tree to get the local-to-global correspondence, and we do not need to explicitly detect planes in the whole scan, which is required in previous works [16], [7], [19]. Thus we can perform global registration in real time. In the traditional point-to-plane ICP (b), the global KD-tree is built on the first 4 point clouds. Plane parameters are estimated from a small patch of the global point cloud. Pose estimation errors reduce the quality of the map, which will in turn increase the pose error. Our framework eliminates this vicious loop, as $\boldsymbol{\pi}_j$ and the poses are jointly optimized in the back-end.

us define as $\mathbf{T}_{k-1}^f$ the tracking pose of the $(k-1)$th scan $\mathbb{L}_{k-1}$. The relationship between the tracking pose and the smoothed pose is:

$$\mathbf{T}_{k-1}^f = \mathbf{T}_k^b. \tag{2}$$

To simplify the notation, we use $\mathbf{T}_k$ to represent $\mathbf{T}_k^f$ or $\mathbf{T}_k^b$, if the meaning can be identified by the context of the following description.

Plane In this paper, we use $\boldsymbol{\pi} = [\mathbf{n}; d]$ to represent a plane, where $\mathbf{n}$ is the plane normal with $||\mathbf{n}||_2 = 1$, and d is the signed distance between the origin and $\boldsymbol{\pi}$. We use $\tilde{\mathbf{p}}$ to represent the homogeneous coordinates of a 3D point $\mathbf{p}$. Thus, a point on the plane $\boldsymbol{\pi}$ should satisfy $\boldsymbol{\pi}^T\tilde{\mathbf{p}} = 0$. As $\boldsymbol{\pi}$ has 3 degree of freedom (DOF), in the optimization we adopt the closest point (CP) vector [17] to parameterize $\boldsymbol{\pi}$, *i.e.*, $CP(\boldsymbol{\pi}) = d\mathbf{n}$.

System Overview Fig. 2 provides an overview of our system. Our system has two components: front-end and back-end. The front-end establishes local-to-global point-to-plane correspondences through an ICP process, and provides a globally consistent pose in real time. The back-end jointly optimizes planes and keyframe poses, and provides information to correct the drift of the pose in the front-end.

IV. Front-End

A. Plane Extraction

We only detect the plane in the whole LiDAR scan at the first pose. We adopt the region growing method introduced in [30] to extract the plane with the following changes. For a spinning LiDAR, the LiDAR point cloud has a structure. We organize the point cloud as a range image [31]. For the points within one scan line, we sort them according to the rotation angle. The K nearest neighbors of a certain point $\mathbf{p}$ in scan line i can only come from the same scan line or the scan line $i-1$ and $i+1$. In the region growing step, we select 5 nearest points for the query point $\mathbf{p}$ in scan line $i-1$, i and $i+1$, respectively. As there may exist occlusion, one plane surface may be broken into several pieces. We try to merge planes with similar parameters. We only keep planes with more than 50 points. We also extract new planes for a new keyframe, but this is only done for a small portion of the LiDAR scan that does not match current global planes.

B. In-Scan Motion

The measurements of a LiDAR scan are not collected at the same time. Thus, the measurements will suffer from the motion distortion. This problem can be solved by modeling the in-scan motion [1], [8]. Specifically, we denote with $\mathbb{L}_k$ and $\mathbb{L}_{k+1}$ the kth and $(k+1)$th scans. Assume $\mathbb{L}_{k+1}$ is collected between the time interval $(t_k, t_{k+1}]$. We denote the relative pose from t_{k+1} to t_k is $\mathbf{T}_{k+1}^k$, and assume we can parameterize $\mathbf{T}_{k+1}^k$ as $\boldsymbol{x}_{k+1}^k = \left[\boldsymbol{\omega}_{k+1}^k; \mathbf{t}_{k+1}^k\right]$, where $\boldsymbol{\omega}_{k+1}^k$ is the angle-axis representation of the rotation matrix in $\mathbf{T}_{k+1}^k$, and $\mathbf{t}_{k+1}^k$ is the translation vector in $\mathbf{T}_{k+1}^k$. Suppose the velocity, angle velocity and rotating axis are constant during $(t_k, t_{k+1}]$. Then the parameterization $\boldsymbol{x}_t^k$ of the relative pose at $t \in (t_k, t_{k+1}]$, denoted as $\mathbf{T}_t^k$, can be computed by the linear interpolation with the form

$$\boldsymbol{x}_t^k = s\boldsymbol{x}_{k+1}^k, \ s = \frac{t - t_k}{t_{k+1} - t_k}. \tag{3}$$

Denote $\mathbf{T}_k$ and $\mathbf{T}_t$ are the poses at t_k and $t \in (t_k, t_{k+1}]$, respectively. Thus, we have

$$\mathbf{T}_t = \mathbf{T}_k\mathbf{T}_t^k. \tag{4}$$

C. Global Registration through ICP Process

We introduce a novel ICP process to establish the local-to-global point-to-plane correspondences, and estimate the relative pose $\mathbf{T}_{k+1}^k$ in real time, as demonstrated in Fig. 3.

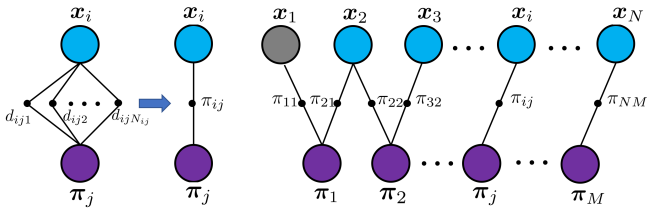
Data Association Assume that there are N global planes that are partially observed by the kth scan $\mathbb{L}_k$, *i.e.*, $\left\{\boldsymbol{\pi}_n \leftrightarrow \mathbb{P}_n^k\right\}_{n=1}^N$. We combine the N planar point set $\mathbb{P}_n^k$ to form $\mathbb{Q}^k = \cup_{n=1}^N \mathbb{P}_n^k$. Then we build a KD-tree on $\mathbb{Q}^k$. Let us denote the relative pose estimated from the ith ICP iteration as $\hat{\mathbf{T}}_{k+1}^{k,i}$. We use $\hat{\mathbf{T}}_{k+1}^{k,i}$ to transform $\mathbf{L}_{k+1}$ into the coordinate system of $\mathbf{L}_k$. This forms a point set $\mathbb{L}_{k+1}^i$. For each point in $\mathbb{L}_{k+1}^i$, we find the $\mathcal{K}$ nearest neighbors in $\mathbb{L}_k$ ($\mathcal{K} = 2$ in our experiments). If the $\mathcal{K}$ nearest neighbors belong to the same plane $\boldsymbol{\pi}_n$, we assign this point to $\mathbb{P}_n^{k+1}$. Otherwise, this point does not match any planes. Here we use the $\mathcal{K}$ nearest neighbors to avoid the ambiguous cases where points are nearby the intersection boundary of two planes. Using the above method, we establish the data association between $\mathbb{L}_{k+1}^i$ and the N global planes:

$$\left\{\boldsymbol{\pi}_n \leftrightarrow \mathbb{P}_n^k \leftrightarrow \mathbb{P}_n^{k+1}\right\}_{n=1}^N \tag{5}$$

We evaluate the distance between $\boldsymbol{\pi}_n$ and each point in $\mathbb{P}_n^{k+1}$. We only keep the points in $\mathbb{P}_n^{k+1}$ whose distances to $\boldsymbol{\pi}_n$ are less than ν ($\nu = 0.2m$ for the first iteration and $\nu = 0.1m$ for the remaining iterations in our experiments). Then we use these local-to-global point-to-plane correspondences to estimate $\mathbf{T}_{k+1}^k$.

State Estimation Assume $\mathbf{p}_{ni}^{k+1} \in \mathbb{P}_n^{k+1}$ is captured at $t \in (t_k, t_{k+1}]$ and $\tilde{\mathbf{p}}_{ni}^{k+1}$ is its homogeneous coordinates. Given $\mathbf{p}_{ni}^{k+1} \leftrightarrow \boldsymbol{\pi}_n$, we have the point-to-plane residual:

$$d_{ni}^{k+1}\left(\boldsymbol{x}_{k+1}^k\right) = \boldsymbol{\pi}_n^T\mathbf{T}_t\tilde{\mathbf{p}}_{ni}^{k+1}, \tag{6}$$



(a) π-factor (b) Plane adjustment problem

Fig. 4. In Fig. (a), $\boldsymbol{x}_i$ represents the ith keyframe, and $\boldsymbol{\pi}_j$ denotes the jth global plane. $\mathbb{P}_{ij}$ is a set of N_{ij} observations of $\boldsymbol{\pi}_j$ captured at $\boldsymbol{x}_i$. d_{ijk} is the signed distance between the kth point in $\mathbb{P}_{ij}$ and $\boldsymbol{\pi}_j$, defined in (7). We introduce the π-factor to compress the N_{ij} constraints into one. Fig. (b) illustrates the factor graph for the plane adjustment with N poses and M planes using the π-factor. $\boldsymbol{x}_1$ is fixed during the optimization.

where $\mathbf{T}_t$ is defined in (4). Stacking all the point-to-plane constraints, we get a residual vector $\mathbf{d}^{k+1}(\boldsymbol{x}_{k+1}^k)$. We use the LM algorithm to minimize the cost $\mathbf{d}^{k+1}(\boldsymbol{x}_{k+1}^k)^T\mathbf{d}^{k+1}(\boldsymbol{x}_{k+1}^k)$. The rotation and translation of $\boldsymbol{x}_{k+1}^k$ are initialized as the identity matrix $\mathbf{I}_3$ and $[0;0;0]$, respectively. We adopt the robust Huber loss during optimization. After we get $\mathbf{T}_{k+1}^k$, we can compute $\mathbf{T}_{k+1} = \mathbf{T}_k\mathbf{T}_{k+1}^k$. We conduct at most three ICP iterations in our experiments.

D. Keyframe Decision

We use the following three criteria to determine whether a new keyframe is needed:

- More than 30% of the points in $\mathbf{L}_{k+1}$ are not matched.
- The distance between current frame and latest keyframe is larger than τ ($\tau = 0.15m$ in our experiments).
- The rotation angle between current frame and latest keyframe is larger than θ ($\theta = 10^\circ$ in our experiments).

We more aggressively bring new keyframes into the map in the initial stage (*i.e.*, the trajectory length is less than $2m$), where $\tau = 0.1m$ and $\theta = 5^\circ$. This is because the LiDAR point cloud is sparse, the plane parameters from one scan are generally of low quality. Not all the keyframes will be kept, some of them will be deleted afterward as done in [32].

E. New Planes Detection

For a new keyframe, we first detect planes from the points which do not match the global planes using the algorithm introduced in Section IV-A. We only keep the planes with more than 50 points. If a new global plane cannot be tracked more than three keyframes, it will be deleted.

V. Back-End

In this section, we introduce our back-end. The back-end jointly optimizes plane parameters and keyframe poses in a sliding window, and also detects the loop for global optimization. We introduce π-factor to speed up the optimization.

A. Plane Adjustment

Our aim is to jointly optimize the plane parameters and the keyframe poses, which is similar to BA in visual SLAM, where 3D points and camera poses are jointly optimized. As BA has a special meaning for the point feature in visual SLAM [33], here we name this joint minimization problem **plane adjustment** (**PA**). Fig. 4 demonstrates the factor graph of a PA problem.

Assume there are N keyframes and M planes. Suppose $\mathbb{P}_{ij}$ is the set of N_{ij} points on plane $\boldsymbol{\pi}_j$ observed at pose $\mathbf{T}_i$, and $\mathbf{p}_{ijk}$ is the kth point of $\mathbb{P}_{ij}$. Assume $\tilde{\mathbf{p}}_{ijk}$ is the homogeneous coordinates of $\mathbf{p}_{ijk}$, the signed distance between $\mathbf{p}_{ijk}$ and $\boldsymbol{\pi}_j$ has the form:

$$d_{ijk}\left(\mathbf{T}_i, \boldsymbol{\pi}_j\right) = \boldsymbol{\pi}_j^T\mathbf{T}_i\tilde{\mathbf{p}}_{ijk}. \tag{7}$$

Here $d_{ijk}\left(\mathbf{T}_i, \boldsymbol{\pi}_j\right)$ is a function of $\mathbf{T}_i$ and $\boldsymbol{\pi}_j$. To simplify the notation, we omit $\mathbf{T}_i$ and $\boldsymbol{\pi}_j$ in the following description. The cost function of PA is:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{N_{ij}} d_{ijk}^2. \tag{8}$$

As N_{ij} is generally very large, PA is generally a very large least-squares problem even for a small place. We introduce the π-factor to solve this problem as described below.

B. π-Factor

Using the definition of d_{ijk} in (7), d_{ijk}^2 has the form:

$$d_{ijk}^2 = \boldsymbol{\pi}_j^T\mathbf{T}_i\tilde{\mathbf{p}}_{ijk}\tilde{\mathbf{p}}_{ijk}^T\mathbf{T}_j^T\boldsymbol{\pi}_j. \tag{9}$$

Then, the least-squares cost for the N_{ij} points has the form:

$$\begin{aligned} c_{ij}(\mathbf{T}_i, \boldsymbol{\pi}_j) &= \sum_{k=1}^{N_{ij}} d_{ijk}^2 = \boldsymbol{\pi}_j^T\mathbf{T}_i \underbrace{\left(\sum_{k=1}^{N_{ij}}\tilde{\mathbf{p}}_{ijk}\tilde{\mathbf{p}}_{ijk}^T\right)}_{\mathbf{G}_{ij}}\mathbf{T}_j^T\boldsymbol{\pi}_j \\ &= \boldsymbol{\pi}_j^T\mathbf{T}_i\mathbf{G}_{ij}\mathbf{T}_i^T\boldsymbol{\pi}_j. \end{aligned} \tag{10}$$

The π-factor for $\mathbf{T}_i$ and $\boldsymbol{\pi}_j$ is defined as

$$\pi_{ij}(\mathbf{T}_i, \boldsymbol{\pi}_j) = \sqrt{c_{ij}(\mathbf{T}_i, \boldsymbol{\pi}_j)}. \tag{11}$$

The π-factor can be easily incorporated into the factor graph, as shown in Fig. 4. For each $\mathbb{P}_{ij}$, we only need to compute $\mathbf{G}_{ij}$ once. $\mathbf{G}_{ij}$ is a constant 4×4 matrix during the optimization. Using the π-factor, we integrate the N_{ij} constraints into one. This significantly reduces the computational cost of the LM algorithm. As a plane has three DOF, a plane is added into (8) if it has been observed more than three times.

Using the definition of the π-factor in (11), we can rewrite the PA cost function E in (8) as:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}\pi_{ij}^2(\mathbf{T}_i, \boldsymbol{\pi}_j). \tag{12}$$

C. Local Plane Adjustment

We adopt a windowed optimization strategy [32] to update the newest w keyframe poses and the corresponding active planes for efficiency, named local plane adjustment (**LPA**). When the number of keyframe poses is larger than the window size, the oldest keyframe is removed from the sliding window. We simply delete this keyframe if the following conditions are met:

- This keyframe does not include a new plane or all its new planes have been deleted.
- The distance between this keyframe and the one before this keyframe is smaller than $0.3m$.

- The rotation angle between this keyframe and the one before this keyframe is smaller than $15°$.

Otherwise, we keep this keyframe for global optimization, and marginalize it and the planes which are not seen by the remaining keyframes in the sliding window for LPA. The marginalization is done by using the Schur complement. We adopt the LM algorithm to solve this minimization problem. Here we sparsify the keyframes to reduce the runtime of the global optimization.

Drift Correction After we smooth the pose using the LPA, we correct the drift in the tracking pose. Let us assume $\mathbf{T}_k^b$ is the smoothed pose of the latest keyframe. Using the relationship between the tracking pose and the smoothed pose in (2), we know the corresponding tracking pose in the front-end is $\mathbf{T}_{k-1}^f$. Then we can get the correction for the drift $\Delta\mathbf{T} = \mathbf{T}_k^b(\mathbf{T}_{k-1}^f)^{-1}$. Assume the latest tracking pose is $\mathbf{T}_m^f$, we calculate $\Delta\mathbf{T}\mathbf{T}_m^f$ to correct the drift. Our experimental results show that the runtime for the LPA is generally less than the runtime of the ICP process. This means we can correct the drift in real time.

D. Global Plane Adjustment

We conduct global plane adjustment (**GPA**) to minimize the cost (12) to globally refine keyframe poses and plane parameters, if a loop is detected.

Trigger Loop Detection We trigger the loop detection, when we find that the latest keyframe approaches an old keyframe. Formally, let us denote the latest keyframe as $\mathbb{L}_n$, and the last keyframe where the loop closure occurred as $\mathbb{L}_l$. Given a previous keyframe $\mathbb{L}_i$, we define the straight distance from $\mathbb{L}_i$ to $\mathbb{L}_n$ as $\delta_{ni} = ||\mathbf{t}_n - \mathbf{t}_i||_2$, where $\mathbf{t}_n$ and $\mathbf{t}_i$ are the translation vectors of $\mathbb{L}_n$ and $\mathbb{L}_i$, respectively, and we define the distance from $\mathbb{L}_i$ to $\mathbb{L}_n$ along the trajectory as $\Delta_{ni} = \sum_{j=i}^{n-1} \delta_{nj}$. We trigger loop detection if $\delta_{il} > 5m$ and $\Delta_{ni} > 5m$ and $\delta_{ni} < 0.5m$. Here δ_{il} is the straight distance between $\mathbb{L}_n$ and $\mathbb{L}_l$. We require $\delta_{il} > 5m$ to avoid calling for unnecessary loop detection at a place where loop closure just happened. If there are multiple keyframes that satisfy the above conditions, we choose the one with the smallest straight distance, and denote it as $\mathbb{L}_o$. We try to register $\mathbb{L}_n$ to $\mathbb{L}_o$ to determine whether a loop occurs. We adopt the RANSAC algorithm to achieve this.

Plane Correspondences Although we do not explicitly extract planes for each scan, the ICP process introduced in section IV-C actually identifies the planes within a scan, as shown in (5). We first fit a plane for each planar set in (5) for $\mathbb{L}_n$ and $\mathbb{L}_o$, respectively. Given the estimated poses, we transform these local plane parameters into the global coordinate system, and calculate their CP vectors. For one plane in $\mathbb{L}_n$, we find the plane with the closest CP vector in $\mathbb{L}_o$, and vice versa. If one plane pair are mutually closest, we keep this pair and verify them in the RANSAC algorithm..

RANSAC The pose between two coordinate systems can be calculated from three plane-to-plane correspondences [34]. We use the distance between two CP vectors to determine the number of inliers. Specifically, let us denote the ith plane-to-plane correspondences as $\boldsymbol{\pi}_i^{new} \leftrightarrow \boldsymbol{\pi}_i^{old}$. Given the transformation $\mathbf{T}_n^o$ from $\mathbb{L}_n$ to $\mathbb{L}_o$ which is estimated from three randomly selected plane-to-plane correspondences, we treat $\boldsymbol{\pi}_i^{new} \leftrightarrow \boldsymbol{\pi}_i^{old}$ as an inlier if $||CP(\boldsymbol{\pi}_i^{new}) - CP(\mathbf{T}_n^{oT}\boldsymbol{\pi}_i^{old})||_2 < \zeta$ ($\zeta = 0.15$ in our experiment, and $CP(\cdot)$ is defined in section III). If we find more than six inliers, we consider that a loop closure occurs, and we run the GPA to minimize the cost (12) to globally refine the poses and plane parameters.

Merge Global Planes The local planes in $\mathbb{L}_o$ and $\mathbb{L}_n$ are all associated to global planes. If the global planes associated to an inlier local plane pair are not the same, we merge the two global planes. We average the two CP vectors and combine the π-factors together.

Update the System The GPA refines all the variables After GPA, we need to update the whole system. The drift of the tracking pose is corrected, as done after LPA in section V-C. In addition, we update the state of the variables in LPA, and conduct re-marginalization for LPA to make the whole system consistent. Specifically, the poses out of the sliding window and all the planes that are not seen by these poses are marginalized using the new states.

TABLE I

STATISTICS OF OUR 3 INDOOR DATASETS. #SCAN REPRESENTS THE NUMBER OF SCANS.

Dataset	1	2	3
#Scan	4427	4504	5812
Length (m)	105.38	117.98	215.08

TABLE II

$\Delta\theta(°)$ AND $\Delta t(m)$ BETWEEN THE START POINT AND THE END POINT.

Algorithm	1		2		3	
	$\Delta\theta$	Δt	$\Delta\theta$	Δt	$\Delta\theta$	Δt
LeGO-LOAM [2]	5.63	0.13	97.8	3.8	91.2	30.92
π-LASM - Undis	0.72	0.052	0.82	0.068	0.98	0.082
π-LASM - GPA	0.95	0.064	0.91	0.083	1.71	0.12
π-LASM - GPA - LPA	4.27	0.11	3.90	0.18	3.63	0.26
π-LASM	0.57	0.043	0.36	0.038	0.46	0.048

VI. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our algorithm. We first perform an ablation study to test the impact of different components. Then we compare our algorithm with the state-of-the-art LOAM variant, LeGO-LOAM [2]. Finally, we provide the runtime analysis. We ran the experiments on a computer with an i7-3770 CPU and 16G memory.

A. Dataset

We used a VLP-16 LiDAR to collect three indoor datasets. Table I lists their statistics. As shown in Fig. 5, the three datasets are challenging. They all have multiple rapid rotating motions. The three datasets all contain a closed loop. That is to say their start and end points are the same. So we can use the difference between the poses of the start and the end points to quantitatively evaluate the accuracy of a SLAM algorithm. Let us denote the poses of the start and the end points are $(\hat{\mathbf{R}}_s, \hat{\mathbf{t}}_s)$ and $(\hat{\mathbf{R}}_e, \hat{\mathbf{t}}_e)$, respectively. We use $\Delta\theta = \angle(\hat{\mathbf{R}}_s(\hat{\mathbf{R}}_e)^{-1})$ and $\Delta t = ||\hat{\mathbf{t}}_s - \hat{\mathbf{R}}_s(\hat{\mathbf{R}}_e)^{-1}\hat{\mathbf{t}}_e||_2$ to evaluate the performance of different algorithms. Here $\angle(\hat{\mathbf{R}}_s(\hat{\mathbf{R}}_e)^{-1})$ means the angle of the angle-axis representation of $\hat{\mathbf{R}}_s(\hat{\mathbf{R}}_e)^{-1}$. As the end point in $\boldsymbol{\pi}$-LSAM may not be a keyframe, we compute $\Delta\theta$ and Δt for the tracking pose.

TABLE III

COMPUTATIONAL TIME (ms) OF DIFFERENT COMPONENTS OF π-SLAM. NEW PLANE DETECTION IS ONLY REQUIRED FOR A KEYFRAME.

Dataset	Front-End			Back-End		
	ICP	Keyframe Decision	New Plane Detection*	LPA	Loop Detection	GPA
1	53.2 $\pm$ 8.9	0.31 $\pm$ 0.11	12.5 $\pm$ 3.6	43.2 $\pm$ 7.1	21.2 $\pm$ 4.9	356.7 $\pm$ 50.4
2	58.5 $\pm$ 7.4	0.28 $\pm$ 0.13	13.8$\pm$ 4.8	45.3 $\pm$ 7.9	23.9 $\pm$ 5.3	330.3 $\pm$86.8
3	55.7 $\pm$ 8.3	0.26 $\pm$ 0.10	13.2$\pm$ 4.1	46.3 $\pm$ 8.3	25.2 $\pm$ 5.8	380.7 $\pm$89.5

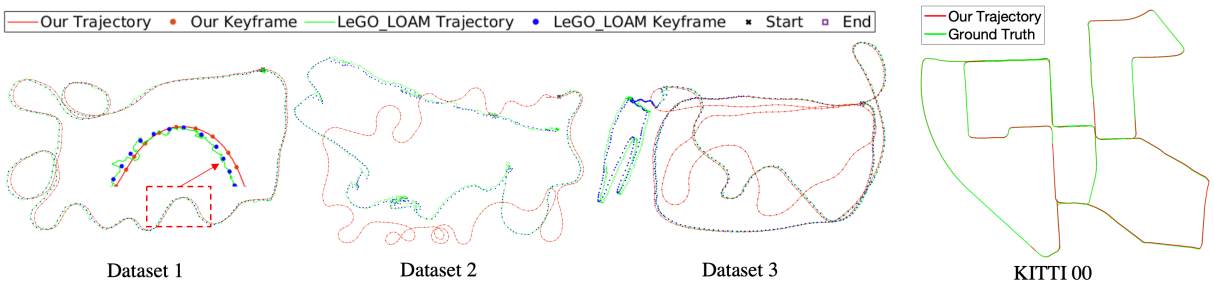


Fig. 5. The results of our algorithm and LeGO-LOAM [2]. We show the trajectory (tracking poses) and the keyframe poses. LeGO-LOAM fails to work on datasets 2 and 3, and its trajectories have obvious jitters. We also show the result of our algorithm on KITTI 00.

B. Ablation Study

Here we evaluate the impact of different components of our algorithm. The size of the sliding window is set to 10. We consider the following variants of our algorithm:

- π-LASM - Undis: The motion unditortion is removed, which means s in (3) is set to 1. We adopt the method in [35] to solve the point-to-plane registration problem.
- π-LASM - GPA: The GPA is removed.
- π-LASM - GPA - LPA : The GPA and LPA are removed.

Table II shows the pose errors between the start point and the end point for the above variants. It is clear that the unditortion, LPA and GPA all benefit the pose estimation.

C. Comparison with the State-of-the-Art

We compare our algorithm with the state-of-the-art LOAM variant, LeGO-LOAM [2]. The pose errors between the start point and the end point are listed in Table II. Fig. 5 shows the trajectories (tracking poses) and the keyframe poses generated by our algorithm and LeGO-LOAM. The keyframes of our algorithm in Fig. 5 are the keyframes kept for GPA. As introduced in section V-C, some keyframes are deleted to balance the efficiency and accuracy. LeGO-LOAM fails to work on datasets 2 and 3. It is clear that the quality of the tracking poses and the keyframe poses of LeGO-LOAM is different. The tracking poses of LeGO-LOAM have many jitters, and its keyframe poses are smoother. But the difference between our tracking pose and the keyframe pose is marginal. This is because the tracking poses of LeGO-LOAM are computed from registering two consecutive scans. But our tracking poses are generated from global registration. On the other hand, we find that π-LASM - GPA - LPA can generate better results than LeGO-LOAM. Besides, as we show in the table III, our LPA component generally achieves real time, which means the drift can be corrected and the plane parameters can be updated in real time.

Although our algorithm is designed for indoor scenes, it is also applicable to outdoor scenes if planes can provide enough constraints for pose estimation. We tested $\boldsymbol{\pi}$-LSAM on sequence 00 of KITTI [36]. Fig. 5 shows the result. The relative position errors of $\boldsymbol{\pi}$-LSAM and LOAM are 0.72% and 0.78%, respectively. We achieve a state-of-the-art result.

D. Runtime

Table III provides the runtime analysis of π-LSAM. As it takes $100ms$ for a VLP-16 LiDAR to finish one scan, our front-end achieves real time. The new plane detection is only required for a new keyframe. As it runs on a small portion of a scan, it is efficient. We find that even with the new plane detection, the front-end of π-LSAM still achieves real time. The time-consuming part is GPA. Its runtime has a large variance, as the loop closure occurs several times in the three datasets. The GPA at the earlier stage is faster, as the number of variables is smaller. The loop detection and GPA are in an individual thread, so they will not block other components. We also test the runtime of the plane extraction algorithm introduced in section IV-A on dataset 3. The runtime for this algorithm is about $70.1 \pm 5.8ms$ on dataset 3. Our ICP process can avoid this time-consuming step. Finally, we test the runtime of LPA and GPA without the π-factor on dataset 3. It shows that the π-factor can on average speed up LPA and GPA 28 times and 36 times, respectively.

VII. CONCLUSION

In this paper, we present a new LiDAR SLAM algorithm, π-LSAM, which uses planes as landmarks. We introduce PA in the back-end, and combine LPA and GPA to balance the efficiency and accuracy. We introduce the π-factor to significantly reduce the computational complexity of PA, and present an efficient loop closure detection algorithm using planes. We introduce an ICP process in the front-end to provide accurate globally consistent tracking poses in real time. The ICP process avoids explicitly extracting planes in the whole scan, and it allows us to use a small local KD-tree to establish the local-to-global data association, which requires large global KD-trees in previous works. Our experiments show that LPA and GPA can significantly improve the pose estimation. Furthermore, π-LSAM achieves real time and significantly outperforms the state-of-the-art LOAM variant, LeGO-LOAM [2].

REFERENCES

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

[2] 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.

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

[4] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE, 2001, pp. 145–152.

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

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

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

[8] 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.

[9] 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.

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

[11] D. Rozenberszki and A. L. Majdik, "Lol: Lidar-only odometry and localization in 3d point cloud maps," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4379–4385.

[12] 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.

[13] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE, 2008, pp. 643–650.

[14] J. Serafin, E. Olson, and G. Grisetti, "Fast and robust 3d feature extraction from sparse point clouds," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4105–4112.

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

[16] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga, "Fast registration based on noisy planes with unknown correspondences for 3-d mapping," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 424–441, 2010.

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

[18] 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.

[19] Y. Yang, P. Geneva, X. Zuo, K. Eckenhoff, Y. Liu, and G. Huang, "Tightly-coupled aided inertial navigation with point and plane features," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6094–6100.

[20] 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.

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

[22] L. Zhou, D. Koppel, H. Ju, F. Steinbruecker, and M. Kaess, "An efficient planar bundle adjustment algorithm," in *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2020, pp. 136–145.

[23] G. Ferrer, "Eigen-factors: Plane estimation for multi-frame and time-continuous point cloud alignment." in *IROS*, 2019, pp. 1278–1284.

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

[25] 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.

[26] B. Steder, G. Grisetti, and W. Burgard, "Robust place recognition for 3d range data based on point features," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1400–1405.

[27] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard, "Place recognition in 3d scans using a combination of bag of words and point feature based relative pose estimation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 1249–1255.

[28] J. Du, R. Wang, and D. Cremers, "DH3D: Deep hierarchical 3d descriptors for robust large-scale 6dof relocalization," in *European Conference on Computer Vision (ECCV)*, 2020.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

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

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

[32] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[33] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[34] F. M. Mirzaei, D. G. Kottas, and S. I. Roumeliotis, "3d lidar–camera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 452–467, 2012.

[35] L. Zhou, S. Wang, and M. Kaess, "A Fast and Accurate Solution for Pose Estimation from 3D Correspondences," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1308–1314.

[36] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.