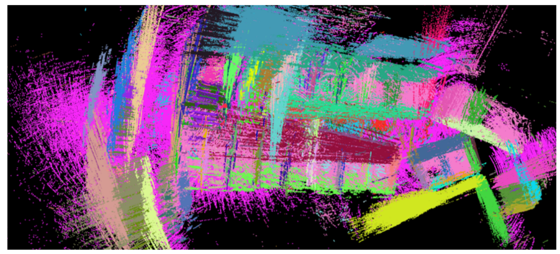
Efficient Second-Order Plane Adjustment

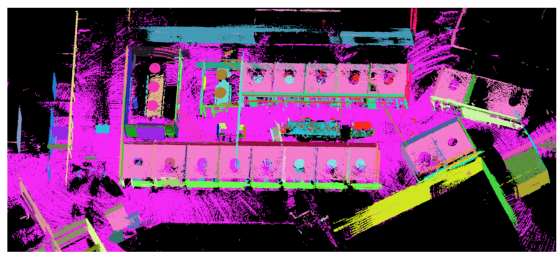
Lipu Zhou
Meituan, Beijing, China
zhoulipu@meituan.com

Abstract

Planes are generally used in 3D reconstruction for depth sensors, such as RGB-D cameras and LiDARs. This paper focuses on the problem of estimating the optimal planes and sensor poses to minimize the point-to-plane distance. The resulting least-squares problem is referred to as plane adjustment (PA) in the literature, which is the counterpart of bundle adjustment (BA) in visual reconstruction. Iterative methods are adopted to solve these least-squares problems. Typically, Newton's method is rarely used for a large-scale least-squares problem, due to the high computational complexity of the Hessian matrix. Instead, methods using an approximation of the Hessian matrix, such as the Levenberg-Marquardt (LM) method, are generally adopted. This paper adopts the Newton's method to efficiently solve the PA problem. Specifically, given poses, the optimal plane have a close-form solution. Thus we can eliminate planes from the cost function, which significantly reduces the number of variables. Furthermore, as the optimal planes are functions of poses, this method actually ensures that the optimal planes for the current estimated poses can be obtained at each iteration, which benefits the convergence. The difficulty lies in how to efficiently compute the Hessian matrix and the gradient of the resulting cost. This paper provides an efficient solution. Empirical evaluation shows that our algorithm outperforms the state-of-the-art algorithms.



(a) Point cloud from perturbed poses



(b) Point cloud from our algorithm

Figure 1. We use Gaussian noises to perturb the poses of dataset D in Fig. 3. The standard deviations for rotation and translation are 3° and $0.3m$, respectively. The resulting point cloud (a) is in a mess. Fig. (b) shows the result from our algorithm. Our algorithm can quickly align the planes, as shown in Fig. 5.

1. Introduction

Planes ubiquitously exist in man-made environments, as demonstrated in Fig. 1. Thus they are generally used in simultaneous localization and mapping (SLAM) systems for depth sensors, such as RGB-D cameras [6, 9, 12–14] and LiDARs [16, 21, 22, 24, 26]. Just as bundle adjustment (BA) [3, 8, 20, 25] is important for visual reconstruction [1, 5, 18, 19], jointly optimizing planes and depth sensor poses, which is called plane adjustment (PA) [24, 26], is critical for 3D reconstruction using depth sensors. This paper focuses on efficiently solving the large-scale PA problem.

The BA and PA problems both involve jointly optimizing 3D structures and sensor poses. As the two problems are similar, it is straightforward to apply the well-studied solutions for BA to PA, as done in [23, 26]. However, planes in PA can be eliminated, so that the cost function of the PA problem only depends on sensor poses, which significantly reduces the number of variables. This property provides a promising direction to efficiently solve the PA problem. However, it is difficult to compute the Hessian matrix and the gradient vector of the resulting cost. Although this problem was studied in several previous works [10, 16], no efficient solution has been proposed. This paper seeks to solve this problem.

The main contribution of this paper is an efficient PA solution using Newton's method. We derive a closed-form solution for the Hessian matrix and the gradient vector for the PA problem whose computational complexity is independent of the number of points on the planes. Our experimental results show that the proposed algorithm converges faster than the state-of-the-art algorithms.

2. Related Work

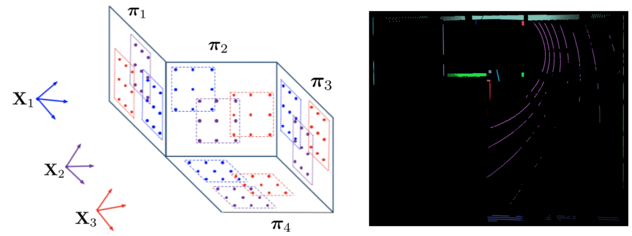
The PA problem is closely related to the BA problem. In BA, points and camera poses are jointly optimized to minimize the reprojection error. Schur complement [3, 20, 25] or the square root method [7, 8] is generally used to solve the linear system of the iterative methods. The keypoint is to generate a reduced camera system (RCS) which only relates to camera poses.

In PA, planes and poses are jointly optimized. Planes are the counterparts of points in BA. Thus, the well-known solutions for the BA problem can be applied to the PA problem [23, 24]. In the literature, two cost functions are used to formulate the PA problem. The first one is the plane-to-plane distance which measures the difference between two plane parameters [12, 13]. The value of the plane-to-plane distance is related to the choice of the global coordinate system, which means the selection of the global coordinate system will affect the accuracy of the results. The second one is the point-to-plane distance, whose value is invariant to the choice of the global coordinate system. The solutions of different choices of the global coordinate system are equivalent up to a rigid transformation. Zhou *et al.* [23] show that the point-to-plane distance can converge faster and lead to a more accurate result. But unlike BA where each 3D point has only one 2D observation at a pose, a plane can generate many points at a pose as demonstrated in Fig. 2. This means the point-to-plane distance probably leads to a very large-scale least-squares problem. Directly adopting the BA solutions is computationally infeasible for a large-scale PA problem. Zhou *et al.* [23] propose to use the QR decomposition to accelerate the computation.

For a general least-squares problem with M variables, the computational complexity of the Hessian matrix is $O(M^2)$. Thus, in the computer vision community, it is ingrained that Newton's method is infeasible for a large-scale optimization problem, as calculating the Hessian matrix is computationally demanding. Instead, Gauss-Newton-based iterative methods are generally adopted. Suppose that $\mathbf{J}$ is the Jacobian matrix of the residuals. The Gauss-Newton method actually approximates the Hessian matrix by $\mathbf{H} \approx \mathbf{J}^T\mathbf{J}$. In theory, Newton's method can lead to a better quadratic approximation to the original cost function, which means the Newton's step probably yields a better result. This in turn may reduce the number of iterations.

The PA problem has a special property that the optimal plane parameters are determined by the poses. That is to say the point-to-plane cost actually only depends on the poses. This property is attractive, as it significantly reduces the number of variables, which makes using the Newton's method possible. In traditional iterative methods, the correlation between the plane parameters and the poses are ignored. Thus, after one iteration, there is no guarantee that the new plane parameters are optimal for the new poses. Using the property of the PA, it is possible to overcome this drawback, which may lead to faster convergence. Several previous works seek to exploit this property of PA. Ferrer [10] explored an algebraic point-to-plane distance and provided a closed-form gradient for the resulting cost. The algebraic cost may result in a suboptimal solution [4], and the first-order optimization generally leads to slow convergence [20]. Liu *et al.* [16] provided analytic forms of the Hessian matrix and the gradient of the genuine point-to-plane cost. Assume that N points are captured from a plane. The computational complexity of the Hessian matrix related to these points is $O(N^2)$. Since N can be large as shown in Fig. 2, this algorithm is computationally demanding and infeasible for a large-scale problem.

In summary, the potential benefits of the special property of the PA problem have not been manifested in previous works. The bottleneck is how to efficiently compute the Hessian matrix and the gradient vector. This paper focuses on solving this problem.



(a) A PA problem with 3 poses and 4 planes (b) Plane Observation

Figure 2. A schematic of the PA problem and the planes detected in a LiDAR scan. Unlike BA where each 3D point only has one observation, many points can be captured from a plane in PA. Assume that N points are captured from a plane. The computational complexity of the Hessian matrix related to these points using BALM [16] is $O(N^2)$. Thus, this method is infeasible for a large-scale problem. In contrast, the computational complexity of our algorithm is independent of N.

3. Problem Formulation

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

3.1. Notations

Planes and Poses A plane can be represented by a four-dimensional vector $\boldsymbol{\pi} = [\boldsymbol{n}; d]$. We denote the rotational and translational components from a depth sensor coordinate system to the global coordinate system as $\mathbf{R} \in SO(3)$

and $\boldsymbol{t} \in \mathbb{R}^3$, respectively. To simplify the notation in the following description, we also use the following two matrices to represent a pose:

$$\mathbf{X} = \begin{bmatrix} \mathbf{R}, & \boldsymbol{t} \\ \mathbf{0}, & 1 \end{bmatrix} \in SE(3) \text{ and } \mathbf{T} = \begin{bmatrix} \mathbf{R}, & \boldsymbol{t} \end{bmatrix}. \quad (1)$$

As $\mathbf{R} \in SO(3)$, a certain parameterization is usually adopted in the optimization [20]. In this paper, we use the Cayley-Gibbs-Rodriguez (CGR) parameterization [11] to represent $\mathbf{R}$

$$\mathbf{R} = \frac{\bar{\mathbf{R}}}{1 + \boldsymbol{s}^T\boldsymbol{s}}, \bar{\mathbf{R}} = \left(1 - \boldsymbol{s}^T\boldsymbol{s}\right)\mathbf{I}_3 + 2[\boldsymbol{s}]_\times + 2\boldsymbol{s}\boldsymbol{s}^T, \quad (2)$$

where $\boldsymbol{s} = [s_1; s_2; s_3]$ is a three-dimensional vector. We adopt the CGR parameterization as it is a minimal representation for $\mathbf{R}$. Furthermore, unlike the angle-axis parameterization that is singular at $\mathbf{I}_3$, the CGR parameterization is well-defined at $\mathbf{I}_3$, and equals to $[0; 0; 0]$ which can accelerate the computation, as described in Section 4.3. We parameterize $\mathbf{X}$ as a six-dimensional vector $\boldsymbol{x} = [\boldsymbol{s}; \boldsymbol{t}]$.

Newton's method This paper adopts the damped Newton's method in the optimization. For a cost function $f(\boldsymbol{z})$, the damped Newton's method seeks to find its minimizer from an initial point. Assume that $\boldsymbol{z}_n$ is the solution at the nth iteration. Given the Hessian matrix $\mathbf{H}_f(\boldsymbol{z}_n)$ and the gradient $\boldsymbol{g}_f(\boldsymbol{z}_n)$ at $\boldsymbol{z}_n$, $\boldsymbol{z}_n$ is updated by $\boldsymbol{z}_{n+1} = \boldsymbol{z}_n + \Delta\boldsymbol{z}$. Here $\Delta\boldsymbol{z}$ is from

$$(\mathbf{H}_f(\boldsymbol{z}_n) + \mu\mathbf{I}))\Delta z = -\boldsymbol{g}_f(\boldsymbol{z}_n), \quad (3)$$

where μ is adjusted at each iteration to make the value of $f(\boldsymbol{z})$ reduce, as done in the Levenberg-Marquardt (LM) algorithm [17].

Matrix Calculus In the following derivation, we will use vector-by-vector, vector-by-scalar, scalar-by-vector derivatives. Here we provide their definitions.

Assume $\boldsymbol{a} = [a_1; \cdots; a_N] \in \mathbb{R}^N$ is a vector function of $\boldsymbol{b} = [b_1, \cdots, b_M] \in \mathbb{R}^M$. The first-order partial derivatives of vector-by-vector $\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}}$, vector-by-scalar $\frac{\partial \boldsymbol{a}}{\partial b_j}$, and scalar-by-vector $\frac{\partial a_i}{\partial \boldsymbol{b}}$ are defined as

$$\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}} = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \cdots & \frac{\partial a_N}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial b_M} & \cdots & \frac{\partial a_N}{\partial b_M} \end{bmatrix}, \frac{\partial \boldsymbol{a}}{\partial b_j} = \begin{bmatrix} \frac{\partial a_1}{\partial b_j} \\ \vdots \\ \frac{\partial a_N}{\partial b_j} \end{bmatrix}, \frac{\partial a_i}{\partial \boldsymbol{b}} = \begin{bmatrix} \frac{\partial a_i}{\partial b_1} \\ \vdots \\ \frac{\partial a_i}{\partial b_M} \end{bmatrix} \quad (4)$$

where $\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}}$ is an $M \times N$ matrix with $\frac{\partial a_j}{\partial b_i}$ as the ith row jth column element, $\frac{\partial \boldsymbol{a}}{\partial b_j}$ is an N-dimensional vector whose ith term is $\frac{\partial a_i}{\partial b_j}$, and $\frac{\partial a_i}{\partial \boldsymbol{b}}$ is an M-dimensional vector whose ith term is $\frac{\partial a_i}{\partial b_i}$.

3.2. Optimal Plane Estimation

Given a set of K points $\{\boldsymbol{p}_i\}$, the optimal plane $\hat{\boldsymbol{\pi}}$ can be estimated by minimizing the sum of squared point-to-plane distances

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}} \sum_i^K \left(\boldsymbol{n}^T\boldsymbol{p}_i + d\right)^2, \; s.t. \; \|\boldsymbol{n}\|_2^2 = 1. \quad (5)$$

There is a closed-form solution for $\hat{\boldsymbol{\pi}}$. Let us define

$$\mathbf{M} = \sum_{i=1}^K (\boldsymbol{p}_i - \bar{\boldsymbol{p}})(\boldsymbol{p}_i - \bar{\boldsymbol{p}})^T = \mathbf{S} - K\bar{\boldsymbol{p}}\bar{\boldsymbol{p}}^T, \quad (6)$$

where $\mathbf{S} = \sum_{i=1}^K \boldsymbol{p}_i\boldsymbol{p}_i^T$ and $\bar{\boldsymbol{p}} = \frac{1}{K}\sum_i^K \boldsymbol{p}_i$. Assume that $\lambda_3(\mathbf{M})$ and $\boldsymbol{\xi}_3(\mathbf{M})$ are the smallest eigenvalue of $\mathbf{M}$ and the corresponding eigenvector, respectively. Using these notations, we can write the optimal plane $\hat{\boldsymbol{\pi}} = [\hat{\boldsymbol{n}}; \hat{d}]$ as

$$\hat{\boldsymbol{n}} = \boldsymbol{\xi}_3(\mathbf{M}), \; \hat{d} = -\hat{\boldsymbol{n}}^T\bar{\boldsymbol{p}}. \quad (7)$$

Furthermore, the cost of (5) at $\hat{\boldsymbol{\pi}}$ equals to $\lambda_3(\mathbf{M})$, *i.e.,*

$$\lambda_3(\mathbf{M}) = \sum_{i=1}^K \left(\hat{\boldsymbol{n}}\boldsymbol{p}_i + \hat{d}\right)^2 = \min_{\boldsymbol{\pi}} \sum_{i=1}^K (\boldsymbol{n}\boldsymbol{p}_i + d)^2. \quad (8)$$

The above property will be used to eliminate planes in PA.

3.3. Plane Adjustment

Assume that there are M planes and N poses. According to section 3.1, the ith plane can be represented by a four-dimensional vector $\boldsymbol{\pi}_i = [\boldsymbol{n}_i; d_i]$. The jth pose is denoted as $\boldsymbol{x}_j$. The observation of $\boldsymbol{\pi}_i$ at $\boldsymbol{x}_j$ is a set of N_{ij} points $\mathbb{Q}_{ij} = \{\boldsymbol{p}_{ijk} \in \mathbb{R}^3\}_{k=1}^{N_{ij}}$. For a 3D point $\boldsymbol{p}_{ijk}$, we use $\tilde{\boldsymbol{p}}_{ijk} = [\boldsymbol{p}_{ijk}; 1]$ to represent the homogeneous coordinates of $\boldsymbol{p}_{ijk}$. Then, the transformation from the local coordinate system at $\boldsymbol{x}_j$ to the global coordinate system can be represented as

$$\boldsymbol{p}_{ijk}^g = \mathbf{R}_j\boldsymbol{p}_{ijk} + \boldsymbol{t}_j = \mathbf{T}_j\tilde{\boldsymbol{p}}_{ijk}, \quad (9)$$

where $\mathbf{T}_j$ is defined in (1). Then the distance d_{ijk} from $\boldsymbol{p}_{ijk}$ to $\boldsymbol{\pi}_i$ has the form

$$\begin{aligned} d_{ijk}(\boldsymbol{\pi}_i, \mathbf{x}_j) &= \boldsymbol{n}_i^T(\mathbf{R}_j\boldsymbol{p}_{ijk} + \boldsymbol{t}_j) + d_i \\ &= \boldsymbol{n}_i^T\mathbf{T}_j\tilde{\boldsymbol{p}}_{ijk} + d_j = \boldsymbol{\pi}_i^T\tilde{\boldsymbol{p}}_{ijk}^g. \end{aligned} \quad (10)$$

The PA problem is to jointly adjust the M planes $\{\boldsymbol{\pi}_i\}$ and the N sensor poses $\{\boldsymbol{x}_j\}$ to minimize the sum of squared point-to-plane distances. Specifically, using (10), we can formulate the cost function of the PA problem as

$$\min_{\{\boldsymbol{\pi}_i\},\{\mathbf{x}_j\}} \sum_{i=1}^M \underbrace{\sum_{j \in obs(\boldsymbol{\pi}_i)} \sum_{k=1}^{N_{ij}} d_{ijk}^2(\boldsymbol{\pi}_i, \mathbf{x}_j)}_{C_i(\boldsymbol{\pi}_i, \mathbb{X}_i),\; \mathbb{X}_i = \{\boldsymbol{x}_j | j \in obs(\boldsymbol{\pi}_i)\}} = \min_{\{\boldsymbol{\pi}_i\},\{\boldsymbol{x}_j\}} \sum_{i=1}^M C_i(\boldsymbol{\pi}_i, \mathbb{X}_i). \quad (11)$$

where $obs(\boldsymbol{\pi}_i)$ represents the indexes of poses where $\boldsymbol{\pi}_i$ can be observed, and $C_i(\boldsymbol{\pi}_i, \mathbb{X}_i)$ accumulates the errors from $N_i = \sum_{j \in obs(\boldsymbol{\pi}_i)} N_{ij}$ points captured at the poses in $\mathbb{X}_i$. According to (6) and (9), we get

$$\mathbf{M}_i(\mathbb{X}_i) = \sum_{j \in obs(\boldsymbol{\pi}_i)} \mathbf{S}_{ij} - N_i \bar{\boldsymbol{p}}_i \bar{\boldsymbol{p}}_i^T, \tag{12}$$

where $\bar{\boldsymbol{p}}_i = \frac{1}{N_i} \sum_{j \in obs(\boldsymbol{\pi}_i)} \sum_{k=1}^{N_{ij}} \boldsymbol{p}_{ijk}^g$ and $\mathbf{S}_{ij} = \sum_{k=1}^{N_{ij}} \boldsymbol{p}_{ijk}^g \left(\boldsymbol{p}_{ijk}^g\right)^T$. Here the elements in $\mathbf{M}_i$, $\mathbf{S}_{ij}$ and $\bar{\boldsymbol{p}}_i$ in (12) are all functions of the poses in $\mathbb{X}_i$. Substituting $\boldsymbol{p}_{ijk}^g$ in (9) into $\mathbf{S}_{ij}$ and $\bar{\boldsymbol{p}}_i$ in (12), we have

$$\begin{aligned} \mathbf{S}_{ij} &= \mathbf{T}_j \underbrace{\sum_{k=1}^{N_{ij}} \tilde{\boldsymbol{p}}_{ijk} \tilde{\boldsymbol{p}}_{ijk}^T}_{\mathbf{U}_{ij}} \mathbf{T}_j^T = \mathbf{T}_j \mathbf{U}_{ij} \mathbf{T}_j^T, \\ \bar{\boldsymbol{p}}_i &= \frac{1}{N_i} \sum_{j \in obs(\boldsymbol{\pi}_i)} \mathbf{T}_j \underbrace{\sum_{k=1}^{N_{ij}} \tilde{\boldsymbol{p}}_{ijk}}_{\tilde{\boldsymbol{p}}_{ij}} = \frac{1}{N_i} \sum_{j \in obs(\boldsymbol{\pi}_i)} \mathbf{T}_j \tilde{\boldsymbol{p}}_{ij}. \end{aligned} \tag{13}$$

Here $\mathbf{U}_{ij}$ and $\tilde{\boldsymbol{p}}_{ij}$ in (13) are constants. We only need to compute them once, and reuse them in the iteration.

According to (7), given poses in $\mathbb{X}_i$, the optimal solution for $\boldsymbol{\pi}_i$ has a closed-form expression $\hat{\boldsymbol{\pi}}_i = [\hat{\boldsymbol{n}}_i; \hat{d}_i]$, where $\hat{\boldsymbol{n}}_i = \boldsymbol{\xi}_3(\mathbf{M}_i(\mathbb{X}_i))$ and $\hat{d}_i = -\hat{\boldsymbol{n}} \bar{\boldsymbol{p}}_i$. As $\mathbf{M}_i$ and $\bar{\boldsymbol{p}}_i$ are functions of the poses in $\mathbb{X}_i$, $\hat{\boldsymbol{\pi}}_i$ is also a function of the poses in $\mathbb{X}_i$. That is to say $\hat{\boldsymbol{\pi}}_i$ is completely determined by the poses in $\mathbb{X}_i$. To simplify the notation, let us define

$$\lambda_{i,3}(\mathbb{X}_i) = \lambda_3(\mathbf{M}_i(\mathbb{X}_i)), \tag{14}$$

which represents the smallest eigenvalue of $\mathbf{M}_i(\mathbb{X}_i)$.

Substituting the optimal plane estimation $\hat{\boldsymbol{\pi}}_i$ into $C_i(\boldsymbol{\pi}_i, \mathbb{X}_i)$ in (11) and using (8), we have

$$\lambda_{i,3}(\mathbb{X}_i) = C_i\left(\hat{\boldsymbol{\pi}}_i, \mathbb{X}_i\right). \tag{15}$$

Using (15), we can formulate the PA problem in (11) as

$$\{\hat{\mathbf{x}}_j\} = \arg\min_{\{\mathbf{X}_j\}} \boldsymbol{\tau}, \ \boldsymbol{\tau} = \sum_{i=1}^{M} \lambda_{i,3}(\mathbb{X}_i). \tag{16}$$

Table 1 summarizes the notations for PA.

The cost function (16) only depends on poses, which significantly reduces the number of variables. However, as it is the sum of squared point-to-plane distances, we cannot apply the Gauss-Newton-based methods to minimize it, where the Jacobian matrix of residuals are required. Here we adopt the Newton's method to solve it. The crux for applying the Newton's method to minimize (16) is how to compute the gradient and the Hessian matrix of (16) efficiently. In the following sections, we provide a closed-form solution for them. Note that we can assign a weight to each point to punish outliers, which also has a close-form solution for planes using the **weighted PCA**. Our algorithm can be extended to this situation. To simplify the notation, we omit the variables of functions in the following description (*e.g.*, $\lambda_{i,3}(\mathbb{X}_i) \to \lambda_{i,3}$).

Notation	Description
$\boldsymbol{\pi}_i$	The ith plane.
$obs(\boldsymbol{\pi}_i)$	The set of indexes of poses which can see $\boldsymbol{\pi}_i$.
$\mathbb{X}_i$	The set of poses which can see $\boldsymbol{\pi}_i$.
$\mathbf{M}_i$	The scatter matrix for $\boldsymbol{\pi}_i$.
$\lambda_{3,i}$	The smallest eigenvalue of $\mathbf{M}_i$.
$\boldsymbol{x}_j$, $\boldsymbol{x}_k$	The jth and kth poses.
x_{jm}, x_{kn}	The mth and nth elements of $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$.
$\mathbb{P}_j$	The set of planes that are visible to $\boldsymbol{x}_j$.
$\mathbb{P}_{jk}$	The set of planes that are visible to $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$.

Table 1. Table of notations.

4. Newton's Iteration for Plane Adjustment

Let us denote the gradient and the Hessian matrix of τ in (16) as $\boldsymbol{g}$ and $\mathbf{H}$, and denote the 6-dimensional gradient vector for $\boldsymbol{x}_j$ as $\boldsymbol{g}_j$ and the 6×6 Hessian matrix block for $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ as $\mathbf{H}_{jk}$ (note that here j can equal to k). Then $\boldsymbol{g}$ and $\mathbf{H}$ can be written in the block form as $\boldsymbol{g} = (\boldsymbol{g}_j) \in \mathbb{R}^{6N}$ and $\mathbf{H} = (\mathbf{H}_{jk}) \in \mathbb{R}^{6N \times 6N}$.

The ith plane $\boldsymbol{\pi}_i$ is observed by the poses in $\mathbb{X}_i$. Assume $\boldsymbol{x}_j \in \mathbb{X}_i$ and $\boldsymbol{x}_k \in \mathbb{X}_i$. Let us define

$$\boldsymbol{g}_j^i = \frac{\partial \lambda_{i,3}}{\partial \boldsymbol{x}_j}, \ \mathbf{H}_{jk}^i = \frac{\partial^2 \lambda_{i,3}}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_k}. \tag{17}$$

According to (16), we have

$$\boldsymbol{g}_j = \sum_{i \in \mathbb{P}_j} \boldsymbol{g}_j^i, \ \mathbf{H}_{jk} = \sum_{i \in \mathbb{P}_{jk}} \mathbf{H}_{jk}^i, \tag{18}$$

where $\mathbb{P}_j$ is the set of planes observed by $\boldsymbol{x}_j$, and $\mathbb{P}_{jk}$ is the set of planes observed by $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ simultaneously. If $j = k$, here $\mathbb{P}_{jk}$ equals to $\mathbb{P}_j$. From (18), we know that the key point to get $\boldsymbol{g}$ and $\mathbf{H}$ is to compute $\boldsymbol{g}_j^i$ and $\mathbf{H}_{jk}^i$ in (17).

4.1. Partial Derivatives of Eigenvalue

According to (15), $\lambda_{i,3}$ is a function of poses in $\mathbb{X}_i$. Assume that x_{jm} and x_{kn} are the mth and nth elements of $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, respectively. We consider the first- and second-order partial derivation $\frac{\partial \lambda_{i,3}}{\partial x_{jm}}$ and $\frac{\partial^2 \lambda_{i,3}}{\partial x_{jm} \partial x_{kn}}$, respectively.

$\lambda_{i,3}$ is a root of the equation $|\mathbf{M}_i(\mathbb{X}_i) - \lambda_i \mathbf{I}_3| = 0$, where $|\cdot|$ denotes the determinant of a matrix. Assume m_{ef} is the eth row fth column term of $\mathbf{M}_i(\mathbb{X}_i)$. $|\mathbf{M}_i(\mathbb{X}_i) - \lambda_i \mathbf{I}_3| = 0$ is a cubic equation with the following form

$$-\lambda_{i,3}^3 + a_i \lambda_{i,3}^2 + b_i \lambda_{i,3} + c_i = 0, \tag{19}$$

where $a_i = m_{11} + m_{22} + m_{33}$, $b_i = m_{12}^2 + m_{13}^2 + m_{23}^2 - m_{11}m_{22} - m_{11}m_{33} - m_{22}m_{33}$, and $c_i = -m_{33}m_{12}^2 + 2m_{12}m_{13}m_{23} - m_{22}m_{13}^2 - m_{11}m_{23}^2 + m_{11}m_{22}m_{33}$. Here a_i, b_i and c_i are all functions of the poses in $\mathbb{X}_i$. It is known that the root of a cubic equation has a closed form. One solution to compute $\frac{\partial \lambda_{i,3}}{\partial x_{jm}}$ and $\frac{\partial^2 \lambda_{i,3}}{\partial x_{jm}\partial x_{kn}}$ is to directly differentiate the root. However, the formula of the root is too complicated. Here we introduce a simple way to compute them. Briefly, we employ the implicit function theorem [15] to compute them. Let us define

$$\boldsymbol{\chi}_i = \begin{bmatrix} \lambda_{i,3}^2 \\ \lambda_{i,3} \\ 1 \end{bmatrix}, \boldsymbol{\eta}_i = \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} \boldsymbol{\kappa}_i = \begin{bmatrix} -3 \\ 2a_i \\ b_i \end{bmatrix}, \boldsymbol{\delta}_{jm}^i = \frac{\partial \boldsymbol{\eta}_i}{\partial x_{jm}}. \tag{20}$$

Using the above notations, we present $\frac{\partial \lambda_{i,3}}{\partial x_{jm}}$ and $\frac{\partial^2 \lambda_{i,3}}{\partial x_{jm}\partial x_{kn}}$ in Lemma 1 and 2. The proofs of the following lemmas and theorems are in the **supplementary material**.

Lemma 1 *Using the notations in (20), we have*

$$\frac{\partial \lambda_{i,3}}{\partial x_{jm}} = -\varphi_i \boldsymbol{\delta}_{jm}^i \cdot \boldsymbol{\chi}_i, \tag{21}$$

where $\cdot$ represents the dot product and $\varphi_i = (\boldsymbol{\kappa}_i \cdot \boldsymbol{\chi}_i)^{-1}$.

Lemma 2 *Using the notations in (20) and (21), we have*

$$\frac{\partial^2 \lambda_{i,3}}{\partial x_{jm}\partial x_{kn}} = -\varphi_i \left(\boldsymbol{\delta}_{jm}^i \cdot \frac{\partial \boldsymbol{\chi}_i}{\partial x_{kn}} + \boldsymbol{\chi}_i \cdot \frac{\partial \boldsymbol{\delta}_{jm}^i}{\partial x_{kn}} - \frac{\partial \lambda_{i,3}}{\partial x_{jm}} \frac{\partial \varphi_i^{-1}}{\partial x_{kn}} \right). \tag{22}$$

Let us define

$$\begin{aligned} \boldsymbol{\alpha}_j^i &= \frac{\partial a_i}{\partial \boldsymbol{x}_j}, \boldsymbol{\beta}_j^i = \frac{\partial b_i}{\partial \boldsymbol{x}_j}, \boldsymbol{\gamma}_j^i = \frac{\partial c_i}{\partial \boldsymbol{x}_j}, \boldsymbol{\Delta}_j^i = [\boldsymbol{\alpha}_j^i, \boldsymbol{\beta}_j^i, \boldsymbol{\gamma}_j^i], \\ \boldsymbol{\alpha}_k^i &= \frac{\partial a_i}{\partial \boldsymbol{x}_k}, \boldsymbol{\beta}_k^i = \frac{\partial b_i}{\partial \boldsymbol{x}_k}, \boldsymbol{\gamma}_k^i = \frac{\partial c_i}{\partial \boldsymbol{x}_k}, \boldsymbol{\Delta}_k^i = [\boldsymbol{\alpha}_k^i, \boldsymbol{\beta}_k^i, \boldsymbol{\gamma}_k^i], \\ \mathbf{H}_{jk}^{a_i} &= \frac{\partial^2 a_i}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_k}, \mathbf{H}_{jk}^{b_i} = \frac{\partial^2 b_i}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_k}, \mathbf{H}_{jk}^{c_i} = \frac{\partial^2 c_i}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_k}. \end{aligned} \tag{23}$$

Using the above lemmas and notations, we can derive $\boldsymbol{g}_j^i$ and $\mathbf{H}_{jk}^i$.

Theorem 1 *Using the notations in (20), (21) and (23), $\boldsymbol{g}_j^i$ and $\mathbf{H}_{jk}^i$ have the forms*

$$\begin{aligned} \boldsymbol{g}_j^i &= -\varphi_i \boldsymbol{\Delta}_j^i \boldsymbol{\chi}_i, \\ \mathbf{H}_{jk}^i &= \varphi_i \left(\mathbf{K}_{jk}^i - \lambda_{3,i}^2 \mathbf{H}_{jk}^{a_i} - \lambda_{3,i} \mathbf{H}_{jk}^{b_i} - \mathbf{H}_{jk}^{c_i} \right), \end{aligned} \tag{24}$$

where $\mathbf{K}_{jk}^i = \boldsymbol{g}_j^i \boldsymbol{u}^T - \boldsymbol{v}(\boldsymbol{g}_k^i)^T$, $\boldsymbol{u} = 2\lambda_{i,3}\boldsymbol{\alpha}_k^i + \boldsymbol{\beta}_k^i + (2a - 6\lambda_{i,3})\boldsymbol{g}_k^i$, and $\boldsymbol{v} = 2\lambda_{i,3}\boldsymbol{\alpha}_j^i + \boldsymbol{\beta}_j^i$, and similar to $\boldsymbol{g}_j^i$, $\boldsymbol{g}_k^i = -\varphi_i \boldsymbol{\Delta}_k^i \boldsymbol{\chi}_i$ is the gradient block for $\boldsymbol{x}_k$.

The formula of $\mathbf{H}_{jk}^i$ in (24) is applicable to the case that $j = k$. From Theorem 1, we know that the key point to get $\boldsymbol{g}_j^i$ and $\mathbf{H}_{jk}^i$ is to get the derivatives of a_i, b_i and c_i in (23).

4.2. Partial Derivatives of a_i, b_i and c_i

According to (19), a_i, b_i and c_i are functions of the elements in $\mathbf{M}_i$. Using this relationship, we can easily derive the partial derivatives in (23). For instance, as $a_i = m_{11} + m_{22} + m_{33}$, we have

$$\frac{\partial a_i}{\partial \boldsymbol{x}_j} = \frac{\partial m_{11}}{\partial \boldsymbol{x}_j} + \frac{\partial m_{22}}{\partial \boldsymbol{x}_j} + \frac{\partial m_{33}}{\partial \boldsymbol{x}_j}. \tag{25}$$

Thus, to get the first- and second-order partial derivatives of a_i, b_i and c_i with respect to $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ in (23), we need to derive the form of $\mathbf{M}_i$ with respect to $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$.

Lemma 3 *In terms of $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, $\bar{\boldsymbol{p}}_i$ in (13) has the form*

$$\bar{\boldsymbol{p}}_i(\boldsymbol{x}_j, \boldsymbol{x}_k) = \mathbf{T}_j \boldsymbol{q}_{ij} + \mathbf{T}_k \boldsymbol{q}_{ik} + \boldsymbol{c}_{ijk}, \tag{26}$$

where $\boldsymbol{q}_{ij} = \frac{1}{N_i}\tilde{\boldsymbol{p}}_{ij}$, $\boldsymbol{q}_{ik} = \frac{1}{N_i}\tilde{\boldsymbol{p}}_{ik}$, and $\boldsymbol{c}_{ijk} = \frac{1}{N_i}\sum_{n\in\mathbb{O}_{jk}} \mathbf{T}_n \tilde{\boldsymbol{p}}_{in}$. Here $\mathbb{O}_{jk} = obs(\boldsymbol{\pi}_i) - \{j, k\}$ represents the set of indexes of the poses that can observe $\boldsymbol{\pi}_i$, excluding the $\boldsymbol{j}$th and $\boldsymbol{k}$th poses.

In terms of $\boldsymbol{x}_j$, $\bar{\boldsymbol{p}}_i$ has the form

$$\bar{\boldsymbol{p}}_i(\boldsymbol{x}_j) = \mathbf{T}_j \mathbf{q}_{ij} + \boldsymbol{c}_{ij}, \tag{27}$$

where $\boldsymbol{c}_{ij} = \mathbf{T}_k \mathbf{q}_{ik} + \boldsymbol{c}_{ijk}$.

Using Lemma 3, we can have the following theorem for $\mathbf{M}_i$ in (12).

Theorem 2 *In terms of $\boldsymbol{x}_j$, $\mathbf{M}_i$ in (12) can be written as*

$$\mathbf{M}_i(\boldsymbol{x}_j) = \mathbf{T}_j \mathbf{Q}_j^i \mathbf{T}_j^T + \mathbf{T}_j \mathbf{K}_j^i + (\mathbf{K}_j^i)^T \mathbf{T}_j^T + \mathbf{C}_j^i, \tag{28}$$

where $\mathbf{Q}_j^i = \mathbf{U}_{ij} - N_j \boldsymbol{q}_{ij}\boldsymbol{q}_{ij}^T$ and $\mathbf{K}_j^i = -N_i \boldsymbol{q}_{ij}\boldsymbol{c}_{ij}^T$. Here $\mathbf{U}_{ij}$ and $\boldsymbol{q}_{ij}$ are defined in (13) and (26), respectively.

In terms of $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, $\mathbf{M}_i$ can be written as

$$\mathbf{M}_i(\boldsymbol{x}_j, \boldsymbol{x}_k) = \mathbf{T}_j \mathbf{O}_{jk}^i \mathbf{T}_k^T + \mathbf{T}_k (\mathbf{O}_{jk}^i)^T \mathbf{T}_j^T + \mathbf{C}_{jk}^i. \tag{29}$$

where $\mathbf{O}_{jk}^i = -N_i \boldsymbol{q}_{ij}\boldsymbol{q}_{ik}^T$.

Here we do not provide the detailed formulas for $\mathbf{C}_j^i$ and $\mathbf{C}_{jk}^i$, as they will be eliminated in the partial derivative. Actually, only $\mathbf{Q}_j^i$, $\mathbf{K}_j^i$, and $\mathbf{O}_{jk}^i$ are required to compute the partial derivatives in (23). Equation (28) is used to compute the first- and second-order partial derivatives of a_i, b_i and c_i with respect to $\boldsymbol{x}_j$. Equation (29) is used to compute the second-order partial derivatives of a_i, b_i and c_i with respect to $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$.

4.3. Efficient Iteration

From Theorem 2, we can easily derive the elements of $\mathbf{M}_i(\boldsymbol{x}_j)$ and $\mathbf{M}_i(\boldsymbol{x}_j, \boldsymbol{x}_i)$. Assume $a(\boldsymbol{x}_j)$ and $b(\boldsymbol{x}_j, \boldsymbol{x}_k)$ are one of the elements in $\mathbf{M}_i(\boldsymbol{x}_j)$ and $\mathbf{M}_i(\boldsymbol{x}_j, \boldsymbol{x}_k)$, respectively. Substituting $\mathbf{T}_j$ and $\mathbf{T}_k$ defined in (1) into (28) and

(29) and expanding them, we can obtain that $a(\boldsymbol{x}_j)$ and $b(\boldsymbol{x}_j, \boldsymbol{x}_k)$ are second-order polynomials in terms of the elements in $\mathbf{T}_j$ and $\mathbf{T}_k$. Using the CGB parameterization in (2), we can write $a(\boldsymbol{x}_j)$ and $b(\boldsymbol{x}_j, \boldsymbol{x}_k)$ as

$$\begin{aligned} a(\boldsymbol{x}_j) &= \boldsymbol{c} \cdot \boldsymbol{h}(\boldsymbol{x}_j) + c_0, \\ b(\boldsymbol{x}_j, \boldsymbol{x}_k) &= \boldsymbol{d} \cdot \boldsymbol{g}(\boldsymbol{x}_j, \boldsymbol{x}_k) + d_0, \end{aligned} \tag{30}$$

where $\boldsymbol{c}$ is determined by $\mathbf{Q}_j^i$ and $\mathbf{K}_j^i$ in (28), $\boldsymbol{d}$ is determined by $\mathbf{O}_{jk}^i$ in (29), $\boldsymbol{h}(\boldsymbol{x}_j)$ and $\boldsymbol{g}(\boldsymbol{x}_j, \boldsymbol{x}_k)$ are two vector functions, c_0 and d_0 are two scalars independent on $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$. Let us first consider the first-order partial derivative of $a(\boldsymbol{x}_j)$ with respect to $\boldsymbol{x}_j$. It has the form

$$\frac{\partial a(\boldsymbol{x}_j)}{\partial \boldsymbol{x}_j} = \frac{\partial \boldsymbol{h}(\boldsymbol{x}_j)}{\partial \boldsymbol{x}_j} \boldsymbol{c}, \tag{31}$$

where the vector-by-vector derivative $\frac{\partial \boldsymbol{h}(\boldsymbol{x}_j)}{\partial \boldsymbol{x}_j}$ is defined in (4). To efficiently compute (31), we consider a special pose $\mathbf{T}_0 = [\mathbf{R}_0, \boldsymbol{t}_0]$ where $\mathbf{R}_0 = \mathbf{I}_3$ and $\boldsymbol{t}_0 = [0; 0; 0]$. Let us denote the parameterization of $\mathbf{T}_0$ as $\boldsymbol{x}_0$. As the CGR parameterization for $\mathbf{I}_3$ is $[0; 0; 0]$, we have $\boldsymbol{x}_0 = [0; 0; 0; 0; 0; 0]$. For $\boldsymbol{x}_j = \boldsymbol{x}_0$, the matrix $\frac{\partial \boldsymbol{h}(\mathbf{x}_j)}{\partial \boldsymbol{x}_j}$ can be easily computed. Similarly, the second-order partial derivatives of $\boldsymbol{h}(\boldsymbol{x}_j)$ and $\boldsymbol{g}(\boldsymbol{x}_j, \boldsymbol{x}_k)$ at $\boldsymbol{x}_j = \boldsymbol{x}_0$ and $\boldsymbol{x}_k = \boldsymbol{x}_0$ are also simple. As $\boldsymbol{h}(\boldsymbol{x}_j)$ and $\boldsymbol{g}(\boldsymbol{x}_j, \boldsymbol{x}_k)$ only depend on $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, we can compute the partial derivatives at $\boldsymbol{x}_0$ once, and then reuse them in the following iterations. Here we introduce a method to make the iteration stay at x_0 for each pose.

Assume that $\{\mathbf{X}_j^n\}$ are the poses after the nth iteration. Then we can update $\mathbf{U}_{ij}$ and $\tilde{\boldsymbol{p}}_{ij}$ in (13) by

$$\mathbf{U}_{ij}^{n+1} = \mathbf{X}_j^n \mathbf{U}_{ij} (\mathbf{X}_j^n)^T \text{ and } \tilde{\boldsymbol{p}}_{ij}^{n+1} = \mathbf{X}_j^n \tilde{\boldsymbol{p}}_{ij}. \tag{32}$$

Substituting $\mathbf{U}_{ij}^{n+1}$ and $\tilde{\boldsymbol{p}}_{ij}^{n+1}$ into (12), we get a new matrix $\mathbf{M}_i(\mathbb{X}_i)^{n+1}$, which can finally lead to a new cost τ^{n+1} in (16). As the points have been transformed by $\{\mathbf{X}_j^n\}$, each pose should start with $\mathbf{X}_0$ for τ^{n+1}. Assume that $\Delta \boldsymbol{x}_j^{n+1}$ is the result from the $(n+1)$th iteration for the jth pose. We can compute the corresponding transformation matrix $\Delta \mathbf{X}_j^{n+1}$ using (1) and (2). Then we can update $\mathbf{X}_j^n$ by

$$\mathbf{X}_j^{n+1} = \Delta \mathbf{X}_j^{n+1} \mathbf{X}_j^n. \tag{33}$$

Furthermore, the update steps in (32) will not introduce additional computation. This is because the damped Newton's method requires to compute the cost τ in (16) to adjust μ in (3) after each iteration, which requires to perform the computation in (32).

4.4. Algorithm Summary

We first construct $\mathbf{H}$ and $\boldsymbol{g}$. For each plane $\boldsymbol{\pi}_i$, we solve the cubic equation (19), and select the smallest root $\lambda_{i,3}$. For $\boldsymbol{x}_j$, we construct $\mathbf{M}(\boldsymbol{x}_j)$ in (28), and calculate the partial derivatives of a_i, b_i and c_i with respect to $\boldsymbol{x}_j$ in (23). Then, we use (24) to compute $\boldsymbol{g}_j^i$ and $\mathbf{H}_{jj}^i$ and use (18) to update $\boldsymbol{g}_j$ and $\mathbf{H}_{jj}$. For $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, $\mathbf{M}_i(\boldsymbol{x}_j, \boldsymbol{x}_k)$ is generated, and then the partial derivatives of a_i, b_i and c_i with respect to $\boldsymbol{x}_j$ and $\boldsymbol{x}_j$ in (23) are computed. Then, $\mathbf{H}_{jk}^i$ can be easily obtained from (24), and $\mathbf{H}_{jk}$ in (18) is computed accordingly. Using $\mathbf{H}$ and $\boldsymbol{g}$, we conduct the damped Newton's step in (3). After each iteration, $\mathbf{U}_{ij}$ and $\tilde{\boldsymbol{p}}_{ij}$ are updated by (32). The proposed algorithm is summarized in Algorithm 1. Let us denote the mean and variance of the number of observations per plane as $\bar{K}$ and σ^2, respectively. According to [8], the computational complexity of the Hessian matrix is $O(M(\bar{K}^2 + \sigma^2))$, which is of the same order as the Schur complement trick.

Algorithm 1: Second-order plane adjustment for N poses and M planes

```
while not converge do
    H = zeros(6N, 6N), g = zeros(6N, 1);
    for i ∈ [1, M] do
        /* Compute g and the diagonal
           terms of H.                          */
        for j ∈ obs(π_i) do
            Compute M_i(x_j) using (28);
            Compute α_j^i, β_j^i, γ_j^i, H_jj^{a_i}, H_jj^{b_i}, H_jj^{c_i}
             using (23);
            Compute H_jj^i and g_j^i using (24);
            H_jj = H_jj + H_jj^i, g_j = g_j + g_j^i;
        /* Compute other terms of H.            */
        for j ∈ obs(π_i) do
            for k ∈ obs(π_i) and k > j do
                Compute H_jk^i using (24);
                H_jk = H_jk + H_jk^i;
                H_kj = H_kj + (H_jk^i)^T;
    Conduct the damped Newton's step in (3) ;
    Update U_ij and p̃_ij using (32);
```

5. Experiments

5.1. Setup

In this section, we compare our algorithm with **EF** [10], **BALM** [16] and the LM solution [23] with plane fitting after a successful LM step (named **LM+PF**). Our damped Newton's method was implemented according to the LM algorithm in Ceres [2]. The damped Newton's method and the LM algorithm are with the same parameters. Specifically, the initial value of the damping factor μ in (3) is set to 10^{-4}. The early stopping tolerances (such as the cost value change and the norm of gradient) are set to 10^{-7}. The

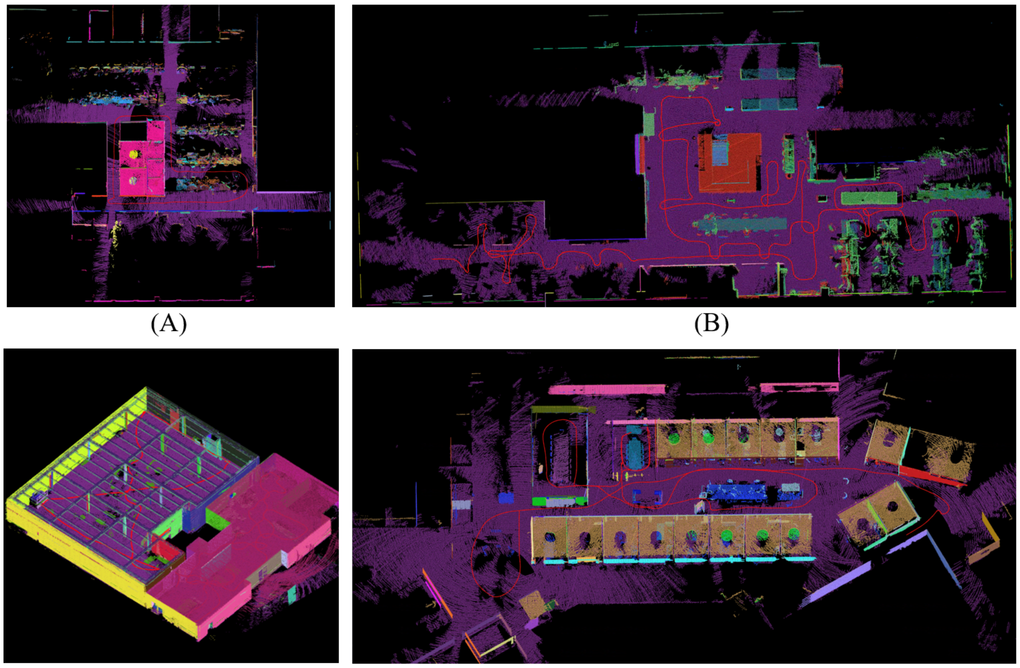


Figure 3. The four datasets used in this paper. There are 4.5×10^6, 16.6×10^6, 16.7×10^6, and 10.5×10^6 points in the 4 datasets. The 4 datasets have 339, 369, 856, and 589 planes, and 472, 1355, 1606, 1184 poses, respectively. Roofs are removed to show the trajectories.

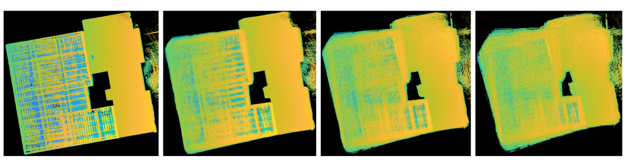


Figure 4. The point clouds of dataset C after the poses were perturbs by the four noise levels.

maximum number of iterations is set to 200 for our algorithm, BALM, and LM+PF, and 10^5 for EF, as EF uses the first-order minimization which requires more iterations to converge. All the experiments were conducted on a desktop with an Intel i7 cpu and 64G memory.

5.2. Datasets

We collected four datasets using a VLP-16 LiDAR. We used the LiDAR SLAM algorithm [24] to detect the planes and establish the plane association. Fig. 3 shows the four datasets. Similar to the evaluation of BA algorithms [3, 8, 25], we perturb the pose, and compare the PA cost in (11) for different algorithms. Specifically, we directly add Gaussian noises to the translation, and randomly generate an angle-axis vector from a Gaussian distribution to perturb the rotation. After the poses are perturb, we use (7) to get the initial plane parameters for LM+PF.

We evaluate the performance of different algorithms under different noise levels. Let us denote the standard deviation (std) of the Gaussian noises for rotation and translation as $\sigma_{\mathbf{R}}$ and $\sigma_{\boldsymbol{t}}$, respectively. We consider four noise levels: $\sigma_{\mathbf{R}} = 0.1^\circ$ and $\sigma_{\boldsymbol{t}} = 0.01m$, $\sigma_{\mathbf{R}} = 1^\circ$ and $\sigma_{\boldsymbol{t}} = 0.1m$, $\sigma_{\mathbf{R}} = 2^\circ$ and $\sigma_{\boldsymbol{t}} = 0.2m$, and $\sigma_{\mathbf{R}} = 3^\circ$ and $\sigma_{\boldsymbol{t}} = 0.3m$. Fig. 4 demonstrates the point clouds of dataset C after the poses are perturbed by the four noise levels. To evaluate the performance of different algorithms suffering from large measurement noises, we also test the case that additional Gaussian noises with the std $\sigma_{pt} = 0.05m$ are added to the LiDAR point cloud, and the poses are perturbed by Gaussian noises with $\sigma_{\mathbf{R}} = 3^\circ$ and $\sigma_{\boldsymbol{t}} = 0.3m$.

5.3. Results

Fig. 5 and Fig. 6 illustrates the results. It is clear that our algorithm converges faster than other algorithms. LM+PF works well at small noises (such as $\sigma_{\mathbf{R}} = 0.1^\circ$ and $\sigma_{\boldsymbol{t}} = 0.01m$). As the noise level increases, LM+PF tends to converge slower. Constructing $\mathbf{H}$ and $\mathbf{g}$ using BALM [16] is computationally demanding. For efficiency, BALM only keeps the centroid of each plane observation (*i.e.*, only 1 point is kept), which results in bad performance. In Fig. 5 and Fig. 6, all points are used to compute the cost in (11) after an iteration of BALM. BALM generally converges slow, as it minimizes a reduced PA problem. EF [10] adopts the first-order optimization method as it only provides a method

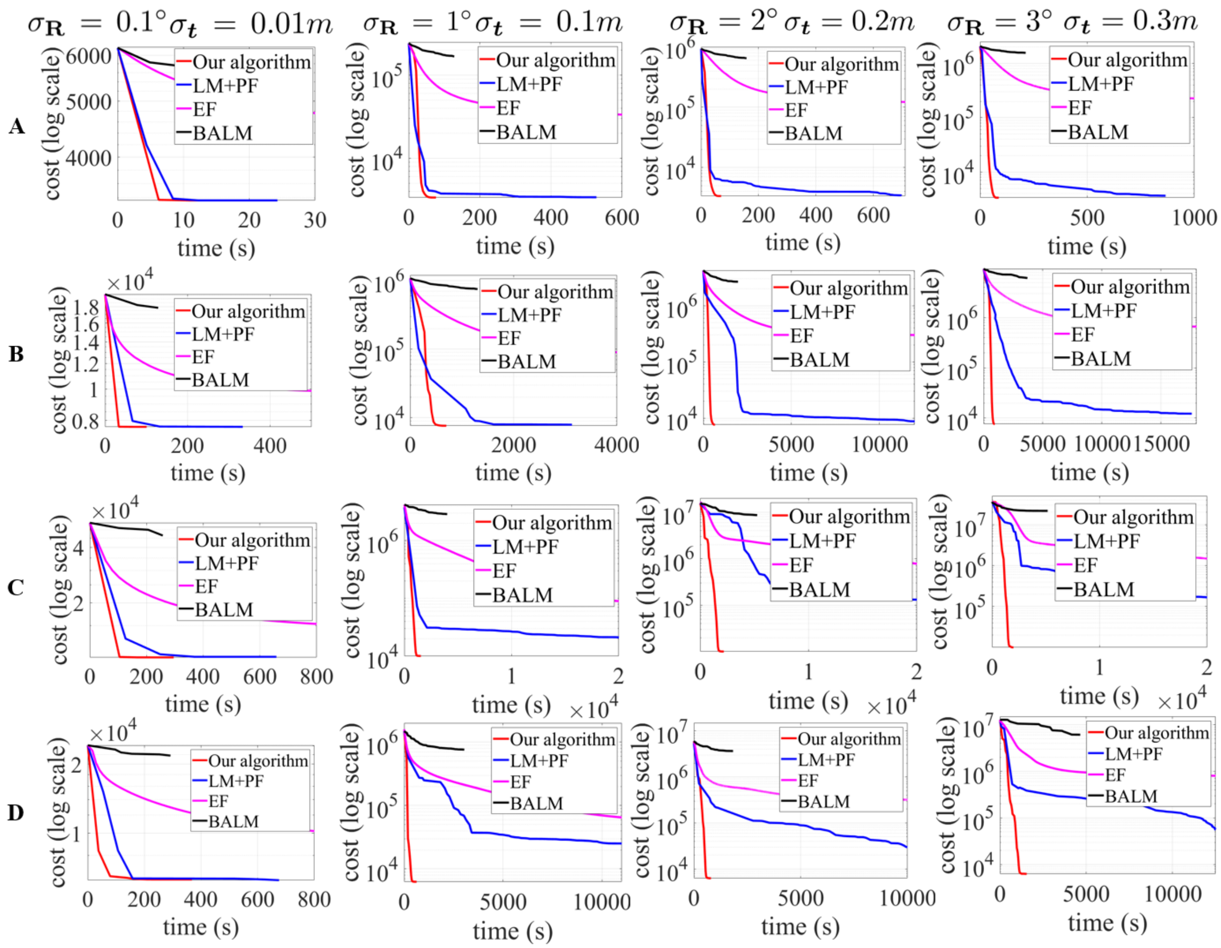


Figure 5. The results of different algorithms using different initial noise levels. It is clear that our algorithm converges significantly faster than other algorithms. The *y*-axis represents the logarithmic scale of the cost in (11).

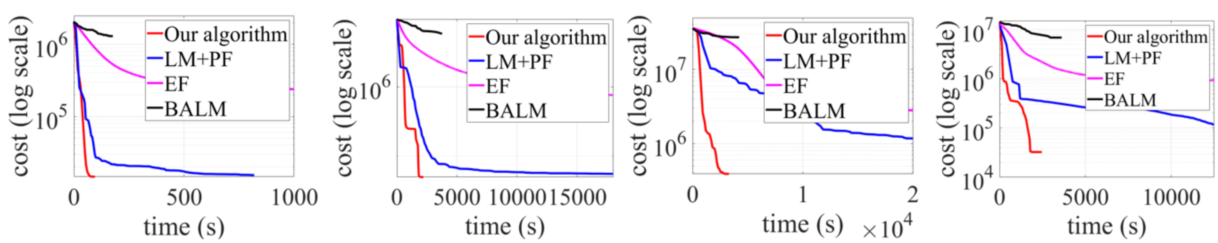


Figure 6. The results of different algorithms for point clouds perturbed by Gaussian noise with std $\sigma_{pt} = 0.05m$. The rotations and translations of the poses are perturbed by Gaussian noise with std$\sigma_{\mathbf{R}} = 3^\circ$ and $\sigma_{\boldsymbol{t}} = 0.3m$, respectively.

to compute the gradient, which results in slow convergence.

6. Conclusion

In the computer vision community, Newton's method is generally considered too expensive for a large-scale least-squares problem. This paper adopts the Newton's method to efficiently solve the PA problem. Our algorithm takes advantage of the fact that the optimal planes are determined by the poses, so that the number of unknowns can be significantly reduced. Furthermore, this property can ensure to obtain the optimal planes when we update the poses. The difficulty lies in how to efficiently compute the Hessian matrix and the gradient vector. The key contribution of this paper is to provide a closed-form solution for them. The experimental results show that our algorithm outperforms the state-of-the-art algorithms.

References

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. 1

[2] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 3 2022. 6

[3] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010. 1, 2, 7

[4] Alex M Andrew. Multiple view geometry in computer vision. *Kybernetes*, 2001. 2

[5] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 2021. 1

[6] Danpeng Chen, Shuai Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Hujun Bao, and Guofeng Zhang. Vip-slam: An efficient tightly-coupled rgb-d visual inertial planar slam. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5615–5621. IEEE, 2022. 1

[7] Nikolaus Demmel, David Schubert, Christiane Sommer, Daniel Cremers, and Vladyslav Usenko. Square root marginalization for sliding-window bundle adjustment. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13260–13268, 2021. 2

[8] Nikolaus Demmel, Christiane Sommer, Daniel Cremers, and Vladyslav Usenko. Square root bundle adjustment for large-scale reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11723–11732, 2021. 1, 2, 6, 7

[9] Hakim Elchaoui Elghor, David Roussel, Fakhreddine Ababsa, and El Houssine Bouyakhf. Planes detection for robust localization and mapping in rgb-d slam systems. In *2015 International Conference on 3D Vision*, pages 452–459. IEEE, 2015. 1

[10] Gonzalo Ferrer. Eigen-factors: Plane estimation for multi-frame and time-continuous point cloud alignment. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1278–1284. IEEE, 2019. 1, 2, 6, 7

[11] Joel A Hesch and Stergios I Roumeliotis. A direct least-squares (dls) method for pnp. In *2011 International Conference on Computer Vision*, pages 383–390. IEEE, 2011. 3

[12] Ming Hsiao, Eric Westman, Guofeng Zhang, and Michael Kaess. Keyframe-based dense planar slam. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5110–5117. IEEE, 2017. 1, 2

[13] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4605–4611. IEEE, 2015. 1, 2

[14] Pyojin Kim, Brian Coltin, and H Jin Kim. Linear rgb-d slam for planar environments. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 333–348, 2018. 1

[15] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002. 5

[16] Zheng Liu and Fu Zhang. BALM: Bundle Adjustment for LiDAR Mapping. *IEEE Robotics and Automation Letters*, 6(2):3184–3191, 2021. 1, 2, 6, 7

[17] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978. 3

[18] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 1

[19] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 1

[20] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. 1, 2, 3

[21] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014. 1

[22] Lipu Zhou, Guoquan Huang, Yinian Mao, Jincheng Yu, Shengze Wang, and Michael Kaess. Plc-lislam: Lidar slam with planes, lines and cylinders. *IEEE Robotics and Automation Letters*, 2022. 1

[23] Lipu Zhou, Daniel Koppel, Hul Ju, Frank Steinbruecker, and Michael Kaess. An efficient planar bundle adjustment algorithm. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 136–145, 2020. 1, 2, 6

[24] Lipu Zhou, Daniel Koppel, and Michael Kaess. Lidar slam with plane adjustment for indoor environment. *IEEE Robotics and Automation Letters*, 6(4):7073–7080, 2021. 1, 2, 7

[25] Lei Zhou, Zixin Luo, Mingmin Zhen, Tianwei Shen, Shiwei Li, Zhuofei Huang, Tian Fang, and Long Quan. Stochastic bundle adjustment for efficient and scalable 3d reconstruction. In *European Conference on Computer Vision*, pages 364–379. Springer, 2020. 1, 2, 7

[26] Lipu Zhou, Shengze Wang, and Michael Kaess. π-lsam: Lidar smoothing and mapping with planes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5751–5757. IEEE, 2021. 1