Efficient Bundle Adjustment for Coplanar Points and Lines

Lipu Zhou[1], Jiacheng Liu[2], Fengguang Zhai[1], Pan Ai[1], Kefei Ren[1], Yinian Mao[1]
Guoquan Huang[1], Ziyang Meng[2], and Michael Kaess[3]

Abstract— **Bundle adjustment (BA) is a well-studied fundamental problem in the robotics and vision community. In man-made environments, coplanar points and lines are ubiquitous. However, the number of works on bundle adjustment with coplanar points and lines is relatively small. This paper focuses on this special BA problem, referred to as π-BA. For a point or a line on a plane, we derive a new constraint to describe the relationship among two poses and the plane, called π-constraint. We distribute π-constraints into different groups. Each group is called a π-factor. We prove that, with some simple preprocessing, the computational complexity associated with a π-factor in the Levenberg-Marquardt (LM) algorithm is $O(1)$, independent of the number of π-constraints packed into the π-factor. In π-BA, π-factors replace original reprojection errors. One problem is how to divide π-constraints into π-factors. Different strategies may result in different numbers of π-factors, which in turn affects the efficiency. It is difficult to get the optimal division. We present a greedy algorithm to overcome this problem. Experimental results verify that our algorithm can significantly accelerate the computation.**

I. Introduction

Bundle adjustment (BA) is important for visual simultaneous localization and mapping (VSLAM) and structure from motion (SfM). Due to its importance, the BA problem has been intensively studied [1]–[16]. Coplanar points and lines are ubiquitous in man-made scenarios. Recently, a number of works [17]–[27] explore leveraging plane information to improve the accuracy and stability of VSLAM, and show promising results. However, the number of works on BA with coplanar points and lines is relatively small. This paper focuses on this special BA problem, referred to as π-BA, and seeks to provide an efficient solution.

π-BA is a non-linear least-squares (NLLS) problem. The Levenberg-Marquardt (LM) algorithm [28] is generally used to solve the NLLS problem. Given a NLLS problem, the LM algorithm first calculates its Jacobian matrix, and then constructs a linear system to update the current solution. The runtime of the LM algorithm depends on constructing and solving the linear system. The computational complexity of solving the linear system in turn depends on the number of unknowns. The crux of our algorithm is that we present new constraints for π-BA which can significantly reduce the number of unknowns and the runtime of constructing the linear system. In fact, we show that the computational complexity of π-BA can be independent of the number of coplanar points and lines, and only depends on the number of planes and poses. The main contributions of this paper are listed below:

- We introduce new constraints for coplanar points and lines, referred to as π-constraints, which depend on the plane where the coplanar points or lines locate, rather than the points and lines themselves. This can significantly reduce the number of unknowns.
- We present an efficient algorithm to optimize the π-constraints. Specifically, π-constraints are divided into different groups. Each group is called a π-factor. We prove that, with some matrix multiplications as preprocessing, the computational complexity of a π-factor in the LM algorithm is $O(1)$, independent of the number of π-constraints in a π-factor.
- How π-constraints are divided determines the number of π-factors, which impacts on the efficiency. It is difficult to find the division that leads to the minimal number of π-factors. We present a simple greedy algorithm to divide π-constraints into π-factors to construct the cost function of π-BA.

Experimental results show that our algorithm can significantly speed up the computation. Thus, this work can benefit the VSLAM and SfM system using planes.

II. Related Work

Bundle Adjustment BA is a nonlinear least-squares problem with special structure. Triggs *et al.* [1] show that the Schur complement trick can accelerate the computation. This provides the theoretical support to make large-scale SfM [29], [30] and real-time SLAM [31]–[33] using BA feasible. Latter works explore ways to further accelerate BA to handle SfM with growing scale. The Schur complement generates a linear system for camera poses, referred to as reduced camera system (RCS) [12], [13]. Agarwal *et al.* [7] introduce the preconditioned conjugate gradients algorithm to efficiently approximate the solution of this linear system. Zhou *et al.* [12] decompose the RCS approximately inside the LM iterations to speed up BA. In addition, parallel computing has been explored to accelerate BA [10], [11]. Recently, Demmel *et al.* [13] present the square root BA which is mathematically equivalent to the Schur complement, but more numerically stable. This allows for solving a BA

*This work was partially done when Lipu Zhou was with the Robotics Institute at Carnegie Mellon University.

[1]Lipu Zhou, Fengguang Zhai, Pan Ai, Kefei Ren, YiNian Mao, and Guoquan Huang are with the UAV Lab, Meituan, Beijing 100012, China. {zhoulipu, zhaifengguang, aipan02, renkefei, maoyinian, huangguoquan}@meituan.com

[2]Jiacheng Liu and Ziyang Meng are with Department of Precision Instrument, Tsinghua University, Beijing 100084, China. liu-jc18@mails.tsinghua.edu.cn, ziyangmeng@tsinghua.edu.cn

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

problem with single-precision floating-point numbers. Lines and planes widely exist in man-made scenarios. Although BA was originally formulated for points, it can be easily extended for lines [34]–[37]. However, it is not straightforward to introduce planes into BA.

Planes for Visual Reconstruction Coplanar points and lines widely exist in man-made environments. Intuitively, these coplanar constraints can benefit VSLAM. Thus, adding plane constraints into VSLAM has gained increasing attention. To use planes in VSLAM, the first step is to detect planes from an image. In the literature, neural networks [17]–[19], [23], [26], [38] and geometry-based methods [20]–[22], [24], [25], [39] were developed for this purpose. Given detected planes, the next step is to construct a cost function to optimize them. In [22], point-to-plane and line-to-plane regularities are added into the cost function. These extra regularities increase the computational cost. Their latter work [23] accelerates the computation by introducing a new parameterization for coplanar points and lines. In [24], the plane regularity is introduced into DSO [40] for coplanar points. It is known that the images of planar points captured at two poses are related by a homography matrix [41]. This relationship is adopted to construct the cost function for coplanar points [20], [25], [39], [42]. This paper considers both coplanar points and lines. We introduce new constraints for coplanar points and lines, and divides them into different groups to speed up the computation.

Planes for Reconstruction with Depth Sensor Planes are common landmarks used in SLAM with depth sensors, such as LiDAR [43]–[51] and RGB-D camera [52], [53]. Similar to BA, jointly optimizing planes and poses, referred to as plane adjustment (PA) [47], [48], has been intensively studied. Generally, there are two ways to construct the cost function of PA. The first one is based on the transformation between plane parameters [52], [53]. The second one adopts the point-to-plane error [44], [46]–[49], [54]. These algorithms are designed for depth sensor, thus they cannot be applied to the camera.

In summary, $\boldsymbol{\pi}$-BA has many applications. This paper focuses on exploring the special structure of $\boldsymbol{\pi}$-BA to speed up the computation.

III. Notations and Preliminaries

This paper uses italic, boldfaced lowercase and boldfaced uppercase letters to represent scalars, vectors and matrices, respectively.

Pose In this paper, a pose represents a rigid body transformation $\mathbf{T} \in SE(3)$ from a camera coordinate system to the world coordinate system. The rotational and translational components of $\mathbf{T}$ are denoted as $\mathbf{R} \in SO(3)$ and $\boldsymbol{t} \in \mathbb{R}^3$, respectively. We parameterize a pose by $\boldsymbol{x} = [\boldsymbol{\omega}; \boldsymbol{t}]$, where $\boldsymbol{\omega}$ is the angle-axis representation of $\mathbf{R}$.

Back Projection As shown in Fig. 1, the back projections of a pixel $\boldsymbol{p}$ and a 2D line $\boldsymbol{l}$ on the image plane are a 3D line $\boldsymbol{L}$ and a plane $\boldsymbol{\pi}$ passing through the origin of the camera coordinate system [41], respectively. Assume that the camera intrinsic matrix is $\mathbf{K}$. Then the direction of $\boldsymbol{L}$

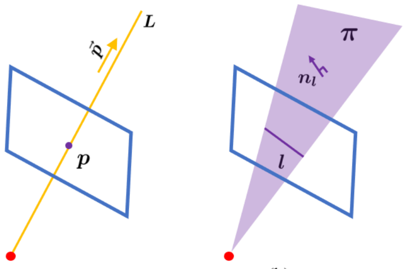


Fig. 1. The back-projections of a pixel $\boldsymbol{p}$ **(a)** and a line $\boldsymbol{l}$ **(b)** are a 3D line $\boldsymbol{L}$ and a plane $\boldsymbol{\pi}$ passing through the camera center, respectively. These relationships will be used to derive the constraints for coplanar points and lines in (4) and (5).

is $\vec{\boldsymbol{p}} = \frac{\mathbf{K}^{-1}\bar{\boldsymbol{p}}}{||\mathbf{K}^{-1}\bar{\boldsymbol{p}}||_2}$, where $\bar{\boldsymbol{p}}$ is the homogeneous coordinates of $\boldsymbol{p}$ (*i.e.*, $\bar{\boldsymbol{p}} = [\boldsymbol{p}; 1]$), and the normal of $\boldsymbol{\pi}$ is $\boldsymbol{n_l} = \frac{\mathbf{K}^T\boldsymbol{l}}{||\mathbf{K}^T\boldsymbol{l}||_2}$. These relationships will be used to derive the constraints from a point and a line on a plane in (4) and (5).

Vector-by-vector Derivative Assume that $\boldsymbol{v} = [v_1, \cdots, v_M]$ is an M-dimensional vector function with respect to an N-dimensional vector $\boldsymbol{\psi} = [\psi_1, \cdots, \psi_N]$. That is to say each element of $\boldsymbol{v}$ is a function of $\boldsymbol{\psi}$. The derivative of $\boldsymbol{v}$ by $\boldsymbol{\psi}$ is an $M \times N$ matrix written as

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\psi}} = [\delta_{ij}] \in \mathbb{R}^{M\times N}, \ \delta_{ij} = \frac{\partial v_i}{\partial \psi_j}, \tag{1}$$

where δ_{ij} is the entry at the ith row and jth column of the matrix. The above formula will be used in (10).

LM Algorithm The LM algorithm [28] is generally adopted to solve a least-squares problem. Given an N-dimensional residual vector $\boldsymbol{e}(\boldsymbol{\chi})$, the corresponding least-squares problem has the form $\arg\min_{\boldsymbol{\chi}} ||\boldsymbol{e}(\boldsymbol{\chi})||_2^2$. To simplify the notation, we omit the variable $\boldsymbol{\chi}$ in the following description (*i.e.*, $\boldsymbol{e}(\boldsymbol{\chi}) \rightarrow \boldsymbol{e}$).

Let us denote the Jacobian matrix of $\boldsymbol{e}$ as $\mathbf{J}_{\boldsymbol{e}}$. The LM algorithm iteratively updates the solution by $\boldsymbol{\chi}_{n+1} = \boldsymbol{\chi}_n + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is computed from the following linear system

$$(\mathbf{J}_{\boldsymbol{e}}^T\mathbf{J}_{\boldsymbol{e}} + \lambda\mathbf{I}_N)\boldsymbol{\delta} = -\mathbf{J}_{\boldsymbol{e}}^T\boldsymbol{e}, \tag{2}$$

where λ is adjusted according to the value of $||\boldsymbol{e}||_2^2$ at the new solution, and $\mathbf{I}_N$ is an identity matrix of size N. Assume that $\boldsymbol{e}$ is divided into different groups. Suppose that $\boldsymbol{e}_i$ is the ith group and $\mathbf{J}_i$ is its corresponding Jacobian matrix, then we have

$$\mathbf{J}_{\boldsymbol{e}}^T\mathbf{J}_{\boldsymbol{e}} = \sum \mathbf{J}_i^T\mathbf{J}_i, \mathbf{J}_{\boldsymbol{e}}^T\boldsymbol{e} = \sum \mathbf{J}_i^T\boldsymbol{e}_i, ||\boldsymbol{e}||_2^2 = \sum \boldsymbol{e}_i^T\boldsymbol{e}_i. \tag{3}$$

In the LM algorithm, $\mathbf{J}_{\boldsymbol{e}}$ is usually computed first. Then $\mathbf{J}_{\boldsymbol{e}}^T\mathbf{J}_{\boldsymbol{e}}$ and $\mathbf{J}_{\boldsymbol{e}}^T\boldsymbol{e}$ are calculated for constructing the equation system (2). Finally, $||\boldsymbol{e}||_2^2$ is computed at the new solution to update λ. The computational complexity of the above process is $O(N)$. **From (2) and (3), we find that $\mathbf{J}_{\boldsymbol{e}}$ and $\boldsymbol{e}$ are not required in the LM algorithm, instead $\mathbf{J}_i^T\mathbf{J}_i$, $\mathbf{J}_i^T\boldsymbol{e}_i$, and $\boldsymbol{e}_i^T\boldsymbol{e}_i$ are essential.** Our algorithm uses this property to speed up the computation. Specifically, we divide constraints into special groups, and show that each group share a lot of computations in each iteration. We accelerate the process by getting rid of the redundant computation.

IV. π-CONSTRAINT

Here we consider the constraint on two poses introduced from a 3D point or a 3D line on a plane, referred to as π-constraint, as demonstrated in Fig. 2. We will show that they have a special form, which can be used to speed up the computation. **The proofs of the following lemmas and theorems are in the Appendix.**

Lemma 1: Suppose that $\boldsymbol{P}$ is a 3D point on a plane $\boldsymbol{\pi} = [\boldsymbol{n}; d]$, and $\boldsymbol{P}$ is observed by two poses $(\mathbf{R}_1, \boldsymbol{t}_1)$ and $(\mathbf{R}_2, \boldsymbol{t}_2)$ with images $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, respectively. Let us denote the directions of the back-projected rays of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ as $\vec{\boldsymbol{p}}_1$ and $\vec{\boldsymbol{p}}_2$, respectively. Then we have

$$\mathbf{R}_1\vec{\boldsymbol{p}}_1 \times (\mathbf{R}_2\vec{\boldsymbol{p}}_2 + \tau_{\boldsymbol{p}_2}(\boldsymbol{t}_2 - \boldsymbol{t}_1)) = \mathbf{0}_{3\times 1}, \tag{4}$$

where $\times$ represents the cross product, $\tau_{\boldsymbol{p}_2} = \frac{\boldsymbol{n}^T\mathbf{R}_2\vec{\boldsymbol{p}}_2}{-\boldsymbol{n}^T\boldsymbol{t}_2 - d}$.

Lemma 2: Suppose that $\boldsymbol{L}$ is a 3D line on a plane $\boldsymbol{\pi} = [\boldsymbol{n}; d]$, and $\boldsymbol{L}$ is observed by two poses $(\mathbf{R}_1, \boldsymbol{t}_1)$ and $(\mathbf{R}_2, \boldsymbol{t}_2)$ with images $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, respectively. Let us denote the normal of the back-projected plane of $\boldsymbol{l}_1$ as $\boldsymbol{n}_{\boldsymbol{l}_1}$, and denote the directions of the back-projected rays of the two endpoints of $\boldsymbol{l}_2$ as $\vec{\boldsymbol{p}}^1_{\boldsymbol{l}_2}$ and $\vec{\boldsymbol{p}}^2_{\boldsymbol{l}_2}$, respectively. Then we have

$$\mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1} \cdot \left(\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \tau^i_{\boldsymbol{l}_2}(\boldsymbol{t}_2 - \boldsymbol{t}_1)\right) = 0, \; i = 1, 2, \tag{5}$$

where $\cdot$ represents the dot product, α is defined in (4) and $\tau^i_{\boldsymbol{l}_2} = \frac{\boldsymbol{n}^T\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2}}{-\boldsymbol{n}^T\boldsymbol{t}_2 - d}$.

The constraints (4) and (5) are functions of $\boldsymbol{\pi}$ in stead of $\boldsymbol{P}$ and $\boldsymbol{L}$. This can obviously reduce the number of unknowns. Fig. 2 illustrates the variables involved in (4) and (5). In the minimization, we parameterize a plane $\boldsymbol{\pi} = [\boldsymbol{n}; d]$ by the closest-point representation $\boldsymbol{\tau} = d\boldsymbol{n}$ [55]. The following theorems show that (4) and (5) have special forms.

Theorem 1: Let us define $\vec{\boldsymbol{p}}_1 = [x_1; y_1; z_1]$ and $\vec{\boldsymbol{p}}_2 = [x_2; y_2; z_2]$, and assume that $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{\tau}$ are the parameterization of $(\mathbf{R}_1, \boldsymbol{t}_1)$, $(\mathbf{R}_2, \boldsymbol{t}_2)$ and $\boldsymbol{\pi}$, respectively. The constraints in (4) can be written as

$$\boldsymbol{c} \cdot \boldsymbol{f}_i(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) = 0, \; i = 1, 2, 3, \tag{6}$$

where $\boldsymbol{c} = [x_1x_2, x_1y_2, x_1z_2, y_1x_2, y_1y_2, y_1z_2, z_1x_2, z_1y_2, z_1z_2]^T$ is a constant vector, and $\boldsymbol{f}_i$ is a vector function of $\boldsymbol{\tau}$, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ where i=1, 2, 3 is for the three constraints in (4).

Theorem 2: Let us define $\boldsymbol{n}_{\boldsymbol{l}_1} = [a_1; b_1; c_1]$ and $\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} = [x^i_{\boldsymbol{l}_2}; y^i_{\boldsymbol{l}_2}; z^i_{\boldsymbol{l}_2}]$. The constraints in (5) can be written as

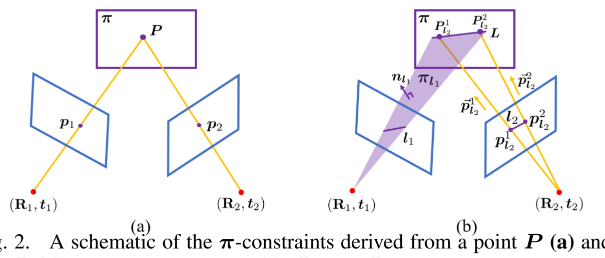
$$\boldsymbol{d}^i \cdot \boldsymbol{g}(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) = 0, \; i = 1, 2, \tag{7}$$

where $\boldsymbol{d}^i = [a_1x^i_{\boldsymbol{l}_2}, a_1y^i_{\boldsymbol{l}_2}, a_1z^i_{\boldsymbol{l}_2}, b_1x^i_{\boldsymbol{l}_2}, b_1y^i_{\boldsymbol{l}_2}, b_1z^i_{\boldsymbol{l}_2}, c_1x^i_{\boldsymbol{l}_2}, c_1y^i_{\boldsymbol{l}_2}, c_1z^i_{\boldsymbol{l}_2}]^T$ is a constant vector, and $\boldsymbol{g}$ is a vector function of $\boldsymbol{\tau}$, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

V. π-FACTOR

In this section, we first introduce the $\boldsymbol{\pi}$-factor, and then describe how to efficiently minimize the least-squares problem with $\boldsymbol{\pi}$-factors.

$\boldsymbol{\pi}$-Factor Formulation Suppose that N points and M lines on a plane $\boldsymbol{\pi}$ are captured at the poses $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, which forms two sets $\{\boldsymbol{p}^n_1 \leftrightarrow \boldsymbol{p}^n_2\}^N_{n=1}$ and $\{\boldsymbol{l}^m_1 \leftrightarrow \boldsymbol{l}^m_2\}^M_{m=1}$. For each $\boldsymbol{p}^n_1 \leftrightarrow \boldsymbol{p}^n_2$, we can compute a constant vector



π
$\boldsymbol{P}$
$\boldsymbol{p}_1$
$\boldsymbol{p}_2$
$(\mathbf{R}_1, \boldsymbol{t}_1)$
$(\mathbf{R}_2, \boldsymbol{t}_2)$
(a)
π
$\boldsymbol{P}^1_{\boldsymbol{l}_2}$
$\boldsymbol{P}^2_{\boldsymbol{l}_2}$
$\boldsymbol{L}$
$\boldsymbol{n}_{\boldsymbol{l}_1}$
$\boldsymbol{\pi}_{\boldsymbol{l}_1}$
$\vec{\boldsymbol{p}}^2_{\boldsymbol{l}_2}$
$\vec{\boldsymbol{p}}^1_{\boldsymbol{l}_2}$
$\boldsymbol{l}_1$
$\boldsymbol{l}_2$
$\boldsymbol{p}^2_{\boldsymbol{l}_2}$
$\boldsymbol{p}^1_{\boldsymbol{l}_2}$
$(\mathbf{R}_1, \boldsymbol{t}_1)$
$(\mathbf{R}_2, \boldsymbol{t}_2)$
(b)

Fig. 2. A schematic of the $\boldsymbol{\pi}$-constraints derived from a point $\boldsymbol{P}$ **(a)** and a line $\boldsymbol{L}$ **(b)** on a plane $\boldsymbol{\pi}$. In Fig. **(a)**, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are the back-projected ray of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, respectively. The constraint from $\boldsymbol{P}$ in *Lemma* 1 is based on the fact that the intersection point between $\boldsymbol{\pi}$ and $\boldsymbol{L}_2$ should be on $\boldsymbol{L}_1$. In Fig. **(b)**, $\boldsymbol{\pi}_{\boldsymbol{l}_1}$ is the back-projected plane of $\boldsymbol{l}_1$, and $\boldsymbol{L}^1_{\boldsymbol{l}_2}$ and $\boldsymbol{L}^2_{\boldsymbol{l}_2}$ are the back-projected rays of the two endpoints of $\boldsymbol{l}_2$, respectively. The constraint from $\boldsymbol{L}$ in *Lemma* 2 is based on the fact that the intersection points $\boldsymbol{P}^i_{\boldsymbol{l}_2}$ $(i = 1, 2)$ between $\boldsymbol{\pi}$ and $\boldsymbol{L}^i_{\boldsymbol{l}_2}$ should be on $\boldsymbol{\pi}_{\boldsymbol{l}_1}$. Note that the endpoints of $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ may not be from the same 3D points.

$\boldsymbol{c}_n$, according to (6). Similarly, for each $\boldsymbol{l}^n_1 \leftrightarrow \boldsymbol{l}^n_2$, we can calculate two constant vectors $\boldsymbol{d}^1_n$ and $\boldsymbol{d}^2_n$, according to (7). Let us define

$$\mathbf{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_N]^T \text{ and } \mathbf{D} = [\boldsymbol{d}^1_1, \boldsymbol{d}^2_1, \cdots, \boldsymbol{d}^1_M, \boldsymbol{d}^2_M]. \tag{8}$$

Stacking all the residuals from the N points and M lines, we get a $(3N + 2M)$-dimensional vector

$$\boldsymbol{\varepsilon}(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{bmatrix} \mathbf{C}\boldsymbol{f}_1(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) \\ \mathbf{C}\boldsymbol{f}_2(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) \\ \mathbf{C}\boldsymbol{f}_3(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) \\ \mathbf{D}\boldsymbol{g}(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) \end{bmatrix}. \tag{9}$$

In this work, $\boldsymbol{\varepsilon}(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2)$ is referred to as **$\boldsymbol{\pi}$-factor**. $\boldsymbol{x}_1$ is called the **reference image**, and $\boldsymbol{x}_2$ is called the **target image**. A $\boldsymbol{\pi}$-factor contains multiple constraints in (4) and (5). We will show that, with certain preprocessing, no matter how many $\boldsymbol{\pi}$-constraints are in a $\boldsymbol{\pi}$-factor, the computational complexity associated with a $\boldsymbol{\pi}$-factor in the LM algorithm is $O(1)$. This can significantly reduce the runtime.

$\boldsymbol{\pi}$-Factor Minimization Now we consider minimizing a least-squares problem with $\boldsymbol{\pi}$-factors. To simplify the notation, we omit the variables of functions in the following description (*e.g.*, $\boldsymbol{\varepsilon}(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2) \to \boldsymbol{\varepsilon}$).

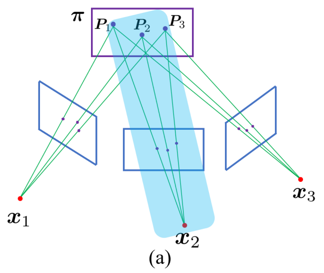
Let us denote the Jacobian matrix of $\boldsymbol{\varepsilon}$ as $\mathbf{J}_{\boldsymbol{\varepsilon}}$. According to (2) and (3), we know that only $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\mathbf{J}_{\boldsymbol{\varepsilon}}$, $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}$ are essential, instead of $\mathbf{J}_{\boldsymbol{\varepsilon}}$ and $\boldsymbol{\varepsilon}$. We will show that with some preprocessing, the computational complexities of $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\mathbf{J}_{\boldsymbol{\varepsilon}}$, $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}$ are all $O(1)$.

Let us define $\mathbf{K} = \mathbf{C}^T\mathbf{C}$ and $\mathbf{Q} = \mathbf{D}^T\mathbf{D}$, and for $\boldsymbol{f}_i$ $(i = 1, 2, 3)$ and $\boldsymbol{g}$ in (9), we define

$$\mathbf{U}_i = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{\tau}}, \quad \mathbf{V}_i = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_1}, \quad \mathbf{W}_i = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_2},$$
$$\mathbf{X} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\tau}}, \quad \mathbf{Y} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_1}, \quad \mathbf{Z} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_2}. \tag{10}$$

The above vector-by-vector derivatives are defined in (1). Using the above symbols, we can further define

$$\boldsymbol{\Delta}^{\boldsymbol{P}}_i = \begin{bmatrix} \mathbf{U}^T_i\mathbf{K}\mathbf{U}_i & \mathbf{U}^T_i\mathbf{K}\mathbf{V}_i & \mathbf{U}^T_i\mathbf{K}\mathbf{W}_i \\ \mathbf{V}^T_i\mathbf{K}\mathbf{U}_i & \mathbf{V}^T_i\mathbf{K}\mathbf{V}_i & \mathbf{V}^T_i\mathbf{K}\mathbf{W}_i \\ \mathbf{W}^T_i\mathbf{K}\mathbf{U}_i & \mathbf{W}^T_i\mathbf{K}\mathbf{V}_i & \mathbf{W}^T_i\mathbf{K}\mathbf{W}_i \end{bmatrix},$$
$$\boldsymbol{\Delta}^{\boldsymbol{L}} = \begin{bmatrix} \mathbf{X}^T\mathbf{Q}\mathbf{X} & \mathbf{X}^T\mathbf{Q}\mathbf{Y} & \mathbf{X}^T\mathbf{Q}\mathbf{Z} \\ \mathbf{Y}^T\mathbf{Q}\mathbf{X} & \mathbf{Y}^T\mathbf{Q}\mathbf{Y} & \mathbf{Y}^T\mathbf{Q}\mathbf{W} \\ \mathbf{Z}^T\mathbf{Q}\mathbf{X} & \mathbf{Z}^T\mathbf{Q}\mathbf{Y} & \mathbf{Z}^T\mathbf{Q}\mathbf{Z} \end{bmatrix}. \tag{11}$$



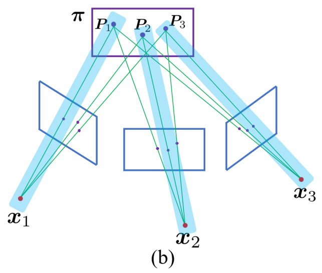


Fig. 3. A schematic of the effect of the reference image selection. Assume that three coplanar points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ and $\boldsymbol{P}_3$ are captured at three images $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$. No matter how the reference images are choose for $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ and $\boldsymbol{P}_3$, there will exist six $\boldsymbol{\pi}$-constraints with the form as Eq. (4). However, since only the $\boldsymbol{\pi}$-constraints with the same reference and target images can be packed into a $\boldsymbol{\pi}$-factor as shown in (9), a different choice of reference images may result in a different number of $\boldsymbol{\pi}$-factors. In Fig. **(a)**, $\boldsymbol{x}_2$ is selected as the reference image of $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, and $\boldsymbol{P}_3$. In Fig. **(b)**, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ are selected as the reference image of $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, and $\boldsymbol{P}_3$, respectively. For Fig. **(a)**, only two $\boldsymbol{\pi}$-factors are required to pack the six $\boldsymbol{\pi}$-constraints. However, six $\boldsymbol{\pi}$-factors are needed for Fig. **(b)**.

The following theorem provides the forms of $\mathbf{J}_{\varepsilon}^T\mathbf{J}_{\varepsilon}$, $\mathbf{J}_{\varepsilon}^T\varepsilon$ and $\varepsilon^T\varepsilon$ using the above symbols.

Theorem 3: $\mathbf{J}_{\varepsilon}^T\mathbf{J}_{\varepsilon}$, $\mathbf{J}_{\varepsilon}^T\varepsilon$, and $\varepsilon^T\varepsilon$ have the forms

$$\mathbf{J}_{\varepsilon}^T\mathbf{J}_{\varepsilon} = \mathbf{\Delta}^{\boldsymbol{L}} + \sum_{i=1}^{3}\mathbf{\Delta}_i^{\boldsymbol{P}},$$
$$\mathbf{J}_{\varepsilon}^T\varepsilon = \left[\mathbf{X},\mathbf{Y},\mathbf{Z}\right]^T\mathbf{Q}\boldsymbol{g} + \sum_{i=1}^{3}\left[\mathbf{U}_i,\mathbf{V}_i,\mathbf{W}_i\right]^T\mathbf{K}\boldsymbol{f}_i, \quad (12)$$
$$\varepsilon^T\varepsilon = \boldsymbol{g}^T\mathbf{Q}\boldsymbol{g} + \sum_{i=1}^{3}\boldsymbol{f}_i^T\mathbf{K}\boldsymbol{f}_i.$$

As the sizes of the matrices in (12) are small, the computation is very efficient. From Eq. (12), we know that given $\mathbf{K}$ and $\mathbf{Q}$, the computational complexities of $\mathbf{J}_{\varepsilon}^T\mathbf{J}_{\varepsilon}$, $\mathbf{J}_{\varepsilon}^T\varepsilon$ and $\varepsilon^T\varepsilon$ are all $O(1)$. Since $\mathbf{K}$ and $\mathbf{Q}$ are reused during the iteration, we only need to compute them once.

VI. π-BA

$\boldsymbol{\pi}$-BA Formulation In $\boldsymbol{\pi}$-BA, $\boldsymbol{\pi}$-factors are used to replace the reprojection errors of coplanar points and lines. For constructing $\boldsymbol{\pi}$-factors, the images with coplanar points and lines are divided into reference images and target images. The factor graph of $\boldsymbol{\pi}$-BA is demonstrated in Fig. 4. Formally, the full cost function is given by

$$\sum_{i\in\mathbb{A}}\sum_{j\in\mathbb{B}_i}\sum_{k\in\mathbb{P}_{ij}}\|\varepsilon\left(\boldsymbol{\tau}_k,\boldsymbol{x}_i,\boldsymbol{x}_j\right)\|_2^2 + C_{other}, \quad (13)$$

where $\mathbb{A}$ denotes the set of reference images, $\mathbb{B}_i$ represents the set of target images associated with the reference image with pose $\boldsymbol{x}_i$, $\mathbb{P}_{ij}$ describes the set of planes visible at both $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and C_{other} includes other costs, such as reprojection errors of non-coplanar points and lines, IMU and GPS.

Greedy Division In $\boldsymbol{\pi}$-BA, $\boldsymbol{\pi}$-constraints are packed into $\boldsymbol{\pi}$-factors. The way of packing affects the efficiency, as demonstrated in Fig. 3. A $\boldsymbol{\pi}$-factor in (9) contains the $\boldsymbol{\pi}$-constraints of a plane with the same reference and target images. Each coplanar 3D point or 3D line is captured in multiple images. In our algorithm, one of these images is

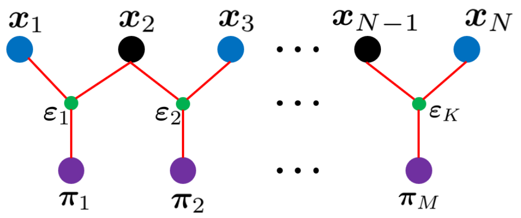


Fig. 4. A factor graph of $\boldsymbol{\pi}$-BA with N poses, M planes, and K $\boldsymbol{\pi}$-factors. The $\boldsymbol{\pi}$-factor is a ternary factor connecting two poses and a plane. Each $\boldsymbol{\pi}$-factor has a reference image and a target images. Black and blue circles represent the reference and target images of $\boldsymbol{\pi}$-factors, respectively.

selected as the reference image, and the remaining ones are treated as target images. The number of $\boldsymbol{\pi}$-factors is determined by this division. As demonstrated in Fig. 3, different divisions may lead to different numbers of $\boldsymbol{\pi}$-factors, which in turn affects the efficiency. It is difficult to get the best division that leads to the smallest number of $\boldsymbol{\pi}$-factors. From *Theorem* 5, we know that given $\mathbf{Q}$ and $\mathbf{K}$, no matter how many $\boldsymbol{\pi}$-constraints are packed into a $\boldsymbol{\pi}$-factor, the computational complexity of handling a $\boldsymbol{\pi}$-factor in the LM algorithm is the same. Intuitively, if we can pack more $\boldsymbol{\pi}$-constraints into a $\boldsymbol{\pi}$-factor, we can save more computational time. Here we introduce an greedy algorithm to get the $\mathbb{A}$ and $\mathbb{T} = \{\mathbb{B}_i | i \in \mathbb{A}\}$ in (13).

Let us use $\mathbb{Q}$ to represent the set of all coplanar points and lines in the 3D space. In our algorithm, we first count the number of coplanar points and lines captured at each image. Assume that the image with pose $\boldsymbol{x}_i$ captures the largest number of coplanar points and lines. Let us denote the set of coplanar points and lines captured at $\boldsymbol{x}_i$ as $\mathbb{Q}_i$. The image with pose $\boldsymbol{x}_i$ is select as the first reference image (*i.e.*, $\mathbb{A} = \{i\}$), and the images which can see any points or lines in $\mathbb{Q}_i$ form the set of corresponding target images $\mathbb{B}_i$. Then we remove $\mathbb{Q}_i$ from $\mathbb{Q}$ (*i.e.*, $\mathbb{Q} = \mathbb{Q} - \mathbb{Q}_i$), and repeat the above process until $\mathbb{Q}$ is empty. Algorithm 1 summarizes the above greedy algorithm.

Algorithm 1 Get the set of reference images $\mathbb{A}$ and target images $\mathbb{T} = \{\mathbb{B}_i | i \in \mathbb{A}\}$ for $\boldsymbol{\pi}$-BA in Eq. (13).

```
$\mathbb{A} \leftarrow \emptyset$, $\mathbb{T} \leftarrow \emptyset$;
$\mathbb{Q} \leftarrow$ {coplanar points and lines};
while $\mathbb{Q} \neq \emptyset$ do
    $x_i \leftarrow$ a pose that captures the largest number of
        elements in $\mathbb{Q}$;
    $\mathbb{Q}_i \leftarrow$ {planar points and lines captured at $x_i$};
    $\mathbb{B}_i \leftarrow$ {indices of poses see any elements in $\mathbb{Q}_i$};
    $\mathbb{A} \leftarrow \mathbb{A} \cup \{i\}$, $\mathbb{T} \leftarrow \mathbb{T} \cup \{\mathbb{B}_i\}$;
    $\mathbb{Q} \leftarrow \mathbb{Q} - \mathbb{Q}_i$;
end while
```

VII. Experimental Results

In this section, we use synthetic and real data to evaluate our algorithm. All the experiments were conducted on a desktop with an Intel i9 processor and 64GB memory.

A. Setup

Algorithms In the experiments, we evaluate the following three algorithms:

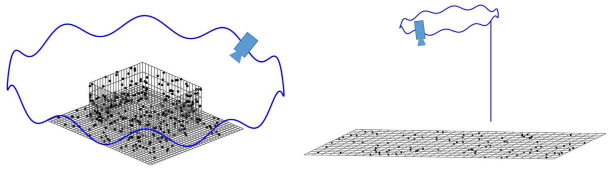


Fig. 5. The two synthetic datasets used in our experiments. They contain 200 and 400 images, respectively.

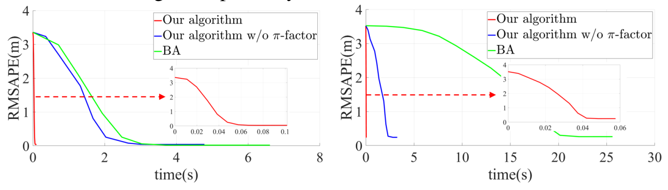


Fig. 6. The root mean square of absolute pose errors (RMSAPE) *w.r.t.* the runtime using the synthetic data in Fig. 5. The insets in the above figures provide close-ups of our algorithm on the two datasets. Our algorithm obviously surpasses the other two algorithms in speed.

- **Our algorithm:** The algorithm introduced in this paper.
- **Our algorithm w/o π-factor:** The π-factor is not adopted in our algorithm. That is to say all the π-constraints are computed individually.
- **BA:** This is the traditional BA algorithm with the reprojection error, implemented by g2o [56].

All the algorithms use the same stopping criteria.

Metrics Previous works generally use the cost with respect to time to evaluate the performance of a BA algorithm [8]–[14]. This is because they use the same cost function. However, as a different cost function is adopted in this work, this method is not suitable. Instead, we adopt the root mean square of absolute pose errors (RMSAPE) to evaluate the convergence speed of different algorithms [57]. Specifically, we first use a similarity transformation matrix to align the estimated poses and the ground truth after each iteration. Then RMSAPE is computed to evaluate the performance.

Initialization Initial camera poses and landmarks are generated by perturbing the ground truth ones with isotropic zero-mean Gaussian noises. We denote the standard deviations (STD) of the Gaussian noises for translation vectors, angle-axis representations of rotation matrices, and landmarks (*i.e.*, 3D points and the endpoints of 3D lines) as $\sigma_{\boldsymbol{t}}$, $\sigma_{\boldsymbol{\theta}}$, and σ_l respectively. In our experiment, we set $\sigma_{\boldsymbol{\theta}} = \frac{\sigma_{\boldsymbol{t}}}{10}$. The initial plane parameters are obtained by fitting a plane to the noisy 3D points and lines.

B. Synthetic Data

We first use synthetic data to compare the performance of different algorithms. Specifically, two scenes with coplanar points and lines are generated, as shown in Fig. 5. In the first scene, a synthesized camera moves around a four-sided fence with a periodic change in the pitch and roll angles. In the second scene, a downward facing camera takes off vertically from a ground plane, and then flies around at the height of $100m$. In the experiment, the resolution of the virtual camera is set to 1280×800 pixels, and the focal length is set to 930 pixels. We generate 200 and 400 images for the two scenes, and add the zero-mean Gaussian noise with STD 1 pixel to the 2D feature points and the endpoints of 2D lines.

The algorithms are initialized by perturbing the landmarks and the poses. For 3D points and lines, we set the STD of the Gaussian noise as $\sigma_l = 0.1m$. For poses, we set $\sigma_{\boldsymbol{t}} = 2m$ and $\sigma_{\boldsymbol{\theta}} = 0.2rad$. Fig. 6 shows the results. It is clear that our algorithm is significantly faster than other algorithms.

C. Real Data

We collect four real datasets in urban environments using two aerial robots. The ground truth poses were acquired from an inertial navigation system (INS) with RTK-GPS. We adopt COLMAP [58], [59] with the ground truth poses to generate 3D points and 3D lines. Then we use PCL [60] to detect planes from the point cloud. Finally, we get the coplanar lines by checking the distances between a plane and the endpoints of 3D lines. These coplanar points and lines are used to evaluate the performance of different algorithms.

The poses and landmarks are perturbed by the same Gaussian noises as the synthetic data, *i.e.*, $\sigma_{\boldsymbol{t}} = 2m$, $\sigma_{\boldsymbol{\theta}} = 0.2rad$, and $\sigma_l = 0.1m$. We also add the noisy GPS positions into the optimization. Fig. 8 provides the results. Our algorithm outperforms the traditional BA in terms of both speed and accuracy. Our algorithm and our algorithm w/o $\boldsymbol{\pi}$-factor converge at the same point, but the first one is much faster. This verifies that $\boldsymbol{\pi}$-factor can significantly reduce the computational cost.

VIII. Conclusions

This paper introduces an efficient algorithm for $\boldsymbol{\pi}$-BA. We present new constraints, referred to as $\boldsymbol{\pi}$-constraints, for coplanar points and lines, and show that the $\boldsymbol{\pi}$-constraint has a special form, so that $\boldsymbol{\pi}$-constraints can share a lot of computations in the LM algorithm. This is the crux of our algorithm. We introduce a greedy algorithm to group $\boldsymbol{\pi}$-constraints into $\boldsymbol{\pi}$-factors to construct the cost function of $\boldsymbol{\pi}$-BA. We prove that, with some simple matrix multiplications as preprocessing, the computational complexity of a $\boldsymbol{\pi}$-factor in the LM algorithm is $O(1)$, independent of the number of $\boldsymbol{\pi}$-constraints in it. Experimental results show that our algorithm can significantly reduce the computational load.

Appendix

A. Proof of Lemma 1

Let us denote the back-projected rays of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ as $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, respectively. In the world coordinate system, the directions of $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are $\mathbf{R}\vec{\boldsymbol{p}}_1$ and $\mathbf{R}\vec{\boldsymbol{p}}_2$, respectively. As shown in Fig. 2, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ pass through the origins of the camera coordinate systems, respectively. Thus, in the world coordinate system, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ passes through the 3D points $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively. As $\boldsymbol{P}$ is the intersection between $\boldsymbol{L}_2$ and $\boldsymbol{\pi}$, we have

$$\boldsymbol{P} = k_{\boldsymbol{p}_2}\mathbf{R}_2\vec{\boldsymbol{p}}_2 + \boldsymbol{t}_2,\ \boldsymbol{n}^T\boldsymbol{P} + d = 0. \tag{14}$$

Substituting the formula of $\boldsymbol{P}$ into $\boldsymbol{n}^T\boldsymbol{P} + d = 0$, we get

$$k_{\boldsymbol{p}_2} = \frac{\alpha}{\beta_{\boldsymbol{p}_2}}, \alpha = -\boldsymbol{n}^T\boldsymbol{t}_2 - d \text{ and } \beta_{\boldsymbol{p}_2} = \boldsymbol{n}^T\mathbf{R}_2\vec{\boldsymbol{p}}_2. \tag{15}$$

Define $\tau_{\boldsymbol{p}_2} = \frac{1}{k_{\boldsymbol{p}_2}}$. As $\boldsymbol{P}$ is on the line $\boldsymbol{L}_1$, we obtain

$$\mathbf{R}_1\vec{\boldsymbol{p}}_1 \times (\boldsymbol{P} - \boldsymbol{t}_1) = \mathbf{0}_{3\times1}. \tag{16}$$

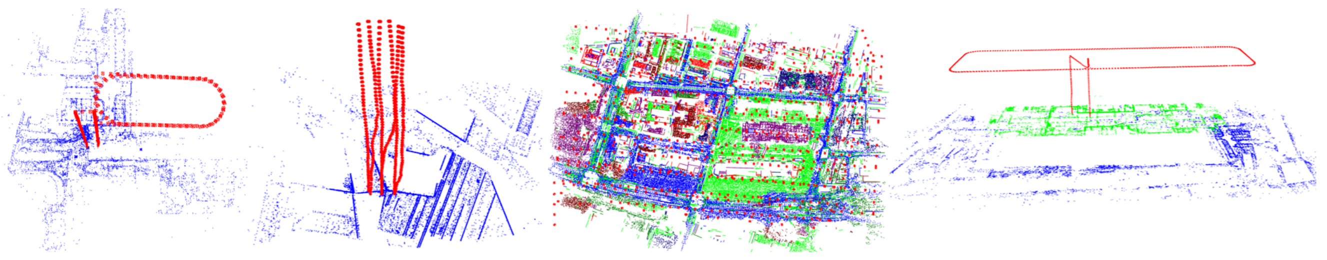


Fig. 7. The four real datasets used in our experiments. They contain 306, 377, 757, and 895 images, respectively. Points on the same plane are colorized by the same color. As planes are unbounded, some planar patches belonging to the same plane are not connected in the third dataset.

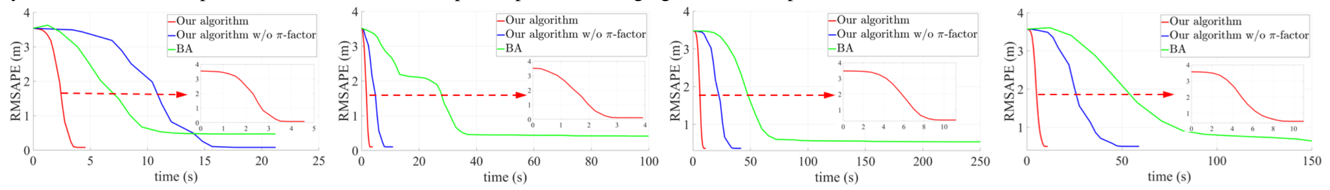


Fig. 8. The root mean square of absolute pose errors (RMSAPE) *w.r.t.* the runtime using the four datasets in Fig. 7. Our algorithm is much faster than the other two algorithms. In addition, the two versions of our algorithm get more accurate results than BA.

Substituting the first equation of (14) into (16) and using $\tau_{\boldsymbol{p}_2} = \frac{1}{k_{\boldsymbol{p}_2}}$ in (15), we have

$$\begin{aligned} &\mathbf{R}_1\vec{\boldsymbol{p}}_1 \times \left(\frac{1}{\tau_{\boldsymbol{p}_2}}\mathbf{R}_2\vec{\boldsymbol{p}}_2 + \boldsymbol{t}_2 - \boldsymbol{t}_1\right) = \mathbf{0}_{3\times 1} \\ \Rightarrow\ &\mathbf{R}_1\vec{\boldsymbol{p}}_1 \times \frac{1}{\tau_{\boldsymbol{p}_2}}\left(\mathbf{R}_2\vec{\boldsymbol{p}}_2 + \tau_{\boldsymbol{p}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)\right) = \mathbf{0}_{3\times 1} \\ \Rightarrow\ &\mathbf{R}_1\vec{\boldsymbol{p}}_1 \times \left(\mathbf{R}_2\vec{\boldsymbol{p}}_2 + \tau_{\boldsymbol{p}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)\right) = \mathbf{0}_{3\times 1}. \end{aligned} \tag{17}$$

B. Proof of Lemma 2

Let us denote the back-projected plane of $\boldsymbol{l}_1$ as $\boldsymbol{\pi}_{\boldsymbol{l}_1}$, and the back-projected rays of the two endpoints $\boldsymbol{p}^i_{\boldsymbol{l}_2}$ $(i = 1, 2)$ of $\boldsymbol{l}_2$ as $\boldsymbol{L}^i_{\boldsymbol{l}_2}$. As illustrated in Fig. 2, in the world coordinate system, $\boldsymbol{\pi}_{\boldsymbol{l}_1}$ passes through the 3D point $\boldsymbol{t}_1$ with the normal $\mathbf{R}\boldsymbol{n}_{\boldsymbol{l}_1}$, and $\boldsymbol{L}^i_{\boldsymbol{l}_2}$ passes through the 3D point $\boldsymbol{t}_2$ with the direction $\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2}$.

Let us first compute the intersection between $\boldsymbol{L}^i_{\boldsymbol{l}_2}$ and $\boldsymbol{\pi}$, denoted as $\boldsymbol{P}^i_{\boldsymbol{l}_2}$. According to (14) and (15), $\boldsymbol{P}^i_{\boldsymbol{l}_2}$ has the form

$$\boldsymbol{P}^i_{\boldsymbol{l}_2} = \frac{\alpha}{\beta^i_{\boldsymbol{l}_2}}\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \boldsymbol{t}_2, \tag{18}$$

where $\alpha = -\boldsymbol{n}^T\boldsymbol{t}_2 - d$ and $\beta^i_{\boldsymbol{l}_2} = \boldsymbol{n}^T\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2}$.

Define $\tau^i_{\boldsymbol{l}_2} = \frac{\beta^i_{\boldsymbol{l}_2}}{\alpha}$. As $\boldsymbol{P}^i_{\boldsymbol{l}_2}$ should be on $\boldsymbol{\pi}_{\boldsymbol{l}_1}$, we have

$$\mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1} \cdot \left(\boldsymbol{P}^i_{\boldsymbol{l}_2} - \boldsymbol{t}_1\right) = 0. \tag{19}$$

Substituting (18) into (19) and using $\tau^i_{\boldsymbol{l}_2} = \frac{\beta^i_{\boldsymbol{l}_2}}{\alpha}$, we get

$$\begin{aligned} &\mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1} \cdot \left(\frac{1}{\tau^i_{\boldsymbol{l}_2}}\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \boldsymbol{t}_2 - \boldsymbol{t}_1\right) = 0 \\ \Rightarrow\ &\mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1} \cdot \frac{1}{\tau^i_{\boldsymbol{l}_2}}\left(\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \tau^i_{\boldsymbol{l}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)\right) = 0 \\ \Rightarrow\ &\mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1} \cdot \left(\mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \tau^i_{\boldsymbol{l}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)\right) = 0. \end{aligned} \tag{20}$$

C. Proof of Theorem 1

Let us define $\boldsymbol{a} = \mathbf{R}_1\vec{\boldsymbol{p}}_1$ and $\boldsymbol{b} = \mathbf{R}_2\vec{\boldsymbol{p}}_2 + \tau_{\boldsymbol{p}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)$. Then equation (4) can be written as $\boldsymbol{a} \times \boldsymbol{b} = \mathbf{0}_{3\times 1}$. Suppose that a_n and b_n represent the nth element of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. Expanding $\boldsymbol{a}$ and $\boldsymbol{b}$, we know that a_n and b_n have the forms as

$$a_n = \boldsymbol{r}^1_n \cdot \vec{\boldsymbol{p}}_1 \text{ and } b_n = \boldsymbol{c}_n \cdot \vec{\boldsymbol{p}}_2, n = 1, 2, 3, \tag{21}$$

where $\boldsymbol{r}^1_n$ is the nth row of $\mathbf{R}_1$, and $\boldsymbol{c}_n$ is a vector function of $\boldsymbol{\pi}$, $\boldsymbol{t}_1$, $\mathbf{R}_2$ and $\boldsymbol{t}_2$. Substituting (21) into $\boldsymbol{a} \times \boldsymbol{b}$, we have

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_2\boldsymbol{c}^T_3\vec{\boldsymbol{p}}_2 - \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_3\boldsymbol{c}^T_2\vec{\boldsymbol{p}}_2 \\ \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_3\boldsymbol{c}^T_1\vec{\boldsymbol{p}}_2 - \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_1\boldsymbol{c}^T_3\vec{\boldsymbol{p}}_2 \\ \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_1\boldsymbol{c}^T_2\vec{\boldsymbol{p}}_2 - \vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_2\boldsymbol{c}^T_1\vec{\boldsymbol{p}}_2 \end{bmatrix}. \tag{22}$$

Note that $\boldsymbol{\tau}$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$ are the parameterizations of $\boldsymbol{\pi}$, $(\mathbf{R}_1, \boldsymbol{t}_1)$, and $(\mathbf{R}_2, \boldsymbol{t}_2)$, respectively. So it is clear that each $\vec{\boldsymbol{p}}^T_1\boldsymbol{r}^1_i\boldsymbol{c}^T_j\vec{\boldsymbol{p}}_2$ in (22) is of a quadratic form with respect to $\vec{\boldsymbol{p}}_1$ and $\vec{\boldsymbol{p}}_2$, whose coefficients are functions of $\boldsymbol{\tau}$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$. Thus, substituting $\vec{\boldsymbol{p}}_1 = [x_1; y_1; z_1]$ and $\vec{\boldsymbol{p}}_2 = [x_2; y_2; z_2]$ into (22) and expanding it, we can get that the elements of $\boldsymbol{a} \times \boldsymbol{b}$ have the form as (6).

D. Proof of Theorem 2

Let us define $\boldsymbol{u} = \mathbf{R}_1\boldsymbol{n}_{\boldsymbol{l}_1}$ and $\boldsymbol{v}^i = \mathbf{R}_2\vec{\boldsymbol{p}}^i_{\boldsymbol{l}_2} + \tau^i_{\boldsymbol{l}_2}\left(\boldsymbol{t}_2 - \boldsymbol{t}_1\right)$ $(i = 1, 2)$. Then equation (5) can be written as $\boldsymbol{u} \cdot \boldsymbol{v}^i = 0$. Suppose that u_n and v^i_n are the nth element of $\boldsymbol{u}$ and $\boldsymbol{v}^i$, respectively. Similar to (21), we have

$$u_n = \mathbf{r}^1_n \cdot \boldsymbol{n}_{\boldsymbol{l}_1} \text{ and } v^i_n = \boldsymbol{d}_n \cdot \vec{\boldsymbol{p}}_{\boldsymbol{l}_2}, n = 1, 2, 3, \tag{23}$$

where $\boldsymbol{r}^1_n$ is the nth row of $\mathbf{R}_1$, and $\boldsymbol{d}_n$ is a vector function of the elements in $\boldsymbol{\pi}$, $\boldsymbol{t}_1$, $\mathbf{R}_2$ and $\boldsymbol{t}_2$. Similar to (22), substituting (23) into $\boldsymbol{u} \cdot \boldsymbol{v}^i = 0$ and expanding it, we can obtain that it has the form as (7).

E. Proof of Theorem 3

Let us first consider the Jacobian $\mathbf{J}_{\boldsymbol{\varepsilon}}$ of $\varepsilon(\boldsymbol{\tau}, \boldsymbol{x}_1, \boldsymbol{x}_2)$ defined in (9). Using the notations in (10), $\mathbf{J}_{\boldsymbol{\varepsilon}}$ has the form

$$\mathbf{J}_{\boldsymbol{\varepsilon}} = \begin{bmatrix} \mathbf{CU}_1 & \mathbf{CV}_1 & \mathbf{CW}_1 \\ \mathbf{CU}_2 & \mathbf{CV}_2 & \mathbf{CW}_2 \\ \mathbf{CU}_3 & \mathbf{CV}_3 & \mathbf{CW}_3 \\ \mathbf{GX} & \mathbf{GY} & \mathbf{GZ} \end{bmatrix}. \tag{24}$$

Substituting ε in (9) and $\mathbf{J}_{\boldsymbol{\varepsilon}}$ in (24) into $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\mathbf{J}_{\boldsymbol{\varepsilon}}$, $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\varepsilon$ and $\varepsilon^T\varepsilon$ and using the block matrix multiplication rule, we obtain that $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\mathbf{J}_{\boldsymbol{\varepsilon}}$, $\mathbf{J}^T_{\boldsymbol{\varepsilon}}\varepsilon$ and $\varepsilon^T\varepsilon$ have the forms as shown in (12).

REFERENCES

[1] 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.

[2] M. Spetsakis and J. Y. Aloimonos, "A multi-frame approach to visual motion perception," *International Journal of Computer Vision*, vol. 6, no. 3, pp. 245–255, 1991.

[3] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[4] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[5] D. Sibley, C. Mei, I. D. Reid, and P. Newman, "Adaptive relative bundle adjustment." in *Robotics: science and systems*, vol. 32, 2009, p. 33.

[6] M. I. Lourakis and A. A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, pp. 1–30, 2009.

[7] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," in *European conference on computer vision*. Springer, 2010, pp. 29–42.

[8] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment," in *CVPR 2011*. IEEE, 2011, pp. 3057–3064.

[9] C. Zach, "Robust bundle adjustment revisited," in *European Conference on Computer Vision*. Springer, 2014, pp. 772–787.

[10] K. Natesan Ramamurthy, C.-C. Lin, A. Aravkin, S. Pankanti, and R. Viguier, "Distributed bundle adjustment," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 2146–2154.

[11] R. Zhang, S. Zhu, T. Fang, and L. Quan, "Distributed very large scale bundle adjustment by global camera consensus," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 29–38.

[12] L. Zhou, Z. Luo, M. Zhen, T. Shen, S. Li, Z. Huang, T. Fang, and L. Quan, "Stochastic Bundle Adjustment for Efficient and Scalable 3d Reconstruction," in *European Conference on Computer Vision*. Springer, 2020, pp. 364–379.

[13] N. Demmel, C. Sommer, D. Cremers, and V. Usenko, "Square root bundle adjustment for large-scale reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 723–11 732.

[14] N. Demmel, D. Schubert, C. Sommer, D. Cremers, and V. Usenko, "Square root marginalization for sliding-window bundle adjustment," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 260–13 268.

[15] T. Tanaka, Y. Sasagawa, and T. Okatani, "Learning to bundle-adjust: A graph network approach to faster optimization of bundle adjustment for vehicular slam," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 6250–6259.

[16] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, "Barf: Bundle-adjusting neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5741–5751.

[17] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up slam: Semantic monocular plane slam for low-texture environments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1222–1229.

[18] C. Liu, J. Yang, D. Ceylan, E. Yumer, and Y. Furukawa, "PlaneNet: Piece-wise planar reconstruction from a single rgb image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2579–2588.

[19] Z. Yu, J. Zheng, D. Lian, Z. Zhou, and S. Gao, "Single-image piece-wise planar 3d reconstruction via associative embedding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1029–1037.

[20] S. Du, H. Guo, Y. Chen, Y. Lin, X. Meng, L. Wen, and F.-Y. Wang, "Gpo: Global plane optimization for fast and accurate monocular slam initialization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6254–6260.

[21] X. Wang, M. Christie, and E. Marchand, "Relative Pose Estimation and Planar Reconstruction via Superpixel-Driven Multiple Homographies," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 625–10 632.

[22] X. Li, Y. He, J. Lin, and X. Liu, "Leveraging planar regularities for point line visual-inertial odometry," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5120–5127.

[23] X. Li, Y. Li, E. P. Örnek, J. Lin, and F. Tombari, "Co-planar parametrization for stereo-slam and visual-inertial odometry," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6972–6979, 2020.

[24] B. Xu, X. Li, J. Li, C. Yuen, J. Dai, and Y. Gong, "PVI-DSO: Leveraging Planar Regularities for Direct Sparse Visual-Inertial Odometry," *arXiv preprint arXiv:2204.02635*, 2022.

[25] X. Wang, M. Christie, and E. Marchand, "TT-SLAM: Dense Monocular SLAM for Planar Environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 11 690–11 696.

[26] P. Ji, Y. Tian, Q. Yan, Y. Ma, and Y. Xu, "Cnn-augmented visual-inertial slam with planar constraints," *arXiv preprint arXiv:2205.02940*, 2022.

[27] F. Shu, J. Wang, A. Pagani, and D. Stricker, "Structure plp-slam: Efficient sparse mapping and localization using point, line and plane for monocular, rgb-d and stereo cameras," *arXiv preprint arXiv:2207.06058*, 2022.

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

[29] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building rome in a day," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 72–79.

[30] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[31] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 363–370.

[32] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[33] 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.

[34] P. Baker and Y. Aloimonos, "Structure from motion of parallel lines," in *Computer Vision-ECCV 2004: 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part IV 8*. Springer, 2004, pp. 229–240.

[35] A. Bartoli and P. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer vision and image understanding*, vol. 100, no. 3, pp. 416–441, 2005.

[36] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, "Pl-slam: Real-time monocular visual slam with points and lines," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 4503–4508.

[37] Q. Wang, Z. Yan, J. Wang, F. Xue, W. Ma, and H. Zha, "Line flow based simultaneous localization and mapping," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1416–1432, 2021.

[38] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "PlanerCNN: 3d plane detection and reconstruction from a single image," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4450–4459.

[39] Z. Zhou, H. Jin, and Y. Ma, "Plane-based content preserving warps for video stabilization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2299–2306.

[40] 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.

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

[42] D. Chen, S. Wang, W. Xie, S. Zhai, N. Wang, H. Bao, and G. Zhang, "Vip-slam: An efficient tightly-coupled rgb-d visual inertial planar slam," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 5615–5621.

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

[44] 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.

[45] K. Favre, M. Pressigout, E. Marchand, and L. Morin, "A plane-based approach for indoor point clouds registration," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 7072–7079.

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

[47] L. Zhou, S. Wang, and M. Kaess, "π-lsam: Lidar smoothing and mapping with planes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5751–5757.

[48] L. Zhou, D. Koppel, and M. Kaess, "Lidar slam with plane adjustment for indoor environment," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7073–7080, 2021.

[49] L. Zhou, G. Huang, Y. Mao, J. Yu, S. Wang, and M. Kaess, "Plc-lislam: Lidar slam with planes, lines and cylinders," *IEEE Robotics and Automation Letters*, 2022.

[50] J. Zhang, C. Zhang, J. Wu, J. Jin, and Q. Zhu, "Lidar-inertial 3d slam with plane constraint for multi-story building," *arXiv preprint arXiv:2202.08487*, 2022.

[51] L. Zhou, "Efficient second-order plane adjustment," *arXiv preprint arXiv:2211.11542*, 2022.

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

[53] 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.

[54] G. 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)*. IEEE, 2019, pp. 1278–1284.

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

[56] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[57] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.

[58] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[59] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[60] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 1–4.