A Complete, Accurate and Efficient Solution for the Perspective-n-Line Problem

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

***Abstract*—This paper presents a complete, accurate, and efficient solution for the Perspective-n-Line (PnL) problem. Generally, the camera pose can be determined from $N \geq 3$ 2D-3D line correspondences. The minimal problem ($N = 3$) and the least-squares problem ($N > 3$) are generally solved in different ways. This paper shows that a least-squares PnL problem can be transformed into a quadratic equation system that has the same form as the minimal problem. This leads to a unified solution for the minimal and least-squares PnL problems. We adopt the Gram-Schmidt process and a novel hidden variable polynomial solver to increase the numerical stability of our algorithm. Experimental results show that our algorithm is more accurate and robust than the state-of-the-art least-squares algorithms [1], [2], [3], [4] and is significantly faster. Moreover, our algorithm is more stable than previous minimal solutions [3], [5], [6] with comparable runtime.**

***Index Terms*—Perspective-n-Line, Localization**

I. Introduction

LINES widely exist in man-made scenes. Thus, they have been extensively used in many robotics and computer vision tasks [7], [8]. Estimating the pose of a camera from N 2D/3D line correspondences, called the PnL problem, is a fundamental problem with many applications, such as structure from motion (SfM) [9], localization [10] and augmented reality [11]. This paper focuses on the PnL problem.

An ideal PnL algorithm should be complete, which means that the algorithm should be applicable to all non-degenerate cases of $N \geq 3$. Completeness is important in practice and theory, as it avoids having to implement multiple solutions to cover all possible inputs in real applications. Although many solutions have been proposed for the PnL problem, it is still difficult to achieve this goal.

Both the minimal and least-squares solutions are important in practice. One challenging problem is to find a unified solution for the minimal and least-squares problems. Typically, the minimal problem is addressed by solving an equation system, and the least-squares problem is formulated as a minimization problem. Theoretically, the minimal problem can also be formulated as a minimization problem and solved by a least-squares solution, such as [1]. However, this strategy is generally too slow to make it a practical solution. An interesting question is whether we can proceed in reverse, *i.e.*, address a least-squares problem by means of a minimal solution, as this may result in a complete and efficient solution. At first glance, this idea seems infeasible, as the least-squares problem is generally considered to be more complicated, which can be demonstrated by comparing the complexity of the resulting polynomial equations from both problems. The minimal problem generally leads to an eighth-order polynomial equation [3], [5], [6], [12], but the least-squares problem typically requires to solve a more complicated equation system. For example, Mirzaei's algorithm [13] requires to find the roots of three fifth-order polynomial equations. In [3], [14], subset-based solutions are presented which need to solve a fifteenth-order univariate equation. This paper shows that the least-squares problem can be solved as simply as the minimal problem.

The main contribution of this paper is a novel PnL algorithm with the following characteristics:

Complete: Our algorithm is applicable to all $N \geq 3$ line correspondences. The central idea of our algorithm is to compress the constraints of a least-squares PnL problem into a quadratic equation system with 3 equations and 3 unknowns of the rotation matrix, which has the same form as the minimal problem [5].

Accurate: Our algorithm achieves high accuracy and robustness by avoiding numerically unstable operations. Generally, the solution of a high-order polynomial system may be more sensitive to noise [15]. We compress the PnL constraints into a quadratic system with 3 equations which has a lower order than the state-of-the-art least-squares solutions [1], [2], [3]. We use the Gram-Schmidt process to remove the potential numerical instability in the constraint compression step. We adopt the hidden variable method introduced in [15] to solve this polynomial system, which shows better numerical stability than E3Q3 [16], RE3Q3 [5] and the Gröbner basis method [17], [15]. The experimental results show that the accuracy and robustness of our algorithm outperform the state-of-the-art least-squares solutions [1], [2], [3], [4] and minimal solutions [3], [5], [6].

Efficient: Our algorithm has $O(N)$ complexity and only needs to solve a small quadratic polynomial system. Our runtime is comparable to previous minimal solutions [3], [5], [6], and it is much faster than the state-of-the-art least-squares solutions [1], [2], [3], [4].

A. Related Work

In non-singular cases, the camera pose can be determined by $N \geq 3$ line correspondences. When $N = 3$, it is the

Manuscript received: October, 15, 2020; Accepted: December, 7, 2020.

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

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

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

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

minimal problem, known as the **P3L problem**. For $N > 3$, it is a least-squares problem. Both minimal and least-squares solutions play important roles in various robotics and computer vision tasks. Due to their importance, many efforts have been made to solve both problems.

P3L problem The P3L problem generally needs to solve an eighth-order univariate equation, except for some specific geometric configurations [3]. Thus it has at most 8 solutions. One widely adopted strategy for the P3L problem is to simplify the problem by introducing some geometrical transformation [3], [6], [12]. Specifically, these algorithms design intermediate coordinate systems to reduce the number of unknowns to one, which results in a univariate equation. Zhou *et al.* [5] adopt quaternions to parametrize the rotations, and introduce a direct algebraic solution for the P3L problem.

Least-Squares PnL problem Early work for the PnL problem mainly focuses on error function formulation and iterative solutions. Liu *et al.* [18] study how to formulate the constraints from 2D-3D point and line correspondences. They decouple the estimation of rotation and translation. Kumar and Hanson [19] propose to jointly optimize rotation and translation in an iterative method. They present a sampling-based method to get an initial estimation. In [20], [21], they propose to start the iteration from a pose estimated by the weak perspective or paraperspective camera model. The accuracy of the iterative algorithm depends on the quality of the initial solution. There is no guarantee that the iterative method will converge when the initial error is large.

The linear formulation plays an important role for most 3D vision problems [22]. Direct Linear Transformation (DLT) provides a straightforward solution for the PnL problem [22]. This method requires at least 6 line correspondences. Přibyl *et al.* [23] introduce a new DLT method based on the Plücker coordinates of the 3D line, and this method needs at least 9 lines. In their later work [24], they combine the above two DLT methods, which shows improved accuracy and reduces the minimum required number of line correspondences to 5. By exploring the similarity between the constraints derived from the PnP problem and the PnL problem, the EPnP algorithm [25] is extended to solve the PnL problem [2], [3]. The EPnP-based PnL algorithm is applicable to $N \geq 4$, but it is not stable when N is small and needs special treatment for the planar PnL problem (*i.e.*, all lines are on the same plane). Linear formulations ignore the constraints on the unknowns. This generally results in a less accurate solution.

To solve the above problem, methods based on polynomial formulation have been proposed. Ansar *et al.* [26] adopt a quadratic system to represent the constraints, and present a linearized approach to solve this system. Their algorithm does not scale well with a large N. Motivated by the RPnP algorithm [27], subset-based PnL approaches are presented in [3], [14]. They divide the N line correspondences into $N-2$ triplets, and minimize the sum of squared polynomials that are derived from these triplets. As their cost functions are of algebraic nature, this may result in a suboptimal solution. Using the Gröbner basis technique [15], it is able to directly solve a polynomial system. This results in a series of direct minimization methods. In the literature, Cayley-Gibbs-Rodrigues (CGR) [13], [28] and quaternion [2] parametrizations are adopted to express the rotation, which results in a polynomial cost function. Then the Gröbner basis technique is used to solve the first optimality conditions of the cost function. The cost functions in these algorithms are not derived from the reprojection error, which may lead to a suboptimal solution. Besides, the Gröbner basis technique may encounter numerical problems as described in [15]. Zhou *et al.* [1] show that the reprojection error from the 2D-3D line correspondence can be approximated by two polynomial distances, and they introduce a hidden variable polynomial solver. Although they show improved accuracy, our experiments demonstrate that this algorithm is still unstable under some challenging conditions, such as large noise and planar configuration.

The PnL problem has some extensions for certain applications. Recently, Hichem *et al.* [29] propose a direct least-squares solution for the PnL problem of a multi-camera system. In some applications, the vertical direction is known from a certain sensor (*e.g.*, IMU). This can be used as a prior for the pose estimation [30]. This paper focuses on the PnL problem for a single camera.

In summary, a desirable PnL solution is the one that is accurate and efficient for any possible inputs. As mentioned above, algorithms based on linear formulations are generally unstable or infeasible for a small N, and typically need special treatment or do not work for the planar configuration. On the other hand, algorithms based on a polynomial formulation generally achieve better accuracy and are applicable to a broader configuration of PnL inputs, but they are more computationally demanding and may encounter numerical problems. Therefore, there exists a demand for improvement over the state-of-the-art PnL solution.

II. PnL Problem and Notations

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

A. Constraint

The PnL problem is to estimate the camera pose including rotation $\mathbf{R}$ and translation $\mathbf{t}$ from $N \geq 3$ 2D-3D line correspondences $\{\mathbf{l}_i \leftrightarrow \mathbf{L}_i\}_{i=1}^{N}$. It is the line counterpart of the PnP problem [31], [32], [33].

For the ith correspondence as shown in Fig. 1, we assume $\mathbf{P}_{ij}$, $j = 1, 2$ are two 3D points on $\mathbf{L}_i$. Let us define the camera intrinsic matrix as $\mathbf{M}$. The projection from $\mathbf{P}_{ij}$ to the camera can be written as $\hat{\mathbf{p}}_{ij} \sim \mathbf{M}(\mathbf{R}\mathbf{P}_{ij} + \mathbf{t})$, where $\hat{\mathbf{p}}_{ij}$ is in homogeneous coordinates. The 2D line $\mathbf{l}_i$ is represented by a three-dimensional vector [22]. As $\hat{\mathbf{p}}_{ij}$ should be on $\mathbf{l}_i$, we have $\mathbf{l}_i^T \hat{\mathbf{p}}_{ij} = 0$. Using the expression of $\hat{\mathbf{p}}_{ij}$, we get $\mathbf{l}_i^T \mathbf{M}(\mathbf{R}\mathbf{P}_{ij} + \mathbf{t}) = \left(\mathbf{M}^T \mathbf{l}_i\right)^T (\mathbf{R}\mathbf{P}_{ij} + \mathbf{t}) = 0$. $\mathbf{M}^T\mathbf{l}$ can also be treated as the normal vector of the back-projection plane of $\mathbf{l}_i$ [22]. As $\mathbf{M}$ is known, we can first compute $\mathbf{M}^T\mathbf{l}_i$. To simplify the notation, we use $\mathbf{l}_i$ to represent $\mathbf{M}^T\mathbf{l}_i$ in the following description. Thus the 2 constraints from $\mathbf{l}_i \leftrightarrow \mathbf{L}_i$ can be written as:

$$\mathbf{l}_i{}^T (\mathbf{R}\mathbf{P}_{ij} + \mathbf{t}) = 0, \quad j = 1, 2. \tag{1}$$

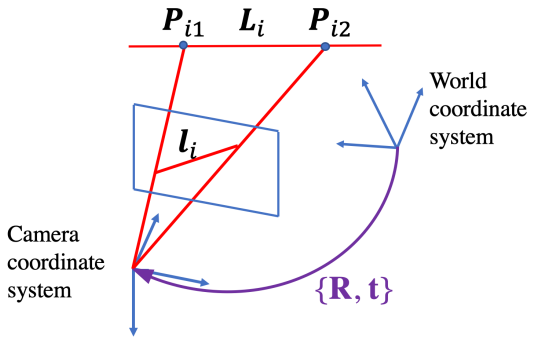


Fig. 1: A schematic of the constraints from $\mathbf{l}_i \leftrightarrow \mathbf{L}_i$

B. Minimal Constraint of $\mathbf{R}$

$N = 3$ is the minimal case of the PnL problem, called the P3L problem. The rotation estimation is crucial for the P3L problem. Basically, we can obtain 3 quadratic equations for the 3 unknowns of $\mathbf{R}$ [5], [12]. We call this quadratic system the minimal constraints of $\mathbf{R}$ (**MCR**). The basic idea of this paper is to compress the $2N$ constraints of N 2D-3D line correspondences into a MCR.

III. Our Algorithm

In this paper, we use the CGR parameterization to represent $\mathbf{R}$ as in [1], [13], [28]. Let us denote a three-dimensional vector as $\mathbf{s} = [s_1, s_2, s_3]$. Then the CGR parameterization for $\mathbf{R}$ can be written as

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

where $\mathbf{I}_3$ is the 3×3 identity matrix and $[\mathbf{s}]_\times$ is the skew matrix of $\mathbf{s}$. Here $\mathbf{s}$ differs from the angle-axis vector. Each element of $\bar{\mathbf{R}}$ is a quadratic in $\mathbf{s}$. The CGR parameterization is degenerate if the rotation angle of $\mathbf{R}$ is 180° [28], [32]. This problem can be solved by randomly rotating the 3D lines to a new frame when the cost of the solution of the original problem exceeds a certain threshold, and then rotating the solution for the new frame back to the original frame [28].

Let us substitute (2) into (1) and multiply $1 + \mathbf{s}^T\mathbf{s}$ to both sides, then we have

$$\mathbf{l}_i^T\bar{\mathbf{R}}\mathbf{P}_{ij} + (1 + \mathbf{s}^T\mathbf{s})\mathbf{l}_i^T\mathbf{t} = 0. \quad (3)$$

Expanding $\mathbf{l}_i^T\bar{\mathbf{R}}\mathbf{P}_{ij}$ in (3), we get a polynomial in $\mathbf{s}$ and $\mathbf{t}$

$$\begin{aligned} &\mathbf{a}_{ij}^T\mathbf{r} + (1 + \mathbf{s}^T\mathbf{s})\mathbf{l}_i^T\mathbf{t} = 0, \\ &\mathbf{r} = \left[s_1^2, s_2^2, s_3^2, s_1s_2, s_1s_3, s_2s_3, s_1, s_2, s_3, 1\right]^T, \end{aligned} \quad (4)$$

where $\mathbf{a}_{ij}$ is a ten-dimensional vector. $(1 + \mathbf{s}^T\mathbf{s})\mathbf{t}$ is a third-order polynomial in $\mathbf{s}$ and $\mathbf{t}$. To simplify (4), we define

$$\boldsymbol{\tau} = (1 + \mathbf{s}^T\mathbf{s})\mathbf{t}. \quad (5)$$

Using this definition, we can rewrite (4) as

$$\mathbf{a}_{ij}^T\mathbf{r} + \mathbf{l}_i^T\boldsymbol{\tau} = 0. \quad (6)$$

Given N 2D-3D correspondences, we have $2N$ equations as (6). Stacking these equations, we have

$$\mathbf{A}\mathbf{r} + \mathbf{B}\boldsymbol{\tau} = \mathbf{0}_{2N\times1}, \quad (7)$$

where $\mathbf{A} = [\mathbf{a}_{11}, \mathbf{a}_{12}, \cdots, \mathbf{a}_{N1}, \mathbf{a}_{N2}]^T$ and $\mathbf{B} = [\mathbf{l}_1, \mathbf{l}_1, \cdots, \mathbf{l}_N, \mathbf{l}_N]^T$. We can treat (7) as a linear equation system in $\boldsymbol{\tau}$. Thus the closed-form expression for $\boldsymbol{\tau}$ is

$$\boldsymbol{\tau} = -\left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{A}\mathbf{r}. \quad (8)$$

Substituting (8) into (7), we have

$$\mathbf{K}\mathbf{r} = \mathbf{0}_{2N\times1}, \mathbf{K} = \mathbf{A} - \mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{A}. \quad (9)$$

We divide the monomials in $\mathbf{r}$ defined in (4) into two groups. The first group $\mathbf{r}_3$ includes 3 monomials, and the remaining one $\mathbf{r}_7$ has 7 terms. For instance, $\mathbf{r}_3 = \left[s_1^2, s_2^2, s_3^2\right]^T$ and $\mathbf{r}_7 = \left[s_1s_2, s_1s_3, s_2s_3, s_1, s_2, s_3, 1\right]^T$. Based on this division, the matrix $\mathbf{K}$ in (9) can be divided into $\mathbf{K}_3$ and $\mathbf{K}_7$, accordingly. Then we can rewrite (9) as

$$\mathbf{K}_3\mathbf{r}_3 + \mathbf{K}_7\mathbf{r}_7 = \mathbf{0}_{2N\times1}. \quad (10)$$

Moving $\mathbf{K}_7\mathbf{r}_7$ to the right hand side of (10), we have $\mathbf{K}_3\mathbf{r}_3 = -\mathbf{K}_7\mathbf{r}_7$. Here we treat the three elements in $\mathbf{r}_3$ as individual unknowns. Consequently, we can have a closed-form expression for $\mathbf{r}_3$ with respect to $\mathbf{r}_7$ as

$$\mathbf{r}_3 = -\left(\mathbf{K}_3^T\mathbf{K}_3\right)^{-1}\mathbf{K}_3^T\mathbf{K}_7\mathbf{r}_7, \quad (11)$$

where $-\left(\mathbf{K}_3^T\mathbf{K}_3\right)^{-1}\mathbf{K}_3^T\mathbf{K}_7$ is a 3×7 matrix. To simplify the notation, we define $\mathbf{C}_7 = \left(\mathbf{K}_3^T\mathbf{K}_3\right)^{-1}\mathbf{K}_3^T\mathbf{K}_7$. Then we can rewrite (11) as:

$$\mathbf{C}\mathbf{r} = \mathbf{0}_{3\times1}, \mathbf{C} = [\mathbf{I}_3, \mathbf{C}_7]. \quad (12)$$

The above equation system includes 3 second-order polynomial equations in s_1, s_2, s_3. Each of them has the form

$$\begin{aligned} f_i =& c_{i1}s_1^2 + c_{i2}s_2^2 + c_{i3}s_3^2 + c_{i4}s_1s_2 + c_{i5}s_1s_3 + \\ & c_{i6}s_2s_3 + c_{i7}s_1 + c_{i8}s_2 + c_{i9}s_3 + c_{i10} = 0, \end{aligned} \quad (13)$$

where $i = 1, 2, 3$. They form a MCR. This compressed quadratic system provides the minimal number of constraints to determine the 3 unknowns s_1, s_2, s_3. Although the above derivation is based on a least-squares problem, it is also applicable to the P3L problem.

One critical step of our algorithm is to divide the monomials of $\mathbf{r}$ into 2 groups to solve for $\mathbf{r}_3$ in (11). There are $84 = \binom{9}{3}$ different combinations. We require the coefficient matrix $\mathbf{K}_3$ for $\mathbf{r}_3$ is full rank. Let us define $\mathbf{K}_9$ as the coefficient matrix of $\left[s_1^2, s_2^2, s_3^2, s_1s_2, s_1s_3, s_2s_3, s_1, s_2, s_3\right]^T$ in (9). If $\mathbf{K}_9$ is full rank, we can arbitrarily select $\mathbf{r}_3$. However, we have the following theorem:

Theorem 1: In case of noiseless data, $\mathbf{K}_9$ is rank-deficient, given an arbitrary number of 2D-3D line correspondences.

The proof can be found in appendix. As $\mathbf{K}_9$ is ran-deficient, a certain input may make $\mathbf{K}_3$ either exactly or approximately rank-deficient, given a constant $\mathbf{r}_3$. We introduce a simple method below to solve this problem.

A. Robust Division

Let us define $Rank\left(\mathbf{K}_9\right)$ as the rank of $\mathbf{K}_9$. In general configuration for $N \geq 3$ lines, we have $Rank\left(\mathbf{K}_9\right) \geq 3$. This is because, otherwise, there are less than 3 linearly independent constraints, which are not enough to determine the 3 unknowns

s_1, s_2 and s_3. This contradicts to the fact that the pose can be determined by $N \geq 3$ correspondences. Therefore, although a $\mathbf{K}_3$ for a certain combination may be rank-deficient, we can always find a combination from the 84 possible combinations whose $\mathbf{K}_3$ is full rank.

Our method is motivated by the QR decomposition with column pivoting [34] that can find linearly independent columns of a matrix. Let us define $\mathbf{k}_n$ as the nth column of $\mathbf{K}_9$. Here we adopt the Gram-Schmidt process with column pivoting to select 3 linearly independent columns from $\mathbf{K}_9$ to generate $\mathbf{K}_3$. Specifically, we have

$$\begin{aligned} i =& \arg\max_n \left\|\mathbf{k}_n\right\|_2, \\ j =& \arg\max_{n\neq i} \left\|\bar{\mathbf{k}}_n\right\|_2, \bar{\mathbf{k}}_n = \mathbf{k}_n - \frac{\mathbf{k}_i \cdot \mathbf{k}_n}{\mathbf{k}_i \cdot \mathbf{k}_i}\mathbf{k}_i, \\ k =& \arg\max_{n\neq i,j} \left\|\tilde{\mathbf{k}}_n\right\|_2, \tilde{\mathbf{k}}_n = \bar{\mathbf{k}}_n - \frac{\bar{\mathbf{k}}_j \cdot \bar{\mathbf{k}}_n}{\bar{\mathbf{k}}_j \cdot \bar{\mathbf{k}}_j}\bar{\mathbf{k}}_j. \end{aligned} \tag{14}$$

Then the ith, jth and kth columns of $\mathbf{K}$ are selected to form $\mathbf{K}_3$, and the corresponding monomials form $\mathbf{r}_3$. The remaining coefficients and terms generate $\mathbf{K}_7$ and $\mathbf{r}_7$.

B. Hidden Variable Quadratic System Solver

We can obtain the rotation matrix by solving the quadratic equation system (13). Gröbner basis approach [15] is widely used to solve the polynomial system. Besides, Kukelova *et al.* [16] introduce an efficient solution for the equation system (13). Zhou *et al.* [5] improve the stability of [16]. However, these approaches all involve computing the inverse of a matrix, thus they may encounter numerical issues. This paper adopts a hidden variable method to solve (12). The general hidden variable method is computationally expensive [15]. But the equation system (13) can be efficiently solved by a customized hidden variable method by exploiting its special structure as described in [15]. Here we briefly introduce this method for completeness. The interested reader can find more details in [15].

We first treat one unknown in (13) as a constant. Without loss of generality, we treat s_3 as a constant, and s_1 and s_2 as unknowns. Then (13) can be written as

$$f_i = c_{i1}s_1^2 + c_{i2}s_2^2 + c_{i4}s_1s_2 + p_{i1}(s_3)s_1 + p_{i2}(s_3)s_2 + p_{i3}(s_3) = 0, \tag{15}$$

where $p_{i1}(s_3) = c_{i5}s_3 + c_{i7}$, $p_{i2}(s_3) = c_{i6}s_3 + c_{i8}$, and $p_{i3}(s_3) = c_{i3}s_3^2 + c_{i9}s_3 + c_{i10}$. Then we introduce an auxiliary variable s_0 to make (13) a homogeneous quadratic equation, which makes all the terms in (15) have degree 2. This generates the following equation system:

$$\begin{aligned} F_i =& c_{i1}s_1^2 + c_{i2}s_2^2 + c_{i4}s_1s_2 + p_{i1}(s_3)s_0s_1 + \\ & p_{i2}(s_3)s_0s_2 + p_{i3}(s_3)s_0^2 = 0. \end{aligned} \tag{16}$$

It is easy to see $F_i = f_i$, when we set $s_0 = 1$.

Let us denote $J(s_0, s_1, s_2)$ as the determinant of the Jacobian matrix of F_0, F_1 and F_2:

$$J(s_0, s_1, s_2) = det\begin{pmatrix} \frac{\partial F_0}{\partial s_0} & \frac{\partial F_0}{\partial s_1} & \frac{\partial F_0}{\partial s_2} \\ \frac{\partial F_1}{\partial s_0} & \frac{\partial F_1}{\partial s_1} & \frac{\partial F_1}{\partial s_2} \\ \frac{\partial F_2}{\partial s_0} & \frac{\partial F_2}{\partial s_1} & \frac{\partial F_2}{\partial s_2} \end{pmatrix}. \tag{17}$$

Let us define $G_i = \frac{\partial J}{\partial s_i}, i = 0, 1, 2$. According to [15], $G_0 = G_1 = G_2 = 0$ at all nontrivial solutions of $F_0 = F_1 = F_2 = 0$. G_i has the from

$$\begin{aligned} G_i =& \frac{\partial J}{\partial s_i} = q_{i1}(s_3)s_1^2 + q_{i2}(s_3)s_2^2 + q_{i3}(s_3)s_1s_2 + \\ & q_{i4}(s_3)s_0s_1 + q_{i5}(s_3)s_0s_2 + q_{i6}(s_3)s_0^2 = 0, \end{aligned} \tag{18}$$

where $q_{ij}(s_3)$ is a polynomial in s_3. G_i has the same form as F_i. We combine them to form a new homogeneous system with respect to s_0, s_1 and s_2 as

$$\mathbf{Q}(s_3)\mathbf{u} = \mathbf{0}_{6\times 1}, \tag{19}$$

where $\mathbf{Q}(s_3)$ is a 6×6 matrix whose elements are polynomials in s_3, and $\mathbf{u} = \left[s_0^2, s_1^2, s_2^2, s_0s_1, s_0s_2, s_1s_2\right]^T$. Based on linear algebraic theory, the homogeneous linear system (19) has a non-trivial solution if and only if $det(\mathbf{Q}(s_3)) = 0$. Here $det(\mathbf{Q}(s_3))$ is the determinant of $\mathbf{Q}(s_3)$. $det(\mathbf{Q}(s_3)) = 0$ is an eighth-order polynomial equation in s_3. There are at most 8 solutions. If a root is a complex number, we keep its real part. After we get s_3, we can back-substitute s_3 into (19). This leads to a linear homogeneous equation system with respect to $\mathbf{u}$. As mentioned above, $F_i = f_i$ when we set $s_0 = 1$. Consequentially, s_1 and s_2 can be computed through the linear system (19) by back-substituting s_3 into (19) and setting $s_0 = 1$. After we obtain s_1, s_2 and s_3, we can compute $\mathbf{R}$ by (2). Then $\boldsymbol{\tau}$ can be calculated by (5). Finally, we can obtain $\mathbf{t}$ by (8).

C. Solution Refinement

The above derivation is of algebraic nature. As in previous works [1], [2], [3], [14], [25], we also use an iterative method to refine the initial solution. We adopt the cost function introduced in [1], which approximates the reprojection error of the line. Specifically, the 3D line $\mathbf{L}_i$ in [1] is represented by the Plüker coordinates. The projection from $\mathbf{L}_i$ to the image plane can be written as $\hat{\mathbf{l}}_i = [\mathbf{R}, -\mathbf{R}[\mathbf{c}]_\times]\mathbf{L}_i$, where $\mathbf{c} = -\mathbf{R}^T\mathbf{t}$. Assume $\bar{\mathbf{p}}_{ij}, j = 1, 2$ are the homogeneous coordinates of the two normalized endpoints of the detected 2D line segment $\mathbf{l}_i$. The error from $\bar{\mathbf{p}}_{ij}$ to $\hat{\mathbf{l}}_i$ is $e = \frac{\hat{\mathbf{l}}_i \cdot \bar{\mathbf{p}}_{ij}}{\sqrt{\hat{\mathbf{l}}_i^1 + \hat{\mathbf{l}}_i^2}}$, where $\hat{\mathbf{l}}_i^1$ and $\hat{\mathbf{l}}_i^2$ are the first and second elements of $\hat{\mathbf{l}}_i$, respectively. In [1], e is approximated by fixing the denominator of e at the initial pose. This leads to a sixth-order polynomial cost in $\mathbf{s}$ and $\mathbf{t}$.

We adopt the damped Newton step to refine the solution. Specifically, for the kth step, we compute the Hessian $\mathbf{H}_k$ and the gradient $\mathbf{g}_k$ of the cost function with respect to $\mathbf{s}$ and $\mathbf{t}$. Then the solution is updated by $[\mathbf{s}_{k+1}, \mathbf{t}_{k+1}] = [\mathbf{s}_k, \mathbf{t}_k] - (\mathbf{H_k} + \lambda\mathbf{I_6})^{-1}\mathbf{g}_k$. Here λ is adjusted in each step according to the Levenberg-Marquardt (LM) algorithm [35] to make the cost reduce. We run at most 5 iterations.

D. Summary of Algorithm

Our PnL solution is applicable to $N \geq 3$ 2D-3D line correspondences. Basically, our algorithm has 4 steps. The first step is to compress the $2N$ constraints (3) into 3 equations (13). This equation system is solved by the introduced hidden variable method. Then we recover $\mathbf{R}$ and $\mathbf{t}$. Finally, we refine

Algorithm 1: Our PnL algorithm

Input: $N \geq 3$ 2D-3D line correspondences

Output: Camera Pose $\mathbf{R}$ and $\mathbf{t}$

1) Compress the $2N$ constraints (3) from N 2D-3D line correspondences:
   - Calculate $\boldsymbol{\tau}$ as a function of $\mathbf{s}$ by (8) and substitute it into (7) to generate a quadratic system (9) in $\mathbf{s}$.
   - Split the quadratic system (9) into two parts by the Gram-Schmidt process with column pivoting.
   - Solve for $\mathbf{r}_3$ to generate an equation system (13).
2) Solve (13) for $\mathbf{s}$ by the hidden variable method.
3) Compute $\mathbf{R}$ by (2), and $\mathbf{t}$ by (8) and (5) .
4) Refine the solution by the damped Newton step.

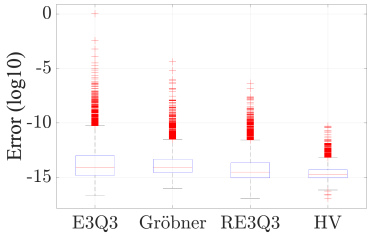


Fig. 2: Comparison of the numerical stability of the introduced hidden variable (**HV**) method, Gröbner [17], E3Q3 [16] and RE3Q3 [5].

the solution by the damped Newton step. We summarize our algorithm in Algorithm 1. As mentioned in [32], the value of the cost function may not enough to select the correct solution for some PnP inputs, and multiple minima are returned for these cases. We have observed this issue for the PnL problem as well. We return multiple solutions when the difference among the costs of physically feasible solutions is not obvious.

The computational complexity of step 2 and 3 is $O(1)$ as they are independent of the number of lines. The major computational cost in step 1 comes from solving the linear least-squares problem (8) and (11). The main computational cost of step 4 comes from calculating the summation of squared distance functions. The computational complexity of these steps increases linearly with respect to N. In summary, the computational complexity of our algorithm is $O(N)$.

IV. Experimental Results

In this section, we first investigate our algorithm, referred to as **MinPnL**. Then we compare our algorithm with previous P3L and least-squares algorithms. The compared algorithms are listed below:

- For the P3L problem, we compare our algorithm with **AlgP3L** [5], **RP3L** [3] and **SRP3L** [6].
- For the least-squares PnL problem, the following algorithms are compared: **OAPnL** [1], **SRPnL** [6] , **ASPnL** [3], **Ansar** [26], **Mirzaei** [13], **LPnL_DLT** [3], **DLT_Combined** [23], **DLT_Plucker** [24], **cvxPnPL**[4], **LPnL_Bar_LS** [3], **LPnL_Bar_ENull** [3], **OPnPL** [2], and **EPnPL_Planar** [2].

We adopt the metrics in [1], [24] to measure the estimation error. Specifically, assuming $\mathbf{R}_{gt}$ and $\mathbf{t}_{gt}$ are the ground truth

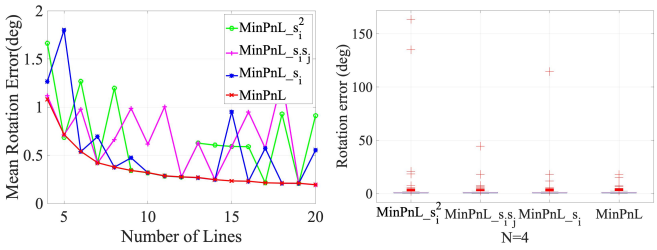


Fig. 3: The effect of the Gram-Schmidt process. Three choices of $\mathbf{r}_3$ are considered, including $[s_1^2, s_2^2, s_3^2]$, $[s_1s_2, s_1s_3, s_2s_3]$ and $[s_1, s_2, s_3]$, referred to as MinPnL_s_i^2, MinPnL_s_is_j and MinPnL_s_i.

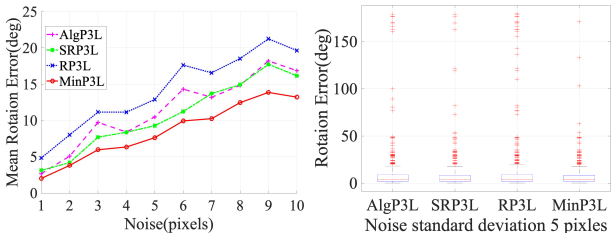


Fig. 4: Comparison of our algorithm with previous P3L algorithms including AlgP3L [5], RP3L [3] and SRP3L [6]. The left figure shows the mean rotation errors, and the right figure shows the boxplot of rotation errors at $\sigma = 5$ pixels.

rotation and translation, and $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$ the estimated ones, we calculate the rotation error $\Delta\theta$ as the angle (degree) of the axis-angle representation of $\mathbf{R}_{gt}^{-1}\hat{\mathbf{R}}$ and the translation error $\Delta\mathbf{t}$ as $\left\|\mathbf{t}_{gt} - \hat{\mathbf{t}}\right\|_2 / \left\|\mathbf{t}_{gt}\right\|_2 \times 100\%$.

A. Experiments with Synthetic Data

In this section, we use synthetic data to evaluate the performance of different algorithms. We first compare the solvers for equation system (13) and study the effect of the Gram-Schmidt process. Then we compare our algorithm with the state-of-the-art P3L and least-squares PnL algorithms.

Data Generation Our synthetic data are generated similar to [1], [3], [14]. Specifically, we set the camera resolution to 640×480 pixels and the focal length to 800. For each trial, the camera is randomly placed within a $[-10m, 10m]^3$ cube and the Euler angles of the rotation are uniformly sampled from $\alpha, \lambda \in [0°, 360°]$ and $\beta \in [0°, 180°]$. Then N 2D-3D line correspondences are randomly generated. We first randomly generate the endpoints of the 2D lines, then the endpoints of the corresponding 3D lines are generated by back-projecting the 2D endpoints into 3D. Depths of the 3D endpoints are within $[4m, 10m]$. Then these 3D endpoints are transformed to the world frame. Here the endpoints of the 2D and 3D lines are matched as in previous works [1], [3], [14]. Thus the solution of [2] is calculated without the line shift step, which aims to align the endpoints of the 2D and 3D lines, to fairly evaluate its accuracy and runtime.

Compare Polynomial Solvers We use the **boxplot** to present the estimation errors. In the boxplot, the central mark of each box indicates the median, and the bottom and top edges indicate the $25th$ and $75th$ percentiles, respectively. The whiskers are extended to ± 2.7 standard deviations, and the errors out of this range are plotted using the symbol "+".

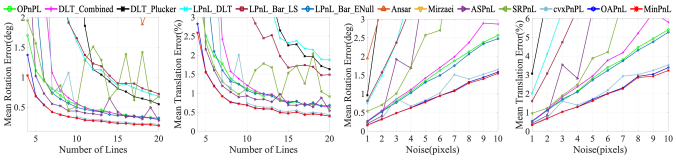
We compare the numerical stability of the introduced hidden variable (**HV**) method, **Gröbner** [17], **E3Q3** [16] and **RE3Q3** [5] using $10,000$ trials. The monomial coefficients and a solution $\mathbf{s}_{gt}$ of (13) are randomly generated from $(-1, 1)$. Then we substitute $\mathbf{s}_{gt}$ into (13) to calculate the constant items of (13). Assume $\hat{\mathbf{s}}$ is the estimated solution. We use $||\hat{\mathbf{s}} - \mathbf{s}_{gt}||_2/||\mathbf{s}_{gt}||_2$ to evaluate the estimation error. Fig. 2 shows the results. It is clear that the hidden variable solver is more stable than other solvers. As these solvers all involve computing the inverse of a matrix, they may generate a large error when the matrix is rank deficiency or near to that condition.

The Effect of the Gram-Schmidt Process One critical step of our algorithm is to reorganize $\mathbf{Kr} = \mathbf{0}$ in (9) as $\mathbf{K}_3\mathbf{r}_3 + \mathbf{K}_7\mathbf{r}_7 = \mathbf{0}$ in (10). There exist 84 choices for $\mathbf{r}_3$. Here we study the impact of different choices on numerical stability. We consider 3 choices of $\mathbf{r}_3$, *i.e.*, $\left[s_1^2, s_2^2, s_3^2\right]$, $[s_1s_2, s_1s_3, s_2s_3]$ and $[s_1, s_2, s_3]$, named as **MinPnL**_s_i^2, **MinPnL**_s_is_j and **MinPnL**_s_i, respectively. We increase the number of correspondences N from 4 to 20, and set the standard deviation of the noise to 2 pixels. For each N, we conduct 500 independent trials. Fig. 3 demonstrates the results. As shown in the boxplot of Fig. 3 for $N = 4$, the fixed choice of $\mathbf{r}_3$ may generate large errors as it may encounter a numerical problem when $\mathbf{K}_3$ is nearly rank-deficient. The Gram-Schmidt process solves this problem.

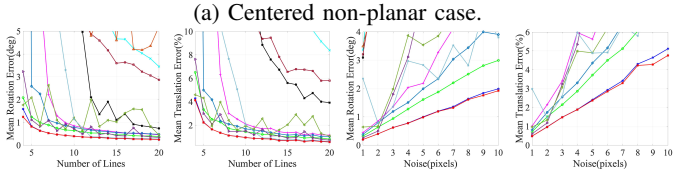
Compare P3L Algorithms We next compare our P3L algorithm (**MinP3L**) with previous P3L algorithms including AlgP3L [5], RP3L [3] and SRP3L [6]. To fairly compare the performance, our results are obtained without running the iterative refinement step, as the compared algorithms do not have such a refinement. We consider the behavior of the P3L algorithms under varying noise levels. We add Gaussian noise to the endpoints of the 2D lines. We conduct 500 independent trials for each noise level. The standard deviation σ increases from 1 to 10 pixels. Fig. 4 shows the results. Our algorithm is more robust than the compared ones. The compared algorithms [3], [5], [6] all have longer tails than our algorithm. This may be due to the numerically unstable operations in these algorithms, such as potentially vanishing denominators in equation (4) of [3] and (20) of [6], and the singular case of RE3Q3 in AlgP3L [5].

Compare Least-Squares Algorithms As done in previous works [1], [3], [14], we consider two configurations of the 2D line segments, including the **centered case** (the 2D line segments are uniformly distributed within the whole image) and the **uncentered case** (the 2D line segments are constrained within $[0, 160] \times [0, 120]$). The following results are from 500 independent trials.

In the first experiment, we consider the performance of the PnL algorithms w.r.t. varying number of correspondences. The standard deviation σ of the Gaussian noise added to the 2D line endpoints is set to 2 pixels. In the second experiment, we consider the situation of varying noise level. σ is from 1 pixel to 10 pixels, and N is set to 10. Fig. 5 shows the mean errors. Typically, solutions from polynomial formulations are more stable than linear solutions. OAPnL [1] and our algorithm achieve the best result among the compared algorithms. The

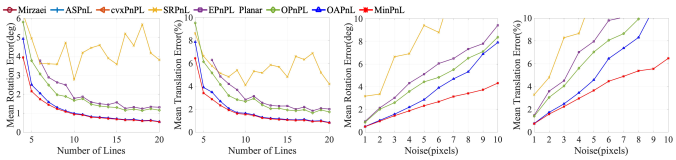


(a) Centered non-planar case.

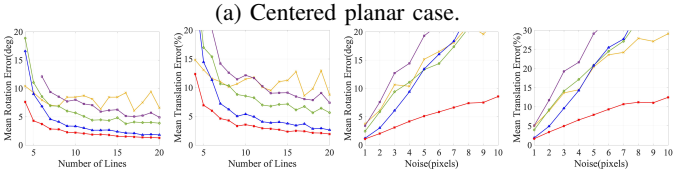


(b) Uncentered non-planar case.

Fig. 5: Comparison of our algorithm (**MinPnL**) with previous PnL algorithms w.r.t. increasing number of lines (**first two columns**, $N \in [4, 20]$, $\sigma = 2$ pixels) and increasing noise level (**last two columns**, $N = 10$, $\sigma \in [1, 10]$ pixels) for centered and uncentered **non-planar** cases.



(a) Centered planar case.



(b) Uncentered planar case.

Fig. 6: Comparison of our algorithm (**MinPnL**) with previous PnL algorithms w.r.t. increasing number of lines (**first two columns**, $N \in [4, 20]$, $\sigma = 2$ pixels) and increasing noise level (**last two columns**, $N = 10$, $\sigma \in [1, 10]$ pixels) for centered and uncentered **planar cases**.

results almost coincide, but our algorithm is more accurate than OAPnL when $N = 4$ or σ is large. cvxPnPL [4] provides comparable results only when $N \geq 10$. Other algorithms provide obviously larger errors.

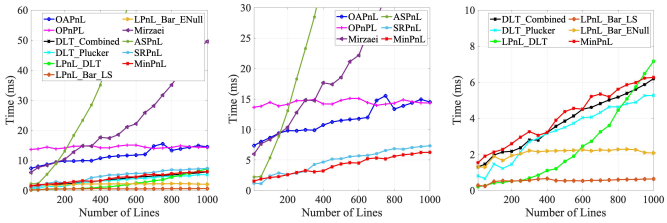
Furthermore, we also evaluate the accuracy of the PnL algorithms in the **planar configuration** (all the 3D lines are in a plane). The planar configuration widely exists in man-made environments. However, many PnL algorithms are infeasible for this configuration as shown in [1], [24]. Here we compare our algorithm with 7 PnL algorithms as shown in Fig. 6. Our algorithm achieves the best results. cvxPnPL[4], ASPnL [3] and Mizaei [28] generate large errors which are beyond the bounds of the figure. Although OAPnL [1] achieves similar results as our algorithm in the non-planar case, our algorithm is clearly more stable for the planar case.

Runtime We evaluate the runtime of different algorithms on a 3.1 Hz Intel Core i7 laptop with 16G memory using Matlab. We first compare the runtime of the P3L algorithms by 10,000 independent trials. The average computational time of our algorithm, RP3L [3], SRP3L [6] and AlgP3L [5] are $0.32ms$, $0.26ms$, $0.22ms$, $0.29ms$, respectively. Our algorithm is slightly slower than them, but is more robust as demonstrated in Fig.

TABLE I: Experimental results with real data. $\Delta\theta(^\circ)$ is the angle of the angle-axis representation of $\mathbf{R}_{gt}^{-1}\hat{\mathbf{R}}$. $\Delta\mathbf{t}(m)$ is the absolute translation error $\left\|\mathbf{t}_{gt}-\hat{\mathbf{t}}\right\|_2$. The best results are displayed in bold font.

Dataset		BB	STR	TFH	MH	COR	MC1	MC2	MC3	ULB	WDC
#images		66	20	72	10	11	3	3	3	3	5
#lines		870	1841	828	30	69	295	302	177	253	380
Mirzaei	$\Delta\theta$	88.18	0.90	32.24	0.46	0.22	4.83	15.47	5.00	2.51	36.52
	$\Delta\mathbf{t}$	168.47	1.92	11.04	0.04	0.10	1.53	7.37	1.82	1.27	6.44
DLT_Comb	$\Delta\theta$	0.40	0.22	0.39	0.41	0.11	0.11	0.15	0.16	0.20	0.23
	$\Delta\mathbf{t}$	1.88	0.38	0.32	0.04	0.04	0.04	0.07	0.05	0.08	0.12
DLT_Plücker	$\Delta\theta$	1.04	0.93	1.11	17.58	0.38	0.28	0.22	0.48	0.77	0.34
	$\Delta\mathbf{t}$	1.88	0.38	0.32	0.04	0.04	0.04	0.07	0.05	0.08	0.12
LPnL_ENull	$\Delta\theta$	0.30	0.11	0.57	0.32	0.10	0.04	0.03	0.07	0.39	0.08
	$\Delta\mathbf{t}$	1.13	0.16	0.45	0.022	0.04	0.01	0.02	0.02	0.18	0.05
LPnL_LS	$\Delta\theta$	1.98	0.15	1.10	0.45	0.13	0.03	0.03	0.09	0.49	0.18
	$\Delta\mathbf{t}$	7.23	0.27	1.05	0.04	0.05	0.01	0.02	0.03	0.22	0.11
ASPnL	$\Delta\theta$	37.82	22.08	7.76	0.25	0.10	0.15	0.20	2.08	4.89	0.51
	$\Delta\mathbf{t}$	76.61	30.47	6.11	0.018	0.03	0.04	0.08	0.74	2.22	0.23
SRPnL	$\Delta\theta$	36.92	0.14	3.40	0.25	0.063	0.58	0.15	48.53	5.58	0.11
	$\Delta\mathbf{t}$	42.98	0.20	2.67	0.018	0.021	0.14	0.066	5.25	2.75	0.049
OPnPL	$\Delta\theta$	0.19	0.09	0.42	0.28	0.07	0.03	0.03	0.06	0.06	0.13
	$\Delta\mathbf{t}$	0.81	0.07	0.31	0.02	0.02	0.01	0.01	0.02	0.02	0.06
cvxPnPL	$\Delta\theta$	**0.18**	0.081	0.40	0.25	0.055	0.035	0.025	0.065	0.12	0.10
	$\Delta\mathbf{t}$	**0.78**	0.058	0.31	0.018	0.019	0.011	0.013	0.020	0.048	0.047
OAPnL	$\Delta\theta$	**0.18**	**0.080**	0.10	0.22	**0.031**	0.011	**0.012**	**0.022**	**0.030**	**0.036**
	$\Delta\mathbf{t}$	**0.78**	0.029	0.070	**0.015**	**0.011**	$3.5e^{-3}$	$\mathbf{6.0}e^{-3}$	$\mathbf{7.4}e^{-3}$	**0.014**	**0.019**
MinPnL	$\Delta\theta$	**0.18**	**0.080**	**0.089**	**0.21**	**0.031**	**0.010**	0.013	**0.022**	**0.030**	**0.036**
	$\Delta\mathbf{t}$	**0.78**	**0.024**	**0.057**	**0.015**	**0.011**	$\mathbf{3.4}e^{-3}$	$6.3e^{-3}$	$\mathbf{7.4}e^{-3}$	**0.014**	**0.019**

4. Next, we compare the runtime of the least-squares PnL algorithms. The results are from 500 independent trials and are illustrated in Fig. 7. The algorithms Ansar [26] and cvxPnPL [4] are too slow for a large N and are not considered here. As shown in Fig 5 and 6, OAPnL [1] and our algorithm are generally the most accurate two algorithms. But our algorithm is significantly faster than OAPnL as shown in Fig. 7(b). In addition, our algorithm has similar runtime compared to the linear algorithm DLT_Combined [24] and DLT_Plucker [23], and slightly slower than LPnL_Bar_ENull [3] when N is less than 100, and faster than LPnL_DLT [3] when N is large.



(a) (b) (c)

Fig. 7: (a) Runtime for all the algorithms. (b) Runtime for algorithms with polynomial solvers. (c) Runtime for our algorithm and algorithms based on linear transformation.

B. Experiments with Real Data

In this section, we use the MPI[1] and VGG datasets[2] to evaluate the PnL algorithms. They include 10 datasets whose characteristics are listed in Table I. Here we adopt the absolute translation error $\left\|\mathbf{t}_{gt}-\hat{\mathbf{t}}\right\|_2$ instead of the relative error used for the synthetic data, as the ground truth translation is $[0;0;0]$ in some cases. Table I presents the results. We find that some algorithms generate large errors even for hundreds of lines, such as Mirzaei [28], ASPnL [3] and SRPnL [14] on the BB dataset. Our algorithm achieves the best result among the compared algorithms, except for the MC2 dataset where it is slightly less accurate than OAPnL [1].

V. Conclusions

This paper presents a complete, accurate and efficient solution for the PnL problem. Instead of formulating a least-squares problem into a minimization problem as in previous works, we compress the $2N$ constraints of N line correspondences into 3 quadratic equations with 3 unknowns. This establishes the connection between a least-squares problem and a minimal problem. We adopt the Gram-Schmidt process to avoid the numerical instability during the constraint compression. We solve this quadratic equation system using a novel hidden variable polynomial solver. We conduct extensive experiments, and the results show our algorithm is more accurate than the state-of-the-art least-squares algorithms [1], [2], [3], [4] especially under the challenging conditions (such as small numbers of lines, large noise and planar case) at a lower cost. On the other hand, our algorithm is more stable than the state-of-the-art P3L algorithms [3], [5], [6] with comparable runtime.

[1]MPI dataset http://resources.mpi-inf.mpg.de/LineReconstruction/

[2]VGG dataset http://www.robots.ox.ac.uk/ vgg/data/data-mview.html

Appendix

Proof. Here we prove Theorem 1. The constraints in (1) can be rewritten as $\mathbf{l}_i^T\mathbf{R}\mathbf{P}_{ij}+\mathbf{l}_i^T\mathbf{t}=0, \quad j=1,2$. Define R_{ij} as the element of $\mathbf{R}$ at the ith row and jth column, and $\mathbf{q}=[R_{11},R_{12},R_{13},R_{21},R_{22},R_{23},R_{31},R_{32},R_{33}]^T$. Expanding the above constraint, we have $\mathbf{w}_{ij}\mathbf{q}+\mathbf{l}_i^T\mathbf{t}=0$. We stack the $2N$ constraints to get

$$\mathbf{W}\mathbf{q}+\mathbf{B}\mathbf{t}=\mathbf{0}_{2N\times 1}. \tag{20}$$

The coefficient matrix $\mathbf{B}$ in (20) is the same one in (7). Similarly to (8), we can have a closed-form expression for $\mathbf{t}$ *i.e.* $\mathbf{t} = -\left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{W}\mathbf{q}$. Substituting it to (20), we get

$$\mathbf{L}\mathbf{q} = \mathbf{0}_{2N\times 1},\, \mathbf{L} = \mathbf{W} - \mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{W}. \qquad (21)$$

We have $Rank\left(\mathbf{L}\right) < 9$. This is because (21) is a homogeneous linear system. As $\mathbf{q}$ is a non-trivial solution of (21), according to linear algebraic theory, $\mathbf{L}$ should be rank-deficient, because otherwise this homogeneous system will only have a trivial (zero) solution, which is in contradiction to the fact that there is a non-trivial solution $\mathbf{q}$ for (21).

As we know, $\mathbf{q}$ is the vectorization of $\mathbf{R}$. Using the definition in (2), we find that every element in $\mathbf{q}$ is a linear combination of the elements in $\frac{\mathbf{r}}{1+\mathbf{s}^T\mathbf{s}}$. For instance, $R_{11} = [1,-1,-1,0,0,0,0,0,1]\frac{\mathbf{r}}{1+\mathbf{s}^T\mathbf{s}}$. Stacking these linear combinations, we have $\mathbf{q} = \frac{\mathbf{E}\mathbf{r}}{1+\mathbf{s}^T\mathbf{s}}$. Substituting this into (20) and multiplying both sides by $1+\mathbf{s}^T\mathbf{s}$ and using the definition of $\boldsymbol{\tau}$ in (5), we have $\mathbf{W}\mathbf{E}\mathbf{r} + \mathbf{B}\boldsymbol{\tau} = \mathbf{0}_{2N\times 1}$. Comparing the above equation with (7), we have $\mathbf{A} = \mathbf{W}\mathbf{E}$. Substituting it into (9) and using the expression of $\mathbf{L}$ in (21), we have $\mathbf{K} = \left(\mathbf{W} - \mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{W}\right)\mathbf{E} = \mathbf{L}\mathbf{E}$. According to algebraic theory, the rank of the product of two matrices is smaller than or equal to the smaller rank of the two matrices. As we know $Rank\left(\mathbf{L}\right) < 9$, thus we have $Rank\left(\mathbf{K}\right) \leq min\left(Rank\left(\mathbf{L}\right), Rank\left(\mathbf{E}\right)\right) < 9$. $\mathbf{K}_9$ is the first 9 columns of $\mathbf{K}$. Consequentially, we have $Rank(\mathbf{K}_9) \leq Rank(\mathbf{K}) < 9$. Thus $\mathbf{K}_9$ is rank-deficient.

□

REFERENCES

[1] L. Zhou, Y. Yang, M. Abello, and M. Kaess, "A robust and efficient algorithm for the pnl problem using algebraic distance to approximate the reprojection distance," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9307–9315.

[2] A. Vakhitov, J. Funke, and F. Moreno-Noguer, "Accurate and linear time pose estimation from points and lines," in *European Conference on Computer Vision*. Springer, 2016, pp. 583–599.

[3] C. Xu, L. Zhang, L. Cheng, and R. Koch, "Pose estimation from line correspondences: A complete analysis and a series of solutions," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1209–1222, 2016.

[4] S. Agostinho, J. Gomes, and A. Del Bue, "CvxPnPL: A unified convex solution to the absolute pose estimation problem from point and line correspondences," *arXiv preprint arXiv:1907.10545*, 2019.

[5] L. Zhou, J. Ye, and M. Kaess, "A stable algebraic camera pose estimation for minimal configurations of 2d/3d point and line correspondences," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 273–288.

[6] P. Wang, G. Xu, and Y. Cheng, "A novel algebraic solution to the perspective-three-line pose problem," *Computer Vision and Image Understanding*, p. 102711, 2018.

[7] R. Gomez-Ojeda, J. Briales, and J. Gonzalez-Jimenez, "PL-SVO: Semi-direct Monocular Visual Odometry by Combining Points and Line Segments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4211–4216.

[8] Y. Li, N. Brasch, Y. Wang, N. Navab, and F. Tombari, "Structure-SLAM: Low-Drift Monocular SLAM in Indoor Environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6583–6590, 2020.

[9] B. Micusik and H. Wildenauer, "Structure from motion with line segments under relaxed endpoint constraints," *International Journal of Computer Vision*, vol. 124, no. 1, pp. 65–79, 2017.

[10] S. Ramalingam, S. Bouaziz, and P. Sturm, "Pose estimation using both points and lines for geo-localization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4716–4723.

[11] F. Zhou, H. B.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ismar," in *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE, 2008, pp. 193–202.

[12] H. H. Chen, "Pose determination from line-to-plane correspondences: existence condition and closed-form solutions," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 530–541, 1991.

[13] F. M. Mirzaei and S. I. Roumeliotis, "Optimal estimation of vanishing points in a manhattan world," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 2454–2461.

[14] P. Wang, G. Xu, Y. Cheng, and Q. Yu, "Camera pose estimation from lines: a fast, robust and general method," *Machine Vision and Applications*, vol. 30, no. 4, pp. 603–614, 2019.

[15] D. A. Cox, J. Little, and D. O'shea, *Using algebraic geometry*. Springer Science & Business Media, 2006, vol. 185.

[16] Z. Kukelova, J. Heller, and A. Fitzgibbon, "Efficient intersection of three quadrics and applications in computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1799–1808.

[17] V. Larsson, K. Astrom, and M. Oskarsson, "Efficient solvers for minimal problems by syzygy-based reduction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 820–829.

[18] Y. Liu, T. S. Huang, and O. D. Faugeras, "Determination of camera location from 2-d to 3-d line and point correspondences," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 1, pp. 28–37, 1990.

[19] R. Kumar and A. R. Hanson, "Robust methods for estimating pose and a sensitivity analysis," *CVGIP: Image understanding*, vol. 60, no. 3, pp. 313–342, 1994.

[20] S. Christy and R. Horaud, "Iterative pose computation from line correspondences," *Computer vision and image understanding*, vol. 73, no. 1, pp. 137–144, 1999.

[21] F. Dornaika and C. Garcia, "Pose estimation using point and line correspondences," *Real-Time Imaging*, vol. 5, no. 3, pp. 215–230, 1999.

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

[23] B. Přibyl, P. Zemčík, and M. Čadík, "Camera pose estimation from lines using plücker coordinates," in *Proceedings of the British Machine Vision Conference (BMVC 2015)*, 2015.

[24] ——, "Absolute pose estimation from line correspondences using direct linear transformation," *Computer Vision and Image Understanding*, vol. 161, pp. 130–144, 2017.

[25] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[26] A. Ansar and K. Daniilidis, "Linear pose estimation from points or lines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 578–589, 2003.

[27] S. Li, C. Xu, and M. Xie, "A robust o (n) solution to the perspective-n-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1444–1450, 2012.

[28] F. M. Mirzaei and S. I. Roumeliotis, "Globally optimal pose estimation from line correspondences," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 5581–5588.

[29] H. Abdellali, R. Frohlich, and Z. Kato, "A direct least-squares solution to multi-view absolute and relative pose from 2d-3d perspective line pairs," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[30] L. Lecrosnier, R. Boutteau, P. Vasseur, X. Savatier, and F. Fraundorfer, "Camera pose estimation based on pnl with a known vertical direction," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3852–3859, 2019.

[31] J. A. Hesch and S. I. Roumeliotis, "A direct least-squares (dls) method for pnp," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 383–390.

[32] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi, "Revisiting the pnp problem: A fast, general and optimal solution," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2344–2351.

[33] L. Kneip, H. Li, and Y. Seo, "Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability," in *European Conference on Computer Vision*. Springer, 2014, pp. 127–142.

[34] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2012, vol. 3.

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