Computer Vision I
CSE 252A, Winter 2007
David Kriegman

Homography Estimation*

1. From 3D to 2D Coordinates

Under homography, we can write the transformation of points in 3D from camera 1 to camera 2 as:

$$\mathbf{X}_2 = H\mathbf{X}_1 \quad \mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^3 \tag{1}$$

In the image planes, using homogeneous coordinates, we have

$$\lambda_1\mathbf{x}_1 = \mathbf{X}_1, \quad \lambda_2\mathbf{x}_2 = \mathbf{X}_2, \quad \text{therefore} \quad \lambda_2\mathbf{x}_2 = H\lambda_1\mathbf{x}_1 \tag{2}$$

This means that $\mathbf{x}_2$ is equal to $H\mathbf{x}_1$ up to a scale (due to universal scale ambiguity). Note that $\mathbf{x}_2 \sim H\mathbf{x}_1$ is a *direct mapping* between points in the image planes. If it is known that some points all lie in a plane in an image[1], the image can be rectified directly without needing to recover and manipulate 3D coordinates.

2. Homography Estimation

To estimate H, we start from the equation $\mathbf{x}_2 \sim H\mathbf{x}_1$. Written element by element, in homogenous coordinates we get the following constraint:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow \mathbf{x}_2 = H\mathbf{x}_1 \tag{3}$$

In inhomogenous coordinates ($x_2' = x_2/z_2$ and $y_2' = y_2/z_2$),

$$x_2' = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1} \tag{4}$$

$$y_2' = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1} \tag{5}$$

Without loss of generality, set $z_1 = 1$ and rearrange:

$$x_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13} \tag{6}$$

$$y_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23} \tag{7}$$

We want to solve for H. Even though these inhomogeneous equations involve the coordinates nonlinearly, the coefficients of H appear linearly. Rearranging equations 6 and 7 we get,

$$\mathbf{a}_x^T\mathbf{h} = \mathbf{0} \tag{8}$$

$$\mathbf{a}_y^T\mathbf{h} = \mathbf{0} \tag{9}$$

*Adapted from lecture notes from CSE252b Spring 2004 with permission from Serge Belongie.

[1] For cameras related by a pure rotation, every scene point can be thought of as lying on a plane at infinity.

where

$$\mathbf{h} = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \quad (10)$$

$$\mathbf{a}_x = (-x_1, -y_1, -1, 0, 0, 0, x'_2 x_1, x'_2 y_1, x'_2)^T \quad (11)$$

$$\mathbf{a}_y = (0, 0, 0, -x_1, -y_1, -1, y'_2 x_1, y'_2 y_1, y'_2)^T . \quad (12)$$

Given a set of corresponding points, we can form the following linear system of equations,

$$A\mathbf{h} = \mathbf{0} \quad (13)$$

where

$$A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix} . \quad (14)$$

Equation 13 can be solved using homogeneous linear least squares, described in the next section.

3. Homogeneous Linear Least Squares

We will frequently encounter problems of the form

$$A\mathbf{x} = \mathbf{0} \quad (15)$$

known as the Homogeneous Linear Least Squares problem. It is similar in appearance to the inhomogeneous linear least squares problem

$$A\mathbf{x} = \mathbf{b} \quad (16)$$

in which case we solve for $\mathbf{x}$ using the pseudoinverse or inverse of A. This won't work with Equation 15. Instead we solve it using Singular Value Decomposition (SVD).

Starting with equation 13 from the previous section, we first compute the SVD of A:

$$A = U\Sigma V^\top = \sum_{i=1}^{9} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \quad (17)$$

When performed in Matlab, the singular values σ_i will be sorted in descending order, so σ_9 will be the smallest. There are three cases for the value of σ_9:

- If the homography is *exactly determined*, then $\sigma_9 = 0$, and there exists a homography that fits the points exactly.
- If the homography is *overdetermined*, then $\sigma_9 \geq 0$. Here σ_9 represents a "residual" or goodness of fit.
- We will not handle the case of the homography being *underdetermined*.

From the SVD we take the "right singular vector" (a column from V) which corresponds to the smallest singular value, σ_9. This is the solution, $\mathbf{h}$, which contains the coefficients of the homography matrix that best fits the points. We reshape $\mathbf{h}$ into the matrix H, and form the equation $\mathbf{x}_2 \sim H\mathbf{x}_1$.

4. Homogeneous Linear Least Squares Alternate Derivation

Starting again with the equation $A\mathbf{h} = 0$, the sum squared error can be written as,

$$f(\mathbf{h}) = \frac{1}{2} (A\mathbf{h} - \mathbf{0})^T (A\mathbf{h} - \mathbf{0}) \quad (18)$$

$$f(\mathbf{h}) = \frac{1}{2} (A\mathbf{h})^T (A\mathbf{h}) \quad (19)$$

$$f(\mathbf{h}) = \frac{1}{2} \mathbf{h}^T A^T A\mathbf{h}. \quad (20)$$

Taking the derivative of f with respect to $\mathbf{h}$ and setting the result to zero, we get

$$\frac{d}{d\mathbf{h}} f = 0 = \frac{1}{2}\left(A^T A + (A^T A)^T\right)\mathbf{h} \tag{21}$$

$$0 = A^T A\mathbf{h}. \tag{22}$$

Looking at the eigen-decomposition of $A^T A$, we see that $\mathbf{h}$ should equal the eigenvector of $A^T A$ that has an eigenvalue of zero (or, in the presence of noise the eigenvalue closest to zero).

This result is identical to the result obtained using SVD, which is easily seen from the following fact,

Fact 1 *Given a matrix A with SVD decomposition $A = U\Sigma V^T$, the columns of V correspond to the eigenvectors of $A^T A$.*

Least-Squares Rigid Motion Using SVD

Olga Sorkine-Hornung and Michael Rabinovich

Department of Computer Science, ETH Zurich

January 16, 2017

Abstract

This note summarizes the steps to computing the best-fitting rigid transformation that aligns two sets of corresponding points.

Keywords: Shape matching, rigid alignment, rotation, SVD

1 Problem statement

Let $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ and $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ be two sets of corresponding points in $\mathbb{R}^d$. We wish to find a rigid transformation that optimally aligns the two sets in the least squares sense, i.e., we seek a rotation R and a translation vector $\mathbf{t}$ such that

$$(R, \mathbf{t}) = \underset{R \in SO(d),\ \mathbf{t} \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2, \tag{1}$$

where $w_i > 0$ are weights for each point pair.

In the following we detail the derivation of R and $\mathbf{t}$; readers that are interested in the final recipe may skip the proofs and go directly Section 4.

2 Computing the translation

Assume R is fixed and denote $F(\mathbf{t}) = \sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2$. We can find the optimal translation by taking the derivative of F w.r.t. $\mathbf{t}$ and searching for its roots:

$$\begin{aligned} 0 \;&=\; \frac{\partial F}{\partial \mathbf{t}} = \sum_{i=1}^{n} 2w_i \left(R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i \right) = \\ &=\; 2\mathbf{t} \left(\sum_{i=1}^{n} w_i \right) + 2R \left(\sum_{i=1}^{n} w_i \mathbf{p}_i \right) - 2 \sum_{i=1}^{n} w_i \mathbf{q}_i. \end{aligned} \tag{2}$$

Denote

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i}{\sum_{i=1}^{n} w_i}, \quad \bar{\mathbf{q}} = \frac{\sum_{i=1}^{n} w_i \mathbf{q}_i}{\sum_{i=1}^{n} w_i}. \tag{3}$$

By rearranging the terms of (2) we get

$$\mathbf{t} = \bar{\mathbf{q}} - R\bar{\mathbf{p}}. \tag{4}$$

In other words, the optimal translation $\mathbf{t}$ maps the transformed weighted centroid of $\mathcal{P}$ to the weighted centroid of $\mathcal{Q}$. Let us plug the optimal $\mathbf{t}$ into our objective function:

$$\begin{aligned} \sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2 &= \sum_{i=1}^{n} w_i \left\| R\mathbf{p}_i + \bar{\mathbf{q}} - R\bar{\mathbf{p}} - \mathbf{q}_i \right\|^2 = & (5) \\ &= \sum_{i=1}^{n} w_i \left\| R(\mathbf{p}_i - \bar{\mathbf{p}}) - (\mathbf{q}_i - \bar{\mathbf{q}}) \right\|^2 . & (6) \end{aligned}$$

We can thus concentrate on computing the rotation R by restating the problem such that the translation would be zero:

$$\mathbf{x}_i := \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{y}_i := \mathbf{q}_i - \bar{\mathbf{q}}. \tag{7}$$

So we look for the optimal rotation R such that

$$R = \underset{R \in SO(d)}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \left\| R\mathbf{x}_i - \mathbf{y}_i \right\|^2 . \tag{8}$$

3 Computing the rotation

Let us simplify the expression we are trying to minimize in (8):

$$\begin{aligned} \left\| R\mathbf{x}_i - \mathbf{y}_i \right\|^2 &= (R\mathbf{x}_i - \mathbf{y}_i)^\mathsf{T}(R\mathbf{x}_i - \mathbf{y}_i) = (\mathbf{x}_i^\mathsf{T} R^\mathsf{T} - \mathbf{y}_i^\mathsf{T})(R\mathbf{x}_i - \mathbf{y}_i) = \\ &= \mathbf{x}_i^\mathsf{T} R^\mathsf{T} R\mathbf{x}_i - \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i - \mathbf{x}_i^\mathsf{T} R^\mathsf{T} \mathbf{y}_i + \mathbf{y}_i^\mathsf{T} \mathbf{y}_i = \\ &= \mathbf{x}_i^\mathsf{T} \mathbf{x}_i - \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i - \mathbf{x}_i^\mathsf{T} R^\mathsf{T} \mathbf{y}_i + \mathbf{y}_i^\mathsf{T} \mathbf{y}_i. \end{aligned} \tag{9}$$

We got the last step by remembering that rotation matrices imply $R^\mathsf{T} R = I$ (I is the identity matrix).

Note that $\mathbf{x}_i^\mathsf{T} R^\mathsf{T} \mathbf{y}_i$ is a scalar: $\mathbf{x}_i^\mathsf{T}$ has dimension $1 \times d$, R^T is $d \times d$ and $\mathbf{y}_i$ is $d \times 1$. For any scalar a we trivially have $a = a^\mathsf{T}$, therefore

$$\mathbf{x}_i^\mathsf{T} R^\mathsf{T} \mathbf{y}_i = (\mathbf{x}_i^\mathsf{T} R^\mathsf{T} \mathbf{y}_i)^\mathsf{T} = \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i . \tag{10}$$

Therefore we have

$$\left\| R\mathbf{x}_i - \mathbf{y}_i \right\|^2 = \mathbf{x}_i^\mathsf{T} \mathbf{x}_i - 2\mathbf{y}_i^\mathsf{T} R\mathbf{x}_i + \mathbf{y}_i^\mathsf{T} \mathbf{y}_i. \tag{11}$$

Let us look at the minimization and substitute the above expression:

$$\begin{aligned} &\underset{R \in SO(d)}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \left\| R\mathbf{x}_i - \mathbf{y}_i \right\|^2 = \underset{R \in SO(d)}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (\mathbf{x}_i^\mathsf{T} \mathbf{x}_i - 2\mathbf{y}_i^\mathsf{T} R\mathbf{x}_i + \mathbf{y}_i^\mathsf{T} \mathbf{y}_i) = \\ = &\underset{R \in SO(d)}{\operatorname{argmin}} \left(\sum_{i=1}^{n} w_i \mathbf{x}_i^\mathsf{T} \mathbf{x}_i - 2 \sum_{i=1}^{n} w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i + \sum_{i=1}^{n} w_i \mathbf{y}_i^\mathsf{T} \mathbf{y}_i \right) = \\ = &\underset{R \in SO(d)}{\operatorname{argmin}} \left(-2 \sum_{i=1}^{n} w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i \right). \end{aligned} \tag{12}$$

$$\begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix} \begin{bmatrix} — & \mathbf{y}_1^\mathsf{T} & — \\ — & \mathbf{y}_2^\mathsf{T} & — \\ & \vdots & \\ — & \mathbf{y}_n^\mathsf{T} & — \end{bmatrix} \begin{bmatrix} & & \\ & R & \\ & & \end{bmatrix} \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_n \\ | & | & & | \end{bmatrix} =$$

$$= \begin{bmatrix} — & w_1\mathbf{y}_1^\mathsf{T} & — \\ — & w_2\mathbf{y}_2^\mathsf{T} & — \\ & \vdots & \\ — & w_n\mathbf{y}_n^\mathsf{T} & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ R\mathbf{x}_1 & R\mathbf{x}_2 & \dots & R\mathbf{x}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} w_1\mathbf{y}_1^\mathsf{T}R\mathbf{x}_1 & & & * \\ & w_2\mathbf{y}_2^\mathsf{T}R\mathbf{x}_2 & & \\ & & \ddots & \\ * & & & w_n\mathbf{y}_n^\mathsf{T}R\mathbf{x}_n \end{bmatrix}$$

Figure 1: Schematic explanation of $\sum_{i=1}^n w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i = \mathrm{tr}(WY^\mathsf{T}RX)$.

The last step (removing $\sum_{i=1}^n w_i \mathbf{x}_i^\mathsf{T}\mathbf{x}_i$ and $\sum_{i=1}^n w_i \mathbf{y}_i^\mathsf{T}\mathbf{y}_i$) holds because these expressions do not depend on R at all, so excluding them would not affect the minimizer. The same holds for multiplication of the minimization expression by a scalar, so we have

$$\underset{R\in SO(d)}{\operatorname{argmin}} \left(-2 \sum_{i=1}^n w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i \right) = \underset{R\in SO(d)}{\operatorname{argmax}} \sum_{i=1}^n w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i. \tag{13}$$

We note that

$$\sum_{i=1}^n w_i \mathbf{y}_i^\mathsf{T} R\mathbf{x}_i = \mathrm{tr}\left(WY^\mathsf{T}RX\right), \tag{14}$$

where $W = \mathrm{diag}(w_1, \dots, w_n)$ is an $n \times n$ diagonal matrix with the weight w_i on diagonal entry i; Y is the $d \times n$ matrix with $\mathbf{y}_i$ as its columns and X is the $d \times n$ matrix with $\mathbf{x}_i$ as its columns. We remind the reader that the trace of a square matrix is the sum of the elements on the diagonal: $\mathrm{tr}(A) = \sum_{i=1}^n a_{ii}$. See Figure 1 for an illustration of the algebraic manipulation.

Therefore we are looking for a rotation R that maximizes $\mathrm{tr}\left(WY^\mathsf{T}RX\right)$. Matrix trace has the property

$$\mathrm{tr}(AB) = \mathrm{tr}(BA) \tag{15}$$

for any matrices A, B of compatible dimensions. Therefore

$$\mathrm{tr}\left(WY^\mathsf{T}RX\right) = \mathrm{tr}\left((WY^\mathsf{T})(RX)\right) = \mathrm{tr}\left(RXWY^\mathsf{T}\right). \tag{16}$$

Let us denote the $d \times d$ "covariance" matrix $S = XWY^\mathsf{T}$. Take SVD of S:

$$S = U\Sigma V^\mathsf{T}. \tag{17}$$

Now substitute the decomposition into the trace we are trying to maximize:

$$\mathrm{tr}\left(RXWY^\mathsf{T}\right) = \mathrm{tr}\left(RS\right) = \mathrm{tr}\left(RU\Sigma V^\mathsf{T}\right) = \mathrm{tr}\left(\Sigma V^\mathsf{T}RU\right). \tag{18}$$

The last step was achieved using the property of trace (15). Note that V, R and U are all orthogonal matrices, so $M = V^\mathsf{T}RU$ is also an orthogonal matrix. This means that the columns of M are orthonormal vectors, and in particular, $\mathbf{m}_j^\mathsf{T}\mathbf{m}_j = 1$ for each M's column $\mathbf{m}_j$. Therefore all entries m_{ij} of M are ≤ 1 in magnitude:

$$1 = \mathbf{m}_j^\mathsf{T}\mathbf{m}_j = \sum_{i=1}^d m_{ij}^2 \;\Rightarrow\; m_{ij}^2 \leq 1 \;\Rightarrow\; |m_{ij}| \leq 1\,. \tag{19}$$

So what is the maximum possible value for $tr(\Sigma M)$? Remember that Σ is a diagonal matrix with non-negative values $\sigma_1, \sigma_2, \ldots, \sigma_d \geq 0$ on the diagonal. Therefore:

$$\operatorname{tr}(\Sigma M) = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_d \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & \ldots & m_{1d} \\ m_{21} & m_{22} & \ldots & m_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ m_{d1} & m_{d2} & \ldots & m_{dd} \end{pmatrix} = \sum_{i=1}^{d} \sigma_i m_{ii} \leq \sum_{i=1}^{d} \sigma_i. \tag{20}$$

Therefore the trace is maximized if $m_{ii} = 1$. Since M is an orthogonal matrix, this means that M would have to be the identity matrix!

$$I = M = V^{\mathsf{T}} R U \;\Rightarrow\; V = RU \;\Rightarrow\; R = VU^{\mathsf{T}}. \tag{21}$$

Orientation rectification. The process we just described finds the optimal *orthogonal* matrix, which could potentially contain reflections in addition to rotations. Imagine that the point set $\mathcal{P}$ is a perfect reflection of $\mathcal{Q}$. We will then find that reflection, which aligns the two point sets perfectly and yields zero energy in (8), the global minimum in this case. However, if we restrict ourselves to rotations only, there might not be a rotation that perfectly aligns the points.

Checking whether $R = VU^{\mathsf{T}}$ is a rotation is simple: if $\det(VU^{\mathsf{T}}) = -1$ then it contains a reflection, otherwise $\det(VU^{\mathsf{T}}) = +1$. Assume $\det(VU^{\mathsf{T}}) = -1$. Restricting R to a rotation is equivalent to restricting M to a reflection. We now want to find a reflection M that maximizes:

$$\operatorname{tr}(\Sigma M) = \sigma_1 m_{11} + \sigma_2 m_{22} + \ldots + \sigma_d m_{dd} =: f(m_{11}, \ldots, m_{dd}). \tag{22}$$

Note that f only depends on the diagonal of M, not its other entries. We now consider the m_{ii}'s as variables $(m_{11}, \ldots, m_{dd})$. This is the set of all diagonals of reflection matrices of order n. Surprisingly, it has a very simple structure. Indeed, a result by A. Horn [1] states that the set of all diagonals of rotation matrices of order n is equal to the convex hull of the points $(\pm 1, ..., \pm 1)$ with an even number of coordinates that are -1. Since any reflection matrix can be constructed by inverting the sign of a row of a rotation matrix and vice versa, it follows that the set we are optimizing on is the convex hull of the points $(\pm 1, ..., \pm 1)$ with an *uneven* number of -1's.

Since our domain is a convex polyhedron, the linear function f attains its extrema at its vertices. The diagonal $(1, 1, \ldots, 1)$ is not in the domain since it has an even number of -1's (namely, zero), and therefore the next best shot is $(1, 1, \ldots, 1, -1)$:

$$\operatorname{tr}(\Sigma M) = \sigma_1 + \sigma_2 + \ldots + \sigma_{d-1} - \sigma_d. \tag{23}$$

This value is attained at a vertex of our domain, and is larger than any other combination of the form $(\pm 1, ..., \pm 1)$ except $(1, 1, \ldots, 1)$, because σ_d is the smallest singular value.

To summarize, we arrive at the fact that if $\det(VU^{\mathsf{T}}) = -1$, we need

$$M = V^{\mathsf{T}} R U = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & -1 \end{pmatrix} \quad \Rightarrow \quad R = V \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & -1 \end{pmatrix} U^{\mathsf{T}}. \tag{24}$$

We can write a general formula that encompasses both cases, $\det(VU^{\mathsf{T}}) = 1$ and $\det(VU^{\mathsf{T}}) = -1$:

$$R = V \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & \det(VU^{\mathsf{T}}) \end{pmatrix} U^{\mathsf{T}}. \tag{25}$$

4 Rigid motion computation – summary

Let us summarize the steps to computing the optimal translation $\mathbf{t}$ and rotation R that minimize

$$\sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2 .$$

1. Compute the weighted centroids of both point sets:

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i}{\sum_{i=1}^{n} w_i}, \quad \bar{\mathbf{q}} = \frac{\sum_{i=1}^{n} w_i \mathbf{q}_i}{\sum_{i=1}^{n} w_i} .$$

2. Compute the centered vectors

$$\mathbf{x}_i := \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{y}_i := \mathbf{q}_i - \bar{\mathbf{q}}, \qquad i = 1, 2, \ldots, n.$$

3. Compute the $d \times d$ covariance matrix

$$S = XWY^{\mathsf{T}},$$

where X and Y are the $d \times n$ matrices that have $\mathbf{x}_i$ and $\mathbf{y}_i$ as their columns, respectively, and $W = \mathrm{diag}(w_1, w_2, \ldots, w_n)$.

4. Compute the singular value decomposition $S = U\Sigma V^{\mathsf{T}}$. The rotation we are looking for is then

$$R = V \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & \det(VU^{\mathsf{T}}) \end{pmatrix} U^{\mathsf{T}}.$$

5. Compute the optimal translation as

$$\mathbf{t} = \bar{\mathbf{q}} - R\bar{\mathbf{p}}.$$

Acknowledgments

We are grateful to Julian Panetta, Zohar Levi and Mario Botsch for their help with improving this note.

References

[1] A. Horn, *Doubly stochastic matrices and the diagonal of a rotation matrix.* Amer. J. Math. 76:620–630 (1954).

A Flexible New Technique for Camera Calibration

Zhengyou Zhang, *Senior Member, IEEE*

Abstract—We propose a flexible new technique to easily calibrate a camera. It only requires the camera to observe a planar pattern shown at a few (at least two) different orientations. Either the camera or the planar pattern can be freely moved. The motion need not be known. Radial lens distortion is modeled. The proposed procedure consists of a closed-form solution, followed by a nonlinear refinement based on the maximum likelihood criterion. Both computer simulation and real data have been used to test the proposed technique and very good results have been obtained. Compared with classical techniques which use expensive equipment such as two or three orthogonal planes, the proposed technique is easy to use and flexible. It advances 3D computer vision one more step from laboratory environments to real world use. The corresponding software is available from the author's Web page.

Index Terms—Camera calibration, calibration from planes, 2D pattern, flexible plane-based calibration, absolute conic, projective mapping, lens distortion, closed-form solution, maximum likelihood estimation, flexible setup.

1 Motivations

CAMERA calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images. Much work has been done, starting in the photogrammetry community (see [2], [4] to cite a few), and more recently in computer vision ([9], [8], [23], [7], [25], [24], [16], [6] to cite a few). We can classify those techniques roughly into two categories: photogrammetric calibration and self-calibration.

- **Three-dimensional reference object-based calibration.** Camera calibration is performed by observing a calibration object whose geometry in 3D space is known with very good precision. Calibration can be done very efficiently [5]. The calibration object usually consists of two or three planes orthogonal to each other. Sometimes a plane undergoing a precisely known translation is also used [23]. These approaches require an expensive calibration apparatus, and an elaborate setup.
- **Self-calibration.** Techniques in this category do not use any calibration object. Just by moving a camera in a static scene, the rigidity of the scene provides in general two constraints [16], [14] on the cameras' internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow us to reconstruct 3D structure up to a similarity [15], [12]. While this approach is very flexible, it is not yet mature [1]. Because there are many parameters to estimate, we cannot always obtain reliable results.

Other techniques exist: vanishing points for orthogonal directions [3], [13], and calibration from pure rotation [11], [20].

The author is with Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399. E-mail: zhang@microsoft.com.

Manuscript received 26 Apr. 2000; revised 25 Aug. 2000; accepted 7 Sept. 2000.
Recommended for acceptance by A. Shashua.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 111998.

Our current research is focused on a desktop vision syste (DVS) since the potential for using DVSs is large. Cameras are becoming inexpensive and ubiquitous. A DVS aims at the general public who are not experts in computer vision. A typical computer user will perform vision tasks only from time to time, so they will not be willing to invest money for expensive equipment. Therefore, flexibility, robustness, and low cost are important. The camera calibration technique described in this paper was developed with these considerations in mind.

The proposed technique only requires the camera to observe a planar pattern shown at a few (at least two) different orientations. The pattern can be printed on a laser printer and attached to a "reasonable" planar surface (e.g., a hard book cover). Either the camera or the planar pattern can be moved by hand. The motion need not be known. The proposed approach, which uses 2D metric information, lies between the photogrammetric calibration, which uses explicit 3D model, and self-calibration, which uses motion rigidity or equivalently implicit 3D information. Both computer simulation and real data have been used to test the proposed technique and very good results have been obtained. Compared with classical techniques, the proposed technique is considerably more flexible: Anyone can make a calibration pattern by him/herself and the setup is very easy. Compared with self-calibration, it gains a considerable degree of robustness. We believe the new technique advances 3D computer vision one step from laboratory environments to the real world.

Note that Triggs [22] recently developed a self-calibration technique from at least five views of a planar scene. His technique is more flexible than ours, but has difficulty to initialize. Liebowitz and Zisserman [13] described a technique of metric rectification for perspective images of planes using metric information, such as a known angle, two equal though unknown angles, and a known length ratio. They also mentioned that calibration of the internal camera parameters is possible provided at least three such rectified planes, although no experimental results were shown.

During the revision of this paper, we notice the publication of an independent but similar work by Sturm and Maybank [21].They use a simplified camera model (image axes are orthogonal to each other) and have studied the degenerate configurations exhaustively for the case of one and two planes, which are very important in practice if only one or two views are used for camera calibration.

The paper is organized as follows: Section 2 describes the basic constraints from observing a single plane. Section 3 describes the calibration procedure. We start with a closed-form solution, followed by nonlinear optimization. Radial lens distortion is also modeled. Section 4 provides the experimental results. Both computer simulation and real data are used to validate the proposed technique. In the Appendix, we provides a number of details, including the techniques for estimating the homography between the model plane and its image.

2 Basic Equations

We examine the constraints on the camera's intrinsic parameters provided by observing a single plane. We start with the notation used in this paper.

2.1 Notation

A 2D point is denoted by $\mathbf{m} = [u, v]^T$. A 3D point is denoted by $\mathbf{M} = [X, Y, Z]^T$. We use $\tilde{\mathbf{x}}$ to denote the augmented vector by adding 1 as the last element: $\tilde{\mathbf{m}} = [u, v, 1]^T$ and $\widetilde{\mathbf{M}} = [X, Y, Z, 1]^T$. A camera is modeled by the usual pinhole: The relationship between a 3D point $\mathbf{M}$ and its image projection $\mathbf{m}$ is given by

$$s\tilde{\mathbf{m}} = \mathbf{A}[\mathbf{R} \quad \mathbf{t}]\widetilde{\mathbf{M}}, \qquad \text{with } \mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where s is an arbitrary scale factor, $(\mathbf{R}, \mathbf{t})$, called the extrinsic parameters is the rotation and translation which relates the world coordinate system to the camera coordinate system, and $\mathbf{A}$ is called the camera intrinsic matrix, with (u_0, v_0) the coordinates of the principal point, α and β the scale factors in image u and v axes, and γ the parameter describing the skew of the two image axes.

We use the abbreviation $\mathbf{A}^{-T}$ for $(\mathbf{A}^{-1})^T$ or $(\mathbf{A}^T)^{-1}$.

2.2 Homography between the Model Plane and Its Image

Without loss of generality, we assume the model plane is on $Z = 0$ of the world coordinate system. Let's denote the ith column of the rotation matrix $\mathbf{R}$ by $\mathbf{r}_i$. From (1), we have

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{A}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \mathbf{A}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}.$$

By abuse of notation, we still use $\mathbf{M}$ to denote a point on the model plane, but $\mathbf{M} = [X, Y]^T$ since Z is always equal to zero. In turn, $\widetilde{\mathbf{M}} = [X, Y, 1]^T$. Therefore, a model point $\mathbf{M}$ and its image $\mathbf{m}$ is related by a homography $\mathbf{H}$:

$$s\widetilde{\mathbf{m}} = \mathbf{H}\widetilde{\mathbf{M}} \quad \text{with} \quad \mathbf{H} = \mathbf{A}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}. \tag{2}$$

As is clear, the 3×3 matrix $\mathbf{H}$ is defined up to a scale factor.

2.3 Constraints on the Intrinsic Parameters

Given an image of the model plane, an homography can be estimated (see the Appendix). Let's denote it by $\mathbf{H} = [\mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3]$. From (2), we have

$$[\mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3] = \lambda\mathbf{A}[\mathbf{r}_1 \quad \mathbf{r}_2 \quad \mathbf{t}],$$

where λ is an arbitrary scalar. Using the knowledge that $\mathbf{r}_1$ and $\mathbf{r}_2$ are orthonormal, we have

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 = 0 \tag{3}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2. \tag{4}$$

These are the two basic constraints on the intrinsic parameters, given one homography. Because a homography has 8 degrees of freedom and there are six extrinsic parameters (three for rotation and three for translation), we can only obtain two constraints on the intrinsic parameters. Note that $\mathbf{A}^{-T}\mathbf{A}^{-1}$ actually describes the image of the absolute conic [15]. In the next section, we will give a geometric interpretation.

2.4 Geometric Interpretation

We are now relating (3) and (4) to the absolute conic [16], [15].

It is not difficult to verify that the model plane, under our convention, is described in the camera coordinate system by the following equation:

$$\begin{bmatrix} \mathbf{r}_3 \\ \mathbf{r}_3^T\mathbf{t} \end{bmatrix}^T\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0,$$

where $w = 0$ for points at infinity and $w = 1$ otherwise. This plane intersects the plane at infinity at a line and we can easily see that

$$\begin{bmatrix} \mathbf{r}_1 \\ 0 \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{r}_2 \\ 0 \end{bmatrix}$$

are two particular points on that line. Any point on it is a linear combination of these two points, i.e.,

$$\mathbf{x}_\infty = a\begin{bmatrix} \mathbf{r}_1 \\ 0 \end{bmatrix} + b\begin{bmatrix} \mathbf{r}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} a\mathbf{r}_1 + b\mathbf{r}_2 \\ 0 \end{bmatrix}.$$

Now, let's compute the intersection of the above line with the absolute conic. By definition, the point $\mathbf{x}_\infty$, known as the *circular point* [18], satisfies: $\mathbf{x}_\infty^T\mathbf{x}_\infty = 0$, i.e., $(a\mathbf{r}_1 + b\mathbf{r}_2)^T(a\mathbf{r}_1 + b\mathbf{r}_2) = 0$, or $a^2 + b^2 = 0$. The solution is $b = \pm ai$, where $i^2 = -1$. That is, the two intersection points are

$$\mathbf{x}_\infty = a\begin{bmatrix} \mathbf{r}_1 \pm i\mathbf{r}_2 \\ 0 \end{bmatrix}.$$

The significance of this pair of complex conjugate points lies in the fact that they are invariant to Euclidean transformations. Their projection in the image plane is given, up to a scale factor, by

$$\widetilde{\mathbf{m}}_\infty = \mathbf{A}(\mathbf{r}_1 \pm i\mathbf{r}_2) = \mathbf{h}_1 \pm i\mathbf{h}_2.$$

Point $\widetilde{\mathbf{m}}_\infty$ is on the image of the absolute conic, described by $\mathbf{A}^{-T}\mathbf{A}^{-1}$ [15]. This gives

$$(\mathbf{h}_1 \pm i\mathbf{h}_2)^T\mathbf{A}^{-T}\mathbf{A}^{-1}(\mathbf{h}_1 \pm i\mathbf{h}_2) = 0.$$

Requiring that both real and imaginary parts be zero yields (3) and (4).

3 Solving Camera Calibration

This section provides the details how to effectively solve the camera calibration problem. We start with an analytical solution, followed by a nonlinear optimization technique based on the maximum-likelihood criterion. Finally, we take into account lens distortion, giving both analytical and nonlinear solutions.

3.1 Closed-Form Solution

Let

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} \equiv \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}. \tag{5}$$

Note that $\mathbf{B}$ is symmetric, defined by a 6D vector

$$\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T. \tag{6}$$

Let the ith column vector of $\mathbf{H}$ be $\mathbf{h}_i = [h_{i1}, h_{i2}, h_{i3}]^T$. Then, we have

$$\mathbf{h}_i^T\mathbf{B}\mathbf{h}_j = \mathbf{v}_{ij}^T\mathbf{b} \tag{7}$$

with

$$\mathbf{v}_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T.$$

Therefore, the two fundamental constraints (3) and (4), from a given homography, can be rewritten as two homogeneous equations in $\mathbf{b}$:

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix}\mathbf{b} = \mathbf{0}. \tag{8}$$

If n images of the model plane are observed, by stacking n such equations as (8), we have

$$\mathbf{V}\mathbf{b} = \mathbf{0}, \tag{9}$$

where $\mathbf{V}$ is a $2n \times 6$ matrix. If $n \geq 3$, we will have in general a unique solution $\mathbf{b}$ defined up to a scale factor. If $n = 2$, we can impose the skewless constraint $\gamma = 0$, i.e., $[0, 1, 0, 0, 0, 0]\mathbf{b} = 0$, which is added as an additional equation to (9). (If $n = 1$, we can only solve two camera intrinsic parameters, e.g., α and β, assuming u_0 and v_0 are known (e.g., at the image center) and $\gamma = 0$, and that is indeed what we did in [19] for head pose determination based on the fact that eyes and mouth are reasonably coplanar. In fact, Tsai [23] already mentions that focal length from one plane is possible, but incorrectly says that aspect ratio is not.) The solution to (9) is well-known as the eigenvector of $\mathbf{V}^T\mathbf{V}$ associated with the smallest eigenvalue (equivalently, the right singular vector of $\mathbf{V}$ associated with the smallest singular value).

Once $\mathbf{b}$ is estimated, we can compute all camera intrinsic parameters as follows. The matrix $\mathbf{B}$, as described in Section 3.1, is estimated up to a scale factor, i.e., $\mathbf{B} = \lambda\mathbf{A}^{-T}\mathbf{A}$ with λ an arbitrary scale. Without difficulty, we can uniquely extract the intrinsic parameters from matrix $\mathbf{B}$.

$$\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2)\\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11}\\
\alpha &= \sqrt{\lambda/B_{11}}\\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)}\\
\gamma &= -B_{12}\alpha^2\beta/\lambda\\
u_0 &= \gamma v_0/\alpha - B_{13}\alpha^2/\lambda.
\end{aligned}$$

exact solution

Once $\mathbf{A}$ is known, the extrinsic parameters for each image is readily computed. From (2), we have

$$\mathbf{r}_1 = \lambda\mathbf{A}^{-1}\mathbf{h}_1, \quad \mathbf{r}_2 = \lambda\mathbf{A}^{-1}\mathbf{h}_2, \quad \mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2, \quad \mathbf{t} = \lambda\mathbf{A}^{-1}\mathbf{h}_3$$

with $\lambda = 1/\|\mathbf{A}^{-1}\mathbf{h}_1\| = 1/\|\mathbf{A}^{-1}\mathbf{h}_2\|$. Of course, because of noise in data, the so-computed matrix $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$ does not, in general, satisfy the properties of a rotation matrix. The best rotation matrix can then be obtained through for example singular value decomposition [10], [26].

3.2 Maximum-Likelihood Estimation

The above solution is obtained through minimizing an algebraic distance which is not physically meaningful. We can refine it through maximum-likelihood inference.

We are given n images of a model plane and there are m points on the model plane. Assume that the image points are corrupted by independent and identically distributed noise. The maximum-likelihood estimate can be obtained by minimizing the following functional:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|\mathbf{m}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)\|^2, \tag{10}$$

where $\hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ is the projection of point $\mathbf{M}_j$ in image i, according to (2). A rotation $\mathbf{R}$ is parameterized by a vector of three parameters, denoted by $\mathbf{r}$, which is parallel to the rotation axis and whose magnitude is equal to the rotation angle. $\mathbf{R}$ and $\mathbf{r}$ are related by the Rodrigues formula [5]. Minimizing (10) is a nonlinear minimization problem, which is solved with the Levenberg-Marquardt Algorithm as implemented in `Minpack` [17]. It requires an initial guess of $\mathbf{A}$, $\{\mathbf{R}_i, \mathbf{t}_i | i = 1..n\}$ which can be obtained using the technique described in the previous section.

Desktop cameras usually have visible lens distortion, especially the radial components. We have included these while minimizing (10). Refer to the technical report, [26], for more details.

3.3 Summary

The recommended calibration procedure is as follows:

1. Print a pattern and attach it to a planar surface.
2. Take a few images of the model plane under different orientations by moving either the plane or the camera.
3. Detect the feature points in the images.
4. Estimate the five intrinsic parameters and all the extrinsic parameters using the closed-form solution, as described in Section 3.1
5. Refine all parameters, including lens distortion parameters, by minimizing (10).

There is a degenerate configuration in my technique when planes are parallel to each other. Refer to the technical report, [26], for a more detailed description.

4 Experimental Results

The proposed algorithm has been tested on both computer simulated data and real data. The closed-form solution involves finding a singular value decomposition of a small $2n \times 6$ matrix, where n is the number of images. The nonlinear refinement within the Levenberg-Marquardt Algorithm takes 3 to 5 iterations to converge. Due to space limitation, we describe in this section one set of experiments with real data when the calibration pattern is at different distances from the camera. The reader is referred to [26] for more experimental results with both computer simulated and real data, and to the following Web page: http://research.microsoft.com/~zhang/Calib/ for some experimental data and the software.

The example is shown in Fig. 1. The camera to be calibrated is an off-the-shelf PULNiX CCD camera with 6 mm lens. The image resolution is 640×480. As can be seen in Fig. 1, the model plane contains 9×9 squares with nine special dots which are used to identify automatically the correspondence between reference points on the model plane and square corners in images. It was printed on a A4 paper with a 600 DPI laser printer and attached to a cardboard.

In total, 10 images of the plane were taken (six of them are shown in Fig. 1). Five of them (called Set A) were taken at close range, while the other five (called Set B) were taken at a larger distance. We applied our calibration algorithm to Set A, Set B, and also to the whole set (called Set A+B). The results are shown in Table 1. For intuitive understanding, we show the estimated angle between the image axes, ϑ, instead of the skew factor γ. We can see that the angle ϑ is very close to $90°$, as expected with almost all modern CCD cameras. The cameras parameters were estimated consistently for all three sets of images, except the distortion parameters with Set B. The reason is that the calibration pattern only occupies the central part of the image in Set B, where lens distortion is not significant and therefore cannot be estimated reliably.

5 Conclusion

In this paper, we have developed a flexible new technique to easily calibrate a camera. The technique only requires the camera to observe a planar pattern from a few different orientations. Although the minimum number of orientations is two if pixels are square, we recommend four or five different orientations for better quality. We can move either the camera or the planar

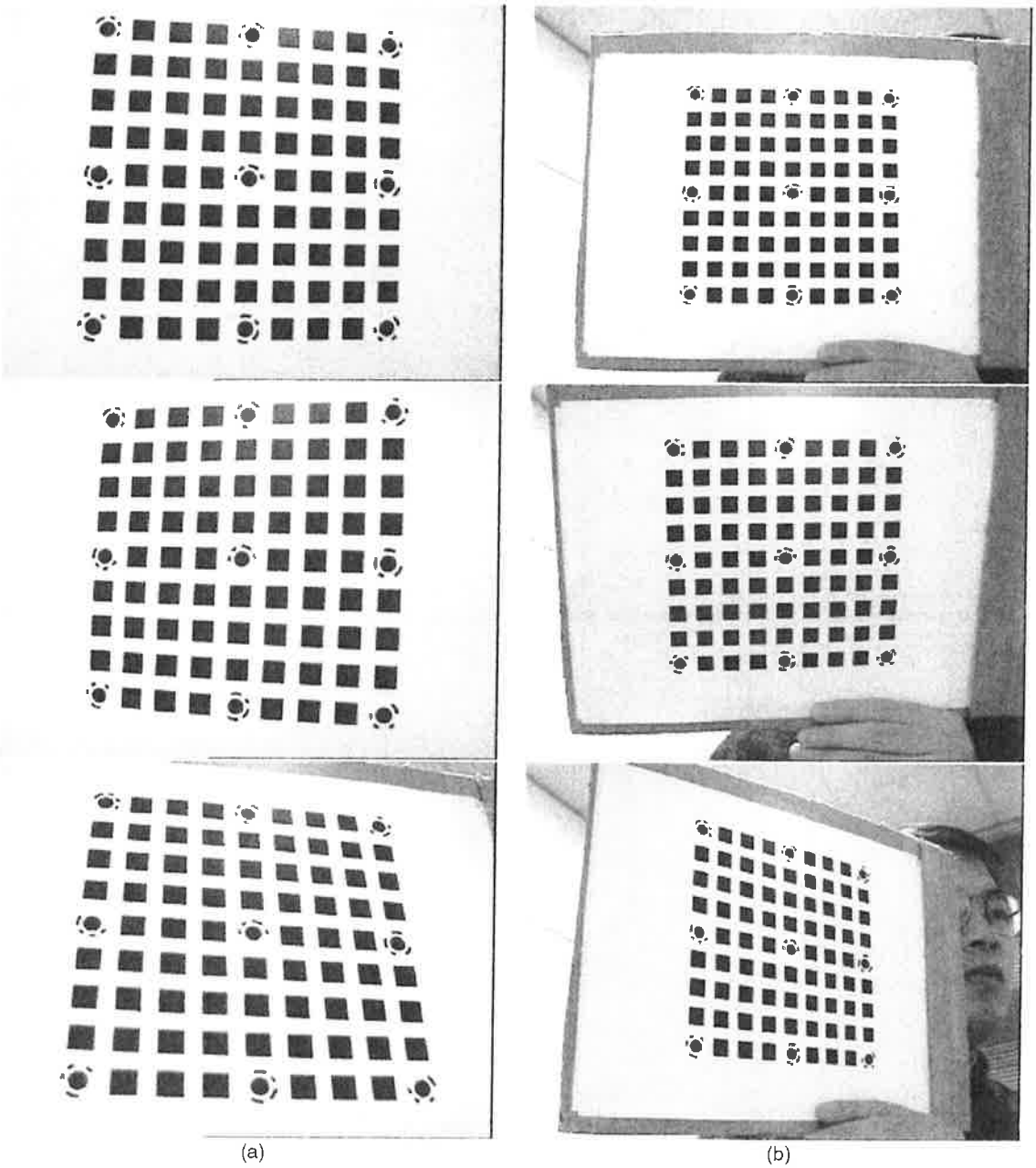

Fig. 1. Two sets of images taken at different distances to the calibration pattern. Each set contains five images. (a) Three images from the set taken at a close distance are shown. (b) Three images from the set taken at a larger distance are shown.

pattern. The motion does not need to be known, but should not be a pure translation. When the number of orientations is only two, one should avoid positioning the planar pattern parallel to the image plane. The pattern could be anything, as long as we know the metric on the plane. For example, we can print a pattern with a laser printer and attach the paper to a reasonable planar surface such as a hard book cover. We can even use a book with known size because the four corners are enough to estimate the plane homographies.

Radial lens distortion is modeled. The proposed procedure consists of a closed-form solution, followed by a nonlinear refinement based on a maximum-likelihood criterion. Both computer simulation and real data have been used to test the proposed technique and very good results have been obtained. Compared with classical techniques which use expensive equipment such as two or three orthogonal planes, the proposed technique gains considerable flexibility.

Appendix

Estimating Homography Between the Model Plane and Its Image

There are many ways to estimate the homography between the model plane and its image. Here, we present a technique based on a maximum-likelihood criterion. Let $\mathbf{M}_i$ and $\mathbf{m}_i$ be the model and image points, respectively. Ideally, they should satisfy (2). In practice, they don't because of noise in the extracted image points. Let's assume that $\mathbf{m}_i$ is corrupted by Gaussian noise with mean $\mathbf{0}$ and covariance matrix $\mathbf{\Lambda}_{\mathbf{m}_i}$. Then, the maximum-likelihood estimation of $\mathbf{H}$ is obtained by minimizing the following functional

TABLE 1
Calibration Results with the Images Shown in Fig. 1

image set	α	β	ϑ	u_0	v_0	k_1	k_2
A	834.01	839.86	89.95°	305.51	240.09	-0.2235	0.3761
B	836.17	841.08	89.92°	301.76	241.51	-0.2676	1.3121
A+B	834.64	840.32	89.94°	304.77	240.59	-0.2214	0.3643

$$\sum_i (\mathbf{m}_i - \hat{\mathbf{m}}_i)^T \mathbf{\Lambda}_{\mathbf{m}_i}^{-1} (\mathbf{m}_i - \hat{\mathbf{m}}_i),$$

where

$$\hat{\mathbf{m}}_i = \frac{1}{\bar{\mathbf{h}}_3^T \mathbf{M}_i} \begin{bmatrix} \bar{\mathbf{h}}_1^T \mathbf{M}_i \\ \bar{\mathbf{h}}_2^T \mathbf{M}_i \end{bmatrix} \quad \text{with } \bar{\mathbf{h}}_i^T \text{ the } i\text{th row of } \mathbf{H}.$$

In practice, we simply assume $\mathbf{\Lambda}_{\mathbf{m}_i} = \sigma^2 \mathbf{I}$ for all i. This is reasonable if points are extracted independently with the same procedure. In this case, the above problem becomes a nonlinear least-squares one, i.e., $\min_{\mathbf{H}} \sum_i \|\mathbf{m}_i - \hat{\mathbf{m}}_i\|^2$. The nonlinear minimization is conducted with the Levenberg-Marquardt Algorithm as implemented in `Minpack` [17]. This requires an initial guess, which can be obtained as follows:

Let $\mathbf{x} = [\bar{\mathbf{h}}_1^T, \bar{\mathbf{h}}_2^T, \bar{\mathbf{h}}_3^T]^T$. Then, (2) can be rewritten as

$$\begin{bmatrix} \widetilde{\mathbf{M}}^T & \mathbf{0}^T & -u\widetilde{\mathbf{M}}^T \\ \mathbf{0}^T & \widetilde{\mathbf{M}}^T & -v\widetilde{\mathbf{M}}^T \end{bmatrix} \mathbf{x} = \mathbf{0}.$$

When we are given n points, we have n above equations, which can be written in matrix equation as $\mathbf{L}\mathbf{x} = \mathbf{0}$, where $\mathbf{L}$ is a $2n \times 9$ matrix. As $\mathbf{x}$ is defined up to a scale factor, the solution is well-known to be the right singular vector of $\mathbf{L}$ associated with the smallest singular value (or equivalently, the eigenvector of $\mathbf{L}^T\mathbf{L}$ associated with the smallest eigenvalue). In $\mathbf{L}$, some elements are constant 1, some are in pixels, some are in world coordinates, and some are multiplication of both. This makes $\mathbf{L}$ poorly conditioned numerically. Much better results can be obtained by performing a simple data normalization prior to running the above procedure.

ACKNOWLEDGMENTS

The author would like to thank Brian Guenter for his software on corner extraction and for many discussions and to Bill Triggs for insightful comments. He would also like to thank Andrew Zisserman for bringing his CVPR '98 work [13] to the authors' attention. It uses the same constraint but in different form. He would also like to thank the members of the Vision Group at Miscrosoft Research for encouragement and discussions. Anandan and Charles Loop have checked the English of an early version. The constructive comments from the anonymous reviewers are gratefully acknowledged which have helped the author to improve the paper.

REFERENCES

[1] S. Bougnoux, "From Projective to Euclidean Space under any Practical Situation, a Criticism of Self-Calibration," *Proc. Sixth Int'l Conf. Computer Vision*, pp. 790-796, Jan. 1998.
D.C. Brown, "Close-Range Camera Calibration," *Photogrammetric Eng.*, vol. 37, no. 8, pp. 855-866, 1971.
[3] B. Caprile and V. Torre, "Using Vanishing Points for Camera Calibration," *Int'l J. Computer Vision*, vol. 4, no. 2, pp. 127-140, Mar. 1990.
[4] W. Faig, "Calibration of Close-Range Photogrammetry Systems: Mathematical Formulation," *Photogrammetric Eng. and Remote Sensing*, vol. 41, no. 12, pp. 1,479-1,486, 1975.
[5] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.
[6] O. Faugeras, T. Luong, and S. Maybank, "Camera Self-Calibration: Theory and Experiments," *Proc Second European Conf. Computer Vision*, pp. 321-334, May 1992.
[7] O. Faugeras and G. Toscani, "The Calibration Problem for Stereo," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 15-20, June 1986.
[8] S. Ganapathy, "Decomposition of Transformation Matrices for Robot Vision," *Pattern Recognition Letters*, vol. 2, pp. 401-412, Dec. 1984.
[9] D. Gennery, "Stereo-Camera Calibration," *Proc. 10th Image Understanding Workshop*, pp. 101-108, 1979.
[10] G. Golub and C. van Loan, *Matrix Computations*, Baltimore: The John Hopkins Univ. Press, third ed. 1996.
[11] R. Hartley, "Self-Calibration from Multiple Views with a Rotating Camera," *Proc. Third European Conf. Computer Vision*, pp. 471-478, May 1994.
[12] R.I. Hartley, "An Algorithm for Self-Calibration from Several Views," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 908-912, June 1994.
[13] D. Liebowitz and A. Zisserman, "Metric Rectification for Perspective Images of Planes," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 482-488, June 1998.
[14] Q.-T. Luong, "Matrice Fondamentale et Calibration Visuelle sur l'Environnement-Vers une plus Grande Autonomie des Systèmes Robotiques," PhD thesis, Université de Paris-Sud, Centre d'Orsay, Dec. 1992.
[15] Q.-T. Luong and O. Faugeras, "Self-Calibration of a Moving Camera from Point Correspondences and Fundamental Matrices," *Int'l J. Computer Vision*, vol. 22, no. 3, pp. 261-289, 1997.
[16] S.J. Maybank and O.D. Faugeras, "A Theory of Self-Calibration of a Moving Camera," *Int'l J. Computer Vision*, vol. 8. no. 2, pp. 123-152, Aug. 1992.
[17] J. More, "The Levenberg-Marquardt Algorithm, Implementation, and Theory," *Numerical Analysis*, G.A. Watson, ed., Springer-Verlag, 1977.
[18] J. Semple and G. Kneebone, *Algebraic Projective Geometry*. Oxford: Clarendon Press, 1952.
[19] I. Shimizu, Z. Zhang, S. Akamatsu, and K. Deguchi, "Head Pose Determination from One Image Using a Generic Model," *Proc. IEEE Third Int'l Conf. Automatic Face and Gesture Recognition*, pp. 100-105, Apr. 1998.
[20] G. Stein, "Accurate Internal Camera Calibration Using Rotation, with Analysis of Sources of Error," *Proc. Fifth Int'l Conf. Computer Vision*, pp. 230-236, June 1995.
[21] P. Sturm and S. Maybank, "On Plane-Based Camera Calibration: A General Algorithm, Singularities, Applications," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 432-437, June 1999.
[22] B. Triggs, "Autocalibration from Planar Scenes," *Proc. Fifth European Conf. Computer Vision*, pp. 89-105, June 1998.
[23] R.Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE J. Robotics and Automation*, vol. 3, no. 4, pp. 323-344, Aug. 1987.
[24] G. Wei and S. Ma, "A Complete Two-Plane Camera Calibration Method and Experimental Comparisons," *Proc. Fourth Int'l Conf. Computer Vision*, pp. 439-446, May 1993.
[25] J. Weng, P. Cohen, and M. Herniou, "Camera Calibration with Distortion Models and Accuracy Evaluation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965-980, Oct. 1992.
[26] Z. Zhang, "A Flexible New Technique for Camera Calibration," Technical Report MSR-TR-98-71, Microsoft Research, Dec. 1998. Available together with the software at http://research.microsoft.com/~zhang/Calib/.

Int J Comput Vis (2009) 81: 155–166
DOI 10.1007/s11263-008-0152-6

EP*n*P: An Accurate $O(n)$ Solution to the P*n*P Problem

Vincent Lepetit · Francesc Moreno-Noguer · Pascal Fua

Received: 15 April 2008 / Accepted: 25 June 2008 / Published online: 19 July 2008

Abstract We propose a non-iterative solution to the P*n*P problem—the estimation of the pose of a calibrated camera from n 3D-to-2D point correspondences—whose computational complexity grows linearly with n. This is in contrast to state-of-the-art methods that are $O(n^5)$ or even $O(n^8)$, without being more accurate. Our method is applicable for all $n \geq 4$ and handles properly both planar and non-planar configurations. Our central idea is to express the n 3D points as a weighted sum of four virtual control points. The problem then reduces to estimating the coordinates of these control points in the camera referential, which can be done in $O(n)$ time by expressing these coordinates as weighted sum of the eigenvectors of a 12×12 matrix and solving a small *constant* number of quadratic equations to pick the right weights. Furthermore, if maximal precision is required, the output of the closed-form solution can be used to initialize a Gauss-Newton scheme, which improves accuracy with negligible amount of additional time. The advantages of our method are demonstrated by thorough testing on both synthetic and real-data.[1]

Keywords Pose estimation · Perspective-n-Point · Absolute orientation

[1]The Matlab and C++ implementations of the algorithm presented in this paper are available online at http://cvlab.epfl.ch/software/EPnP/.

V. Lepetit · F. Moreno-Noguer (✉) · P. Fua
Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
e-mail: fmorenoguer@gmail.com

1 Introduction

The aim of the Perspective-n-Point problem—P*n*P in short—is to determine the position and orientation of a camera given its intrinsic parameters and a set of n correspondences between 3D points and their 2D projections. It has many applications in Computer Vision, Robotics, Augmented Reality and has received much attention in both the Photogrammetry (McGlove et al. 2004) and Computer Vision (Hartley and Zisserman 2000) communities. In particular, applications such as feature point-based camera tracking (Skrypnyk and Lowe 2004; Lepetit and Fua 2006) require dealing with hundreds of noisy feature points in real-time, which requires computationally efficient methods.

In this paper, we introduce a non-iterative solution with better accuracy and much lower computational complexity than non-iterative state-of-the-art methods, and much faster than iterative ones with little loss of accuracy. Our approach is $O(n)$ for $n \geq 4$ whereas all other methods we know of are either specialized for small fixed values of n, very sensitive to noise, or much slower. The specialized methods include those designed to solve the P3P problem (Gao et al. 2003; Quan and Lan 1999). Among those that handle arbitrary values of n (Fischler and Bolles 1981; Dhome et al. 1989; Horaud et al. 1989; Haralick et al. 1991; Quan and Lan 1999; Triggs 1999; Fiore 2001; Ansar and Daniilidis 2003; Gao et al. 2003), the lowest-complexity one (Fiore 2001) is $O(n^2)$ but has been shown to be unstable for noisy 2D locations (Ansar and Daniilidis 2003). This is currently addressed by algorithms that are $O(n^5)$ (Quan and Lan 1999) or even $O(n^8)$ (Ansar and Daniilidis 2003) for better accuracy whereas our $O(n)$ approach achieves even better accuracy and reduced sensitivity to noise, as depicted by Fig. 1 in the $n = 6$ case and demonstrated for larger values of n in the result section.

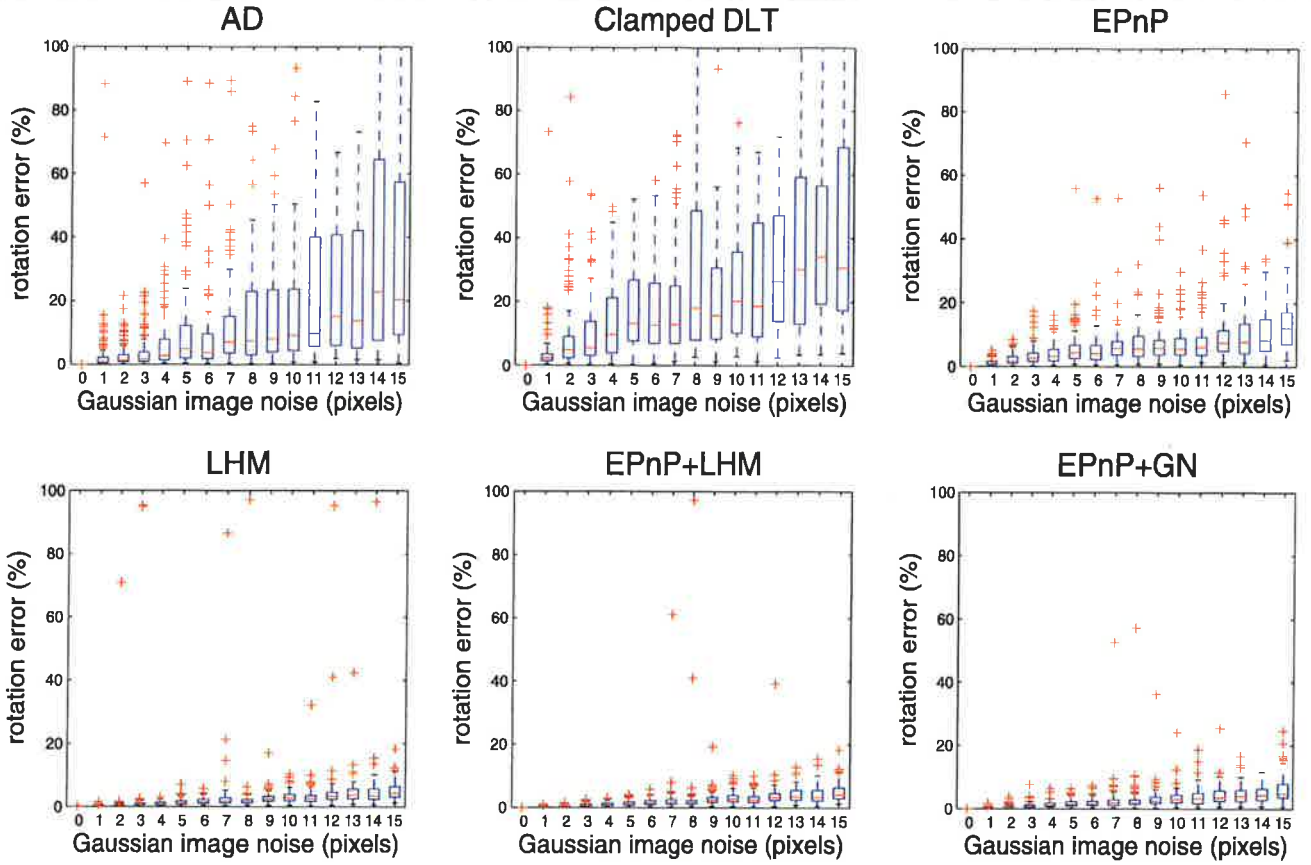

Fig. 1 Comparing the accuracy of our method against state-of-the-art ones. We use the boxplot representation: The boxes denote the first and third quartiles of the errors, the lines extending from each end of the box depict the statistical extent of the data, and the crosses indicate observations that fall out of it. *Top row*: Accuracy of non-iterative methods as a function of noise when using $n = 6$ 3D-to-2D correspondences: AD is the method of Ansar and Daniilidis (2003); Clamped DLT is the DLT algorithm after clamping the internal parameters with their known values; and EPnP is our method. *Bottom row*: Accuracy of iterative methods using $n = 6$: LHM is Lu's et al. method (Lu et al. 2000) initialized with a weak perspective assumption; EPnP + LHM is Lu's et al. algorithm initialized with the output of our algorithm; EPnP + GN, our method followed by a Gauss-Newton optimization

A natural alternative to non-iterative approaches are iterative ones (Lowe 1991; DeMenthon and Davis 1995; Horaud et al. 1997; Kumar and Hanson 1994; Lu et al. 2000) that rely on minimizing an appropriate criterion. They can deal with arbitrary numbers of correspondences and achieve excellent precision when they converge properly. In particular, Lu et al. (2000) introduced a very accurate algorithm, which is fast in comparison with other iterative ones but slow compared to non-iterative methods. As shown in Figs. 1 and 2, our method achieves an accuracy that is almost as good, and is much faster and without requiring an initial estimate. This is significant because iterative methods are prone to failure if poorly initialized. For instance, Lu's et al. approach relies on an initial estimation of the camera pose based on a weak-perspective assumption, which can lead to instabilities when the assumption is not satisfied. This happens when the points of the object are projected onto a small region on the side of the image and our solution performs more robustly under these circumstances. Furthermore, if maximal precision is required our output can be used to initialize Lu's et al., yielding both higher stability and faster convergence. Similarly, we can run a Gauss-Newton scheme that improves our closed-form solution to the point where it is as accurate as the one produced by Lu's et al. method when it is initialized by our method. Remarkably, this can be done with only very little extra computation, which means that even with this extra step, our method remains much faster. In fact, the optimization is performed in constant time, and hence, the overall solution still remains $O(n)$.

Our central idea is to write the coordinates of the n 3D points as a weighted sum of four virtual control points. This reduces the problem to estimating the coordinates of the control points in the camera referential, which can be done in $O(n)$ time by expressing these coordinates as weighted sum of the eigenvectors of a 12×12 matrix and solving a small *constant* number of quadratic equations to pick the right weights. Our approach also extends to planar config-

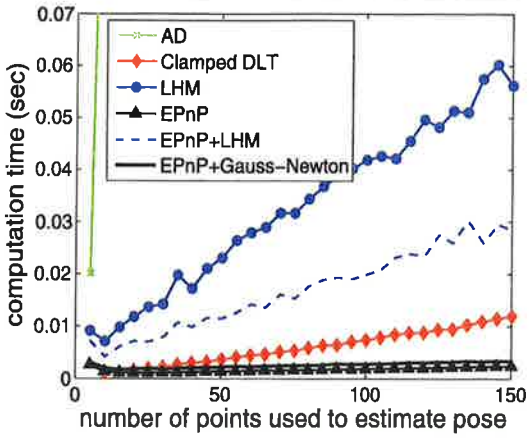

Fig. 2 Comparing computation times of our method against the state-of-the-art ones introduced in Fig. 1. The computation times of a MATLAB implementation on a standard PC, are plotted as a function of the number of correspondences. Our method is both more accurate—see Fig. 1—and faster than the other non-iterative ones, especially for large amounts of noise, and is almost as accurate as the iterative LHM. Furthermore, if maximal precision is required, the output of our algorithm can be used to initialize a Gauss-Newton optimization procedure which requires a negligible amount of additional time

urations, which cause problems for some methods as discussed in Oberkampf et al. (1996), Schweighofer and Pinz (2006), by using three control points instead of four.

In the remainder of the paper, we first discuss related work focusing on accuracy and computational complexity. We then introduce our new formulation and derive our system of linear and quadratic equations. Finally, we compare our method against the state-of-the-art ones using synthetic data and demonstrate it using real data. This paper is an expanded version of that in Moreno-Noguer et al. (2007), where a final Gauss-Newton optimization is added to the original algorithm. In Sect. 4 we show that optimizing over a reduced number of parameters, the accuracy of the closed-solution proposed in Moreno-Noguer et al. (2007) is considerably improved with almost no additional computational cost.

2 Related Work

There is an immense body of literature on pose estimation from point correspondences and, here, we focus on non-iterative approaches since our method falls in this category. In addition, we will also introduce the Lu et al. (2000) iterative method, which yields very good results and against which we compare our own approach.

Most of the non-iterative approaches, if not all of them, proceed by first estimating the points 3D positions in the camera coordinate system by solving for the points depths. It is then easy to retrieve the camera position and orientation as the Euclidean motion that aligns these positions on the given coordinates in the world coordinate system (Horn et al. 1988; Arun et al. 1987; Umeyama 1991).

The P3P case has been extensively studied in the literature, and many closed form solutions have been proposed such as Dhome et al. (1989), Fischler and Bolles (1981), Gao et al. (2003), Haralick et al. (1991), Quan and Lan (1999). It typically involves solving for the roots of an eight-degree polynomial with only even terms, yielding up to four solutions in general, so that a fourth point is needed for disambiguation. Fisher and Bolles (1981) reduced the P4P problem to the P3P one by taking subsets of three points and checking consistency. Similarly, Horaud et al. (1989) reduced the P4P to a 3-line problem. For the 4 and 5 points problem, Triggs (1999) derived a system of quadratic polynomials, which solves using multiresultant theory. However, as pointed out in Ansar and Daniilidis (2003), this does not perform well for larger number of points.

Even if four correspondences are sufficient in general to estimate the pose, it is nonetheless desirable to consider larger point sets to introduce redundancy and reduce the sensitivity to noise. To do so, Quan and Lan (1999) consider triplets of points and for each one derive four-degree polynomials in the unknown point depths. The coefficients of these polynomials are then arranged in a $\frac{(n-1)(n-2)}{2} \times 5$ matrix and singular value decomposition (SVD) is used to estimate the unknown depths. This method is repeated for all of the n points and therefore involves $O(n^5)$ operations.[2] It should be noted that, even if it is not done in Quan and Lan (1999), this complexity could be reduced to $O(n^3)$ by applying the same trick as we do when performing the SVD, but even then, it would remain slower than our method. Ansar and Daniilidis (2003) derive a set of quadratic equations arranged in a $\frac{n(n-1)}{2} \times (\frac{n(n+1)}{2} + 1)$ linear system, which, as formulated in the paper, requires $O(n^8)$ operations to be solved. They show their approach performs better than Quan and Lan (1999).

The complexity of the previous two approaches stems from the fact that quadratic terms are introduced from the inter-point distances constraints. The linearization of these equations produces additional parameters, which increase the complexity of the system. Fiore's method (Fiore 2001) avoids the need for these constraints: He initially forms a set of linear equations from which the world to camera rotation and translation parameters are eliminated, allowing the direct recovery of the point depths without considering the inter-point distances. This procedure allows the estimation of the camera pose in $O(n^2)$ operations, which makes real-time performance possible for large n. Unfortunately, ignoring nonlinear constraints produces poor results in the presence of noise (Ansar and Daniilidis 2003).

[2] Following Golub and Van Loan (1996), we consider that the SVD for a $m \times n$ matrix can be computed by a $O(4m^2n + 8mn^2 + 9n^3)$ algorithm.

By contrast, our method is able to consider nonlinear constraints but requires $O(n)$ operations only. Furthermore, in our synthetic experiments, it yields results that are more accurate than those of Ansar and Daniilidis (2003).

It is also worth mentioning that for large values of n one could use the Direct Linear Transformation (DLT) algorithm (Abdel-Aziz and Karara 1971; Hartley and Zisserman 2000). However, it ignores the intrinsic camera parameters we assume to be known, and therefore generally leads to less stable pose estimate. A way to exploit our knowledge of the intrinsic parameters is to clamp the retrieved values to the known ones, but the accuracy still remains low.

Finally, among iterative methods, Lu's et al. (2000) is one of the fastest and most accurate. It minimizes an error expressed in 3D space, unlike many earlier methods that attempt to minimize reprojection residuals. The main difficulty is to impose the orthonormality of the rotation matrix. It is done by optimizing alternatively on the translation vector and the rotation matrix. In practice, the algorithm tends to converge fast but can get stuck in an inappropriate local minimum if incorrectly initialized. Our experiments show our closed-form solution is slightly less accurate than Lu's et al. when it find the correct minimum, but also that it is faster and more stable. Accuracies become similar when after the closed-form solution we apply a Gauss-Newton optimization, with almost negligible computational cost.

3 Our Approach to the P*n*P Problem

Let us assume we are given a set of n *reference points* whose 3D coordinates are known in the world coordinate system and whose 2D image projections are also known. As most of the proposed solutions to the P*n*P Problem, we aim at retrieving their coordinates in the camera coordinate system. It is then easy and standard to retrieve the orientation and translation as the Euclidean motion that aligns both sets of coordinates (Horn et al. 1988; Arun et al. 1987; Umeyama 1991).

Most existing approaches attempt to solve for the depths of the reference points in the camera coordinate system. By contrast, we express their coordinates as a weighted sum of virtual *control points*. We need 4 non-coplanar such control points for general configurations, and only 3 for planar configurations. Given this formulation, the coordinates of the control points in the camera coordinate system become the unknown of our problem. For large n's, this is a much smaller number of unknowns that the n depth values that traditional approaches have to deal with and is key to our efficient implementation.

The solution of our problem can be expressed as a vector that lies in the kernel of a matrix of size $2n \times 12$ or $2n \times 9$. We denote this matrix as $\mathbf{M}$ and can be easily computed from the 3D world coordinates of the reference points and their 2D image projections. More precisely, it is a weighted sum of the null eigenvectors of $\mathbf{M}$. Given that the correct linear combination is the one that yields 3D camera coordinates for the control points that preserve their distances, we can find the appropriate weights by solving small systems of quadratic equations, which can be done at a negligible computational cost. In fact, for n sufficiently large—about 15 in our implementation—the most expensive part of this whole computation is that of the matrix $\mathbf{M}^\top\mathbf{M}$, which grows linearly with n.

In the remainder of this section, we first discuss our parameterization in terms of control points in the generic non-planar case. We then derive the matrix $\mathbf{M}$ in whose kernel the solution must lie and introduce the quadratic constraints required to find the proper combination of eigenvectors. Finally, we show that this approach also applies to the planar case.

3.1 Parameterization in the General Case

Let the reference points, that is, the n points whose 3D coordinates are known in the world coordinate system, be

$$\mathbf{p}_i, \quad i = 1, \ldots, n.$$

Similarly, let the 4 control points we use to express their world coordinates be

$$\mathbf{c}_j, \quad j = 1, \ldots, 4.$$

When necessary, we will specify that the point coordinates are expressed in the world coordinate system by using the ^{w} superscript, and in the camera coordinate system by using the ^{c} superscript. We express each reference point as a weighted sum of the control points

$$\mathbf{p}_i^w = \sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j^w, \quad \text{with} \sum_{j=1}^{4} \alpha_{ij} = 1, \tag{1}$$

where the α_{ij} are homogeneous barycentric coordinates. They are uniquely defined and can easily be estimated. The same relation holds in the camera coordinate system and we can also write

$$\mathbf{p}_i^c = \sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j^c. \tag{2}$$

In theory the control points can be chosen arbitrarily. However, in practice, we have found that the stability of our method is increased by taking the centroid of the reference points as one, and to select the rest in such a way that they form a basis aligned with the principal directions

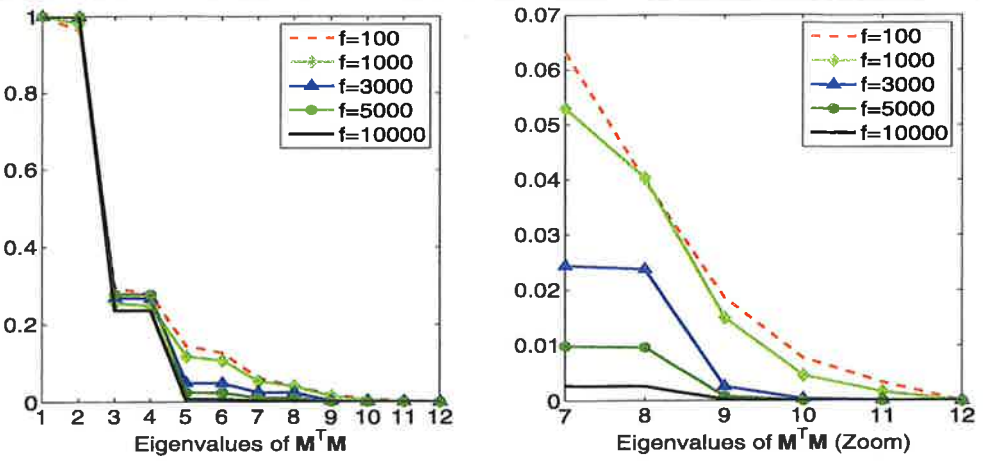

Fig. 3 *Left*: Singular values of $\mathbf{M}^\top\mathbf{M}$ for different focal lengths. Each curve averages 100 synthetic trials. *Right*: Zooming in on the smallest eigenvalues. For small focal lengths, the camera is perspective and only one eigenvalue is zero, which reflects the scale ambiguity. As the focal length increases and the camera becomes orthographic, all four smallest eigenvalues approach zero

of the data. This makes sense because it amounts to conditioning the linear system of equations that are introduced below by normalizing the point coordinates in a way that is very similar to the one recommended for the classic DLT algorithm (Hartley and Zisserman 2000).

3.2 The Solution as Weighted Sum of Eigenvectors

We now derive the matrix $\mathbf{M}$ in whose kernel the solution must lie given that the 2D projections of the reference points are known. Let $\mathbf{A}$ be the camera internal calibration matrix and $\{\mathbf{u}_i\}_{i=1,\dots,n}$ the 2D projections of the $\{\mathbf{p}_i\}_{i=1,\dots,n}$ reference points. We have

$$\forall i, \quad w_i \begin{bmatrix} \mathbf{u}_i \\ 1 \end{bmatrix} = \mathbf{A}\mathbf{p}_i^c = \mathbf{A}\sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j^c, \tag{3}$$

where the w_i are scalar projective parameters. We now expand this expression by considering the specific 3D coordinates $[x_j^c, y_j^c, z_j^c]^\top$ of each $\mathbf{c}_j^c$ control point, the 2D coordinates $[u_i, v_i]^\top$ of the $\mathbf{u}_i$ projections, and the f_u, f_v focal length coefficients and the (u_c, v_c) principal point that appear in the $\mathbf{A}$ matrix. Equation 3 then becomes

$$\forall i, \quad w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix}. \tag{4}$$

The unknown parameters of this linear system are the 12 control point coordinates $\{(x_j^c, y_j^c, z_j^c)\}_{j=1,\dots,4}$ and the n projective parameters $\{w_i\}_{i=1,\dots,n}$. The last row of (4) implies that $w_i = \sum_{j=1}^{4} \alpha_{ij} z_j^c$. Substituting this expression in the first two rows yields two linear equations for each reference point:

$$\sum_{j=1}^{4} \alpha_{ij} f_u x_j^c + \alpha_{ij}(u_c - u_i) z_j^c = 0, \tag{5}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_v y_j^c + \alpha_{ij}(v_c - v_i) z_j^c = 0. \tag{6}$$

Note that the w_i projective parameter does not appear anymore in those equations. Hence, by concatenating them for all n reference points, we generate a linear system of the form

$$\mathbf{M}\mathbf{x} = \mathbf{0}, \tag{7}$$

where $\mathbf{x} = [\mathbf{c}_1^{c\top}, \mathbf{c}_2^{c\top}, \mathbf{c}_3^{c\top}, \mathbf{c}_4^{c\top}]^\top$ is a 12-vector made of the unknowns, and $\mathbf{M}$ is a $2n \times 12$ matrix, generated by arranging the coefficients of (5) and (6) for each reference point. Unlike in the case of DLT, we do not have to normalize the 2D projections since (5) and (6) do not involve the image referential system.

The solution therefore belongs to the null space, or kernel, of $\mathbf{M}$, and can be expressed as

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i \tag{8}$$

where the set $\mathbf{v}_i$ are the columns of the right-singular vectors of $\mathbf{M}$ corresponding to the N null singular values of $\mathbf{M}$. They can be found efficiently as the null eigenvectors of matrix $\mathbf{M}^\top\mathbf{M}$, which is of small constant (12×12) size. Computing the product $\mathbf{M}^\top\mathbf{M}$ has $O(n)$ complexity, and is the most time consuming step in our method when n is sufficiently large, about 15 in our implementation.

3.3 Choosing the Right Linear Combination

Given that the solution can be expressed as a linear combination of the null eigenvectors of $\mathbf{M}^\top\mathbf{M}$, finding it amounts to computing the appropriate values for the $\{\beta_i\}_{i=1,\dots,N}$ coefficients of (8). Note that this approach applies even when the system of (7) is under-constrained, for example because

β_1 and β_2 only appear in the quadratic terms and we solve for them using a technique called "linearization" in cryptography, which was employed by Ansar and Daniilidis (2003) to estimate the point depths. It involves solving a linear system in $[\beta_{11}, \beta_{12}, \beta_{22}]^\top$ where $\beta_{11} = \beta_1^2$, $\beta_{12} = \beta_1\beta_2$, $\beta_{22} = \beta_2^2$. Since we have four control points, this produces a linear system of six equations in the β_{ab} that we write as:[3]

$$\mathbf{L}\boldsymbol{\beta} = \boldsymbol{\rho}, \tag{13}$$

where $\mathbf{L}$ is a 6×3 matrix formed with the elements of $\mathbf{v}_1$ and $\mathbf{v}_2$, $\boldsymbol{\rho}$ is a 6-vector with the squared distances $\|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2$, and $\boldsymbol{\beta} = [\beta_{11}, \beta_{12}, \beta_{22}]^\top$ is the vector of unknowns. We solve this system using the pseudoinverse of $\mathbf{L}$ and choose the signs for the β_a so that all the $\mathbf{p}_i^c$ have positive z coordinates.

This yields β_1 and β_2 values that can be further refined by using the formula of (11) to estimate a common scale β so that $\mathbf{c}_i^c = \beta(\beta_1\mathbf{v}_1^{[i]} + \beta_2\mathbf{v}_2^{[i]})$.

Case $N = 3$: As in the $N = 2$ case, we use the six distance constraints of (12). This yields again a linear system $\mathbf{L}\boldsymbol{\beta} = \boldsymbol{\rho}$, although with larger dimensionality. Now $\mathbf{L}$ is a square 6×6 matrix formed with the elements of $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$, and $\boldsymbol{\beta}$ becomes the 6D vector $[\beta_{11}, \beta_{12}, \beta_{13}, \beta_{22}, \beta_{23}, \beta_{33}]^\top$. We follow the same procedure as before, except that we now use the inverse of $\mathbf{L}$ instead of its pseudo-inverse.

Case $N = 4$: We now have four β_a unknowns and, in theory, the six distance constraints we have been using so far should still suffice. Unfortunately, the linearization procedure treats all 10 products $\beta_{ab} = \beta_a\beta_b$ as unknowns and there are not enough constraints anymore. We solve this problem using a *relinearization* technique (Kipnis and Shamir 1999) whose principle is the same as the one we use to determine the control points coordinates.

The solution for the β_{ab} is in the null space of a first homogeneous linear system made from the original constraints. The correct coefficients are found by introducing new quadratic equations and solving them again by linearization, hence the name "relinearization". These new quadratic equations are derived from the fact that we have, by commutativity of the multiplication

$$\beta_{ab}\beta_{cd} = \beta_a\beta_b\beta_c\beta_d = \beta_{a'b'}\beta_{c'd'}, \tag{14}$$

where $\{a', b', c', d'\}$ represents any permutation of the integers $\{a, b, c, d\}$.

[3] We use the indices a and b for the β's in order to differentiate from the indices i and j used for the 3D points.

3.4 The Planar Case

In the planar case, that is, when the moment matrix of the reference points has one very small eigenvalue, we need only three control points. The dimensionality of $\mathbf{M}$ is then reduced to $2n \times 9$ with 9D eigenvectors $\mathbf{v}_i$, but the above equations remain mostly valid. The main difference is that the number of quadratic constraints drops from 6 to 3. As a consequence, we need use of the relinearization technique introduced in the $N = 4$ case of the previous section for $N \geq 3$.

4 Efficient Gauss-Newton Optimization

We will show in the following section that our closed-form solutions are more accurate than those produced by other state-of-the-art non-iterative methods. Our algorithm also runs much faster than the best iterative one we know of Lu et al. (2000) but can be slightly less accurate, especially when the iterative algorithm is provided with a good initialization. In this section, we introduce a refinement procedure designed to increase the accuracy of our solution at very little extra computational cost. As can be seen in Figs. 1 and 2, computing the solution in closed form and then refining it as we suggest here yields the same accuracy as our reference method (Lu et al. 2000), but still much faster.

We refine the four values $\boldsymbol{\beta} = [\beta_1, \beta_2, \beta_3, \beta_4]^\top$ of the coefficients in (8) by choosing the values that minimize the change in distance between control points. Specifically, we use Gauss-Newton algorithm to minimize

$$\text{Error}(\boldsymbol{\beta}) = \sum_{(i,j) \text{ s.t. } i<j} \left(\|\mathbf{c}_i^c - \mathbf{c}_j^c\|^2 - \|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2 \right), \tag{15}$$

with respect $\boldsymbol{\beta}$. The distances $\|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2$ in the world coordinate system are known and the control point coordinates in camera reference are expressed as a function of the $\boldsymbol{\beta}$ coefficients as

$$\mathbf{c}_i^c = \sum_{j=1}^{4} \beta_j \mathbf{v}_j^{[i]}. \tag{16}$$

Since the optimization is performed only over the four β_i coefficients, its computational complexity is independent of the number of input 3D-to-2D correspondences. This yields fast and constant time convergence since, in practice, less than 10 iterations are required. As a result, the computational burden associated to this refinement procedure is almost negligible as can be observed in Fig. 2. In fact, the time required for the optimization may be considered as constant, and hence, the overall complexity of the closed-form solution and Gauss-Newton remains linear with the number of input 3D-to-2D correspondences.

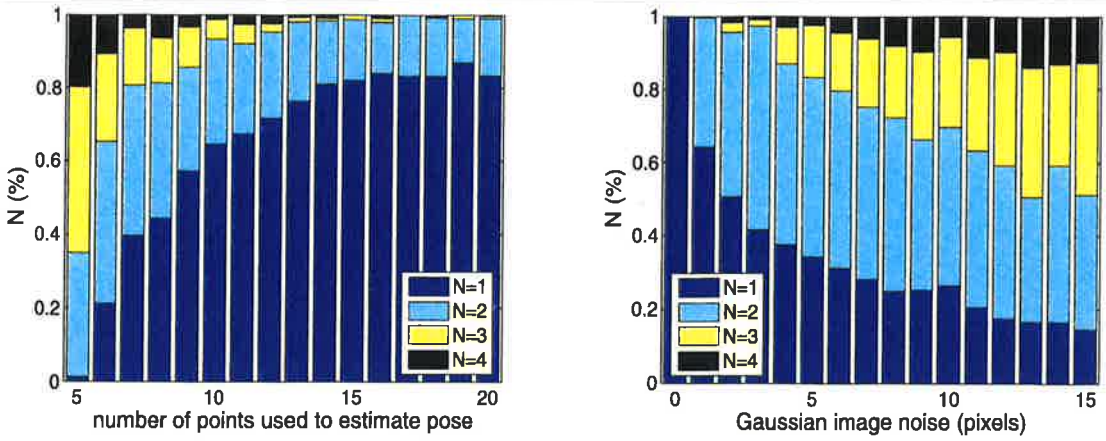

Fig. 4 Effective number N of null singular values in $\mathbf{M}^\top\mathbf{M}$. Each vertical bar represents the distributions of N for a total of 300 experiments. On the *left*, we plot the results for a fixed image noise of $\sigma = 10$ pixels and an increasing number of reference points, and the results on the *right* correspond to a fixed $n = 6$ number of reference points and increasing level of noise in the 2D projections

the number of input correspondences is (4) or (5), yielding only (8) or (10) equations, which is less than the number of unknowns.

In theory, given perfect data from at least six reference points imaged by a perspective camera, the dimension N of the null-space of $\mathbf{M}^\top\mathbf{M}$ should be exactly one because of the scale ambiguity. If the camera becomes orthographic instead of perspective, the dimension of the null space increases to four because changing the depths of the four control points would have no influence on where the reference points project. Figure 3 illustrates this behavior: For small focal lengths, $\mathbf{M}^\top\mathbf{M}$ has only one zero eigenvalue. However, as the focal length increases and the camera becomes closer to being orthographic, all four smallest eigenvalues approach zero. Furthermore, if the correspondences are noisy, the smallest of them will be tiny but not strictly equal to zero.

Therefore, we consider that the effective dimension N of the null space of $\mathbf{M}^\top\mathbf{M}$ can vary from 1 to 4 depending on the configuration of the reference points, the focal length of the camera, and the amount of noise, as shown in Fig. 4. In this section, we show that the fact that the distances between control points must be preserved can be expressed in terms of a small number of quadratic equations, which can be efficiently solved to compute $\{\beta_i\}_{i=1,\ldots,N}$ for $N = 1, 2, 3$ and 4.

In practice, instead of trying to pick a value of N among the set $\{1, 2, 3, 4\}$, which would be error-prone if several eigenvalues had similar magnitudes, we compute solutions for all four values of N and keep the one that yields the smallest reprojection error

$$\text{res} = \sum_i \text{dist}^2\left(\mathbf{A}[\mathbf{R}|\mathbf{t}]\begin{bmatrix}\mathbf{p}_i^w\\1\end{bmatrix}, \mathbf{u}_i\right), \tag{9}$$

where $\text{dist}(\tilde{\mathbf{m}}, \mathbf{n})$ is the 2D distance between point $\mathbf{m}$ expressed in homogeneous coordinates, and point $\mathbf{n}$. This improves robustness without any noticeable computational penalty because the most expensive operation is the computation of $\mathbf{M}^\top\mathbf{M}$, which is done only once, and not the solving of a few quadratic equations. The distribution of values of N estimated in this way is depicted by Fig. 4.

We now turn to the description of the quadratic constraints we introduce for $N = 1, 2, 3$ and 4.

Case $N = 1$: We simply have $\mathbf{x} = \beta\mathbf{v}$. We solve for β by writing that the distances between control points as retrieved in the camera coordinate system should be equal to the ones computed in the world coordinate system when using the given 3D coordinates.

Let $\mathbf{v}^{[i]}$ be the sub-vector of $\mathbf{v}$ that corresponds to the coordinates of the control point $\mathbf{c}_i^c$. For example, $\mathbf{v}^{[1]}$ will represent the vectors made of the three first elements of $\mathbf{v}$. Maintaining the distance between pairs of control points $(\mathbf{c}_i, \mathbf{c}_j)$ implies that

$$\|\beta\mathbf{v}^{[i]} - \beta\mathbf{v}^{[j]}\|^2 = \|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2. \tag{10}$$

Since the $\|\mathbf{c}_i^w - \mathbf{c}_j^w\|$ distances are known, we compute β in closed-form as

$$\beta = \frac{\sum_{[i,j]\in[1:4]} \|\mathbf{v}^{[i]} - \mathbf{v}^{[j]}\| \cdot \|\mathbf{c}_i^w - \mathbf{c}_j^w\|}{\sum_{[i,j]\in[1:4]} \|\mathbf{v}^{[i]} - \mathbf{v}^{[j]}\|^2}. \tag{11}$$

Case $N = 2$: We now have $\mathbf{x} = \beta_1\mathbf{v}_1 + \beta_2\mathbf{v}_2$, and our distance constraints become

$$\|(\beta_1\mathbf{v}_1^{[i]} + \beta_2\mathbf{v}_2^{[i]}) - (\beta_1\mathbf{v}_1^{[j]} + \beta_2\mathbf{v}_2^{[j]})\|^2 = \|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2. \tag{12}$$

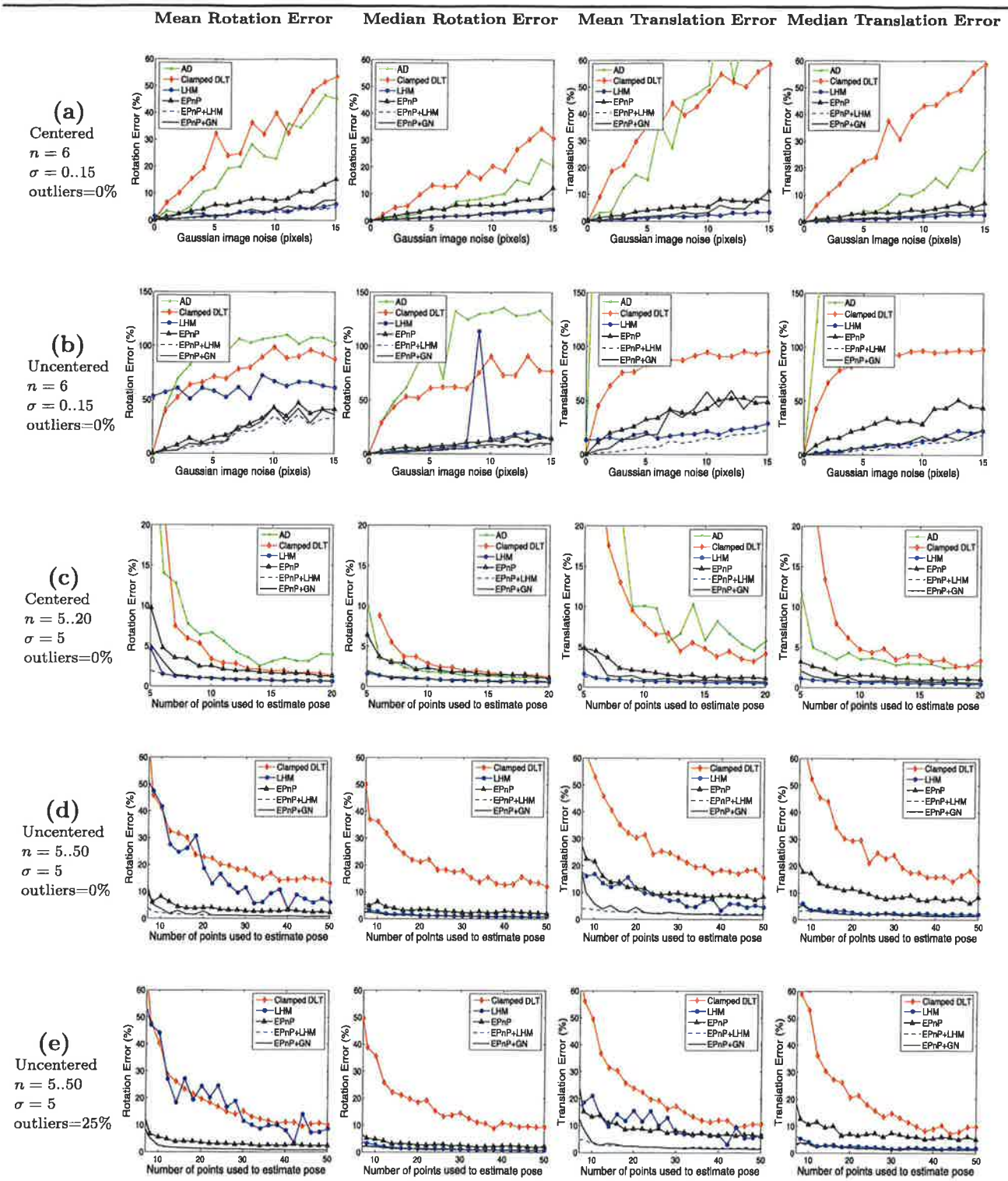

Fig. 5 (Color online) Non Planar case. Mean and median rotation and translation errors for different experiments

5 Results

We compare the accuracy and speed of our approach against that of state-of-the-art ones, both on simulated and real image data.

5.1 Synthetic Experiments

We produced synthetic 3D-to-2D correspondences in a 640×480 image acquired using a virtual calibrated camera with an effective focal length of $f_u = f_v = 800$ and a

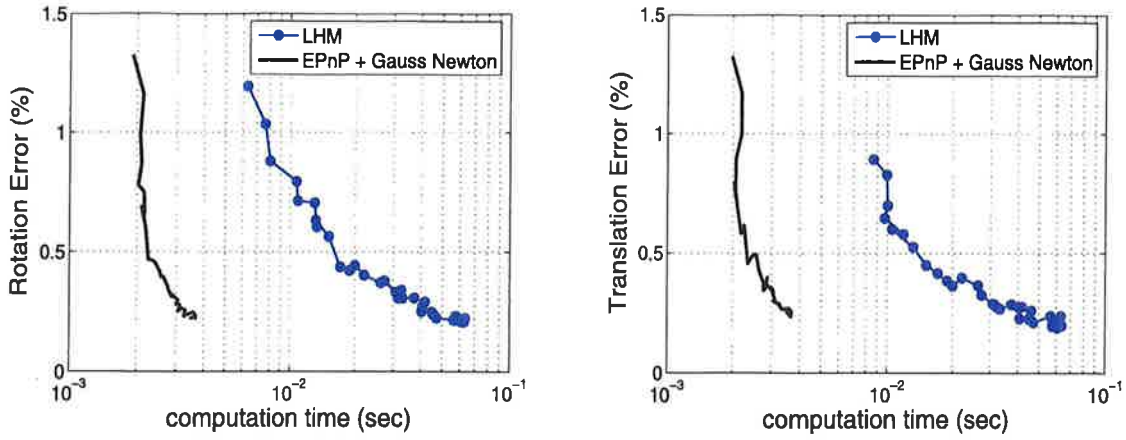

Fig. 6 Median error as a function of required amount of computation, itself a function of the number of points being used, for our approach and the LHM iterative method (Lu et al. 2000)

principal point at $(u_c, v_c) = (320, 240)$. We generated different sets for the input data. For the *centered* data, the 3D reference points were uniformly distributed into the x,y,z interval $[-2, 2] \times [-2, 2] \times [4, 8]$. For the *uncentered* data, the ranges were modified to $[1, 2] \times [1, 2] \times [4, 8]$. We also added Gaussian *noise* to the corresponding 2D point coordinates, and considered a percentage of *outliers*, for which the 2D coordinate was randomly selected within the whole image.

Given the true camera rotation $\mathbf{R}_{true}$ and translation $\mathbf{t}_{true}$, we computed the relative error of the estimated rotation $\mathbf{R}$ by $E_{rot}(\%) = \|\mathbf{q}_{true} - \mathbf{q}\|/\|\mathbf{q}\|$, where $\mathbf{q}$ and $\mathbf{q}_{true}$ are the normalized quaternions corresponding to the rotation matrices. Similarly, the relative error of the estimated translation $\mathbf{t}$ is determined by $E_{trans}(\%) = \|\mathbf{t}_{true} - \mathbf{t}\|/\|\mathbf{t}\|$.

All the plots discussed in this section were created by running 300 independent MATLAB simulations. To estimate running times, we ran the code 100 time for each example and recorded the average run time.

5.1.1 The Non-Planar Case

For the non-planar case, we compared the accuracy and running times of our algorithm, which we denote as EPnP, and EPnP + GN when it was followed by the optimization procedure described above, to: *AD*, the non-iterative method of Ansar and Daniilidis (2003); *Clamped DLT*, the DLT algorithm after clamping the internal parameters with their known values; *LHM*, the Lu's et al. (2000) iterative method initialized with a weak perspective assumption; EPnP + LHM, Lu's et al. algorithm initialized with the output of our algorithm.

On Fig. 1, we plot the rotational errors produced by the three non-iterative algorithms, and the three iterative ones as a function of noise when using $n = 6$ points. We use the boxplot representation,[4] where each column depicts the distribution of the errors for the 300 different simulations. A concise way to summarize the boxplot results is to plot both the mean and median results for all simulations: The difference between the mean and the median mainly comes from the high errors represented as red crosses in the boxplots. The greater it is, the less stable the method. This is shown in Fig. 5a, where in addition to the rotation error we also plot the translation error. The closed form solution we propose is consistently more accurate and stable than the other non-iterative ones, especially for large amounts of noise. It is only slightly less accurate than the LHM iterative algorithm. When the Gauss-Newton optimization is applied the accuracy of our method becomes then similar to that of LHM and, as shown in Fig. 5b, it even performs better when instead of using well spread data as in the previous case, we simulate data that covers only a small fraction of the image.

In Fig. 5c, we plot the errors as a function of the number of reference points, when the noise is fixed to $\sigma = 5$. Again, EPnP performs better than the other non-iterative techniques and very nearly as well as LHM. It even represents a more stable solution when dealing with the uncentered data of Fig. 5d and data which includes outliers, as in Fig. 5e. Note that in all the cases where LHM does not converge perfectly, the combination EPnP + LHM provides accurate results, which are similar to the EPnP + GN solution we propose. In the last two graphs, we did not compare the performance of AD, because this algorithm does not normalize the 2D coordinates, and hence, cannot deal well with uncentered data.

[4]The boxplot representation consists of a box denoting the first $Q1$ and third $Q3$ quartiles, a horizontal line indicating the median, and a dashed vertical line representing the data extent taken to be $Q3 + 1.5(Q3 - Q1)$. The red crosses denote points lying outside of this range.

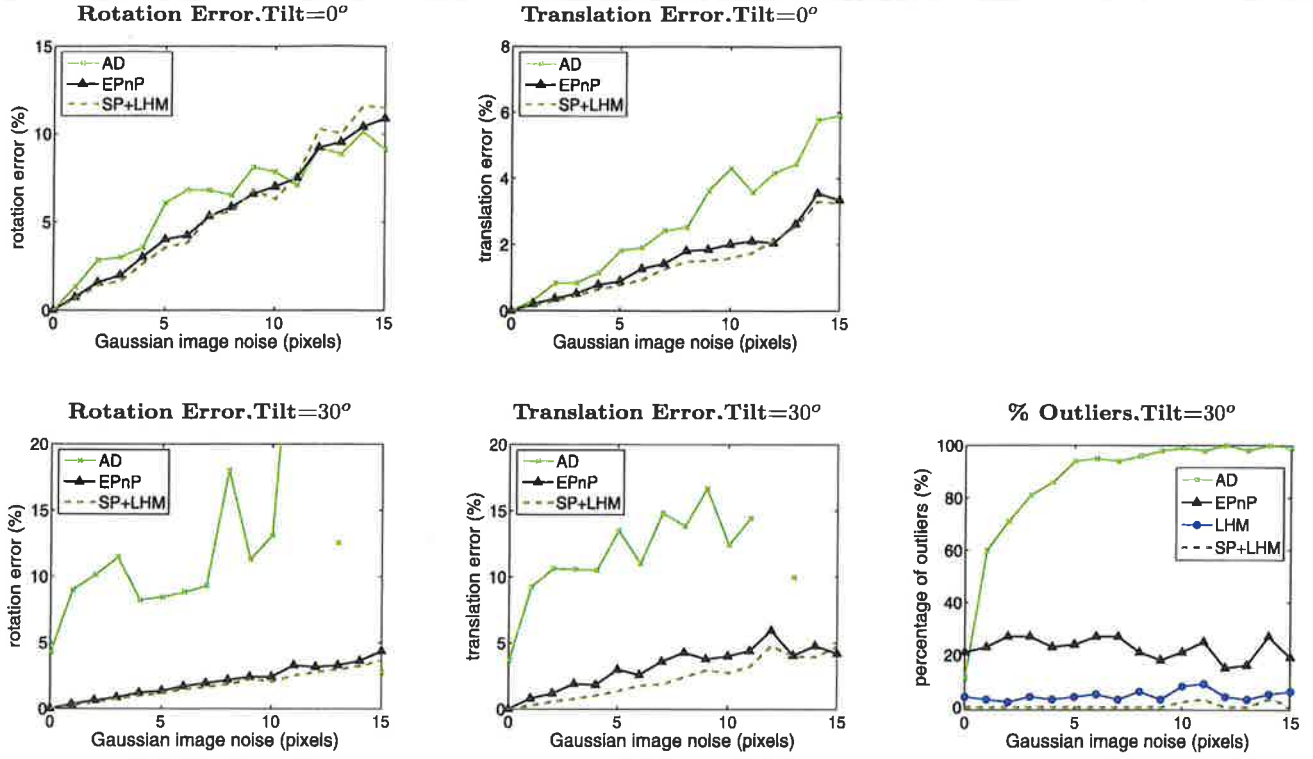

Fig. 7 Planar case. Errors as a function of the image noise when using $n = 6$ reference points. For Tilt $= 0°$, there are no pose ambiguities and the results represent the mean over all the experiments. For Tilt $= 30°$, the errors are only averaged among the solutions not affected by the pose ambiguity. The *right-most figure* represents the number of solutions considered as outliers, which are defined as those for which the average error in the estimated 3D position of the reference points is larger than a threshold

As shown in Fig. 2, the computational cost of our method grows linearly with the number of correspondences and remains much lower than all the others. It even compares favorably to clamped DLT, which is known to be fast. As shown in Fig. 6, EPnP + GN requires about a twentieth of the time required by LHM to achieve similar accuracy levels. Although the difference becomes evident for a large number of reference points, it is significant even for small numbers. For instance, for $n = 6$ points, our algorithm is about 10 times faster than LHM, and about 200 times faster than AD.

5.1.2 The Planar Case

Schweighofer and Pinz (2006) prove that when the reference points lie on a plane, camera pose suffers from an ambiguity that results in significant instability. They propose a new algorithm for resolving it and refine the solution using Lu's et al. (2000) method. Hereafter, we will refer to this combined algorithm as SP + LHM, which we will compare against EPnP, AD, and LHM. We omit Clamped DLT because it is not applicable in the planar case. We omit as well the EPnP + GN, because for the planar case the closed-form solution for the non-ambiguous cases was already very accurate, and the Gauss-Newton optimization could not help to resolve the ambiguity in the rest of cases.

Figure 7 depicts the errors as a function of the image noise, when $n = 10$ and for reference points lying on a plane with tilt of either 0 or 30 degrees. To obtain a fair comparison we present the results as was done in Schweighofer and Pinz (2006) and the errors are only averaged among those solutions not affected by the pose ambiguity. We also report the percentage of solutions which are considered as outliers. When the points are lying on a frontoparallel plane, there is no pose ambiguity and all the methods have a similar accuracy, with no outliers for all the methods, as shown in the first row of Fig. 7. The pose ambiguity problem appears only for inclined planes, as shown by the bottom-row graphs of Fig. 7. Note that for AD, the number of outliers is really large, and even the errors for the inliers are considerable. EPnP and LHM produce a much reduced number of outliers and similar results accuracy for the inliers. As before, the SP + LHM method computes the correct pose for almost all the cases. Note that we did not plot the errors for LHM, because when considering only the correct poses, it is the same as SP + LHM.

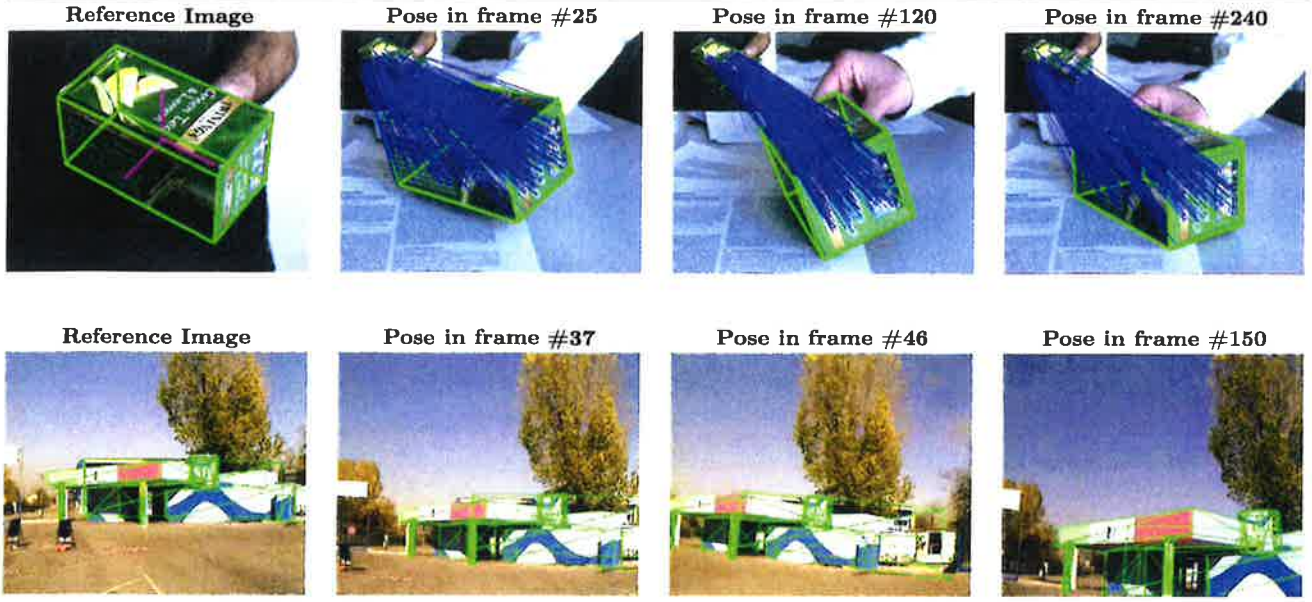

Fig. 8 Real images. *Top. Left*: Calibrated reference image. *Right*: Reprojection of the model of the box on three video frames. The camera pose has been computed using the set of correspondences depicted by the thin blue lines. *Bottom. Left*: Calibrated reference image. *Right*: Reprojection of the building model on three video frames. The registration remains accurate even when the target object is partially occluded

As in the non-planar case, the EPnP solution proposed here is much faster than the others. For example for $n = 10$ and a tilt of 30°, our solution is about 200 times faster than AD, 30 times faster than LHM, even though the MATLAB code for the latter is not optimized.

5.2 Real Images

We tested our algorithm on noisy correspondences, that may include erroneous ones, obtained on real images with our implementation of the keypoint recognition method of (Lepetit and Fua 2006). Some frames of two video sequences are shown in Fig. 8. For each case, we trained the method on a calibrated reference image of the object to be detected, for which the 3D model was known. These reference images are depicted in Fig. 8-left. At run time, the method generates about 200 correspondences per image. To filter out the erroneous ones, we use RANSAC on small subsets made of 7 correspondences from which we estimate the pose using our PnP method. This is effective because, even though our algorithm is designed to work with a large number of correspondences, it is also faster than other algorithms for small numbers of points, as discussed above. Furthermore, once the set of inliers has been selected, we use all of them to refine the camera pose. This gives a new set of inliers and the estimation is iterated until no additional inliers are found. Figure 8-right shows different frames of the sequences, where the 3D model has been reprojected using the retrieved pose.

6 Conclusion

We have proposed an $O(n)$ non-iterative solution to the PnP problem that is faster and more accurate than the best current techniques. It is only slightly less accurate than one the most recent iterative ones (Lu et al. 2000) but much faster and more stable. Furthermore, when the output of our algorithm is used to initialize a Gauss-Newton optimization, the precision is highly improved with a negligible amount of additional time.

Our central idea—expressing the 3D points as a weighted sum of four virtual control points and solving in terms of their coordinates—is very generic. We demonstrated it in the context of the PnP problem but it is potentially applicable to problems ranging from the estimation of the Essential matrix from a large number of points for Structure-from-Motion applications (Stewènius et al. 2006) to shape recovery of deformable surfaces. The latter is particularly promising because there have been many approaches to parameterizing such surfaces using control points (Sederberg and Parry 1986; Chang and Rockwood 1994), which would fit perfectly into our framework and allow us to recover not only pose but also shape. This is what we will focus on in future research.

Acknowledgements The authors would like to thank Dr. Adnan Ansar for providing the code of his PnP algorithm. This work was supported in part by the Swiss National Science Foundation and by funds of the European Commission under the IST-project 034307 DYVINE (Dynamic Visual Networks).

References

Abdel-Aziz, Y. I., & Karara, H. M. (1971). Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proc. ASP/UI symp. close-range photogrammetry* (pp. 1–18).

Ansar, A., & Daniilidis, K. (2003). Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(5), 578–589.

Arun, K. S., Huang, T. S., & Blostein, S. D. (1987). Least-squares fitting of two 3-D points sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *9*(5), 698–700.

Chang, Y., & Rockwood, A. P. (1994). A generalized de Casteljau approach to 3D free-form deformation. In *ACM SIGGRAPH* (pp. 257–260).

DeMenthon, D., & Davis, L. S. (1995). Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, *15*, 123–141.

Dhome, M., Richetin, M., & Lapreste, J.-T. (1989). Determination of the attitude of 3d objects from a single perspective view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(12), 1265–1278.

Fiore, P. D. (2001). Efficient linear solution of exterior orientation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(2), 140–148.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications ACM*, *24*(6), 381–395.

Gao, X. S., Hou, X. R., Tang, J., & Cheng, H. F. (2003). Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(8), 930–943.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations. Johns Hopkins studies in mathematical sciences*. Baltimore: Johns Hopkins Press.

Haralick, R. M., Lee, D., Ottenburg, K., & Nolle, M. (1991). Analysis and solutions of the three point perspective pose estimation problem. In *Conference on computer vision and pattern recognition* (pp. 592–598).

Hartley, R., & Zisserman, A. (2000). *Multiple view geometry in computer vision*. Cambridge: Cambridge University Press.

Horaud, R., Conio, B., Leboulleux, O., & Lacolle, B. (1989). An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics, and Image Processing*, *47*(1), 33–44.

Horaud, R., Dornaika, F., & Lamiroy, B. (1997). Object pose: The link between weak perspective, paraperspective, and full perspective. *International Journal of Computer Vision*, *22*(2), 173–189.

Horn, B. K. P., Hilden, H. M., & Negahdaripour, S. (1988). Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, *5*(7), 1127–1135.

Kipnis, A., & Shamir, A. (1999). Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Advances in cryptology—CRYPTO'99* (Vol. 1666/1999, pp. 19–30). Berlin: Springer.

Kumar, R., & Hanson, A. R. (1994). Robust methods for estimating pose and a sensitivity analysis. *Computer Vision and Image Understanding*, *60*(3), 313–342.

Lepetit, V., & Fua, P. (2006). Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(9), 1465–1479.

Lowe, D. G. (1991). Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(5), 441–450.

Lu, C.-P., Hager, G. D., & Mjolsness, E. (2000). Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(6), 610–622.

McGlove, C., Mikhail, E., & Bethel, J. (Eds.) (2004). *Manual of photogrametry. American society for photogrammetry and remote sensing* (5th edn.).

Moreno-Noguer, F., Lepetit, V., & Fua, P. (2007). Accurate non-iterative $o(n)$ solution to the pnp problem. In *IEEE international conference on computer vision*. Rio de Janeiro, Brazil.

Oberkampf, D., DeMenthon, D., & Davis, L. S. (1996). Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, *63*, 495–511.

Quan, L., & Lan, Z. (1999). Linear N-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(7), 774–780.

Schweighofer, G., & Pinz, A. (2006). Robust pose estimation from a planar target. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(12), 2024–2030.

Sederberg, T. W., & Parry, S. R. (1986). Free-form deformation of solid geometric models. *ACM SIGGRAPH*, *20*(4).

Skrypnyk, I., & Lowe, D. G. (2004). Scene modelling, recognition and tracking with invariant image features. In *International symposium on mixed and augmented reality* (pp. 110–119). Arlington, VA.

Stewènius, H., Engels, C., & Nister, D. (2006). Recent developments on direct relative orientation. *International Society for Photogrammetry and Remote Sensing*, *60*, 284–294.

Triggs, B. (1999). Camera pose and calibration from 4 or 5 known 3D points. In *International conference on computer vision* (pp. 278–284).

Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(4).

Pinhole Camera Coordinate Transform Solver

Owen Lu
Electroimpact Inc.
owenl@electroimpact.com

***Abstract*— In order to understand the location of 4 cameras in relation to the tool-point coordinate system an algorithm was implemented using the method proposed by Zhengyou Zhang in his paper "A Flexible New Technique for Camera Calibration". It uses at least 3 images of a checkerboard pattern to locate the camera and define the intrinsic/extrinsic parameters of a pinhole camera model. A set of transforms then can be combined with tool-point transforms to get the relative transform from the tool-point to the camera. Refinements for lens distortion are assumed to be taken care of in the Keyence system so no non-linear solvers are used. This allows the images to appear as if they were taken from a pinhole camera.**

I. Introduction

The main goal of this calibration routine is not to find the intrinsic parameters of the camera, but instead to find its location in relation to the tool-point coordinates. This allows us to know the position of the camera as we move it through space, and project the images it takes onto a 3D surface model virtually.

II. Conventions and conversion

Dependent on whether the data is stored in rows or columns the transform has a slightly different form. We assume that a vector is a column vector, and thus, the rotation matrix appears to the left of the vector when applying the transform, shown in Eq, 1.

$$x' = \mathbf{R}x + t \tag{1}$$

$$x = [x_1 \quad x_2 \quad x_3]^T \tag{2}$$

$$t = [t_1 \quad t_2 \quad t_3]^T \tag{3}$$

The reason that this convention is being used is due to explicit solutions given to solve camera intrinsics and extrinsic in the paper using the same convention.

Augmentation of coordinate systems is also common, which allows the transform in Eq. 1 to be applied as one matrix multiplication.

Let $\bar{x}$ be the augmented vector. It follows that the transform can be written as a matrix multiplication as shown in Eq. (5).

$$\bar{x} = [x_1 \quad x_2 \quad x_3 \quad 1]^T \tag{4}$$

$$x' = [\mathbf{R} \quad t]\bar{x} \tag{5}$$

A key point is to notice that although $\bar{x}$ is augmented, after multiplication of the transform matrix, x' is not augmented. Another convention instead of using $[\mathbf{R} \quad t]$ as the transform matrix is to expand it even further from a 3×4 matrix into a 4×4 matrix to keep the augmentation. This allows successive transforms to be applied very easily. An example of such a matrix is shown in Eq. 6.

$$\mathbf{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Therefore, it follows that the augmented transformed vector $\bar{x}'$ can be written as the product of $\mathbf{T}$ and x.

$$\bar{x}' = \mathbf{T}\bar{x} \tag{7}$$

It is also common for the vectors to be written as row vectors instead of column vectors. Given that we have computed $\mathbf{T}$ already, to convert to row vectors we simply need the transpose of $\mathbf{T}$.

$$\bar{x}'^T = \bar{x}^T\mathbf{T}^T \tag{8}$$

This allows us to easily convert conventions depending on the implementations in different libraries. Notably, Matrix3D transforms in the standard C# Media3D assembly uses the transform $\mathbf{T}^T$ convention, which is why the transform must be converted if it is first obtained using the column vector convention.

III. Algorithm Summary

Within the C# implementation, it is assumed that the coordinates within a checkerboard frame are known and that the corners of the checkerboard within an image can be detected with the Keyence system. That is, we assume that we have one set of points that have the locations of the corners in a coordinate system, and n sets of points that correspond to the locations of the corners within each image.

The algorithm is then broken down into 4 major steps

1. Initial homography estimation
2. Solve for camera intrinsic matrix
3. Solve for a set of camera extrinsic transforms
4. Combine camera extrinsic transforms with tool-point transforms to solve tool-point to camera transform

These steps are described in detail in the following sections.

IV. Homography Estimation

Due to the checkerboard pattern being contained within a plane, we can define a coordinate system that makes all checkerboard corners have Z coordinate equal to zero. Originally the equation that maps 3D points onto the 2D image is in Eq. 9.

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \mathbf{A}[\mathbf{r_1} \quad \mathbf{r_2} \quad \mathbf{r_3} \quad \mathbf{t}]\begin{bmatrix}x\\y\\z\\1\end{bmatrix} \tag{9}$$

However, since $Z = 0$, column $\boldsymbol{r_3}$ of the rotation matrix can be removed for the analysis. Therefore, we can relate the vector on the left with an augmented 2D vector $[X \quad Y \quad 1]^T$ on the right via homography $\mathbf{H}$.

$$\begin{bmatrix}u\\v\\1\end{bmatrix} = \mathbf{H}\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{10}$$

$$\mathbf{H} = \mathbf{A}[\mathbf{r_1} \quad \mathbf{r_2} \quad \mathbf{t}] \tag{11}$$

Writing $\mathbf{H}$ in terms of its individual entries gives Eq. 12. Note that $\mathbf{H}$ is only unique up to scale.

$$\mathbf{H} = \begin{bmatrix}H_{11} & H_{12} & H_{13}\\H_{21} & H_{22} & H_{23}\\H_{31} & H_{32} & H_{33}\end{bmatrix} \tag{12}$$

From Eq. 10 and Eq. 12, we can rewrite the equations for pixel coordinates u and v.

$$u = \frac{xH_{11} + yH_{12} + H_{13}}{xH_{31} + yH_{32} + H_{33}} \tag{13}$$

$$y = \frac{xH_{21} + yH_{22} + H_{23}}{xH_{31} + yH_{32} + H_{33}} \tag{14}$$

Let $\mathbf{h}$ be the listed elements of $\mathbf{H}$ in row major order.

$$\mathbf{h} = [H_{12}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33}]^T \tag{15}$$

Then Eq. 13 and Eq. 14 can be written as two homogeneous equations using $\mathbf{h}$.

Let

$$\boldsymbol{g_x} = [-x, -y, -1, 0, 0, 0, ux, uy, u] \tag{16}$$

$$\boldsymbol{g_y} = [0, 0, 0, -x, -y, -1, vx, vy, v] \tag{17}$$

$$\boldsymbol{g_x}\mathbf{h} = 0 \tag{18}$$

$$\boldsymbol{g_y}\mathbf{h} = 0 \tag{19}$$

Therefore, for each pair of points (u, v) and (x, y) we get 2 equations. Since each point gives two equations, we can construct the matrix $\mathbf{G}$ which concatenates all of the equations by stacking them vertically.

Matrix $\mathbf{G}$ must then be a $2n \times 9$ matrix where n is the number of coordinate pairs. This formulates a homogenous system of equations.

$$\mathbf{G}\mathbf{h} = \mathbf{0} \tag{20}$$

At this point, Singular Value Decomposition (SVD) is used to solve for the least squares solution to $\mathbf{h}$ which is returned as the right hand singular vector associated with the smallest singular value σ_9.

V. Intrinsic Estimation

The basic principle behind intrinsic estimation is that each image taken will result in 2 constraints on the intrinsic parameters of the camera. Taking a sufficient number of pictures allows the intrinsic matrix to be solved. The derivation is not shown here, but the method to construct constraint matrix $\mathbf{V}$ to solve transformed variables $\mathbf{b}$, which can be used to calculate intrinsic matrix $\mathbf{A}$, is given.

Let v_{ij} be calculated as below from estimated homography $\mathbf{H}$.

$$v_{ij} = \begin{bmatrix}H_{1i}H_{1j}\\H_{1i}H_{2j} + H_{2i}H_{1j}\\H_{2i}H_{2j}\\H_{3i}H_{1j} + H_{1i}H_{3j}\\H_{3i}H_{2j} + H_{2i}H_{3j}\\H_{3i}H_{3j}\end{bmatrix}^T \tag{21}$$

Then $\mathbf{V}$ can be formulated by stacking 2 equations for each of the m images taken. Therefore, $\mathbf{V}$ is a $2m \times 6$ matrix.

$$\begin{bmatrix}v_{12}\\v_{11} - v_{12}\end{bmatrix}\mathbf{b} = \mathbf{0} \tag{22}$$

We need at least 3 images in order to have a unique solution to $\mathbf{b}$ up to scale.

$$\mathbf{b} = [b_1, b_2, b_3, b_4, b_5, b_6]^T \tag{23}$$

The intrinsic matrix $\mathbf{A}$ is a 3×3 that maps 3D points onto the 2D image pixels.

$$\mathbf{A} = \begin{bmatrix}\alpha & \gamma & u_0\\0 & \beta & v_0\\0 & 0 & 1\end{bmatrix} \tag{24}$$

Each parameter in $\mathbf{A}$ can then be calculated from $\mathbf{b}$. From Zhang's paper w,e have the following formulae for the parameters which should be calculated in the same sequence.

$$v_0 = \frac{(b_2 b_4 - b_1 b_5)}{b_1 b_3 - b_2^2} \tag{25}$$

$$\phi = b_6 - \frac{\left(b_4^2 - v_0(v_2 b_4 - b_1 b_5)\right)}{b_1} \tag{26}$$

$$\alpha = \sqrt{\frac{\phi}{b_1}} \tag{27}$$

$$\beta = \sqrt{\frac{\phi b_1}{b_1 b_3 - b_2^2}} \tag{28}$$

$$\gamma = -\frac{b_2 \alpha^2 \beta}{\phi} \tag{29}$$

$$u_0 = \frac{\gamma u_0}{\alpha} - \frac{b_4 \alpha^2}{\phi} \tag{30}$$

At this point matrix, $\mathbf{A}$ can be solved via substitution.

VI. Extrinsic Estimation

Now that $\mathbf{A}$ is known it is a straightforward calculation to find the rotation and translation matrices defined in Eq. 9.

Let

$$\mathbf{R} = [\mathbf{r_1} \quad \mathbf{r_2} \quad \mathbf{r_3}] \tag{31}$$

$$\mathbf{H} = [\mathbf{h_1} \quad \mathbf{h_2} \quad \mathbf{h_3}] \tag{32}$$

Let $\mathbf{C} = \frac{\mathbf{A}^{-1}}{\|\mathbf{A}^{-1}\mathbf{h_1}\|} = \frac{\mathbf{A}^{-1}}{\|\mathbf{A}^{-1}\mathbf{h_2}\|}$

$$\mathbf{r_1} = \mathbf{C}\mathbf{h_1} \tag{33}$$

$$\mathbf{r_2} = \mathbf{C}\mathbf{h_2} \tag{34}$$

$$\mathbf{r_3} = \mathbf{r_1} \times \mathbf{r_2} \tag{35}$$

$$\mathbf{t} = \mathbf{C}\mathbf{h_3} \tag{36}$$

At this point, we have all the parameters to construct transformation matrix $\mathbf{T}$ in Eq. 6.

VII. Combining tool point transforms

The purpose of performing this calibration method was to find the transform from the tool-point to the camera. Up to this point, the algorithm has found the rotation and translation of the camera relative to the checkerboard coordinates. In our application, the checkerboard is located in space and thus, the transform to convert a point from the checkerboard coordinates to the tool-point and vice-versa is assumed to be known.

Thus, the process is to find the aggregate transform from tool-point to checkerboard to camera m times and average the transform.

Suppose we have two transforms defined by $\mathbf{R_1}, \mathbf{t_1}$ and $\mathbf{R_2}, \mathbf{t_2}$ applied in sequence.

Then the aggregate transform has the following properties

$$\mathbf{T_a} = [\mathbf{R_2}\mathbf{R_1} \quad \mathbf{R_2}\mathbf{t_1} + \mathbf{t_2}] \tag{37}$$

Then all we must do is compute the aggregate rotation matrix and translation as in Eq. 37 to find the aggregate transform from tool-point to the camera.

VIII. Averaging multiple transforms

The concept of averaging, in this case, means the following. Suppose $\bar{p}$ is a point that we wish to transform by $\mathbf{T_a}$. Since there are m estimates of $\mathbf{T_a}$ we need to average them.

$$\bar{p}'_i = \mathbf{T}_{a_i}\bar{p} \tag{38}$$

An easy method would simply be to add all the estimations of p_i and average them.

$$\bar{p}'_{avg} = \frac{1}{m}\sum_{i=1}^{m}\bar{p}'_i = \frac{1}{m}\left(\sum_{i=1}^{m}\overline{\mathbf{T}_{a_i}}'\right)\bar{p} \tag{39}$$

This means that the averaged transformed can be thought of as adding all the transforms element by element and averaging them.

IX. Rotation Matrix Correction

Due to estimation error, the rotation matrix component found by averaging in Eq. 39 will not, in general, satisfy the rotation matrix. We then find the matrix that satisfies both the properties of the rotation matrix and minimizes the Frobenius norm σ.

$$\sigma = \|\mathbf{R_{avg}} - \mathbf{R_c}\| \tag{40}$$

This can be done also with the SVD. Suppose we use SVD to obtain matrices $\mathrm{U}\Sigma\mathrm{V}^{\mathrm{T}}$, then the expression for the best fit rotation matrix can be calculated using $\mathbf{U}\mathbf{V}^T$.

$$\mathbf{R_{avg}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \tag{41}$$

$$\mathbf{R_c} = \mathbf{U}\mathbf{V}^T \tag{42}$$

If the rotation matrix $\mathbf{R}_c$ is improper, then use a negative left singular vector $\mathbf{u}_3$ in order to computer $\mathbf{R}_c$.

X. Camera calibration results

Using a phone camera, 4 images of a checkerboard were taken. This is shown in Figure 1.

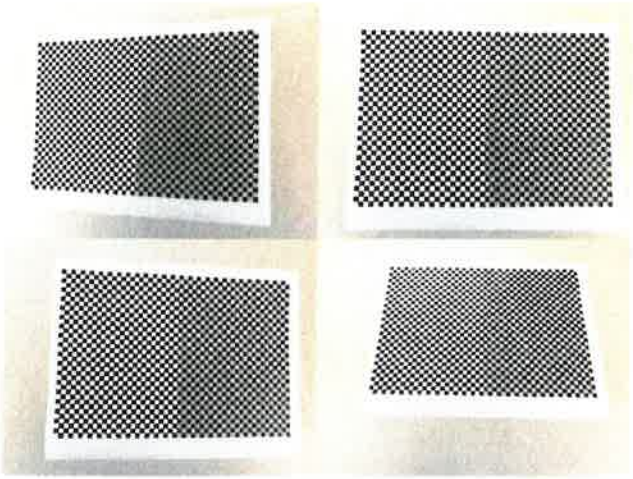

Figure 1 – All 4 images used to calibrate the camera.

Since matrices $\mathbf{A}$, $[\mathbf{R}\ \mathbf{t}]$ can be calculated, we can apply the transforms on the checkerboard corners and also compare them to the image points. We simply multiply the matrices to a $4 \times n$ matrix containing the data points. This is shown in Eq, 43.

$$\begin{bmatrix} u_1' & \dots & u_n' \\ v_1' & \dots & v_n' \\ w_1' & \dots & w_n' \end{bmatrix} = \mathbf{A}[\mathbf{R}\ \mathbf{t}] \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{43}$$

Since there is an arbitrary scale w_i' we simply divide u_i' and v_i' by the scale w_i' for each point. This gives the pixel coordinates on the image. Using this method, a reprojection was created, showing good agreement in Figure 2.

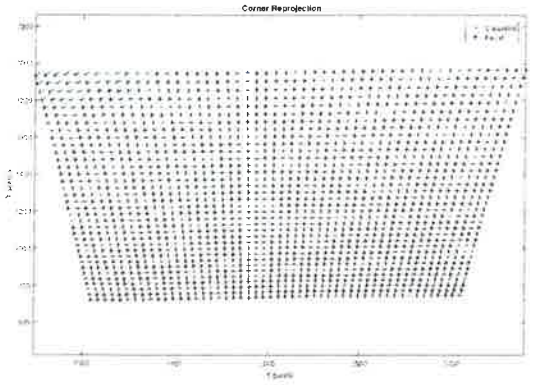

Figure 2 - Reprojection verification

XI. Conclusion

Although we are primarily concerned with estimating the matrix $[\mathbf{R}\ \mathbf{t}]$, looking at the pixel reprojection errors can give us insight on how well this model in general is performing. Thus, we considered the norm of the errors and their distribution as well as their mean. From Figure 3 we can see each pixel axis has a mean error centered around 0. In practice i,t is approximately 0.4 pixels mean error in each axis.

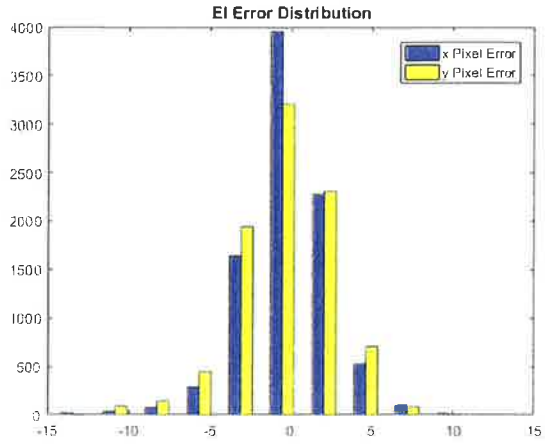

Figure 3 - Histogram of u and v pixel error for all 4 images

The Euclidian error is calculated shown in Eq. 46. It uses the standard deviations in each of the axes.

$$e_u = std(u' - u) \tag{44}$$

$$e_v = std(v' - v) \tag{45}$$

$$\delta = \sqrt{e_v^2 + e_u^2} \approx 4 \tag{46}$$

The result is 4 pixel standard deviation error showing good agreement. This shows the validity of the calibration procedure in producing accurate results and that the transforms are valid over multiple angles.

General Camera Transform Solver

Owen Lu
Electroimpact Inc.
owenl@electroimpact.com

Abstract— **Assuming an intrinsic camera matrix, the pose of the camera can be estimated by knowing n 3D coordinates and their projections onto the image plane. The previous calibration model assumed that points were within a plane in 3D in order to calculate its relative pose. This meant that for each picture taken in a different orientation, the two transforms must be composed in order to get a relative transform. As this new algorithm allows arbitrary 3D points to be used in pose calibration, a set of images of points within multiple non-coplanar checkboards can be used to create a set of points forming a volume in their native tool point coordinate system. That is, no coordinate convention is assumed, and coordinate transforms are direct. This paper details an exact method to calculating the pose of a camera using the Efficient Perspective n Point (EPnP) method proposed by Lepetit et. Al.**

I. Introduction

The previous implementation of the camera calibration algorithm implicitly assumes that the world coordinates are intrinsically defined by the calibration checkerboard pattern. Because of this, closed form solutions for the camera intrinsic parameters can easily be written after the homography and intrinsic constraint equations are solved.

In other words, the world coordinates are fixed and all assumed to exist in a plane which makes the calculation of the pose easy relative to this fixed coordinate system. However, transforms relative to other coordinates can only be calculated via composition, which compounds the error.

The problem is that within moving coordinate system C_1 the transform defined by R, t, which describes the camera pose, is assumed to be a constant. However, calibration points expressed in C_1 contained in X, an $n \times 3$ matrix, are non-coplanar as C_1 moves relative to the plane to take m unique pictures. This cluster of points cannot be used to calibrate the pose of the camera if the old algorithm is used.

II. General approach

Central to the EPnP approach is the homogenous barycentric coordinate system in three dimensions which is defined by 4 non-coplanar points dubbed "control points" by Lepetit et. Al. The basic principle is that if all 3D points can be calculated as a weighted sum of 4 non-coplanar points within world coordinates, only these 4 control points points need to be solved in the camera coordinate system to efficiently recover all n 3D points within the camera coordinate system. At this point, any algorithm to determine the rigid transform between n points in world coordinates mapped to n points in camera coordinates can be used. The result is the camera pose. In principle there are only a few key equations, and how they are solved is completely up to the person implementing the algorithm. The authors suggest some methods such as linearization and relinearization, but in this paper Gauss-Newton gradient solvers are used in order to get a general solution form regardless of varying numbers of right hand singular vectors generated by the constraint equations.

III. Algorithm Summary

The algorithm is broken down into 8 detailed steps.

1. Define control points in world coordinates for the homogeneous barycentric system
2. Calculate the $\alpha_{j\in[1,2,3,4]}$ weights based on the control points for each n data points
3. For each of the n data points, generate 2 constraints on the control points described in the camera coordinates. Concatenate all equations and form M such that $\mathrm{M}x = 0$ where x is a 12 vector containing the 4 control points and their components within the camera system.
4. Solve for the right singular vectors of M to describe the general solution space of x.
5. Use Gauss-Newton optimization method to generate the best linear combination x_1, x_2, x_3, x_4 which are the right singular vectors associated with the 4 smallest singular values of M. Let x be this best linear combination from a least squares optimization.
6. Recover the control points in camera space from x.
7. Use control points in camera space and the previously calculated weights in step 2 to find all n points in camera space.
8. Solve the rigid body rotation problem R, t which defines the pose of the camera.

These steps are described in detail in the following sections.

IV. Implementation

In general, since the model of the system is a linear model we attempt to describe as much of the model as possible as matrix operations.

First, the control points which are arbitrary are selected. Note they must be non-coplanar. We store these inside the 4×3 matrix C_w describing the control points in the world frame.

1. Define the control points

We simply pick the orthogonal basis vectors of the rectangular world system and add the origin. The choice is arbitrary, but this choice is well conditioned which results in numerical stability.

$$\mathbf{C}^w = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \\ \mathbf{c}_4 \end{bmatrix}$$

2. Calculate the weights for each point

Let $\mathbf{X}^w$ be the points in the world coordinate system where the ith in $\mathbf{X}^w$ is a 1×3 vector describing the x, y, z coordinates.

To solve the weights of the barycentric system we can augment the matrices $\mathbf{C}^{wT}$ and $\mathbf{X}^T$ by adding a row of ones.

Let:
The augmented $\mathbf{C}^{wT}$ be $\mathbf{C}_\star^{wT}$
The augmented $\mathbf{X}^{wT}$ be $\mathbf{X}_\star^{wT}$

$$\mathbf{C}_\star^{wT}\alpha = \mathbf{X}_\star^{wT}$$

Where α is a $4 \times n$ matrix where each column contains a vector of the control points weights for the ith point.

$$\alpha = \begin{bmatrix} \alpha_{1_1} & \cdots & \alpha_{1_n} \\ \alpha_{2_1} & \cdots & \alpha_{2_n} \\ \alpha_{3_1} & \cdots & \alpha_{3_n} \\ \alpha_{4_1} & \cdots & \alpha_{4_n} \end{bmatrix}$$

Since $\mathbf{C}_\star^{wT}$ is square we can simply invert it.

$$\alpha = inv(\mathbf{C}_\star^{wT})\mathbf{X}_\star^T$$

At this point, α has been calculated.

3. Generate the constraint matrix

Matrix $\mathbf{M}$ is derived from concatenating 2 constraint equations from each of the n points. We assume that for the ith point

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{A} \sum_{j=1}^{4} \alpha_{j_i} \mathbf{c}_j^T \quad (1)$$

It is assumed that matrix $\mathbf{A}$ describing the camera is skew-less.

$$\mathbf{A} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

This gives rise to two constraints, formed from the u_i, v_i pixel equations. This means that

$$\sum_{j=1}^{4} \alpha_{j_i} f_u x_j + \alpha_{j_i}\ u_0 - u_i\ z_j = 0$$
$$\sum_{j=1}^{4} \alpha_{j_i} f_v y_j + \alpha_{j_i}\ v_0 - v_i\ z_j = 0$$

Where $c_j = [x_j, y_j, z_j]$

The resultant vector is a $2n \times 12$ matrix which is solved for its right singular values. This is the same as solving for x in the following equation.

$$\mathbf{M}x = 0$$

4. Solve for the right singular values of M

In practice, finding the 4 eigenvectors of $\mathbf{M}^T\mathbf{M}$ that are associated with the smallest eigenvalues is more numerically stable than finding the null space of $\mathbf{M}$ since noise can make it such that the null space is empty for example.

The maximum number of null vectors is 4 which is the reason we find the 4 eigenvectors associated with the smallest eigenvalues. This basically means that the vectors are sent very close to the origin. Roughly satisfying our above equation.

At this point, we have four vectors x_1, x_2, x_3, x_4.

Assume that the corresponding eigenvalues have the following relationship: $\lambda_1 < \lambda_2 < \lambda_3 < \lambda_4$.

5. Gauss-Newton algorithm to linear combinations

Therefore, x_1 is the best single null vector approximation, to $\mathbf{M}x = 0$. It follows that x_2 is the second best approximation and so forth.

Thus, it makes sense to attempt to create a solution x which is a linear combination of $x_{i \in \{1,2,3,4\}}$.

We wish to solve the weights β_i that gives x such that the norms between corresponding points in world space and camera space are equal. This creates an equivalent scale between the two coordinate systems and also refines their directions.

In practice, it is helpful to use the 2-norm squared and compare the values since the expressions are simpler.

For any pair of points in camera space, the norms squared should be equal to that of the pair of points in world space.

$$\|p_a^c - p_b^c\|^2 = \|p_a^w - p_b^w\|^2$$

Since we have defined every point as a weighted sum of control points, there are only 4 points that we need to consider. Since there are only 6 ways to pair 2 points within a set of 4, we have only 6 equations to be summed for overall squared error.

We define each the 6 differences as such

Let

$$\begin{aligned}
\delta_1^w &= \mathbf{c}_1 - \mathbf{c}_2 \\
\delta_2^w &= \mathbf{c}_1 - \mathbf{c}_3 \\
\delta_3^w &= \mathbf{c}_1 - \mathbf{c}_4 \\
\delta_4^w &= \mathbf{c}_2 - \mathbf{c}_3 \\
\delta_5^w &= \mathbf{c}_2 - \mathbf{c}_4 \\
\delta_6^w &= \mathbf{c}_3 - \mathbf{c}_4
\end{aligned}$$

$$\sigma^2 = \sum_i^6 \left(\|\beta_1 \delta_{i\,1}^c + \beta_2 \delta_{i\,2}^c + \beta_3 \delta_{i\,3}^c + \beta_4 \delta_{i\,4}^c\|^2 - \|\delta_i^w\|^2 \right)^2$$

We notice that squaring the norm squared yields quartic equations, which creates negative solutions. For example, if β was the vector of weights that minimized σ^2, then $-\beta$ would also be a solution to the minimum.

In practice, this means that the rotation and translation are negative, which results in the points within the camera system having a negative z coordinate. This is impossible due to camera coordinate conventions, therefore, the points are then multiplied by (-1) and the proper transform is found. This problem is avoided if the 2-norm squared residuals are used instead of the 2-norm squared-squared residuals. However, the derivative computation is elegantly expressed using the σ^2 expression from earlier which is helpful in using Gauss-Newton methods.

The problem is then state below:

Minimize σ^2 with respect to $(\beta_1, \beta_2, \beta_3, \beta_4)$

Since Gauss-Newton requires the derivatives of the residuals to be computed for each data point i

Let

$$r_i = \|\beta_1 \delta_{i\,1}^c + \beta_2 \delta_{i\,2}^c + \beta_3 \delta_{i\,3}^c + \beta_4 \delta_{i\,4}^c\|^2 - \|\delta_i^w\|^2$$

We formulate the Jacobian, which holds the derivative of r_i with respect to $\beta_{i\in\{1,2,3,4\}}$ in the columns, evaluated at (β) in its ith row. β is the vector containing all β_i. This is basically a derivative matrix calculated for each data point. Note that superscript s denotes the iteration count. This means that we must initialize β^0.

Let

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

$$r = \begin{bmatrix} r_1 \\ \vdots \\ r_6 \end{bmatrix}$$

$$\Delta_{ij} = \delta_i \cdot \delta_j$$

$$\Delta = [\Delta_1, \Delta_2, \Delta_3, \Delta_4]$$

$$\mathbf{J}_{ij} = \frac{\partial r_i(\beta^s)}{\partial \beta_j} = 2\beta^T \Delta_j$$

$$\beta^{s+1} = \beta^s - inv\ \mathbf{J}^T\mathbf{J}\ \mathbf{J}^T r(\beta^s)$$

This results in a general method to solve for β regardless of the number of approximate singular vectors x. For example, if we are only considering using vector x_1 to solve the system. Then β_1 is initialized and $\beta_{i\in\{2,3,4\}} = 0$.

Calculating the linear combination to find x is then the linear combination with the β values

$$x = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$$

Then the transformed control points in camera space can be calculated.

6. *Recover the camera control points*

Let

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{12} \end{bmatrix}$$

Then the transformed control points can be recovered in camera space.

$$\begin{aligned}
\mathbf{c}_1^c &= [x_1, x_2, x_3] \\
\mathbf{c}_2^c &= [x_4, x_5, x_6] \\
\mathbf{c}_3^c &= [x_7, x_8, x_9] \\
\mathbf{c}_4^c &= [x_{10}, x_{11}, x_{12}]
\end{aligned}$$

$$\mathbf{C}^c = \begin{bmatrix} \mathbf{c}_1^c \\ \mathbf{c}_2^c \\ \mathbf{c}_3^c \\ \mathbf{c}_4^c \end{bmatrix}$$

7. *Calculate the coordinates in camera space*

At this point since the weights in α can be used to calculate all the points $\mathbf{X}^c$

$$\mathbf{X}^c = \alpha \mathbf{C}^c$$

Although scales should be very close at this point, we compute the scale factor to equate the norms again.

Let

$$d_i^c = \|\mathbf{X}_i^c\|$$

$$d_i^w = \|\mathbf{X}_i^w\|$$

Then we compute λ such that the squared error in norms r^2

$$r^2 = \sum_{i=1}^{n} \lambda d_i^c - d_i^w \ ^2$$

Taking the derivative of r^2 with respect to lambda

$$\frac{dr^2}{d\lambda} = \sum_{i=1}^{n} 2 \ \lambda d_i^c - d_i^w \ d_i^c$$

Setting the derivative to zero allows us to express λ in closed form by using dot products, or equivalently, inner products.

$$\lambda = \frac{d^w \cdot d^c}{d^c \cdot d^c} = \frac{d^{wT} d^c}{d^{cT} d^c}$$

At this point, the scales are the closest possible using a single scaling factor with respect to a squared residual error.

At this point, we check if the z values contained in $\mathbf{X}^c$ are negative. If they are, then they are all negative, so we multiply $\mathbf{X}^c$ by (-1) to get the correct points.

8. Solve the rigid body transformation problem

Now we have two matrices with the 3D points in camera space and world space. The last step is to solve the rigid rotation problem. Start by calculating the centroids.

$$x^c = mean \ \mathbf{X}^c$$
$$x^w = mean(\mathbf{X}^w)$$

The by abuse of notation matrix $\mathbf{X}$ subtracted by the centroid x denotes a point by point subtraction.

$$\delta\mathbf{X}^c = \mathbf{X}^c - x^c$$

$$\delta\mathbf{X}^w = \mathbf{X}^w - x^w$$

Let

$$\mathbf{S} = \delta\mathbf{X}^w diag \ 1, \ldots, 1_n \ \delta\mathbf{X}^{cT}$$

Compute the Singular Value Decomposition (SVD) of S

$$\mathbf{S} = \mathbf{U}\Sigma\mathbf{V}^T$$

Finally, the rotation and translation of the camera with respect to the reference coordinate frame is calculated.

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T$$

$$\mathbf{t} = x^c - \mathbf{R}x^w$$

In practice, the process is completed 4 times to solve 4 different transforms $\mathbf{R}, \mathbf{t}$ using different numbers of null vectors. For example, we compute the linear weights β for 1 vector, 2 vectors, 3 vectors, and finally 4 vectors. We always keep the smallest null vectors first, and add the next smallest to compute the transformed control points. Finally, we compare the reprojection error of all 4 sets in order to pick out the best transform.