IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE

An Efficient Hidden Variable Approach to Minimal-Case Camera Motion Estimation

Richard Hartley, *Fellow, IEEE,* Hongdong Li, *Member, IEEE*

Abstract—In this paper we present an efficient new approach for solving two-view minimal-case problems in camera motion estimation, most notably the so-called five-point relative orientation problem, and the six-point focal-length problem. Our approach is based on the *hidden variable technique* used in solving multivariate polynomial systems. The resulting algorithm is conceptually simple, which involves a relaxation which replaces monomials in all but one of the variables to reduce the problem to the solution of sets of linear equations, as well as solving a polynomial eigenvalue problem (polyeig). To efficiently find the polynomial eigenvalues, we make novel use of several numeric techniques, which include quotient-free Gaussian elimination, Levinson-Durbin iteration, and also a dedicated root-polishing procedure. We have tested the approach on different minimal cases and extensions, with satisfactory results obtained. Both the executables and source codes of the proposed algorithms are made freely downloadable.

Index Terms—Camera calibration, camera motion estimation, epipolar geometry, minimal solver, polynomial root finding.

1 Introduction

This paper studies the classical problem of estimating relative camera motion from two views. We are interested in minimal case problems. In particular, we focus on the so-called "minimal case orientation" from two views, for example, estimating the relative camera pose given five corresponding points in two fully-calibrated views, or given six points if the cameras are semi-calibrated where only the focal length is unknown. Since the relative pose of two views is faithfully described by an *essential matrix* E, the task is equivalent to estimating the essential matrix (EM) from five or six points. The application of minimal solvers such as the 5-point algorithm is particularly relevant in the context of RANSAC, in which speed is essential, since computations are repeated many times.

The most well-known method for solution of the minimal case 5-point problem is that of Nistér [1], which relies on reducing a set of polynomial equations using Gauss-Jordan elimination. This algorithm represents the state of the art. It is very fast, due to very careful implementation in C. One downside of this is however, as of today, no authoritative source-code implementation (such as Nistér's own code) is readily available (due to perhaps intellectual property restriction issue, as the authors are aware of).

The present paper is related to our previous work [2], [3], in which we mainly aimed at providing a very conceptually-simple, and easy-to-implement algorithms for such minimal case problems. Our previous two algorithm implementations–the five-point algorithm in [2] and the six-point algorithm in [3] are very concise, using only about 20 lines of Matlab code, yet critically relying on the Symbolic-Math toolbox (and an internal Maple engine) of Matlab, rendering the algorithms practically inefficient in terms of execution time.

In this paper, we substantially revise and extend the previous algorithms. While keep the elegance and simplicity in the algorithms' concept, we provide computationally efficient numerical implementations for both the (new) five-point and six-point algorithms in Matlab and C with excellent speed performance, without using any off-the-shelf symbolic mathematics devices (as we did previously). Our new algorithms, freely available on the web (from the authors' web-site), are substantially faster than most other publicly available implementations, and is competitive with Nistér's own optimized binaries [4].

Polynomial eigenvalue problem. At the heart of our new approach is the need to solve a polynomial eigenvalue problem. Given a square matrix $\mathtt{C}(w)$ with polynomial entries in the single variable w, the polynomial eigenvalue problem finds those values of w for which $\mathtt{C}(w)$ is singular. The Matlab function *polyeig* implements a solution of this problem. However, for the geometric problems we are interested in, only real eigenvalues are of interest, and in some cases only positive ones. We describe an algorithm, written in C++, which finds the required solution, with a substantial speed advantage. First, we directly find $\det(\mathtt{C}(w))$, which is a polynomial in w, followed by use of Sturm sequences to find the real (or positive real) roots of this polynomial. Computing the polynomial determinant is carried out by quotient-free Gauss-Jordan elimination [5]. Efficient numeric techniques are used to maintain accuracy, most notably the propose of using a least-squares technique (i.e. Levinson-Durbin iteration) for carrying out polynomial division,

• *Richard Hartley and Hongdong Li are with the RSISE, Research School of Engineering, College of Engineering and Computer Science, the Australian National University and NICTA (National ICT Australia). Email: firstname.lastname@anu.edu.au*

using Toeplitz matrices. This is proven crucial for acceptable accuracy.

Prior Arts. An excellent background for the 5-point and 6-point geometric problems considered here is given in previous works, and it would be redundant to repeat it here. The most-cited, and most directly comparable method is that of Nistér [4]. Other important methods have been proposed by Stewenius [6], which is one of the fastest implementations currently available (to the authors). In addition, methods based on Gröbner basis and modular arithmetic have been proposed [7], [8]for solving minimal case vision problems. Kukelova et al [9] suggested the use of the polynomial eigenvalue problem, and the Matlab *polyeig* function as a means to solving the 5-point problem.[1]

However, earlier formulation of the problem in [2], [3] actually involves solving a polynomial eigenvalue problem, though it was not explicitly identified as such there. We use *polyeig* in a Matlab implementation of our algorithm, but our fastest implementation is in C++, running considerably fast.

2 Basic Two-view Geometry

We assume the reader is very familiar with epipolar geometry ([10], [11]). However, for ease presentation, we nevertheless list some basic results without elaboration.

Two corresponding image points $\mathbf{x}$ and $\mathbf{x}'$ from a pair of images are related by a fundamental matrix $\mathtt{F}$ according to the relationship

$$\mathbf{x}'^{\top}\mathtt{F}\mathbf{x} = 0. \tag{1}$$

A valid $\mathtt{F}$ must satisfy the following singularity condition:

$$\det \mathtt{F} = 0. \tag{2}$$

The fundamental matrix depends only on the relative displacement and orientation of the two cameras, and their calibration matrices. It has seven degrees of freedom, since it has 9 elements, but it is defined only up to scale, and satisfies the singularity condition.

Fully-calibrated case. If the cameras are fully-calibrated, with calibration matrices $\mathtt{K}$ and $\mathtt{K}'$, then the fundamental matrix is related to an *essential matrix*, denoted by $\mathtt{E}$, as follows.

$$\mathtt{K}'^{-\top}\mathtt{E}\mathtt{K}^{-1} = \mathtt{F}. \tag{3}$$

Since $\mathtt{K}$ and $\mathtt{K}'$ are nonsingular, the matrix $\mathtt{E}$ must also satisfy the condition $\det \mathtt{E} = 0$.

The essential matrix $\mathtt{E}$ is a faithful representation of the relative placement (translation and rotation, up to a scale) of the two cameras, and hence it has only five degrees of freedom. Consequently, to be a valid essential matrix, $\mathtt{E}$ must further satisfy two more constraints, which may be expressed in the condition

$$2\,\mathtt{E}\mathtt{E}^{\top}\mathtt{E} - \mathrm{tr}(\mathtt{E}\mathtt{E}^{\top})\mathtt{E} = 0. \tag{4}$$

This actually gives nine equations in the elements of $\mathtt{E}$, but only two of them are algebraically independent.

Using (3) to replace $\mathtt{E}$ by $\mathtt{F}$ in (4) leads to a condition

$$2\,\mathtt{K}'^{\top}\mathtt{F}\mathtt{K}.\mathtt{K}^{\top}\mathtt{F}^{\top}\mathtt{K}'.\mathtt{K}'^{\top}\mathtt{F}\mathtt{K} - \mathrm{tr}(\mathtt{K}'^{\top}\mathtt{F}\mathtt{K}.\mathtt{K}^{\top}\mathtt{F}^{\top}\mathtt{K}')\,\mathtt{K}'^{\top}\mathtt{F}\mathtt{K} = \mathtt{0}$$

which may be rewritten as

$$2\,\mathtt{F}\Omega\mathtt{F}^{\top}\Omega'\mathtt{F} - \mathrm{tr}(\mathtt{F}\Omega\mathtt{F}^{\top}\Omega')\,\mathtt{F} = \mathtt{0} \tag{5}$$

where Ω and Ω' represent $\mathtt{K}\mathtt{K}^{\top}$ and $\mathtt{K}'\mathtt{K}'^{\top}$ respectively.

Unknown focal-length case. For most modern CCD or CMOS cameras, it is often reasonable to assume the cameras have square-shaped pixels, and the principal point coincide with the image center. In this case, if a pair of images are taken with a single moving camera with fixed intrinsic, the only unknown camera parameter is the constant but unknown focal length, and we may solve for the relative pose from six or more point correspondences. Assuming the only unknown calibration parameter of the camera is the focal length, the calibration matrix of the cameras is equal to $\mathtt{K} = \mathrm{diag}(f, f, 1)$, where f is the focal length. In this case $\Omega = \Omega' = \mathtt{K}\mathtt{K}^{\top} = \mathrm{diag}(f^2, f^2, 1)$.

3 Problem Formulation

In this section, we will present the main mathematical formulations of the five-point problem, and the six-point (focal length) problem in sequel. In particular, we will show how these minimal case problems are formulated as the polynomial eigenvalue problems, and how they can be addressed by the "hidden variable technique".

3.1 The five-point problem

To illustrate this (hidden variable) technique, we first consider the five-point problem, i.e., estimating the essential matrix $\mathtt{E}$ from five fully-calibrated point correspondences.

To do this, we use the three equations (1), (2) and (4). Note that (1) is satisfied by the essential matrix, as long as the points $\mathbf{x}$ are expressed in calibrated image coordinates. Given 5 points correspondences $\mathbf{x}'_i \leftrightarrow \mathbf{x}_i$, we use (1) to form a set of 5 equations in the 9 entries of the essential matrix. The four-dimensional (right) null-space of this matrix provides a basis of 4 linearly independent matrices $\mathtt{E}_w$, $\mathtt{E}_x$, $\mathtt{E}_y$, $\mathtt{E}_z$. Since the essential matrix must lie in this null space, it may be written in the form

$$\mathtt{E} = w\mathtt{E}_w + x\mathtt{E}_x + y\mathtt{E}_y + z\mathtt{E}_z\ , \tag{6}$$

where w, x, y and z are unknown variables. The next step is to use equations (2) and (4) to solve for these variables, and hence find the value of the essential matrix $\mathtt{E}$.

By expanding the determinant in (2) we obtain a cubic equation in the variables w, x, y and z. In a similar

1. Given a matrix $\mathtt{C}(w)$ with entries that are polynomials in w, the polynomial eigenvalue problem finds the "eigenvalues" w and corresponding "eigenvectors" $\mathbf{X}$ such that $\mathtt{C}(w)\mathbf{X} = 0$.

way, expanding the products in (4) leads to 9 equation, one for each entry of the matrix, also cubic in these variables. Although these equations are algebraically redundant, we use them all, resulting in a set of 10 cubic equations in the 4 variables. Since the equations obtained in this way are homogeneous, we may set one of the variables z to 1 (assuming that the true value of z is not zero in any sought solution). Note that even when the true z happens to be zero,i.e. a case which would fail out algorithm when applied alone, it will cause little effect when the algorithm is used in combination with RANSAC.

Next, we treat one of the variables w as a *parameter* (i.e. a *hidden variable*), and express each equation as a combination of monomials in the remaining variables x and y. Since the equations are cubic, we need to consider the set of monomials in x, y of degrees at most 3. These may be formed into a vector

$$\mathbf{X} = (1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3)^\top \ . \tag{7}$$

Note that there are 10 such monomials. The equations (2) and (4) may be expressed as linear combinations of these monomials, where the coefficients are polynomials in the hidden variable w. We obtain a set of 10 equations in the 10 monomials.

Example. To give the reader a concrete example, let us consider the following problem in which

$$\mathtt{E} = w \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 0 \end{bmatrix} + x \begin{bmatrix} 0 & 0 & 0 \\ -2 & 1 & 2 \\ 0 & 1 & 0 \end{bmatrix} + y \begin{bmatrix} 0 & 0 & 3 \\ 0 & 0 & 1 \\ 2 & -2 & 0 \end{bmatrix} + z \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The determinant of this matrix is

$$\begin{aligned} \det \mathtt{E} = & -w^3 - 2w^2x + 4w^2y + 7wxy - 6x^2y - 4wy^2 \\ & + 6xy^2 + w^2z + 3wxz - 2x^2z - 2wyz + 2xyz \end{aligned}$$

which with $z = 1$ is equal to

$$\begin{aligned} (w^2 - w^3, 3w - 2w^2, -2w + 4w^2, -2, 2 + 7w, -4w, 0, \\ -6, 6, 0) \cdot (1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3)^\top \ . \end{aligned}$$

This gives one linear equation in the monomials. The other equations resulting from (4) may be expressed in a similar way, resulting in a set of equations as given in table 1. Since this set of equations must have a non-zero solution for some value of w, the determinant of the matrix must be zero. The determinant is a polynomial $p(w)$, which may be solved to find possible values of w. For each root of $p(w)$, one may then compute the null-space of the matrix to find the value of the monomial vector $(1, x, y, \ldots, y^3)$. (Note that one must find the solution in which the first entry is 1.) Since x and y are entries in this vector, we immediately obtain their values. Finally, the essential matrix is obtained by substituting the solution $(w, x, y, 1)$ in (6).

It is of some importance to note that the polynomial $p(w)$ obtained as the determinant of the coefficient matrix has degree 10. To see this, observe that in the example equation set in table 1 the polynomials in w occuring in the columns of the matrix have degrees $3, 2, 2, 1, 1, 1, 0, 0, 0, 0$ respectively. Thus, the degree of the determinant is at most 10. To see that this is always true, observe that each entry in the matrix corresponds to one or more terms in a cubic polynomial, and hence must have degree at most 3. However, the columns of the matrix are multiplied by the monomials $(1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3)$, these monomials having degrees $0, 1, 1, 2, 2, 2, 3, 3, 3, 3$ respectively. Since the degree of each product can not exceed 3. the terms in each column have the stated degrees in w, and hence, the degrees of the columns sum to 10 as required. this confirms a well-known result that the five-point minimal case problem have 10 (complex) solutions in general.

3.2 Relationship to the hidden variable technique

The above algorithm can be loosely viewed as an instance of the *hidden variable* technique, a well-known technique for solving systems of polynomial equations [12] in several variables.

In the original form of the hidden variable technique, all but one of the variables of the polynomial system are temporarily treated as constants, resulting in a set of equations in the remaining univariate. By forming the *algebraic resultant*, one may eliminate this one variable, thereby reducing the number of equations and variables.

In our above described five-point algorithm, however, we treat just one of the variables as a parameter, and use linear methods to eliminate all the other variables at once. This leads to a single polynomial equation in the parameter, which is easily solved. Subsequently the values of the other variables may be computed. Note however, we do not claim this method as a fully general-purpose technique for solving polynomial equations, but it works very well for computation of the essential matrix under varying conditions, as we will show later, which is the main topic of this paper.

To give a general description to our (modified) hidden variable technique, consider a system of M homogeneous polynomial equations of degree d in N variables, say, $p_i(x_1, x_2, ..., x_N) = 0$, for $i = 1, 2, ..., M$. In the typical two view cases one encounters, the set of equations is redundant, so $M > N$. We select one of the unknowns (for example, x_1) as a so-called hidden variable. Now, we may think of each equation as being expressed as a linear combination of all the monomials in $x_2, \ldots, x_n$ of degree at most d, where the coefficients are polynomials in x_1. In this way, the complete set of equations may be expressed in matrix form as

$$\mathtt{C}(x_1)\mathbf{X} = 0 \ , \tag{9}$$

where the entries of the coefficient matrix $\mathtt{C}$ are polynomials in the *hidden variable* x_1, and $\mathbf{X}$ is a vector consisting of the monomials of degree at most d in all other $N-1$ variables (say, $x_2, x_3, \cdots, x_N$). If by good luck the number of equations equals the number of monomial terms in the vector $\mathbf{X}$ (i.e. the matrix $\mathtt{C}$ is square), then

$$\begin{bmatrix}
w^2-w^3 & 3w-2w^2 & -2w+4w^2 & -2 & 2+7w & -4w & 0 & -6 & 6 & 0 \\
-2w & -4w-6w^2 & 4-6w+4w^2 & -8-4w & -4-8w & 12-8w & 0 & -24 & -12 & 0 \\
2w^2 & 2+6w-2w^2 & -4+8w+8w^2 & 4-6w & 8+10w & -12+14w & 0 & 12 & 6 & 0 \\
2+2w^3 & 4w+6w^2 & 12+2w+2w^2 & -2+4w & 8+8w & 20+18w & 0 & -6 & 24 & 6 \\
-2w^2 & 4-4w-6w^2 & 2w+4w^2 & -14w & 20+12w & 4-8w & -16 & -12 & 8 & 0 \\
-2w+2w^2+2w^3 & -2+6w+8w^2 & -8w+12w^2 & 4+20w & -12+10w & -4-2w & 10 & -8 & 8 & 0 \\
2w+2w^2 & 4+8w+6w^2 & 2+2w+12w^2 & 2+20w & 32w & 6+8w & 16 & 24 & 12 & 2 \\
2w^3 & -4w+6w^2 & 6w+2w^2 & -8-4w & -4+4w & -12-10w & -4 & 8 & -16 & -4 \\
2w+2w^2 & 6w+6w^2 & 8w-4w^2 & 4+2w & -4-10w & 12+12w & -6 & -4 & 14 & 4 \\
2 & 4w+6w^2 & 6-6w-4w^2 & 14w & -8w & 18-16w & 4 & -22 & -12 & 0
\end{bmatrix}$$

TABLE 1: *Example of a matrix* $\mathtt{C}(w)$ *resulting from the* 5*-point algorithm. This matrix satisfies the equation* $\mathtt{C}(w)(1,x,y,x^2,xy,y^2,x^3,x^2y,xy^2,y^3)^\top = \mathbf{0}$.

the equation system will have non-trivial solutions only if $\det \mathtt{C}(x_1) = 0$. This leads to a polynomial equation in x_1, which may be solved to find x_1. For each solution x_1, the vector $\mathbf{X}$ may be found as the generator of the null-space of $\mathtt{C}(x_1)$, and the values of the other variables $x_2, \ldots, x_N$ may be extracted from $\mathbf{X}$.

Note that in this procedures one eliminates $N-1$ variables all at once, quickly leading to a solution for the hidden variable x_1.

Relaxation. It is easy to see that the above general algorithm we have just described may be considered as a relaxation procedure, in that we have effectively replaced the monomials x^2, xy, y^2, x^3, x^2y, xy^2 and y^3 by independent variables, and have relaxed (ignored) the evident relationship they bear to each other and to the variables x and y.

It is clear from the derivation that any valid essential matrix corresponding to the point correspondences will be found in this way. It is not, however, self-evident that every solution found in this way will give rise to a valid essential matrix. There appears to be no guarantee that the solution for the null-space generator of the matrix $\mathtt{C}(w)$ in (9) gives a monomial vector $(1, x, y, \ldots, y^3)$ satisfying these implicit constraints.

Using results of [13], however, it can be proved that all the solutions found in this way are valid essential matrices, at least for generic inputs. It was shown there that in order to find the essential matrices compatible with 5 point matches, it is necessary to solve a 10-th order polynomial defined over the field extension of the rationals containing the input data. In addition, this polynomial is generically irreducible and unique. Since the polynomial given by our method has degree 10, it must be the minimal polynomial required to find the solutions w.

3.3 The six-point problem

Now we consider the six-point relative orientation problem, assuming the only unknown calibration parameter of the camera is the focal length. We show how this problem, despite more difficult, can be similarly addressed by the hidden variable technique.

Given six point correspondences, the set of equations arising from (1) has a 3-dimensional null-space, and we may write

$$\mathtt{F} = x\mathtt{F}_x + y\mathtt{F}_y + z\mathtt{F}_z \ . \tag{10}$$

The condition $\det \mathtt{F} = 0$ provides, as before, a cubic equation in the unknowns x, y and z. We write $w = f^2$, the fourth unknown variable. Now, from (5), we obtain 9 further equations in all four variables. The equations so obtained are different from those obtained in the five-point problem, in that they are cubic and homogeneous in x, y and z, but additionally quadratic and non-homogeneous in w.

We treat w as a hidden variable, and express the equations in terms of the monomials $(z^3, xz^2, yz^2, x^2z, xyz, y^2z, x^3, x^2y, xy^2, y^3)$. Once again we have 10 equations in 10 monomial unknowns. The first equation (corresponding to (2)) does not involve w; the subsequent rows are quadratic in w. Consequently one expects the determinant $p(w) = \det \mathtt{C}(w)$ to have degree 18, which is indeed the case. However, calculations showed that $p(w)$ always has the form $p(w) = w^3\bar{p}(w)$, where $\bar{p}(w)$ is a degree 15 polynomial. Since a focal length of 0 is meaningless, we can ignore the zero-roots corresponding to the factor w^3 and find the 15 roots of $\bar{p}$.

We give a proof that $p(w) = \det \mathtt{C}(w)$ is of the form $w^3\bar{p}(w)$. We denote the last 9 rows of the matrix $\mathtt{C}(w)$ by $\hat{\mathtt{C}}(w)$. The entries of $\hat{\mathtt{C}}(w)$ are derived from (5), where $\Omega = \Omega' = \text{diag}(f^2, f^2, 1) = \text{diag}(w, w, 1)$.

We wish to find the rank of the matrix $\hat{\mathtt{C}}(0)$; when $w = 0$, and $\Omega = \text{diag}(0, 0, 1)$, we find that $\Omega\,\mathtt{F}^\top\,\Omega = f_{33}\,\Omega$, where f_{33} is the bottom right element of $\mathtt{F}$. In this case, the matrix in (5) simplifies to

$$\begin{aligned} 2\,\mathtt{F}\,\Omega\,\mathtt{F}^\top\,\Omega\,\mathtt{F} - \text{tr}(\mathtt{F}\,\Omega\,\mathtt{F}^\top\,\Omega)\,\mathtt{F} &= 2\,f_{33}\,\mathtt{F}\,\Omega\,\mathtt{F} - f_{33}^2\,\mathtt{F} \\ &= f_{33}\,(2\,\mathtt{F}\,\Omega\,\mathtt{F} - f_{33}\,\mathtt{F}) = f_{33}\,\mathtt{G}, \end{aligned} \tag{11}$$

where $\mathtt{G}$ is so defined. Now, each element of $\mathtt{F}$ is a linear expression in the unknowns x, y and z, so the matrix $\mathtt{G} = 2\mathtt{F}\,\Omega\,\mathtt{F} - f_{33}\mathtt{F}$ has entries that are quadratic in these unknowns. Thus, if $\text{vec}(\mathtt{G})$ is a vector made up of the 9 elements of $\mathtt{G}$, then we may write $\text{vec}(\mathtt{G})$ as a matrix product

$$\begin{aligned}\mathrm{vec}(\mathtt{G}) &= \mathtt{B}_{9\times 6}(z^2, xz, yz, x^2, xy, y^2)^\top \\ &= \mathbf{b}_{zz}\, z^2 + \mathbf{b}_{xz}\, xz + \ldots + \mathbf{b}_{yy}\, y^2\end{aligned}$$

where $\mathbf{b}_{zz}, \mathbf{b}_{xz} \ldots, \mathbf{b}_{yy}$ are the columns of $\mathtt{B}$ that are multiplied with the monomials $z^2, xz, \ldots, y^2$ in this product. Now, f_{33} is also linear in x, y and z; if $f_{33} = a_z z + a_x x + a_y y$, we may write

$$\begin{aligned}&\hat{\mathtt{C}}(0)(z^3, xz^2, \ldots, y^3)^\top = \mathrm{vec}(f_{33}\,\mathtt{G}) = f_{33}\,\mathrm{vec}(\mathtt{G}) \\ &\quad = (a_z z + a_x x + a_y y)(\mathbf{b}_{zz} z^2 + \mathbf{b}_{xz} xz + \ldots + \mathbf{b}_{yy} y^2) \\ &\quad = (a_z \mathbf{b}_{zz}) z^3 + (a_x \mathbf{b}_{zz} + a_z \mathbf{b}_{xz}) xz^2 + \ldots + (a_y \mathbf{b}_{yy}) y^3.\end{aligned}$$

The point being made here is that the 10 columns of $\hat{\mathtt{C}}(0)$ may be written as linear combinations of the 6 columns $\mathbf{b}_{zz}, \mathbf{b}_{xz}, \ldots, \mathbf{b}_{yy}$, and hence $\hat{\mathtt{C}}(0)$ has rank at most 6. Consequently, $\mathtt{C}(0)$ has rank at most 7, so all 8×8 minors have vanishing determinant. It follows that the smallest degree term in $\det \mathtt{C}(w)$ is the w^3 term, so $\det \mathtt{C}(w)$ has three zero-roots, as claimed.

We may use this observation to reduce the matrix $\mathtt{C}(w)$ prior to taking the determinant, so that a polynomial of degree 15 is directly obtained. This provides benefits in terms of accuracy and stability. Since it has rank 6, row operations on the matrix $\hat{\mathtt{C}}(0)$ can be used to reduce it to a form in which the first three rows are identically zero. Applying these operations to the polynomial matrix $\hat{\mathtt{C}}(w)$ yields a matrix in which these first 3 rows are polynomials divisible by w. They may therefore be divided by w, thereby reducing $\mathtt{C}(w)$ to a matrix with determinant of degree 15. The first row of $\mathtt{C}(w)$ contains real (degree-zero) entries; the next three rows have degree 1 and the final six rows have degree 2. The determinant will be expected to have degree 15, which confirms the number of solutions to this problem determined in [7].

An alternative parametrization is to set $w = 1/f^2$, and set $\Omega = \mathrm{diag}(1, 1, w)$. Proceeding in the same way as before, one obtains a polynomial of degree 15 directly, and hence 15 possible solutions for the focal length and the essential matrix.

4 Efficient Numerical Solution

In this section, we will explain in detail how to numerically solve the obtained polynomial system very efficiently and accurately. Apart from our problem formulation, this represents a novel contribution of this paper, and had not been reported before.

From the above section, it is clear that: central to our method is the problem of computing the null-space of the matrix $\mathtt{C}(w)$. This is essentially the polynomial eigenvalue problem, which has been considered in several papers ([9], [14]). In fact, there is a Matlab function–*p*olyeig–which directly finds a solution to this problem. However, as we will shown in the experiment section, the computational efficiency of Matlab's polyeig is rather low. In the absence of a freely available C (or C++) implementation of the polynomial eigenvalue problem, we decided to implement our own efficient algorithm to solve this problem, consisting of the two steps of computing $\det \mathtt{C}(w)$, finding the real roots of this polynomial, substituting the roots, and finding the null-space. The last step is easy, but we will discuss the first two steps in more detail.

It is worth noting that, the algorithms that we are considering in this paper are for minimal configurations, which means that the solutions will be exact. Conditions such as matrices having non-trivial null space, or polynomials having double roots or non-constant greatest common factor (GCD) will hold precisely, except for floating-point round-off error. They will not be affected by possible "noise" in the input data. Nevertheless, it will be seen that accumulation of round-off error in the computation of matrix determinants can lead to extreme inaccuracies if care is not taken.

4.1 Polynomial determinant.

Finding the determinant of a polynomial matrix is a classic problem. If one applies the usual Gaussian elimination algorithm to matrices of polynomials, the calculation leads to intermediate matrices whose entries are quotients of polynomials. The degrees of numerator and denominator in these quotients can become quite large. This leads to a complex and inefficient algorithm. We realized that this problem can be avoided by using the *quotient-free Gaussian elimination* algorithm, e.g. [5], in which polynomial quotient terms are avoided.

Specifically, let $\mathtt{A}$ be a matrix with polynomial entries (or more generally, entries in a commutative ring). The following algorithm finds its determinant.

```
m = 1; // m is a polynomial
for k=1 to n-1 do begin
  for i=k+1 to n do begin
    for j=k+1 to n do begin
      A(i,j)=(A(k,k)A(i,j)-A(i,k)A(k,j))/m;
    end
  end
  m = A(k,k);
end
return A(n,n);
```

Pivoting is then used to avoid the eventuality that $\mathtt{A}_{kk} = 0$. This algorithm requires polynomial multiplication, but the division by m keeps the degree as low as possible. Crucial is the observation that the division by m will be exact (without remainder) so that rational polynomial expressions do not occur. This trick is stated by Bareiss ([5]) to have been known to Jordan.[2]

Note that this method may also be used to find determinants of integer matrices, without using rational or real arithmetic. This remark may be used to understand the main issue addressed by the quotient-free algorithm.

2. There are two Jordans whose names are associated with matrix computations, Wilhelm Jordan (as in Gauss-Jordan elimination) and Camille Jordan (Jordan normal form). Presumably Bareiss means Wilhelm Jordan.

4.2 Toeplitz matrices.

The quotient $(\mathtt{A}_{kk}\mathtt{A}_{ij} - \mathtt{A}_{ik}\mathtt{A}_{kj})/m$ may be computed by polynomial long-division. Unfortunately, this is numerically unstable, and even for matrices of the dimensions we need to handle, accumulated error due only to round-off can make the results wildly wrong. We address this problem by using a least-squares technique that mitigates the effect of round-off error.

In dividing a polynomial p by m we find a quotient q and remainder r such that $p = qm + r$, while minimizing $\|r\|$, where $\|r\|$ is the sum of squares of coefficients of r, and r has degree at most equal to that of p. (By contrast, in long-division the degree of the remainder r is at most equal to $\deg(m) - 1$). Now, observe that a polynomial product mq may be written as a matrix-vector product $\mathtt{M}\mathbf{q}$, where $\mathtt{M}$ is a Toeplitz matrix and $\mathbf{q}$ is the vector of coefficients of q. A Toeplitz matrix is a matrix in which each descending diagonal from left to right is constant. It is an easy observation that the above-mentioned least-squares problem naturally leads to a Toeplitz system, where the dimensionality of the system (i.e. columns of the Toeplitz matrix) equals to the number of the coefficients of polynomial q.

Using this, we may express the problem as one of finding $\mathbf{q}$ that minimizes $\|\mathtt{M}\mathbf{q} - \mathbf{p}\|^2$, where $\mathbf{q}$ and $\mathbf{p}$ are the vectors of coefficients of the polynomials p and q, and $\mathtt{M}$ is the Toeplitz matrix formed from the entries of m. The least-squares solution $\mathbf{q}$ may be computed by solving the linear equation system $\mathtt{M}^\top\mathtt{M}\mathbf{q} = \mathtt{M}^\top\mathbf{p}$, where $\mathtt{M}^\top\mathtt{M}$ is a symmetric Toeplitz matrix.

4.3 Levinson-Durbin algorithm.

Solving sets of equations involving Toeplitz matrices is especially efficient, for example, by the Levinson-Durbin algorithm [15]. It can solve a set of Toeplitz-structured linear equations in time $O(n^2)$, where n is the dimension of the system of equations.[3]

Given a set of equations $\mathtt{A}\mathbf{x} = \mathbf{b}$ where $\mathtt{A}$ is a Toeplitz matrix, the Levinson-Durbin algorithm consists of two stages. In the first stage, a set of *back-vectors* are computed, that depend only on the matrix $\mathtt{A}$; in the second stage, the solution $\mathbf{x}$ is computed as a linear combination of back vectors. In the quotient-free determinant algorithm, the division by any given m takes place several times. This can be used further to speed up the algorithm. Specifically, the back vectors for the matrix $\mathtt{M}^\top\mathtt{M}$ may be computed one for each m, and then used several times to compute the solution to the different linear systems corresponding to the different polynomial divisions.

Using this least-squares technique for polynomial division gives a large improvement in accuracy, and allows us to find the determinant of the matrices we need to consider.

3. A convenient description of the Levinson-Durbin algorithm is given on the web page http://en.wikipedia.org/wiki/Levinson_recursion.

More details. We give a more detailed explanation of the advantage of the least-squares Toeplitz method over straight long-division. In computing the quotient p/m at line 5 of the quotient-free Gaussian elimination algorithm we expect the division to be exact, so that $p/m = q$, or $p = mq$. However, because of numerical round-off error, the polynomial $p = \mathtt{A}_{kk}\mathtt{A}_{ij} - \mathtt{A}_{ik}\mathtt{A}_{kj}$ may be inexact; let $p' = p + \delta_p$ be the actual noisy value of p. The quotient p'/m is no longer exact. Instead, we compute a noisy quotient q' and remainder r such that $p' - r = mq'$. We then accept the quotient q', and ignore the remainder r. Knowing that p' is noisy, but not knowing the correct p, it makes sense therefore to seek the smallest "increment" r such that $p' - r$ is an exact multiple of m. This leads to the least-squares problem just discussed, solved using Toeplitz matrices to find q. In this method, the only restriction on r is that its degree is no larger than that of p.

Example. To illustrate this, consider a specific numerical example. Let m be the polynomial $x + 2$, and p be a polynomial of degree 10 divisible exactly by m so that the correct quotient is $q = p/m$. Let p be perturbed by noise to a polynomial $p' = p + \delta_p$, where δ_p is the polynomial $\varepsilon(1 - x + x^2 - \ldots + x^{10})$, and ε is a small value. By the well-known Polynomial Remainder Theorem (a.k.a. little Bezout theorem), long division of p' by m results in the remainder $r = \delta_p(-2) = 2047\varepsilon$, and the quotient is $q' = q + \delta_q$ with error $\delta_q = \varepsilon(x^9 - 3x^8 + 7x^7 - \ldots - 1023)$, where $\|\delta_q\| \approx 1180.7\varepsilon$. On the other hand, the least squares division of p' by m results in a quotient $q' = q + \delta_q$ with error of magnitude $\|\delta_q\| = \|q - q'\| \approx 2.58\varepsilon < \|\delta_p\|$ and a remainder $\|r\| \approx 1.732\varepsilon$. Thus the Toeplitz algorithm gives result better than long division by some orders of magnitude.

Numerical tests. To illustrate the improvement in numerical accuracy for computation of polynomial determinants using the Toeplitz method, we computed the determinants of matrices $\mathtt{C}(w)$ of varying dimensions (from 6 to 20) with quadratic polynomial entries, chosen at random. To assess the accuracy of the polynomial determinant, we compute $d = \det \mathtt{C}(1)$ and $d' = \det \mathtt{C}(w)$ evaluated at $w = 1$. In other words, d is computed by evaluating the polynomial entries of the matrix at the value $w = 1$, followed by computing the determinant of the matrix with real entries. On the other hand, d' is found by first computing the polynomial determinant, and then evaluating the resulting polynomial at the value $w = 1$. With exact computation, the two results should be equal. We expect the value of d to be relatively accurate, since both evaluating polynomials and taking the determinant of a real matrix can be accomplished accurately. Therefore, we assess the accuracy of the result

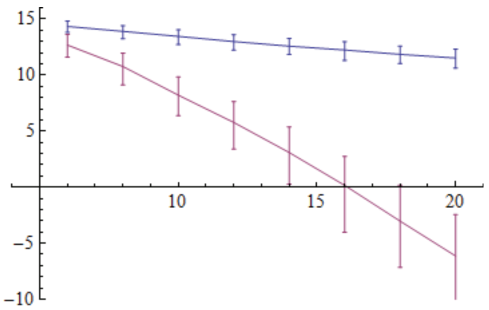


Fig. 1: *Plots of the number of significant figures of accuracy for quotient-free Gauss-Jordan elimination, using the Toeplitz method (top) and long division (bottom). The horizontal axis represents the dimension of the matrix, the vertical axis represents the relative accuracy* $-\log_{10}\varepsilon_{\rm rel}$ *(see text for a definition of* $\varepsilon_{\rm rel}$*). The graph shows the median relative accuracy, with error bars at 25-th and 75-th percentile. Note that a relative error of* 2.2×10^{-16} *(plotted as* 15.65 *on a negative logarithmic scale) approximately represents the smallest possible (non-zero) double precision roundoff error. A negative log error of* 7.22 *represents single-precision accuracy and any value less than* 2 *(1% relative error) is quite inaccurate. Note that the Toeplitz algorithm gives highly accurate results for all degrees up past* 20*, whereas the long-division algorithm fails completely for degrees past* 14.

by evaluating $\varepsilon_{\rm rel} = |(d'-d)/d|$, which measures the relative error. For each dimension, the determinants of 1000 different matrices are computed. In Fig 1, the negative log relative error $-\log(\varepsilon_{\rm rel})$ is plotted. This is the number of significant figures of accuracy of the result. Double precision arithmetic was used for all computations.

4.4 Finding roots of polynomial.

Since we are only interested in finding real roots of the polynomial $\det \mathtt{C}(w)$, we choose the method of Sturm sequences to do this. The Sturm sequence method was also used in Nistér's work [1], and the reader is also referred to a brief example in [12]. This method is quick and efficient, and is easily adapted to find only positive real roots, with a consequent substantial time saving. For example, in the 6 point problems, the polynomial involved is in fact a polynomial in $w = f^2$, where f is the focal length. In this case, we need only to find positive roots of the polynomial.

4.5 Root polishing

Although for most applications, the above algorithm gives results of sufficient accuracy, it would be nice to do better. This is easily effected by a simple polishing method that corrects the result to achieve machine accuracy. Previously this was done by an iterative optimization approach, similar to bundle-adjustment. Although [16] considers optimization of the essential matrix, little attention has been given in published work to achieving ultimate speed for 5-point bundle-adjustment.

In this paper, we propose a new and efficient polishing algorithm, which is specifically designed for solve the minimal case camera motion problems. Details are given below.

Any essential matrix has Singular Value Decomposition $\mathtt{E} = \mathtt{U}\widehat{\mathtt{I}}\mathtt{V}^\top$, where $\widehat{\mathtt{I}} = \mathrm{diag}(1,1,0)$ and $\mathtt{U}$ and $\mathtt{V}^\top$ are rotation matrices. Expressing the rotations in terms of Euler angle decomposition, we obtain $\mathtt{E} = (\mathtt{R}_u\,\mathtt{R}_v\,\mathtt{R}_w)\,\widehat{\mathtt{I}}\,(\mathtt{R}^\top{}_z\,\mathtt{R}^\top{}_y\,\mathtt{R}^\top{}_x)$, where $\mathtt{R}_u$, $\mathtt{R}_v$ and $\mathtt{R}_w$ are rotations about the x, y and z axes, respectively, as are also $\mathtt{R}_x$, $\mathtt{R}_y$, $\mathtt{R}_z$. Since $\mathtt{R}_z$ commutes with $\widehat{\mathtt{I}}$, we then move it across to the left and absorb it, resulting in a decomposition

$$\mathtt{E} = \mathtt{R}_u\,\mathtt{R}_v\,\mathtt{R}_w\,\widehat{\mathtt{I}}\,\mathtt{R}^\top{}_y\,\mathtt{R}^\top{}_x. \tag{12}$$

where to be specific,

$$\mathtt{R}_u = \begin{bmatrix} cu & -su & 0\\ su & cu & 0\\ 0 & 0 & 1\end{bmatrix} \;;\; \mathtt{R}_v = \begin{bmatrix} cv & 0 & sv\\ 0 & 1 & 0\\ -sv & 0 & cv\end{bmatrix} \;;\; \mathtt{R}_w = \begin{bmatrix} 1 & 0 & 0\\ 0 & cw & -sw\\ 0 & sw & cw\end{bmatrix}$$

where su, cu, *etc*, are the sine and cosine of the rotation angles; Rotations $\mathtt{R}_x$ and $\mathtt{R}_y$ have the same form as $\mathtt{R}_u$ and $\mathtt{R}_v$. The angles of each of these 5 rotations consititute a minimal parametrization of $\mathtt{E}$, with parameter vector $\theta = (\theta_u,\theta_v,\theta_w,\theta_x,\theta_y)$.

The singularities associated with representing rotations in terms of Euler angles do not concern us, since we will be using only small angle representations, as will be explained soon. Our strategy is to use iterative refinement of $\mathtt{E}$ based on this localized parametrization about the essential matrix $\widehat{\mathtt{I}}$.

In the iterative step, we need to compute a Jacobian with respect to the parameters. Consider a point correspondence $\mathbf{x}_i = (x_i, y_i, z_i) \leftrightarrow \mathbf{x}'_i = (x'_i, y'_i, z'_i)$ and define

$$\begin{aligned}\varepsilon_i &= \mathbf{x}'^\top_i\,\mathtt{E}\,\mathbf{x}_i\\ &= \mathbf{x}'_i\,(\mathtt{R}_u\,\mathtt{R}_v\,\mathtt{R}_w)\,\widehat{\mathtt{I}}\,(\mathtt{R}^\top{}_y\,\mathtt{R}^\top{}_x)\,\mathbf{x}_i\end{aligned}$$

Taking the derivative with respect to the parameter vector $\theta = (\theta_u,\theta_v,\theta_w,\theta_x,\theta_y)$ at the value $\theta = \mathbf{0}$, we obtain

$$\begin{aligned}\mathtt{J}_i|_0 &= \left.\frac{\partial\varepsilon_i}{\partial\theta}\right|_{\theta=\mathbf{0}}\\ &= (y_i z'_i,\; -x_i z'_i,\; x_i y'_i - y_i x'_i,\; z_i y'_i,\; -z_i x'_i)\end{aligned} \tag{13}$$

The matrix $\mathtt{J}$ is obtained by stacking the five rows corresponding to $\mathtt{J}_i|_0$ for each i.

The complete iterative algorithm for computing $\mathtt{E}$ is then as follows. Given an initial estimate $\mathtt{E}_0$ for the essential matrix, and point correspondences $\mathbf{x}^0_i \leftrightarrow \mathbf{x}^{0\prime}_i$,

1) Compute the decomposition $\mathtt{E}_0 = \mathtt{U}\widehat{\mathtt{I}}\mathtt{V}^\top$,
2) Repeat the following steps until convergence
   a) For $i = 1,\ldots,5$, transform the input points according to

$$\begin{aligned}\mathbf{x}'_i &\leftarrow \mathtt{U}^\top\mathbf{x}^{0\prime}_i\\ \mathbf{x}_i &\leftarrow \mathtt{V}^\top\mathbf{x}^0_i.\end{aligned}$$

   b) Compute the error vector ε, where

$$\varepsilon_i = \mathbf{x}'^\top_i\,\widehat{\mathtt{I}}\,\mathbf{x}_i = x'_i x_i + y'_i y_i.$$

   c) Using (13), evaluate $\mathtt{J}$ in terms of the $\mathbf{x}'_i$ and $\mathbf{x}_i$, and solve the equation $\mathtt{J}\delta_\theta = -\varepsilon_i$ to obtain $\delta_\theta = (\delta_{\theta_u},\ldots,\delta_{\theta_y})$

d) Set

$$\mathtt{U} \leftarrow \mathtt{U}\,\mathtt{R}_u(\delta_u)\,\mathtt{R}_v(\delta_v)\,\mathtt{R}_w(\delta_w)$$
$$\mathtt{V} \leftarrow \mathtt{V}\,\mathtt{R}_x(\delta_x)\,\mathtt{R}_y(\delta_y)$$

3) Output $\mathtt{E} = \mathtt{U}\,\widehat{\mathtt{I}}\,\mathtt{V}^\top$.

Observe how by transforming the input points $\mathbf{x}_i^0$ and $\mathbf{x}_i^{0\prime}$ by the current estimates of $\mathtt{U}$ and $\mathtt{V}$, we obtain a set of points that nominally satisfy the equation $\mathbf{x}_i'^\top\,\widehat{\mathtt{I}}\,\mathbf{x}_i = 0$. Localization about $\widehat{\mathtt{I}}$ makes the form of the Jacobian $\mathtt{J}$ and the error vector ε particularly simple.

Notes.

1) The words "until convergence" are to be interpreted freely. In practice, we use a fixed number of iterations (two or three). In addition, in some cases, because of round-off error near convergence, or non-convergence, the error $\|\varepsilon\|$ may actually increase after some iteration. In this case, we roll back to the previous values of $\mathtt{U}$ and $\mathtt{V}$, and exit the loop.
2) For efficiency, the updates to $\mathtt{U}$ and $\mathtt{V}$ in step (d) of the algorithm should be computed by applying separate Givens rotations to $\mathtt{U}$. In addition, if $\delta_{\theta_i} < 10^{-8}$, then $\cos(\theta_i) = 1.0$ and $\sin(\theta_i) = \theta_i$ within machine precision, and the update is particularly efficiently computed.
3) Computational overhead. The polishing algorithm is extremely rapid, expending no more than about $2\mu s$ per iteration. In addition, if it is used only when the error exceed some very small threshold, then the time requirement is even smaller. With polishing invoked only when the RMS error (per point) exceeds 10^{-12}, we observed a time penalty of only $0.3\mu s$ per essential matrix computed.

5 Implementations and Experiments

We concentrated our implementation effort on the 5-point and 6-point (pose plus focal length) algorithms. Both implementations (source code) can be found on the authors' web site [4].

5.1 Implement the five-point algorithm

The five-point algorithm was implemented both in Matlab and C++. Steps in the algorithm are given as follows:

1) From five point correspondences $\mathbf{x}_i' \leftrightarrow \mathbf{x}_i$ compute a basis $(\mathtt{E}_w, \mathtt{E}_x, \mathtt{E}_y, \mathtt{E}_z)$ for the space of possible essential matrices.
2) Setting $z = 1$ and treating w as a hidden variable, form the equation set $\mathtt{C}(w)\mathbf{X} = 0$, where $\mathbf{X}$ is the vector of 10 monomials in x and y.
3) Find the values of w for which this set of equations has a solution, and for each such w, solve for $\mathbf{X}$, and hence obtain values of x and y.
4) Reconstitute the required essential matrix $\mathtt{E}$ as a linear combination of the basis.

In this algorithm, steps 1 and 4 are straight-forward linear algebra, and step 2 is a simple matter of efficient bookkeeping. Step 3 in the algorithm may be done in various ways, and this is where our Matlab and C++ implementations differed.

The Matlab implementation is very simple and relatively efficient, without any significant attempt being made to achieve maximum speed. However, unlike earlier implementations described in [2], [3], it does not make use of any symbolic programming, and hence runs orders of magnitude faster. It uses the Matlab *polyeig* function to carry out step 3 of the algorithm.

The algorithm was also coded in C++ and interfaced to Matlab as a *mex* file, to allow direct comparison of timing. The algorithm used in the C++ implementation was described in section 4. In order to limit the number of polynomial divisions computed during the Gauss-Jordan elimination, columns of the matrix are eliminated in order of increasing polynomial-degree of the elements they contain. Note that the last 4 columns of the matrix in table 1 contain only constant terms, and columns $4-6$ only linear terms. To take advantage of this, we eliminate columns from the right.

Similarly to the Matlab version, we use Gauss-Jordan elimination on the last 4 columns of $\mathtt{C}(w)$ to reduce it to size 6×6 before finding its determinant. Note that these columns consist of constant (degree 0) entries, so Gauss-Jordan elimination is simple. Note that it is possible at the cost of a little more complexity to reduce the matrix to size 3×3, similar to the procedure outlined in [4], however this operation is specific to the 5-point problem, and hence not used in our generic procedure.

Finally, we considered Nistér's algorithm [4], which is essentially equivalent to carrying out column elimination on this matrix, with specific advantage being taken of the particular form of the matrix. We carefully re-implemented the algorithm following [4], with expended considerable effort to make our implementation as fast as possible and close to his original implementation. This is the algorithm referred to as 5pt-mex-opt below.

5.2 Test of speed

We give speed comparison for our and other people's implementations of the 5-point algorithm. The methods compared are as follows.

1) A pure Matlab implementation of our algorithm, as described in this paper (algorithm **5pt-Matlab**)
2) A hybrid Matlab/C++ implementation of our algorithm, in which the C++ routine is passed the input matches, and returns a the polynomial matrix $\mathtt{C}(w)$ and the equation $\mathtt{C}(w)\mathbf{X} = 0$ is solved using the Matlab *polyeig* function (algorithm **5pt-polyeig-hybrid**).
3) A hybrid method in which the C++ routine returns the determinant $\det(\mathtt{C}(w))$, which is a degree-10

4. http://users.cecs.anu.edu.au/~hartley,//users.cecs.anu.edu.au/~hongdong

polynomial. In Matlab, the roots of this polynomial are found using the companion-matrix technique, and for each real root the null space of $\mathtt{C}(w)$ (also computed in C++) is computed to provide solutions for x and y (algorithm **5pt-companion**). This allows a timing comparision between the use of the companion-matrix and Sturm sequence methods for root-finding.

4) A pure C++-*mex* implementation of our algorithm in which Matlab passes the input data to the mex routine, and a list of essential matrices is returned (algorithm **5pt-mex**).
5) A C++-*mex* implementation similar to 5pt-mex, in which special optimizations suggested by the specific structure of $\mathtt{C}(w)$ are incorporated. The resulting algorithm is very much the same as the one described in [4] and gives a good indication of how Nistér's algorithm will perform in an implementation with reasonable care taken to optimize for speed (algorithm **5pt-mex-opt**).
6) The Gröbner -basis algorithm (programmed by Chris Engel, dated 2004/07/22) by Stewenius et al; reported in [6], and available online. This has a Matlab front end with a C-*mex* helper (with non-transparent machine-generated C-code) which does most of the work. (algorithm **5pt-Stewenius-GB**).
7) The five-point algorithm using Matlab's *polyeig* function, proposed by Kukelova et al [9] (algorithm **5pt-Kukelova**).

The timing results are given in the following table. Times given are for a single computation of **all** Essential matrices from one set of five point matches. On the average, each set of point correspondences led to 4.6 essential matrices. The computer used is a 2009 laptop running at 2.5GHz. The results are obtained by averaging over 10,000 different random data sets, with a total of 100 runs for each data set (except for the pure Matlab implementations, which were run for fewer iterations). Timing is in microseconds (μs).

algorithm	**time** (μs)
5pt-Matlab	2971
5pt-Kukelova	1567
5pt-polyeig-hybrid	606
5pt-companion	305
5pt-mex	71
5pt-mex-opt	34
5pt-Stewenius-GB	250

Observations. Comparing 5pt-mex with 5pt-polyeig shows that our C++ method (see section 4) for solving the polynomial eigenvalue problem is many times faster than using the Matlab *polyeig* function. Of course, using *polyeig* has the advantage of convenience since it avoids the need to take the determinant of a polynomial matrix.

Similarly, comparing 5pt-mex with 5pt-companion shows the speed advantage of using Sturm sequences to find the roots of the polynomial, compared with using the companion matrix method.

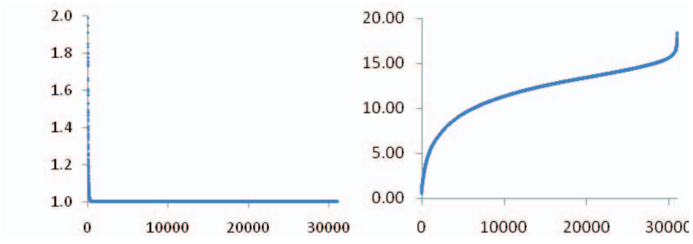


Fig. 2: *On the left the ratio s_1/s_2 of the first two singular values of the essential matrix found by trials with our 5-point algorithm. For a valid matrix, this ratio should be 1. In a small number of cases the algorithm gives an invalid matrix, due to numerical errors. On the right, for the same data, a plot of $-\log_{10}(1-s_2)$, assuming that $s_1 = 1$. This graph shows that for the majority of the estimates, the two singular values differ by about 10^{-10}.*

Next, using the problem-specific optimizations of [4] gives our fastest implementation. For comparison, a timing of $100-120\mu s$ is given for Nistér's algorithm in [4] for a 550 MHz computer from around 2004. Extrapolation would suggest a running time on our computer of less than $20\mu s$ for this algorithm but this is uncertain, since clock speed is not the only determiner of speed. Our implementation 5pt-mex-opt does not run as fast as this but comes close.

Finally, comparison was also made with the 5-point algorithm of Kukelova et al [9] . Their times are 8ms for a Matlab implementation using *polyeig* (our test of Kukelova's this five-point code gave $1567\mu s$ on our computer). This should be compared with our 5pt-Matlab timing of 2.96ms.

Also according to them, the Gröbner implementation (presumably the same as the Stewenius-GB algorithm which we tested) took $1ms$. For us, it ran in $251\mu s$, which suggests they were using a 4–5 times slower computer.

In Fig 2 we provide a plot of results derived from randomly generated 5-point data to illustrate the accuracy of the essential matrices computed over 7500 trials, resulting in 31000 found essential matrices. The results show the ratio of the two non-zero singular values of $\mathtt{E}$. Note that there are occasional invalid essential matrices computed. This is not a serious problem in the context of a RANSAC algorithm, as has been remarked in [1], [4].

5.3 Implement and test of the 6-point algorithm

We also implemented our six-point algorithm in both C++ and Matlab using the same approach, and most of the code shared with our 5-point algorithm. Running time is slower for the 6-point algorithm, partly because the polynomial $\det \mathtt{C}(w)$ has degree 15 instead of 10.

We implemented and tested three versions of our algorithm: *6pt-Matlab-polyeig*, which is based on pure Matlab using Matlab's native *polyeig* function; *6pt-polyeig-hybrid*, which partially uses C++-mex code to reduce the matrix size; *6pt-mex*, which is a C++-mex implementation and

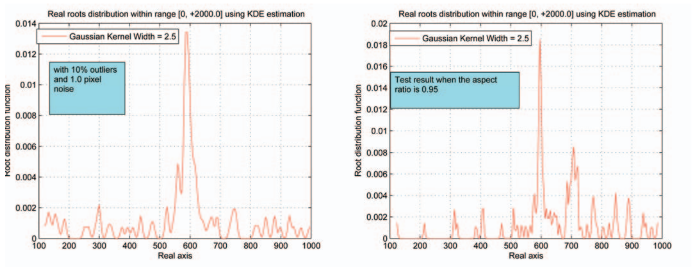


Fig. 3: *Estimate of the focal length from a pair of images; Left: with 10% outliers ; Right: with an error in the aspect ratio estimation: the true aspect-ratio is 0.95 but 1.00 was assumed.*

0.01%	0.1%	1.0%	50%	Method
11.15	12.29	13.40	15.53	5pt-Kukelova
9.92	11.61	13.03	15.37	5pt-Stewenius-GB
11.01	12.09	13.24	15.47	5pt-Matlab
4.26	6.60	9.05	14.07	5pt-mex
5.21	7.75	10.21	14.50	5pt-mex-opt

TABLE 2: *Accuracy of different 5-point algorithm methods for random data. For each method, one million essential matrices were computed, by each of the different algorithms. The computed essential matrix was evaluated on how well it fitted the input data, using the error measure $C(\mathtt{E})$, given in (14). We show $-\log(C(\mathtt{E}))$ for different percentiles. This indicates the approximate number of significant figures of accuracy in the result. See the discussion in the text.*

is called by Matlab. We made little effort to optimize our Matlab codes.

Speed is given in the following table. For comparison, we also tested Kukelova et al's Matlab code (6pt-Kukelova) based on polyeig [9], which is about five times faster than our pure Matlab implementation. The difference is accounted for by the way that Kukelova et al pre-compute the polynomial matrix $C(w)$, which is optimized (and presumably Macaulay-generated according to their paper [9]). For the C++ code provided by Stewenius et al for [7], our tests gave a timing of about 700 microseconds for their Gröbner basis based 6-point algorithm (*6pt-Stewenius-GB*).

algorithm	**time** (μs)
6pt-Matlab-polyeig	6747
6pt-polyeig-hybrid	1049
6pt-mex	166
6pt-Kukelova	1177
6pt-Stewenius-GB	700

Kernel voting. Our 6 point algorithm gives a direct solution for the focal length f, where $w = f^2$. There may be up to 15 solutions from a minimal set of 6 correspondences. One may find a unique best estimate for the focal length by a procedure similar to Kernel Density Estimation (otherwise known as Parzen windows) using many sampled minimal 6-point sets, either drawn from the correspondences in one set of images, or several pairs in a sequence of images with a non-zooming camera. The idea is simply to sum kernel functions centred at each of the individual focal length estimates. The result of such an experiment is shown in Fig 3, where the robustness of the technique to outliers and incorrect pixel aspect ratio is shown. Such a *kernel-voting* may be used as an alternative to RANSAC to remove outliers if the outlier ratio is low (e.g. less than 20% outliers).

6 Accuracy

Besides execution time, accuracy is another important performance index for camera motion estimation. In evaluating an algorithm for computing the essential matrix, there are two possible criteria. First, one requires that the matrix should be a valid essential matrix, satisfying the conditions (2) and (4). Second, it should fit the input data exactly, so that $\mathbf{x}_i'^{\top}\mathtt{E}\mathbf{x}_i = 0$ for each input correspondence $\mathbf{x}_i' \leftrightarrow \mathbf{x}_i$. The different algorithms perform differently against these two criteria. Since the latter often gives more consistent result empirically, and the result can be directly verified on the input images, we therefore choose the point-matching criterion.

$$C(\mathtt{E}) = \sqrt{\sum_{i=1}^{5}(\mathbf{x}_i'\widehat{\mathtt{E}}\mathbf{x}_i)^2} \tag{14}$$

where $\widehat{\mathtt{E}}$ is obtained from $\mathtt{E}$ (the output of the algorithm) by correcting to an exact essential matrix using the Singular Value Decomposition (SVD). That is, $\widehat{\mathtt{E}} = \mathtt{U}\widehat{\mathtt{I}}\mathtt{V}^{\top}$ where $\widehat{\mathtt{I}} = \mathrm{diag}(0, 0, 1)$, and $\mathtt{E} = \mathtt{U}\widehat{\mathtt{I}}\mathtt{V}^{\top}$ is the SVD. In addition, the points $\mathbf{x}_i'$ and $\mathbf{x}_i$ were normalized as 3-vectors such that $\|\mathbf{x}_i'\| = \|\mathbf{x}_i\| = 1$. In this way, we may compute a consistent accuracy measure for all the methods. Accuracy results are given in table 2.

6.1 Observations.

For all methods, the median error (50-th percentile, shown in the last column) is around 10^{-15}, which is close to machine precision. The first column of numbers shows that occasionally, for one case in $10,000$ (0.01 percentile), accumulated error reduces the precision of the result to about 4 or 5 decimal places for our *5pt-mex* and *5pt-mex-opt* implementations. Note that this is still better than the expected accuracy of the input data in most cases. (Four figure accuracy corresponds roughly to 0.1 pixel error for an image with a focal length of 1000 pixels). Thus, the accuracy of the computed result is adequate in 99.99% of the cases. For applications such as RANSAC, this is more than accurate enough.

Consistently, algorithms *5pt-mex* and *5pt-mex-opt* were substantially less accurate than the others. As pointed out by one reviewer, this is possibly due to the stopping criterion used in the implementation of Sturm-sequences. We set an error of 10^{-12} as the required relative accuracy of root-finding in all our experiments. Sharpening this to 10^{-16} gives a significant improvement in accuracy, but with significant increase in computation time.

0.01%	0.2%	1.0%	50%	Method
9.46	11.03	11.95	14.85	5pt-Kukelova
4.22	6.57	8.03	13.09	5pt-Stewenius-GB
6.94	8.69	9.70	13.44	5pt-Matlab
1.59	3.77	6.50	13.66	5pt-mex
1.59	2.96	5.00	12.75	5pt-mex-opt

TABLE 3: *Accuracy, before polishing, for small-disparity (1-degree) point correspondences. Displayed results show the number of decimal places of accuracy. All the algorithms lose some accuracy on small-disparity correspondences. However, results are accurate to over 6 decimal places at least 99% of the time (except for* 5pt-mex-opt*). Shown are the results of over one million essential matrix calculations.*

0.01%	0.1%	1.0%	50%	Method
15.80	15.88	15.98	16.36	5pt-Kukelova
15.80	15.88	15.98	16.36	5pt-Stewenius-GB
15.79	15.88	15.98	16.36	5pt-Matlab
15.79	15.88	15.98	16.36	5pt-mex
15.80	15.88	15.98	16.36	5pt-mex-opt

TABLE 4: *Accuracy, after polishing, for random data. This should be compared with the results given in table 2. After polishing, all algorithms give close to optimal results in at least* 99.99% *of cases. Our new algorithm* 5pt-mex *gives fewer than* 15 *decimal places of accuracy in only about* 20 *cases out of one million computed.*

We also note that the algorithm *5pt-Kukelova* was consistently better than the other algorithms by a fraction of a significant digit accuracy (before the root polishing step is applied).

6.2 With more difficult data

The previous tests were done with 5 randomly selected point correspondences. Such point correspondences are not quite realistic, in that they may have arbitrarily large disparity. In most real situations, the *essential disparity* of point correspondences is usually small.

We define the term *essential disparity* to denote the disparity between corresponding points, once the effect of gross image misalignment is removed. One may make a formal definition as follows. If $\mathtt{E} = \mathtt{U}\hat{\mathtt{I}}\mathtt{V}^\top$ is the essential matrix, and $\mathbf{x} \leftrightarrow \mathbf{x}'$ is a point correspondence, then the essential disparity is the angle between the vectors $\mathtt{U}^\top\mathbf{x}'$ and $\mathtt{V}^\top x$.

We carry out experiments on sets of 5 points correspondences, for which the average essential disparity (over 5 points) is one degree. Comparisons of the accuracy before and after root polishing under different conditions are provided in Table-3, -4 and -5.

7 Other Extensions

The general technique described in this paper can be extended to solve several other two-view minimal case problems. Some examples will be discussed briefly in this section, only formulations are given and without going into full implementation detail.

0.01%	0.2%	1.0%	50%	Method
15.69	15.80	15.88	16.28	5pt-Kukelova
15.64	15.80	15.88	16.28	5pt-Stewenius-GB
15.70	15.80	15.88	16.28	5pt-Matlab
2.06	15.71	15.87	16.28	5pt-mex
1.93	7.89	15.84	16.28	5pt-mex-opt

TABLE 5: *Accuracy, after polishing, for small-disparity correspondences. These results show that almost all algorithms (except* 5pt-mex-opt*) are obtaining close to ultimate accuracy* 99.8% *of the time, even on point sets with small essential disparity. If polishing is invoked only when the error exceeds* 10^{-12}*, then it adds only about* $2\mu s$ *to the average computation time for each* 5*-point correspondence set.*

7.1 Non-square pixels

By similar techniques, one can solve for the case where $\mathtt{K}_1 = \mathtt{K}_2 = \mathrm{diag}(f_1, f_2, 1)$, that is the two cameras are the same, but pixels are not square. From 7 or more points, one may compute a fundamental matrix, satisfying the condition $\det(\mathtt{F}) = 0$. In the case of 7 points, there may be 3 possible solutions [10].

Knowing $\mathtt{F}$, equations (5) give rise to 9 equations in the variables $w_1 = f_1^2$ and $w_2 = f_2^2$. These equations are quadratic in w_1 and w_2. We treat w_1 as a hidden variable in the equations (5) to obtain 9 equations of the form $\mathtt{C}(w_1)(1, w_2, w_2^2)^\top = 0$. From the condition that $\mathtt{C}(w_1)$ has rank at most 2, we obtain a polynomial equation, which may be solved to find w_1. We observe that this equation is of degree 3, so there is at least one real solution, and in some cases three. Subsequently, solving for the null-space of $\mathtt{C}(w_1)$ one computes $\mathbf{X} = (1, w_2, w_2^2)$, and hence w_2.

Note that the method just describes involves relaxation, since we ignore the condition that the vector $\mathbf{X}$ has the form $(1, w_2, w_2^2)$. Despite this, the solution found appears always to be valid.

The method used here differs from that described previously in section 3 in that the matrix $\mathtt{C}(w_1)$ is not square, so we can not simply compute $\det \mathtt{C}(w_1)$ to get a polynomial in w_1. A way of handling this situation is described later.

Finally, note that this problem has been considered previously in [17], where a solution was given in terms of the Kruppa equations.

7.2 Varying focal lengths

We may also address the orientation problem for two cameras with different focal lengths f_1 and f_2. In this case, the calibration matrices are $\mathtt{K}_1 = \mathrm{diag}(f_1, f_1, 1)$ and $\mathtt{K}_2 = \mathrm{diag}(f_2, f_2, 1)$.

Starting from a known fundamental matrix $\mathtt{F}$, the equations (5) are quadratic involving monomials $(1, w_1, w_2, w_1 w_2)$ only. Choosing w_1 as the hidden variable, we may write a set of 9 equations of the form $\mathtt{C}(w_1)\mathbf{X} = 0$, where $\mathbf{X} = (1, w_2)$. Since this 9×2 system has rank 1, we obtain a polynomial in w_1 which may be solved to find w_1. As before, computing the null-space of $\mathtt{C}(w_1)$ provides the estimate of w_2. Note that in this case no relaxation is involved, since in computing the solution $\mathbf{X} = (1, w_2)$ one may enforce the condition that the first entry is 1 by scaling the solution.

This problem of focal-length estimation has been studied frequently before, by a variety of methods ([10], [17]). Since this is a problem with an exact solution, most methods work comparably, however the one described here is perhaps conceptually the simplest.

7.3 Non-square systems

In the two problems just considered the matrix $\mathtt{C}(w_1)$ was not square. Nevertheless, we claim that this leads to a polynomial equation in w_1. Indeed, it is easily seen that if $p(w_1)$ is the greatest common divisor of all $(n-1)\times(n-1)$ determinants of $\mathtt{C}$, (where n is the number of columns) then $p(w_1) = 0$ is a necessary and sufficient condition for $\mathtt{C}(w_1)$ to be rank-deficient.

It is not necessary to compute all such sub-determinants. One approach is to compute one or two of them and find the common roots directly. Spurious common roots (if any) can be detected, since they will fail to make $\mathtt{C}(w_1)$ rank-deficient.

A second approach is to compute $\det(\mathtt{C}(w_1)^\top \mathtt{C}(w_1))$ and find its roots. However, this introduces extra irrelevant complex roots. It may be shown however that any real root of $\det(\mathtt{C}(w_1)^\top \mathtt{C}(w_1))$ will make $\mathtt{C}(w_1)$ rank-deficient. Furthermore, any real root will be a double root.

Finally, the matrix $\mathtt{C}(w_1)$ may be reduced directly to row-echelon form using quotient-free Gaussian elimination ([5]) as described above. This method leads directly to the correct polynomial $p(w_1)$ and then the null-space of the matrix is quickly computed from the row-echelon form.

8 Conclusions

The variant of the hidden variable technique that we have described above gives a conceptually very simple and computationally efficient way of solving geometric problems with speed competitive with highly optimized problem-specific methods. To our knowledge, it is still by far one of the fastest implementations publicly available for the 6-point problems. Compared with our original codes provided in [2] and [3] (which dated back four years ago using Matlab's Symbolic-Math toolbox), the new algorithms and their implementations described in this paper are far more efficient—the new versions are both more than 100,000 times faster than the old ones. Since these algorithms are typically used in a RANSAC environment, they need to be *as fast as possible*, even if some accuracy is sacrificed [4]. Pure Matlab or Symbolic-Math (Maple) implementations are in general of limited practical use for this purpose.

In addition, the simple root polishing algorithm we have described allows us to obtain precision in the computed essential matrix close to machine accuracy, at a very minimal cost of less than $0.5\mu s$ per computed matrix. This provides more accurate results than any other we have tested, and extremely rapidly.

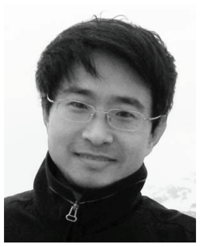
Acknowledgement

The author wish to thank the Editors and Anonymous reviewers for invaluable comments, thank David Nis̀ter and Fredrik Kahl for invaluable discussions related to this work. The work is partially funded by Australian Research Council–ARC Grants. NICTA is funded by the Australian Government and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] D. Nistér, "An efficient solution to the five-point relative pose problem," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.
[2] H. Li and R. Hartley, "Five-point motion estimation made easy," in *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, (Washington, DC, USA), pp. 630–633, 2006.
[3] H. Li, "A simple solution to the six-point two-view focal-length problem," in *ECCV (4)*, pp. 200–213, 2006.
[4] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 756–777, 2004.
[5] E. H. Bareiss, "Sylvester's identity and multistep integer-preserving gaussian elimination," *Mathematics of Computation*, vol. 22, no. 103, pp. 565–578, 1968.
[6] H. Stewénius, C. Engels, and D. Nistér, "Recent developments on direct relative orientation," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60, pp. 284–294, June 2006.
[7] H. Stewénius, F. Kahl, D. Nistér, and F. Schaffalitzky, "A minimal solution for relative pose with unknown focal length," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, (San-Diego, USA), pp. 789–794, June 2005.
[8] H. Stewénius and C. Engels, "An efficient minimal solution for infinitesimal camera motion," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (USA), June 2006.
[9] Z. Kukelova, M. Bujnak, and T. Pajdla, "Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems," in *BMVC08*, 2008.
[10] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision – 2nd Edition*. Cambridge University Press, 2004.
[11] O. D. Faugeras and S. J. Maybank, "Motion from point matches: multiplicity of solutions," in *IEEE Workshop on Motion*, (Irvine, CA.), 1989.
[12] D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*. Springer, 2nd ed., 2005.
[13] D. Nistér, R. Hartley, and H. Stewenius, "Using Galois theory to prove structure from motion algorithms are optimal," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
[14] A. W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, (Los Alamitos, CA, USA), 2001.
[15] G. Cybenko, "The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations," *SIAM Sci Stat*, vol. 1, pp. 303–310, 1980.
[16] U. Helmke, K. Hüper, P. Y. Lee, and J. B. Moore, "Essential matrix estimation using gauss-newton iterations on a manifold," *International Journal of Computer Vision*, vol. 74, no. 2, pp. 117–136, 2007.
[17] R. I. Hartley, "Estimation of relative camera positions for uncalibrated cameras," in *ECCV '92: Proceedings of the Second European Conference on Computer Vision*, (London, UK), pp. 579–587, Springer-Verlag, 1992.

Richard Hartley Richard Hartley is currently with the Computer Vision Group at the Research School of information Sciences and Engineering (RSISE), Australian National University (ANU), Canberra, and also with National ICT Australia (NICTA), a government-funded research institute. He worked in the areas computer-aided electronic design system, described in his book Digit Serial Computation, and Computer Vision in particularly in multi-view geometry. He is the winner of Significant Senior Computer Vision Researcher Award at ICCV 2011. He is the Program Chair for ICCV 2013 (Sydney). He is a Fellow of Australian Science Academy, a Fellow of the IEEE and a Member of the IEEE Computer Society.



Hongdong Li Dr. Hongdong Li is currently a Fellow with the Research School of Engineering (RSISE) at the Australian National University (ANU), Canberra. He was also seconded to the National ICT Australia (NICTA) as a senior research scientist. His current research interests include multiview computer vision, structure from motion, image processing and restoration, computational photography, bionic vision, and discrete optimization in MRFs. He serves constantly as Program Committee Member and Reviewer for ICCV, CVPR, ECCV, TPAMI, TIP, IJCV, CVIU etc. He has been a Member of the IEEE and the IEEE Computer Society since 2001.