# Routines for Relative Pose of Two Calibrated Cameras from 5 Points

**Bill Triggs**
INRIA Rhône-Alpes,
655 avenue de l'Europe, 38330 Montbonnot, France.
*http://www.inrialpes.fr/movi/people/Triggs*
*Bill.Triggs@inrialpes.fr*

July 23, 2000

## 1 Introduction

This report describes a library of C routines for finding the **relative pose** of two calibrated perspective cameras given the images of five unknown 3D points. The relative pose is the translational and rotational displacement between the two camera frames, also called **camera motion** and **relative orientation**.

As images contain no record of the overall spatial scale factor, the scale of the inter-camera translation can not be recovered. So the relative pose problem has 5 degrees of freedom: three angles giving the angular orientation of the second camera in the frame of the first, and two angles giving the direction of the inter-camera translation. According to the usual coplanarity (epipolar) constraint, each pair of corresponding image points gives one constraint on these degrees of freedom, so in principle 5 point pairs are enough for a solution. There are a number of ways to parametrize the problem algebraically and solve the resulting system of polynomial constraints to obtain the solution. The method below uses a quaternion based formulation and multiresultants and eigendecomposition for the solution. There are a great many previous references on this problem, although relatively few give algorithms suitable for numerical implementation. For a sample and further references, see [2, 10, 11, 6, 4, 3, 1, 5, 9, 7].

A well-formulated polynomial system for the 5 point problem generically has 20 possible algebraic solutions [1, 4, 5]. However many of these are often complex, and of the real ones at least half have negative point depths, corresponding to invisible points *behind* the camera. In fact, the 20 solutions fall into 10 "twisted pairs" — solution pairs differing only by an additional rotation of the second camera by 180° about the axis joining the two cameras. The two members of a twisted pair always give opposite *relative* signs for the two depths of any 3D point, so they can not both be physically realizable. Hence, the 5 point relative pose problem has at most 10 real, physically feasible solutions. More often it has between one and five such solutions (*e.g.*, [10]), but in rare cases 10 feasible solutions are indeed possible [1, 4, 3].

The relative pose problem has various singularities [7, 11], both for special point configurations and for special motion types. We will not go detail on these. However, note that for an inter-camera translation of zero, even though the inter-camera rotation can still be found accurately, the corresponding translation direction is undefined and the point-camera distances can not be recovered as there is no baseline for triangulation. In fact, pure camera rotation is a singular case in all formulations of the general 5 point problem of which we are aware, and has to be handled as a special case.

# 2 Code Organization

The main entry point to the library is the driver routine `relorient5()`. This checks for zero translation using `relorient_approx_rot()`, then calls the main multiresultant based solver `relorient5m()`. LAPACK is used for linear algebra, namely LU decomposition, SVD and nonsymmetric eigensystems. Various small utility routines are also needed, *e.g.* for converting quaternions to and from $3\times3$ rotation matrices. For usage of the routines and further information, see the comments at the start of each file.

A test program is provided in `test_relorient5.c`. This can either generate random motions and scenes, run the method on these, and accumulate error statistics, or read a file containing a set of 5 point pairs and print the possible solutions found. A random 5 point data file `points.1` is included. See the comments in this for the format and expected output from it.

Some older MATLAB (or OCTAVE, *etc.*) routines for the basic multiresultant method are also included, but no attempt has been made to make the interface exactly the same.

# 3 Problem Formulation and Method

This section formulates the relative pose problem algebraically, and briefly describes the multiresultant solution method. The approach is essentially the same as that of Emeris in [2], but with many small differences of detail and some additional processing steps.

## 3.1 The Coplanarity Constraints

Given a pure projective camera with internal calibration matrix $\boldsymbol{K}$ [1], the $3 \times 4$ camera projection matrix is $\mathbf{P} = \boldsymbol{K}\,(\boldsymbol{R}\,|\,\boldsymbol{t})$. Here, $\boldsymbol{R} = \boldsymbol{R}(\mathbf{q})$ is a $3\times3$ rotation matrix giving the angular orientation of the camera, $\mathbf{q}$ is the corresponding quaternion, and $\boldsymbol{t}$ encodes the camera position. ($\boldsymbol{t}$ is the position of the world frame origin in this camera's coordinate frame, and $-\boldsymbol{R}\,\boldsymbol{t}$ is the position of the camera centre in the world frame).

The $3\times4$ calibration-corrected perspective projection matrix $(\boldsymbol{R}(\mathbf{q})\,|\,\boldsymbol{t})$ parametrizes the camera's pose. We assume that the cameras are internally calibrated and pre-normalize the measured homogeneous image points by (correcting for lens distortion

[1] $\boldsymbol{K} = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix}$ for a perspective camera with focal length $f$ and principal point $(u_0, v_0)$.

and then) multiplying them by $\boldsymbol{K}^{-1}$, so that the effective perspective matrix becomes $(\boldsymbol{R}\,|\,\boldsymbol{t})$. Such normalized 'points' represent 3D vectors expressed in camera frame coordinates, pointing outwards along the optical ray of the corresponding image point. For convenience, we assume that these 3-vectors have been normalized to unit vectors. So by "image point", we really mean a unit-norm 3D direction vector giving the outwards direction of the corresponding optical ray in the camera frame.

We use 3D coordinates centred on the frame of the first camera, so the two camera matrices are $(\boldsymbol{I}\,|\,\boldsymbol{0})$ and $(\boldsymbol{R}(\mathbf{q})\,|\,\boldsymbol{t})$. We need to recover $\boldsymbol{R}$ or $\mathbf{q}$, and $\boldsymbol{t}$ up to scale. Let $\boldsymbol{x}_i, \boldsymbol{y}_i$, $i = 1 \ldots 5$, be the normalized image points (optical ray direction vectors) in respectively the first and the second cameras. Each image point / direction vector has an associated 3D **depth** (point-camera distance), called $\lambda_i$ for $\boldsymbol{x}_i$ and $\mu_i$ for $\boldsymbol{y}_i$. The corresponding 3D point is just $\lambda_i\,\boldsymbol{x}_i$ in the first camera frame and $\mu_i\,\boldsymbol{y}_i$ in the second, so by elementary 3D geometry:

$$\mu_i\,\boldsymbol{y}_i \;=\; (\boldsymbol{R}\,|\,\boldsymbol{t})\,\left(\begin{smallmatrix}\lambda_i \boldsymbol{x}_i \\ 1\end{smallmatrix}\right) \;=\; \boldsymbol{R}\,(\lambda_i\,\boldsymbol{x}_i) + \boldsymbol{t} \tag{1}$$

Among other things, this is the basis of intersection / triangulation for finding the point depths $\lambda_i, \mu_i$. As the three terms of (1) must be coplanar, we get the well-known **coplanarity** or **epipolar constraint**:

$$\left[\boldsymbol{y}_i, \boldsymbol{R}\,\boldsymbol{x}_i, \boldsymbol{t}\right] \;=\; 0 \tag{2}$$

where $\left[\,\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\,\right]$ is the 3-vector triple product $\boldsymbol{a} \cdot (\boldsymbol{b} \wedge \boldsymbol{c}) = \det(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$. The coplanarity constraint (2) gives one scalar constraint on $(\boldsymbol{R}\,|\,\boldsymbol{t})$ per point pair. However note that the constraint vanishes identically for $\boldsymbol{t} \to \boldsymbol{0}$, so any relative pose method based on it is likely to fail in this case.

We will use a quaternion based form of the coplanarity constraint for our relative pose method. If you are not familiar with quaternion algebra, you will have to take the rest of this section on trust. Quaternions are a way of encoding 3D rotations, algebraically convenient in the sense that only 4 numbers (a 4-vector $\mathbf{q}$ containing a 3-vector $\boldsymbol{q}$ and a scalar $q_0$) with one nonlinear constraint ($\|\mathbf{q}\|^2 = 1$) are needed, rather than the 9 components of a $3 \times 3$ rotation matrix $\boldsymbol{R}$ subject to the 6 quadratic constraints $\boldsymbol{R}\,\boldsymbol{R}^\top = \boldsymbol{I}$. Quaternions have a bilinear product that encodes rotation composition, and a conjugation operation $\overline{\mathbf{q}}$ which reverses the sign of the vector part. Any 3-vector $\boldsymbol{v}$ can be considered to be a quaternion $\mathbf{v}$ with vanishing scalar part $v_0 = 0$. In quaternion notation, the rotated vector $\boldsymbol{R}\,\boldsymbol{x}_i$ is written as the quaternion product $\mathbf{q}\,\boldsymbol{x}_i\,\overline{\mathbf{q}}$ (where juxtaposition denotes quaternion multiplication), and the triple product of three 3-vectors is the scalar part of their quaternion product $\left[\,\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\,\right] = (\boldsymbol{a}\,\boldsymbol{b}\,\boldsymbol{c})_0$.

Putting these elements together, we can write the coplanarity constraint (2) as a bilinear constraint in two quaternion (4-vector) unknowns $\mathbf{q}$ and $\mathbf{p} = \overline{\mathbf{q}}\,\boldsymbol{t}$:

$$0 \;=\; \left[\boldsymbol{y}_i, \boldsymbol{R}\,\boldsymbol{x}_i, \boldsymbol{t}\right] \;=\; (\boldsymbol{y}_i\,\mathbf{q}\,\boldsymbol{x}_i\,\overline{\mathbf{q}}\,\boldsymbol{t})_0 \;=\; (\boldsymbol{y}_i\,\mathbf{q}\,\boldsymbol{x}_i\,\mathbf{p})_0 \;=\; \mathbf{q}^\top\,\mathbf{B}(\boldsymbol{y}_i, \boldsymbol{x}_i)\,\mathbf{p} \tag{3}$$

where the $4 \times 4$ matrix $\mathbf{B}$ turns out to be:

$$\mathbf{B}(\boldsymbol{y}_i, \boldsymbol{x}_i) \;=\; \begin{pmatrix} \boldsymbol{x}_i\,\boldsymbol{y}_i^\top + \boldsymbol{y}_i\,\boldsymbol{x}_i^\top - (\boldsymbol{y}_i \cdot \boldsymbol{x}_i)\,\boldsymbol{I} & \boldsymbol{y}_i \wedge \boldsymbol{x}_i \\ -(\boldsymbol{y}_i \wedge \boldsymbol{x}_i)^\top & -\boldsymbol{y}_i \cdot \boldsymbol{x}_i \end{pmatrix} \tag{4}$$

(Here, '$\wedge$' is cross product, and the scalar component of the quaternions is written last). We get one of these bilinear equations for each point pair. Also, owing to the form of $\mathbf{p} = \overline{\mathbf{q}}\,\boldsymbol{t}$, there is a bilinear consistency constraint between $\mathbf{p}$ and $\mathbf{q}$:

$$p_0\, q_0 - \boldsymbol{p} \cdot \boldsymbol{q} \;=\; (\mathbf{p}\,\mathbf{q})_0 \;=\; (\overline{\mathbf{q}}\,\boldsymbol{t}\,\mathbf{q})_0 \;=\; 0 \tag{5}$$

This gives a total of $5 + 1 = 6$ bilinear equations on the $4 + 4 = 8$ components of $\mathbf{q}, \mathbf{p}$. As $\mathbf{q}, \mathbf{p}$ are defined only up to scale they have just 6 degrees of freedom between them, and the polynomial system turns out to be (generically) well-constrained, with 20 roots.

## 3.2 Sparse Multiresultant Polynomial Solver

### 3.2.1 General Approach

Of the many ways to solve the above 6 polynomial system, we will use a multiresultant approach. We can not describe this in any detail here. See [2] for a description and references, and [8] for a general tutorial on methods of solving polynomial systems using matrices. In our case, the method builds a large ($60 \times 60$) but fairly sparse matrix from the polynomial system using multiresultant techniques, and uses linear algebra to reduce this to a $20 \times 20$ nonsymmetric matrix whose eigenvalues and eigenvectors encode the 20 roots of the system.

To get a general idea of the approach, note that any polynomial is a sum of monomials in its unknowns, multiplied by scalar coefficients. If we choose a set of monomials, we can represent any polynomial on them as a row vector whose entries are the coefficients and whose columns are labelled by the corresponding monomials. This allows us to use linear algebra to manipulate systems of polynomials. In fact, each row and column of the $60 \times 60$ and $20 \times 20$ matrices that we build corresponds to a specific monomial in the unknown variables $\mathbf{q}$ and $\mathbf{p}$. The real art of the method lies in finding suitable sets of row and column monomials, where "suitable" means that the resulting matrices are both nonsingular and relatively small. Everything else follows almost inevitably from this choice. The choice requires some advanced theory in principle, and brute force search in practice.

Suppose that we restrict attention to polynomials on a given monomial set $A$. In linear algebra language, to evaluate the polynomial at a point (set of variable values), we dot-product the polynomial's row vector with a column vector containing the corresponding monomials evaluated at the point. If the point is a root of the polynomial, the dot product (polynomial value) must vanish. So the root's monomial vector is orthogonal to the polynomial's row vector. If we can generate a series of independent polynomials with the same root, these will give a series of linear constraints on the monomial vector. With luck, these will eventually suffice to restrict the monomial vector to a 1D subspace, and hence give it uniquely up to scale. Given this, it is a trivial manipulation to read off the corresponding variable values at the root from the up-to-scale monomial vector. If there are several roots, their monomial vectors all lie in the orthogonal complement of the constraints. As different monomial vectors are linearly independent, we can only hope to constrain the monomial vector to a subspace of dimension equal to at least the number of independent roots, but it turns out that eigendecomposition methods can be used to extract the roots from this residual subspace.

To create a series of independent polynomials with the same root, we work as follows. Given a set $A$ of column-label monomials and an input polynomial $p$, we can form the set of all multiples of $p$ by arbitrary monomials $q$, such that the monomials of the polynomial $q\,p$ are all contained in $A$. This corresponds to forming the set of row vectors $q\,p$ whose nonzero entries lie entirely within the columns labelled by $A$. If $p$ has a root at some point, $q\,p$ must as well, so all of these rows will be orthogonal to the root's monomial vector. If we are interested in the simultaneous roots of a system of several polynomials, we can form the set of admissible row vectors for polynomial separately, and stack them together into a big "multiresultant" matrix to get further constraints on the root monomials.

If the system is generic and has only a single isolated root, it turns out that this construction eventually succeeds in isolating the 1D subspace spanned by the root's monomial vector. All that is needed for this is a sufficiently large (and otherwise suitable) column monomial set $A$. There exist a number of theoretical multiresultant construction methods that give sufficient sets for $A$ under various kinds of genericity assumptions on the input polynomials. We will not go into these, because the details are complicated and in any case they seldom give *minimal* sets for $A$. A practical multiresultant method can usually do better by some kind of combinatorial search over the possible monomial sets $A$, which is exactly how the monomial sets given below were generated.

In our case there are multiple roots so we can not use the above construction as it stands. However, by treating one of the variables in the problem as if it were a constant (*i.e*. part of the coefficients, not the monomials), we can run through the same process to get a multiresultant matrix whose entries are no longer scalars, but rather polynomials in the "hidden" (treated-as-constant) variable. Roots of the system are still null-vectors of this matrix, provided that the hidden variable is given its correct value for the root. So we can find roots by looking for values of the hidden variable for which the multiresultant matrix is singular (has a nontrivial null space, corresponding to the root's monomial vector). If the matrix is actually linear in the hidden variable (which ours is), this requirement already has the form of a so-called generalized eigenproblem for which standard linear algebra methods exist. If not, it can still be converted into an eigenproblem — *e.g*. by using companion matrices — but we will not need this here.

### 3.2.2 Details of the 5 Point Method

In the implementation of the 5 point relative pose method, the multiresultant matrix is constructed by taking the following 10 multiples of each of the $5 + 1 = 6$ input polynomials (3) and (5):

$$[1, q_1, q_2, q_3, q_1^2, q_1\, q_2, q_1\, q_3, q_2^2, q_2\, q_3, q_3^3]$$

These multiples give a $60\times60$ multiresultant matrix with columns labelled respectively by the following three lists of 10, 30 and 20 monomials:

$$\begin{gathered}[p_1\,q_1^3, p_1\,q_1^2\,q_2, p_1\,q_1^2\,q_3, p_1\,q_1\,q_2^2, p_1\,q_1\,q_2\,q_3,\\ p_1\,q_1\,q_3^2, p_1\,q_1^2, p_1\,q_1\,q_2, p_1\,q_1\,q_3, p_1\,q_1]\end{gathered} \tag{6}$$

$$\begin{gathered}[p_1\,q_2^3, p_1\,q_2^2\,q_3, p_1\,q_2\,q_3^2, p_1\,q_3^3, p_2\,q_1^3, p_2\,q_1^2\,q_2, p_2\,q_1^2\,q_3, p_2\,q_1\,q_2^2,\\ p_2\,q_1\,q_2\,q_3, p_2\,q_1\,q_3^2, p_2\,q_2^3, p_2\,q_2^2\,q_3, p_2\,q_2\,q_3^2, p_2\,q_3^3, p_1\,q_2^2,\\ p_1\,q_2\,q_3, p_1\,q_3^2, p_2\,q_1^2, p_2\,q_1\,q_2, p_2\,q_1\,q_3, p_2\,q_2^2, p_2\,q_2\,q_3,\\ p_2\,q_3^2, p_1\,q_2, p_1\,q_3, p_2\,q_1, p_2\,q_2, p_2\,q_3, p_1, p_2]\end{gathered} \tag{7}$$

$$\begin{gathered}[q_1^3, q_1^2\,q_2, q_1^2\,q_3, q_1\,q_2^2, q_1\,q_2\,q_3, q_1\,q_3^2, q_2^3, q_2^2\,q_3, q_2\,q_3^2, q_3^3,\\ q_1^2, q_1\,q_2, q_1\,q_3, q_2^2, q_2\,q_3, q_3^2, q_1, q_2, q_3, 1]\end{gathered} \tag{8}$$

Note that in the above monomials, we have normalized $\mathbf{p}, \mathbf{q}$ to $p_0 = 1, q_0 = 1$. (Equivalently, the above monomials could each be homogenized separately in $p_0$ for $\mathbf{p}$ and $q_0$ for $\mathbf{q}$). The component $p_3$ does not appear above, because it has been treated as a constant and "hidden" in the polynomial coeffients. This means that the coefficients of the $60\times60$ multiresultant matrix are linear polynomials in $p_3$ (because $(3,5)$ are linear in $p_3$), with coefficients given by the above $\mathbf{B}(\boldsymbol{y}_i, \boldsymbol{x}_i)$ matrices (4) for (3), and constant coefficients $\mathbf{B} = \mathrm{diag}(-1,-1,-1,1)$ for (5).

The ordering of the above column monomials was chosen so that: (*i*) the first 10 monomials give a nonsingular leading $10\times10$ submatrix with constant coefficients on the 10 rows from (5); (*ii*) only the last 20 columns contain non-constant entries, *i.e.* nonzero linear terms in $p_3$. These properties are used in three steps as follows.

First, in the implementation, we have already eliminated the first 10 columns using the constant $10\times10$ submatrix from the (5) rows. This reduces the problem to a $50\times50$ one involving only coplanarity equations (3), which decreases the matrix decomposition work required for this stage by about 40% without any significant increase in complexity.

Second, we build the reduced $50\times50$ multiresultant matrix $\mathbf{M}$ from the $\mathbf{B}$ matrices, as a $50\times50$ constant matrix $\mathbf{M}_0$ and a $50\times20$ one $\mathbf{M}_1$ with

$$\mathbf{M} \;=\; \mathbf{M}_0 + p_3\,(\mathbf{0}_{50\times30}\,|\,\mathbf{M}_1)$$

This is already in the form of a generalized eigensystem in $p_3$:

$$(\mathbf{M}_0 + p_3\,(\mathbf{0}_{50\times30}\,|\,\mathbf{M}_1))\;\mathbf{x} \;=\; \mathbf{0}$$

so it could be solved directly using, *e.g.* LAPACK's `dgegv()` or `dggev()`. However, many of the columns do not involve $p_3$ so there would be many unwanted roots at infinity. Instead, we extract the $20\times20$ submatrix $\mathbf{A}$ containing the last 20 rows of $\mathbf{M}_0^{-1}\,\mathbf{M}_1$, and solve the standard nonsymmetric eigensystem[2] $(\mathbf{A} + \lambda\,\mathbf{I})\;\mathbf{x} = \mathbf{0}$, where $\lambda = 1/p_3$.

[2]Strictly speaking, this is not the best numerical approach. It would be stabler to use pivoted LU decom-

Finally, the eigenvalues and eigenvectors of the $20 \times 20$ matrix give the 20 possible roots. The eigenvectors $\mathbf{x}$ are monomial vectors in the 20 monomials of (8), from which $\mathbf{q}$ and hence $\boldsymbol{R}(\mathbf{q})$ can be recovered. $\boldsymbol{t}$ is then recovered by linear least squares from the coplanarity equations (2), and the point depths are found by triangulation using (1).

### 3.3 The Zero Rotation Singularity

One further trick is included in the implementation. The above formulation happens to have an inessential singularity when the inter-camera rotation is zero, which is annoying because rotations near zero are common in practice. To get around this, we perturb the input data with a random rotation matrix, solve, then undo the effects of the random rotation on the solution. This does not solve the problem, but it moves it randomly around in the solution space so that occasional failures occur for all rotation values, rather than guaranteed failures for some particular values, and none elsewhere. This is usually preferable, especially if the method will be used in a RANSAC style loop, as it spreads and randomizes the unreliability.

Another more systematic approach would be to have several multiresultant routines with different monomial sets and hidden variables, and hence different singularities. Or equivalently and more simply, to call the same routine several times for each point set, each time perturbing the points with a different input rotation and undoing the effects of this afterwards. The problem would then reduce to the choice of a small set of perturbing rotations that suffices to avoid all of the observed inessential singularities of the method. We have not done this, as the randomized method seems to perform sufficiently well.

### 3.4 The Zero Translation Singularity

For inter-camera translations near zero (*i.e.* much smaller than the point-camera distances), the coplanarity constraints (2, 3) degenerate and the above multiresultant method becomes ill-conditioned. As a partial work-around for this, the driver `relorient5()` also runs a method designed for the zero-translation case. This simply assumes that the translation is exactly zero and finds the $3 \times 3$ rotation that best aligns the input points $\boldsymbol{y}_i \approx \boldsymbol{R}\boldsymbol{x}_i$. The point-camera distances can not be recovered in this case as there is no baseline for triangulation, so they are set arbitrarily to 1.

## 4 Performance

The method seems to work tolerably well in practice, given the almost universal ill-conditioning of minimal-data pose problems.

position to reduce the system to a $20 \times 20$ *generalized* eigensystem:

$$\big(\{\mathbf{L}^{-1}\mathbf{M}_0\} + p_3\,\{\mathbf{L}^{-1}(\mathbf{0}\,|\,\mathbf{M}_1)\}\big)\;\mathbf{x} = \mathbf{0}$$

where $\{-\}$ means "take the final $20 \times 20$ submatrix". This would allow $\mathbf{L}^{-1}\mathbf{M}_0 = \mathbf{U}$ to be ill-conditioned or even singular without causing the method to fail. However, we have kept the above approach for now as it is simple and it seems to work well in practice.

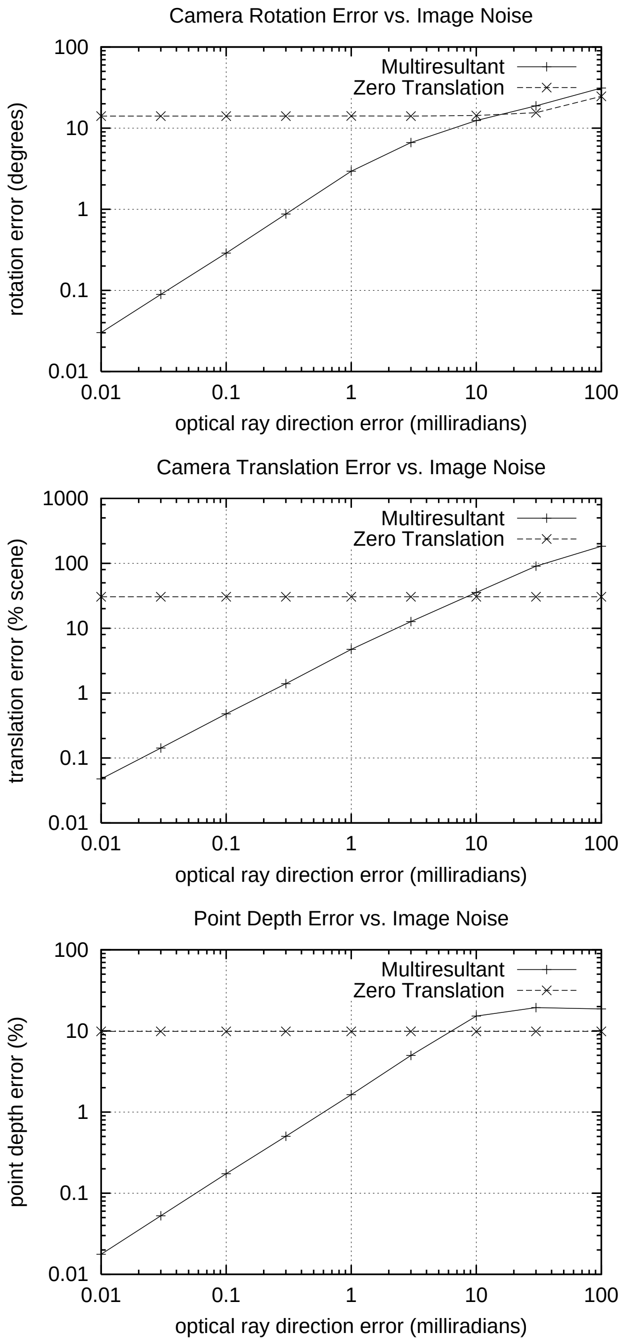



Figure 1: The rotation, translation and point depth error as a function of image noise for the 5 point multiresultant and zero translation methods.

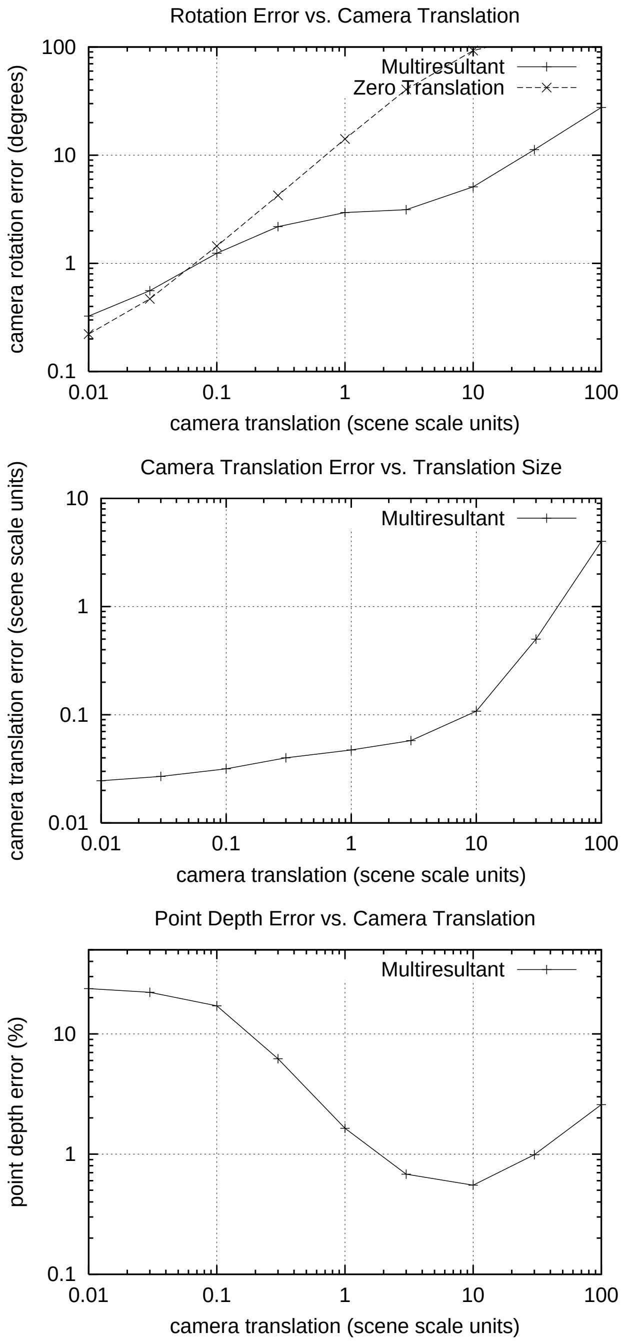



Figure 2: The rotation, translation and point depth error as a function of camera translation distance for the 5 point multiresultant and zero translation methods.

The experiments shown below use the following default values. The scene contains 5 randomly distributed 3D points drawn from a unit standard deviation Gaussian distribution 4 units from the first camera. The inter-camera translation is a random Gaussian vector with standard deviation 1 unit. The inter-camera rotation is random Gaussian with standard deviation $20^\circ$. The image (optical ray direction vector) noise is Gaussian with standard deviation $10^{-3}$ radians. Each experiment is run for 1000 trials and the median error values are reported. The spatial distance unit used in the translation error graphs is the "scene scale", by which we mean the smallest point-camera distance. This is preferable to the common policy of setting $\|\boldsymbol{t}\|$ to 1, as the latter muddies the concept of "small translations".

As the multiresultant method usually gives multiple solutions, we need to select a single "best" one to report. We choose the solution with minimum rotation error against the ground truth, and report its rotation, translation and depth errors. This is biased towards rotation errors in the sense that even if another solution has smaller translation error (which does happen), it is not selected.

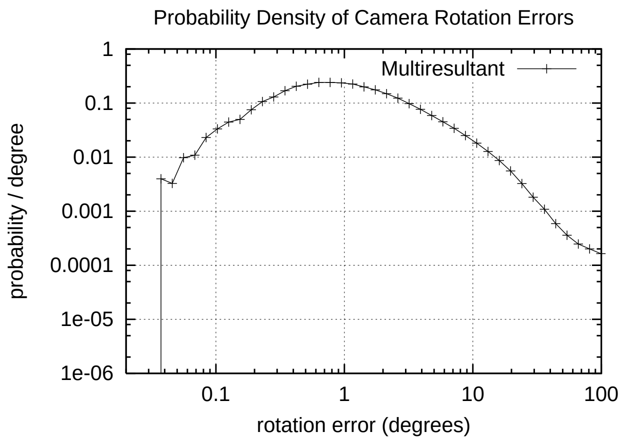
Figure 1 shows the median camera rotation and translation and point depth errors as a function of image noise. The multiresultant method seems to be usable up to a noise of around 10 milliradians on the measured optical ray directions, which would translate to an image noise of 10 pixels or more for most close-range photogrammetric cameras. The zero translation method gives a strongly biased solution here because the camera translations are non-negligible, but for zero translation its error also scales linearly with noise.

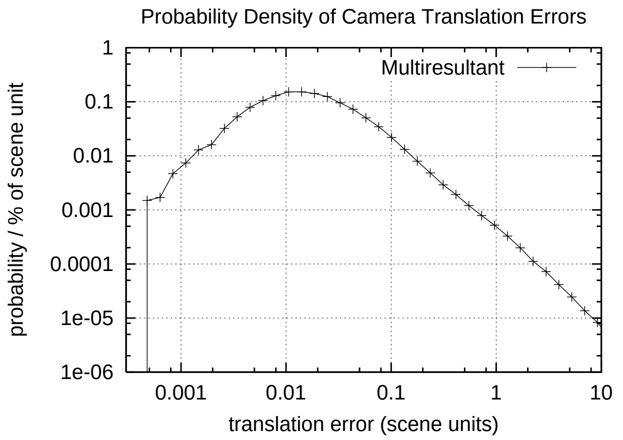
Figure 2 shows how the errors scale with camera translation distance, from $10^{-2}$ up to $10^{2}$ scene units. (In the latter case, the second camera moves far away from the points, so an accurate solution can not be expected and all of the errors increase). The zero translation method makes a moderate improvement over the multiresultant one for small translations. The improvement is not large in these particular experiments, but it does become larger as the noise level and/or the translation distance decrease[3]. As would be expected: (*i*) the zero translation method gives very biased results at large camera translations; and (*ii*) the point-camera distances can not be recovered accurately for small baselines.

Figure 3 gives probability densities (*i.e.* relative frequencies) versus error size, for the rotation, translation and point depth errors in trials with the standard noise, rotation and translation distributions given above. The error distributions stretch to quite large values, which after some threshold would have to be declared as cases where the method failed. However, note that the random problem generator makes no attempt to detect cases where the generated data is intrinsically ill-conditioned, *e.g.* because the five random 3D points happen to be tightly clustered or nearly aligned. It is likely that at least some of the observed large error values are due to such ill-conditioned 5-tuples. Many of these should already be recognizably ill-conditioned in the images, and hence could potentially be avoided in a real relative pose problem.

Figure 4 shows that neither method is particularly sensitive to coplanarity of the input points. Here, the random spherical scene is compressed in the $z$-direction by an

[3] The multiresultant method fails for $\boldsymbol{t} \to \boldsymbol{0}$ in the noiseless case, but even a small amount of noise seems to perturb it enough to avoid this singularity, although still with some loss of accuracy.



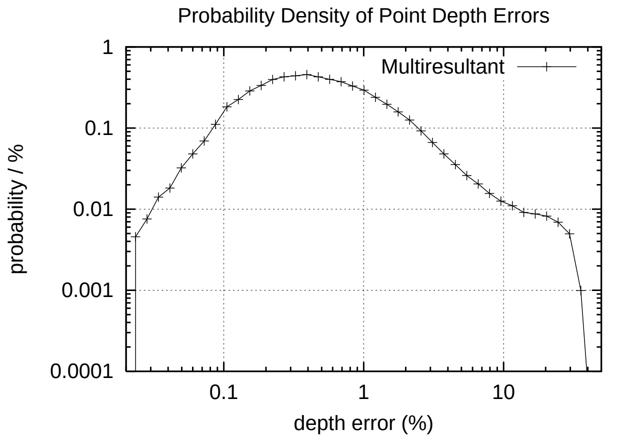




Figure 3: Probability densities for the various sizes of rotation, translation and point depth errors.

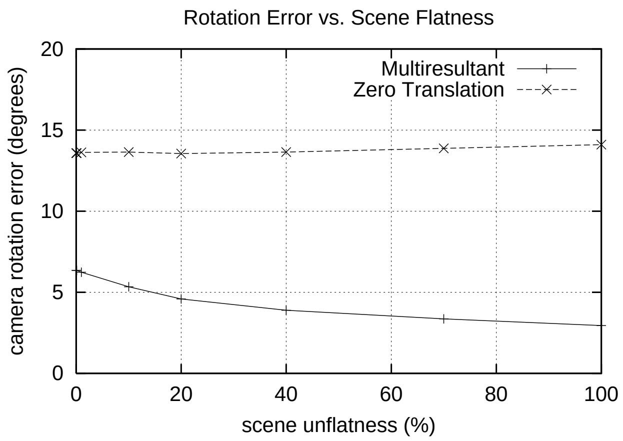



Figure 4: The rotation error as a function of scene flatness.

unflatness factor between 0 and 1 before running the method. The moderate decrease in the multiresultant method's error as the unflatness increases is probably due at least in part to the increase in the scatter of the 3D points, which provides greater geometric strength.

The run time of the method is about 5 milliseconds per call to the driver `relorient5()` on my 440 MHz Sun UltraSparc 10, for optimized code using the (free but slow) BLAS distributed with LAPACK. This is acceptable for one-off use, but slower than one would like for a routine that might be called many times in a RANSAC loop. Most of the time is spent in LU and eigendecomposition of the 5 point multiresultant matrix. This could not be reduced substantially within the current approach unless a more compact multiresultant matrix could be found.

## References


[1] M. Demazure. Sur deux problèmes de reconstruction. Technical report, INRIA, 1988.

[2] I. Z. Emeris. A general solver based on sparse resultants: Numerical issues and kinematic applications. Technical Report RR-3110 (http://www.inria.fr/RRRT/RR-3110.html), INRIA, Sophia Antipolis, France, January 1997.

[3] O. Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT Press, 1993.

[4] O. D. Faugeras and S. J. Maybank. Motion from point matches: Multiplicity of solutions. In *IEEE Workshop on Computer Vision*, 1989.

[5] A. Heyden and G. Sparr. Reconstruction from calibrated cameras - a new proof of the kruppa-demazure theorem. *J. Mathematical Imaging & Vision*, 10:1–20, 1999.

[6] B. Horn. Relative orientation. *Int. J. Computer Vision*, 4:59–78, 1990.

[7] J. Krames. Zur Ermittlung eines Objektes aus zwei Perspektiven (Ein Beitrag zur Theorie der "gefährlichen Örter"). *Monatshefte für Mathematik und Physik*, 49:327–354, 1941.

[8] B. Mourrain. An introduction to linear algebra methods for solving polynomial equations. In *HERMCA'98*, 1998. See also: http://www-sop.inria.fr/saga/logiciels/multires.html.

[9] A. N. Netravali, T. S. Huang, A. S. Krishnakumar, and R. J. Holt. Algebraic methods in 3D motion estimation from two-view point correspondences. *Int. J. Imaging Systems & Technology*, 1:78–99, 1989.

[10] J. Philip. A non-iterative algorithm for determining all essential matrices corresponding to five point pairs. *Photogrammetric Record*, 15(88):589–599, October 1996.

[11] B. P. Wrobel. Minimum solutions for orientation. In *Workshop on Calibration and Orientation of Cameras in Computer Vision*, Washington D.C., August 1992. Proceedings still unpublished, paper available from author.