International Journal of Computer Vision, 15, 123–141 (1995)
© 1995 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Model-Based Object Pose in 25 Lines of Code

DANIEL F. DEMENTHON AND LARRY S. DAVIS
Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD 20742

Received October 1992; Revised March 1994; Accepted May 1994

Abstract. In this paper, we describe a method for finding the pose of an object from a single image. We assume that we can detect and match in the image four or more noncoplanar feature points of the object, and that we know their relative geometry on the object. The method combines two algorithms; the first algorithm, *POS* (Pose from Orthography and Scaling) approximates the perspective projection with a scaled orthographic projection and finds the rotation matrix and the translation vector of the object by solving a linear system; the second algorithm, *POSIT* (POS with ITerations), uses in its iteration loop the approximate pose found by POS in order to compute better scaled orthographic projections of the feature points, then applies POS to these projections instead of the original image projections. POSIT converges to accurate pose measurements in a few iterations. POSIT can be used with many feature points at once for added insensitivity to measurement errors and image noise. Compared to classic approaches making use of Newton's method, POSIT does not require starting from an initial guess, and computes the pose using an order of magnitude fewer floating point operations; it may therefore be a useful alternative for real-time operation. When speed is not an issue, POSIT can be written in 25 lines or less in Mathematica; the code is provided in an Appendix.

1 Introduction

Computation of the position and orientation of an object (*object pose*) using images of feature points when the geometric configuration of the features on the object is known (a *model*) has important applications, such as calibration, cartography, tracking and object recognition. Fischler and Bolles (1981) have coined the term *Perspective-n-Point problem* (or P*n*P problem) for this type of problem with *n* feature points.

Researchers have formulated closed form solutions when a few feature points are considered in coplanar and noncoplanar configurations (Abidi and Chandra 1991; DeMenthon and Davis 1992; Egli, Miller and Setterholm 1987; Fischler and Bolles 1981; Horaud, Conio and Leboulleux 1989; Roberts 1965). However, pose computations which make use of numbers of feature points larger than can be dealt with in closed form solutions may be more robust because the measurement errors and image noise average out between the feature points and because the pose information content becomes highly redundant. The most straightforward method, first described by Roberts (1965), consists of finding the elements of the *perspective projection matrix* (Roberts 1965; Sutherland 1974; Faugeras 1993)—which expresses the mapping between the feature points on the object and their image projections in homogeneous coordinates—as solutions of a linear system. The 11 unknown elements in this matrix can be found if at least six matchings between image points and object points are known. Also notable among pose computations are the methods proposed by Tsai (1987), Lowe (1985, 1991), and Yuan (1989) (these papers also provide good critical reviews of photogrammetric calibration techniques). The methods proposed by Tsai are especially useful when the focal length of the camera, the lens distortion and the image center are not known. When these parameters have already been calibrated, the techniques proposed by Lowe and by Yuan may be sufficient. However, both techniques rely on the Newton-Raphson method, which presents two significant drawbacks: first, an approximate pose must be provided to initiate the iteration process; second, at each iteration step, the pseudoinverse matrix of a Jacobian of dimensions $2N \times 6$ (Lowe; N is the number of feature points) or $N \times 6$ (Yuan) must be found, a computationally expensive operation.

The method described in this paper relies on linear algebra techniques and is iterative like the methods of Lowe and Yuan, but it does not require an initial pose estimate and does not require matrix inversions in its iteration loop. At the first iteration step, the method finds an approximate pose by multiplying an *object matrix* (which depends only on the distribution of feature points on the object and is precomputed) by two vectors (which depend only on the coordinates of the images of these feature points). The two resulting vectors, once normalized, constitute the first two rows of the rotation matrix, and the norms of these vectors are equal to the scaling factor of the projection, which provides the translation vector. We show that these operations amount to assuming that the involved image points have been obtained by a scaled orthographic projection (SOP in the following). We refer to this part of the algorithm as "POS" (Pose from Orthography and Scaling). The works of Tomasi (1991) and Ullman and Basri (1991) apply related techniques (see also (Huttenlocher and Ullman 1988) for related work with three points).

The next iterations apply exactly the same calculations, but with "corrected" image points. The basic idea is that since the POS algorithm requires an SOP image instead of a perspective image to produce an accurate pose, we have to compute SOP image points, using the pose found at the previous iteration. The process consists in shifting the feature points of the object in the pose just found, to the lines of sight (where they would belong if the pose was correct), and obtain a scaled orthographic projection of these shifted points. We call this iterative algorithm "POSIT" (POS with ITerations). Four or five iterations are typically required to converge to an accurate pose.

The POSIT algorithm requires an order of magnitude fewer computations than the techniques mentioned above: For N matchings between object points and image points, POSIT requires around $24N$ arithmetic operations and two square root calculations per iteration. For 8 feature points and four iteration steps, around 800 arithmetic operations and 8 square roots are needed. As a comparison, Roberts' method would solve a linear system of $2N$ equations using a pseudoinverse matrix, which requires about $1012N + 2660$ arithmetic operations (adds, multiplies and divides); for $N = 8$, the total count is at least 10 times more than for POSIT. With Lowe's method, for each iteration, around $202N + 550$ operations are required; for 8 feature points and four iteration steps, the total count of arithmetic operations is around 10 times more than POSIT. Yuan's method seems to be the most expensive, with around $12N^3 + 21N^2$ arithmetic operations to set up the iteration loop, then $36N^2 + 108N + 460$ operations at each iteration; for eight feature points and four iteration steps, the total count of operations is around 25 times more than for POSIT.

Based on these comparisons, we believe that the proposed method has definite advantages over previous approaches for real-time applications.

In later sections of this paper, we test the accuracy and stability of the method by considering a large set of simulated situations with increasing amounts of random image perturbation (Haralick 1992). In all these situations, the algorithm appears to remain stable and to degrade gracefully as image noise is increased.

2 Notations

In Fig. 1, we show the classic pinhole camera model, with its center of projection O, its image plane G at a distance f (the focal length) from O, its axes Ox and Oy pointing along the rows and columns of the camera sensor, and its third axis Oz pointing along the optical axis. The unit vectors for these three axes are called **i**, **j** and **k** (vectors are written in bold characters).

An object with feature points M_0, $M_1, \ldots, M_i$, $\ldots, M_n$ is positioned in the field of view of the camera. The coordinate frame of reference for the object is centered at M_0 and is (M_0u, M_0v, M_0w). We call M_0 the reference point for the object. Only the object

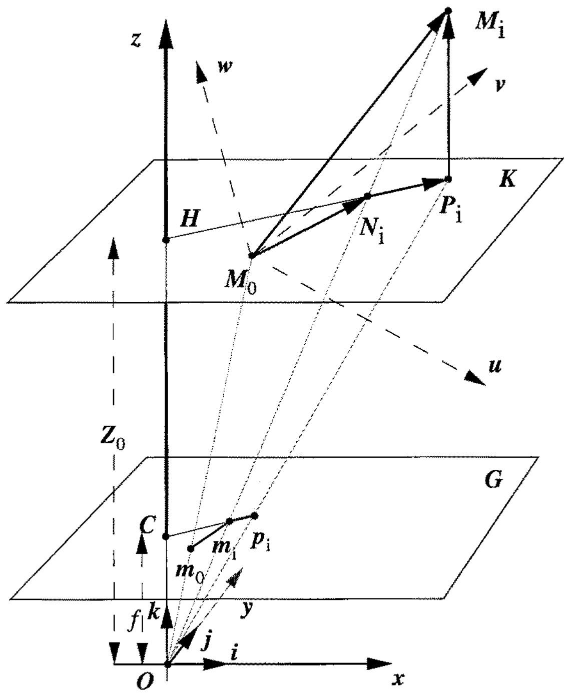


Fig. 1. Perspective projection (m_i) and scaled orthographic projection (p_i) for an object point M_i and a reference point M_0.

points M_0 and M_i are shown in Fig. 1. The shape of the object is assumed to be known; therefore the coordinates (U_i, V_i, W_i) of the point M_i in the object coordinate frame of reference are known. The images of the points M_i are called m_i, and their image coordinates (x_i, y_i) are known. The coordinates (X_i, Y_i, Z_i) of the same points M_i in the camera coordinate system are unknown, because the pose of the object in the camera coordinate system is unknown. We next show how to find the rotation matrix and translation vector of the object directly, without solving explicitly for the coordinates (X_i, Y_i, Z_i).

3 Problem Definition

Our goal is to compute the rotation matrix and translation vector of the object. The rotation matrix $\mathbf{R}$ for the object is the matrix whose rows are the coordinates of the unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ of the camera coordinate system expressed in the object coordinate system (M_0u, M_0v, M_0w). Indeed, the purpose of the rotation matrix is to transform the object coordinates of vectors such as $\mathbf{M_0M}_i$ into coordinates defined in the camera system; the dot product $\mathbf{M_0M}_i \cdot \mathbf{i}$ between the first row of the matrix and the vector $\mathbf{M_0M}_i$ correctly provides the projection of this vector on the unit vector $\mathbf{i}$ of the camera coordinate system, i.e. the coordinate $X_i - X_0$ of $\mathbf{M_0M}_i$, as long as the coordinates of $\mathbf{M_0M}_i$ and of the row vector $\mathbf{i}$ are expressed in the same coordinate system, here the coordinate system of the object.

The rotation matrix can therefore be written as

$$\mathbf{R} = \begin{bmatrix} i_u & i_v & i_w \\ j_u & j_v & j_w \\ k_u & k_v & k_w \end{bmatrix}$$

where i_u, i_v, i_w are the coordinates of $\mathbf{i}$ in the coordinate system (M_0u, M_0v, M_0w) of the object.

To compute the rotation, we only need to compute $\mathbf{i}$ and $\mathbf{j}$ in the object coordinate system. The vector $\mathbf{k}$ is then obtained by the cross-product $\mathbf{i} \times \mathbf{j}$. The translation vector, $\mathbf{T}$ is the vector $\mathbf{OM}_0$ between the center of projection, O, and the reference point M_0, the origin of the object coordinate frame of reference. Therefore the coordinates of the translation vector are X_0, Y_0, Z_0. If this point M_0 has been chosen to be a visible feature point for which the image is a point m_0, this translation vector $\mathbf{T}$ is aligned with vector $\mathbf{Om}_0$ and is equal to $\frac{Z_0}{f}\mathbf{Om}_0$. Therefore to compute the object translation, we only need to compute its z-coordinate Z_0. Thus the object pose is fully defined once we find $\mathbf{i}, \mathbf{j}$ and Z_0.

4 Scaled Orthographic Projection and Perspective Projection

4.1 Analytical Definition

Scaled orthographic projection (*SOP*) is an approximation to "true" perspective projection. In this approximation, for a given object in front of the camera, one assumes that the depths Z_i of different points M_i of the object with camera coordinates (X_i, Y_i, Z_i) are not very different from one another, and can all be set to the depth of one of the points of the object, for example the depth Z_0 of the reference point M_0 of the object (see Fig. 1). In SOP, the image of a point M_i of an object is a point p_i of the image plane G which has coordinates

$$x'_i = fX_i/Z_0, \quad y'_i = fY_i/Z_0,$$

while for perspective projection an image point m_i would be obtained instead of p_i, with coordinates

$$x_i = fX_i/Z_i, \quad y_i = fY_i/Z_i$$

The ratio $s = f/Z_0$ is the *scaling factor* of the SOP. The reference point M_0 has the same image m_0 with coordinates x_0 and y_0 in SOP and perspective projection. The image coordinates of the SOP projection p_i can also be written as

$$\begin{aligned} x'_i &= fX_0/Z_0 + f(X_i - X_0)/Z_0 \\ &= x_0 + s(X_i - X_0) \\ y'_i &= y_0 + s(Y_i - Y_0) \end{aligned} \qquad (1)$$

4.2 Geometric Construction of SOP

The geometric construction for obtaining the perspective image point m_i of M_i and the SOP image point p_i of M_i is shown in Fig. 1. Classically, the perspective image point m_i is the intersection of the line of sight of M_i with the image plane G. In SOP, we draw a plane K through M_0 parallel to the image plane G. This plane is at a distance Z_0 from the center of projection O. The point M_i is projected on K at P_i by an orthographic projection. Then P_i is projected on the image plane G at p_i by a perspective projection. The vector $\mathbf{m_0p}_i$ is parallel to $\mathbf{M_0P}_i$ and is scaled down from $\mathbf{M_0P}_i$ by the scaling factor s equal to f/Z_0. Equation 1 simply expresses the proportionality between these two vectors.

5 Fundamental Equations for Perspective

We now consider equations that characterize "true" perspective projection and relate the unknown row vectors **i** and **j** of the rotation matrix and the unknown Z_0 coordinate of the translation vector to the known coordinates of the vectors $\mathbf{M_0M_i}$ in the object coordinate system, and to the known coordinates x_i and y_i of the image points m_0 and m_i. Solving these equations for these unknowns would provide all the information required to define the object pose, as we have seen in Section 3. These equations are

$$\mathbf{M}_0\mathbf{M}_i \cdot \frac{f}{Z_0}\mathbf{i} = x_i(1+\varepsilon_i) - x_0, \tag{2}$$

$$\mathbf{M}_0\mathbf{M}_i \cdot \frac{f}{Z_0}\mathbf{j} = y_i(1+\varepsilon_i) - y_0 \tag{3}$$

in which the terms ε_i are defined as

$$\varepsilon_i = \frac{1}{Z_0}\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k} \tag{4}$$

and **k** is defined as $\mathbf{k} = \mathbf{i} \times \mathbf{j}$

PROOF. In Fig. 1, consider points M_0, M_i of the object, and the plane K parallel to the image plane through M_0. The line of sight for M_i intersects plane K in N_i, and M_i projects onto plane K in P_i. The vector $\mathbf{M}_0\mathbf{M}_i$ is the sum of three vectors

$$\mathbf{M}_0\mathbf{M}_i = \mathbf{M}_0\mathbf{N}_i + \mathbf{N}_i\mathbf{P}_i + \mathbf{P}_i\mathbf{M}_i \tag{5}$$

The vector $\mathbf{M}_0\mathbf{N}_i$ and its image $\mathbf{m}_0\mathbf{m}_i$ are proportional in the ratio Z_0/f. The two vectors $\mathbf{N}_i\mathbf{P}_i$ and $\mathbf{Cm}_i$ are also proportional in the two similar triangles Cm_iO and $N_iP_iM_i$, in a ratio equal to the ratio of the z coordinates of the other two corresponding vectors of these triangles, $\mathbf{P}_i\mathbf{M}_i$ and $\mathbf{OC}$. This ratio is $\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}/f$. The sum of the three vectors can then be expressed as

$$\mathbf{M}_0\mathbf{M}_i = \frac{Z_0}{f}\mathbf{m}_0\mathbf{m}_i + \frac{\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}}{f}\mathbf{Cm}_i + \mathbf{P}_i\mathbf{M}_i \tag{6}$$

We take the dot product of this expression with the unit vector **i** of the camera coordinate system. The dot product $\mathbf{P}_i\mathbf{M}_i \cdot \mathbf{i}$ is zero; the dot product $\mathbf{m}_0\mathbf{m}_i \cdot \mathbf{i}$ is the x-coordinate, $x_i - x_0$, of the vector $\mathbf{m}_0\mathbf{m}_i$; the dot product $\mathbf{Cm}_i \cdot \mathbf{i}$ is the coordinate x_i of $\mathbf{Cm}_i$. With the definition $\varepsilon_i = \frac{1}{Z_0}\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}$, one obtains Eq. 2. Similarly, one obtains Eq. 3 by taking the dot product of expression 6 with unit vector **j**. □

6 Fundamental Equations and POS

We now show that in the right hand sides of the fundamental equations, the terms $x_i(1+\varepsilon_i)$ and $y_i(1+\varepsilon_i)$, are in fact the coordinates x_i' and y_i' of the points p_i, which are the scaled orthographic projections of the feature points M_i (Fig. 1):

Consider the points M_0, M_i, the projection P_i of M_i on the plane K, and its image p_i. We call the coordinates of p_i in the image plane x_i' and y_i'. The vector $\mathbf{M}_0\mathbf{M}_i$ is the sum of two vectors $\mathbf{M}_0\mathbf{P}_i$ and $\mathbf{P}_i\mathbf{M}_i$. The first vector, $\mathbf{M}_0\mathbf{P}_i$, and its image $\mathbf{m}_0\mathbf{p}_i$ are proportional in the ratio Z_0/f. Consequently,

$$\mathbf{M}_0\mathbf{M}_i = \frac{Z_0}{f}\mathbf{m}_0\mathbf{p}_i + \mathbf{P}_i\mathbf{M}_i$$

We take the dot product of this vector with unit vector **i** of the camera coordinate system; the dot product $\mathbf{P}_i\mathbf{M}_i \cdot \mathbf{i}$ is zero, and the dot product $\mathbf{m}_0\mathbf{p}_i \cdot \mathbf{i}$ is the x coordinate, $x_i' - x_0$, of the vector $\mathbf{m}_0\mathbf{p}_i$. We obtain

$$\mathbf{M}_0\mathbf{M}_i \cdot \frac{f}{Z_0}\mathbf{i} = x_i' - x_0 \tag{7}$$

and similarly

$$\mathbf{M}_0\mathbf{M}_i \cdot \frac{f}{Z_0}\mathbf{j} = y_i' - x_0, \tag{8}$$

Comparing these equations with Eqs. 2 and 3, one sees that the coordinates of p_i can be written $x_i' = x_i(1+\varepsilon_i)$ and $y_i' = y_i(1+\varepsilon_i)$.

7 POS and POSIT

The Eqs. 2 and 3 can also be written

$$\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{I} = x_i(1+\varepsilon_i) - x_0, \tag{9}$$

$$\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{J} = y_i(1+\varepsilon_i) - y_0, \tag{10}$$

with

$$\mathbf{I} = \frac{f}{Z_0}\mathbf{i}, \quad \mathbf{J} = \frac{f}{Z_0}\mathbf{j}$$

The basic idea behind the proposed method is that if values are given to ε_i, Eqs. 9 and 10 provide linear systems of equations in which the only unknowns are respectively the coordinates of **I** and **J**. Once **I** and **J** have been computed, **i** and **j** are found by normalizing **I** and **J**, and Z_0 is obtained from the norm of either **I** or **J**. We call this algorithm, which finds the pose by solving linear systems, *POS* (Pose from Orthography and Scaling). Indeed, finding the pose of the object by using fixed values of ε_i in Eqs. 2 and 3 amounts to

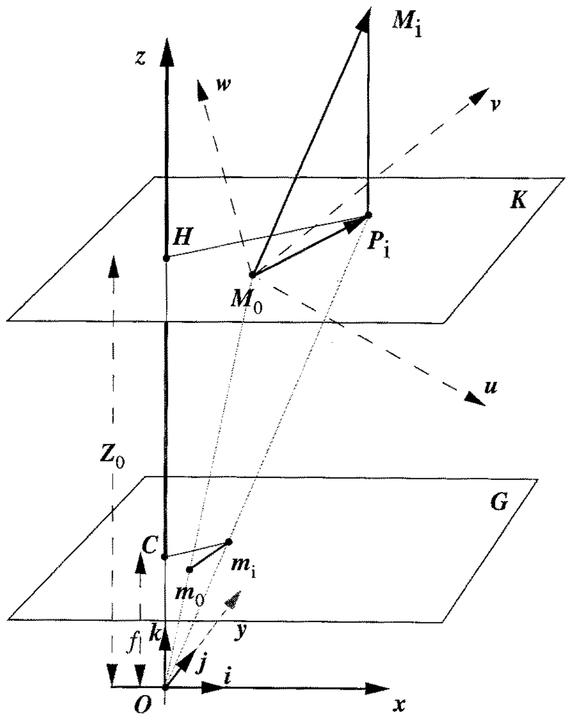


Fig. 2. The initial loop in POSIT looks for a pose of the object such that the points m_i are the scaled orthographic projections of the points M_i of the object.

finding the pose for which the points M_i have as scaled orthographic projections the image points p_i with coordinates $x_i(1+\varepsilon_i)$ and $y_i(1+\varepsilon_i)$, as we have seen in the previous section.

The solutions of the POS algorithm are only approximations if the values given to ε_i are not exact. But once the unknowns **i** and **j** have been computed, *more exact values can be computed for the* ε_i, and the equations can be solved again with these better values. We call this iterative algorithm *POSIT* (POS with Iterations). This algorithm generally makes the values of **i**, **j** and Z_0 converge toward values which correspond to a correct pose in a few iterations.

Initially, we can set $\varepsilon_i = 0$. Assuming ε_i null implies $x'_i = x_i$, $y'_i = y_i$ and amounts to assuming that p_i and m_i coincide. Figure 2 describes this configuration. Note from the definition of the terms ε_i (Eq. 4) that they are the z-coordinates of vectors between two object points, divided by the distance of the object to the camera; these ratios are small when the ratio of object size to distance is small, so that the pose found at the very first iteration may be acceptable in this case. When tracking an object, the initial values for the terms ε_i are preferably chosen equal to the values obtained at the last iteration of the pose computation for the previous image.

8 Solving the Systems of Equations (POS algorithm)

Within the iterative algorithm described in the previous section, we have to solve Eqs. 9 and 10. We rewrite these equations in a somewhat more compact form

$$\mathbf{M_0M}_i \cdot \mathbf{I} = \xi_i,$$
$$\mathbf{M_0M}_i \cdot \mathbf{J} = \eta_i,$$

with

$$\mathbf{I} = \frac{f}{Z_0}\mathbf{i}, \quad \mathbf{J} = \frac{f}{Z_0}\mathbf{j}, \quad \xi_i = x_i(1+\varepsilon_i) - x_0,$$
$$\eta_i = y_i(1+\varepsilon_i) - y_0,$$

and where the terms ε_i have known values computed at the previous iteration steps of the algorithm. We express the dot products of these equations in terms of vector coordinates *in the object coordinate frame of reference*:

$$[U_i \quad V_i \quad W_i][I_u \quad I_v \quad I_w]^T = \xi_i,$$
$$[U_i \quad V_i \quad W_i][J_u \quad J_v \quad J_w]^T = \eta_i,$$

where the T exponent expresses the fact that we consider a transposed matrix, here a column vector. These are linear equations where the unknowns are the coordinates of vector **I** and vector **J**. The other parameters are known: x_i, y_i, x_0, y_0 are the known coordinates of m_i and m_0 (images of M_i and M_0) in the camera coordinate system, and U_i, V_i, W_i are the known coordinates of the point M_i in the object coordinate frame of reference.

Writing Eq. 9 for the n object points M_1, M_2, $M_i, \ldots, M_n$ and their images, we generate linear systems for the coordinates of the unknown vectors **I** and **J**:

$$\mathbf{A\,I} = \mathbf{x}', \quad \mathbf{A\,J} = \mathbf{y}' \tag{11}$$

where **A** is the matrix of the coordinates of the object points M_i in the object coordinate frame of reference, $\mathbf{x}'$ is the vector with i-th coordinate ξ_i and $\mathbf{y}'$ is the vector with i-th coordinate η_i. Generally, if we have at least three visible points other than M_0, and all these points are *noncoplanar*, matrix **A** has rank 3, and the solutions of the linear systems in the least square sense are given by

$$\mathbf{I} = \mathbf{B\,x}', \quad \mathbf{J} = \mathbf{B\,y}' \tag{12}$$

where **B** is the *pseudoinverse* of the matrix **A**.

We call **B** the *object matrix*. Knowing the geometric distribution of feature points M_i, we can precompute this pseudoinverse matrix **B**, either by the operation $[\mathbf{A}^T\ \mathbf{A}]^{-1}\mathbf{A}^T$, or by decomposing matrix **A** by Singular Value Decomposition (SVD) (Press et al. 1988). One can find in (Press et al. 1988) a discussion showing that the solutions computed by Eq. 12 for the systems of Eq. 11 indeed minimize the norms of the residual vectors $|\mathbf{A}\,\mathbf{I} - \mathbf{x}'|$ and $|\mathbf{A}\,\mathbf{J} - \mathbf{y}'|$. The Singular Value Decomposition has the advantage of giving a clear diagnosis about the rank and condition of matrix **A**; this is useful in photogrammetric applications, when one has to make sure that the feature points on the terrain can be considered noncoplanar before applying this algorithm (for an extension of this algorithm to coplanar points using a matrix **A** of rank 2, see (Oberkampf, DeMenthon and Davis 1993)).

Once we have obtained least square solutions for **I** and **J**, the unit vectors **i** and **j** are simply obtained by normalizing **I** and **J**. As mentioned earlier, the three elements of the first row of the rotation matrix of the object are then the three coordinates of vector **i** obtained in this fashion. The three elements of the second row of the rotation matrix are the three coordinates of vector **j**. The elements of the third row are the coordinates of vector **k** of the z-axis of the camera coordinate system and are obtained by taking the cross-product of vectors **i** and **j**.

Now the translation vector of the object can be obtained. It is vector $\mathbf{OM}_0$ between the center of projection, O, and M_0, the origin of the object coordinate system. This vector $\mathbf{OM}_0$ is aligned with vector $\mathbf{Om}_0$ and is equal to $Z_0\mathbf{Om}_0/f$, i.e. $\mathbf{Om}_0/s$. The scaling factor s is obtained by taking the norm of vector **I** or vector **J**, as can be seen from the definitions of these vectors in terms of **i** and **j** provided with Eqs. 9 and 10.

If the ε_i terms used in Eqs. 9 and 10 are accurate, the rotation matrix and translation matrix just computed can be considered accurate representations of the pose; otherwise, they can be used to compute more accurate values for ε_i, and then the process is repeated, as will be explained below in greater details.

9 Geometric Interpretation for the System Solutions

Geometrically, Eq. 9 states that if the tail of **I** is taken to be at $\mathbf{M}_0$, the head of **I** projects on $\mathbf{M}_0\mathbf{M}_i$ at a point H_i defined by the algebraic measure

$$\overline{M_0H_i} = \frac{x_i'}{|\mathbf{M}_0\mathbf{M}_i|}$$

In other words, the head of **I** must belong to the plane perpendicular to $\mathbf{M}_0\mathbf{M}_i$ at H_i. If we use four feature points, M_0, M_1, M_2, M_3, and these points are chosen to be noncoplanar, then **I** is completely defined as the vector with its tail at M_0 and its head at the intersection of the three planes perpendicular to $\mathbf{M}_0\mathbf{M}_1$, $\mathbf{M}_0\mathbf{M}_2$ and $\mathbf{M}_0\mathbf{M}_3$ at H_1, H_2 and H_3 respectively. Analytically, we solve a system of three equations, and the matrix of the system has rank 3. We would use as matrix **B** the inverse of matrix **A** instead of its pseudoinverse.

If more than four feature points are used, the corresponding planes generally do not intersect exactly at a single point, but we would like to choose as head of **I** the point which minimizes the sum of the squares of its distances to these planes. Analytically, the system of equations is overdetermined, the matrix of the system of equations is still of rank 3, and the solution in the least square sense is obtained by using the pseudoinverse **B** of the matrix **A** in Eq. 12.

10 Pseudocode for the POSIT Algorithm

We can now summarize the description of the POSIT algorithm for N feature points (including the reference point M_0) with the following pseudocode

```
• Step 0 (preliminary step, executed once for a set of simultaneously visible feature points):
  Write the matrix A with dimension $(N-1) \times 3$; each row vector is a vector $\mathbf{M}_0\mathbf{M}_i$ connecting the reference feature point $M_0$ to another feature point $M_i$; compute the $3 \times (N-1)$ object matrix B as the pseudoinverse matrix of A;
• Step 1: $\varepsilon_{i(0)} = 0, (i = 1 \cdots N-1), n = 1$;
• Step 2, beginning of loop:
  Compute i, j, and $Z_0$:
    — Compute the image vector $\mathbf{x}'$ with $N-1$ coordinates of the form $x_i(1+\varepsilon_{i(n-1)}) - x_0$, and the image vector $\mathbf{y}'$ with $N-1$ coordinates of the form $y_i(1+\varepsilon_{i(n-1)}) - y_0$.
    — Multiply the $3 \times (N-1)$ object matrix B and the image vectors ($N-1$ coordinates) to obtain vectors I and J with 3 coordinates: $\mathbf{I} = \mathbf{B}\,\mathbf{x}'$ and $\mathbf{J} = \mathbf{B}\,\mathbf{y}'$;
```

  - Compute the scale s of the projection as the average between the norms of $\mathbf{I}$ and $\mathbf{J}$: $s_1 = (\mathbf{I}\cdot\mathbf{I})^{1/2}$, $s_2 = (\mathbf{J}\cdot\mathbf{J})^{1/2}$, $s = (s_1 + s_2)/2$;
  - Compute unit vectors $\mathbf{i}$ and $\mathbf{j}$: $\mathbf{i} = \mathbf{I}/s_1, \mathbf{j} = \mathbf{J}/s_2$;
- Step 3: Compute new ε_i:
  - Compute unit vector $\mathbf{k}$ as the cross-product of $\mathbf{i}$ and $\mathbf{j}$;
  - Compute the z-coordinate Z_0 of the translation vector as $Z_0 = f/s$ where f is the camera focal length;
  - Compute $\varepsilon_{i(n)} = \frac{1}{Z_0}\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}$;
- Step 4: If $|\varepsilon_{i(n)} - \varepsilon_{i(n-1)}|$ >Threshold, $n = n + 1$; Go to Step 2.
- Step 5: Else output pose using values found at last iteration: the full translation vector $\mathbf{OM}_0$ is $\mathbf{OM}_0 = \mathbf{Om}_0/s$; the rotation matrix is the matrix with row vectors $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$; for applications where the rotation matrix must be perfectly orthonormal, renormalize this matrix: compute $\mathbf{k}' = \mathbf{k}/|\mathbf{k}|$, $\mathbf{j}' = \mathbf{k}' \times \mathbf{i}$, and output the matrix with row vectors $\mathbf{i}, \mathbf{j}'$, and $\mathbf{k}'$.

10.1 Geometric Interpretation of the Algorithm

The iterative algorithm described analytically in the previous section can also be described geometrically as follows:

- Step 0 (preliminary step): compute object matrix $\mathbf{B}$ as the pseudoinverse matrix of $\mathbf{A}$;
- Step 1. Assume that the scaled orthographic projections p_i of M_i are superposed with the image points m_i of M_i: $p_{i(0)} = m_i$.
- Steps 2: Compute an approximate pose assuming that the projection model is a scaled orthographic projection.
- Step 3: *Shift the object points from their found approximate positions to positions at the same depth but on their lines of sight* (a deformation of the object).
  Find the images of these shifted points by a scaled orthographic projection model.
- Step 4: If these scaled orthographic image points are not the same as those found at previous iteration, go back to Step 2 using these image points instead of the original image points.
- Step 5: Else, exact pose = last approximate pose.

PROOF. Our goal here is to show that the steps of the geometric description of the algorithm are equivalent to the analytic description provided by the pseudocode.

Steps 1 and 2: As we have seen in Section 6, finding the pose using Eqs. 2 and 3 with calculated values for ε_i (analytical description) amounts to finding the pose for which the points M_i project in points p_i with coordinates $x_i(1+\varepsilon_i)$ and $y_i(1+\varepsilon_i)$ by a scaled orthographic projection (geometric description). At step 1, assuming ε_i to be zero (analytical description) implies $x'_i = x_i$, $y'_i = y_i$ and amounts to assuming that p_i and m_i coincide (Fig. 2).

Step 3: Object points M_i are shifted to the lines of sight of the original image points m_i at constant depth, then projected into points p_i by a scaled orthographic projection (geometric description). The coordinates of p_i for these shifted points M_i are $x_i(1+\varepsilon_i)$ and $y_i(1+\varepsilon_i)$, as we have just seen, with $\varepsilon_i = \frac{1}{Z_0}\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}$ (Eq. 4). This dot product and the term Z_0 are not affected by the shift of M_i in a direction perpendicular to the vector $\mathbf{k}$; thus the terms ε_i can be obtained without accounting for this shift, simply by multiplying the $\mathbf{M}_0\mathbf{M}_i$—defined once and for all in the object reference frame—by the vector $\mathbf{k}$ defined in the same frame, which is the third row of the rotation matrix (analytical description).

Step 4: Once the ε_i terms don't change from one iteration to the next (analytic description), the expressions for the coordinates of the points p_i don't change (geometric description). One or the other test can be used. With the geometric description, no artificial threshold needs to be introduced for the definition of change; one exits the loop once the quantized positions of the points p_i (in pixels) stop moving. □

11 An Intuitive Interpretation of the POSIT Algorithm

The process by which an object pose is found by the POSIT algorithm can be seen from a third, somewhat more intuitive, viewpoint:

- What is known is the distribution of the feature points on the object and the images of these points by perspective projection.
- If we could build SOP images of the object feature points from a perspective image, we could apply the POS algorithm to these SOP images and we would obtain an exact object pose.
- Computing exact SOP images requires knowing the exact pose of the object. However, we can apply POS to the actual image points; we then obtain an approximate depth for each feature point, and we position the feature points on the lines of sight at

these depths. Then we can compute an SOP image. At the next step we apply POS to the SOP image to find an improved SOP image. Repeating these steps, we converge toward an accurate SOP image and an accurate pose.

12 Deformation Measure

The POS algorithm uses at least one more feature point than is strictly necessary to find the object pose. At least four noncoplanar points including M_0 are required for this algorithm, whereas three points are in principle enough if the constraints that **i** and **j** be of equal length and orthogonal are applied (Oberkampf, DeMenthon and Davis 1993). Since we do not use these constraints in POS, we can verify *a posteriori* how close the vectors **i** and **j** provided by POS are to being orthogonal and of equal length. Alternatively, we can verify these properties with the vectors **I** and **J**, since they are proportional to **i** and **j** with the same scaling factor s. We construct a *deformation measure* G, for example as

$$G = |\mathbf{I} \cdot \mathbf{J}| + |\mathbf{I} \cdot \mathbf{I} - \mathbf{J} \cdot \mathbf{J}|$$

The POSIT algorithm finds the translation vector and the transformation matrix that transform the object onto the camera coordinate system so that its feature points fall on the lines of sight of the image points. The transformation matrix may not exactly be an orthonormal rotation matrix, but may comprise a deformation component which slightly deforms the object to adjust it to the lines of sight. The deformation measure G is zero if the transformation matrix is a rotation matrix. The deformation measure G generally becomes large, of course, when the *wrong correspondence* has been established between image points and object points; therefore we have used this measure to solve the correspondence problem when only a small number of correspondence ambiguities exist, applying the POS algorithm with various correspondence permutations and choosing the correspondence permutation that yields the minimum deformation measure as a preliminary step before completing the POSIT algorithm (see also (Basri and Weinshall 1992) for a systematic analysis of this type of approach).

What are possible causes for finding a nonzero deformation measure when the correct correspondence permutation is used? In the ideal case of an object at the correct pose and no image noise, Eqs. 2 and 3 would be exactly verified. Returning to the geometric interpretation of Section 9, the planes perpendicular to $\mathbf{M}_0\mathbf{M}_i$ at points H_{xi} defined at abscissae $(x_i(1+\varepsilon_i) - x_0)/|\mathbf{M}_0\mathbf{M}_i|$ would exactly meet at a single point, the head of **I** corresponding to this correct pose, and the planes perpendicular to $\mathbf{M}_0\mathbf{M}_i$ at points H_{yi} defined at abscissae $(y_i(1+\varepsilon_i) - y_0)/|\mathbf{M}_0\mathbf{M}_i|$ would exactly meet at the head of **J** corresponding to this correct pose. For these two vectors the deformation measure would be zero. During the iteration process of the POSIT algorithm, the terms ε_i are first set to zero, then computed until they reach the values corresponding to the correct pose. Accordingly, the points H_{xi} and H_{yi} are initially different from the correct points, and move toward these points during the iteration. Thus initially the planes do not generally cut at single points, and there is little chance that the vectors **I** and **J** found at the minimum added squared distance between these planes can be equal and perpendicular, and the resulting deformation measure is not zero. As the points H_{xi} and H_{yi} move toward their correct positions at successive iteration steps, the resulting deformation measure tends to zero.

This scenario assumes perfect image detection and camera modelizing. In practice, the coordinates x_i and y_i are not the coordinates of the ideal geometric perspective projections of the M_i, and the planes obtained at convergence generally do not meet at single points. Therefore the final deformation measure is generally not zero (during the iteration it may even go through a value slightly smaller than its final value); accordingly, the final resulting matrix is not exactly orthogonal, but comprises a deformation component which slightly warps the object so that it better fits the noisy image and the assumed camera model. For most applications, an orthonormal transformation matrix is more useful. The output matrix is easily corrected, for instance by normalizing the vector **k** obtained by the cross-product of **i** and **j**, then replacing **j** by the cross-product of **k** and **i**).

13 Illustration of the Iteration Process in POSIT

We can illustrate the iteration process of POSIT with an example using synthetic data. The object is a cube (cube size 10 cm, image 512 $\times$ 512, focal length 760 pixels, corners of the cubes are at a distance between three and four times the cube size from the center of projection of the camera). The cube is assumed transparent, and the feature points are the corners of the cube. We use a full cube in this experiment without

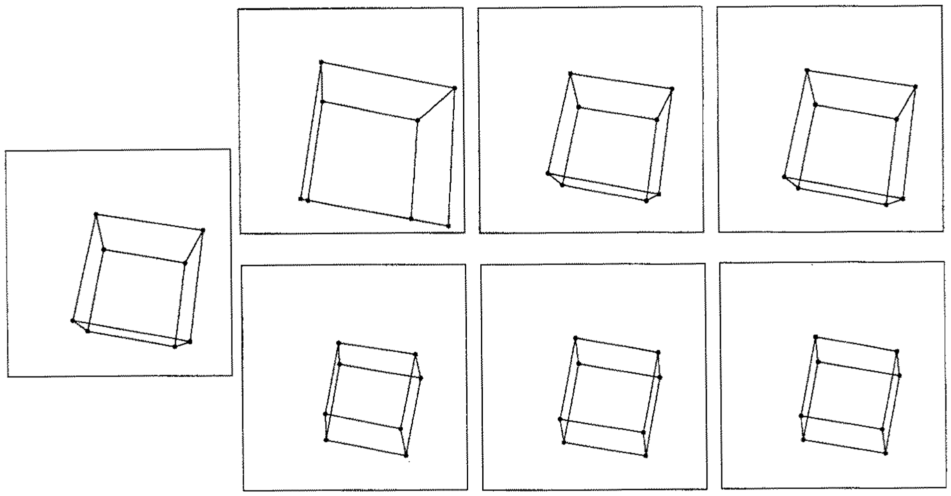


Fig. 3. Perspective images (upper row) and scaled orthographic projections (lower row) for the poses computed in the first three iterations (left to right) of the POSIT algorithm for a cube and its image (left).

simulating hidden faces, because it is interesting to see the converging projections of the parallel edges in the perspective image being transformed into parallel projections in the SOP image (in fact it is not difficult to do actual experiments with eight corners of a cube, using light emitting diodes mounted in a cubic arrangement on a transparent frame).

The projection on the left of Fig. 3 is the given perspective image for the cube. The enclosing square is the boundary of the 512 $\times$ 512 pixel image area. The projections of the cube edges which are drawn on the figure are not used by the algorithm, but are useful for studying the evolution of the scaled orthographic projections of the cube. Because the distance-to-size ratio for the cube is small, some cube edges show a strong convergence in the image. One can get an idea of the success of the POSIT algorithm by computing at each iteration the perspective image of the cube for the transformation found at this iteration. The three projections at the top of Fig. 3 are such perspective projections at three iterations. Note that from left to right, these projections are getting closer and closer to the original image. POSIT does not compute these images. Instead, POSIT computes SOP images using only the actual image corners and the depths it computed for the corners. These images are shown on the bottom row of Fig. 3. Notice that, from left to right, the images of the cube edges become more and more parallel, an indication that the algorithm is getting closer to the correct scaled orthographic projection of the cube, in which parallel lines project onto parallel image lines.

14 Protocol of Performance Characterization

In this section, we attempt to follow the recommendations of Haralick for performance evaluation in computer vision (Haralick 1992). We compute the orientation and position errors of the POS algorithm at the first iteration loop (an approximation which assumes that the perspective image is a scaled orthographic projection), and for the POSIT algorithm once it has converged. Synthetic images are created for two objects, and the poses of the objects computed by POS and POSIT from these images are compared with the actual poses which were used to produce the images. For each of the two objects, we consider ten distances from the camera, 40 random orientations for each distance, and three levels of image noise for each combination.

14.1 Objects

The two objects are

1. A configuration of four points (tetrahedron), such that the three line segments joining the reference point to the other three points are equal (10 cm) and perpendicular to each other (Fig. 4, left)
2. The eight corners of a 10 cm cube. One of the corners is the reference point (Fig. 4, right).

14.2 Object Positions

The reference points of the objects are positioned on the optical axis. The distance from the camera center to the

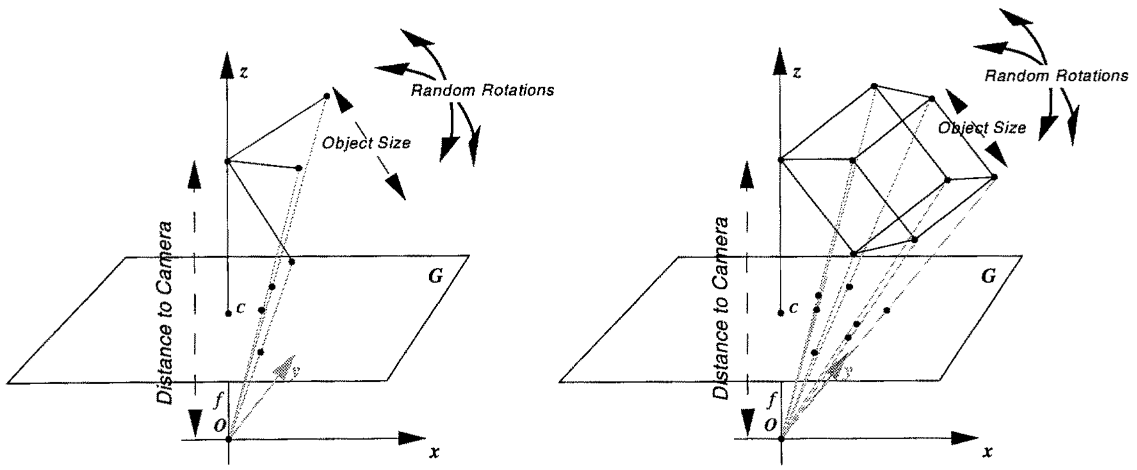


Fig. 4. Definition of objects and parameters used for estimating rotation and translation errors of the POSIT algorithm.

reference point is expressed as a ratio to a characteristic dimension of the object (here this dimension is 10 cm for both objects). Ten distances are considered, from four times to 40 times the object size. These distance-to-size ratios are used as the horizontal coordinates on all the orientation and position error plots.

Around each of these reference point positions, the objects are rotated at 40 random orientations. The rotation matrices defining these 40 orientations are computed from three Euler angles chosen by a random number generator in the range $(0, 2\pi)$.

14.3 *Image Generation*

We obtain images by perspective projection with a focal length of 760 pixels. Here we do not clip the image, in order to allow for large offsets of the images. When the reference point of the cube is 40 cm from the image plane on the optical axis and when the cube is completely on one side of the optical axis, the point at the other end of the diagonal of the cube may be 30 cm from the image plane and have an image 355 pixels from the image center. Only a wide-angle camera with a total angular field of more than 50 degrees would be able to see the whole cube in this position.

We specify three levels of random perturbation and noise in the image. At noise level 1, the real numbers computed for the coordinates of the perspective projections are rounded to integer pixel positions. At noise level 2, the integer positions of the lowest level are disturbed by vertical and horizontal perturbations of ± 1 pixel around the integer positions. These are created by a uniform random number generator. At noise level 3, the amplitude of the perturbations are ± 2 pixels. When the objects are at 400 cm from the camera, the image may be as small as 20 pixels, and a perturbation of two pixels on each side of the image produces a 20% perturbation in image size.

14.4 *Orientation and Position Errors*

For each of the synthetic images, the orientation and position of the object are computed by the POS algorithm (at the first iteration loop, with $\varepsilon_i = 0$), then by the POSIT algorithm at the end of five iterations. These orientations and positions are compared to the actual orientation and position of the object used to obtain the image. We compute the axis of the rotation required to align the coordinate system of the object in its actual orientation with the coordinate system of the object in its computed orientation. The *orientation error* is defined as the rotation angle in degrees around this axis required to achieve this alignment. Details of this computation are given in Appendix B. The *position error* is defined as the norm of the translation vector required to align the computed reference point position with the actual reference point, divided by the distance of the actual reference point position from the camera. Thus the position error is a relative error, whereas the orientation error is a measure in degrees.

14.5 *Combining the Results of Multiple Experiments*

As mentioned above, for each distance-to-size ratio, many rotations are considered. We compute the average and standard deviation of the orientation and position errors over all these rotations and plot the averages with their standard deviation error bars as a function of the distance-to-size ratios. Each plot shows the results both for POS, and for POSIT after five iterations. The plots for the orientation error are shown in Fig. 5, and the plots for the position errors are shown in Fig. 6. In each of these two figures, the plots in the left column are for the tetrahedron, and the plots in the right column are for the cube. The top diagrams are for the lowest image noise level, the middle diagrams for the medium noise level, and the bottom diagrams for the highest noise level.

15 Analysis of the Pose Error Diagrams

15.1 Comparison between POS and POSIT

At short to medium range and low to medium noise, POSIT gives poses with less than two degree rotation errors and less than 2% position errors. Errors increase linearly in proportion to the object range, because of the pixel quantization of the camera; indeed, one can displace a point twice as much when it is twice as far from the camera before its image point jumps to the next pixel. The effect of image noise also increases with range; when the distance ratio is 40, the image points are grouped in a 20 pixel regions, and perturbations of several pixels significantly modify the relative geometry of these image points, and the resulting pose computation. The effects described so far are caused by the imaging process, and are probably characteristic of all the algorithms computing a pose from a single image obtained by a CCD camera.

The iterations of POSIT provide clear improvements over the initial estimate provided by POS ($\varepsilon_i = 0$) when the objects are very close to the camera; on the other hand, they provide almost no improvement when the objects are far from the camera. When the objects are close to the camera, the so-called *perspective distortions* are large, and scaled orthographic projection is a poor approximation of perspective; therefore the performance of POS is poor. For the shortest distance-to-size ratio (4), errors in orientation evaluation are in the 10 degree region, and errors in position evaluation are in the 10% region. When the objects are very far, there is almost no difference between SOP and perspective. This can be seen analytically: the terms ε_i in Eq. 4 are negligible when the objects are far, so that the perspective equations become identical to the SOP equations. Thus POS gives the best possible results, and the iterations of POSIT cannot improve upon them. POS gives its best performance for distances around 30 times the object size for low image noise, and around 20 times for high image noise—half way between the large errors of perspective distortion at short distances and the large errors of pixel quantization at large distances—with orientation errors in the three degree region and position level in the 3% region.

15.2 Comparison between Cube and Tetrahedron

The long error bars at short range for POS seem to be due to the fact that the apparent image size can be very different depending on the orientation. For example, the cube looks like an object of size 10 cm when a face is parallel to the image plane, but one dimension is 70% larger when a cube diagonal is parallel to the image plane. In this last configuration, the reference point projects at the image center whereas the opposite corner is offcentered by more than 323 pixels, with a large resulting perspective distortion. The tetrahedron does not have as large apparent size changes, which may explain the fact that at short viewing distance the average error and standard deviation produced by POS are smaller for this shape than for the cube. This is more an artifact of the problem of defining object size with a single number than a specific advantage of the tetrahedron over the cube.

At high noise level and long range, the performance with the cube becomes almost twice as good as with the tetrahedron for POS and POSIT, probably because the least square method averages out the random errors on the points, and the averaging improves when more points are made available to the method.

16 Convergence Analysis for POSIT

With the distance-to-size ratios used in the rotation and translation error evaluations above, the POSIT algorithm would converge in four or five iterations. The convergence test used here consists of quantizing (in pixels) the coordinates of the image points in the SOP images obtained at successive steps, and terminating when two successive SOP images are identical (see Appendix A).

One can apply POSIT with 1D images of a 2D world, and in this case one can analytically show that the quantities determining the algorithm's convergence are ratios of image coordinates over focal length, i.e. tangents of the angles between the optical axis and the lines of sight. When all the ratios are smaller than 1, the algorithm converges. The feature points are then

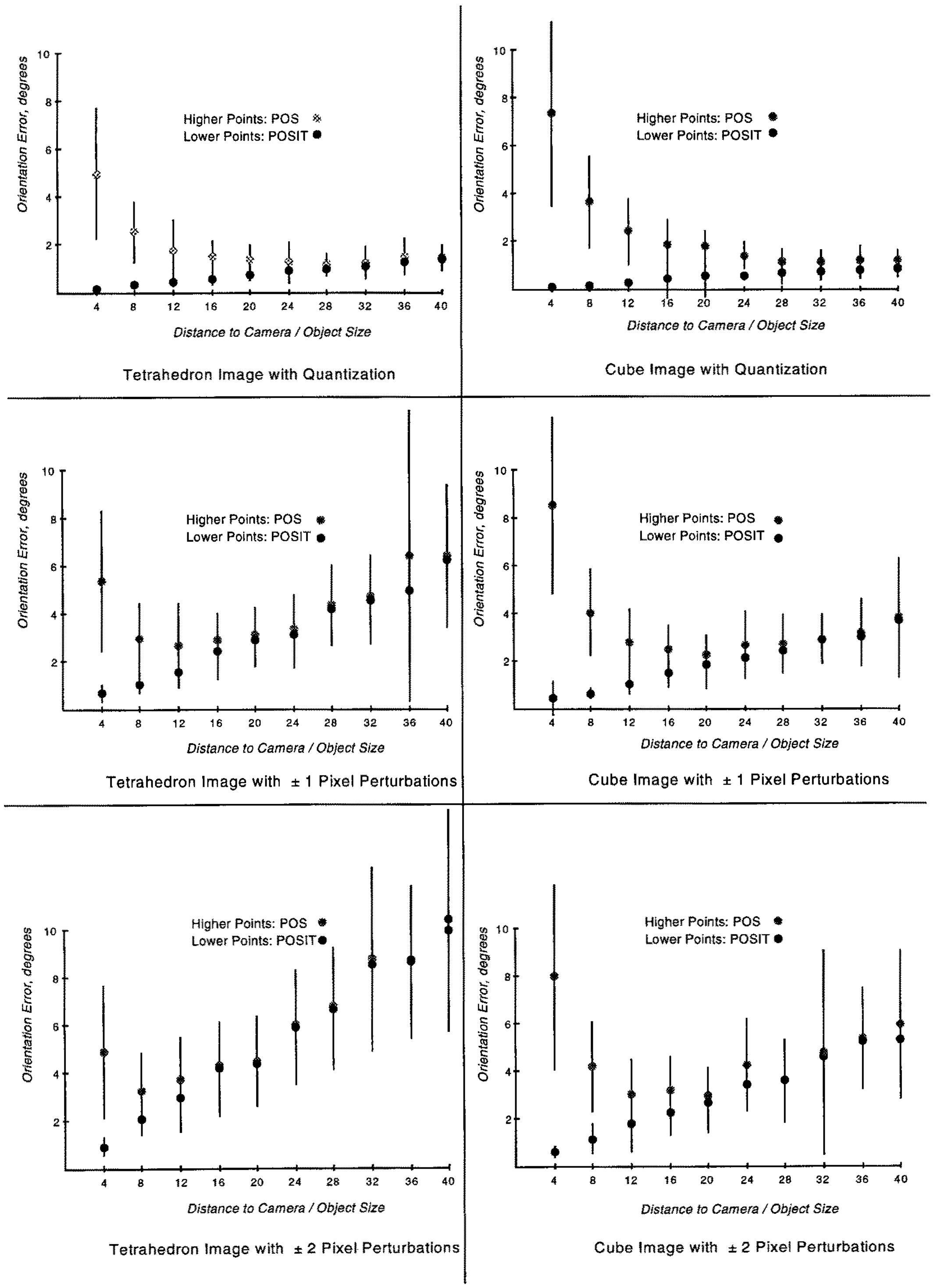


Fig. 5. Angular orientation errors for a tetrahedron (left) and for a cube (right) for 10 distances from the camera, with three image noise levels (quantization, ± 1 pixel, ± 2 pixels).

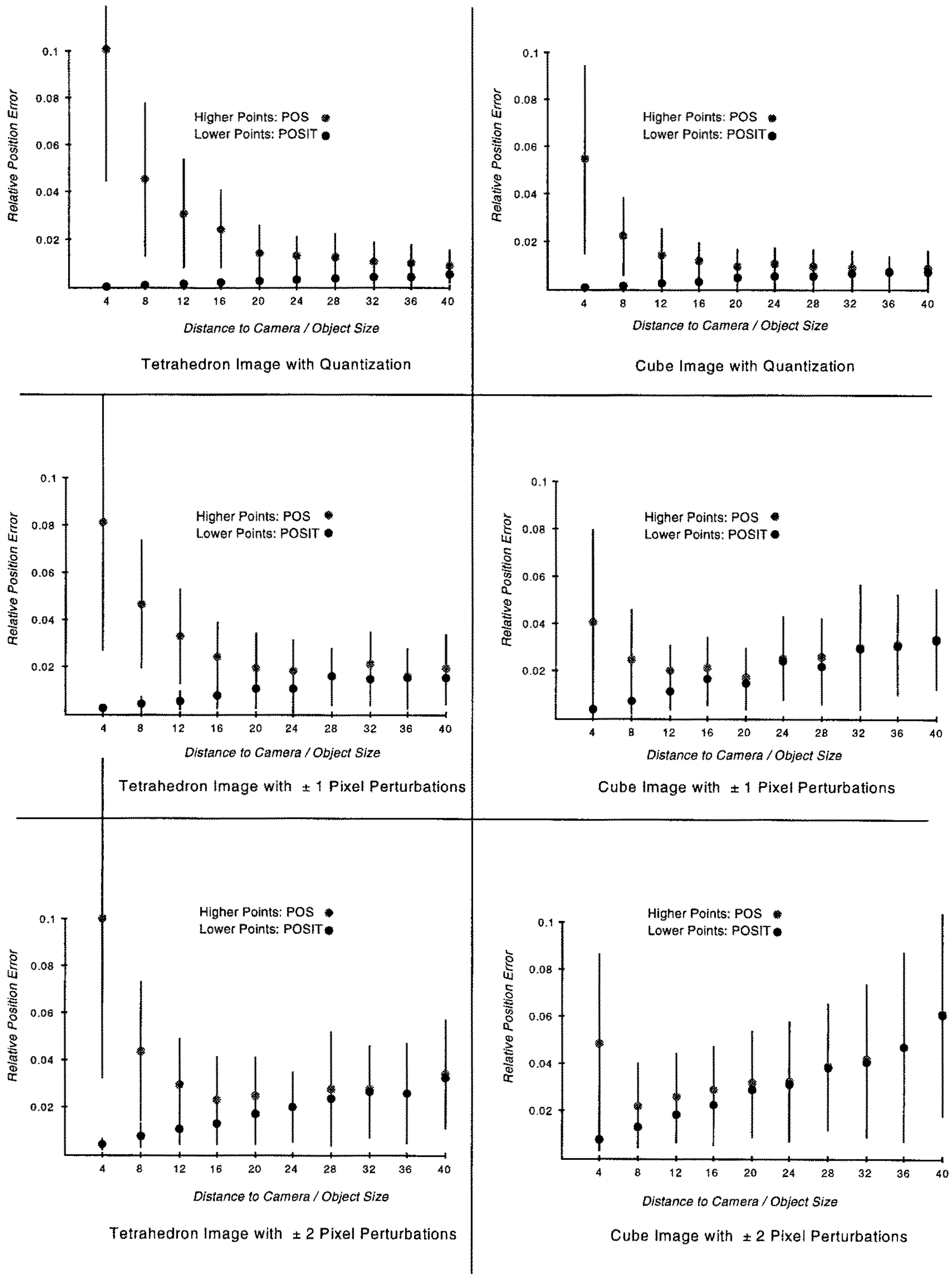
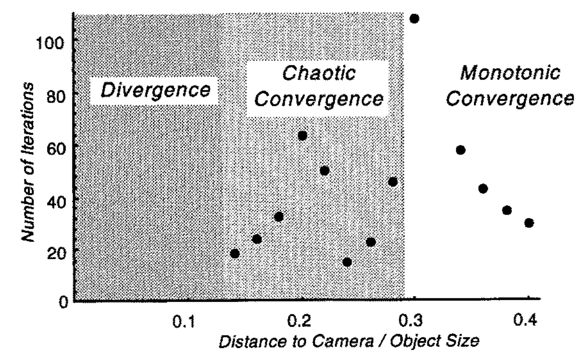


Fig. 6. Relative position errors for a tetrahedron (left) and for a cube (right) for 10 distances from the camera, with three image noise levels (quantization, ±1 pixel, ±2 pixels).



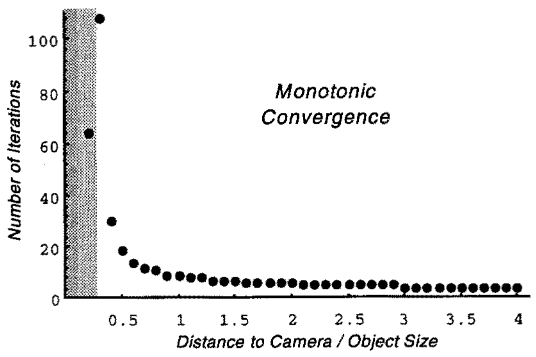


Fig. 7. Number of iterations for POSIT as a function of distance to camera. *Top*: Definition of distance and object size. *Middle*: Convergence results at very short ranges. Convergence occurs if the 10 cm cube is more than 1.2 cm away from the camera. *Bottom*: Number of iterations for a wider range of distances.

seen with a view angle of less than 45 degrees. Therefore with a camera with a 90 degree total field of view, the algorithm would converge for all possible image points. When all the view angles are more than 45 degrees, the algorithm diverges. Thus with an object with all its image points at the periphery of the field of a 110 degree camera the algorithm would diverge. In mixed situations with small and large angles, mixed results are obtained; image points close to the image center contribute to convergence, and balance the negative effect of peripheral image points.

A mathematical analysis of the conditions of convergence in the more interesting case when POSIT is applied to 2D images in a 3D world has eluded us so far; however, in simulations, convergence appears to be similarly dependent on the angles of the lines of sight. A cube is displaced along the camera optical axis (Fig. 7). One face is kept parallel to the image plane, because at the shorter ranges being considered, the cube cannot be rotated much without intersecting the image plane. The distance used to calculate the distance-to-object size ratio in the plots is the distance from the center of projection to the cube. Noise of ± 2 pixels is added to the perspective projection. For a cube of 10 cm, four iterations are required for convergence until the cube is 30 cm from the center of projection. The number gradually climbs to eight iterations as the cube reaches 10 cm from the center of projection, and 20 iterations for 5 cm. Then the number increases sharply to 100 iterations for a distance of 2.8 cm from the center of projection. In reference to our prior 1D observations, at this position the images of the close corners are more than two focal lengths away from the image center, but the images of the far corners are only half a focal length away from the image center and probably contribute to preserving the convergence.

Up to this point the convergence is monotonic. At still closer ranges the mode of convergence changes to a nonmonotonic mode, in which SOP images of successive steps seem subjected to somewhat random variations from step to step until they hit close to the final result and converge rapidly. The number of iterations ranges from 20 to 60 in this mode, i.e. less than for the worse monotonic case, with very different results for small variations of object distance. We label this mode "chaotic convergence" in Fig. 7. Finally, when the cube gets closer than 1.2 cm away from the center of projection, the differences between images increase rapidly and the algorithm clearly diverges. Note, however, that in order to see the close corners of the cube at this range a camera would require a total field of more than 150 degrees, i.e. a focal length of less than 1.5 mm for a 10 mm CCD chip, an improbable configuration. This preliminary convergence analysis and the simulations of the previous section indicate that for ordinary cameras, the POSIT algorithm seems to converge without problems in a few iterations.

We know, however, that some specific configurations of noncoplanar feature points have been shown to produce *ambiguous* images for some isolated positions of the object. Here we call the image ambiguous when distinct poses of the object in space can produce the same image with the same correspondences. In other words, once we have found a rigid transformation that positions each of the feature points on a line of sight of an image point, a second rigid transformation could be found that moves each feature point to another position on the same line of sight. Examples

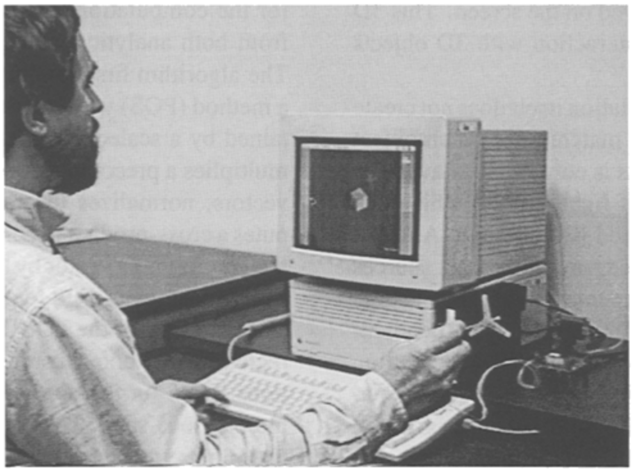


Fig. 8. Video-based 3D mouse. The signal from the camera is sent to the black box at the right of the computer. This box detects the images of the light sources of the mouse and transmits their coordinates to the computer. The computer runs a driver based on POSIT.

of such configurations can be found in (Fischler and Bolles 1981). The POSIT algorithm for a noncoplanar configuration of points produces a single pose. If the feature points on the object happen to be located in space in a configuration that produces an ambiguous image, the POSIT algorithm will compute only one of the possible poses, and there are good chances that this pose may not be the one occupied by the object. We note in our defense that numerical algorithms relying on the Newton-Raphson method (Lowe 1985; Lowe 1991; Yuan 1989) do not behave differently in this case. On the other hand, Roberts' method finds the perspective projection matrix which maps world points to image points expressed with homogeneous coordinates. For ambiguous images, this mapping is not unique, and this fact can be detected by the fact that the matrix of the linear system used in this method is singular. Therefore, Roberts' method seems preferable for applications in which detecting pathological configurations is important. An analysis of the geometry of such configurations can be found in (Faugeras 1993; Maybank 1992). For an ambiguous image to occur, the object points and the camera's center of projection must belong to the same twisted cubic curve in space. Such a curve is defined by six points, therefore one can make sure that the event will not happen by choosing as feature points seven object points that cannot all belong to the same cubic curve. When fewer points are considered, ambiguous images can occur but probably remain unlikely. When the POSIT algorithm is used in tracking a moving object using fewer points, one can in principle detect the fact that the algorithm has computed the wrong pose from an ambiguous image by noticing a discontinuity in the sequence of poses.

17 Real-Time Experiments

We originally developed the POSIT algorithm for the purpose of providing 60 pose computations per second in a video-based 3D mouse system able to run on a personal computer (DeMenthon 1993; DeMenthon 1995; DeMenthon and Fujii 1994). Figure 8 is a general view of a prototype. The 3D mouse comprises several small infrared sources. A camera is positioned next to the computer display and faces the user. This camera is equipped with a filter that blocks the visible light spectrum and transmits wave lengths longer than 1 μm. In the images, the light sources appear as bright spots on a dark background and are easy to detect. The black box along the right side of the computer contains a microcontroller that computes the centroids of these bright spots for every image field and transmits the centroid coordinates to a serial port of the computer. In our latest implementation, we have integrated the camera and microcontroller functions into a very small "smart" camera that can receive image processing code through the serial line, and can send simple image processing results through the serial line (this camera may be useful in other applications as well, such as range scanning and robot navigation). From the centroid coordinates received through its serial port, the computer calculates the pose of the 3D mouse 60 times per second, and computes the corresponding perspective image of a 3D cursor, which is displayed on the screen. This 3D cursor allows an intuitive interaction with 3D objects represented on the screen.

We find that the pose calculation itself does not create any problems, provided the matching between object light sources and image spots is correct. We have used for the mouse alignments of light sources which can be easily detected and matched in the image. Alternatively, we have used a tetrahedron of four light sources arranged so that the line segments between one source and the three others are equal and mutually perpendicular. This configuration simplifies the pose calculation, because the object matrix **B** is then a 3×3 identity matrix. With this mouse, we choose the matching which minimizes a combination of the deformation measure of Section 12 and the difference from the previous pose. The matching is nontrivial in some conditions. In particular, with the tetrahedron configuration, one can often find two matchings which would correspond to two poses which are symmetric with respect to a plane parallel to the image plane. When the two poses are close together, it is difficult to choose the better pose. If the user never attempts to point the mouse toward himself (a maneuver which has a good chances of resulting in the hand occluding a light source anyway), then one pose can be rejected. Also, image spots are very often close together, and the matching may be difficult in these conditions too. When image spots get closer, they may end up merging, and when this occurs a single centroid is detected. We keep track of the assignments of spots that are close together, so that when they merge, we can assign the same image centroid to two line sources. With these precautions, we obtain a reasonably robust and usable system in which the 3D cursor responds smoothly and predictably to the rotations and translations of the 3D mouse in space. Details about this system can be found in (DeMenthon 1993).

From these experiments, it seems to us that with a fast algorithm such as POSIT, a video-based approach may be an attractive alternative in the growing field of interactive 3D graphics, where both mechanical, magnetic, acoustic and optical pose trackers are being developed (Meyer, Applewhite and Biocca 1992).

18 Summary and Discussion

We have presented an algorithm, POSIT, that can compute the pose of an object from an image containing several noncoplanar feature points of the object. We have described in pseudocode form the steps required for the computation, explaining the role of each step from both analytical and geometrical points of view. The algorithm first computes an approximate pose by a method (POS) which assumes that the image was obtained by a scaled orthographic projection. This step multiplies a precomputed object matrix and two image vectors, normalizes the resulting vectors, then computes a cross-product to complete the rotation matrix; it then multiplies a vector by the norm used in the normalization just mentioned to obtain the translation vector. The next step of the POSIT algorithm computes "corrected" image points using scaled orthographic projections based on the approximate object pose found at the previous step. These two steps are repeated until no improvement is detected. Simulations show that the algorithm converges in a few iterations in the domain of useful configurations of a camera and an object. We have characterized the performance of the algorithm by a number of experiments on synthetic data with increasing levels of image noise. The POSIT algorithm appears to remain stable and to degrade gracefully with increasing image noise levels.

POSIT may be a useful alternative to popular pose algorithms because of the following advantages:

1. It does not require an initial pose estimate;
2. Its code is easy to implement. In compact languages such as Mathematica, only around 25 lines of code are necessary (Appendix A);
3. It can run ten times faster than those algorithms, since it typically requires an order of magnitude fewer arithmetic operations;

One of the objections that may be raised is that since POSIT does not make full use of the fact that the rotation matrix is orthonormal, it is bound to be less accurate than algorithms that account for this fact. This is probably the case when the minimum number of feature points (4) is considered, but the difference should disappear as the number of points is increased and the pose information available in the image becomes more redundant. Comparative experiments would be useful in deciding about this issue. If indeed some algorithms are shown to provide an advantage in accuracy, and if the considered application requires such additional accuracy, the advantages of POSIT mentioned above may still make it useful for producing an initial pose for these algorithms.

Before going to such lengths, one has to remember that there are intrinsic limitations on pose calculation from single images that no algorithm using single images may be able to overcome. For example, object displacements in the direction of the optical axis move the feature points more or less along the lines of sight (more so as the object size/distance decreases), so that rather large object displacements can occur before they translate into jumps to neighboring pixels in the image. Methods using line features (Dhome et al. 1989) would have the same problems in detecting these displacements. In some applications, it is possible to obtain greater accuracy by combining the information obtained from two cameras with optical axes at a large angle (ideally 90 degrees), at the expense of added complexity in calibration and computation.

In photogrammetric applications, the feature points are often coplanar or almost coplanar. In these situations, the method described in this paper must be significantly modified, because the matrix **A** describing the positions of the feature points in the scene has rank 2. This extension of the POSIT algorithm to planar scenes is described in (Oberkampf, DeMenthon and Davis 1993).

Finally, assigning the proper correspondence between object points and their images is a required preliminary step for the POSIT algorithm; this problem has been addressed only briefly. In Section 12, we suggest that the algorithm be run for different point correspondences, and that the correct correspondence corresponds to the minimal deformation factor. In our 3D mouse experiments (Section 17), we have combined this technique with comparisons between successive pose solutions to produce robust correspondence assignments; this is a feasible technique only if a few correspondence permutations have to be examined. Methods which do not depend exponentially on the number of points and combine the search for the correct pose and the search for the correct correspondence have been proposed (Breuel 1992; DeMenthon 1993); we find that the search is painfully slow because it takes place in a high–dimensional transformation space. For these methods to become attractive, novel criteria for further pruning the search tree will have to be discovered.

Acknowledgments

We thank Thor Bestul for valuable discussions, and Azriel Rosenfeld for his comments on earlier drafts of this paper; we also express our thanks to the three reviewers for very useful insights, which led us in particular to develop the analytical formulation of POSIT. The support of the Defense Advanced Research Agency under Contract DACA76-92-C-0009 is gratefully acknowledged. A shorter version of this paper considering only the geometric interpretation of the algorithm was published in the Proceedings of the 1992 European Conference on Computer Vision (DeMenthon and Davis 1992).

Appendix A: A Mathematica Program Implementing POS and POSIT

Compute the pose of an object given a list of 2D image points, a list of corresponding 3D object points, and the object matrix (the pseudoinverse matrix for the list of object points). The first point of the image point list is taken as a reference point. The outputs are the pose computed by POS using the given image points and the pose computed by POSIT.

```
GetPOSIT[imagePoints_,objectPoints_,
objectMatrix_, focalLength_]:= Module[
{objectVectors, imageVectors, IVect,
JVect, ISquare, JSquare, IJ,
imageDifference, row1, row2, row3,
scale1, scale2, scale,
oldSOPImagePoints,
SOPImagePoints, translation, rotation,
count = 0, converged = False},
objectVectors = (#-objectPoints[[1]])&
/@ objectPoints;
oldSOPImagePoints=imagePoints;
(* loop until difference between 2 SOP
images is less than one pixel *)
While[! converged,
  If[count==0,
   (* we get image vectors from image
   of reference point for POS: *)
   imageVectors = Map[(#
   - imagePoints
   [[1]])&, imagePoints],
   (* else count>0, we compute a SOP
   image first for POSIT: *)
   SOPImagePoints =
   imagePoints(1+(objectVectors.row3)
   /translation[[3]]);
   imageDifference = Apply[Plus,
   Abs[Round[Flatten
   [SOPImagePoints]]-
   Round[Flatten[oldSOPImagePoints]]]];
   oldSOPImagePoints = SOPImagePoints;
  imageVectors = Map[(#
  - SOPImagePoints[[1]])&,
  SOPImagePoints]
 ]; (* end else count>0*)
 {IVect, JVect}
 = Transpose[objectMatrix .
 imageVectors];
 ISquare = IVect.IVect; JSquare
 = JVect.JVect; IJ = IVect.JVect;
 {scale1, scale2} = Sqrt[{ISquare,
 JSquare}];
 {row1, row2} = {IVect/scale1,
 JVect/scale2};
 row3 = RotateLeft[row1]
 RotateRight[row2] -
RotateLeft[row2] RotateRight[row1];
  (* cross-product *)
  rotation={row1, row2, row3};
  scale = (scale1 + scale2)/2.0;
  (* scaling factor in SOP *)
  translation
  = Append[imagePoints[[1]],
  focalLength]/scale;
  converged = (count>0)
  && (imageDifference<1);
  count++
]; (* End While *)
Return[{rotation, translation}]]

(* Example of input and output: *)

focLength = 760;
cube ={{0,0,0},{10,0,0},{10,10,0},
      {0,10,0},{0,0,10},
      {10,0,10},{10,10,10},{0,10,10}};
cubeMatrix = PseudoInverse[cube]//N;
cubeImage = {{0,0},{80,-93},{245,-77},
  {185,32},{32,135},
  {99,35},{247, 62},{195, 179}};
{POSITRot,POSITTrans} =
GetPOSIT[cubeImage, cube, cubeMatrix,
focLength]
Out[1] = {{{0.49010, 0.85057, 0.19063},
   {-0.56948, 0.14671, 0.80880},
   {0.65997, -0.50495, 0.55629}},
   {0, 0, 40.02637}}
```

Appendix B: Angular Error

In our performance evaluation, the object has a coordinate system in a known orientation, and the POS and POSIT algorithms compute from the image of the object a coordinate system that is in a different orientation. We want to compute how far off the computed orientation is from the actual orientation. We find the axis of the rotation required to align the coordinate system of the object in its actual orientation with the coordinate system of the object in its computed orientation. The angular error is the rotation angle in degrees around this axis required to achieve this alignment. The axis of rotation and the angle for the alignment can be readily found with quaternions, but we propose a more direct method here. Given the two unit vectors **i** and **i**′ of the x-axes of the two coordinate systems, the axis of rotation must belong to a plane with respect to which **i** and **i**′ are mirror images of each other. Therefore this plane is perpendicular to the vector **i**′ − **i**. Similarly, the axis belongs to the plane perpendicular to **j**′ − **j** and to the plane perpendicular to **k**′ − **k**. Thus the axis must have a direction **n** perpendicular to both **i**′ − **i**, **j**′ − **j** and **k**′ − **k**. The coordinates of **n** satisfy the homogeneous system composed of the equation

$$(i'_x - i_x)n_x + (i'_y - i_y)n_y + (i'_z - i_z)n_z = 0$$

and two similar equations in **j**′ − **j** and **k**′ − **k**. This system is solved by Singular Value Decomposition. Then the required angle of the rotation is the angle which brings the plane (**n**, **i**) to the plane (**n**, **i**′), i.e. the angle between the cross product **n** × **i** and the cross product **n** × **i**′. The angle between (**n**, **j**) and (**n**, **j**′) and the angle between (**n**, **k**) and (**n**, **k**′) may be slightly different; thus we compute the average of these three angles.

References

1. Abidi, M.A. and Chandra 1991. T. A New Efficient and Direct Solution for Pose Estimation Using Quadrangular Targets: Algorithm and Evaluation. Dept. of Electrical and Computer Engineering, The University of Tennessee, to be published in *IEEE Trans. on Pattern Analysis and Machine Intelligence.*
2. Basri, R. and Weinshall, D. 1992. Distance Metric between 3D Models and 2D Images for Recognition and Classification. MIT A.I. Memo No. 1373.
3. Breuel, T.M. 1992. Fast Recognition using Adaptive Subdivisions of Transformation Space. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Champaign, IL, pp. 445–451.
4. DeMenthon, D.F. 1993. De la Vision Artificielle à la Réalité Synthétique: Système d'interaction avec un ordinateur utilisant l'analyse d'images vidéo. Doctoral Thesis, Université Joseph Fourier – Grenoble I, Laboratoire TIMC/IMAG.
5. DeMenthon, D.F. 1993. Recognition and Tracking of 3D Objects by 1D Search. Proc. ARPA Image Understanding Workshop, Washington, DC, pp. 653–659.

6. DeMenthon, D.F. and Davis, L.S. 1992. New Exact and Approximate Solutions of the Three-Point Perspective Problem. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 14, pp. 1100–1105.
7. DeMenthon, D.F. and Davis, L.S. 1992. Model-Based Object Pose in 25 Lines of Code. *Computer Vision–ECCV 92*, Lecture Notes in Computer Science 588, G. Sandini (Ed.), pp. 335–343, Springer-Verlag.
8. DeMenthon, D.F. 1993. Computer Vision System for Position Monitoring in Three Dimensions using Non-Coplanar Light Sources Attached to a Monitored Object. U.S. Patent 5,227,985, (Application 07/747124, August 1991)
9. DeMenthon, D.F. and Fujii, Y. 1994. Three Dimensional Pointing Device Monitored by Computer Vision. U.S. Patent 5,297,061, (Application 08/063489, May 1993).
10. DeMenthon, D.F. 1995. Computer Vision System for Accurate Monitoring of Object Pose, U.S. Patent 5,388,059 (Patent Application 08/098470, December 1992).
11. Dhome, M., Richetin, M., Lapreste, J.T., and Rives, G. 1989. Determination of the Attitude of 3D Objects from a Single Perspective View. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 11, pp. 1265–1278.
12. Egli, W.H., Miller, J.W. and Setterholm, J.M. 1987. Method and Apparatus for Determining Location and Orientation of Objets. U.S. Patent 4, 672, 562.
13. Faugeras, O. 1993. Three-Dimensional Computer Vision—a Geometric ViewPoint. MIT Press.
14. Fischler, M.A. and Bolles, R.C. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Comm. ACM*, Vol. 24, pp. 381–395.
15. Haralick, R.M. 1992. Performance Characterization in Computer Vision, University of Washington C.S. Technical Report, July 1991; also Performance Characterization in Image Analysis: Thinning, a Case in Point. Pattern Recognition Letters, Vol. 13, pp. 5–12.
16. Horaud, R., Conio, B., and Leboulleux, O. 1989. An Analytical Solution for the Perspective-4-Point Problem. *Computer Vision, Graphics, and Image Processing*, Vol. 47, pp. 33–44.
17. Huttenlocher and D. and Ullman, S. 1988. Recognizing Solid Objects by Alignment. *Proc. DARPA Image Understanding Workshop*, pp. 1114–1122.
18. Lowe, D.G., 1985. Perceptual Organization and Visual Recognition. Kluwer Academic Publishers.
19. Lowe, D. G., 1991. Fitting Parameterized Three-Dimensional Models to Images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 13, pp. 441–450.
20. Maybank, S.J. 1992. The Projective Geometry of Ambiguous Surfaces. *Phil. Trans. R. Soc. Lond. A 332*, pp. 1–47.
21. Meyer, K., Applewhite, H.L., and Biocca, F.A. 1992. A Survey of Position Trackers. Presence, Vol. 1, pp. 173–200, Spring.
22. Oberkampf, D., DeMenthon, D.F., and Davis, L.S. 1993. Iterative Pose Estimation using Coplanar Feature Points. *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 626–627, New York, 1993; full version: Center for Automation Research Technical Report CAR-TR-677, University of Maryland.
23. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Veterling. W.T. 1988. *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK.
24. Roberts, L.G. 1965. Machine Perception of Three-Dimensional Solids. In *Optical and Electrooptical Information Processing*, J. Tippet et al., eds., MIT Press.
25. Sutherland, I.E. 1974. Three–Dimensional Input by Tablet. *Proceedings of the IEEE*, Vol. 62, pp. 453–461.
26. Tomasi, C. 1991. Shape and Motion from Image Streams: A Factorization Method. Technical Report CMU-CS-91-172. Carnegie Mellon University.
27. Tsai, R.Y. 1987. A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses. *IEEE J. Robotics and Automation*, Vol. 3, pp. 323–344.
28. Ullman, S. and Basri, R. 1991. Recognition by Linear Combinations of Models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 13, pp. 992–1006.
29. Yuan, J.S.C. 1989. A General Photogrammetric Method for Determining Object Position and Orientation. *IEEE Trans. on Robotics and Automation*, Vol. 5, pp. 129–142.