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 20742-3411, USA

Abstract. We find the pose of an object from a single image when the relative geometry of four or more noncoplanar visible feature points is known. We first describe an algorithm, *POS* (Pose from Orthography and Scaling), that solves for the rotation matrix and the translation vector of the object by a linear algebra technique under the scaled orthographic projection approximation. We then describe an iterative algorithm, *POSIT* (POS with ITerations), that uses the pose found by POS to remove the "perspective distortions" from the image, then applies POS to the corrected image instead of the original image. POSIT generally converges to accurate pose measurements in a few iterations. Mathematica 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. Researchers have formulated closed form solutions when a few feature points are considered in coplanar and noncoplanar configurations (see [5] for a review). However, numerical pose computations can make use of larger numbers of feature points and tend to be more robust; the pose information content becomes highly redundant; the measurement errors and image noise average out between the feature points. Notable among these computations are the methods proposed by Tsai [7] and by Yuan [9].

The method we describe here can also use many noncoplanar points and applies a novel iterative approach. Each iteration comprises two steps.

1. In the first step we approximate the "true" perspective projection (*TPP*) with a scaled orthographic projection approximation (*SOP*). Finding the rotation matrix and translation vector from image feature points with this approximation is very simple. We call this algorithm "POS" (Pose from Orthography and Scaling) (see [6] for similar solutions without scaling, and [8] for similar equations applied to object recognition without pose computation).
2. We use the approximate pose from the first step to displace the TPP image feature points toward the positions they would have if they were SOP projections.

We stop the iteration when the image points are displaced by less than one pixel. Since the POS algorithm in the first step requires an SOP image instead of a TPP image to produce an accurate pose, using the displaced points of the second step instead of the TPP

* The support of the Defense Advanced Research Projects Agency (ARPA Order No. 6989) and the U.S. Army Topographic Engineering Center under Contract DACA76-89-C-0019 is gratefully acknowledged, as is the help of Sandy German in preparing this report.

points yields an improved pose at the second iteration, which in turns leads to displaced image points closer to SOP points, etc. We call this iterative algorithm "POSIT" (POS with ITerations). Four or five iterations are typically required to converge to an accurate pose.

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**.

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 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 coordinates of the same point in the *camera* coordinate system are called (X_i, Y_i, Z_i).

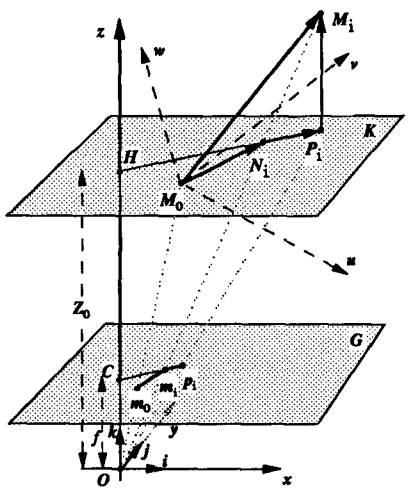

Fig. 1. Perspective projection and scaled orthographic projection for object point M_i and object reference point M_0.

3 Scaled Orthographic Projection and Perspective Projection

Consider a point M_i of the object (Fig. 1). In "true" perspective projection (TPP), its image is a point m_i of the image plane G which has coordinates

$$x_i = fX_i/Z_i, \quad y_i = fY_i/Z_i \tag{1}$$

Scaled orthographic projection (*SOP*) is an approximation to TPP. One assumes that the depths Z_i of different points M_i of the object are not very different from one another, and can all be set to the depth Z_0 of the reference point M_0 of the object. In SOP, the image of a point M_i is a point p_i of the image plane G which has coordinates

$$x'_i = fX_i/Z_0, \quad y'_i = fY_i/Z_0 \tag{2}$$

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 TPP. The image coordinates of the SOP projection p_i can also be written as

$$x'_i = x_0 + s(X_i - X_0), \quad y'_i = y_0 + s(Y_i - Y_0) \tag{3}$$

The geometric construction for obtaining the TPP image point m_i of M_i and the SOP image point p_i of M_i is shown in Fig. 1. Classically, the TPP 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 m_0p_i is parallel to M_0P_i and is scaled down from M_0P_i by the scaling factor $s = f/Z_0$. Eq. (3) simply expresses the proportionality between these two vectors.

4 Approximate Pose from SOP (POS)

We find an approximate pose by assuming that the TPP image points m_i can be approximated by the SOP image points p_i (Fig. 1). Our goal is to recover the coordinates of the three unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ of the camera coordinate system in the object coordinate system using the SOP approximation. Indeed *these three vectors expressed in the object coordinate system are the row vectors of the rotation matrix* $\mathbf{R}$. The translation vector $\mathbf{T}$ for the object is the vector $\mathbf{OM}_0$. Once we find the scaling factor of the SOP, this vector $\mathbf{OM}_0$ is simply a scaled up version of the image vector $\mathbf{Om}_0$. We call this pose calculation method *POS* (Pose from Orthography and Scaling).

We modify the two expressions of Eq. (3). After expressing the coordinates $X_i - X_0$ and $Y_i - Y_0$ of the vector $\mathbf{M_0M}_i$ as dot products of $\mathbf{M_0M}_i$ with unit vectors $\mathbf{i}$ and $\mathbf{j}$, we obtain

$$x_i - x_0 = s\,\mathbf{i} \cdot \mathbf{M_0M}_i, \quad y_i - x_0 = s\,\mathbf{j} \cdot \mathbf{M_0M}_i$$

We define $\mathbf{I}$ and $\mathbf{J}$ as scaled down versions of the unit vectors $\mathbf{i}$ and $\mathbf{j}$

$$\mathbf{I} = s\,\mathbf{i}, \quad \mathbf{J} = s\,\mathbf{j} \tag{4}$$

which yields

$$x_i - x_0 = \mathbf{I} \cdot \mathbf{M_0M}_i, \quad y_i - y_0 = \mathbf{J} \cdot \mathbf{M_0M}_i \tag{5}$$

These can be viewed as *linear equations* where the unknowns are the coordinates of vector $\mathbf{I}$ and vector $\mathbf{J}$ in the object coordinate system. The other parameters are known.

Writing Eq. (5) for the object points $M_0, M_1, M_2, M_i, \ldots, M_n$ and their images, we generate a linear system for the coordinates of the unknown vector $\mathbf{I}$ and a linear system for the unknown vector $\mathbf{J}$:

$$\mathbf{A\,I} = \mathbf{x}, \quad \mathbf{A\,J} = \mathbf{y} \tag{6}$$

where **A** is the matrix of the coordinates of the object points M_i in the object coordinate system and **x** and **y** are the vectors of the x and y coordinates of the image points m_i offset by the coordinates of the image point m_0.

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

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

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**.

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**. The POS method uses at least one more point than is strictly necessary to find the object pose. At least four noncoplanar points including M_0 are required for this method, whereas three points are in principle enough if the constraints that **i** and **j** be of equal length and orthogonal are applied (see [3] for a simple pose solution for three or more coplanar points). 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** which are proportional to **i** and **j** with the same scaling factor s. We construct a *goodness measure* G, for example as

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

The goodness measure G becomes large when the results are poor and can be used for quickly testing the quality of the computed pose and for detecting wrong correspondences between image points and object points.

The POS algorithm provides a computationally inexpensive method for directly obtaining the translation and rotation of an object; the accuracy of POS may be sufficient for tracking the motions of an object in space, finding initial estimates for iterative methods, or testing whether image and object points can be matched. Furthermore, when an object is far from the camera, it is useless to try to improve on the pose found by POS.

5 From Approximate Pose to Exact Pose: The POSIT Algorithm

5.1 Basic Idea

In this section, we present an iterative algorithm, POSIT (POS with Iterations) , which uses POS at each iteration. Less than five iterations are typically sufficient. The basic idea for iterating toward a more accurate pose is the following:

If we could build an SOP image of the object feature points from a TPP image, we could apply the POS algorithm to this SOP image and we would obtain an exact object pose.

Computing an exact SOP image requires knowing the exact pose of the object. However, once we have applied POS to the actual image, we have an approximate depth for each feature point, and we position the feature points at these depths *on the lines of sight.* Then we can compute an SOP image. At the next iteration, we apply POS to the SOP image to find an improved SOP image. The algorithm generally converges after a few iterations and provides an accurate SOP image and an exact pose.

5.2 Finding an SOP image from a TPP image

Eq. (1) and Eq. (2) show that the SOP vector $\mathbf{Cp}_i$ is aligned with the TPP vector $\mathbf{Cm}_i$ and the proportionality factor is Z_i/Z_0:

$$\mathbf{Cp}_i = \frac{Z_i}{Z_0}\mathbf{Cm}_i \tag{7}$$

The coordinates Z_i can be computed by

$$Z_i = Z_0 + \mathbf{k} \cdot \mathbf{M_0M}_i \tag{8}$$

where $\mathbf{k}$ is the unit vector along the optical axis Oz. Expressed in the object coordinate system, $\mathbf{k}$ is the third row of the rotation matrix of the object, and $\mathbf{M_0M}_i$ is a known vector. Eq. (7) and Eq. (8) yield for the SOP image points p_i

$$\mathbf{Cp}_i = (1 + \frac{s}{f}(\mathbf{k} \cdot \mathbf{M_0M}_i))\mathbf{Cm}_i \tag{9}$$

where we have replaced $1/Z_0$ by s/f, the ratio of the scaling factor of the SOP by the camera focal length.

Expression (9) provides at each iteration of the POSIT algorithm the approximated positions of the SOP image points p_i in relation to the image points m_i if we use the third row of the computed rotation matrix and the computed scaling factor.

6 Illustration of the Iteration Process in POSIT

To illustrate the iteration process of POSIT, we apply the method to synthetic data. The object is a cube; the points of interest are the eight corners (one can easily experiment with eight visible corners using light emitting diodes). The projection on the left of Fig. 2 is the given image for the cube (the shown projections of the cube edges are not used by the algorithm). The distance-to-size ratio for the cube is small, thus some parallel cube edges show strong convergence in the image. One can get an idea of the success of the POS algorithm by computing TPP image of the cube at the found poses at successive iterations (Fig. 2, top row). Notice that from left to right these projections become more similar to the given image. POSIT does not compute these images. Instead, POSIT computes SOP images using Eq. (9 (Fig. 2, bottom row). Notice that from left to right the edges of the cube become more parallel in these SOP images, since orthographic projection preserves parallelism.

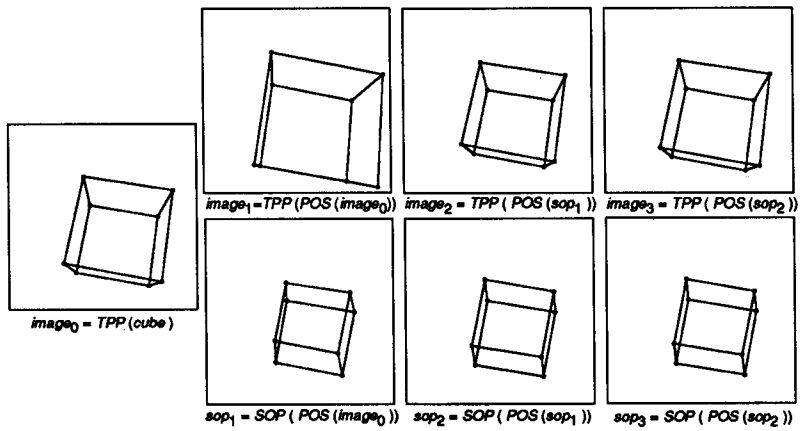

Fig. 2. TPP images (top) and SOP images (bottom) for cube poses computed at successive steps by POSIT algorithm.

7 Performance Characterization

We try to follow the recommendations of Haralick for performance evaluation in computer vision [4]. We compute the orientation and position errors of the POS and POSIT algorithms for a cube (see [3] for more experiments). One corner of the cube is taken as the reference point and is allowed to slide along the optical axis at 10 distances from the center of projection, from four times to 40 times the size of the cube. These distance-to-size ratios are used as the horizontal coordinates in the error plots. Around each of these reference point positions, the cube can swivel at 40 random orientations.

We obtain synthetic images by perspective projection with a focal length of 760 pixels. Note that only a wide-angle camera with a total angular field of more than 50° would be able to see the whole cube when it is closest to the camera. We specify three levels of random perturbation and noise in the image. At noise level 1, the computed image coordinates are rounded to integer values. At noise level 2, perturbations of $\pm$ 1 pixel are added to the image coordinates. At noise level 3, the amplitudes of the perturbations are $\pm$ 2 pixels.

For each of the images, the orientation and position of the object are computed by the POS algorithm, then by the POSIT algorithm until it converges. We then 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 [3]. The *relative 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. For both POS and POSIT, we show the average errors with their standard deviation error bars as a function of the distance-to-size ratios (Fig. 3).

8 Results

At very low to medium range and low to medium noise, POSIT gives poses with less than 2° rotation errors and less than 2% position errors. POSIT provides dramatic im-

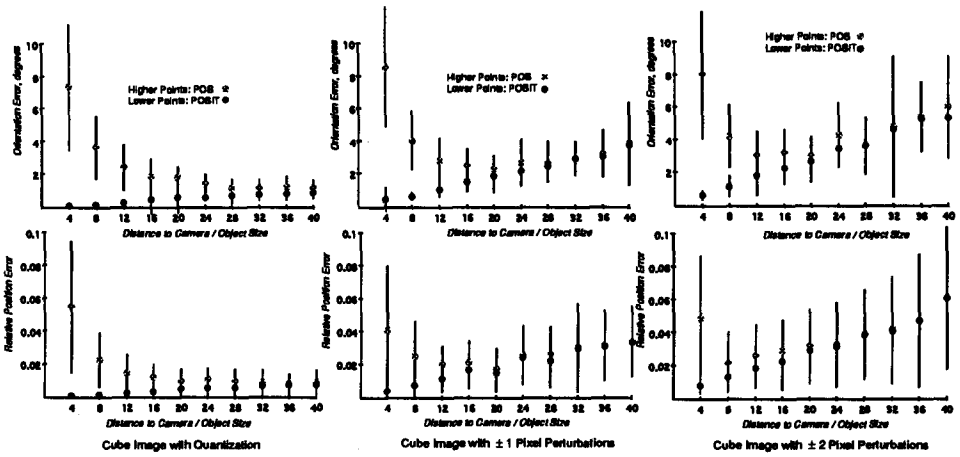

Fig. 3. Orientation and position errors for a cube at various distances at three image noise levels.

provements over POS when the objects are very close to the camera, and almost no improvements when the objects are far from the camera. When the objects are close to the camera, the so-called *perspective distortions* are large, and the approximation that the image is an SOP is poor; therefore the performance of POS is poor. When the objects are very far, there is almost no difference between SOP and TPP; thus POS gives the best possible results, and iterating with POSIT cannot improve upon them. Also, when the object is far, pose errors increase with the distance ratios, since at long range perturbations of a few pixels are a large percentage of the image size.

9 Convergence Analysis

We now explore with simulations the effect of the distance of an object to the camera on the convergence of the POSIT algorithm (Fig. 4). The convergence test 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). A cube is displaced along the camera optical axis. One face is kept parallel to the image plane. The abscissa in the plots is the distance from the center of projection to that face, in cube size units. Noise of $\pm$ 2 pixels is added to the perspective projection. Four iterations are required for convergence until the cube is at three times its size from the center of projection. The number gradually climbs to eight iterations for a distance of 1, and 20 iterations for 0.5. Then the number increases sharply to 100 iterations for a distance ratio of 0.28 from the center of projection. 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 are subjected to somewhat random variations from iteration to iteration 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. 4. Finally, when the distance ratio becomes less than 0.12, 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°, i.e. a focal

length of less than 1.5 mm for a 10 mm CCD chip, an improbable configuration. In all our experiments, the POSIT algorithm has been reliably converging in a few iterations in the range of *practical* camera and object configurations. We are in the process of analyzing the convergence process by analytical means, but so far have succeeded only for objects and orientations chosen to yield simple expressions. Convergence seems to be guaranteed if the image features are at a distance from the image center shorter than the focal length.

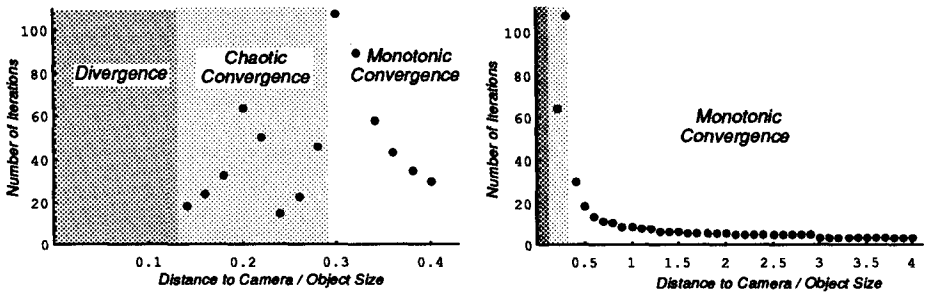

Fig. 4. Number of iterations as a function of distance to camera at very close ranges (left) and for a wider range of distances (right).

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, firstPose, 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 = (# - 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 = (# - 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;
   If[count==0, firstPose = {rotation, translation}];
   converged = (count>0) && (imageDifference<1);
   count++
];    (* End While *)
Return[{firstPose,{rotation, translation}}]]

(* Example of input:*)

fLength = 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}};

{{POSRot,POSTrans},{POSITRot,POSITTrans}} =
GetPOSIT[cubeImage, cube, cubeMatrix, fLength];
```

References

1. H.S. Baird, "*Model-Bases Image Matching Using Location*", MIT Press, Cambridge, MA, 1985.
2. T.A. Cass, "Feature Matching for Object Localization in the Presence of Uncertainty", MIT A.I. Memo 1113, May 1990.
3. D. DeMenthon and L.S. Davis, "Model-Based Object Pose in 25 Lines of Code", Center for Automation Research Technical Report CAR-TR-599, December 1991.
4. R.M. Haralick, "Performance Characterization in Computer Vision", University of Washington C.S. Technical Report, July 1991.
5. R. Horaud, B. Conio and O. Leboulleux, "An Analytical Solution for the Perspective-4-Point Problem", Computer Vision, Graphics, and Image Processing, vol. 47, pp. 33–44, 1989.
6. C. Tomasi, "Shape and Motion from Image Streams: A Factorization Method", Technical Report CMU-CS-91-172, Carnegie Mellon University, September 1991.
7. R.Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," IEEE J. Robotics and Automation, vol. 3, pp. 323–344, 1987.
8. S. Ullman and R. Basri, "Recognition by Linear Combinations of Models", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 13, pp. 992-1006, 1991.
9. J.S.C. Yuan, "A General Photogrammetric Method for Determining Object Position and Orientation", IEEE Trans. on Robotics and Automation, vol. 5, pp. 129–142, 1989.

This article was processed using the LaTeX macro package with ECCV92 style