CAR-TR-969 EAR-99-05844
CS-TR-4257 May 2001

SoftPOSIT: An Algorithm for Registration of 3D Models to Noisy Perspective Images Combining Softassign and POSIT

[1]Daniel DeMenthon, [2]Phil David and [1]Hanan Samet

[1]Center for Automation Research
University of Maryland
College Park, MD 20742-3275
daniel@cfar.umd.edu
[2]Army Research Laboratory
2800 Powder Mill Road, Adelphi, MD 20783-1197

Abstract

We propose an algorithm, *SoftPOSIT*, for automatic model-to-image registration. This algorithm combines two techniques, (1) a solution to the correspondence problem by an iterative technique called *Softassign*, and (2) a solution to the pose problem by an iterative technique called *POSIT*. These two techniques are combined into a single iteration loop. The present report focuses on the description of the algorithm. Results of a performance evaluation obtained from Monte Carlo simulations under a variety of levels of clutter, occlusion, and image noise will be presented in a forthcoming report.

The support of this research by the National Science Foundation under contracts EAR-99-05844 and IIS-00-8-6162 is gratefully acknowledged.

1 Introduction

One of the goals of this research is to automatically update city models using cameras mounted on vehicles. Examples of useful city model updates would be the automatic addition of texture maps to simple wireframe models, or the update of damaged buildings after earthquakes. This task requires a very accurate alignment between real-life images and synthetic images from the city model, i.e. a very accurate camera pose estimate. Otherwise, updates obtained from video streams would result in the deterioration of the original city model instead of its improvement. With a typical 30 degree camera field of view, an error of 2 degrees in the estimate of the camera rotation would create a shift of around 40 pixels between what we expect to see from the city model and what we actually see. An approximate camera pose can be measured by navigation sensors. A GPS sensor can provide two horizontal translation components of the camera. The pan angle of the camera can be estimated from the motion direction of the vehicle along a trajectory obtained by tracing the GPS positions. This pan estimate can be improved by a compass measurement. The roll and tilt angles of a vehicle camera are affected by braking and acceleration, road profile (including potholes), and passenger load variations. Pan, tilt and roll estimates can be estimated by inertial navigation systems (INS) comprising gyroscopes and accelerometers, but such systems are subject to drift and must be periodically reset to accurate absolute measurements. These measurements can be obtained using an automatic registration between video stream data and synthetic data from the city model. This registration task is the focus of this report.

Other applications of this research include autonomous navigation in urban environments when a model of the city is available, and, importantly, object recognition.

Automatic registration of 3D models to images is a difficult problem. The main reason is that registration comprises two coupled problems, the correspondence problem and the pose problem, each easy to solve only if the other has been solved first:

1. Solving the correspondence problem consists of finding matching image features and model features. If the camera pose is known, one can relatively easily determine the matching features. Looking at the model from the point of view defined by the camera pose, one defines as matching features the model features that are observed to be close to the features of the new image from that point of view.

2. Solving the pose problem consists of finding the rotation and translation of the camera with respect to the model. Given matching image and model features one can easily determine the pose that best aligns those matches.

The classic approach to solving these two coupled problems has been a hypothesize-and-test approach, in which (a) guesses are made about matches between a few image features and a few model features, (b) the camera pose is computed using these matches, and (c) the remaining model features are projected on the image and a measure of the quality of their superposition with image features is computed. This process is repeated for all reasonable groups of matching guesses, and the correct camera pose and superposition are chosen from among those that provide the highest measure of superposition [12]. However, this type of approach is combinatorial and generally expensive for complex models and images.

The SoftPOSIT algorithm combines two techniques to simultaneously solve the correspondence and pose problems:

1. a solution to the correspondence problem by an iterative technique called Softassign developed by Gold, Rangarajan and others [11] and having its roots in improvements of the original Hopfield-Tank neural network framework;

2. a solution to the pose problem by an iterative technique called POSIT (Pose from Orthography and Scaling with ITerations) developed by DeMenthon and Davis [7], who showed that the pose problem for perspective projection can be solved by iteratively refining the pose computed for scaled orthographic projection.

These two techniques are combined into a single iteration loop. A global objective function is defined that captures the nature of the problem in terms of both pose and correspondence and combines the formalisms of both iterative techniques. The correspondence and the pose are determined *simultaneously* by minimizing this global objective function. This objective function is minimized when both the correct pose and an optimal correspondence are found. The expressions for this objective function and its minimization are closely related to those proposed by Gold et al. [11] when they successfully solved the correspondence and pose problem for matching two images or two 3D models. However, their work does not address the more difficult problem of registration between a 3D model and its perspective image. Solving this problem is the subject of this report. In the remainder of this introduction, we give an overview of the steps of our method.

1.1 Correspondence Problem

Following the framework introduced in [11], the correspondence between image features and model features in the global objective function is expressed as a *correspondence matrix* $\mathbf{M}_0$ in which each term m_{jk} is a *weight* for the match between an image point p_j and a model point P_k. This weight is a function of the distance d_{jk} between the scaled orthographic projection of the model point P_k and the scaled orthographic projection after that model point P_k is displaced onto the line of sight of image point p_j (for a camera pose that is refined during the iteration process). This weight is equal to one if P_k falls on the line of sight of p_j, and less than one otherwise. There are additional weights in a slack column and a slack row which can become equal to one in case of *no-match* for image points or model points. At the beginning of the iteration, the matrix M_0 is a *fuzzy* correspondence matrix which permits partial matches, with all weights lower than one, but with the sums of the weights always adding up to one in each row and each column. The degree of fuzziness is controlled by a parameter β. As the iteration proceeds, the correspondence matrix is hardened into an *assignment matrix* M by a deterministic annealing process obtained by increasing the parameter β. In this assignment matrix, the weight for one match has become equal to one for each column and for each row, and the other weights have become zero, in a winner-take-all manner. The winning match for point p_j at row j is the point P_k with the smallest distance d_{jk}.

1.2 Pose Problem

Scaled orthographic projection (SOP) is an approximation of perspective projection that has become popular in the computer vision community because it allows one to solve the pose problem with linear algebra and with fewer unknowns than with projective geometry. However, with this approximation, parallel lines in a model will always produce parallel lines in images. When the size of an object or a scene is large compared to its distance from the camera, strong convergence of parallel lines can be observed in the image, and registration results between a model of the world and its image are difficult with this approximation.

The POSIT algorithm preserves the computational simplicity of the pose problem with the SOP model of projection, while obtaining the correct pose corresponding to full perspective projection. This result is achieved by observing that pose computation using an SOP model *can* provide an exact pose, provided it uses image points obtained by SOP. Therefore, after an approximate pose has been found for the image frame using SOP, the locations of the image points can be "corrected" into points close to SOP image points to offset the errors of this linear solution at the next iteration step.

1.3 Merging Correspondence and Pose Computations

The proposed global objective function expresses the sum of the squared distances between all corrected image feature points and all SOP projections of the feature points of the model, as seen from the (unknown) camera pose, each distance being *weighted* by the qualities of the matches between image and model points. These weights are the terms of the correspondence matrix. This objective function is positive or null, with a minimum at zero only when all the weighted distances are zero. This function is minimized by an iteration loop. Each iteration step of the loop comprises three phases:

1. The correspondence variables of the objective function are optimized assuming the pose to be fixed.

2. The pose variables are optimized assuming the correspondence variables to be fixed.

3. The positions of the image feature points are corrected in a way that will offset the effects of the SOP approximation at the next iteration step.

Fig. 1 shows an example of a computation result obtained by SoftPOSIT for a 15-point model. One point of the model was not detected in the image, and two detected image points were noise. Fig. 2 shows the evolution of the projections of a cube as its pose computation evolves toward the correct pose.

In the following sections, we examine each step of the method in detail; we then provide pseudocode for the algorithm. In the appendix, we present a Matlab implementation and an example of input and output.

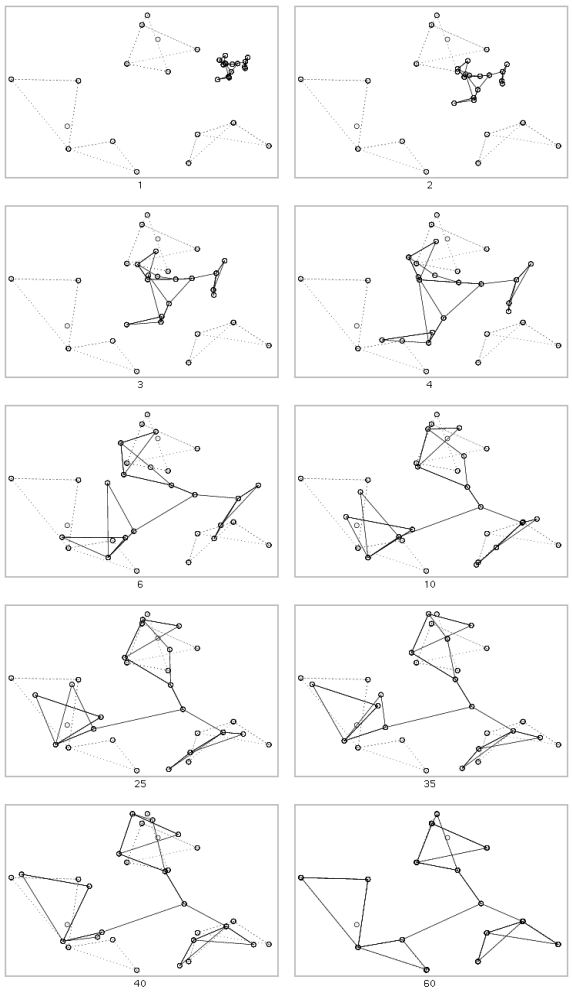

Figure 1: Evolution of perspective projections for a 15-point object (solid lines) being aligned by the SoftPOSIT algorithm onto an image(dashed lines) with one missing point and two clutter points.

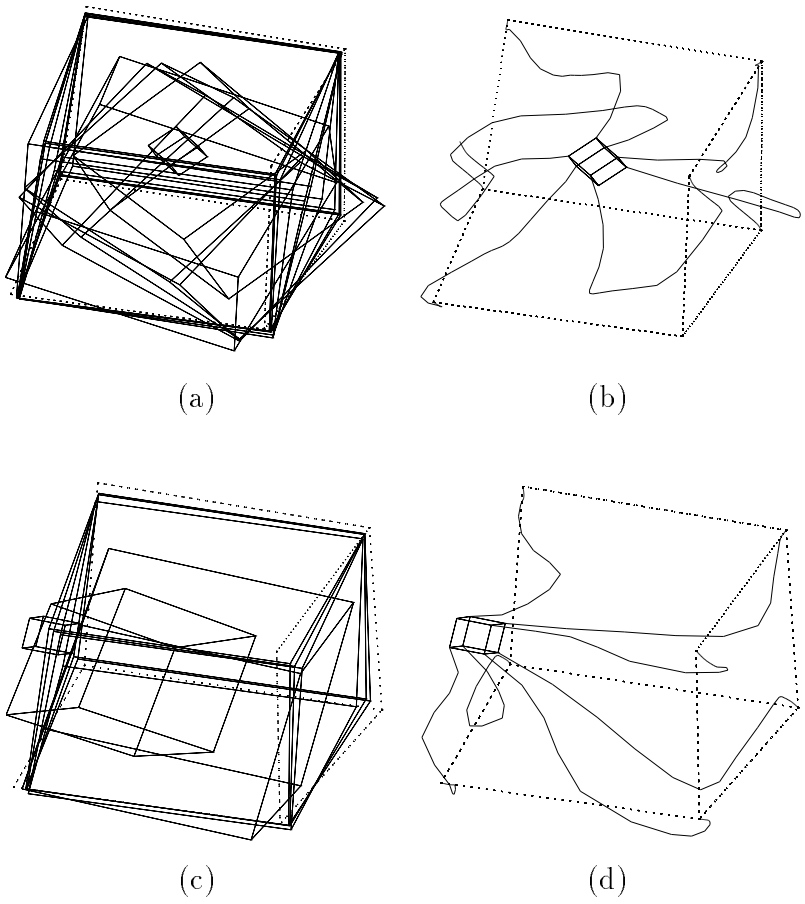

Figure 2: Evolution of the perspective projections of a cube starting from random initial poses (small cube with solid lines) onto an image with one missing point (dashed lines). The two left images show intermediary cube projections as they move toward the image points. The two right images show the trajectories of the projections of the cube vertices as they move toward the image points.

2 A Projective Geometry Formulation of POSIT

One of the building blocks of the new algorithm is the POSIT algorithm, first described in [6], and presented in detail in [7] and in projective geometry form in [8]. We summarize this algorithm below, and then present a variant of the algorithm using the closed form minimization of an objective function. It is this objective function which is modified to analytically characterize the global pose-correspondence problem in a single equation.

Consider a pinhole camera of focal length f and an image feature point p with its two Euclidean coordinates x and y and its three homogeneous coordinates (wx, wy, w). This point p is the perspective projection of the 3D point P with homogeneous coordinates $(X, Y, Z, 1)$ in the frame of reference of an object.

In our problem, there is an unknown coordinate transformation between the object and the camera, represented by a rotation matrix $R = [\mathbf{R}_1\ \mathbf{R}_2\ \mathbf{R}_3]^T$ and a translation vector $\mathbf{T} = (T_x, T_y, T_z)$: $\mathbf{P}_c = R\,\mathbf{P} + \mathbf{T}$, where $\mathbf{P}$ and $\mathbf{P}_c$ correspond to the same point expressed in the object and the camera coordinate systems, respectively. The vectors $\mathbf{R}_1^T$, $\mathbf{R}_2^T$, $\mathbf{R}_3^T$ are the row vectors of the rotation matrix; they are the unit vectors of the camera coordinate system expressed in the model coordinate system. The translation vector $\mathbf{T}$ is the vector from the center of projection O of the camera to the origin P_0 of the object expressed in the camera coordinate system. Therefore the coordinates of the perspective projection p are related to the coordinates of the world point P by

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1^T & T_x \\ \mathbf{R}_2^T & T_y \\ \mathbf{R}_3^T & T_z \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

or

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} f\mathbf{R}_1^T & fT_x \\ f\mathbf{R}_2^T & fT_y \\ \mathbf{R}_3^T & T_z \end{bmatrix} \begin{bmatrix} \mathbf{P_0P} \\ 1 \end{bmatrix},$$

where $\mathbf{P_0P} = (X, Y, Z)$. The homogeneous image point coordinates are defined up to a multiplicative constant; therefore the validity of the equality is not affected if we multiply all the elements of the perspective projection matrix by the same term $1/T_z$. We also make use of the scaling factor $s = f/T_z$ (the reason for this appellation becomes clear below). We obtain

$$\begin{bmatrix} wx \\ wy \end{bmatrix} = \begin{bmatrix} s\mathbf{R}_1^T & sT_x \\ s\mathbf{R}_2^T & sT_y \end{bmatrix} \begin{bmatrix} \mathbf{P_0P} \\ 1 \end{bmatrix}. \tag{1}$$

with

$$w = \mathbf{R}_3 \cdot \mathbf{P_0P}/T_z + 1. \tag{2}$$

In the expression for w the dot product $\mathbf{R}_3 \cdot \mathbf{P_0P}$ represents the projection of the vector $\mathbf{P_0P}$ (from the origin of the model to its point P) onto the optical axis of the camera. Indeed, in the world coordinate system where P is defined, $\mathbf{R}_3$ is the unit vector of the optical axis. When

the depth range of the model along the optical axis of the camera is small with respect to the model distance, $\mathbf{R}_3 \cdot \mathbf{P_0 P}$ is small with respect to T_z, and therefore w is close to one. In this case, perspective projection gives results that are similar to the following transformation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s\mathbf{R}_1^T & sT_x \\ s\mathbf{R}_2^T & sT_y \end{bmatrix} \begin{bmatrix} \mathbf{P_0 P} \\ 1 \end{bmatrix}. \tag{3}$$

This expression defines the *scaled orthographic* projection p' of the 3D point P. The factor s is the scaling factor of this scaled orthographic projection. If $s = 1$, note that this equation expresses (1) a transformation of world point components from a world coordinate system to a camera coordinate system, and (2) the use of two of the three world point coordinates as image coordinates: this is the definition of a pure orthographic projection. With a factor s different from one, this image is scaled and approximates a perspective image because the scaling is inversely proportional to the distance T_z from the camera center of projection to the object origin P_0 ($s = f/T_z$).

Returning to the general perspective equations (1, 2), we can rewrite the first equation as

$$\begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} s\mathbf{R}_1 & s\mathbf{R}_2 \\ sT_x & sT_y \end{bmatrix} = \begin{bmatrix} wx & wy \end{bmatrix}. \tag{4}$$

Assume that for each image point p with coordinates x and y the corresponding homogeneous coordinate w has been computed at a previous computation step and is known. Then we are able to calculate wx and wy, and the previous equation expresses the relationship between the unknown pose components $s\mathbf{R}_1$, $s\mathbf{R}_2$, sT_x, sT_y, and the known image components wx and wy and known world coordinates X, Y, Z. If we know K world points P_k and their K corresponding image points p_k *and their homogeneous components* w_k, we can then write two linear systems of equations

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \vdots & \vdots & \vdots & \\ X_K & Y_K & Z_K & 1 \end{bmatrix} \begin{bmatrix} s\mathbf{R}_1 & s\mathbf{R}_2 \\ sT_x & sT_y \end{bmatrix} = \begin{bmatrix} w_1x_1 & w_1y_1 \\ w_2x_2 & w_2y_2 \\ \vdots & \vdots \\ w_Kx_K & w_Ky_K \end{bmatrix}. \tag{5}$$

This set of two linear systems can be solved for the unknown components of vectors $s\mathbf{R}_1$, $s\mathbf{R}_2$ and the unknowns sT_x and sT_y, provided the rank of the matrix of world point coordinates is at least 4. Thus, at least four of the points of the model for which we use the image points must be noncoplanar. If this is the case, we can find a least square solution for the set of unknowns by using the pseudoinverse A' of the matrix A of model points:

$$\begin{bmatrix} s\mathbf{R}_1 & s\mathbf{R}_2 \\ sT_x & sT_y \end{bmatrix} = A' \begin{bmatrix} w_1x_1 & w_1y_1 \\ w_2x_2 & w_2y_2 \\ \vdots & \vdots \\ w_Kx_K & w_Ky_K \end{bmatrix}, \tag{6}$$

with

$$A = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \vdots & \vdots & \vdots & \\ X_K & Y_K & Z_K & 1 \end{bmatrix} \tag{7}$$

We call A the object matrix, as it depends only on the point configuration in the 3D model. The pseudoinverse A' of A can be precomputed when the model is given.

After the unknown $s\mathbf{R}_1$ and $s\mathbf{R}_2$ are obtained, we can extract s, $\mathbf{R}_1$ and $\mathbf{R}_2$, by imposing the condition that $\mathbf{R}_1$ and $\mathbf{R}_2$ must be unit vectors. Then we can obtain $\mathbf{R}_3$ as the cross-product of $\mathbf{R}_1$ and $\mathbf{R}_2$:

$$s = (|s\mathbf{R}_1|\,|s\mathbf{R}_2|)^{1/2} \text{ (geometric mean)}, \tag{8}$$

$$\mathbf{R}_1 = (s\mathbf{R}_1)/s, \tag{9}$$

$$\mathbf{R}_2 = (s\mathbf{R}_2)/s, \tag{10}$$

$$\mathbf{R}_3 = \mathbf{R}_1 \times \mathbf{R}_2, \tag{11}$$

$$T_x = (sT_x)/s;\ T_y = (sT_y)/s;\ T_z = f/s. \tag{12}$$

An additional intermediary step that improves performance and quality of results consists of using the unit vectors $\mathbf{R}'_1$ and $\mathbf{R}'_2$ that are mutually perpendicular and closest to $\mathbf{R}_1$ and $\mathbf{R}_2$ in the least square sense. These vectors can be found by Singular Value Decomposition (see Matlab code in the appendix).

How can we compute the w_k components required to compute the right-hand side rows $(w_k x_k, w_k y_k)$ corresponding to image point p_k $(1 \leq k \leq K)$? Setting $w_k = 1$ for every point is a good first step because it amounts to solving the problem with a scaled orthographic model of projection as we saw above. Once we have the pose result for this first step, we can compute better estimates for w_k as

$$w_k = \mathbf{R}_3 \cdot \mathbf{P_0 P}_k / T_z + 1. \tag{13}$$

as shown by Eq. (2). Then we can solve the linear systems of Eq. (5) again to obtain a refined pose. This process is repeated, and the iteration is stopped when the process becomes stationary.

We now look at a geometric interpretation of this method in order to propose a variant using an objective function. Consider (Fig. 3) the center of projection O of the camera, its axis Oz, and its image plane Π at distance f from O. The image center (principal point) is called c. Consider an object, the origin P_0 of its coordinate system, a point P of this object, and a corresponding image point p. The line of sight corresponding to the image point p is called L. The goal of pose calculation when correspondence is known is to move the object so as to place all the points P on the object as close as possible to the lines of sight L of the corresponding image points p.

A plane Π' parallel to the image plane Π is chosen to pass through the origin P_0 of the object coordinate system. This plane cuts the camera axis in H $(OH = T_z)$. The point P projects on plane Π' in P', and the image of P' on the image plane Π is called p'.

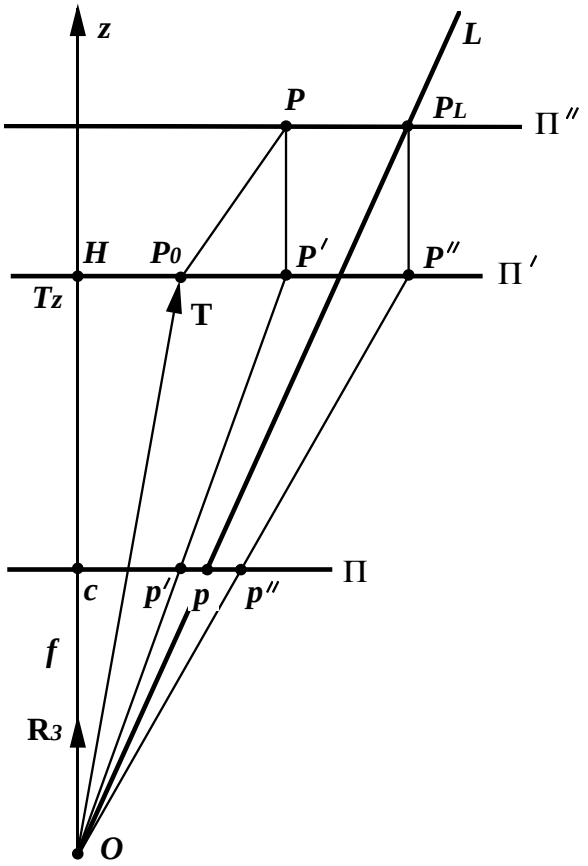

Figure 3: Geometric interpretation of POSIT computation. The image point p', scaled orthographic projection of world point P, is computed by one side of the POSIT equations. The image point p'', scaled orthographic projection of point P_L on the line of sight of p, is computed by the other side of the equation. The equations are satisfied when the two points are superposed, which requires that world point P be on the line of sight of image point p. The plane of the figure is chosen to contain the plane of the optical axis and the line of sight L. The points P_0, P, P' and p' are generally out of that plane.

A plane Π'' also parallel to the image plane Π passes through point P and cuts the line of sight L at P_L. The point P_L projects onto the plane Π' at P'', and the image of P'' on the image plane Π is called p''.

The plane defined by line L and the camera axis is chosen as the plane of the figure. Therefore, the image points p and p'' are also in the plane of the figure. Generally P_0 and P are out of the plane of the figure, and therefore p' is also out of the plane of the figure.

Consider again the equations of perspective (Eqs. (1, 2)):

$$\begin{bmatrix} wx \\ wy \end{bmatrix} = \begin{bmatrix} s\mathbf{R}_1^T & sT_x \\ s\mathbf{R}_2^T & sT_y \end{bmatrix} \begin{bmatrix} \mathbf{P_0P} \\ 1 \end{bmatrix}. \tag{14}$$

with $w = \mathbf{R}_3 \cdot \mathbf{P_0P}/T_z + 1$. We can see that $\mathbf{cp'} = s(\mathbf{R}_1 \cdot \mathbf{P_0P} + T_x, \mathbf{R}_2 \cdot \mathbf{P_0P} + T_y)$. Indeed the terms in parentheses are the x and y camera coordinates of P and therefore also of P', and the scaling factor s scales down these coordinates to those of the image p' of P'. In other words, the column vector of the right-hand side of Eq. (14) represents the vector $\mathbf{cp'}$ in the image plane.

On the other hand, $\mathbf{cp''} = (wx, wy) = w\mathbf{cp}$. Indeed the z-coordinate of P in the camera coordinate system is $\mathbf{R}_3 \cdot \mathbf{P_0P} + T_z$, i.e. wT_z. It is also the z-coordinate of P_L. Therefore $\mathbf{OP_L} = wT_z\mathbf{Op}/f$. The x and y camera coordinates of P_L are also those of P'', and the scaling factor $s = f/T_z$ scales down these coordinates to those of the image p'' of P''. Thus $\mathbf{cp''} = w\mathbf{cp}$. In other words, the column vector of the left-hand side of Eq. (14) represents the vector $\mathbf{cp''}$ in the image plane. The image point p'' can be interpreted as a correction of the image point p from a perspective projection to a scaled orthographic projection of a point P_L located on the line of sight at the same distance as P.

When P is on the line of sight L of p, P and P_L are superposed, the image points p' and p'' are superposed, $\mathbf{cp''} = \mathbf{cp'}$, and Eq. (14) is satisfied. Conversely, if Eq.(14) is satisfied, then $\mathbf{cp''} = \mathbf{cp'}$, P and P_L are superposed, and therefore P is on the line of sight of the image p.

When we try to match the points P_k of an object to the lines of sight L_k of image points p_k, it is unlikely that all or even any of the points will fall on their corresponding lines of sight, or equivalently that $\mathbf{cp''}_k = \mathbf{cp'}_k$ or $\mathbf{p'}_k\mathbf{p''}_k = \mathbf{0}$. The solution to the linear systems of Eqs. (5) is a least square optimization of the set of constraints $\mathbf{p'}_k\mathbf{p''}_k = \mathbf{0}$.

Alternatively, we can minimize a global objective function E equal to the sum of the squared distances $d_k^2 = \mathbf{p'}_k\mathbf{p''}_k^2$ between image points $\mathbf{p'}_k$ and $\mathbf{p''}_k$:

$$E = \sum_k d_k^2 = \sum_k ((\mathbf{M} \cdot \mathbf{S}_k - w_k x_k)^2 + (\mathbf{N} \cdot \mathbf{S}_k - w_k y_k)^2), \tag{15}$$

where we introduce the vectors $\mathbf{M}$, $\mathbf{N}$, and $\mathbf{S}_k$ with four homogeneous coordinates to simplify the subsequent notation.

$$\begin{aligned} \mathbf{M} &= (M_1, M_2, M_3, M_4) = s(\mathbf{R}_1, T_x), \\ \mathbf{N} &= (N_1, N_2, N_3, N_4) = s(\mathbf{R}_2, T_y), \\ \mathbf{S}_k &= (\mathbf{P_0P_k}, 1). \end{aligned}$$

We call $\mathbf{M}$ and $\mathbf{N}$ the *pose vectors*.

Note, referring again to Fig. 3, that $\mathbf{p}'\mathbf{p}'' = s\mathbf{P}'\mathbf{P}'' = s\mathbf{P}\mathbf{P}_L$. Therefore minimizing this objective function consists of minimizing the scaled sum of distances of model points to lines of sight, when distances are taken along directions parallel to the image plane.

This objective function is minimized iteratively. Initially, the w_k are all set to one. Then the following two operations take place at each iteration step:

1. Optimize the pose vectors $\mathbf{M}$ and $\mathbf{N}$ assuming the terms w_k are known .

2. Compute the correction terms w_k using the pose vectors $\mathbf{M}$ and $\mathbf{N}$ just computed (Eq.(2)).

We now focus on the optimization of the pose vectors $\mathbf{M}$ and $\mathbf{N}$. The pose vectors that will minimize the objective function E at a given iteration step are those for which all the partial derivatives of the objective function with respect to the coordinates of these vectors are zero. This condition provides 4×4 linear systems for the coordinates of $\mathbf{M}$ and $\mathbf{N}$ whose solutions are

$$\mathbf{M} = (\sum_k \mathbf{S}_k \mathbf{S}_k^T)^{-1} (\sum_k w_k x_k \mathbf{S}_k), \tag{16}$$

$$\mathbf{N} = (\sum_k \mathbf{S}_k \mathbf{S}_k^T)^{-1} (\sum_k w_k y_k \mathbf{S}_k), \tag{17}$$

The matrix $L = \sum_k \mathbf{S}_k \mathbf{S}_k^T$ is a 4×4 matrix that can be precomputed.

With either method, the point p'' can be viewed as the image point p "corrected" for scaled orthographic projection using w computed at the previous step of the iteration. The next iteration step finds the pose such that the scaled orthographic projection of each point P is as close as possible to its corrected image point.

3 Pose Calculation with Unknown Correspondences

When correspondences are unknown, each image feature point p_j can potentially match any of the model feature points P_k, and therefore must be corrected using the value of w specific to the coordinates of P_k:

$$w_k = \mathbf{R}_3 \cdot \mathbf{P_0}\mathbf{P}_k / T_z + 1. \tag{18}$$

Therefore for each point p_j and each point P_k we generate a corrected image point p''_{jk}, aligned with the image center c and with p_j, and defined by

$$\mathbf{cp}''_{jk} = w_k \mathbf{cp}_j. \tag{19}$$

We make use of the squared distances between these corrected image points p''_{jk} and the scaled orthographic perspective projections p'_k of the points P_k whose positions are provided by

$$\mathbf{cp}'_k = \begin{bmatrix} \mathbf{M} \cdot \mathbf{S}_k \\ \mathbf{N} \cdot \mathbf{S}_k \end{bmatrix}. \tag{20}$$

These squared distances are

$$d_{jk}^2 = (\mathbf{p}''_{jk}\mathbf{p}'_k)^2 = (\mathbf{M}\cdot\mathbf{S}_k - w_k x_j)^2 + (\mathbf{N}\cdot\mathbf{S}_k - w_k y_j)^2, \tag{21}$$

where x_j and y_j are the image coordinates of the image point p_j, $\mathbf{S}_k$ is the vector $(S_{k1}, S_{k2}, S_{k3}, S_{k4}) = (\mathbf{P_0 P_k}, 1)$, and $\mathbf{M}$ and $\mathbf{N}$ are pose vectors introduced in the previous section and recomputed at each iteration step. The term w_k is defined by Eq. (18).

The simultaneous pose and correspondence problem can be formulated as a minimization of the global objective function

$$E = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}\, d_{jk}^2 = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}((\mathbf{M}\cdot\mathbf{S}_k - w_k x_j)^2 + (\mathbf{N}\cdot\mathbf{S}_k - w_k y_j)^2), \tag{22}$$

where the m_{jk} are weights for each of the squared distances d_{jk}^2, and J and K are the number of image and model points, respectively. They are correspondence variables and define the assignments between image and model feature points. Note that when all the assignments are well-defined, this objective function becomes equivalent to the objective function defined in Eq. (15).

This objective function is minimized iteratively, with the following three operations at each iteration step:

1. Optimize the correspondence variables assuming everything else is fixed (Section 3.2).
2. Optimize the pose vectors $\mathbf{M}$ and $\mathbf{N}$ assuming everything else is fixed (Eqs. 23 and 24 in Section 3.1).
3. Compute the correction terms w_k using the pose vectors $\mathbf{M}$ and $\mathbf{N}$ just computed (Eqs. 18 and 8–12).

3.1 Pose Problem

We now focus on the optimization of the pose vectors $\mathbf{M}$ and $\mathbf{N}$. As in the previous section, the pose vectors that will minimize the objective function E at a given iteration step are those for which all the partial derivatives of the objective function with respect to the coordinates of these vectors are zero. This condition provides 4×4 linear systems for the coordinates of $\mathbf{M}$ and $\mathbf{N}$ whose solutions are

$$\mathbf{M} = (\sum_{k=1}^{K} m'_k \mathbf{S}_k \mathbf{S}_k^T)^{-1} (\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk} w_k x_j \mathbf{S}_k), \tag{23}$$

$$\mathbf{N} = (\sum_{k=1}^{K} m'_k \mathbf{S}_k \mathbf{S}_k^T)^{-1} (\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk} w_k y_j \mathbf{S}_k), \tag{24}$$

with $m'_k = \sum_{j=1}^{J} m_{jk}$. The terms $\mathbf{S}_k\mathbf{S}_k^T$ are 4×4 matrices. Therefore computing $\mathbf{M}$ and $\mathbf{N}$ requires the inversion of a single 4×4 matrix, $L = (\sum_{k=1}^{K} m'_k \mathbf{S}_k \mathbf{S}_k^T)$, a fairly inexpensive operation. Note that L cannot be precomputed because the term in slack row $J+1$ can be non-zero and therefore $m'_k = \sum_{j=1}^{J} m_{jk}$ is not a constant.

3.2 Correspondence Problem

We optimize the correspondence variables m_{jk} assuming that the parameters d_{jk}^2 in the expression for the objective function E are known and fixed. Our aim is to find a zero-one *assignment matrix*, $M = \{m_{jk}\}$, that explicitly specifies the matchings between a set of J image points and a set of K model points, and that minimizes the objective function E. This assignment matrix M has one row for each of the J image points p_j and one column for each of the K model points P_k. A slack row $J+1$ and a slack column $K+1$ are added. A one in the slack column $K+1$ at row j indicates that image point p_j has not found any match among the model features. A one in the slack row $J+1$ at column k indicates that the model point P_k is not seen in the image and does not match any image feature. *The objective function E will be minimum if the assignment matrix M matches image and model points with the smallest distances d_{jk}^2.* This problem can be solved by the iterative *Softassign* technique [11]. The iteration for the assignment matrix M is started with a matrix M_0 in which each element m_{jk}^0 is $\exp(-\beta(d_{jk}^2 - \alpha))$, with β very small, and all elements in the slack row and slack column are set to one. The parameter α acts as a threshold, determining when to select no match (slack assignment) over a match with distance d_{jk}. See [11] for an analytical justification. The continuous matrix M_0 converges toward the discrete matrix M due to two mechanisms that are used concurrently:

1. First, a technique due to Sinkhorn [19] is applied. When each row and column of a square correspondence matrix is normalized (several times, alternatingly) by the sum of the elements of that row or column respectively, the resulting matrix has positive elements with all *rows and columns summing to one.*

2. The term β is increased as the iteration proceeds. As each term of a row or column of M_0 is normalized by the sum of the terms of that row or column by the Sinkhorn technique, the terms m_{jk}^0 corresponding to the smaller d_{jk}^2 of the row or column tend to converge to one as β is increased, while the other terms converge to zero. This is a deterministic annealing process [10] known as *Softmax* [3]. This is a desirable behavior, since we in fact want to assign the correspondence to the match that has the smallest distance in a row or column.

This combination of deterministic annealing and Sinkhorn's technique in an iteration loop was called *Softassign* by Gold and Rangarajan [11] and was first used in [14]. The matrix M resulting from an iteration loop that comprises these two substeps is the assignment that minimizes the global objective function $E = \sum_{j=1}^{J} \sum_{k=1}^{K} m_{jk}\, d_{jk}^2$. As the following pseudocode shows, these two substeps are grafted into the iteration loop of SoftPOSIT, along with the substeps that optimize the pose and correct the image points by scaled orthographic distortions.

3.3 Pseudocode for SoftPOSIT

The SoftPOSIT algorithm can be summarized as follows:
Inputs:

1. A list of image feature points $\mathbf{p}_j = (x_j, y_j)$.

2. A list of world points $\mathbf{S}_k = (X_k, Y_k, Z_k, 1) = (\mathbf{P_0}\mathbf{P}_k, 1)$ in the object.

Initialize slack elements of assignment matrix M to one, β to β_0 (β_0 around 0.00005 if nothing is known about the pose, β_0 larger if an initial pose can be guessed)
Initialize pose vectors $\mathbf{M}$ and $\mathbf{N}$ using expected pose or a random pose within expected range.
Initialize $w_k = 1$.
Do A until $\beta > \beta_{final}$ (β_{final} around 0.5) (*Deterministic annealing loop*)

- Compute the squared distances $d_{jk}^2 = (\mathbf{M} \cdot \mathbf{S}_k - w_k x_j)^2 + (\mathbf{N} \cdot \mathbf{S}_k - w_k y_j)^2$
- Compute $m_{jk}^0 = \exp(-\beta(d_{jk}^2 - \alpha))$
- **Do B until** ΔM **small** (*Sinkhorn's method*)
  - Update matrix M by normalizing across all rows: $m_{jk}^1 = m_{jk}^0 / \sum_{k=1}^{K+1} m_{jk}^0$
  - Update matrix M by normalizing across all columns: $m_{jk}^0 = m_{jk}^1 / \sum_{j=1}^{J+1} m_{jk}^1$
- **End Do B**
- Compute 4×4 matrix $L = (\sum_{k=1}^{K} m'_k \mathbf{S}_k \mathbf{S}_k^T)$ with $m'_k = \sum_{j=1}^{J} m_{jk}$
- Compute L^{-1}
- Compute pose vector $\mathbf{M} = L^{-1}(\sum_{j=1}^{J} \sum_{k=1}^{K} m_{jk} w_k \, x_j \mathbf{S}_k)$
- Compute pose vector $\mathbf{N} = L^{-1}(\sum_{j=1}^{J} \sum_{k=1}^{K} m_{jk} w_k \, y_j \mathbf{S}_k)$
- Compute $s = |(M_1, \, M_2, \, , M_3)|$; $\mathbf{R}_1 = (M_1, M_2, M_3)/s$; $\mathbf{R}_2 = (N_1, N_2, N_3)/s$; $\mathbf{R}_3 = \mathbf{R}_1 \times \mathbf{R}_2$
- Compute $w_k = \mathbf{R}_3 \cdot \mathbf{P_0} \mathbf{P}_k / T_z + 1$
- $\beta = \beta_{update} \beta$ (β_{update} around 1.025)

End Do A
Outputs: A rotation matrix $R = [\mathbf{R}_1 \; \mathbf{R}_2 \; \mathbf{R}_3]^T$, a translation vector $\mathbf{T} = (T_x, T_y, T_z)$, and an assignment matrix $em\mathbf{M} = \{m_{jk}\}$ between the list of image points and the list of world points of the input.

4 Related Work

This work was directly inspired by recent work of Gold et al. [11], who demonstrated the power of the *Softassign* approach on two related but simpler problems — matching sets of 2D points using an affine transform (to allow for skew), and matching sets of 3D points while recovering a rigid body motion. They did not address the more complex problem of perspective mapping between 3D objects and 2D images that we consider here. This problem has remained the subject of very active research in two areas, automatic registration of 3D site models to images of scenes for detection of unusual activity or photogrammetry, and object recognition.

Efforts on automatic registration of video images and site models have mostly aimed at images in which camera distances are large compared to variations in elevation within the site, so that the site can be assumed to be planar [9]. The proposed approach is complementary to these efforts and will work best with images taken at relatively close range with full perspective effects due to variations in terrain elevations and differences in building levels.

The problem addressed here is one that is encountered when taking a model-based approach to the object recognition problem (the other main approach being the appearance-based approach in which multiple views of the object are compared to the image), and as such has received considerable attention. Baird's original tree-pruning method [1] was of exponential time complexity for unequal point sets. The alignment method of [20] is $O(J^4K^3 \log K)$. Geometric hashing [15, 12] applied to affine transforms is very slow for large data sets, being of high-order polynomial complexity.

Many methods hypothesize the model pose from the correspondences of a small number of model and image points, and then verify the pose by projecting the remaining model features and trying to match them to image features. In [8] we proposed a different approach using a binary search by bisection of pose boxes in two 4D spaces, extending the research of [1, 4, 2] on affine transforms, but it had high-order complexity. The approach taken by Jurie [13] was inspired by our work and belongs to the same family. An initial volume of pose space is guessed, and all the correspondences compatible with this volume are first taken into account. Then the pose volume is recursively reduced until it can be viewed as a single pose. As a Gaussian error model is used, boxes of pose space are pruned not by counting the number of correspondences that are compatible with the box as in [8], but on the basis of the probability of having an object model in the image within the range of poses defined by the box.

All the methods mentioned so far settle for affine approximations of perspective. Few researchers have addressed the full perspective problem. Among them, Wunsch and Hirzinger [21] formalize the abstract problem in a way similar to the approach advocated here as the optimization of an objective function combining correspondence and pose constraints. However, the correspondence constraints are not represented analytically. Instead, each model feature is explicitly matched to the line of sight closest to it. The closest 3D point on the line of sight is found, and the pose that brings each model feature closest to that 3D point is selected, an easier 3D to 3D pose problem. The process is repeated until a minimum is reached for the objective function. Good results are presented.

5 Conclusions

In this report we have given a detailed description of a new algorithm, SoftPOSIT, for matching a model and an image using a full perspective model. This algorithm, unlike most previous algorithms for this problem, does not have to hypothesize small sets of matches and then verify the remaining image points. Instead, all possible matches are treated identically throughout the search for an optimal pose. The algorithm appears to have time complexity $O(JK)$, an encouraging characteristic. The goal of this report was to give a detailed enough description so that the reader can implement SoftPOSIT from the provided pseudocode. In addition, Matlab code is presented in the Appendix.

The SoftPOSIT algorithm is currently being evaluated with large-scale Monte Carlo simulations under a variety of levels of clutter, occlusion, and image noise. Preliminary results are good. Full results of our experiments will be presented and discussed in a forthcoming report.

References

[1] Baird, H.S., "Model-Based Image Matching Using Location", MIT Press, Cambridge, MA, 1985.

[2] Breuel, T.M., "Fast Recognition using Adaptive Subdivisions of Transformation Space", IEEE Conf. on Computer Vision and Pattern Recognition, Champaign, IL, pp. 445–451, 1992.

[3] Bridle, J.S., "Training Stochastic Model Recognition as Networks can Lead to Maximum Mutual Information Estimation of Parameters", Advances in Neural Information Processing Systems, vol. 2, pp. 211–217, 1990.

[4] Cass, T.A, "Polynomial-Time Object Recognition in the Presence of Clutter, Occlusion, and Uncertainty", Computer Vision–ECCV 92, Lecture Notes in Computer Science 588, G. Sandini (Ed.), pp.834–842, Springer-Verlag, 1992.

[5] Chellappa, R., L.S. Davis, D. DeMenthon, A. Rosenfeld, Q. Zheng, "Site-Model-Based Change Detection and Image Registration", Image Understanding Workshop, Washington, DC, April 1993.

[6] DeMenthon, D., L.S. Davis, "Model-Based Object Pose in 25 Lines of Code", European Conf. on Computer Vision, pp. 335–343, 1992.

[7] DeMenthon, D. , L.S. Davis, "Model-Based Object Pose in 25 Lines of Code", International Journal of Computer Vision, 15, pp. 123–141, 1995.

[8] DeMenthon, D., "Recognition and Tracking of 3D Objects by 1D Search", DARPA Image Understanding Workshop, Washington, DC, April 1993.

[9] Ely, R.W., J.A. Digirolamo, J.C. Lundgren, "Model Supported Positioning", Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision II, SPIE Aerospace Sensing & Dual Use Sensors and Controls, Orlando, FL, 1995.

[10] Geiger, D., A.L. Yuille, "A Common Framework for Image Segmentation", International Journal of Computer Vision, 6, pp. 227–243, 1991.

[11] Gold, S., A. Rangarajan, C.P. Lu, S. Pappu, E. Mjolsness, "New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence", Pattern Recognition, 31, pp. 1019–1031, 1998.

[12] Grimson, E., "Object Recognition by Computer: The Role of Geometric Constraints", MIT Press, Cambridge, MA, 1990.

[13] Jurie, F., "Solution of the Simultaneous Pose and Correspondence Problem Using Gaussian Error Model", Computer Vision and Image Understanding, 73, 3, pp. 357–373, 1999.

[14] Kosowsky, J.J., A.L. Yuille, "The Invisible Hand Algorithm: Solving the Assignment Problem with Statistical Physics", Neural Networks, 7, pp.477–490, 1994.

[15] Lamdan, Y., J. Schwartz, H. Wolfson, "Object Recognition by Affine Invariant Matching", IEEE Conf. on Computer Vision and Pattern Recognition, Ann Arbor, MI, pp. 335–344, 1988.

[16] Manjunath, B.S., R. Chellappa, "A Unified Approach to Boundary Perception: Edges, Textures, and Illusory Contours", USC-SP Report 167, 1991.

[17] Oberkampf, D., D. DeMenthon, L. S. Davis, "Iterative Pose Estimation Using Coplanar Feature Points", Computer Vision and Image Understanding, 63, 3, pp. 495–511, 1996.

[18] Pappu, S., S. Gold, A. Rangarajan, "A Framework for Non-Rigid Matching and Correspondence", Advances in Neural Information Processing Systems, vol. 8, pp. 795–801, 1996.

[19] Sinkhorn, R., "A Relationship between Arbitrary Positive Matrices and Doubly Stochastic Matrices", Annals Math. Statist., 35, pp. 876–879, 1964.

[20] Ullman, S., "Aligning Pictorial Descriptions: An Approach to Object Recognition", Cognition, 32, pp. 193–254, 1989.

[21] Wunsch, P., G. Hirzinger, "Registration of CAD Models to Images by Iterative Inverse Perspective Matching", Int. Conf. on Pattern Recognition, Vienna, Austria, pp. 78-83, 1996.

APPENDIX
Matlab Implementation of the SoftPOSIT Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SoftPOSIT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Copyright 2001, Daniel F. DeMenthon and University of Maryland  %%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [rot, trans, assignMat, newImage, foundPose] = ...
    softPOSIT(imagePts, worldPts, beta0, initRot, initTrans, ...
                  focalLength, center)
%
% USAGE:
%     [rot, trans, assignMat, newImage, foundPose] = ...
%         softPOSIT(imagePts, worldPts, beta0, initRot, initTrans,
%                         focalLength, center)
%
% Return the rotation and translation of the world object given
% uncorresponded world points and image points. World and image coordinates
% are homogeneous coordinates represented as row vectors.
%     If the center is not given, assume the image coordinates are
% w.r.t. centered image reference (i.e., w.r.t. the point in the image
% through which the optic axis passes). This version uses an assignment
% matrix first set to the identity.
%     beta0 defines the initial fuzziness of the correspondence matrix.
% If we know that our initial pose is close to the actual pose, then
% beta0 should be close to 0.1 so that the correspondence matrix is not
% completely fuzzy;  otherwise, if the pose is completely unknown, beta0
% should be close to 0.0001 so that all possibilities of correspondence
% are possible in the beginnning.
%
% EXAMPLE:
% Inputs:
%     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]
%     truncCube = [ 0 0 0; 10 0 0; 10 10 0; 0 10 0; 0 0 10; 10 0 10; 10 10 10]
%     truncCube2 = [ 0 0 0; 10 0 0; 10 10 0; 0 10 0; 0 0 10; 10 0 10]
%     cubeImage = [0 0; 80 -93; 245 -77; 185 32; 32 135; 99 35; 247 62; 195 179]
%     focalLength = 760
%     approxTrans = [-60;-60;100]; % this cube projects far from original
%     approxRot = eye(3);
%
% [rr,tt,MM,im] = softPOSIT(cubeImage, truncCube, 0.000045, approxRot,
%                                 approxTrans, 760)
%
% Outputs:
%     rr = [0.4898, -0.8507, -0.1906; -0.5696, -0.1467, -0.8087;
%           0.6600, 0.5047, -0.5565];
```

```
%     tt = [10.4155; 9.5569; 40.5511]';
%     MM = [0 0 0 0 0 0 0 1 0;
%           0 0 0 0 0 0 1 0 0;
%           0 0 0 0 0 1 0 0 0;
%           0 0 0 0 1 0 0 0 0;
%           0 0 0 1 0 0 0 0 0;
%           0 0 1 0 0 0 0 0 0;
%           0 1 0 0 0 0 0 1 0;
%           1 0 0 0 0 0 0 0 0;
%           0 0 0 0 0 0 0 0 1];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

msg = nargchk(6, 7, nargin);
if (~isempty(msg))
        error(strcat('ERROR:', msg));
end

clear msg;

if nargin == 6 % image coordinates are already centered, center is not given
        center = [0, 0];
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

alpha = 25;

betaFinal = 0.5;                    % Best for 7 points.
betaUpdate = 1.025;                 % Very good.
epsilon0 = 0.01;                    % Used to initialize assignement matrix.

maxCount = 2;
maxRandomPoseCount = 10;
minBetaCount = 20;
maxBetaCount = 1000;

maxDelta = 20;                      % Standard deviation error.

nbImagePts = size(imagePts, 1);     % Number of image points (M below).
nbWorldPts = size(worldPts, 1);     % Number of world points (N below).

minNbPts = min([nbImagePts;nbWorldPts]);

% Convert to normalized image coordinates. With normalized coordinates, (0,0)
```

```
% is the point where the optic axis penetrates the image, and the focal length
% is 1.
centeredImage(:,1) = (imagePts(:,1) - center(1))/focalLength;
centeredImage(:,2) = (imagePts(:,2) - center(2))/focalLength;
centeredImage

imageOnes = ones(nbImagePts, 1);   % Column M-vector.

% The homogeneous world points -- append a 1 to each 3-vector. An Nx4 matrix.
worldOnes = ones(nbWorldPts, 1);   % Column N-vector.
wk = worldOnes;                    % Initialized depths of world points.
homogeneousWorldPts = [worldPts, worldOnes];

% Initial rotation and translation as passed into this function.
rot = initRot;
trans = initTrans;

% Draw a picture of the cube on the original image plane.
newImage = world2Image(worldPts, rot, trans, focalLength);
% plotPts(imagePts, newImage);
% plotCube(imagePts, newImage);
% plotTrunc2(imagePts, newImage);
plotBldgs(imagePts, newImage);

% Rows of the camera matrices (for both perspective and SOP).  Note:
% the scale factor is s = f/Tz = 1/Tz since f = 1.  These are column 4-vectors.
r1T = [rot(1,:)/trans(3), trans(1)/trans(3)]';
r2T = [rot(2,:)/trans(3), trans(2)/trans(3)]';

betaCount = 0;
poseConverged = 0;
assignConverged = 0;
beta = beta0;

% The assignment matrix includes a slack row and slack column.
assignMat = ones(nbImagePts+1,nbWorldPts+1) + epsilon0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Deterministic annealing loop.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while beta < betaFinal & ~assignConverged
% while betaCount < 200
% while beta < betaFinal & ~poseConverged
% while beta < betaFinal
% while beta < betaFinal & ~poseConverged & ~assignConverged

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Given the current pose and w[i], compute the distance matrix, d[j,k].
    % d[j,k] is the squared distance between the j-th corrected SOP image
    % point and the SOP projection of the k-th scene point.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% SOP (weak perspective projection) of world points.
projectedU = homogeneousWorldPts * r1T;          % Column N-vector.
projectedV = homogeneousWorldPts * r2T;          % Column N-vector.

% MxN matrices with identical rows equal to the X (replicatedProjectedU)
% and Y (replicatedProjectedV) SOP projections of the world points.  The
% number of rows M is equal to the number of image points.
replicatedProjectedU = imageOnes * projectedU';
replicatedProjectedV = imageOnes * projectedV';

% For all possible pairings of image to world points, map the
% perspective image point into the corrected SOP image point using
% the world point's current estimate of w[i].  Here, j is the index of
% an image point and k is the index of a world point.  wkxj is an
% MxN  matrix where the j-th,k-th entry is w[k]*x[j]/f.  wkyj is an
% MxN matrix where the j-th,k-th entry is w[k]*y[j]/f.
wkxj = centeredImage(:,1) * wk';
wkyj = centeredImage(:,2) * wk';

% distMat[j,k] = d[j,k]^2.  The focalLength^2 term here maps the distances
% back to the original (unnormalized) image, so distances are in terms
% of original pixels.
distMat = focalLength*focalLength*((replicatedProjectedU - wkxj).^2 + ...
                                   (replicatedProjectedV - wkyj).^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use the softAssign algorithm to compute the best assignment of image to
% world points based on the computed distances.  The use of alpha
% determines when to favor assignments instead of using slack.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prevAssignMat = assignMat;
assignMat(1:nbImagePts, 1:nbWorldPts) = exp(-beta*(distMat - alpha));
% assignMat = sinkhornSlack(assignMat) % Do not normalize slack row and col.
assignMat = sinkhornImp(assignMat);    % Improved Sinkhorn normalization, see below

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use POSIT to calculate the pose that minimizes the objective function
% (equation (18), the weighted sum of the differences between corrected
% image points and SOP's of the corresponding world points), given the
% current assignment.  The pose parameters and the w[i] are iteratively
% updated until some convergence criterion is satisfied.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute Sum{k=1,N}(m'[k]S[k]S[k]^T) -- this is used in equations (11) and
% (12) in the paper, and below in the POSIT loop, to compute an optimal
% pose.  In the paper, L == sumSkSkT.
%     First, compute the sum of the elements w/in each column,
% ignoring the slack row and column. This is a row N-vector.
summedByColAssign = sum(assignMat(1:nbImagePts, 1:nbWorldPts), 1);
```

```
%     Now, compute the 4x4 matrix L.
sumSkSkT = zeros(4, 4);
for k = 1:nbWorldPts
    sumSkSkT = sumSkSkT + summedByColAssign(k) * ...
                    homogeneousWorldPts(k,:)' * homogeneousWorldPts(k,:);
end

objectMat = inv(sumSkSkT);                              % Inv(L), a 4x4 matrix.

poseConverged = 0;                                  % Initialize for POSIT loop.
count = 0;

% POSIT loop.  We put a cap on number of steps here, so at first it does
% not converge.  We currently do just one iteration of POSIT.  When the
% annealing temperature is low enough, one iteration of POSIT is
% sufficient since the assigments and pose will converge simultaneously.
% while ~poseConverged & count < maxCount

    % Compute the term in the equation for the optimal pose vectors (M and
    % N in equations (11) and (12) in the paper) that depends on the current
    % assigment and the w[i]. These are Sum{all j,k}(m[j,k]w[k]x[j]S[k]) and
    % Sum{all j,k}(m[j,k]w[k]y[j]S[k]).  Here, (w[k]x[j],w[k]y[j]) is our
    % best estimate of the SOP of the k-th scene point whose homogeneous
    % perspective image coordinate is (x[j],y[j]).
    weightedUi = zeros(4,1);                        % column vector
    weightedVi = zeros(4,1);                        % column vector
    for j = 1:nbImagePts
        for k = 1:nbWorldPts
            weightedUi = weightedUi + assignMat(j,k) * wk(k) * ...
                            centeredImage(j,1) * homogeneousWorldPts(k,:)';
            weightedVi = weightedVi + assignMat(j,k) * wk(k) * ...
                            centeredImage(j,2) * homogeneousWorldPts(k,:)';
        end % for k
    end % for j

    % Compute the pose vectors. M = s(R1,Tx) and N = s(R2,Ty) where the
    % scale factor is s = f/Tz, and f = 1.  These are column 4-vectors.
    r1T= objectMat * weightedUi;                % M
    r2T = objectMat * weightedVi;               % N

    % Compute the rotation matrix and translation vector corresponding to
    % the computed pose vectors.
    if 1
        disp(' *** Renormalization of rotation matrix ***');
        [U, S, V] = svd([r1T(1:3)'; r2T(1:3)']');
        A = U * [1 0; 0 1; 0 0] * V';
        r1 = A(:,1);
        r2 = A(:,2);
        r3 = cross(r1,r2);
        Tz = 2 / (S(1,1) + S(2,2));
        Tx = r1T(4) * Tz;
        Ty = r2T(4) * Tz;
```

```
        r3T= [r3; Tz];
    else
        % Standard calculation of R and T.  The rotation matrix may not be
        % orthonormal.  The object must distort when the rotation matrix
        % is not orthonormal.
        r1TSquared = r1T(1)*r1T(1) + r1T(2)*r1T(2)+ r1T(3)*r1T(3);
        r2TSquared = r2T(1)*r2T(1) + r2T(2)*r2T(2)+ r2T(3)*r2T(3);
        Tz = sqrt(2/(r1TSquared+r2TSquared));   % Chang & Tsai's suggestion.
        r1N = r1T*Tz;                   % Column 4-vectors: (R1,Tx).
        r2N = r2T*Tz;                   %                   (R2,Ty).
        r1 = r1N(1:3);                  % Three rows of the rotation matrix.
        r2 = r2N(1:3);
        r3 = cross(r1,r2);
        r3T= [r3; Tz];                  % Column 4-vector: (R3,Tz).
        Tx = r1N(4);
        Ty = r2N(4);
    end % if

    % Compute updated estimates for the w[i] (the third homogeneous
    % coordinate of the perspective image points).  The improved w[i] are
    % used above to map (correct) the perspective image coordinates into
    % the corresponding scaled orthographic projection image coordinates.
    wk= homogeneousWorldPts * r3T /Tz;

    % Determine if the pose as computed by POSIT has converged.
    % The pose is considered to have converged when the sum of the
    % weighted distances between the corrected SOP image points and
    % the SOP's of the world points is less than some threshold.
    % This distance is in terms of pixels in the original image
    % coordinate frame.
    delta = sqrt(sum(sum(assignMat(1:nbImagePts,1:nbWorldPts) ...
                                  .* distMat))/minNbPts);
    delta
    poseConverged = delta < maxDelta

    count = count + 1;                          % Number of POSIT iterations.

% end % of POSIT loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Updates for deterministic annealing and checks for convergence.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update the ''annealing temperature'' and determine if the assignments
% have converged.

beta = betaUpdate * beta
betaCount = betaCount + 1    % Number of deterministic annealing iterations.
assignConverged = poseConverged & betaCount > minBetaCount

% Form the best estimates for the translation vector and rotation matrix.
trans = [Tx; Ty; Tz];
```

```
    rot = [r1'; r2'; r3'];

    % Has the pose converged?
    foundPose = (delta < maxDelta & betaCount > minBetaCount);


end % of Deterministic annealing loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Print the answer that will be returned by this function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rot
trans
assignMat

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of softPOSIT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Sinkhorn algorithm for matrices with slack row and column.
%
% Normalize across rows and columns to find an assignment matrix (a
% doubly stochastic matrix).
%
% This version treats the slack row and column differently from other rows
% and columns: the slack values are not normalized with respect to other
% slack values, only with respect to the nonslack values.  This may work
% better than the original Sinkhorn algorithm which treats all rows and
% columns identically.  This is true primarily when there needs to be more
% than one assignment to a slack row or column.  I.e., when there are two
% or more missing image points or model points.
%
% DSD returns a vector of the doubly-stochastic distances of the matrix
% throughout the sinkhorn iteration.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [normalizedMat, dsd] = sinkhornSlack(M)

  iMaxIterSinkhorn=60;        % In PAMI paper
  fEpsilon2 = 0.001;          % Used in termination of Sinkhorn Loop.

  iNumSinkIter = 0;
  [nbRows, nbCols] = size(M);

  fMdiffSum = fEpsilon2 + 1;  % Set "difference" from last M matrix above
                              % the loop termination threshold
```

```
  dsd(1) = dsdist(M,1);       % Measure of doubly stochasticness.

  while(abs(fMdiffSum) > fEpsilon2 & iNumSinkIter < iMaxIterSinkhorn)
      Mprev = M;  % Save M from previous iteration to test for loop termination

      % Col normalization (except outlier row - do not normalize col slacks
      % against each other)
      McolSums = sum(M, 1); % row vector
      McolSums(nbCols) = 1; % don't normalize slack col terms against each other
      McolSumsRep = ones(nbRows,1) * McolSums ;
      M = M ./ McolSumsRep;

      % Row normalization (except outlier row - do not normalize col slacks
      % against each other)
      MrowSums = sum(M, 2); % column vector
      MrowSums(nbRows) = 1; % don't normalize slack row terms against each other
      MrowSumsRep = MrowSums * ones(1, nbCols);
      M = M ./ MrowSumsRep;

      iNumSinkIter=iNumSinkIter+1;
      fMdiffSum=sum(abs(M(:)-Mprev(:)));

      dsd(iNumSinkIter+1) = dsdist(M,1);    % Measure of doubly stochasticness.
  end %Sinkhorn

  % iNumSinkIter
  normalizedMat = M;

  return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Phil's improve Sinkhorn algorithm for matrices with slack row and column.
%
% normalizedMat = sinkhornImp(M)
%
% Apply an improved Sinkhorn algorithm to map matrix M to a doubly
% stochastic matrix.
%
% The Sinkhorn algorithm modified for slack rows and columns treats the
% slack row and column differently from other rows and columns: the slack
% values are not normalized with respect to other slack values, only with
% respect to the nonslack values.  This may work better than the original
% Sinkhorn algorithm which treats all rows and columns identically.
% This is true primarily when there needs to be more than one assignment
% to a slack row or column.  I.e., when there are two or more missing
% image points or model points.
%
% A problem with this modified Sinkhorn algorithm is the following.
% Suppose all rows except the slack row are normalized.  It is possible that
```

```
% a nonslack value which was previously maximum in its row and column to now
% have a value that is less than the slack value for that column. (This
% nonslack value will still be greater than the slack value for that
% row.)  The same sort of thing can happen when columns are normalized.
% Intuitivitly, this seems like a bad thing: nonslack values that start
% off as maximum in their row and column should remain maximum in their
% row and column throughout this iteration.  The current algorithm
% attempts to prevent this from happening as follows.  After performing
% row normalizations, the values in the slack row are set so that their
% ratio to the nonslack value in that column which was previously maximum
% is the same as this ratio was prior to row normalization.  A similar
% thing is done after column normalizations.
%
% DSD returns a vector of the doubly-stochastic distances of the matrix
% throughout the sinkhorn iteration.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [normalizedMat, dsd] = sinkhornImp(M)

iMaxIterSinkhorn=60;          % In PAMI paper
fEpsilon2 = 0.001;            % Used in termination of Sinkhorn Loop.

iNumSinkIter = 0;
[nbRows, nbCols] = size(M);

fMdiffSum = fEpsilon2 + 1;    % Set "difference" from last M matrix above
                              % the loop termination threshold

% Get the positions and ratios to slack of the nonslack elements that
% are maximal in their row and column.
[posmax, ratios] = maxPosRatio(M);

dsd(1) = dsdist(M,1);         % Measure of doubly stochasticness.

while(abs(fMdiffSum) > fEpsilon2 & iNumSinkIter < iMaxIterSinkhorn)
    Mprev = M;  % Save M from previous iteration to test for loop termination

    % Col normalization (except outlier row - do not normalize col slacks
    % against each other)
    McolSums = sum(M, 1);  % Row vector.
    McolSums(nbCols) = 1;  % Don't normalize slack col terms against each other.
    McolSumsRep = ones(nbRows,1) * McolSums ;
    M = M ./ McolSumsRep;

    % Fix values in the slack column.
    for i = 1:size(posmax,1)
      M(posmax(i,1),nbCols) = ratios(i,1)*M(posmax(i,1),posmax(i,2));
    end

    % Row normalization (except outlier row - do not normalize col slacks
    % against each other)
```

```
    MrowSums = sum(M, 2);  % Column vector.
    MrowSums(nbRows) = 1;  % Don't normalize slack row terms against each other.
    MrowSumsRep = MrowSums * ones(1, nbCols);
    M = M ./ MrowSumsRep;

    % Fix values in the slack row.
    for i = 1:size(posmax,1)
      M(nbRows,posmax(i,2)) = ratios(i,2)*M(posmax(i,1),posmax(i,2));
    end

    iNumSinkIter=iNumSinkIter+1;
    fMdiffSum=sum(abs(M(:)-Mprev(:)));

    dsd(iNumSinkIter+1) = dsdist(M,1);      % Measure of doubly stochasticness.
end

normalizedMat = M;

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```