Computers and Electrical Engineering 36 (2010) 1165–1180

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Gravitational pose estimation [☆]

H. Fatih Ugurdag, Sezer Gören [*], Ferhat Canbay

Department of Computer Engineering, Bahcesehir University, Ciragan Cad. Osmanpasa, Mektebi Sok, Besiktas 34349, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 21 December 2009
Accepted 19 May 2010
Available online 17 June 2010

Keywords:
Pose estimation
Camera calibration
Correspondenceless
Gravitational field
Occlusion

ABSTRACT

Problem of relative pose estimation between a camera and rigid object, given an object model with feature points and image(s) with respective image points (hence known correspondence) has been extensively studied in the literature. We propose a "correspondenceless" method called gravitational pose estimation (GPE), which is inspired by classical mechanics. GPE can handle occlusion and uses only one image (i.e., perspective projection of the object). GPE creates a simulated gravitational field from the image and lets the object model move and rotate in that force field, starting from an initial pose. Experiments were carried out with both real and synthetic images. Results show that GPE is robust, consistent, and fast (runs in less than a minute). On the average (including up to 30% occlusion cases) it finds the orientation within 6° and the position within 17% of the object's diameter. SoftPOSIT was so far the best correspondenceless method in the literature that works with a single image and point-based object model like GPE. However, SoftPOSIT's convergence to a result is sensitive to the choice of initial pose. Even "random start SoftPOSIT," which performs multiple runs of SoftPOSIT with different initial poses, can often fail. However, SoftPOSIT finds the pose with great precision when it is able to converge. We have also integrated GPE and SoftPOSIT into a single method called GPEsoftPOSIT, which finds the orientation within 3° and the position within 10% of the object's diameter even under occlusion. In GPEsoftPOSIT, GPE finds a pose that is very close to the true pose, and then SoftPOSIT is used to enhance accuracy of the result. Unlike SoftPOSIT, GPE also has the ability to work with three points as well as planar object models.

1. Introduction

Position and orientation (pose) estimation of a three-dimensional (3D) object with respect to a camera – or vice versa – from the object's two-dimensional (2D) image(s) is a subtask in robotics, computer graphics, computer and machine vision. It is, for example, used for robot guidance in manufacturing applications. In augmented reality, it is required for accurately inserting computer graphics objects into photographic/movie scenes. In computer vision, pose estimation is instrumental for object recognition. In machine vision, it may sometimes be needed for making precise measurements. And in a variety of tasks, it is used for camera calibration (i.e., finding the pose of camera with respect to the world coordinates) before carrying out the actual task.

There is a significant amount of literature that uses a CAD model of an object's surface with 3D [1–5] or 2D images [6,7] of the object to estimate pose. Pose estimation with 3D imaging requires sophisticated hardware such as a laser and apparatus to sweep the laser to scan the object. Also, not every object has a smooth surface that can be easily modeled and scanned. On

☆ Reviews processed and proposed for publication to the Editor-in-Chief by Associate Editor Dr. F. Sahin
* Corresponding author. Tel.: +90 212 381 0569; fax: +90 212 381 0550.
E-mail address: sgoren@bahcesehir.edu.tr (S. Gören).

0045-7906/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2010.05.007

the other hand, surface-based pose estimation with a 2D image has all of the challenges of its 3D version. Furthermore, it is not easy to recover surface contour lines in a 2D image. If structured lighting is not used, that may be extremely difficult. In that case, the object outline in the 2D image can be used [8] with the risk of not being able to distinguish some viewing angles from each other.

When pose estimation needs to have one or more of the following properties, it needs to be based on 2D imaging and be "feature-based" (as opposed to surface-based): (1) tracked object does not have smooth surface. (2) There is no tracked object and instead we have a set of markers as in the case of augmented reality. (3) Structured lighting is not used. (4) Background is crowded. (5) Real-time operation is needed. (6) An inexpensive solution is required. (7) Highly accurate pose estimation is desired.

By 2D above, we mean a regular camera is used – no laser triangulation or any other 3D imaging technique is employed. By feature-based, we mean the object model models only certain features of the object, and the pose estimation algorithm's preprocessing computer vision step identifies those same features in the image taken (i.e., feature extraction).

To make the problem easier, the features can be taken as points [9–15] or lines [15–20] on the object. In problems where we have the option of designing the tracked object at our wish, we can sometimes use striking colors to make the features stand out. It may seem best if we make each feature a different color so that we have an easier feature extraction problem prior to pose estimation – one with "known correspondence." However, a vision system is not like a human eye and usually has problem differentiating multiple colors without (or sometimes even with) controlled lighting. The easiest approach (we have come to follow in our own work) in robot guidance is that we put infrared LEDs (i.e., create feature points) on the tracked robot and an infrared filter on the camera lens. When that is the case, all feature points are of the same color, and they can be easily identified and not be confused with the background. However, then the challenge is that we have to use a "correspondenceless" pose estimation method like our proposed *gravitational pose estimation (GPE)*.

Most point-feature based pose estimation techniques require correspondence information. Given matching object and image feature pairs, the pose that best aligns those pairs is determined. The pose can be found in closed form [9] by solving polynomial equations, if there are 3–5 points matched. More points are desirable to compensate for occlusion as well as errors introduced by imaging system and vision algorithms used for feature (i.e., point) detection. For 6 or more points, non-linear and linear approximate methods [9–11,13–15,21–22] are proposed. Gradient-descent and least-squares type methods can be used if correspondence is given. However, such brute-force techniques suffer from local optima and do not work if correspondence is unknown.

There are relatively few point-based methods [12,23–25] that work with unknown correspondence. The method by Liu and Rodrigues in [12] requires two images and assumes that camera translation and rotation is contained in the XY plane of the world coordinates. Lin et al. [23] propose a correspondenceless method in the context of motion estimation and hence assume a sequence of images in close proximity of each other. Aloimonos and Herve [24] find the pose change of an object that has a planar surface, using two cameras with parallel axes. We will later discuss SoftPOSIT [25] in more detail.

There are several methods in the literature that work in the presence of partially reliable correspondence information, but are not really correspondenceless. RANSAC [26] is one of them. It can work with a marginal feature detector that supplies correspondence information between image points and object points, where some of the correspondences may be incorrect. A very recent work [27] operates under very similar assumptions but does not get caught in local optima. In addition to requiring (partially reliable) correspondence information, this method minimizes the distance between true and computed image points, whereas our method GPE minimizes the distance from the true position of the object in 3D. There are other methods such as [28] that use detected contours of the object to come up with accurate correspondence information between image and object points. There are similar methods that use object recognition techniques to match lines or curves in the image and object model [20,29].

There is only one technique in the literature that is exactly in the same category as GPE, i.e., correspondenceless, point-based, uses a single 2D image, is designed to handle unrestricted 3D pose [25]. The work of David and DeMenthon et al. in [25] called "random start *SoftPOSIT*" has been the state of the art so far. This method requires at least four feature points of the object to be used, and the points cannot be coplanar. On the other hand, GPE can work with coplanar points and can even work with three points. Note that with three points the problem has more than one solution.

Random start *SoftPOSIT* runs SoftPOSIT multiple times and takes the best solution. SoftPOSIT algorithm is an integration (into a single loop) of the iterative *SoftAssign* algorithm by Gold [30] (for computing correspondences) and the iterative *POSIT* (Pose from Orthography and Scaling with ITerations) algorithm by DeMenthon [21] (for computing pose with given correspondence info.) SoftPOSIT performs a search starting from an initial guess for the object's pose. A global objective function is defined, which combines the two separate goals of finding correspondence and finding pose. The objective function is minimized by combining the formalisms of both iterative techniques. The function has many local optima, and as a result, even the employed annealing process may not sometimes help. Thus, there is no guarantee of finding the global minimum with a single initial guess. To overcome this, "random start SoftPOSIT" is proposed in [25], where the search algorithm is launched from a number of different initial guesses. The algorithm outputs the first solution that meets a specified termination criterion.

Our proposed method GPE and SoftPOSIT are complementary techniques. Preliminary results from GPE were presented in [31]. This paper, on the other hand, extends GPE to handle occlusion (i.e., missing feature points in the image), compares GPE to random start SoftPOSIT, and then integrates it with SoftPOSIT to create a method called *GPEsoftPOSIT*. Although GPE can theoretically handle clutter, we did not produce results for it in this work because we target applications where easily identifiable feature points are used. One example was given above with infrared LEDs. In such cases, a cluttered background would not mislead pose estimation by introducing false feature points.

Based on experimental results with synthetic images, we show that GPE's strength is in its robustness, and it always finds a close enough solution irrespective of true pose and initial pose. On the other hand, SoftPOSIT's sensitivity to initial pose quite often cannot be adequately compensated by starting it from multiple random initial poses. SoftPOSIT's strength is in its precision of identifying the true pose when it succeeds to converge. This led us to come up with GPEsoftPOSIT. In GPEsoftPOSIT, we first estimate the pose with GPE – which brings the pose to a very accurate position and within a few degrees of the orientation. Then, we feed this pose to SoftPOSIT as initial pose. This gives us a robust and precise method and also runs fast as it does not have to do a random start from many initial poses.

The paper is structured as follows. In Section 2, the problem formulation is detailed. GPE and GPEsoftPOSIT are presented in Sections 3 and 4, respectively, followed by the results and conclusion in Sections 5 and 6.

2. Problem formulation

Image of a 3D point (on the image sensor of a regular camera) is the perspective projection of the point on to the image plane – though less accurate approximations (e.g. orthographic projection) are possible. If the coordinates of the point are (x, y, z) with respect to the camera coordinate frame (*origin* is the camera lens' center and the X–Y axes are along the edges of the rectangular image sensor), then its image is located at $(x.f/z, y.f/z, f)$ where f is the distance between the camera lens and image plane – called f because it is approximately equal to the focal length of the lens. In real life, image plane is behind the camera lens at $z = -f$ and image is inverted. However, most treat it as if the image plane is at $z = f$ without any loss of generality – a simple flip of all three coordinates does the job. All points on line (kx, ky, kz), where k is a free parameter, produce the same image point on the image plane. Therefore, for each image point, we can construct a unique line in 3D (called *line of sight*) on which the object point must be located.

A camera image does not consist of feature points; it instead consists of pixels, each of which is assigned a value indicating brightness and color if any. However, it is possible to transform an image to a collection of feature points by computer vision techniques. Each of the feature points on the image plane corresponds to a point on the object and gives us a line of sight (LOS) in 3D when extended through the lens center. Thus, from the image of an object we obtain a *pencil of lines*, and we know that they go through the object as in Fig. 1.

GPE (and similar techniques) require a rigid object model, which is a set of 3D *object points* (i.e., feature points) expressed by their coordinates in the *object coordinate frame*. Then, the answer to the pose estimation problem is the pose where the object points are as close to the LOSes as possible. From the above discussion we can devise an *energy function*, which is expected to be *zero* when the correct pose is found – hence gives us an absolute measure of knowing if we have reached the global optimum in our search. Below are the steps to calculate the energy of a suggested pose for the object:

1. For the suggested pose, transform object points' coordinates in the object model from the object coordinate frame to the camera coordinate frame. (Everything from this point on is computed in the camera coordinate frame.)
2. Construct the LOSes from the image points.
3. Let d_{ij} be the shortest, hence perpendicular distance, between the i th point of the object model and the j th LOS. Calculate and sort all d_{ij} 's in ascending order. Place them in a list called *DistanceList*.
4. Pair the LOSes with object points until there is no LOS left (assuming no false feature points). Do this in such a way that an object point is paired with only one LOS and an LOS is paired with only one object point. Start with the smallest d_{ij} (the topmost element in *DistanceList*), and assign the i th point of the object model to the j th LOS. Erase all distances in *DistanceList* that belong to object point i (all d_{ik} where k is a free parameter) as well as all distances that belong to LOS j (all d_{kj} where k is free). Continue with the smallest d_{mn} in the updated *DistanceList*, and pair object point m and LOS n. Continue until all LOSes are paired.
5. Calculate the energy of object model in the suggested pose, which is equal to the sum of squares of the distances between all LOS-object point pairs:

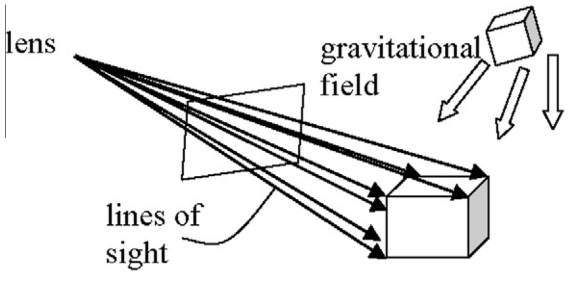


Fig. 1. An object in the gravitational field of the lines of sight.

$$E = \sum d_{ij}^2 \tag{1}$$

It should be noted that one can calculate the average distance between the points and lines as in:

$$d_{ave} = \sqrt{(E/\#\text{lines})} \tag{2}$$

When the estimated (suggested) pose is equal to the correct pose, d_{ave} should be quite small with respect to the dimensions of the object. Although this is currently our criterion, a maximum feature distance criterion may also be imposed.

3. GPE algorithm

The algorithm starts by forming the LOSes. A line can be represented (though not canonically) by a point on it and a (unit-length) direction vector. As we want the object points to coincide with the LOSes, it is quite intuitive to form a gravitational field where LOSes attract object points as shown in Fig. 1. Since we do not want object points to move after they reach the LOSes, it makes sense to make the force (gravitational field) proportional to the distance between object points and LOSes. The outline of the algorithm is as follows:

1) Pair the object points and LOSes on a 1-to-1 basis as described in Section 2 in the energy calculation instructions.
2) Calculate the force on each object point through Eq. (3), where $\vec{f_i}$ is the force on object point i and d_{ij} is the distance between the i^{th} object point and its paired LOS j, and $\hat{r}ij$ is the unit vector that points from point i to line j and is perpendicular to line j.

$$\vec{f}_i = d_{ij}\hat{r}_{ij} \tag{3}$$

3) Use classical mechanics to calculate the translational, $\vec{a}_t$, and rotational acceleration, $\vec{a}_r$, as follows:
a) Calculate effective force, $\vec{f}_{cm}$ applied to the object's center of mass (COM) by:

$$\vec{f_{cm}} = \sum_i \hat{f_i} \tag{4}$$

Note that force on an unpaired object point (a possibly occluded point) is zero. However, unpaired points still contribute to the COM calculations, i.e., they are still considered part of the object.

b) Calculate translational acceleration, $\vec{a}_t$, which is the acceleration of object's COM, by (5), where m_{tot} is the total mass of the object. (Normally, we assume every feature point has a mass of 1. However, if we have different degrees of certainty in feature extraction or have reasons to believe that some feature points have to be aligned more carefully with LOSes, then those points may be assigned a larger mass than other points.)

$$\vec{a}_i = \vec{f}_{cm}/m_{tot} \tag{5}$$

c) Calculate total torque, $\vec{t}_{cm}$, around the COM using the summation of cross-products in (6), where $(\vec{r}_{cm})_i$ is the vector that gives the relative displacement of point i with respect to COM.

$$\vec{t}_{cm} = \sum_i (\vec{r}_{cm})_i \times \vec{f}_i \tag{6}$$

d) Calculate the rotational inertia matrix of the object (at the current pose,) $\mathbf{R}$, through (7), where $(\mathbf{R})_{pq}$ is a component of matrix $\mathbf{R}$ that is located in row p and column q, m_i is the mass of object point i, and $((\vec{r}_{cm})_i)_k, ((\vec{r}_{cm})_i)_p, and((\vec{r}_{cm})_i)_q$ denote the k th, p th, and q th components of the vector $(\vec{r}_{cm})_i$, respectively.

$$(\mathbf{R})_{\mathbf{pq}} = \sum_{\mathbf{i}} \mathbf{m_i} \sum_{\mathbf{k}} \{((\vec{\mathbf{r}}_{\mathbf{cm}})_{\mathbf{i}})_{\mathbf{k}}((\vec{\mathbf{r}}_{\mathbf{cm}})_{\mathbf{i}})_{\mathbf{k}} - ((\vec{\mathbf{r}}_{\mathbf{cm}})_{\mathbf{i}})_{\mathbf{p}}((\vec{\mathbf{r}}_{\mathbf{cm}})_{\mathbf{i}})_{\mathbf{q}}\} \tag{7}$$

e) Calculate angular acceleration, $\vec{a}_r$, using (8), where $\mathbf{R}^{-1}$ is the inverse of rotational inertia matrix, $\mathbf{R}$.

$$\vec{a}_r = \mathbf{R}^{-1}\vec{t}_{cm} \tag{8}$$

4) Let the pose of object be expressed by vectors $\vec{n}, \vec{s}, \vec{a}$ (unit vectors by definition) and $\vec{p}$, where $\vec{n}$ is the orientation of X-axis of the object model (object coordinate frame) expressed in the camera coordinate frame, $\vec{s}$ is the orientation of Y-axis, $\vec{a}$ is the orientation of Z-axis, and $\vec{p}$ is the position of object. Then:
a) Calculate $\Delta\vec{p}$ where $\Delta\vec{p} = \vec{a}_t$ and $\Delta\vec{p}$ is the change in $\vec{p}$, so add $\Delta\vec{p}$ to old $\vec{p}$ to get the new $\vec{p}$.
b) Rotate $\vec{n}, \vec{s}, and\vec{a}$ around the vector $\vec{a}_r$ by an angle equal to $|\vec{a}_r|$ radians, where $|\vec{a}_r|$ denotes the length of the vector $\vec{a}_r$. In order to do this, we follow the steps below for each of $\vec{n}, \vec{s},$ and $\vec{a}$:
i) Let $\vec{v}$ be a vector to be rotated around $\vec{a}_r$ shown as in Fig. 2. Normalize $\vec{v}$ (i.e., make it unit-length) to get unit vector $\hat{v}$. Identify the two components of $\hat{v}$, $\hat{v}_p$ and $\hat{v}_n$, where $\hat{v}_p$ is parallel to $\vec{a}_r$, and $\hat{v}_n$ is normal (i.e., perpendicular) to $\vec{a}_r$, using normalization, dot-product, and scalar multiplication in (9) and vector subtraction (10).

$$\vec{v}_p = (\vec{a}_r \cdot \hat{v})\hat{a}_r \quad (9)$$

$$\vec{v}_n = \vec{v} - \vec{v}_p \quad (10)$$

ii) Construct unit vector $\vec{v}_{nn}$ from $\vec{v}_n$ using (11). Vector $\hat{v}_{nm}$ and $\vec{a}_r$, and is the direction vector $\vec{v}$ moves when it is rotated sround $\vec{a}_r$.

$$\vec{v}_{nm} = \hat{a}_r \times \hat{v}_n \quad (11)$$

iii) Calculate the next $\vec{v}$ denoted as $\vec{v}_{next}$ using (12):

$$\vec{v}_{next} = \vec{v}_p + |\vec{v}_n|(\cos(|\vec{a}_r|)\,\hat{v}_n + \sin(|\vec{a}_r|)\,\hat{v}_{nm}) \quad (12)$$

5) Calculate the energy E at the next pose – pair object points and LOSes – use (1). If the energy obtained is not small enough, go to step 2 and continue to move the object in the gravitational field until it gets stuck in a locally minimum energy state. If it is stuck, then update the minimum pose and energy, provided current one is the best. If the energy is still not small enough (and the maximum number of iterations is not reached), then go to step 6 to shake the object (*shake* means a random rotation).
6) Generate an angular acceleration, $\vec{a}_r$, randomly. Apply step 4 to calculate the next orientation. Then go to step 1. As an example, a 10-point object is shown in Fig. 3. We ran GPE by taking the true pose as

$\vec{n} = (1,0,0), \vec{s} = (0,1,0), \vec{a} = (0,0,1), \vec{p} = (0,0,5)$ and initial pose as $\vec{n} = (0,1,0), \vec{s} = (0,0,1), \vec{a} = (1,0,0)$, $\vec{p} = (20,20,20)$.

The initial energy came out to be around 8000. The criterion for a local minimum was set such that the absolute change in the energy is less than 0.0001 for 30 consecutive iterations. The criterion for termination was either getting an energy of less than 0.0002 or to reaching iteration 1000. (For our larger set of experiments, we have used up to 50,000 iterations since GPE runs quite fast.) The result of this initial experiment was impressive. The number of shakes was only 2. The program stopped with energy less than 0.0002 after the 215th iteration. The trajectory of the object during the program execution is plotted in Fig. 4. In addition, energy versus iteration count is given in Fig. 5.

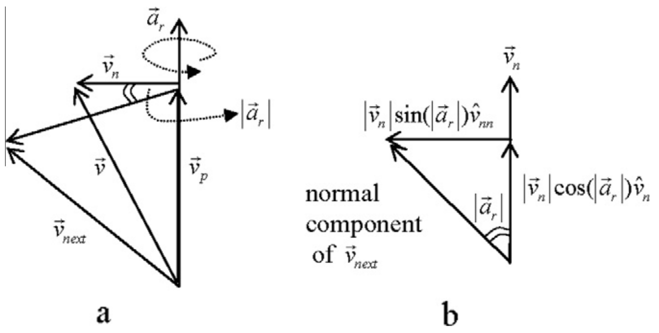

$\vec{a}_r$
$\vec{v}_n$
$|\vec{a}_r|$
$\vec{v}$
$\vec{v}_p$
$\vec{v}_{next}$
a
$\vec{v}_n$
$|\vec{v}_n|\sin(|\vec{a}_r|)\hat{v}_{nn}$
normal component of $\vec{v}_{next}$
$|\vec{a}_r|$
$|\vec{v}_n|\cos(|\vec{a}_r|)\hat{v}_n$
b

Fig. 2. Calculation of $\vec{v}_{next}$. (a) 3D view and (b) top view.

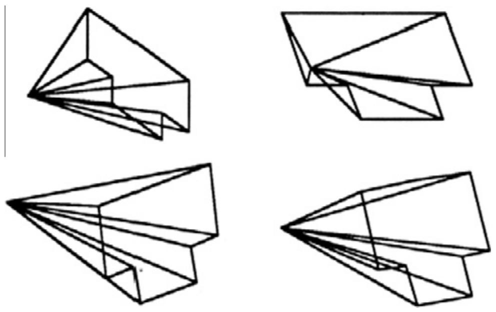


Fig. 3. Four different views of a 10-point synthetic object (3 of the points are collinear).

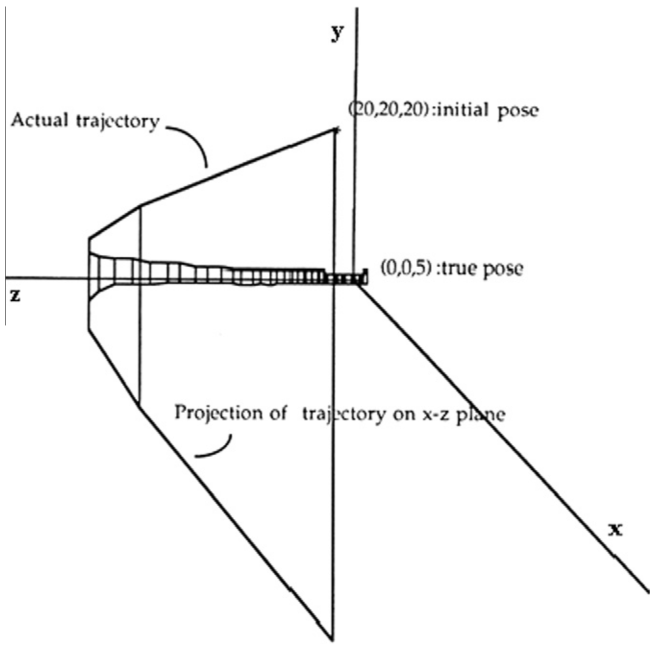


Fig. 4. Trajectory of the above object during GPE's search for the true pose.

4. GPEsoftPOSIT

GPEsoftPOSIT is an integration of the two algorithms, GPE and SoftPOSIT. The main idea here is to run SoftPOSIT with an initial guess that is close to the true pose instead of trying many random starts. The pose found by GPE is taken as the initial guess for the object's pose when running SoftPOSIT. Whenever the ratio of object points matched to LOSes to the total number of object points reaches a predetermined criterion (here it is set to 0.7) and also SoftPOSIT satisfies its own definition of convergence, then SoftPOSIT terminates and outputs the guess for the object's pose. If SoftPOSIT is unable to find a pose, then the initial guess for the object's pose (the result of GPE) is presented as the output.

SoftPOSIT has a parameter, called β_0, which corresponds to the fuzziness of the correspondence matrix. If the initial pose is close to the actual pose, then β_0 should be around 0.1. In our case, since our initial pose comes from GPE and we know that it is close to the actual pose, we set β_0 to 0.1. As an exercise, the 10-point synthetic object in Fig. 3 and its image for a given pose were constructed. Using the object model and image, GPE was run and its result was piped to SoftPOSIT. The projected model and the original image are shown in blue and red, respectively in Fig. 6a–e. The projected model shown in Fig. 6a corresponds to the object at the pose found by GPE. The iterations of SoftPOSIT, on the other hand, are shown in Fig. 6b–e.

5. Experimental results

We first validated GPE using real images, and then we compared GPE, GPEsoftPOSIT, and random start SoftPOSIT on synthetic data with and without occlusion.

5.1. Real image experiments

In this section, we test GPE (without SoftPOSIT) with real images taken by a simple webcam, and our results are still impressive. (A webcam or any other cheap camera has a fish-eye effect. Please notice that the A4 paper's borders in Fig. 7 look curved.) We have a fixture (see Fig. 8) where we can attach an object, rotate the object by means of an arm, and adjust the angle manually. As we have a protractor attached to the arm, we exactly know what the actual angle is. In Fig. 7, the images of a black polygon are taken when the object is brought to angles 70°, 75°, 80°, 85°, 90°, 95°, 100°, 105°, and 110° on the protractor. All of the images taken are shown in Fig. 7a–i.

Although GPE can measure all 6 degrees of freedom of a 3D pose, it is hard to measure the same 6 degrees of freedom mechanically as it requires a very fancy apparatus. Instead we have a simple mechanical fixture where we can measure one degree of freedom (a rotation angle) with great precision. From GPE we receive two 3D poses and compare them to find the rotation angle and then compare it with the angle read from the fixture's protractor. Note that the plane that the arm traverses when rotated (i.e., the plane of the protractor) and plane of the A4 sheet (where the polygon is drawn) are the same.

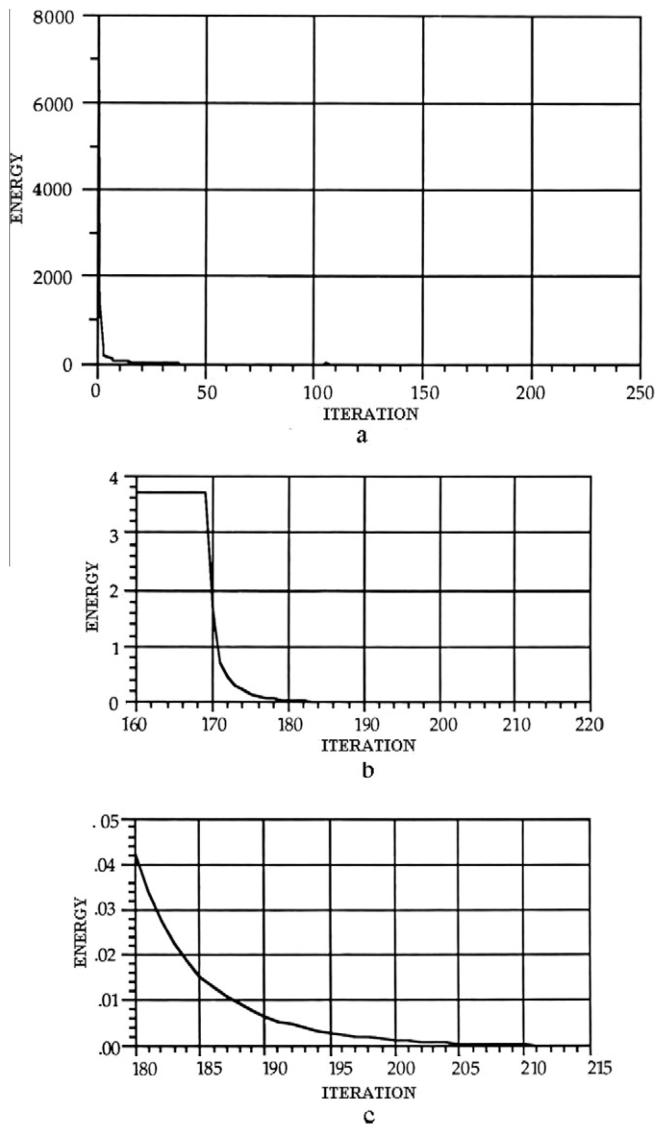


Fig. 5. Energy versus iterations. (a) Overall plot; (b) zoomed in near the last shake and (c) final slope.

In the real image experiment, the object model contains corners of the polygon. We did the feature extraction manually by pointing to the corners of the polygon with the mouse and reading the coordinates by means of image viewing software. We plan to automate feature extraction in the future. The object coordinate frame is defined by the A4 sheet. The *Z*-axis is orthogonal to the sheet and the *X* and *Y* axes are along the edges of the sheet. The directions of the *X*, *Y*, *Z* axes of the object model are expressed, respectively by $\vec{n}, \vec{s}, and\vec{a}$ in the camera coordinate frame.

To estimate the relative angle of the object in an image with respect to another image, GPE is run twice. We take vectors $\vec{n}_1, \vec{s}_1,$ and $\vec{a}_1$ estimated by GPE for the 1st image and $\vec{n}_2, \vec{s}_2,$ and $\vec{a}_2$ for the 2nd image. Then it is possible to calculate the rotation angle (denoted as θ) using Eqs. (13)–(15), when the plane of the A4 sheet is identical to the plane of the protractor's rotation plane – as in our case.

$$\theta_n = \text{acos}(\vec{n}_i \cdot \vec{n}_2) \tag{13}$$

$$\theta_s = \text{acos}(\vec{s}_1 \cdot \vec{s}_2) \tag{14}$$

$$\theta = \frac{\theta_n + \theta_s}{2} \tag{15}$$

To evaluate the accuracy of GPE, the relative angle between every image pair was measured by both GPE and protractor. The results of this experiment are given as a chart in Fig. 9. In the chart, *ACTUAL2* (horizontal) axis denotes the angle value for

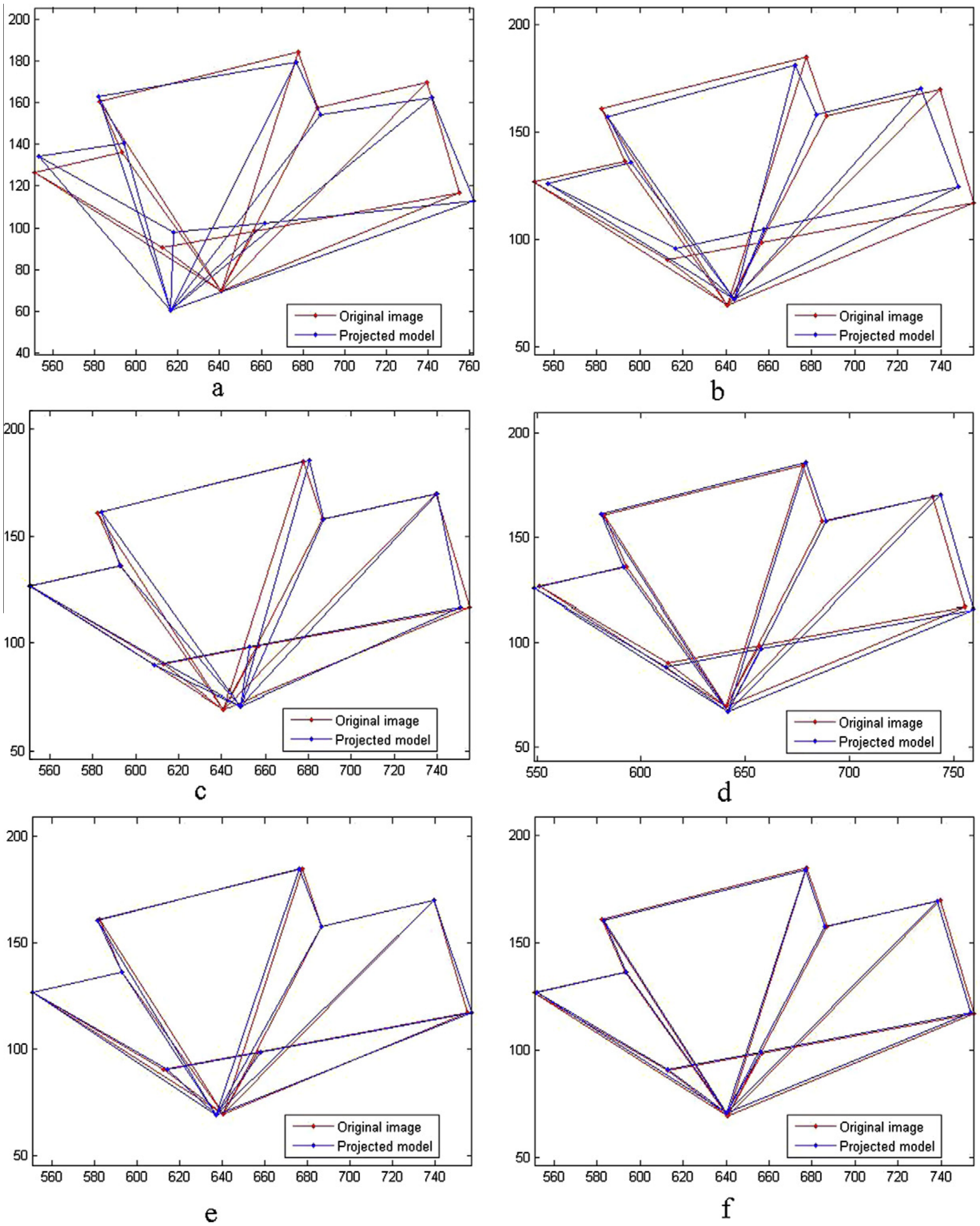


Fig. 6. Converging to optimum using GPEsoftPOSIT. (a) Object at the pose found by GPE has been projected to the image plane. (b–f) Iterations of SoftPOSIT to improve the pose.

image 2, whereas the numbers next to word ''vs'' denotes the protractor angle for *image 1*. After running GPE twice, we compute the difference between the two angles. *ESTIMATED2* (vertical) axis in Fig. 9 is the angle estimated for image 2 by GPE. That is, in fact, the difference estimated by two runs of GPE ((θ) plus the angle of image 1 from the protractor.

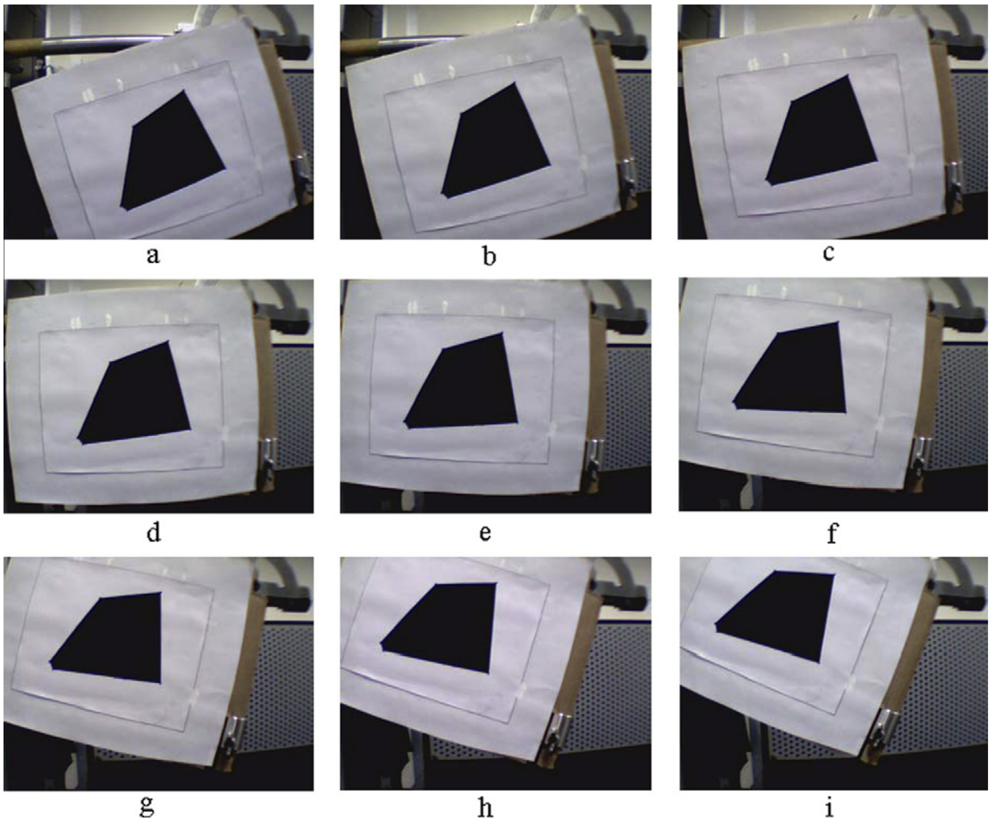


Fig. 7. Images of a real object taken at various angles: (a) 70°; (b) 75°; (c) 80°; (d) 85°; (e) 90°; (f) 95°, (g) 100°; (h) 105° and (i) 110°.

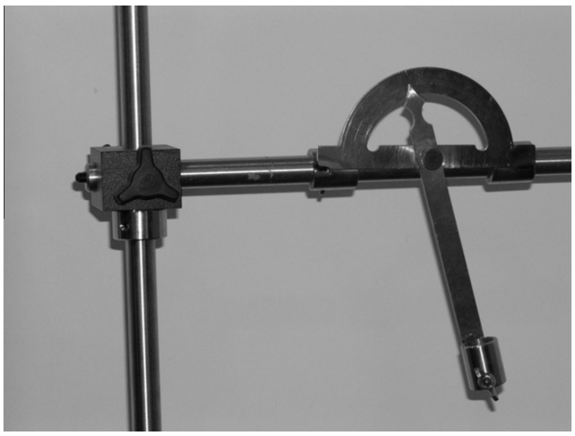
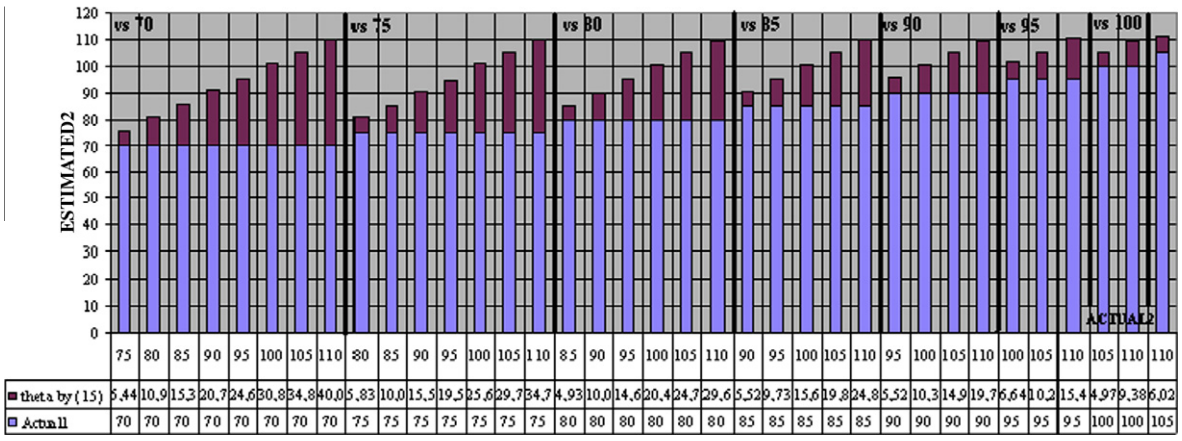


Fig. 8. The fixture used for the real image experiment.

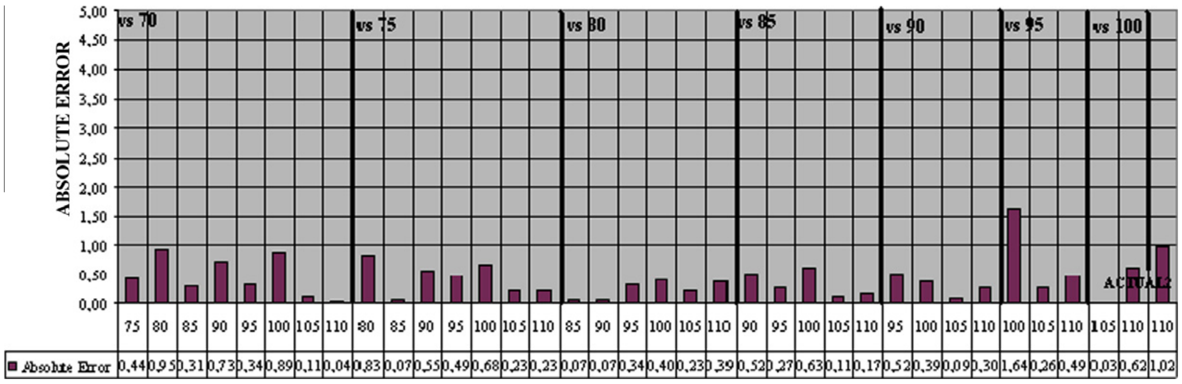
Eight groups of image pairs are included in Fig. 9:

(i) vs 70: Image 1 at 70° versus image 2 at {75°, 80°, 85°, 90°, 95°, 100°, 105°, 110°}.
(ii) vs 75: Image 1 at 75° versus image 2 at {80°, 85°, 90°, 95°, 100°, 105°, 110°}.
(iii) vs 80: Image 1 at 80° versus image 2 at {85°, 90°, 95°, 100°, 105°, 110°}.
(iv) vs 85: Image 1 at 85° versus image 2 at {90°, 95°, 100°, 105°, 110°}.
(v) vs 90: Image 1 at 90° versus image 2 at {95°, 100°, 105°, 110°}.
(vi) vs 95: Image 1 at 95° versus image 2 at {100°, 105°, 110°}.
(vii) vs 100: Image 1 at 100° versus image 2 at {105°, 110°}.
(viii) vs 105: Image 1 at 105° versus image 2 at 110°.



	75	80	85	90	95	100	105	110	80	85	90	95	100	105	110	85	90	95	100	105	110	90	95	100	105	110	95	100	105	110	100	105	110	105	110	110
■ theta by (15)	5,44	10,9	15,3	20,7	24,6	30,8	34,8	40,0	5,83	10,0	15,5	19,5	25,6	29,7	34,7	4,93	10,0	14,6	20,4	24,7	29,6	5,52	9,73	15,6	19,8	24,8	5,52	10,3	14,9	19,7	6,64	10,2	15,4	4,97	9,38	6,02
□ Actual1	70	70	70	70	70	70	70	70	75	75	75	75	75	75	75	80	80	80	80	80	80	85	85	85	85	85	90	90	90	90	95	95	95	100	100	105

Fig. 9. Estimation of relative angle of the object between two image pairs of Fig. 7.



	75	80	85	90	95	100	105	110	80	85	90	95	100	105	110	85	90	95	100	105	110	90	95	100	105	110	95	100	105	110	100	105	110	105	110	110
■ Absolute Error	0,44	0,95	0,31	0,73	0,34	0,89	0,11	0,04	0,83	0,07	0,55	0,49	0,68	0,23	0,23	0,07	0,07	0,34	0,40	0,23	0,39	0,52	0,27	0,63	0,11	0,17	0,52	0,39	0,09	0,30	1,64	0,26	0,49	0,03	0,62	1,02

Fig. 10. Absolute error in estimated relative angle in Fig. 9.

Table 1
Performance of pose estimation algorithm.

Image (°)	Number of shakes	Run-time (s)
75	69	12.6
80	97	15.2
85	104	8.3
90	99	7.9
95	71	9.6
100	69	22.1
105	60	15.6
110	63	21.3

Absolute estimation error (GPE minus protractor) is presented in Fig. 10 in degrees. The average absolute error for the 36 image pairs is 0.43°. We also present Table 1 where the run-time of the pose estimation algorithm and the number of shakes for the eight experiments where image 1 is at 70°, and image 2 is varied between 75° and 110°.

All of the experiments were carried out on a laptop running Cygwin (i.e., Linux emulator in Windows) with a 1.6GHz Pentium processor and 1GB RAM. The images that we used in this experiment have coplanar points. Since SoftPOSIT requires at least four non-coplanar points, we were unable to utilize GPEsoftPOSIT in this real image experiment. However, we have put together an extensive set of synthetic test images to evaluate random start SoftPOSIT, GPEsoftPOSIT, and GPE. Next, we present these results.

5.2. Synthetic image experiments

In this section, we show results of GPEsoftPOSIT and compare them with GPE and random start SoftPOSIT, using three randomly generated objects with 6, 10, and 15 vertices. For each object, 10 different orientations and positions were generated (using step 1 below) for each configuration. We have used up to 3 × 4 = 12 configurations (*relative distance* = 3, 7, 10; number of points occluded = 0, 1, 2, 3) with each object model.

The ratio of the distance of the object from the lens center (i.e., magnitude of the position vector) over the diameter of the object is what we call "relative distance." Note that relative distance is also roughly the reciprocal of the "viewing angle" of the object in radians. Viewing angle is the top angle of the "pencil of lines" from the lens center to the object (Fig. 1).

$$relative\_distance = \frac{|\vec{p}|}{2.object\_radius} \tag{16}$$

In order to calculate the radius of an n-point object, we first compute the centroid (also COM if all points have equal mass) of the object (denoted by $\vec{c}$). Then we sum the squares of distances from object points (denoted by $\vec{x}_k$ for object point k) to the centroid. We then take the average followed by a square-root. Hence, the object radius is, in a way, a mean object point distance from the centroid – or more accurately an *rms* (root-mean-square) displacement from the centroid – see (17).

The two steps below outline our synthetic image creation process:

True pose generation: We generate a rotation matrix, $\mathbf{R}_{\alpha\beta\delta}$, using random angles α, β, and δ between $[-\pi, \pi]$ by

$$\mathbf{R}_{\alpha\beta\delta} = \mathbf{R}_\alpha \mathbf{R}_\beta \mathbf{R}_\delta = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} \tag{17}$$

$$\mathbf{R}_\alpha = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{18}$$

$$\mathbf{R}_\beta = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{19}$$

$$\mathbf{R}_\delta = \begin{bmatrix} \cos\delta & -\sin\delta & 0 \\ \sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{20}$$

where $\mathbf{R}_\alpha$ shows the rotation about X-axis, $\mathbf{R}_\beta$ rotation about Y-axis, and $\mathbf{R}_\delta$ rotation about Z-axis. Then a random position vector $\vec{p}$ with a length of $2.relative\_distance.object\_radius$ is generated.

Image construction: With 3D object points in the object model and the true pose parameters, we compute the coordinates of every object point with respect to the camera coordinate frame ($\vec{v}_c$) using the transformation $\vec{v}_c^T = \vec{v}_o^T \mathbf{R}_{\alpha\beta\delta} + \vec{p}$, where $\vec{v}_o$ is the coordinates of the object point in the object coordinate frame and superscript of T indicates transposed, hence, row vectors. Finally, we project object points on to the 2D image plane.

For each configuration of each object model, we have constructed 10 images – each at a different and random pose. Then, we have evaluated the performance of GPE, random start SoftPOSIT, and GPEsoftPOSIT by comparing the absolute deviation of each method from the true pose.

For random start SoftPOSIT, we ran SoftPOSIT 500 times with each test image. Whenever the fraction of the number of 3D–2D matched points is greater than 0.7 and SoftPOSIT is able to converge to a pose, random start SoftPOSIT terminates and outputs the guess for the pose of the object. On the other hand, if no convergence is achieved even after 500th run, we terminate the random start SoftPOSIT algorithm. In random start SoftPOSIT, since we do not know whether the randomly generated initial pose is close to the true pose, we set the parameter β_0 of SoftPOSIT to 0.0001 enabling that all possibilities of correspondence are possible in the beginning.

To test GPEsoftPOSIT algorithm, SoftPOSIT is run only once using the pose that is output by GPE as the initial pose. In GPEsoftPOSIT, the parameter β_0 is set to 0.1 since we know that the initial pose is close to the actual pose. If the fraction of the number of 3D–2D matched points is greater than 0.7 and SoftPOSIT is able to converge to a pose, then it outputs the improved guess for the pose of the object. If it fails to get convergence, then the initial pose, which is the result of GPE, is presented as the output. (A predetermined initial pose is used for GPE whether it is used stand-alone or as part of GPEsoftPOSIT.)

If the true pose of the object is expressed by vectors $\vec{n}, \vec{s}, \vec{a}, and \vec{p}$, and the estimated pose is expressed by vectors $\vec{n}_G, \vec{s}_G, \vec{a}_G, and \vec{p}_G$, the orientation errors, $\Delta\theta_n$,$\Delta\theta_s$, and $\Delta\theta_a$ and the relative position error, ε_p are calculated using the following equations:

$$\Delta\theta_n = \text{acos}(\vec{n}.\vec{n}_G) \tag{21}$$

$$\Delta\theta_s = \text{acos}(\vec{s}.\vec{s}_G) \tag{22}$$

$$\Delta\theta_a = \text{acos}(\vec{a}.\vec{a}_G) \tag{23}$$

Table 2
Comparison of GPEsoftPOSIT, GPE, random start softPOSIT.

# obj. pts	# occ. pts	Rel. Dist.	GPEsoftPOSIT					GPE					Random start softPOSIT					
			$\overline{\Delta\theta_n}$	$\overline{\Delta\theta_s}$	$\overline{\Delta\theta_a}$	$\overline{\varepsilon_p}$	CPU	$\overline{\Delta\theta_n}$	$\overline{\Delta\theta_s}$	$\overline{\Delta\theta_a}$	$\overline{\varepsilon_p}$	CPU	$\overline{\Delta\theta_n}$	$\overline{\Delta\theta_s}$	$\overline{\Delta\theta_a}$	$\overline{\varepsilon_p}$	CPU	# fails
6	0	3	0.81	0.89	0.47	0.01	28.03	3.11	3.89	3.30	0.06	28.01	–	–	–	–	95.88	10
		7	0.14	0.28	0.28	0.01	28.79	1.79	2.89	3.36	0.10	28.77	–	–	–	–	96.88	10
		10	3.07	2.05	3.14	0.14	26.72	4.85	3.72	4.63	0.22	26.70	–	–	–	–	96.00	10
	1	3	30.24	30.45	13.87	0.46	21.60	33.14	33.08	16.09	0.53	19.45	–	–	–	–	93.11	10
		7	49.59	55.42	61.29	0.95	19.50	50.13	56.22	61.97	1.02	18.92	–	–	–	–	94.09	10
		10	22.01	21.70	23.21	1.45	19.17	22.58	22.52	23.71	1.49	18.72	–	–	–	–	93.94	10
10	0	3	2.59	3.24	2.64	0.02	24.37	5.28	6.07	5.63	0.03	24.34	0.02	0.02	0.02	0.00	51.81	4
		7	3.79	4.49	4.24	0.07	25.11	7.19	8.27	8.43	0.14	25.10	0.02	0.03	0.03	0.00	68.85	6
		10	2.91	3.08	3.46	0.10	26.62	5.80	5.64	5.96	0.23	26.60	–	–	–	–	99.08	10
	1	3	1.96	2.63	2.15	0.02	22.58	5.84	6.76	4.57	0.04	22.57	0.02	0.02	0.02	0.01	67.31	7
		7	1.85	2.52	2.72	0.05	23.21	3.11	4.49	4.40	0.14	23.21	0.04	0.05	0.06	0.00	69.65	7
		10	2.95	2.96	2.08	0.20	23.32	6.65	6.08	4.62	0.36	23.29	0.18	0.19	0.21	0.00	84.99	9
	2	3	3.83	3.10	3.44	0.02	20.75	6.07	5.32	6.25	0.04	20.73	89.86	1.52	88.50	0.01	77.29	8
		7	4.16	4.04	4.67	0.13	21.62	5.84	5.58	6.91	0.21	21.60	0.01	0.01	0.02	0.00	79.01	8
		10	6.87	7.56	6.19	0.48	21.78	8.76	9.69	7.84	0.58	21.76	–	–	–	–	95.23	10
	3	3	5.11	8.58	8.93	0.06	19.01	6.49	9.64	10.45	0.07	18.97	0.05	0.06	0.08	0.00	93.02	9
		7	3.50	4.91	4.86	0.32	19.84	4.05	5.45	5.54	0.33	19.82	–	–	–	–	94.83	10
		10	5.42	5.00	2.54	0.24	20.05	8.52	8.20	4.75	0.41	20.05	–	–	–	–	95.03	10
15	0	3	0.97	0.62	0.98	0.02	75.33	7.44	4.63	7.36	0.07	75.31	0.00	0.00	0.00	0.00	51.43	4
		7	1.11	1.45	1.35	0.04	69.97	4.78	4.66	6.01	0.08	69.95	0.01	0.01	0.00	0.00	84.99	8
		10	1.85	1.72	1.76	0.03	66.40	4.62	4.23	4.04	0.14	66.38	0.06	0.06	0.03	0.01	90.23	9
	1	3	1.31	1.39	0.69	0.01	61.06	6.09	6.20	5.77	0.04	58.06	0.00	0.00	0.00	0.00	90.02	9
		7	1.35	1.16	1.42	0.02	60.91	5.19	4.99	5.11	0.07	58.61	0.01	0.01	0.01	0.00	87.89	8
		10	3.23	2.87	2.47	0.06	62.67	6.16	5.75	5.23	0.11	60.55	0.02	0.02	0.02	0.00	91.22	9
	2	3	1.38	1.23	0.52	0.02	53.99	8.90	6.95	8.36	0.08	50.64	0.01	0.01	0.01	0.00	88.66	8
		7	5.70	4.70	6.41	0.07	54.25	8.20	6.63	9.07	0.12	51.29	0.01	0.01	0.01	0.00	87.94	9
		10	2.85	2.18	2.82	0.20	53.77	7.81	6.52	8.09	0.34	51.53	–	–	–	–	95.13	10
	3	3	4.62	4.88	2.75	0.05	46.50	11,13	7.72	9.11	0.11	44.50	0,01	0,01	0,01	0,00	86.15	9
		7	6.68	4.65	8.36	0.11	47.50	10.06	7.96	11.04	0.16	45.50	0.06	0.06	0.04	0.00	78.33	8
		10	3.15	2.66	2.58	0.13	48.30	6.68	5.20	6.20	0.26	46.30	0.09	0.08	0.05	0.01	78.58	8

Angle unit, degree; position unit, multiples of object diameter; time unit, second.

$$\varepsilon_p = \frac{|\vec{p}_G - \vec{p}|}{2 \cdot object_{radius}} \tag{24}$$

The results of GPE, GPEsoftPOSIT, and random start SoftPOSIT are presented in Table 2 where a total of 30 different configurations are shown. For each configuration, we have 10 different tests (i.e., images). Hence, the total number of experiments is 300. The averages of orientation errors ($\Delta\theta_n$, $\Delta\theta_s$, and $\Delta\theta_a$) and average of relative position errors (ε_p) obtained from the 10 tests for each configuration are listed in Table 2 as $\overline{\Delta\theta_n}$, $\overline{\Delta\theta_s}$, and $\overline{\Delta\theta_a}$ in degrees and $\overline{\varepsilon_p}$ in multiples of the object diameter.

Random start SoftPOSIT was unable to converge to a pose in 257 tests out of 300 (see the rightmost column in Table 2 titled #fails). Having a number listed there does not mean random start SoftPOSIT converged to a pose in all cases. A row in Table 2 reports average numbers for the corresponding configuration out of the tests that converged.

On the other hand, GPE (and hence GPEsoftPOSIT) is an algorithm that does not have convergence problems. Hence, it converged in all of the 300 tests. It produced good results in all experiments except the 6/1 (#object points/#occluded points) case. However, note that random start SoftPOSIT was not able to converge even in the 6/0 case. GPE and GPEsoftPOSIT worked fine in 6/0 case. Since we cannot obtain acceptable results in 6/1, naturally we could not obtain meaningful results

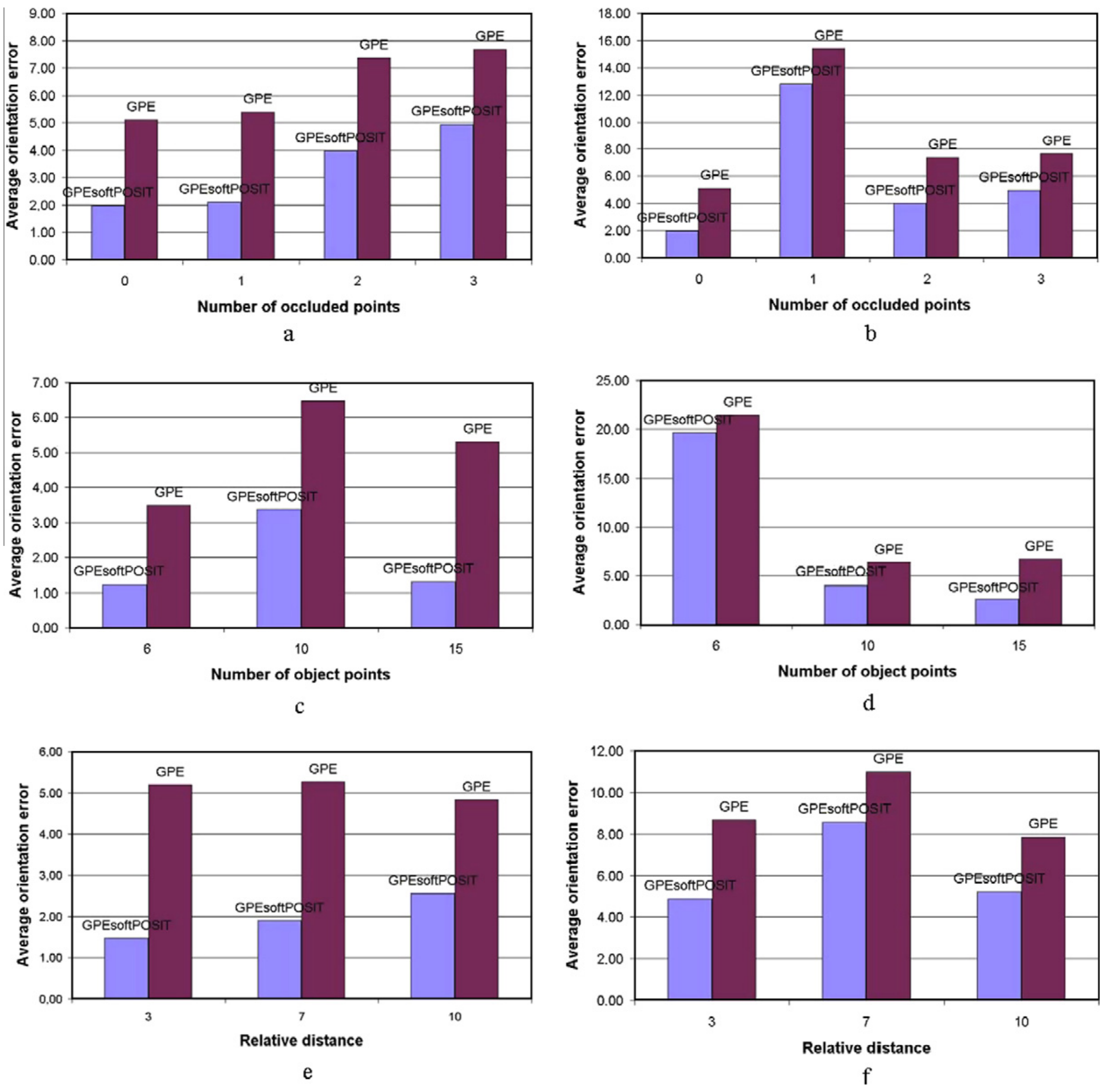


Fig. 11. Average orientation error (in degrees) versus: (a) number of occluded points (without 6/1); (b) number of occluded points (with 6/1); (c) number of object points (without occlusion); (d) number of object points (for all tests); (e) relative distance (without occlusion) and (f) relative distance (for all tests).

with 6/2 and 6/3. This is true for all three methods. Therefore, we exclude 6/2 and 6/3 by having 30 configurations instead of 36. However, it is understandable that any occlusion affects the results in the 6-point object case. Since there are very few points, every point is critical in finding the true pose.

GPE is especially very good in finding the position. Note that we evaluate position error in relative terms to the size of the object. That is why ε_p, as shown in equation (25), is calculated by normalizing the position error by the object diameter (i.e., twice the object radius).

Looking at the error figures in Table 2, the best performing algorithm is apparently GPEsoftPOSIT. Figs. 11 and 12 try to summarize the tendencies of the orientation and position error, respectively. Before we look at the bar graphs in Figs. 11 and 12 more closely, let us state the expected tendencies:

- Error should go up as the number of occluded points goes up.
- Error should go down as the number of object points goes up.
- Error should go up as the relative distance goes up.

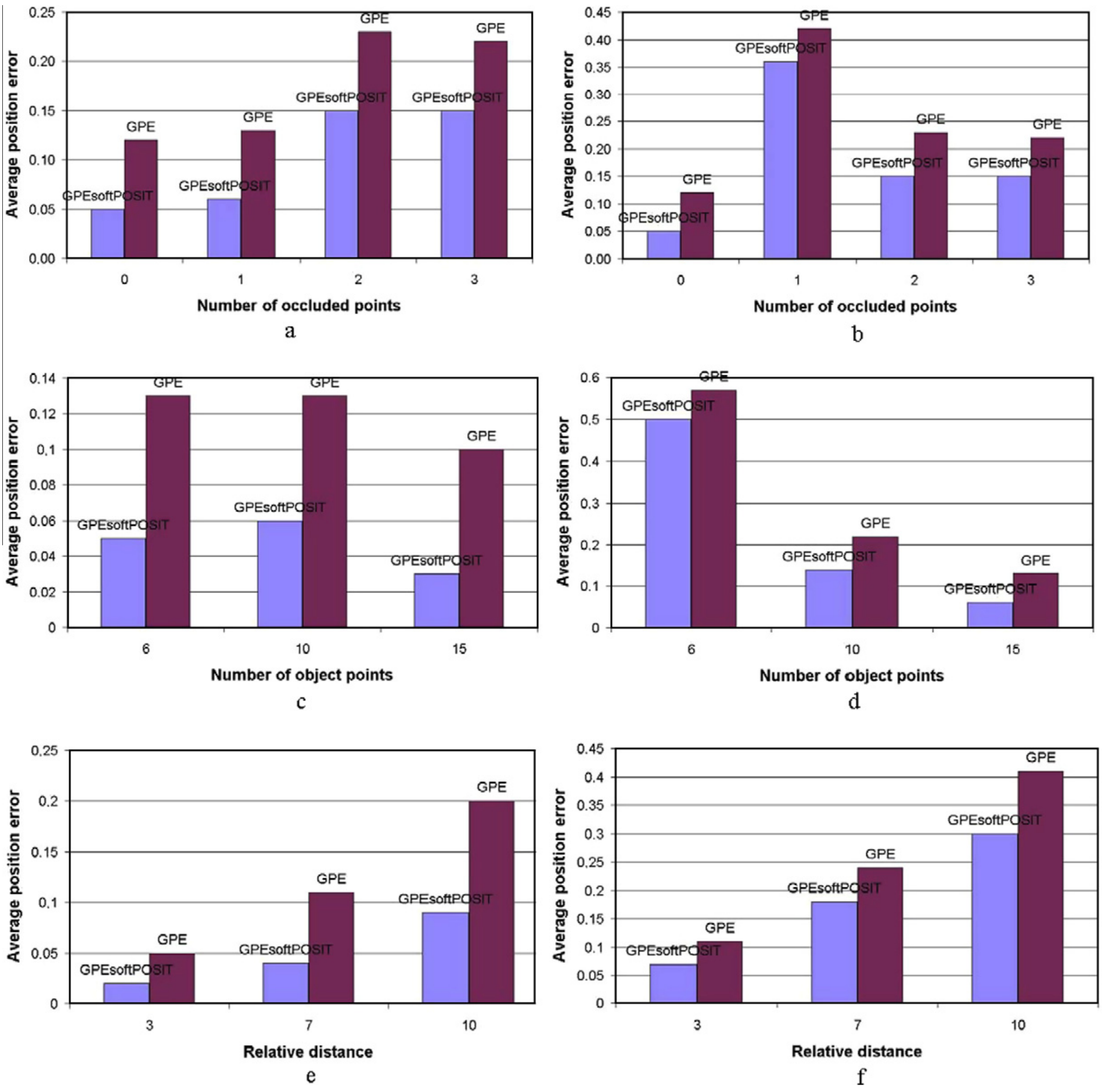


Fig. 12. Average position error (error/object_diameter – see Eq. (25)) versus: (a) number of occluded points (without 6/1); (b) number of occluded points (with 6/1); (c) number of object points (without occlusion); (d) number of object points (for all tests); (e) relative distance (without occlusion) and (f) relative distance (for all tests).

If we exclude the gross errors introduced by 6/1, we can more or less see that error goes up as the number of occluded points goes up (Figs. 11 and 12a). In Fig. 12c and d, as expected we see that (relative) position error goes down as the number of object points goes up. In Fig. 11c and d, this tendency is not obvious for orientation error. This may be due to the fact that our images are synthetic. In real-life (i.e., real images), more object points compensate for the inaccuracies introduced by pixelization, imaging system, and feature detection. Similarly in Figs. 11 and 12e and f, we see the expected tendency in position error graphs but not orientation error graphs. We have plotted errors both for no occlusion and also for all tests. If no occlusion tests have an error of x (Figs. 11 and 12c and e), the average of all tests seem to be around $2x$ (Figs. 11 and 12d and f).

CPU times are listed in Table 2. GPE and GPEsoftPOSIT almost have equal times as GPEsoftPOSIT runs SoftPOSIT only once. GPE and GPEsoftPOSIT take between half to 1 min, while 500 runs of SoftPOSIT in random start SoftPOSIT take between 50 and 95 s. The computational complexity of GPE is governed by the number of iterations and number of feature points. In most tests, GPE went up to the preset maximum of 50,000 iterations. The most compute-intensive operation inside a GPE iteration is the computation of correspondence (see the pairing algorithm in Section II). The complexity of this algorithm is, on the other hand, dominated by the sorting of LOS and object point distances, which is in the order of $n^2 \log n$, where n is the number of object points.

6. Conclusion and future work

We have presented an n-point feature-based correspondenceless pose estimation method (called GPE for gravitational pose estimation) for rigid objects, which uses a single perspective image with no restrictions on the object's pose. GPE is based on an intuition from classical mechanics.

We have demonstrated good results with both synthetic and real-world images. Our real-world images were taken with a cheap webcam with an obvious fish-eye effect (as can be seen in Fig. 7,) and we still have quality results. We have successfully tested GPE against occlusion as well – with a test suite of 300 synthetic images.

In this paper, we integrate GPE with SoftPOSIT for fine-tuning of the pose. After the integration, we achieve nearly zero-error in 62% of the test cases (185 out of the 300). On the average (including up to 30% occlusion cases but excluding the 6/1 case,) GPE finds the orientation within 6° and the position within 17% of the object's diameter. GPEsoftPOSIT, on the other hand, is within 3° and 10% of the object's diameter in the same set of tests.

In future work, we plan to include automatic feature extraction in real image experiments, use an automatically calculated initial pose, and further improve on run-time. We will also look at incremental techniques for real-time operation, which start with the correspondence information found for the prior frame.

Acknowledgment

The authors wish to thank Daniel DeMenthon for making SoftPOSIT publicly available and for answering our questions regarding SoftPOSIT.

References

[1] Besl PJ, McKay ND. A method for registration of 3-D shapes. IEEE Trans Pattern Anal Mach Intell 1992;14(2):239–56.
[2] Besl PJ. Geometric modeling and computer vision. Proc IEEE 1988;76(8):936–58.
[3] Besl PJ. The free-form surface matching problem, In: Freeman H, editor. Machine vision for three-dimensional scenes, Academic Press; 1990. p. 25–71.
[4] Arun KS, Huang TS, Blostein SD. Least square fitting of two 3-D point sets. IEEE Trans Pattern Anal Mach Intell 1987;9(5):698–700.
[5] Li H, Hartley R. The 3D-3D registration problem revisited. In: Proc Int Conf Comput Vision 2007:1–8.
[6] Liu Y, Wang Y, Evaluating 3D-2D correspondences for accurate camera pose estimation from a single image. In: Proc IEEE Int Conf Systems Man Cybernetics 2003;23:703–8.
[7] Liu Y. Eliminating false matches for the projective registration of free-form surfaces with small translational motions. IEEE Trans Systems Man Cybernetics 2005;35(3):607–24.
[8] Khotanzad A, Liou J-H. Recognition and pose estimation of unoccluded three-dimensional objects from a two-dimensional perspective view by banks of neural networks. IEEE Trans Neural Networks 1996;7(4):897–906.
[9] Haralick RM, Joo H, Lee C, Zhuang X, Vaidya VG, Kim MB, et al. IEEE Trans Systems Man Cybernetics 1989;19(6):1426–46.
[10] Rosin PL. Robust pose estimation. IEEE Trans Systems Man Cybernetics 1999;29(2):297–302.
[11] Quan L, Lan Z, Linear N-point camera pose determination. IEEE Trans Pattern Anal Mach Intell 1999;21(8):774–80.
[12] Liu Y, Rodrigues MA. Statistical image analysis for pose estimation without point correspondences. Pattern Recogn Lett 2001;22(11):1191–206.
[13] Yu YK, Wong KH, Chang MMY. Pose estimation for augmented reality applications using genetic algorithm. IEEE Trans Systems Man Cybernetics 2005;35(6):1295–301.
[14] Zhang Z, Zhu D, Zhang J. An improved pose estimation algorithm for real-time vision applications. In: Proc. IEEE Int. Conf. Communications, Circuits and Systems: in; 2006. pp. 402–406.
[15] Ansar A, Daniilidis K. Linear pose estimation from points and lines. IEEE Trans Pattern Anal Mach Intell 2003;25(5):578–89.
[16] Dhome M, Richetin M, Lapreste J-T, Rives G. Determination of the attitude of 3-D objects from a single perspective view. IEEE Trans Pattern Anal Mach Intell 1989;11(12):1265–1278.
[17] Chang CC, Tsai WH. Reliable determination of object pose from line features by hypothesis testing. IEEE Trans Pattern Anal Mach Intell 1999;21(11):1235–41.
[18] Qin L, Zhu F. The judgment method for the unique solution of real-time pose estimation from particular line correspondences. In: Proc IEEE Int Conf Mechatronics Automat 2006:1216–1220.
[19] Wang G, Wu J, Ji Z. Single view based pose estimation from circle or parallel lines. Pattern Recogn Lett 2008;29(7):977–85.

[20] Jurie F. Solution of the simultaneous pose and correspondence problem using Gaussian error model. Comput Vision Image Understanding 1999;73(3):357–73.
[21] DeMenthon D, David P. Model-based object pose in 25 lines of code. Int J Comput Vision 1995;15(1–2):123–41.
[22] Fiore PD. Efficient linear solution of exterior orientation. IEEE Trans Pattern Anal Mach Intell 2001;23(2):140–8.
[23] Lin Z, Lee H, Huang TS. Finding 3D point correspondences in motion estimation. In: Proc IEEE Int Conf Pattern Recogn 1986:303–305.
[24] Aloimonos J, Herve J. Correspondenceless stereo and motion: planar surfaces. IEEE Trans Pattern Anal Mach Intell 1990;12(5):504–10.
[25] David P, DeMenthon D, Duraiswami R, Samet H. SoftPOSIT: simultaneous pose and correspondence determination. Int J Comput Vision 2004;59(3):259–84.
[26] Fischler MA, Bolles RC. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 1981;24:381–95.
[27] Enqvist O, Kahl F. Robust optimal pose estimation. In: Proc Eur Conf Comput Vision 2008:141–153.
[28] Huttenlocher DP, Ullman S. Object recognition using alignment. In: Proc Int Conf Comput Vision 1987:102–111.
[29] Jacobs D, Basri R. 3-D to 2-D pose determination with regions. Int J Comput Vision 1999;34(2-3):123–45.
[30] Gold S, Rangarajan A, Lu C-P, Pappu S, Mjolsness E. New algorithms for 2D and 3D point matching: pose estimation and correspondence. Pattern Recogn 1998;31(8):1019–31.
[31] Ugurdag HF, Gören S, Canbay F. Correspondenceless pose estimation from a single 2D image using classical mechanics. In: Proc IEEE Int Symp Comput Inform Sci 2008;132:1–6.