A Comparison of Iterative 2D-3D Pose Estimation Methods for Real-Time Applications

Daniel Grest, Thomas Petersen and Volker Krüger

Aalborg University Copenhagen, Denmark
Computer Vision Intelligence Lab
{dag,vok}@cvmi.aau.dk

Abstract. This work compares iterative 2D-3D Pose Estimation methods for use in real-time applications. The compared methods are available for public as C++ code. One method is part of the openCV library, namely POSIT. Because POSIT is not applicable for planar 3D-point configurations, we include the planar POSIT version. The second method optimizes the pose parameters directly by solving a Non-linear Least Squares problem which minimizes the reprojection error. For reference the Direct Linear Transform (DLT) for estimation of the projection matrix is inlcuded as well .

1 Introduction

This work deals with the 2D-3D pose estimation problem. Pose Estimation has the aim to find the rotation and translation between an object coordinate system and a camera coordinate system. Given are correspondences between 3D points of the object and their corresponding 2D projections in the image. Additionally the internal parameters focal length and principal point have to be known.

Pose Estimation is an important part of many applications as for example structure-from-motion [11], marker-based Augmented Reality and other applications that involve 3D object or camera tracking [7]. Often these applications require short processing time per image frame or even real-time constraints[11]. In that case pose estimation algorithms are of interest, which are accurate and fast. Often, lower accuracy is acceptable, if less processing time is used by the algorithm. Iterative methods provide this feature.

Therefore we compare three popular methods with respect to their accuracy under strict time constraints. The first is POSIT, which is part of openCV [6]. Because POSIT is not suited for planar point configurations, we take the planar version of POSIT also into the comparison (taken from [2]. The second method we call CamPoseCalib (CPC) from the class name of the BIAS library [8]. The third method is the Direct Linear Transform for estimation of the projection matrix (see section 2.3.2 of [7]), because it is well known, used often as a reference [9] and easy to implement.

Even though pose estimation is studied long since, new methods have been developed recently. In [9] a new linear method is developed and a comparison is given, which focuses on linear methods. We compare here iterative algorithms, which are available in C++, under the constraint of fixed computation time as required in real-time applications.

2 2D-3D Pose Estimation

Given are correspondences between 3D-points $\boldsymbol{p}_i$, which project into a camera image at position $\boldsymbol{p'}_i$ (see Fig. 1). Pose estimation from these 2D-3D correspondences is about finding the rotation and translation between camera and object coordinate systems.

2.1 CamPoseCalib (CPC)

The approach of *CamPoseCalib* is to estimate the relative rotation and translation of an object from an initial position and orientation (pose) to a new pose. The correspondences $(\boldsymbol{p}_i, \tilde{\boldsymbol{p}}'_i)$ are given for the new pose. Figure 1 illustrates this. The method was originally published in [1]. Details about the implementation used can be found in [5].

The algorithm can be formulated as a non-linear least squares problem, which minimizes the reprojection error d:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} (\boldsymbol{r}_i(\boldsymbol{\theta}))^2 \qquad (1)$$

for m correspondences. The residui functions $\boldsymbol{r}_i(\theta)$ represent the reprojection error $d = \boldsymbol{r}_i(\theta)^2 = r_x^2 + r_y^2$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z, \theta_\alpha, \theta_\beta, \theta_\gamma)^T$ are the 6 pose parameters, three for translation and three angles of rotation around the world axes. More specifically, the residui functions give the difference between moved, projected 3D point $\boldsymbol{m'}(\boldsymbol{p}_i, \boldsymbol{\theta})$ and the target point:

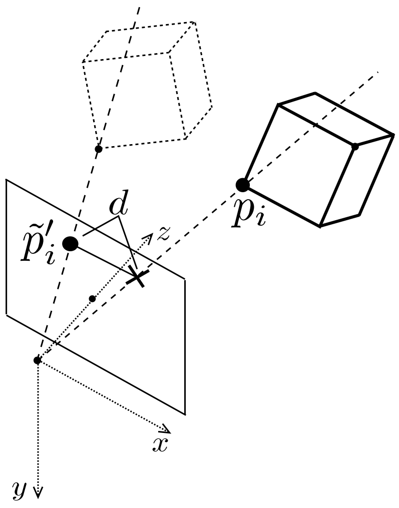

Fig. 1. *CamPoseCalib* estimates the pose by minimizing the reprojection error d between initial projected points from given correspondences $(\boldsymbol{p}_i, \tilde{\boldsymbol{p}}'_i)$

$$\boldsymbol{r}_i(\boldsymbol{\theta}) = \boldsymbol{m'}(\boldsymbol{p}_i, \boldsymbol{\theta}) - \tilde{\boldsymbol{p}}' \qquad (2)$$

The projection with pixel scales s_x, s_y and principal point $(c_x, c_y)^T$ is:

$$\boldsymbol{m'}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} s_x \frac{m_x(\boldsymbol{p},\boldsymbol{\theta})}{m_z(\boldsymbol{p},\boldsymbol{\theta})} + c_x \\ s_y \frac{m_y(\boldsymbol{p},\boldsymbol{\theta})}{m_z(\boldsymbol{p},\boldsymbol{\theta})} + c_y \end{pmatrix} \qquad (3)$$

where $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{p}) = (m_x, m_y, m_z)^T$ is the rigid motion in 3D:

$$\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{p}) = (\theta_x, \theta_y, \theta_z)^T + R_x(\theta_\alpha) R_y(\theta_\beta) R_z(\theta_\gamma) \boldsymbol{p} \qquad (4)$$

In order to avoid Euler angle problems, a compositional approach is used, that accumulates a rotation matrix during the overall optimization, rather than the rotation angles around camera axes x, y, z, which are estimated each iteration. More details in page 38-43 of [5].

The solution to the optimization problem is found by the *Levenberg-Marquardt (LM)* algorithm, which estimates the change in parameters in each iteration by:

$$\boldsymbol{\Delta\theta} = -(J^T J + \lambda I)^{-1} J^T \boldsymbol{r}(\boldsymbol{\theta_t}) \tag{5}$$

where I is the identity matrix and J is the Jacobian with the partial derivatives of the residui functions (see page 21 of [5]).

The inversion of $J^T J$ requires $det(J^T J) > 0$, which is achieved by 3 correspondences, because each correspondence gives two rows in the Jacobian and there are 6 parameters. The configuration requirement of 3D and 2D points is, that neither of them are lying on a line. However, due to the LM extension a solution that minimizes the reprojection error is always found, even for a single correspondence. Of course it will not give the correct new pose, but it returns a pose which is close to the initial pose.

The implementation in BIAS [8] also allows to optimize the internal camera parameters and has the option to estimate an initial guess, both is not used within this comparison.

2.2 POSIT

The second pose estimation algorithm uses a scaled orthographic projection (SOP), which resembles the real perspective projection at convergence. The SOP approximation leads to a linear equation system, which gives the rotation and translation directly , without the need of a starting pose. A scale value is introduced for each correspondence, which is iteratively updated. We give a brief overview of the method here. More details about POSIT can be found in [4, 3].

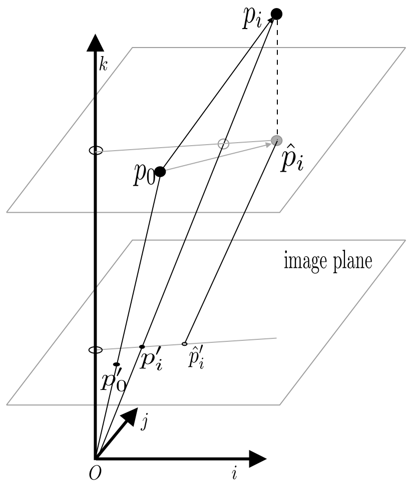

Fig. 2. POSIT estimates the pose by using a scaled orthographic projection (SOP) from given correspondences $\boldsymbol{p}_i, \boldsymbol{p'}_i$. The SOP of $\boldsymbol{p}_i$ is here shown as $\boldsymbol{\hat{p}'}_i$ with a scale value of 0.5.

Figure 2 illustrates this. The correspondences are $\boldsymbol{p}_i, \boldsymbol{p'}_i$. The SOP of $\boldsymbol{p}_i$ is here shown as $\boldsymbol{\hat{p}'}_i$ with a scale value of 0.5. The POSIT algorithm estimates the rotation by finding the values for $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ in the object coordinate system, whose origin is $\boldsymbol{p}_0$. The translation between object and camera system is $O\boldsymbol{p_0}$.

For each SOP 2D-point a scale value can be found such that the SOP $\boldsymbol{\hat{p}'}_i$ equals the correct perspective projection $\boldsymbol{p'}_i$. The

POSIT algorithm refines iteratively these scale values. Initially the scale value (w in the following) is set to one.

The POSIT algorithm works as follows:

1. Initially set the unknown values $w_i = 1$ for each correspondence.
2. Estimate pose parameters from the linear equation system
3. Estimate new values w_i by $w_i = \frac{\boldsymbol{p}_i^T \boldsymbol{k}}{t_z} + 1$
4. Repeat from step 2 until the change in w_i is below a threshold or maximum iterations are reached

The initially chosen $w_i = 1$ approximates the real configuration of camera position and scene points well, if the fraction of object elongation to camera distance is small.

If the 3D points lie in one plane the POSIT algorithm needs to be altered. A description of the co-planar version of POSIT can be found in [10].

3 Experiments

There are several experiments on synthetic data conducted, whose purpose is to reveal the advantages and disadvantages of the different methods. We use implementations as available for the public for download of *CamPoseCalib* [8] and the two *POSIT* methods from Daniel DeMenthons homepage [2]. The C++ sources are compiled with *Microsoft's Visual Studio 2005 C++ compiler* in standard release mode settings. The POSIT method is also part of openCV [6]. Experiments showed, that the openCV version is about two times faster than our compilation. However we chose to use our self compiled version, because we want to compare the algorithms rather than binary realeases or compilers.

In order to resemble a realistic setup, we chose the following values for all experiments. Some values are changed as stated in the specific tests.

- 3D points are randomly distributed in a 10x10x10 box
- camera is positioned 25 units away, facing the box
- internal camera parameters are $s_x = s_y = 882$, $c_x = 600$ and $c_y = 400$, which corresponds to a real camera with 49 degree opening angle in y-direction and an image resolution of 1200x800 pixels
- the number of correspondences is 10.
- Gaussian noise is added to the 2D positions with a variance of 0.2 pixels
- each test is run 100 times with varying 3D points

The accuracy is measured in the following tests by comparing the estimated translation and rotation of the camera to the known groundtruth.

The translation error is measured as the Euclidean distance between estimated camera position and real camera position divided by the distance of the camera to the center of the 3D points. For example in the first test, an translation error of 100% means 25 units difference.

The rotational error is measured as the Euclidean distance between the rotation quaternions representing the real and the estimated orientation.

3.1 Test 1: Increasing Noise

In many applications the time for pose estimation is bound by an upper limit. Therefore, we compare here the accuracy of different methods, which are given the same calculation time. The time chosen for each iterative algorithm is the same time as for the non-iterative DLT.

Normal distributed noise is added to the 2D positions with changing variance. The following settings are used:

- 2D-noise is increased from 0 to 3.3 pixels standard deviation (variance 10)
- The initial pose guess for CPC: rotation is two degrees off and position is 3.4% away from the real position
- Initial scale value of POSIT is 1 for all points
- Number of iterations for CPC is 9 and for POSIT 400

The initial guess for CPC is 2 degrees and 0.034 units off. This resembles a tracking scenario as in augmented reality applications.

In Figure 10 the accuracy of all methods is shown with boxplots. A boxplot shows the median (red horizontal line within boxes) instead of the mean, as well as the outliers (red crosses). The blue boxes denote the first and third quartile (the median is the second quartile).

The left column shows the difference in estimated camera position, the right column the difference in orientation as the Euclidian length of the difference rotation quaternion. The top row shows CPC, which accuracy is better than POSIT (middle row) and DLT (bottom row).

3.2 Test 2: Point Cloud Morphed to Planarity

In many applications the spatial configuration of the 3D points is unknown as in structure-from-motion. Especially interesting is the case, where the points lie in a plane or are close to a plane. In order to test the performance of the different algorithms, the point cloud is transformed into a plane by reducing its thickness each time by 30%.

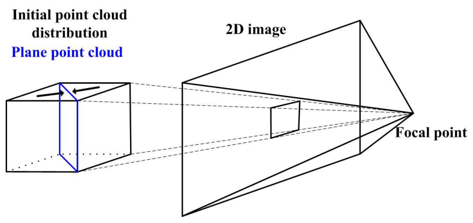

Fig. 3. Test 2: Initial box shaped point cloud distribution is changed into planarity.

Figure 3 illustrates the test. The plane is chosen not to face the camera directly (the plane normal is not aligned with the optical axis), because a correct pose is in that case also found, if the camera is on the opposite side of the plane. Because the POSIT algorithm can't handle coplanar points, the planar POSIT version is tested in addition to CPC and DLT.

Figure 4 shows the translation error versus the thickness of the box (rotational errors are similar). As visible, the DLT error increases greatly when the box gets thinner than 0.2 and fails to give correct results for a thickness smaller

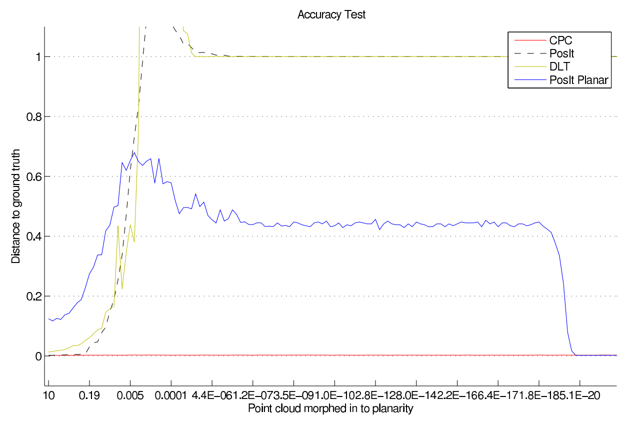

Fig. 4. Test 2: Point cloud is morphed into planarity. Shown is the mean of 100 runs.

than 1E-05 (the algorithm returns $(0, 0, 0)^T$ as position in that case). The normal POSIT algorithm performs similar to the DLT. Interesting to note is, that the planar POSIT algorithm works only correctly, if the 3D points are very close to coplanar (a thickness of 1E-20). Important is the observation, that there is a thickness range, where non of the POSIT algorithms estimates a correct result. The CPC algorithm is unaffected by a change in the thickness, while the accu-

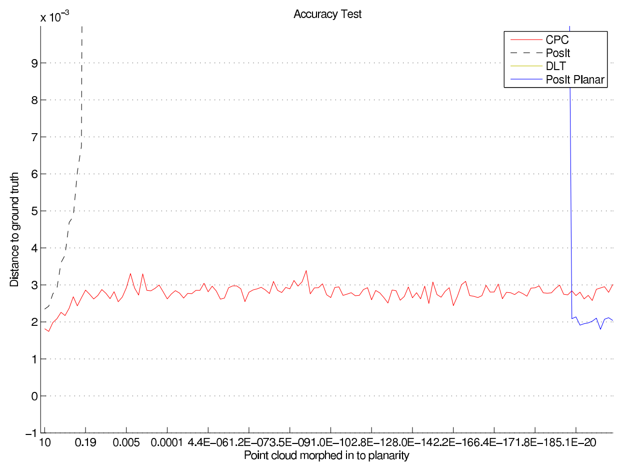

Fig. 5. Test 2: Point cloud is morphed into planarity. Shown is a closeup of the same values as in Fig. 4

racy of the planar POSIT is slightly better for nearly coplanar points as visible on in Figure 5.

3.3 Test 3: Different Starting Pose for CPC

The iterative optimization of CPC requires an initial guess of the pose. The performance of CPC depends on how close these initial parameters are to the real ones. Further there is the possibility, that CPC gets stuck in a local minimum during optimization. Often a local minimum is found, if the camera is positioned exactly on the opposite side of the 3D points.

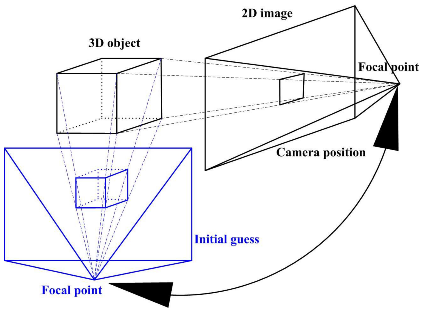

Fig. 6. Test 3 illustrated. The initial camera pose for CPC is rotated on a circle.

In order to test this dependency, the initial guess of CPC is changed, such that the camera is at the same distance to the point cloud circling around it. Figure 6 illustrates this, the orientation of the initial guess is changed such that the camera faces the point cloud at all times.

Figure 7 shows the mean and standard deviation of the rotational error (translation is similar) versus the rotation angle of the initial guess. Higher angles mean a worse starting point. The initial pose is opposite to the real one for 180 degrees. If the initial guess is worse than 90 degrees the accuracy decreases. For angles

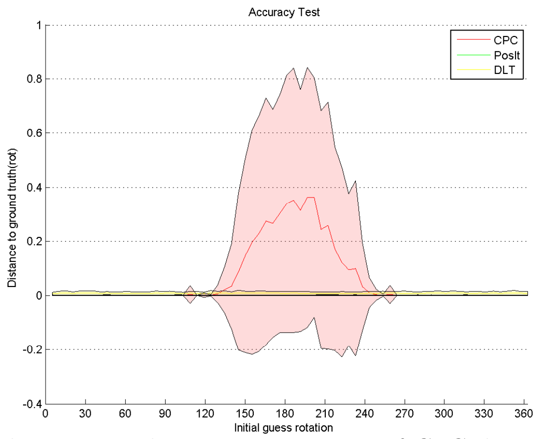

Fig. 7. Mean and variance. The rotation accuracy of CPC decreases significantly, if the starting position is on the opposite side of the point cloud.

around 180 degrees the deviation and error becomes very high, which is due to the local minimum on the opposite side. Figure 8 shows a close-up of the mean of Figure 7. Here it is visible, that the accuracy of CPC is slightly better than CPC and significantly better than DLT for angles smaller 90 degrees. Figure 9 shows the mean and standard deviation of the computation time for CPC, POSIT and

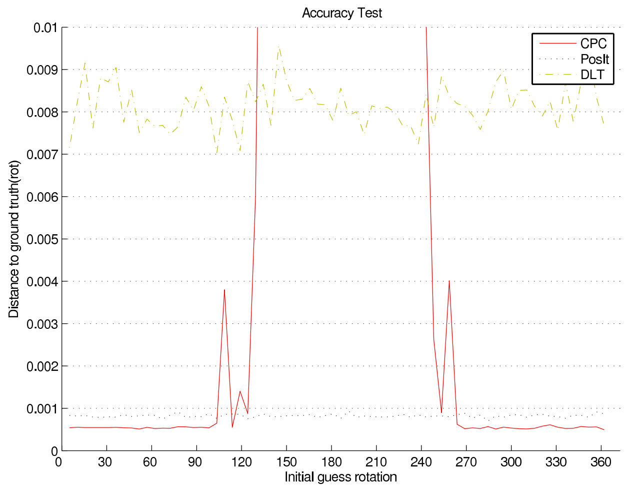

Fig. 8. A closeup of the values of figure 7. The accuracy of CPC is better than the other methods for an initial angle that is within 90 degrees of the actual rotation.

DLT. If the initial guess is worse than 30 degrees, CPC uses more time because of the LM iterations. However, even in worse cases it is only 2 times slower.

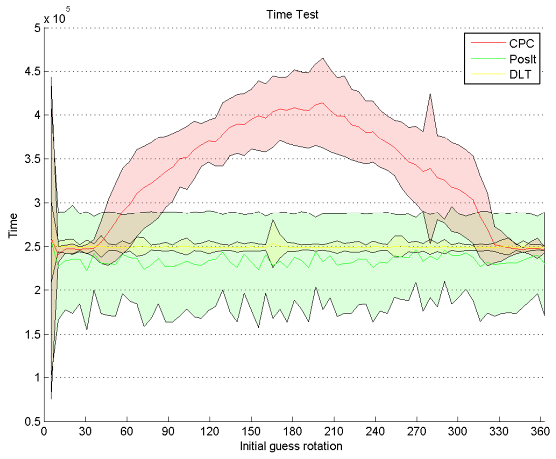

Fig. 9. Timings. Mean and variance.

From the accuracy and timing results for this test it can be concluded, that CPC is the more accurate method compared to POSIT, if given the same time and an initial guess which is within 30 degrees of the real one.

4 Conclusions

The first test showed, that CPC is more accurate than the other methods given the same computation time and an initial pose which is only 2 degrees off the real one, which is similar to the changes in real time tracking scenarios. CPC is also more accurate if the starting angle is within 30 degrees as test 3 showed. POSIT has the advantage, that it is not in the need of a starting pose and is available as an highly optimized version in openCV.

In test 2 the point cloud was changed into a planar surface. Here the POSIT algorithms gave inaccurate results for a box thickness from 0.2 to 1E-19 making the POSIT methods not applicable for applications where the 3D configuration of points is close to co-planar as in structure-from-motion applications.

The planar version of POSIT was most accurate, if the 3D points are arranged exactly in a plane. Additionally it can return 2 solutions: camera positions on both sides of the plane. This is advantageous because in applications where a planar marker is observed, the pose with smaller reprojection error is not necessarily the correct one, because of noisy measurements.

References

1. H. Araujo, R. Carceroni, and C. Brown. A Fully Projective Formulation to Improve the Accuracy of Lowe's Pose Estimation Algorithm. *Journal of Computer Vision and Image Understanding*, 70(2), 1998.
2. Daniel DeMenthon. www.cfar.umd.edu/~daniel, 2008.
3. P. David, D. Dementhon, R. Duraiswami, and H. Samet. SoftPOSIT: Simultaneous Pose and Correspondence Determination. *Int. J. Comput. Vision*, 59(3):259–284, 2004.
4. D.F. DeMenthon and L.S. Davis. Model-Based Object Pose in 25 Lines of Code. *International Journal of Computer Vision*, 15:335–343, 1995.
5. Daniel Grest. *Marker-Free Human Motion Capture in Dynamic Cluttered Environments from a Single View-Point.* PhD thesis, MIP, Uni. Kiel, Kiel, Germany, 2007.
6. Intel. openCV: Open Source Computer Vision Library. opencvlibrary.sourceforge.net, 2008.
7. Vincent Lepetit and Pascal Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–104, 2005.
8. MIP Group Kiel. Basic Image AlgorithmS (BIAS) open-source-library, C++. www.mip.informatik.uni-kiel.de, 2008.
9. F. Moreno-Noguer, V. Lepitit, and P. Fua. Accurate Non-Iterative O(n) Solution to the PnP Problem . In *ICCV*, Brazil, 2007.
10. D. Oberkampf, D. F. DeMenthon, and L.S. Davis. Iterative pose estimation using coplanar feature points. *CVIU*, 63(3):495–511, 1996.
11. B. Williams, G. Klein, and I. Reid. Real-time SLAM Relocalisation. In *Proc. of Internatinal Conference on Computer Vision (ICCV)*, Brazil, 2007.

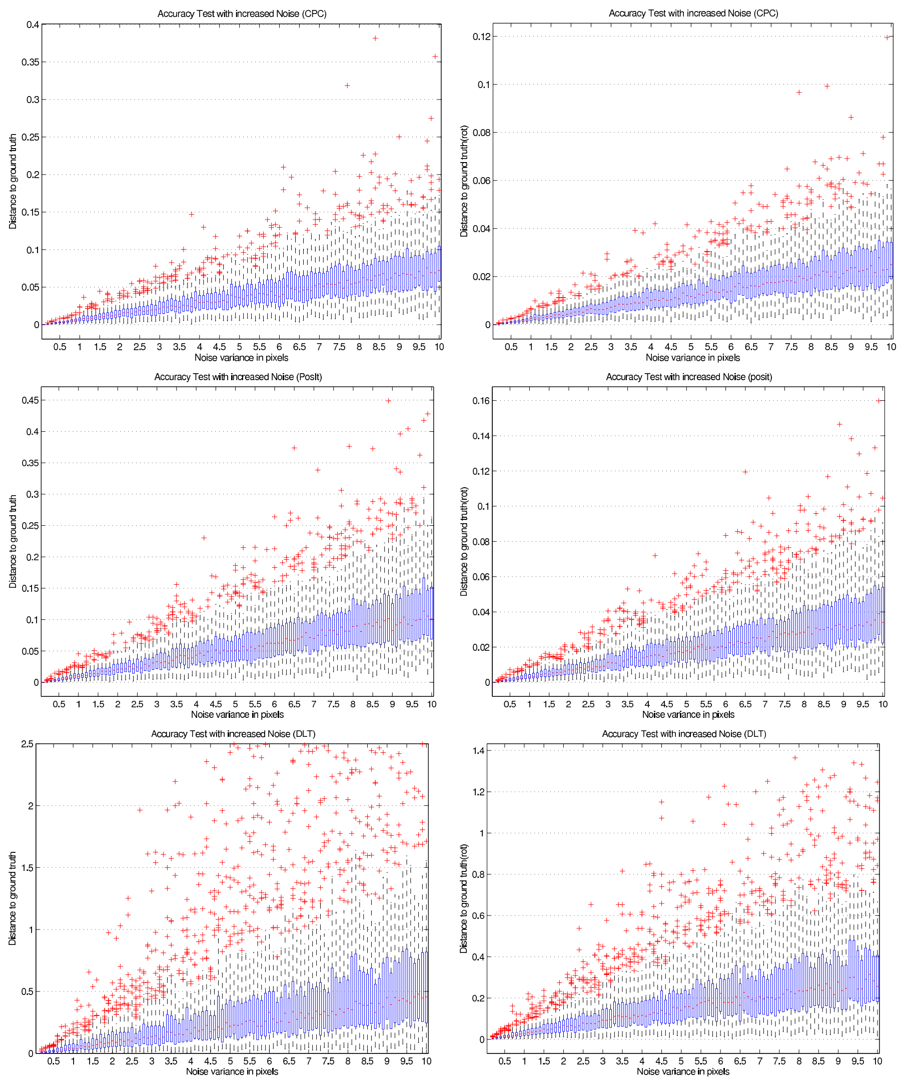

Fig. 10. Test 1: Increasing noise. Left: translation. Right: rotation. CPC (top) estimates the translation and rotation with a higher accuracy than POSIT (middle) and DLT (bottom). All algorithms used the same run-time.