A Comparison of 2D-3D Pose Estimation Methods

Thomas Petersen

Aalborg University 2008

Aalborg University - Institute for Media Technology
Computer vision and graphics
Lautrupvang 15, 2750 Ballerup, Denmark
Phone +45 9940 2480
469573.g.portal.aau.dk
www.aau.dk

Abstract

This thesis describes pose estimation as an increasingly used area in augmentation and tracking with many different solutions and methods that constantly undergo optimization and each has drawbacks and benefits. But the aim is always speed, accuracy or both when it comes to real applications. Pose estimation is used in many areas but primarily tracking and augmentation issues, where another large area of finding 2D-2D correspondences is crucial research area today. Software like ARToolKit [11] tracks a flat marker and is able to draw 3D objects on top of it for augmentation purposes. It is very fast, because the accuracy is not the largest issue when the eye has to judge if it looks real or augmented. But the speed must be high for the eye to see it as real as the background.

There is not really a common standard of how to compare methods for pose estimation and there is no standard method to compare with. In this thesis efford is made to get a fair comparison and there is included a simple very known method as comparator.

In total there is 4 methods tested, they calculate the perspective from known 2D-3D correspondences from image to point cloud. All have different limitations such as minimum amount of 2D-3D correspondence pairs or sensitivity to noise that makes it unpredictable in noisy conditions. The benefits and drawbacks are listed for each method for easy comparison. The 3 methods are nonlinear CPC, PosIt and PosIt for coplanar points, while DLT is a linear method that is used because it is easy to implement and good for comparison.

All tests are done on fictive data to allow some extreme cases and to have ground truth for accurate comparisons. In short the tests made are: Noise

test, increased number of points, planarity issues, distance to object and initial guesses.

The findings were many and shows that the methods are working very differently. So when choosing a method, one has to consider the application of it, and what data is available to the method.

Preface

This thesis was made at Computer Vision and Graphics (CVG), at Aalborg University Copenhagen (AAU) in partial fulfillment of the requirements for acquiring the Master of Science in Engineering (M.Sc.Eng).

The thesis was carried out over a period of 4 month for the amount of 30 ECTS points, completing 1 full semester.

AAU, June 2008

Thomas Petersen

Acknowledgements

This thesis has been made with help that I would like to thank for: First of all my supervisor and teacher Daniel Grest for providing code and assistance along with it, and also to help guide me in the right direction. I also like to thank family and friends for support throughout this period, and especially my brother for proofreading my report.

Contents

Chapter 1

Introduction

This thesis is about various methods for pose estimation and tests to reveal how they react in common and uncommon situations. A pose is defined as the position and orientation of the camera in relation to the 3D structure. In order to estimate a pose, a 2D image, a 3D object and some camera parameters are needed depends on the method. The 2D image should show the points of the 3D object, which is called 2D-3D correspondences. A minimum number of correspondences are required for each method to run. Fischler and Bolles [**?**] have used the term "Perspective n-Point problem" or "PnP problem", for this type of problem with n feature points.

Pose estimation is used in tracking applications in many varieties including surveillance and augmentation. With the increase in computer power the usage will expand to other areas where it used to be impossible. For closed environments at factories, robots can track and handle products at a very fast pace. Other areas like augmentation the usage is limited to mostly movies and toys at the moment. Augmentation is about changing a picture, movie or live video by adding content that follows objects seen. Vincent Lepetit and Pascal Fua made a compilation of tracking methods and usablities in [12] and Ronald T. Azuma made an augmented reality survey in [2].Example of augmentation can be seen on figure 1.1.

Most usages today include simple fixed 2D markers for augmentation in games

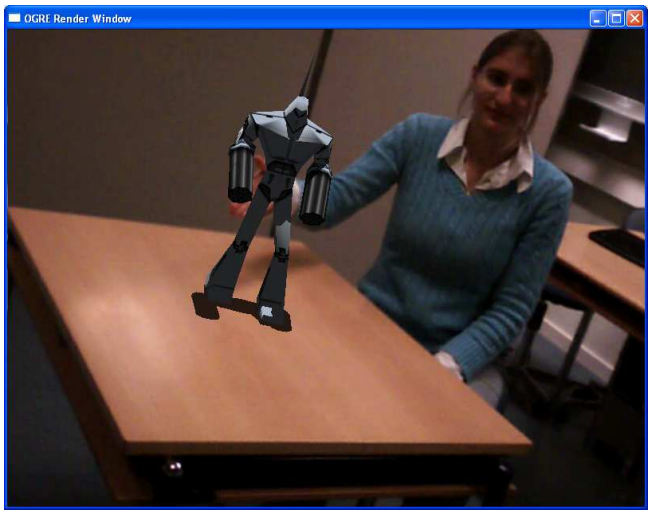


Figure 1.1: Example of augmentation in Ogre.

for example Hitlab developed ARToolKit [11] and they made projects mainly around helping visual perception by 3D in areas like hospitals, learning and treating phobias. Another company that makes virtual reality called Total Immersion uses marker tracking but also markerless tracking for movies, airplane controls, enhanced TV and much more. Common for companies like these is that their products aim for kids or learning aspects, but also the industry for movies and presentations. All these areas seems to be focused on luxury but there is plans for more serious projects in health care with 3D bodies for operations for example.

Some of the recent projects use structure from motion [1] to go from normal vacation images to build a 3D model of famous buildings. When a family is in vacation they should be able to take a picture of a building and the camera knows what building and provides some info about the buildings history. It uses GPS to narrow down the search of buildings and from maybe hundreds of images of one single building it can be pretty sure it guesses the right one.

Tracking people by their faces or body is also going to be the future on normal streets to find criminals or track suspicious people. On the website Earthcam[2] you can log in and see cameras from all over the world in famous places with high population. This might just be the beginning of a surveillance era where we will be watched in all public places. England has a very dense camera network in

[1] http://www.mip.informatik.uni-kiel.de/tiki-index.php?page=3D+reconstruction+from+images
[2] http://www.earthcam.com/network/

London that is being tested with face recognition. It was proven to be useful at the London bombing in 7th july 2005 where images of the terrorists was found from several cameras around London. China uses face recognition to keep track of hooligans that serves 1 year for making problems, and they can easy deny them access to soccer games. This all illustrates the importance of survaillance and that it might be used for tracking other things than faces in just a matter of time.

Several algorithms with the same goal have been tested against each other under various circumstances. The goal for all algorithms was to go from 2D-3D correspondences to a camera pose as seen on figure 1.2. The input for the system is a known 3D structure made of points and these points have been related to 2D points on an photo taken of the same 3D structure. This step of 2D-2D matching is not described here but is also a vast area of research today. For the pose to be defined both translation and rotation needs to be found, and in the results here both are plotted, because the algorithms proved to not necessarily give same accuracy for both.

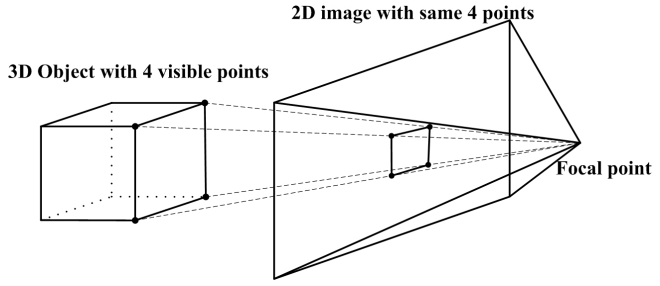


Figure 1.2: Perspective projection illustrated for 4 points observed from camera.

The tested algorithms are CPC(CamPoseCalib) from [3], DLT [4], PosIt [6] and a planar version of PosIt [5]. There is no single place where all these have been compared for speed and accuracy nor the special cases where 1 algorithm might not work properly. So this report is about special cases and common standards of comparison between such algorithms.

Chapter 2

Related Work

Many people have come up with innovative ideas for optimizing this process of finding a pose from 2D-3D correspondences. Ranging from simple ones like DLT [4] to complex ones like PosIt [6]. Not only complexity vary a lot but also the approach can be either iteratively like [3] [6] [11] or non iterative like [4] or [8].

Most of the algorithms are based on a linear or nonlinear system of equations that needs to be solved, but how they get these equations and how many parameters to be estimated is very different. All algorithms are not made for the same purpose, still they all aim for speed and accuracy in their area. For example [8] is made to work well for many points since the system of equations has the same computation time for higher amount of points. The idea is to use 4 control points that are not coplanar, and they define all points according to these control points so they only need to find the control points.

DLT [4] was first used in photogrametry and then later in computer vision. It is the simplest method for finding the pose, by finding all the 12 parameters in the projection matrix it will be slower than any other algorithm. Note that it is expected to be the weaker of the methods tested and included because it is often used as a simple method that does not require initial guess and it is easy to implement.

PosIt [6] is a method developed by Daniel DeMenthon in 1995 where it was published as a pose finder in 25 lines of code (mathematica). A fast method that does not work with perspective projection as most other methods, but instead uses orthographic projection that is simpler to work with and can approximate perspective to find the correct pose.

Coplanar PosIt [5] was also made by DeMenthon because PosIt does not work well for planar or close to planar point clouds. Another issue when dealing with planar point clouds is that there is often 2 solutions that are very close, and while normal algorithms choose either one, this planar version returns both. This is needed in some cases when there is no prior knowledge and a decision has to be made which pose is the right one. With prior knowledge you can limit the pose change from each frame in a video sequence so it does not guess the opposite pose.

CPC stands for CamPoseCalib and is the method from BIAS [10], which is an image library made in Kiel 2006. It is based on the Gauss-Newton method and nonlinear least squares optimization, originally published in [3]. A flexible method, because it has a lot of options that are not described here. However these additional methods (like outlier mitigation) are not used in the comparison. More about Gauss-Newton and this special implementation can can be found in chapter 3 of [9].

Chapter 3

Theory

3.1 Overview

The 4 different algorithms have very unique theoretical background and this is useful for understanding the test outcome in chapter 4. This chapter will give basic information on notation and coordinate system information before explaining the tested methods.

1. Notation
2. Pinhole Camera Model
3. Non-linear Least Squares
4. Pose and rigid motion
5. Posit
6. PosIt for coplanar points
7. Projection matrix
8. Direct Linear Transform(DLT)
9. Boxplot

3.2 Notation

Because of all the algorithms have different origin they come with various notation that is presented here with one common notation:

$s, \theta, x \in \mathbb{R}$	a scalar value
$m \in \mathbb{R}^2$	2D points from perspective projection
$M \in \mathbb{R}^3$	3D points
$p' \in \mathbb{R}^2$	2D point from SOP(scaled orthographic projection)
$A, R, P \in \mathbb{R}^{n \times m}$	matrix of size $n \times m$
functions:	
$m(\theta, p)$	3D motion function for point m and parameters θ
Algorithms:	
CPC	Gauss Newtons method from BIAS [10].
PosIt	BIAS version of PosIt [6].
DLT	Direct linear transform from [4].

3.3 Pinhole Camera Model

The basic of projection comes from a model where all light from a scene travels through a small hole 3.1. x

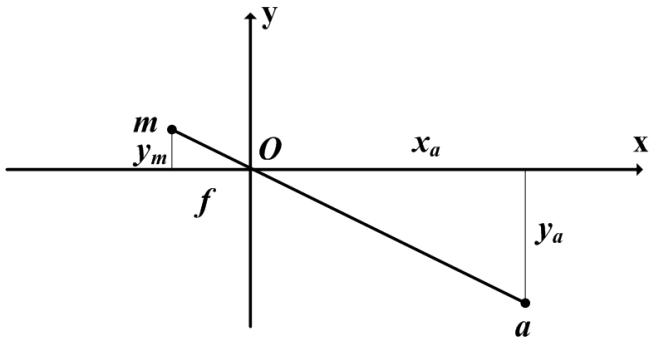


Figure 3.1: Pinhole Camera Model.

Figure 3.1 shows simple pinhole princip where:

$$\frac{y_m}{f} = \frac{y_a}{x_a} \tag{3.1}$$

$$y_m = -\frac{f y_a}{x_a} \tag{3.2}$$

The 3D point a is projected through origin O to imageplane along y_m with normal in x direction to 2D image point m. The coordinate y_m found in 3.2 is the first coordinate of the 2D point in the image, the second can be found in the same manner by looking from above(along y) and do the same calculations.

3.4 Non-linear Least Squares

The parameters describing a pose are estimated by minimizing an objective function. The minimization is a non-linear least squares problem and is solved with Gauss-Newtons method. The objective function depends on the input we need to estimate. A general non-linear least squares problem reads:

$$\hat{\phi} = \arg\min_{\phi} \sum_{i=1}^{m} (r_i(\phi))^2 \tag{3.3}$$

This minimization is iterative and there are 2 stop criteria: when overall error is lower than a threshold between image points observed and back projected points from the 3D model. The threshold is called epsilon and in CPC it is defined for both the rotational change and the 2D-2D change. The second stop vriteria is when a certain number of iterations is reached it stops.

3.4.1 Gauss-Newton

CPC from [10] is using the Gauss-Newton method to estimate the pose. This method uses first order derivatives of the residual (for residual see equation 3.4).

$$r_i(\phi) \tag{3.4}$$

Gauss-Newton approximates the hessian by leaving out the second order derivatives. The method called Newtons method uses also second order derivatives.

This method works analytically and not numerically, meaning it does not "shake" the object to try and find derivatives, instead it already has the function for the derivative directly. A "Shake" means a small change in parameters and then checking how that affected the error rate. The disadvantage about numerical approach is the size of the "shake" needs to be predefined and then does not have to show the actual slope in the parameter space for a given situation.

For minimizing the objective function we need the Jacobian (first order derivative of the residual functions) and from that update the parameters as seen in equation 3.5:

$$\theta_{t+1} = \theta_t - (J^T J)^{-1} J^T r(\theta_t) \tag{3.5}$$

Where J is the jacobian found according to [9].

3.4.2 Levenberg-Marquardt

This extension to the Gauss-Newton method interpolates between Gauss-Newton and gradient descend. This addition makes Gauss-Newton more robust meaning it can start far off the correct minimum and still find it. But if the initial guess is good then it can actually be a slower way to find the correct pose. This works like dampening the parameter change that happens each iteration to make sure that Gauss-Newton always descends in parameter space. More details about Levenberg-Marquardt in [4].

3.5 Pose and rigid motion

In order to describe an object in 3D space all we need is rotation and translation. Let a_{obj} be a 3D point in an object we like to know the world coordinates of, and (α, β, λ) be the Euler rotations around the 3 world axis. Then the a_w in world coordinate system is given by:

$$a_w = (\theta_x, \theta_y, \theta_z)^T + (R_{ex}(\alpha) \circ R_{ey}(\beta) \circ R_{ez}(\lambda))(a_{obj}) \tag{3.6}$$

Where (e_x, e_y, e_z) denotes the world coordinate system and the whole thing is build on a translation and a rotation part.

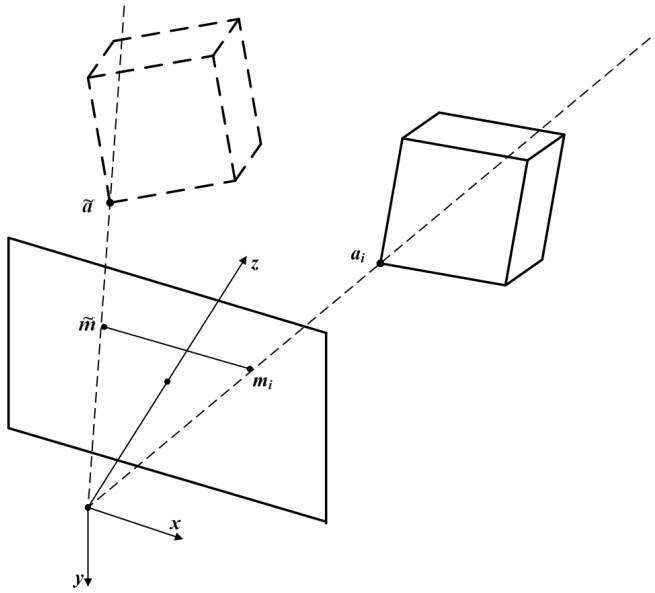


Figure 3.2:

This leads to a function that describes the motion from 1 point in space to another:

$$m(\theta, a_0) = (\theta_x, \theta_y, \theta_z)^T + (R_x(\theta_\alpha) \circ R_y(\theta_\beta) \circ R_z(\theta_\lambda))(a_0) \tag{3.7}$$

Now to combine the camera projection with the rigid motion described above we get a transform from $a = (x, y, z)$ in world coordinates, to $m' = (x, y)$ in image coordinates:

$$m'(a, \theta) = \begin{pmatrix} s_x \frac{m_x(a,\theta)}{m_z(a,\theta}) + c_x \\ s_y \frac{m_y(a,\theta)}{m_z(a,\theta)} + c_y \end{pmatrix} \tag{3.8}$$

$m(a, \theta)$ is the function 2.3, and (s_x, s_y) is the focal length of the camera in x and y direction in pixels, and $(c_x, c_y)^T$ is the principal point of the camera.

Now for the actual pose estimation when given a 3D point a_i and a 2D image point $\tilde{m_i}$, the relative motion can be described by nonlinear least squares that minimizes the residual that describes the connection between the 3D and the

2D (a_i, m_i).

$$d = r_i(\theta) = (m'(a_i, \theta) - \tilde{m_i}) \tag{3.9}$$

Where d is the 2D distance between 2 poses given a change in parameters θ. This is used for all iterations in CPC to find the next proper pose.

3.6 PosIt

This method is published by Daniel F. DeMenthon and Larry S. Davis in May 1995. PosIt can find the pose of an object from a single image and does not require an initial guess. It requires 4 or more point correspondences between 3D and 2D in order to find the pose of the object. The method is split in 2 where the first part is POS (Pose from Orthography and Scaling) that approximates the perspective projection with scaled orthographic projection and finds the rotation matrix and translation vector from a linear set of equations. The second part is POSIT which iteratively uses a scale factor for each point to enhance the found orthographic projection and then uses POS on the new points instead of the original ones until a threshold is met, either the number of iterations or the 2D pixel difference averaged over all 2D correspondences.

3.6.1 Scaled Orthographic Projection

Scaled Orthographic Projection (SOP) is an approximation to perspective projection because it assumes that all the points a_i with (X_i, Y_i, Z_i) in camera coordinate system, have Z_i that is close to each other and can be set to the same Z_0 as taken from the reference point a_0 on figure 3.3. In SOP, the image of a point a_i of an object is a point p'_i of the image plane G which has coordinates:

$$x'_i = f\frac{X_i}{Z_0}, y'_i = f\frac{Y_i}{Z_0} \tag{3.10}$$

While in perspective projection instead of p'_i we get m_i:

$$x_i = f\frac{X_i}{Z_i}, y_i = f\frac{Y_i}{Z_i} \tag{3.11}$$

The ratio called $s = f/Z_0$ is the scaling factor of the SOP. Note that the reference point a_0 is the same in perspective and orthographic projection $(x_0, y_0) = (x'_0, y'_0) = (0, 0)$.

3.6.2 The PosIt coordinate system

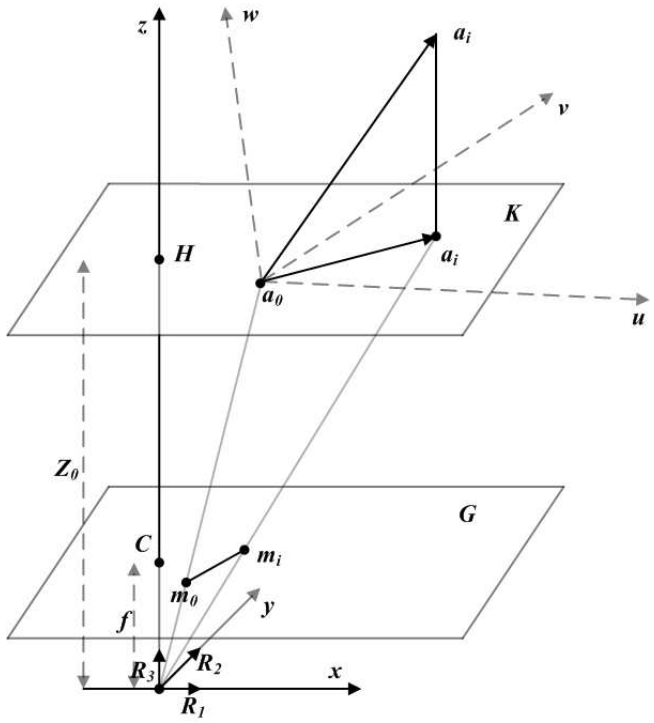


Figure 3.3: PosIt coordinate system.

To find the rotation and translation we look first at the rotation matrix:

$$R = \begin{bmatrix} R_{1u} & R_{1v} & R_{1w} \\ R_{2u} & R_{2v} & R_{2w} \\ R_{3u} & R_{3v} & R_{3w} \end{bmatrix} = \begin{bmatrix} R_1^T \\ R_2^T \\ R_3^T \end{bmatrix} \tag{3.12}$$

Seen on figure 3.3, (a_0u, a_0v, a_0w) are the coordinates of 3D point a_1 in the object coordinate system . If we can find R_1 and R_2 we can calculate R_3 as the crossproduct $R_1 \times R_2$. The translation vector T is the vector a_0O between the

center of projection O and origin of object coordinate system a_0. a_0 should be visible image point m_0 and T is then aligned with Om_0 equal to $\frac{Z_0}{f}Om_0$. So the pose defined by translation and rotation is found when we know R_1, R_2 and Z_0.

3.6.3 The PosIt Algorithm

Starting with basic perspective projection that consist of a rotation matrix where R_1^T, R_2^T, R_3^T is the rows vectors of the matrix and a translation vector that is between the origin O center of the camera and a_0 which is $0, 0, 0$ in object coordinates. To project an object point a into image point m:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} fR_1^T & fT_x \\ fR_2^T & fT_y \\ R_3^T & T_z \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} \tag{3.13}$$

The coordinates of the image point are homogenous and not affected by multiplying with constants. All the elements in the perspective projection matrix are then multiplied with $1/T_z$. Also the scaling factor $s = f/T_z$, that relates the distance to the object with the view of the camera.

$$\begin{bmatrix} wx \\ wy \end{bmatrix} = \begin{bmatrix} sR_1^T & sT_x \\ sR_2^T & sT_y \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} \tag{3.14}$$

But now the w is instead:

$$w = R_3 \cdot a/T_z + 1 \tag{3.15}$$

In order for w to become 1 we need T_z to become much larger than $R_3 \cdot a$, and then the projection would be the same for perspective projection. The dot product $R_3 \cdot a$ describes the projection of object point a unto the optical axis of the camera. Where R_3 is the unit vector of the optical axis. When the depth range of the object is small compared to the distance from the object to the camera the perspective projection becomes the same as orthographic projection.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sR_1^T & sT_x \\ sR_2^T & sT_y \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} \tag{3.16}$$

The case of $s = 1$ is pure orthographic projection. The value s being different from 1 can make up the difference to perspective projection. For each 2D point PosIt estimates this scaling.

Now the general perspective equation can be rewritten as:

$$\begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} sR_1 & sR_2 \\ sT_x & sT_y \end{bmatrix} = \begin{bmatrix} wx & wy \end{bmatrix} \tag{3.17}$$

Now when the iterations start we set initial $w = 1$ and then the only unknowns are sR_1, sR_2 and sT_x, sT_y that can be found with a system of 2 linear equations assuming the rank of the matrix A is a least 4. So at least 4 points from all object points have to be non-coplanar.

$$XsR(1,1) + YsR(1,2) + ZsR(1,3) + sT_x = wx \tag{3.18a}$$
$$XsR(2,1) + YsR(2,2) + ZsR(2,3) + sT_x = wy \tag{3.18b}$$

Where $R(1,1)$ means the first row and first column in the rotation matrix R.

$$I = \begin{bmatrix} sR_1 \end{bmatrix} \tag{3.19}$$

$$J = \begin{bmatrix} sR_2 \end{bmatrix} \tag{3.20}$$

So I and J are the scaled rows of the rotation matrix. To make a system of equations we define K as the matrix we want to find, b as the 2D points and A as the 3D points:

$$K = \begin{bmatrix} I^T & sT_x \\ J^T & sT_y \end{bmatrix} \tag{3.21}$$

$$A = \begin{bmatrix} X_0 & Y_0 & Z_0 & 1 \\ X_0 & Y_0 & Z_0 & 1 \\ X_1 & Y_1 & Z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 \end{bmatrix} \tag{3.22}$$

$$b = \begin{bmatrix} x_0 \\ y_0 \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \tag{3.23}$$

The problem then simplifies to 3.24:

$$AK = b \tag{3.24}$$

This can be solved as system of equations with least squares method. If n is larger than 4 it is overdetermined:

$$\hat{K} = (A^T A)^{-1} A^T b \tag{3.25}$$

After finding I and J we can extract s and R_1, R_2 by the condition that they must be unit vectors, and R_3 as the crossproduct between R_1 and R_2. T_x and T_y is found by deviding sT_x, sT_y with s.

As the iterations continue w from 3.18 for each point, becomes increasingly accurate until a stabil result is found or max number of iterations is reached. The scale factor w is updated according to 3.15.

3.7 PosIt for coplanar points

When dealing with planes there rises doubt about the pose because several poses that are very different have the same orthographic projection 3.4. This planar version of PosIt is finding all the poses and then choosing the best match. More specific it finds 2 poses for each iteration and either picks one or continue with both options. This might look like a tree with n iterations and 2^n solutions but only 2 poses are kept at any time.

If the rank of A 3.22 is 3 instead of the 4 as required for regular PosIt, then we have points distributed on a plane. Checking if the points are coplanar can be done by calculating the rank with SVD (Singular Value Decomposition).

3.7.1 Finding poses

On figure 3.4 is illustrated the case where the object plane can have 2 very different positions that returns the same 2D image correspondences. This is object plane 1 and object plane 2 that both can be result of PosIt when dealing with planarity. Given equation 3.19 and 3.20 if we could find 2 solution for both I and J that would create 2 poses.

$$I = I_0 + \lambda u \tag{3.26}$$

Where u is the unit vector normal to D figure 3.4 and λ is the coordinate of the head of I along u. Same thing for J, μ is the coordinate of the head of J along u:

$$J = J_0 + \mu u \tag{3.27}$$

The unknowns are λ, μ and u. u can be found because it is the normal to the object plane D thus we know that $a_0 a_i . u = 0$. u will then be the unit vector of the null space of the matrix A. SVD and this knowledge can give you u and then we only need λ and μ. To find them 2 more constrains are needed:

$$\lambda\mu = -I_0 . J_0 \tag{3.28}$$

Equation 3.28 means I and J are perpendicular. And 3.29 means I and J have the same length:

$$\lambda^2 - \mu^2 = I_0^2 - J_0^2 \tag{3.29}$$

Equation 3.28 and 3.29 Now we have 2 equations with 2 unknowns and because this can be solved using a complex version and it gives 2 different solutions for both I and J (see more in [7]).

$$I = I_0 + \rho cos(\theta)u, J = J_0 + \rho sin(\theta)u \tag{3.30}$$

$$I = I_0 - \rho cos(\theta)u, J = J_0 - \rho sin(\theta)u \tag{3.31}$$

The 2 solutions can be illustrated in the following manner:

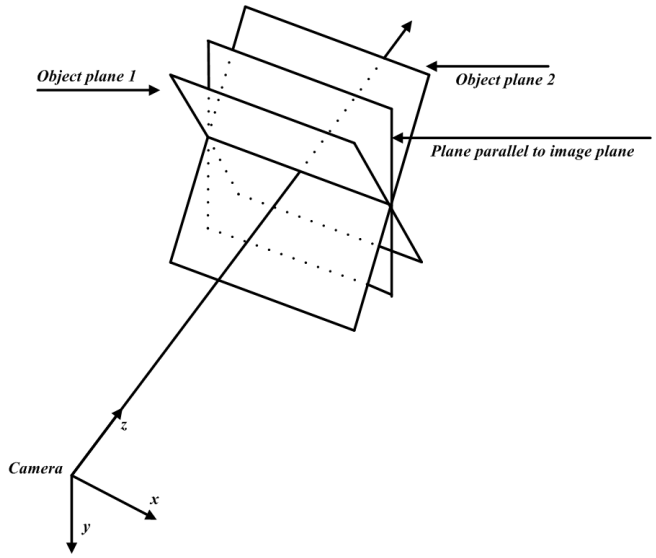


Figure 3.4: Shows how object plane 1 and object plane 2 looks the same seen from the Camera, and the planar version of PosIt returns both these solutions.

The values (I, J) for the first solution is symmetrical to the second solution (I, J) around the plane parallel to the image plane used to make the SOP approximation.

Figure 3.5 has different vectors I whose head project onto the plane D in Q projects unto a_0a_1 at H_{x1}. The following equation gives H_{xi}:

$$\overline{a_0 H_{xi}} = \frac{x_i^{'}}{|a_0 a_i|} \tag{3.32}$$

Given that $x_i^{'} = x_i(1 - w_i) - x_0$, where w_i is the scale factor for each point, then H_{xi} can be found with equation 3.32 from the correspondences alone. This means that if 3 points on a plane defined by the lines a_0a_1, a_0a_2 and a_1a_2 then the corresponding H_{x1} and H_{x2} will intersect at the vertical line that goes through Q and this should be a point since we want the head of vector I. This is the case if 1 more points is added that is not in the same plane, then H_{x3} will intersect u exactly at the head of I. That means for coplanar points there can be more solutions than 1.

Then find the translation vector that is unique for both pairs of (I, J). And 2

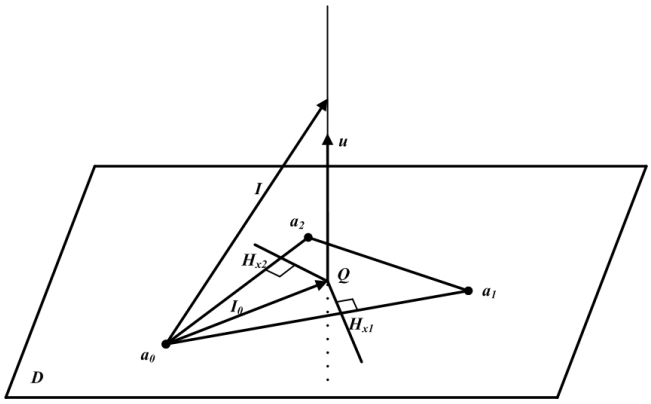


Figure 3.5: Shows how the vector I is defined by the points a_i.

rotation matrices by normalizing I and J we got the 2 first rows and last row is crossproduct of the first 2 rows. Unlike the regular PosIt these solutions can be a pose where the camera looks away from the points, easily detected by checking the sign of all $Z_i > 0$ and discard the pose if any point is behind the camera.

A distance measure E is used to distinguish the poses from one another, distance between the image points observed and the back projected from the calculated pose. E has a threshold that is predefined depending on the image noise and anything lower than the threshold is discarded.

3.8 Projection matrix

In the DLT (Direct Linear Transform) the parameters of the projection matrix are estimated, so this section will explain what exactly the parameters are. Perspective projection and the pinhole camera principle is basic knowledge for understanding this section. The idea is to go from 3D points m to 2D points a with a projection matrix P:

$$P = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.33}$$

The transform from 2D point image $m = (x, y)$ to homogenized 3D point $a =$

$(x, y, z, 1)^T$, $m = Pa$ then looks like:

$$\begin{pmatrix} m_x \\ m_y \\ z \end{pmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix} \tag{3.34}$$

Where z is a scale factor that is used to normalize the 2D point m by $\frac{m_x}{z}$ and $\frac{m_y}{z}$. The value of f_x and f_y is the focal length of the camera in x and y direction respectively. The rest of the projection matrix is rotation and translation parameters and the whole matrix is defined up to scale so it can be multiplied with a constant and remains the same. For more information on calibration and finding these parameters see [13].

3.9 Direct Linear Transform(DLT)

Direct Linear Transform (DLT) is the simple algorithm compared to CPC and PosIt.

DLT is around 10 times slower than 1 iteration of CPC because it estimates 11 parameters and CPC only 6. After finding the 11 parameters the last one can be found because the projection matrix P is defined up to scale. This method is the simplest of the tested methods. It linearly estimates all 12 parameters in the projection matrix and need at least 6 points to do so. The projection matrix P can be described by vectors as follows:

$$P = (q_1^T, q_{14}, q_2^T, q_{24}, q_3^T, q_{34}) \tag{3.35}$$

The $2D$ image coordinates and $3D$ object coordinates:

$$(a_i, m_i) = ((a_{i1}, a_{i2}, a_{i3})^T, (u_i, v_i)^T) \tag{3.36}$$

From $m_i = Pa_i$ follows:

$$q_1^T a_i - u_i q_3^T a_i + q_{14} - u_i q_{34} = 0 \tag{3.37a}$$
$$q_2^T a_i - v_i q_3^T a_i + q_{24} - v_i q_{34} = 0 \tag{3.37b}$$

Because P is defined up to scale we can set the last q in P to 1 and thus:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 & 0 & 0 & -u_1a_{11} & -u_1a_{12} & -u_1a_{13} & -u_1 \\ 0 & 0 & 0 & 0 & a_{11} & a_{12} & a_{13} & 1 & -v_1a_{11} & -v_1a_{12} & -v_1a_{13} & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & 1 & 0 & 0 & 0 & 0 & -u_na_{n1} & -u_na_{n2} & -u_na_{n3} & -u_n \\ 0 & 0 & 0 & 0 & a_{n1} & a_{n2} & a_{n3} & 1 & -v_na_{n1} & -v_na_{n2} & -v_na_{n3} & -v_n \end{pmatrix} \tag{3.38}$$

Now we can say $Aq = 0$. To find the q we need to have enough points n in a_n and in this case 6 for 12 parameters and more will be over determined. Then find q with SVD as the smallest eigenvalue under the assumption that $|q| = 1$. See [1] for details.

3.10 Boxplot

This plot type is also known as box and whisker plot, it shows median, quartiles and whiskers/outliers. The boxplot was invented in 1977 by the American statistician John Tukey. Quartile 1 ($Q1$) is the number which 25% of the data is lower than, and quartile 3 ($Q3$) is the number which 25% of the data is higher than and $Q2$ is the median value. Now the interquartile range (IQR) is $Q3$-$Q1$ and is used to decide which data points are marked as outliers. The standard value for this is $1.5IQR$ lower than $Q1$ or $1.5IQR$ higher than $Q3$.

On figure 3.6 from [1] you see 3 axes, the $1st$ one is boxplot of a standard normal distribution, $2nd$ shows how much of the data the different parts of the boxplot contains and the $3rd$ axes shows how the standard normal distribution is presented using variance as comparison to the boxplot method. Note that no outliers are shown here but later on they will be red X's.

[1] www.wikipedia.org

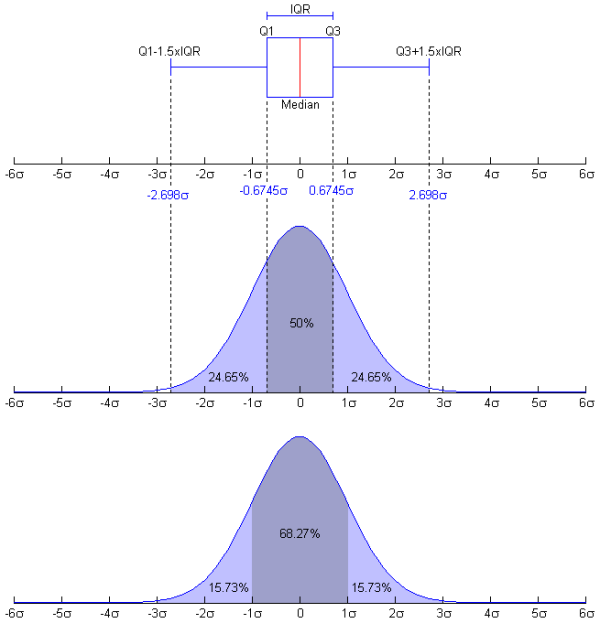


Figure 3.6: Shows standard boxplot compared to gaussian spread.

Chapter 4

Experiments

4.1 Tests

Here is a short description of each test and why it is relevant for testing followed by the actual test. Several angles are approached under each of these sections to find out what happens to accuracy and time in some extreme cases.

1. Overall algorithm and test setup
2. General limitations for all algorithms
3. Test 1: Increasing noise level
4. Test 2: Increased number of points
5. Test 3: Points shape is morphed into planarity
6. Test 4: Camera is moved very close to the points
7. Test 5: Testing the initial guess for CPC
8. Test 6: Different implementation of PosIt

4.1.1 Overall algorithm and test setup

These tests were conducted on synthetic data points with values uniformly distributed between -5 and $+5$ on x, y and z axis. One of these points will always be $0, 0, 0$ in 3D space because the PosIt implementation requires this. All the points are projected with a virtual camera position to make the ground truth and gaussian noise is added to the 2D points to resemble real image noise. The noise has mean 0 and variance 0.2 as a standard value unless otherwise written in the specific test.

For making the ground truth projection the following was chosen: Camera was set to $0, 0, -25$ in 3D space. The view direction set to $0, 0, 0$ (looking at $0, 0, 0$). Focal length set to 882 on both x and y axis, this is the same as 49 degrees field of view in y direction. This is chosen because it resembles a real camera focal length. Image size is defined to be 1200×800. Note that for PosIt the points need to be defined from the center and not upper left corner. (pixel coordinate $600, 400$ is $0, 0$ for PosIt). The camera perspective can be seen on figure 4.1 (a), and the points including camera position seen from the side can be seen on figure 4.1 (b). Note that the green surface is only to see the perspective more clearly:

(a) 10 points(gray) as seen from the camera perspective.

(b) 10 points(gray) and camera(red) seen from the side.

Figure 4.1:

To test the accuracy, the run time had to be the same for all algorithms. For applications that have limited time available and want to find the best possible accuracy. This was done by setting the stop criteria to maximum number of iterations and then, because each iteration takes a certain time, it is easy to align them. The DLT is one iteration so all the other algorithms are aiming for that same run time. The inaccuracy between run times is limited by iteration step size in time. This means they are not exactly the same run time but as close as it gets, on average within 3% of the total run time. For CPC to have

same run time as PosIt, CPC is run with 9 iterations and PosIt 400. This shows that PosIt has less to calculate during one iteration than CPC. Also note that CPC is provided the correct pose estimate at the first run of the test and after that uses the previous guess, unless otherwise noted in the test description.

PLOTS: 2 types of plots are used for showing the results, one is boxplot as described in section 3.9, and second is using mean and standard deviation as area plot. The reason for using standard deviation instead of variance is more robustness when it comes to scaling and it is easier to interpret. AXES: The translational error is the Euclidean distance between the perfect translation and the guessed translation. Result is divided by the distance from the camera to the world center. A translation error of 1 is then relative to a guessed camera position of 0,0,0 where the correct would be 0,0,-25. The rotational error is the Euclidean distance between 2 Quaternion. All time measuring is done in nanoseconds and include only the computation of the result and not the conversions and initializations.

All the tests are done 100 times.

The descriptions below use above numbers unless otherwise noted.

4.1.2 General limitations for all algorithms

In order to work properly all the algorithms have some basic demands for the input. This concerns the points but also the parameters required to find poses.

4.1.2.1 CPC requires

1. Initial guess of pose including projection matrix with focal lengths, skew, image size and center. The implementation also has a function to find a good initial pose, however this has not been used.

2. 3 or more 2D-3D correspondences.

3. 2D Points are defined from $(0, 0)$ up to the size of the image or $(1200, 800)$ in this thesis (origin top left).

4. Iteration number and limit on parameter change for rotational and translation error.

4.1.2.2 PosIt requires

1. Focal length, which is assumed to be same for x and y.
2. 4 or more non-coplanar 2D-3D correspondences.
3. 1st point must be $(0, 0, 0)^T$ in 3D and $(0, 0)^T$ in 2D.
4. 2D points are defined from the center of the image as $(0, 0)$ (Origin at image center).
5. Iteration number and pose accuracy.
6. Stop criteria either iterations or pose accuracy.

4.1.2.3 PosIt Planar requires

Same as PosIt. But for proper result the 3D points sould be perfectly coplanar see Test 3.

4.1.2.4 DLT requires

1. 2D Points are defined from $(0, 0)$ up to the size of the image or $(1200, 800)$ in this thesis (Origin at top left).

4.1.3 Test 1: Increasing noise level

The noise level on the 2D points are increased variance from 0.0 to 10.0 (about 3.3 pixels) with 100 steps of size 0.1 on an image of 1200×800 and no possibility of points being outside the image. A total of 10 points was used.

10.0 noise level means noise with the variance of 10.0 is added to the 2D back projected ground truth, to get the 2D points passed to the methods. Considering a gaussian distribution the noise could be more than 10 pixels, but with a very little likelihood.

On figure 4.2 is a 3D box that illustrates an area with 10 randomly distributed points, and shows how it is projected on to the 2D image. At the bottom of figure 4.2 is shown how the noise works on the 2D points as gaussians around each point.

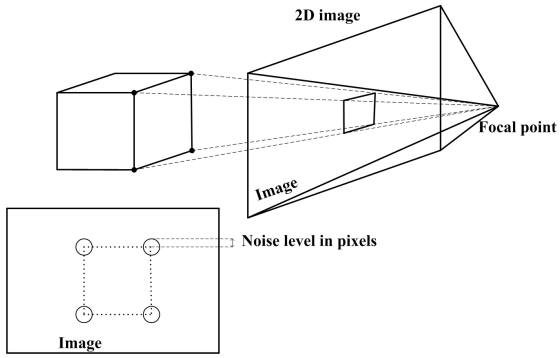


Figure 4.2: Test 1 illustrated

Each iteration of the methods takes a certain time and in this test they all have same run time as seen on figure 4.3.

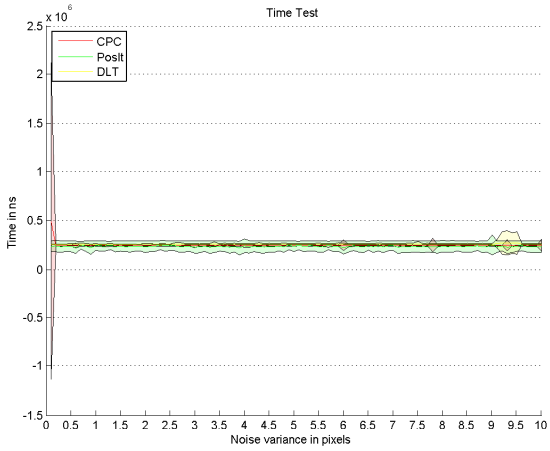
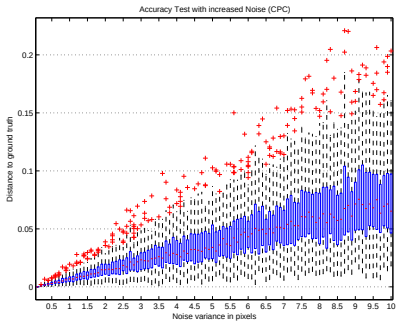


Figure 4.3: Run time for all algorithms when noise increases. Shown with mean and standard deviation.

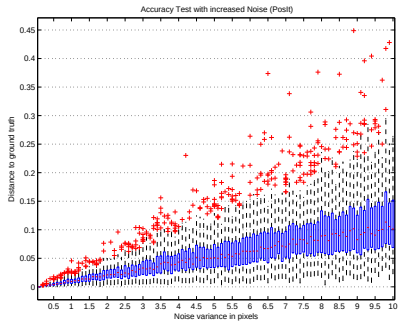
The run time aimed for in this test is about 0.235 ms. Thats the run time of DLT and the other methods needs various iteration number to get the same runtime. Here is the amount of iterations used for each algorithm:

1. CPC 9 iterations
2. PosIt 135 iterations
3. DLT 1

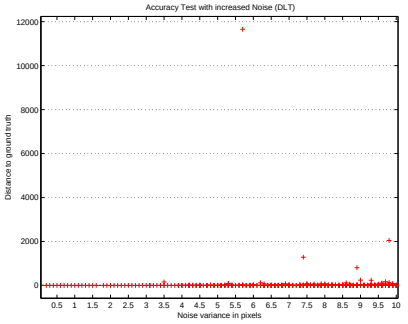
The stop criteria has been set to iterations so it will not stop when it reaches a specific accuracy.



(a) CPC translation error for increased noise.



(b) PosIt translation error for increased noise.



(c) DLT translation error for increased noise.

Figure 4.4: (a) shows increased noise for CPC. (b) shows PosIt and (c) shows DLT (see 4.5 for zoom in version).

Figure 4.4 shows the translation error for increased noise. Note that 0 noise level means 0 errors for both (a), (b) and (c). CPC (a) performs better than the 2 other methods and since they are all linear (see figure 4.5 for proof that the run time of DLT also is linear with number of points) this is the case for any noise level. PosIt (b) note y-axis larger than (a). DLT (c) shows here to be very sensitive to noise and the outliers size makes DLT not applicable for high noise levels at all. See figure 4.5 for DLT zoom in version.

On figure 4.5 is a zoomed version of DLT translation error for better comparison. DLT is about 5x as inaccurate as PosIt on figure 4.4 (b). And note that there are no extreme outliers in the beginning with low noise levels.

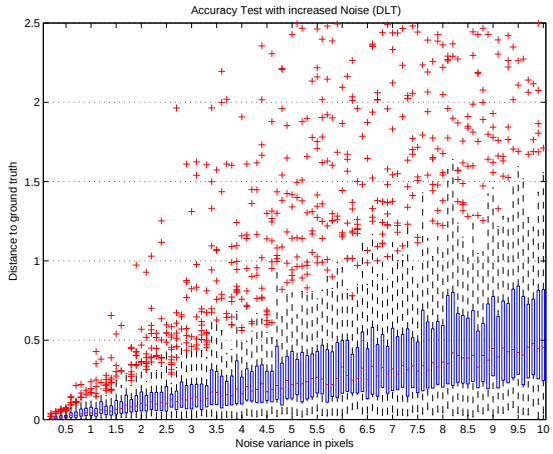
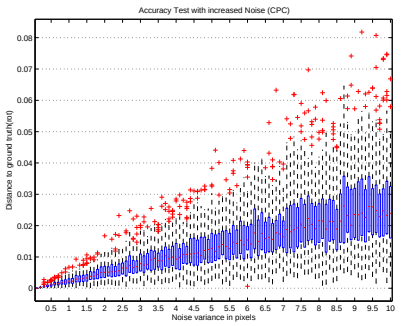
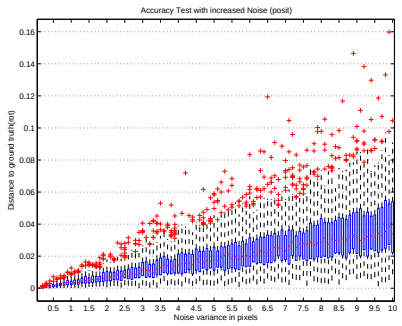


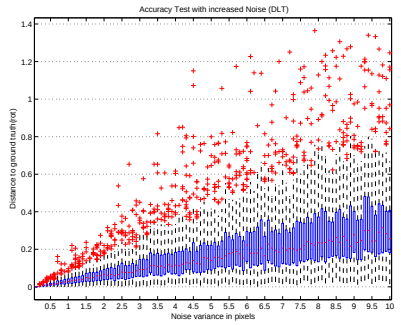
Figure 4.5: DLT translation error for increased noise, zoom in version.



(a) CPC rotational error for increased noise.



(b) PosIt rotational error for increased noise.



(c) DLT rotational error for increased noise.

Figure 4.6: This figure shows rotational error for increased noise level. (a) shows CPC. (b) shows PosIt and (c) shows DLT.

Figure 4.6 shows (a) CPC, (b) PosIt and (c) DLT rotational error for increased noise level. Note that the y-axes are different and CPC is the better of the 3 methods then PosIt and last DLT.

4.1.3.1 Maximum accuracy

Now for testing the maximum accuracy of a fixed noise level. CPC start at 1 iteration and increases up to 20 iterations. PosIt will start with 15 iterations that give the same run time as 1 iteration for CPC. Note that the stop criteria is set to iterations meaning it will not stop when a certain accuracy is reached, but keep going until all the iterations are complete.

On figure 4.7 is the translation error for increased iterations with mean and

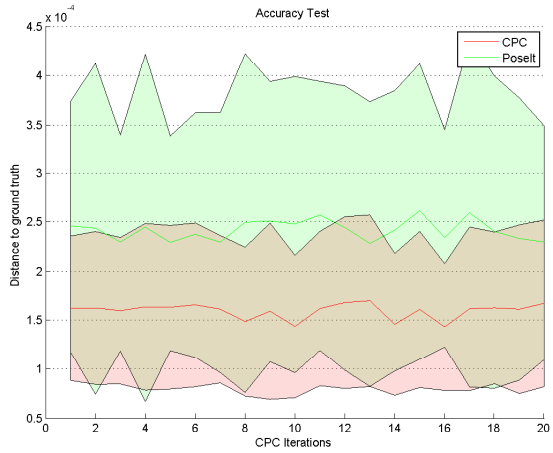


Figure 4.7: Translation error for increased iterations with mean and standard deviation.

standard deviation. DLT is removed because it is not iterative and as such makes no sense to draw it, but it had accuracy of 1.4E-3 for comparison. Note that PosIt starts at 15 iterations that takes just as long as 1 CPC iteration. The noise level of 0.2 also sets a limit of how well the methods can calculate the pose that is why the change here is small.

On figure 4.8 is rotational error for increased iterations with mean and standard deviation. Interestingly CPC is the most accurate both considering the spread and the mean. DLT has accuracy of about 8E-4 for comparison. It was expected to show an improvement in accuracy but the noise limits it.

The time plotted on figure 4.9 is to show that the methods have almost the same run time. CPC also has less spread and gets accuracy better than PosIt and DLT according to 4.7. Also notice that PosIt has larger spread when it uses more iterations, maybe CPC does too but it is less in that case.

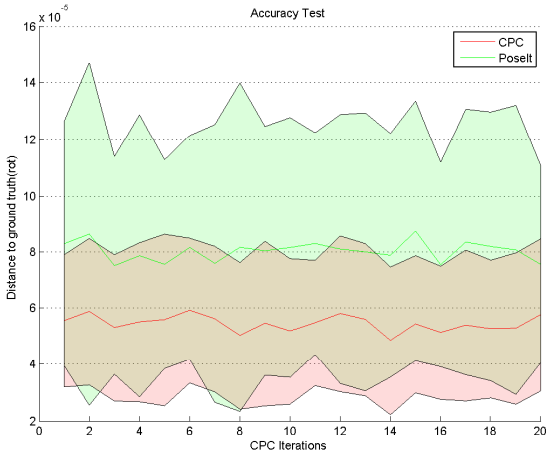


Figure 4.8: Rotational error for increased iterations with mean and standard deviation.

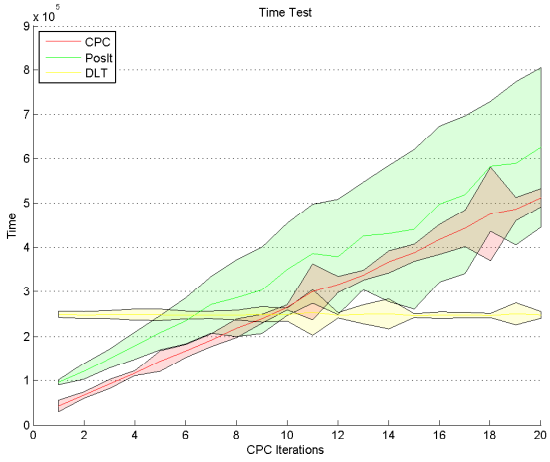


Figure 4.9: Run time for increased iterations with mean and standard deviation.

4.1.4 Test 2: Increased number of points

This test is made to find out what happens if we add more points to calculate the output pose. The result is plotted from $3-6$ (depends on the method) points up to 40 points and you can see the output is for a fixed number of iterations.

The CPC method has the correct projection matrix as initial guess, but because of the noise added the accuracy will not be perfect.

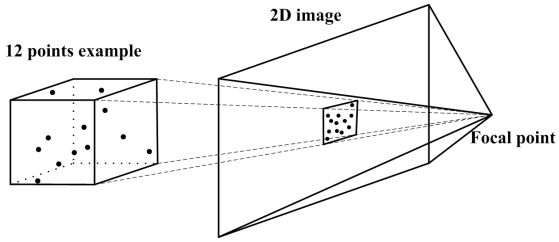


Figure 4.10: Test 2 illustrated

On figure 4.10 is illustrated how 12 points from the point cloud is captured by the camera onto the 2D image plane.

It is also interesting to see what happens when the methods are fed with minimum number of points possible for it to run and then increases the number of points to see how it affects the accuracy and run time. The minimum number of points the methods need are:

1. CPC 3
2. PosIt 4
3. PosItPlanar 4
4. DLT 6

This is because CPC estimates 6 parameters and therefore needs 3 points with x, y. PosIt needs 4 because they must not be planar and 3 points will always be planar. PosItPlanar should work with 3 point according to the theory, but version used here is limited to 4 points minimum. PosIt for coplanar points are not used in this test. To begin with the methods have the same run time but then the number of points affects them differently.

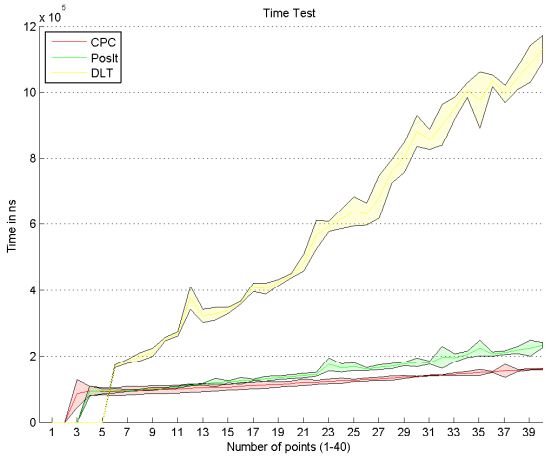
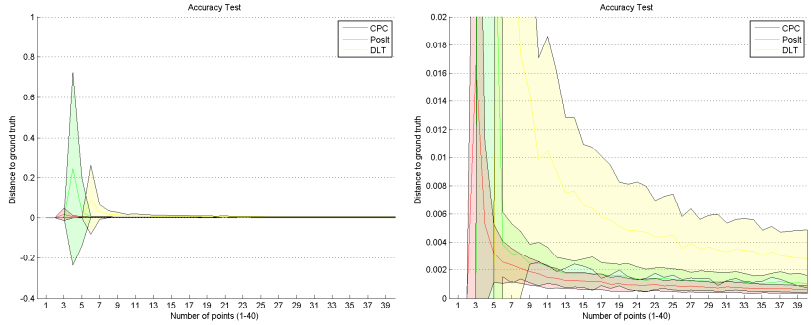


Figure 4.11: Run time for all methods with increased number of points and fixed number of iterations. Shown with mean and standard deviation.

On figure 4.11 is the run time of each algorithm as the number of points increases to 40 with mean and standard deviation. DLT stands out as the worst method and more than doubles in time when the number of points is doubled. So DLT is linearly increasing time for more points added. PosIt and CPC are close but CPC is the best at handling more points. CPC's run time becomes 1.84 times slower from 6-45 points. PosIt's run time becomes 2.76 times slower from 6-45 points. DLT's run time becomes 7.92 times slower from 6-45 points.

Figure 4.12 shows all the algorithms translation error with mean and standard deviation. (a) shows that PosIt spikes up to 0.7 at 4 (the minimum of points to run) which is bad when a guessed translation of 0,0,0 would be 1 in translation error. This could be because PosIt is not working properly for coplanar points (see test 3). PosIt becomes a lot more stable at 5-6 points. Zoomed in on (a) gives (b) which clearly shows that CPC is the better for any number of points (3-40).

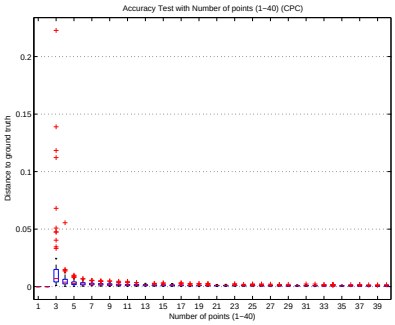


(a) Translational error for increased number of points.

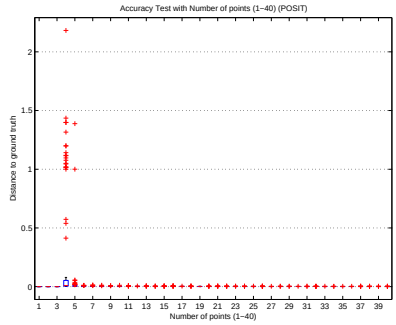
(b) translational error for increased number of points zoomed in version.

Figure 4.12: Translational error for all 3 methods with mean and standard deviation.

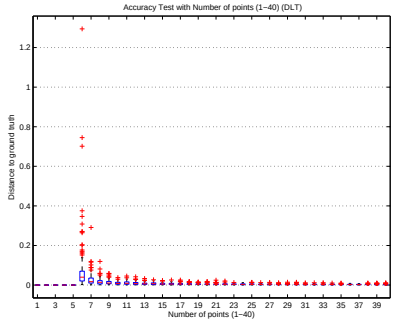
Figure 4.13 shows the translation error with respect to the number of points, as boxplots without zooming, and figure 4.14 shows the zoom in versions. CPC is the most accurate in this test and also works with least amount of points. Common for the methods is that the minimum amount of points they can run with has outliers that is extreme and would be a really bad pose estimate. CPC does not have outliers as extreme as DLT and PosIt with minimum amount of points and DLT has less extreme outliers for 6 points than PosIt for 4 points. But overall CPC is the best because it runs for 3 points minimum and the others 4 (PosIt) and 6 (DLT).



(a) CPC translation error for increased number of points.



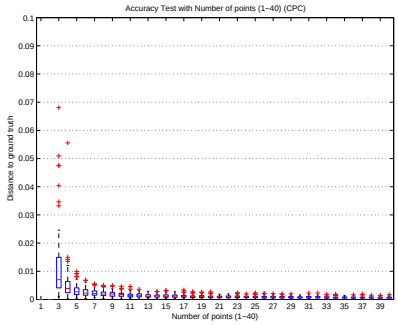
(b) PosIt translation error for increased number of points.



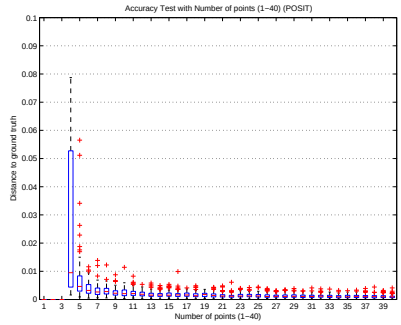
(c) DLT translation error for increased number of points.

Figure 4.13: Translation error for increased number of points.

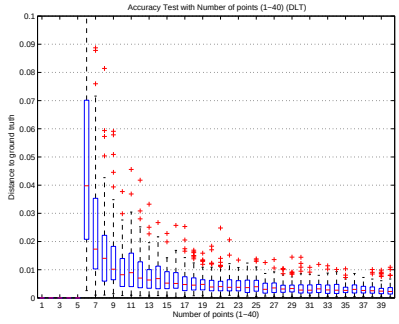
Figure 4.14 shows translation error for increased number of points in a zoomed in version that has same y-axes on (a),(b) and (c). CPC on (a) shows that IQR (section 3.10) stops between 0.02 and 0.03 while (b) PosIt stops between 0.07 and 0.08. The IQR of DLT(c) is larger than 0.1 worse than the 2 other methods. So it means CPC is the better choice when pose estimating from many points.



(a) CPC translation error zoom in for increased number of points.



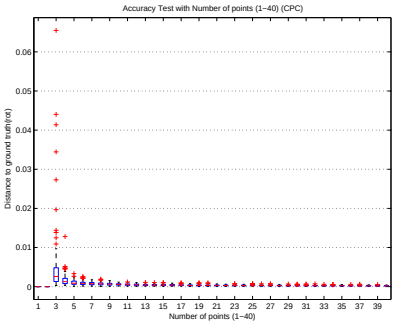
(b) PosIt translation error zoom in for increased number of points.



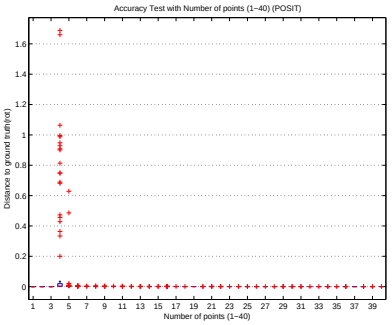
(c) DLT translation error zoom in for increased number of points.

Figure 4.14: Translation error for increasedn number of points. Note they have same y-axes.

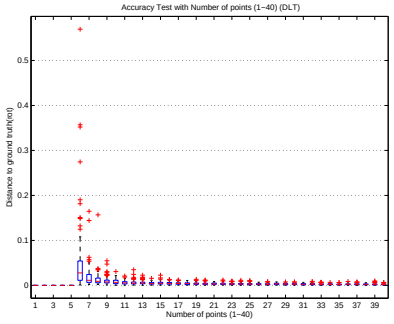
Figure 4.15 show rotational error for increased number of points and figure 4.16 shows the zoomed version. On CPC (a) note that the outliers are not as extreme as neither PosIt(b) or DLT(c). PosIt(b) is actually the least accurate for the methods minimum number of points.



(a) CPC rotational error for increased number of points.



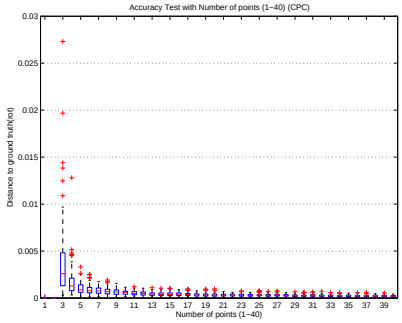
(b) PosIt rotational error for increased number of points.



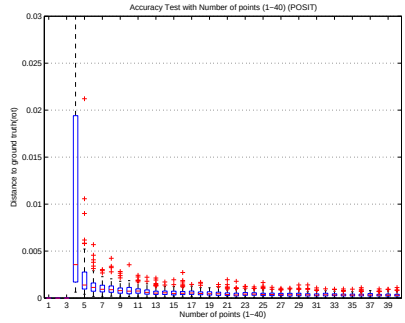
(c) DLT rotational error for increased number of points.

Figure 4.15: Rotational error for increased number of points.

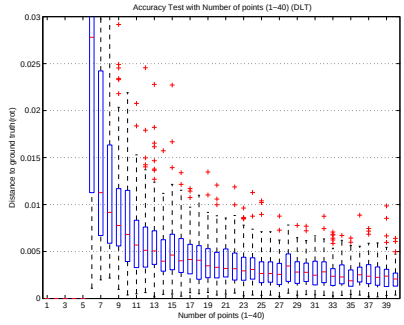
Figure 4.16 shows rotational error for increased number of points. Note that the y-axes are the same on the 3 plots. The median of (a) CPC is about same as (b) PosIt but the IQR is much smaller and therefore CPC is the better one. DLT (c) again stands out as the worst of the 3.



(a) CPC rotational error zoom in for increased number of points.



(b) PosIt rotational error zoom in for increased number of points.



(c) DLT rotational error zoom in for increased number of points.

Figure 4.16: Rotational error for increased number of points. Note same y-axes.

4.1.5 Test 3: Points shape is morphed into planarity

This test has a fixed viewport and all the points are slowly moved into a plane so there is no volume to exploit by primarily posit.

The point cloud starts as a cube shape and then change into a plane angled 0.25 π towards the image plane. Decreasing the oblongity by 30 % each step (oblong is how rectangular it is).

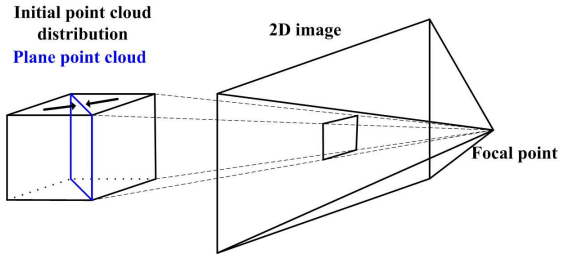
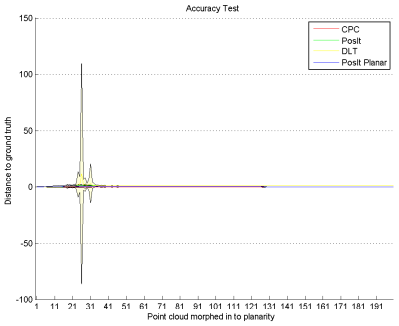


Figure 4.17: Test 3 illustrated

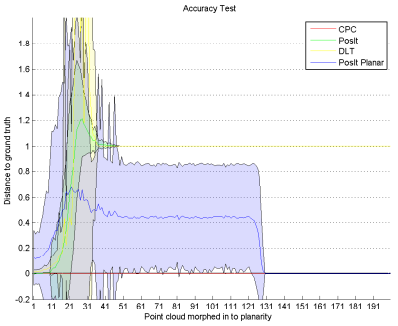
Figure 4.17 shows a black box where the points are randomly inside to begin with. I goes from -5 to +5 on x,y and z. Then at each iteration the cube is made 30% shorter in 1 diagonal until only 1 diagonal is left.

A few changes have been made to run this test, for example the PosIt method has lower accuracy just to separate it from the other methods on the plots, it is plotted for comparison only and not accuracy or time. The planar version of PosIt runs at 2 iterations.

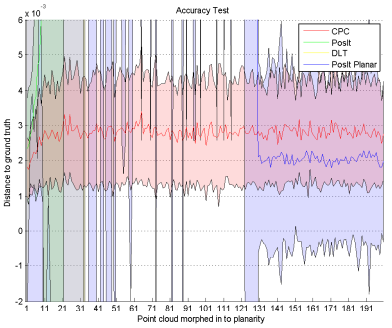
In this test is used 10 points starting with a point cloud shape of a cube and then through 40 iterations it is morphed in to a plane. Every step decreases the diagonal by 30% and the final thickness of the point cloud is about 1E-30 which is relatively close to 0. The first figure 4.18 (a) shows that DLT does not work well when the points are on a plane and it becomes very inaccurate and starts guessing at a translation of 0,0,0 which is an error of 1. On figure 4.18 (b) is zoomed in and is proven that PosIt does not work with planar points or close to planarity it also guess 0,0,0. The coplanar version of PosIt is correctly worse than posit for nonplanar points but as the points become planar it does not improve accuracy actually it gets worse. Only at around 130 (1E-19 thickness) it suddenly drops to a good estimate. With a closer look it is actually better translation error than CPC and takes the same computation time as seen on figure 4.19. After trying to increase the number of iterations CPC still was not as accurate as the planar version of PosIt.



(a) All methods translation error, for points moved to planarity.



(b) All methods translation error zoomed, for points moved to planarity.



(c) All methods translation error, for points moved to planarity zoomed even more to show CPC and posit planar when points are coplanar.

Figure 4.18: Translation error for all methods when point cloud is morphed into planarity. Shown with mean and standard deviation.

On figure 4.19 is the run time for all methods as the point cloud is morphed planar. CPC (red) has the same runtime no matter how coplanar the points are. PosIt (green) begins with low spread and then increases in error and spread as the points become planar. It is completely wrong and unstable from 20 morphs but stabilizes at a wrong pose after 130 morphs. The reason PosIt starts lower is that it has limit on the epsilon which stops it at a certain accuracy while the other methods are set to stop only by iterations. PosIt for planar points (blue) begins at same run time as DLT and CPC but then decreases as the points become coplanar and the spread becomes very high. After 130 morphs the planar PosIt again takes the same time as CPC and have low spread. DLT (yellow) starts at the same run time as PosIt planar and CPC but then drops as the points becomes planar and it can not find a proper pose.

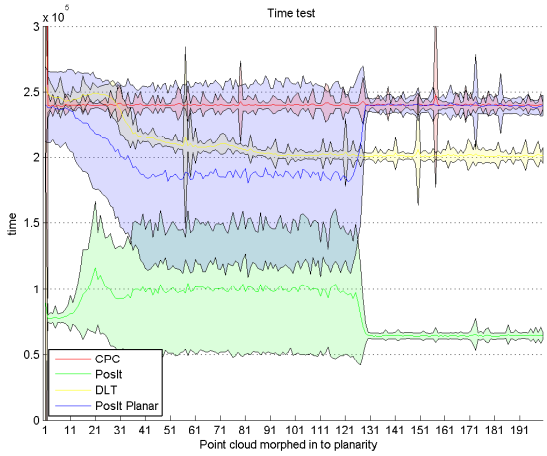
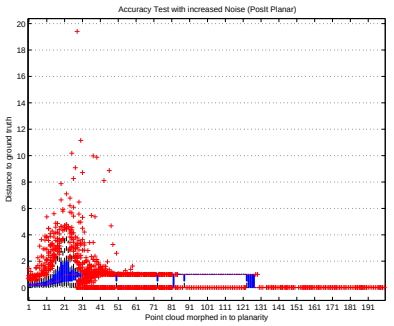
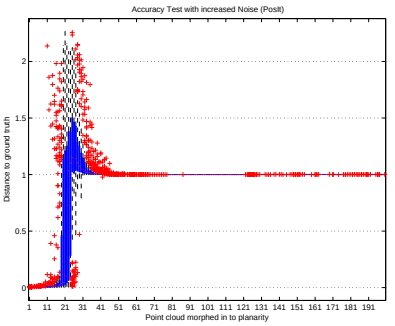


Figure 4.19: Shows run time for all methods when point cloud is moved to planarity. Shown with mean and standard deviation.

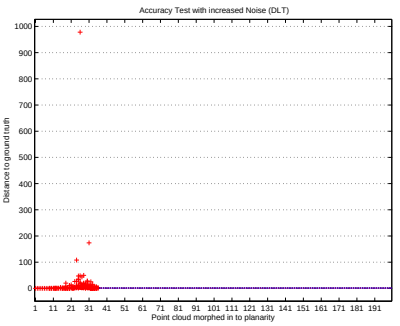
Translation for all 4 methods on figure 4.20 as points are morphed into planarity as boxplots, otherwise same as 4.18. Coplanar PosIt(a) begins stable but with bad accuracy and then becomes very unstable for about 20 morphs. It becomes stable at 50 morphs at 1 which is a guess of 0,0,0 translation. At 130 it suddenly starts working and is more accurate than CPC. PosIt(b) works in the beginning but after 20 morphs it is unstable and completely wrong as it does not work for coplanar points. DLT(c) works in the beginning and then becomes very unstable at about 20 morphs and after that it settles on a pose guess of 0,0,0 because it does not work for planar points. CPC(d) begins with a translation error of 0.0018 and after about 10 morphs it settles on a translation error of 0.0028.



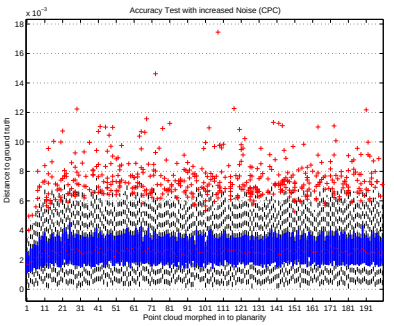
(a) Translation error for Coplanar PosIt when points are morphed to planarity.



(b) Translation error for PosIt when points are morphed to planarity.



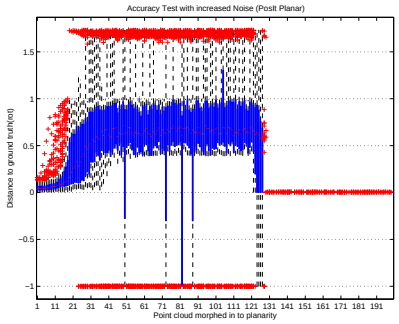
(c) Translation error for DLT when points are morphed to planarity.



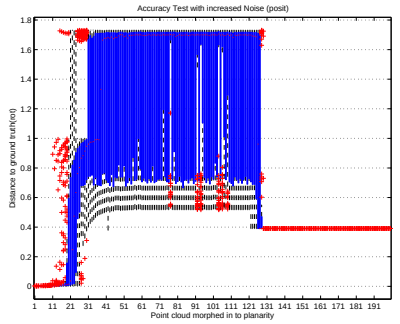
(d) Translation error for CPC when points are morphed to planarity.

Figure 4.20: Translation error for all methods when point cloud is morphed into planarity. Shown as boxplots.

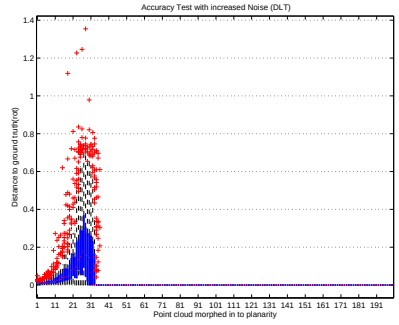
Rotational error for all 4 methods on figure 4.21. Coplanar PosIt(a) begins similar to PosIt with worse accuracy when about 20 morphs to planarity is over. Then it is wrong until about 130 morphs where it suddenly becomes better. Not better than CPC but close. PosIt (b) begins with good accuracy for a box with points because of the volume, then becomes increasingly worse as the points move into planarity and is not finding anywhere near proper pose until about 130 where it settles on a fixed error of 0.4. DLT (c) begins with good accuracy but then gets a lot worse with planar points and suddenly the result is 0. This is because it guessed a pose with angles 0,0,0 which happens to be the correct result but it does not mean that DLT works for planar points as seen on figure 4.20 DLT guesses 0,0,0 as the translation also. CPC (d) is very affected by coplanar points and becomes less accurate but never unstable as it settles on 0.0008 rotation error.



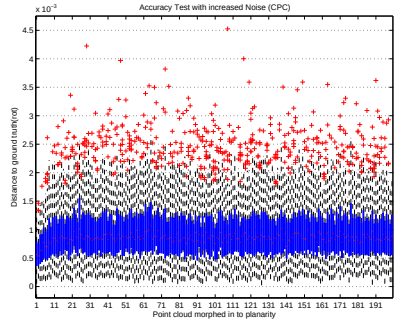
(a) Rotational error for Coplanar PosIt when points are morphed to planarity.



(b) Rotational error for PosIt when points are morphed to planarity.



(c) Rotational error for DLT when points are morphed to planarity.



(d) Rotational error for CPC when points are morphed to planarity.

Figure 4.21: Rotational error for all methods when point cloud is morphed into planarity. Shown as boxplots.

Conclusion must be that PosIt for planar points must have a planar object and not an object that is almost planar. PosIt for coplanar points is more accurate than CPC, PosIt and DLT and even with more iterations CPC can not get more accurate than it is considering translation error. CPC is still more accurate when it comes to rotational error. This shows that for planar points the planar version of PosIt is the better choice.

4.1.6 Test 4: Camera is moved very close to the points

In this test there is a fixed number of points (10) and the viewport is moved closer to the cloud to see how the increasing depth compared to distance affects primarily the posit algorithm.

Camera is moved to -12.5 from -25, towards the center and at this position the first points can be outside the image, so can not move it any further.

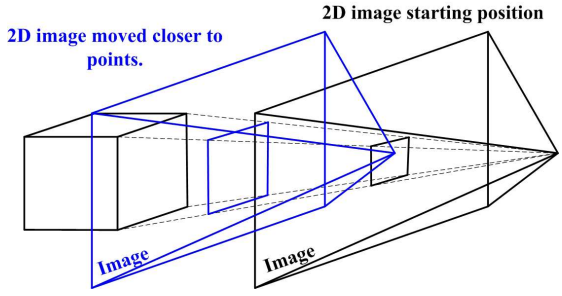
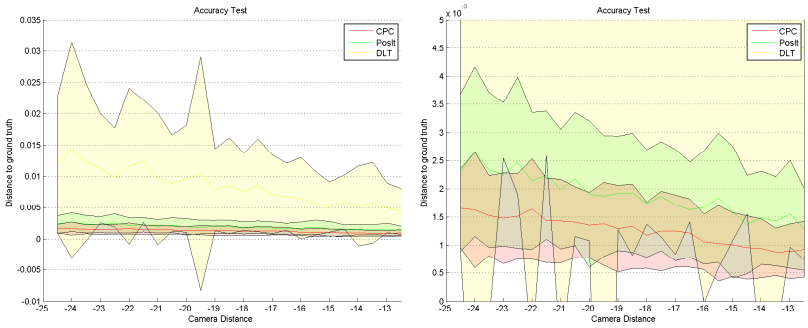


Figure 4.22: Test 4 illustrated

Figure 4.22 shows an illustration of camera starting position (black) and an example pose after moving the camera closer to the point cloud (blue).

This is a test to find out what happens to PosIt mainly, when the SOP image is very wrong because the points are spread widely on the Z axis. This means the depths of the points vary a lot, seen from the camera. The point cloud is standard -5 to 5 and the camera starts at a distance of 25 away and then moved 1 closer each step.

Figure 4.23 shows translation error when the camera is moved closer to the point cloud. On (b) it is seen that CPC is the better method but PosIt does not have problems with the SOP at this very close distance.



(a) Translation error for all methods when the camera is moved closer to the points cloud.

(b) Translation error for all methods when the camera is moved closer to the points cloud. Zoomed in.

Figure 4.23: Translation error for all methods when camera is moved closer. With mean and standard deviation.

As seen on figure 4.23 the error slowly decreases as the camera closes in on the points. This is because of the fixed noise level and pixel size. When the camera is far from the point cloud, there is more noise relative to the size of the 3D object than when it is close as seen on figure 4.24.

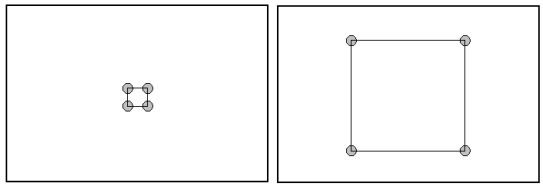
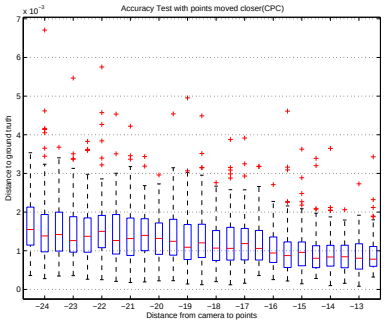
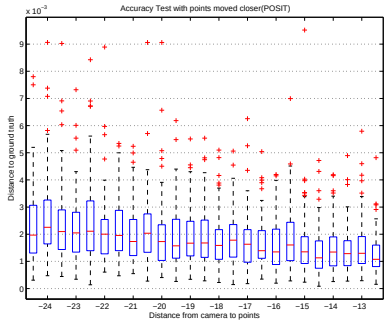


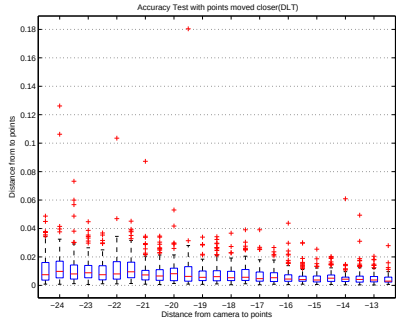
Figure 4.24: Left square shows a 2D image of a box consisting of 4 points, and the noise affecting the image is marked as a circle around each of the 4 visible points. Right square shows a 2D image of the same box and same 4 visible points but moved closer to the camera and the noise is kept the same.



(a) CPC translation error for camera moved closer to point cloud.



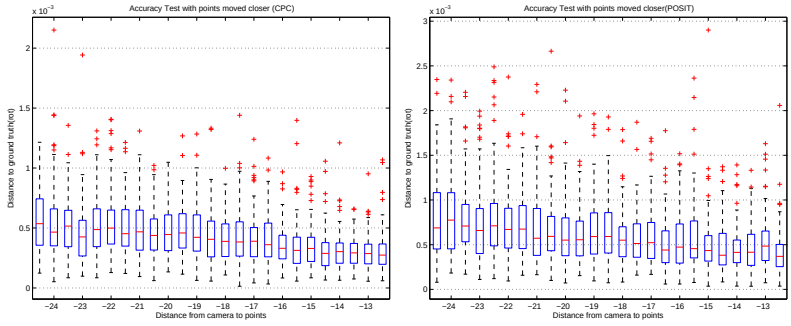
(b) PosIt translation error for camera moved closer to point cloud.



(c) DLT translation error for camera moved closer to point cloud.

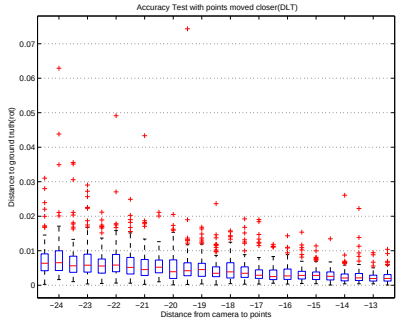
Figure 4.25: Translation error for camera moved closer to point cloud. Note the axes are different on the 3 plots.

Figure 4.25 shows boxplots of translation error when camera is moved closer to the object points. CPC(a) improves by a factor of 1.83 from first step to the last. PosIt(b) 1.77 and DLT(c) by 2.68. This is expected that DLT is very sensitive when it comes to noise and PosIt slightly more effected by noise compared to CPC.



(a) CPC rotation error for camera moved closer to point cloud.

(b) PosIt translation error for camera moved closer to point cloud.



(c) DLT translation error for camera moved closer to point cloud.

Figure 4.26: Rotational error for camera moved closer to point cloud.

Figure 4.26 shows rotational error when camera is moved closer to the object. CPC(a) improved rotational error by a factor of 1.857 from first frame to the last, PosIt(b) by 1.92 and DLT(c) by 3.21. This means that the distance to the object and the noise are affecting DLT a lot more than the 2 other methods. As expected CPC did only react on the noise level but PosIt that works because it can extract the depth did not have problems when depth became large compared to the camera distance.

4.1.7 Test 5: Testing the initial guess for CPC

The camera is fixed but the initial guess is moved around the center of the point cloud and in a full circle for illustrative purpose also because half a circle would be enough. The camera is always looking at $0, 0, 0$ and keeps constant 25 distance to this center position as it goes 360 degrees around the center of the point cloud.

Each step in the spin is 5.1 degree, from 0 to 360 in 70 steps, so it gets back to where it started.

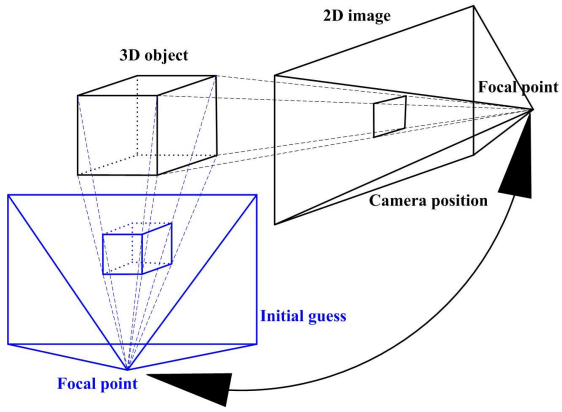
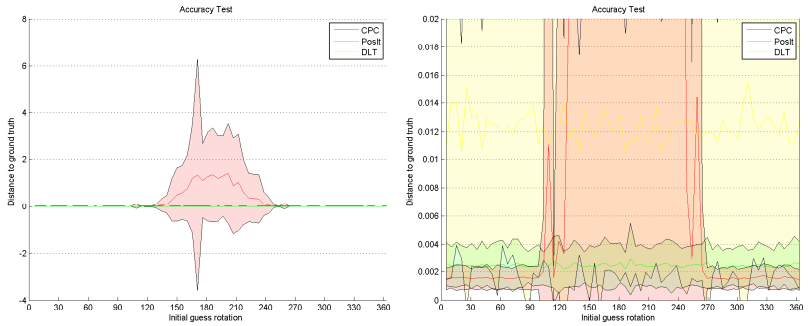


Figure 4.27: Test 5 illustrated.

Figure 4.27 shows the fixed camera (black) and an example of rotated initial guess (blue).

Previously the initial guess for CPC was given to be the true for the first frame and after that it would use the previous frame. This is not a complete fair comparison and that is why this test is made to show what happens when the initial guess is far from the true pose. The guess starts with being correct as in previous tests but after that the guessed pose turns around the point cloud 360 degrees. Figure 4.28 shows all methods but only CPC is varying because of the initial guess. Yet it cannot reach this accuracy when the initial guess is opposite side of the point cloud but works better than the other methods for angles less than 90 degrees see figure 4.28 (b) for zoom in version. With an angle less than 30 degrees CPC keeps the same time (see figure 4.30) as the other methods and also better accuracy (see figure 4.28 and 4.29).

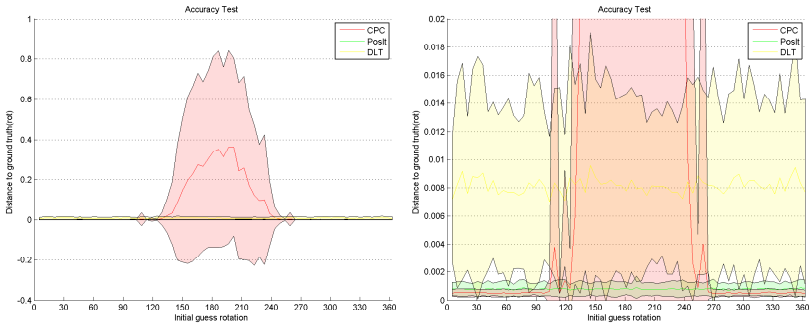


(a) Translation error for all methods when CPC is given increasingly wrong initial guess.

(b) Translation error for all methods when CPC is given increasingly wrong initial guess. Zoomed in.

Figure 4.28: Translation error for all methods when CPC is given increasingly wrong initial guess. Shown with mean and standard deviation

Figure 4.29 shows the rotational error for increasingly wrong initial guess for CPC. Shown with mean and standard deviation. Note that sometimes it finds the correct rotation this is because of the point destribution. This can be seen when looking at the numbers that makes this plot, sometimes it does find the correct pose with minimal error. Figure (b) shows that after about 90 degrees CPC is not stable anymore.

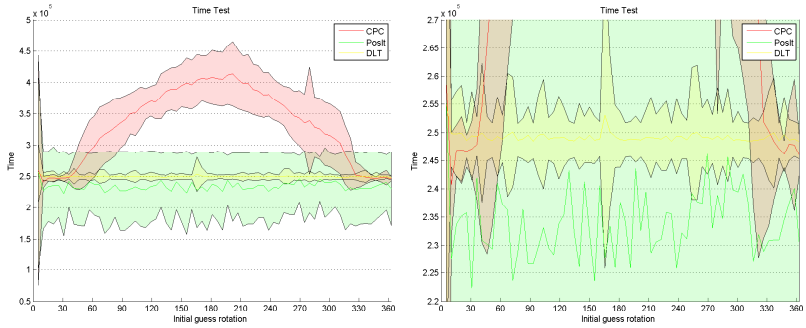


(a) Rotational error for all methods when CPC is given increasingly wrong initial guess.

(b) Rotational error for all methods when CPC is given increasingly wrong initial guess. Zoomed in.

Figure 4.29: Rotation error for all methods when CPC is given varying initial guess. With Mean and standard deviation.

On figure 4.30 (a) you can see that CPC uses more and more time for same number of iterations to try and keep the accuracy and uses the most time exactly opposite of the correct pose. When the initial guess is worse that 30 degrees CPC starts increasing the time it takes to find a proper pose (seen on 4.30 (b)) but it does find correct pose according to figure 4.29 (b) and 4.28 (b).



(a) Time for all methods when CPC is given increasingly wrong initial guess.

(b) Time for all methods when CPC is given increasingly wrong initial guess. Zoomed in.

Figure 4.30: Time for all methods when CPC is given varying initial guess. Shown with mean and standard deviation.

Interesting that sometimes CPC finds the correct pose when starting opposite of the correct pose and sometimes it does not. This can be seen on figure 4.28 because it shows error of 0 at 180 degrees and same for rotation figure 4.29, but specifically when checking the actual results the plots are based on the error is close to 0 for both rotation and translation in some cases. In this test the only thing that changes are the point correspondences so when choosing object for real application one needs to make sure it is working properly when initial guess is way off. This is mostly not the case in video sequences because there is not much change from 1 frame to the next.

4.1.8 Test 6: Different implementation of PosIt

OpenCV is an open source image library by Intel [1]. It is highly optimized and recommended for fastest version of the PosIt method. The reason not to use this faster version in this thesis is a fair comparison between CPC and the PosIt that are both compiled by me and written in BIAS [10]. The OpenCV version is here tested as in Test 1 for noise increasing to show how fast it is compared to the PosIt version used in this thesis.

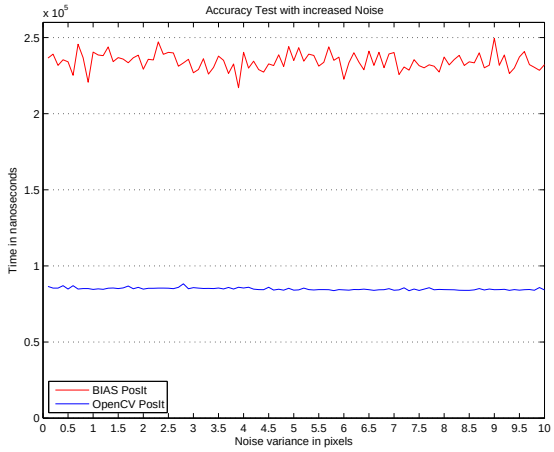


Figure 4.31: 2 different versions of PosIt plotted run time when the accuracy is the same.

[1] http://www.intel.com/technology/computing/opencv/index.htm

Figure 4.31 shows PosIt from BIAS and PosIt from OpenCV with mean time for the same number of iterations over 100 runs as the noise increases (noise is not relevant but the test is taken from Test 1). OpenCV PosIt has an average time of 0.849 ms and the BIAS version has 2.34 ms that is 2.7 times slower but otherwise they are exactly the same. CPC could be optimized as well for better performance by using a different compiler or different compiler settings. But using the slower version will keep the methods comparable.

Chapter 5

Conclusion

Gauss-Newton with Levenberg-Marquardt modification, PosIt, PosIt for coplanar points and finally the direct linear transform has been tested in various extremes to reveal positive and negative sides.

First was tested how they were affected by noise. Then tested what happens when more point correspondences are added. And tested what happens when the object is at the limit of planarity. Then tested what happens when the object is close to the camera. Finally tested what happens when initial guess is very bad for iterative method CPC.

CPC is iterative and relies on minimizing the back projected error of a pose when changing the parameters of the projection matrix according to first order derivatives of the residual and the approximated hessian without the second order derivatives which is considered time consuming.

PosIt is approximating the depth in the image for every 2D point it has correspondence for, starting from pure orthographic projection and then iteratively improves this to estimate perspective projection.

The planar version of PosIt works like PosIt except it finds 2 rotation matrices for every previous found pose and keeps choosing the better 2 poses. Ending up with 2 poses instead of 1 like PosIt.

DLT uses SVD to linearly find the 11 parameters of the projection matrix and therefore requires more points than the other methods and it also has other drawbacks, but it is used as comparison for the other methods.

Noise affect the methods very differently, as expected DLT is the worst but CPC is slightly better than PosIt handling noise. This is important if the noise level in a practical application is really high it might not work accurate enough.

When the methods get more point correspondences it mostly affect DLT that increased run time by a factor 8, CPC only with a factor of 1.84 and PosIt in the middle with a factor of 2.76.

Interestingly the planarity was affecting all the methods in some way. Mostly PosIt which error slowly increased as the object became planar. And the planar version of PosIt that did not work at all for near planar objects but gave a decent guess for objects with volume, meaning it worked for planar objects and non-planar but not the transition between these. And CPC became less accurate when the object was planar.

Important thing about CPC is the initial guess and as proven in Test 5 the point distribution determines how well the initial guess have to be. Also shown is that the initial guess has to deviate more than 30 degrees before anything happens to the run time of CPC.

-CPC is best when you got a movie sequence and know the pose from the last frame, it is very accurate but can give very wrong results if the initial guess is more off than 90 degrees.

-PosIt has the drawback of coplanar points but it is almost as fast as CPC and does not require initial guess which means it is perfect if you got 1 image of points that are sure not to be coplanar.

-Coplanar PosIt is just as fast and more accurate for translation error compared to CPC in the case where points are coplanar, it can also give a valuable clue to how likely the pose is to be the right one by providing the 2 best poses possible. This is unique for this method and cannot be done by either of the other methods. Negative that it needs perfectly 2D marker as in ARToolKit.

-DLT was used as a comparator for the other methods. If the 2D-3D correspondences are without noise the DLT works good but extreme sensitivity to noise makes it a bad choice for any real application.

Appendix A

The Test Program

Because the methods are implemented in C++ it is natual to write the testing program in that lagnauge also. The values from each test is stored in *.txt and read by a Matlab program that draws the graphs. Here is a description of all functions in C++ and their usage.

void init() -Initializes camera position, kalibration matrix and glut camera settings.

void drawFloor(float size, int nrQuadsPerRow) -Makes a green square in the middle of the points with size 10x10 to give a better 3D effect.

void drawPoint() -Draws 1 point at the current position.

void display() -Glut draws the scene with objects.

void reshape(int w, int h) -Resets the glut camera settings.

void SaveData(const char* file, int type,int x, int y) -Saves the translation error, rotation error and run time for last run method to *.txt.

void SavePMatrix() -Stores correct projection matrices for each frame to make ground truth for comparison.

void AddNoise() -2D points made from ground truth projection of the 3D object is added noise.

void AddPlanarMovement() -3D points are morphed 30% closer to planarity parallel to image plane.

void AddPlanarMovement2() - 3D points are morphed in the diagonal 30% towards planarity, angled 0.25 π.

void ProjectPointsTruth() - Projects the 3D object points with ground truth projection matrix to make perfect 2D points.

void CamPoseCalib_algo() -Gauss-Newton method. Fills matrices with translation, rotation error and the run time.

void CamPoseCalib_algo_time(float time_limit) -Sets a time limit for CPC so it increases/decreases number of iteration until it reaches the desired run time.

void OpenCVPoseIT() -PosIt method from OpenCV. Fills matrices with translation, rotation error and the run time.

void OpenCVPoseIT_time(float time_limit) -Sets a time limit for PosIt so it increases/decreases number of iteration until it reaches the desired run time.

void PositWithPlanar() -Both PosIt and PosIt for coplanar points depending on the variable "dim". When dim is 3, it means the points are spread in 3D and it will use PosIt. If dim is 2, it means the points are on a plane and it runs planar version of PosIt.

void ARToolkit() -ARToolKit not used.

double arGetTransMatT( ) -ARToolKit not used.

double arGetTransMat2T(double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num) -ARToolKit not used.

double arGetTransMat3T(double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num, double *dist_factor, double cpara[3][4]) -ARToolKit not used.

double arGetTransMat4T(double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num) -ARToolKit not used.

double arGetTransMat5T(double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num, double *dist_factor, double cpara[3][4]) -ARToolKit not used.

static double arGetTransMatSubT(double rot[3][3], double ppos2d[][2], double pos3d[][3], int num, double *dist_factor, double cpara[3][4]) -ARToolKit not used.

void DLT() -Direct Linear transform method. Fills matrices with translation, rotation error and the run time.

void keyPressed(unsigned char key, int x, int y) -Starts the test chosen when "f" is pressed. Single methods can be tested with "i".

void animation() -Can make things rotate. Not used.

void mousePressed(int button, int state, int x, int y) -Used to move the glut camera for viewing the points and camera.

void mouseMotion(int x, int y) -If mouse pressed it can rotate the glut camera.

void noiseTest_time() -Test 1 with fixed time.

void noiseTest_acc() -Test 1 with different number of iterations.

void createSamples() -Creates array with 60 3D points randomly between -5 and +5 on all 3 axes.

int main(int argc, char** argv) -Sets window size and position. Initaializes the glut functions for viewing color 3D and more.

Bibliography

[1] Y. Abdel-Aziz and H. Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. pages 1–18, 1971.

[2] Ronald T. Azuma. A survey of augmented reality. 6 (4), 1997. Hughes Research Laboratories 3011 Malibu Canyon Road, MS RL96 Malibu, CA 90265.

[3] Helder Araújo Rodrigo L. Carceroni Christopher M. Brown. A fully projective formulation for lowe's tracking alogrithm. pages 227–238, 1998.

[4] Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization.* John Wiley & Sons, 2001.

[5] D. Oberkampf & L. S. Davis Daniel DeMenthon. Iterative pose estimation using coplanar feature points. 63 (nr. 3), 1996.

[6] Daniel F. DeMenthon & Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[7] Philip David Daniel DeMenthon Ramani Duraiswami and Hanan Samet. Simultaneous pose and correspondence determination. pages 698–714, 2002.

[8] Vincent Lepetit & Pascal Fua Francesc Moreno-Noguer. Accurate non-iterative o(n) solution to the pnp problem. *ICCV07(1-8)*, 2007.

[9] Daniel Grest. Marker-free human motion capture in dynamic cluttered environments from a single view-point. 2007.

[10] J. F. Evers-Senne J. M. Frahm D. Grest K. Köser B. Streckel J. Bias (basic image algorithms). 2006.

[11] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *IWAR '99, San Francisco*, pages 85–94, 1999.

[12] Vincent Lepetit and Pascal Fua. Monocular model-based 3d tracking of rigid objects: A survey. 1(1), 2005. Computer Vision Laboratory, CH-1015 Lausanne, Switzerland.

[13] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. pages 666–673, 1999. Corfu Greece.