PoseLab: A Levenberg-Marquardt Based Prototyping Environment for Camera Pose Estimation

Michiel Darcis, Wout Swinkels, Aydin Emre Güzel and Luc Claesen
Department of Engineering Technology
Hasselt University
Diepenbeek, Belgium

***Abstract*—A pose identifies the 3D position and 3D rotation of an object with respect to a reference coordinate system. Cameras can be used to determine the pose of objects in a scene. Fiducial markers on a object make it easier to segment the object. For applications where either high-accuracy, high speed and/or very low latency are required, dedicated hardware architectures are needed as regular processors are not performant enough. The dedicated hardware needs to be designed such that the constraints are taken into account in order to meet the requirements. A software prototyping environment PoseLab is presented which can be used to evaluate the effects of various design trade-offs.**

I. Introduction

In many applications it is interesting to know the 3D position and 3D rotation of objects in space with respect to a known coordinate system. The combination of 3D position and 3D rotation is also referred to as the 6 degrees of freedom pose or 6DoF pose. There are a lot of methods that use a camera to determine the 6DoF pose of an object. The position information is usually embedded in known features of a printed color booklet or by a simple black and white pattern that can be detected by an image sensor in a tablet or a VR-headset. It is currently common practice in publicity, virtual models of furniture, gaming etc... Movies often benefit from using human actors to animate 3D virtual persons or animals. The movements of head, limbs, arms, fingers and body is tracked with cameras using visual markers attached to key feature points on the body. 6DoF position determination is also useful in VR-headsets and hand controlled tools. Robotics benefits from accurate position estimation of objects to be manipulated. The objects can be identified by simple dots made out of retro reflective material, so that the reflection of light sources mounted near a camera and pointing to the object are easily detectable.

Camera-based pose estimation is also starting to be used in applications that require high speed and high accuracy. An example of such an application is prospective motion correction [1]. The resolution of MRI images that can be obtained is limited by the movement of the patient inside the scanner. Even small movements, caused for example by the heartbeat or breathing, lower the quality of the resulting image. Camera-based pose estimation is used to track the movement of the patients head in real-time. A marker attached to the head is used by a camera to calculate the pose of the patient. This information is used to adjust the imaging pulse sequence of the scanner accordingly.

The problem today is that software-based methods are computationally expensive and this makes them difficult to integrate in applications where high speed and high accuracy are important. Currently, the prospective motion correction application can only reach 80 Hz [2]. Application-specific hardware is needed to effectively achieve higher frame rates and enable low latencies. Direct hardware implementation has the potential to realize the algorithms exploiting the capabilities of the parallel processing in dedicated hardware circuits. It also has the advantage to be able to directly interface to the image sensor, eliminating unnecessary delays.

Creating a hardware solution is not an automatic process but demands the intelligence of human engineers to be able to obtain efficient results. It requires the understanding of the mathematical problem at hand, the algorithmic options and possibilities to reformulate the problem such that it is suitable for exploiting the capabilities provided by a direct hardware implementation. In order to be able to evaluate various algorithmic and architectural alternatives as well as the potential exploitation of the intricacies of the hardware and image sensor interfaces, a mathematical framework is needed to help the engineer trade-off specific application requirements and possible architectural choices.

This paper presents a prototyping environment PoseLab. It can be used by the engineer to test and evaluate various design trade-offs to help create efficient hardware solutions for camera pose estimation applications. In section II, the basics are explained of how the pinhole camera model is employed to obtain the pose of the camera. Section III describes the Levenberg-Marquardt optimization method that is used to compute the pose. PoseLab is implemented in MATLAB with the goal that every detail of the algorithm can be adjusted. Some of the implementation details are explained in section IV. Section V describes the experiments that were performed to verify the correctness of PoseLab.

II. Pinhole model

To extract the 6DoF information of a camera from known 3D points, a model is needed that describes how a 3D point in the world is projected onto the 2D image plane. The pinhole

camera model is used for this purpose and is illustrated in figure 1. The mapping of a 3D point (X, Y, Z) to the 2D pixel coordinates (x, y) can be summarized in matrix equation (1) [3], [4].

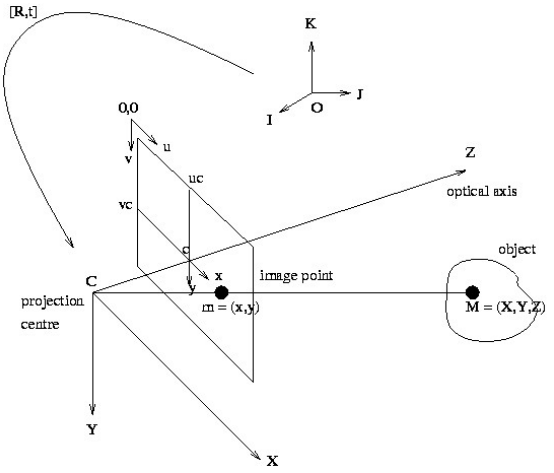


Fig. 1. Projection of 3D point onto 2D image plane using the pinhole camera model [5]

$$\tau \begin{bmatrix} x & y & 1 \end{bmatrix}^T = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T \quad (1)$$

In (1), $\boldsymbol{K}$ is the 3×3 camera calibration matrix and is assumed to be known. This matrix can be estimated using the Caltech camera calibration toolbox [6] or the OpenCV library [7]. Both tools use Zhang's calibration method [8]. However, there are other calibration methods such as Tsai's technique [9]. The extra 1 coordinate comes from the use of normalized homogeneous coordinates. The scaling factor τ results from the fact that homogeneous points are equivalent when their coordinate values are proportional. The 3×3 rotation matrix $\boldsymbol{R}$ and 3×1 translation vector $\boldsymbol{t}$ describe the 6DoF pose of the camera with respect to world coordinate system that is used to represent the 3D points in the world. Despite having 9 parameters, the rotation matrix $\boldsymbol{R}$ has only 3DoF. Using an axis-angle transformation [10], the rotation can be described by only 3 parameters.

The goal is to find the 6DoF pose described by $\boldsymbol{R}$ and $\boldsymbol{t}$. In the assumption that $\boldsymbol{K}$ is already estimated, $\boldsymbol{R}$ and $\boldsymbol{t}$ can be calculated from (1) when enough 2D points $\boldsymbol{x}_j = (x_j, y_j)$ and their corresponding 3D points $\boldsymbol{X}_j = (X_j, Y_j, Z_j)$ are known. This is called the perspective-n-point problem [11]. Each 2D-3D point correspondence gives rise to 2 equations, 1 for x and 1 for y. Thus to estimate the 6 pose parameters, at least 3 2D-3D point correspondences are needed to find a solution for $\boldsymbol{R}$ and $\boldsymbol{t}$. To eliminate ambiguities however, a 4th point correspondence is needed.

The known points identifying the position of an object in a scene should have known coordinates in the object coordinate system. In order to make these easily identifiable spheres, corners on a flat checkerboard etc.. are used. Cameras have a limited accuracy to determine the (x, y) position in the camera image. This is amongst others related to the pixel resolution, noise, characteristics of the point feature in space etc.. When using cameras to determine the pose of an object, the accuracy of the position measurement can be increased by using more than 4 correspondence points.

Another important aspect of the pinhole camera model is that it does not include the lens of the camera. In practice, lenses introduce distortions in the image [3]. To be able to correctly use the pinhole model, these distortions need to be eliminated beforehand. This is done via distortion parameters. The two most important kinds of distortion are radial and tangential distortion. The parameter values that model these distortions can also be found using the Caltech toolbox or OpenCV library.

III. Levenberg-Marquardt for camera pose estimation

There are many solutions for the perspective-n-point problem such as EPnP [12], DLS [13] and P3P [14]. However, the most state-of-the-art methods use the Levenberg-Marquardt non-linear optimization algorithm [15]–[17].

A. Cost Function

Levenberg-Marquardt is used to find the parameters of a model so that it matches the experimental data. In this case, the model is the pinhole camera and the 2D-3D point correspondences are used as the experimental data. A cost function is setup to represent how much the model disagrees with the data. Finding the parameters of the model that minimize this cost function ensures that the best possible fit is found. For Levenberg-Marquardt, the cost function is defined as the summation of the squared error of every data point.

For camera pose estimation, the reprojection error, illustrated in figure 2, is used as the error metric. It is the difference between the 2D image point $\boldsymbol{x}_j$ and the reprojection $\hat{\boldsymbol{x}}_j$ of the corresponding 3D point on the image plane using the pinhole camera model. This reprojection error consists of two parts, the difference in x-coordinate and the difference in y-coordinate as shown by equations (2) and (3). The reprojection depends on the calibration matrix $\boldsymbol{K}$, distortion parameters $\boldsymbol{q}$, rotation matrix $\boldsymbol{R}$, translation vector $\boldsymbol{t}$ and the 3D coordinate $\boldsymbol{X}_j$. For a total of m point correspondences, a vector valued function $\boldsymbol{f}(\boldsymbol{\Theta})$ is defined as shown by (4). The 6DoF pose parameters are put together in a vector $\boldsymbol{\Theta}$. Using this notation, the cost function can be defined as in equation (5). The factor $\frac{1}{2}$ is there to simplify the derivation of the derivatives.

$$d_{j,x} = x_{j,x} - \hat{x}_{j,x}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{X}_j) \quad (2)$$

$$d_{j,y} = x_{j,y} - \hat{x}_{j,y}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{X}_j) \quad (3)$$

$$\boldsymbol{f}(\boldsymbol{\Theta}) = [d_{1,x}, d_{1,y}, ..., d_{m,x}, d_{m,y}]^T \quad (4)$$

$$F(\boldsymbol{\Theta}) = \frac{1}{2} \boldsymbol{f}(\boldsymbol{\Theta})^T \boldsymbol{f}(\boldsymbol{\Theta}) \quad (5)$$

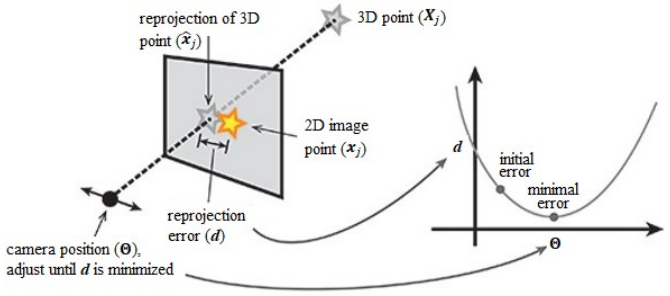


Fig. 2. Minimizing the reprojection error [18]

B. Finding the minimum

To find the minimum of cost function $F(\boldsymbol{\Theta})$, the function is approximated at $\boldsymbol{\Theta}$ by a Taylor series as shown by equation (6). The goal is to find step $\boldsymbol{h}$ that minimizes this approximation. This can be done by taking the derivative of (6) with respect to $\boldsymbol{h}$ and set it equal to zero. This results in equation (7). The vector $\boldsymbol{g} = \left[\frac{\partial F}{\partial \Theta_1}, ..., \frac{\partial F}{\partial \Theta_6}\right]^T$ is the gradient which contains the first-order partial derivatives with respect to every parameter. Matrix $\boldsymbol{H}$ is the Hessian matrix and is defined as $\boldsymbol{H}_{i,j} = \frac{\partial F}{\partial \Theta_i \partial \Theta_j}$ with $i,j \in [1..6]$.

$$F(\boldsymbol{\Theta} + \boldsymbol{h}) = F(\boldsymbol{\Theta}) + \boldsymbol{g}^T\boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{H}\boldsymbol{h} \tag{6}$$

$$\boldsymbol{H}\boldsymbol{h} = -\boldsymbol{g} \tag{7}$$

It can be shown that $\boldsymbol{g}$ is equal to $\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta})$ [15]. The matrix $\boldsymbol{J}$ is called the Jacobian matrix and is given by (8). It contains the first-order partial derivatives of all the entries in $\boldsymbol{f}(\boldsymbol{\Theta})$ with respect to each parameter in $\boldsymbol{\Theta}$. It can also be proven that the Hessian $\boldsymbol{H}$ is equal to $\boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{f}^{''}(\boldsymbol{\Theta})^T\boldsymbol{f}(\boldsymbol{\Theta})$ [15]. In the assumption that $\boldsymbol{f}(\boldsymbol{\Theta})$ is linear, $\boldsymbol{f}^{''}(\boldsymbol{\Theta}) = 0$, Hessian $\boldsymbol{H}$ can be approximated by $\boldsymbol{J}^T\boldsymbol{J}$. Substituting $\boldsymbol{g}$ and $\boldsymbol{H}$ in (7) leads to equation (9), known as the matrix notation of the normal equations.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial d_{1x}}{\partial \Theta_1} & \cdots & \frac{\partial d_{1x}}{\partial \Theta_6} \\ \frac{\partial d_{1y}}{\partial \Theta_1} & \cdots & \frac{\partial d_{1y}}{\partial \Theta_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial d_{mx}}{\partial \Theta_1} & \cdots & \frac{\partial d_{mx}}{\partial \Theta_6} \\ \frac{\partial d_{my}}{\partial \Theta_1} & \cdots & \frac{\partial d_{my}}{\partial \Theta_6} \end{bmatrix} \tag{8}$$

$$(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta}) \tag{9}$$

The minimum of $F(\boldsymbol{\Theta})$ is found in an iterative manner. In every iteration, the step $\boldsymbol{h}$ is calculated that minimizes the linear approximation using (9). This is then used to update the parameters $\boldsymbol{\Theta}$. This process is repeated until convergence.

C. Levenberg-Marquardt

Using (9) to find step $\boldsymbol{h}$ is also known as the Gauss-Newton method. The Levenberg-Marquardt algorithm expands on the Gauss-Newton method by introducing a damping parameter $\lambda > 0$. Equation (10) shows the new normal equations.

When λ is small, equation (10) reduces to (9) and the Levenberg-Marquardt method becomes the Gauss-Newton method. The advantage of Gauss-Newton is that it converges very rapidly when close to the minimum. For large values of λ, (10) reduces to $\lambda\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta})$ or $\boldsymbol{h} = -\frac{1}{\lambda}\cdot\boldsymbol{g}$. Step $\boldsymbol{h}$ is now along the direction of the gradient which represents the steepest descent direction of the cost function. The size of the step is controlled by the factor $\frac{1}{\lambda}$. This approach is known as the gradient descent method. It ensures a rapid decrease when the current solution is far from the minimum. By controlling the value of λ at every iteration, Levenberg-Marquardt can switch between Gauss-Newton and gradient descent to provide rapid convergence. The damping factor λ thus controls the size and the direction of step $\boldsymbol{h}$.

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta}) \tag{10}$$

IV. PoseLab implementation details

PoseLab uses the Levenberg-Marquardt optimization method described above to find the 6DoF pose of a camera. This section explains some of the details on how PoseLab is implemented in MATLAB.

A. Axis-angle transformation

As mentioned earlier, the rotation of the camera can be described by 3 parameters using an axis-angle representation. A 3×1 vector $\boldsymbol{w} = [w_x, w_y, w_z]^T$ indicates the axis of rotation. The magnitude $\|\boldsymbol{w}\|$ specifies the angle of rotation around $\boldsymbol{w}$. It can be shown, through a power series expansion, that $e^{\lfloor \boldsymbol{w} \rfloor_x}$ represents a three dimensional rotation matrix [10]. Here, $\lfloor \boldsymbol{w} \rfloor_x$ is the skew-symmetric matrix of the vector $\boldsymbol{w}$. The relationship between a vector $\boldsymbol{w}$ and its corresponding rotation matrix can be derived from the power series and is described by (11). In the case where $\boldsymbol{w} = \beta\hat{\boldsymbol{w}}$, with $\hat{\boldsymbol{w}}$ being the unit vector of the rotation axis and β the angle around this axis, equation (11) simplifies to (12). This is known as the Rodrigues transformation and is the most widely implemented representation. For this reason, the Rodrigues transformation is used by PoseLab to switch between the rotation matrix $\boldsymbol{R}$ and the axis-angle representation $\boldsymbol{w}$.

$$e^{\lfloor \boldsymbol{w} \rfloor_x} = \boldsymbol{R} = \boldsymbol{I} + \frac{sin(\|\boldsymbol{w}\|)}{\|\boldsymbol{w}\|}\lfloor \boldsymbol{w} \rfloor_x + \frac{1 - cos(\|\boldsymbol{w}\|)}{\|\boldsymbol{w}\|^2}\lfloor \boldsymbol{w} \rfloor_x^2 \tag{11}$$

$$\boldsymbol{R} = \boldsymbol{I} + sin(\beta)\lfloor \hat{\boldsymbol{w}} \rfloor_x + (1 - cos(\beta))\lfloor \hat{\boldsymbol{w}} \rfloor_x^2 \tag{12}$$

B. Reversing lens distortions

A necessary step before a 2D-3D correspondence can be used in the optimization, is to correct the 2D image point for lens distortion. The implementation in PoseLab is based on the OpenCV library [19]. Equations (13) and (14) show the distortion model that is utilized. It contains 8 distortion parameters, 6 for radial distortion $(k_1, ..., k_6)$ and 2 for tangential

distortion (p_1, p_2). The value r is the distance of the distortion free coordinate to the point where the image plane intersects the optical axis, known as the principal point.

$$x_{dist} = x \cdot \frac{1 + k_3 r^6 + k_2 r^4 + k_1 r^2}{1 + k_6 r^6 + k_5 r^4 + k_4 r^2} + (2p_1 xy + p_2(r^2 + 2x^2)) \tag{13}$$

$$y_{dist} = y \cdot \frac{1 + k_3 r^6 + k_2 r^4 + k_1 r^2}{1 + k_6 r^6 + k_5 r^4 + k_4 r^2} + (p_1(r^2 + 2y^2) + 2p_2 xy)) \tag{14}$$

An iterative algorithm is used to remove the lens distortions [20], [21]. First, the lens distortion is estimated using the coordinate, and thus r, of the original distorted point. However, when the distortion is again applied to the undistorted point using its coordinate, the reprojection will not be at the original image point and this gives rise to an error. The Euclidean distance is used as the error metric. The algorithm then uses the coordinate of the undistorted point to refine the estimate of the lens distortions. This is repeated until a point is found that, when the distortion model is applied, leads to an error that is smaller than a predetermined tolerance. Another stopping criterion is when a maximum number of iterations is reached.

C. Calculating the Jacobian

The Jacobian matrix is given by equation (8). The computation of this matrix is taken from [17]. Here, the calculation happens on a row-per-row basis. Before executing the Levenberg-Marquadt optimization, symbolic expressions need to be found for the row vectors $\left[\frac{\partial d_{jx}}{\partial \Theta_1}, \cdots, \frac{\partial d_{jx}}{\partial \Theta_6}\right]$ and $\left[\frac{\partial d_{jy}}{\partial \Theta_1}, \cdots, \frac{\partial d_{jy}}{\partial \Theta_6}\right]$. The expressions are determined using the symbolic math toolbox of MATLAB. Equations (2) and (3) are written in symbolic form and then the $jacobian$ function of the toolbox can be used to find the symbolic expressions of the two 2 row vectors. During the Levenberg-Marquardt iterations, the Jacobian can then be constructed by substituting the symbolic values in the symbolic expressions with the values available at the current iteration.

D. Calculating the step

Previously, it was stated that (10) should be solved for $\boldsymbol{h}$ to obtain the step. However, there is another way to calculate step $\boldsymbol{h}$. It can be shown that (10) is the solution to the linear least squares problem shown by (15). This linear least squares problem, $\boldsymbol{A}\boldsymbol{x}^* \approx \boldsymbol{b}$, can also be solved via orthogonal transformation. This method is more accurate as the normal equations can be ill-conditioned [22]. Here, $\boldsymbol{A}$ is the matrix $\begin{bmatrix} \boldsymbol{J} \\ \sqrt{\lambda}\boldsymbol{I} \end{bmatrix}$ and $\boldsymbol{b}$ is equal to $-\begin{bmatrix} \boldsymbol{f}(\boldsymbol{\Theta}) \\ \boldsymbol{0} \end{bmatrix}$. The vector $\boldsymbol{x}^*$ that needs to be found corresponds in this case to the step $\boldsymbol{h}$.

First, the QR decomposition of the matrix $\boldsymbol{A}$ needs to be calculated so that $\boldsymbol{Q}^T\boldsymbol{A} = \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{bmatrix}$, where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{R}$ an upper triangular matrix. The solution for $\boldsymbol{x}^*$ is then found by solving $\boldsymbol{R}_1\boldsymbol{x}^* = \boldsymbol{Q}_1^T\boldsymbol{b}$ using backward substitution [23].

$$\begin{bmatrix} \boldsymbol{f}(\boldsymbol{\Theta}) \\ \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{J} \\ \sqrt{\lambda}\boldsymbol{I} \end{bmatrix} \boldsymbol{h} \approx \boldsymbol{0}. \tag{15}$$

E. Updating the damping factor

To obtain an appropriate value for the damping factor λ, the mechanisms proposed in [15] are used. Here, λ is initialized as a value proportional to the maximum value of the diagonal of matrix $\boldsymbol{J}^T\boldsymbol{J}$. This can be expressed as $\eta \cdot max\left\{diag(\boldsymbol{J}^T\boldsymbol{J})\right\}$ where η is a small value and was chosen to be 10^{-8}. The update of λ at every iteration is based on the gain ratio ϱ defined by equation (16). When the gain ratio is equal or less than zero, step $\boldsymbol{h}$ does not lead to a decrease in cost function. In this case, λ is multiplied by a factor υ, which is initialized to 2. The factor υ itself is doubled and a new iteration is started in an attempt to find a better step $\boldsymbol{h}$. If the gain ratio ϱ is larger than zero, the cost function decreases by applying step $\boldsymbol{h}$ and λ is multiplied by $max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$. The factor υ is set equal to 2. This method does not give rise to flutter which can be present in other approaches and slows down convergence.

$$\varrho = \frac{F(\boldsymbol{\Theta}) - F(\boldsymbol{\Theta} + \boldsymbol{h})}{\frac{1}{2}\boldsymbol{h}^T(\lambda\boldsymbol{h} - \boldsymbol{g})} \tag{16}$$

F. Stopping criteria

In [15] they suggest 3 ways to stop the Levenberg-Marquardt from iterating. Firstly, it is known that the gradient $\boldsymbol{g} = \boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta})$ should be equal to 0 at the minimum. This condition is met if the supremum norm $\|\boldsymbol{g}\|_\infty$ is less than a predetermined threshold ε_1. The threshold is chosen to be 10^{-8}. Secondly, the algorithm also needs to stop when the step $\boldsymbol{h}$ is too small. This is when the Euclidean norm $\|\boldsymbol{h}\|$ is smaller than $\varepsilon_2(\|\boldsymbol{\Theta}\| + \varepsilon_2)$. Here, ε_2 is also set equal to 10^{-8}. Finally, a maximum number of iterations is set to prevent an infinite loop.

V. Experiments

A. The Test Bench

A test bench was made to verify the correctness of PoseLab [24]. An A4 1200 dpi flatbed scanner (1200 dpi $\times$ 1200 dpi) is mounted on a table. A custom construction has been made to firmly attach an 8×11 checkerboard marker to the scanner bar. The rear camera of a Samsung Galaxy S8 is used to take the images and is clamped on a stand. A bluetooth click device is used to take images remotely. This way, there is no need to touch the camera which can alter its position. The AdaFruit Arduino Motorshield v1 [25] is used to drive the stepper motor of the scanner. The laptop connected to the Arduino is put on a different table.

The stepper motor moves the checkerboard pattern along the shaft on the flatbed scanner. It can be assumed that a precise linear motion is possible because in a scanner it is important to position the head correctly with an accuracy of 1200 dpi which corresponds to 20 μm. The measurements have been done for movements of entire steps taken by the stepper motor.

This is done to get better repeatable results and to avoid less accurate positioning when microstepping would be used. It was measured that the checkerboard moved 0.673 mm $\pm$ 0.003 mm when one full step is taken.

B. The Procedure

A total of 4 experiments have been made. In each case, the checkerboard starts at the end of the scanner and is moved linearly along the scanner. The checkerboard is moved 10 full steps after which an image is taken. This is repeated until a total of 250 steps are made. This leads to a total of 26 positions or 26 images per experiment. For each image, the pose of the camera is calculated with respect to the world coordinate system defined in the upper left corner of the checkerboard as illustrated in figure 3. Figure 4 shows the starting position for each of the 4 experiments. In experiment 1, the checkerboard is moved in the Y-direction. Secondly, the checkerboard is moved in the Z-direction. In the third experiment, the checkerboard is turned 45 degrees and moves away from the camera. Finally, the camera is turned roughly 45 degrees with respect to the scanner so that the movement is not along the optical axis.

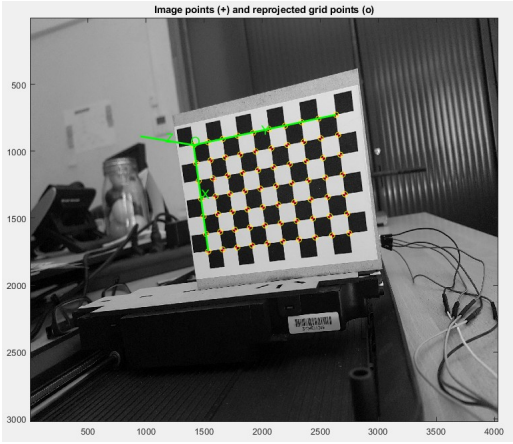


Fig. 3. The coordinate system

C. Results

For every experiment there were a total of 26 estimated positions. Because a linear movement was made, it should be possible to fit straight lines to the course of each coordinate. Table 1 gives the slopes of these straight lines. To evaluate how well the straight lines match the estimated coordinate values, the deviations from the straight line, or residuals, for the X-, Y- and Z-coordinates for every experiment are given in figure 5.

TABLE I
SLOPES OF THE FITTED STRAIGHT LINES (MM PER 10 STEPS)

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
X	-0.0260	-0.3565	-0.0711	-0.3299
Y	-6.691	0.0446	-4.8659	-0.1034
Z	-0.0561	6.861	4.8655	6.8632

D. Discussion

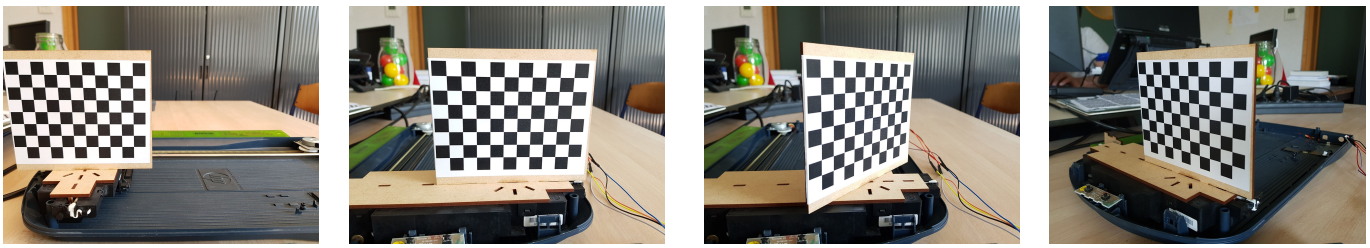
Observing the plots of the residuals, it can be concluded that the fitted straight lines matched the data well. From the Y-coordinate of experiment 1 and the Z-coordinate of experiment 2 and 4, it is possible to retrieve the size of a full step of the stepper motor. Because 10 steps are taken between images, dividing the slope by 10 gives the change in the coordinate value as a result of a single step. In experiment 1, this leads to a change of 0.6691 mm in Y-coordinate per step. In experiment 2, a step changes the Z-coordinate by 0.6861 mm. Similarly, a step in experiment 4 leads to a change of 0.6863 mm. These values are close to the measured step size of 0.673 mm.

The deviation from the measured value can have many causes. The main cause is that the test setup is a low cost solution. There is no way to align the optical axis of the camera in the correct manner. Furthermore, the checkerboard is not perfectly parallel to the scanner. These reasons not only cause the measured step size to be different, but also induces a change in the values of the other coordinates. That is why their slopes are not equal to 0, as indicated in table I.

From the residuals, it can be noted that the size of the deviations from a linear motion is generally smaller in the Z-coordinate than in the X- and Y-coordinate. This is unusual because measuring a change in depth is harder than measuring a change in the X- or Y-direction. A reason can be that the distortion parameters were not accurately estimated because the checkerboard was placed in the middle of the images used during calibration while the distortion is most significant at the edges of the image. In addition, the accuracy of the pose estimation itself plays a role. The ability to accurately retrieve the 2D-3D point correspondences is the main obstacle. Many factors need to be taken into account. Some of them are: resolution, quality of the marker, lightning conditions and the distance to the camera. The quality of the camera, and in particular the lens, also has an influence.

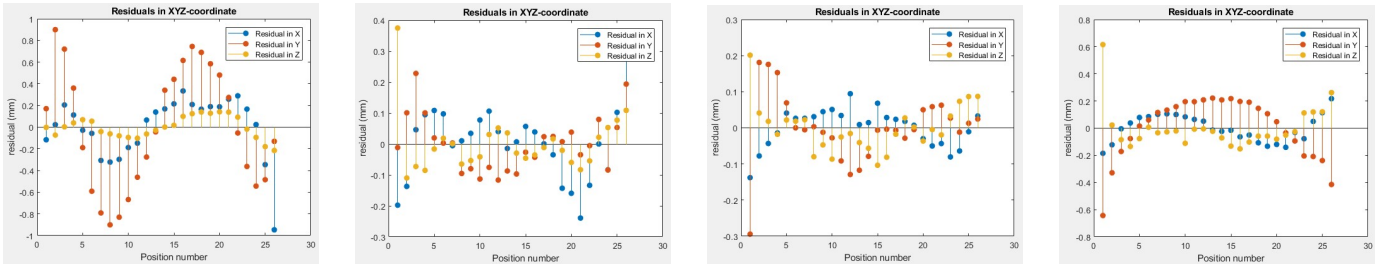
VI. CONCLUSION

Currently, the software algorithms to calculate the pose of a camera are too computationally expensive to achieve high frame rates in real-time applications. Application-specific hardware will be needed to meet the requirements. This paper presents PoseLab, an experimentation platform created in MATLAB. Because everything in the implementation is fully adaptable, an engineer can use it to evaluate various application-specific trade-offs and architectural choices which will facilitate the development of dedicated hardware solutions. PoseLab calculates the pose of the camera using 2D coordinates in an image with known 3D coordinates in the world. After the 2D coordinates are compensated for lens distortions, the Levenberg-Marquardt optimization method is used to find the optimal pose of the camera. To verify that PoseLab is correctly implemented, a total of 4 linear movement experiments were made. The calculated values agreed well with the specified linear movement.



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3 (d) Experiment 4

Fig. 4. Starting positions of the different experiments



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3 (d) Experiment 4

Fig. 5. Deviations from the fitted straight lines for every experiment

REFERENCES

[1] J. Maclaren, M. Herbst, O. Speck, and M. Zaitsev, "Prospective motion correction in brain imaging: A review," *Magnetic Resonance in Medicine*, vol. 69, no. 3, pp. 621–636, 2013.

[2] J. Maclaren, B. S. R. Armstrong, R. T. Barrows, K. A. Danishad, T. Ernst, C. L. Foster, K. Gumus, M. Herbst, I. Y. Kadashevich, T. P. Kusik, Q. Li, C. Lovell-Smith, T. Prieto, P. Schulze, O. Speck, D. Stucht, and M. Zaitsev, "Measurement and Correction of Microscopic Head Motion during Magnetic Resonance Imaging of the Brain," *PLoS ONE*, vol. 7, no. 11, p. e48088, 2012. [Online]. Available: http://dx.plos.org/10.1371/journal.pone.0048088

[3] G. Bradski and A. Kaehler, "Camera Models and Calibration," in *Learning OpenCV*, 1st ed. Sebastopol: O'Reilly Media, 2008, pp. 370–403. [Online]. Available: http://shop.oreilly.com/product/0636920022497.do

[4] C. Stachniss, "Photogrammetry I - 15a - Camera Extrinsics and Intrinsics," 2015. [Online]. Available: https://www.youtube.com/watch?v=DX2GooBIESs&index=28&list=PLKEuOzA9RZ2WnWf7vj2hQEXRIeB8hpxHa

[5] R. Owens, "Camera calibration," 1997. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT9/node2.html

[6] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab," 2015. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[7] "OpenCV Camera Calibration and 3D Reconstruction," 2018. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[8] Z. Zhang, "A Flexible New Technique for Camera Calibration (Technical Report)," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2002.

[9] R. Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[10] R. Hartley and A. Zisserman, "Representations of rotation matrices," in *Multiple View Geometry in computer vision*, 2nd ed. New York: Cambridge University Press, 2004, pp. 583–587.

[11] R. Duraiswami, "Lecture 23: 3-D Pose Object Recognition," University of Maryland Institure for Advanced Computer Studies, Tech. Rep., 2005. [Online]. Available: http://legacydirs.umiacs.umd.edu/ ramani/cmsc426/Lecture23_3Dpose.pdf

[12] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2009.

[13] J. A. Hesch and I. Stergios, "A Direct Least - Squares (DLS) Method for P n P," in *IEEE International Conference on Computer Vision*, 2011, p. 4.

[14] G. Xiao-Shan, H. Xiao-Rong, and T. Jianliang, "Complete solution classification for the perspective-three-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[15] K. Madsen, H. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems," Technical University of Denmark, Tech. Rep., 2004.

[16] R. Hartley and A. Zisserman, "Iterative Estimation Methods," in *Multiple View Geometry in computer vision*, 2nd ed. New York: Cambridge University Press, 2004, pp. 597–602.

[17] P. Mittrapiyanuruk, "A Memo on How to Use the Levenberg-Marquardt Algorithm for Refining Camera Calibration Parameters," Purdue University, Tech. Rep. [Online]. Available: https://engineering.purdue.edu/kak/computervision/ECE661_Fall2012/homework/hw5_LM_handout.pdf

[18] M. Gourlay and R. Held, "Head-Mounted-Display Tracking for Augmented and Virtual Reality," 2017. [Online]. Available: http://informationdisplay.org/IDArchive/2017/JanuaryFebruary/FrontlineTechnologyHeadMountedDisplay.aspx

[19] OpenCV, "undistort." [Online]. Available: https://github.com/opencv/opencv/blob/master/modules/imgproc/src/undistort.cpp

[20] P. Drap and J. Lefèvre, "An Exact Formula for Calculating Inverse Radial," *MDPI Sensors*, no. 1, pp. 1–18, 2016.

[21] P. Abeles, "Inverse Radial Distortion Formula." [Online]. Available: http://peterabeles.com/blog/?p=73

[22] S. Marschner, "CS3220 Lecture Notes : QR factorization and orthogonal transformations," Cornell University, Tech. Rep. March, 2009.

[23] H. Douglas, "Backward substitution," 2005. [Online]. Available: https://ece.uwaterloo.ca/ ece204/howtos/backward/

[24] T. Aerts, "Calibration and evaluation system for 3D camera systems," Ph.D. dissertation, Hasselt University, 2018.

[25] L. Ada, "Adafruit Motor Shield," 2015. [Online]. Available: https://learn.adafruit.com/adafruit-motor-shield/overview