2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Novel Levenberg-Marquardt based methods for application-specific hardware enabled high-speed, high-accuracy, six degrees of freedom camera-based pose estimation

PROMOTOR :
Prof. dr. ir. Luc CLAESEN

COPROMOTOR :
Prof. dr. ir. Nele MENTENS

BEGELEIDER :
De heer Wout SWINKELS

Michiel Darcis

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Novel Levenberg-Marquardt based methods for application-specific hardware enabled high-speed, high-accuracy, six degrees of freedom camera-based pose estimation

PROMOTOR :
Prof. dr. ir. Luc CLAESEN

COPROMOTOR :
Prof. dr. ir. Nele MENTENS

BEGELEIDER :
De heer Wout SWINKELS

Michiel Darcis
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

UHASSELT KU LEUVEN

Acknowledgements

My interest in cameras and images started during my Bachelor's thesis where I worked on an image processing application. When looking for a topic for my Master's thesis, I wanted to further my knowledge in the computer vision area. In this work, I had to catch up on a lot of theory. However, this was more a blessing than a curse. I loved diving into the books. I did not want to stop until I fully understood how it worked. This brought with it the necessary frustrations, but the experience you have when something finally makes sense is so rewarding.

My sincere thanks goes to Prof. dr. ir. Luc Claesen. This thesis would not be possible without his ideas, vast knowledge and guidance. I could always walk into his office whenever I had questions. It is also funny to think how at the start of his career, he was the supervisor of my uncle. Now many years later, he got to have another member of the family under his wings.

I would also like to thank ing. Wout Swinkels. Sharing an office with him was a great experience. His advice and support were very important to this thesis. I am also grateful for the many discussions and the laughter we had during the year.

This Master's thesis is the cherry on the cake of my student career at the University of Hasselt. Looking back at the beginning, I can conclude that I have grown a lot as a person. I would like to thank my friends Karel Bertrands en Sander Denorme who have taken this journey with me. Finally, I want to thank my family for always believing in me.

Michiel Darcis June 2018

Contents

List of Tables

List of Figures

Abstract

This Master's thesis addresses the measurement problem of camera-based six degrees-of-freedom (6-DoF) pose determination. The intention is to use this method in applications where speed and accuracy are critical. Examples are position feedback for haptic man-machine interface systems for use in maxillo-facial surgery planning, prospective motion correction and stereotactic surgery. They require high accuracies (tens of microns), sampling rates higher than 1kHz and latencies shorter than 1msec. The goal of this thesis is to set up a detailed experimental model for the development of complex algorithms that enables the evaluation of architectural design decisions.

Software-based methods for 6-DoF camera based pose estimation exist. However, due to much too low frame rates (60Hz) and much too high latencies (tens of msecs), they are not suited for the envisioned measurements. They are also not directly implementable in hardware. The applications will demand dedicated cameras with FPGA based hardware architectures, directly interfacing with and employing all capabilities of modern image sensors.

To enable future development of 6-DoF camera based hardware architectures, this thesis has studied pose estimation methods and worked out a Levenberg-Marquardt optimization based prototyping environment "PoseLab" that will enable to analyze, subdivide, evaluate and experiment with various trade-offs (accuracy, sampling frequencies, latencies, camera resolution, hardware complexity, ...).

Abstract (Nederlands)

Deze masterproef behandelt het probleem van camera gebaseerde pose bepaling met 6 vrijheidsgraden. Het is de bedoeling om deze methode in de toekomst te gebruiken voor contactloze positiebepalingen waar de snelheid en accuraatheid een kritieke rol spelen. Voorbeelden hiervan zijn positie terugkoppeling voor haptische interface systemen in maxillo faciale chirurgie, prospectieve bewegingscorrectie en stereotactische chirurgie. Deze vereisen een hoge accuraatheid, bemonsteringsfrequenties groter dan 1kHz en vertragingen lager dan 1msec. Het doel van deze masterproef is het opzetten van een gedetailleerd experimenteermodel voor het ontwikkelen van gespecialiseerde algoritmen waarbij architecturale ontwerpbeslissingen kunnen geëvalueerd worden.

Software methodes om de camera pose te bepalen bestaan maar zijn dankzij hun te lage frame rates (60Hz) en te hoge vertragingen (tientallen msec) niet geschikt. Ze zijn ook niet meteen om te zetten naar hardware. Voor performante toepassingen zullen speciale FPGA gebaseerde architecturen nodig zijn die interfacen met de beeldsensor en de mogelijkheden van parallelle hardware ten volle benutten.

Om de ontwikkeling van camera gebaseerde hardware architecturen mogelijk te maken, zijn in deze thesis de methodes om de pose te bepalen bestudeerd. Hieruit is een framework "PoseLab" ontwikkeld om de pose te berekenen via de Levenberg-Marquardt optimalisatie. Dit zal gebruikt worden voor het analyseren van verschillende afwegingen (accuraatheid, bemonsteringsfrequenties, vertragingen, camera resolutie, hardware complexiteit, ...).

Chapter 1

Introduction

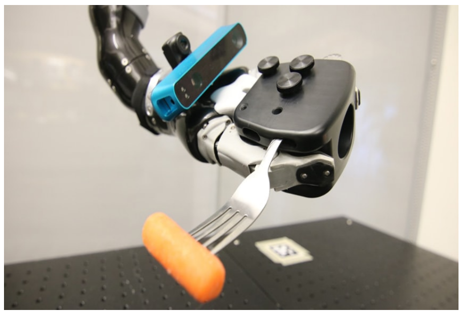
1.1 Camera-based pose estimation applications

This Master's thesis is situated in the research area of pose estimation where the position and orientation of an object with respect to a camera needs to be determined in relation to a predefined world coordinate system. A pose corresponds to the 3D position as well as the 3-axis orientations of an object. It is represented as six degrees of freedom also abbreviated as 6-DoF. Camera pose estimation is used in a wide variety of computer vision applications. It plays an important role in augmented reality [1]. The pose of the camera is utilized to accurately display virtual objects on the image in such a way that they are perceived to be part of the real world. This is illustrated by figure 1.1a. Another important domain is that of robotics. Autonomous navigation relies on knowing the position of the robot in the world. A camera based system can be used for this purpose [2]. Also, knowing the pose of a robot arm enables the creation of intelligent grasping systems [3]. An example can be seen in figure 1.1b. Furthermore, camera pose estimation is employed in the creation of 3D digital models. It is possible to create models solely based on multiple images of an object. Figure 1.1c shows a model of the Colosseum created using only the images made by tourists.

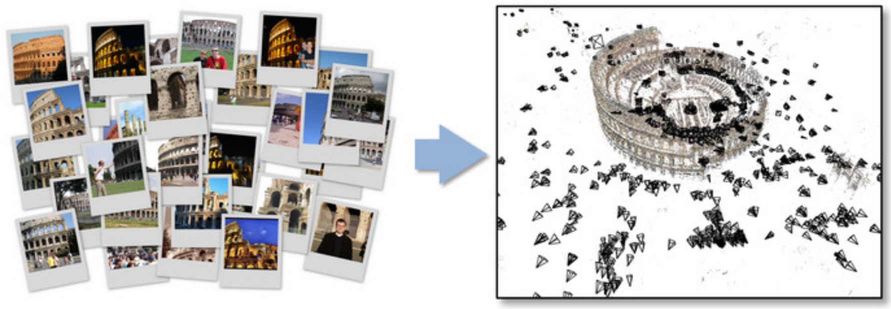
Camera pose estimation is also starting to be used in applications that require high speed and high accuracy. An example of such an application is the tracking of a patient's head inside a magnetic resonance imaging (MRI) scanner, also known as prospective motion correction [7]. The resolution of MRI 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 pose estimation is used to track the movement of the patient in real-time to correct the imaging pulse sequence of the scanner. A marker is placed on the patients head, as illustrated by figure 1.2a. A camera utilizes this marker to calculate the position of the patient. This information is used to adjust the scanner accordingly. Figure 1.2b shows the improvement of the image quality.

Another real-time application in the medical world that needs high accuracy is a haptic feedback system for planning orthognathic surgeries. In orthognatic surgery, corrections are made to the position of the upper and/or lower jaw [9]. Plaster

(a) Augmented reality [4]

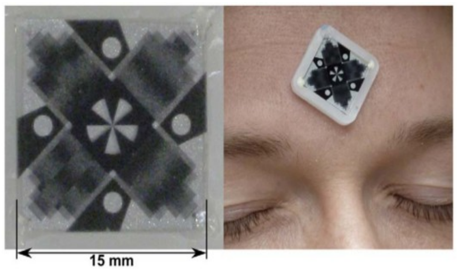


(b) Robotics [5]

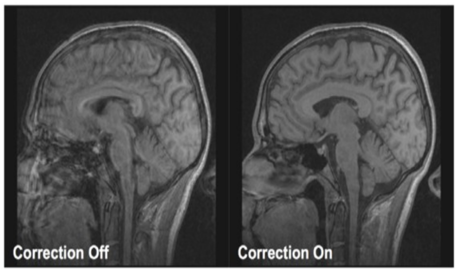


(c) 3D modeling [6]

Figure 1.1: Examples of camera pose estimation applications



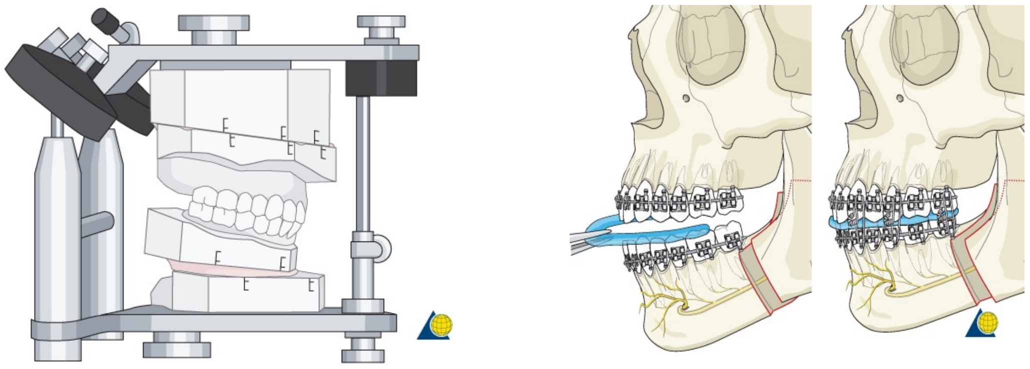
(a) Marker placed on the patient's head [8, p. 2]



(b) Improvement of the image quality [8, p. 4]

Figure 1.2: Prospective motion correction

cast models are used to determine the correct positioning of the jaws. The surgeon moves the models until the optimal occlusion is achieved, as shown by figure 1.3a. The relative position of the jaws are indicated by reference lines on the models. Having found the correct position of the models, intermediate splints are created to be used during the surgery. Figure 1.3b illustrates the use of the splint. Next to plaster casts, the surgeon can also use digital models. This makes the fabrication of the splints easier as 3D printing techniques can be used. However, the surgeon does not perceive any haptic feedback while working with digital models. This makes it difficult to determine the optimal occlusion. A solution to this problem is to use camera-based pose estimation to determine the position of the jaws relative to each other. A camera is connected to one jaw, while a marker is attached to the other. This makes it possible for the surgeon to use the plaster cast models to get haptic feedback, whilst the pose estimation is used to move the digital models accordingly. This way, the surgeon has real-time positioning information of the plaster casts inside his hand. Next to the requirement of high-speed pose estimation, a high accuracy is needed because a slight deviation in the splint can cause discomfort for the patient later on [10]. In [11], a software based proof of concept has been realized, but in order to be practically useful the accuracy, sample rate and latency must be improved by several orders of magnitude.



(a) Determining the optimal occlusion using plaster cast models [12]

(b) Using an intermediate splint to correctly position the jaws [13]

Figure 1.3: Orthognathic surgery planning

1.2 Problem statement

Camera pose estimation is an extensively researched area because of its wide variety of applications. However, the algorithms in use today are computationally expensive. This makes it difficult to achieve high frame rates which are crucial for real-time applications that demand high accuracy. Currently, the propspective motion correction application can only reach 80 frames per second [8]. Having a higher frame rate will result in better MRI images than currently possible. However, computational speed prohibits the realization of this goal. It is possible to use lower resolution images to increase the frame rate. However, this will also reduce the accuracy of the pose estimation. This is not an option for the applications where a high accuracy is equally important as speed. The computational resources need to be enhanced to lower the time required for the processing of a frame. A possibility is to use faster processors or graphical processing units [14]. However, their inherent latency times from camera to action as well as high power consumption make them far from ideal.

Application-specific hardware is needed to effectively achieve higher frame rates and enable low latencies. Advances in field programmable gate arrays (FPGA's) make them suitable for creating complex system-on-chip solutions. They are also reconfigurable, which improves the development time. However, the making of dedicated hardware designs brings its own challenges. Firstly, creating a hardware solution is more difficult than building the equivalent software program. In software, many libraries and built-in functions are available to ease the development. It is not needed to know every detail of the program. This is different in hardware. Here, complete knowledge of the full system is required to correctly implement the digital logic. There are no libraries that make abstractions of certain components of the algorithm. Secondly, it is much more difficult to test and debug a hardware system in comparison to a software program. Software is a sequence of execution statements. A debugger can scan the code line-by-line, showing the states of the variables at that moment. In hardware, many things are running in parallel. This makes it difficult to observe how the logic is behaving.

On the other side, using dedicated hardware also brings new opportunities. The software algorithms of today are designed to calculate the camera pose in a sequential way. Direct hardware implementation in FPGA's has the potential to realize the algorithms exploiting the capabilities of the parallel processing directly in dedicated hardware circuits. This is not an automatic process, but requires the intelligence of human engineers to be able to obtain efficient results. It requires the understanding of the mathematics 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. Dedicated hardware also has the advantage to be able to directly interface to the image sensor, thereby directly exploiting the capabilities provided and possible shortcutting unnecessary delays (such as frame buffers, communication buffers, networking overhead, frame data copying from CPU memory to GPU memories etc...). These delays add up to the latency of the pose estimation, thereby limiting the frequency bandwidth of the control systems based on the pose measurements.

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.

1.3 Objectives

This Master's thesis is part of a larger project. The goal of this project is to create a dedicated hardware solution for the haptic feedback application discussed earlier. The objective is to improve the frame rate, as well as the accuracy. A thousand frames per second with an accuracy of twenty micrometers at two centimeters distance is the ultimate target. This Master's thesis is an essential phase in realizing this objective. The aim of the research done here is to gain a detailed understanding, which is required for a hardware implementation, of the state of the art camera pose estimation methods. More in particular, the use of the Levenberg-Marquardt optimization algorithm is studied. In addition, a framework is realized in Matlab that allows to experiment with the available design trade-offs. This is absolutely necessary when evaluating hardware architectural alternatives.

The first objective of this Master's thesis is to create a software implementation of the Levenberg-Marquardt optimization algorithm to determine the pose of a camera based on 2D-3D point correspondences. Normally, This method is applied as a black box function provided by software libraries such as OpenCV [15]. The goal here is to open up the black box and compute the camera pose without using calls to external libraries. This ensures that all the calculations required to obtain the camera pose are fully known. The choice of starting with a software implementation is made because it is much easier for debugging and experimentation.

The software implementation will be used as a framework to create an algorithm that is designed to run on dedicated hardware and optimized for a specific application. Here, the first steps will be taken to adapt the Levenberg-Marquardt based pose estimation to take advantage of the hardware interface to the camera. This leads to the second goal of this Master's thesis, to enable Levenberg-Marquardt to take in new 2D-3D point correspondences as the camera is reading out the individual rows of the frame.

1.4 Materials and Methods

The evaluation framework made in this thesis has been given the name PoseLab. For the implementation of PoseLab, MATLAB® is used. It has been chosen because camera pose estimation is a mathematical problem and uses matrices extensively. MATLAB makes it easy to work with matrices and other mathematical tools. It is a very high-level programming language, which speeds up the design process. There is also an open source camera calibration toolbox, developed at Caltech, available for MATLAB [16]. This is used for determining the intrinsic parameters of the camera, a necessary step prior to camera pose estimation. MATLAB is not the fastest software and would not be ideal for real-time software applications. However, this does not matter for this thesis, as dedicated hardware architectures are the target implementation platform. But before being able to do that, it is necessary to very well model, evaluate and experiment all architectural decisions by accurate mathematical models. MATLAB is a suitable environment for this evaluation.

A test setup is used to verify the correctness of the implementation of the Levenberg-Marquardt based camera pose estimation method. The test setup has been realized by Aerts Thomas [17]. It uses the stepper motor of a flatbed scanner to accurately move a 8x11 checkerboard pattern. The size of the squares is 12x12 mm. The AdaFruit Arduino Motorshield v1 drives the stepper motor [18]. A Java application interfaces with the test bench in order to move the pattern to a desired position. The camera of the Samsung Galaxy S8 is used to take the pictures. It has an image sensor resolution of 12 megapixels and outputs images of size 4032x3024. The camera operates in pro mode to enable manual focus. This is done to ensure that the focal length is the same for every image. The shutter speed and ISO of the camera have been set to 1/45 s and 125 respectively.

1.5 Outline

1.5.1 Introduction

An overview is given on the role of camera pose estimation in computer vision. Two applications that require high-speed, high-accuracy solutions are discussed, namely prospective motion correction and orthognathic surgery planning. Then, the shortcoming of speed of the current software algorithms is discussed and how dedicated hardware is needed to overcome this problem. Hereafter, the objective of this Master's thesis is explained. Finally, the materials that have been used are given.

1.5.2 Theory

This chapter discusses the theory needed to understand the research that has been done in this Master's thesis. It starts by explaining the pinhole camera which is used to model the projection of a point in the real world onto the image sensor of the camera. After that, Zhang's calibration method [19] is discussed which determines the intrinsic parameters of the pinhole model. These are distinct for each camera. Then, it is described how the position and orientation of the camera in the world can be recovered. Finally, the Levenberg-Marquardt optimization method is explained in detail.

1.5.3 Camera pose estimation algorithm implementation

This chapter describes the implementation of the camera pose estimation algorithm. First, it is explained how the intrinsic parameters are obtained for the camera used in this thesis. Then, the functions used inside the algorithm are explained in detail. The functions perform the following tasks: axis-angle transformation, reverse lens distortion, calculate the Jacobian, calculate the error vector and determine the step. Finally, the flow of the camera pose estimation algorithm is illustrated with a flowchart.

1.5.4 Experiments

A total of four experiments have been completed. First, the setup to conduct the experiments is described. Secondly, the procedure is explained. After this, the results are given and discussed. Next, a comparison is made between PoseLab and the Caltech toolbox. Finally, The behavior of Levenberg-Marquardt when new point correspondences are added is examined.

1.5.5 Conclusion

In this chapter, an overview is given of the work done in this thesis. To end, the work that needs to be done in the future is discussed.

Chapter 2

Theory

2.1 Pinhole camera model

The pinhole camera is a model which describes how a 3D point in the real world is projected to a 2D point in the image [20], [21]. Figure 2.1 illustrates the setup. In this model, the camera has an infinitesimally small hole where it takes in the light. This point is referred to as the optical center. A light ray coming from a distant object passes the optical center and is captured on the image plane where the image sensor is located. The assumption is made that only one light ray from a particular point in the world enters the camera. This process creates an image where the object in front of the camera is upside down.

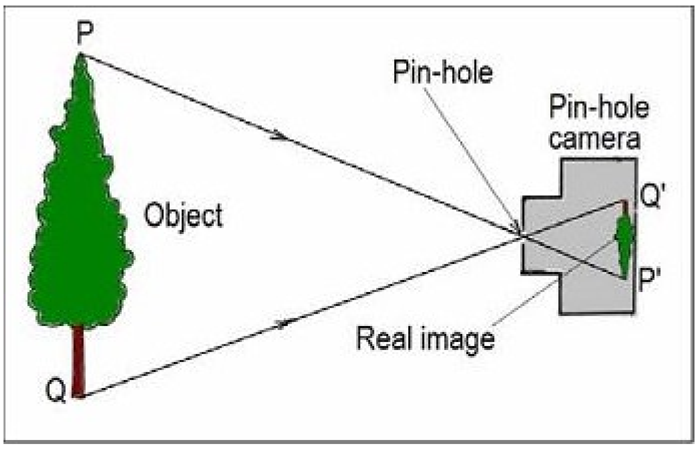


Figure 2.1: The pinhole camera model [22]

The mathematical formulation of the projection can be easily derived. Figure 2.2 shows how a 3D world point is projected onto the image plane. By putting the image plane in front of the optical center, the image is inverted in the vertical direction and brings it the right-side up. The camera coordinate system is defined at the optical center where one axis, the z-axis in this case, is in the viewing direction of the camera. This is called the optical axis. The point where the image plane intersects the optical axis is known as the principal point. The distance between the principal point and the optical center is defined as the focal length f. A point in the real world is described by its coordinates (X_c, Y_c, Z_c) expressed in the camera coordinate system. The corresponding point on the image plane has coordinates (x_{image}, y_{image}), also expressed in the camera coordinate system. Equations

(2.1) and (2.2) can be easily derived from similar triangle geometry.

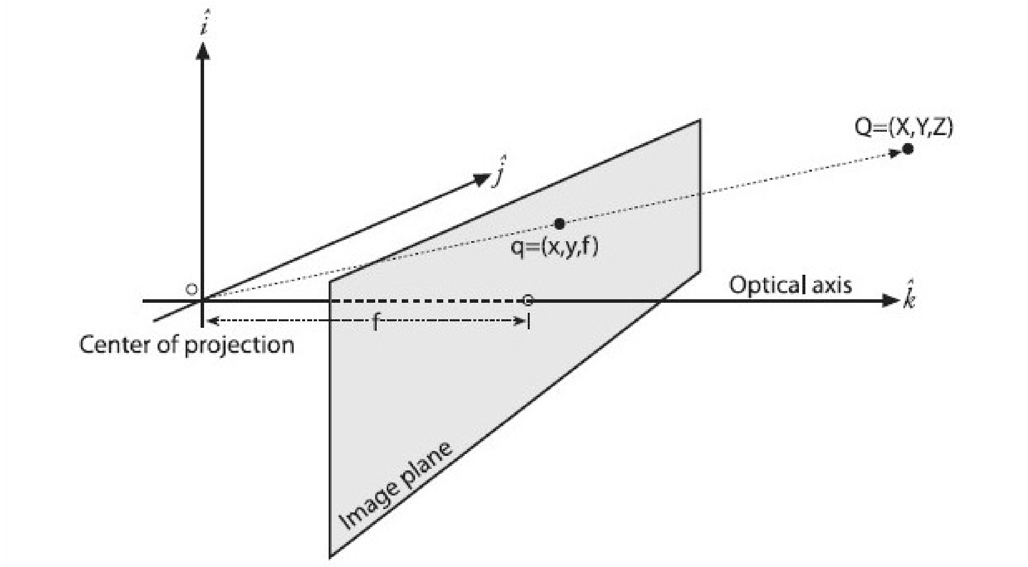


Figure 2.2: Projection of a 3D world point onto the image plane [20, p. 372]

$$x_{image} = f\frac{X_c}{Z_c} \tag{2.1}$$

$$y_{image} = f\frac{Y_c}{Z_c} \tag{2.2}$$

Note that the depth information is lost. This is because the collection of 3D points that make up a line that intersects the optical center are projected onto the same point in the image plane. This leads to ambiguities and implies that if the x_{image} and y_{image} coordinates of a single image point are known, the 3D coordinate in the world cannot be uniquely found. Only the direction of the light ray can be recovered and the information of depth is lost.

However, this model simplifies a real camera too much. Extensions on this basic model are needed to achieve decent results. The first extension corrects for the fact that the principal point is not in the center of the image sensor as assumed in equations (2.1) and (2.2). This is caused by imperfections during the production process. Two new parameters c_x and c_y are added to take this into account. These are the coordinates of the principle point in the image sensor coordinate system which has the origin in the upper left corner of the image. Shifting the (x_{image}, y_{image}) coordinate by (c_x, c_y) respectively, pixel coordinates (x, y) are obtained. Another adjustment is needed to equations (2.1) and (2.2) to allow rectangular pixels in addition to square pixels. Rectangular pixels are often used in low-end cameras. Two different focal lengths f_x and f_y, in pixels, are needed to make this possible. These values are the product of the physical focal length, in mm, and the size of the imager elements in the x or y direction, in pixels per mm. Taking the corrections described here into account, the equations for the x and y

coordinate become:

$$x = f_x \frac{X_c}{Z_c} + c_x \tag{2.3}$$

$$y = f_y \frac{Y_c}{Z_c} + c_y \tag{2.4}$$

The perspective projection described by equations (2.3) and (2.4) can be reduced to a matrix equation. The convention for projective transformations is to make equations more elegant by using homogeneous coordinates instead of Cartesian coordinates. By using homogeneous coordinates points at infinity can be represented by finite coordinate values. These homogeneous coordinates have an extra dimension and the property that two points are equivalent when their coordinate values are proportional. A possible homogeneous coordinate corresponding to a specific Cartesian coordinate can be found by simply appending a one as an extra coordinate. This is called a normalized homogeneous coordinate. Using homogeneous coordinates, the perspective projection can then be described by equation (2.5). The Cartesian pixel coordinates, x and y, can be recovered by using the property that two points are equivalent if one is a multiple of the other. Dividing u and v by w, gives the normalized homogeneous coordinate $[x \quad y \quad 1]^T$.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.5}$$

Another possible extension to the model is to let the pixels have a parallelogram shape. A new parameter s, the skew coefficient, is introduced and represents the angle between the x- and y-axis of the pixel. The model is then given by equation (2.6). This equation also introduces the scaling factor τ which results from the equivalence between homogeneous points whose coordinate values are proportional.

$$\tau \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.6}$$

Up to now, the 3D points in the world are given with respect to the camera coordinate system. In practice, 3D points are often given in a predefined coordinate system. A point in this world coordinate system needs to be expressed in the camera coordinate system in order to make use of equation (2.6). This can be done by applying the appropriate translation and rotation as shown by figure 2.3. In Cartesian coordinates, the relationship is described by equation (2.7). Here, $\boldsymbol{C}_w \in \mathbb{R}^{3\times1}$ is the position of the optical center expressed in the world coordinate system, $\boldsymbol{x}_w \in \mathbb{R}^{3\times1}$ is a 3D point expressed in the world coordinate system, $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ is a rotation matrix and $\boldsymbol{X}_c \in \mathbb{R}^{3\times1}$ is the same 3D point expressed in the camera coordinate system. For homogeneous coordinates, the matrix equation is given by (2.8) [23]. However, (2.7) can be expanded to $\boldsymbol{X}_c = \boldsymbol{R}\boldsymbol{X}_w - \boldsymbol{R}\boldsymbol{C}_w$. In most literature, the term $-\boldsymbol{R}\boldsymbol{C}_w \in \mathbb{R}^{3\times1}$ is renamed to the translation vector $\boldsymbol{t}$.

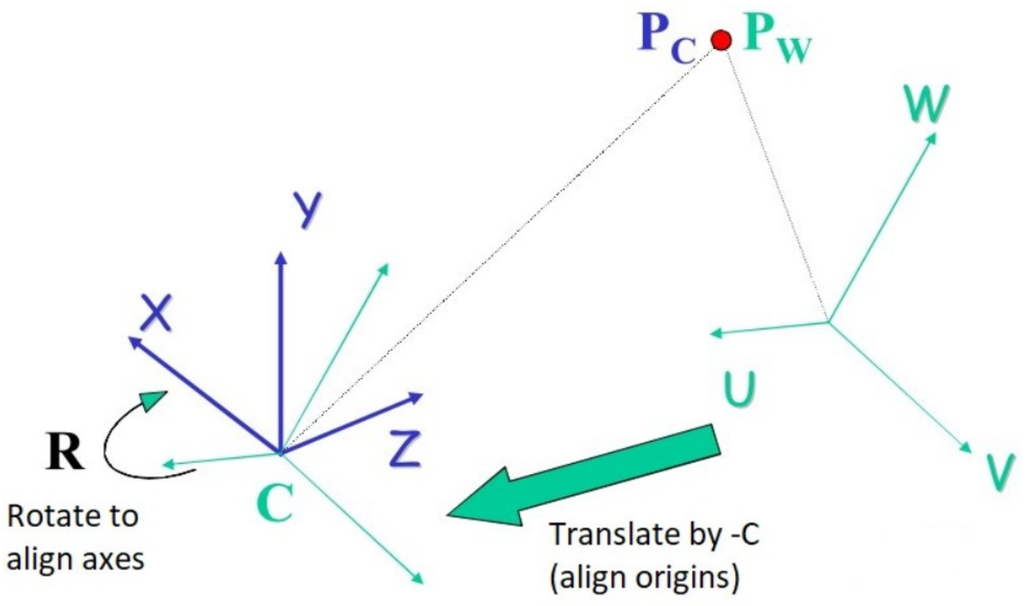


Figure 2.3: Relationship between camera coordinate system and world coordinate system [23, p. 18]

The final equation to express a 3D point, defined in the world coordinate system, in the camera coordinate system is given by (2.9).

$$\boldsymbol{X}_c = \boldsymbol{R}(\boldsymbol{X}_w - \boldsymbol{C}_w) \tag{2.7}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 & 0 \\ r21 & r22 & r23 & 0 \\ r31 & r32 & r33 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.8}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.9}$$

Combining equations (2.6) and (2.9) into (2.10), a 3D point in the world can be mapped to the 2D image point of the camera.

$$\tau \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.10}$$

Note that the fourth column of the first matrix contains only zeros and thus will not contribute to the result of the matrix multiplication. This means that the fourth column of the first matrix and the fourth row of the second matrix can be removed. This leads to the final matrix equation of the perspective projection given by (2.11). Matrix $\boldsymbol{K}$ contains the intrinsic parameters that are specific to the camera and is referred to as the calibration matrix. Matrix $\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix}$ contains the extrinsic parameters which represent the position and orientation of the camera with respect to the world coordinate system.

$$\tau \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \boldsymbol{X} \tag{2.11}$$

As mentioned earlier, the pinhole camera has an infinitesimally small point where it takes in the light. This way, only a small amount of light can enter the camera which results in the need of a very long exposure time. Real cameras make use of lenses to catch more light. In theory, a perfect lens exists which will not introduce distortions. However, difficulties in manufacturing prohibits the perfect lens from being made. Imperfect lenses introduce lens distortions in the image. The two most important categories of lens distortions are the radial and tangential distortion.

Figure 2.4 illustrates radial distortion. It has the effect that pixels are bent more the further away they are from the center of the image. This is the result of light rays at the edges of the lens being bent more than light rays at the optical center. Radial distortion can be modeled by equations (2.12) and (2.13). Here, (x, y) is the pixel coordinate without radial distortion, r is the distance of this coordinate to the principal point, k_{1-3} are the radial distortion coefficients and (x_{dist}, y_{dist}) is the pixel coordinate as captured by the camera with radial distortion.

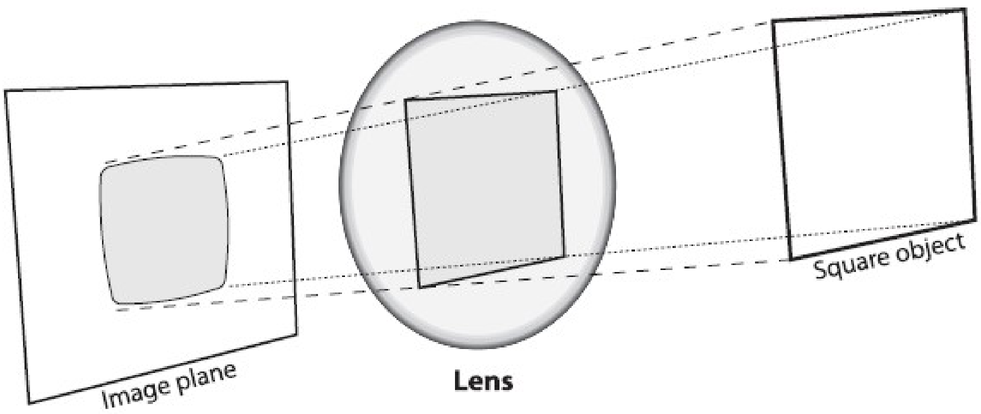


Figure 2.4: Radial distortion [20, p. 376]

$$x_{dist} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.12}$$

$$y_{dist} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.13}$$

Figure 2.5 illustrates tangential distortion. This is caused by the image plane not being completely parallel to the lens. Imperfections in the manufacturing process are responsible for this. One of the reasons can be the use of cheap glue. Tangential distortion can be modeled by equations (2.14) and (2.15). There are two tangential distortion coefficients p_1 and p_2.

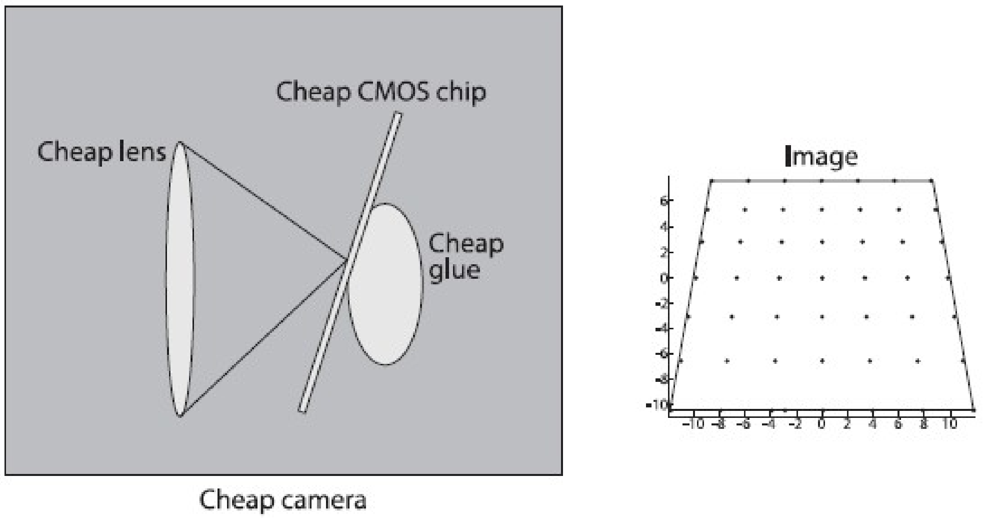


Figure 2.5: Tangential distortion [20, p. 377]

$$x_{dist} = x + [2p_1 y + p_2(r^2 + 2x^2)] \tag{2.14}$$

$$y_{dist} = y + [p_1(r^2 + 2y^2) + 2p_2 x] \tag{2.15}$$

Combining radial and tangential distortion, there are a total of five distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$. There are other types of distortions but these do not have a significant effect compared to radial and tangential distortions.

2.2 Camera calibration

As stated earlier, the calibration matrix contains parameters which are specific to each camera. Estimating these parameters for a particular camera is known as camera calibration [24]. The most widely used method today has been proposed by Zhang in 1998 [19]. This method makes use of a 2D pattern to generate known 3D points in the world. This pattern is often a checkerboard because the corner points are easily extracted [25]. The world coordinate system is on the pattern itself. The Z-axis is defined perpendicular to the pattern. By doing this, the 3D coordinates of every point on the pattern are known. It also results in the Z-coordinate being 0 for every point as the pattern lays in a two dimensional plane. This will simplify equation (2.11) by eliminating the Z-coordinate as shown by (2.16). Here, $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the first and second column of the 3×3 rotation matrix respectively.

$$\tau \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{2.16}$$

Equation (2.16) can be seen as a linear transformation of a 2D point to another 2D point. This linear transformation is a homography $\boldsymbol{H}$ which in this case is described by $\boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix}$. The next step is then to take multiple images of the pattern at different positions and angles, and estimate the homography $\boldsymbol{H}$ between the pattern and the image. To get an estimate for $\boldsymbol{H}$, a system of linear equations needs to be solved. The mathematical details are described in [19]. The matrix $\boldsymbol{H}$ has 8 degrees of freedom (not 9 because homogeneous coordinates are used). Because every observed point gives 2 equations, one for x and one for y, at least 4 points are needed to estimate the homography.

The calibration matrix $\boldsymbol{K}$ can be extracted from the homography $\boldsymbol{H} = \begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3\end{bmatrix} = \boldsymbol{K}\begin{bmatrix}\boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t}\end{bmatrix}$. This is done by exploiting the fact that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are orthonormal ($\boldsymbol{r}_1^T\boldsymbol{r}_2 = 0$ and $||\boldsymbol{r}_1|| = ||\boldsymbol{r}_2|| = 1$) which leads to the following two constraints given by equations (2.17) and (2.18).

$$\boldsymbol{h}_1^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_2 = 0 \tag{2.17}$$

$$\boldsymbol{h}_1^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_1 = \boldsymbol{h}_2^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_2 \tag{2.18}$$

Now define a matrix $\boldsymbol{B} = \boldsymbol{K}^{-T}\boldsymbol{K}^{-1}$ which is symmetric and positive definite. This matrix $\boldsymbol{B}$ can be found by solving a system of linear equations $\boldsymbol{V}\boldsymbol{b} = 0$ where $\boldsymbol{b} = [b_{11}, b_{12}, b_{13}, b_{22}, b_{23}, b_{33}]^T$. Matrix $\boldsymbol{V}$ is constructed using the the fact that $\boldsymbol{h}_i^T\boldsymbol{B}\boldsymbol{h}_j$ can be written as $\boldsymbol{v}_{ij}^T\boldsymbol{b}$. Here, $\boldsymbol{v}_{ij}$ is defined as $\begin{bmatrix}h_{i1}h_{j1} & h_{i1}h_{j2}+h_{i2}h_{j1} & h_{i3}h_{j1}+h_{i1}h_{j3} & h_{i3}h_{j2}+h_{i2} & h_{j3} & h_{i3}h_{j3}\end{bmatrix}$. The constraints described earlier can then be formulated as: $\boldsymbol{v}_{12}^T\boldsymbol{b} = 0$ and $\boldsymbol{v}_{11}^T b - \boldsymbol{v}_{22}^T b = 0$. For one image, the matrix $\boldsymbol{V}$ is described by $\begin{bmatrix}\boldsymbol{v}_{12}^T \\ \boldsymbol{v}_{11}^T - \boldsymbol{v}_{22}^T\end{bmatrix}$. Using only one image, the matrix $\boldsymbol{V}$ is not large enough to get a solution for $\boldsymbol{b}$. Remember that multiple pictures are taken and that for each image a homography is estimated. The matrix $\boldsymbol{V}$ is then found by stacking the results for every image in the vertical direction. Because $\boldsymbol{b}$ has 6 degrees of freedom, 3 different images are needed to obtain a solution.

Once $\boldsymbol{b}$, and thus $\boldsymbol{B}$, is found by solving the system of linear equations described above, the calibration matrix $\boldsymbol{K}$ can be extracted by performing a Cholesky decomposition.

$$chol(\boldsymbol{B}) = \boldsymbol{A}\boldsymbol{A}^T \tag{2.19}$$

$$\boldsymbol{A} = \boldsymbol{K}^{-T} \tag{2.20}$$

In addition to the intrinsic parameters, this method can also recover the extrinsic parameters for every image once $\boldsymbol{K}$ is known. They are determined using equations (2.21) to (2.24) where $\mu = 1/||\boldsymbol{K}^{-1}\boldsymbol{h}_1||$.

$$\boldsymbol{r}_1 = \mu\boldsymbol{K}^{-1}\boldsymbol{h}_1 \tag{2.21}$$

$$\boldsymbol{r}_2 = \mu\boldsymbol{K}^{-1}\boldsymbol{h}_2 \tag{2.22}$$

$$\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2 \tag{2.23}$$

$$\boldsymbol{t} = \mu\boldsymbol{K}^{-1}\boldsymbol{h}_3 \tag{2.24}$$

However, the solution for the intrinsic and extrinsic parameters can still be refined because the algebraic distance that is minimized has no physical meaning. This can be done by finding the maximum likelihood estimate. It also makes it possible to determine the distortion coefficients. The maximum likelihood estimate can be found by minimizing equation (2.25). This equation represents the squared reprojection error of the m points on the pattern across all n images. Here, $\hat{\boldsymbol{x}}$ is the reprojection of the jth point on the pattern, $\boldsymbol{X}_j$, in image i. The vector $\boldsymbol{x}_{ij}$ is holding the pixel coordinates of the jth point in image i. The vector $\boldsymbol{q}$ contains the distortion coefficients. This non-linear minimization problem can be solved by using the Levenberg-Marquardt optimization algorithm which will be discussed in section 2.4.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} ||\boldsymbol{x}_{ij} - \hat{\boldsymbol{x}}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}_i, \boldsymbol{t}_i, \boldsymbol{X}_j)||^2 \tag{2.25}$$

2.3 6 degrees of freedom camera pose estimation

The pose of the camera is described by the extrinsic parameters as stated earlier. The rotation matrix $\boldsymbol{R}$ represents the orientation of the camera with respect to the world coordinate system. The matrix $\boldsymbol{R}$ has dimension 3×3 and thus has nine parameters that need to be determined. However, a rotation matrix has only three degrees of freedom. These are the angles of rotation around the X, Y and Z axis which are called roll (θ), pitch (ϕ) and yaw (γ) respectively. The rotation matrices that describe the rotation around each of the three axes are given by equations (2.26) to (2.28). The rotations around every axis need to be combined to get the resulting rotation matrix. The values of the roll, pitch and yaw angles needed to obtain the correct rotation matrix depend on the sequence in which the rotations around the individual axes are applied. The convention is to first rotate around the X-axis, then the Y-axis and finally the Z-axis. Equation (2.29) describes the calculation of the rotation matrix $\boldsymbol{R}$ using roll, pitch and yaw angles in the ZYX-sequence.

$$\boldsymbol{R}_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.26}$$

$$\boldsymbol{R}_Y = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix} \tag{2.27}$$

$$\boldsymbol{R}_Z = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.28}$$

$$\boldsymbol{R} = \boldsymbol{R}_Z \boldsymbol{R}_Y \boldsymbol{R}_X \tag{2.29}$$

Because the rotation matrix only has three degrees of freedom, it is possible to parameterize the orientation using three parameters. This way, only three parameters need to be determined instead of nine. The axis-angle notation [26], shown in figure 2.6, is utilized for this purpose. It uses a 3×1 vector $\boldsymbol{w} = [w_x, w_y, w_z]^T$

to indicate the axis of rotation. The magnitude $\|\boldsymbol{w}\|$ specifies the angle of rotation around $\boldsymbol{w}$. It can be shown through a Taylor series expansion that $e^{\lfloor \boldsymbol{w} \rfloor_x}$ represents a three dimensional rotation matrix. Here, $\lfloor \boldsymbol{w} \rfloor_x$ is the skew-symmetric matrix of the vector $\boldsymbol{w}$ as given by equation (2.30). The relationship between a vector $\boldsymbol{w}$ and its corresponding rotation matrix can be derived from the Taylor series expansion and is described by (2.31). 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 (2.31) simplifies to (2.32). This is known as the Rodrigues transformation.

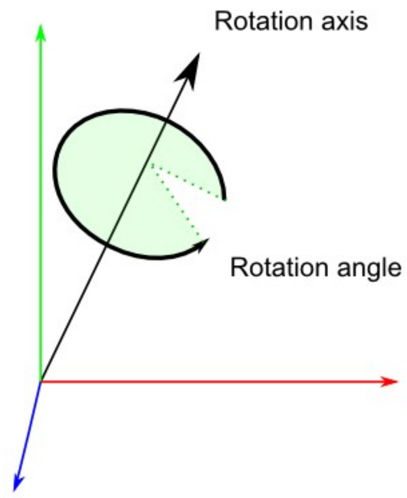


Figure 2.6: Axis-angle representation [27]

$$\lfloor \boldsymbol{w} \rfloor_x = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \tag{2.30}$$

$$e^{\lfloor \boldsymbol{w} \rfloor_x} = \boldsymbol{R} = \boldsymbol{I} + \frac{sin(\|\boldsymbol{w}\|)}{\|\boldsymbol{w}\|}\lfloor \boldsymbol{w} \rfloor_x + \frac{1 - cos(\|\boldsymbol{w}\|)}{\|\boldsymbol{w}\|^2}\lfloor \boldsymbol{w} \rfloor_x^2 \tag{2.31}$$

$$\boldsymbol{R} = \boldsymbol{I} + sin(\beta)\lfloor \hat{\boldsymbol{w}} \rfloor_x + (1 - cos(\beta))\lfloor \hat{\boldsymbol{w}} \rfloor_x^2 \tag{2.32}$$

As mentioned earlier, the vector $\boldsymbol{t}$ represents the translation of the camera with respect to the world coordinate system. It also has three degrees of freedom. Recall that $\boldsymbol{t}$ implicitly gives the position $\boldsymbol{C}_w$ of the camera through the relation $\boldsymbol{t} = -\boldsymbol{R}\boldsymbol{C}_w$. To recover the position of the camera, equation (2.33) is used.

$$\boldsymbol{C}_w = -\boldsymbol{R}^{-1}\boldsymbol{t} \tag{2.33}$$

The pose of a camera has six degrees of freedom. Three for rotation and three for translation. Thus, a total of six parameters need to be determined. The process of finding these is called camera pose estimation. The most accurate and widely used methods are based on the perspective-n-point (PnP) problem. Here, it is assumed that the intrinsic parameters and distortion coefficients of the camera are known. Given n known 3D world points and the corresponding 2D image points, equation

(2.11) can be used to solve for the extrinsic parameters. At least three point correspondences are needed to find a solution for the camera pose. However, having more point correspondences will always lead to a better accuracy.

There exists a considerable amount of algorithms which can be used to solve the PnP problem. The first set of algorithms are non-iterative. In this category, the most important algorithms are: P3P [28], efficient PnP (EPnP) [29] and direct least squares (DLS) [30]. The second category consists of iterative algorithms. These require more computing time but will give more accurate results. They minimize the total reprojection error as given by equation (2.25) but only for one image. The Levenberg-Marquardt optimization algorithm leads to the most accurate results.

2.4 Levenberg-Marquardt optimization

The Levenberg-Marquardt algorithm is a non-linear optimization method [31], [32], [33]. It is used to find the parameters of a model so that it matches the experimental data. This is done by finding the minimum of a cost function. The cost function represents how well the model agrees with the experimental data. A large value for the cost function means that the data predicted by the model using the current parameters is far off the observed data. The measure of how much the model disagrees with the data is often taken to be the summation of the squared error of every data point.

In the context of camera calibration and pose estimation, the error $\boldsymbol{d}_j \in \mathbb{R}^{2\times 1}$ of the jth data point is defined as the difference between the 2D image point and the reprojection of the corresponding 3D world point using the model described by equation (2.11). Notice that each point correspondence gives rise to two error terms. One for the x-value and one for the y-value. This is shown by equations (2.34) and (2.35). Define a vector-valued function $\boldsymbol{f}(\boldsymbol{\Theta}) : \mathbb{R}^p \rightarrow \mathbb{R}^{2m}$ for m data points that takes as input the $p \times 1$ vector $\boldsymbol{\Theta}$ which contains the p parameters to be determined, and outputs the $2m \times 1$ error vector $[d_{1x}, d_{1y}, ..., d_{mx}, d_{my}]^T$. When estimating the pose of the camera, vector $\boldsymbol{\Theta}$ contains the six parameters that describe the orientation and position as discussed in section 2.3. The resulting sum of squared errors cost function is given by equation (2.36). The factor $\frac{1}{2}$ is there to simplify the derivation of the derivatives. Note that equation (2.36) is the same as (2.25), but this time only for one image.

$$d_{jx} = x_{jx} - \hat{x}_{jx}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{X}_j) \tag{2.34}$$

$$d_{jy} = x_{jy} - \hat{x}_{jy}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{X}_j) \tag{2.35}$$

$$F(\boldsymbol{\Theta}) = \frac{1}{2}||\boldsymbol{f}(\boldsymbol{\Theta})||^2 = \frac{1}{2}\boldsymbol{f}(\boldsymbol{\Theta})^T\boldsymbol{f}(\boldsymbol{\Theta}) \tag{2.36}$$

The minimum of cost function $F(\boldsymbol{\Theta})$ is found in an iterative manner. The parameters in $\boldsymbol{\Theta}$ are set to an initial value $\boldsymbol{\Theta}_0$. Then, step $\boldsymbol{h}_1$ is calculated. This vector contains the correction for each parameter so that $F(\boldsymbol{\Theta}_0 + \boldsymbol{h}_1) < F(\boldsymbol{\Theta}_0)$. After this, $\boldsymbol{\Theta}_1$ is appointed the value $\boldsymbol{\Theta}_0 + \boldsymbol{h}_1$. Again, a step $\boldsymbol{h}_2$ is calculated so that $F(\boldsymbol{\Theta}_1 + \boldsymbol{h}_2) < F(\boldsymbol{\Theta}_1)$. This iterative process of taking steps towards lower values for cost function $F(\boldsymbol{\Theta})$ is carried out until $F(\boldsymbol{\Theta})$ reaches its minimum.

To find step $\boldsymbol{h}$, the cost function $F(\boldsymbol{\Theta})$ is approximated by a Taylor series expansion. This is given by equation (2.37). The vector $\boldsymbol{g} = \left[\frac{\partial F}{\partial \Theta_1}, ..., \frac{\partial F}{\partial \Theta_p}\right]^T$ 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 $i,j \in [1..p]$. It contains the second-order partial derivatives. The goal is to find the step $\boldsymbol{h}$ which minimizes the approximation. This can be done by differentiating equation (2.37) with respect to $\boldsymbol{h}$ and set it to zero which leads to equation (2.38).

$$F(\boldsymbol{\Theta} + \boldsymbol{h}) = F(\boldsymbol{\Theta}) + \boldsymbol{g}^T\boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{H}\boldsymbol{h} \tag{2.37}$$

$$\boldsymbol{H}\boldsymbol{h} = -\boldsymbol{g} \tag{2.38}$$

The next step is to determine the gradient $\boldsymbol{g}$ and Hessian $\boldsymbol{H}$. It can be shown that $\boldsymbol{g}$ is equal to $\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta})$. The $2m \times p$ matrix $\boldsymbol{J}$ is called the Jacobian matrix. It contains the first-order partial derivatives of all the entries in $\boldsymbol{f}(\boldsymbol{\Theta})$ with respect to each parameter in $\boldsymbol{\Theta}$. In the case of camera pose estimation, $\boldsymbol{J}$ is given by equation (2.39). It can be proven that the Hessian $\boldsymbol{H}$ is equal to $\boldsymbol{J}^T\boldsymbol{J}+\boldsymbol{f}''(\boldsymbol{\Theta})^T\boldsymbol{f}(\boldsymbol{\Theta})$. 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 (2.38) leads to equation (2.40), known as the matrix notation of the normal equations, which needs to be solved to obtain step $\boldsymbol{h}$.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial d_{1x}}{\partial \Theta_1} & \dots & \frac{\partial d_{1x}}{\partial \Theta_6} \\ \frac{\partial d_{1y}}{\partial \Theta_1} & \dots & \frac{\partial d_{1y}}{\partial \Theta_6} \\ \vdots & & \vdots \\ \frac{\partial d_{mx}}{\partial \Theta_1} & \dots & \frac{\partial d_{mx}}{\partial \Theta_6} \\ \frac{\partial d_{my}}{\partial \Theta_1} & \dots & \frac{\partial d_{my}}{\partial \Theta_6} \end{bmatrix} \tag{2.39}$$

$$(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta}) \tag{2.40}$$

Using (2.40) to calculate step $\boldsymbol{h}$ is known as the Gauss-Newton method. Levenberg and Marquardt made an extension to this method by introducing a damping factor $\lambda > 0$ as shown by (2.41). The influence of the damping factor will be discussed by looking at the case where λ is very small, and when λ is very high. When λ is small, equation (2.41) reduces to (2.40) 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, (2.41) 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 of the gradient which represents the direction of the fastest reduction in the cost function. The size of the step is controlled by factor $\frac{1}{\lambda}$. The higher the value of λ, the smaller the step. This approach of trying to reach the minimum 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. Damping factor λ thus controls the size and the direction of step $\boldsymbol{h}$.

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta}) \tag{2.41}$$

As stated earlier, equation (2.40) is the matrix notation of the normal equations. In general, the normal equations $(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{x}^* = \boldsymbol{A}^T\boldsymbol{b}$ give the analytical solution to the linear least squares problem $\boldsymbol{A}\boldsymbol{x}^* \approx \boldsymbol{b}$. It finds the parameters $\boldsymbol{x}^*$ that minimize the squared error defined as $||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2$. Comparing equation (2.40) to the general case, it can be seen that it represents the normal equations which form the solution for the linear least squares problem of $\boldsymbol{f}(\boldsymbol{\Theta}) + \boldsymbol{J}\boldsymbol{h} \approx \boldsymbol{0}$. Similarly, equation (2.41) represents the normal equations of the linear least squares problem $\begin{bmatrix}\boldsymbol{f}(\boldsymbol{\Theta})\\ \boldsymbol{0}\end{bmatrix} + \begin{bmatrix}\boldsymbol{J}\\ \sqrt{\lambda}\boldsymbol{I}\end{bmatrix}\boldsymbol{h} \approx \boldsymbol{0}$. However, normal equations are not the only solutions to linear least squares problems. They can also be solved by orthogonal transformation. The first step is to determine the QR decomposition of the matrix $\boldsymbol{A}$ 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. It can be shown that the solution is found by solving $\boldsymbol{R}_1\boldsymbol{x}^* = \boldsymbol{Q}_1^T\boldsymbol{b}$ using backward substitution [34]. If matrix $\boldsymbol{A}$ is of size $m \times n$ and $m > n$, then the matrix $\boldsymbol{Q}_1$ contains the first n columns of $\boldsymbol{Q}$. This method is more accurate than the normal equations.

The damping factor λ needs to be updated at every iteration to calculate the optimal step $\boldsymbol{h}$. This update is done based on the gain ratio ϱ defined by equation (2.42). It is the actual decrease of the cost function F with respect to the decrease predicted by the linear model L. Recall that $\boldsymbol{f}(\boldsymbol{\Theta})$ is assumed to be linear in order to approximate the Hessian $\boldsymbol{H}$ as $\boldsymbol{J}^T\boldsymbol{J}$. It can be shown that the denominator can be written out as $\frac{1}{2}\boldsymbol{h}^T(\lambda\boldsymbol{h} - \boldsymbol{g})$. It can also be proven that this is always positive. A small value for ϱ means that $L(\boldsymbol{h})$ is a bad approximation of $F(\boldsymbol{\Theta} + \boldsymbol{h})$ and thus the damping factor λ needs to be increased to take smaller steps. For a large value of ϱ, λ may be decreased to increase the size of the steps because $L(\boldsymbol{h})$ is a good approximation. The method used to calculate the new value for λ is as follows. First, an extra factor υ is introduced. If the gain ratio ϱ is larger than zero, the cost function decreases by applying step $\boldsymbol{h}$ and λ is multiplied by $max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$ while υ is set equal to two. 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 υ. The factor υ itself is doubled and a new iteration is started in an attempt to find a better step $\boldsymbol{h}$. When multiple iterations do not find a decrease in cost function, λ becomes large and the steps will be small in order to eventually seek a decrease in cost function.

$$\varrho = \frac{F(\boldsymbol{\Theta}) - F(\boldsymbol{\Theta} + \boldsymbol{h})}{L(\boldsymbol{0}) - L(\boldsymbol{h})} \tag{2.42}$$

The algorithm needs to be stopped when the minimum is reached. In this point, the gradient $\boldsymbol{g}$ of cost function $F(\boldsymbol{\Theta})$ should be equal to zero. A viable stopping criterion is then to determine if the supremum norm $\|\boldsymbol{g}\|_\infty$ is smaller than a chosen value ε_1. A second way to stop the algorithm is when the step $\boldsymbol{h}$ becomes too small or $\|\boldsymbol{h}\| < \varepsilon_2(\|\boldsymbol{\Theta}\| + \varepsilon_2)$. This formulation also takes the size of the parameters into account. When $\boldsymbol{\Theta}$ is large, the algorithm stops when $\boldsymbol{h}$ is too small relative to $\boldsymbol{\Theta}$ with a factor of ε_2. When $\boldsymbol{\Theta}$ is small, step $\boldsymbol{h}$ is considered too small if it is less than ε_2^2. Lastly, the algorithm needs to be stopped if it reaches a certain number of iterations in order to prevent an infinite loop.

Chapter 3

Camera pose estimation algorithm implementation

3.1 Obtaining intrinsic parameters

Before the pose can be determined, the intrinsic parameters and distortion coefficients of the camera need to be known. The camera calibration toolbox from Caltech is used for this task. It utilizes the method of Zhang [19] as discussed in 2.2. The camera of the Galaxy S8 has been calibrated using the 8x11 checkerboard pattern of the test bench. This pattern contains 70 corners with known world coordinates. The origin of the world coordinate system is placed in the upper left corner as shown by figure 3.1. The X-axis is along vertical direction, while the Y-axis is along the horizontal direction. To obtain a right-handed coordinate system, the Z-axis must point towards the camera.+

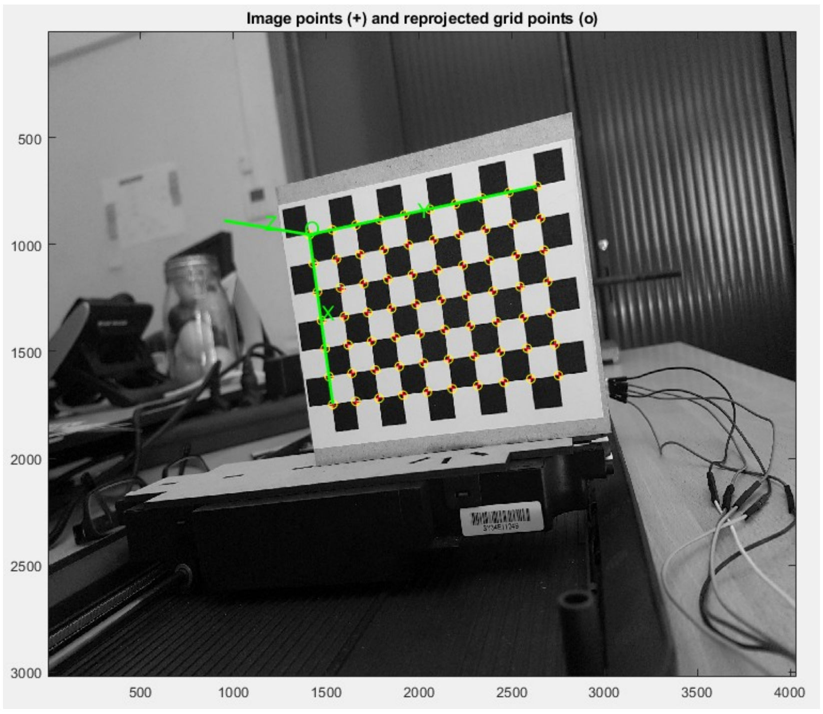


Figure 3.1: The world coordinate system

A total of 25 images are used to calibrate the camera. These images are shown by figure 3.2. The corners are extracted using the corner extraction engine of the toolbox. After this, the calibration is executed. The toolbox uses a gradient descent method, instead of Levenberg-Marquardt, to minimize the reprojection error. Table 3.1 shows the resulting intrinsic parameters and distortion coefficients. Note that the skew coefficient and the third radial distortion coefficient are not estimated by default. However, a skew coefficient of 0 is often a good assumption in most practical applications. the third radial distortion coefficient is left out of the calibration because the uncertainty is larger than the coefficient itself. Figure 3.3 shows the reprojection error for every corner on the checkerboard in every calibration image.

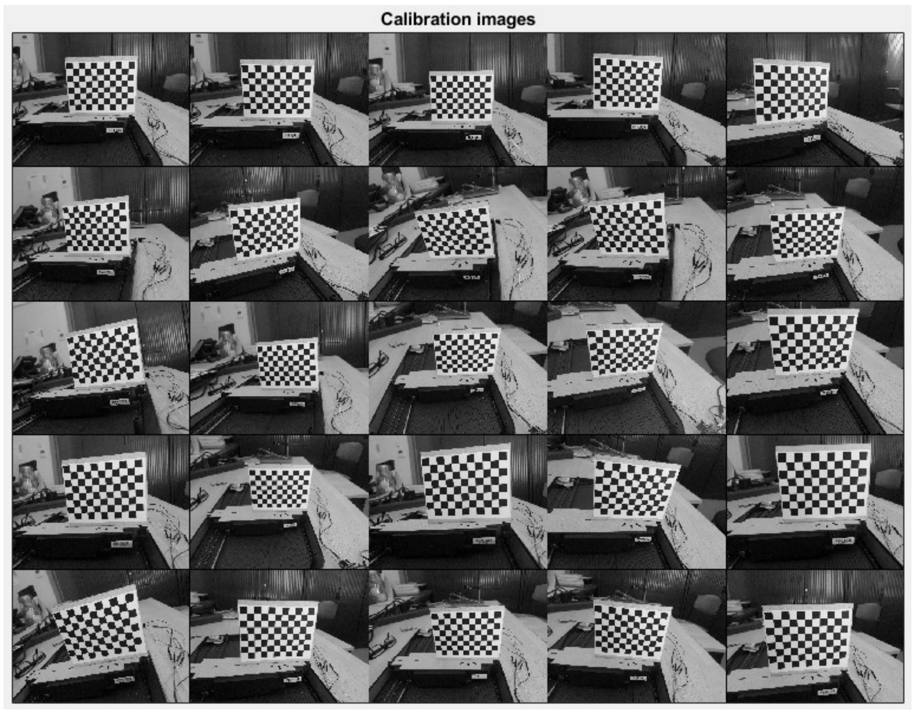


Figure 3.2: Calibration images

Table 3.1: Camera calibration results for the Samsung Galaxy S8

Focal length (pixels)	f_x	3237.13525 ± 17.67636
	f_y	3238.04681 ± 16.17327
Principal point (pixels)	c_x	1977.83019 ± 11.55264
	c_y	1510.92708 ± 14.03050
skew	s	0.00000 ± 0.00000
Radial distortion	k_1	0.11378 ± 0.00959
	k_2	-0.29215 ± 0.05808
	k_3	0.00000 ± 0.00000
Tangential distortion	p_1	0.00294 ± 0.00125
	p_2	-0.00290 ± 0.00144
Mean reprojection error (pixels)	x	0.65425
	y	0.64650

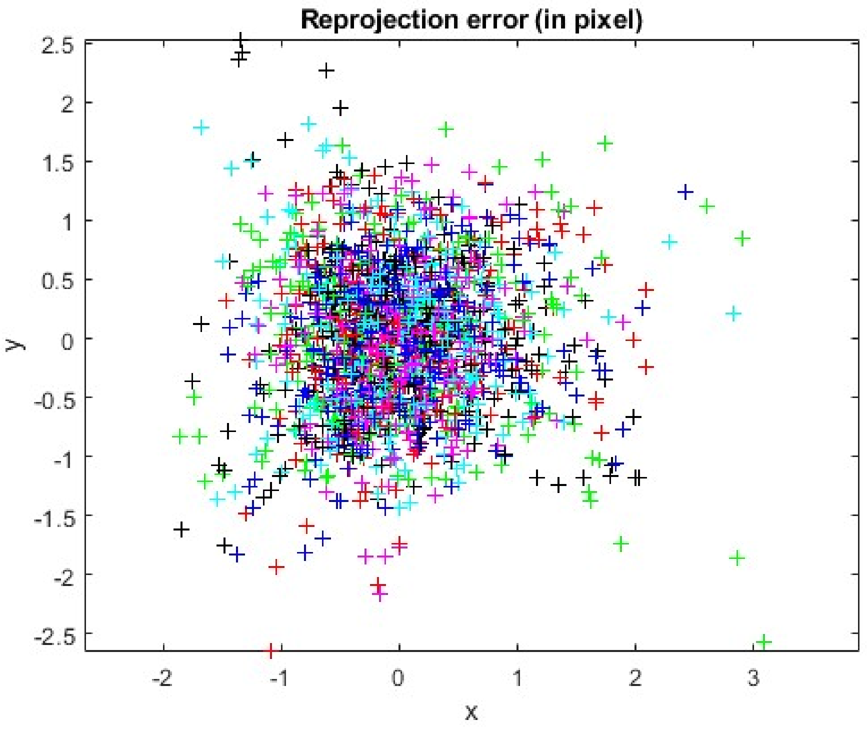


Figure 3.3: Reprojection error

3.2 Axis-angle transformation

The pose of a camera consists of a rotation matrix and translation vector. However, the rotation matrix has nine elements but only three degrees of freedom. In Levenberg-Marquardt, it is desired to optimize for as few parameters as possible. The axis-angle transformation, as discussed in 2.3, is used to parameterize the rotation using a rotation vector containing three parameters instead of nine. Having fewer parameters to estimate increases the speed of the algorithm. This section discusses the implementation of the axis-angle transformation. The function is described by algorithm 1 and is based on [16], [26]. It transforms a rotation matrix into the axis-angle representation and vice-versa.

If the input is a 3×1 rotation vector, the function will return the corresponding rotation matrix. The first step is to check that the rotation angle θ is bigger than a certain small value ϵ_1. In the Caltech toolbox, ϵ_1 is set to *eps* which is the relative accuracy of a floating point number in MATLAB. When θ is too small, there is no rotation and the identity matrix will be returned. Else, there is a rotation and the equivalent rotation matrix is returned using equation (2.31). It may be necessary to normalize the input if the rotation angle θ is larger than π.

If the input is a 3×3 matrix that satisfies $\boldsymbol{in}^T = \boldsymbol{in}^{-1}$ and $det(\boldsymbol{in}) = \pm 1$, then it is a rotation matrix and the function will return the corresponding axis-angle representation. A value ϵ_2, which is set to $(10 \cdot 10^{20}) \cdot eps$ in the Caltech toolbox, is used to test these conditions. First, the rotation matrix is projected to the special orthogonal group SO(3). This ensures that the determinant is equal to +1. This is also known as a proper rotation. The projection is done by performing a singular value decomposition and multiplying the matrix containing the left singular vectors with the transpose of the matrix containing the right singular vectors. The next step is to obtain the unit rotation axis v. It corresponds to the eigenvector belonging to the eigenvalue of 1. It can be shown that the rotation angle θ is equal to $atan2((\boldsymbol{v}^T \cdot \boldsymbol{v}_{hat}), (trace(\boldsymbol{R}) - 1))$. Here, $\boldsymbol{R}$ is the rotation matrix and $\boldsymbol{v}_{hat} = [\boldsymbol{R}(3,2) - \boldsymbol{R}(2,3), \boldsymbol{R}(1,3) - \boldsymbol{R}(3,1), \boldsymbol{R}(2,1) - \boldsymbol{R}(1,2)]^T$. The $trace$ operation calculates the sum of the diagonal elements. The $atan2$ function determines the four-quadrant inverse tangent. The unit rotation vector is then multiplied by the rotation angle to obtain the axis-angle representation.

Algorithm 1 Axis-angle transformation

```
function AxisAngle($in$)
    if $in \in \mathbb{R}^{3\times 1}$ then
        $\theta \leftarrow ||in||$
        if $\theta < \epsilon_1$ then
            $R \leftarrow I_3$
        else
            $t \leftarrow in$
            if $\theta > \pi$ then
                $t \leftarrow \frac{(\theta - 2\cdot\pi)\cdot in}{\theta}$
            end if
            $\theta \leftarrow ||t||$
            $t_x \leftarrow \begin{bmatrix} 0 & -t(3) & t(2) \\ t(3) & 0 & -t(1) \\ -t(2) & t(1) & 0 \end{bmatrix}$
            $R \leftarrow I_3 + \frac{sin(\theta)}{\theta} t_x + \frac{1-cos(\theta)}{\theta^2} t_x^2$
        end if
        return $R$
    else if $in \in \mathbb{R}^{3\times 3}$ and $||in^T \cdot in - I_3|| < \epsilon_2$ and $abs(det(in) - 1) < \epsilon_2$ then
        $R \leftarrow in$
        $[U, S, V] \leftarrow svd(R)$
        $R \leftarrow U \cdot V^T$
        $[V, D] \leftarrow eig(R)$
        $v \leftarrow$ Eigenvector in $V$ belonging to the eigenvalue equal to 1
        $v_{hat} = [R(3,2) - R(2,3), R(1,3) - R(3,1), R(2,1) - R(1,2)]^T$
        $\theta \leftarrow atan2((v^T \cdot v_{hat}), (trace(R) - 1))$
        return $v * \theta$
    end if
end function
```

3.3 Reversing lens distortion of image points

A necessary step before a 2D-3D correspondence can be used in the optimization, is to correct the 2D image point for lens distortion. Doing this beforehand, allows Levenberg-Marquardt to use a simpler reprojection model that does not take the lens distortion into account. This makes the required calculations easier throughout the whole optimization step. Algorithm 2 shows how to correct the image points. The implementation is based on the OpenCV library [35]. Equations (3.1) and (3.2) show the distortion model that is utilized. It contains 8 distortion parameters, 6 for radial distortion and 2 for tangential distortion. In contrast to (2.12) and (2.13), the radial distortion term is extended with a denominator part. In this thesis however, the extra radial distortion parameters are not estimated during the calibration and will thus be equal to zero.

Prior to inverting the distortion, the image coordinate needs to be normalized. This makes the coordinate independent of the intrinsic parameters. Next, the lens distortion is reversed using an iterative approach [36]. First, the lens distortion is estimated using the coordinate 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 and after denormalization, leads to an error that is smaller than a predetermined tolerance. Another stopping criterion is when the maximum number of iterations is reached.

$$x_{dist} = x \cdot \frac{1 + k_3 r^6 + k_2 r^4 + k_1 r^2}{1 + k_6 r^6 + k_5 r^4 + k_4 r^2} + (2p_1 xy + p_2(r^2 + 2x^2)) \tag{3.1}$$

$$y_{dist} = y \cdot \frac{1 + k_3 r^6 + k_2 r^4 + k_1 r^2}{1 + k_6 r^6 + k_5 r^4 + k_4 r^2} + (p_1(r^2 + 2y^2) + 2p_2 xy)) \tag{3.2}$$

Algorithm 2 Undistort image points

```
function UNDISTORT($ImagePoints, K, DistCoeffs, maxError, maxIters$)
    Extract intrinsic parameters from $K$
    $f_x \leftarrow K(1,1)$
    $f_y \leftarrow K(2,2)$
    $c_x \leftarrow K(1,3)$
    $c_y \leftarrow K(2,3)$

    for all $ImagePoints$ do
        //Normalize
        $x \leftarrow (x_{image} - c_x)/f_x$
        $y \leftarrow (y_{image} - c_y)/f_y$

        $x_0 \leftarrow x$
        $y_0 \leftarrow y$
        $itercounter \leftarrow 0$
        while $error > maxError$ and $itercounter < maxIters$ do
            //Undo distortion, $DistCoeffs = (k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6)$
            $r \leftarrow \sqrt{x^2 + y^2}$
            $x \leftarrow (x_0 - (2p_1xy + p_2(r^2 + 2x^2))) \cdot \frac{1+k_6r^6+k_5r^4+k_4r^2}{1+k_3r^6+k_2r^4+k_1r^2}$
            $y \leftarrow (y_0 - (p_1(r^2 + 2y^2) + 2p_2xy)) \cdot \frac{1+k_6r^6+k_5r^4+k_4r^2}{1+k_3r^6+k_2r^4+k_1r^2}$

            //Determine error
            $r \leftarrow \sqrt{x^2 + y^2}$
            $x_{dist} \leftarrow x \cdot \frac{1+k_3r^6+k_2r^4+k_1r^2}{1+k_6r^6+k_5r^4+k_4r^2} + (2p_1xy + p_2(r^2 + 2x^2))$
            $y_{dist} \leftarrow y \cdot \frac{1+k_3r^6+k_2r^4+k_1r^2}{1+k_6r^6+k_5r^4+k_4r^2} + (p_1(r^2 + 2y^2) + 2p_2xy))$

            $x_{proj} \leftarrow x_d \cdot f_x + c_x$
            $y_{proj} \leftarrow y_d \cdot f_y + c_y$

            $error = \sqrt{(x_{proj} - x_{image})^2 + (y_{proj} - y_{image})^2}$

            $itercounter \leftarrow itercounter + 1$
        end while
        //Denormalize
        $x_{undistorted} \leftarrow x \cdot f_x + c_x$
        $y_{undistorted} \leftarrow y \cdot f_y + c_y$
        return $(x_{undistorted}, y_{undistorted})$
    end for
end function
```

3.4 Computing the Jacobian and error vector

The Jacobian matrix is given by equation (2.39). The computation of this matrix is taken from [33]. Here, the calculation happens on a row-per-row basis. Before executing the Levenberg-Marquadt optimization, symbolic expressions need to be found for the first-order partial derivatives with respect to the six pose parameters for the reprojection error equations (2.34) and (2.35). The expressions are determined using the symbolic math toolbox of MATLAB and is summarized by algorithm 3. First, the rotation matrix is written in symbolic form in function of the three variables of the corresponding rotation vector using the Rodrigues transform. Secondly, a translation vector is made out of three other symbolic variables. Then, the symbolic expression of the reprojection of a world coordinate is calculated. The calibration matrix $\boldsymbol{K}$ is assumed to be given and contains the real values instead of symbolic variables. Note, that the third column of the rotation matrix and the Z-coordinate are left out because a flat checkerboard is used as pattern. Also, it does not contain the distortion coefficients as the obtained image points will be undistorted beforehand. After this, the symbolic expressions of the error terms d_{jx} and d_{jy} are found by introducing new variables for the x-coordinate and y-coordinate of an extracted, and undistorted, image point. Finally, the symbolic expressions of the column vectors $\left[\frac{\partial d_{jx}}{\partial \Theta_1}, \cdots, \frac{\partial d_{jx}}{\partial \Theta_6}\right]$ and $\left[\frac{\partial d_{jy}}{\partial \Theta_1}, \cdots, \frac{\partial d_{jy}}{\partial \Theta_6}\right]$ are computed using the *jacobian* function of the toolbox.

The result of algorithm 3 will be two long symbolic expressions that are a function of the six pose variables and the XY-coordinates of the world point. The Jacobian can now by calculated by simply substituting the variables with the values of the pose parameters and the coordinates of the world point. This is shown by algorithm 4. For every corner on the checkerboard, its world coordinates and the current values of the six pose parameters are used to evaluate the expressions of $\boldsymbol{J}_x$ and $\boldsymbol{J}_y$ to calculate the two corresponding rows of the Jacobian. It is advised to calculate $\boldsymbol{J}_x$ and $\boldsymbol{J}_y$ once for a certain calibration matrix $\boldsymbol{K}$ and copy them into the *CalculateJacobian* function instead of using the MATLAB function *subs* to evaluate the symbolic expressions. Because $\boldsymbol{J}_x$ and $\boldsymbol{J}_y$ are very long, the *subs* function will be very slow.

As discussed earlier, the error vector $\boldsymbol{f}(\boldsymbol{\Theta})$ contains the reprojection error in the x-coordinate and y-coordinate for every world point. It is also calculated on a row-per-row basis and described by algorithm 5. The implementation is based on [33]. For every point, equations (2.34) and (2.35) are applied and the result is stored in the error vector.

Algorithm 3 Finding symbolic expression of partial derivatives

```
function SymbolicJacobian($K$)
    //Define the necessary symbolic variables
    syms $txs\ tys\ tzs\ wxs\ wys\ wzs\ Xs\ Ys\ xim\ yim$ real

    //Symbolic expression of Rodrigues representation, [wxs], of rotation matrix
    $\theta \leftarrow sqrt(wxs^2 + wys^2 + wzs^2)$
    $t_x \leftarrow [0\ -wzs\ wys;\ wzs\ 0\ -wxs;\ -wys\ wxs\ 0]$
    $R \leftarrow I_3 + sin(\theta)t_x + (1 - cos(\theta))t_x^2$

    //Symbolic expression translation vector
    $t \leftarrow [txs; tys; tzs]$

    //Symbolic expression of reprojection
    $[u, v, w] \leftarrow K \cdot [R(:, 1), R(:, 2), t] \cdot [Xs; Ys; 1]$;
    $x \leftarrow u/w$
    $y \leftarrow v/w$

    //Symbolic expressions of error terms
    $d_{jx} \leftarrow xim - x$
    $d_{jy} \leftarrow yim - y$

    //Symbolic expression of first-order partial derivatives
    $J_x \leftarrow jacobian(dx, [wxs, wys, wzs, txs, tys, tzs])$
    $J_y \leftarrow jacobian(dy, [wxs, wys, wzs, txs, tys, tzs])$
    return $J_x, J_y$
end function
```

Algorithm 4 Computing the Jacobian

```
function CalculateJacobian($J_x, J_y, \boldsymbol{\Theta}, WorldPoints$)
    for i = 1:Number of WorldPoints do
        $X \leftarrow WorldPoints(1, i)$
        $Y \leftarrow WorldPoints(2, i)$

        $J$(2·(i-1)+1,:) $\leftarrow$ substitute variables in $J_x$ with pose parameters in vector $\boldsymbol{\Theta}$ and XY-coordinate of the $i^{th}$ world point
        $J$(2·(i-1)+2,:) $\leftarrow$ substitute variables in $J_y$ with pose parameters in vector $\boldsymbol{\Theta}$ and XY-coordinate of the $i^{th}$ world point
    end for
    return $J$
end function
```

Algorithm 5 Computing the error vector

```
function CalculateError($K, R, t, ImagePoints, WorldPoints$)
    for i = 1:Number of WorldPoints do
        $X \leftarrow WorldPoints(1, i)$
        $Y \leftarrow WorldPoints(2, i)$
        $x \leftarrow ImagePoints(1, i)$
        $y \leftarrow ImagePoints(2, i)$

        $[u', v', w'] \leftarrow K \cdot [R(:, 1), R(:, 2), t] \cdot [X; Y; 1]$;
        $x' \leftarrow u'/w'$
        $y' \leftarrow v'/w'$

        $f$(2·(i-1)+1,:) $\leftarrow x - x'$
        $f$(2·(i-1)+2,:) $\leftarrow y - y'$
    end for
    return $f$
end function
```

3.5 Determining the step

Recall that the step $\boldsymbol{h}$ is the solution of the linear least squares problem $\begin{bmatrix} \boldsymbol{f}(\boldsymbol{\Theta}) \\ \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{J} \\ \sqrt{\lambda}\boldsymbol{I} \end{bmatrix} \boldsymbol{h} \approx \boldsymbol{0}$. This can be determined in two ways. The first possibility is to solve the augmented normal equations described by equation (2.41). In this thesis however, the solution via orthogonal transformation is chosen because it is mathematically more accurate. The implementation is shown by algorithm 6. It takes as input the Jacobian matrix, the damping factor and the error vector. A linear least squares problem is generally written as $\boldsymbol{A}\boldsymbol{x}^* \approx \boldsymbol{b}$. 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 next step is to compute the QR-decomposition of $\boldsymbol{A}$. Only the first six columns of $\boldsymbol{Q}$ and the first six rows of $\boldsymbol{R}$ will be needed.

In MATLAB, this is indicated by adding a zero as the second argument to the *qr* function. To find the solution for $\boldsymbol{h}$, $\boldsymbol{R}_1\boldsymbol{h} = \boldsymbol{Q}_1^T\boldsymbol{b}$ needs to be solved. This can be done via backward substitution because $\boldsymbol{R}_1$ is an upper triangular matrix [37]. Backward substitution starts by calculating the last unknown, $\boldsymbol{h}(6)$ in this case. This is then used to solve for $\boldsymbol{h}(5)$. These two known parameters then make it possible to calculate $\boldsymbol{h}(4)$, and so on.

Algorithm 6 Computing the step

```
function STEP($J, \lambda, f$)
    $A \leftarrow \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix}$
    $b \leftarrow -\begin{bmatrix} f \\ [0,0,0,0,0,0]^T \end{bmatrix}$
    $[Q_1, R_1] \leftarrow qr(A, 0)$

    //Solve $R_1 h = Q_1^T b$ through backward substitution
    $qtb \leftarrow Q_1^T b$
    for i = 6:-1:1 do
        $h(i) \leftarrow (qtb(i) - R(i,:) \cdot h)/R(i,i)$
    end for

    return $h$
end function
```

3.6 The complete algorithm

This section describes the full camera pose estimation algorithm used in the PoseLab framework. The code in [31], [33] was taken as a starting point. Figure 3.4 shows the flowchart of the implemented algorithm. The corresponding pseudocode can be found in appendix A.

Certain variables need to be initialized before Levenberg-Marquardt can start finding the optimal extrinsic parameters. First, the calibration matrix and distortion coefficients are required. They are retrieved from the results of a calibration performed previously. In addition, the symbolic expressions necessary to compute the Jacobian matrix need to be obtained by using the function *SymbolicJacobian* described in 3.4. The Levenberg-Marquardt optimization also requires an initialization of the pose parameters. Normally, an analytical method, such as P3P and EPnP, is used to determine the pose initialization. When such a method returns the orientation as a rotation matrix, an axis-angle transformation has to be applied. In a real-time application however, the pose calculated on the previous frame can be used as an initialization. Finally, 2D-3D point correspondences are needed. In a dedicated hardware solution where the frame is read row-by-row, the algorithm can start as soon as the first points are found.

The algorithm then continues by computing the Jacobian and the error vector

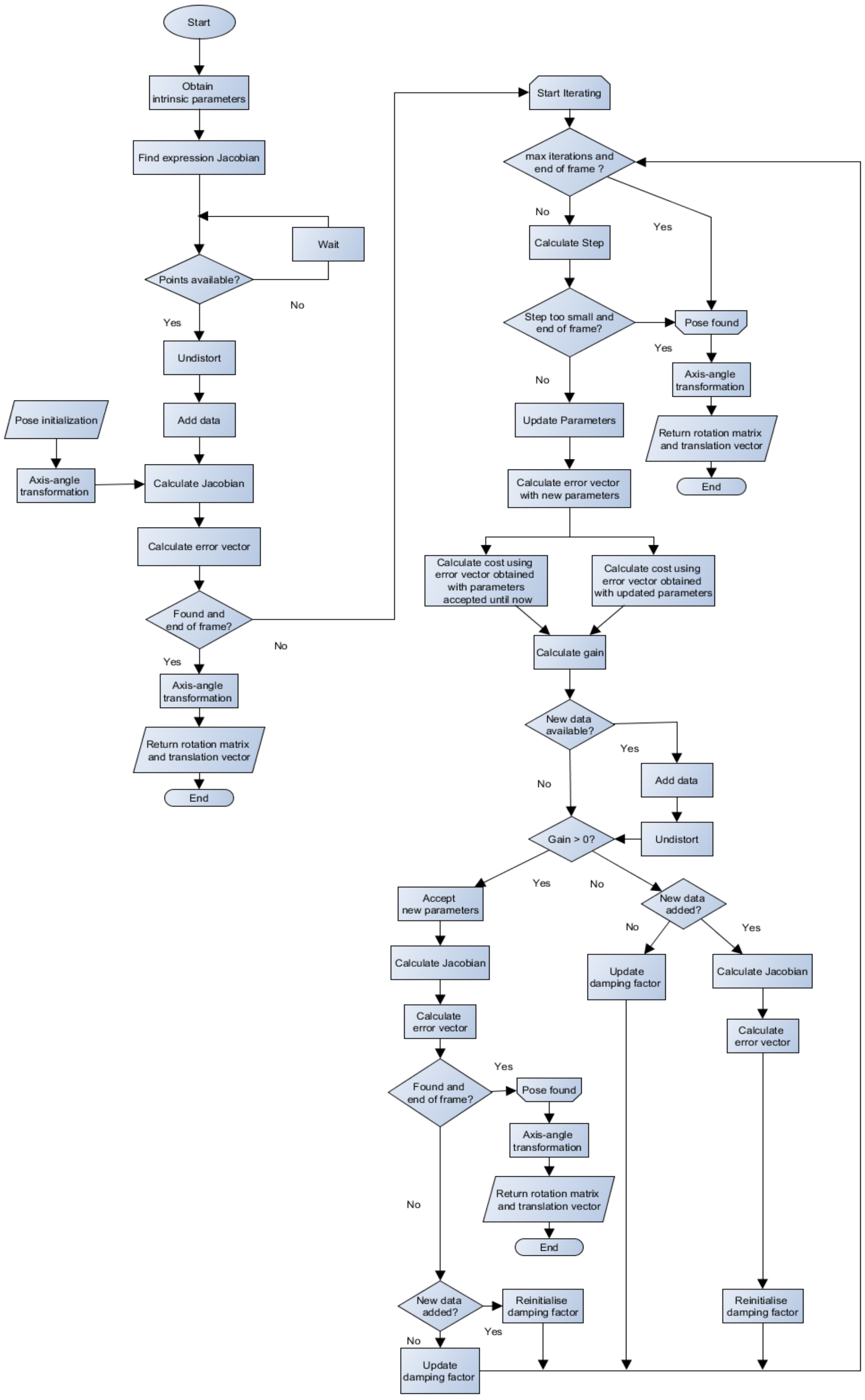

Start
Obtain intrinsic parameters
Find expression Jacobian
Wait
Points available?
No
Yes
Undistort
Add data
Pose initialization
Axis-angle transformation
Calculate Jacobian
Calculate error vector
Found and end of frame?
No
Yes
Axis-angle transformation
Return rotation matrix and translation vector
End
Start Iterating
max iterations and end of frame ?
No
Yes
Calculate Step
Step too small and end of frame?
Pose found
Yes
No
Axis-angle transformation
Return rotation matrix and translation vector
End
Update Parameters
Calculate error vector with new parameters
Calculate cost using error vector obtained with parameters accepted until now
Calculate cost using error vector obtained with updated parameters
Calculate gain
New data available?
Yes
No
Add data
Undistort
Gain > 0?
Yes
No
Accept new parameters
New data added?
No
Yes
Calculate Jacobian
Update damping factor
Calculate Jacobian
Calculate error vector
Calculate error vector
Found and end of frame?
Yes
Pose found
Axis-angle transformation
Return rotation matrix and translation vector
End
No
New data added?
Reinitialise damping factor
Yes
No
Update damping factor
Reinitialise damping factor

Figure 3.4: Flowchart of the camera pose estimation algorithm

at the initialization parameters by using algorithms 4 and 5. In case all the 2D-3D point correspondences are known beforehand, such as a software application, the initial parameters may already minimize cost function $F(\boldsymbol{\Theta})$. It is therefore checked if the gradient $\boldsymbol{g} = \boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{\Theta})$ is equal to zero. This condition is met if the supremum norm $\|\boldsymbol{g}\|_\infty$ is less than a predetermined threshold ε_1. In this context, the supremum norm is the same as the maximum absolute value in the gradient vector. The threshold is chosen to be 10^{-8}. When already at the minimum, the pose of the camera is described by the initialization parameters. The Rodrigues representation is converted into a rotation matrix via an axis-angle transformation and is returned together with the translation vector. In a hardware implementation however, not all the correspondences may be known at the start. That is why an extra condition that the camera needs to be at the end of the frame has to be met in order to stop the algorithm.

When the initialization parameters do not minimize the cost function, the iteration process begins. Before this can start, the damping factor λ needs to be initialized. In [38], they suggest to use a value proportional to the maximum value of the diagonal of matrix $\boldsymbol{J}^T\boldsymbol{J}$. This can be expressed as $\eta \cdot max\left\{diag(\boldsymbol{J}^T\boldsymbol{J})\right\}$. Here, η is a small value and was chosen to be 10^{-8}. The factor υ, used when the step does not lead to a decrease in cost function, is initialized to 2. After this, the Levenberg-Marquardt iterations start. If the maximum number of iterations is not reached and the camera is not at the end of the frame, the step $\boldsymbol{h}$ is calculated as described in 3.5. Then, it is checked whether the step 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}. When the step is too small, and in case of a hardware implementation the camera has reached the end of the frame, the camera pose parameters are found and the corresponding rotation matrix and translation vector are returned. Else, the parameters are increased by $\boldsymbol{h}$.

Next, it is determined whether the step has led to a decrease in the cost function. This is done by evaluating the gain ratio ϱ described by 2.42. To do this, the error vector $\boldsymbol{f}(\boldsymbol{\Theta} + \boldsymbol{h})$ is calculated at the updated parameters. Then, the cost function is computed two times using equation (2.36). The error vector $\boldsymbol{f}(\boldsymbol{\Theta})$ is used to calculate the cost function $F(\boldsymbol{\Theta})$ at the old parameters, and $\boldsymbol{f}(\boldsymbol{\Theta} + \boldsymbol{h})$ determines the cost function $F(\boldsymbol{\Theta} + \boldsymbol{h})$ at the updated parameters. The numerator of the gain ratio is then found by subtraction. The denominator is computed as $\frac{1}{2}\boldsymbol{h}^T(\lambda\boldsymbol{h} - \boldsymbol{g})$.

While the algorithm is iterating, the camera is at the same time reading the frame row-by-row. Before the correct actions are taken based on the value of the gain ratio, it is checked if new 2D-3D point correspondences are found by the camera. If they are available, they are added to the data. This will lead to two extra rows in the Jacobian and error vector per point correspondence. Different calculations will be needed if data is added.

If the gain is calculated to be greater than zero, the step has led to a decrease in cost function and the updated parameters are accepted as the current best estimate for the camera pose. The Jacobian and error vector are evaluated at these new parameters, and possibly with extra data. The gradient is recalculated and its supremum norm is tested against ε_1 to see if the parameters minimize the cost function. If they do, and the camera has reached the end of the frame, the camera pose is found and the rotation matrix and translation vector are returned. If not, more iterations are needed and the damping factor λ needs to be updated. It was chosen to reinitialize λ as described before when new data was added. If no extra data was added, λ is updated by multiplying it with $max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$ and setting υ equal to 2.

If the gain is calculated to be less than or equal to zero, the step does not cause a decrease in the cost function. The updated parameters are not accepted and the subsequent actions depend on whether new data is added or not. When no new data is added, λ is multiplied by υ in order to try and find a new step that does lead to a decrease in the cost function. The factor υ itself is multiplied by 2. If new data is added, the Jacobian, error vector and gradient need to be recalculated. The damping factor λ has to be reinitialized because there may have been successive iterations that have failed to decrease the cost function. This leads to a very large value of λ because it is multiplied by υ on each failed iteration. The initialization is done as described before.

Chapter 4

Experiments

4.1 The setup

The experiments were done using the test bench of Thomas Aerts [17]. Figure 4.1 shows the setup. 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 the checkerboard marker to the scanner bar. The camera is clamped on a stand of the chemistry lab. A bluetooth click device is used to take images remotely. This way, there is no need to push the button which can alter the position of the camera. The laptop connected to the Arduino is put on a different table. This is done in order to minimize the effects on the test setup due to the movements of the person sitting behind the laptop.

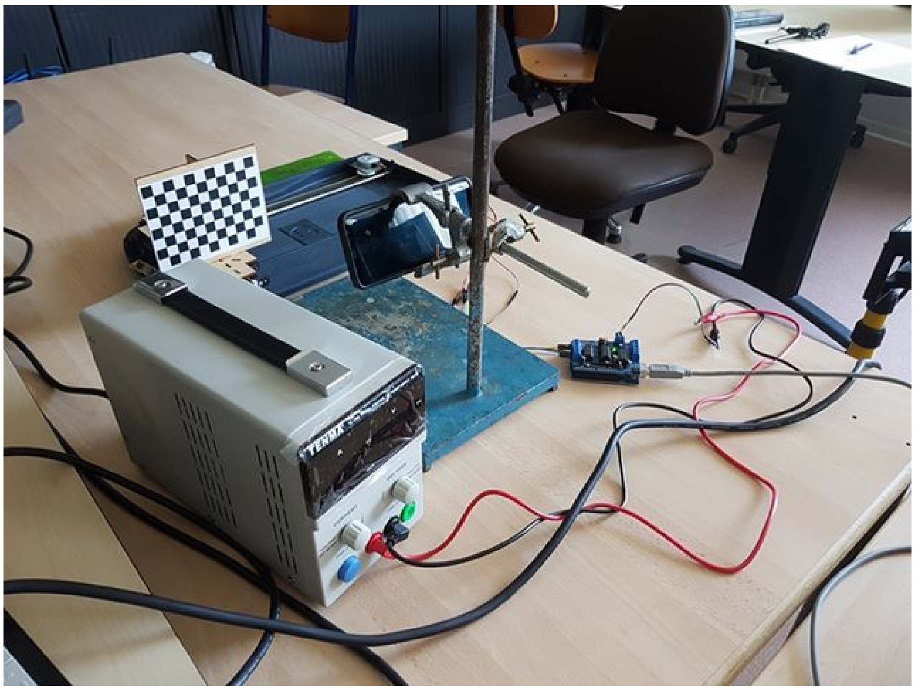


Figure 4.1: The test bench

A 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 20μm. In the experiments done in this thesis, 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.

In order to measure how far the marker is displaced as a result of one full step of the stepper motor, modifications have been made to the scanner as shown by figure 4.2. A ruler was glued to the scanner. It can measure up to 0.025 inches or 0.635 mm. In addition, a piece of wood was glued to the scanner bar holding the marker. The edge of the wood was used to read the value of the ruler. The marker was placed at one side of the scanner and the value read of the ruler was 11.575 inches or 294.005 mm. Then, the stepper motor made 250 steps and the ruler was read again. This time, it was a value of 4.95 inches or 125.73 mm. Thus, a total distance of 168.275 mm was traveled. This leads to a displacement of 0.673 mm $\pm$ 0.003 mm per step.

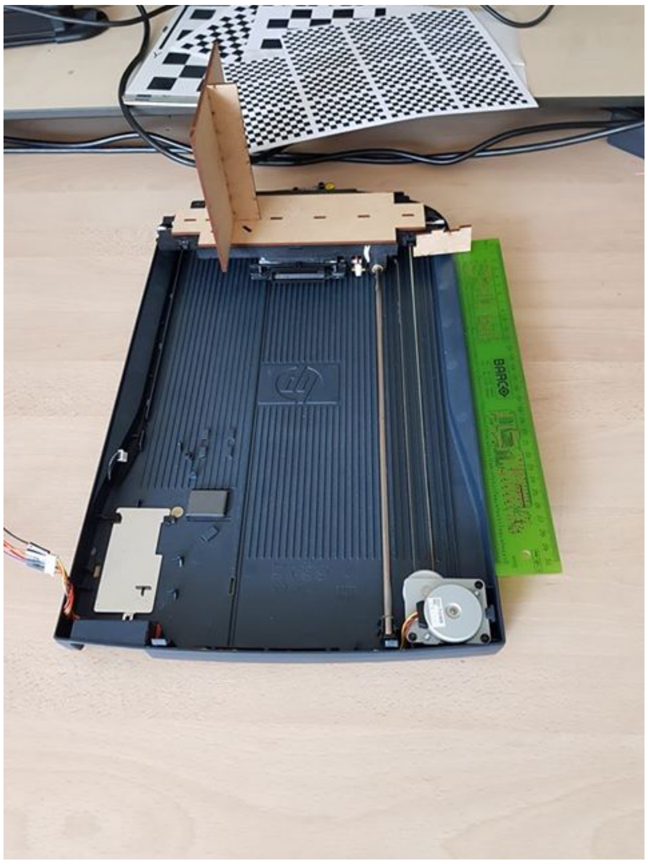
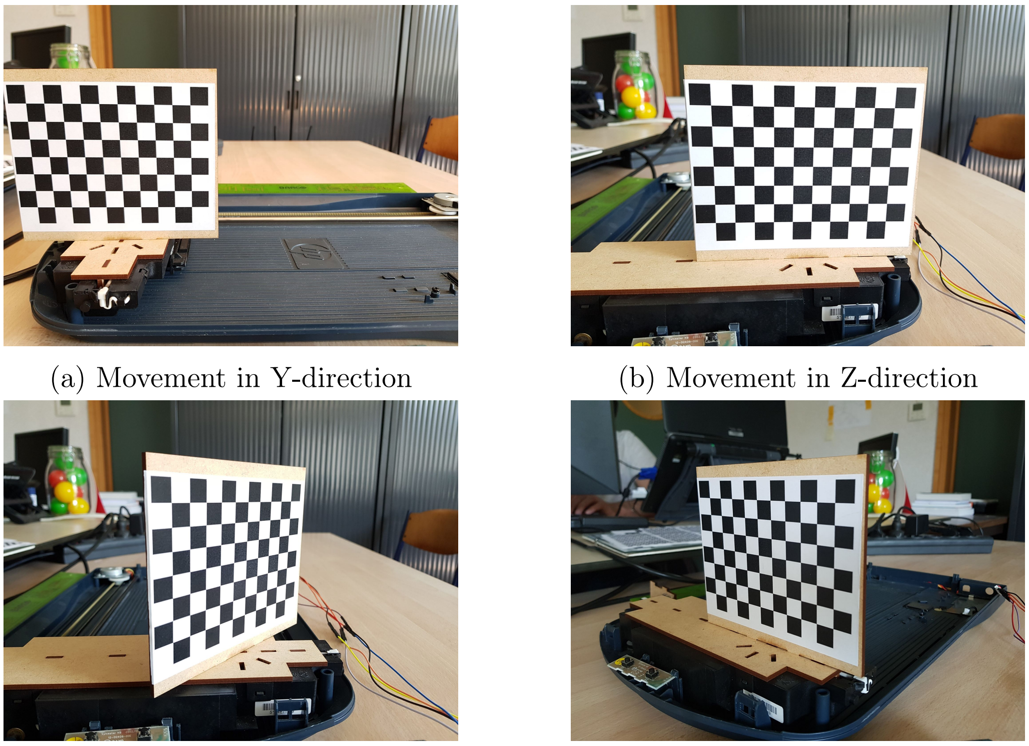


Figure 4.2: Setup to measure the displacement of a step

4.2 The procedure

To test the PoseLab framework, four experiments have been made. In each case, the marker is moved linearly along the scanner. The starting point is at the end of the scanner as seen in figure 4.1. Then, the marker is moved 10 full steps at a time. After each 10 steps, an image is taken. This is done 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 described earlier. Figure 4.3 shows the starting position for each experiment. First, the marker is moved in the Y-direction. Secondly, the scanner and marker are turned 90 degrees to simulate movement in the Z-direction. Then, the marker 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.


(a) Movement in Y-direction

(b) Movement in Z-direction

(c) Marker turned 45 degrees

(d) Camera turned with respect to the scanner

Figure 4.3: Starting positions of the different experiments

For every image, the first step is to calculate the pose using the *Compute Extrinsic* function of the Caltech toolbox. Using the calibration results from section 3.1, the function starts by detecting the corners of the checkerboard using its corner extraction engine. Based on the found 2D-3D point correspondences, the extrinsic parameters are calculated. Finally, the rotation matrix and translation vector are returned. Additionally, the Rodrigues representation of the rotation matrix is given. The extracted corners and their respective world coordinates are also saved.

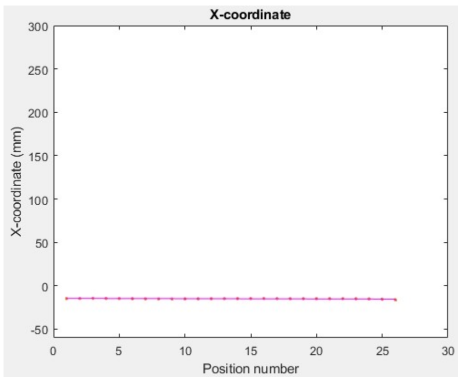
Next, the camera pose is computed by PoseLab. The same corner coordinates and corresponding world coordinates as in the Caltech toolbox are used. In addition to the 2D-3D point correspondences, the MATLAB function requires the calibration matrix, distortion coefficients, and an initialization rotation matrix and translation vector. To get an initialization, the result of the Caltech toolbox is taken and a value of 0.05 rad was added to the roll and yaw angles, 0.04 rad to the pitch angle, 2 mm to the X-coordinate and 3 mm to the Y-coordinate and Z-coordinate. These values are chosen arbitrarily. Furthermore, because the long term goal is to create a hardware solution and this is still a software implementation, the addition of new data whilst iterating needs to be simulated. This is done by adding a new row, consisting of 10 checkerboard corners, to the used 2D-3D point correspondences after every 5 iterations of Levenberg-Marquardt.

4.3 Results

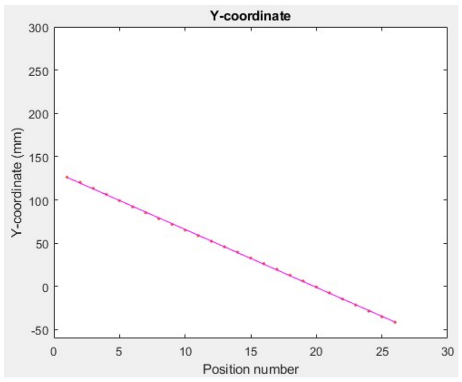
This section gives the results of the four different experiments. For every experiment, the value of each pose parameter is plotted against the position number during the experiment. Despite the fact that there is no way to measure and verify the rotation, the rotational parameters are given for completeness. The orange dots in the plots are the values as calculated by the MATLAB PoseLab. Because the scanner creates a linear motion, straight lines can be fitted to the coordinates of the camera position.

4.3.1 Movement in Y-direction

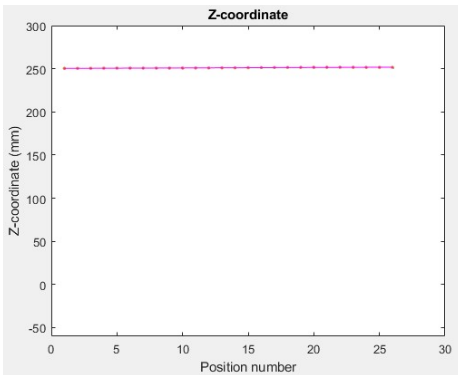
Figure 4.4 shows the calculated pose parameters for every image in the experiment. The slope of the line fitted to the Y-coordinate is calculated to be -6.691 mm per 10 steps. The slope of the X-coordinate was determined to be -0.0260 mm per 10 steps. The slope of the Z-coordinate was -0.0561 mm per 10 steps. Figure 4.5 shows the deviations from the calculated values to the fitted line for all three coordinates. These errors are also called the residuals.



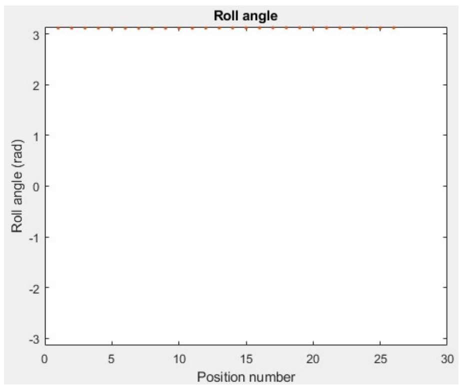
(a)



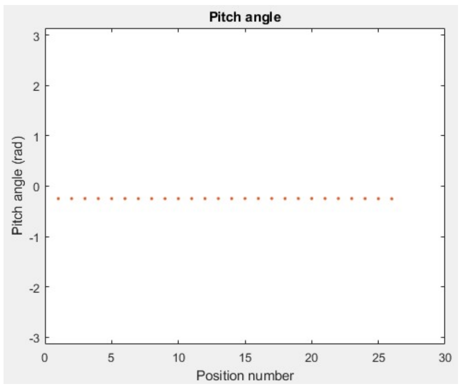
(b)



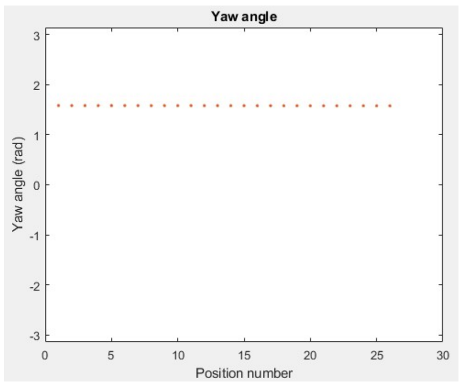
(c)



(d)



(e)



(f)

Figure 4.4: Pose parameters for every position for a movement in the Y-direction

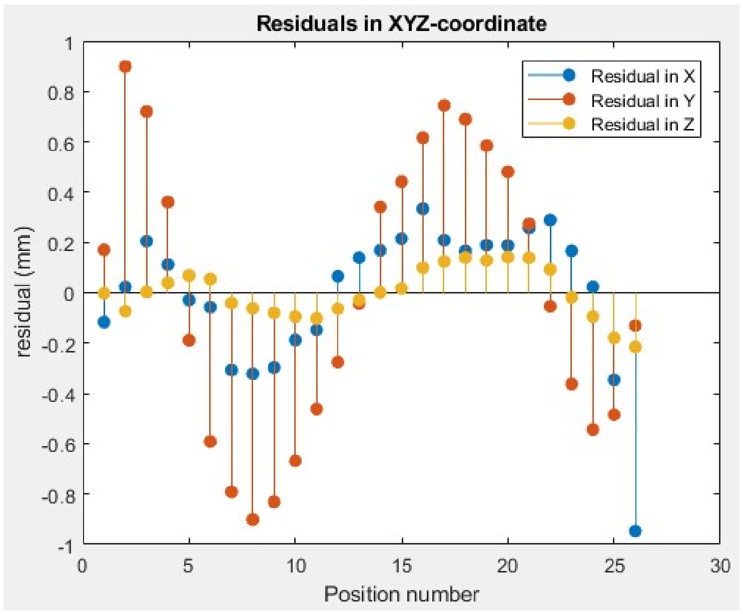
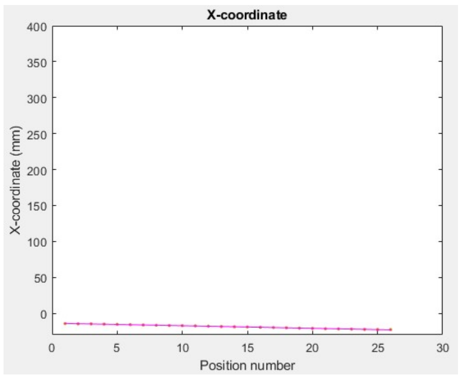


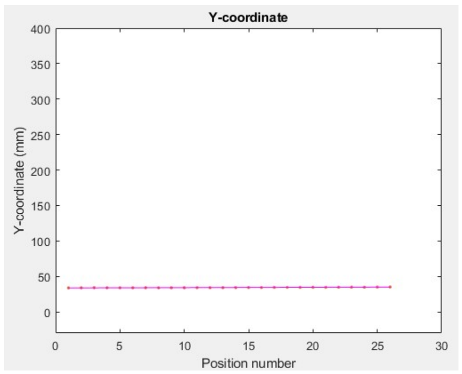
Figure 4.5: Residuals of XYZ-coordinate for a movement in the Y-direction

4.3.2 Movement in Z-direction

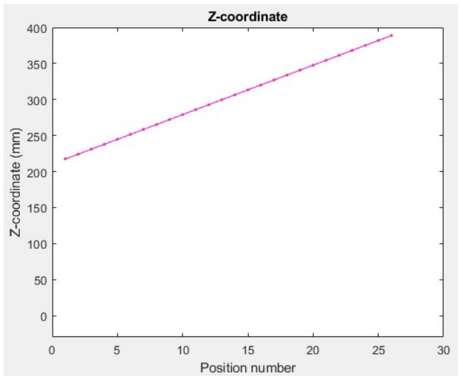
The pose parameters during this experiment are given by figure 4.6. The slope of the Z-coordinate was calculated to be 6.861 mm per 10 steps. This leads to an average change of 0.6861 mm in the Z-coordinate per full step. The slope of the X-coordinate and Y-coordinate were -0.3565 mm per 10 steps and 0.0446 mm per 10 steps respectively. The residuals are shown by figure 4.7.



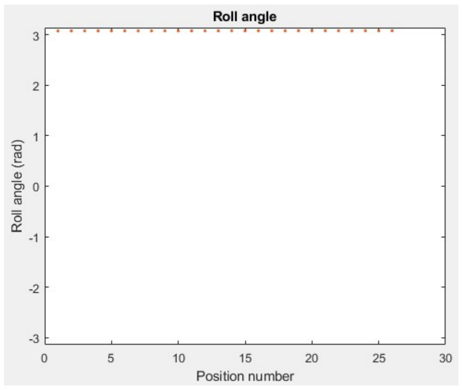
(a)



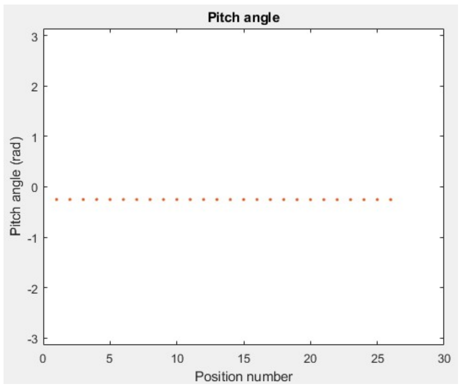
(b)



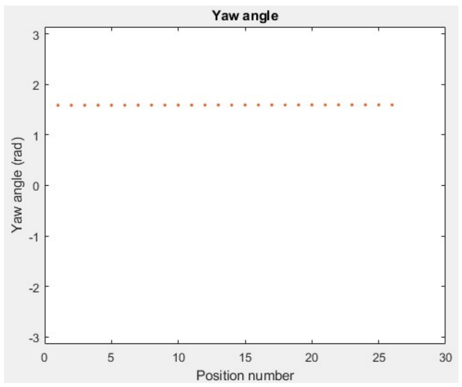
(c)



(d)



(e)



(f)

Figure 4.6: Pose parameters for every position for a movement in the Z-direction

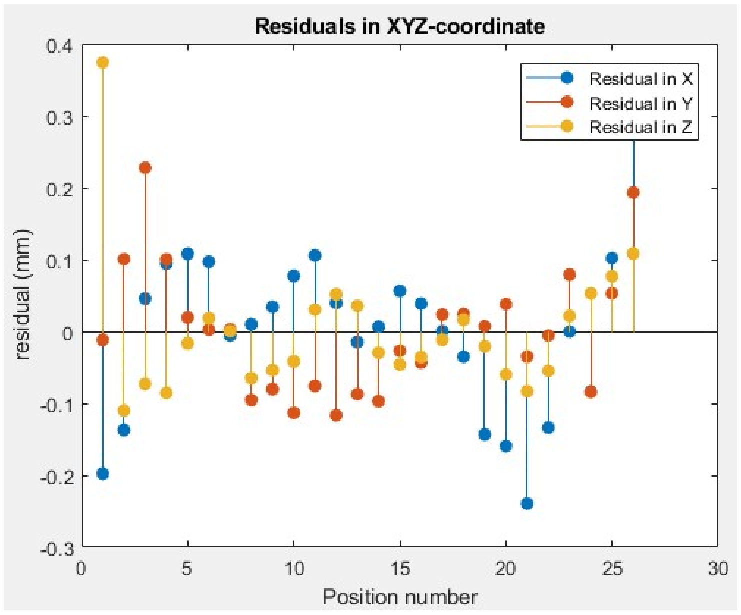
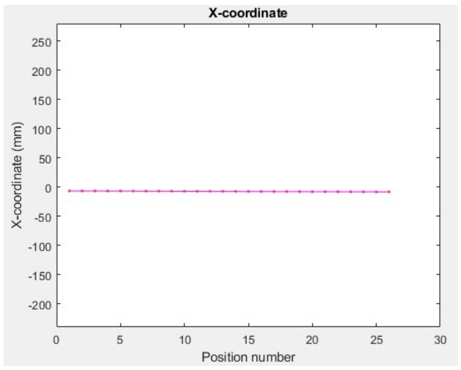


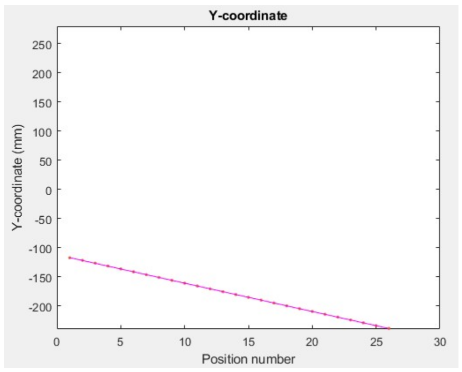
Figure 4.7: Residuals of XYZ-coordinate of each image for a movement in the Z-direction

4.3.3 Marker turned 45 degrees

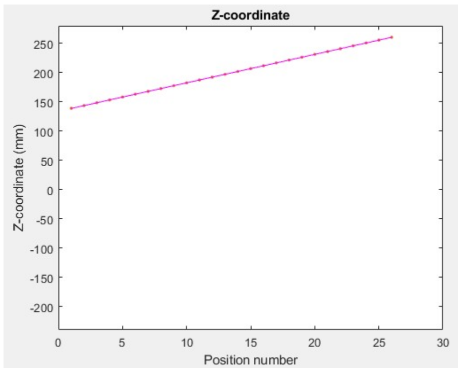
Figure 4.8 shows the calculated pose parameters for the experiment where the marker is turned 45 degrees and moved away from the camera. The slope of the Z-coordinate was 4.8655 mm per 10 steps, whilst the slope of the Y-coordinate was -4.8659 mm per 10 steps. The X-coordinate had a slope of -0.0711 mm per 10 steps. The residuals are shown by figure 4.9.



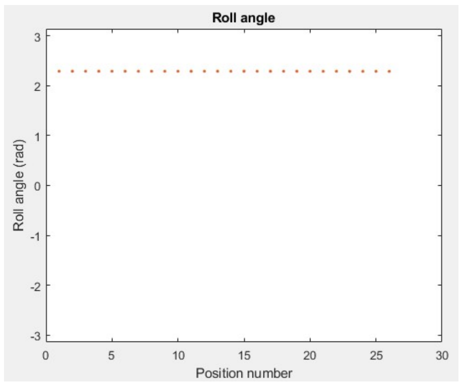
(a)



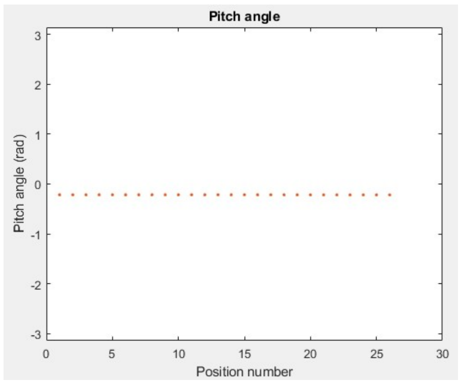
(b)



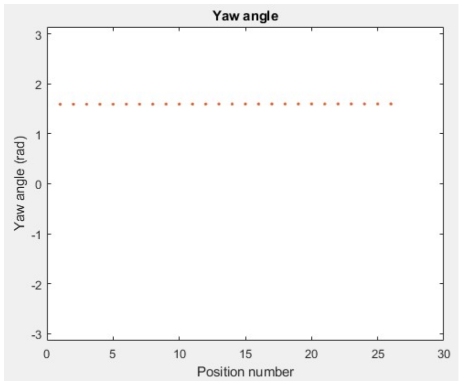
(c)



(d)



(e)



(f)

Figure 4.8: Pose parameters for every position for the marker turned 45 degrees

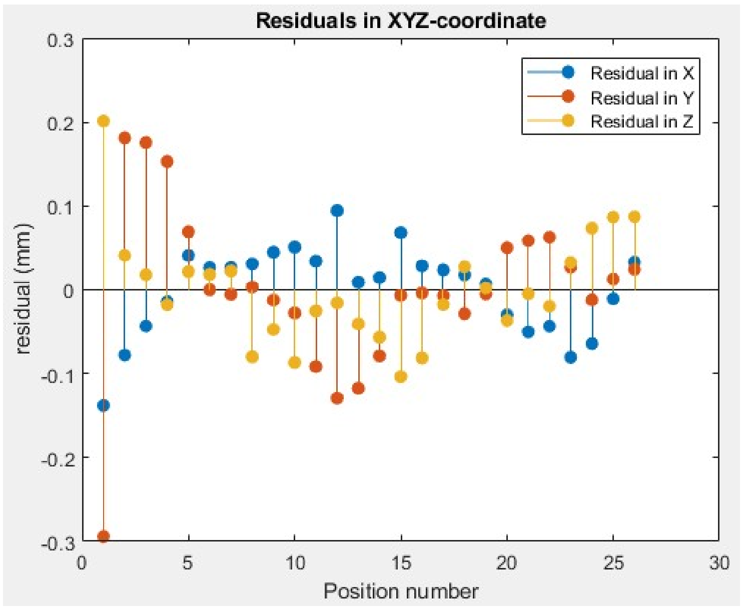
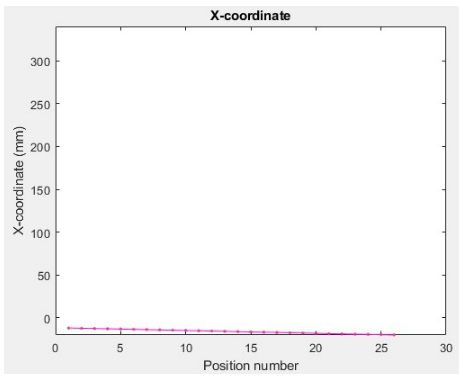


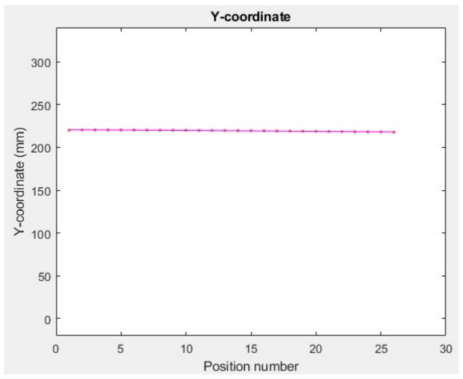
Figure 4.9: Residuals of XYZ-coordinate of each image for the marker turned 45 degrees

4.3.4 Camera turned with respect to scanner

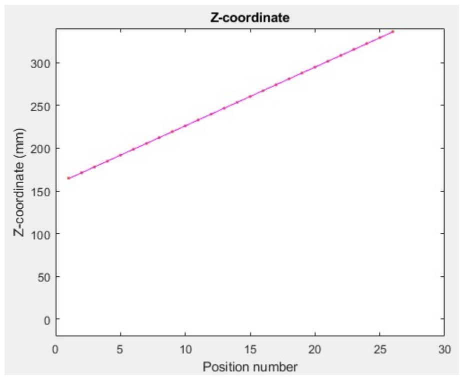
Figure 4.10 shows the calculated pose parameters for the experiment where the camera was turned approximately 45 degrees in order to not align the movement of the marker with the optical axis. The slope of the Z-coordinate was 6.8632 mm per 10 steps. The slope of the X-coordinate and Y-coordinate were -0.3299 mm per 10 steps and -0.1034 mm per 10 steps respectively. The residuals are shown by figure 4.11.



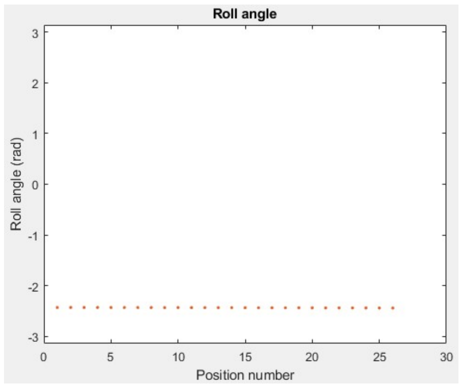
(a)



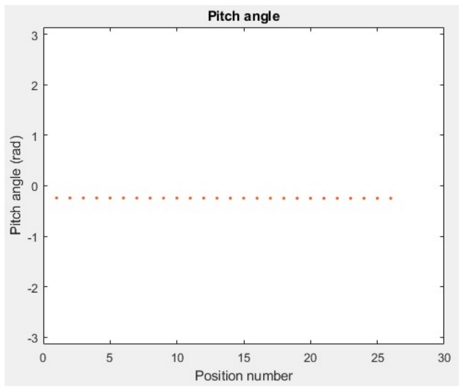
(b)



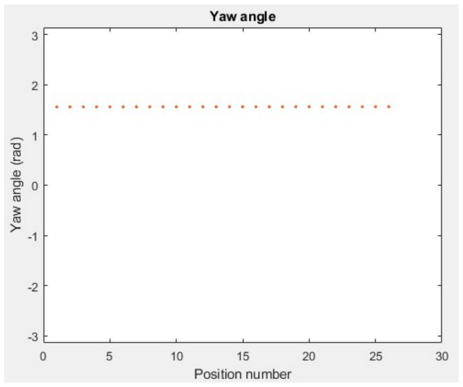
(c)



(d)



(e)



(f)

Figure 4.10: Pose parameters for every position for the camera turned approximately 45 degrees

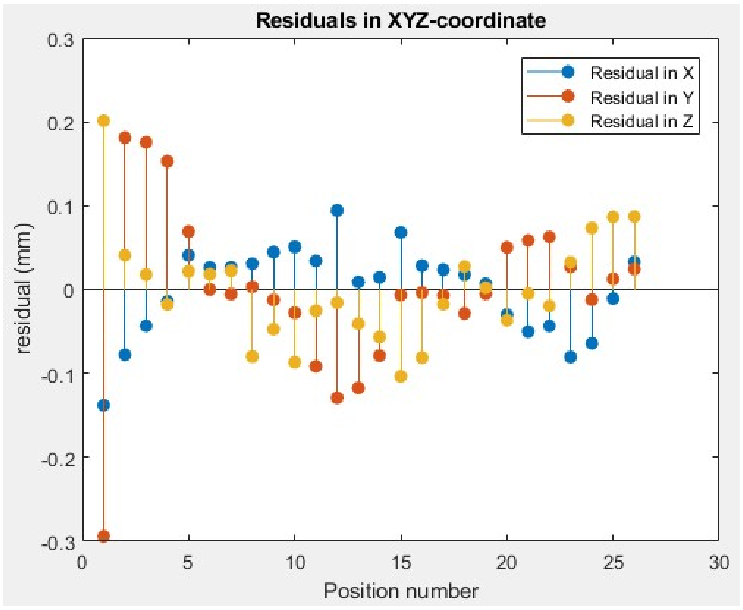


Figure 4.11: Residuals of XYZ-coordinate of each image for the camera turned approximately 45 degrees

4.4 Discussion

A modified flatbed scanner has been used to create a precise linear movement. This can clearly be seen in the resulting plots for the X, Y and Z-coordinates. The fitted straight lines matched the data well. From the Y-coordinate of the first experiment and the Z-coordinate of the second and fourth experiment, 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 value of the coordinate as a result of a single step. In the first experiment, this leads to a change of 0.6691 mm in Y-coordinate per step. In the second experiment, a step changes the Z-coordinate by 0.6861 mm. Similarly, a step in the fourth experiment leads to a change of 0.6863 mm. These values are close to the measured step size of 0.6731 mm.

The deviation from the measured value can have many causes. The main cause is that the test 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. Also, the manner in which the stepper motor is driven has an effect on the stability and reproducibility.

Observing the plots of 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 the sort of images used during the calibration. In most images, the checkerboard is in the middle. Because the lens distortion is most significant on the edges of the image, the checkerboard corners do not give a lot of information on the amount of distortion.

This may lead to incorrect values of the distortion coefficients. 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.

4.5 Comparison to Caltech toolbox

In addition to finding the 2D-3D point correspondences for each image, the Caltech toolbox also calculates the camera pose. To compare the results with the MATLAB PoseLab, the absolute differences for each pose parameter are examined. The mean difference and the standard deviation are given by table 4.1. It can be seen that results of PoseLab are in accordance with the Caltech toolbox. However, it is difficult to tell how accurate the calculated parameters are because the test bench does not allow the verification of absolute camera poses. One reason for the differences is that the toolbox uses gradient descent as optimization method, whilst here Levenberg-Marquardt is used. Another possibility is the quality of the initialization.

Table 4.1: Absolute difference between Caltech toolbox and PoseLab for each pose parameter

Pose parameter	Mean	Standard deviation
x (mm)	0.0056	0.0128
y (mm)	0.0028	0.0066
z (mm)	0.0020	0.0019
θ (rad)	1.0184e-05	2.4659e-05
ϕ (rad)	2.0706e-05	4.9513e-05
γ (rad)	5.3242e-06	6.4381e-06

4.6 Addition of point correspondences whilst iterating

In the PoseLab framework developed in this thesis, it is made possible to add new data while iterating. For a single image, the number of point correspondences, damping factor and cost are examined in function of the iteration number. The tenth image of the first experiment is taken as test. At the end of each iteration, the size of the buffer containing the point correspondences, the damping factor and cost are saved. Figures 4.12 to 4.14 show the result.

From 4.12, it can be clearly seen how the buffer contains 10 new point correspondences, a row of checkerboard corners, after every 5 iterations. Figure 4.13 illustrates how the damping factor is reinitialized when new point correspondences are added. The cost, shown by figure 4.14, starts at a value of 7.5e4. After the first iteration, the cost is reduced to 30. When the number of correspondences increases, the overall cost rises because there are simply more points. However, the reduction in cost due to extra iterations becomes less as the number of point correspondences increases. This is because the information about the camera pose carried by new data is less as more point correspondences are known. There is still a reduction but this cannot be clearly seen on the figure because of the scale.

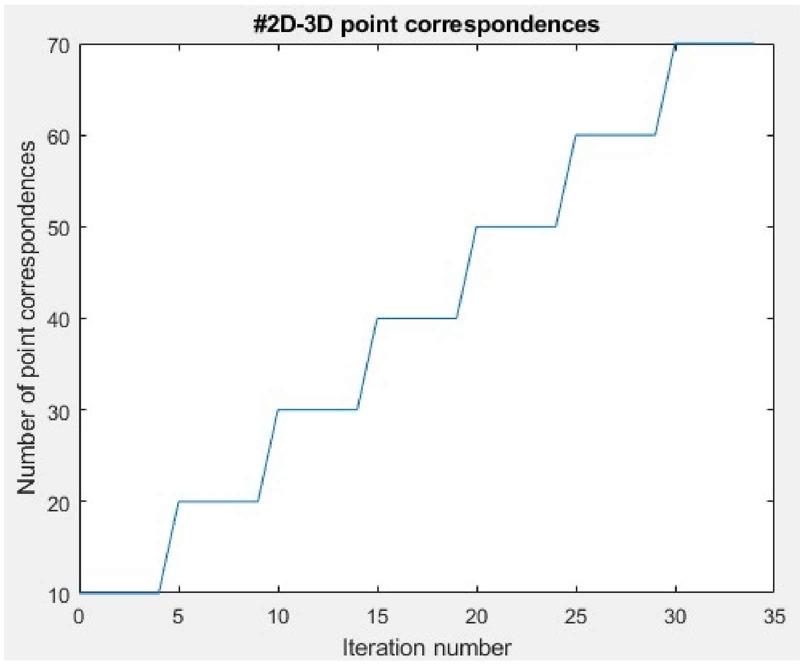


Figure 4.12: Number of 2D-3D point correspondences in function of iteration number

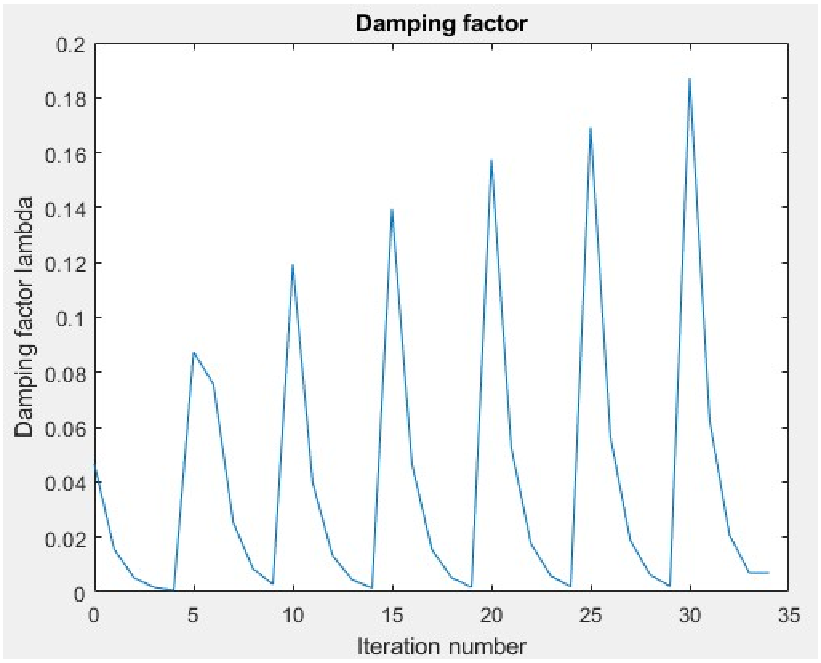


Figure 4.13: Value of the damping factor in function of iteration number

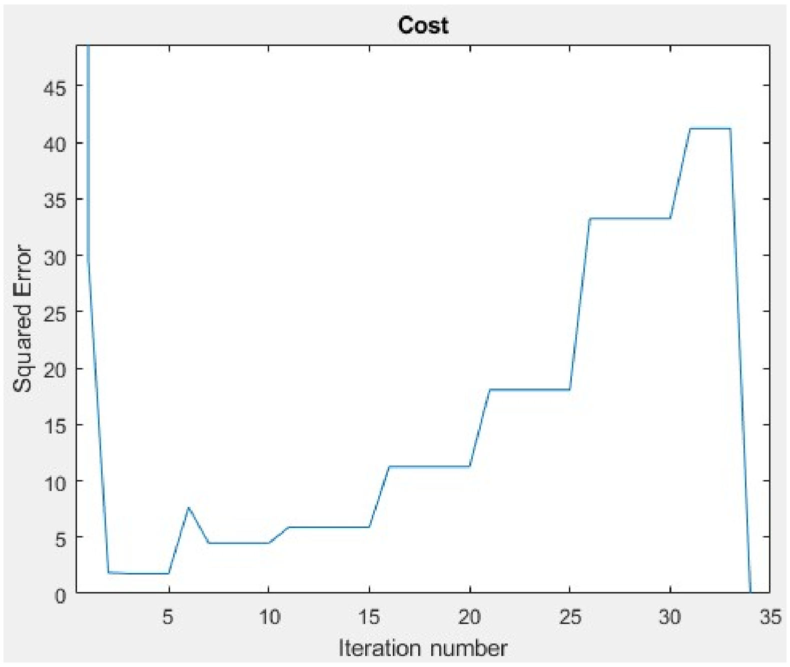


Figure 4.14: Value of the cost at every iteration

Chapter 5

Conclusion

5.1 PoseLab implementation

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 obtain the desired processing speeds. This requires knowledge of every calculation needed to retrieve the pose of the camera. An extensive literature review is needed to understand every step. This thesis explains the basic theory of the pinhole camera model, camera calibration, 6 degrees of freedom camera pose estimation and the Levenberg-Marquardt optimization.

The first goal of this Master's thesis is to create the PoseLab experimentation platform in MATLAB that uses the Levenberg-Marquardt optimization method to compute the camera pose. The second goal is to make it possible to accept 2D-3D point correspondences whilst iterating. The implementation is discussed of the necessary functions to perform the axis-angle transformation, lens distortion reversion and the computation of the Jacobian matrix, error vector and step. The flow chart of the complete algorithm is also given. To allow the addition of new data, changes need to be made to the Levenberg-Marquardt algorithm. When new data is added, the damping factor is reinitialized and the Jacobian and error vector are recomputed.

To verify the correctness of the PoseLab framework, a test setup has been used to create linear movement of a checkerboard marker. The linearity is clearly seen in the plots of the camera position coordinates. The step size of the stepper motor derived from calculated positions are in agreement with the measured value. The step sizes derived from the experiments were in accordance with the measured step size. However, because the test bench was a low cost solution, the values did not perfectly match. There were also slight deviations from the linear movement. There may be a lot of reasons for this. One possible cause is the incorrect calibration of the distortion coefficients. The correctness is also verified by comparing the calculated poses with the ones determined by the Caltech toolbox.

5.2 Future work

This Master's thesis is the basis for a project that is trying to create an application-specific hardware solution to accurately calculate the camera pose in real-time. The PoseLab framework will serve as an experimental test bench to be able to quantitatively evaluate hardware architectural alternatives. Besides the possibility to add 2D-3D point correspondences to the algorithm whilst iterating, other hardware specific modification will be made.

In addition to further developing specialized algorithms, better verification of the calculated camera pose is needed. Here, a low cost test setup is used to create a linear movement. It was not possible to verify rotational angles and thus a test setup must be designed that can do this. Next to rotational angles, another desired upgrade is for the test bench to be able to validate the absolute pose of the camera. Currently, it is only possible to determine the relative motion with respect to a starting position. Next to the pose, other aspects of the camera-based pose estimation will need to be tested such as the influence of the resolution, the marker, the lightning conditions, etc. on the achieved accuracy.

When the specialized algorithm is completely verified, the software will serve as a blueprint to facilitate the creation of application specific pose estimation hardware architectures. It will be opted to first implement the algorithm on an FPGA as it is more flexible than an integrated circuit. Once the hardware design is completed, tests can be done to measure the improvements in the speed and accuracy of the pose estimation.

Bibliography

[1] R. Yu, T. Yang, J. Zheng, and X. Zhang, "Real-time camera pose estimation based on multiple planar markers", *Proceedings of the 5th International Conference on Image and Graphics, ICIG 2009*, pp. 640–645, 2010. DOI: 10.1109/ICIG.2009.93.

[2] C. Wang and X. Guo, "Feature-based RGB-D camera pose optimization for real-time 3D reconstruction", *Computational Visual Media*, vol. 3, no. 2, pp. 95–106, 2017, ISSN: 2096-0433. DOI: 10.1007/s41095-016-0072-2. [Online]. Available: http://link.springer.com/10.1007/s41095-016-0072-2.

[3] T. Liu, Y. Guo, S. Yang, S. Yin, and J. Zhu, "Monocular-Based 6-Degree of Freedom Pose Estimation Technology for Robotic Intelligent Grasping Systems", *Sensors*, vol. 17, no. 2, p. 334, 2017, ISSN: 1424-8220. DOI: 10.3390/s17020334. [Online]. Available: http://www.mdpi.com/1424-8220/17/2/334.

[4] R. Chin, *eDrawings for iOS with Augmented Reality*, 2013. [Online]. Available: http://blogs.solidworks.com/solidworksblog/2013/02/augmented-reality-in-edrawings.html (visited on 04/28/2018).

[5] B. Spice, *Robot's In-Hand Eye Maps Surroundings, Determines Hand's Location*, 2016. [Online]. Available: https://www.cmu.edu/news/stories/archives/2016/may/robot-hand-camera.html (visited on 04/28/2018).

[6] T. Malisiewicz, *The Future of Real-Time SLAM and Deep Learning vs SLAM*, 2016. [Online]. Available: http://www.computervisionblog.com/2016/01/why-slam-matters-future-of-real-time.html (visited on 04/28/2018).

[7] 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, ISSN: 07403194. DOI: 10.1002/mrm.24314.

[8] 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, e48088, 2012, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0048088. [Online]. Available: http://dx.plos.org/10.1371/journal.pone.0048088.

[9] L. Ramirez, *Corrective jaw surgery.* [Online]. Available: http://www.maxillofacialcentre.com/index.php/services/corrective-jaw-surgery (visited on 04/28/2018).

[10] Y. Sun, "Digital intermediate splint fabrication and image-guided maxillary positioning", PhD thesis, Hasselt University, 2013.

[11] J. Stals, "Pose estimation with a camera for orthogrnathic surgery planning", Master's thesis, Hasselt University, 2015.

[12] *Mobilization/positioning*. [Online]. Available: `https://www2.aofoundation.org/wps/portal/!ut/p/a0/04%7B%5C_%7DSj9CPykssy0xPLMnMz0vMAfGjzOKN%7B%5C_%7DA0M3D2DDbz9%7B%5C_%7DUMMDRyDXQ3dw9wMDAx8jfULsh0VAdAsNSU!/?approach=%7B%5C&%7Dbone=CMF%7B%5C&%7Dclassification=95b-Mandible,%20Sagittal%7B%5C&%7DcontentUrl=srg/95b/05-RedFix/P315-BSSO-Hunsuc/05%7B%5C_%7DMobilizationPosit` (visited on 04/29/2018).

[13] *Mock surgery and fabrication of splints*. [Online]. Available: `https://www2.aofoundation.org/wps/portal/surgerypopup?contentUrl=/srg/95b/05-RedFix/X300-SpecCond-PlanningOrthSurg/X300-06%7B%5C_%7DMockSurgandFabofSplints.jsp%7B%5C&%7DsoloState=precomp%7B%5C&%7Dtitle=%7B%5C&%7DLanguage=en` (visited on 04/28/2018).

[14] Z. Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6DOF pose estimation for textureless objects", *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 2441–2448, 2016, ISSN: 10504729. DOI: `10.1109/ICRA.2016.7487396`.

[15] *OpenCV Camera Calibration and 3D Reconstruction*, 2018. [Online]. Available: `https://docs.opencv.org/2.4/modules/calib3d/doc/camera%7B%5C_%7Dcalibration%7B%5C_%7Dand%7B%5C_%7D3d%7B%5C_%7Dreconstruction.html?highlight=calib%7B%5C#%7Dsolvepnp`.

[16] J.-Y. Bouguet, *Camera Calibration Toolbox for Matlab*, 2015. [Online]. Available: `http://www.vision.caltech.edu/bouguetj/calib%7B%5C_%7Ddoc/` (visited on 04/29/2018).

[17] T. Aerts, "Calibration and evaluation system for 3D camera systems", Master's thesis, Hasselt University, 2018.

[18] L. Ada, *Adafruit Motor Shield*, 2015. [Online]. Available: `https://learn.adafruit.com/adafruit-motor-shield/overview` (visited on 05/30/2018).

[19] 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, ISSN: 01628828. DOI: `10.1109/34.888718`. arXiv: `arXiv:1011.1669v3`.

[20] G. Bradski and A. Kaehler, "Camera Models and Calibration", in *Learning OpenCV*, 1st ed., Sebastopol: O'Reilly Media, 2008, pp. 370–403, ISBN: 978-1-4493-1465-1. DOI: `10.1109/MRA.2009.933612`. arXiv: `arXiv:1011.1669v3`. [Online]. Available: `http://shop.oreilly.com/product/0636920022497.do`.

[21] C. Stachniss, *Photogrammetry I - 15a - Camera Extrinsics and Intrinsics*, 2015. [Online]. Available: `https://www.youtube.com/watch?v=DX2GooBIESs%7B%5C&%7Dindex=28%7B%5C&%7Dlist=PLKEuOzA9RZ2WnWf7vj2hQEXRIeB8hpxHa` (visited on 05/28/2018).

[22] PAKSC, *Simple Pinhole Camera Science*, 2015. [Online]. Available: `https://paksc.org/pk/diy-projects/homemade-simple-pinhole-camera/`.

[23] R. Collins, “Lecture 12 : Camera Projection”, in, Penn State University, 2007. [Online]. Available: `http://www.cse.psu.edu/%7B~%7Drtc12/CSE486/lecture12.pdf`.

[24] C. Stachniss, *Photogrammetry I - 16b - DLT & Camera Calibration*, 2015. [Online]. Available: `https://www.youtube.com/watch?v=Ou9Uj75DJX0%7B%5C&%7Dlist=PLKEuOzA9RZ2WnWf7vj2hQEXRIeB8hpxHa%7B%5C&%7Dindex=31` (visited on 05/28/2018).

[25] Z. Wang, Z. Wang, and X. Xu, “Extraction of the corner of checkerboard image”, *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pp. 6783–6788, 2008. DOI: `10.1109/WCICA.2008.4593961`.

[26] Z. A. Hartley Richard, “Representations of rotation matrices”, in *Multiple View Geometry in computer vision*, 2nd ed., New York: Cambridge University Press, 2004, pp. 578–587.

[27] *Tutorial 17 : Rotations.* [Online]. Available: `http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/` (visited on 04/13/2018).

[28] 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.

[29] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi, “Revisiting the PnP problem: A fast, general and optimal solution”, *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2344–2351, 2013, ISSN: 1550-5499. DOI: `10.1109/ICCV.2013.291`.

[30] 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, ISBN: 9781457711022.

[31] K. Madsen, H. Nielsen, and O. Tingleff, “Methods for non-linear least squares problems”, Technical University of Denmark, Tech. Rep., 2004, pp. 18–30.

[32] Z. A. Hartley Richard, “Iterative Estimation Methods”, in *Multiple View Geometry in computer vision*, 2nd ed., New York: Cambridge University Press, 2004, pp. 597–602.

[33] P. Mittrapiyanuruk, “A Memo on How to Use the Levenberg-Marquardt Algorithm for Refining Camera Calibration Parameters”, Purdue University, Tech. Rep., pp. 5–10. [Online]. Available: `https://engineering.purdue.edu/kak/computervision/ECE661%7B%5C_%7DFall2012/homework/hw5%7B%5C_%7DLM%7B%5C_%7Dhandout.pdf`.

[34] S. Marschner, “CS3220 Lecture Notes : QR factorization and orthogonal transformations”, *Computing*, vol. 2, no. March, pp. 1–8, 2009.

[35] OpenCV, *Undistort.* [Online]. Available: `https://github.com/opencv/opencv/blob/master/modules/imgproc/src/undistort.cpp` (visited on 05/28/2018).

[36] P. Abeles, *Inverse Radial Distortion Formula.* [Online]. Available: `http://peterabeles.com/blog/?p=73` (visited on 05/28/2018).

[37] H. Douglas, *Backward substitution*, 2005. [Online]. Available: `https://ece.uwaterloo.ca/%7B~%7Dece204/howtos/backward/` (visited on 05/13/2018).

[38] H. B. Nielsen, “Damping parameter in marquardt’s method”, Technical University of Denmark, Lyngby, Tech. Rep., 1999, p. 7. [Online]. Available: `http://www2.imm.dtu.dk/documents/ftp/tr99/tr05%7B%5C_%7D99.pdf`.

Appendices

Appendix A: Pseudocode camera pose estimation algorithm

Algorithm 7 Camera pose estimation

```
function CAMERAPOSE($ImagePoints, WorldPoints, K, DistCoeffs, R_0, t_0$)
    $J_x, J_y \leftarrow SymbolicJacobian(K)$

    //Stopping criteria
    $\varepsilon_1 \leftarrow 10^{-8}$
    $\varepsilon_2 \leftarrow 10^{-8}$
    $i_{max} \leftarrow 500$

    //Initialize parameters
    $tx \leftarrow t_0(1)$
    $ty \leftarrow t_0(2)$
    $tz \leftarrow t_0(3)$

    $w \leftarrow AxisAngle(R_0)$
    $wx \leftarrow w(1)$
    $wy \leftarrow w(2)$
    $wz \leftarrow w(3)$

    //Add available 2D-3D correspondences to the data buffers
    $impBuffer \leftarrow$ undistorted 2D image coordinates
    $wpBuffer \leftarrow$ 3D world coordinates

    //Compute Jacobian, error vector and gradient
    $J \leftarrow CalculateJacobian(J_x, J_y, [wx; wy; wz; tx; ty; tz], wpBuffer)$
    $f \leftarrow CalculateError(K, AxisAngle([wx; wy; wz]), [tx; ty; tz], impBuffer, wpBuffer)$
    $g \leftarrow J^T \cdot f$
```

```
//Pose found?
**if** $\|\boldsymbol{g}\|_\infty \leq \varepsilon_1$ **and** $endOfFrame$ **then**
    **return** $AxisAngle([wx; wy; wz]), [tx; ty; tz]$
**end if**

//initialize $\lambda$ and $\upsilon$
$\lambda \leftarrow 10^{-8} \cdot max(diag(J^T J))$
$\upsilon \leftarrow 2$

$i \leftarrow 0$
//Iterate
**while** $i < i_{max}$ **and** $endOfFrame$ **do**
    $i \leftarrow i + 1$

    $h \leftarrow Step(J, \lambda, f)$
    **if** $\|h\| \leq \varepsilon_2(\| [wx; wy; wz; tx; ty; tz] \| + \varepsilon_2)$ **and** $endOfFrame$ **then**
        **return** $AxisAngle([wx; wy; wz]), [tx; ty; tz]$
    **else**
        //updated parameters
        $wx_{upd} \leftarrow wx + h(1); wy_{upd} \leftarrow wy + h(2); wz_{upd} \leftarrow wz + h(3);$
        $tx_{upd} \leftarrow tx + h(4); ty_{upd} \leftarrow ty + h(5); tz_{upd} \leftarrow tz + h(6);$
        $f_{upd} \leftarrow CalculateError(K, AxisAngle([wx_{upd}; wy_{upd}; wz_{upd}]), [tx_{upd}; ty_{upd}; tz_{upd}],$
                        $impBuffer, wpBuffer)$

        //Cost function
        $F \leftarrow \frac{1}{2} \cdot f^T \cdot f$
        $F_{upd} \leftarrow \frac{1}{2} \cdot f_{upd}^T \cdot f_{upd}$

        //gain
        $\varrho \leftarrow \frac{F - F_{upd}}{\frac{1}{2} \cdot h' \cdot (\lambda \cdot h - g)}$

        //Add new 2D-3D correspondences if available
        Add new undistorted 2D image coordinates to $impBuffer$
        Add new 3D world coordinates to $wpBuffer$

        **if** $\varrho > 0$ **then**
            //Accept parameters
            $wx \leftarrow wx_{upd}; wy \leftarrow wy_{upd}; wz \leftarrow wz_{upd};$
            $tx \leftarrow tx_{upd}; ty \leftarrow ty_{upd}; tz \leftarrow tz_{upd};$

            //Recalculate Jacobian, error vector and gradient
            $J \leftarrow CalculateJacobian(J_x, J_y, [wx; wy; wz; tx; ty; tz], wpBuffer)$
            $f \leftarrow CalculateError(K, AxisAngle([wx; wy; wz]), [tx; ty; tz],$
                        $impBuffer, wpBuffer)$
            $g \leftarrow J^T \cdot f$
```

```
                //Pose found?
                if $\|\boldsymbol{g}\|_\infty \le \varepsilon_1$ and $endOfFrame$ then
                    return $AxisAngle([wx; wy; wz]), [tx; ty; tz]$
                end if

                //update $\lambda$
                if Data was added then
                    $\lambda \leftarrow \lambda \cdot max(\left[\frac{1}{3}, 1 - (2 \cdot \varrho - 1)^3\right])$
                else
                    $\lambda \leftarrow 10^{-8} \cdot max(diag(J^T J))$
                end if
                $\upsilon \leftarrow 2$
            else
                if Data was added then
                    //Recalculate Jacobian, error vector and gradient
                    $J \leftarrow CalculateJacobian(J_x, J_y, [wx; wy; wz; tx; ty; tz], wpBuffer)$
                    $f \leftarrow CalculateError(K, AxisAngle([wx; wy; wz]), [tx; ty; tz],$
                                $impBuffer, wpBuffer)$
                    $g \leftarrow J^T \cdot f$

                    //Reinit $\lambda$
                    $\lambda \leftarrow 10^{-8} \cdot max(diag(J^T J))$
                    $\upsilon \leftarrow 2$
                else
                    //Increase $\lambda$
                    $\lambda \leftarrow \lambda \cdot \upsilon$
                    $\upsilon \leftarrow 2 \cdot \upsilon$
                end if
            end if
        end if
    end while
    return $AxisAngle([wx; wy; wz]), [tx; ty; tz]$
end function
```

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Novel Levenberg-Marquardt based methods for application-specific hardware enabled high-speed, high-accuracy, six degrees of freedom camera-based pose estimation

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Darcis, Michiel

Datum: **4/06/2018**