Tampere University

Peter Todorov

MULTI-CAMERA CALIBRATION

Calibration of Multi-camera Rig for Wide Field of View Light Field Capture

Faculty of Information Technology and Communication Sciences
Bachelor's Thesis
May 2019

ABSTRACT

Peter Todorov: Multi-camera Calibration
Bachelor's Thesis
Tampere University
Signal Processing and Machine Learning
May 2019
Examiners: Suren Vagharshakyan and Laura Goncalves Ribeiro

3-D display technology is under development, and it has not yet become common among people. In this project, we use light field display to create a multi-view image. This type of display makes it possible to visualize objects from different perspectives on the screen by changing the viewing angle. The display does not require additional equipment from the viewer, unlike the older 3-D displays where the viewer can feel the depth of the image by using the polarizing 3-D glasses. The aim of this study is not to get into the technology behind the light field displays but do research on how to create tools to provide proper content for light field displays. In more detailed, this study focuses on methods in multi-camera calibration where the multi-view image data has to be parametrized accurately to know where and how these images have been obtained. Such requirements need accurate calibration.

In general, camera calibration consists of color, focus, distortions and position calibration in which the focus in this thesis is on the position calibration between cameras. In one camera calibration, the goal is to remove the distortion caused by the camera lens and components to which errors have occurred during the manufacturing process. Firstly, the distortions caused by the lens will cause straight lines in the real world to be curved in the image. Secondly, imperfect parallelism between the camera lens and the image sensor gives the image projective impression. Thirdly, slight distortions can happen by the component errors which make the image skewed or stretched.

The calibration of multiple cameras must take into account the relative translations and rotations of the cameras. The unknown parameters describing the position and distortions of a camera are estimated by minimizing the error between the model and the actual measurements. The final result of the minimization is finally seen on the light field display where the image should not have significant artifacts. To achieve this result, accurate methods are required to calibrate the translations and rotations of the cameras.

The study provides tools for more accurate calibration and illustrates the less accurate pairwise calibration method. The calibration method can be seen in the image quality and on that basis it is required to use better methods of calibrating a multi-camera camera rig.

Keywords: camera calibration, calibration pattern, ChArUco

PREFACE

This thesis was done for the Laboratory of Signal Processing at Tampere University, and the project is done for the CIVIT laboratory. At the beginning of the thesis, I was rather uncertain of what is the purpose of the project, and how does the mathematics work. The topic was quite demanding, and a very deep jump to real computer vision after having only a few introductory courses in signal processing. In the end, I learned a lot and was able to get into the topic. I suppose this topic is highly beneficial for the work I start after the thesis.

I want to thank my Ph.D. student supervisors Suren Vagharshakyan and Laura Goncalves Ribeiro for teaching me in many areas in this field, and being very patient on things that I did not understand immediately. I also want to thank professor Joni Kämäräinen for setting up great deadlines for the thesis and giving suggestions for this thesis to make it more compact and finish it in time. Thank you for professor Atanas Gotchev for giving me an opportunity of continuing this project after finishing the thesis. It meets my interests and both of our benefits. For the language part, I want to thank English teacher Anni Karo who gave me unexpectedly detailed feedback of the language in the thesis. Her effort in giving feedback is highly recommended. I also want to thank my girlfriend and peers for setting up a competitive atmosphere during the studies.

In Tampere, 3rd May 2019

Peter Todorov

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

c_x	x coordinate of the principal point
c_y	y coordinate of the principal point
DLT	direct linear transformation
$\boldsymbol{E}$	essential matrix
$\boldsymbol{F}$	fundamental matrix
f	focal length in the world units
f_x	focal length divided by the size of a pixel in x direction, the length is in pixels
f_y	focal length divided by the size of a pixel in y direction, the length is in pixels
$\boldsymbol{I}$	identity matrix
$\boldsymbol{K}$	camera calibration matrix, intrinsic parameters
κ	kappa, radial distortion parameter
NumPy	a Python library for scientific computing
OpenCV	open source computer vision, a library of programming functions mainly aimed at real-time computer vision
o_x	x coordinate of the optical center
o_y	y coordinate of the optical center
$\boldsymbol{P}$	projection matrix
p_x	pixel length in x direction
p_y	pixel length in y direction
$\boldsymbol{R}$	rotation matrix, extrinsic parameters
$\boldsymbol{S}$	singular value matrix
s	skew term, intrinsic parameter
SVD	singular value decomposition
$\boldsymbol{t}$	translation vector, extrinsic parameters
θ	theta, an angle formed by the intersection of two curves in a plane
$\boldsymbol{U}$	left orthogonal matrix
$\boldsymbol{V}$	right orthogonal matrix
$\boldsymbol{W}$	orthogonal matrix

1 INTRODUCTION

Visual information from the real world is carried by the *light field*. It means that the light field is formed by light vectors consisting of light rays. A collection of light rays creates the visual information of the surroundings. [23] Many computational imaging operations can be done by capturing the light field properly. This study works on the visualization of multi-view images taken from different perspectives. The images are shown on a single light field display to create the feel of depth on a 2-D plane. This is one of the many methods of creating a hologram effect.

In this project, the light field is captured by a camera rig of 20 cameras. The multi-camera images are stacked together to form a 3-D cube. This cube can then be shown on the light field display to make an object look different when looking at it from different viewing angles.

First of all, in order to visualize multi-camera images and 3-D view of the captured scene, the relative positions of all the cameras must be known. Secondly, one needs to capture the scene at the same time instant, and this is done by synchronously controlling all the cameras' shutters. The third required step is to correct the optical distortions caused by the camera lenses and to equalize the colors in order to compensate for the eventual color deviations between the different camera sensors. These procedures are referred to as calibration. In this study, it was seen that with the manufacturer measures of the camera rig, the result was extremely poor and the light field image had many artifacts. This as a baseline proofs why more sophisticated calibration method – done by using computer vision – is required for the multi-camera rig.

The aim of this thesis is to create tools to develop multi-camera calibration. The experimental work was performed on the premises of the Centre for Immersive Visual Technologies (CIVIT). In the CIVIT laboratory, they have been able to take satisfactory light field images. These images were taken by a camera system in the robot arm. Unfortunately, the robot arm cannot be used for objects that are not stable, such as a human. That is why the light field image for an unstable object must be captured in a short time period. Multiple camera system where cameras are able to see the object from different perspectives will solve this problem.

This thesis is structured as follows. Chapter 2 goes through the theory behind the camera, and how it is modeled. It deals with the camera geometry of the model, and the different distortions caused by the lenses. Chapter 3 covers the calibration of a different number of

cameras. It will tell in more detailed what kind of steps the calibration procedure consists of, what kind of calibration objects can be used for different purposes, and how to get the information of cameras' relative positions. Chapter 4 describes the project steps, problems, and solutions for them. The results are covered in Chapter 5. It analyses the condition-dependent results of ChArUco checkerboard marker detection, and pair-wise multi-camera calibration errors. The final chapter, Chapter 6, covers the conclusions of the project. It presents the ideas for further development, introduces briefly the results, and gives suggestions for further studies.

2 CAMERA MODEL

Vision starts with detecting light from the world. Much of the light is absorbed to the surroundings but some of the light is reflected from the objects to our eyes. The geometry of this arrangement – the light rays traveling from the object, through the lens in our eye or camera, to the retina or imager – is important to model in computer vision applications. [22, p. 637]

In order to create the mathematical model of the camera, we need to define the components that affect the direction or focus of the light rays and form of the image. During the vision, the 3-D space points are projected to a 2-D plane, and finally, a 2-D image is formed. Between the 3-D points and 2-D plane, there are typically a lens or multiple lenses, camera aperture and the *image sensor* also called as *imager*, that converts the optical image to an electronic signal [30]. These together can create a complex camera model but we start with a commonly used, simple *pinhole camera model*. A *pinhole* is a tiny hole in an imaginary wall which passes only the light rays going through the pinhole. [22, p. 637] The model is covered in the following section.

2.1 Pinhole Camera Model

The simplest way to model a camera is to use a pinhole model which basics are shown in Figure 2.1. In the model, the center of projection is in the camera center which is the origin of the Cartesian coordinate system. By definition, the plane $Z = f$ is called the *image plane*. In the pinhole camera model, the point in the real world coordinates $\boldsymbol{X} = (X, Y, Z)$ is mapped to the point on the image plane where the line from the world coordinate to the center of projection cuts the image plane. This projection is illustrated in Figure 2.2. Based on similar triangles, the world coordinate point (X, Y, Z) can be mapped to the image plane point as $(fX/Z, fY/Z, f)$. Leaving the last image coordinate out, the coordinate transformation can be shown as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix} \tag{2.1}$$

that tells the central projection mapping from world coordinate system to image coordinate system. [16, pp. 153–154]

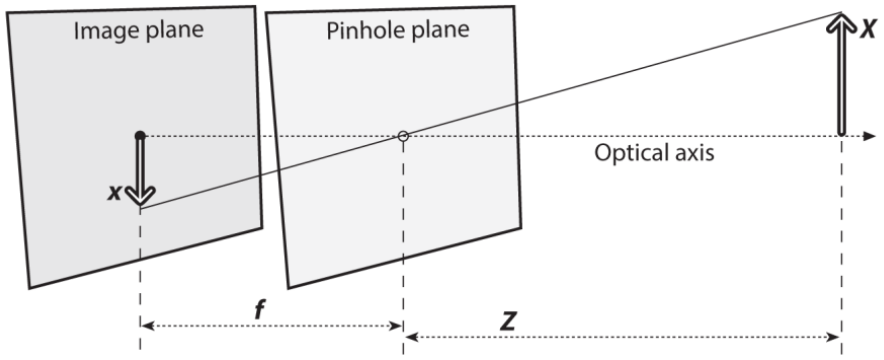

Figure 2.1. *Pinhole model. Light rays go through the pinhole aperture and the image is formed on to the image plane by the rays. One point in the world coordinate system can reach only one point or small area on the image plane so the image will be focused. [22, p. 639]*

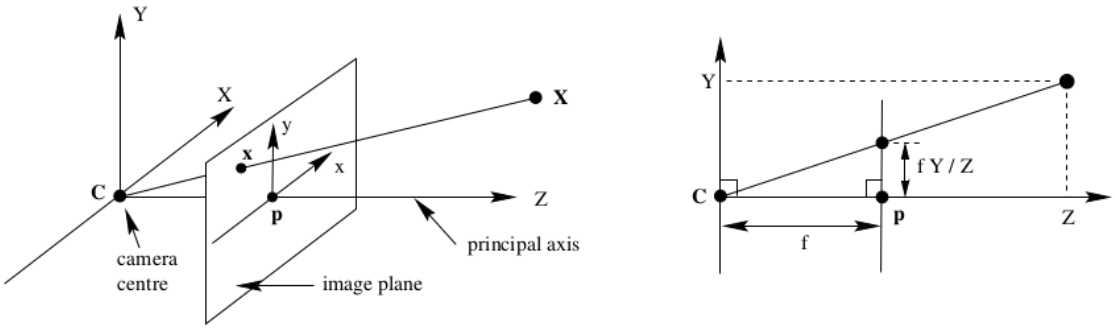

Figure 2.2. *Pinhole camera model. The point C is the camera center placed in the coordinate origin and p is the principal point. Image plane is placed in front of the coordinate origin. [16, p. 154]*

The projection is centered through the *camera center*, also known as *optical center*. The vertical line from the camera center perpendicular to the image plane is called the *principal axis*. The ray going on this line is the *principal ray*, and the point where it meets the image plane is called the *principal point*. The plane in parallel with the image plane through the camera center is called the *principal plane*. The distance *f* between the camera center and the image plane is called the *focal length*. Figure 2.2 shows how these map to the image. [16, p. 154]

2.2 Projective Geometry

The relation that maps the points in the world coordinates (X, Y, Z) to the image coordinates (X, Y) is called the *projective transform* [13, p. 424]. When working with such transforms, it is convenient to use the *homogeneous coordinates* instead of Cartesian coordinates. 3-D point coordinates in the world can be written using inhomogeneous coordinates $\boldsymbol{x} = (x, y, z) \, \epsilon \, \mathbb{R}^3$, or homogeneous coordinates $\widetilde{\boldsymbol{X}} = (\widetilde{X}, \widetilde{Y}, \widetilde{Z}, \widetilde{W}) \, \epsilon \, \mathbb{P}^3$. [37, p. 33] The transformation from inhomogeneous coordinates to homogeneous coordinates are done by adding the extra coordinate W so that the dimensionality will be inceased by

1. All the points, Euclidian and ideal, in the homogeneous coordinate system are treated equally in the projective plane. The homogeneous coordinate system is widely used in computer graphics because it makes projective and affine transformations as matrix multiplications in a convenient way. [4] Projection transform to homogeneous coordinates is shown in Figure 2.3.

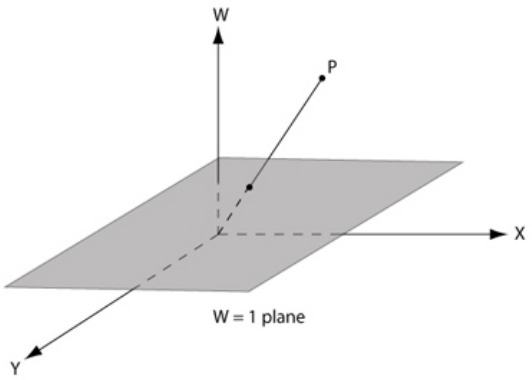

Figure 2.3. *Projective transform to homogeneous coordinates. Point $\boldsymbol{p}$ projected onto the $w = 1$ plane. [6]*

By representing the world and image coordinate points in homogeneous coordinates, the central projection is done by a linear mapping between the coordinates. Equation 2.1 can be continued, and be written as a matrix multiplication

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX \\ fY \\ 1 \end{bmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2}$$

where the matrix is diag(f, f, 1)[$\boldsymbol{I}|\boldsymbol{0}$] in which the notation [$\boldsymbol{I}|\boldsymbol{0}$] means a 3x3 diagonal indentity matrix plus a zero column vector. [16]

Changing the coordinates from the world to the image is done by transforming the world Cartesian coordinates into homogeneous coordinates $(X, Y, Z, 1)^T$. The image coordinate in homogeneous 3-vector $(X, Y, W)^T$ is obtained by multiplying the homogeneous world coordinate by the 3×4 homogeneous *camera projection matrix*. Equation 2.2 can then be expressed as

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X} \tag{2.3}$$

where $\boldsymbol{x}$ is the image coordinate point in homogeneous coordinates, $\boldsymbol{P}$ is the camera projection matrix, and $\boldsymbol{X}$ is the world coordinate point in homogeneous coordinates. [16, p. 154]

In Equation 2.1, the origin of image plane coordinates is assumed to be at the principal point. In most cases, it is close to the principal point, but the deviation must still be taken

into account in the coordinate projection as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} fX/Z + c_x \\ fY/Z + c_y \end{bmatrix} \tag{2.4}$$

where $(c_x, c_y)^T$ are the principal point coordinates. The camera central projection equation is now formed as

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX + Zc_x \\ fY + Zc_y \\ Z \end{bmatrix} = \begin{bmatrix} f & & c_x & 0 \\ & f & c_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.5}$$

where it can be seen that the principle point settles perfectly to the projection matrix. From Equation 2.5 the projection matrix can be separated as

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I}|\boldsymbol{0}]\boldsymbol{X} \tag{2.6}$$

where the $\boldsymbol{K}$ is defined as

$$\boldsymbol{K} = \begin{bmatrix} f & & c_x \\ & f & c_y \\ & & 1 \end{bmatrix} \tag{2.7}$$

and it is called the *camera calibration matrix*. [16, pp. 154–155]

In Equation 2.6, it is assumed that the location of the camera is at the origin of the Cartesian coordinate system, and the direction of the camera's principal axis is down along the z-axis. This coordinate system is called the *camera coordinate frame*. In space, points are expressed in terms of a different Cartesian coordinate frame which is known as the *world coordinate frame*. These two coordinate frames are related via rotation and translation. The rotation matrix is 3×3 matrix that includes the orientation related to x, y, and z-axis. In 3 dimensional space, as here the world coordinates are, the rotation matrices for each axis are defined as

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \tag{2.8}$$

$$\boldsymbol{R}_y = \begin{bmatrix} cos\theta_y & 0 & sin\theta_y \\ 0 & 1 & 0 \\ -sin\theta_y & 0 & cos\theta_y \end{bmatrix} \tag{2.9}$$

$$\boldsymbol{R}_z = \begin{bmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

and applying rotation over every axis, the rotation matrix will come to form

$$\begin{aligned} \boldsymbol{R} \\ &= \boldsymbol{R}_x \boldsymbol{R}_y \boldsymbol{R}_z \\ &= \begin{bmatrix} cos\theta_y cos\theta_z & -cos\theta_y sin\theta_z & sin\theta_y \\ cos\theta_x sin\theta_z + cos\theta_z sin\theta_x sin\theta_y & cos\theta_x cos\theta_z - sin\theta_x sin\theta_y sin\theta_z & -cos\theta_y sin\theta_x \\ sin\theta_x sin\theta_z - cos\theta_x cos\theta_z sin\theta_y & cos\theta_z sin\theta_x + cos\theta_x sin\theta_y sin\theta_z & cos\theta_x cos\theta_y \end{bmatrix} \end{aligned} \tag{2.11}$$

where the θ_x is rotation on x-axis, θ_y is rotation on y-axis and θ_z is rotation on z-axis. [16, pp. 155–157]

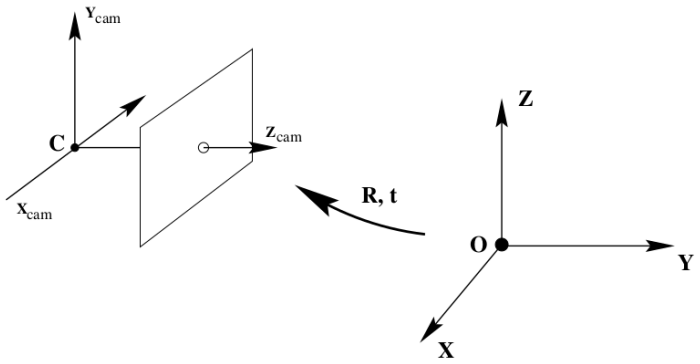

Figure 2.4. *By applying rotation $\boldsymbol{R}$ and translation $\boldsymbol{t}$ to world coordinate frame, it can be known how it is related to the camera coordinate frame. [16, p. 156]*

The translation is the offset from one coordinate system origin to another. In 3-D space it is expressed as 3 dimensional vector (x, y, z). By knowing rotation and translation between the world and camera coordinate frames it is possible to know exactly how they are related to each other. [16, p. 155] The relation has been shown in Figure 2.4.

The transformation from world coordinates to image coordinates can now be done as $\boldsymbol{x} = \boldsymbol{RX} + \boldsymbol{t}$. Now the camera projection matrix comes to form

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}] \tag{2.12}$$

where $\boldsymbol{t} = -\boldsymbol{RC}$ in which $\boldsymbol{C}$ is the camera center in world coordinates. [16, p. 156]

In the image sensor, it is possible to have non-square pixels. So in that sense, the pixels could have a different size in x and y-direction. The camera calibration matrix can now

be shown as

$$\boldsymbol{K} = \begin{bmatrix} f_x & & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \tag{2.13}$$

where f_x is the focal length f divided by the length of a pixel in x direction $f_x = f/p_x$, and f_y is defined in a corresponging way $f_y = f/p_y$. [16, pp. 156–157]

To add one more change to the camera calibration matrix it becomes to the following form

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \tag{2.14}$$

where s is the *skew* term [16, p. 157]. The skew is covered in Section 2.3.

2.3 Lens Distortion

The camera has usually a lens to create the pinhole structure and gather more light rays from a point in the world rather than one light ray as in pinhole model. The light rays from one point in the world are then focused on one point on the image plane. The lens improves the lighting and reduces need of exposure. [12, pp. 159–187] Lens light rays collection and focusing are shown in Figure 2.5.

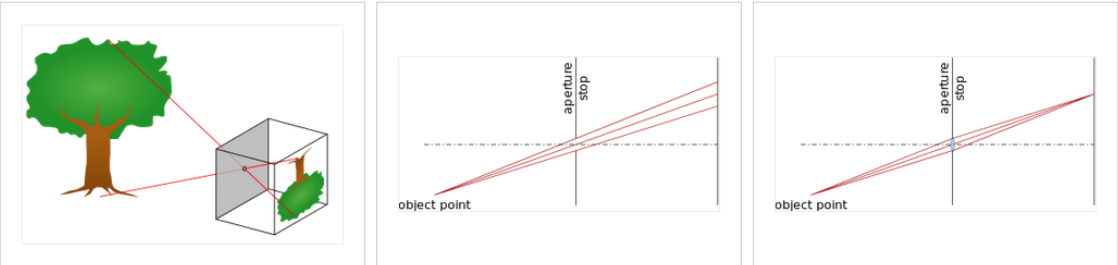

Figure 2.5. *On the left, only one light ray from one point goes through the pinhole, and the image is dark. In the middle, large pinhole let many light rays from a point go through the hole, and spread to a large area on the image plane. Then the image becomes blurry. On the right, many light rays go through the lens, and the rays are focused to one point on the image plane. The image is getting more light and it is sharp. [5]*

The models in the previous section assume that the cameras do not have any distortion. It means that the straight lines in the world are also straight lines in the image. Unfortunately, many lenses, especially wide-angle lenses have considerable *radial distortion*. Radial distortion creates curves to the image, and the closer you look the image edges, the stronger are the curves. [37, p. 58] Strong radial distortion can be made by the fish-eye lens which is strongly curved. Fish-eye effect is shown in Figure 2.6.

There are 2 types of radial distortion: *barrel* distortion and *pincushion* distortion. In

Figure 2.6. *Image taken from the streets by a full frame fish-eye lens. Radial distortion is strong especially near to the edges. [14]*

barrel distortion, the coordinates in an image are displayed away from the image center. Pincushion distortion, on the other hand, displays the coordinates in an image towards the image center. [37, pp. 58–59] These two radial distortion types are shown in Figure 2.7.

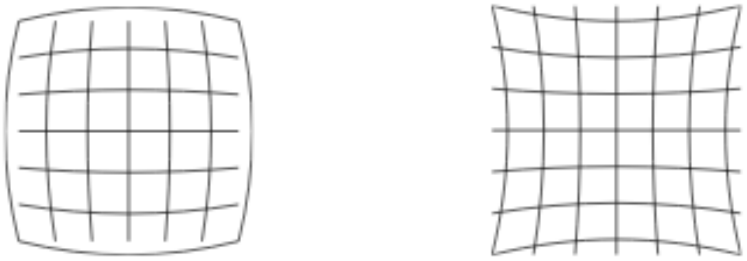

Figure 2.7. *Radial distortion effects. On the left the barrel distortion causes the lines bulge out away from the image center. On the right the pincushion distortion causes the lines bend towards the image center. [7, p. 30]*

Considering (x_c, y_c) to be the image coordinates obtained by the perspective division, and shifting by the optical center (o_x, o_y) before scaling by the focal length f, the coordinates can be calculated as

$$\begin{aligned} x_c &= \frac{\boldsymbol{r}_x \cdot \boldsymbol{x} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{x} + t_z} \\ y_c &= \frac{\boldsymbol{r}_y \cdot \boldsymbol{x} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{x} + t_z} \end{aligned} \tag{2.15}$$

where the $\boldsymbol{r}_x$, $\boldsymbol{r}_y$ and $\boldsymbol{r}_z$ are the three rows of rotation matrix $\boldsymbol{R}$, $\boldsymbol{x}$ is the world coordinate point, and t_x, t_y and t_z are the translation components of translation vector $\boldsymbol{t}$. Improving the radial undistortion, it is possible to add low-order polynomials, and that is what the

simplest radial distortion models use. It is shown in Equation 2.16

$$\begin{aligned} \hat{x}_c &= x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \\ \hat{y}_c &= y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \end{aligned} \tag{2.16}$$

where $r_c^2 = x_c^2 + y_c^2$, and κ_1 and κ_2 are called *radial distortion parameters*. [37, pp. 58–59]

Radial distortion might be significant depending on the lens. Another common distortion is called *tangential distortion*. This distortion is not caused only by the lens but from the assembly process of the camera as a whole. In an ideal situation, the camera lens and the camera sensor are perfectly in parallel. During the camera manufacturing process, the sensor and lens may not be assembled in parallel which causes tangential distortion. The effect of tangential distortion causes the image to have a perspective view. Tangential distortion is shown in Figure 2.8. [22, p. 647]

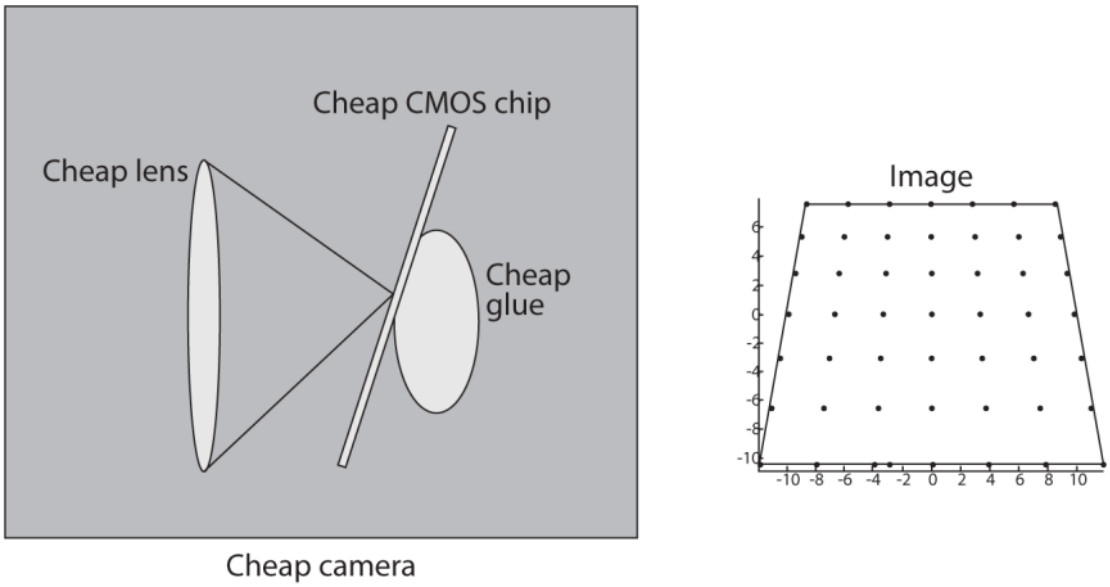

Figure 2.8. *Tangential distortion in a camera. The lens is not fully parallel to the image plane which may be due to the glue on the back of the camera. Resulting image showed on the right. [22, p. 647]*

Skew happens in some image sensors, but in most normal cameras it is zero [16, p. 157]. Skew means that the image sensor contains imperfections that cause the x- and y-axes of the image not to be perpendicular. It makes the image have affine form which is shown in Figure 2.9 [40]. The skew coefficient is defined by the angle between the x and y pixel axes [38].

Decentering distortion is also caused by lenses but it is not covered in this thesis. More information can be found from [3].

Figure 2.9. *Applying affine transformation to such grid, it becomes skewed. Some image sensors might have skewness that is taken into account in a calibration. [1]*

2.4 Intrinsic and Extrinsic Parameters

Camera parameters are generally divided into intrinsic and extrinsic parameters. Extrinsic parameters do the transform from object coordinates to a camera centered coordinate frame which basically means that applying rotation and translation, the coordinate frame can be changed. [18] In Section 2.2, it is described how the rotation and translation are used for the coordinate frame transform.

In calibration, extrinsic parameters are used to express an arbitrary object point P at location (X_i, Y_i, Z_i) in image coordinates to transform it to camera coordinates (x_i, y_i, z_i) [18]. By multiplying the object point P with the rotation matrix and adding translation to it, the coordinate frame transform is achieved. This projection transform from the world coordinate system to the image coordinate system is covered in more detailed in Section 2.2.

The camera intrinsic parameters include the parameters in camera calibration matrix, which are the focal lengths f_x and f_y, principal point parameters c_x and c_y, and the skew s which is assumed to be 0 in many cases such as in the calibration functions of OpenCV library [18][22, p. 646]. The origin of the image coordinate system is in the upper left corner of the image array so the principal point coordinate values (c_x, c_y) are typically close to half of the image width and height in pixels [18].

The pinhole camera model is a very simple approximation of the projection for a real camera. It enables a straightforward mathematical relationship between the world and image coordinates. However, it does not give high accuracy, and that is why a more complex camera model is required. Nonetheless, the pinhole is used as a basis with some extensions. The extensions are corrections for the systematically distorted image coordinates. As mentioned in Section 2.3, the radial distortion is the most common lens distortion, and because of its significant effect, the radial distortion parameters are added to the model as an extension. Mostly, two radial distortion coefficients are able to remove the distortion well enough. [18]

Accurate calibration is achieved by a proper camera model that combines the pinhole

model with the correction for radial and tangential distortion coefficients. These distortion coefficients combined with intrinsic parameters are called as *physical camera parameters* since they have an obvious physical meaning. As mentioned previously, the camera calibration procedure is to define optimal values for these parameters. [18]

3 CAMERA CALIBRATION

Camera calibration is required phase in 3-D computer vision for extracting metric information from 2-D images [40]. In this study, the calibration is a part of creating a light field image that can be shown on a light field display. Camera calibration, in the context of 3-D machine vision, is the mechanism of determining the internal camera geometric and optical characteristics which are so-called intrinsic parameters. Another determination is the 3-D position and orientation of the camera frame relative to a certain world coordinate system. Those are known as the extrinsic parameters. Often, the accuracy of the camera calibration is strongly related to the total performance of the machine vision system. [18]

The classic approach for camera calibration solves the problem of minimizing a nonlinear error function to get accurate estimations for parameters in camera matrix [18]. In stereo or multi-camera calibration, the relations, rotations and translations, between cameras are required to compute. After the calibration is done, the unknown parameters are found in the mathematical model which was fitted to the real measurements. [15] In terms of light field display application, the multi-camera calibration result for the light field image is strongly related to the accuracy of estimated parameters.

Multi-camera systems also require color calibration between each camera. Color calibration is not covered in this thesis but more information can be found from [24]. This thesis explains the principles of commonly used Zhang's [40] approaches to camera calibration. The implementation of Zhang's method is available in Matlab toolboxes and in OpenCV. Zhang's method relies on correspondences between image coordinate points and known world reference points. Those correspondences can be obtained many ways of which one way is through the use of 2-D calibration targets.

3.1 Calibration Pattern

In the mathematical camera model, there are multiple unknown parameters that have to be solved. For this problem, we need external information or measurements of the scene to determine the unknown parameters in the model. The usage of calibration patterns is a way to obtain easily interpretable and more accurate measurements. The 3-D coordinate measurements of the space are taken from the calibration pattern. [37, p. 327] There are multiple different calibration patterns of which three are covered in this thesis.

If a calibration pattern is not available, it is also possible to perform calibration simultaneously with structure and pose recovery by using, for example, vanishing points. This is

known as *self-calibration*. However, to get accurate results such an approach requires a large amount of imagery. [37, p. 328] The 3-D coordinates of the object points are also included in the set of unknown parameters in self-calibration. However, in this thesis, the calibration procedure is performed with a known object. [18]

3.1.1 Chessboard

Very common calibration pattern is *chessboard* pattern where the corner coordinates (in world coordinates) in between white and black squares are known. This kind of board is called the *checkerboard* [11]. The corner coordinates of projection on 2-D image plane are easy to detect accurately [40]. Figure 3.1 shows the chessboard checkerboard.

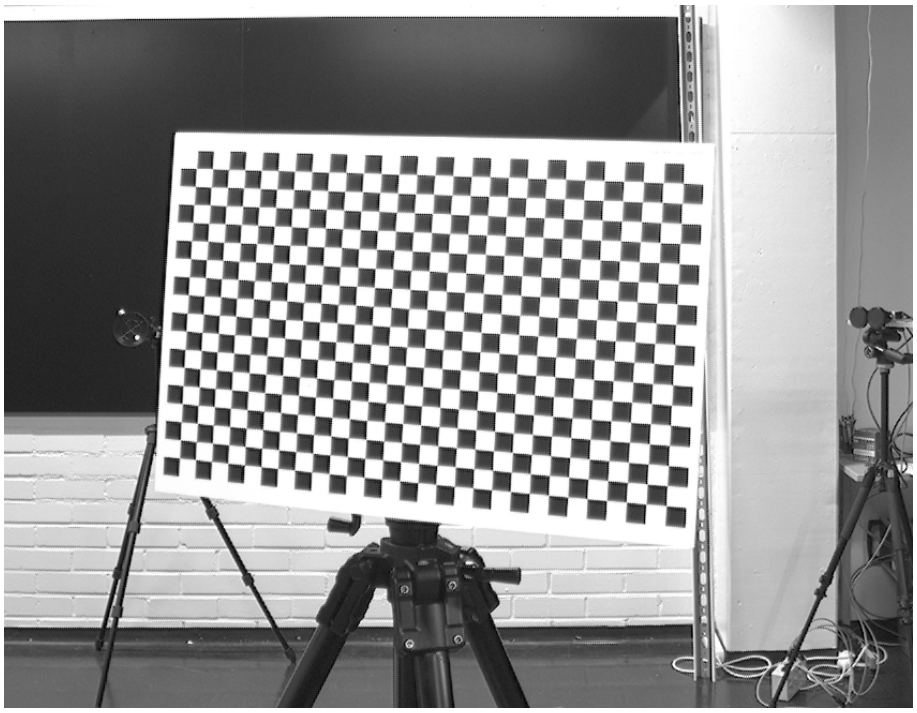

Figure 3.1. *Chessboard calibration pattern.*

Using chessboard in the calibration, it requires that all the corners are visible and detected. Otherwise, if not all the corners were detected, it would be hard for a computer to know which corners were detected and define the 3-D coordinates for them. It would require human labeling work to identify the detected corners or additional information from the calibration pattern as in ChArUco. Because of the requirement of fully visible chessboard pattern, chessboard pattern also limits the possible number of positions in the calibration procedure. In stereo camera case, the chessboard must always been placed perfectly in the overlapping areas of two cameras' views. In case of wide multi-camera rig, the overlapping area between the outermost cameras is small which reduces the chessboard positions.

3.1.2 ChArUco

Another type of calibration pattern is called ChArUco calibration pattern. It combines ArUco [9] and chessboard pattern where the white squares of a chessboard pattern are filled with *markers*. The marker is a QR-code [39] which tells a unique ID (identification number) for the marker. Because the detection of corners in the checkerboard is the intention, the ChArUco board makes it possible to define an ID for every corner on the board. The ID for a corner enables to know which corners are detected from the checkerboard. It is useful in many situations such as in pose estimation and in the case of a partially visible checkerboard. [20] ChArUco board is shown in Figure 3.2.

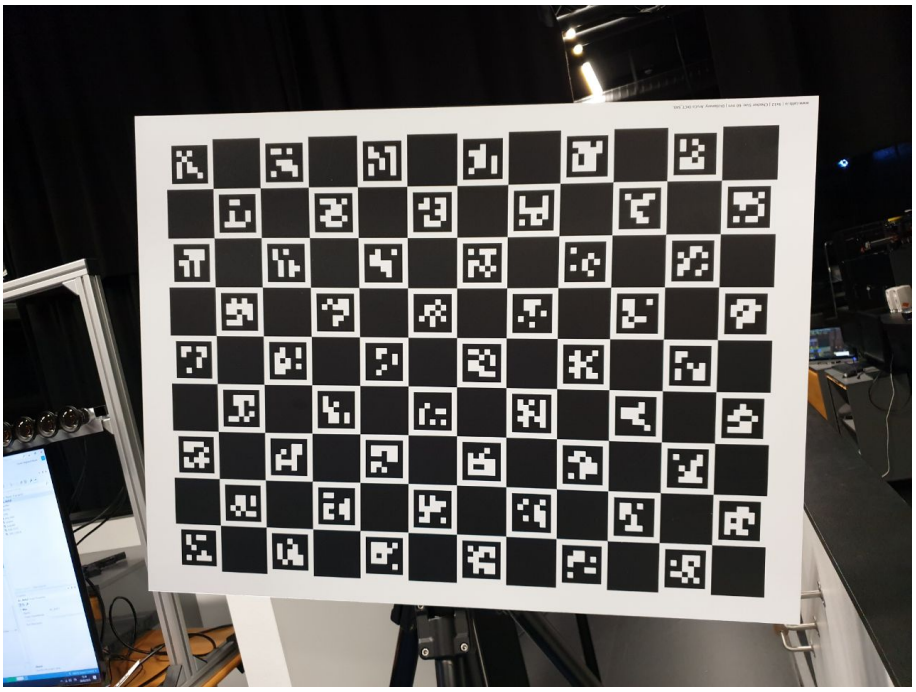

Figure 3.2. *ChArUco calibration pattern. The white squares are filled with the markers to identify the corners. Each corner is surrounded by two markers.*

Using ChArUco detection in the code, one must define the ChArUco dictionary which tells the resolution of a marker, and the maximum number of IDs that are used in the detection [20]. In this study, the marker resolution of the checkerboard is 5×5 and the number of markers is 54. The predefined dictionaries are discretized so used dictionary is defined to be `DICT_5x5_100`.

One disadvantage of ChArUco is the difficulty of marker detection since the markers are small and require the image to be focused. It also requires enough pixels for the markers so that they would be clear enough for detection. During the study, this was noticed to be one major problem when gathering data.

3.1.3 Cicle-grid

The third covered calibration pattern is circle-grid where the 3-D points are detected from the centers of the circles. A circle-grid pattern is shown in Figure 3.3. Circle, as a calibration pattern, enables the point detection in cases where the image is not totally focused. However, it has disadvantages in asymmetric projection where the circle center have to be corrected.

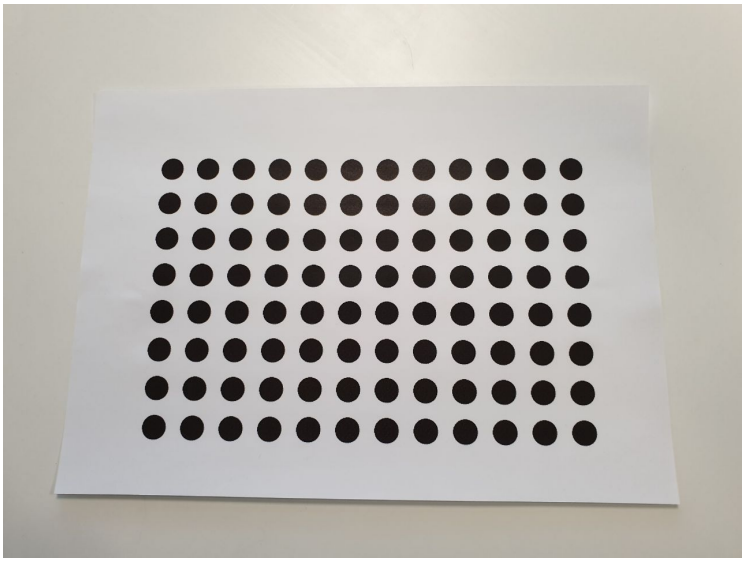

Figure 3.3. *Circle-grid calibration pattern.*

Two- and three-dimensional objects, which are not coplanar with the image plane, are distorted. This is also true for any arbitrarily shaped objects. [18] There has been written whole paper of correcting asymmetric circle center projection, more detailed mathematics information can be found from [25]. The asymmetric projection is shown in Figure 3.4.

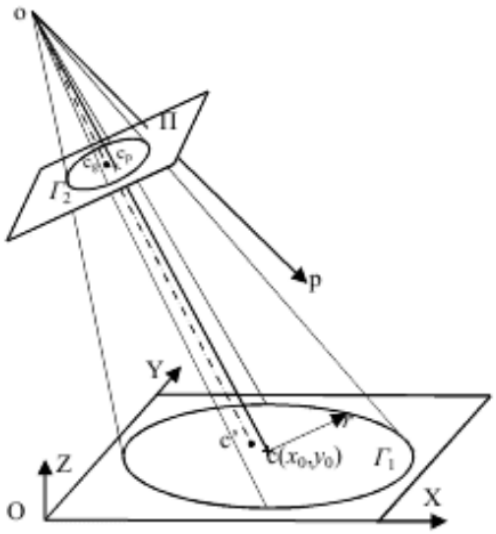

Figure 3.4. *A circle is mapped on the image plane as an ellipse if it is not coplanar with the image plane. As a result, the center of the circle is shifted from the center of the ellipse. [29]*

After correcting the asymmetric projection, the camera parameters are recomputed. The obtained parameters are not optimal in the sense of least squares but the iterations for these values are not needed since the remaining error is so small. [18] This is one calibration step and it corrects the systematic bias in all parameters caused by the asymmetric projection [25].

3.2 Reprojection Error

The error metrics defined for estimation are done from the image points. It is usually done by computing the Euclidian distance between the projected image coordinate point, and the measured image coordinate point. [16, p. 95]

In the estimation process, the total error function will be minimized in the least squares fashion [18]. The process is iterated where the reprojection error is computed during the iterations, and the final target is to get the minimum reprojection error.

3.3 Single Camera Calibration

Calibration of a single camera – finding the individual cameras' intrinsic parameters – is the easiest step to start, and it is also gathered in multi-camera calibration [19]. The following sections use the explained projection transform and lens distortions in the camera calibration. Each calibration step covered here increases the accuracy of the camera calibration in a way to create minimal reprojection error. [18].

The 3-D world coordinate points that are used in the calibration are measured from the calibration patterns covered in Section 3.1. To gain better accuracy in the calibration, there has to be taken 10 to 20 images of the calibration pattern [10]. These images must be focused and taken from different rotations, distances and they should cover a wide area in the field of view of a camera [37, p. 340]. Also, a higher number of object points in the calibration object will provide more data for the calibration and result in better accuracy. The corresponding 2-D image coordinate points for the world coordinate points are measured where the object points appear in the image.

3.3.1 Linear Parameter Estimation

The radial and tangential distortion components are nonlinear. By removing them, the equations become linear. The pinhole camera model is used as a basis in *direct linear transformation* (DLT), which ignores the nonlinearity caused by the radial and tangential distortions. In DLT, the calibration procedure is divided into two steps. In the first step, the linear transformation from the world coordinates (X_i, Y_i, Z_i) to image coordinates (u_i, v_i) is computed. A homogeneous 3 $\times$ 4 matrix representation for projection matrix $\boldsymbol{P}$ – as in

Equation 2.12 – does the projection as follows:

$$\begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{3.1}$$

where the image coordinates are in homogeneous form, and can be transformed to inhomogeneous by dividing with the term w_i. [18]

The parameters $p_{11}, ..., p_{34}$ of the DLT matrix can be solved by eliminating w_i. If $\boldsymbol{A}$ and $\boldsymbol{p}$ are defined as

$$\boldsymbol{A} = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1u_1 & -Y_1u_1 & -Z_1u_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1v_1 & -Y_1v_1 & -Z_1v_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -X_iu_i & -Y_iu_i & -Z_iu_i & -u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -X_iv_i & -Y_iv_i & -Z_iv_i & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -X_Nu_N & -Y_Nu_N & -Z_Nu_N & -u_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -X_Nv_N & -Y_Nv_N & -Z_Nv_N & -v_N \end{bmatrix}$$

$$\boldsymbol{p} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{21} & p_{22} & p_{23} & p_{24} & p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}^T,$$

the following matrix equation for N control points satisfies the equation

$$\boldsymbol{Ap} = \boldsymbol{0}. \tag{3.2}$$

By replacing the correct image coordinate values (u_i, v_i) with observed values from projection transform (U_i, V_i), the parameters in p can be estimated by using least squares. The optimized parameters $p_{11}, ..., p_{34}$ do not have any physical meaning in terms of intrinsic and extrinsic parameters but despite of it this optimization procedure can be called as implicit camera calibration stage. [18][35]

DLT is used to provide proper initial parameters that can be refined in the nonlinear estimation. A method of estimating intrinsic and extrinsic parameters from projection matrix $\boldsymbol{P}$ has been proposed by Melen [28]. In the method, a set of 11 physical camera parameters are extracted from the DLT projection matrix. The decomposition is as follows:

$$\boldsymbol{P} = \lambda \boldsymbol{V}^{-1} \boldsymbol{B}^{-1} \boldsymbol{F} \boldsymbol{M} \boldsymbol{T} \tag{3.3}$$

where λ is an overall scaling factor and the matrices $\boldsymbol{M}$ and $\boldsymbol{T}$ define the rotation and translation from the world coordinate system to the camera coordinate system. The ma-

trices $\boldsymbol{V}$, $\boldsymbol{B}$ and $\boldsymbol{F}$ contain the focal length, principal point and coefficients for the linear distortion. [18]

3.3.2 Nonlinear Estimation

Linear parameter estimation is efficient but it has disadvantages. One disadvantage is that the lens distortion effects are not generally corrected. There are presented a few solutions for this, for example in [33], but the second disadvantage is not that easy to be fixed. Due to the simplicity and nonlinear algorithm that the linear estimation provides, the constraints in the intermediate parameters are not considered. In the presence of noise, the accuracy is poor, and nonlinearity is needed in the calibration. [18]

When the noise is added to the image, the best estimate for the camera parameters can be achieved by minimizing the residual between the model and N observations (U_i, V_i), where $i = 0, ..., N-1$. The noise can be considered to be white Gaussian noise, and in that sense, the objective function is expressed as a sum of squared residuals

$$F = \sum_{i=0}^{N-1} (U_i - u_i)^2 + \sum_{i=0}^{N-1} (V_i - v_i)^2 \tag{3.4}$$

where the N is the number of object points in the image, (U_i, V_i) is the projected point based on the intrinsic and extrinsic parameters, and (u_i, v_i) is the ground truth that is the target of the optimization. [18]

The least squares estimation based on the maximum-likelihood criterion is used to minimize Equation 3.4 [40]. The simultaneous estimation of the parameters is obtained by applying an iterative algorithm for the nonlinear camera model. The fastest convergence for this problem has been shown to be the *Levenberg-Marquardt optimization method*. Despite the fast convergence, the method could stick in the local minimum without the proper initial parameter values, and causes the calibration to be inaccurate. In this step, the DLT comes to need for making the initial values for the optimization. With these initial parameters, the global minimum of the Equation 3.4 is usually obtained after a few iterations. [18]

Research shows that two coefficients for both radial and tangential distortion are typically enough [17]. The total number of estimated intrinsic parameters are then eight. Singularity limits the number of parameters that can be estimated from a single view, and that is why more views are required to compute all the intrinsic parameters. The number of extrinsic parameters depends on the number of different camera views, but for a single view, the number of extrinsic parameters is six. [18]

One last step to increase the accuracy is to do back projection from the image coordinates to world coordinates. By doing this we can gain a very small increase in accuracy. The mathematical notations for back projection are covered in [18].

3.4 Stereo Calibration

The process of computing the geometrical relationship between the two cameras in space is called stereo calibration. As discussed in the single-camera case, the relationship between the camera and world coordinate frames is defined by the intrinsic and extrinsic parameters. In stereo calibration, the rotation and translation are considered between camera and calibration patterns as well as between the two cameras whose relative position is fixed. [2, p. 21] The new unknown parameters come from the relative position.

To estimate the relationship between two cameras, we need corresponding points taken from two perspectives by the cameras. The corresponding points in stereo images can be related by the *fundamental matrix*. The fundamental matrix $\boldsymbol{F}$ and a corresponding point $\boldsymbol{x}$ define an *epipolar line* on which the corresponding point $\boldsymbol{x'}$ on the other image must lie. The *epipolar geometry* is the intrinsic projective geometry between two views. [16, pp. 239–240] Figure 3.5 shows how the epipolar line is defined in epipolar geometry.

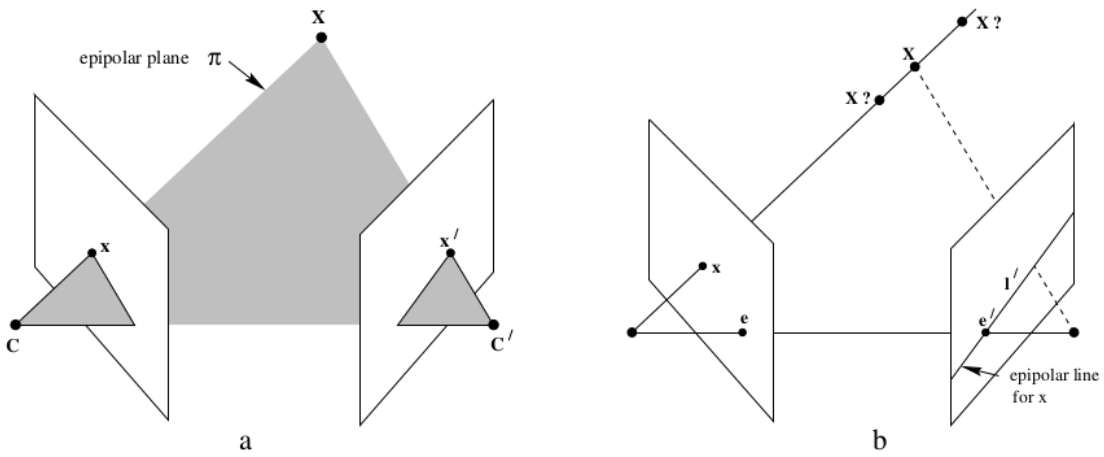

Figure 3.5. *Epipolar geometry. Camera centers $\boldsymbol{C}$ and $\boldsymbol{C'}$, and the object point $\boldsymbol{X}$ define the epipolar plane π. The plane defines the epipolar line and the corresponding point $\boldsymbol{x'}$ must lie on the epipolar line $\boldsymbol{l'}$. [16, p. 240]*

The fundamental matrix encapsulates intrinsic geometry. It is a 3 $\times$ 3 matrix of rank 2. [16, p. 239] It is defined by the equation

$$\boldsymbol{x'}^T \boldsymbol{F} \boldsymbol{x} = 0 \tag{3.5}$$

where $\boldsymbol{x}$ and $\boldsymbol{x'}$ are the corresponding points in stereo images. Detailed mathematical computation for fundamental matrix is done in [16, p. 279].

Another describing matrix for the relationship between two cameras is the *essential matrix*. It relates points to each other in the world coordinates, not in the image coordinates. The difference between the fundamental matrix and essential matrix is that the essential matrix $\boldsymbol{E}$ is totally geometrical and does not know anything about imagers. It relates the location, in the world coordinates for the corresponding point P which is seen by the left camera to the location of the same point as seen by the right camera. The fundamental

matrix $\boldsymbol{F}$ relates the points in image coordinates on the image plane of one camera to the points on the image plane of the other camera. The essential matrix can be represented as

$$\boldsymbol{E} = [\boldsymbol{t}]_\times \boldsymbol{R} = \boldsymbol{R}[\boldsymbol{R}^T \boldsymbol{t}]_\times \tag{3.6}$$

where $[\boldsymbol{t}]_\times$ is the matrix representation of the cross product with $\boldsymbol{t}$. The essential matrix is also represented in normalized image coordinates respectively

$$\boldsymbol{y'}^T \boldsymbol{E} \boldsymbol{y} = 0 \tag{3.7}$$

where the $\boldsymbol{y}$ and $\boldsymbol{y'}$ are the corresponding normalized image coordinates in stereo images. The relationship between fundamental matrix and essential matrix is

$$\boldsymbol{E} = \boldsymbol{K'}^T \boldsymbol{F} \boldsymbol{K} \tag{3.8}$$

where the $\boldsymbol{K'}$ and $\boldsymbol{K}$ are the stereo cameras intrinsic matrices. More information can be found from [16, pp. 239–308].

The essential and fundamental matrices make it possible to calculate the rotation and translation between the cameras. The parameters in the matrices are then optimized in the same manner as in single camera calibration by calculating the residual error and minimizing it with the least squares. The rotation and translation between cameras are computed by using *singular value decomposition* (SVD) with the following form

$$\boldsymbol{E} = \boldsymbol{U} \boldsymbol{S} \boldsymbol{V}^T \tag{3.9}$$

where the $\boldsymbol{U}$ is the 3×3 left orthogonal matrix and the $\boldsymbol{V}$ is the 3×3 right orthogonal matrix. The resulting 3×3 singular-value matrix $\boldsymbol{S}$ contains only two non-zero and equal singular values according to the properties of the essential matrix. [16, pp. 257–258]

The SVD of $\boldsymbol{E}$ gives the extrinsic parameters of the second camera with respect to the first camera [2, p. 22]. In the case of fundamental matrix, a projective ambiguity exists, and the camera matrices may be retrieved from the essential matrix up to scale and a four-fold ambiguity. This means that the ambiguity makes there four possible solutions. [16, p. 258] The two are for rotation and two for translation [2, p. 22]. They can be determined as

$$\begin{aligned} \boldsymbol{R}_1 &= \boldsymbol{U} \boldsymbol{W} \boldsymbol{V}^T \\ \boldsymbol{R}_2 &= \boldsymbol{U} \boldsymbol{W}^T \boldsymbol{V}^T \\ \boldsymbol{t}_1 &= +\boldsymbol{u}_3 \\ \boldsymbol{t}_2 &= -\boldsymbol{u}_3 \end{aligned} \tag{3.10}$$

where $\boldsymbol{u}_3$ is the third column of $\boldsymbol{U}$ and $\boldsymbol{W}$ is the following orthogonal matrix

$$\boldsymbol{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

More information and mathematical proofs can be found from [16, pp. 258–259].

Because stereo camera system has the relative position as a new unknown parameter in addition to single camera calibration, the single camera calibration can be extended to find the relative position. The standard approach is to calibrate two cameras independently, and then calibrate only the relative pose between them. The measurements from one camera can improve the calibration of the other camera which may not lead to the optimal solution. Moreover, the independent depth camera calibration may require a high precision 3-D calibration object that can be avoided using joint calibration. [19]

3.5 Multi-Camera Calibration

Calibration for multiple cameras means that the relationships – the new unknown parameters, rotations and translations – between all cameras are estimated. Good estimation accuracy is gained when multiple cameras are calibrated simultaneously in order to determine the relative geometry between cameras [27, p. 27]. Computing intrinsic parameters stays the same, and can be done individually for each camera as in stereo-calibration. However, accurate extrinsic computation is still challenging. [2, p. 23]

One and easy way to calibrate multiple cameras is pair-wise stereo calibration. The pair-wise multi-camera calibration does stereo calibration for adjacent camera pairs. As a result, the calibration gives the relations between the cameras pairs. The results of this method are covered in the next chapter but as mentioned before, the calibration should be done simultaneously. However, this method provides good initial estimates for better calibration methods. New techniques for multi-camera calibration have been proposed in [8][21][36]. The pair-wise calibration provides coarse values for these methods after which the values are then iteratively refined [2, pp. 23–24].

4 EXPERIMENT

The aim of the project was introduced in Chapter 1, and this chapter describes the actual project. The project started by detecting the markers in ChArUco checkerboard. More commonly used chessboard calibration pattern is not optimal for this project because the views of all the cameras in the rig do not overlap with each other's views at small distances.

The camera calibration application in Matlab works for chessboard checkerboard patterns. The application finds the corners of the checkerboard from the image but it rejects those images from the calibration part where the found number of corners is not consistent. In the case of this project, it means that it would be difficult to take the images with chessboard calibration pattern, and only a few of the images would pass to the calibration part. The solution for this is to use a ChArUco board which gives IDs for the found corners, and that is how the calibration does not need to see all the available corners in the checkerboard.

In this study, the ChArUco corner detection was done in Python [32], and the library that was used is OpenCV [31]. Once the image coordinates are detected they are transferred to Matlab [26] which is then used for the estimation of intrinsic and extrinsic parameters.

4.1 ChArUco Corner Detection

Corner detection was done by using OpenCV 4.0.0 in Python 3.7.2. OpenCV provides functions that have ChArUco marker detection, and the corner detection for the corners between the found markers. The final result of the detection can be verified by drawing the corners to the image, and visually check whether the corners have been detected correctly. During the process, few problems were encountered due to camera hardware, and Python to Matlab conversion. These problems are described here as well as the scripts that help the process of taking images.

The hardware problem was a poor resolution, and the condition problem was a poor lighting. In certain situations, they caused zero percent marker detection rate (the number of detected corners divided by the maximum number of corners). In the cases where the distance was too high between the camera and the ChArUco board, it was visually checked from the images that the markers were not clear enough for recognition. When the distance was suitable, and the markers were clear for the human eye, they were still not recognized. It was later noticed to be caused by poor lighting. When the lighting was

improved, the corner detection rate increased. For time-saving and debugging purpose, I created three scripts to help the process of getting satisfying images. The scripts are real-time marker/corner detection for a video stream, corner drawing for still image, and computation of detection rate for a set of images.

Python can be used from Matlab in many cases but Matlab has some limitations to Python. In this project, it was noticed that Matlab (version R2018b) does not support the NumPy library in Python version 3.7. Because of this constraint, and to avoid any compatibility problems, the detection of image coordinates is done separately with the calibration. The image coordinates from ChArUco board are read in Python and the results are written to *.mat* files. In Matlab, the image coordinates are then read, and used in the calibration.

The corner detection has been created for a few different cases due to the errors that were encountered during the project. The real-time marker detection has been done to see how well the markers are detected with different distances and angles. The Program A.1 shows the general frame how to detect the corners: which functions to call, what parameters to give, and how to draw corners. The constant parameters that are always defined for ChArUco board in every script are covered in Table 4.1.

Table 4.1. *ChArUco checkerboard specific parameters needed in corner detection.*

Parameter	Description
CORNERS_X	The number of checkerboard corners in x direction
CORNERS_Y	The number of checkerboard corners in y direction
MARKER_DICT	Marker dictionary, depends on the marker resolution and the maximum number of needed marker IDs
SQUARE_LENGTH	The side length of a square in ChArUco checkerboard in millimeters
MARKER_LENGTH	The side length of a marker in ChArUco checkerboard in millimeters

The drawing of the corners to still image has been done to see which corners were detected. It enables zooming to see how well the detection was in detailed. The image can be given as an argument in the command line. The program call is done in the following way

```
$ python draw_corners_still_image.py path/to/the/image.jpg
```

where the path to the image is given as an argument. The code of the corners drawing in still image is provided in B.1.

Computing the detection rate for a set of images is informative, and helps to see fast how good images were taken in the capturing process. The command line interface is similar to the still image case. The path to the images or many paths of the images can be given as arguments for the code. The call is as follows

```
$ python corner_detection_rate.py path/*.jpg another/path/*.jpg
```

where the number of arguments is not limited. The code is provided in C.1.

The number of detected image coordinate points is not fixed. This is why Python *dict* is used for the writing, and the reading format in Matlab is *cell* structure. Running the code is similar to previous ones:

```
$ python write_corners_to_file.py path/*.jpg another/path/*.jpg
```

Writing the image coordinates to files is provided in D.1. In Matlab, the coordinates are read to matrix which is easier format than cell. Matrix also enables targeting the matching extrinsic parameters with the image by using indexing. The code is provided in E.1.

All of the Python codes above use the one function that finds the corners of ChArUco board. The function `getCharucoCorners` takes as parameters one image or the image name, and the parameters defined in Table 4.1. At first, the function creates a ChArUco board object that is defined by the parameters in Table 4.1. Secondly, the markers are detected. In this step, the marker resolution must be set correctly. The maximum number of markers can be set to maximum that the dictionary offers but it is unnecessary to have a too big dictionary. The third step is to interpolate the corners between detected markers. It has been done by calculating the corresponding *homography* between the ChArUco plane and the ChArUco image projection. The homography is only performed using the closest markers of each ChArUco corner to reduce the effect of distortion. [31] The code of detecting ChArUco corners is below in the Program 4.1.

```
1  def getCharucoCorners(image, corners_x, corners_y, marker_dict,
2                        square_length, marker_length):
3
4      # Define marker dictionary
5      dictionary = cv2.aruco.getPredefinedDictionary(marker_dict)
6
7      # Define Charuco board
8      board_squares_x = corners_x + 1
9      board_squares_y = corners_y + 1
10     board = cv2.aruco.CharucoBoard_create(board_squares_x,
11                                           board_squares_y,
12                                           square_length/1000,
13                                           marker_length/1000,
14                                           dictionary)
15
16     # Read image if image name is given
17     if (type(image) == str):
18         image = cv2.imread(image)
19
20     # Detect markers
21     (markers, ids, _) = cv2.aruco.detectMarkers(image,
22                                                 dictionary)
```

```
23      charuco_corners = np.array([])
24      charuco_ids = np.array([])
25
26      # At least one marker is detected
27      if ids is not None:
28          # Interpolate corners between markers
29          (_, c_corners, c_ids) = \
30              cv2.aruco.interpolateCornersCharuco(markers,
31                                                  ids,
32                                                  image,
33                                                  board)
34
35          # At least one corner is detected
36          if c_ids is not None:
37              charuco_corners = c_corners[:, 0]
38              charuco_ids = c_ids[:, 0]
39
40      return charuco_corners, charuco_ids
```

Program 4.1. *Compute ChArUco corners with OpenCV.*

The OpenCV's drawing function `cv2.aruco.drawDetectedCornersCharuco` takes the corners as a parameter in shape of `(number_of_detected_corners, 1, 2)` and IDs in shape of `(number_of_detected_corners, 1)`. Corners and IDs are needed without the extra dimension, so that is why a wrapper has been created for this. It is provided in F.1.

4.2 Camera Parameters Estimation

The simultaneous estimation of intrinsic and extrinsic parameters was started but due to the timing of Bachelor's thesis, it was not finished. The main principle is to estimate all the camera parameters simultaneously computing the reprojection error between the estimated corner coordinates and the detected ChArUco corners during the iterations. Nonlinear least squares estimation is used for the iteration. However, pair-wise camera calibration was done using Matlab Camera Calibration Toolbox to evaluate how good estimate it can provide.

Pair-wise calibration using the Toolboxes in Matlab is a straightforward process, and only images must be given for the calibration application. The error in pair-wise multi-camera calibration can be seen by the simple case of three cameras. Considering the case of cameras 1, 2 and 3. From single camera calibration, the camera center can be calculated from the extrinsic parameters $\boldsymbol{R}$ and $\boldsymbol{t}$. Because the translation error calculated between the camera centers $\boldsymbol{C}_1$, $\boldsymbol{C}_2$ and $\boldsymbol{C}_3$ is easy to imagine so we can only take that into consideration.

We can formulate the relation between the rotation matrix $\boldsymbol{R}$ and the camera's orientation $\boldsymbol{R}_c$ with respect to the world coordinate axes. The relations are formulated as follows:

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{C} \\ \boldsymbol{0} & 1 \end{bmatrix}^{-1} \\
&= \left[\begin{bmatrix} \boldsymbol{I} & \boldsymbol{C} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix} \right]^{-1} \\
&= \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{I}_c & \boldsymbol{C} \\ \boldsymbol{0} & 1 \end{bmatrix}^{-1} \\
&= \begin{bmatrix} \boldsymbol{R}_c^T & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{C} \\ \boldsymbol{0} & 1 \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{R}_c^T & -\boldsymbol{R}_c^T \boldsymbol{C} \\ \boldsymbol{0} & 1 \end{bmatrix}
\end{aligned}$$

from where we can obtain the following relations:

$$\begin{aligned}
\boldsymbol{R} &= \boldsymbol{R}_c^T \\
\boldsymbol{t} &= -\boldsymbol{R}\boldsymbol{C}
\end{aligned}$$

and finally the camera center $\boldsymbol{C}$ can be defined to be

$$\boldsymbol{C} = -\boldsymbol{R}^T \boldsymbol{t}$$

where the rotation matrix is orthogonal, and the transpose of it can be used instead of the inverse. [34] For better accuracy, the camera center is calculated from every calibration image and then the camera center is determined to be the average over all the calculated camera center coordinates. The translation between 2 camera centers is then defined as

$$\boldsymbol{t}_{mn} = \boldsymbol{C}_n - \boldsymbol{C}_m$$

and the translations between the 3 camera pairs are then $\boldsymbol{t}_{12}$, $\boldsymbol{t}_{23}$ and $\boldsymbol{t}_{13}$. If the pair-wise calibration would be accurate, the equation $\boldsymbol{t}_{13} = \boldsymbol{t}_{12} + \boldsymbol{t}_{23}$ should satisfy.

5 RESULTS

Here, the results present the problems, solutions, and accuracies for the experiment covered in Chapter 4. The problem in ChArUco corner detection was noticed to be condition dependency, and the solution provided shows how the conditions are improved to gain a higher corner detection rate. The pair-wise stereo calibration estimates the cameras' relative poses in the calibration method and compares them with multiple other stereo calibration estimations.

5.1 ChArUco Corner Detection Performance

The ChArUco marker detection worked well with lower resolution images taken in real time with a web camera. At first, the detection did not work with the images that were taken by the cameras in the camera rig in this project. It was noticed that the images were not good enough for detecting any markers. I did some post-processing with Gimp2 to improve the edges in the images but it was not enough to make the code detect the markers. Later on, it was noticed that bringing the checkerboard closer to the cameras and decreasing the shutter speed, the code was able to detect about half of the visible markers from the image. Despite the increase in the detection rate, the quality became poor due to overexposing. An example of the overexposed image is seen in Figure 5.1.

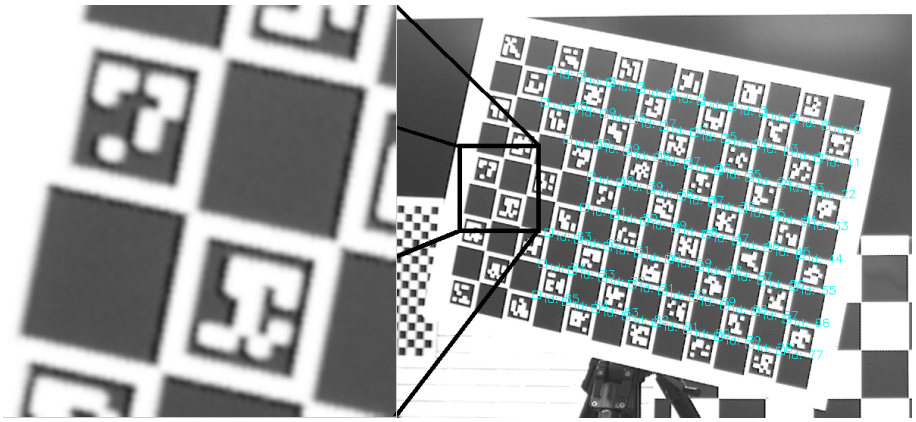

Figure 5.1. *Detecting corners from the ChArUco board was achieved by decreasing shutter speed. The corners are blurry, and a single corner has a gap. Markers are also unclear.*

It is seen in Figure 5.1 that the quality is not satisfied. Because of the gap in a corner, the corner detection could have an error of many pixels. Figure 5.2 shows the images

that have visually clearer markers but are little dark, and the OpenCV marker detection cannot detect any markers in 1000 images that were taken in similar conditions.

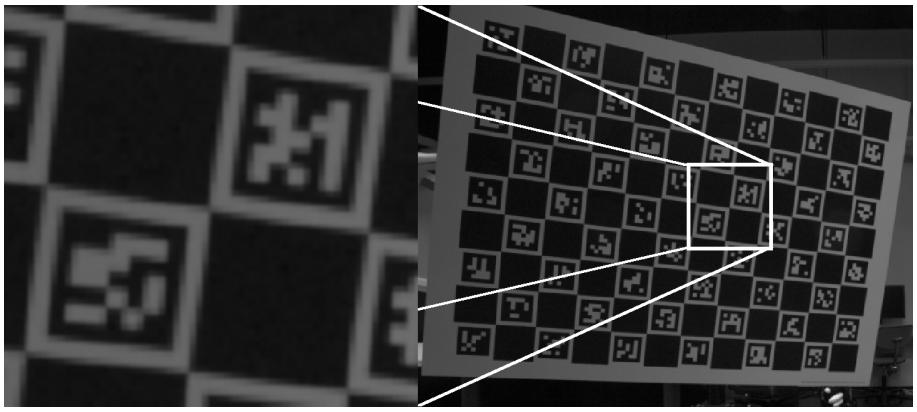

Figure 5.2. *Shutter speed is increased, and the markers look visually clearer. The checkerboard is close to the camera. Marker detection cannot detect any marker.*

The image in Figure 5.3 is post-processed by using *bilinear interpolation* and *Laplacian filter* and changing brightness and contrast of the image. However, the marker detection could not find any marker even if the black and white are stronger as in the image in Figure 5.1.

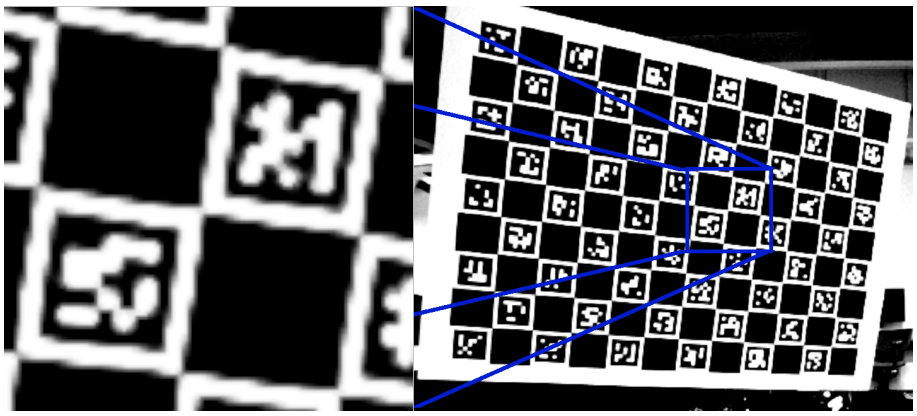

Figure 5.3. *Post-process done for the dark image. Marker detection cannot detect any marker.*

In order to take images where the desired corners can be detected, one needs to consider the resolution, lighting, shutter speed, aperture, the distance of the checkerboard from cameras, and larger squares in the checkerboard. In this project, the resolution is in its maximum of 1920 $\times$ 1200 and it sets the boundaries. In the ChArUco board, the starting step is to detect the markers. Because the resolution is fixed, the other parameters have to be set that the markers can be detected.

First of all, the lighting could be increased easily by having more powerful lights. If the lighting cannot be increased, then the shutter speed will have to compensate for it. The second is to bring the checkerboard closer to the camera. In that case, the aperture must be set so that the checkerboard looks sharp. If these do not help the detection,

then the checkerboard could be changed. Bigger markers or smaller marker resolution increases the size of the small squares in a marker. If these steps do not satisfy, then the marker detection code in OpenCV could be improved. Otherwise, having other calibration pattern or higher resolution cameras could be considerable. Figure 5.4 shows the result of having better lighting conditions where more light is directed to the ChArUco board. All the corners were detected at small distances for less than 10 images that were taken, and the markers were visually clearer as well. The markers were detected in longer distances which was not achieved by low light or overexposure. The detection rate for few images was noticed to be about 90 percent in such long distances as in Figure 5.5.

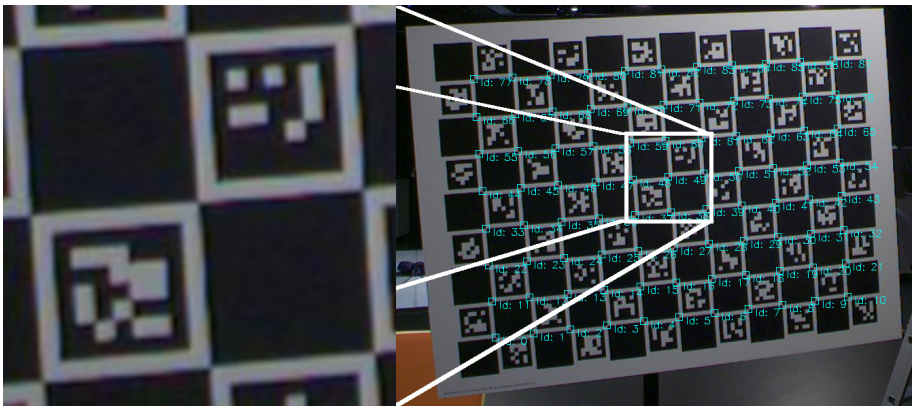

Figure 5.4. *Marker detection works with more light directed to the board. All the markers are detected and the edges are visually clearer.*

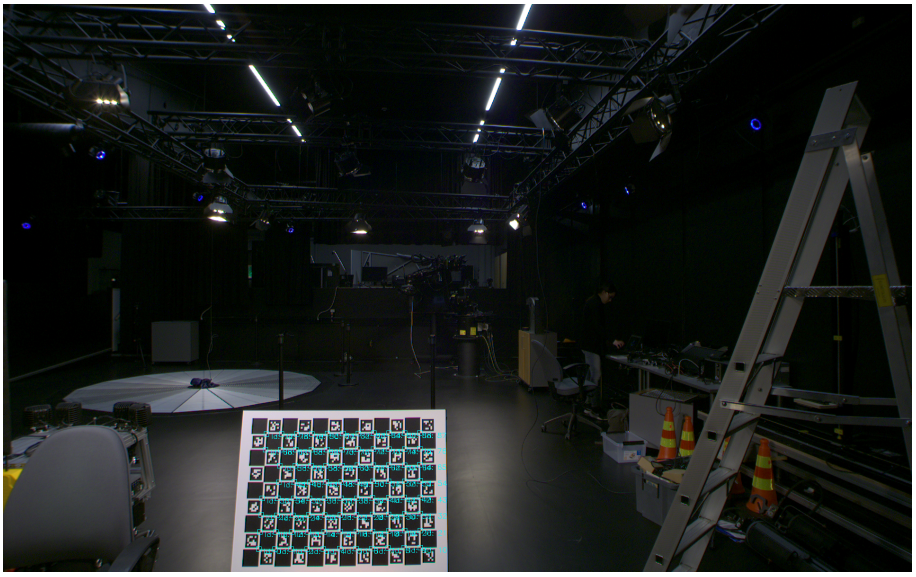

Figure 5.5. *ChArUco corner detection works for fairly long distance between camera and the board with more light directed to the board. All the markers except one are detected. It means in this case that 86 out of 88 corners are detected.*

The overall corner detection – and visual quality – for ChArUco board was noticed to improve significantly by having more light. The lack of high-resolution cameras was compensated by the directed light.

5.2 Camera Parameter Estimation Accuracy

The simultaneous camera parameters estimation for multiple cameras has very high re-projection error which is why pair-wise calibration result is shown here. A single stereo calibration resulted in 0.2 pixel reprojection error on average for every image and corners. From the multi-camera calibration, the desired reprojection error is also close to that level what it was in the stereo calibration case. Figure 5.6 shows the reprojection error from one pair of stereo calibration.

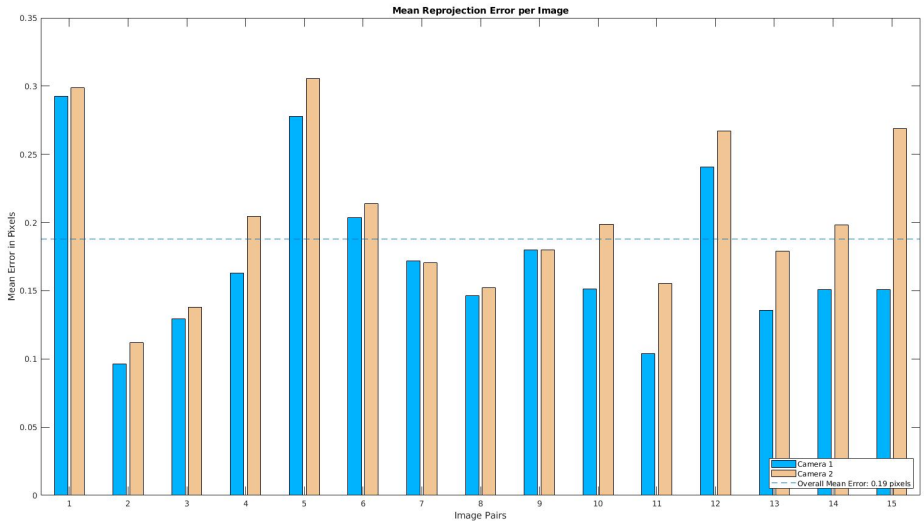

Figure 5.6. *Stereo calibration mean reprojection error per image. The average error of every image is less than 0.2 pixels.*

The calibration pattern for this pair-wise calibration is chessboard instead of ChArUco. The calibration was done for 20 pairs: $1-2, 2-3, ..., 19-20$. The translation vectors between camera centers were computed for the mentioned pair-wise calibrations, and for the camera pairs: $1-n, n=3, ..., 20$. This means that the translation error has been computed for example for camera pair $1-3$ by comparing the distance between the pair $1-3$ to the distance between the camera pairs $1-2$ and $2-3$. All the camera pairs had 12 to 15 calibration images from where all the corresponding corners were found. To make sure how the errors were computed, here are a few examples of the translation errors between camera pairs

$$\begin{aligned}
\boldsymbol{e}_{13} &= \boldsymbol{t}_{13} - (\boldsymbol{t}_{12} + \boldsymbol{t}_{23}) \\
\boldsymbol{e}_{14} &= \boldsymbol{t}_{14} - (\boldsymbol{t}_{12} + \boldsymbol{t}_{23} + \boldsymbol{t}_{34}) \\
\boldsymbol{e}_{15} &= \boldsymbol{t}_{15} - (\boldsymbol{t}_{12} + \boldsymbol{t}_{23} + \boldsymbol{t}_{34} + \boldsymbol{t}_{45})
\end{aligned}$$

where $\boldsymbol{t}_{mn} = (\Delta x_{mn}, \Delta y_{mn}, \Delta z_{mn})$ is the translation vector between the camera centers $\boldsymbol{C}_m$ and $\boldsymbol{C}_n$. The total distance errors are then computed by the Euclidian distance of the error vectors. The distance and translation errors are shown in Table 5.1. The table has also relative error computed by dividing the Euclidian distance error with the distance

between the camera pair $1 - n$.

The stereo calibrations resulted always approximately 0.2 pixel reprojection error which is something that is desired from the calibration. However, the pair-wise calibration for multiple cameras resulted in much higher error in the estimation of the cameras' positions. From these results, it can be said that this calibration method – by alone – does not provide desirable accuracy.

Table 5.1. *Pair-wise camera calibration translation errors. The Euclidian distance is computed from the translation error vector.*

Camera pair	**Translation error (mm)**			**Euclidian distance (mm)**	**Relative error (%)**
	x	y	z		
1–3	0.1250	0.2589	0.6619	0.7216	0.4223
1–4	-0.0052	0.5714	0.1322	0.5865	0.2295
1–5	-0.1727	0.5515	-0.6597	0.8770	0.2577
1–6	-0.0487	0.3787	-0.2861	0.4771	0.1123
1–7	-0.4140	-0.0427	-4.7568	4.7750	0.9357
1–8	-0.3949	0.3680	-0.4150	0.6809	0.1145
1–9	-0.5073	-0.9412	-0.4595	1.1638	0.1712
1–10	-0.7556	-0.5832	-0.5664	1.1099	0.1452
1–11	-0.4483	0.7343	-0.0896	0.8650	0.1017
1–12	-0.7386	-1.1734	0.1134	1.3911	0.1488
1–13	-0.8197	-1.4132	-9.4045	9.5453	0.9355
1–14	-1.6854	-0.0423	-3.8854	4.2354	0.3834
1–15	-1.1457	0.8476	0.1707	1.4353	0.1206
1–16	-1.4839	-1.5267	1.2548	2.4713	0.1939
1–17	-2.1753	-2.2384	-0.2934	3.1350	0.2307
1–18	-1.6641	-1.1881	-0.6170	2.1358	0.1478
1–19	-1.5476	-0.3863	-0.6332	1.7162	0.1121
1–20	-2.5888	-2.5303	6.0790	7.0752	0.4384

The error from pair-wise calibration was thought to be cumulating by increasing the number of cameras but it is not seen clearly in these results. There may be other factors that caused the absolute and relative errors to increase with the higher number of cameras. The translation errors in millimeters are shown in Figure 5.7 and the relative errors are shown in Figure 5.8.

The translation error is high. The manufacturer's measures of the camera's positions in the camera rig were millimeters and the errors pair-wise stereo calibration gave, have only slightly higher accuracy. The cameras are mounted to straight row on the aluminum bar which means that there are no high differences between camera positions in y-axis and z-axis. However, the highest peaks in Figure 5.7 are from the situation where the pair-wise calibration between camera pair $1 - n$ gave a high difference between the camera positions in y and z directions. One thing that affects the accuracy of stereo calibration

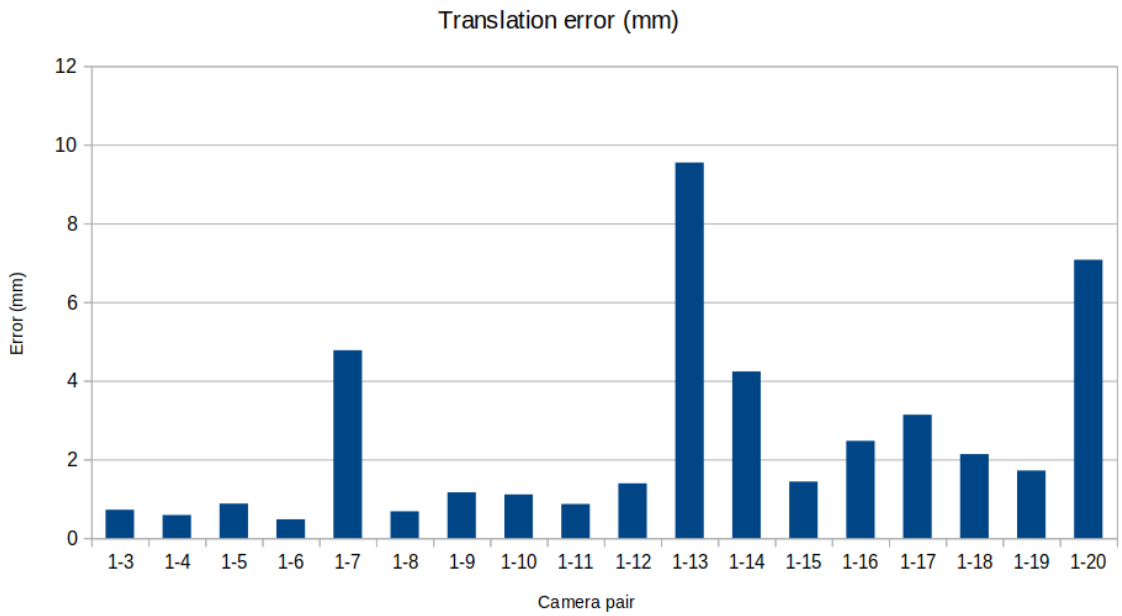

Figure 5.7. *Pair-wise calibration errors in millimeters with increasing number of cameras.*

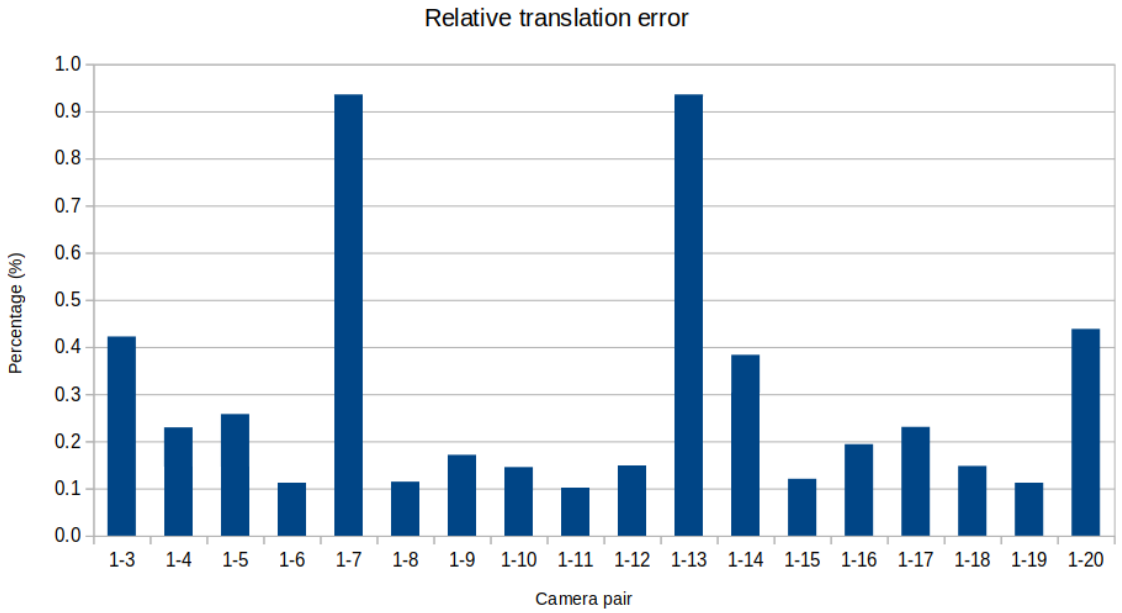

Figure 5.8. *Relative pair-wise calibration errors with increasing number of cameras.*

between camera pairs $1-n$ is the lack of widely positioned checkerboard in the calibration images. The cameras are focused between 2 and 3 meters so the overlapping area for the long-distance camera centers will become small. Even if the number of used calibration images is recommended for the accurate result – more than 10 images for every camera pair – the checkerboard was positioned almost in the same place, to the small overlapping area. The problem is illustrated in Figure 5.9.

The relative pair-wise calibration errors in Figure 5.8 have almost the same relations between the other calibration errors as in the absolute error values in Figure 5.7. The only major difference is that the shorter the translations are between the camera centers,

the higher the relative calibration error is. It is due to the fact that the smaller the distance with the cameras, the smaller is the angle between the views. Decent error in depth in the world causes an only small error in the images and the error is not then optimized. With a longer distance between cameras, the same decent depth error will be noticed better and it affects the optimization of the camera parameters. This just means that the longer the distance is with the cameras, the more accurate is the measure of depth.

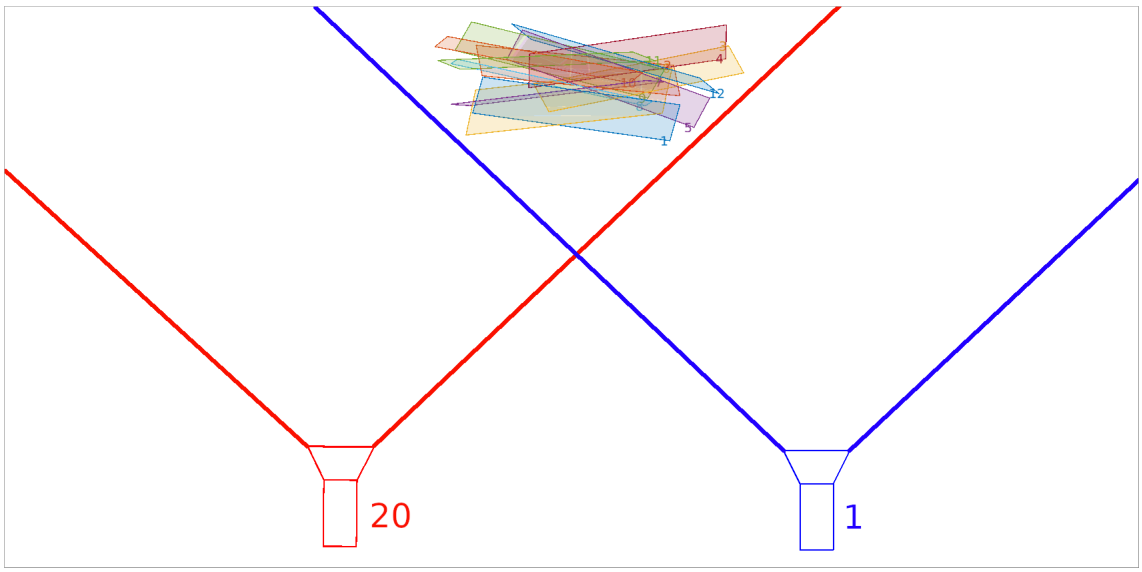

Figure 5.9. *Cameras 1 and 20 have long distance which decreases the overlapping area and possible positions for the checkerboard. The positions for checkerboard are drawn to the overlapping area.*

All the calibrations between all possible camera pairs gave the 0.2 pixel reprojection error. That result can be achieved even if the estimation for camera center position has many millimeters error in y and z directions. One reason for that could be the small position variation for checkerboard in the calibration images as in Figure 5.9.

6 CONCLUSION

The goal of this thesis was to do ChArUco checkerboard corner detection and multi-camera calibration of a camera rig of 20 cameras. The corner detection consists of marker detection and corner interpolation between detected markers. The result of this step outputs the image coordinates of the corners and the corresponding IDs for them. The calibration for multiple cameras was supposed to do simultaneously to get accurate results. Due to the timing, it was done pair-wisely which was proved at the same time to be inaccurate for the purpose of knowing the cameras' relative positions in capturing the light field.

The scripts built during this thesis are able to detect corners in different ChAcUro boards in good conditions. However, the cameras set the boundaries, since the price of cameras is fairly high, and cannot be replaced. This is why the marker detection should be improved or the conditions should be tested better to capture images from where the all visible markers can be detected. Other codes are beneficial for debugging this marker detection problem. The real-time marker detection A.1 helps to see when, at what distances and lighting, the markers are detected. The still image corner visualization B.1 helps to see how the corners were detected, and how did the conditions look. File writing D.1 and reading E.1 for the image coordinates and IDs work, and can be used for later usage in Matlab.

The multi-camera calibration, done by stereo calibrating camera pairs, was shown to be inaccurate for the purpose of capturing light field. Based on the results we can require more accurate methods which are in the case of this project using the ChArUco checkerboard and simultaneous multi-camera calibration. The expected error from these methods is less than one pixel as it is in stereo calibration.

REFERENCES

[1] *Affine and Projective Transformations*. URL: https://www.graphicsmill.com/docs/gm/affine-and-projective-transformations.htm (visited on 03/26/2019).

[2] S. Beriault. Multi-camera system design, calibration and three-dimensional reconstruction for markerless motion capture. PhD Thesis. 2008.

[3] D. C. Brown. Decentering Distortion of Lenses. (1965).

[4] A. Butterfield & G. E. Ngondi. *A Dictionary of Computer Science*. 2016.

[5] *Camera Lens*. Mar. 13, 2019. URL: https://en.wikipedia.org/wiki/Camera_lens (visited on 03/26/2019).

[6] *Cartography for Swiss Higher Education*. Jan. 26, 2012. URL: http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/Transform_learningObject2.html (visited on 03/10/2019).

[7] D. H. Castro. From Images to Point Clouds. Practical Considerations for Three-Dimensional Computer Vision. PhD Thesis. 2015.

[8] X. Chen, J. Davis & P. Slusallek. Wide area camera calibration using virtual calibration objects. *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*. Vol. 2. 2000, 520–527 vol.2.

[9] *Detection of Aruco Markers*. URL: https://docs.opencv.org/3.2.0/d5/dae/tutorial_aruco_detection.html.

[10] J.-E. H. Dong-Joong Kang & M.-H. Jeong. *Detection of Calibration Patterns for Camera Calibration with Irregular Lighting and Complicated Backgrounds*. 2008.

[11] M. Drennan. An Implementation of Camera Calibration Algorithms. (2008).

[12] H. Eugene. *Optics*. Fifth edition. Pearson, 2017. ISBN: 987-1-292-09693-3.

[13] D. Forsyth & J. Ponse. *Computer Vision: A Modern Approach*. Pearson, 2003.

[14] *Full Frame Fisheye Lenses*. URL: https://int-store.bitplayinc.com/products/36-full-frame-fisheye-lens (visited on 03/21/2019).

[15] Y. Furukawa & J. Ponse. Accurate Camera Calibration from Multi-View Stereo and Bundle Adjustment. *International Journal of Computer Vision* 84.3 (2009). ISSN: 1573-1405.

[16] R. Hartley & A. Zisserman. *Multiple View Geometry in Computer Vision*. Second edition. Cambridge University Press, 2003.

[17] J. Heikkila & O. Silven. Calibration procedure for short focal length off-the-shelf CCD cameras. 1 (1996), 166–170 vol.1. ISSN: 1051-4651.

[18] J. Heikkila & O. Silven. A four-step camera calibration procedure with implicit image correction. (1997), 1106–1112. ISSN: 1063-6919.

[19] D. Herrera C., J. Kannala & J. Heikkilä. Joint Depth and Color Camera Calibration with Distortion Correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.10 (2012), 2058–2064. ISSN: 0162-8828.

[20] D. Hu, D. DeTone, V. Chauhan, I. Spivak & T. Malisiewicz. Deep ChArUco: Dark ChArUco Marker Pose Estimation. (2018).

[21] I. Ihrke, L. Ahrenberg & M. Magnor. External Camera Calibration for Synchronized Multi-Video Systems. *Journal of WSCG* 12 (Feb. 2004).

[22] A. Kaehler & G. Bradski. *Learning OpenCV 3. Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2017.

[23] M. Levoy & P. Hanrahan. Light Field Rendering. SIGGRAPH 96, 1996, 31–42.

[24] K. Li, Q. Dai & W. Xu. Collaborative color calibration for multi-camera systems. *Elsevier* (2011).

[25] Q. Liu & H. Su. Camera calibration based on correction of asymmetric circle center projection. 17 (2009), 3103–3108.

[26] *Matlab Documentation*. URL: `https://se.mathworks.com/help/matlab/`.

[27] G. Medioni & S. B. Kang. *Emerging Topics in Computer Vision*. Prentice Hall, 2005. ISBN: 978-0-13-101366-7.

[28] T. Melen. Geometrical modelling and calibration of video cameras for underwater navigation. (1994).

[29] X. L. Meng, W. T. He, X. Q. Che & C. Zhao. Correction of the Asymmetrical Projection for Accurate Camera Calibration Using Co-Planar Circular Feature. 397 (2013), 1016–1020.

[30] S. K. Moore. Self-Powered Image sensor Could Watch You Forever. *IEEE Spectrum* (2018).

[31] *OpenCV Documentation*. URL: `https://docs.opencv.org/`.

[32] *Python Documentation*. URL: `https://docs.python.org/3/`.

[33] S.-W. Shih, Y.-P. Hung & W.-S. Lin. Accurate linear technique for camera calibration considering lens distortion by solving an eigenvalue problem. *Optical Engineering - OPT ENG* 32 (1993), 138–149.

[34] K. Simek. *Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix*. 2012.

[35] K. H. Strobl & G. Hirzinger. More accurate pinhole camera calibration with imperfect planar target. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE Computer Society, 2011, 1068–1075.

[36] T. Svoboda, D. Martinec & T. Pajdla. A Convenient Multi-Camera Self-Calibration for Virtual Environments. *Presence* 14 (Aug. 2005), 407–422.

[37] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[38] R. Unnikrishnan & M. Hebert. Fast Extrinsic Calibration of a Laser Rangefinder to a Camera. (2005).

[39] *What is a QR-code*. URL: `https://www.qrcode.com/en/about/`.

[40] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), 1330–1334. ISSN: 0162-8828.

A REAL-TIME MARKER DETECTION

```
# Checkerboard parameters
CORNERS_X = 11
CORNERS_Y = 8
MARKER_DICT = 5
SQUARE_LENGTH = 60
MARKER_LENGTH = 47


def main():
    # Start capturing frames from camera
    cap = cv2.VideoCapture(0)
    while True:

        # Read one frame
        _, frame = cap.read()
        charuco_corners, charuco_ids \
            = getCharucoCorners(frame,
                                CORNERS_X,
                                CORNERS_Y,
                                MARKER_DICT,
                                SQUARE_LENGTH,
                                MARKER_LENGTH)

        # At least one corner is detected, draw corner
        if (charuco_ids.size > 0):
            drawDetectedCornersCharuco(frame,
                                       charuco_corners,
                                       charuco_ids)

        # Draw image or quit
        cv2.imshow('video', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

```

```
35      # Release camera instance and close window
36      cap.release()
37      cv2.destroyAllWindows()
```

Program A.1. *A real-time ChArUco marker detection, and a frame for how to use corner detection functions.*

B STILL IMAGE CORNER VISUALIZATION

```
1  import cv2
2  import numpy as np
3  import sys
4  from get_charuco_corners import getCharucoCorners, \
5                                  drawDetectedCornersCharuco
6
7  CORNERS_X = 11
8  CORNERS_Y = 8
9  MARKER_DICT = 5
10 SQUARE_LENGTH = 60
11 MARKER_LENGTH = 47
12
13
14 def main():
15
16     file_name = "../data/test_im/img3.bmp"
17
18     # If argument was given
19     if len(sys.argv) > 1:
20         file_name = sys.argv[1:][0]
21
22     frame = cv2.imread(file_name)
23     charuco_corners, charuco_ids \
24         = getCharucoCorners(frame,
25                             CORNERS_X,
26                             CORNERS_Y,
27                             MARKER_DICT,
28                             SQUARE_LENGTH,
29                             MARKER_LENGTH)
30
31     # If any corner is detected, draw corner
32     if (charuco_ids.size > 0):
33         drawDetectedCornersCharuco(frame, charuco_corners,
34                                    charuco_ids)
```

```
35          print("Number␣of␣detected␣corners:",
36                charuco_corners.shape[0])
37
38      # Draw image or quit
39      cv2.imshow('Image', frame)
40      if cv2.waitKey(0) & 0xff == 27:
41          cv2.destroyAllWindows()
```

Program B.1. *Drawing corners for still image.*

C CHARUCO CORNER DETECTION RATE

```
CORNERS_X = 11
CORNERS_Y = 8
MARKER_RESOLUTION = 5
SQUARE_LENGTH = 60
MARKER_LENGTH = 47


def main():
    detected_images = 0
    detected_corners = 0
    image_names = glob("../data/RigCalibImagesV3/*.bmp")

    # If argument was given
    if len(sys.argv) > 1:
        image_names = sys.argv[1:]

    for i in range(len(image_names)):
        charuco_corners, charuco_ids = \
            getCharucoCorners(image_names[i],
                              CORNERS_X,
                              CORNERS_Y,
                              MARKER_RESOLUTION,
                              SQUARE_LENGTH,
                              MARKER_LENGTH)

        # If any corner is detected, count it
        if (charuco_ids.size > 0):
            detected_images += 1
            detected_corners += charuco_corners.shape[0]

        print("Image: {}/{}, Detected: {}"
              .format(i+1, len(image_names),
                      detected_images),
              end="\r")
```

```
35
36      print("\nSuccessfully␣detected␣corners␣in␣images:␣{}/{}"
37            .format(detected_images, len(image_names)))
38      print("Detection␣rate␣on␣average:␣{}␣%"
39            .format(round(detected_corners /
40                    (CORNERS_X * CORNERS_Y * len(image_names))
41                    * 100, 2)))
```

Program C.1. *Compute detection rate for a set of images.*

D WRITE IMAGE COORDINATES TO FILES

```
1  import numpy as np
2  from glob import glob
3  import scipy.io
4  import sys
5  from get_charuco_corners import getCharucoCorners
6
7  CORNERS_COORS_FNAME = "coords.mat"
8  CORNERS_IDS_FNAME = "ids.mat"
9  CORNERS_X = 11
10 CORNERS_Y = 8
11 MARKER_DICT = 5
12 SQUARE_LENGTH = 60
13 MARKER_LENGTH = 47
14 NUMBER_OF_CAMERAS = 20
15
16
17 def writeToFile(corners_coords, corners_ids):
18     scipy.io.savemat(CORNERS_COORS_FNAME, mdict=corners_coords)
19     scipy.io.savemat(CORNERS_IDS_FNAME, mdict=corners_ids)
20     print("Corners␣and␣IDs␣has␣been␣written␣to␣files:",
21           CORNERS_COORS_FNAME, CORNERS_IDS_FNAME)
22
23
24 def main():
25     detected_corners_images = 0
26     image_names = glob("../data/RigCalibImagesV3/*.bmp")
27     coords = {}
28     ids = {}
29     images_no_corners = []
30
31     # If argument was given, change image location(s)
32     if len(sys.argv) > 1:
33         image_names = sys.argv[1:]
34
```

```
35      for i in range(len(image_names)):
36          # Detect corners
37          charuco_corners, charuco_ids = \
38              getCharucoCorners(image_names[i],
39                                CORNERS_X,
40                                CORNERS_Y,
41                                MARKER_DICT,
42                                SQUARE_LENGTH,
43                                MARKER_LENGTH)
44
45          # Add corners to dictionary
46          key = "pos_" + str(int(i / NUMBER_OF_CAMERAS)) \
47                + "_cam_" + str(int(i % NUMBER_OF_CAMERAS))
48          coords[key] = charuco_corners
49          ids[key] = charuco_ids
50
51          # Count the number of detected images
52          # Save image names without detected corners
53          if (charuco_ids.size > 0):
54              detected_corners_images += 1
55          else:
56              parsed_name = image_names[i].split('/')[-1]\
57                                          .split('.')[0]
58              images_no_corners.append(parsed_name)
59
60          print("Image: {}/{}, Images of detected corners: {}"
61                .format(i+1, len(image_names),
62                        detected_corners_images),
63                end="\r")
64
65      print("\nDetected corners in images: {}/{}"
66            .format(detected_corners_images, len(image_names)))
67
68      # Print image names without detected corners
69      if (len(images_no_corners)):
70          print("Images without detected corners:")
71          for i in range(len(images_no_corners)):
72              print("{}. {}".format(i+1, images_no_corners[i]))
73
74      writeToFile(coords, ids)
```

Program D.1. *Write image coordinates and IDs to .mat files.*

E READ IMAGE COORDINATES FROM FILES

```
function [imCorners, imIds] = ...
read_coordinates_from_mat(maxNumberOfCorners, numberOfImages)

% Read checkerboard corners and corner IDs
charucoCornersXY = load('coords.mat');
cornerIds = load('ids.mat');

% Get image names
cornerFields = fieldnames(charucoCornersXY);
idFields = fieldnames(cornerIds);

% Read all images if numberOfImages == -1
if numberOfImages == -1
    numberOfImages = numel(cornerFields);
end
imCorners = zeros(maxNumberOfCorners, 2, numberOfImages);
imIds = zeros(numberOfImages, maxNumberOfCorners);

% Loop all images
for i = 1:numberOfImages
    corners = charucoCornersXY.(cornerFields{i});
    ids = cornerIds.(idFields{i});
    missingNumberOfCorners = maxNumberOfCorners - length(ids);

    % Add -1 to the end
    cornersFilled = [corners;...
                     -1*ones(missingNumberOfCorners, 2)];
    idsFilled = [ids -1*ones(1, missingNumberOfCorners)];
    imCorners(:, :, i) = cornersFilled;
    imIds(i, :) = idsFilled;
end
end
```

Program E.1. *Read image coordinates and IDs from .mat files.*

F DRAW DETECTED CHARUCO CORNERS

```
1 def drawDetectedCornersCharuco(img, corners, ids):
2     id_color = (255, 255, 0)
3     corners = corners.reshape((corners.shape[0], 1,
4                                corners.shape[1]))
5     ids = ids.reshape((ids.size, 1))
6     cv2.aruco.drawDetectedCornersCharuco(img, corners, ids,
7                                          id_color)
```

Program F.1. *Draw detected ChArUco corners wrapper.*