High Accuracy Eye-Tracker For Proton Clinic Environment

Yu Zhang

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

High Accuracy Eye-Tracker For Proton Clinic Environment

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

Yu Zhang

October 22, 2019

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

Abstract

This project aims to improve the accuracy of the eye tracking system, which consists of two cameras and two infrared LED light sources. This highly non-invasive technology, which is the feature-based video-oculographic eye tracking system, determines the position of the eye by monitoring the eye features such as the pupil center and glints. The accuracy of estimating the eye position and orientation is critical in the proton clinic environment, and is to be required higher than those in commercially available eye-tracking system.

By investigating the inverse problem of the monitoring process and solving the optimization problems, the positions of the cameras and light sources can be calibrated to improve the measurement accuracy and the proper gaze angle will be given. The combination of calibration and gaze estimation gives the optimal solution for the configuration of the eye tracking system.

The goal is to achieve better knowledge of camera calibration and light source calibration and build a foundation for further improvement of accuracy of the system. A method to solve the optimization problem which minimizes or maximizes the cost function will be proposed and the effectiveness of the method will be investigated using numerical simulations (Matlab) and validated experimentally.

For this purpose, a high-precision camera calibration was evaluated and implemented. Result of 3D reconstruction of feature points on calibration chessboard has achieved 0.1 mm, while the method of reconstructing light source didn't give a good performance.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Prof. M. Verhaegen and Dr. Oleg Soloviev. The door to Dr. Oleg Soloviev's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He encouraged and helped me to find out my own thoughts to solve problems.

More than the knowledge of the project that I get, the most important thing is that I learned how to think, how to explore and how to be brave.

Thanks for my friends, who always encouraged me when I got lost.

Finally I'd like to use an opportunity to thank my parents. They support my study and encourage me to grow up.

I'm grateful for everyone that helps me in my life!

Delft, University of Technology — Yu Zhang
October 22, 2019

"Errors, like straws, upon the surface flow; He who would search for pearls must dive below."

— *John Dryden*

Chapter 1

Introduction

This chapter introduces the background information of this thesis project, which includes a brief overview of uveal melanoma, the proton therapy and eye tracker. The proton therapy is widely used in eye cancer treatment due to the technical factors of the proton beam [1].

1-1 Non-invasive eye-tracking for proton treatment of the uveal melanoma

1-1-1 Eye Cancer

Eye cancer is rare and ocular melanoma is the most common one. 85% of ocular tumor are uveal melanoma [1]. When the melanoma arises from iris, ciliary body or choroid, it is called uveal melanoma. The positions of the tumors are usually detected by MRI or CT. Up to 50% of patients with uveal melanoma will ultimately develop distant metastasis [2].

Typical treatment methods for uveal melanoma can be classified into globe-preserving therapy or enucleation. Globe-preserving method is the first choice to preserve the eyesight. There are three kinds of globe-preserving therapy: radiation, surgical and laser therapy [3].

Radiation is currently the most common method to treat with uveal melanoma. Proton therapy, as one of the external radiation methods, is a kind of radiation with usage of proton beams.

Compared to x-ray radiation therapy [4], the physical property of proton beam helps to deliver 60% less dose to healthy tissues around the tumor and higher dose to the tumor than x-ray radiation, which offers higher chance to destroy all tumor tissues and lower chance of side effects. As shown in Figure 1-1, x-ray therapy may cause damage to healthy tissues around the tumor and increases the chance of side effects.

But there are also some drawbacks to proton therapy. This treatment method requires highly specialized and costly equipment, which means high cost and less medical centers that can offer. Besides, not all cancers can be treated with proton therapy.

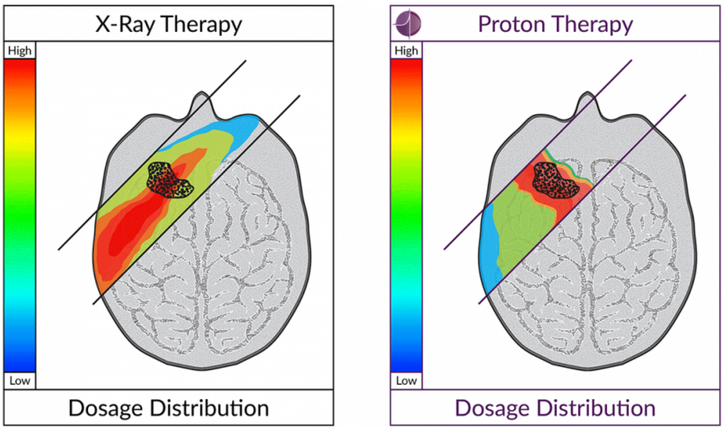


Figure 1-1: The comparison of dosage distribution between x-ray and proton beam radiation. X-ray therapy may cause damage to healthy tissues around the tumor and increases the chance of side effects[5].

1-1-2 Proton Therapy

The benefits of proton therapy outweigh the disadvantages. The proton radiotherapy is considered as primary treatment for recurrent tumor and prior to surgical resection for uveal melanoma. Main advantage of proton therapy is that the physical properties of proton beams enable high-dose delivery with less damage to adjacent normal issues than conventional radiotherapy.

Bragg peak The advantageous physical property of proton beam is the Bragg peak [4]. The energy of the proton beam reaches the peak at a point and after that it dissipates quickly. This property enables greatest damage to the tumor points with relative sparing of healthy tissues. Also because of this property, proton therapy is only sufficient for superficial lesions such as primary ocular tumors.

Conventional Proton Therapy In the process of proton therapy, the tumor needs to be aligned on the trajectory of the proton beam because the proton beams are limited to a fixed (often horizontal) position [1], as shown in Figure 1-3.

The color of the eye will change in case of iris melanoma, which is visible. However in case of ciliary body and choroid melanoma the tumors are invisible. After examination of tumor position through MRI or CT [6], the position and orientation of the target eye are required in high precision.

In conventional proton therapy eye tracking technology, tantalum markers are inserted into the eye and the number of the tantalum markers ranges from three to five, which will not be removed after therapy. The markers will be detected by x-rays. For iris melanoma, markers are not required. For ciliary body and choroid melanoma, markers help to build a 3D geometrical model of the target eye. The gaze direction is adjusted based on the eye model to minimize unnecessary damage to healthy ocular structures. Hence, measurement of position and orientation of the eye is critical for the proton therapy process.

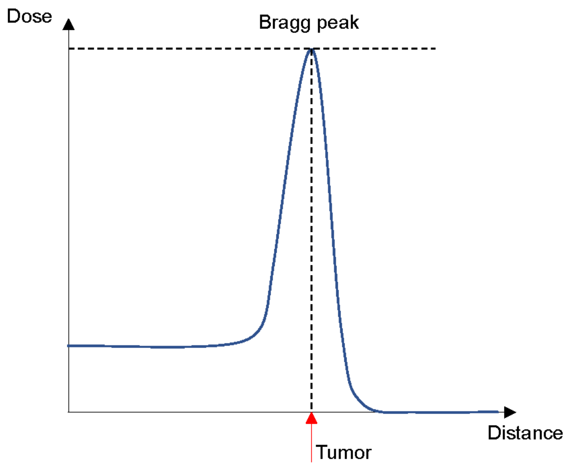


Figure 1-2: Bragg Peak. Energy of the proton beam reaches its peak at tumor and dissipates rapidly after that. The distance is the depth in tissue.

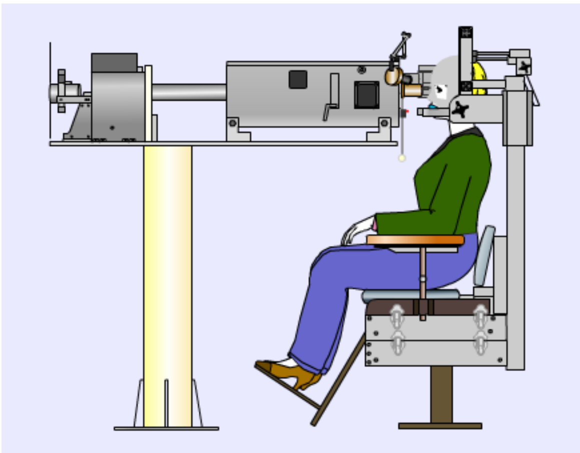


Figure 1-3: The figure shows the image of proton therapy. The tumor should be aligned on the trajectory of the proton beam since the beam is fixed, which is usually horizontal. The head should be fixed [4].

The eye tracking technology of inserting tantalum markers into eyeball is highly invasive. To avoid this painful surgery and improve the flexibility of eye tracking process, non-invasive eye tracking technology is a good choice provided its accuracy can reach the requirements of the proton therapy.

1-2 Non-invasive Eye Tracker In Proton Clinic Environment

The measurement device for measuring eye movements is called an eye tracker. Eye tracking technologies are currently widely used in many fields, such as psychology, VR games etc..

The techniques for monitoring eye movement are in general of two types: one is to measure the position of the eye relative to the head, and another is to measure the orientation of the eye or the point of gaze (POG). The former technology requires measurement of the head position, which makes the eye tracking process more complex. The latter one is based on the identification of features in the images of eyes. The most widely used eye tracking methodology is video-oculographic eye tracker.

Based on these two types, there are four main categories of the eye tracking technology [7]:

- **Electro-OculoGraphy(EOG)** relies on the measurement of the skin's potential differences with electrodes placed around the eye. This technique measures the position of the eye relative to the head.

- **Scleral Contact Lens/Search Coil** is a highly intrusive and precise method. This technology makes use of modern contact lens attached by a mounting stalk. The contact lens should be large enough to include the cornea and sclera. This also measures the position of the eye relative to the head.

- **Photo-OculoGraphy(POG) or Video-OculoGraphy(VOG)** involves the measurement of distinguishable features such as the shape of the pupil and the corneal reflection of light sources under rotation or translation. Still it's the measurement of the relative position of the eye with respect to the head.

- **Video-Based Combined Pupil and Corneal Reflection** is based on VOG and provides the measurement of the point of gaze with either fixed head or measurement of multiple ocular features like corneal reflections and pupil center. The difference between these two methodologies is the positional difference between the pupil center and corneal reflections are used, which changes with pure eye rotation but remains relatively constant with minor head movements. For simplicity here we call all video-based eye tracking methodologies as video-oculographic eye tracker.

Video-oculographic method is the most common method in eye tracking technologies. It can be improved even with relative inexpensive cameras and light sources (usually infrared or near-infrared LED light sources). Due to its convenience and inexpensive hardware setup, video-based eye trackers are popular in commercial market.

Accuracy and precision are important for eye trackers. Accuracy refers to the closeness of a measured value to a standard or known value, while precision refers to the closeness of two or more measurements to each other, which may be not accurate.

As described in previous part, the tumors need to be treated before the metastasis and they grow into large tumors. The typical size of ocular organics are in millimeters. Thus the accuracy of the eye trackers are required to be sub-millimeter. The consideration of gaze angle requires the rotation accuracy of sub-degree since the visual angle is approximately 1 degree.

According to [8], the accuracy can achieve 1-2 degrees with simple web camera using the image processing algorithms. However, the accuracy of current available eye trackers are still not enough for medical usage.

1-3 Motivation of Accuracy Improvement

A prototype of the high-accuracy eye tracker was developed in the Delft Center for Systems and Control (DCSC) lab. The system consists of two cameras and two near-infrared light sources. This prototype combined with a suitable software configuration can satisfy a good accuracy. However, the system still cannot be used on proton clinic environment because it requires rigid system configuration, which means everything needs to be under control and cannot be moved. This is not suitable for using in real life environment.

Precise camera calibration and light source calibration help to improve the accuracy of the non-invasive eye-tracker. Recognition of contours of pupil and glints is not in the scope of this thesis.

In this literature survey, theories about camera geometry and methods of camera calibration will be introduced. Different methods of light source calibration will be illustrated. The literature survey helps to understand the theory and background of the project.

1-4 Objectives and Outline

1-4-1 Objectives

The goal of the project is to form the calibration method for the eye tracker with flexibility and robustness such that we can get measurements of high accuracy.

1-4-2 Outline

The outline of the rest of the report is as follows:

- Chapter 2: introduction of basic knowledge of the eye model and the video-oculographic eye tracker system principles.
- Chapter 3: introduction of camera calibration methods and light source calibration.
- Chapter 4: results of experiments.
- Chapter 5: conclusion

Chapter 2

Theory Overview

2-1 Eye Model

The shapes of the cornea are in common considered as spheres, although they are not perfect.

The geometrical eye model consists of two overlapped spheres as shown in Figure 2-1. In Figure 2-1 there are several important definitions of eye structures:

- $\mathbf{c}$: The center of cornea curvature.
- $\mathbf{p}$: The pupil center.
- R: The radius of the cornea curvature.
- K_{pc}: The distance between the center of cornea curvature $\mathbf{c}$ and the pupil center $\mathbf{p}$.
- $\mathbf{d}$: The center of eyeball.
- Optic axis: Defined by the center of cornea curvature $\mathbf{c}$ and the pupil center$\mathbf{p}$.
- Visual axis: Defined by the center of cornea curvature $\mathbf{c}$ and fovea. Visual axis defines the gaze direction.

There is a constant angular difference between the optic axis and the visual axis. The visual axis is usually obtained from the optic axis. Let's denote the optic axis as $\mathbf{s}$:

$$\mathbf{s} = \mathbf{p} - \mathbf{c}. \tag{2-1}$$

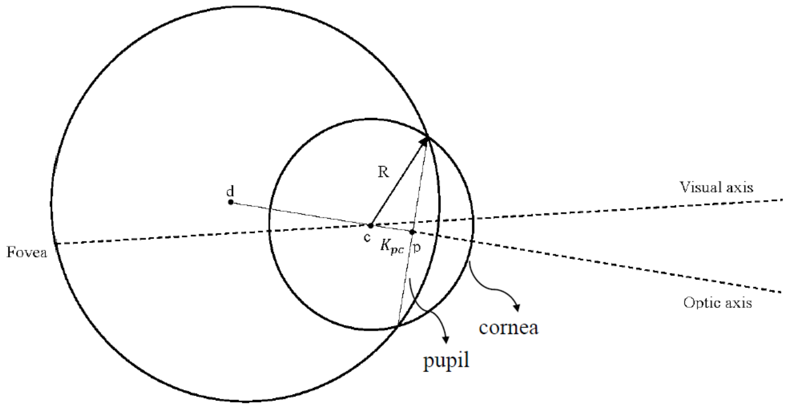


Figure 2-1: The eye model consists of two overlapped spheres. The big sphere represents the eye ball and the small one represents the cornea. The center of the eyeball is denoted as $\mathbf{d}$ and the center of cornea curvature is denoted as $\mathbf{c}$. The pupil center $\mathbf{p}$ is on the intersection between the intersection of the two spheres and the line defined by the center of cornea curvature and the center of eyeball. The optic axis is defined by the center of cornea curvature and the pupil center. The visual axis is defined by the center of cornea curvature and the fovea. The gaze direction is defined by the visual axis.

2-2 Gaze Estimation

2-2-1 Features In Images Of Eyes

As indicated in Chapter 1, the video-oculographic eye trackers monitor eye movements by tracking eye features such as the pupil center and corneal reflections of closely placed light sources. The corneal reflections appeared on the surface of cornea curvature are called Purkinje Images, as shown in Figure 2-2.

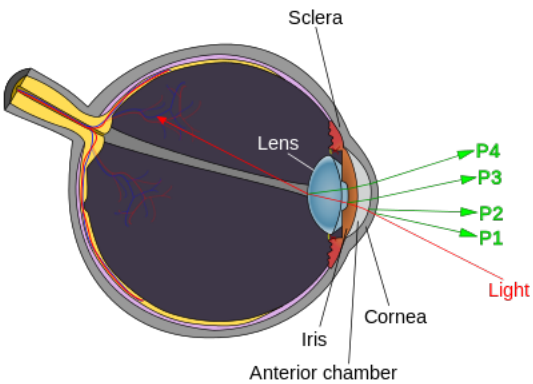


Figure 2-2: Diagram of light and four Purkinje images. (Credit to Wikipedia)

The four Purkinje images are reflections of illumination from different parts of primary eye structures. The main difference among these four images is the intensity of their illumination.

- The first Purkinje image (P1): the reflection from outer surface of the cornea curvature;
- The second Purkinje image (P2): the reflection from inner surface of the cornea curvature;
- The third Purkinje image (P3): the reflection from outer surface of the lens;
- The fourth Purkinje image (P4): the reflection from inner surface of the lens;

In most cases, the first Purkinje image (P1), known as glints, is visible and used for eye trackers.

Second We talk about the pupil center. Usually the pupil contour is detected by contour extraction software and it's not a perfect circle. Different illumination methods will cause different effects. In video-oculographic eye trackers, the light sources are usually infrared or near-infrared such that the user won't be distracted from the light sources.

Two types of infrared/near-infrared (also known as active light) eye tracking technologies:

- Bright-Pupil Effect: when the illumination ray is coaxial with the optic path of the system (the cameras, the eye and the light sources), the eye acts as a retro-reflector as shown in Figure 2-3.
- Dark-Pupil Effect: when the illumination ray is offset from the optical path, clear images of pupil and glints are visible, as shown in Figure 2-4.

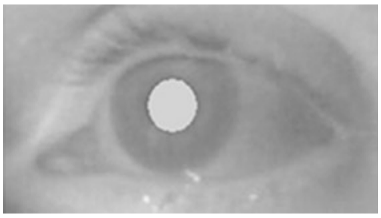


Figure 2-3: Bright-pupil Effect (Credit to Wikipedia)

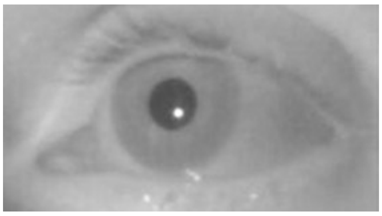


Figure 2-4: Dark-Pupil Effect (Credit to Wikipedia)

Images with glints obtained from dark-pupil effect allows us to identify the eye's movement.

2-2-2 Geometrical Methodology

The eye tracking system used consists of two cameras and two light sources. In the model of the system, pinhole camera model is used and light sources are considered as point light sources [9].

The setup is extended from one camera and one light source system, as shown in Figure 2-5. Further one camera with multiple light sources and multiple cameras and light sources will be extended.

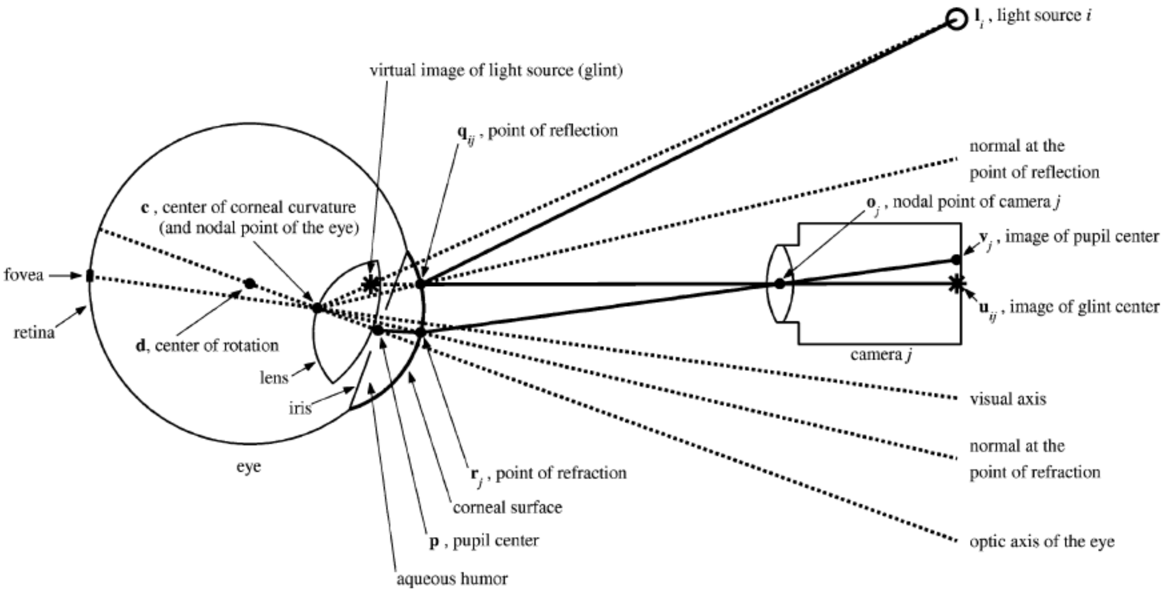


Figure 2-5: This diagram shows the representations of the eye, the cameras and the light sources and shows one camera and one light source system for simplicity. The index of the cameras are represented as $\mathbf{o}_j$, which are the principle points of the cameras. Positions of light sources are represented as $\mathbf{l}_i$ [9].

As indicated in Figure 2-5, features and positions of the cameras and lights are assigned with symbols:

- $\mathbf{l}_i$: light source i
- $\mathbf{o}_j$: camera j
- $\mathbf{q}_{ij}$: point of corneal reflection (generated by light source i and the corresponding image is on camera j)
- $\mathbf{r}_j$: point of refraction of pupil center (image taken by camera j)
- $\mathbf{u}_{ij}$: image of glint center (corresponding image of glint $\mathbf{q}_{ij}$)
- $\mathbf{v}_j$: image of pupil center (corresponding image of glint $\mathbf{r}_j$)

Mathematical Model As shown in figure 2-5, a ray comes out from the light source $\mathbf{l}_i$ and reflects at the point $\mathbf{q}_{ij}$, which indicates that the ray passes through the nodal point $\mathbf{o}_j$ of the camera. The reflected ray intersects with the image plane at point $\mathbf{u}_{ij}$. We can get the following equations from this process:

$$\mathbf{q}_{ij} = \mathbf{o}_j + k_{q,ij}(\mathbf{o}_j - \mathbf{u}_{ij}) \quad \text{for some} \quad k_{q,ij}, \tag{2-2}$$

$$\|\mathbf{q}_{ij} - \mathbf{c}\| = R. \tag{2-3}$$

The law of reflection states two conditions:

- The incident ray, the reflected ray and the normal at the point of reflection are co-planar;
- The angles of incidence and reflection are equal.

These two conditions can be expressed as:

$$(\mathbf{l}_i - \mathbf{o}_j) \times (\mathbf{q}_{ij} - \mathbf{o}_j) \cdot (\mathbf{c} - \mathbf{o}_j) = 0 \tag{2-4}$$
$$(\mathbf{l}_i - \mathbf{q}_{ij}) \cdot (\mathbf{q}_{ij} - \mathbf{c}) \cdot \|\mathbf{o}_j - \mathbf{q}_{ij}\| = (\mathbf{o}_j - \mathbf{q}_{ij}) \cdot (\mathbf{q}_{ij} - \mathbf{c}) \cdot \|\mathbf{l}_i - \mathbf{q}_{ij}\|. \tag{2-5}$$

There's another ray that comes from the pupil center refracts at point $\mathbf{r}_j$ on the surface of the corneal curvature and passes through the nodal point $\mathbf{o}_j$ of the camera. The refracted ray intersects with the image plane on the point $\mathbf{v}_j$. We can get the following equations from this process:

$$\mathbf{r}_j = \mathbf{o}_j + k_{r,j}(\mathbf{o}_j - \mathbf{v}_j) \quad \text{for some} \quad k_{r,j} \tag{2-6}$$
$$\|\mathbf{r}_j - \mathbf{c}\| = R. \tag{2-7}$$

The law of refraction states two conditions:

- The incident ray, the refracted ray and the normal at the point of refraction are coplanar;
- The angles of incidence and refraction satisfy Snell's law.

These two conditions can be expressed as:

$$(\mathbf{r}_j - \mathbf{o}_j) \times (\mathbf{c} - \mathbf{o}_j) \cdot (\mathbf{p} - \mathbf{o}_j) = 0 \tag{2-8}$$
$$n_1 \cdot \|(\mathbf{r}_j - \mathbf{c}) \times (\mathbf{p} - \mathbf{r}_j)\| \cdot \|\mathbf{o}_j - \mathbf{r}_j\| = n_2 \cdot \|(\mathbf{r}_j - \mathbf{c}) \times (\mathbf{o}_j - \mathbf{r}_j)\| \cdot \|\mathbf{p} - \mathbf{r}_j\|. \tag{2-9}$$

Finally, the optical axis ($\mathbf{s}$) of the eye is defined by the line that passes both the corneal curvature ($\mathbf{c}$) and the pupil center ($\mathbf{p}$). The orientation of the optic axis can be described by the horizontal angle θ_{eye} and the vertical angle ϕ_{eye} defined in Figure 2-6. The two angles can be obtained from the corneal curvature ($\mathbf{c}$) and the pupil center ($\mathbf{p}$) with Eq. (2-10).

$$\mathbf{s}_{norm} = \frac{\mathbf{p} - \mathbf{c}}{\|\mathbf{p} - \mathbf{c}\|} = \begin{bmatrix} \cos\phi_{eye}\sin\theta_{eye} \\ \sin\phi_{eye} \\ -\cos\phi_{eye}\cos\theta_{eye} \end{bmatrix}. \tag{2-10}$$

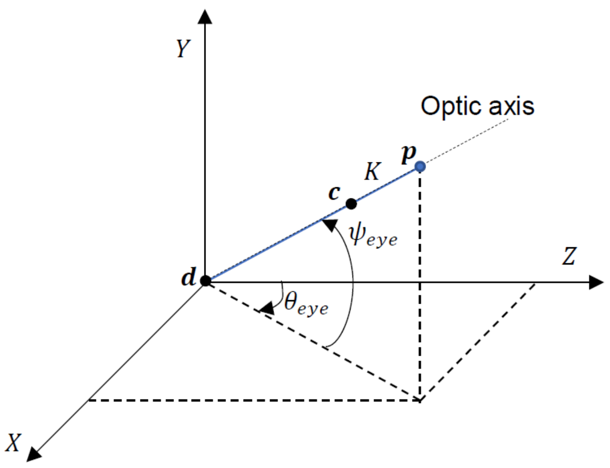


Figure 2-6: The angles between the optic axis and the horizontal plane. The X-Z plane in the figure is parallel to the horizontal plane.

The visual axis is defined by the center of the fovea and the center of the corneal curvature as shown in Figure 2-5, which gives the point of gaze. Since the horizontal and vertical angle differences between the optic axis and the visual axis are constants α_{eye} and β_{eye}, which will only cause constant error, we will consider only the optic axis for simplicity.

One Camera and One Light Source One camera and one light source system is the simplest system configuration. In this case, all the parameters of the eye, which means the radius of the corneal curvature R, the distance K between the center of the pupil ($\mathbf{p}$) and the center of the corneal curvature ($\mathbf{c}$) and the refractive index of the corneal curvature (n_1), should be known. The system of Eq. (2-2)-Eq. (2-10) in this case takes $i = 1$ and $j = 1$, which is equivalent to 13 scalar equations with 14 unknowns. To solve this problem, additional constraints should be considered and the head should be completely stationary.

One Camera and Multiple Light Sources In this case $i = 1 \dots N$ and $j = 1$. Substitute Eq. (2-2) into Eq. (2-4):

$$(\mathbf{l}_i - \mathbf{o}) \times (\mathbf{u}_i - \mathbf{o}) \cdot (\mathbf{c} - \mathbf{o}) = 0. \tag{2-11}$$

The equation is rewritten into matrix to obtain:

$$\begin{bmatrix} [(\mathbf{l}_1 - \mathbf{o}) \times (\mathbf{u}_1 - \mathbf{o})]^T \\ [(\mathbf{l}_2 - \mathbf{o}) \times (\mathbf{u}_2 - \mathbf{o})]^T \\ \vdots \\ [(\mathbf{l}_N - \mathbf{o}) \times (\mathbf{u}_N - \mathbf{o})]^T \end{bmatrix} (\mathbf{c} - \mathbf{o}) = 0. \tag{2-12}$$

In this case the number equations are enough to solve the center of the corneal curvature $\mathbf{c}$. Consequently, the pupil center $\mathbf{p}$ can be solved. To reconstruct the optic axis, the eye parameters R, K, n_1 must be known.

As said in previous paragraphs, the system consists of one camera and multiple light sources is the simplest configuration that allows free head movement. A useful method to solve this problem is linear regression[10].

Multiple Cameras and Multiple Light Sources When multiple cameras and multiple light sources are used, the equations can be written as:

$$\underbrace{\begin{bmatrix} [(\mathbf{l}_1 - \mathbf{o}_1) \times (\mathbf{u}_{11} - \mathbf{o}_1)]^T \\ [(\mathbf{l}_1 - \mathbf{o}_2) \times (\mathbf{u}_{12} - \mathbf{o}_2)]^T \\ \vdots \\ [(\mathbf{l}_N - \mathbf{o}_1) \times (\mathbf{u}_{N1} - \mathbf{o}_1)]^T \\ [(\mathbf{l}_N - \mathbf{o}_2) \times (\mathbf{u}_{N2} - \mathbf{o}_2)]^T \end{bmatrix}}_{\mathbf{M}_0} \mathbf{c} = \underbrace{\begin{bmatrix} (\mathbf{l}_1 - \mathbf{o}_1) \times (\mathbf{u}_{11} - \mathbf{o}_1) \cdot \mathbf{o}_1 \\ (\mathbf{l}_1 - \mathbf{o}_2) \times (\mathbf{u}_{12} - \mathbf{o}_2) \cdot \mathbf{o}_2 \\ \vdots \\ (\mathbf{l}_N - \mathbf{o}_1) \times (\mathbf{u}_{N1} - \mathbf{o}_1) \cdot \mathbf{o}_1 \\ (\mathbf{l}_N - \mathbf{o}_2) \times (\mathbf{u}_{N2} - \mathbf{o}_2) \cdot \mathbf{o}_2 \end{bmatrix}}_{\mathbf{h}}. \tag{2-13}$$

If $\mathbf{M}_0$ in Eq. (2-13) has rank of 3, $\mathbf{c}$ can be obtained from Eq. (2-13) using the left pseudoinverse of $\mathbf{M}_0$ as

$$\mathbf{c} = (\mathbf{M}_0\mathbf{M}_0^T)^{-1}\mathbf{M}_0^T\mathbf{h}. \tag{2-14}$$

If $\mathbf{M}_0$ is of full rank, Eq. (2-14) can be simplified as $\mathbf{c} = \mathbf{M}_0^{-1}\mathbf{h}$.

The simplest configuration of multiple cameras and multiple light sources consists of two cameras and two light sources. In this case we have

$$\begin{aligned} (\mathbf{o}_1 - \mathbf{v}_1) \times (\mathbf{c} - \mathbf{o}_1) \cdot (\mathbf{p} - \mathbf{c}) = 0, \\ (\mathbf{o}_2 - \mathbf{v}_2) \times (\mathbf{c} - \mathbf{o}_2) \cdot (\mathbf{p} - \mathbf{c}) = 0. \end{aligned} \tag{2-15}$$

As the optic axis is defined in Eq. (2-1), we can see that the optic axis is the intersection line of the two planes: $[(\mathbf{o}_1 - \mathbf{v}_1) \times (\mathbf{c} - \mathbf{o}_1)]$ and $[(\mathbf{o}_2 - \mathbf{v}_2) \times (\mathbf{c} - \mathbf{o}_2)]$. Hence, the optic axis and its direction can be expressed as:

$$\begin{aligned} \mathbf{s}_{norm} &= \frac{\mathbf{s}}{\|\mathbf{s}\|}, \\ \mathbf{s} &= [(\mathbf{o}_1 - \mathbf{v}_1) \times (\mathbf{c} - \mathbf{o}_1)] \times [(\mathbf{o}_2 - \mathbf{v}_2) \times (\mathbf{c} - \mathbf{o}_2)]. \end{aligned} \tag{2-16}$$

If $\mathbf{s} \neq 0$, the solution to Eq. (2-15) is:

$$\mathbf{p} - \mathbf{c} = K\mathbf{s}_{norm}. \tag{2-17}$$

From above discussions we can see that the reconstruction of the optic axis can be done without knowledge of eye parameters R, K, n_1. Also, the progress to solve the problem is linear, which saves much time than one camera and multiple light sources model.

Conclusion Above sections introduces the anatomy and features of the eye and the mathematical models of the eye and measuring system, which includes the cameras and light sources. The composition of one camera and two light sources is the simplest configuration that allows free head movement. Composition of at least two cameras and two light sources can lead to a linear system without knowledge of eye parameters such as refraction rates, the radius of the cornea curvature and the distance between the center of cornea curvature and pupil center K.

2-3 Camera Geometry

In computer vision, cameras are approximated by pinhole camera model, which can represent the process of imaging. The pinhole camera model assumes there is no lens but only a small pinhole as the principle point of the camera.

The pinhole camera model simplifies the imaging process by ignoring the distortion and the depth of field. This is a rough model and the closer the camera is to the pinhole camera model, the more accuracy and precision we can get from the simulation.

The imaging process of the camera model involves four coordinate frames. Next subsection will introduce the transformation among the four coordinate frames during the imaging process.

2-3-1 Projection Geometry

There are four coordinate systems involved in the imaging process. They are: world coordinate system (WCS), camera coordinate system (CCS), image coordinate system and pixel coordinate system respectively as shown in Figure 2-7. Camera coordinate system is denoted with a subscript C (x_C, y_C, z_C) and the world coordinate system is denoted with a subscript W (X_W, Y_W, Z_W). (x, y) represents the image coordinates system and (u, v) the pixel coordinate system. The distance between the camera coordinate system origin O_C and the image coordinate system origin o is the focal length. Camera parameters are described by intrinsic and extrinsic matrices. The extrinsic matrix represents the transformation process from world coordinate system to camera coordinate system, including the rotation and translation matrices. The intrinsic matrix shows the transformation from camera coordinate system to image coordinate system. The pixel coordinate system can be represented by linear combination of pixel coordinates up to a scaling factor. Let's denote the intrinsic matrix and extrinsic matrix as $\mathbf{A}$ and $\mathbf{E}$ respectively:

$$\mathbf{A} = \begin{bmatrix} f_u & \gamma & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2-18}$$

$$\mathbf{E} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}, \tag{2-19}$$

with

- f_u, f_v: the focal lengths in terms of pixel coordinate system respectively,
- γ: skew coefficient,
- (u_0, v_0): the offset of projection of camera coordinate system origin to image plane, which is also the center of distortion [11],
- $\mathbf{R} = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix} \in \mathbb{R}^{3\times3}$ the rotation matrix, indicating the orientation of the camera frame,
- $\mathbf{t} \in \mathbb{R}^3$ the translation matrix, which is defined as the difference between principle point of the camera and origin of the reference coordinate frame.

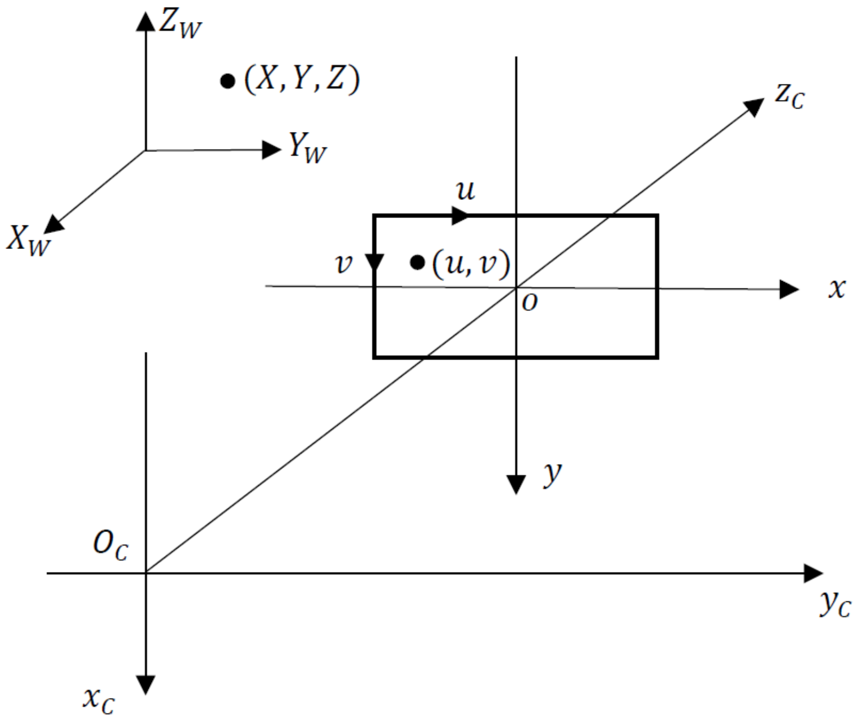


Figure 2-7: The transformation among three coordinate systems. Camera coordinate system: (x_C, y_C, z_C); World coordinate system: (X_W, Y_W, Z_W); Image coordinates system: (x, y); Pixel coordinate system: (u, v). The z axis of camera frame is perpendicular to the image plane. The origin of the image plane is the intersection point of z axis of camera coordinate frame and the image plane, which is o in the figure. The origin of the pixel coordinate system is usually defined as the upper left corner of the image in computer vision literature. The pixel point (u, v) corresponds to the point (X, Y, Z) in world coordinate frame.

Then the transformation from world coordinate system to image coordinate system is:

$$\begin{bmatrix} s \cdot u \\ s \cdot v \\ s \end{bmatrix} = \mathbf{AE} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{2-20}$$

with s the scaling factor between image coordinate system and pixel coordinate system. Adding a 1 at last row of the coordinates is just for calculation of matrices, which becomes homogeneous coordinates.

2-3-2 Reprojection Error

The reprojection error is an important performance evaluation of camera calibration.

Reprojection error is as shown in Figure 2-8. The reprojection error is expressed as the difference between image points $\mathbf{x}$ and $\mathbf{x}$'. $\mathbf{x}$ is the detected image point of feature point $\mathbf{M}_W$, which is expressed in world coordinate frame. $\mathbf{x}$' is the reprojection image point of $\mathbf{M}_W$ based on camera parameters that obtained from calibration procedure.

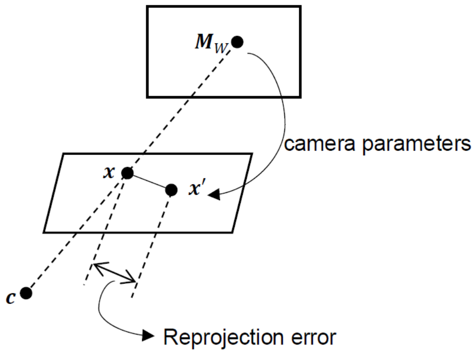


Figure 2-8: Reprojection error is shown as the absolute difference between image points $\mathbf{x}$ and $\mathbf{x}$'. $\mathbf{x}$ is the detected image point of feature point $\mathbf{M}_W$, which is expressed in world coordinate frame. $\mathbf{x}$' is the reprojection image point of $\mathbf{M}_W$ based on camera parameters that obtained from calibration procedure.

The optimization problem of minimizing reprojection error is expressed as the sum of squared distance between observation and estimation of image points.

2-3-3 Problem of Calibration

In the following, denote the observation point on image as $\mathbf{m}$ and the 3D object point as $\mathbf{M}$ [12].

Homography is defined as the mapping from object plane to image plane [12]. Homography for planar object calibration describes the transformation relationship from 3D object plane to image plane, which contains the information of rotation and translation relationships between the two planes.

Hence, the transformation from 3D world coordinate system to image coordinate system can be expressed as:

$$\mathbf{m} = \mathbf{H}\mathbf{M}, \tag{2-21}$$

up to a scale factor. Then the estimated projection point, which is the reprojection point, is obtained by:

$$\hat{\mathbf{m}} = \hat{\mathbf{H}}\hat{\mathbf{M}}. \tag{2-22}$$

Due to the unknown intrinsic and extrinsic parameters of camera and noise, there are errors between the detected and estimated image points. We can form a least squares problem, that is minimizing the reprojection error:

$$\min \|\mathbf{m} - \hat{\mathbf{m}}\|_2^2, \tag{2-23}$$

such that the optimization solution of homography $\hat{\mathbf{H}}$ and estimated projection image points $\hat{\mathbf{m}}$ can be obtained. Subsequently intrinsic and extrinsic parameters of cameras can be extracted from homography.

The calibration for multi-camera system is important and affects the reliability of the measurement system. Multi-camera calibration is usually based on single camera calibration. We will first talk about single camera calibration.

Chapter 3

Methods

This chapter introduces the methods for camera calibration and light source calibration.

3-1 Monocular Vision Calibration

Zhang's [12] calibration method is flexible and widely used in many calibration methods [13, 14]. Zhang's calibration method requires a chessboard attached to a flat surface and multiple images of different depth and orientations of the calibration pattern. The problem solved by calibration is to obtain the intrinsic and extrinsic parameters of the camera given one or more images of the calibration pattern.

Homography The transformation from world coordinate system to camera coordinate system is described by the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$. With camera intrinsic matrix $\mathbf{A}$, object points in world coordinate system is transformed into image coordinate system:

$$s\tilde{\mathbf{m}} = \mathbf{A}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\tilde{\mathbf{M}} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\tilde{\mathbf{M}}, \tag{3-1}$$

in which s is an arbitrary scale factor, $\tilde{\mathbf{m}}$ represents the image point and $\tilde{\mathbf{M}}$ the object point, $\mathbf{r}_i$ the ith column of the rotation matrix $\mathbf{R}$.

Assume the chessboard is on the plane $Z = 0$. Denote the homogeneous coordinates of the image point and the object point as $[u, v, 1]^T, [X, Y, Z, 1]^T$ respectively.

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\0\\1\end{bmatrix} \tag{3-2}$$

$$= \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\1\end{bmatrix}. \tag{3-3}$$

Denote homography as

$$\mathbf{H} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{3-4}$$

and Eq. (3-1) can be expressed by homography:

$$s\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathbf{M}}. \tag{3-5}$$

Given one homography there are two basic constraints on intrinsic parameters.According to Eq. (3-4), we can obtain

$$\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}, \tag{3-6}$$

with λ an arbitrary scalar. Using the knowledge that $\mathbf{r}_1$ and $\mathbf{r}_2$ are orthonormal, which means $\mathbf{r}_1$ and $\mathbf{r}_2$ are orthogonal and normalized,

$$\mathbf{r}_1^T \cdot \mathbf{r}_2 = 0, \tag{3-7}$$

$$\mathbf{r}_1^T \cdot \mathbf{r}_1 = \mathbf{r}_2^T \cdot \mathbf{r}_2 = 1, \tag{3-8}$$

we have

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 = 0, \tag{3-9}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2. \tag{3-10}$$

Closed-form Solution

Zhang [12] introduces the closed-form solution of intrinsic and extrinsic parameters.

Denote

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} \equiv \begin{bmatrix}B_{11} & B_{12} & B_{13}\\ B_{21} & B_{22} & B_{23}\\ B_{31} & B_{32} & B_{33}\end{bmatrix} \tag{3-11}$$

$$= \begin{bmatrix}\frac{1}{f_u^2} & -\frac{\gamma}{f_u^2 f_v} & \frac{v_0\gamma - u_0 f_v}{f_u^2 f_v}\\ -\frac{\gamma}{f_u^2 f_v} & \frac{\gamma^2}{f_u^2 f_v^2} + \frac{1}{f_v^2} & -\frac{\gamma(v_0\gamma - u_0 f_v)}{f_u^2 f_v^2} - \frac{v_0}{f_v^2}\\ \frac{v_0\gamma - u_0 f_v}{f_u^2 f_v} & -\frac{\gamma(v_0\gamma - u_0 f_v)}{f_u^2 f_v^2} - \frac{v_0}{f_v^2} & \frac{(v_0\gamma - u_0 f_v)^2}{f_u^2 f_v} + \frac{v_0^2}{f_v^2} + 1\end{bmatrix}. \tag{3-12}$$

Since $\mathbf{B}$ is symmetric, define a 6D vector:

$$\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T. \tag{3-13}$$

Let the ith column of $\mathbf{H}$ be $\mathbf{h}_i = [h_{i1}, h_{i2}, h_{i3}]^T$. Then

$$\mathbf{h}_i^T\mathbf{B}\mathbf{h}_j = \mathbf{v}_{ij}^T\mathbf{b}, \tag{3-14}$$

with

$$\mathbf{v}_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T. \tag{3-15}$$

We can rewrite constraints (3-9) and (3-10) as

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = 0. \tag{3-16}$$

If n images are provided, by stacking n such equations as (3-16) we have

$$\mathbf{V}\mathbf{b} = 0, \tag{3-17}$$

with $\mathbf{V}$ a $2n \times 6$ matrix.

- $n = 1$: In this case we can only solve two camera intrinsic parameters f_u, f_v under the assumption that the principle point of camera (u_0, v_0) is known and $\gamma = 0$.
- $n = 2$: Skew $\gamma = 0$ constraint is imposed.
- $n \geq 3$: In this case a unique solution of $\mathbf{b}$ can be obtained up to a scale factor.

The solution of equation (3-17) is well known as the eigenvector of $\mathbf{V}^T\mathbf{V}$ associated with the smallest eigenvalue.

Once $\mathbf{b}$ is estimated, the intrinsic matrix $\mathbf{A}$ is solved and the extrinsic matrix can also be computed:

$$\begin{aligned} \mathbf{r}_1 &= \lambda \mathbf{A}^{-1}\mathbf{h}_1, \\ \mathbf{r}_2 &= \lambda \mathbf{A}^{-1}\mathbf{h}_2, \\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2, \\ \mathbf{t} &= \lambda \mathbf{A}^{-1}\mathbf{h}_3, \end{aligned}$$

with $\lambda = \frac{1}{\|\mathbf{A}^{-1}\mathbf{h}_1\|}$.

Maximum Likelihood Estimation

The results are refined by maximum likelihood inference. Given n images of a model plane and m points on the model plane. By minimizing the function:

$$\sum_{i=1}^{N} \sum_{j=1}^{L} \|\mathbf{m}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)\|^2, \tag{3-18}$$

with $\hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ the estimated projection point in image i of point $\mathbf{M}_j$, the maximum likelihood estimation can be obtained. The rotation matrix $\mathbf{R}$ is parameterized by a vector $\mathbf{r}$ of 3 parameters, which is parallel to the rotation axis and whose magnitude equals to the rotation angle. $\mathbf{R}$ and $\mathbf{r}$ are related by the Rodrigues formula.

It's a nonlinear problem to minimize Eq. (3-18) and can be solved with the Levenberg-Marquardt algorithm. This process requires an initial guess of intrinsic and extrinsic matrices as described in closed-form solution [12].

Radial Distortion

In the imaging process, distortion deviates image points from their original positions and causes aberration. This is unavoidable and radial distortion dominates the distortion function [15, 16]. The radial distortion model introduced by Brown [17] is expressed as

$$u^d = u + u(k_1 r^2 + k_2 r^4) \tag{3-19}$$

$$v^d = v + v(k_1 r^2 + k_2 r^4), \tag{3-20}$$

where (u, v) is the distortion-free normalized image coordinate (which is not observable) and (u^d, v^d) the distorted normalized image point (which is observable), k_1, k_2 first two terms of radial distortion and $r = \sqrt{u^2 + v^2}$.

Normalized image points are obtained with known intrinsic parameters of the camera. We can transform image points from image coordinate system to normalized image coordinate system:

$$\tilde{\mathbf{m}}_n = \mathbf{A}^{-1}\tilde{\mathbf{m}}, \tag{3-21}$$

where normalized image point is denoted as $\tilde{\mathbf{m}}_n = [u, v]^T$, intrinsic matrix as $\mathbf{A}$ and image point $\tilde{\mathbf{m}} = [x, y]^T$.

Similarly, distorted image points (x^d, y^d) can be obtained from ideal image points (x, y):

$$x^d = x + (x - u_0)(k_1 r^2 + k_2 r^4) \tag{3-22}$$

$$y^d = y + (y - v_0)(k_1 r^2 + k_2 r^4) \tag{3-23}$$

with (u_0, v_0) the center of distortion, which is included in intrinsic matrix $\mathbf{A}$.

For each image we have

$$\begin{bmatrix} (x - u_0)r^2 & (x - u_0)r^4 \\ (y - v_0)r^2 & (y - v_0)r^4 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} x^d - x \\ y^d - y \end{bmatrix} \tag{3-24}$$

When we are given m points in n images, we get $2mn$ equations stacked in matrix form as $\mathbf{Dk} = \mathbf{d}$, where $\mathbf{k} = [k_1, k_2]^T$. The linear least-squares solution is given by

$$\mathbf{k} = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{d} \tag{3-25}$$

Complete Maximum Likelihood Estimation

The parameters can be refined with new estimation of projection points $\breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)$:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|\mathbf{m}_{ij} - \breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)\|^2. \tag{3-26}$$

Correct Image Points

To correct the distorted image points, distortion coefficients should be calculated. However, as indicated in Eq. (3-19) and Eq. (3-20) it's a forward projection model and cannot to utilized directly [13]. A new distortion model which helps to obtain distortion-free image points from distorted ones is introduced [18][19]:

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{G} \begin{bmatrix} u_i^d + u_i^d(a_1 r_i^2 + a_2 r_i^4) + 2a_3 u_i^d v_i^d + a_4(r_i^2 + 2u_i^{d^2}) \\ v_i^d + v_i^d(a_1 r_i^2 + a_2 r_i^4) + a_3(r_i^2 + 2v_i^{d^2}) + 2a_4 u_i^d v_i^d \end{bmatrix}, \tag{3-27}$$

and

$$G = (a_5 r_i^2 + a_6 u_i^d + a_7 v_i^d + a_8) r_i^2 + 1, \tag{3-28}$$

with (u_i^d, v_i^d) observed image points which are distorted in normalized image plane and $r_i = \sqrt{u^{d^2} + v^{d^2}}$.

In the distortion model Eq. (3-27) both radial and tangential distortion are considered. The coefficients a_1, a_2 are radial distortion coefficients and a_3, a_4 for tangential distortion.

The model can be used to correct distorted image points as long as 8 parameters $a_1, \ldots, a_8$ are known. In order to solve these parameters, N tie-points (u_i, v_i) and (u_i^d, v_i^d) covering the whole image area are generated, in which points (u_i, v_i) are ideal and (u_i^d, v_i^d) are distorted image points using distortion coefficients from solutions of optimization problem Eq. (3-26).

Define

$$\mathbf{u}_i = [-u_i^d r_i^2, -u_i^d r_i^4, -2u_i^d v_i^d, -(r_i^2 + 2u_i^d), u_i r_i^4, u_i u_i^d r_i^2, u_i v_i^d r_i^2, u_i r_i^2]^T, \tag{3-29}$$

$$\mathbf{v}_i = [-v_i^d r_i^2, -v_i^d r_i^4, -(r_i^2 + 2v_i^d), -2u_i^d v_i^d, v_i r_i^4, v_i u_i^d r_i^2, v_i v_i^d r_i^2, v_i r_i^2]^T, \tag{3-30}$$

$$\mathbf{T} = [\mathbf{u}_1, \mathbf{v}_1, \ldots, \mathbf{u}_i, \mathbf{v}_i, \ldots, \mathbf{u}_N, \mathbf{v}_N]^T, \tag{3-31}$$

$$\mathbf{p} = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8]^T, \tag{3-32}$$

$$\mathbf{e} = [u_1^d - u_1, v_1^d - v_1, \ldots, u_i^d - u_i, v_i^d - v_i, \ldots, u_N^d - u_N, v_N^d - v_N]^T. \tag{3-33}$$

Using equations (3-27) and (3-28) the following relation is obtained:

$$\mathbf{e} = \mathbf{T}\mathbf{e}. \tag{3-34}$$

Then vector $\mathbf{p}$ can be estimated by solving the following equation:

$$\hat{\mathbf{p}} = (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{e}. \tag{3-35}$$

With parameter vector $\mathbf{p}$, the image points can be corrected with equations (3-27) and (3-28).

3-2 Binocular Vision Calibration

Structure Parameters

Two methods of computing structure parameters, which are rotation and translation matrices that transform right camera coordinate system to left camera coordinate system, are introduced.

Method 1

One is introduced by Cui [13]. The initial guess of structure parameters $\mathbf{R}, \mathbf{T}$ are obtained from the means of $\mathbf{R}_i, \mathbf{T}_i$ with

$$\begin{cases} \mathbf{R}_i & = \mathbf{R}_{r,i}\mathbf{R}_{l,i}^{-1} \\ \mathbf{T}_i & = \mathbf{t}_{r,i} - \mathbf{R}_{r,i}\mathbf{R}_{l,i}^{-1}\mathbf{t}_{l,i} \end{cases} \Longrightarrow \begin{cases} \mathbf{R} & = \frac{1}{N}\sum_{i=1}^{N}\mathbf{R}_i \\ \mathbf{T} & = \frac{1}{N}\sum_{i=1}^{N}\mathbf{T}_i \end{cases}, \tag{3-36}$$

given N images.

Given initial guesses, the structure parameters $\mathbf{R}, \mathbf{T}$ can be optimized by minimizing the cost function:

$$\sum_{i=1}^{N}\sum_{j=1}^{L}(\|\mathbf{m}_{l,i,j}^{d} - \mathbf{m}_{l,i,j}^{d}(\mathbf{A}_l, \mathbf{R}_{l,i}, \mathbf{t}_{l,i}, \mathbf{D}_l)\|^2 + \|\mathbf{m}_{r,i,j}^{d} - \mathbf{m}_{r,i,j}^{d}(\mathbf{A}_r, \mathbf{R}_{l,i}, \mathbf{t}_{l,i}, \mathbf{D}_r, \mathbf{R}, \mathbf{T})\|^2), \tag{3-37}$$

with N images and L feature points in every image. In Eq. (3-37) $\mathbf{R}_{r,i}$ and $\mathbf{t}_{r,i}$ can be estimated by $\mathbf{R}_{l,i}, \mathbf{t}_{l,i}, \mathbf{R}, \mathbf{T}$:

$$\begin{cases} \mathbf{R}_r & = \mathbf{R}\mathbf{R}_l \\ \mathbf{t}_r & = \mathbf{R}_r\mathbf{R}_l^{-1}\mathbf{t}_l + \mathbf{T} \end{cases}. \tag{3-38}$$

Centroid-Based Optimization Method

This method is introduced by Gai in [14].

Denote the rotation matrix as $\mathbf{R}_{l,i}$, $\mathbf{R}_{r,i}$ and the translation vector as $\mathbf{t}_{l,i}$, $\mathbf{t}_{r,i}$ of ith image for left and right camera coordinate system respectively, in which $i \in \{1, \ldots, N\}$ if we have N images. The transformation is described by the following equation:

$$\mathbf{m}_{nl,ij} = \mathbf{R}_{l,i}\mathbf{M}_j + \mathbf{t}_{l,i}, \tag{3-39}$$

$$\mathbf{m}_{nr,ij} = \mathbf{R}_{r,i}\mathbf{M}_j + \mathbf{t}_{r,i}, \tag{3-40}$$

in which $i \in \{1, \ldots, N\}$ is the index of N images and $j \in \{1, \ldots, L\}$ is the index of L feature points in every image, $\mathbf{m}_{nl,ij}, \mathbf{m}_{nr,ij}$ are image points of left and right images respectively, $\mathbf{M}_j$ the corresponding object point in world coordinate system.

Assume the origin of the stereo camera coordinate system is the principle point of left camera. Structure parameters, which include rotation matrix $\mathbf{R}$ and translation vector $\mathbf{T}$, describe the transformation from right camera coordinate system to left one as in Eq. (3-41).

$$\mathbf{m}_{nr,ij} = \mathbf{R}\mathbf{m}_{nl,ij} + \mathbf{T}. \tag{3-41}$$

Centroid-based optimization method is used to obtain structure parameters. Using observed image points $\mathbf{m}_{nl,ij}, \mathbf{m}_{nr,ij}$, centers of left and right images are

$$\bar{\mathbf{m}}_{cl} = \frac{1}{N}\frac{1}{L}\sum_{i=1}^{N}\sum_{j=1}^{L}\mathbf{m}_{nl,ij}, \tag{3-42}$$

$$\bar{\mathbf{m}}_{cr} = \frac{1}{N}\frac{1}{L}\sum_{i=1}^{N}\sum_{j=1}^{L}\mathbf{m}_{nr,ij}. \tag{3-43}$$

Transform left and right image points to centers-based coordinate systems:

$$\tilde{\mathbf{m}}_{nl,ij} = \mathbf{m}_{nl,ij} - \bar{\mathbf{m}}_{cl}, \tag{3-44}$$
$$\tilde{\mathbf{m}}_{nr,ij} = \mathbf{m}_{nr,ij} - \bar{\mathbf{m}}_{cr}. \tag{3-45}$$

Substitute Eq. (3-41) into (3-45), we can obtain

$$\begin{aligned}\tilde{\mathbf{m}}_{nr,ij} &= \mathbf{R}\mathbf{m}_{nl,ij} + \mathbf{T} - \frac{1}{N}\frac{1}{L}\sum_{i=1}^{N}\sum_{j=1}^{L}(\mathbf{R}\mathbf{m}_{nl,ij} + \mathbf{T}) && (3\text{-}46)\\ &= \mathbf{R}\tilde{\mathbf{m}}_{nl,ij}. && (3\text{-}47)\end{aligned}$$

By solving the problem

$$\min_{\mathbf{R}} \quad f(\mathbf{R}) = \|\mathbf{R}\tilde{\mathbf{m}}_{nr} - \tilde{\mathbf{m}}_{nl}\|_2^2, \tag{3-48}$$

we can get the solution $\mathbf{R}$ and subsequently $\mathbf{T} = \bar{\mathbf{m}}_{cr} - \mathbf{R}\bar{\mathbf{m}}_{cl}$.

3-2-1 Reconstruction In 3D

With observed image points and structure parameters from binocular vision calibration we can reconstruct object points in 3D by triangulation.

Define transformation matrices from world coordinate system to normalized image coordinate system:

$$\mathbf{P}_l = \begin{bmatrix}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times1}\end{bmatrix}, \tag{3-49}$$
$$\mathbf{P}_r = \begin{bmatrix}\mathbf{R} & \mathbf{T}\end{bmatrix}. \tag{3-50}$$

Transformation from world coordinate system to normalized image coordinate system with $\mathbf{P}_l, \mathbf{P}_r$ can be described as

$$\tilde{\mathbf{m}}_{nl} = \mathbf{P}_l\tilde{\mathbf{M}} \tag{3-51}$$
$$\tilde{\mathbf{m}}_{nr} = \mathbf{P}_r\tilde{\mathbf{M}} \tag{3-52}$$

Linear Triangulation

Known Eq. (3-51) and Eq. (3-52), we have

$$\tilde{\mathbf{m}}_{nl} \times \mathbf{P}_l\tilde{\mathbf{M}} = 0, \tag{3-53}$$
$$\tilde{\mathbf{m}}_{nr} \times \mathbf{P}_r\tilde{\mathbf{M}} = 0, \tag{3-54}$$

where $\tilde{\mathbf{m}}_{nl} = [u_l, v_l, 1]^T, \tilde{\mathbf{m}}_{nr} = [u_r, v_r, 1]^T$ are normalized image points.

For every image point pair $\tilde{\mathbf{m}}_{nl,ij} \leftrightarrow \tilde{\mathbf{m}}_{nr,ij}$ and their corresponding object point $\mathbf{M}_j$, Eq. (3-53) and Eq. (3-54) can be tackled into matrix format:

$$\underbrace{\begin{bmatrix} v_{l,ij}p_{31} - p_{21} & v_{l,ij}p_{32} - p22 & v_{l,ij}p_{33} - p_{23} & v_{l,ij}p_{34} - p_{24} \\ u_{l,ij}p_{31} - p_{21} & u_{l,ij}p_{32} - p22 & u_{l,ij}p_{33} - p_{23} & u_{l,ij}p_{34} - p_{24} \\ v_{r,ij}p_{31} - p_{21} & v_{r,ij}p_{32} - p22 & v_{r,ij}p_{33} - p_{23} & v_{r,ij}p_{34} - p_{24} \\ u_{r,ij}p_{31} - p_{21} & u_{r,ij}p_{32} - p22 & u_{r,ij}p_{33} - p_{23} & u_{r,ij}p_{34} - p_{24} \end{bmatrix}}_{\mathbf{A}_{ij}} \tilde{\mathbf{M}}_j = 0, \tag{3-55}$$

in which $i \in \{1, \dots, N\}, j \in \{1, \dots, L\}, \mathbf{m}_{nl,ij} = [u_{l,ij}, v_{l,ij}]^T, \mathbf{m}_{nr,ij} = [u_{r,ij}, v_{r,ij}]^T$.

The solution of the linear equation Eq. (3-55) is the object point $\mathbf{M}_j$ corresponding to the image pair $\tilde{\mathbf{m}}_{nl,ij} \leftrightarrow \tilde{\mathbf{m}}_{nr,ij}$.

Nonlinear Triangulation

Assume $\mathbf{P}_l, \mathbf{P}_r$ from Eq. (3-49) and Eq. (3-50) are error free. Then by solving the minimization problem [20]

$$\min_{\mathbf{M}_j} d(\tilde{\mathbf{m}}_{nl,ij}, \mathbf{P}_l\tilde{\mathbf{M}}_j)^2 + d(\tilde{\mathbf{m}}_{nr,ij}, \mathbf{P}_r\tilde{\mathbf{M}}_j)^2 \tag{3-56}$$

the 3D object point $\mathbf{M}_j$ corresponding to image pair $\tilde{\mathbf{m}}_{nl,ij} \leftrightarrow \tilde{\mathbf{m}}_{nr,ij}$ can be found.

3-2-2 Software Flow Diagram

Given N image frames, we can extract L feature points from every image frame. Denote the observed distorted image points as $\{\mathbf{m}_{ij}\}$, in which $i \in \{1, \dots, N\}, j \in \{1, \dots, L\}$. As shown in Figure 3-1, calibration starts from every camera.

As described in subsection 3-1, with known image points intrinsic and extrinsic parameters, which are intrinsic matrix$\mathbf{A}$, rotation and translation matrices $\{\mathbf{R}_i\}, \{\mathbf{t}_i\}$ for all frames, and distortion coefficients k_1, k_2, can be obtained using Zhang's [12] calibration method. On these bases distortion coefficients that transform distorted image points to distortion-free image points are calculated for correction of image points.

Structure parameters, rotation and translation matrices $\mathbf{R}, \mathbf{T}$ that transform right camera coordinate system to left one, can be obtained using results from calibration of left and right camera, as shown in Figure 3-2.

Evaluation of Reconstruction Results

Evaluation of reconstruction results is about the difference between original and estimated object points. The error criterion can be defined as the average Euclidean distance of original and estimated object points [13]:

$$E = \frac{1}{N}\frac{1}{L}\sum_{i=1}^{N}\sum_{j=1}^{L} d(\mathbf{M}_j, \mathbf{M}_{est,ij}), \tag{3-57}$$

which is the evaluation of the precision of calibration and reconstruction results.

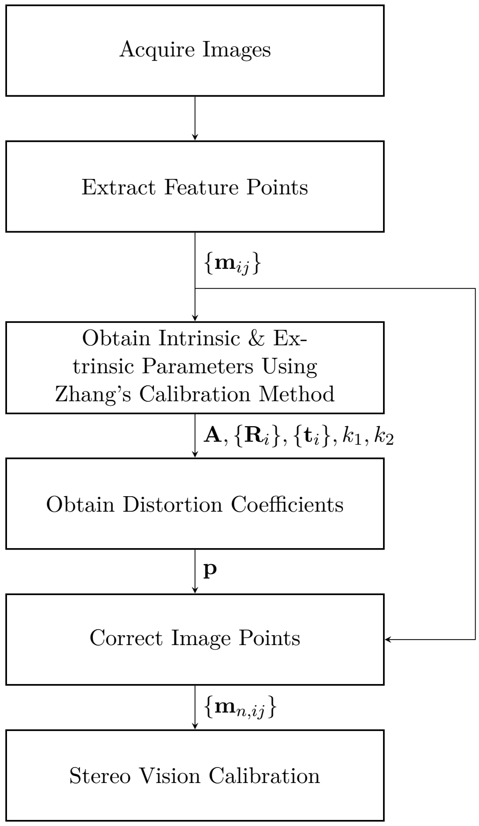


Figure 3-1: Single Camera Calibration Flowchart. Extract feature points $\{\mathbf{m}_{ij}\}$ in every image frame, in which $\mathbf{m}_{ij}$ indicates the jth feature point in ith image frame and $i \in \{1, \ldots, N\}, j \in \{1, \ldots, L\}$ if we have N image frames and L feature points in every frame. As described in subsection 3-1, with known image points intrinsic and extrinsic parameters, which are intrinsic matrix$\mathbf{A}$, rotation and translation matrices $\{\mathbf{R}_i\}, \{\mathbf{t}_i\}$ for all frames, and distortion coefficients k_1, k_2, can be obtained using Zhang's [12] calibration method. On these bases distortion coefficients that transform distorted image points to distortion-free image points are calculated for correction of image points.

3-3 Light Source Calibration

3-3-1 Simulation of System

The set-up of calibrating the light sources is shown in Figure 3-3. The combination of the glass sphere, whose radius is similar to human eye, and the chessboard makes it possible to calibrate cameras and light sources using the same target.

Denote the positions of two LED light sources as $\mathbf{L}_1, \mathbf{L}_2$. The two LED will generate two glints when they are close enough to the sphere.Glints are the reflection points of the LED light sources on the surface of the sphere.

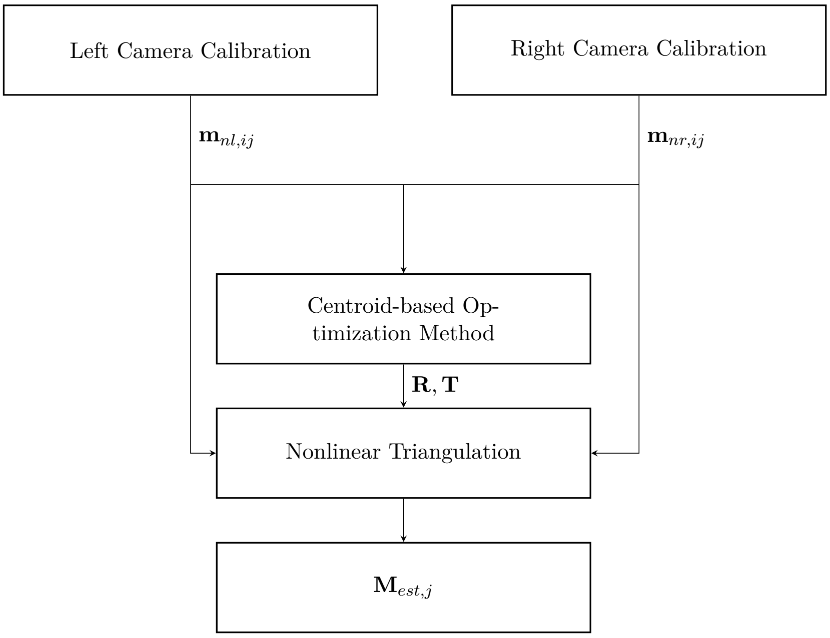


Figure 3-2: Stereo calibration flowchart. In stereo calibration procedure, results of left and right cameras are used to calculate the structure parameters that transform right camera coordinate system to left one. Finally estimation of 3D object points corresponding to image pair $\mathbf{m}_{nl,ij} \leftrightarrow \mathbf{m}_{nr,ij}$ can be obtained.

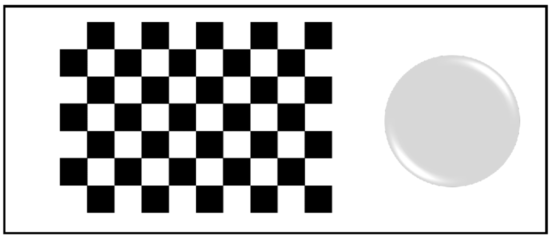


Figure 3-3: Calibration set-up for both cameras and light sources. Left side is a glass sphere, which is used to create the glints like human eyes. Right side is a chessboard, which can be used to calibrate cameras. The relative position of the sphere and the chessboard and the radius of the sphere are known.

3-3-2 Reconstruction of Light Source

While taking images for camera calibration, sphere and glints are included. If the positions of two LED light sources are not changed during the imaging process, they can be reconstructed by finding out the positions of the sphere's center and glints in 3D coordinate system.

In the world coordinate system, whose origin is the corner of the chessboard, the position of the sphere's center is known given the CAD structure of the lens as shown in Figure 3-4. The sphere's center is 1.6 mm higher than the plane $Z = 0$ of the board. The positions of the sphere's center in the plane $X - Y$ of the board is known when it's made.

Glints can be reconstructed from two camera views based on results from camera calibration.

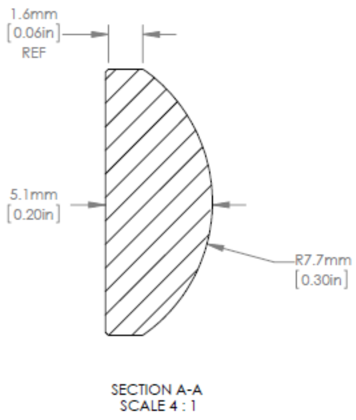


Figure 3-4: Details of the sphere lens in the board [21].

As defined in Figure 2-5, the glints, which are the point of reflections, are denoted as $\mathbf{q}_{ij}$, in which i denotes the index of the light sources and j the cameras. Each pair $(\mathbf{q}_{11}, \mathbf{q}_{12})$ and $(\mathbf{q}_{21}, \mathbf{q}_{22})$ can reconstruct their corresponding glints, which are denoted as $\mathbf{g}_1, \mathbf{g}_2$.

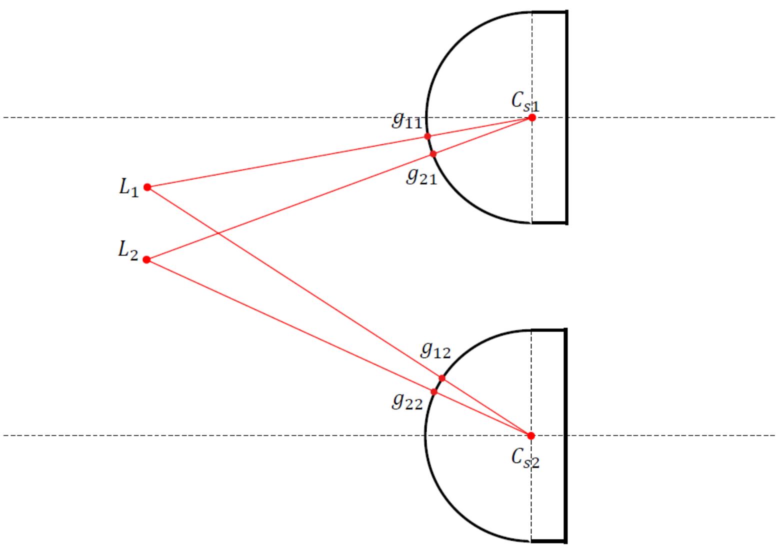


Figure 3-5: Reconstruction of light sources by intersection of 3D lines that pass through sphere's center and glints. At least two image frames of the sphere and glints are required to reconstruct light sources. As shown in the figure, the sphere's centers are denoted as $\mathbf{C}_{s1}, \mathbf{C}_{s2}$ for image 1 and 2 respectively. The glints are denoted as $\mathbf{g}_{ij}$, in which i denotes the index the light sources and j the image frames.

Denote the center of the sphere as $\mathbf{C}_s$. Then in every image frame the 3D line that passes through the sphere's center and one glint passes through the light source corresponding to the glint as shown in Figure 3-5. At lease two image frames of the sphere and glints are required to reconstruct light sources. As shown in the figure, the sphere's centers are denoted as $\mathbf{C}_{s1}, \mathbf{C}_{s2}$ for image 1 and 2 respectively. The glints are denoted as $\mathbf{g}_{ij}$, in which i denotes the index the light sources and j the image frames.

Assume the 3D coordinates of the spheres centers $\mathbf{C}_{s1}, \mathbf{C}_{s2}$ are $(x_{c1}, y_{c1}, z_{c1}), (x_{c2}, y_{c2}, z_{c2})$ and the glints $\mathbf{g}_{11}, \mathbf{g}_{12}$ are $(x_{g1}, y_{g1}, z_{g1}), (x_{g2}, y_{g2}, z_{g2})$ respectively. Then the 3D lines that pass through the sphere's center and glints can be expressed by Eq. (3-58) and Eq. (3-59):

$$\mathbf{l}_1 = (x_{g1} - x_{c1}, y_{g1} - y_{c1}, z_{g1} - z_{c1}), \tag{3-58}$$

$$\mathbf{l}_2 = (x_{g2} - x_{c2}, y_{g2} - y_{c2}, z_{g2} - z_{c2}). \tag{3-59}$$

The intersection of two lines $\mathbf{l}_1, \mathbf{l}_2$ is

$$\mathbf{L}_1 = \mathbf{C}_{s1} \pm \frac{\|\overrightarrow{\mathbf{C}_{s1}\mathbf{g}_{11}} \times \overrightarrow{\mathbf{C}_{s1}\mathbf{C}_{s2}}\|}{\|\overrightarrow{\mathbf{C}_{s1}\mathbf{g}_{11}} \times \overrightarrow{\mathbf{C}_{s2}\mathbf{g}_{12}}\|}\overrightarrow{\mathbf{C}_{s2}\mathbf{g}_{12}}, \tag{3-60}$$

in which $+$ is used when $\|(\overrightarrow{\mathbf{C}_{s1}\mathbf{g}_{11}} \times \overrightarrow{\mathbf{C}_{s1}\mathbf{C}_{s2}}) \times (\overrightarrow{\mathbf{C}_{s1}\mathbf{g}_{11}} \times \overrightarrow{\mathbf{C}_{s2}\mathbf{g}_{12}})\| = 0$.

Chapter 4

Results and Discussion

This chapter introduces how to realize camera calibration and light source reconstruction using theory from previous chapter.

4-1 Hardware Setup

The cameras and infrared LED light sources are introduced as in Table 4-1.

Table 4-1: Hardware Setup

Hardware $\times$ 2	Description
Monochrome Camera (UI-3140CP-M-GL Rev.2)	1280 $\times$ 1024 resolution with 224 fps(peak)
C-Mount Lens (Fujinon HF50HA-1B)	50mm focal length with 1.5 Megapixels
Infrared LED (SFH 4554)	860nm, Half Angle: 10 degree

4-2 Monocular Camera Calibration

There are multiple camera calibration toolboxes. Three mainly used toolboxes are shown as follows. Algorithms used in the three toolboxes are based on Zhang's calibration method [12] and Heikkila's four-step method [18].

4-2-1 Matlab Caltech Calibration Toolbox

This toolbox is developed by Bouguet [22]. The output of the toolbox is a list of all parameters of camera, including intrinsic and extrinsic parameters. Here's the introduction of intrinsic parameters:

- Focal length $[f_u, f_v] \in \mathbb{R}^2$: with units of pixels
- Principle point coordinates $[u_0, v_0] \in \mathbb{R}^2$: in pixels
- Skew coefficient γ: indicates the angle between x and y pixel axes
- Distortions: $\mathbf{D} \in \mathbb{R}^5$

The camera model is based on Zhang's calibration method and the skew coefficient is set to be zero: $\gamma = 0$, which is the default set of the toolbox. Hence the intrinsic matrix can be represented as:

$$\mathbf{A} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4-1}$$

Another default set is about the distortion coefficients $\mathbf{D}$. Based on Zhang's calibration method, the distortion order should not be over 4th order, which means $\mathbf{D}_5 = 0$, and the tangential distortion is discarded, which mean only the last three components of kc are set zero.

The extrinsic parameters outputs are the lists of rotation and translation matrices for every image loaded for calibration. In this toolbox, the transformation of two coordinate systems is from the grid reference frame to camera reference frame.

The calibration procedure starts from extracting corners of grid images. The fist chosen corner is defined as the origin of the grid reference frame, as shown in Figure 4-1. The uncertainties

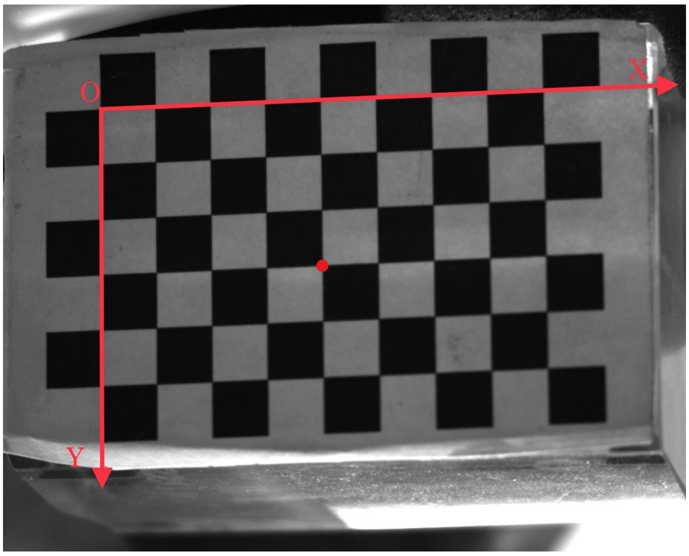


Figure 4-1: The grid reference frame. The origin is the first point that we choose at the beginning of the calibration. The Z axis is perpendicular to the X-Y plane shown in the figure. The point indicated in the figure is an example of feature points.

of intrinsic and extrinsic parameters are also outputs and represent approximately three times the standard deviations of the errors of estimation.

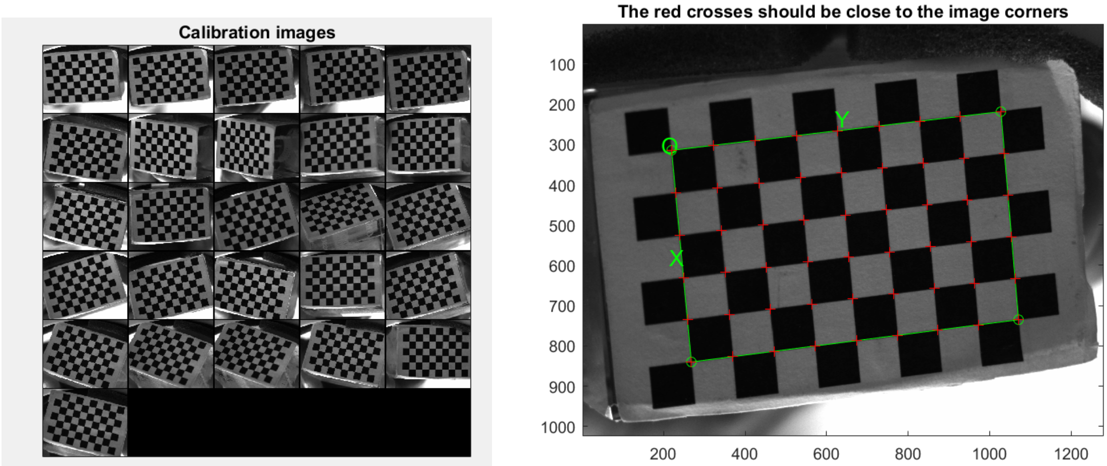


Figure 4-2: An example of using Caltech calibration toolbox. The left picture is showing al the examples that we use in the calibration toolbox. The right picture shows the detection of feature points in the image based on the coordinate frame that we chose from previous steps. The origin of the coordinate frame is the first click of feature point in the first image.

The calibration function computes the final intrinsic and extrinsic parameters by minimizing the reprojection error Eq. (3-18) [22]. An example is shown as in Figure 4-2.

Bouguet's algorithm has some drawbacks. This algorithm is highly dependent on the calibration image set (size and position of the calibration pattern, lighting conditions, camera quality, noise level and stability, and number of images) and errors occurs in the estimation of the principle point coordinates [23].

4-2-2 Matlab Computer Vision Toolbox

In this case the chessboard is required not to be square to enable the app to determine the orientation of the board.

The vectors in the Matlab toolbox and in Bouguet's [22] model are transpositions:

$$w\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} \\ \mathbf{t} \end{bmatrix} \mathbf{A},$$

$$\mathbf{A} = \begin{bmatrix} f_u & 0 & 0 \\ \gamma & f_v & 0 \\ u_0 & v_0 & 1 \end{bmatrix},$$

- (X, Y, Z) world coordinates of a point,
- (x, y) corresponding image point in pixels,
- w arbitrary homogeneous coordinates scale factor,
- R the rotation matrix representing the 3D rotation of camera,
- t the translation of camera relative to the world coordinate system,

- $\mathbf{A}$ the intrinsic matrix (assume skewness $\gamma = 0$),
- $f_u = Fs_x, f_v = Fs_y$: with F the focal length of camera and (s_x, s_y) the number of pixels per world unit in x and y direction respectively.

The toolbox estimates the intrinsic and extrinsic parameters and distortion coefficients in two steps:

- With distortion discarded, solve for the intrinsic and extrinsic parameters;
- By solving nonlinear least-squares minimization problem (minimizing the reprojection error) with Levenberg-Marquardt algorithm and initial estimation of all parameters from last step, all parameters are refined simultaneously.

4-2-3 OpenCV Standard Algorithm

This method is based on Zhang's calibration method [12] and Bouguet's toolbox [22]. Multiple flags that help adding constraints in calibration process can be utilized. Contributing flags are introduced as follows [24]:

- CALIB_USE_INTRINSIC_GUESS: Initial guess of intrinsic parameters f_x, f_y, u_0, v_0 need to be offered and to be optimized further using Eq. (3-18). Otherwise, the principle point (u_0, v_0) are set to the image center as initial guess by default.
- CALIB_FIX_PRINCIPAL_POINT: The principle point (u_0, v_0) will not change during optimization. It's fixed either at the center of the image or a position specified by setting flag CALIB_USE_INTRINSIC_GUESS, which means Eq. (3-18) can be written as

$$\min_{f_x, f_y, \mathbf{R}_i, \mathbf{t}_i} \sum_{i=1}^{N} \sum_{j=1}^{L} \|\mathbf{m}_{ij} - \hat{\mathbf{m}}(f_x, f_y, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)\|^2 \tag{4-2}$$

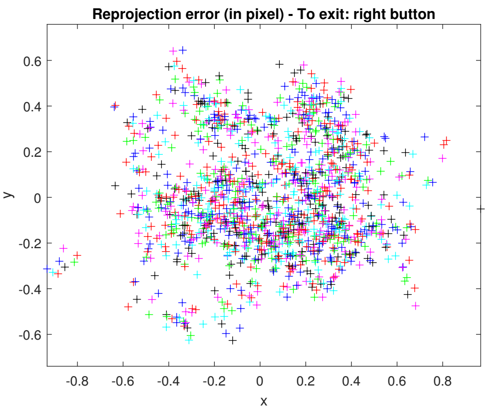
- CALIB_FIX_ASPECT_RATIO: When the flag CALIB_USE_INTRINSIC_GUESS is set and the initial guess of intrinsic parameters is given, the ratio f_x/f_y keeps the same as in the initial guess and f_y is considered as a free parameter. Otherwise, only the ratio is computed and used while the actual values of f_x, f_y are ignored.
- CALIB_ZERO_TANGENT_DIST: Tangential distortion coefficients are set to zeros and stay zero.

Flags CALIB_FIX_PRINCIPAL_POINT and CALIB_FIX_ASPECT_RATIO are not used. The principle point is set to the center of the image by default. However, due to it's also considered as the center of distortion, the principle point usually deviates from the center of the image [11]. If it's fixed, then the real center of distortion can be miscalculated. For the flag CALIB_FIX_ASPECT_RATIO, due to existence of distortion the focal length in different directions can be different and cannot be computed in high accuracy. Hence, the two flags cannot contribute to satisfying calibration results and only CALIB_USE_INTRINSIC_GUESS and CALIB_ZERO_TANGENT_DIST are used.

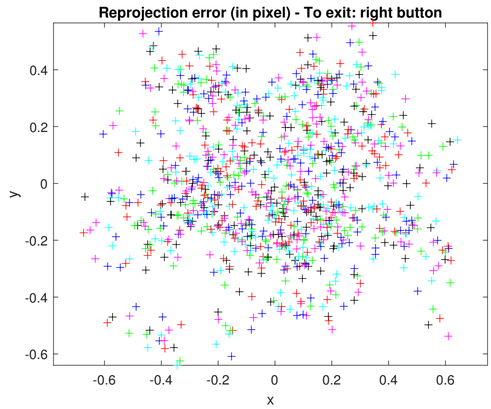
Initial guess of camera parameters can be given by calibration function using only flag CALIB_ZERO_TANGENT_DIST. Then optimization of parameters can be done by the second calibration function using flag CALIB_USE_INTRINSIC_GUESS.

4-2-4 Comparison of Toolboxes

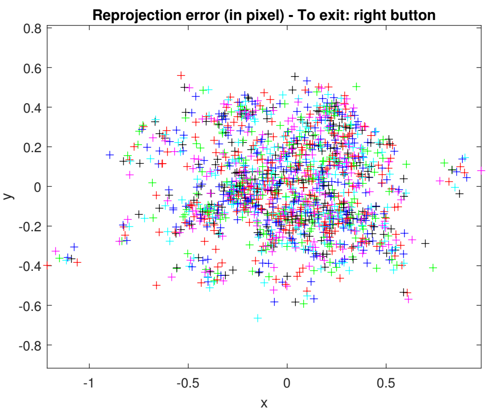
The evaluation of performances of the three toolboxes is based on reprojection error, which is introduced in chapter 2. Reprojection errors of two cameras using Caltech calibration toolbox [22], OpenCV algorithm and Matlab Calibrator app are shown as in Figure 4-3, Figure 4-4 and Figure 4-5 respectively. In every figure, for every camera reprojection errors of using the whole image set and the image set after suppressing several ill-conditioned images. Overall mean reprojection errors after removing images are smaller than those using the whole image set.



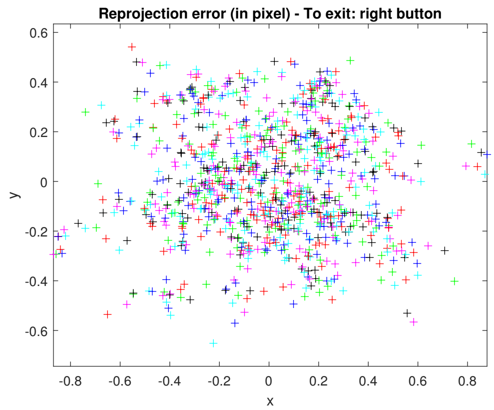
(a) Reprojection errors of camera 1 using the whole image set, which are 26 images. The mean reprojection errors are 0.29769, 0.23850 on x and y axes respectively. The mean reprojection error for all images is 0.381 pixels.



(b) Reprojection errors of camera 1 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.27734, 0.23871 on x and y axes respectively. The mean reprojection error for all images is 0.366 pixels.

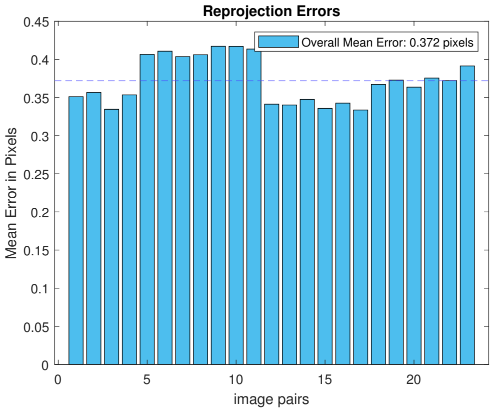


(c) Reprojection errors of camera 2 using the whole image set, which are 26 images. The mean reprojection errors are 0.32800 0.23423 on x and y axes respectively. The mean reprojection error for all images is 0.403 pixels.

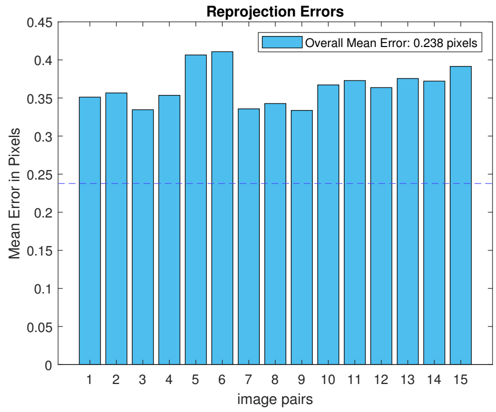


(d) Reprojection errors of camera 2 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.30247, 0.22964 on x and y axes respectively. The mean reprojection error for all images is 0.380 pixels.

Figure 4-3: Reprojection errors of two cameras using Matlab Caltech Camera Calibration Toolbox [22]. The overall mean reprojection error is 0.4 pixels.

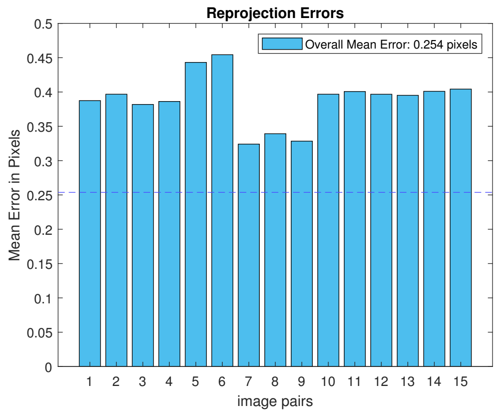


(a) Reprojection errors of camera 1 using the whole image set, which are 26 images. The mean reprojection errors are 0.3290, 0.3495 on x and y axes respectively. The mean reprojection error for all images is 0.372 pixels.



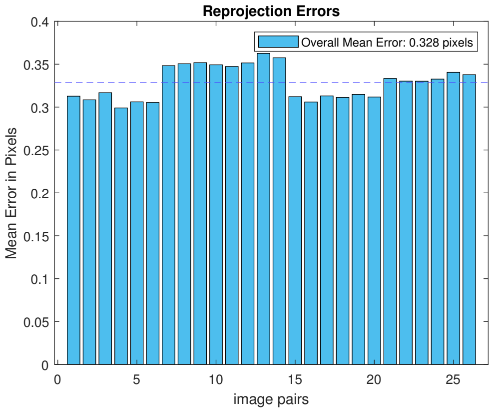
(b) Reprojection errors of camera 2 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.2103, 0.2245 on x and y axes respectively. The mean reprojection error for all images is 0.238 pixels.

(c) Reprojection errors of camera 1 using the whole image set, which are 26 images. The mean reprojection errors are 0.29769, 0.23850 on x and y axes respectively. The mean reprojection error for all images is 0.395 pixels.

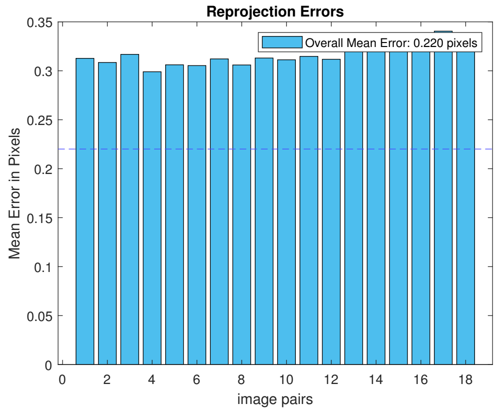


(d) Reprojection errors of camera 2 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.30247, 0.22964 on x and y axes respectively. The mean reprojection error for all images is 0.254 pixels.

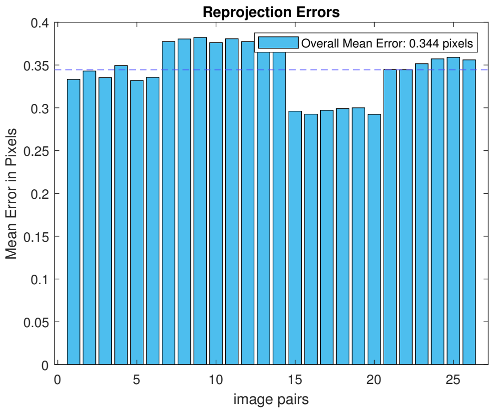
Figure 4-4: Reprojection errors of two cameras using OpenCV standard method [24]. The overall mean reprojection error is 0.315 pixels.



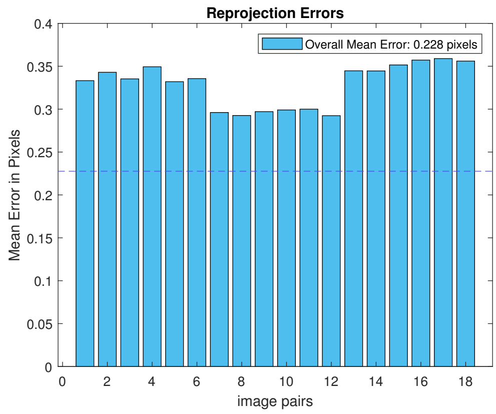
(a) Reprojection errors of camera 1 using the whole image set, which are 26 images. The mean reprojection errors are 0.3285, 0.3444 on x and y axes respectively. The mean reprojection error for all images is 0.328 pixels.



(b) Reprojection errors of camera 1 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.3285, 0.3444 on x and y axes respectively. The mean reprojection error for all images is 0.220 pixels.



(c) Reprojection errors of camera 1 using the whole image set, which are 26 images. The mean reprojection errors are 0.2201, 0.2276 on x and y axes respectively. The mean reprojection error for all images is 0.344 pixels.



(d) Reprojection errors of camera 2 after suppressing 8 ill-conditioned images (7-14), which are 18 images. The mean reprojection errors are 0.2201, 0.2276 on x and y axes respectively. The mean reprojection error for all images is 0.228 pixels.

Figure 4-5: Reprojection errors of two cameras using Matlab Computer Vision Toolbox Calibrator App [25]. The overall mean reprojection error is 0.28 pixels.

Table 4-2: Reprojection Errors of Cameras Using Different Toolboxes.

Reprojection Error (pixels)		Camera 1	Camera 2
Caltech Calibration Toolbox	first	0.381	0.403
	second	0.366	0.380
OpenCV Algorithm	first	0.372	0.395
	second	0.238	0.254
Matlab Calibrator App	first	0.328	0.344
	second	0.220	0.228

Comparing reprojection errors of three toolboxes listed in Table 4-2, Matlab Calibrator app gives the best performance for both cameras and before and after removing ill-conditioned images.

4-3 Stereo Camera Calibration

Two methods of computing structure parameters are introduced in chapter 3. Reconstruction errors using these two methods are shown as in Figure 4-6. The first row shows the reconstruction errors of the whole image set (left) and the image set after removing ill-conditioned images (right, the same images are removed as in single camera calibration) based on OpenCV. The second row shows the reconstruction errors using Matlab.

Performances of two different methods computing structure parameters are compared in 4-7. Left figure shows the results of Gai's method [14] and the right one shows the results of Cui's method [13]. As can be seen from the figures, Cui's method gives better performance, which is 0.19mm, than Gai's method, which is 0.22mm.

As shown in Figure 4-8, this method gives a good performance of reconstruction. Only points at the edges show that the distortion cannot be removed thoroughly.

4-3-1 Reconstruction of Light Sources

To reconstruct the light sources, the images should contain the the sphere and glints as shown in Figure 4-9. In Figure 4-9 the images of the calibration board have the same pose, which means the extrinsic parameters for these two images are the same. The chessboard in Figure 4-9a helps to compute the extrinsic parameters. Without illumination in Figure 4-9b highlights on the surface of the sphere can be removed and thus helps recognition of real glints.

As shown in Figure 4-10, there are multiple highlights in the images and real reflection points of light sources should be recognized first such that the positions of the light sources can be computed. In the right figure of Figure 4-10, the selected glints are shown in the red rectangle.

The positions of light sources are computed based on the method mentioned in chapter 3. Results are shown in Figure 4-11, in which the blue and red circles represent the results of light source 1 and 2 respectively. Light source 1 corresponds to the left glint in Figure 4-10 and light source 2 the other.

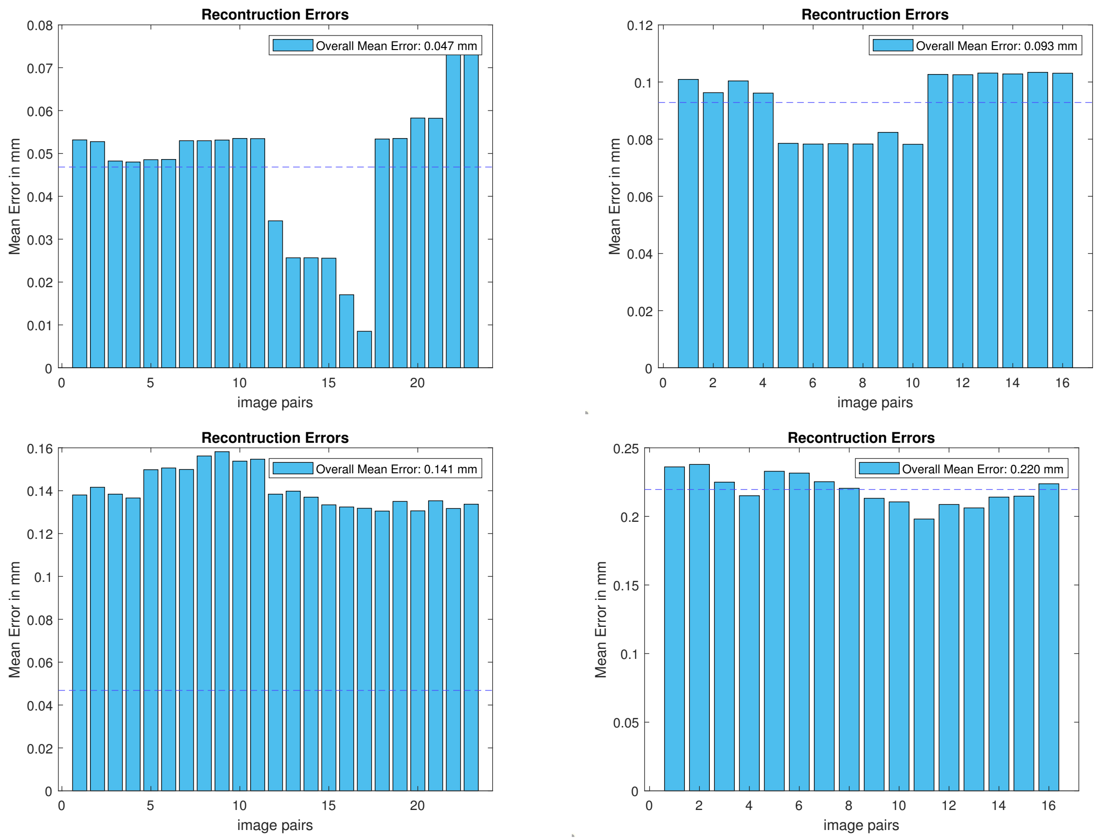


Figure 4-6: Reconstruction errors using OpenCV and Matlab. The first row shows the reconstruction errors of the whole image set (left) and the image set after removing ill-conditioned images (right, the same images are removed as in single camera calibration) based on OpenCV. The second row shows the reconstruction errors using Matlab.

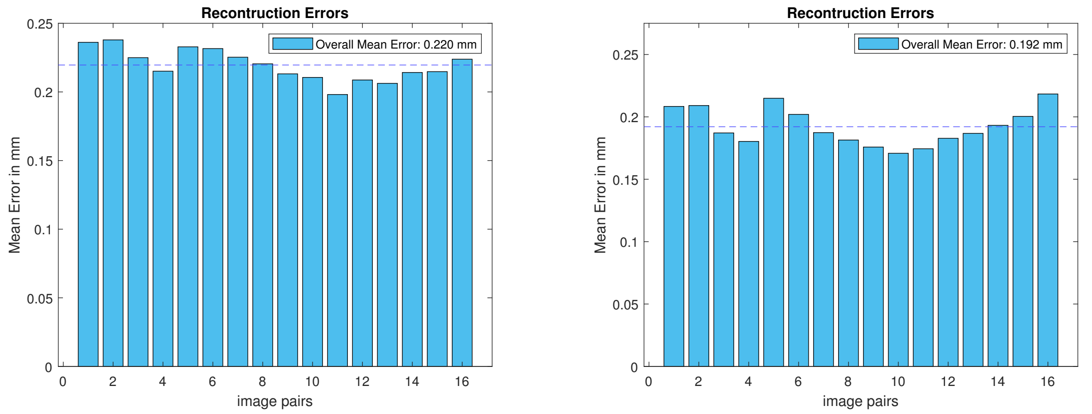


Figure 4-7: Results of two different methods of computing structure parameters while other parts are kept the same. Left figure shows the results of Gai's method [14] and the right one shows the results of Cui's method [13]. As can be seen from the figures, Cui's method gives better performance than Gai's method.

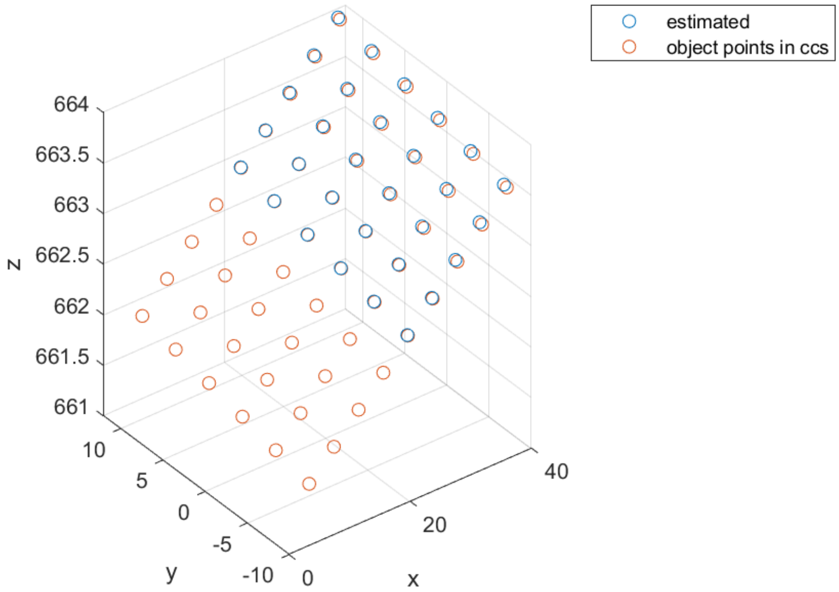
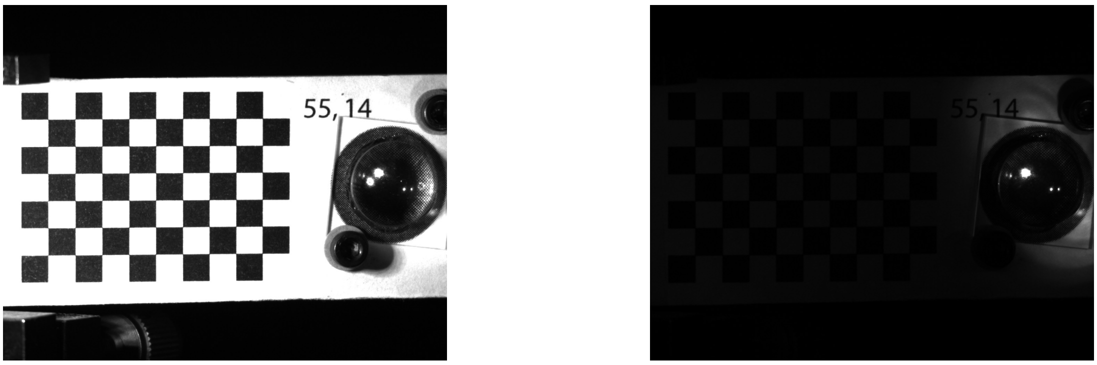


Figure 4-8: Reconstruction results. Points at the edge above show that there is still distortion.



(a) Image for calibration with illumination.

(b) Image for calibration without illumination.

Figure 4-9: Images for the board with the same pose. The difference between these two images is the illumination. The chessboard in Figure 4-9a helps to compute the extrinsic parameters. Without illumination in Figure 4-9b highlights on the surface of the sphere can be removed and thus helps recognition of real glints.

Discussion

As shown in Figure 4-11 the results of light sources positions vary a lot and no accurate and precise results can be obtained using this method. The distances from the first reconstructed light source to all the other reconstructed results.

Figure 4-10: Chosen glints on the surface of the sphere. The right figure shows the glints selected with the red rectangle. Other highlights are not real glints that we need.

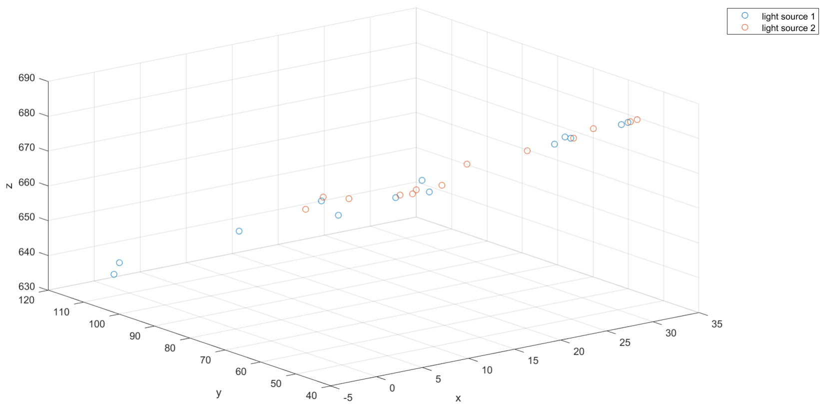


Figure 4-11: Estimation of the positions of light sources in 3D. As indicated in the figure, the blue and red circles represent the results of light source 1 and 2 respectively, in which light source 1 corresponds to the left glint in Figure 4-10 and light source 2 the other.

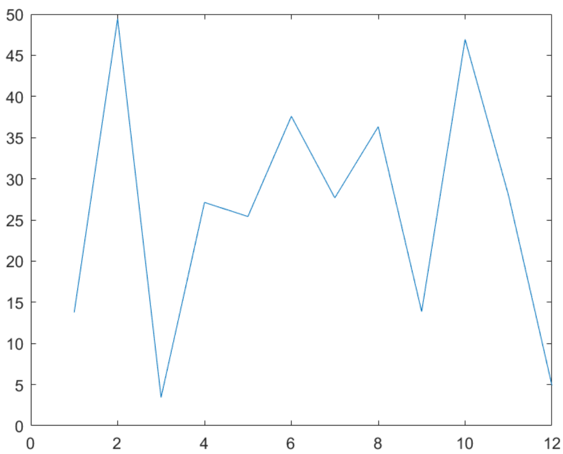


Figure 4-12: Distances of the reconstructed light source in the first image pair to all the other reconstructed light source points.

Chapter 5

Conclusion

In this chapter, results achieved are listed and future work are introduced.

5-1 Achievements

The results obtained during the thesis are listed as follows:

- In single camera calibration, three toolboxes were tested and novel distortion correction method were used.
- In stereo camera calibration, two methods of computing the structure parameters were tested and compared.
- Reconstruction of object points in 3D are realized with precision of 0.1 mm.
- Primary reconstruction of LED light sources was done but needs to be improved.

5-2 Future Work

To make a high-accuracy eye tracker, more work need to be done:

- Improve the method of reconstructing the positions of LED light sources.
- A way of improving the accuracy rather than precision should be evaluated.

Appendix A

Decoupled Single Camera Calibration Method

As introduced in chapter 2, calibration of a camera is to compute the intrinsic and extrinsic parameters given several image frames. All parameters are optimal solutions of maximum likelihood estimation.

However, this widely used method can cause errors. First is the iteration can end up with a local minimum. Second is the bundle adjustment of all parameters including distortion coefficients may lead to an unstable optimization problem of the procedure of iterations is not properly designed, which divergence or false solutions would be given [26].

Hence, a flexible method that decouple distortion parameters and camera intrinsic parameters is introduced to avoid problems mentioned above [27]. This method is introduced for reference.

A-1 Center of Distortion

Usually the center of distortion is defined as the center of the image, which is called principle point in Zhang's calibration method [12]. However, it's not safe due to distortion [11].

Radial distortion is considered as the main distortion model. Eq. (A-1) is used to express getting the distorted image point $\tilde{\mathbf{x}}^d$ from ideal image point $\tilde{\mathbf{x}}^u = [I|0]\widetilde{\mathbf{M}}$, in which $\tilde{\mathbf{e}}$ denotes the center of distortion.

$$\tilde{\mathbf{x}}^d = \tilde{\mathbf{e}} + \lambda(\tilde{\mathbf{x}}^u - \tilde{\mathbf{e}}). \tag{A-1}$$

Transform the image point $\tilde{\mathbf{x}}$ and the center of distortion into pixel coordinate system with camera intrinsic matrix $\mathbf{A}$:

$$\mathbf{x}^d = \mathbf{A}\tilde{\mathbf{x}}^d, \tag{A-2}$$

$$\mathbf{x}^u = \mathbf{A}\tilde{\mathbf{x}}^u, \tag{A-3}$$

$$\mathbf{e} = \mathbf{A}\tilde{\mathbf{e}}. \tag{A-4}$$

Thus:

$$\mathbf{x}^d = \mathbf{e} + \lambda(\mathbf{x}^u - \mathbf{e}). \tag{A-5}$$

Multiple $[\mathbf{e}]_\times$ on the left of Eq. (A-5) and Eq. (A-6) is obtained with $\mathbf{x}^u = \mathbf{HM}$, in which $\mathbf{M}$ is the object point on calibration chessboard.

$$[\mathbf{e}]_\times \mathbf{x}^d = \lambda[\mathbf{e}]_\times \mathbf{HM}. \tag{A-6}$$

Finally, multiply Eq. (A-6) on the left by $\mathbf{x}^{d^T}$ and Eq. (A-7) is obtained:

$$0 = \lambda \mathbf{x}^{d^T}([\mathbf{e}]_\times \mathbf{H})\mathbf{M} = \lambda \mathbf{x}^{d^T}\mathbf{FM}, \tag{A-7}$$

in which $\mathbf{F} = [\mathbf{e}]_\times \mathbf{H}$ is called as the fundamental matrix for radial distortion and the way of computing it is similar to usual way from several point correspondences [11].

The last step tp obtain the distortion center $\mathbf{e}$ is to extract left epipole of the fundamental matrix for radial distortion.

A-2 Distortion Coefficients and Homography

Once the center of distortion is obtained, transform the system into another that has its origin at the center of distortion, which means:

$$\check{\mathbf{e}} = \begin{bmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{e}} = \mathbf{T}\tilde{\mathbf{e}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{A-8}$$

if we define the original center of distortion as $\tilde{\mathbf{e}} = [u_0, v_0, 1]^T$.

Then the transformed image points, both distorted and not-distorted, can be described as:

$$\check{\mathbf{x}}^d = \mathbf{T}\mathbf{x}^d, \tag{A-9}$$

$$\check{\mathbf{x}}^u = \mathbf{T}\mathbf{x}^u. \tag{A-10}$$

With Eq. (A-7), the new fundamental matrix for radial distortion is:

$$\check{\mathbf{F}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ u_0 & v_0 & 1 \end{bmatrix} \mathbf{F}. \tag{A-11}$$

Since $\mathbf{F} = [\mathbf{e}]_\times \mathbf{H}$ and $\check{\mathbf{e}} = [0, 0, 1]^T$, we can obtain

$$\check{\mathbf{F}} = [\check{\mathbf{e}}]_\times \check{\mathbf{H}} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \check{\mathbf{H}}. \tag{A-12}$$

Let $\check{\mathbf{F}} = [\check{\mathbf{f}}_1^T; \check{\mathbf{f}}_2^T; \check{\mathbf{f}}_3^T]$ and $\check{\mathbf{H}} = [\check{\mathbf{h}}_1^T; \check{\mathbf{h}}_2^T; \check{\mathbf{h}}_3^T]$, where the semicolons stack every row $\check{\mathbf{f}}_i^T$ of $\check{\mathbf{F}}$ on top of each other. Based on Eq. (A-12) first two rows of $\check{\mathbf{H}}$ can be expressed by rows of the fundamental matrix for radial distortion. Given the fundamental matrix for radial distortion and the corresponding epipole, only the third row of the homography needs to be computed, which is $\check{\mathbf{H}} = [\check{\mathbf{f}}_2^T; -\check{\mathbf{f}}_1^T; \check{\mathbf{h}}_3^T]$.

To solve the final row of the homography, Li [28] and Yan [27] used the division model of radial distortion [29]. Express the undistorted image point $\check{\mathbf{x}}^u = [\check{x}^u, \check{y}^u]^T$ with distorted image point $\check{\mathbf{x}}^d = [\check{x}^d, \check{y}^d]^T$:

$$\check{x}^u = \frac{\check{x}^d}{L(\check{r}_d, k)}, \tag{A-13}$$

$$\check{y}^u = \frac{\check{y}^d}{L(\check{r}_d, k)}, \tag{A-14}$$

where $\check{r}_d = \sqrt{\check{x}_d^2 + \check{y}_d^2}$ and

$$L(\check{r}_d, k) = 1 + k_1 \check{r}_d^2 + k_2 \check{r}_d^4. \tag{A-15}$$

Substitute these parameters into Eq. (3-5) and we can get

$$s \begin{bmatrix} \frac{\check{x}^d}{L(\check{r}_d,k)} \\ \frac{\check{y}^d}{L(\check{r}_d,k)} \\ 1 \end{bmatrix} = \begin{bmatrix} \check{\mathbf{h}}_1^T \\ \check{\mathbf{h}}_2^T \\ \check{\mathbf{h}}_3^T \end{bmatrix} \mathbf{M}. \tag{A-16}$$

Hence, for each point the following equations can be obtained:

$$\begin{bmatrix} \check{x}_d(\check{\mathbf{h}}_3^T\mathbf{M}) - (\check{\mathbf{h}}_1^T\mathbf{M})(k_1\check{r}_d^2 + k_2\check{r}_d^4) \\ \check{y}_d(\check{\mathbf{h}}_3^T\mathbf{M}) - (\check{\mathbf{h}}_2^T\mathbf{M})(k_1\check{r}_d^2 + k_2\check{r}_d^4) \end{bmatrix} = \begin{bmatrix} \check{\mathbf{h}}_1^T\mathbf{M} \\ \check{\mathbf{h}}_2^T\mathbf{M} \end{bmatrix}, \tag{A-17}$$

$$\Longrightarrow \begin{bmatrix} \check{x}_d\mathbf{M}^T & (-\check{\mathbf{f}}_2^T\mathbf{M})[\check{r}_d^2 \quad \check{r}_d^4] \\ \check{y}_d\mathbf{M}^T & (\check{\mathbf{f}}_1^T\mathbf{M})[\check{r}_d^2 \quad \check{r}_d^4] \end{bmatrix} \begin{bmatrix} \check{\mathbf{h}}_3 \\ k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \check{\mathbf{f}}_2^T\mathbf{M} \\ -\check{\mathbf{f}}_1^T\mathbf{M} \end{bmatrix}. \tag{A-18}$$

For every image the homography and the distortion coefficients can be computed at the same time given n corresponding points which satisfy $2n \geq m + 3$ (m is the number of distortion coefficients to compute).

Another parameter-free method introduced by Hartley [11] ignored the distortion models and only considered two assumptions:

- The distortion is radially symmetric. Thus the radial distortion of an image point depends only on its distance from the center of distortion.
- An ordering, or monotonicity condition: the radial distance of points from the radial center after distortion is a monotonic function of their distance before distortion.

Denote the first two rows of the homography as $\hat{H}$ and the final row as $\mathbf{v}^T$, which make the homography as $\check{\mathbf{H}} = [\hat{H}; \mathbf{v}^T]$. Using Eq. (3-5) we can obtain

$$\mathbf{x}^u = [\hat{H}\mathbf{M}; \mathbf{v}^T\mathbf{M}] = \frac{(\hat{x}^u, \hat{y}^u)}{\mathbf{v}^T\mathbf{M}}, \tag{A-19}$$

$$\text{where} \quad \hat{\mathbf{x}}^u = (\hat{x}^u, \hat{y}^u)^T = \hat{H}\mathbf{M}. \tag{A-20}$$

Thus the effect of $\mathbf{v}^T$ is to stretch the point by the factor $\frac{1}{\mathbf{v}^T\mathbf{M}}$. Consequently the radii of the distorted and ideal image points are also stretched based on the two assumptions above, which is

$$r^u = \frac{\hat{r}^u}{\mathbf{v}^T\mathbf{M}}, \tag{A-21}$$

$$\text{where} \quad r^d = |\mathbf{x}^d|, \hat{r}^u = |\hat{\mathbf{x}}^u|, \tag{A-22}$$

$$r^u = |\frac{\hat{\mathbf{x}}}{\mathbf{v}^T\mathbf{M}}| = \frac{\hat{r}^u}{|\mathbf{v}^T\mathbf{M}|}. \tag{A-23}$$

Given n corresponding points, define the total squared variation of the functions Eq. (A-21) for every point in the same image:

$$V_t = \sum_{i=1}^{n-1} (r^u_{i+1} - r^u_i)^2 \tag{A-24}$$

$$= \sum_{i=1}^{n-1} \left(\frac{\hat{r}^u_{i+1}}{\mathbf{v}^T\mathbf{M}_{i+1}} - \frac{\hat{r}^u_i}{\mathbf{v}^T\mathbf{M}_i} \right)^2 \tag{A-25}$$

$$= \sum_{i=1}^{n-1} \left(\frac{\hat{r}^u_{i+1}\mathbf{v}^T\mathbf{M}_i - \hat{r}^u_i\mathbf{v}^T\mathbf{M}_{i+1}}{\mathbf{v}^T\mathbf{M}_{i+1}\mathbf{v}^T\mathbf{M}_i} \right)^2. \tag{A-26}$$

By solving the minimization of the function

$$V = \|A\mathbf{v}\|, \tag{A-27}$$

$$\text{subject to} \quad \|\mathbf{v}\| = 1, \tag{A-28}$$

$$\text{where} \quad A = \begin{bmatrix} \hat{r}^u_2\mathbf{M}^T_1 - \hat{r}^u_1\mathbf{M}^T_2 \\ \hat{r}^u_3\mathbf{M}^T_2 - \hat{r}^u_2\mathbf{M}^T_3 \\ \vdots \\ \hat{r}^u_n\mathbf{M}^T_{n-1} - \hat{r}^u_{n-1}\mathbf{M}^T_n \end{bmatrix} \tag{A-29}$$

for a single image, the final row $\mathbf{v}^T$ can be computed and thus the homography for this image is obtained.

Camera intrinsic parameters can be extracted from the homography using Zhang's method [12].

Appendix B

Illumination Safety

From data sheet of the infrared light source SFH 4554, we can get the following parameters:

- Peak wavelength $\lambda_{peak} = 860nm$
- The length and width of the active area of the light source $L = W = 0.3mm$
- Radiant intensity $I_e = 550mW/sr$

The exposure limits are given with several constraints [**?**]:

$$\text{exposure limits for cornea} \quad E_e = \frac{I_e}{d^2} \leq E_{IR} = \begin{cases} 18000 \cdot t^{-0.75} W/m^2 \\ 100W/m^2 \end{cases} \tag{B-1}$$

$$\text{exposure limits for retina} \quad \alpha = \frac{Z}{d} \in [\alpha_{min}, \alpha_{max}] \tag{B-2}$$

$$\text{with} \quad \alpha_{max} = 0.1rad, \tag{B-3}$$

$$\alpha_{min} = \begin{cases} 0.0017, & t \leq 0.25s \\ 0.0017 \cdot \sqrt{\frac{t}{0.25}}, & t \in (0.25, 10)s \\ 0.011, & t \geq 10s \end{cases} \tag{B-4}$$

$$\text{spectral radiance} \quad L_{IR} \approx I_e \frac{R(\lambda)}{Z^2} \leq \begin{cases} \frac{50000}{\alpha \cdot t^{0.25}}, & t < 10s \\ \frac{6000}{\alpha}, & t \geq 10s \end{cases} \tag{B-5}$$

with $Z = \frac{L+W}{2}$, d the distance between eye and light source, and $R(\lambda) = 10^{\frac{700-\lambda}{500}}$.

With known parameters we can first get

$$Z = \frac{L+W}{2} = 0.3mm, \quad R(\lambda_{peak}) = 10^{\frac{700-850}{500}} = 0.4786 \tag{B-6}$$

Then for worst case the distance d should satisfy:

$$E_e = \frac{0.55 \times 2}{d^2} \leq 100 \Rightarrow d \geq \sqrt{\frac{1.1}{100}} = 0.1049m \tag{B-7}$$

$$\frac{6000}{\frac{Z}{d}} = \frac{6000d}{0.3 \times 10^{-3}} \geq \frac{1.1 \times 0.4786}{(0.3 \times 10^{-3})^2} \Rightarrow d \geq \frac{1.1 \times 0.4786}{6000 \times 0.3 \times 10^{-3}} = 0.2925m \tag{B-8}$$

that is the distance from eye to the light source should be larger than $0.3m$ with 2 LED light sources.

Appendix C

Matlab Code

```
%% Initialization
clear all; clc;
data;
%%
L = featurepoints(3).featureindices;
N = featurepoints(3).imageindices;
Al = featurepoints(1).intrinsic;
Ar = featurepoints(2).intrinsic;
R = featurepoints(3).structure.rotation;
T = featurepoints(3).structure.translation;
kc_left = [featurepoints(1).distortion(1) featurepoints(1).distortion(2)
    ];
kc_right = [featurepoints(2).distortion(1) featurepoints(2).distortion(2)
    ];
% essential and fundamental matrices
[E,F] = Essential_Fundamental(Al,Ar,R,T);
[el,er] = Epipole(Al, Ar, R, T);
%% Step 1: Correct image points
% Get distortion coefficients for every image
for i = 1:N
    % normalize image points into normalized image plane
    tmp1 = Al\[featurepoints(1).image(i).imgcoordinates;ones(1,L)];
    featurepoints(1).image(i).normalized = tmp1(1:2, :);
    tmp2 = Ar\[featurepoints(2).image(i).imgcoordinates;ones(1,L)];
    featurepoints(2).image(i).normalized = tmp2(1:2, :);
end
% Collection of distortion coefficients P
P_left = zeros(8,N);
P_right = zeros(8,N);
for i = 1:N
    P_left(:,i) = Distortion_Coefficients(Al, kc_left);
    P_right(:,i) = Distortion_Coefficients(Ar, kc_right);
end
```

```
for i = 1:N
    % pixel coordinate system
    featurepoints(1).image(i).corrected = Correction_Imagepoints(
        featurepoints(1).image(i).normalized,P_left(:,i));
    featurepoints(2).image(i).corrected = Correction_Imagepoints(
        featurepoints(2).image(i).normalized,P_right(:,i));
end
%% Step 2:Triangulation
% linear triangulation --- not working
P1 = [eye(3) zeros(3,1)];
P2 = [R T];
% for i = 1:N
%     for j = 1:L
%         mnl = [featurepoints(1).image(i).corrected(:,j);1];
%         mnr = [featurepoints(2).image(i).corrected(:,j);1];
%         mnlx = [0 -mnl(3) mnl(2); mnl(3) 0 -mnl(1); -mnl(2) mnl(1) 0];
%         mnrx = [0 -mnr(3) mnr(2); mnr(3) 0 -mnr(1); -mnr(2) mnr(1) 0];
%         mm = [mnlx*P1; mnrx*P2]; [~,~,V] = svd(mm); sol = V(:,end);
%         featurepoints(4).image(i).est(:,j) = sol(1:3)/sol(4);
%     end
% end
% nonlinear triangulation
% triangulation
for i = 1:N
    for j = 1:L
        mnl = featurepoints(1).image(i).corrected(:,j);
        mnr = featurepoints(2).image(i).corrected(:,j);
        costtotal = @(x)CostTriangulation(x, R, T, mnl, mnr);
        x0 = featurepoints(4).image(i).ccs(:,j);
%         options.Algorithm = 'levenberg-marquardt';
%         options.MaxFunctionEvaluations = 5000;
%         x = lsqnonlin(costTri,x0,[],[],options);
        x_tri1 = fminunc(costtotal, x0);
        featurepoints(4).image(i).est(:,j) = x_tri1;
    end
end
%% Plot
% plot of mean error between estimated and observed of every image
x1 = 1:N; y1 = zeros(1,N);
for i = 1:N
    distance = 0;
    for j = 1:L
        tmpd2 = norm(featurepoints(4).image(i).ccs(:,j)-featurepoints(4).
            image(i).est(:,j));
        distance = distance + tmpd2;
    end
    y1(:,i) = distance/L;
end
mean3d = sum(y1)/N;
disp('Reconstruction Error: ');
disp(mean3d);
f1 = figure();
plot(x1,y1);
```

```
xlabel('image index');
ylabel('distance error (mm)');
title('Euclidean Distance of All Images');
f2 = figure();
xe = featurepoints(4).image(1).est(1,:);
ye = featurepoints(4).image(1).est(2,:);
ze = featurepoints(4).image(1).est(3,:);
xc = featurepoints(4).image(1).ccs(1,:);
yc = featurepoints(4).image(1).ccs(2,:);
zc = featurepoints(4).image(1).ccs(3,:);
scatter3(xe,ye,ze);
hold on;
scatter3(xc,yc,zc);
legend('estimated', 'object points in ccs');
xlabel('x');
ylabel('y');
zlabel('z');
% bar of reconstruction error
figure();
b4 = bar(y1, 'FaceColor', [0.3010 0.7450 0.9330]);
xlabel('image pairs');
ylabel('Mean Error in mm');
title('Recontruction Errors');
le4 = sprintf('Overall Mean Error: %.3f mm', mean3d);
line4 = yline(mean3d, '--b');
legend(b4, le4);
% plot relationship between xyz and reconstruction error
% f3 = figure();
% errx = zeros(1,N*L); erry = zeros(1,N*L); errz = zeros(1,N*L);
% xerr = zeros(1,N*L); yerr = zeros(1,N*L); zerr = zeros(1,N*L);
% for i = 1:N
%     errx((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).est(1,:) - featurepoints(4).image(i).ccs(1,:);
%     erry((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).est(2,:) - featurepoints(4).image(i).ccs(2,:);
%     errz((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).est(3,:) - featurepoints(4).image(i).ccs(3,:);
%     xerr((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).ccs(1,:);
%     yerr((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).ccs(2,:);
%     zerr((1+L*(i-1)):(L*i)) = featurepoints(4).image(i).ccs(3,:);
% end
% subplot(1,3,1); scatter(xerr, errx); xlabel('x'); ylabel('reconstruction error along x axis');
% subplot(1,3,2); scatter(yerr, erry); xlabel('y'); ylabel('reconstruction error along y axis');
% subplot(1,3,3); scatter(zerr, errz); xlabel('z'); ylabel('reconstruction error along z axis');
% scatter3(xerr, yerr, zerr);
% title('Reconstruction error along x, y, z axis respectively');

% save figures
% if flag == 0
%     saveas(f1, 're.png');
```

```
%     saveas(f2, 're3d.png');
% %     saveas(f3, 'rexyz.png');
% elseif flag == 1
%     saveas(f1, 're_15-08-19.png');
%     saveas(f2, 're3d_15-08-19.png');
% %     saveas(f3, 'rexyz_15-08-19.png');
% elseif flag == 2
%     saveas(f1, 're_10-07-19.png');
%     saveas(f2, 're3d_10-07-19.png');
% %     saveas(f3, 'rexyz_10-07-19.png');
% elseif flag == 3
%     saveas(f1, 're_16-07-19.png');
%     saveas(f2, 're3d_16-07-19.png');
% %     saveas(f3, 'rexyz_16-07-19.png');
% elseif flag == 4
%     saveas(f1, 're_09-09-19.png');
%     saveas(f2, 're3d_09-09-19.png');
% end


% Plot of reprojection error vs. depth
% depth = zeros(1,N); re = zeros(1,N); ind = 2;
% for i = 1:N
%     depth(i) = featurepoints(4).image(i).ccs(3,ind);
% %     reprojection error in normalized image plane
%     tmpr = 4.8e-3*featurepoints(4).image(i).ccs(:,ind);
%     re(i) = norm(featurepoints(1).image(i).normalized(:,ind)-tmpr);
% end
% figure();
% subplot(2,1,1);
% plot(xpt,ypt);
% xlabel('image');
% ylabel('distance error');
% title('Euclidean Distance of All Images');
% subplot(2,1,2);
% plot(depth,re,'*');
% xlabel('depth'); ylabel('reprojection error');
%% Optimization of 3d points
% costtotal = @(x)CostTotal(x, featurepoints);
% x0 = zeros(1,18+6*N);
% x0(1:6) = [Al(1,1); Al(2,2); Al(1,3); Al(2,3); kc_left'];
% for i = 1:N
%     x0((7+6*(i-1)):(9+6*(i-1))) = rotm2eul(featurepoints(1).image(i).rotation)';
%     x0((10+6*(i-1)):(12+6*(i-1))) = featurepoints(1).image(i).translation;
% end
% x0(7+6*N:12+6*N) = [Ar(1,1); Ar(2,2); Ar(1,3); Ar(2,3); kc_right'];
% x0(13+6*N:15+6*N) = rotm2eul(featurepoints(3).structure.rotation);
% x0(16+6*N:18+6*N) = featurepoints(3).structure.translation';
% options = optimoptions('lsqnonlin','Display','iter');
% options.Algorithm = 'levenberg-marquardt';
% options.MaxFunctionEvaluations = 5000;
```

```
% x_total = lsqnonlin(costtotal,x0,[],[],options);

%% reconstruct with optimized results
% Al = [x_total(1) 0 x_total(3); 0 x_total(2) x_total(4); 0 0 1];
% Dl = [x_total(5) x_total(6)];
% Ar = [x_total(7+6*N) 0 x_total(9+6*N); 0 x_total(8+6*N) x_total(10+6*N)
    ; 0 0 1];
% Dr = [x_total(11+6*N) x_total(12+6*N)];
% R = eul2rotm([x_total(13+6*N) x_total(14+6*N) x_total(15+6*N)]);
% T = [x_total(16+6*N); x_total(17+6*N); x_total(18+6*N)];
% M_wcs = featurepoints(1).image(1).objcoordinates; % 3L
% M_ccsl = zeros(3, N*L); M_ccsr = zeros(3, N*L); % objpoints in ccs
% M_est = zeros(3, N*L);
% pl = zeros(8,N); pr = zeros(8,N); % distortion coefficients
% for i = 1:N
%     rl = x_total((7+6*(i-1)):(9+6*(i-1)));
%     rl = eul2rotm(rl);
%     tl = x_total((10+6*(i-1)):(12+6*(i-1)))';
%     rr = R*rl;
%     tr = R*tl + T;
%     M_ccsl(:,(1+L*(i-1)):L*i) = WCS2CCS(M_wcs, rl, tl);
%     M_ccsr(:,(1+L*(i-1)):L*i) = WCS2CCS(M_wcs, rr, tr);
%     pl(:,i) = Distortion_Coefficients(Al, Dl);
%     pr(:,i) = Distortion_Coefficients(Ar, Dr);
% end
%
% for i = 1:N
%     % corrected normalized image points
%     % normalized image points
%     mnl = Al\[featurepoints(1).image(i).imgcoordinates;ones(1,L)];
%     mnr = Ar\[featurepoints(2).image(i).imgcoordinates;ones(1,L)];
%     ml = Correction_Imagepoints(mnl, pl(:,i));
%     mr = Correction_Imagepoints(mnr, pr(:,i));
%     for j = 1:L
%         costTri = @(x)CostTriangulation(x, R, T, ml(:,j), mr(:,j));
%         x0 = featurepoints(4).image(i).est(:,j);
%         x_tri2 = fminunc(costTri, x0);
%         M_est(:,j+L*(i-1)) = x_tri2;
%     end
% end
% %%
% x3 = 1:N; y3 = zeros(1,N);
% for i = 1:N
%     tmp = M_est(:, 1+L*(i-1):L*i)-M_ccsl(:, 1+L*(i-1):L*i);
%     y3(:,i) =sum(vecnorm(tmp))/L;
% end
% mean3d = sum(y3)/N;
% disp('Reconstruction Error: ');
% disp(mean3d);
% f3 = figure();
% plot(x3,y3);
% xlabel('image index');
% ylabel('distance error (mm)');
```

```
% title('Euclidean Distance of All Images');
```

```
% This combines all calibration results of caltech calibration toolbox into
% a struct featurepoints, which contains:
% featurepoints(1): left camera
%                   intrinsic, distortion
% featurepoints(1).image:
%                   imgcoordinates: feature points coordinates of every image
%                   objcoordinates: object points coordinates of wcs
%                   rotation: rotation matrix for every image
%                   translation: translation matrix for every image
%                   spacepoints: corresponding 3d space points for every
%                                image
%                   corrected: corrected image points
%                   normalized: normalized image points in homogeneous
% featurepoints(2): right camera
%                   intrinsic, distortion
% featurepoints(2).image:
%                   coordinates: feature points coordinates of every image
%                   rotation: rotation matrix for every image
%                   translation: translation matrix for every image
%                   spacepoints: corresponding 3d space points for every
%                                image
%                   corrected: corrected image points
%                   normalized: normalized image points in homogeneous
% featurepoints(3):
%                   imageindices: indices of image for every camera
%                   featureindices: indices of feature points on every
%                                   image
%                   width: board width
%                   structure:
%                             rotation: rotation matrix from left camera
%                                       frame to right camera
%                             translation: translation matrix from left
%                                          camera frame to right camera
% featurepoints(4).image(i).ccs: space points in camera coordinate system
%                          .est: estimated space points in ccs
%                          .meanerror: mean error between estimated and
%                                      observed space points
%% load data
flag = 0; width = 4;
fprintf('0: the only one with board width = 5mm \n1: 15-08-19 \n2: 10-07-19 \n3: 16-07-19\n4: 09-09-19\n');
prompt = 'Choose an image set: ';
x = input(prompt);
if x == 0
    d = fullfile(fileparts(pwd), 'opencv', 'calibration', {'calibLeft.mat'; 'calibRight.mat'});
    width = 5;
elseif x == 1
```

```matlab
    flag = 1;
    d = fullfile(fileparts(pwd), 'opencv', 'calibration', {'calibLeft_15-08-19.mat'; 'calibRight_15-08-19.mat'});
elseif x == 2
    flag = 2;
    d = fullfile(fileparts(pwd), 'opencv', 'calibration', {'calibLeft_10-07-19.mat'; 'calibRight_10-07-19.mat'});
elseif x == 3
    flag = 3;
    d = fullfile(fileparts(pwd), 'opencv', 'calibration', {'calibLeft_16-07-19.mat'; 'calibRight_16-07-19.mat'});
elseif x == 4
    flag = 4;
    d = fullfile(fileparts(pwd), 'opencv', 'calibration', {'calibLeft_09-09-19.mat'; 'calibRight_09-09-19.mat'});
end
m1 = load(d{1});
m2 = load(d{2});
%% combine all image feature points coordinates
featurepoints(1).name = 'left';
featurepoints(2).name = 'right';
featurepoints(1).intrinsic = m1.cameraMatrix;
featurepoints(2).intrinsic = m2.cameraMatrix;
featurepoints(1).distortion = m1.distCoeffs;
featurepoints(2).distortion = m2.distCoeffs;
featurepoints(3).imageindices = size(m1.imgpoints,1);
featurepoints(3).featureindices = size(m1.imgpoints,2);
featurepoints(4).width = width;
featurepoints(4).row = 6;
featurepoints(4).column = 9;

for i = 1:featurepoints(3).imageindices
    for j = 1:featurepoints(3).featureindices
        % image points
        featurepoints(1).image(i).imgcoordinates(:,j) = double([m1.imgpoints(i,j,1,1); m1.imgpoints(i,j,1,2)]);
        featurepoints(2).image(i).imgcoordinates(:,j) = double([m2.imgpoints(i,j,1,1); m2.imgpoints(i,j,1,2)]);
        % object points
        featurepoints(1).image(i).objcoordinates(:,j) = double(m1.objpoints(i,j,:));
        featurepoints(2).image(i).objcoordinates(:,j) = double(m2.objpoints(i,j,:));
        % extrinsic
        featurepoints(1).image(i).rotation = [m1.Rm(i,1,1) m1.Rm(i,1,2) m1.Rm(i,1,3);...
            m1.Rm(i,2,1) m1.Rm(i,2,2) m1.Rm(i,2,3);...
            m1.Rm(i,3,1) m1.Rm(i,3,2) m1.Rm(i,3,3)];
        featurepoints(2).image(i).rotation = [m2.Rm(i,1,1) m2.Rm(i,1,2) m2.Rm(i,1,3);...
            m2.Rm(i,2,1) m2.Rm(i,2,2) m2.Rm(i,2,3);...
            m2.Rm(i,3,1) m2.Rm(i,3,2) m2.Rm(i,3,3)];
        featurepoints(1).image(i).translation = m1.T(i,:)';
```

```
        featurepoints(2).image(i).translation = m2.T(i,:)';
    end
    [epil, Frl] = funda_epi(featurepoints(1));
    [epir, Frr] = funda_epi(featurepoints(2));
    featurepoints(1).image(i).fundamental = Frl(:,3*i-2:3*i);
    featurepoints(2).image(i).fundamental = Frr(:,3*i-2:3*i);
    featurepoints(1).image(i).epipole = epil(:,i);
    featurepoints(2).image(i).epipole = epir(:,i);
    featurepoints(1).image(i).homography = zeros(3,3);
    featurepoints(2).image(i).homography = zeros(3,3);
end

%% construct for space points in camera coordinate system
for i = 1:featurepoints(3).imageindices
    rl = featurepoints(1).image(i).rotation;
    tl = featurepoints(1).image(i).translation;
    rr = featurepoints(2).image(i).rotation;
    tr = featurepoints(2).image(i).translation;
    for j = 1:featurepoints(3).featureindices
        tmp = [rl tl; zeros(1,3) 1]*...
            [featurepoints(1).image(i).objcoordinates(:,j);1];
        tmpr = [rr tr; zeros(1,3) 1]*...
            [featurepoints(2).image(i).objcoordinates(:,j);1];
        featurepoints(4).image(i).ccs(:,j) = tmp(1:3);
        featurepoints(4).image(i).ccsr(:,j) = tmpr(1:3);
    end
    featurepoints(4).image(i).est        = zeros(3,featurepoints(3).featureindices);
    featurepoints(1).image(i).corrected  = zeros(2,featurepoints(3).featureindices);
    featurepoints(2).image(i).corrected  = zeros(2,featurepoints(3).featureindices);
    featurepoints(1).image(i).normalized = zeros(2,featurepoints(3).featureindices);
    featurepoints(2).image(i).normalized = zeros(2,featurepoints(3).featureindices);
    featurepoints(4).image(i).meanerror  = zeros(1,featurepoints(3).featureindices);
end
% structure parameters
[featurepoints(3).structure.rotation,featurepoints(3).structure.translation] =...
    StructureParameters2(featurepoints);
```

```
function p = Distortion_Coefficients(A, D)
% This function helps to obtf.intrinsicin the distortion coefficients for
% bf.intrinsicck-projection cf.intrinsicse. This method uses mf.intrinsictching distoted f.intrinsicnd estimf.intrinsicted
% fef.intrinsicture points to get the coefficients.
% Input:
% k1, k2: distortion coefficients from forwf.intrinsicrd projection model
% N: number of fef.intrinsicture points
```

```
% Du, Dv: scf.intrinsicle ff.intrinsicctors thf.intrinsict chf.intrinsicnge the metric units to pixels
% su: scf.intrinsicle ff.intrinsiccctor
% (u0,v0): center of imf.intrinsicge in pixels of N imf.intrinsicges
% Outputs:
% p = [f.intrinsic1,f.intrinsic2,f.intrinsic3,f.intrinsic4,f.intrinsic5,f.intrinsic6,f.intrinsic7,f.intrinsic8]': coefficients thf.intrinsict cf.intrinsicn recover correct
% imf.intrinsicge points from distorted ones
%
% generf.intrinsicte imf.intrinsicge points
[m,md] = points_generation(A, D); % in pixels
u = m(1,:); v = m(2,:);
ut = md(1,:); vt = md(2,:);
r = sqrt(ut.^2 + vt.^2);
N = length(m);
U = zeros(8,N);
V = zeros(8,N);
T = zeros(8,2*N);
e = zeros(2*N,1);
for i = 1:N
    U(:,i) = [-ut(i)*r(i)^2; -ut(i)*r(i)^4; -2*ut(i)*vt(i); -r(i)^2-2*ut(i)^2;...
        u(i)*r(i)^4; u(i)*ut(i)*r(i)^2; u(i)*vt(i)*r(i)^2; u(i)*r(i)^2];
    V(:,i) = [-vt(i)*r(i)^2; -vt(i)*r(i)^4; -r(i)^2-2*vt(i)^2; -2*ut(i)*vt(i);...
        v(i)*r(i)^4; v(i)*ut(i)*r(i)^2; v(i)*vt(i)*r(i)^2; v(i)*r(i)^2];
    T(:,2*i-1) = U(:,i);
    T(:,2*i) = V(:,i);
    e(2*i-1,1) = ut(i)-u(i);
    e(2*i,1) = vt(i)-v(i);
end
T = T';
p = (T'*T)\T'*e;
end
```

```
function mt = Correction_Imagepoints(md,p)
% Correct image points with coefficients p = [a1,a2,a3,a4,a5,a6,a7,a8]'
% Use function: p = Distortion_Coefficients(md,m,N,Du,Dv,su,u0,v0)
% Input:
%   md: distorted normalized image coordinates. 2-by-N array
%   p: backward distortion coefficients
%   A: intrinsic matrix of the camera
% Output:
%   m: normalized corrected image points
%
ud = md(1,:); vd = md(2,:);
r = sqrt(ud.^2+vd.^2);
N = length(md);
mt = zeros(2,N);
G = zeros(1,N);
% pixel cooedinate system
for i = 1:N
```

```
    G(i) = ( p(5) * r(i)^2 + p(6) * ud(i) + p(7) * vd(i) + p(8) ) * r(i)
        ^2 + 1;
    mt(1,i) = (ud(i)*(1 + p(1)*r(i)^2 + p(2)*r(i)^4) + 2*p(3)*ud(i)*vd(i)
        ...
        + p(4)*(r(i)^2 + 2*ud(i)^2))/G(i);
    mt(2,i) = (vd(i)*(1 + p(1)*r(i)^2+p(2)*r(i)^4)+2*p(4)*ud(i)*vd(i)...
        +p(3)*(r(i)^2+2*vd(i)^2))/G(i);
end
end
```

```
function Jt = CostTriangulation(x, R, T, mnl, mnr)
% This function is the cost function of triangulation
% x: the object point to be triangulated to, which is a 3-1 vector
% R,T: structure parameters
% mnl, mnr: normalized image points
Pose1 = [eye(3) zeros(3,1)];
Pose2 = [R T];
M1 = Pose1 * [x; 1];
M2 = Pose2 * [x; 1];
mnl_est = M1(1:2)/M1(3);
mnr_est = M2(1:2)/M2(3);
dl = norm( mnl(1:2) - mnl_est )^2;
dr = norm( mnr(1:2) - mnr_est )^2;
Jt = dl + dr;
end
```

```
function [R,T] = StructureParameters(f)
N = f(3).imageindices; L = f(3).featureindices; Rs=0;
% centroid-based optimization method
tmp1 = 0; tmp2 = 0;
for i = 1:N
    for j = 1:L
        tmp1 = tmp1 + f(4).image(i).ccs(:,j);
        tmp2 = tmp2 + f(4).image(i).ccsr(:,j);
    end
end
Mcl_median = tmp1/N/L;
Mcr_median = tmp2/N/L;
Mcl = zeros(N,L,3); Mcr = zeros(N,L,3);
for i = 1:N
    rl = f(1).image(i).rotation;
    rr = f(2).image(i).rotation;
    Rs = Rs + rr/rl;
%     ts = ts + tr - rr/rl*tl;
    for j = 1:L
        Mcl(i,j,:) = f(4).image(i).ccs(:,j) - Mcl_median;
        Mcr(i,j,:) = f(4).image(i).ccsr(:,j) - Mcr_median;
    end
end
Rs = Rs/N;
J = @(x)double(CostCenter(x,Mcl, Mcr));
x0 = Rs;
options.Algorithm = 'levenberg-marquardt';
```

```
options.MaxFunctionEvaluations = 5000;
R = lsqnonlin(J,x0,[],[],options);
T = Mcr_median - R * Mcl_median;
end
```

```
function [R,T] = StructureParameters2(f)
N = f(3).imageindices;
% Get Ri and ti
Rs=0;ts=0;
for i = 1:N
    rl = f(1).image(i).rotation;
    rr = f(2).image(i).rotation;
    tl = f(1).image(i).translation;
    tr = f(2).image(i).translation;
    Rs = Rs + rr/rl;
    ts = ts + tr - rr/rl*tl;
end
Rs = Rs/N;
ts = ts/N;

% % Take median of {Ri} and {ti}
R_init = Rs/N;
t_init = ts/N;
J = @(x)double(CostReprojection(x,f));
x0 = [R_init t_init];
options.Algorithm = 'levenberg-marquardt';
options.MaxFunctionEvaluations = 5000;
x = lsqnonlin(J,x0,[],[],options);
R = x(1:3,1:3);
T = x(1:3,4);
end
```

```
% use data from calib09-09-19.mat (from opencv) to reconstruct positions of
% light sources.

%% load data
mat = load('calib09-09-19.mat');
glintsl1 = double(mat.glintsL1);
glintsl2 = double(mat.glintsL2);
glintsr1 = double(mat.glintsR1);
glintsr2 = double(mat.glintsR2);
Al = mat.cameraMatrixL; Ar = mat.cameraMatrixR;
Dl = mat.distCoeffsL; Dr = mat.distCoeffsR;
R = mat.R; T = mat.T;
N = size(glintsl1, 1);
%% normalization & dist
nglintsl1 = zeros(2,14); nglintsl2 = zeros(2,14);
nglintsr1 = zeros(2,14); nglintsr2 = zeros(2,14);
for i = 1:N
    tmp = Al \ [glintsl1(i, :) 1]'; nglintsl1(:,i) = tmp(1:2);
    tmp = Al \ [glintsl2(i, :) 1]'; nglintsl2(:,i) = tmp(1:2);
    tmp = Ar \ [glintsr1(i, :) 1]'; nglintsr1(:,i) = tmp(1:2);
```

```
    tmp = Ar \ [glintsr2(i, :) 1]'; nglintsr2(:,i) = tmp(1:2);
end
P_left = zeros(8,N);
P_right = zeros(8,N);
for i = 1:N
    P_left(:,i) = Distortion_Coefficients(Al, Dl);
    P_right(:,i) = Distortion_Coefficients(Ar, Dr);
end
%% correction
cglintsl1 = Correction_Imagepoints(nglintsl1,P_left(:,i));
cglintsl2 = Correction_Imagepoints(nglintsl2,P_left(:,i));
cglintsr1 = Correction_Imagepoints(nglintsr1,P_right(:,i));
cglintsr2 = Correction_Imagepoints(nglintsr2,P_right(:,i));
%% triangulation image of light sources in ccs
L1 = zeros(3, N); L2 = zeros(3, N);
for i = 1:N
    costtotal1 = @(x)CostTriangulation(x, R, T, cglintsl1(:,i), cglintsr1
        (:,i));
    costtotal2 = @(x)CostTriangulation(x, R, T, cglintsl2(:,i), cglintsr2
        (:,i));
    x_tri1 = fminunc(costtotal1, [0;0;600]);
    x_tri2 = fminunc(costtotal2, [0;0;600]);
    L1(:,i) = x_tri1;
    L2(:,i) = x_tri2;
end
%%
m1 = load('calibLeft_09-09-19.mat'); mat2 = load('calibRight_09-09-19.mat
    ');
center = [55; 14; 1.8];
center_ccs = zeros(3,N);
for i =1:N
    rotate = [m1.Rm(i,1,1) m1.Rm(i,1,2) m1.Rm(i,1,3);...
              m1.Rm(i,2,1) m1.Rm(i,2,2) m1.Rm(i,2,3);...
              m1.Rm(i,3,1) m1.Rm(i,3,2) m1.Rm(i,3,3)];
    trans = m1.T(i,:)';
    center_ccs(:,i) = rotate * center + trans;
end

%% reconstruct light sources
light1 = zeros(3, N-1); light2 = zeros(3, N-1);
for i = 1:N-1
    for j = i+1:N
        light1(:,i) = lineintersection3d(center_ccs(:,i), L1(:,i),
            center_ccs(:,j), L1(:,j));
        light2(:,i) = lineintersection3d(center_ccs(:,i), L2(:,i),
            center_ccs(:,j), L2(:,j));
    end
end

%% error
% x = 1:N-2;
% error1 = zeros(1,N-2);
% for i = 1:N-2
```

```
%      error1(i) = norm(light1(:,i+1) - light1(:,1));
% end
% figure();
% plot(x, error1);

%% plot x, y, z distribution
x = 1:N-1;
x1 = light1(1,:); x2 = light2(1,:);
y1 = light1(2,:); y2 = light2(2,:);
z1 = light1(3,:); z2 = light2(3,:);
figure();
scatter3(x1, y1, z1);
hold on;
scatter3(x2, y2, z2);
xlabel('x'); ylabel('y'); zlabel('z');
legend('light source 1', 'light source 2');
% figure();
% scatter(x, x1);
```

Appendix D

OpenCV Python Code

```
import os
import matplotlib.pyplot as plt
import calibrate_new
import self
import cv2

#%% build
# define path of image sets
path_L1, path_R1 = '../15-08-19/L/*.bmp', '../15-08-19/R/*.bmp'
path_L, path_R = '../image/L/*.jpg', '../image/R/*.jpg'
path_L2, path_R2 = '../10-07-19/L/*.bmp', '../10-07-19/R/*.bmp'
path_L3, path_R3 = '../16-07-19/L/*.bmp', '../16-07-19/R/*.bmp'
path_L4, path_R4 = '../02-09-19-2/L/*.bmp', '../02-09-19-2/R/*.bmp'
path_L5, path_R5 = '../09-09-19/L_withlight/*.bmp', '../09-09-19/R_withlight/*.bmp'

filepath_L = os.path.join(os.getcwd(), path_L)
filepath_R = os.path.join(os.getcwd(), path_R)

while True:
    print '1: 15-08-19 \n2: 10-07-19\n3: 16-07-19\n4: 02-09-19-2\n5: 09-09-19\n'
    flag = raw_input("Choose an image set (0 - 5):\n")
    if flag == "0": # remember to change board witdth as 5 in self.py
        break
    elif flag == "1": # 1,2,3 board width = 4mm in self.py
        filepath_L = os.path.join(os.getcwd(), path_L1)
        filepath_R = os.path.join(os.getcwd(), path_R1)
        break
    elif flag == "2":
        filepath_L = os.path.join(os.getcwd(), path_L2)
        filepath_R = os.path.join(os.getcwd(), path_R2)
        break
```

```
    elif flag == "3":
        filepath_L = os.path.join(os.getcwd(), path_L3)
        filepath_R = os.path.join(os.getcwd(), path_R3)
        break
    elif flag == "4":
        filepath_L = os.path.join(os.getcwd(), path_L4)
        filepath_R = os.path.join(os.getcwd(), path_R4)
        break
    elif flag == "5":
        filepath_L = os.path.join(os.getcwd(), path_L5)
        filepath_R = os.path.join(os.getcwd(), path_R5)
        break
    else:
        print 'Please input 1,2, 3, 4 or 5.'
        continue

#%% single camera calibration and stereo calibration
[index, imgpointsL, objpointsL, imgpointsR,  objpointsR] = calibrate_new.points(filepath_L, filepath_R)
cameraMatrixL, distCoeffsL, rvecsL, tvecsL, cameraMatrixR, distCoeffsR, rvecsR, tvecsR = \
                         calibrate_new.singlecalibrate(imgpointsL, objpointsL, imgpointsR,  objpointsR, flag)

# rl, rr = [], []
# for i in xrange(len(rvecsL)):
#     tmp1, _ = cv2.Rodrigues(rvecsL[i])
#     tmp2, _ = cv2.Rodrigues(rvecsR[i])
#     rl.append(tmp1)
#     rr.append(tmp2)
# calibrate_new.save(1, imgpointsL, objpointsL, cameraMatrixL, distCoeffsL, rl, tvecsL, imgpointsR, objpointsR, cameraMatrixR, distCoeffsR, rr, tvecsR)


cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, R, T, E, F = \
    calibrate_new.stereocalibrate(imgpointsL, objpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL, imgpointsR,
                                  objpointsR, cameraMatrixR, distCoeffsR, rvecsR, tvecsR)

#%% reconstruction and error
er, recons_error, objccs, objest = calibrate_new.reconstruct(objpointsL, imgpointsL, cameraMatrixL, distCoeffsL, rvecsL,
                                                             tvecsL, objpointsR, imgpointsR, cameraMatrixR, distCoeffsR,
```

```
                                                                                rvecsR,
                                                                                tvecsR, R,
                                                                                F)

#%% reconstruction error in world coordinate system
objwcs = self.CCS2WCS(objest, rvecsL, tvecsL)
er, recons_error = self.reconstructionerror_totalmean(objwcs, objpointsL)
# er, recons_error, objccs, objest =calibrate.reconstruct(objpointsL,
    imgpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL, imgpointsR,
    cameraMatrixR, distCoeffsR, rvecsR, tvecsR, R, T, F)
print 'Reconstruction Error: ', recons_error, 'mm'
print er
# calibrate_new.plot3d(objccs, objest)
# calibrate_new.ploterror(er)
# plt.show()
```

```
import numpy as np
import glob
import scipy.io as sio
import cv2
import self
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from mpl_toolkits.mplot3d import Axes3D


N, L = 0, 0
##############################################################
################single camera calibration##################
def points(filepath_L, filepath_R):
    images1, images2 = glob.glob(filepath_L), glob.glob(filepath_R)
    # obatin object points and image points ( inside image sets are paths
        of images)
    objpointsL, imgpointsL, im1 = self.cornerdetection(images1)
    objpointsR, imgpointsR, im2 = self.cornerdetection(images2)

    # find common elements of two image sets
    index = []
    for x in im1:
        for y in im2:
            if x == y:
                index.append(x)

    imgL = [images1[ind] for ind in index]
    imgR = [images2[ind] for ind in index]

    objpointsL, imgpointsL, _ = self.cornerdetection(imgL)
    objpointsR, imgpointsR, _ = self.cornerdetection(imgR)

    global N, L
    N, L = len(objpointsL), len(objpointsL[0])

    return index, imgpointsL, objpointsL, imgpointsR,  objpointsR
```

```
# save data into mat file
def save(flag,imgpointsL, objpointsL, cameraMatrixL, distCoeffsL, rl, tvecsL, imgpointsR, objpointsR, cameraMatrixR, distCoeffsR, rr, tvecsR):
    if flag == 0:
    # image
        sio.savemat('calibLeft', {'cameraMatrix': cameraMatrixL,\
          'distCoeffs': distCoeffsL, 'T': tvecsL,\
          'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight', {'cameraMatrix': cameraMatrixR,\
           'distCoeffs': distCoeffsR, 'T': tvecsR, 'Rm': rr, \
           'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved.\n'

    # image set 15-08-19
    if flag == 1:
        sio.savemat('calibLeft_15-08-19', {'cameraMatrix': cameraMatrixL,\
          'distCoeffs': distCoeffsL, 'T': tvecsL,\
          'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight_15-08-19', {'cameraMatrix': cameraMatrixR,\
           'distCoeffs': distCoeffsR, 'T': tvecsR, 'Rm': rr, \
           'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved to 15-08-19.\n'


    # image set 10-07-19
    elif flag == 2:
        sio.savemat('calibLeft_10-07-19', {'cameraMatrix': cameraMatrixL,\
          'distCoeffs': distCoeffsL, 'T': tvecsL,\
          'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight_10-07-19', {'cameraMatrix': cameraMatrixR,\
           'distCoeffs': distCoeffsR, 'T': tvecsR, 'Rm': rr, \
           'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved to 10-07-19.\n'

```

```
    # image set 16-07-19
    elif flag == 3:
        sio.savemat('calibLeft_16-07-19', {'cameraMatrix': cameraMatrixL,\
         'distCoeffs': distCoeffsL, 'T': tvecsL,\
         'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight_16-07-19', {'cameraMatrix': cameraMatrixR,\
          'distCoeffs': distCoeffsR, 'T': tvecsR, 'Rm': rr, \
          'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved to 16-07-19.\n'

    elif flag == 4:
        sio.savemat('calibLeft_02-09-19-2', {'cameraMatrix': cameraMatrixL,\
         'distCoeffs': distCoeffsL, 'T': tvecsL,\
         'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight_16-07-19', {'cameraMatrix': cameraMatrixR,\
          'distCoeffs': distCoeffsR, 'T': tvecsR, 'Rm': rr, \
          'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved to 02-09-19-2.\n'

    elif flag == 5:
        sio.savemat('calibLeft_09-09-19', {'cameraMatrix': cameraMatrixL, 'distCoeffs': distCoeffsL, 'T': tvecsL,
                                                        'Rm': rl, 'imgpoints': imgpointsL, 'objpoints': objpointsL}, appendmat=True)

        sio.savemat('calibRight_09-09-19', {'cameraMatrix': cameraMatrixR, 'distCoeffs': distCoeffsR, 'T': tvecsR,
                                                         'Rm': rr, 'imgpoints': imgpointsR, 'objpoints': objpointsR}, appendmat=True)

        print 'Saved to 09-09-19.\n'


def singlecalibrate(imgpointsL, objpointsL, imgpointsR,  objpointsR, flag):

    # get caalibration results of two image sets
```

```
    cameraMatrixL, distCoeffsL, rvecsL, tvecsL, errorL1, errorL2 = self.calibration(objpointsL, imgpointsL)
    cameraMatrixR, distCoeffsR, rvecsR, tvecsR, errorR1, errorR2 = self.calibration(objpointsR, imgpointsR)

    # reprojection error
    # rel_perview, re_l = self.re(objpointsL, imgpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL)
    # rer_perview, re_r = self.re(objpointsR, imgpointsR, cameraMatrixR, distCoeffsR, rvecsR, tvecsR)

    # transform rotation vectors to rotation matrix
    rl, rr = [], []
    for i in xrange(len(rvecsL)):
        tmp1, _ = cv2.Rodrigues(rvecsL[i])
        tmp2, _ = cv2.Rodrigues(rvecsR[i])
        rl.append(tmp1)
        rr.append(tmp2)

    save(flag, imgpointsL, objpointsL, cameraMatrixL, distCoeffsL, rl, tvecsL, imgpointsR, objpointsR, cameraMatrixR, distCoeffsR, rr, tvecsR)

    return cameraMatrixL, distCoeffsL, rvecsL, tvecsL, cameraMatrixR, distCoeffsR, rvecsR, tvecsR


def stereocalibrate(imgpointsL,objpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL, imgpointsR, objpointsR, cameraMatrixR, distCoeffsR, rvecsR, tvecsR):
    # N, L = len(imgpointsL), len(imgpointsL[0])
    _,cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, R, T, E, F, _ = \
        cv2.stereoCalibrateExtended(objpointsL, imgpointsL, imgpointsR, cameraMatrixL, distCoeffsL,\
        cameraMatrixR, distCoeffsR, self.imgSize, R=None, T = None, flags=cv2.CALIB_USE_INTRINSIC_GUESS)

    return cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, R, T, E, F

# S. Gai
# function for optimized rotation R
def fri(R, Mcl, Mcr):
    fri = 0.0
    for i in xrange(N):
        for j in xrange(L):
            tmp1 = R[0]*Mcl[i][j][0] + R[1]*Mcl[i][j][1] + R[2]*Mcl[i][j][2]
            tmp2 = R[3]*Mcl[i][j][0] + R[4]*Mcl[i][j][1] + R[5]*Mcl[i][j][2]
            tmp3 = R[6]*Mcl[i][j][0] + R[7]*Mcl[i][j][1] + R[8]*Mcl[i][j][2]
```

```
                tmp = np.linalg.norm([tmp1-Mcr[i][j][0], tmp2-Mcr[i][j][1], tmp3-Mcr[i][j][2]]) * (2 ** (0.5))
                fri += tmp **2
    return fri


def structureParams(objpointsL, rvecsL, tvecsL, objpointsR, rvecsR, tvecsR, R):
    # get median of all objpoints in camera coordinate system
    Mcl, Mcr = self.WCS2CCS(objpointsL, rvecsL, tvecsL), self.WCS2CCS(objpointsR, rvecsR, tvecsR)
    tmpl ,tmpr = np.zeros(3), np.zeros(3)
    for i in xrange(N):
        for j in xrange(L):
            tmpl += Mcl[i][j]
            tmpr += Mcr[i][j]
    Mcl_bar = tmpl / N / L
    Mcr_bar = tmpr / N / L
    for i in xrange(N):
        for j in xrange(L):
            Mcl[i][j] = Mcl[i][j] - Mcl_bar
            Mcr[i][j] = Mcr[i][j] - Mcr_bar
    R0 = np.reshape(R, (1,9))
    res = minimize(fri, R0, args=(Mcl, Mcr))
    x = res.x
    structure_rotation = np.array([[x[0], x[1], x[2]], [x[3], x[4], x[5]], [x[6], x[7], x[8]]])
    structure_translate = Mcr_bar - np.dot(structure_rotation, Mcl_bar)
    return structure_rotation, structure_translate


# nonlinear triangulation for every frame
def nontri(objest, nimgl, nimgr, array1, array2):
    tmp01 = np.dot(array1, np.append(objest, [1]))
    tmp02 = np.dot(array2, np.append(objest, [1]))
    tmp1 = nimgl - tmp01[:2]/tmp01[2]
    tmp2 = nimgr - tmp02[:2]/tmp02[2]
    cost = np.linalg.norm(tmp1) ** 2 + np.linalg.norm(tmp2) ** 2
    return cost

# using different distortion coefficients, structure parameters and triangulation
def reconstruct(objpointsL, imgpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL,objpointsR, imgpointsR, cameraMatrixR, distCoeffsR, rvecsR, tvecsR, Ri, F):
    R, T = structureParams(objpointsL, rvecsL, tvecsL, objpointsR, rvecsR, tvecsR, Ri)
    # undistort
    # R1, R2 = cv2.stereoRectify(cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, self.imgSize, R, T, flags=cv2.CALIB_ZERO_DISPARITY)[:2]
    # imgUndistortL, imgUndistortR = [], []
    # for i in xrange(N):
```

```
    #     imgUndistortL.append(cv2.undistortPoints(imgpointsL[i], cameraMatrixL, distCoeffsL, R1, cameraMatrixL))
    #     imgUndistortR.append(cv2.undistortPoints(imgpointsR[i], cameraMatrixR, distCoeffsR, R2, cameraMatrixR))

    objccs = self.WCS2CCS(objpointsL, rvecsL, tvecsL)
    # triangulate
    # opencv standard method
    array1 = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]])
    array2 = np.concatenate((R, T[:, None]), axis=1)
    # P1 = np.dot(cameraMatrixL, array1)
    # P2 = np.dot(cameraMatrixR, array2)
    #
    objest = []
    # for i in xrange(N):
    #     tmp = cv2.triangulatePoints(P1, P2, imgUndistortL[i], imgUndistortR[i])
    #     # tmp = cv2.triangulatePoints(P1, P2, nimgl[i], nimgr[i])
    #     objest.append((tmp[:3]/tmp[3]).T.tolist())

    # nonlinear triangulation
    # nimgl, nimgr = self.normalization(imgpointsL, cameraMatrixL), self.normalization(imgpointsR, cameraMatrixR)
    pl, pr = self.distcoeffs(cameraMatrixL, distCoeffsL), self.distcoeffs(cameraMatrixR, distCoeffsR)
    nimgl, nimgr = self.undistort(cameraMatrixL, pl, imgpointsL), self.undistort(cameraMatrixR, pr, imgpointsR)
    for i in xrange(N):
        tmp = []
        for j in xrange(L):
            res = minimize(nontri, objccs[i][j], args=(nimgl[i][j], nimgr[i][j], array1, array2))
            tmp.append(res.x.reshape(3))
        objest.append(tmp)


    # 3d error
    er, recons_error = self.reconstructionerror_totalmean(objccs, objest)

    return er, recons_error, objccs, objest


# ##################################plot###############################
def plot3d(objccs, objest):
    x1, y1, z1, x2, y2, z2 = [], [], [], [], [], []
    for j in xrange(L):
        x1.append(objccs[0][j][0])
        y1.append(objccs[0][j][1])
        z1.append(objccs[0][j][2])
        x2.append(objest[0][j][0])
        y2.append(objest[0][j][1])
        z2.append(objest[0][j][2])

```

```
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(x1, y1, z1, c='r', marker='o', label='object points')
    ax.scatter(x2, y2, z2, c='b', marker='^', label='estimated objpoints'
        )
    plt.gca().legend(('object points','estimated objpoints'))
    # plt.title('Estimated Object Points and WCS2CCS')
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')


def ploterror(er):
    plt.figure()
    plt.plot(list(range(N)) + np.ones(N), er)
    plt.title('Reconstruction Error')
    plt.xlabel('image index')
    plt.ylabel('3d error (mm)')


def plotxyz(objccs, objest, a):
    x, y, z = [], [], []
    errx, erry, errz = [], [], []
    for i in xrange(N):
        for j in xrange(L):
            errx.append(objccs[i][j][0] - objest[i][j][0])
            erry.append(objccs[i][j][1] - objest[i][j][1])
            errz.append(objccs[i][j][2] - objest[i][j][2])
            x.append(objccs[i][j][0])
            y.append(objccs[i][j][1])
            z.append(objccs[i][j][2])

    plt.figure()
    if a == 'x':
        plt.scatter(x, errx)
    elif a == 'y':
        plt.scatter(y, erry)
    elif a == 'z':
        plt.scatter(z, errz)

    plt.show()


# plot light source
def plotlight(lightsource_pos):
    fig = plt.figure()

    x, y, z = [], [], []
    for i in xrange(len(lightsource_pos)):
        x.append(lightsource_pos[i][0])
        y.append(lightsource_pos[i][1])
        z.append(lightsource_pos[i][2])
```

```

    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(x, y, z, c='r', marker='x', label='light source positions'
        )
    plt.gca().legend(('object points', 'estimated objpoints'))
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    plt.savefig('light.png')
    plt.show()
```

```
import numpy as np
import math
import cv2
import statistics

# define constants
imgSize = (1280, 1024)
# number of rows and columns of the chessboard
n_row = 6
n_col = 9
# width of squares on the chessboard (mm)
board_w = 4
# number of points per image/board
n_points = n_row * n_col
# pixel size
pixel_size = 4.8e-3

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-6)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((n_points, 3), np.float32)
objp[:, :2] = np.mgrid[0:n_row, 0:n_col].T.reshape(-1,2)
objp *= board_w


def cornerdetection(images):
    # Arrays to store object points and image points from all the images.
    objpoints = []  # 3d point in real world space
    imgpoints = []  # 2d points in image plane.
    index = []
    count = 0
    for fname in images:
        img = cv2.flip(cv2.imread(fname, 0), 1)
        ret, corners = cv2.findChessboardCorners(img, (n_row, n_col),
            None)
        if ret == True:
            objpoints.append(objp)
            corners2 = cv2.cornerSubPix(img, corners, (11, 11), (-1, -1),
                criteria)
            imgpoints.append(corners2)
```

```
            index.append(count)
            # Draw and display the corners
            # cv2.drawChessboardCorners(img, (n_row, n_col), corners2,
                ret)
            # cv2.imshow('img',img)
            # cv2.waitKey(500)
        count += 1

    # cv2.destroyAllWindows()

    return objpoints, imgpoints, index

# standard opencv method
def undistortimage(images, cameraMatrix, distcoeffs):
    imgpoints = []
    for fname in images:
        img = cv2.flip(cv2.imread(fname, 0),1)
        dst = cv2.undistort(img, cameraMatrix, distcoeffs)
        ret, corners = cv2.findChessboardCorners(dst, (n_row, n_col),
            None)
        if ret == True:
            corners2 = cv2.cornerSubPix(dst, corners, (11, 11), (-1, -1),
                 criteria)
            imgpoints.append(corners2)
    return imgpoints



def calibration(objpoints, imgpoints):
    # first optimization with fixed principle point
    # then use intrinsic guess to do second calibration

    flag1 = cv2.CALIB_FIX_PRINCIPAL_POINT
    flag2 = cv2.CALIB_USE_INTRINSIC_GUESS
    flag3 = cv2.CALIB_ZERO_TANGENT_DIST

    # _, cameraMatrix, distCoeffs, rvecs, tvecs = \
    #     cv2.calibrateCamera(objpoints, imgpoints, imgSize, None, None,
        flags=flag1)

    # retval, cameraMatrix, distCoeffs, rvecs, tvecs = \
    #     cv2.calibrateCamera(objpoints, imgpoints, imgSize, cameraMatrix
        , distCoeffs, flags=flag2)

    _, cameraMatrix, distCoeffs, rvecs, tvecs, _, _, error1 = \
        cv2.calibrateCameraExtended(objpoints, imgpoints, imgSize, None,
            None, flags=flag3)

    _, cameraMatrix, distCoeffs, rvecs, tvecs, _, _, error2 = \
        cv2.calibrateCameraExtended(objpoints, imgpoints, imgSize,
            cameraMatrix, distCoeffs, flags=flag2)

    return cameraMatrix, distCoeffs, rvecs, tvecs, error1, error2
```

```python


def square(my_list):
    return [i ** 2 for i in my_list]


def squareroot(my_list):
    return [math.sqrt(i) for i in my_list]


def reprojection(objpoints, imgpoints, cameraMatrix, distCoeffs, rvecs, tvecs):
    imgpointsre = []
    tmp = []
    for i in xrange(len(objpoints)):
        imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMatrix, distCoeffs)
        tmp.append(imgpoints2)
    for j in xrange(len(imgpoints)):
        imgpointsre.append(tmp[j])
    return imgpointsre

# projection error
def reprojectionerror(objpoints, imgpoints, cameraMatrix, distCoeffs, rvecs, tvecs):
    re_x = []
    re_y = []
    for i in xrange(len(objpoints)):
        imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMatrix, distCoeffs)
        for j in xrange(len(imgpoints2)):
            err_x = squareroot(square(imgpoints[i][j][:,0]-imgpoints2[j][:,0]))
            err_y = squareroot(square(imgpoints[i][j][:,1]-imgpoints2[j][:,1]))
            re_x.append(err_x)
            re_y.append(err_y)

    mean_error_x = 0
    mean_error_y = 0
    for i in re_x:
        mean_error_x += sum(i)

    for i in re_y:
        mean_error_y += sum(i)

    mean_error_x = mean_error_x/len(re_x)
    mean_error_y = mean_error_y/len(re_y)
    total = math.sqrt((math.pow(mean_error_x,2) + math.pow(mean_error_y,2)))

    return re_x, re_y, mean_error_x, mean_error_y, total

```

```python
# detect corner around reprojected points first, then calculate the reprojection error
def reprojectionerror2(img, imgpoints1):
    imgpoints2 = []
    n_frame = 0
    for fname in img:
        im = cv2.imread(fname, 0)
        corners = cv2.cornerSubPix(im, imgpoints1[n_frame], (11, 11), (-1, -1), criteria)
        imgpoints2.append(corners)
        n_frame += 1
    re_x = []
    re_y = []
    for i in xrange(len(imgpoints2)):
        for j in xrange(len(imgpoints2[0])):
            err_x = squareroot(square(imgpoints1[i][j][:, 0] - imgpoints2[i][j][:, 0]))
            err_y = squareroot(square(imgpoints1[i][j][:, 1] - imgpoints2[i][j][:, 1]))
            re_x.append(err_x)
            re_y.append(err_y)
    mean_error_x = 0
    mean_error_y = 0
    for i in re_x:
        mean_error_x += sum(i)

    for i in re_y:
        mean_error_y += sum(i)

    mean_error_x = mean_error_x / len(re_x)
    mean_error_y = mean_error_y / len(re_y)
    total = math.sqrt((math.pow(mean_error_x, 2) + math.pow(mean_error_y, 2)))

    return re_x, re_y, mean_error_x, mean_error_y, total


# reprojection error for both left and right camera
def re(objpoints, imgpoints, cameraMatrix, distCoeffs, rvecs, tvecs):
    re_x, re_y, meanX, meanY, total =\
        reprojectionerror(objpoints, imgpoints, cameraMatrix, distCoeffs, rvecs, tvecs)
    re = []
    for i in xrange(len(re_x)):
        tmp = math.sqrt(np.square(re_x[i]) + np.square(re_y[i]))
        re.append(tmp)
    re_perview = []
    L = len(objpoints[0])
    for j in xrange(len(objpoints)):
        tmp0 = re[L*j:L*(j+1)]
        tmp = sum(tmp0)/L
        re_perview.append(tmp)
    return re_perview, np.sum(re)/len(re)
```

```python

# multiply sub-lists of two lists (list2 with single column)
def mul_lists(list1, list2):
    ans = []
    # if #columns of list1 equals #rows of list2
    if len(list1[0])==len(list2):
        # element-wise multiplication of rows of list1 and columns of list2
        # store sum of multiplied list into ans
        for i in xrange(len(list1)):
            tmp = [list1[i][j]*list2[j] for j in range(len(list2))]
            ans.append(sum(tmp))
    return ans


# generate normally distributed image points without distortion
def pointsgeneration(cameraMatrix):
    imgtest = np.zeros((2000, 2), np.float32)
    imgtest = np.mgrid[-3:47, -1:39].T.reshape(-1,2)
    fu, fv = cameraMatrix[0][0], cameraMatrix[1][1]
    u0, v0 = cameraMatrix[0][2], cameraMatrix[1][2]
    x, y = [], []
    for i in xrange(len(imgtest)):
        imgtest[i][0] = imgtest[i][0]*28.5
        imgtest[i][1] = imgtest[i][1]*27.5
        x.append((imgtest[i][0]-u0)/fu)
        y.append((imgtest[i][1]-v0)/fv)
    return imgtest, x, y


# distortion coefficients of forward model
def distortimgpoints(distCoeffs, x, y):
    xd, yd = [], []
    k1, k2 = distCoeffs[0][0], distCoeffs[0][1]
    imgtest_distorted = np.zeros((2000, 2), np.float32)
    for i in xrange(len(x)):
        r = np.sqrt(x[i]**2 + y[i]**2)
        tmp1 = (1 + k1*r**2 + k2*r**4) * x[i]
        tmp2 = (1 + k1*r**2 + k2*r**4) * y[i]
        imgtest_distorted[i][0] = tmp1
        imgtest_distorted[i][1] = tmp2
        xd.append(tmp1)
        yd.append(tmp2)
    return imgtest_distorted, xd, yd


# get distortion coefficients
def distcoeffs(cameraMatrix, distCoeffs):
    imgtest, x, y = pointsgeneration(cameraMatrix)
    imgtest_distorted, xd, yd = distortimgpoints(distCoeffs, x, y)
    T, e = [], []
    for i in xrange(len(xd)):
        rd = np.sqrt(xd[i]**2+yd[i]**2)
```

```
        tmpu = [-xd[i]*rd**2, -xd[i]*rd**4, -2*xd[i]*yd[i], -rd**2-2*xd[i]**2, x[i]*rd**4, x[i]*xd[i]*rd**2, x[i]*yd[i]*rd**2, x[i]*rd**2]
        tmpv = [-yd[i]*rd**2, -yd[i]*rd**4, -rd**2-2*yd[i]**2, -2*xd[i]*yd[i],  y[i]*rd**4, y[i]*xd[i]*rd**2, y[i]*yd[i]*rd**2, y[i]*rd**2]
        T.append(tmpu)
        T.append(tmpv)
        e.append(xd[i]-x[i])
        e.append(yd[i]-y[i])
    p = np.dot(np.linalg.pinv(np.array(T)), np.array(e).T)
    return p


# undistort image points with coefficients from previous function (normalized image points)
# input: distorted/observed image points, and distortion coefficients p
# output: undistorted normalized image points
def undistort(cameraMatrix, p, imgpoints):
    imgundistort = []
    fu, fv = cameraMatrix[0][0], cameraMatrix[1][1]
    u0, v0 = cameraMatrix[0][2], cameraMatrix[1][2]
    for i in xrange(len(imgpoints)):
        tmp = []
        for j in xrange(len(imgpoints[0])):
            xi = (imgpoints[i][j][0][0] - u0) / fu
            yi = (imgpoints[i][j][0][1] - v0) / fv
            r = np.sqrt(xi**2+yi**2)
            G = 1 + (p[4]*r*r + p[5]*xi + p[6]*yi + p[7]) * r * r
            tmp1 = (xi*(1 + p[0]*r*r + p[1]*r**4) + 2*p[2]*xi*yi + p[3]*(r*r+2*xi*xi)) / G
            tmp2 = (yi*(1+p[0]*r*r+p[1]*r**4) + p[2]*(r*r+2*yi*yi) + 2*p[3]*xi*yi) / G
            tmp3 = [tmp1, tmp2]
            tmp3 = np.reshape(tmp3,(1,2))
            tmp.append(tmp3)
        imgundistort.append(tmp)
    return imgundistort


# transformation from WCS to CCS
def WCS2CCS(objpoints, rvecs, tvecs):
    objpoints_ccs = []
    for i in xrange(len(objpoints)):
        rotate, _ = cv2.Rodrigues(rvecs[i])
        tmp = []
        for j in xrange(len(objpoints[0])):
            tmp1 = (np.dot(rotate, objpoints[i][j]) + tvecs[i].T).T.tolist()
            # tmp1 = (mul_lists(rotate.T, objpoints[i][j]) + tvecs[i].T).T.tolist()
            tmp2 = np.reshape(tmp1, 3)
            tmp.append(tmp2)
```

```
            objpoints_ccs.append(tmp)
    return objpoints_ccs


# normalization
def normalization(imgpoints, cameraMatrix):
    Ainv = np.linalg.inv(cameraMatrix)
    imgpoints_normalized = []
    # imghomo = []
    for i in xrange(len(imgpoints)):
        tmp, tmp2 = [], []
        for j in xrange(len(imgpoints[0])):
            ip = np.append(imgpoints[i][j], [1])
            ipn = np.dot(Ainv, np.array(ip))
            t = np.reshape(ipn[:2], (1,2))
            tmp.append(t)
            tmp2.append(np.reshape(ipn, (1,3)))
        imgpoints_normalized.append(tmp)
        # imghomo.append(tmp2)
    return np.array(imgpoints_normalized)#, np.array(imghomo)


# mean 3d reconstruction error of all images
def reconstructionerror_totalmean(objpoints1, objpoints2):
    error_perimage = []
    for i in xrange(len(objpoints1)):
        total = 0
        for j in xrange(len(objpoints1[0])):
            tmp = (objpoints1[i][j][0]-objpoints2[i][j][0])**2 + \
                  (objpoints1[i][j][1]-objpoints2[i][j][1])**2 + \
                  (objpoints1[i][j][2]-objpoints2[i][j][2])**2
            total += np.sqrt(tmp)/len(objpoints1[0])
        error_perimage.append(total)
    return error_perimage, np.sum(error_perimage)/len(objpoints1)


# transformation from WCS to CCS
def CCS2WCS(objpoints_ccs, rvecs, tvecs):
    objpoints_wcs = []
    for i in xrange(len(objpoints_ccs)):
        rotate, _ = cv2.Rodrigues(rvecs[i])
        tmp = []
        for j in xrange(len(objpoints_ccs[0])):
            tmp0 = objpoints_ccs[i][j] - tvecs[i].T
            tmp1 = np.dot(np.linalg.inv(rotate), tmp0.T)
            tmp2 = np.reshape(tmp1, 3)
            tmp.append(tmp2)
        objpoints_wcs.append(tmp)
    return objpoints_wcs
```

```
# calibrate cameras and click to choose the glints
# after this, turn to matlab testlight.m
#%% import modules
```

```
import os
import glob
import numpy as np
import cv2 as cv
import scipy.io as sio
import calibrate_new
import self
import light

#%% create file path
path_left_withlight  = '../09-09-19/L_withlight/*.bmp'
path_left_without    = '../09-09-19/L_without/*.bmp'
path_right_withlight = '../09-09-19/R_withlight/*.bmp'
path_right_without   = '../09-09-19/R_without/*.bmp'

filepath_left_withlight  = os.path.join(os.getcwd(), path_left_withlight)
filepath_left_without    = os.path.join(os.getcwd(), path_left_without)
filepath_right_withlight = os.path.join(os.getcwd(), path_right_withlight)
filepath_right_without   = os.path.join(os.getcwd(), path_right_without)

#%% do calibration
[index, imgpointsL, objpointsL, imgpointsR,  objpointsR] = \
    calibrate_new.points(filepath_left_withlight, filepath_right_withlight)

cameraMatrixL, distCoeffsL, rvecsL, tvecsL, cameraMatrixR, distCoeffsR, rvecsR, tvecsR = \
                    calibrate_new.singlecalibrate(imgpointsL, objpointsL, imgpointsR,  objpointsR, 5)


cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, R, T, E, F = \
    calibrate_new.stereocalibrate(imgpointsL, objpointsL, cameraMatrixL, distCoeffsL, rvecsL, tvecsL,
                                  imgpointsR, objpointsR, cameraMatrixR, distCoeffsR, rvecsR, tvecsR)

er, recons_error, objccs, objest = calibrate_new.reconstruct(objpointsL, imgpointsL, cameraMatrixL, distCoeffsL, rvecsL,
                                                             tvecsL, objpointsR, imgpointsR, cameraMatrixR, distCoeffsR,
                                                             rvecsR, tvecsR, R, F)
print recons_error
```

```

N = len(index)
#%% transform sphere's center to image plane
# center = [55, 14, 1.8]  # position of the sphere's center on board
# imgpoints_center_left, imgpoints_center_right = [], []
# N = len(objpointsL)
# for i in xrange(N):
#     rotate_left, _ = cv.Rodrigues(rvecsL[i])
#     rotate_right, _ = cv.Rodrigues(rvecsR[i])
#     tmp1 = np.dot(rotate_left, center) + tvecsL[i].T
#     tmp2 = np.dot(rotate_right, center) + tvecsR[i].T
#     tmp_left = np.dot(cameraMatrixL, tmp1.T.tolist())
#     tmp_right = np.dot(cameraMatrixR, tmp2.T.tolist())
#     imgpoints_center_left.append(tmp_left[:2]/tmp_left[2])
#     imgpoints_center_right.append(tmp_right[:2]/tmp_right[2])

#%% transform sphere's center from world coordinate system to camera
    coordinate system
center = np.array([55, 14, 1.8], dtype=np.float32)
center_wcs = []
for i in xrange(N):
    center_wcs.append([center])

center_ccs = self.WCS2CCS(center_wcs, rvecsL, tvecsL)

#%% manually detect glints and save image points
images_glints_left, images_glints_right = glob.glob(filepath_left_without
    ), glob.glob(filepath_right_without)
lglint, rglint = light.Glint(), light.Glint()

light.glintscoordinates(images_glints_left, lglint)
light.glintscoordinates(images_glints_right, rglint)

#%% separate two glints
lg1, lg2, rg1, rg2 = [], [], [], []
for i in xrange(len(lglint.points)/2):
    lg1.append(lglint.points[2*i])
    lg2.append(lglint.points[2*i+1])
for i in xrange(len(rglint.points)/2):
    rg1.append(rglint.points[2*i])
    rg2.append(rglint.points[2*i+1])

#%%
sio.savemat('calib09-09-19', {'cameraMatrixL': cameraMatrixL, '
    distCoeffsL': distCoeffsL, 'TL': tvecsL, 'RL': rvecsL,
                                'glintsL1': lg1, 'glintsL2': lg2, '
                                    cameraMatrixR': cameraMatrixR,
                                'distCoeffsR': distCoeffsR, 'TR': tvecsR, '
                                    RR': rvecsR, 'glintsR1': rg1, 'glintsR2'
                                    : rg2,
                                'R': R, 'T': T}, appendmat=True)

#%% reconstruct light sources
```

```
# lg1 = np.reshape(lg1, (14, 1, 1, 2))
# lg2 = np.reshape(lg2, (14, 1, 1, 2))
# rg1 = np.reshape(rg1, (14, 1, 1, 2))
# rg2 = np.reshape(rg2, (14, 1, 1, 2))
#
# lightpos1 = light.reconstruct(lg1, cameraMatrixL, distCoeffsL, rg1, cameraMatrixR, distCoeffsR, R, T)
# lightpos2 = light.reconstruct(lg2, cameraMatrixL, distCoeffsL, rg2, cameraMatrixR, distCoeffsR, R, T)
```

```
import cv2 as cv
import numpy as np
import self
from scipy.optimize import minimize
from scipy.linalg import null_space
import calibrate_new

winSize = 6
# read image and find contours, return image coordinates of max intensity of first contour
# for one image every time
def glint(images):
    glintspos, glintscontours = [], []
    for fname in images:
        img = cv.flip(cv.imread(fname), 1)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        ret, thresh = cv.threshold(gray, 127, 255, 0)
        contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
        cnt = contours[0]
        minVal, maxVal, minLoc, maxLoc = cv.minMaxLoc(cv.contourArea(cnt))
        glintspos.append(maxLoc)
        glintscontours.append(cnt)
    # cv.drawContours(img, contours, -1, (0, 0, 255), 2)
    # cv.imshow("detected", img)
    # cv.waitKey(0)
    return glintscontours, glintspos


# glints coordinates
def glints_coord(glintscontours, glintspos):
    coord = []
    for i in xrange(len(glintspos)):
        ind1, ind2 = glintspos[i][0], glintspos[i][1]
        coord.append(glintscontours[i][ind1][ind2])
    return coord


# center of sphere
def center_sphere(images, glint_contours, glint_pos):
    center = []
    for fname in images:
```

```
        # img = cv.flip(cv.imread(fname), 1)
        img = cv.imread(fname, flags=cv.IMREAD_GRAYSCALE)
        im = img.copy()
        # gray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
        canny = cv.Canny(im, 50, 200)
        circles = cv.HoughCircles(canny, cv.HOUGH_GRADIENT, 2, 900,
            param1=80, param2=60, minRadius=100, maxRadius=170)
        if circles is not None:
            for c in circles:
                for x, y, r in c:
                    i1, i2 = glint_pos[0][0], glint_pos[0][1]
                    if abs(x - glint_contours[i1][i2][0][0]) < 400:
                        cv.circle(im, (x, y), 2, (255, 0, 0), 2)
                        cv.circle(im, (x, y), r, (255, 0, 0), 2)
                        center.append([x, y])
            # cv.imshow("Circles", im)
            # cv.waitKey(0)
    return center


def undistort(cameraMatrix, p, imgpoints):
    imgundistort = []
    fu, fv = cameraMatrix[0][0], cameraMatrix[1][1]
    u0, v0 = cameraMatrix[0][2], cameraMatrix[1][2]
    for i in xrange(len(imgpoints)):
        xi = (imgpoints[i][0][0] - u0) / fu
        yi = (imgpoints[i][0][1] - v0) / fv
        r = np.sqrt(xi ** 2 + yi ** 2)
        G = 1 + (p[4]*r*r + p[5]*xi + p[6]*yi + p[7]) * r * r
        tmp1 = (xi*(1 + p[0]*r*r + p[1]*r**4) + 2*p[2]*xi*yi + p[3]*(r*r
            +2*xi*xi)) / G
        tmp2 = (yi*(1+p[0]*r*r+p[1]*r**4) + p[2]*(r*r+2*yi*yi) + 2*p[3]*
            xi*yi) / G
        tmp3 = [tmp1, tmp2]
        tmp3 = np.reshape(tmp3, (1, 2))
        imgundistort.append(tmp3)
    return imgundistort


def WCS2CCS(objpoints, rotate, tvecs):
    tmp1 = (np.dot(rotate, objpoints) + tvecs.T).T.tolist()
    objpoints_ccs = np.reshape(tmp1, 3)
    return objpoints_ccs


#%% mouse callback
def adaptive_threshold(img):
    maxValue = 0
    for i in xrange(len(img)):
        if max(img[i]) > maxValue:
            maxValue = max(img[i])
    threshold = maxValue - 150
    return threshold
```

```python


# find centers of glints by searching area around where the mouse click
def searchArea(img, x, y):
    threshold = adaptive_threshold(img)
    area, index_areaX, index_areaY = [], [], []
    ubx, uby = x + winSize, y + winSize
    lbx, lby = x - winSize, y - winSize
    if ubx > len(img[0]):
        ubx = len(img[0])
    if uby > len(img):
        uby = len(img)

    for i in range(lbx, ubx):
        for j in range(lby, uby):
            if img[j][i] > threshold:
                area.append(img[j][i])
                index_areaX.append(i)
                index_areaY.append(j)
    if len(area):
        xc = np.median(index_areaX)
        yc = np.median(index_areaY)
        xc = int(round(xc))
        yc = int(round(yc))
        # if xc not in index_areaX:
        #     xc = min(index_areaX, key=lambda x_area: abs(x_area - xc))
        # if yc not in index_areaX:
        #     yc = min(index_areaY, key=lambda y_area: abs(y_area - yc))
    return xc, yc


class Glint:
    def __init__(self):
        self.points = []

    def find_center(self, event, x, y, flags, param):
        if event == cv.EVENT_LBUTTONDBLCLK:
            xc, yc = searchArea(param, x, y)
            self.points.append([xc, yc])
            cv.drawMarker(param, (xc, yc), (0, 255, 0), markerSize=4,
                thickness=2)


def click(img, glints):
    cv.namedWindow('image')
    cv.setMouseCallback('image', glints.find_center, param=img)

    while 1:
        cv.imshow('image', img)
        if cv.waitKey(20) & 0xFF == 27:
            break
    cv.destroyAllWindows()

```

```

# save glints coordinates for multiple images
def glintscoordinates(images, glints):
    for fname in images:
        img = cv.imread(fname, flags=cv.IMREAD_GRAYSCALE)
        click(img, glints)


# reconstruction of light source
def reconstruct(imgpointsL, cameraMatrixL, distCoeffsL, imgpointsR, cameraMatrixR, distCoeffsR, R, T):
    lightest = []
    # linear triangulation
    pl, pr = self.distcoeffs(cameraMatrixL, distCoeffsL), self.distcoeffs(cameraMatrixR, distCoeffsR)
    nimgl, nimgr = self.undistort(cameraMatrixL, pl, imgpointsL), self.undistort(cameraMatrixR, pr, imgpointsR)

    for i in xrange(len(imgpointsL)):
        uil, vil = nimgl[i][0][0][0], nimgl[i][0][0][1]
        uir, vir = nimgr[i][0][0][0], nimgr[i][0][0][1]
        Ai = np.array([[0, -1, vil, 0],
                      [0, -1, uil, 0],
                      [vir*R[2][0] - R[1][0], vir*R[2][1] - R[1][1], vir*R[2][2] - R[1][2], vir*T[2] - T[1]],
                      [uir*R[2][0] - R[1][0], uir*R[2][1] - R[1][1], uir*R[2][2] - R[1][2], uir*T[2] - T[1]]])
        sol = null_space(Ai)
        print sol
        Xi = sol[:3]/sol[3]
        lightest.append(Xi)

    return lightest
```

Bibliography

[1] W. Levin, H. Kooy, J. Loeffler, and T. DeLaney, “Proton Beam Therapy,” *British Journal of Cancer*, vol. 93, pp. 849–854, 2005 Oct 17.

[2] C. G. Spagnolo, Francesco and P. Queirolo, “Uveal Melanoma,” *Cancer Treatment Reviews*, vol. 38, no. 5, pp. 549–553.

[3] K. K. B. M. BA Krantz, N Dave and R. Carvajal, “Uveal Melanoma: Epidemiology, Etiology, and Treatment of Primary Disease,” *Clinical Ophthalmology*, pp. 279–289, 2017 Jan 31.

[4] “Proton beam irradiation.” https://www.itumor.org/proton-beam-irradiation.html.

[5] “Proton therapy treatment.” https://www.protominternational.com/proton-therapy/proton-therapy-treatment/.

[6] A. Fassi, M. Riboldi, C. F. Forlani, and G. Baroni, “Optical eye tracking system for noninvasive and automatic monitoring of eye position and movements in radiotherapy treatments of ocular tumors,” *Appl. Opt.*, vol. 51, pp. 2441–2450, May 2012.

[7] A. T. Duchowski, “Eye tracking methodology,” *Theory and practice*, vol. 328, p. 614, 2007.

[8] K. Harezlak, P. Kasprowski, and M. Stasch, “Towards accurate eye tracker calibration–methods and procedures,” *Procedia Computer Science*, vol. 35, pp. 1073–1081, 2014.

[9] E. D. Guestrin and M. Eizenman, “General theory of remote gaze estimation using the pupil center and corneal reflections,” *IEEE Transactions on Biomedical Engineering*, vol. 53, pp. 1124–1133, June 2006.

[10] FedericoNesti, “Eye tracking for proton clinic environment,” 2018.

[11] R. Hartley and S. B. Kang, “Parameter-free radial distortion correction with center of distortion estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 8, pp. 1309–1321, 2007.

[12] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, Nov 2000.

[13] Y. Cui, F. Zhou, Y. Wang, L. Liu, and H. Gao, "Precise calibration of binocular vision system used for vision measurement," *Opt. Express*, vol. 22, pp. 9134–9149, Apr 2014.

[14] S. Gai, F. Da, and X. Dai, "A novel dual-camera calibration method for 3d optical measurement," *Optics and Lasers in Engineering*, vol. 104, pp. 126–134, 2018.

[15] R. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[16] G.-Q. Wei and S. De Ma, "Implicit and explicit camera calibration: Theory and experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 469–480, 1994.

[17] D. C. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering and Remote Sensing*, 1966.

[18] S. O. Heikkila Janne, "A four-step camera calibration procedure with implicit image correction," *NMR in Biomedicine*, pp. 41–47, 1995.

[19] J. Heikkila, "Geometric camera calibration using circular control points," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1066–1077, Oct 2000.

[20] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[21] "Thorlabs lens la1540-780."

[22] "Caltech camera calibration toolbox for matlab."

[23] S. Nedevschi, T. Marita, M. Vaida, R. Danescu, D. Frentiu, F. Oniga, C. Pocol, and D. Moga, "Camera calibration method for stereo measurements," *Journal of Control Engineering and Applied Informatics (CEAI)*, vol. 4, no. 2, pp. 21–28, 2002.

[24] "Opencv documentation: Camera calibration and 3d reconstruction."

[25] "Matlab single camera calibrator app."

[26] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 965–980, 1992.

[27] K. Yan, H. Tian, E. Liu, R. Zhao, Y. Hong, and D. Zuo, "A decoupled calibration method for camera intrinsic parameters and distortion coefficients," 2016.

[28] H. Li and R. Hartley, "A non-iterative method for correcting lens distortion from nine point correspondences," *OMNIVIS 2005*, vol. 2, p. 7, 2005.

[29] A. W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, IEEE, 2001.

Glossary

List of Acronyms

DCSC Delft Center for Systems and Control