HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Start date: 09/01/2015

Submitted at: 05/31/2016

Thesis

in fulfillment of the

requirements for the degree of

Master of Science

in Computer Science in Media

at the Faculty of Digital Media

Sarah Häfele

Matriculation number: 249670

Three-dimensional Reconstruction from Stereo High-speed Cameras

Advisor: *Prof. Dr. Ruxandra Lasowski*

Co-Advisor: *Dr. Bernd Reinke (Optronis)*

Master-Thesis

Three-dimensional Reconstruction from Stereo High-speed Cameras

Sarah Häfele

May 31, 2016

Hochschule Furtwangen
Fakultät Digitale Medien

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Master-Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben.

Declaration of Authorship

I hereby declare that I have authored this master's thesis independently and without any other sources than the ones declared. All directly or indirectly used sources and resources are explicitly marked and acknowledged as references.

Furtwangen, May 31, 2016,

Sarah Häfele

Abstract

The field of Computer Vision is broad and has been a highly discussed topic in recent research endeavors. It includes applications for automotive safety, gesture recognition, medical imaging and industrial automation and inspection. An area of extreme focus is the digitization of real world objects via camera enabled 3-D reconstruction. This can be achieved in several ways, including sometimes expensive high-tech lasers or depth cameras. Although there are consumer-friendly depth cameras, 3-D reconstruction with normal cameras must be addressed in order to make the technology accessible to the consumer market. Opening the technology to a wider audience would allow the development of more diverse applications.

This thesis addresses the issue of reconstructing a three-dimensional object from an image sequence captured by stereo high-speed cameras. The pipeline is based on 3-D reconstruction algorithms and was developed in the MATLAB environment.

When compared with lasers, cameras do not provide as detailed an image of an object. This lack of data can be made up for with the inclusion of additional images generated by the high-speed cameras. This statement is supported with experimental sessions which address several different approaches to 3-D reconstruction from stereo high-speed cameras.

"Any sufficiently advanced technology is indistinguishable from magic."

– Arthur C. Clarke's third law

Contents

1 Introduction

1.1 Preface

Figure 1.1: Promotional image for the *NVIDIA 3D Vision Pro* technology (*source:* [Qui10]).

Since the early years of science fiction story telling in books and movies, the concept of enriching our environment with digital information and other virtual content fascinates humanity. Movie franchises like *Star Trek* and *Star Wars* introduced unbelievable technology to a broad audience. First and foremost stories of space travel via huge ships with the help of fantastic inventions are

meant to be entertaining. Having said that, the writers try to put the sci-fi technology into the context of a possible future science to make the shows more believable and exciting. The *Holodeck* in Star Trek is a good example of such an attempt. The Holodeck is an environment simulator, which creates virtual worlds with the help of holographic projections in special rooms. It is used by the crew of the Enterprise for entertainment but also for training purposes. Although we can not yet imagine how to create a holographic image with which we can interact in such a way, the idea of fully accessable virtual worlds, the *virtual reality* (see subsection 2.1.1)[1], has been a heavily researched field since the mid 19th century (compare with [Bat09], [Tön10, p.3], [Dör+13, p.19 et seq.] and [BC03, p.4 et seq.]).

However, in many ways the actual development of the required hardware and software is still in its early stages (see section 2.2 for a historical overview of this topic), and although the concept was promising, we spent too many years "trapped" in the second dimension, leading to disappointments and the loss of people's confidence in the technology.

But with the improvement of sensors, graphics computation and displays the development of AR and VR applications can finally make significant progress. While the entertainment industry seems to be the most anticipated and closely followed development by the general public, the research on *computer vision*[2] is still mostly driven by the military, NASA, the medical field and other technical industries. The tracking, recognition, analyzation, and finally digitization of real world objects and persons are important research fields for 3-D computer

[1] The Holodeck may fall into the category of either a VR or an AR environment. The projection creates virtual content and the users can fully interact with it. The Holodeck itself is just a blank room, which makes the people interacting with it the only "real components". Therefore there is no virtual overlayed "real" environment but only the virtual world the users have access to. According to the definitions used in this thesis (subsection 2.1.1), the Holodeck is more of a VR application.

[2] An explanation on this topic can be found at subsection 2.1.2.

vision. In the future, surgeries will be fully supported with an AR overlay which displays the insides of the patient in real-time[3] and military drones could be able to distinguish between civilian and military buildings. Autonomous cars will reduce the risks on the streets and customers could design their individualized products interactively (see Figure 1.1 for the vision NVIDIA presented in their promotional image for their new 3-D technology).

Despite all the recent successes, researchers are still struggling to reconstruct 3-D shapes and analyze/ interpret images in their entirety. But why is it so difficult? Why are people still disappointed? This misperception can be dated back to the early stages of robotics and artificial intelligence, in which it was initially believed that computer vision would be the easiest step to achieve ([Sze11, p.5]).

Richard Szeliski, author of *Computer Vision - Algorithms and Applications*, calls it an "inverse problem": we try to find a solution on a basis of insufficient information provided by our imperfect technologies. We must turn to a physics- and mathematics-based approach to fill in these missing gaps and to create better *models* of our real world. These include, for example, the modeling of shading and light reflection or the computation of the natural movement of colliding objects (compare with [Sze11, p.3 et seqq.]).

In conclusion, the problems with computer vision can be broken down to mathematical algorithms, which are still merely models and subject to a multitude of errors. Some of these algorithms shall be the topics of this thesis.

[3] The technology, with which the surgeons can see in real-time stereo vision with the help of tracking markers and stereoscopic glasses, is already far advanced (see [Low16] for more examples).

1.2 About this Thesis

The present thesis is submitted in fulfillment of the requirements for the degree of Master of Science in *Computer Science in Media* of the faculty of *Digital Media*. The high-speed cameras and their software components as well as technical support were provided by the company *Optronis*.

In this thesis the challenge of 3-D reconstructing an image sequence captured with uncalibrated high-speed cameras is discussed. The research included not only historical background but first and foremost the mathematical principles behind the process of reconstruction. For the analysis several experimental sessions with different camera set-ups were taken and were used as the basis for a calibration and reconstruction pipeline in the software MATLAB.

Optronis is a German company which is specialized in the field of ultra-fast optical measuring systems since 1986[4]. Their products are used for research and science in several fields. Both their high-speed cameras for precision measurements and their streak camera technology, which makes photons visible, help monitoring and analyzing processes that are otherwise invisible for the human eye ([Opt16]).

High-speed cameras have the potential to provide a huge amount of data in a short time, which is not yet easy to be processed in real-time but will certainly be a big topic in the future (as seen in chapter 4). This can fill in the lack of data and solve the *inverse problem* which was addressed in the introduction above.

This thesis is loosely linked to my bachelor's thesis, in which the *Oculus Rift*, a head-mounted display[5], was transformed into an AR device with the help of two webcams. The software was developed in C# and the shading language *Cg*

[4]Check Optronis' webpage for more detailed information about the company and its products: `http://www.optronis.com`

[5]More information about head-mounted displays can be found in subsection 2.1.1. The bachelor's thesis is digitally attached (in German).

in the *Unity Engine* ([Häf14]).

The chapter 2 defines basic terms and gives a short historical background on the industry, ending with the current state-of-the-art of the field. It is followed by chapter 3, which covers the mathematical principles of 3-D reconstruction algorithms. An overview of interesting related works is given in chapter 4 to complete the background information.

The experimental part starts with the definition and explanation of the architecture and its components in chapter 5. The documentation of the camera calibration and then the reconstruction pipeline follows in chapter 6. The experimental chapter is concluded with a summary of results taken from the experimental sessions.

The closure in section 7.1 takes a quick glance into the future and sums up the overall issues and results.

To be able to quickly find a specific topic, a glossary is added. The printed version includes short documentation sheets of the experimental sessions. The MATLAB code as well as several examples can be found in the digital appendix. Since the data captured during the experimental sessions is too big it could not be delivered with the thesis.

2 Background

2.1 Definitions

This section covers and differentiates between the basic non-mathematical definitions and terms used in this thesis.

2.1.1 Virtual and Augmented Reality

The words virtual reality (VR) and augmented reality (AR) can be constantly read in the recent news all over the internet and other media. However, they are used in various contexts and are often defined differently. The definition used in this thesis is therefore only one small aspect of the whole field and should help to differentiate between the two concepts.

Virtual reality

Virtual reality can be defined as "an electronic simulation in which images are generated in real time (...) from a stored database and displayed in such a way as to facilitate real-time interaction with the database (...)." [Lat95, p.148]

A virtual reality application uses a combination of software and hardware to let the user experience immersion, interaction and imagination (the three I's of virtual reality) in a virtual world. Whereas the first two I's should be clear and understandable, the last one is often underrated or even left out. Since virtual

reality only counts on virtual content and given the fact that computer graphics are still not a perfect representation of our "reality", the user's imagination is a very important aspect of an application ([BC03, p.3 et seq.]).

Typically, a common computer game can not be classified as a virtual reality application unless it includes the collection and usage of other data than only the keyboard and mouse inputs. For this purpose, a VR application often comes with other interfaces such as a head-mounted display (HMD) [1]. An imaginary example for a perfect virtual reality environment is Star Trek's Holodeck, as mentioned before (section 1.1).

Augmented reality

In comparison to virtual reality, augmented reality extends our "real world" with virtual (digital) content, which must be embedded seamlessly to provide a fully immersive experience. The users can interact with virtual objects and can be provided with relevant additional information about their environment. The virtual data is computed in real-time and is displayed as an overlay on top of the "real world" in front of the user's eyes. AR highly relies on its technologies. Relevant topics, amongst others, are object recognition, location tracking, interfaces[2] and real-time computation (compare with [Tön10, p.1 et seqq.]).

The game *Ingress* by *Niantic, Inc.* (see Figure 2.1 for a screenshot), in which two factions fight over virtual portals, is an example for a location-based mobile AR game. The portals, which are linked to real places all over the world, and

[1] The idea of a HMD started with Morton Heilig's U.S. patent in 1960 (see [Hei60]). Recent popular models include the *Oculus Rift* (see [Ocu16]) and *Google Cardboard* (see [Dev16]). The HMD tracks, amongst other things, the head position and its movement and displays the stereo image.

[2] A common interface for AR was and still is the head-up display (HUD), often used for military application (see [BC03, p.6]). But also immaterial displays, such as fog screens, are researched (see [Tön10, p.25 et seq.]).

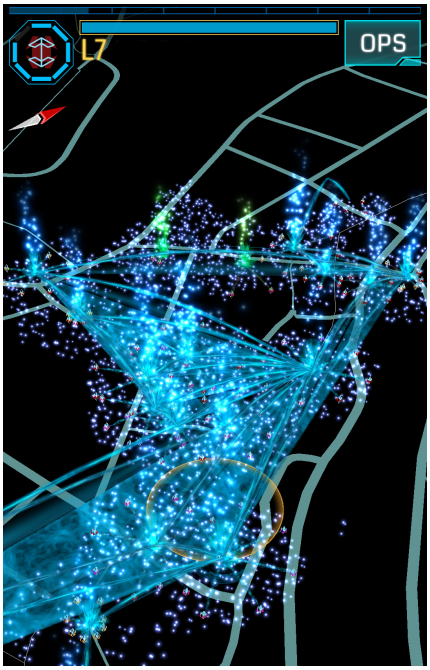


Figure 2.1: Screenshot of the mobile AR game Ingress showing the portals around the Hochschule Furtwangen University (*source:* [Häf14]).

other game functions are displayed on the user's mobile device in real-time. The users travel to these real places to conquer or defend the portals in the name of their faction ([Nia16]).

2.1.2 Computer vision

Computer vision (also referred to as "machine vision") is the "automated extraction of information from images" ([Low16]). It also refers to a whole field of science, which combines several disciplines[3] to research how to make a computer see. Computer vision is differentiated from the wide field of image processing, in which images are processed in different ways to produce new images (compare with [Low16]).

As already stated in the preface, the visual modeling of perception was, for many years, underestimated by scientists studying Artificial Intelligence. Although it is still unclear whether computer vision should be modeled on the basis of biological systems, understanding how biological vision in general, and human depth perception in particular function, seems to be a necessity[4] (compare with *Oliver Faugeras'* foreword in [HZ11, p.xi]). For this a short overview shall be given in subsection 2.1.3.

One of the research fields in Computer Vision is the 3-D reconstruction from image data (other fields are listed in subsection 2.2.3). Two important procedures to reconstruct scenes are *structure from motion* and *stereo correspondence* algorithms. The former uses multiple views of one or more cameras to reconstruct scenes and the latter is described in more detail in the following chapters, since it was used in this thesis.

[3] Among which are mathematics, computer science, as well as phsyics, the psychology of perception and the neuro sciences ([HZ11, p.xi]).

[4] Attempts to find new approaches have not yet been successful.

2.1.3 Human Depth Perception

Our three-dimensional perception is actually based on a two-dimensional image which is projected onto our retina, but from these 2-D images we are still able to get information about how far away an object is. This information is called *depth cues*, whose connection to the actual depth of field we learn from experience. Depth cues need to be applied to stereoscopic videos to create more accurate depth results ([Gol15, p.226] and [Hof09, p.28]).

These depth cues can be classified into three different groups (freely adapted from [Gol15, p.226 et seqq.] and [Hof09, p.28 et seqq.]):

i Oculomotor depth cues

These cues provide depth information for objects which are in closer range (one meter or below). Our brain receives and analyzes feedback from the eye muscles which are contracted differently depending on the distance of the object which gets focused on. We can subconsciously *feel* the muscles' contraction and draw conclusions from it. The oculomotor cues are based on two factors: *accommodation* and *convergence*[5].

ii Monocular depth cues

As the name suggests these depth cues can be perceived even with only a single eye (which is one of the reasons that people with only one eye can still perceive depth). They use the previously mentioned accommodation in combination with *pictorial* as well as *movement-produced depth cues* to estimate depth.

Pictorial depth cues

Information about depth which can be displayed in a two-dimensional

[5]The former is based on the contraction of the ciliary muscles, which allow the lens of the eye to take on different shapes in order to change their focal length. The latter uses the information of the eyes' movement as they converge on a moving object to estimate distances.

picture. Among them are ([Gol15, p.227 et seqq.]):

- *Occlusion*: An overlapped object seems to be further away than the object in front of it. We can not determine the absolute distance of these objects, since these are only relative cues.
- *Elevation*: Relative to the horizon we perceive objects which are closer to it as further away than the ones below or higher than it.
- *Relative and familiar size*: Two objects with the same size change their perceived size according to their position relative to the observer. Objects which are closer appear larger and vice versa. We use our experience to estimate sizes of objects and their distances to us (Figure 2.2 shows the difference in sizes of the people which illustrates relative sizes).
- *Perspective*: Parallel lines converge in the distance and meet in infinity. This concept was often used by artists of the Renaissance to create the optical illusion of depth in their paintings (see Figure 2.2 in which the lines converge towards one vanishing point).
- *Aerial perspective*: Objects in the far distance appear to be hazy[6] and often times have a blue tone with lower contrasts (note that the painter in Figure 2.2 used cool colors for distant objects and warmer colors for the foreground).
- *Texture gradient*: Textures appear less detailed the further they are from the observer. At the same time elements appear more dense relative to the distant of the viewer.

[6] In computer games this phenomenon is often called *distance fog*.

- *Shading*: Shadows help us to determine positions and they amplify the three-dimensional appearance of objects.

Movement-produced depth cues

With the movement of our head or our whole body we receive more cues to determine depth. One of the most important cues is the *motion parallax*. This depth cue occurs when we see closer objects pass us in a faster speed and more distant objects in a slower pace. It is also often used in 2-D side-scrolling video games (also called *side-scrollers*) to illustrate depth of field. Another movement-produced depth cue is the occlusion of objects due to movement ([Gol15, p.231 et seq.]).

Figure 2.2: Pictorial depth cues in Pietro Perugino's fresco *Entrega de las llaves a San Pedro* in the Sistine Chapel, Vatican City, 1481-1482

iii Binocular or stereoscopic depth cues

Stereoscopic depth cues are stimuli for which we need both eyes to experience depth of field. They mainly take the differences between the

right and the left image seen by our eyes into account, which makes the interpupillary distance the bases of the stereoscopic vision[7].

2.2 History

2.2.1 First Developments

In the late 50s of the 19th century *Morton Leonard Heilig* dreamed about the revolution of cinematic story telling and designed with his "stereoscopic-television apparatus for individual use" the first head-mounted display (see Figure 2.3). The helmet was planned to display stereoscopic images, play surround sound and even add smell to the sensory experience, but was never actual built. His ideas were ahead of their time and could not yet be realized with the technology of the time (compare with [Tön10, p.3] and [BC03, p.4 et seq.]).

In the decades to follow, several attempts to bring input and display devices for virtual reality onto the consumer market failed due to their high costs and their lack of software. In 1992, *Jaron Lanier* and his company *VPL Research* released the *DataGlove*[8] at the unaffordable prize point of $9000 USD. The low number of games for *Nintendo's* cheaper *PowerGlove* led to its discontinuation in 1993 soon after its introduction ([BC03, p.8 et seq.] and [Dör+13, p.19 et seq.]).

[7] Computer vision algorithms use these principles as a model for creating or analyzing depth with the help of two or more cameras (see chapter 3 for more details).

[8] The *datagloves* (also called *wire-* or *cybergloves*) are another step into more immersive input devices, since hand gestures can be directly used to interact with the virtual world.

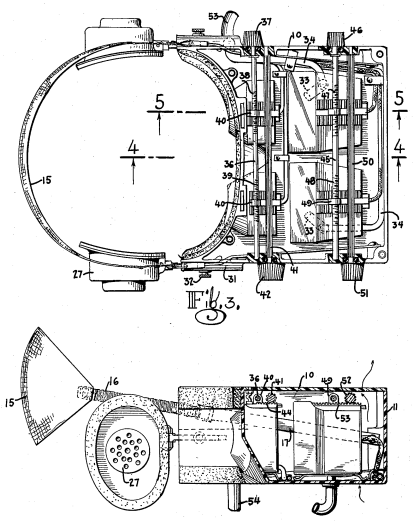


Figure 2.3: Morton Heilig's stereoscopic-television apparatus for individual use (*source: [Hei60]*).

2.2.2 Disappointments

The 80s and 90s were characterized by heavily marketed products, that ultimately disappointed the users and failed to live up to the "mediahype". The expectations for virtual reality applications were too high and the technological development was not fast enough. The hardware was too costly[9], the market for VR was too small and there were no investment funds available for companies. These deficits led to the bankruptcy of many companies and caused a serious crisis for the VR industry ([BC03, p.10]).

With the improvement of computer hardware in the late 90s, the speed of graphics computation began to meet the requirements for VR and the hardware was finally affordable for consumers ([BC03, p.10 et seq.], [Dör+13] and [Tön10, p.3]).

[9]The fastest graphics system, the *Reality Engine* by *Silicon Graphics*, cost $100,000 USD at its release in the 1990s([BC03, p.10]).

2.2.3 Today's Usage of Computer Vision

Fortunately, due to recent successes in the industry the computer vision plays an increasingly important role in many fields. *David Lowe*, Senior Research Scientist at Google Seattle, separated today's most relevant industrial applications for computer vision into the following categories[10] (freely adapted from [Low16]):

Automotive driver assistance and traffic management,
such as visual warning systems and real-time traffic management.

Eye and head tracking,
which is also included in head-mounted displays.

Film and video,
e.g. tracking of the ball in sports for sport analysis or motion tracking systems for animated movies.

Gesture recognition,
such as full-body motion sensing with the Mircosoft Kinect and the use of gesture recognition as computer inputs for gaming or other computer-related activities.

Industrial automation and inspection,
like systems for inspection and process control in the electronic or food and agriculture industry.

Medical and biomedical,
e.g. the detection and tracking of markers in real-time stereo vision for surgeons.

[10] Although he states that his list is not complete and not up-to-date, it can be used as an overview. Go to his homepage to read more about specific applications: [Low16]

Object recognition and AR for mobile devices,
including real-time product recognition and image refinement applications, such as face-swapping apps.

Photography,
e.g. automated panorama stitching for multiple images.

Safety monitoring,
monitoring of areas to warn of accidents, e.g. in swimming pools.

Security and biometrics,
detection and counting of people for security purposes, fingerprint or license plate recognition.

3-D modeling,
3-D scanning of objects or the 3-D reconstruction from a number of photographs.

Web and cloud applications,
such as image retrieval based on content.

Regarding all these examples, the entertainment industry seems to be the slowest candidate, which is partly owed to its users high expectations regarding the (graphics) quality.

3 Mathematical Principles

Games and other virtual applications are based on mathematical models and often times need to be calculated in real-time on a computer system. The most common operations and algorithms for capturing and reconstructing 3-D models involve vectors and matrices, which are also the main concepts the MATLAB programming language is built on. Other commonly used branches of mathematics in computer vision are trigonometry, algebra, statistics and calculus ([Gre14, p.165]).

The following sections shall introduce the mathematical backgrounds of some of the most important algorithms in computer vision. For this, the basic 2-D and 3-D primitives in combination with the 3-D to 2-D projection need to be discussed first. Mathematical problems in multiple view geometry can often be broken down into 2-D space problems. Readers who have already studied computer graphics can skip the basic chapters. The explanations shall only serve as an overview and summary of the topics ([Sze11, p.29] and ([Gre14, p.165 et seq.]).

3.1 Projective and Single-View Geometry

Three-dimensional shapes can be described with geometrical primitives, such as points, lines and planes. A virtual world is created out of such 3-D objects, which all have a position, orientation and scale. Typically, geometry needs to

start with a *point* ([Sze11, p.29 et seqq.] and [Gre14, p.166 et seqq.]).

3.1.1 2-D and 3-D points

A 2-D point can represent a pixel coordinate in an image and can be described as an ordered pair of real numbers

$$\mathbf{x} = (x, y) \in \mathbb{R}^2 \tag{3.1}$$

or alternatively,

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.2}$$

Points have *equivalence classes*, which are represented by adding an extra coordinate, creating a coordinate triple. This means that $x = (x, y, 1)$, $x = (2x, 2y, 2)$ and $x = (kx, ky, k)$, for any $k \neq 0$, represent the same point, they only differ by a multiple. By dividing through k the original coordinates are retrieved. These coordinate triples are called *homogeneous coordinates*. The equation $x = (x, y, 0)$ represents a *point at infinity*, since the coordinate points can not be divided by 0. Those points form a line in the two-dimensional projective space, which is called the *line at infinity* ([Sze11, p.30] and [HZ11, p.2]).

The same principles apply for points in three-dimensional coordinate systems. Thus, a 3-D point can be represented using inhomogeneous coordinates:

$$\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \tag{3.3}$$

or alternatively using homogeneous coordinates by adding another coordinate:

$$\mathbf{x} = (x, y, z, w) \in \mathbb{P}^3 \tag{3.4}$$

Note that the *points at infinity* in 3-D space form the *plane at infinity* ([Sze11, p.31] and [HZ11, p.2]).

3.1.2 Coordinate Systems

Types of coordinate systems. As stated above, a point is represented with its coordinates in a n-dimensional space, whereas in computer vision n usually has the value 2 or 3. Hence, coordinate systems play an important role in computer vision tools. While the *Cartesian coordinate system* is the most commonly used coordinate system in the field, it is certainly not the only one. The following list will introduce three important coordinate systems, which use different concepts: ([Gre14, p.166 et seq.].

i *Cartesian coordinate systems*: Two or three fixed perpendicular axis which form the 2-D or 3-D space. The points are represented by (P_x, P_y) or (P_x, P_y, P_z) and an illustration can be seen in Figure 3.1.

ii *Cylindrical coordinate systems*: A system with a vertical axis h, a radial axis r and the yaw angle Θ. The points are represented by (P_h, P_r, P_Θ).

iii *Spherical coordinate systems*: A system specified by the radial distance r of a point from the origin, the yaw angle Θ and a pitch angle ϕ. Thus, a point is represented by (P_r, P_ϕ, P_Θ).

This thesis is concentrating on points represented in the Cartesian coordinate system, since it is the most used system in computer graphics and computer vision environments.

Left-handed vs. right-handed orientation. Three-dimensional Cartesian coordinate systems can be arranged in two different directions: right-handed and left-handed. They differ only in the orientation of one of the three axis,

for example the z-axis points to the front in a right-handed, and to the rear in a left-handed coordinate system (as illustrated in Figure 3.1). Often the y-axis points downwards, defining a left-handed coordinate system. The resulting point representations in both systems are not much different from each other: by negating the y-coordinate of the points the coordinate system becomes right-handed. Also it is important to note that the actual position of the point in space does not change, it is only the interpretation of it which is defined by this orientation. It is up to the programmer to chose a system and it is important to be consistent throughout one environment ([HZ11, p.164 et seq.] and [Gre14, p.167 et seq.]).

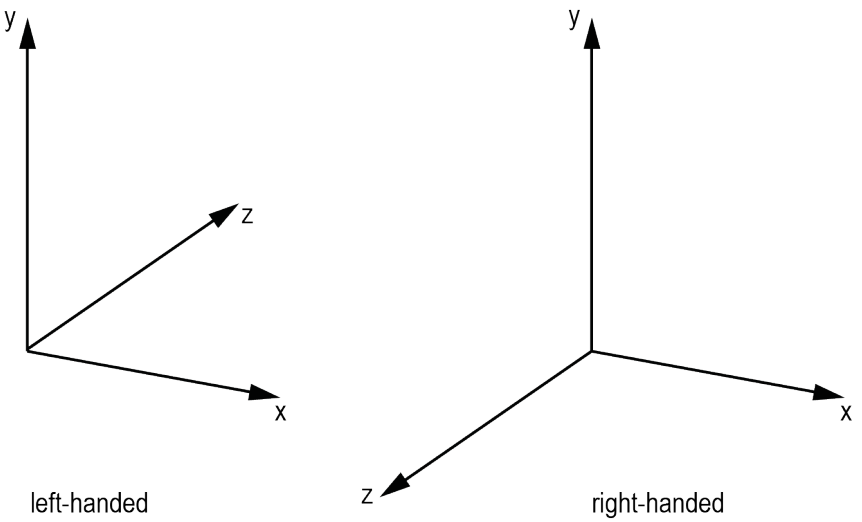


Figure 3.1: Left- and right-handed Cartesian coordinate systems (*source: own illustration based on* [Gre14, p.167]).

3.1.3 3-D to 2-D Projections

To display a model of the 3-D world on a computer screen one has to map the 3-D objects to a 2-D representation. In this process the 3-D structure gets

projected on a two-dimensional image, losing one dimension. Thus, there is a minimum of two coordinate systems involved: the world-space and the 2-D image coordinates. Their relationship to each other can be modeled with the help of a *central projection*: a ray from a point X_i in real world space (*world point*) passes through the fixed *center of projection* C and intersects with the *image plane* (as seen in Figure 3.2). This intersection x_i represents the image of the real world point X_i ([HZ11, p.6 et seq.]).

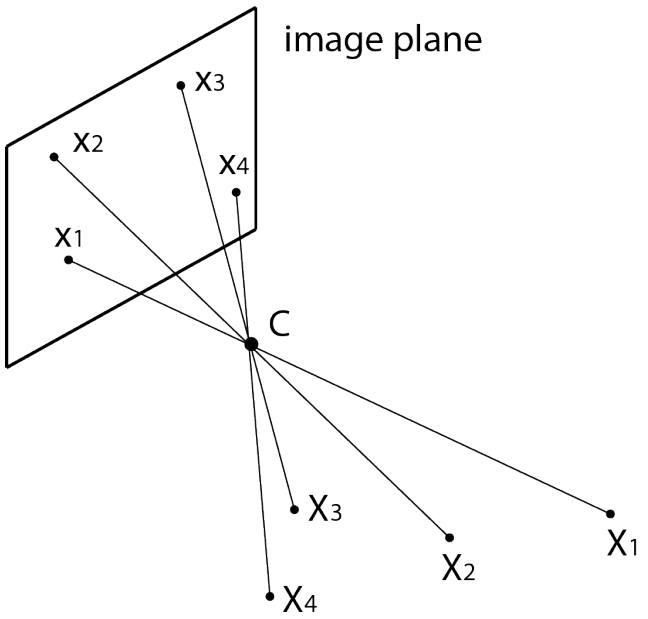


Figure 3.2: Projection of the world points X_i to the image plane through the center of projection C (*source: own illustration based on* [HZ11, p.8]).

This model can be seen as a simple camera, in which the center of projection is the lens of the camera. The mapping from the projective spaces $\mathbb{P}^3$ to $\mathbb{P}^2$ may be represented by a 3×4 matrix P, which takes the homogeneous coordinates $(X, Y, Z, T)^T$ of a point in $\mathbb{P}^3$ and the center of projection into account. The most general imaging projection is known as the *camera matrix* P. Hence, the

projection of a three-dimensional world point (in its homogeneous coordinates) to a two-dimensional image point (in its homogeneous coordinates) with this simple camera concept can be expressed as ([HZ11, p.7] and [Sze11, p.42 et seqq.]):

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = P_{3\times 4} \begin{pmatrix} X \\ Y \\ Z \\ T \end{pmatrix} \tag{3.5}$$

3.1.4 The projective Camera

Real-world 3-D objects can be reconstructed by using one or multiple cameras to digitize them (later chapters will cover this process in more detail). As stated above, a camera is the mapping of the 3-D world space to a 2-D image. This mapping is represented with certain matrices P. Thinking of real cameras one has to take their own properties, like their focal length and aspect ratio, into account as well. These properties are called *internal camera parameters* or *intrinsics* and are stored in a 3×3 matrix K ([HZ11, p.152 et seq.]).

Cameras can be classified into *finite cameras* and cameras which have their center at infinity (e.g. *affine cameras* for parallel projection). Each camera model has its own matrix which represents its particular camera mapping. The models important for this thesis are using the *central projection* (see Equation 3.5) as the basic principal ([HZ11, p.153]).

The pinhole camera. The pinhole camera is the simplest finite camera model and shall serve as an introduction for *projective geometry* and its mathematical expressions. The mapping of a pinhole camera model can be expressed by

rewriting Equation 3.5 as:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.6}$$

with $Z = f$ as the *image plane* or *focal plane* and the world-point $\mathbf{X} = (X, Y, Z)^T$. The right part of the equation consists of the multiplication between the camera matrix P for the pinhole model and the world point in homogeneous coordinates. Alternately P can be written as $P = diag(f, f, 1)[I|0]$. With this the image point can be written in short as ([HZ11, p.153 et seq.]):

$$\mathbf{x} = P\mathbf{X} \tag{3.7}$$

Taking into account the fact that the origin of coordinates in the image plane may not be at the *principal point* $(p_x, p_y)^T$, this principle point offset must be addressed with ([HZ11, p.155]):

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.8}$$

Whereas

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{3.9}$$

is the *camera calibration matrix*. Equation 3.8 can then be written in short as

$$\mathbf{x} = K[I|0]\mathbf{X}_{cam} \tag{3.10}$$

Camera rotation and translation Real world points are usually expressed in a different coordinate system of their own: the *world coordinate system* or *world frame or space*. This system stands in relation to the camera coordinate system via a rotation and translation as illustrated in Figure 3.3.

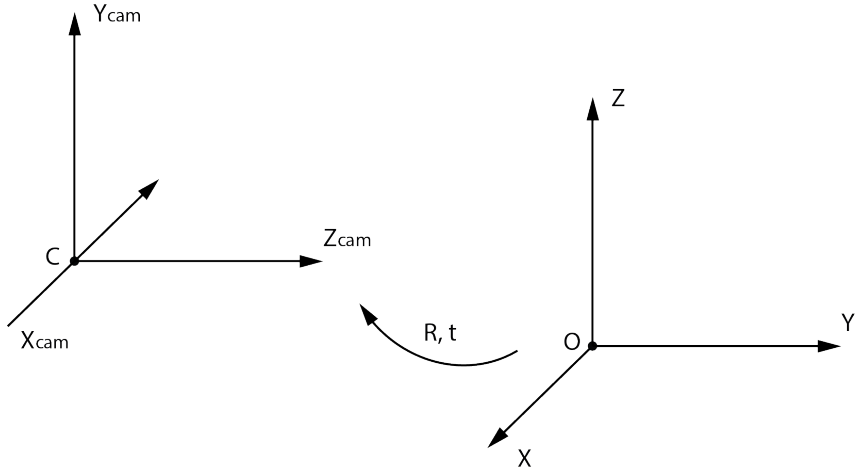


Figure 3.3: Transformation between world and camera coordinate systems (*source: own illustration based on* [HZ11, p.156]).

The rotation R is represented by a 3×3 matrix. The image of a world point in the camera coordinate system is then equated with ([HZ11, p.155 et seq.]):

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{3.11}$$

whereas $\tilde{C}$ represents the camera center in world coordinates. In combination with Equation 3.10 this leads to the general mapping given by a pinhole camera:

$$\mathbf{x} = KR[I| - \tilde{C}]\mathbf{X} \tag{3.12}$$

with $\mathbf{X}$ now written in world coordinates, K representing the *internal camera parameters* and R and $\tilde{C}$ representing the *external camera parameters* giving a projective camera 11 independent transformation parameters ([HZ11, p.156 et seq.] and [Luh+14, p.274].

Note For computer vision the Euclidean space $\mathbb{R}^n$ is problematic in many ways. Parallel lines meet "at infinity", which is merely a model but typically all points are equal in the real world - there is no origin. Euclidean as well as affine transformations "preserve"[1] points at infinity, making them different. For convenience, computer vision uses the projective space $\mathbb{P}^n$ to map 3-D objects to 2-D images. This is achieved by extending the Euclidean space with a line (or a plane) at infinity. Now points at infinity are not any different than other points and they are projected to other points with projective transformations. A projective camera therefore is a map from $\mathbb{P}^3$ to $\mathbb{P}^2$. However, it is still important to remember that cameras are Euclidean devices ([HZ11, p.1 et seqq. and 165]).

3.2 Epipolar and Two-View Geometry

One way to recover 3-D structure is to use the data of two (or multiple) camera views to search for corresponding points in image planes to calculate the absolute position of these points in world space. This section will cover the topic of *epipolar geometry* which describes the intrinsic projective relationship between two views. Those two views can be captured with two cameras simultaneously

or with a single camera sequentially, which is geometrically equivalent. Both views have their own camera matrix P, P' and their own sets of image points $\mathbf{x} = P\mathbf{X}$ and $\mathbf{x}' = P'\mathbf{X}$, whereas $\mathbf{x}$ and $\mathbf{x}'$ are called *corresponding points* since they represent the same 3-D world point $\mathbf{X}$ just from different viewing perspectives ([HZ11, p.238]).

The concepts discussed in this chapter can address the following issues (freely adapted from [HZ11, p.238]):

i **Correspondences geometry:** Retrieving the image point $\mathbf{x}'$ with the given corresponding image point $\mathbf{x}$.

ii **Camera geometry:** Retrieving the camera matrices P and P' with the given set of corresponding image points.

iii **Scene geometry:** Retrieving the position of the 3-D world point $\mathbf{X}$ with the given set of corresponding image points and the two known camera matrices P and P'.

3.2.1 The Structure of Epipolar Geometry

First, the structure of the relationship between the two views shall be defined (Figure 3.4 illustrates the entities involved). The two *camera centers* C and C' are connected through the *baseline*, which also defines the axis of a pencil of image planes[2]. A *3-D world point* X gets projected onto the two image planes as the points x in the first camera view and as x' in the second. X, x and x' are *coplanar* - they lie in the plane π. Most importantly, the rays back-projected from the two image points, intersecting in X also lie in π. This means that the plane π can be defined by the baseline and the ray from x. Going further, x' then needs to lie on l', which is the image in the second

[2]The word *pencil* means here a family of geometric objects with certain common properties, such as common lines, etc.

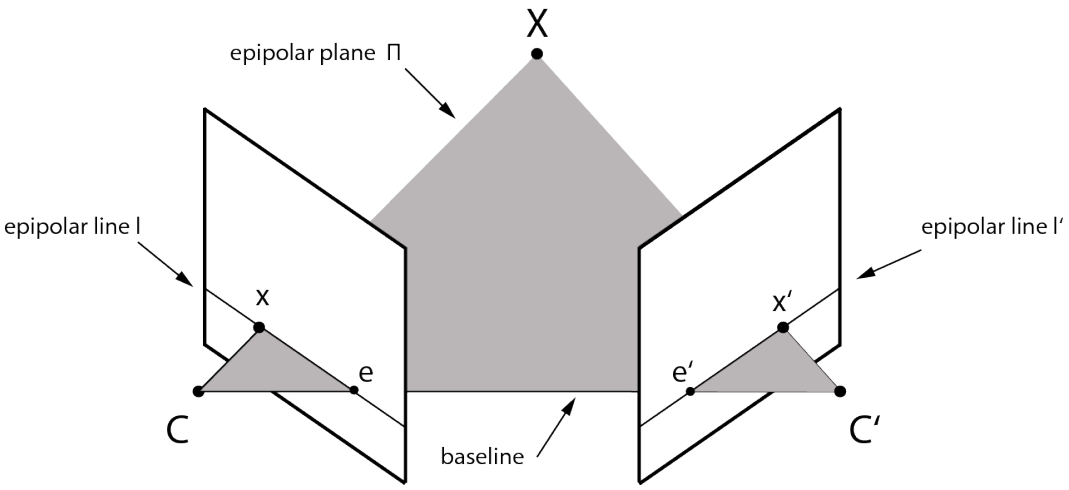


Figure 3.4: Involved entities of the point correspondence geometry (*source: own illustration based on* [HZ11, p.240]).

view of the back-projected ray from x in the first view. This line is called the *epipolar line*, which corresponds to x ([HZ11, p.239 et seq.])[3].

Other entities involved are the *epipoles*, which are the points of intersection of the baseline (joined camera centers) with the image planes: the epipole e is the image in the first view of camera center C' and vice versa e' in the second view is the image of C. With different world points X the corresponding epipolar planes "rotate" about the baseline, building a set (a pencil) of many epipolar planes with their epipolar lines (as shown in Figure 3.5). All these epipolar lines intersect at the epipole.

3.2.2 Fundamental and Essential Matrix

The concept explained above can be represented algebraically as well - with the so called *fundamental matrix*. The fundamental matrix can be derived in many

[3]Vice versa, there is an epipolar line l in the first image corresponding to x'.

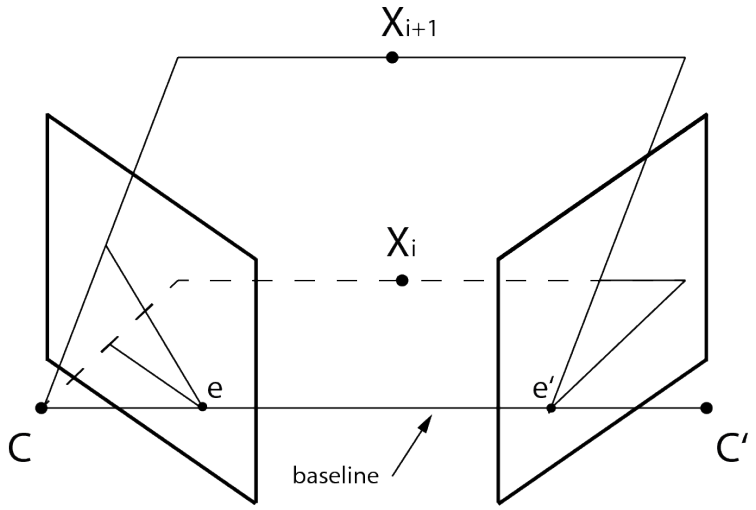


Figure 3.5: Pencil of epipolar planes with varying world points $\mathbf{X}$ (*source: own illustration based on* [HZ11, p.240]).

ways, for the following derivation the projective geometry (see subsection 3.1.4) as well as the epipolar geometry is needed.

Note It is important to clarify in which direction the rotation is interpreted, namely which camera view gets multiplied first or which coordinate system is used as a basis. The results are equivalent but differ in their appearance. In common literature there is no standardized order which makes it slightly difficult to compare different results with each other. One has to choose an order and stick with it. All values are then only to be interpreted relatively to the chosen order of camera pairs.

As mentioned above, each camera has its own projection matrix. Using parts of Equation 3.12 the two projection matrices can be defined as ([Luh+14, p.309]):

$$P = KR[I|0] \qquad P' = K'R'[I|-b] \tag{3.13}$$

with the rotation matrices R and R', the calibration matrices K and K' and the base vector b. As already stated above, the corresponding image point $\mathbf{x}'$ lies on the epipolar line l. The map $\mathbf{x} \mapsto l'$ can then be defined by the fundamental matrix F. The fundamental matrix F is a homogeneous 3×3 matrix which can be described with 8 degrees of freedom. It stores the relative orientation data, such as the interior orientation. It satisfies the following equation (taken from [HZ11, p.245 et seq.], [LZ99, p.2] and [Luh+14, p.310])[4]:

$$\mathbf{x}'^T F \mathbf{x} = 0 \qquad \text{for any pair of corresponding points } \mathbf{x} \leftrightarrow \mathbf{x}' \tag{3.14}$$

This equation is also called the *coplanarity condition* since it derives from the fact that the rays are coplanar.

For the *epipolar lines* the following equations can be used ([HZ11, p.246]):

$$l' = F\mathbf{x} \qquad \text{epipolar line corresponding to } \mathbf{x} \tag{3.15}$$

$$l = F^T\mathbf{x}' \qquad \text{epipolar line corresponding to } \mathbf{x}' \tag{3.16}$$

Since all epipolar lines intersect at the *epipoles*, these two points can be derived from Equation 3.16 and Equation 3.15 respectively:

$$Fe = 0 \qquad F^T e' = 0 \tag{3.17}$$

Note To sum it up, the fundamental matrix F describes the relationship between the camera P and a second camera P' - whichever cameras are designated as the first and the second camera is up to the programmer. F^T

[4]Note: [Luh+14] actually defines it the other way around: $\mathbf{x}^T F \mathbf{x}' = 0$. For the sake of clarity, the following euqations are all based on the order presented by [HZ11]. To achieve the other result one has to *transpose* the fundamental matrix, which changes the order of the camera pair.

then describes the opposite order P', P ([HZ11, p.245].

The *essential matrix* can be used for a calibrated camera system (if the camera calibration matrix K is already known). The essential matrix can be seen as a specialization of the fundamental matrix and the following relationship between both can be formed ([Luh+14, p.310] and [HZ11, p.257]):

$$E = K'^T F K \tag{3.18}$$

With this computed geometry the relative pose and calibration for a pair of cameras in world space is known. The epipolar lines corresponding to a pixel in one image can now be used to find the corresponding pixel in the second view. For this, different corresponding algorithms can be used, such as *optical flow*. Due to the computed epipolar geometry, the corresponding points only need to be searched along the epipolar lines. To speed up the search, the image pairs can be *rectified* first ([Sze11, p.472 et seq.]).

3.2.3 Image Rectification and Disparity

By *rectifying* image pairs corresponding horizontal lines become epipolar lines, which means that all corresponding image points are then lying on the same horizontal lines. This makes the search for corresponding points much easier since the pixels do not have to be compared on *skew lines* but can be compared only on the same *scan lines* ([LZ99, p.1]).

Note Although image rectification works the best for cameras which are standing next to each other, it can still be achieved in other scenarios as well. However, the input images should not be too verged and can not have too big of a size difference. Rectification is not a must and there are other methods to make the search for corresponding points easier, e.g. *plane sweep*.

> In general, rectifying an arbitrary collection of images with non-collinear optical centers simultaneously is not possible ([Sze11, p.472 et seq.]).

During rectification the pixel coordinates of the image pairs are transformed into other pixel coordinates with the help of 2-D projective transformations (*homographies*). This is usually achieved by first rotating both cameras so that their viewing directions are perpendicular to their *baseline*. Then, the cameras' *up vectors* have to be made perpendicular to the camera center line. If the focal lengths are different, the images then need to re-scaled, magnifying smaller images. Figure 3.6 illustrates possible rectification steps ([LZ99, p.1], [Sze11, p.473] and [Luh+14, p.436]).

Since the process of rectification is a science in its own, the author would like to suggest [LZ99] as further readings for full details of this procedure.

The *standard rectified geometry* leads to the following "simple inverse relationship" between disparities d and 3-D depth values Z ([Sze11, p.473]):

$$d = f\frac{B}{Z} \tag{3.19}$$

with the *focal length* f (in pixels), baseline B and

$$x' = x + d(x, y,) \qquad \text{with } y' = y \tag{3.20}$$

for the computation of corresponding pixel coordinates.

With this the so called *disparity map* $d(x, y)$ can be estimated. The term *disparity* is used here to describe the difference in locations of corresponding points. The information for corresponding pixels at similar locations is stored in a *disparity space image* (DSI) $C(x, y, d)$ ([Sze11, p.473]).

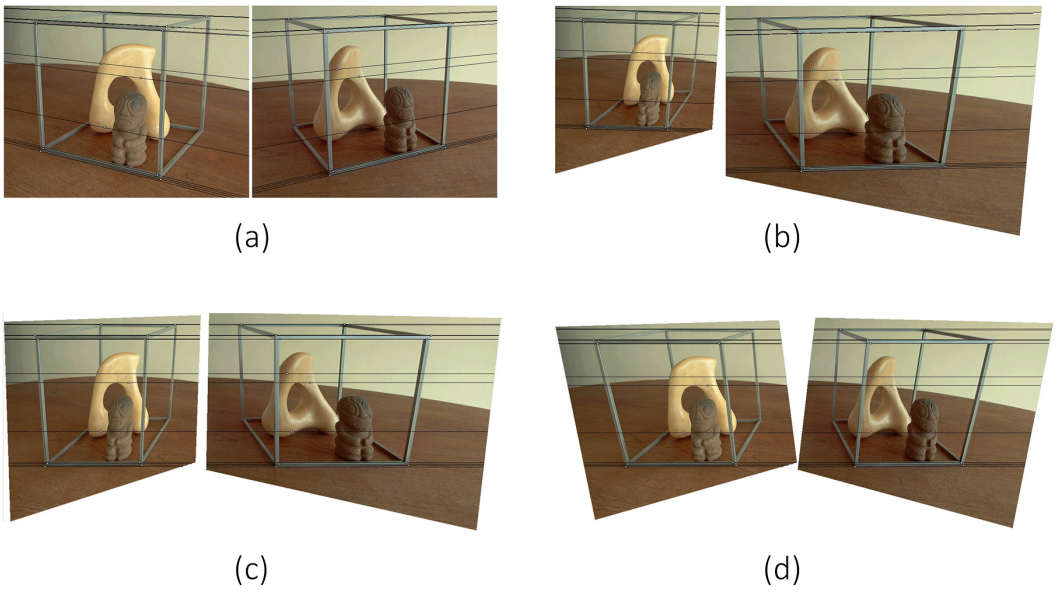


Figure 3.6: Stages of the rectification algorithm proposed by [LZ99]; **(a)** Original image pair with epipolar lines; **(b)** image transformation with projective mapping to make epipolar lines parallel to each other; **c** image transformation which rectifies the image pair - the epipolar lines are now horizontally aligned; **d** Final rectified image with reduced horizontal distortion (*source:* [LZ99, p.12]).

4 Related Works

Due to the broad nature of the field of computer vision, it is difficult to find work that is similar to the following procedure. The ultimate goal for this thesis is to reconstruct a 3-D sequence using images captured with uncalibrated high-speed cameras, the technique generates a lot of data that must then be processed. The algorithms used belong to the field of stereo calibration. In section 7.2 the connection between this thesis and the field of robotics will be briefly addressed.

[Low16] provides a comprehensive overview of the applications and recent research in the field of computer vision, all of which are worthy of quotation in this thesis.

A research group from Furtwangen University studied the reconstruction of facial expressions with MATLAB and stereo calibration algorithms. Although the research did not involve any high-speed cameras it served as a basis for this thesis.

The paper *Automatic rectification of long image sequences* from K. Okuma, J. J. Little and D. G. Lowe from 2004 researched the topic on compensating transformations, made by a fixed camera during the capture of a long sequence of images. The proposed algorithm corrects other projection errors as well and rectifies the images. The computation was tested on long sequences of a hockey match. The mathematical background follows the one used in this paper. To eliminate outliers and select inliers the *RANSAC* algorithm was used. The research takes the approach one step further and maps the projected images

onto a model ([OL04]).

An even older piece of research on the topic, *Temporally Coherent Stereo: Improving Performance Through Knowledge of Motion* from V. Tucakov and D. G. Lowe from 1997, computes the stereo over disparities and creates a depth map. The algorithm uses depth maps and estimates the motions of a robot to move it further. The relatively old research shows the close connection of the algorithms presented in this thesis to the field of robotics ([TL97]).

The more recent study *Using Stereo for Object Recognition* from S. Helmer and D. G. Lowe from 2010 gives a good overview of the topic of more recent object recognition. It also uses depth information for recognition. The paper mentions the challenges of shape based objects regarding their difficult reconstruction ([HL10]).

A closely related topic is the structure from motion algorithm which should be briefly mentioned at this point. The research group *BigSFM*[1] tries to reconstruct the world from internet photos. The research includes feature detection, image matching, optimizations and object recognition. The website provides many recent projects, papers and open source code.

The reconstruction of stereo high-speed sequences could not be found in any other recent paper. MATLAB provides several tutorials and examples regarding the Computer Vision System Toolbox and the Stereo Camera Calibrator in particular.

Computer vision is a heavily researched topic, this is especially in the field of robotics where the algorithms are used for path finding, object recognition, and obstacle avoidance.

[1] The documentation and other information can be found at `http://www.cs.cornell.edu/projects/bigsfm/`.

5 Components

This chapter describes the hardware and software used in the experiments. An illustration of the final architecture can be found in section 5.3.

5.1 Hardware

5.1.1 The High-speed Cameras and related Equipment

The two cameras used in the scenario are color high-speed cameras of the model *CR3000x2* (*CamRecord* series) from *Optronis*.

The following list summarizes the model's specifications (a full data sheet is included in the digital appendix of this thesis):

- **Input and output interfaces (I/O-interfaces):**
  - Gigabit-Ethernet (GigE)
  - VGA video output
  - Interfaces for external trigger, camera synchronization and low-voltage power supply (see Figure 5.1a)
- **Mechanical interfaces:**
  - Nikon F-Mount or C-Mount
  - Screw threads for camera tripods etc. on three sides (1/4-20 UNC), as seen in Figure 5.1b

- **Sensor resolution:** 1696 x 1710 pixel
- **Frame rate:** 540 fps (at max. sensor resolution)[1]

The image sequences can be directly exported into the file formats BMP, TIFF, AVI or MPEG. Since the captured data uses a lot of storage, a sufficiently large hard disk is necessary. In the following experiments the cameras will have this set-up (from the perspective of the cameras' viewing direction):

Right camera. CR3000x2 camera with the number *1838-ST-C-013* and 8 GB ring buffer, set as synchronization master due to the smaller buffer size[2].

Left camera CR3000x2 camera with the number *1838-ST-C-013* and 16 GB ring buffer.

Lense. Two simple *Nikon* 50 mm lenses with fixed focal lengths were used (see Figure 5.1).

Cables and connections. To connect the cameras with each other, with the trigger (see below) and with the computer, the following data cables were used:

- Three *cat 7 Ethernet cables*: for connecting the two cameras with the computer.
- Three *coaxial cables* with *BNC connectors*: for the trigger and for the synchronization.

Furthermore a gigabit Ethernet switch is needed for the data transfer between the two cameras and the computer.

[1] The cameras can have higher frame rates with lower sensor resolutions.
[2] The smaller buffer size serves as a benchmark for the overall capture duration since this camera could not store more footage.

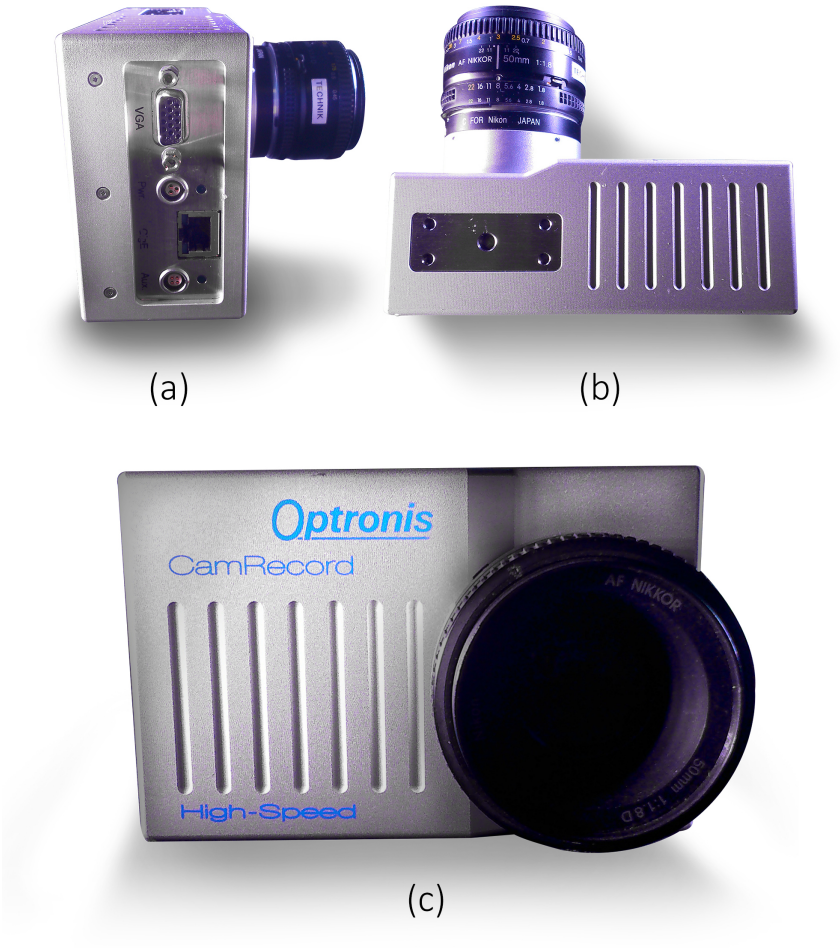


Figure 5.1: The CR3000x2 high-speed camera from different angles: **(a)** I/O-interfaces at the side; **(b)** top view with one of the three screw threads; **(c)** frontal view.

Synchronized triggering. The cameras can be triggered by software (see subsection 5.2.1) or with a mechanical external trigger (see Figure 5.2). The latter was used in the final architecture. The external trigger and the two cameras are connected with two coaxial cables with *BNC connectors*.

Figure 5.2: External mechanical trigger.

Lighting. The high frame rate of the cameras results in serious lighting issues. The scene has to be very brightly illuminated to get satisfying image results, which is especially true for a successful camera calibration. In addition, lighting must not be arbitrary, because the dim cycles of certain lamp filaments are visible as flicker on the images due to the high speed of the cameras. Another type of artifact can arise with the usage of *HMI lighting*. With these lights the electrical arc traveling through the gas can be visible on the output images (called *Arc Wander*)[3]. To avoid these problems, either lamps with constantly lit filaments or special LED lights need to be used. LED lights come in a variety of refresh rates and generally the ones designed for the film industry will not show any flicker as long as they are not dimmed. Hence, three *EUROLITE* LED lights and one MultiLED from *Optronis* were used to illuminate the scene. The final lighting set-up can be seen in Figure 5.3.

[3] A nice summary of this topic can be found at `http://www.lovehighspeed.com/lighting-for-high-speed/`. The website states that a frame rate of 1000 fps needs

Figure 5.3: Lighting set-up consisting of three *EUROLITE LED CLS-18 QCL* and one MultiLED

5.1.2 Calibration Pattern

The stereo calibration toolbox for MATLAB (explained in section 5.2.2) uses a checkerboard pattern to calibrate cameras. It does not natively support other calibration patterns. It is important to have an odd number of squares in one direction and an even number in the other direction, which lets the app determine the orientation of the pattern. The longer side will be the x-direction ([Mat16b]). A checkerboard pattern can be loaded directly in MATLAB with

```
1 open checkerboardPattern.pdf;
```

A simple pattern is included in the appendix of this thesis. The pattern needs to be printed out as clean as possible and needs to be affixed to a flat surface. In the first few experiments a checkerboard with a square size of 30 cm was used. Because the cameras' field of view is small (the reasons for

5.25 times more light than a frame rate of 25 fps.

this are explained in chapter 6), the size of the checkerboard needed to be reduced. The final checkerboard pattern has a square size of 3 mm (as seen in Figure 5.4). The process of checkerboard recognition is explained in more detail in section 6.1.

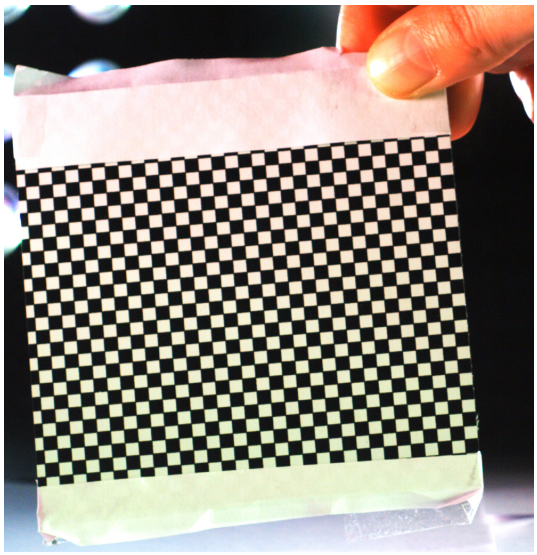


Figure 5.4: Checkerboard pattern with a square size of 3 mm for camera calibration

5.2 Software

This thesis was written in LaTeX. Other software used is explained in the following sections.

5.2.1 Timebench

The software *TimeBench*[4] was developed by *Optronis* and is included in the delivery of their high-speed camera series. It lets the user capture high-speed image sequences and comes with several tools to adjust, analyze and export these sequences ([Opt16]).

This software was chosen due to its close connection to the high-speed cameras that were used. It is also well optimized to process large amounts of image data. The software can also be used with third party video cameras.

TimeBench was first used to create image pair sequences of the calibration pattern in several positions and angles for the camera calibration process (see section 6.1). The software was then used to record a high-speed image sequence of moving objects, the objects were later reconstructed in MATLAB (see section 6.2).

5.2.2 MATLAB

For the camera calibration as well as for the 3-D reconstruction, the commercial platform *MATLAB* by *MathWorks* was used. MATLAB is a software that is specialized in solving mathematical problems (especially matrix operations) and visualizing data. The MATLAB programming language is matrix-based and implements object-oriented concepts like classes and inheritance. The program comes with a large library of built-in toolboxes, which can be used to address many different engineering and scientific problems. Another of MATLAB's strengths lies with its active community, which provides custom scripts and discusses recent developments in the scientific field ([Mat16a]).

A large portion of the MATLAB community is involved in researching the algorithms that drive the field of multiple view geometry in computer vision

[4]More information about TimeBench can be found on the official website at `http://www.optronis.com/en/products/high-speed-cameras/software.html`.

(see chapter 4 for examples). MathWorks also implements new content on this topic regularly.

The reasons stated above led to the usage of MATLAB as the development environment of choice in this thesis. Initially MATLAB R2015b was used, but due to newly added functions for data refinement the software needed to be updated to MATLAB R2016a.

The 3-D reconstruction pipeline was programmed and organized in one MATLAB script file. To obtain the required stereo parameters a built-in toolbox was used, which will be discussed in the following sections. Documentation for all functions used and a step-by-step-guide can be found in chapter 6.

One of the biggest problems with MATLAB is its documentation. Although it often provides good examples and a lot of details, it is not explicit enough in other important areas. Certain algorithms used in the built-in functions are often times not explained, making it difficult to find mistakes when the data output is unexpected[5]. The toolboxes often lack information as well, for example an explanation of how the pipelines work, which coordinate system is used, etc. These problems restrict the programmer quite a bit and force one to either find a work around or use third party programs.

Computer Vision System Toolbox

The *Computer Vision System Toolbox* from MATLAB is a library with tools which are used for detecting, matching, tracking or simulating in computer vision. In this thesis the toolbox is used for camera calibration and 3-D scene reconstruction.

The Computer Vision System Toolbox implements a right-handed world coordinate system in which the x-axis points to the right, the y-axis points

[5] This "blackbox" problem with which you can not see how the output is created may be owed to the fact that MATLAB is (and is used as) a commercial software.

down and the z-axis points into the viewing direction of the camera, as shown in Figure 5.5[6].

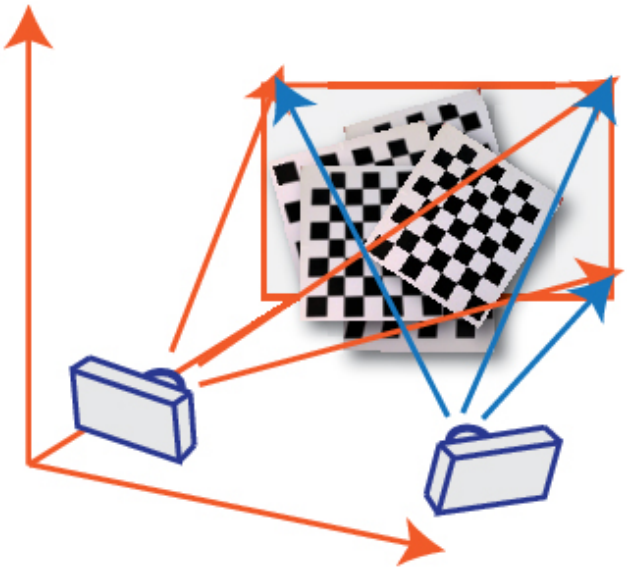


Figure 5.5: Camera-based coordinate system of MATLAB (*source:* ([Mat16b]).

Stereo Calibration Toolbox for MATLAB

The *Stereo Calibrator Toolbox* for MATLAB is an MATLAB application which estimates the stereo parameters of an uncalibrated stereo camera system. The toolbox was used to calibrate the two high-speed cameras in the final pipeline ([Mat16b]).

Camera Calibration Toolbox for MATLAB

The *Camera Calibration Toolbox* for MATLAB is another popolar alternative to calibrate a camera system. It was used to evaluate the results taken from

[6]Taken from `http://www.mathworks.com/help/vision/gs/coordinate-systems.html`.

the Stereo Calibration Toolbox. It is also included in the open source computer vision library *OpenCV*. The website provides several example projects which can be used as tutorials and it includes general information about camera calibration, which makes it a good starting point for further studies ([Bou15]).

5.3 Experimental Architecture

The final experimental architecture of the thesis (as seen in Figure 5.6) includes four lights, two synchronized high-speed cameras on a camera rig with an external trigger and the two softwares *TimeBench* and *MATLAB* with toolboxes.

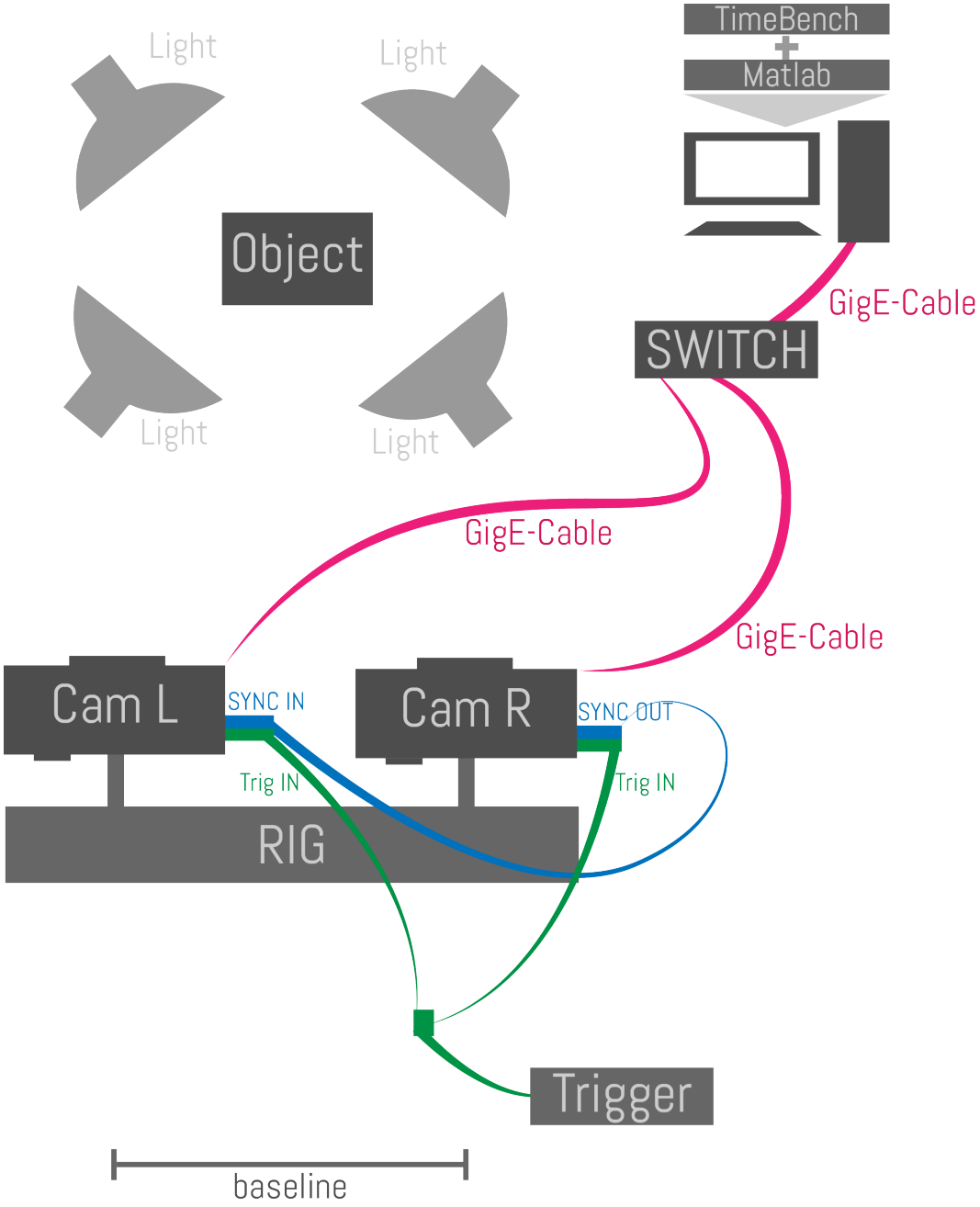


Figure 5.6: Hardware architecture of the final experimental set-up.

6 Experiments

The experiments chapter first describes the final experimental pipeline, which is divided into the Stereo Camera Calibration (section 6.1) and into the actual reconstruction (section 6.2) and secondly follows several experimental sessions to point out problems and results which occurred during the tests.

6.1 Stereo Camera Calibration

The stereo camera calibration estimates the intrinsic and extrinsic parameters of a set-up of two uncalibrated cameras. The calibration is done in two steps, first image pairs of the checkerboard pattern are captured (subsection 6.1.1), then MATLAB's *Stereo Camera Calibrator Toolboox* is used to estimate the parameters (subsection 6.1.2). The camera set-up used in the camera calibration must not be changed at all for the rest of the session: the sequence recorded for the actual 3-D reconstruction needs to have the same parameters.

6.1.1 Capturing Sequence of Calibration Pattern in TimeBench

The two cameras should be set-up next to each other with little to no rotation. For stereo display (anaglyphic display) the cameras need to be placed about 55 mm apart, which is about the distance between our eyes. Since this thesis does not create stereoscopic displays, the cameras' baseline is set to approximately

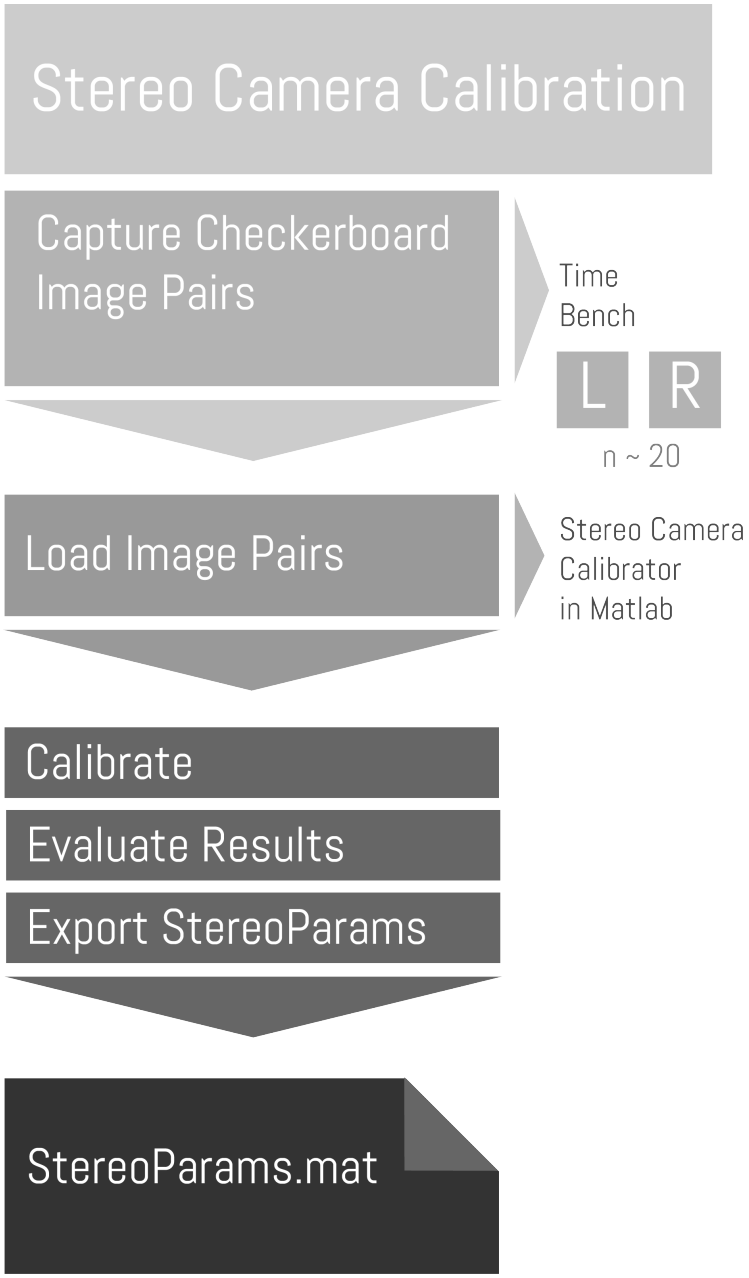


Figure 6.1: Steps of the stereo camera calibration in TimeBench and in the Stereo Camera Calibrator for MATLAB, using n image pairs.

19 cm, which is the minimum distance the two cameras can be apart from each other due to the camera bodies[1] ([Mat16b]).

TimeBench should only be started after the two high-speed cameras have already been connected to the computer, otherwise the cameras may not be recognized by the software. After the hardware was installed (see section 5.3 for the cable set-up), the two cameras need to be dragged into a *synchronization group* and the following configurations are recommended (Figure 6.2 shows a screenshot of the connected and synchronized cameras in Timebench):

i Both cameras: set the frame rate to 70 fps[2].

ii Right camera: check the box "Is Master" to set the right camera as the synchronization master.

iii Both cameras: set the exposure time slightly shorter than 1/frame rate.

iv Right camera: do not forget to white balance.

v Save images as *bmp*.

Once the cameras are synchronized and configured the capturing of the calibration images can begin. The checkerboard pattern should be hold roughly at the same distance the objects will be placed in the actual scene. It needs to be in focus and must be fully visible in the view of both cameras. After starting the capturing sequence by clicking "Start" in the software and also activating the external trigger, the checkerboard pattern needs to be moved in different angles till the sequence stops. Out of the recorded image sequence 10-20 different image pairs have to be chosen (Figure 6.7 shows the timeline

[1] Cameras which are placed farther apart from each other resolve in greater reconstruction accuracy.

[2] The smaller frame rate gives more time for positioning the calibration pattern in different angles, since the data amount is decreased to the lower frame rate.

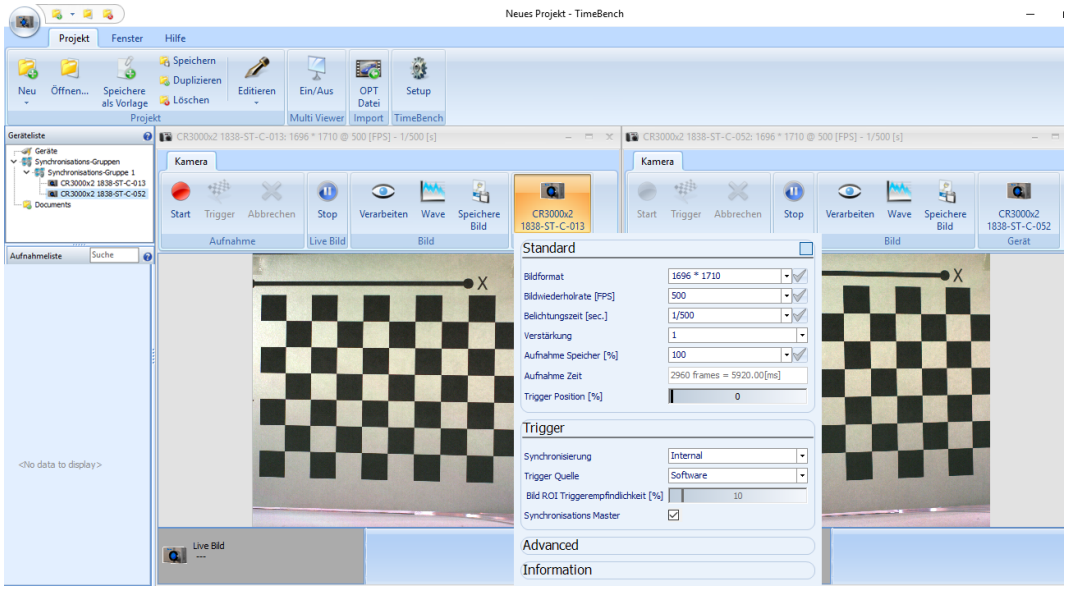


Figure 6.2: The views of two synchronized cameras and their capturing options in TimeBench (*source: software TimeBench owned by* [Opt16]).

with the captured images of a session). Extreme angles are not likely to be detected later in MATLAB ([Mat16b]).

6.1.2 Estimating Stereo Parameters in MATLAB

The selected image pairs are then loaded into the Stereo Camera Calibration Toolbox in MATLAB. They can be added by clicking **Add images** in the app. In the final experimental architecture the left camera is set as camera one and the right camera as camera two. For this step it is important to know which size the squares in the checkerboard pattern are, in this case 3 mm. The images then get analyzed by the software. This process can take some time depending on the amount and resolution of the images in the set. The result window shows how many stereo pairs were added and how many were rejected. The rejection can be caused by several factors, which will be discussed and

summarized in section 6.3. The data browser displays a list of all added image pairs, which should be checked to insure that they are matched with their corresponding image partners. Clicking on a pair of images will display the detected points in a green color and, these points may be checked more closely using the zoom function ([Mat16b]).

After reviewing the accepted data, the button **Calibrate** will start the camera calibration. A diagram shows the overall **reprojection errors** in pixels from all images, which is the difference between the detected and the reprojected points, as shown in Figure 6.3. The error should not be higher than 1 pixel but is recommended to be below 0.5 pixel. Outliers, which means errors much higher than the acceptable value, can be selected by moving the red horizontal line. The respective images can then be deleted in the data browser. After that "Calibrate" needs to be clicked again ([Mat16b]).

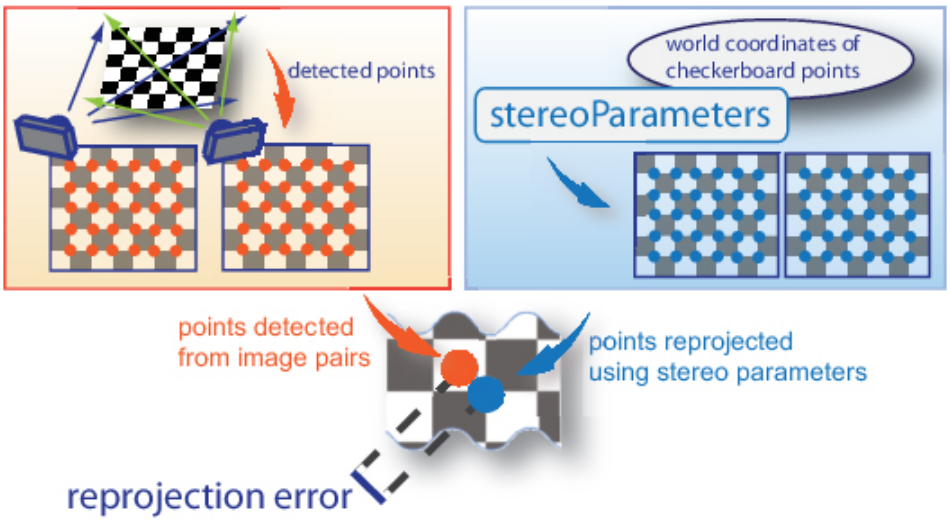


Figure 6.3: Calculation of the reprojection error (*source:* [Mat16b]).

The bottom right window shows the estimated **extrinsic parameters** in a *camera-centric view*. If the pattern was stationary and the cameras were

moved, the plot can be changed to a *pattern-centric view*. This plot should be examined for whether the relative poses of the cameras, the baseline and the checkerboard positions are matching the expected positions.

Another factor to be reviewed is the *rectification*. By clicking on **Show rectified** in the middle of the screen, the rectified image pair should be shown. If the rectified images are still distorted and not row-aligned the calibration was not accurate enough. This problem is further discussed in the session examples and in the summarized results (section 6.3).

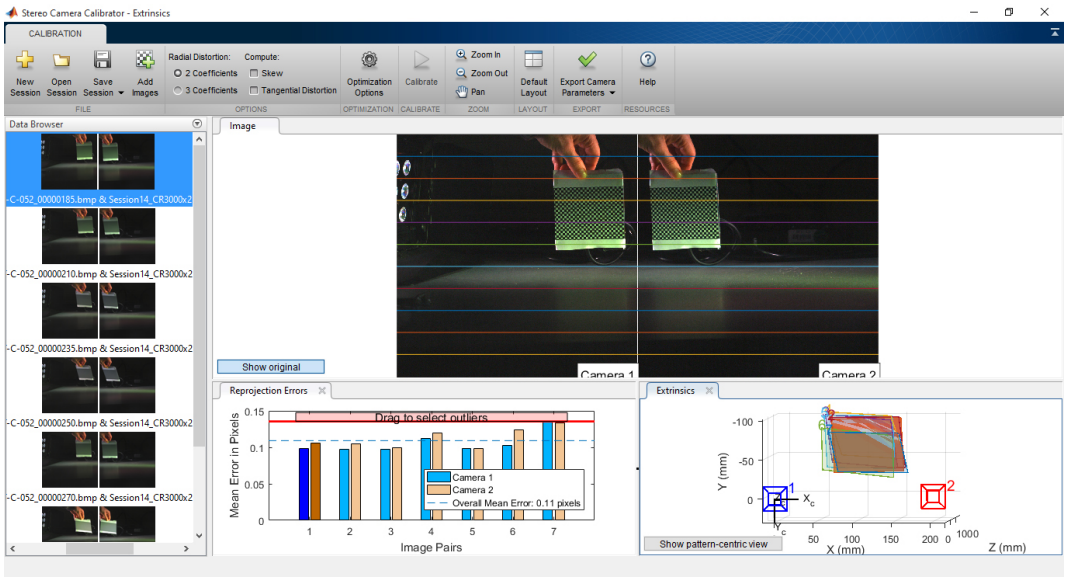


Figure 6.4: Calibrated camera system in the Stereo Calibrator with: **(left)** data browser with set of accepted image pairs; **(middle)** rectified images of one image pair; **(bottom left)** reprojection errors; **(bottom right)** estimated extrinsic parameters.

The last step is to export the calibrated stereo camera parameters. **Export camera parameters** creates a *stereoParameters object* in MATLAB's workspace and **Generate MATLAB script** saves the calibration steps of the session as well.

6.1.3 The stereoParameters Object

The stereoParameters object stores the intrinsic and extrinsic camera parameters as well as the lens distortion ([Mat16b]).

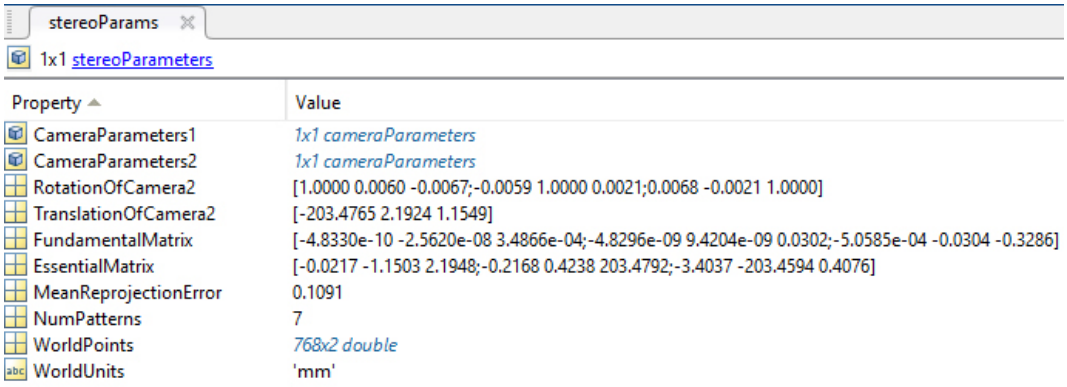


Figure 6.5: The stereoParameter object of session 14 in MATLAB.

Sine the experiments resulted in a rectification problem (see section 6.3), the fundamental matrix computed by the Stereo Camera Calibrator needed to be evaluated. The MATLAB file *testEssentialandFundamentalMatrix.m* addresses this problem.

6.2 Reconstruction

For the 3-D reconstruction of a scene a high-speed stereo sequence is captured with *TimeBench* and this data is then analyzed and computed in MATLAB. The estimated extrinsic and intrinsic camera parameters are retrieved from the camera calibration from section 6.1 with which the cameras' positions are now known in world coordinates. The pipeline of reconstruction is illustrated in Figure 6.6. The corresponding MATLAB *depthEstimationFromStereoImages.m* is included in the digital appendix of this thesis.

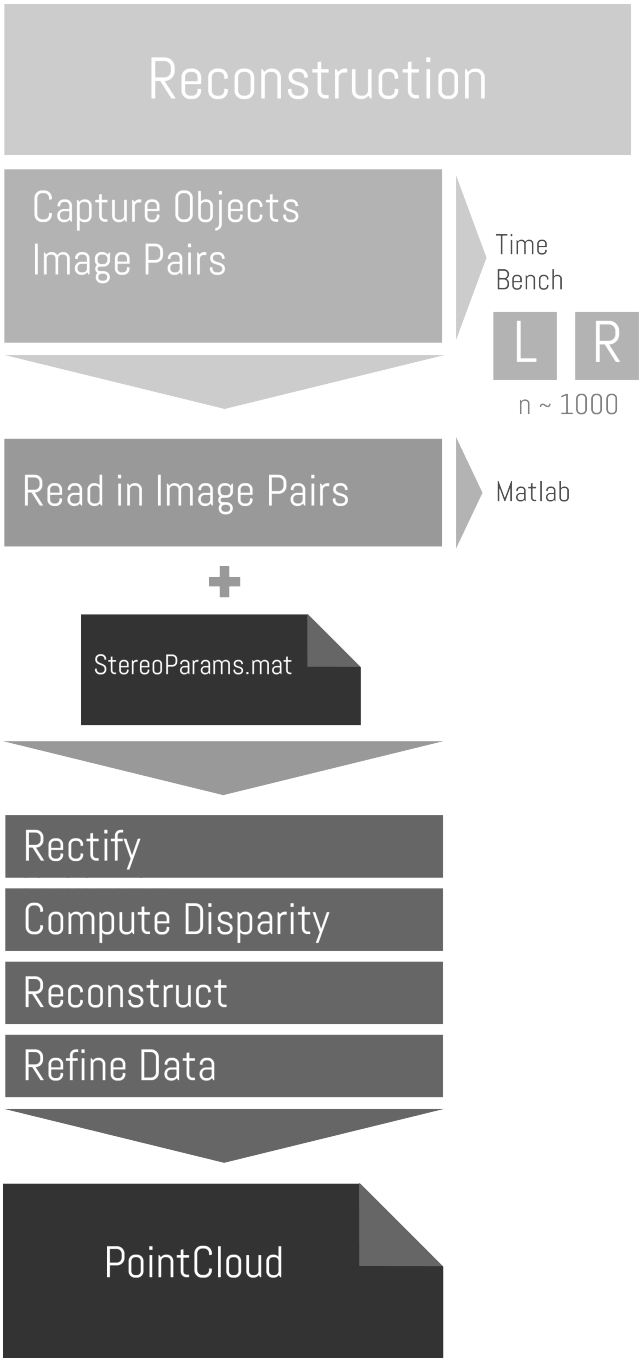


Figure 6.6: Steps of the reconstruction in TimeBench and in MATLAB, using a set of n image pairs.

6.2.1 Capturing a Sequence of moving Objects in TimeBench

The capturing of a sequence of moving objects to be reconstructed follows the procedure as explained in subsection 6.1.1. The cameras must be in the exact same position with the exact same adjustments as in the calibrated scene[3].

The selection of objects which are later to be reconstructed (in the following simply called *objects*) is an important step. The feature detection algorithms need enough different textures and forms in order to detect corresponding points. This topic is discussed in detail in the examples and in the results (section 6.3). The objects need to be brightly illuminated and should not cast too many shadows. They need to be in focus as well. It is recommended that objects be used which are asymmetrical and which have many different textures. Transparent objects should be avoided.

The cameras' frame rate is increased to 500 fps and the sequence is captured in *TimeBench*. Instead of saving only single images, the whole sequence (or the part of interest) is exported (as *bmp*-files). It is of course important to save exactly the same frames of both output data sets. The ring buffer allows a number of approximately 1000 *bmp*-images from each camera to be captured, which results in 2000 images to be processed.

6.2.2 3-D Reconstruction in MATLAB

Loading images and stereo parameters After loading the *stereoParameters*[4] object the image sequences needs to be saved in two variables. For a better

[3]Since it might not be possible to recreate the exact same set-up, the cameras should not be moved at all during the calibration and reconstruction procedure. The only entities which are allowed to be changed in terms of position are the lights.

[4]Stereo parameters can be gained in different ways. In this case they are the output of the Stereo Camera Calibration Toolbox from MATLAB.

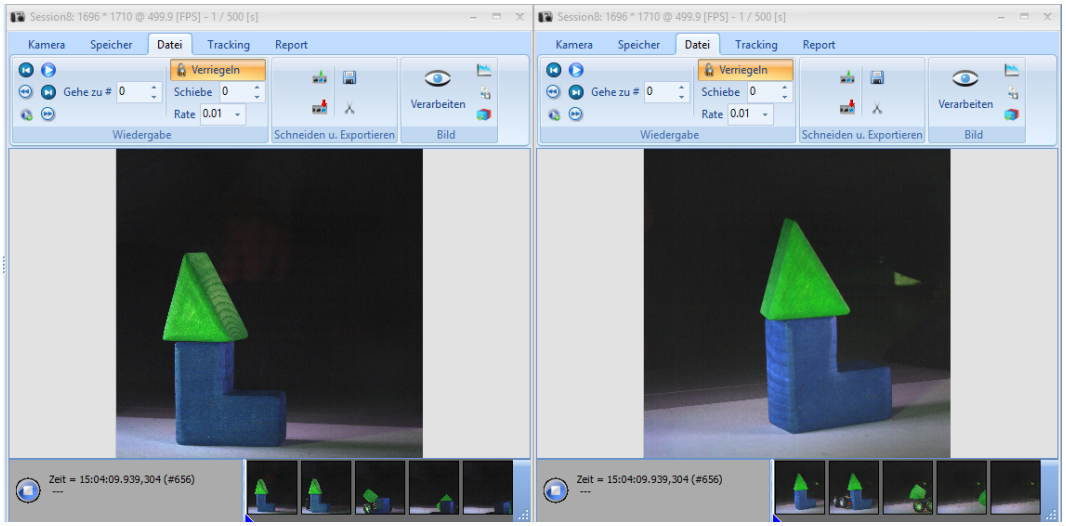


Figure 6.7: Image sequence of a captured scene in TimeBench (*source: software TimeBench owned by* [Opt16]).

overview the following procedures will be explained for only one pair of images, $I1$ (left camera) and $I2$ (right camera). In case of a whole image sequence the steps need to be repeated in a loop for all images.

```
1 load('stereoParams.mat');
2 I1 = imread('D:\left.bmp');
3 I2 = imread('D:\right.bmp');
```

Listing 6.1: Load stereoParams and images.

Rectification The two images must now be rectified (the mathematical background is summarized in subsection 3.2.3). The MATLAB function *rectifyS-*

tereoImages[5] returns the undistorted and rectified images $J1$ and $J2$ from $I1$ and $I2$, taking the estimated stereo parameters into account. If the stereo parameters are not yet estimated (which means the cameras are uncalibrated) the images can still be rectified by first searching *features*, for example with the *detectSURFFeatures* function and then find point correspondeces in the images. The function *estimateUncalibratedRectification* lets one compute the rectification transformations. After this the images can be rectified with the same function[6].

```
1 [J1,J2] = rectifyStereoImages(I1,I2, stereoParams);
2 figure
3 imshowpair(J1, J2, 'montage');
4 title('Rectified Images');
```

Listing 6.2: Image rectification.

Disparity map The disparity map[7] returns the disparities for corresponding pixels of stereo images as an M-by-N 2D grayscale image with the same size as the input data, which are the rectified images $J1$ and $J2$. The *disparityRange* depends on the the cameras' baseline and the distance between the cameras and the objetcs. To get better results the rectified images can be viewed as a *stereo anaglyph* and the distance between two corresponding images can be measured with the *Distance tool*. According to the results the disparity range

[5] The documentation of the MATLAB object can be found at `http://www.mathworks.com/help/vision/ref/rectifystereoimages.html`.

[6] An example for the rectification of images from uncalibrated cameras can be found at: `http://www.mathworks.com/help/vision/examples/uncalibrated-stereo-image-rectification.html`.

[7] The documentation of disparity can be found at `http://www.mathworks.com/help/vision/ref/disparity.html`

can be modified, whereby the bigger the range the bigger the computation time. The numbers of the range must be divisible by 16 with an array range of less than 1629.

```
disparityRange = [0 1600];
disparityMap = disparity(rgb2gray(J1),rgb2gray(J2),'...
    DisparityRange',disparityRange);
figure
imshow(disparityMap, disparityRange);
title('Disparity Map');
colormap jet
colorbar
```

Listing 6.3: Display disparity map.

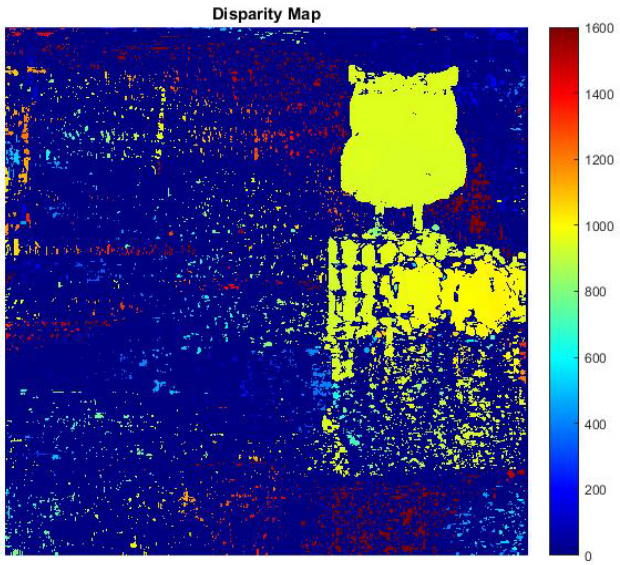


Figure 6.8: The output of the disparity function displays the disparity map.

Reconstruct scene The scene can now be reconstructed with the disparity map. The function *reconstructScene*[8] returns the detected and computed 3-D world point coordinates in a M-by-N-by-3 array - a so called *point cloud*. Due to this fact the output displays absolute values and can be used to measure the real sizes of the reconstructed objects. The point cloud represents the 3-D world points relative to camera 1 in the stereo system computed with the camera calibration. The disparity map's data type determines the type of the point cloud (*single* or *double*). The point clouds of the set of image pairs can be displayed in a point cloud *player* instead of only showing one point cloud.

```
points3D = reconstructScene(disparityMap, stereoParams);
points3D = points3D ./ 1000;
ptCloud = pointCloud(points3D, 'Color', J1);
player3D = pcplayer([-3, 3], [-3, 3], [0, 8], '...
    VerticalAxis', 'y', ...
     'VerticalAxisDir', 'up');
view(player3D, ptCloud);
```

Listing 6.4: Reconstruction of the scene, saved as a point cloud.

6.2.3 Data Refinement

Obviously the scene reconstruction is far from ideal. A noisy or incorrect output can have different causes. If an error in the camera calibration can be excluded, the feature detection is likely to be not accurate enough to get a good result.

[8] See the MATLAB documentation at `http://www.mathworks.com/help/vision/ref/reconstructscene.html`.

Figure 6.9: Zoom into the point cloud of the reconstructed scene.

Note Since most of the filters needs to "run" through the whole array of the point cloud the processes take some time. For one point cloud this might be acceptable, but for the package of point clouds produced by a whole sequence of stereo images it is highly time-consuming.

Filter out distances The first step should be to single out the objects from all noises which were detected around them and which are not interesting for the analysis. For this, two separate MATLAB files were written: the *reducePtCData.m* and the *filterOutPtCDist.m*.

The first file reduces the point cloud data by deleting *NaN*- and *inf*-data values. The output point cloud is unsorted and a lot smaller than the original one.

The second file takes this point cloud and filters out the distances which are defined by a so called *axis-aligned bounding box*. The specialty of this kind of

bounding box is that it can be defined by only two 3-D vectors for the minimum and maximum coordinates which should still be in the scene. With this it can be checked rather easily if a point is inside or outside the axis-aligned bounding box. The point cloud points outside of the box are deleted ([Gre14, p.216 et seq.]).

Denoise MATLAB has a built-in function to decrease the noise of point clouds: *pcdenoise*[9]. It removes *outliers* and was not efficient in the experiments of this thesis.

Bundle adjustment The *Bundle adjustment* in computer vision is a way to minimize re-projection errors. It is only efficient if enough camera views are provided and is therefore often used in *Structure from Motion* algorithms, where a bundle of views from different angles of an object is captured ([Sze11, p.320 et seq.] and [Luh+14, p.322 et seq.]).

Matlab provides a built-in method that refines camera poses and 3D points: the *bundleAdjustment*[10]. At this point it should be mentioned that the code for this method is rather sparsely documented, which could make the usage of the built-in method tedious. However, there are a lot more solutions to choose from provided by the community[11]. The bundle adjustment was not efficient in the experiments.

[9] The documentation can be found at `http://www.mathworks.com/help/vision/ref/pcdenoise.html`.

[10] See a full description of the method at `mathworks.com/help/vision/ref/bundleadjustment.html`.

[11] for example `https://github.com/tikroeger/BA_Matlab`.

6.2.4 Export

The refined cloud object can be analyzed and reused in MATLAB. But to process and rework the 3-D model it is sometimes easier to export the point cloud as a mesh and to import it into another program. MATLAB provides the function *pcwrite*[12] to write a 3-D point cloud into a *PLY*-file. It can be chosen between the *binary* or the *ascii* format, whereas the former saves space.

```
1 pcwrite(ptCloud,'stillOwl','PLYFormat','binary');
```

Listing 6.5: Export of the point cloud into a PLY file with binary format.

The exported point cloud can be for example imported into the software *MeshLab*. MeshLab[13] is open source and is specialized in processing and editing of unstructured 3-D meshes ([CR16]).

6.3 Summarized Experimental Results

This section summarizes the issues and results of the experiments, sorted by topic. In total around 17 different experimental sessions were recorded and analyzed. The earlier tests in which the experimental structure was not yet established are not included. All relevant sessions are briefly documented and included in the appendix. Their calibration patterns and examples of the image sequences are in the digital content. The last section describes the most successful experiment in greater detail (subsection 6.3.1).

[12] The description of the method can be found at `http://mathworks.com/help/vision/ref/pcwrite.html`

[13] The tool was developed with the support of the 3D-CoForm project, which aimed to establish a universal state-of-the-art in 3-D-digitisation and 3-D-documentation. The research reports and other information can be found at `http://www.3d-coform.eu/`.

Time. The calibration takes a long time and the cameras must not be moved at all during the entire process. Only after all necessary calibration images are captured can the success or failure of the calibration process be verified. In many sessions the calibration had to be repeated, sometimes because the cameras were moved by accident and sometimes because of a failed calibration in MATLAB.

Illumination vs. depth of field. The high-speed cameras need a lot of light due to their high frame rate. By keeping the cameras' aperture as wide open as possible one can maximize the efficacy of the lighting, but this results in a very narrow depth of field, causing blurred images and making the calibration process extremely difficult. The checkerboard needs to be moved to different angles in order to get a satisfactory amount of calibration images, but this is hard to achieve when the depth of field is too narrow. Adding more lights results in in reflections on the objects and checkerboards which can also lead to calibration errors.

The early experimental sessions were too dark or the objects out of focus. The best results could be achieved with an aperture of 8, 4 LEDs and a distance of about 1,5 m between the cameras and the objects of interest.

Baseline and camera position. Several camera positions were tested to achieve the best results. In the first sessions (number *1* to *3*) the cameras were set-up on two different camera tripods with a relatively large baseline. The cameras were not parallel to each other. In *session 4* the two cameras were set parallel to each other, but still with a relatively large baseline, since they were still on two separate tripods. The baseline in *session 9* was decreased which resulted in more images pairs being accepted by the camera calibrator but still did not improve the process. From *session 13* onward the cameras were placed on a rig and as parallel to each other as possible. The form of the camera case

and the cables still result in a baseline between 15 to 20 cm.

Hardware problems The cameras disconnected several times after which the TimeBench software and sometimes even the computer had to be restarted which was quite time-consuming. In a few sessions the cameras produced black images.

Problems with the checkerboard. The checkerboard detection is the most important process for the camera calibration. Next to the time-consuming factor, the checkerboard was the cause of many problems. In the early sessions (*1-5*) a big checkerboard in the size of a DIN A4 paper was used. The small depth of field often resulted in checkerboard images that were out of focus, or even out of frame. The lighting was challenging as well: too much and the reflections ruined the images, too little and the images became too dark to be able to be calibrated. By limiting the checkerboard to fewer squares (in *session 2*) and editing the images in *Photoshop*[14], more image pairs were accepted by the Stereo Camera Calibrator. Stereo camera calibration creates another problem: if one checkerboard image is unreadable its corresponding pair image is rendered useless as well, this results in far fewer images pairs being accepted.

Session 6 used a smaller checkerboard, which was easier to handle but was printed at a low quality. The tape used to attach the checkerboard to the paper reflected too much light and the square were not clean enough. The process of capturing the calibration images was improved by recording the calibration sequence in a lower frame rate (50-80 fps) in *session 11* which allowed more time to move the checkerboard pattern to different angles.

Session 12 introduced an even smaller checkerboard which was printed professionally and which had a much smoother surface. This final checkerboard

[14]The contrast was increased and disturbing artifacts were masked.

had the ideal form and improved the calibration process. The square size was now 3 mm.

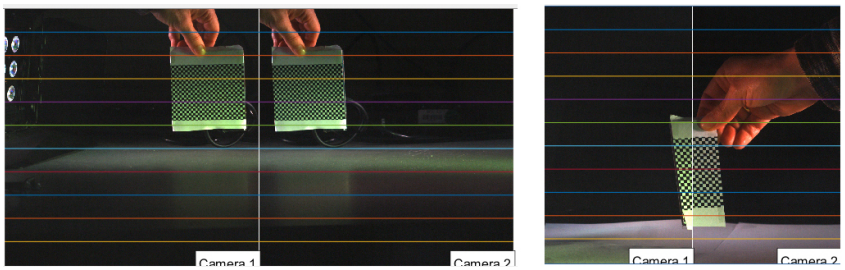
Objects to be reconstructed There were different objects of interest involved in the experiments. In *session 1 to 11* simple toy blocks were pushed over with a toy car. In several sessions different moving objects like a balloon (*session 4*) or a toy magic wand (*session 3*) were captured as well. Since the feature detection algorithms need as many different features as possible, the symmetric toy blocks were first altered with graffiti stickers (*session 12*) and then completely exchanged with asymmetrical objects (*session 13*).

Rectification The biggest problem in the camera calibration process was and still is, the image rectification. The rectified images are either black or do not have the correct size. Changing the cameras' position, adding more image pairs or improving the overall image quality did not change the result. The fundamental matrix computed by the Stereo Camera Calibrator seems to be incorrect. A test with normal *SLR* cameras in *session 13* was successful. In *session 14* the rectification finally worked out fine (a discussion of this session can be found below). After this session it seemed like the error occurred when the cameras were not parallel to each other and when the baseline was too large. A later session (*17*) with almost the same set-up disproved this theory. To this date it is still unclear what causes this error. Several forum posts mention similar problems but the given answers could not be applied to this situation. Solutions were: adding more image pairs and reducing the projection error by deleting outliers. All this is not relevant in these experiments since enough image pairs are accepted and the mean error is way below the recommended maximum.

6.3.1 Session 14

Session 14 is the only session in which the rectification works perfectly with the footage of the high-speed cameras. The cameras were placed on a rig parallel to each other with almost no rotation and a baseline of about 20,7 cm. The cameras were triggered with an external trigger with the option *falling edge*. The checkerboard's squares had the size of 3 mm and the images for calibration were captured with the help of an image sequence at 70 fps.

The actual image sequences were captured at 500 fps. Additionally several still images were taken. Although only 7 out of 19 image pairs were accepted in the Stereo Camera Calibration the rectification was successful. Figure 6.10 shows the comparison between the successful rectification of session 14 and the corrupted one of session 17 which had a very similar camera set-up.



(a) (b)

Figure 6.10: Comparison of the rectification of **session 14** and **session 17**.

The extrinsic camera parameters are computed correctly (see Figure 6.11) and the overall mean error is with 0.11 very low.

The scenes were reconstructed smoothly. The car was the most difficult object for the algorithms to recognize. A screenshot of the reconstruction can be seen in Figure 6.9.

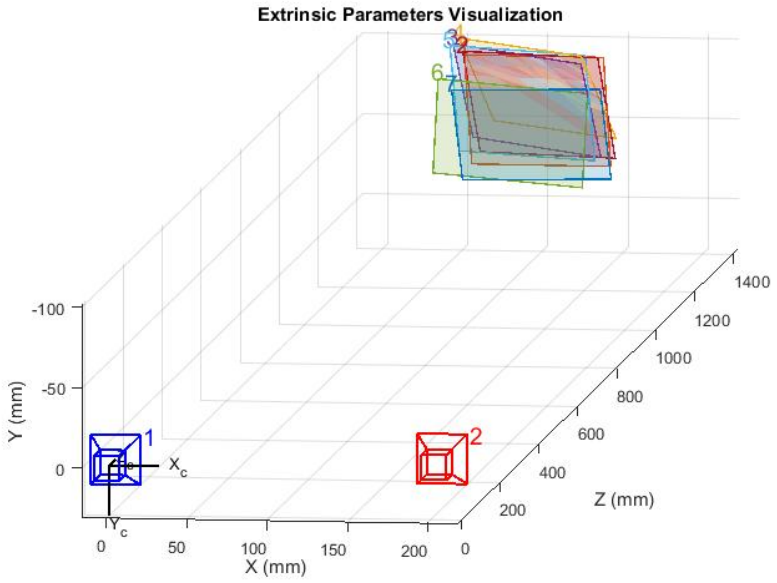


Figure 6.11: Extrinsic parameters of session 14.

7 Closure

7.1 Synopsis

This thesis followed the experimental pipeline in order to reconstruct a scene from stereo high-speed cameras. The mathematical background is complex and MATLAB's built-in functions lack detailed documentation of how exactly output is calculated. Since MATLAB uses its own, special way to compute matrices and to process images, the evaluation of the collected data is difficult to interpret and the cause of problems is hard to find.

Although the stereo correspondence algorithms were successful and led to quite satisfying reconstruction results, a different approach should be tested: the high-speed cameras capture a huge amount of data in a short amount of time. If they are moved around an object there will be many views from different angles of this particular object. This is ideal for another algorithm: the *structure from motion algorithm*. The lighting would still be a problem, maybe even more so due to the change in the cameras position.

The lighting situation should be improved with 4 similar lights, which should emit a consistent light stream so that the scene is evenly lit but not too bright.

The objects should have complex, bright textures with not too many dark spots. The wheels of the toy car are black and did not reflect enough light to be recognized.

The reflections on the white table were another problem. This was addressed

by covering the table with a black cloth. Unfortunately the cloth absorbed too much light and the scene was too dark to be usable. An ideal surface would be neutral gray, matte and non reflective.

The form of the camera case is suboptimal for stereo image capturing since the camera centers are shifted to the left and the cables on the right side of the case add to the distance between the two cameras. This results in a quite a large camera baseline. Stereo anaglyphic views are not possible with this set-up. The cameras could be placed over each other, which is also a common set-up for anaglypic image capturing. This approach was not followed since the field of view changes drastically.

The point clouds can be stitched together to create one big point cloud which contains more information of one object and is more complete[1].

Last but not least the rectification problem needs to be analyzed further with different camera set-ups and other changed variables. A forum post was already created to ask for the communitie's input.

In many ways stereo high-speed cameras make the process of 3-D reconstruction more difficult, but there are many potential improvements and myriad of possible applications in the future.

7.2 Future Work

This section will take a brief look into the future and will also include some suggestions for further reading.

One of the largest fields in which computer vision is applicable is that of robotics. Mobile robots need to "see" their surroundings in order to navigate around and through obstacles, and even immobile robots need to have "vision"

[1] This approach is followed in the example which can be found at `http://www.mathworks.com/help/vision/examples/3-d-point-cloud-registration-and-stitching.html`.

if they are to interact with real world objects. For this the scene can be captured with cameras and then be reconstructed with algorithms such as the ones discussed in this thesis. The next step is to recognize patterns in these computed meshes for which different routines can be programmed which define how the robot has to react or interact in certain situations. Automatic, self-driving vehicles are taking these calculations to the next level. High-speed cameras have the potential to provide a lot of information in a short amount of time for such systems[2].

[2]The paper [Low01] can be used for further readings on this topic.

Index

List of Figures

Bibliography

[Bat09] D. A. Batchelor. *The Science of Star Trek*. 2009. URL: http://www.nasa.gov/topics/technology/features/star_trek.html (visited on 05/17/2016) (cit. on p. 2).

[BC03] G. C. Burdea and P. Coiffet. *Virtual Reality Technology*. 2. ed. Hoboken, NJ, USA: John Wiley & Sons, 2003. ISBN: 0-471-36089-9 (cit. on pp. 2, 8, 14, 15).

[Bou15] J.-Y. Bouguet. *Camera Calibration Toolbox for Matlab*. 2015. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/ (visited on 05/16/2016) (cit. on p. 46).

[CR16] P. Cignoni and G. Ranzuglia. *MeshLab*. 2016. URL: http://meshlab.sourceforge.net/ (visited on 05/03/2016) (cit. on p. 64).

[Dev16] G. Developers. *Google Cardboard Project*. 2016. URL: https://developers.google.com/cardboard/ (visited on 05/16/2016) (cit. on p. 8).

[Dör+13] R. Dörner et al. "Virtual und Augmented Reality (VR / AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität". In: ed. by R. Dörner, W. Broll, P. Grimm, and B. Jung. Berlin, Heidelberg, DE: Springer Berlin Heidelberg, 2013. Chap. Einleitung, pp. 1–31. ISBN: 978-3-642-28903-3. DOI: 10.1007/978-3-642-28903-3_1.

URL: http://dx.doi.org/10.1007/978-3-642-28903-3_1 (cit. on pp. 2, 14, 15).

[Gol15] E. B. Goldstein. *Wahrnehmungspsychologie: der Grundkurs*. 9. ed. Berlin; Heidelberg; DE: Stürtz, [2015]. ISBN: 978-3-642-55073-7 (cit. on pp. 11–13).

[Gre14] J. Gregory. *Game engine architecture*. 2. ed. Boca Raton, FL, USA: CRC Press, 2014. ISBN: 978-1-4665-6001-7 (cit. on pp. 19–22, 63).

[Häf14] S. Häfele. *Konzeption und prototypische Realisierung eines HMD basierten Augmented-Reality-Systems*. Bachelor's thesis, see digital annex. 2014 (cit. on pp. 5, 9).

[Hei60] M. L. Heilig. *Stereoscopic-television apparatus for individual use*. US Patent 2,955,156. 1960. URL: http://www.google.com/patents/US2955156 (visited on 05/16/2016) (cit. on pp. 8, 15).

[HL10] S. Helmer and D. G. Lowe. "Using Stereo for Object Recognition". In: *International Conference on Robotics and Automation (ICRA)*. Anchorage, Alaska, USA, 2010. URL: http://www.cs.ubc.ca/~lowe/papers/10helmer.pdf (cit. on p. 36).

[Hof09] F. Hofmeyer. "Stereoskope HD-Produktion: Grundlagen, Konzepte, Testergebnisse". In: ed. by N. Hottong and D. Lesik. Furtwangen, DE: Faculty of Digital Media Hochschule Furtwangen, 2009. Chap. Tiefenwahrnehmung, pp. 28–55. ISBN: 3-9810384-60 (cit. on p. 11).

[HZ11] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. 2. ed. Cambridge, UK: Cambridge Univ. Press, 2011. ISBN: 978-0-521-54051-3 (cit. on pp. 10, 20–32).

[Lat95] R. Latham. *The Dictionary of Computer Graphics and Virtual Reality*. 2. ed. New York, NY, USA: Springer Berlin Heidelberg, 1995. ISBN: 0-387-94405-2 (cit. on p. 7).

[Low01] D. G. Lowe. "Local feature view clustering for 3D object recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. Kanuai, Hawaii, USA: IEE, 2001. URL: `http://www.cs.ubc.ca/~lowe/papers/cvpr01.pdf` (cit. on p. 73).

[Low16] D. G. Lowe. *The Computer Vision Industry*. 2016. URL: `http://www.cs.ubc.ca/~lowe/vision.html` (visited on 05/16/2016) (cit. on pp. 3, 10, 16, 35).

[Luh+14] T. Luhmann, S. Robson, S. Kyle, and J. Boehm, eds. *Close-range photogrammetry and 3D imaging*. 2. ed. Berlin, DE [et al.]: De Gruyter, 2014. ISBN: 978-3-11-030269-1 (cit. on pp. 27, 30–33, 63).

[LZ99] C. Loop and Z. Zhang. "Computing Rectifying Homographies for Stereo Vision". In: *IEEE Conference on Computer Vision and Pattern Recognition*. Colorado, USA: IEE, 1999. URL: `http://research.microsoft.com/pubs/68542/tr99-21.pdf` (cit. on pp. 31–34).

[Mat16a] MathWorks. *MATLAB*. 2016. URL: `http://www.mathworks.com/products/matlab/` (visited on 05/18/2016) (cit. on p. 43).

[Mat16b] MathWorks. *Stereo Calibration App*. 2016. URL: `http://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html` (visited on 05/19/2016) (cit. on pp. 41, 45, 51–53, 55).

[Nia16] NianticInc. *Ingress Help*. 2016. URL: `https://support.ingress.com/hc/` (visited on 05/18/2016) (cit. on p. 10).

[Ocu16] OculusVR. *Oculus Rift Development Kit 2*. 2016. URL: `https://www.oculus.com/en-us/dk2/` (visited on 05/03/2016) (cit. on p. 8).

[OL04] K. Okuma and D. G. Lowe. "Automatic rectification of long image sequences". In: *Asian Conference on Computer Vision (ACCV)*. Jeju Island, KR, 2004. URL: `http://www.cs.ubc.ca/~lowe/papers/okuma04b.pdf` (cit. on p. 36).

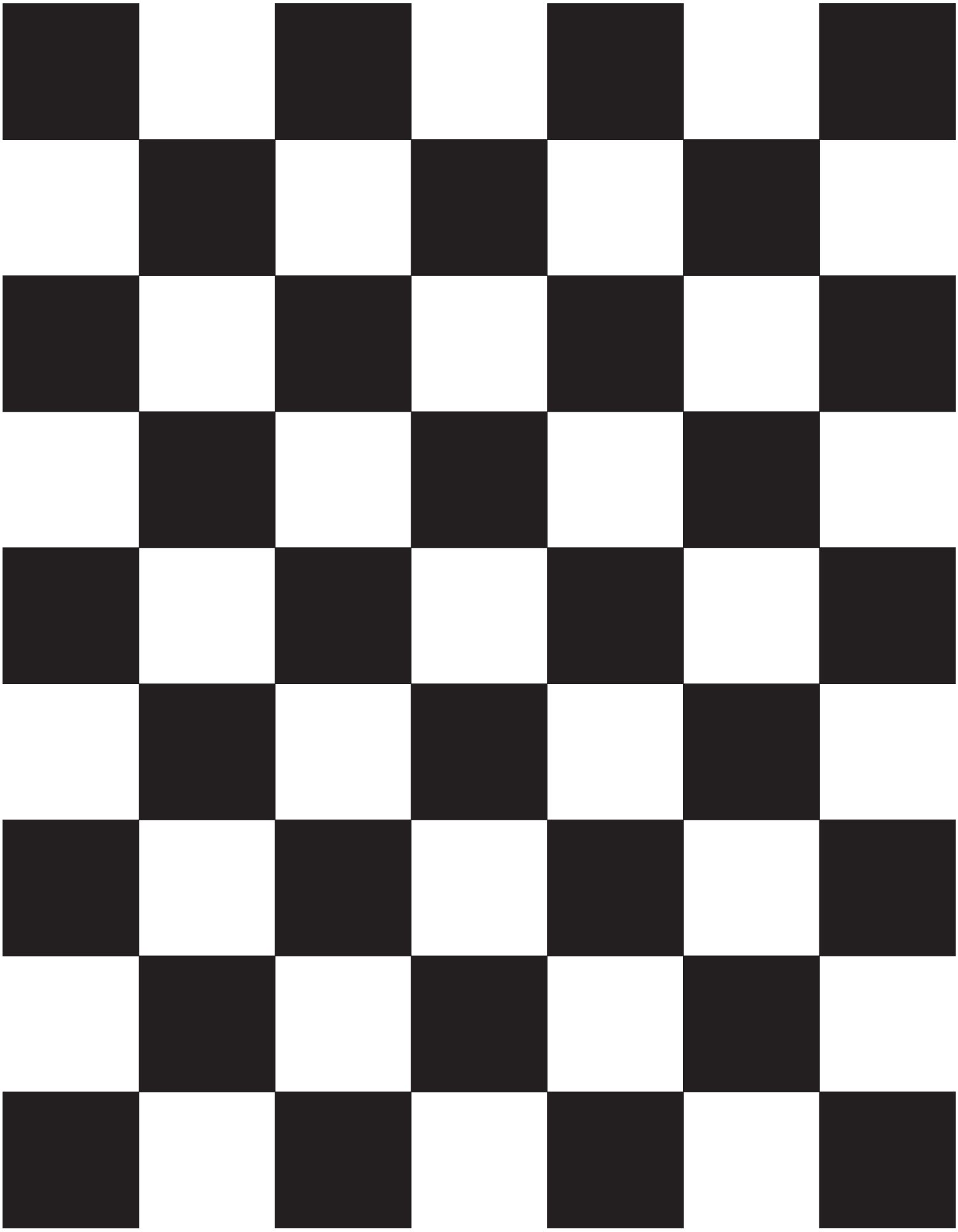
[Opt16] Optronis. *Optronis - Make time visible*. 2016. URL: `http://www.optronis.com` (visited on 05/19/2016) (cit. on pp. 4, 43, 52, 58).

[Qui10] D. Quick. "NVIDIA 3D Vision Pro has professional market in its stereoscopic sights". In: *gizmag* (2010). URL: `http://www.gizmag.com/nvidia-3d-vision-pro/15876/` (visited on 05/17/2016) (cit. on p. 1).

[Sze11] R. Szeliski. *Computer Vision: Algorithms and Applications*. London; Heidelberg [et al.]: Springer Berlin Heidelberg, 2011. ISBN: 978-1-84882-934-3 (cit. on pp. 3, 19–21, 24, 32, 33, 63).

[TL97] V. Tucakov and D. G. Lowe. "Temporally coherent stereo: improving performance through knowledge of motion". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Albuquerque, NM, USA, 1997. URL: `http://www.cs.ubc.ca/~lowe/papers/tucakov97.pdf` (cit. on p. 36).

[Tön10] M. Tönnis. *Informatik im Fokus - Augmented Reality: Einblicke in die Erweiterte Realität*. Berlin, Heidelberg, DE: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-14178-2 (cit. on pp. 2, 8, 14, 15).



[Session #01]

Description

First experimental sessions. Several different seuqences are taken with simple toy blocks and a toy car.

Calibration Stats

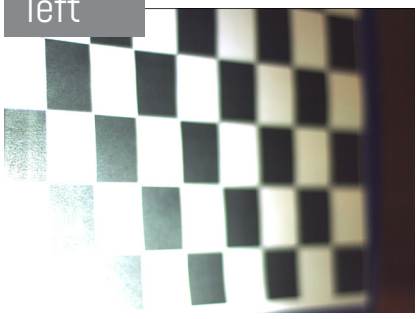
N/A

Results

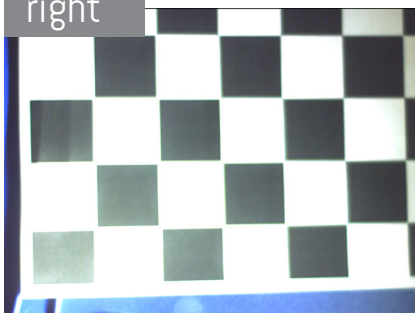
Big problems with the checkerboard: The calibration pattern is not fully visible and does not stay in focus. Reflections all over it. Not enough images. Error with rectification: calibration fails. No reconstruction possible. Scene sometimes too dark, not in focus, different color.

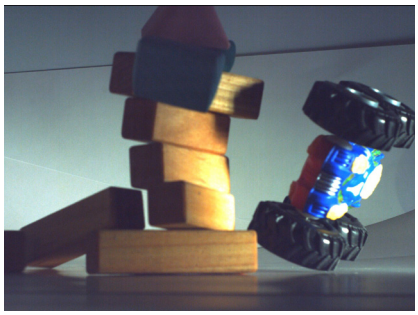
Images

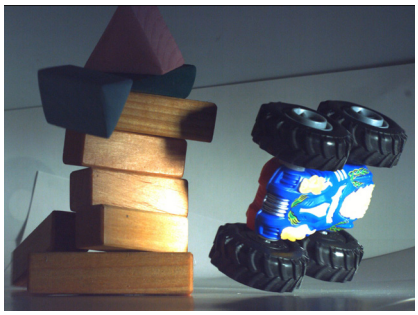
left



right







[Session #02]

Description

Decreased size of checkerboard, since the last session had problems with it. One stereo seuqence is taken with simple toy blocks and a toy car.

Calibration Stats

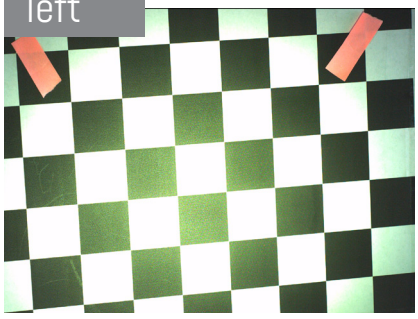
N/A

Results

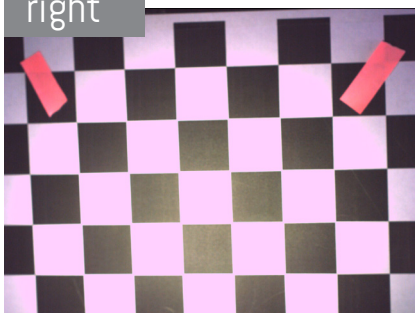
Still sometimes checkerboard not completely visible. Still reflections on checkerboard. Not enough calib. images. Error with rectification: calibration fails. No reconstruction possible. Different coloration of the footage. Not in focus.

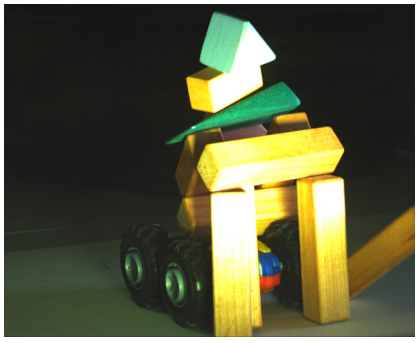
Images

left



right





[Session #03]

Description

Two calibrations needed, since first were rejected in MATLAB due to out of focus checkerboard patterns. Different toy blocks and a folding toy magic wand are used for more details. Checkerboard moved to smoother surface.

Calibration Stats

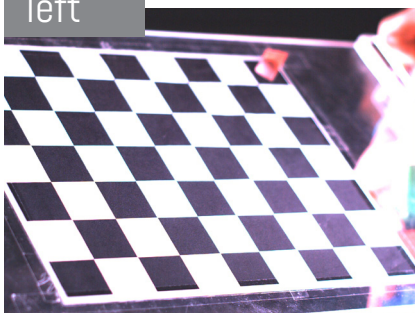
5/8 calibration images accepted, image sequence: 500 fps

Results

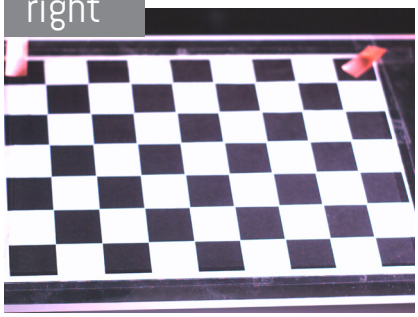
A few camera disconnects in TimeBench. Still checkerboard sometimes not fully visible. Still error with rectification: calibration fails. No reconstruction possible. Different coloration of the footage. Not in focus.

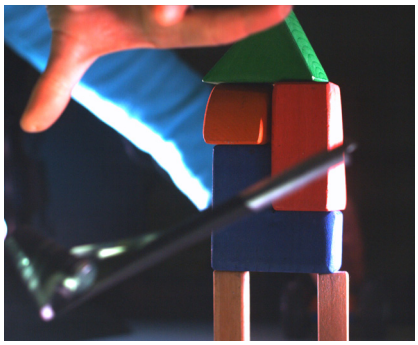
Images

left



right





[Session #04]

Description

Parallel cameras, better printed out checkerboard, calibration with a sequence. Checkerboard images edited in photoshop. Experiments with toy blocks and a balloon.

Calibration Stats

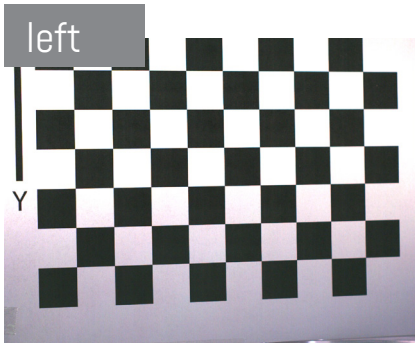
checkerboard sqaure size: 2 cm, 500 fps

Results

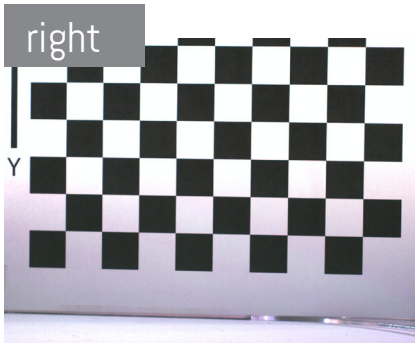
Calibration with sequence gives synchronized and a lot more image pairs! But takes a lot of time to render. Edited checkerboard images are more likely to be added in MATLAB. Still failed rectification. Black images in image sequence.

Images

left



right



[Session #06]

Description

Smaller checkerboard, images edited in photoshop. No image sequence taken.

Calibration Stats

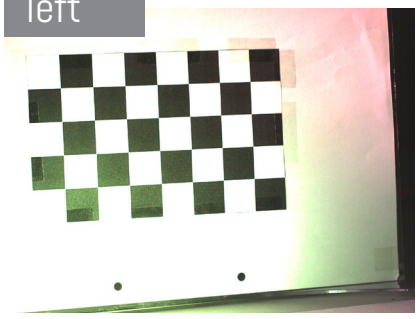
N/A

Results

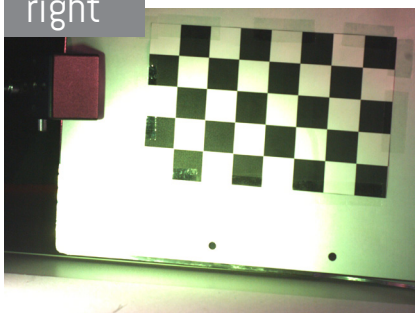
No added images. The checkerboard is not good enough, the printout not clean., too blurry, too many relfections because of the tape.

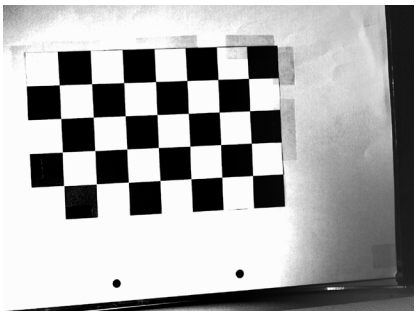
Images

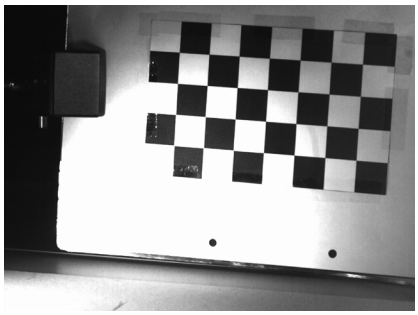
left



right







[Session #08]

Description

Smaller checkerboard which got copied from the last one so the tape does not reflect anymore, image sequence with toy blocks and toy car.

Calibration Stats

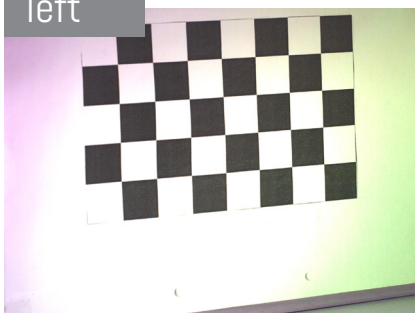
N/A

Results

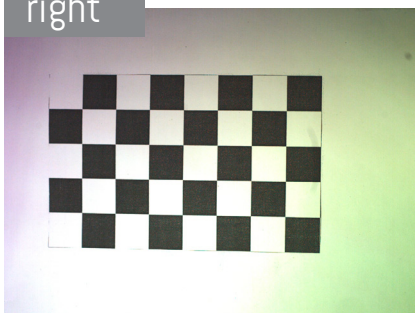
No reflections on the checkerboard anymore, but still too many rejected calibration images due to bad quality of checkerboard.

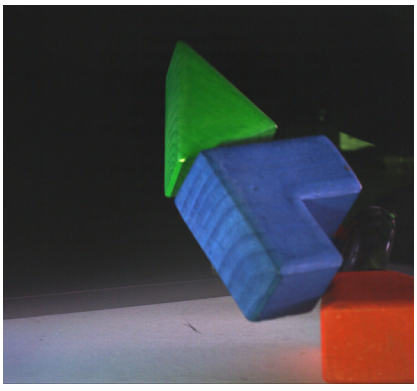
Images

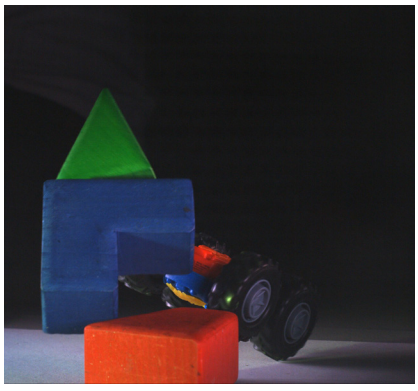
left



right







[Session #09]

Description

Smaller camera baseline

Calibration Stats

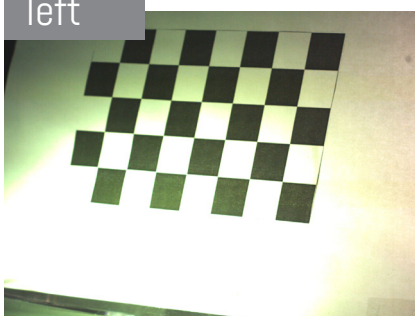
N/A

Results

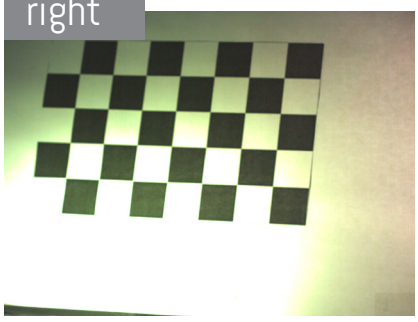
A lot more added pairs (14/17) in MATLAB, but Overall Mean Error: 10.16 px way too high! With outliers deleted still error of 0.96 px; rectification still black!

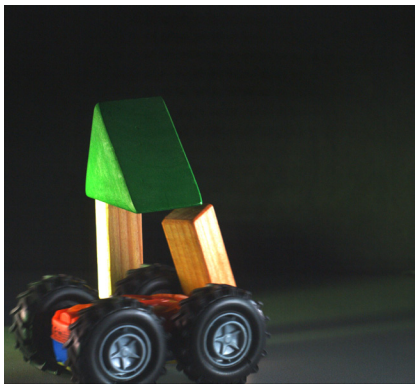
Images

left



right





[Session #10]

Description

Different camera calibrator used in MATLAB.

Calibration Stats

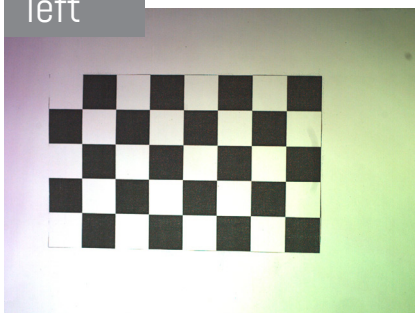
N/A

Results

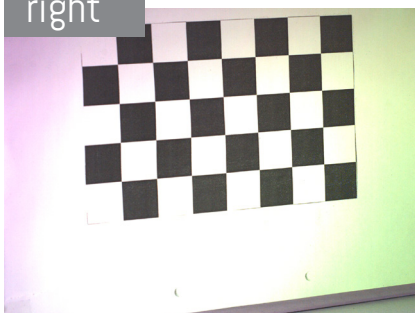
After deleted outliers: Added 13/15 with OME: 0.97 px; Rectified image now not only black but also only one small row. Tried calibration with Camera Calibration Toolbox for Matlab from Jean-Yves Bouguet, still not successful.

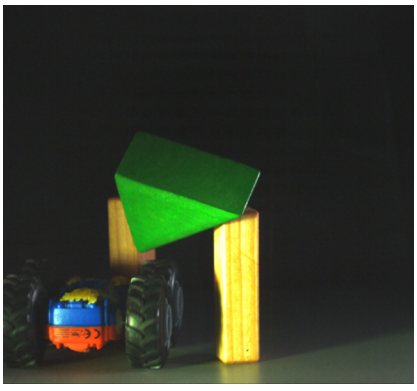
Images

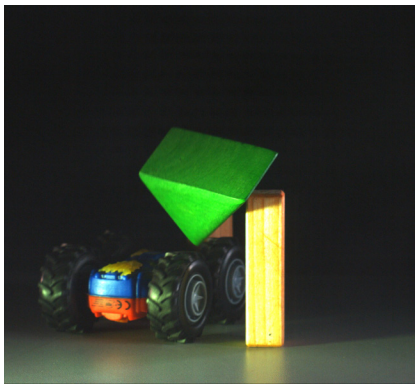
left



right







[Session #11]

Description

Session with Mr. Reinke from Optronis: calibration sequence with lower frame rate, and external trigger, again bigger checkerboard since depth of field was bigger.

Calibration Stats

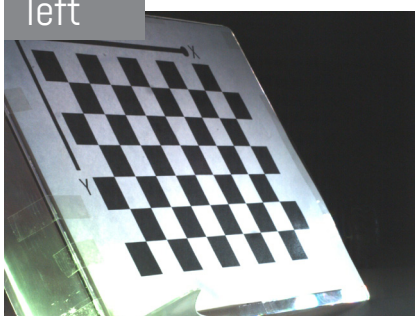
calib sequence: 50 fps, image sequence: 500 fps

Results

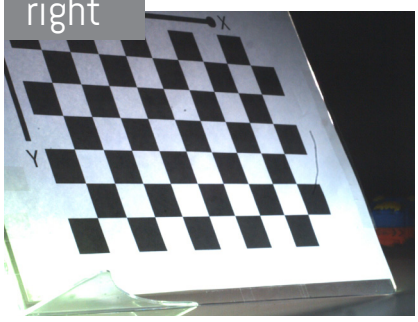
Trying to figure out rectification problem, conclusion/ theory: the essential and fundamental matrix are not computed correctly. Still no successful rectification.

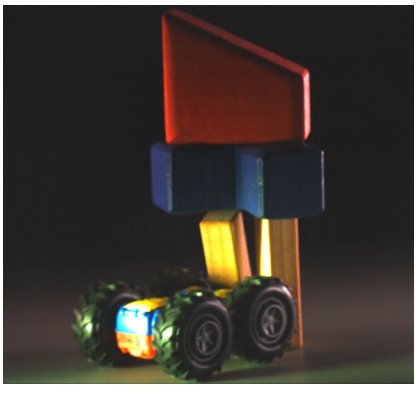
Images

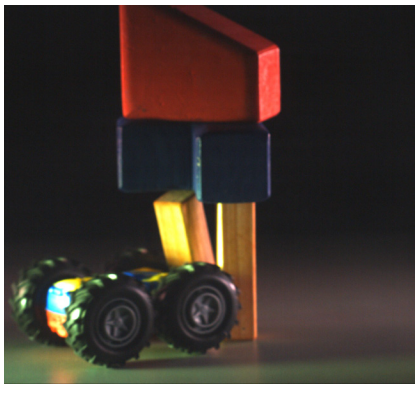
left



right







[Session #12]

Description

Even smaller checkerboard, a lot smoother. Added texture to toy blocks.

Calibration Stats

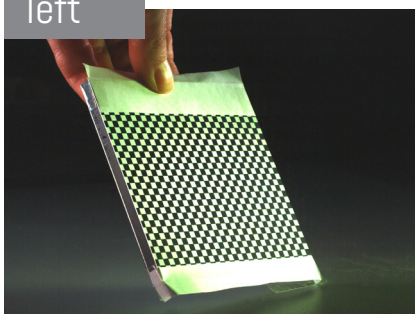
square size: 3 mm, calib sequence: 50 fps, image sequence: 500 fps

Results

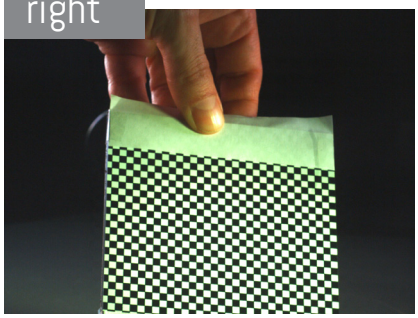
Still only 10/24 added, but smaller checkerboard is a lot easier to handle in this depth of field.

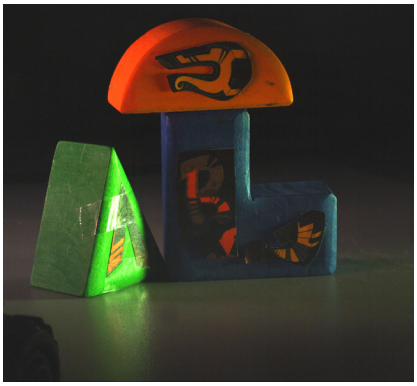
Images

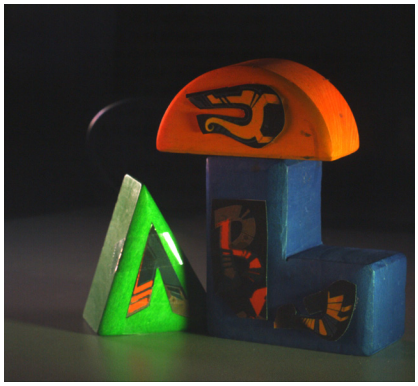
left



right







[Session #13]

Description

Test with normal, parallel SLR cameras on a camera rig, now no toy blocks anymore but asymmetric obects with a lot of texture,

Calibration Stats

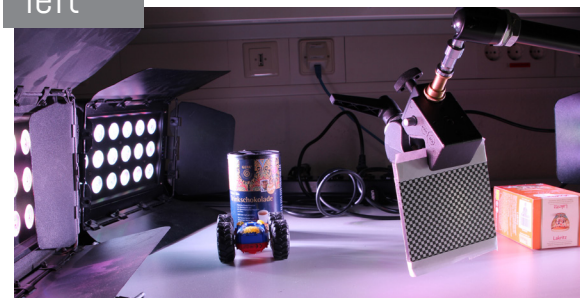
baseline 16,6 cm, square size: 3 mm

Results

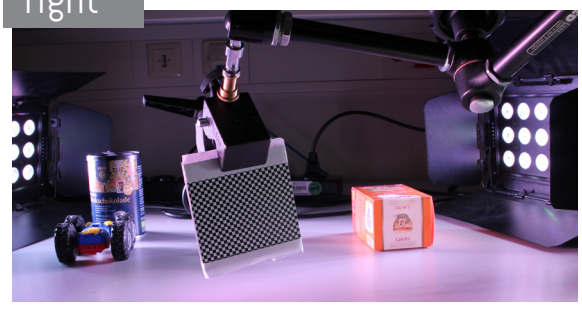
Depth of field is much better; image rendering and processing takes a lot longer time, since the image resolution is much bigger; rectification works

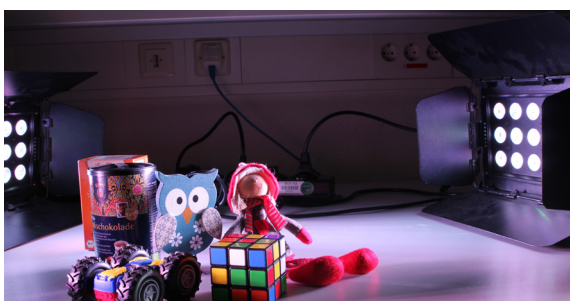
Images

left



right





[Session #14]

Description

High-speed cameras on a rig, parallel (almost no rotation)

Calibration Stats

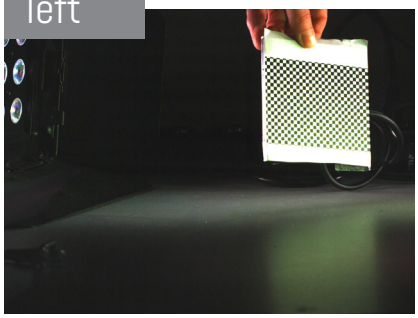
square size: 3 mm, calib sequence: 50 fps, image sequence: 500 fps, baseline: 20,7 cm

Results

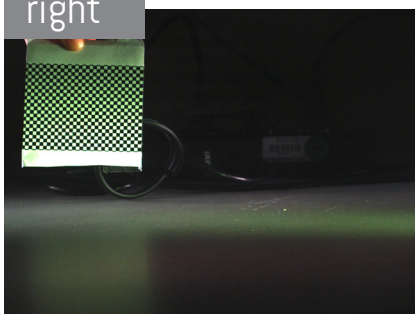
7/19 image pairs added, rectification finally works! Reconstruction of all scenes works! Real world sizes for data evaluation:
Cube: 5,6 x 5,6 cm; Owl: height: 12 cm
Car: width: 10 cm, depth: 8.5cm, height: 5cm

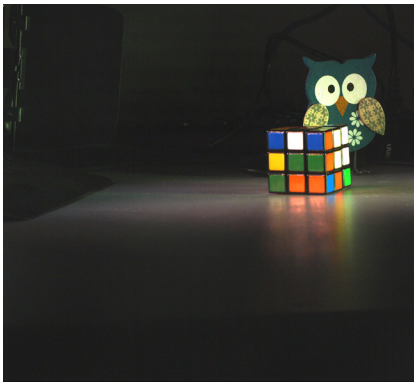
Images

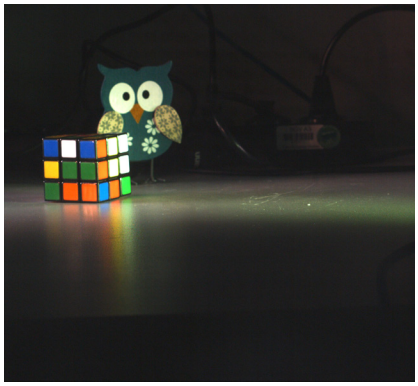
left



right







[Session #15]

Description

High-speed cameras on a rig, small rotation, no image sequence taken since rectification failed

Calibration Stats

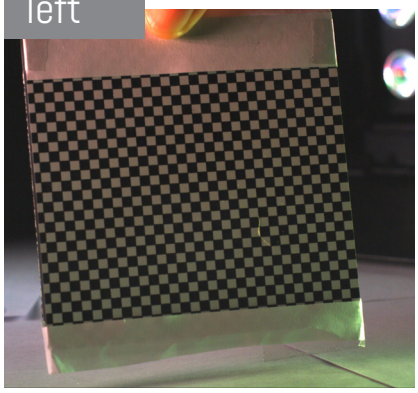
square size: 3 mm, calib sequence: 50 fps, image sequence: 500 fps, baseline: 15,5 cm

Results

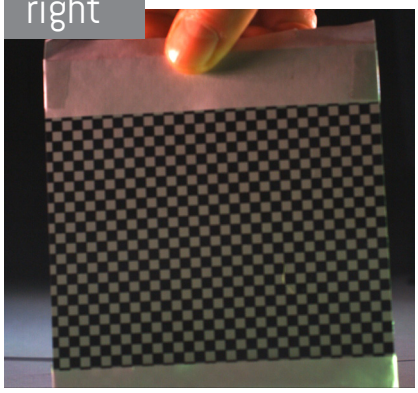
only 6/24 pairs accepted, railed rectification!

Images

left



right



[Session #16]

Description

High-speed cameras on a rig, small rotation, no image sequence taken since rectification failed again.

Calibration Stats

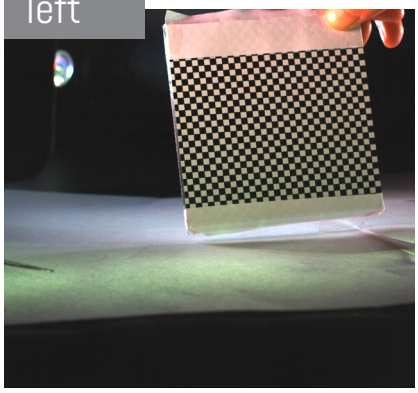
square size: 3 mm, calib sequence: 50 fps, baseline: 15,5 cm due to small rotation

Results

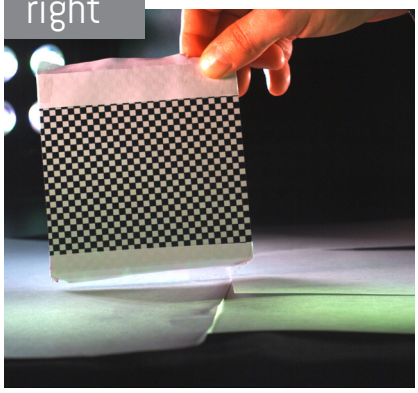
only 6/24 pairs accepted, again failed rectification!

Images

left



right



[Session #17]

Description

Maybe the translation on the y-axis is the problem? The cameras are again almost parallel to avoid that.

Calibration Stats

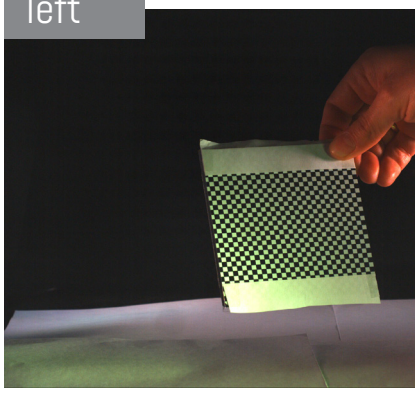
square size: 3 mm, calib sequence: 50 fps, baseline: about 20 cm

Results

19/23 added, but again failed rectification!

Images

left



right