Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Hybrid Tracking Technology for Virtual Rock Climbing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Medieninformatik**

eingereicht von

**Ludwig Steindl, Bsc**
Matrikelnummer 00542071

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Wien, 1. Mai 2018

Ludwig Steindl

Horst Eidenberger

# Hybrid Tracking Technology for Virtual Rock Climbing

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Media Informatics**

by

**Ludwig Steindl, Bsc**
Registration Number 00542071

to the Faculty of Informatics
at the TU Wien

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Vienna, 1$^{st}$ May, 2018

Ludwig Steindl

Horst Eidenberger

# Erklärung zur Verfassung der Arbeit

Ludwig Steindl, Bsc
Johnstraße 61/25
1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Mai 2018

Ludwig Steindl

# Acknowledgements

At this point, I would like to express my deepest gratitude to Prof. Eidenberger. Not only for his outstanding supervision and guidance through this diploma thesis, but for introducing me into the world of machine learning. I want to acknowledge Dipl.Ing. Vonach who provided valuable insights during concept development and performed the 3D printing of the marker enclosures. On a personal note, I want to honor my wife and my mother for their indispensable support throughout the years.

# Kurzfassung

*Virtual Reality (VR)* Anwendungen entkoppeln die Benutzer von der realen Welt, indem sie vollständig in eine künstliche Umgebung eingehüllt werden. Mithilfe einer VR-Brille und schallisolierter Kopfhörer wird die visuelle und auditive Wahrnehmung auf den virtuellen Inhalt der Simulation beschränkt. Die meisten Applikationen unterlassen die Darstellung der Gliedmaßen innerhalb des virtuellen Erlebnisses. Sobald die Anwendung jedoch die Interaktion mit realen Objekten erfordert, muss die Simulation um dieses Merkmal erweitert werden. Ermöglicht wird dies durch ein *Motion Capture* System, welches die Position und Pose der Hände und Füße in Echtzeit bestimmt und an die Applikation übergibt.

Diese Diplomarbeit beschreibt die Entwicklung des VreeTrackers, eines hybriden *Tracking*-Systems, das optische Positionserkennung mit sensorbasierter Lagebestimmung kombiniert. Weiters ermittelt der entwickelte Prototyp die Handpose mithilfe eines markerlosen Ansatzes. Der VreeTracker ist Teil einer virtuellen Kletteranwendung namens VreeClimber. Der VreeClimber ist eine VR-Kletterwand, welche die Sicherheit einer Simulation und die haptische Interaktion mit realen Objekten verknüpft. Als Grundvoraussetzung wird bestimmt, dass alle Hardwarekomponenten günstige und leicht verfügbare Geräte sind. Erforderliche Modifikationen an gekauften Produkten und die eigenständige Neuentwicklung individueller Hardwarekomponenten sind neben der Implementierung der *Tracking*-Software zwei grundlegende Aufgabenbereiche des Projekts.

Der entwickelte Prototyp wird gemäß gängiger räumlicher und zeitlicher Leistungskennzahlen evaluiert. Da optische *Tracking*-Systeme grundsätzlich von Verdeckungsproblemen beeinträchtigt werden können, wird die Robustheit des *Tracking*-Prozesses ebenfalls analysiert. Alle Ergebnisse werden mit der HTC Vive verglichen. Dieses *Virtual Reality* System nutzt ein hochmodernes *Tracking*-Verfahren, welches auf Lasertechnologie basiert. Die markerlose Ermittlung der Handpose wird mittels qualitativer Methoden evaluiert.

Aufgrund der hohen Komplexität des Themas ist die Entwicklung eines hybriden *Motion Capture* Systems im Rahmen einer Diplomarbeit ein ambitioniertes Unterfangen. Dennoch erzielen die Komponenten der Positions- und Lagebestimmung Ergebnisse, die durchwegs mit den Werten der HTC Vive verglichen werden können. Die Ermittlung der Handpose durch Erkennung von Merkmalspunkten gelingt präzise und stabil. Die Berechnung der Werte kann annähernd in Echtzeit durchgeführt werden.

# Abstract

Virtual reality applications detach the user from the real world by fully immersing the person into an artificial environment. With the help of a head-mounted display and soundproof headphones, the visual and auditory senses are limited to the virtual content provided by the simulation. Most applications refrain from representing the user's limbs in the virtual experience. However, if the application requires the participant to interact with real objects, this feature can no longer be omitted. To allow an accurate representation of the extremities within the simulation, the position and pose of the hands and feet need to be determined in real-time by a motion capture system.

This diploma thesis discusses the development of the VreeTracker, a hybrid tracking system that combines optical position tracking with inertial orientation sensing. Furthermore, the resulting prototype estimates the hand pose by a markerless approach. The VreeTracker is embedded in a virtual rock climbing adventure called VreeClimber. The VreeClimber consists of an indoor climbing wall that couples the safety of a virtual simulation and the haptic interaction with real objects. As a prerequisite, all necessary hardware components of the tracking system need to be affordable, easily available off-the-shelf devices. Therefore, in addition to implementing the tracking software, the development process includes all necessary modifications of said consumer products as well as the development of individual hardware components.

The developed prototype is evaluated by common spatial and temporal performance metrics. Since optical tracking technology often suffers from occlusion issues, the tracking robustness is also analyzed. All results are compared to the HTC Vive, a virtual reality system that uses laser-based state-of-the-art tracking technology. The markerless hand pose estimation component is evaluated by qualitative measures.

Considering the complexity of the chosen topic, developing a hybrid motion capture system in the course of a diploma thesis is an ambitious endeavor. Still, the optical position tracking and the inertial orientation sensing components achieve competitive results. The feature point detection algorithm of the hand pose estimation component calculates precise and steady hand pose values almost in real-time.

# Contents

**Kurzfassung** **ix**

**Abstract** **xi**

**Contents** **xiii**

**1 Introduction** **1**
1.1 Motivation . . . . . 1
1.2 Problem Statement & Aim of the Work . . . . . 3
1.3 Methodological Approach . . . . . 3
1.4 Structure of the Work . . . . . 4

**2 State of the Art** **5**
2.1 Fundamental Principles . . . . . 5
  2.1.1 Optical Sensing . . . . . 5
  2.1.2 Inertial Sensing . . . . . 8
2.2 Practical Applications . . . . . 9
  2.2.1 ioTracker . . . . . 9
  2.2.2 Oculus Rift . . . . . 10
  2.2.3 HTC Vive . . . . . 11

**3 Theoretical Background** **15**
3.1 Optical Tracking . . . . . 15
  3.1.1 Light . . . . . 15
  3.1.2 Computer Vision . . . . . 17
3.2 Inertial Tracking . . . . . 23
  3.2.1 Inertial Sensors . . . . . 23
  3.2.2 Sensor Fusion . . . . . 24
  3.2.3 Sensor Calibration . . . . . 25
  3.2.4 Data Representation . . . . . 26
3.3 Digital Image Processing . . . . . 27
  3.3.1 Color Models . . . . . 27
  3.3.2 Image Morphology . . . . . 28

4 **System Design** **31**
4.1 Requirements . . . . . 31
4.1.1 Tracking Scenario . . . . . 31
4.1.2 Quality Characteristics . . . . . 32
4.1.3 Sensing Technologies . . . . . 32
4.1.4 Conclusion . . . . . 34
4.2 System Overview . . . . . 35
4.3 Optical Position Tracking Component . . . . . 36
4.3.1 System Setup . . . . . 36
4.3.2 Hardware Components . . . . . 37
4.3.3 Software Architecture . . . . . 37
4.4 Visual Hand Pose Estimation Component . . . . . 40
4.4.1 Problem Definition . . . . . 40
4.4.2 System Setup . . . . . 40
4.4.3 Software Architecture . . . . . 40
4.5 Inertial Orientation Tracking Component . . . . . 42
4.5.1 Hardware Architecture . . . . . 42
4.5.2 Software Architecture . . . . . 43
4.6 Conclusion . . . . . 43

5 **Implementation** **45**
5.1 Hardware Development . . . . . 45
5.1.1 Camera Rig . . . . . 45
5.1.2 Wearable Tracker . . . . . 50
5.2 Software Development . . . . . 55
5.2.1 Position Tracking . . . . . 55
5.2.2 Hand Pose Estimation . . . . . 61
5.2.3 Orientation Tracking . . . . . 63
5.3 Virtual Environment . . . . . 64
5.4 Conclusion . . . . . 65

6 **Evaluation** **67**
6.1 Setup and Methodology . . . . . 67
6.1.1 Spatial Performance Metrics . . . . . 67
6.1.2 Temporal Performance Metrics . . . . . 68
6.1.3 Tracking Robustness . . . . . 68
6.2 Spatial Performance . . . . . 69
6.2.1 Positional Accuracy . . . . . 69
6.2.2 Positional Precision . . . . . 69
6.2.3 Orientational Precision . . . . . 72
6.3 Temporal Performance . . . . . 74
6.3.1 Update Rate . . . . . 74
6.3.2 Latency . . . . . 74
6.4 Robustness . . . . . 74

6.4.1 Occlusions . . . . . . . . . . 74
6.4.2 Packet Loss . . . . . . . . . . 75
6.5 Hand Pose Estimation Component . . . . . . . . . . 76
6.6 Conclusion . . . . . . . . . . 76

**7 Conclusions and Future Work** **77**

**List of Figures** **79**

**List of Tables** **81**

**Acronyms** **83**

**Bibliography** **85**

CHAPTER 1

# Introduction

## 1.1 Motivation

The commercial launch of affordable *Virtual Reality (VR)* technology in the entertainment sector enabled the general public to experience a highly immersive way of playing video games in their own homes. This allows the player to undergo dangerous adventures in an almost tangible fashion without risking severe injury. *The Climb* [19] illustrates how realistic and accordingly exciting the 360° graphics can be despite standing firmly on the ground and operating the avatar[1] with a handheld controller. However, to fully perceive the thrill of climbing a mountain, a more sophisticated way of user interaction needs to be implemented.

In the late 90s, *Hoffmann* [26] introduced the concept of *passive haptics* and discussed how the interaction with real objects can benefit the virtual experience. Based on this proposition, *Biggs* [9] argues that augmenting the *Virtual Environment (VE)* with physical objects *"can improve the user's sense of presence"* and thereby further increases the immersion. These suggestions are the theoretical foundation for the following project.

The *VreeClimber* is a novel climbing simulator that combines state of the art VR technology with the haptic interaction possibilities of a real climbing wall. In many respects, the provisional test setup displayed in *Figure 1.1a* resembles a traditional indoor climbing wall. Nonetheless, the integration of VR introduces exciting new opportunities that have not been possible before. For instance, the user can enjoy the atmosphere of a beautiful alpine scenery without traveling to a high mountain region.

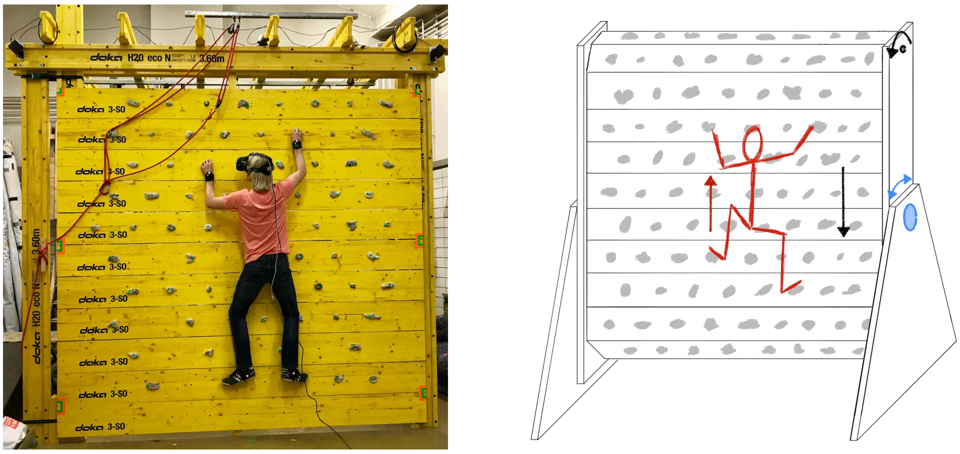
In contrast to similar research projects [36], the VreeClimber attempts to go even further. A rigid climbing wall poses multiple problems. First of all, it needs to be several meters high in order to be reasonably attractive for climbing. This limits the field of application

---

[1] virtual character

tremendously. Secondly, the inclination of the wall is predetermined by its construction which narrows the creative possibilities of the virtual content.

*Figure 1.1b* illustrates the next evolutionary stage of the VreeClimber. It adds the principle of a vertical treadmill where the wall elements move downwards to counteract the person's ascent. Thereby, the climber could continue his endeavor indefinitely while staying only a few feet above the floor. Furthermore, the wall can be pivoted along its horizontal axis which adds an additional *Degree of Freedom (DoF)*. With this new set of features, the possibilities are only limited by the content developer's imagination. The player could witness surreal adventures like climbing the inside of a volcano or ascending the outer walls of a cathedral without facing certain death or criminal prosecution.



(a) Rigid test wall. (b) Rough sketch of revolving wall.

Figure 1.1: Evolutionary stages of the VreeClimber.

One fundamental principle of VR is to fully encapsulate the user in an artificial environment by equipping the person with a *Head-Mounted Display (HMD)* and soundproof headphones. As a result, the visual and auditory perception is limited to the virtual content provided by the simulation. Understandably, when augmenting the virtual experience with the help of physical objects, those need to be accurately re-created in the virtual world. With the help of existing solutions, for instance [51], a high quality replica of the physical climbing wall can be created. It acts as the basis for the virtual content and can be embedded in any conceivable setting.

Furthermore, depicting the user's extremities needs to be considered. For one thing, portraying the limbs can improve the player's feeling of immersion in the simulation. More importantly, when reaching for an object, the eyes guide the movement by providing vital spatial information [44]. In order to intuitively interact with the wall, the user must be presented an accurate representation of the hands and feet. To do so, those need to be tracked in real-time by a motion capture system. This is where this thesis comes into play.

## 1.2 Problem Statement & Aim of the Work

Body Tracking is an immensely popular research area with numerous fields of application, ranging from clinical research [18] to entertainment purposes [64]. As one might expect, several products, partly highly sophisticated systems, are available on the market. However, a thorough research reveals that they are either extremely expensive [63], associated with a particular technology [60] or simply not suitable for a virtual climbing simulator.

The aim of this thesis is to develop an affordable motion capture system called *VreeTracker* that is completely composed of easily replaceable, off-the-shelf components. It should first detect, then track the limbs of an inexperienced climber on a revolving climbing wall. In addition to determining the *three-dimensional (3D)* position and orientation of all four extremities, the poses of the hands need to be estimated. These features can later be used to control the virtual character. Since the player will experience the simulation from a first-person perspective, collecting tracking data of the torso is of little interest. Still, the system should be expandable to meet potential future requirements. Although the precision of the tracking system is of paramount importance, its usability has top priority. Necessary hardware components must not interfere with the user's mobility and should be simple to manage for the operator.

Following the work of *Ribo et al.* [70], five formal requirements are specified:

- *Accuracy:* The tracking system needs to accurately determine the position and orientation of multiple objects. The margin of error should be within a few millimeters.
- *Jitter:* When repeatedly measuring the position of a resting object, the resulting values should be identical. Deviations between individual measurements should not exceed the sub-millimeter range.
- *Robustness:* As long as reasonable movements are performed, the detection of an object should not be disrupted.
- *Mobility:* If tracking technology needs to be attached to the user, it must not hinder the person's range of motion.
- *Prediction:* If the timespan between two measurements exceeds acceptable limits, intermediate values need to be estimated.

## 1.3 Methodological Approach

The methodological approach to implementing the proposed application is composed of two distinct phases. At first, a profound literature review is conducted in order to determine which techniques have been successfully deployed in similar projects. This step also includes an evaluation of commercially available products that could be utilized for the tracking system. Based on the findings of the research phase, a prototype is designed and subsequently implemented. Similar to [21], the development of the motion capture system is *"physically test-driven"*. After realizing a feature, the component is immediately evaluated on the physical test wall with the help of suitable metrics. *Eidenberger and Mossel* [21] describe this approach as *"rapid evolutionary prototyping"*. After finishing the implementation, the quality of the complete tracking system is evaluated.

## 1.4 Structure of the Work

Chapter 2 of this thesis introduces basic principles and gives an overview of the latest state of the art. The main focus lies on tracking systems that relate to current VR technology. Chapter 3 establishes the theoretical foundation of this work. It elaborates the areas of *optical* and *inertial sensor tracking* and imparts necessary background knowledge about *hand detection* techniques. In Chapter 4, the design process is illustrated. After defining the requirements and outlining the motion capture system, deeper insights into the hardware and software components are provided from a design standpoint. Its structure corresponds to the three cornerstones *positional tracking*, *orientation tracking* and *visual hand detection*. Chapter 5 discusses the implementation of the prototype. In Chapter 6, the final prototype is evaluated primarily on quantitative metrics. Finally, Chapter 7 critically reflects on the results, debates the prototype's limits and identifies avenues for future research.

CHAPTER 2

# State of the Art

## 2.1 Fundamental Principles

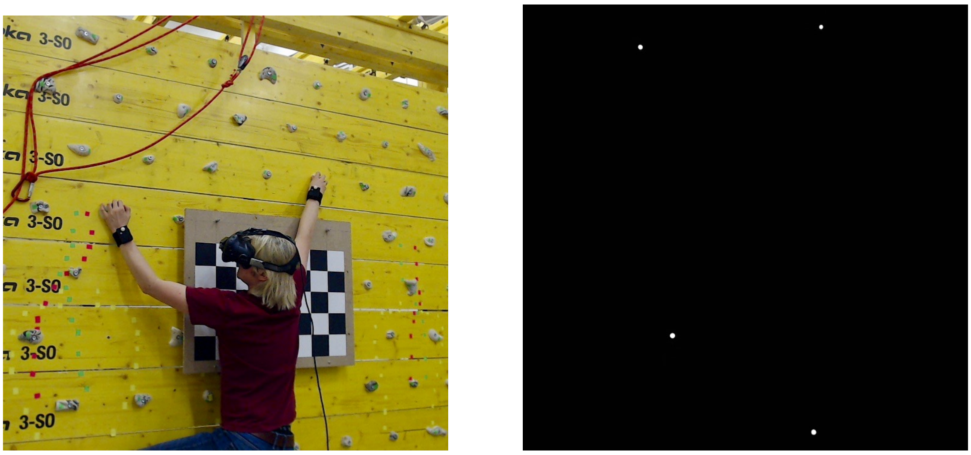
3D tracking systems determine the *position* and *orientation* of an object [12]. Several highly different approaches have been introduced over the years. Each concept presents certain advantages and drawbacks. Selecting the best solution is no trivial task and strongly depends on the concrete use case.

Modern VR applications primarily focus on tracking human motion. Typical scenarios range from rotational head tracking in confined spaces to full skeleton tracking in large areas. Judging from commercially available products [27, 60], *optical* and *inertial sensing* have become the most popular techniques in this territory. Most vendors combine both concepts and form *hybrid sensing methods* [12]. The following chapter introduces basic concepts used by human pose tracking technologies associated with room-sized VR applications. More detailed information on these topics can be found in the corresponding literature [12].

### 2.1.1 Optical Sensing

#### Marker-Based Tracking

The fundamental principle of optical marker-based tracking systems is to detect an artificial landmark, also referred to as a *marker*, with one or more optical sensors [12]. State-of-the-art technologies commonly avoid the visible light spectrum and use the *Near Infrared (NIR)* spectrum instead. This circumvents issues with low or varying lighting conditions. A band pass filter removes the visible bandwidth of the light spectrum before it arrives at the optical sensor. Thereby, it becomes solely sensitive to infrared (IR) light [66]. As shown in *Fig. 2.1*, the sensor identifies the markers as bright spots that can be easily separated from the dark background.



(a) RGB image.

(b) IR image.

Figure 2.1: Comparison of corresponding RGB and IR images *(different camera angles)*.

When placing the landmark on an object of interest, the object itself is indirectly detected. This drastically improves the capabilities of an optical tracking system in comparison to a markerless technique since it renders the computationally expensive recovery of natural features [59] unnecessary.

### Outside-In vs. Inside-Out Tracking

Optical tracking systems usually pursue one of two converse strategies. In an *outside-in* tracking system, the sensors are placed at fixed locations in the room [12]. In a typical setup, they are either mounted on the wall or positioned around the area in which the object moves around. In contrast, an *inside-out* system attaches the sensor to the movable object itself and determines its position with the help of fixed reference points that are situated in the room [12].

### Active Markers

Markers can be classified into two categories. *Active* tracking systems use light emitting landmarks [12]. In many cases, light diffusing spheres are fitted on simple IR LEDs to uniformly emit IR light in every direction. The diffuser is necessary because most IR LEDs only emit a narrow light beam. The spherical shape makes the marker impervious to perspective distortion and therefore optimizes the spatial precision of the tracking system.

Some systems can uniquely identify each marker by controlling its individual high frequency blinking pattern [12, 31]. This can be a major advantage compared to less elaborate systems but implies the use of expensive high speed cameras and complex synchronizing methods. The biggest drawback of active markers is their need for electrical current. Depending on the use case, either every marker is equipped with its own power source or multiple markers are connected to a joint battery pack via cables. While the former is far more comfortable for the user, it can be very time consuming to regularly equip a high number of markers with new (rechargeable) batteries. The latter scenario can significantly limit the user's range of motion.

### Passive Markers

*Passive* markers follow an opposite approach. Instead of evenly emitting light in every direction, they only reflect IR light back towards its origin [66]. Passive tracking setups use external IR lamps that are positioned in close proximity to their corresponding optical sensor [68]. When this unit is directed towards a marker, the sensor registers the reflection of the landmark as a bright spot. Most vendors use small spheres that are coated with a retro-reflective material [67]. In contrast to active markers, these spheres can be built at low cost, are very lightweight, and do not require a power source. Nevertheless, problems occur when optical sensors are facing each other since the sensors register the IR lamps on the opposite side.

### Position Tracking

In optical tracking systems, it is common standard to calculate the 3D position of an object via *Projective Triangulation* [66]. In most cases, the position of a single marker can only be computed if it is detected from at least two different perspectives. However, if multiple markers are placed on a rigid body in a strategic pattern, sometimes called *constellation*, a single perspective can suffice [31].

### Orientation Tracking

To additionally determine an object's spatial orientation, also known as *Pose Estimation* [66], multiple markers need to be arranged in a pre-defined geometric configuration acting as a single *Rigid-Body Target* [67]. The challenge behind this idea is to successfully detect the pre-calibrated targets within the discovered markers. This feature is often called *Model-Fitting* [66]. The complexity of this task increases tremendously with a rising number of targets. By creating different marker arrangements, each target can be uniquely identified by the tracking system.

When considering the target design [67], it is important to avoid similarities between different targets. Otherwise, the targets would not be distinguishable. In addition, self-similarities must be evaded to correctly estimate the target's pose.

### Occlusions

By definition, optical tracking technologies are organized as distributed systems where a clear line-of-sight between the sensors and the landmarks must be maintained. As a result, *occlusions* are a chief concern [12]. In a real-life scenario, markers can easily be obscured by other objects. To keep this common issue under control, some systems are highly scalable and allow the integration of several sensors to facilitate room sized applications where multiple objects can be tracked simultaneously from various angles [66].

### Performance Metrics

Some key characteristics need to be established to assess the quality of an optical tracking system. More detailed information can be gathered in *Burdea and Coiffet's* work on *Virtual Reality Technology* [13].

The most common spatial parameters are *accuracy* and *precision*. Although both terms are often used as synonyms, there must be made a clear distinction. Accuracy describes how close a calculated data point is to its actual value [10]. In other words, it expresses the 'correctness' of a measurement. In contrast, precision indicates if multiple measurements lead to the same result [10]. Thereby, it shows how robust the readings are in fact. Precision is often referred to as *jitter* [13] because a poor precision value lets an object appear trembling.

Two additional characteristics are settled in the time domain. The *update rate* states how many values are generated per second [13]. High update rates are necessary to depict smooth movements of an object. *Latency* is known as the time delay between the occurrence of an event and its detection [13]. According to current believe [35], latency is the primary cause for motion sickness within VR environments. Common terms for this issue are *Simulator Sickness* [62] or *Cyber Sickness* [20].

## 2.1.2 Inertial Sensing

### Sensor-Based Tracking

As the name suggests, inertial tracking systems apply inertial measurement technologies to determine an object's spatial position and orientation. Typically, the sensor data of a 3-axis *accelerometer* and a 3-axis *gyroscope* is combined to form an *Inertial Measurement Unit (IMU)* [12]. The accelerometer measures the *linear acceleration* of an object. The gyroscope contributes its *angular velocity*. In principle, the data of these two sensors is sufficient to calculate the position and orientation of an item. However, accelerometers and gyroscopes only deliver relative changes referring to an arbitrary origin [12]. An *Attitude and Heading Reference System (AHRS)* further integrates a 3-axis magnetometer which provides the necessary heading reference for a full-featured orientation tracking.

At this point, the positional data is still just the relative shift from a random starting point [12]. Without additional information, inertial tracking systems cannot calculate an object's absolute position. In the context of VR, inertial sensing technologies are primarily used for orientation tracking in hybrid sensing solutions [40].

### Data Transmission

Inertial trackers operate as autonomous units. Therefore, the size of the tracking area depends only on the technology used for data transmission [12]. Basic systems simply use cables to send the tracking data to a processing unit. Elaborate systems connect the sensor array directly to a small microcontroller which processes the raw sensor data and transmits the computed tracking information over a wireless network.

### Error Accumulation

Inertial sensors are prone to *error accumulation* due to sensor bias [12]. This characteristic is known as *drifting* [13]. By design, inertial sensors only register angular or directional change. Any measurement inaccuracy, however small, is gradually added up until the discrepancy can no longer be neglected. Accelerometers are especially vulnerable to drifting [12]. This is another reason why inertial sensing is mainly used for orientation tracking. While optical sensing technology can calculate an object's position with millimeter-level accuracy [66], inertial navigation systems often have a margin of error within several hundred meters [12]. Gyroscopes are also susceptible to error accumulation. However, this gyroscopic drift can be compensated by including the magnetometer's sensor data into the calculations [12].

## 2.2 Practical Applications

The following chapter introduces three distinct tracking systems which are deployed in various VR applications. While *Pintaric and Kaufmann* scientifically documented their work on the *ioTracker* [66, 67], the exact procedures of the *Oculus Rift* [31, 61, 32, 1] or the *HTC Vive* [37, 50, 31, 32, 1] are not officially disclosed. Consequently, details on the latter technologies are primarily based on high-value online articles.

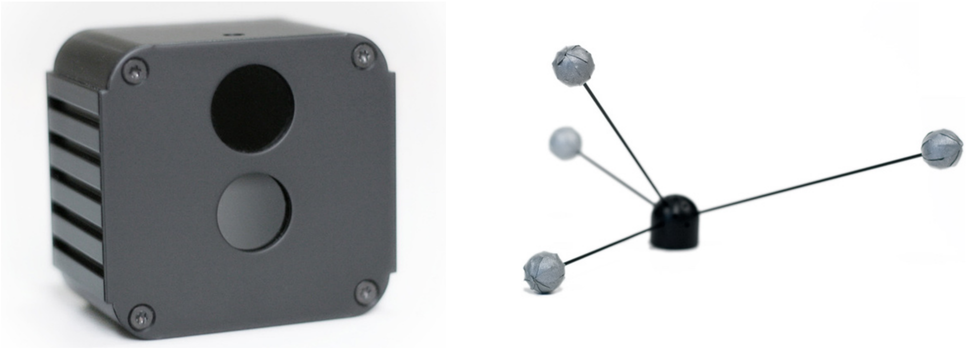
### 2.2.1 ioTracker

#### Sensors

The *ioTracker* [68] is an optical marker-based outside-in tracking system [66]. It consists of 4-8 monochrome IR cameras which are operating in a synchronized framework. Each camera has a video resolution of 640x480 pixels, a field of view of 90 degrees, and delivers 60 frames per second. Aside of the camera itself, each sensor enclosing holds an IR bandpass filter in front of the camera lens and an integrated IR strobe light right next to it *(see Fig. 2.2a)*.

#### Landmarks

The system uses passive retro-reflective spheres as landmarks [67]. The position of each marker can be triangulated if it is detected by at least two cameras. Multiple markers are combined to a single target via carbon-fiber rods *(see Fig. 2.2b)*. These targets can then be mounted on to any object of interest. Using pose estimation algorithms, the system is able to compute the orientation of 12 independent targets within an area of 4m x 4m x 3m [66].



(a) IR sensor.

(b) Rigid-body target.

Figure 2.2: Components of the ioTracker [68].

### Evaluation

The ioTracker is designed as a highly versatile tracking technology. Due to the flexible applicability of the targets, any object or body part can be indirectly detected as long as it has a landmark attached to it. With the capability of installing 8 cameras, the ioTracker handles the fundamental line-of-sight constraint of optical systems very well. Still, target occlusions or even self-occluding targets are key issues. The update rate of 60 frames per second is high enough to guarantee smooth movements. The system's latency of 18ms – 40ms, depending on the number of targets that need to be identified [66], is hardly noticeable in typical applications [40]. The ioTracker achieves excellent results in the spatial properties precision (±0.1mm) and accuracy (±5mm) [66].

## 2.2.2 Oculus Rift

### Head Tracking

The *Rift* [60] is an HMD that was originally developed by *Oculus VR*.[1] HMDs are head-worn displays that resemble diving goggles. Their intention is to fully immerse the user in a virtual environment. In VR games, the player can intuitively change the viewport by turning the head in the appropriate direction. Room-sized applications also adapt the perspective in accordance to the user's movement. The key prerequisite for these features is an accurate tracking of the HMD [12].

---

[1] Since 2014, the company is owned by Facebook [65].

### Sensing Technology

The Rift uses a hybrid sensing technology that combines an optical marker-based outside-in tracking system with inertial sensors [40, 31]. In a so-called *'seated application'*, where the player does not change the position during the game, only the orientation tracking of the head is performed. The Rift utilizes an integrated IMU to determine the head pose without the help of external components [40]. This allows an easy and fast setup in simple environments.

To facilitate room-sized applications, the Rift has several IR LEDs strategically spread across the surface. As shown in *Fig. 2.3*, the LEDs are forming a distinctive pattern called *constellation* [31]. By individually controlling their high frequency blinking patterns, each LED can be clearly identified. Based on the detected formation, the Rift is able to calculate the HMD's position with a single IR camera [31]. Only to improve occlusion issues in 360° setups, the use of three independent cameras is recommended [61].

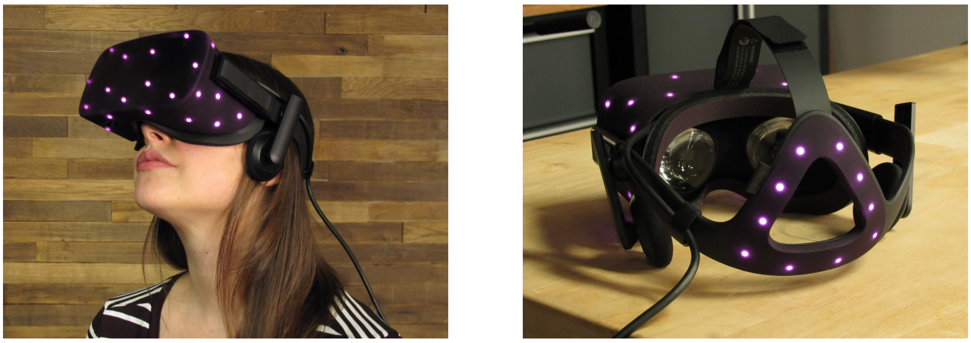

(a) Constellation pattern (front). (b) Constellation pattern (back).

Figure 2.3: Constellation pattern on the Oculus Rift [29].

### Evaluation

Both tracking components are of high quality [38]. The IMU-based orientation tracker has an update rate of 1.000 measurements per second and a latency of only 2ms. The optical position tracker shows an update rate of 60 frames per second. According to their own specification [61], the Oculus Rift can manage a tracking area of 10 feet x 10 feet.

## 2.2.3 HTC Vive

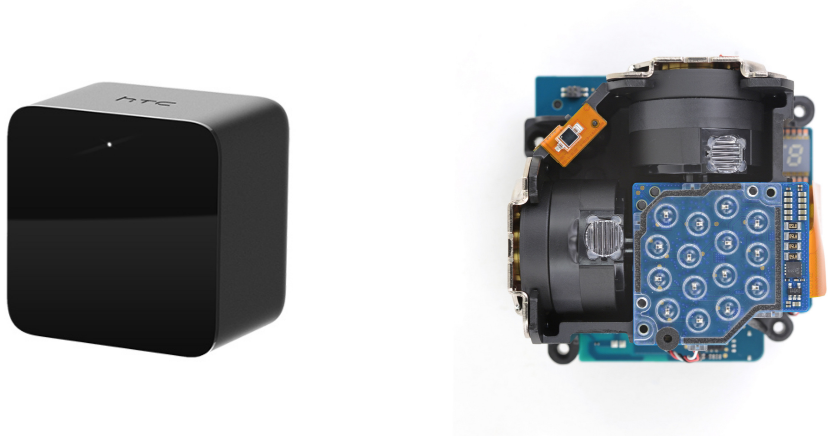
### Overview

The Vive [27] is an HMD that was developed by *HTC* in cooperation with *Valve*. Like the Oculus Rift, it uses a hybrid sensing technology that reconciles inertial sensors with an optical tracking system to determine the position and orientation of the headset [37, 50]. In contrast to the Rift, the HTC Vive incorporates an optical inside-out tracking system [12]. This means that the landmarks are situated outside the tracking area and the sensors are mounted on the HMD. Instead of IR LEDs, the Vive uses synchronized laser sweeps to calculate the relative position and orientation to the landmarks [37, 50].

### Lighthouse

The tracking system utilizes two highly sophisticated landmarks called lighthouses *(see Fig. 2.4)* [37, 50]. Each unit consists of two orthogonally aligned IR lasers and an IR flash. The lasers are used to project a horizontal respectively vertical line into the tracking area. By rotating both lasers around their own axes in an interleaved fashion, the laser lines are alternately sweeping through the room at a constant velocity. Since the tracking system cannot handle two simultaneous laser sweeps, both lighthouses take turns. This procedure is coordinated by a periodic IR flash. The following pattern is repeated 30 times per second:

1. flash – horizontal sweep from lighthouse A
2. flash – vertical sweep from lighthouse A
3. flash – horizontal sweep from lighthouse B
4. flash – vertical sweep from lighthouse B



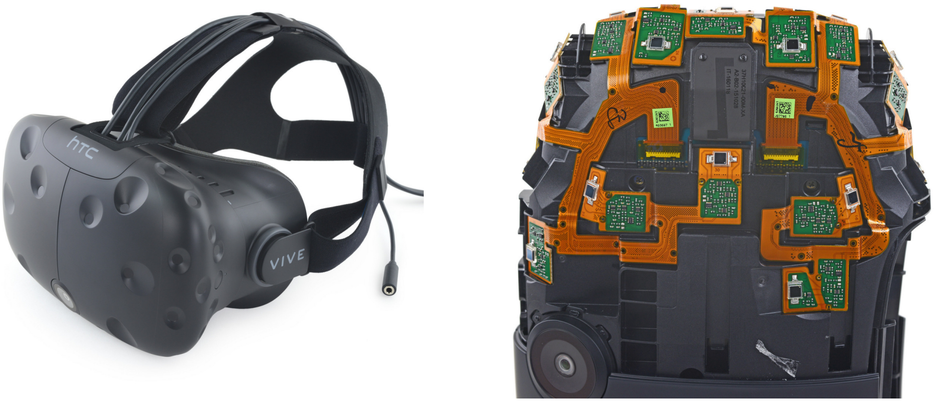
(a) Enclosure [27]. (b) Electronics [28].

Figure 2.4: HTC Vive's Lighthouse.

### Sensing Technology

As shown in *Fig. 2.5*, the surface of the HMD is strategically covered with IR sensitive photodiodes acting as sensors. The system calculates the relative angle to the lighthouses by measuring the time difference between an IR flash and the moment when the laser hits a sensor [1]. The exact position and orientation of the headset can be calculated if at least 5 photodiodes detected the laser beam [50].

The HMD additionally uses inertial sensing data of an integrated IMU to update its position and rotation in-between laser sweeps [37, 50]. This drastically increases the update rate and equally decreases the latency because the IMU delivers over 1.000 new data samples per second [37]. Common drift issues that usually accompany inertial sensor readings do not apply in this case since the inertial data gets corrected by the laser tracking every couple milliseconds.



(a) HMD.

(b) Photodiodes.

Figure 2.5: HTC Vive's HMD [28].

### Evaluation

Due to these innovative techniques, the tracking system shows an accuracy of ±1mm, a precision of ±0,15mm, a latency of 1ms, and an update rate of over 1.000 Hz [37]. Like similar systems, the Vive allows a 360° tracking area of approximately 4m x 4m [1].

CHAPTER 3

# Theoretical Background

## 3.1 Optical Tracking

### 3.1.1 Light

Light represents an essential building block of optical tracking systems since it carries *visual information* in form of electromagnetic waves [33]. Depending on the wavelength, light can be categorized into *ultraviolet (UV) light* (15nm to 380nm), *visible light* (380nm to 780nm), and *infrared (IR) light* (780nm to 1mm) [33].[1] *Fig. 3.1* depicts these areas within the electromagnetic spectrum.

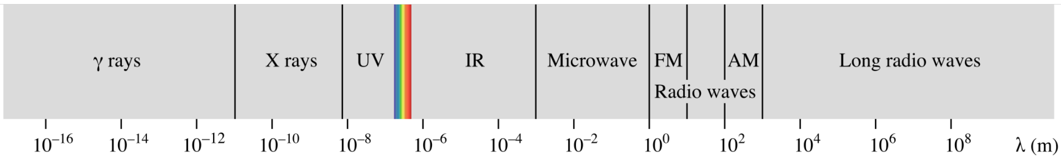


Figure 3.1: The electromagnetic spectrum [24].

#### Near Infrared (NIR) Light Spectrum

The IR band adjacent to the visible spectrum is known as *Near Infrared (NIR)* and ranges from 780nm to 1400nm. In the context of machine vision, this area is of particular interest because common CCD and CMOS camera sensors are sensitive to NIR radiation up to a wavelength of 1100nm [33]. Therefore, ordinary image sensors can be used for NIR applications. However, the simultaneous detection of visible and NIR light is usually undesirable. With the help of optical filters, the range can be limited as required.

---

[1] The limits, which are not consistent among various sources, are based on *DIN 5031*.

### Optical Filters

Three types of optical filters can be distinguished [33]. *Short-wavelength pass filters* pass light with a wavelength shorter than the cut-off value and filter light with a longer wavelength. *Long-wavelength pass filters* pass light with a wavelength longer than the cut-off value and filter light with a shorter wavelength. *Bandpass filters* only pass light with a wavelength around a specified value.

In most cases, one of two filters is used to confine the responsive bandwidth of an image sensor. *IR suppression filters* are bandpass filters that cut out UV and IR light and let visible light through [33]. They are integrated in most RGB cameras to enhance the image quality. *Daylight suppression filters* are long-wavelength pass filters that remove visible light before it arrives at the camera sensor [33]. They are typically deployed in NIR applications.

### Spectral composition of common light sources

As *Fig. 3.2* shows, the spectral composition of common light sources vary significantly [71]. Natural daylight and incandescent lamps have a very broad distribution that reaches far into the IR spectrum. This characteristic has a disruptive influence on certain NIR applications where ambient light jeopardizes the correct detection of IR light emitting or reflecting objects. In this context, fluorescent lights and white LEDs are preferred. They show a much narrower composition that is mostly limited to the visible range and does not interfere with NIR applications.

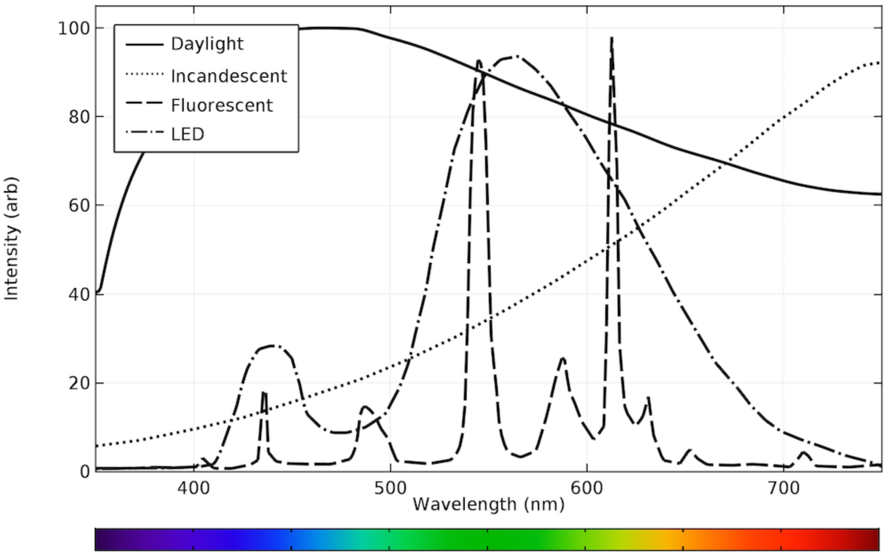


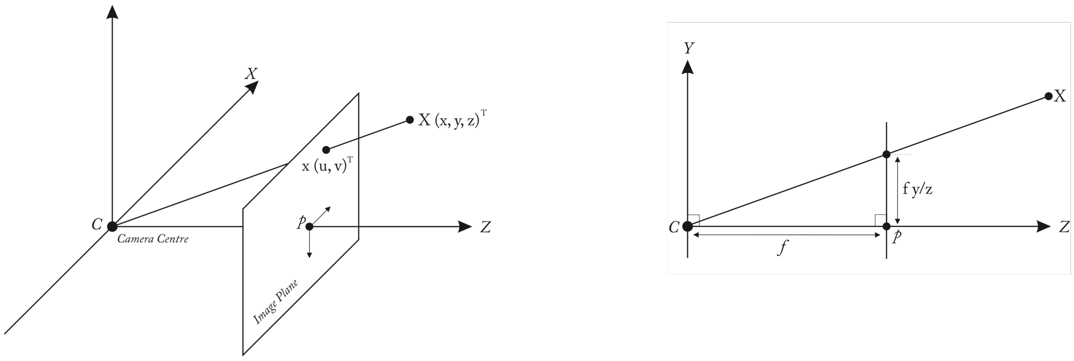
Figure 3.2: Emission spectra of common light sources [71].

### 3.1.2 Computer Vision

#### Pinhole Camera Model

*Camera models* are used to project a section of the 3D world onto a 2D plane [25]. The *pinhole camera model* offers a simple design and is implemented in many computer vision frameworks [52]. It describes the mapping of a point from $3D \mapsto 2D$ space with the help of *central projection* [25].

*Fig. 3.3a* illustrates the the operational pinciple of the pinhole camera model [25]. The *camera center* $C$, also known as the *optical center* [59], corresponds to the origin of the Euclidean coordinate system and acts as the center of projection. The $Z$-axis of the coordinate system is called *principal axis.* The plane perpendicular to the principal axis is named *principal plane.* The *image plane* lies parallel to the principal plane at a distance $Z = f$ from the camera center. The junction $p$ between the image plane and the principal axis is called *principal point.* The 3D point is defined as $X(x, y, z)^T$. The projected 2D point $x(u, v)^T$ lies at the intersection of the image plane and the line segment $\overline{CX}$.



(a) Overview.

(b) Geometric relationships.

Figure 3.3: The pinhole camera model [59].

*Fig. 3.3b* indicates that the point $X(x, y, z)^T$ is projected to $(\frac{fx}{z}, \frac{fy}{z}, f)^T$. Since $(u, v)^T$ is a 2D point, the third coordinate can be omitted. The mathematical expression reads as follows:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{fx}{z} \\ \frac{fy}{z} \end{pmatrix} \tag{3.1}$$

As *Mossel* points out in [59], this *perspective scaling operation* is undesirable and can be circumvented by using *projective geometry* instead of *Euclidean geometry.*[2] Following this conclusion, *Eq. 3.1* can be expressed as a matrix multiplication of *homogenous coordinates* with the *homogenous scaling factor* $\lambda = z$ *(see Eq. 3.2).*

---

[2] Refer to [59] for further information on the mapping between Euclidean and projective spaces.

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.2}$$

The homogenous $3 \times 4$ matrix shown in *Eq. 3.2* is called *camera projection matrix* $P$ [25]. *Eq. 3.3* illustrates that the camera projection matrix can be deconstructed into the *camera calibration matrix* $K$ and the *canonical projection matrix* $P_0$ [14]. The camera calibration matrix holds the *intrinsic camera parameters* [59]. They describe the internal camera properties of the central projection. The canonical projection matrix constitutes the principal form of the *extrinsic camera parameters* which describe the spatial relationship between the camera coordinate system and the world coordinate system.

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P_0} = K \cdot P_0. \tag{3.3}$$

### Intrinsic Camera Parameters

At this point, the camera projection matrix contains only one intrinsic camera parameter called *focal length* $f$. As mentioned in *Ch. 3.1.2*, it is defined as the distance between the image plane and the camera center [59].[3] However, the pinhole camera model considers only the theoretical operating principle of the center projection and neglects common irregularities of real cameras. Consequently, the camera calibration matrix includes additional intrinsic camera parameters [59].

So far, it has been assumed that the principal point lies in the center of the image pane. In reality, many computer vision frameworks set the origin of the coordinate system at the top left corner of the image. The actual position of the principal point is stated by the *offset values* $p_x$ and $p_y$ [59]. Ideally, the pixels of the image sensor are perfectly square, i.e. their sides are of equal length and are perpendicular to each other. Any flaws in these regards can be corrected with the *scale factors* $m_x$ and $m_y$, as well as the *skew coefficient* $s$ [59]. *Eq. 3.4* shows the complete camera calibration matrix.

$$K = \begin{bmatrix} fm_x & s & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

---

[3] In the field of optics, the camera center is called *focal point* [42] and the image plane is known as the *focal plane* [25].

### Extrinsic Camera Parameters

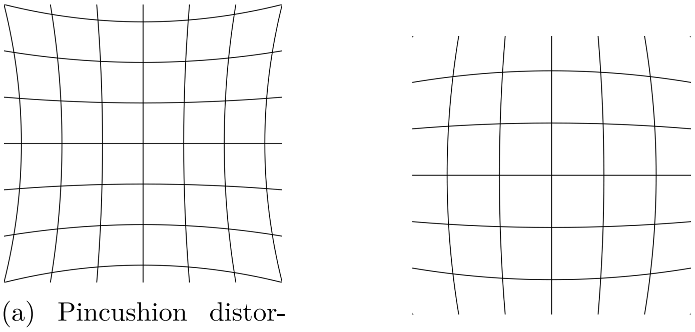
The pinhole camera model assumes that the position and orientation of the *camera coordinate system* corresponds to the *world coordinate system*, i.e. the camera center is aligned with the origin of the world coordinate system. Since this is usually not the case, the *extrinsic camera parameters* specify the transformation between both coordinate systems [14]. They consist of a $3 \times 3$ *rotation matrix* $R$ and a $3 \times 1$ *translation vector* $t$. Both parameters can be merged into a single $4 \times 4$ or $3 \times 4$ matrix that expresses a rotation followed by a translation [14].[4]

Considering the intrinsic and extrinsic parameters, the projection $X(x,y,z)^T \mapsto x(u,v)^T$ can now be mathematically described as follows [14]:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} fm_x & s & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \cdot \underbrace{\left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array}\right]}_{(R|t)} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.5}$$

### Lens Distortion

*Lens distortion* is another common issue of optical systems that the pinhole camera model does not account for. It can be distinguished between *radial distortion* and *tangential distortion* [59]. Radial distortion occurs when entering rays of light are refracted unevenly across the lens. In consequence, straight lines are either warped inwards or outwards. The former defect is called *pincushion distortion*, the latter deformity is named *barrel distortion (see Fig. 3.4).* Tangential distortion happens when the image plane and the camera lens are not parallel to one another [59]. The correct mapping between the distorted coordinates $(u', v')^T$ and their corresponding image coordinates $(u, v)^T$ is a non-trivial mathematical problem that can only be approximated [59].



(a) Pincushion distortion.

(b) Barrel distortion.

Figure 3.4: Types of radial lens distortion [59]

---

[4] The mathematical derivation is explained in [14].

### Camera Calibration

*Camera calibration* is the process of calculating the intrinsic and extrinsic camera parameters as well as estimating the radial and tangential lens distortion [59]. Various approaches meeting different requirements have been developed. Most of them derive the *geometric relationship* between the 3D world and the 2D image with the help of a predetermined calibration pattern.

Zhang [81] implemented a highly popular calibration algorithm that is used in many computer vision frameworks [52]. It suggests the use of a planar pattern like the checkerboard whose geometric characteristics are known. After capturing this *reference target* from multiple angles, its *feature points*, i.e. the intersections between black an white areas, are detected *(see Fig. 3.5)*. Based on these, the camera parameters can be calculated and stored in a projection matrix. In his original paper [81], Zhang summarizes his work as follows:

1. Print a known calibration pattern on a planar surface.
2. Capture multiple images of the planar pattern from various angles.
3. Detect the pattern's feature points in the images.
4. Estimate the intrinsic and extrinsic parameters.
5. Model the lens distortion.
6. Refine all parameters.

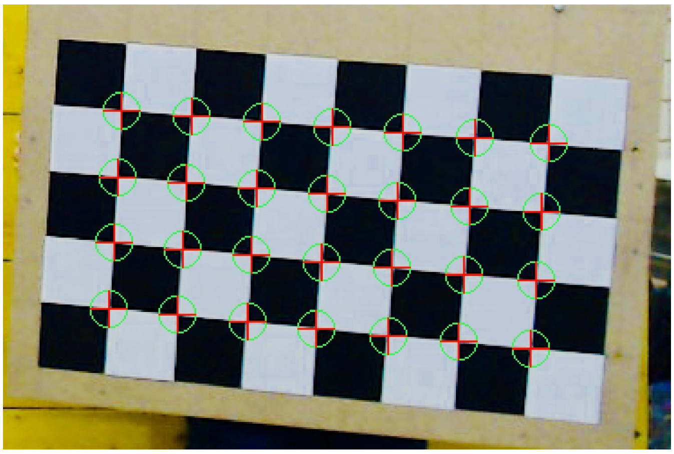
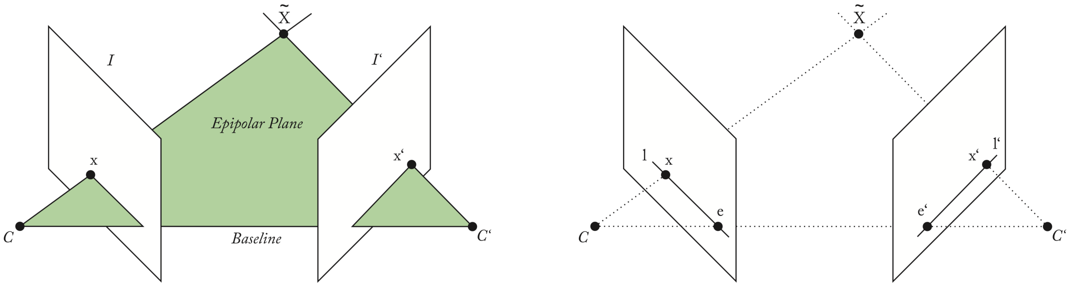


Figure 3.5: Calibration pattern showing detected feature points.

### Epipolar Geometry

Calculating the 3D position of an object from two corresponding camera views is a fundamental requirement for any stereo vision application. The underlying model for this *projective triangulation* is based on *epipolar geometry* [59].

*Fig. 3.6* illustrates the operating principle of epipolar geometry [59]. The camera centers $C$ and $C'$, and the 3D point $\tilde{X}$ span the *epipolar plane*. Following the pinhole camera model, the 2D points $x$ and $x'$ are the projections of the point $\tilde{X}$ on the image planes $I$ and $I'$. The line segment between both optical centers is called *baseline*. The points of intersection between the baseline and the image planes are called epipoles $e$ respectively $e'$. They represent the projection of the opposite camera center on the respective image plane. The epipolar lines $l$ and $l'$ are formed by the intersections of the epipolar plane and the image planes. They connect the image points with their relating epipoles.



(a) Epipolar plane.

(b) Epipoles and epipolar lines.

Figure 3.6: The epipolar geometry [59].

The *stereo correspondence problem* describes the task of correctly matching the corresponding image points $x$ and $x'$ [59]. This can be highly challenging when many points need to be allocated in real time. However, the *epipolar constraint* states that for any point $x$ on the image plane $I$, the corresponding point $x'$ must lie on the epipolar line $l'$. This fact drastically simplifies the correspondence problem since the search for the correct image point can be restricted to candidates along the appropriate epipolar line.[5]

After solving the stereo correspondence problem and performing the camera calibration, epipolar geometry can be used to triangulate the 3D position of the point $\tilde{X}$ in world coordinates. The *triangulation problem* states that the two rays $\overrightarrow{Cx}$ and $\overrightarrow{C'x'}$ intersect at $\tilde{X}$ if $x$ and $x'$ fulfill the epipolar constraint [59]. However, small inaccuracies that are not adequately covered by the camera calibration lead to a slight deviation between the actual 3D point and its calculated counterpart. This can be shown if the calculated point is re-projected onto the image planes. The distance between the original projection and the re-projection on the image plane is called *re-projection error* [25].

---

[5] Since epipolar geometry cannot distinguish between multiple points that lie on the same epipolar line, it cannot fully solve the stereo correspondence problem [59].

### Predictive Filtering

When the triangulation process is repeated periodically, the position of $\tilde{X}$ can be tracked over time. In an ideal scenario, where the calculated position of $\tilde{X}$ matches its real position perfectly and the intervals between two measurements are virtually zero, a continuous movement of $\tilde{X}$ can be mapped precisely. As one might expect, this is usually not the case in practical applications. As mentioned before, optical triangulation contains slight inaccuracies which result in some amount of jitter. Secondly, since optical readings are always subject to a certain degree of digital image processing, the time intervals between data points can never be zero. If the update rate exceeds a reasonable threshold, the application exhibits a noticeable latency and fast movements cannot be depicted smoothly. These issues can be handled with the help of *predictive filters* [59].

Predictive filters like the *Kalman filter*[6] [34] estimate the value of a signal that cannot be measured or is corrupted by noise [54]. The Kalman filter's operating principle is based on a simple feedback loop that consists of two stages [11]. In the *Time Update* stage, the filter predicts a future value of the signal based on past measurements. In the *Measurement Update* stage, the filter improves this prediction by including a new measurement. In other words, the algorithm uses its knowledge about prior measurements to predict a future value and consequently adapts its estimation with every new measurement.

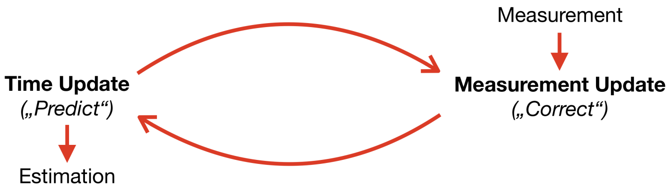
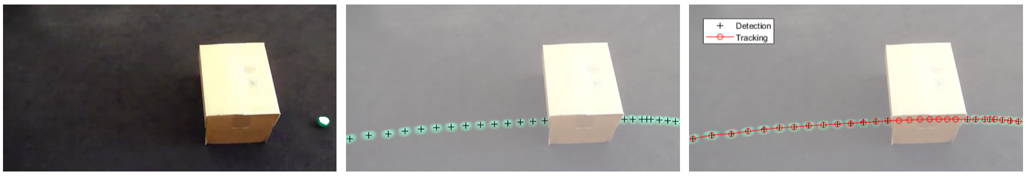


Figure 3.7: The Kalman filter's feedback loop (based on [11, 80]).

As *Fig. 3.8* illustrates, the Kalman filter can also be applied to handle object occlusions. In this scenario, the trajectory of a rolling ball is successfully recovered by the Kalman filter despite being momentarily concealed by another object.



(a) Tracking scenario. (b) Raw tracking data. (c) Filtered tracking data.

Figure 3.8: Occlusion handling using the Kalman filter [55].

---

[6] The Kalman filter is a versatile algorithm that has been successfully implemented in many safety-critical applications for almost 60 years [43]. For instance, NASA used it in the 1960s to estimate the trajectories of spacecrafts during the Apollo missions [22]. Since then, the Kalman filter has lost none of its importance. Today, it is applied in many ubiquitous technologies including radar and GPS [53].

## 3.2 Inertial Tracking

*Inertial tracking systems* are based on the *principle of inertia*, also known as *Newton's first law of motion*. It states that an object will maintain its state of motion unless it is influenced by an external force [8]. Consequently, when measuring this force, an object's motion can be mapped.

### 3.2.1 Inertial Sensors

**Accelerometer**

Accelerometers specify the *linear acceleration* of an objects along one axis by measuring the forces affecting a *proof mass* [8]. When three accelerometers are combined and aligned orthogonally, the 3D acceleration can be determined. Translative motion can be derived from linear acceleration by numerically integrating each individual value twice [8]. On principle, the sensor detects gravity as an acceleration since it constitutes a force. Consequently, the influences of gravity on the sensor need to be compensated to provide accurate results.

Among others, two types of error accompany accelerometers and need to be considered [8]. Due to the *zero offset error* a resting accelerometer might show a linear acceleration other than zero. The *scale error* indicates an inaccuracy in the mapping between force and acceleration. These discrepancies are causing an *error accumulation* which is commonly known as *drifting*.

**Gyroscope**

Gyroscopes meter the *angular rate of rotation* around a single axis [8]. While traditional gyroscopes are based on a rotating disc within movable gimbals, modern sensors observe the behavior of a resonating proof mass. Similar to accelerometers, three gyroscopes are orthogonally combined to form a 3-axis gyroscope. By numerically integrating each value once, the 3D rotation angle of an object can be calculated [8]. Gyroscopes are vulnerable to the same sources of error as accelerometers. Thereby, the concepts of the zero offset error and the scale error can be relayed.

**Magnetometer**

A Magnetometer can be described as a digital compass. It measures the surrounding magnetic field to calculate the direction of Earth's magnetic north [8]. By design, the magnetometer is affected by electro-magnetic fields. As a consequence, two common effects are distorting the results of the measurements [8]. The *hard iron effect* describes the constant influence of electromagnetic materials that are part of the measurement device. The *soft iron effect* is characterized by varying influences from disruptive elements in close proximity.

### 3.2.2 Sensor Fusion

#### Inertial Measurement Unit (IMU)

The combination of a triaxial accelerometer with a triaxial gyroscope is called *Inertial Measurement Unit (IMU)* [8]. If the initial position and rotation of an object is known, an IMU can be used to map an object's motion through space. Hybrid navigation systems often apply IMUs to enhance the robustness of the tracking system. For instance, most GPS navigation devices incorporate IMU readings into the position calculation to ensure a continuous operation in difficult environments.

IMUs have a fundamental weak point. As mentioned before, the initial position and orientation cannot be calculated based on inertial forces alone and thereby must be known in advance. Since inertial sensors simply have no way of determining their absolute spatial position, this issue can only be handled in hybrid sensing solutions [8]. The challenge of determining the absolute orientation is more complex. As mentioned earlier, accelerometers detect gravity as an inertial force. Consequently, the direction of the gravitational acceleration can be used as a vertical reference for the gyroscope. However, an additional horizontal reference is necessary to compute the horizontal orientation, also known as the heading. This is where the magnetometer comes into play.

#### Attitude and Heading Reference System

An *Attitude and Heading Reference System (AHRS)* builds on the data from the IMU and assimilates the magnetometer measurements to calculate the absolute orientation of an object successfully [8]. In addition, these horizontal and vertical references can be used to reduce gyroscopic drifting since rotational deviations are corrected continuously [16]. Every AHRS is based on a sensor fusion algorithm that transforms the raw sensor data into positional and orientational information. The *Madgwick filter* [47, 48], the *Mahony filter* [49], and the Kalman filter [34] represent the most popular techniques and are widely used for inertial navigation [16]. AHRS algorithms handle the following three aspects of sensor fusion:

1. *Data Smoothing*: Raw sensor readings are usually fluctuating heavily due to signal noise and other varying factors [16]. Therefore, the sensor data needs to be smoothed first.

2. *Pose Calculation*: The positional shift and the absolute orientation is computed by fusing the sensor readings from the accelerometer and the gyroscope with the gravity calculations from the accelerometer and the heading information from the magnetometer [8].

3. *Drift Correction*: By matching the gyroscope readings with the results from the sensor fusion calculations, the parameters causing rotational drift can be located and adjusted [16].

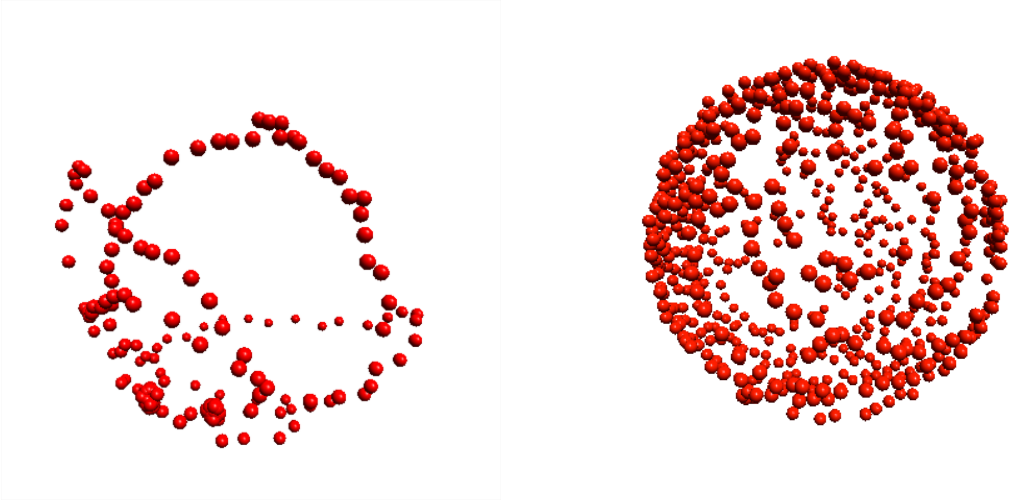
### 3.2.3 Sensor Calibration

#### Factory Calibration

Inertial sensors need to be calibrated in order to keep the inevitable error accumulation to a minimum. Many sensor manufacturers perform a factory calibration that covers known and constant influences. This involves minimizing the zero offset and scale errors in accelerometers and gyroscopes since those parameters are independent from the environment. However, some uncertainties remain and need to be addressed by the user before adequate measurements can be carried out. Accelerometers need to determine the direction of gravity to counteract its force. This can be done easily by placing the sensor on a solid surface. In this connection, the sensor might also revise the accelerometer's and gyroscope's zero offset errors.

#### Magnetometer Calibration

The calibration of the magnetometer, i.e. modeling the hard iron and soft iron effects, is a little more complex. The task can be classified as a *sphere fitting problem* [16]. At first, the sensor needs to be rotated randomly until every conceivable pose has been taken into account. Every pose represents a data point in a point cloud that an algorithm attempts to fit on a sphere. *Fig. 3.9* shows the point cloud during and after sensor calibration.



(a) Point cloud during calibration [69].

(b) Point cloud after calibration [2].

Figure 3.9: Magnetometer calibration.

### 3.2.4 Data Representation

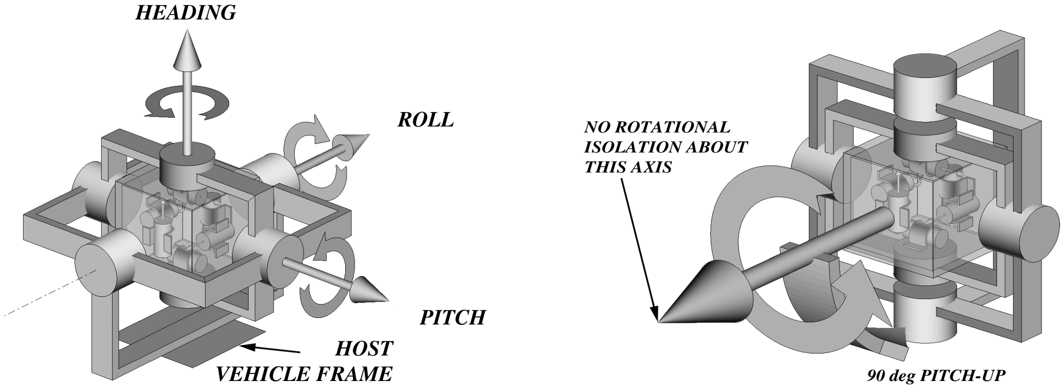
#### Euler Angles

Euler angles describe the pose of an object by specifying the angles $(\theta, \phi, \psi)$ with regard to the coordinate system's axes $(x, y, z)$ [17]. As *Eq. 3.6* illustrates, the 3D rotation $R_{xyz}$ through an Euler angle $e = [\theta, \phi, \psi]^T$ consists of three sequentially executed rotations $R_x(\theta)$, $R_y(\phi)$, $R_z(\psi)$ [17].

$$R_{xyz} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos(\theta) \end{bmatrix}}_{R_x(\theta)} \cdot \underbrace{\begin{bmatrix} cos(\phi) & 0 & -sin(\phi) \\ 0 & 1 & 0 \\ sin(\phi) & 0 & cos(\phi) \end{bmatrix}}_{R_y(\phi)} \cdot \underbrace{\begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_z(\psi)} \tag{3.6}$$

#### Gimbal Lock

Euler angles suffer from an anomaly called *gimbal lock* where minor angular changes may result in erratic output changes of ±180° [23]. The term originates from gimbal-based gyroscopes *(see Fig. 3.10)*. Similar to Euler angles, they derive the orientation of an object by combining the rotation angles of the individual axes. However, if the orientations of two gimbals coincide, the system looses the ability to rotate around a certain axis, i.e. the gimbals are locked. To circumvent this issue, most AHRS algorithms base their calculations on *quaternions* [39] which represent orientation data in form of 4-dimensional vectors.



(a) Euler angles.

(b) Gimbal lock.

Figure 3.10: Euler angles demonstrated on an triaxial gimbal [23].

## 3.3 Digital Image Processing

### 3.3.1 Color Models

*Color models* are used to characterize the structure of color images [15]. Different color models can be used to extract varying features from image data. Consequently, the choice of the optimal color model depends on the individual use case.

#### RGB Color Model

The *RGB color model* is used to display color on television screens and computer monitors. It is based on the principle of additive color mixing where many colors, although not every one, can be formed by combining the three primary colors red, green and blue. *Fig. 3.11* illustrates an *RGB* image and its individual color channels.

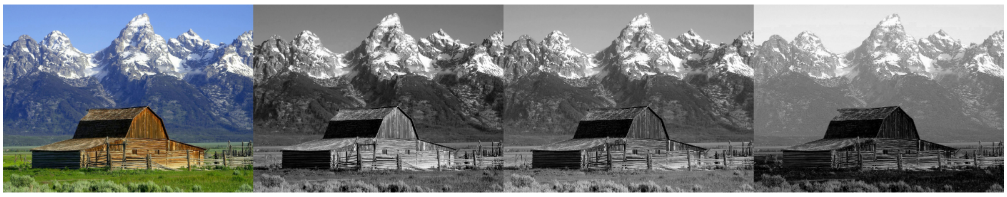


Figure 3.11: An *RGB* image splitted into individual color channels (based on [78]).

#### YCbCr Color Model

The $YC_bC_r$ *color model* is primarily used for digital television encoding and image compression [15]. It is composed of a luminance channel $Y$ and two chroma channels $C_b$ and $C_r$ *(see Fig. 3.12)*. The chroma channels describe the difference between the luminance and the blue respectively red color. An $R'G'B'$ image[7] can be easily converted into an $Y'C_bCr$ image by using the following linear transformation [75]:

$$\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \cdot \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{3.7}$$

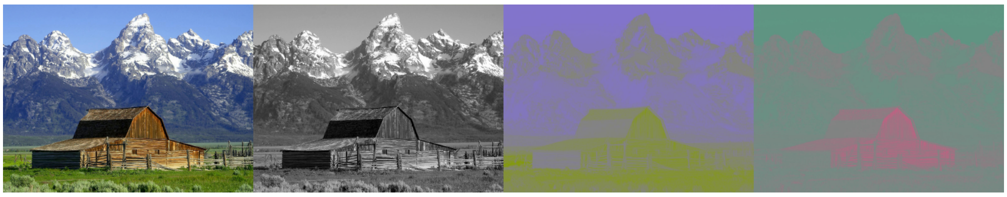


Figure 3.12: An $YC_bC_r$ image composed of the luminance and chroma channels [78].

---

[7] An $R'G'B'$ image is defined as the gamma-corrected counterpart to a conventional *RGB* image [75].

### 3.3.2 Image Morphology

*Mathematical Morphology* is a simple image processing technique that can be used for noise removal, edge detection and other types of image enhancements [72]. The basic idea is to eliminate or emphasize certain aspects of the image in order to obtain necessary information. In most cases, morphology is performed on *binary images.* In contrast to common color images, every pixel of a binary image stores a binary state, i.e. 1 or 0, instead of a color value. There are various ways to convert a greyscale image to a binary image. *Thresholding* offers a simple but effective solution where all pixel values below a threshold are set to 0 and all values equal or above are set to 1. A pixel that stores a logical 1 is called *foreground pixel.* Consequently, elements containing a value of logical 0 are named *background pixels* [72].

The fundamental principle of image morphology is to compare the pixels of a binary image with a binary reference object called *structuring element* [72]. Like the binary image, the structuring element is a two-dimensional array containing logical pixel values. Depending on the size of the array and which of its pixels store a logical 1, different shapes can be depicted. As *Fig. 3.13* illustrates, the most common shapes are discs, rectangles or lines. The most central data entry of the structuring element is called *center pixel* and denotes its origin. The pixels around the origin containing a value of logical 1 are called *neighborhood.*

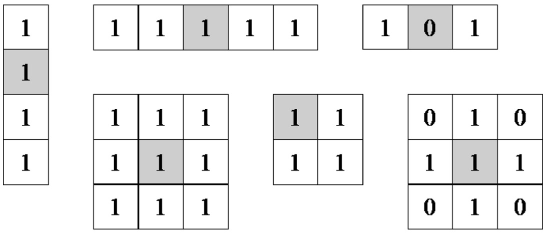


Figure 3.13: Overview of structuring elements [72].

#### Primitive Operations

Image morphology defines two *primitive operations* called *dilation* and *erosion (see Fig. 3.14).* The dilation of a binary image $A$ with a structuring element $B$, formally described as $A \oplus B$, consists of two nested steps [72].

1. Slide the structuring element over the binary image by successively superimposing every background pixel of $A$ with the center pixel of $B$.
2. For each step, determine if any pixels of $B$'s neighborhood coincide with a foreground pixel of $A$. In that case, the background pixel superimposed by $B$'s center pixel is changed to a foreground pixel.

The erosion of a binary image $A$ with a structuring element $B$ proceeds in a similar fashion [72]. It is formalized as $A \ominus B$. In contrast to dilation, each foreground pixel of $A$ is superimposed with the center pixel of the structuring element $B$ and switched to a background pixel if any pixels of $B$'s neighborhood coincide with a background pixel of $A$.

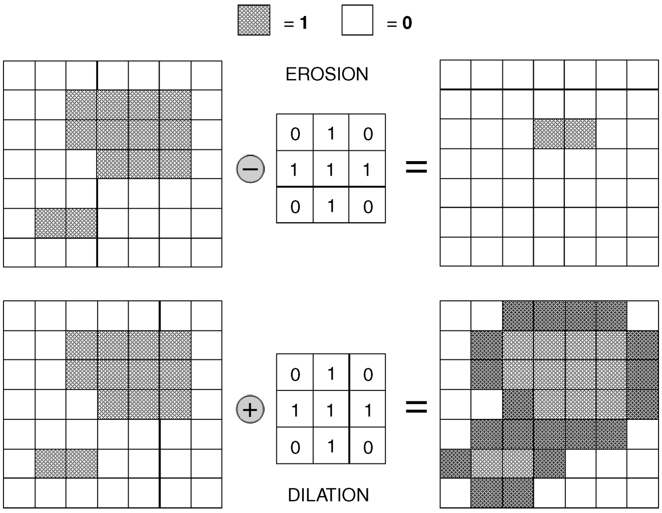


Figure 3.14: Primitive operations of image morphology [72].

### Advanced Operations

A variety of *advanced operations* can be executed by combining primitive operations [72]. The morphological *opening* operation performs an *"erosion followed by a dilation with the same structuring element"* [72]. It is formally described as $A \circ B = (A \ominus B) \oplus B$. This operation can be used to remove small foreground objects from the binary image. When a dilation is performed before an erosion using the same structuring element, formally denoted as $A \bullet B = (A \oplus B) \ominus B$, the operation is called *closing*. It is used to fill small holes in a foreground object. Based on this concept, further features like *skeletonization* or *region filling* can be implemented [72].

CHAPTER 4

# System Design

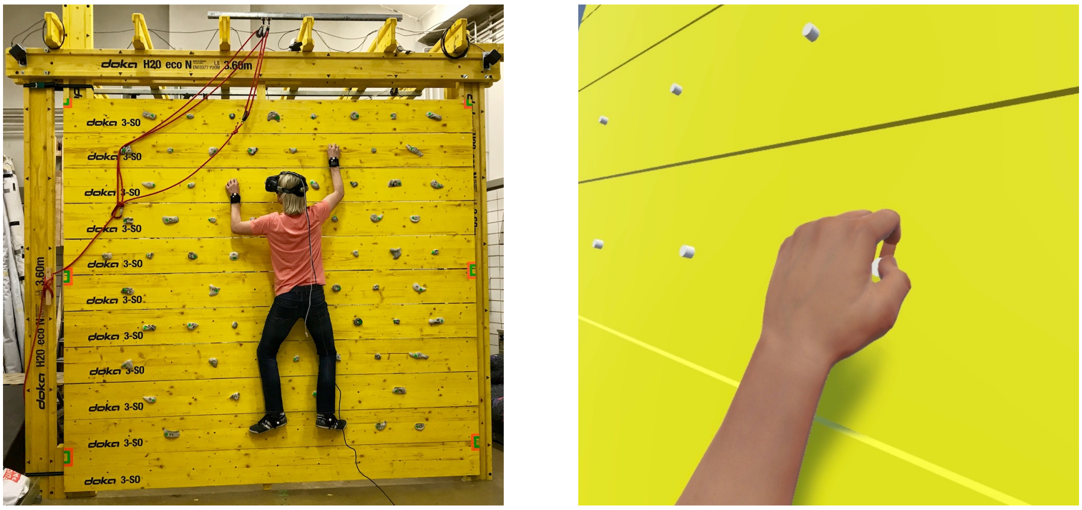
So far, an overview of the current state-of-the-art has been presented and the theoretical background of the applied methods has been established. The following chapter initiates the practical part of this thesis. As stated in *Chapter 1*, aim of this work is to implement a human motion capture system for a novel VR rock climbing application called *VreeClimber*. The developed prototype is named *VreeTracker*. At first, this chapter describes the tracking scenario and its requirements in more detail. It continues by evaluating acclaimed sensing techniques for suitability. Finally, the VreeTracker's principle design is illustrated.

## 4.1 Requirements

### 4.1.1 Tracking Scenario

The VreeClimber is a three meters wide and three meters high indoor rock climbing wall. It incorporates moving wall elements that shift downwards as the user ascends on the wall. Thereby, the upwards motion of the user is counteracted and the climbing experience can be prolonged indefinitely despite the limited height of the climbing wall. With the help of an HMD and soundproof headphones, the user is immersed in a virtual world *(see Fig. 4.1)*. As stated before, the climber's visual and auditory perception is restricted to the virtual content provided by the application. Virtual models of the user's hands and feet provide the necessary visual feedback for climbing the wall. The VreeTracker controls these models. It detects the user's extremities in the real world, describes their position and pose in form of numeric values, and streams the resulting data to the VR application.

While the *position* of the limbs is self-explanatory, the term *"pose"* needs to be clarified. The pose of a foot is described as its absolute orientation in the real world. Naturally, humans can change the form of their feet by moving their toes. However, since the climbing adventure requires the use of footwear, the pose of the toes is irrelevant.



(a) Tracking scenario. (b) Virtual view from user's perspective.

Figure 4.1: Comparison of the real world and the virtual view.

The pose of the hands is a different matter. For one thing, the human hand can assume various shapes, i.e. poses, by moving the fingers. Further on, it is important to take the pose of the hand into account since the climber continuously opens and closes the hand when gripping a climbing handle. Therefore, the pose of a hand is interpreted as its absolute orientation in the real world in combination with its shape.

### 4.1.2 Quality Characteristics

In order to evaluate quality of the implemented prototype, certain characteristics need to be established. Firstly, mathematical metrics like accuracy, precision, latency and update rate can be used to determine the *quality of the measurement* itself. The *robustness* of the system indicates if continuous measurements can be performed without interruptions by disruptive factors. The third cornerstone describes the *usability* of the motion capture system. On the one hand, the climber's mobility must not be impaired by wearable devices in case they are needed. Reasonable metrics are the size and weight of the object as well as the point of application. In addition, the system needs to be easily manageable for the operators. The setup should be simple and fast and no cumbersome procedures should be necessary during active operation.

### 4.1.3 Sensing Technologies

*Chapter 2* illustrated that the current state of the art favors optical and inertial sensing techniques. Since many systems combine both approaches to form various hybrid solutions, it is necessary to analyze the advantages and disadvantages of both methods in order to form an individual solution that meets the VreeClimber's needs.

### Optical marker-based Sensing

- *Quality of Measurements:* The biggest advantage of optical marker-based sensing techniques is the quality of the position detection. Most systems are capable of determining the absolute position of an object within a margin of error of only a few millimeters. Since the markers are easily detected by optical sensors, the method works highly efficient and usually achieves adequate update rates.
- *Robustness:* A potential drawback of optical tracking is the need for a direct line of sight between the markers and the sensor. This makes occlusions a common issue that affects the robustness of the tracking system negatively. By strategically placing multiple sensors around the markers, occlusions can be handled satisfactorily.
- *Mobility:* Judging the mobility of an optical marker-based tracking system is more complex. Optical markers are usually very small and lightweight which makes them a preferred choice in many situations. However, depending on the point of application, the sheer necessity of a marker can be an exclusion criterion. In addition, if the optical system is used to determine the orientation of the object, a rigid-body target consisting of multiple markers needs to be applied. In many cases, the size of the rigid-body target exceeds the designated limits.
- *Calibration:* The intrinsic calibration can be cumbersome, but only needs to be performed once per camera. As long as the internal camera parameters stay the same, a re-calibration is not necessary. An extrinsic re-calibration is mandatory when the cameras are moved. To optimize the quality of measurements, it is recommended to re-calibrate the external parameters prior to each tracking session. Nevertheless, this task can be performed within minutes and might be executed automatically by the tracking system.

### Optical markerless Sensing

- *Quality of Measurements:* In contrast to optical marker-based tracking systems, markerless techniques detect natural features in image data. This approach is computationally far more expensive and usually produces results that are inferior in terms of accuracy, precision, and update rate. Depending on the necessary image processing, latency can become a fatal issue in real-time applications.
- *Robustness:* Optical markerless tracking systems are based on the same principles as marker-based systems. Therefore, occlusions are equally problematic.
- *Mobility:* The key benefit of optical markerless tracking systems is the fact that they operate without the need for artificial landmarks. Thereby, the user's mobility is not affected by any wearable tracking devices.
- *Calibration:* Optical markerless tracking systems use the same calibration algorithms as marker-based systems. Consequently, the same calibration directives apply.

**Intertial Sensing**

- *Quality of Measurements:* Inertial sensors are capable of delivering high quality orientation data with update rates that simply cannot be matched by optical systems. However, due to error accumulation, inertial techniques are a poor choice for determining the position of an object. Additionally, since the sensors only deliver a positional shift relative to an arbitrary origin, the absolute position cannot be determined.

- *Robustness:* Since inertial sensors solely base their readings on inertial forces, they operate completely autonomous and are independent of external components. Still, strong electromagnetic fields in close proximity might affect the readings.

- *Mobility:* Depending on the concrete scenario, inertial sensors might impair the user's mobility. They can be built relatively light and have a reasonable size, but still might be too large or heavy for certain use cases.

- *Calibration:* Inertial sensors need to be regularly re-calibrated. Some vendors apply black-box calibration algorithms that are automatically performed by an integrated sensor chip. Other sensors lack this feature and need to be re-calibrated manually at regular intervals.

### 4.1.4 Conclusion

Based on these findings, the developed prototype combines the beneficial properties of all three approaches and forms a new hybrid sensing technique. It uses optical marker-based sensing to determine the position of the climber's extremities and inertial sensors to calculate their absolute orientation. Both tasks can be performed with the help of small wearable devices that are mounted on the user's wrists and feet.

Estimating the shape of the climber's hands is a far more challenging task. Any device capable of detecting or marking the position and orientation of the fingers would dramatically impair the user during the ascend. Consequently, only a markerless design meets the requirement of reasonably unrestricted mobility. However, the detection of natural features is complex and the quality of measurements is usually inferior. That being said, the following two assumption justify the compromise between quality and mobility.

- *The climber only executes simple climbing motions:* Since the climber is engaged in a thrilling adventure, the likelihood of an unnatural hand pose is minimal. In most cases, the user will restrict the hand poses to simple gripping motions. Therefore, the complexity can be reduced.

- *The human brain accepts a slightly different hand pose as the truth:* The precise depiction of the hand pose is of secondary importance since the brain has no visual information about the real hand pose. It is the author's assumption that it might accept a slightly different pose as the truth.

## 4.2 System Overview

The motion capture system consists of three mostly independent components:

- *Optical Position Tracking:* Two stereoscopic IR cameras capture a total of four active IR markers which are attached to the climber's limbs. A computer vision algorithm detects the markers in the IR images, calculates their 3D positions from the 2D camera views, assigns all markers to their corresponding extremity, and transmits the position data to the *VR game engine*[1]. Furthermore, the hands' position data is sent to the *visual hand pose estimation* component.
- *Visual Hand Pose Estimation:* Two RGB cameras are used to visually search for the hands in close proximity to their corresponding IR marker position. An image processing algorithm extracts visual features from the images, describes the estimated hand poses with numeric values and relays the information to the VR game engine.
- *Inertial Orientation Tracking:* Wrist- and foot-worn inertial sensors calculate their absolute orientation with the help of a sensor fusion algorithm and broadcast the orientation data to the VR game engine.

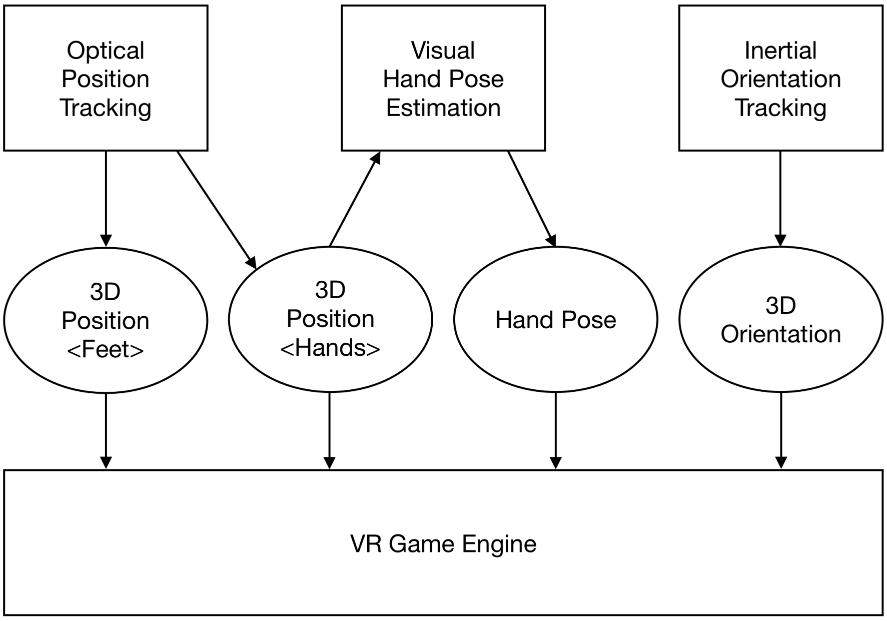

Optical Position Tracking
Visual Hand Pose Estimation
Inertial Orientation Tracking
3D Position <Feet>
3D Position <Hands>
Hand Pose
3D Orientation
VR Game Engine

Figure 4.2: System overview.

---

[1] The VR game engine is the software framework in which the virtual simulation runs. Content developers use it to create virtual environments. The game engine applies the tracking data to control a virtual character.

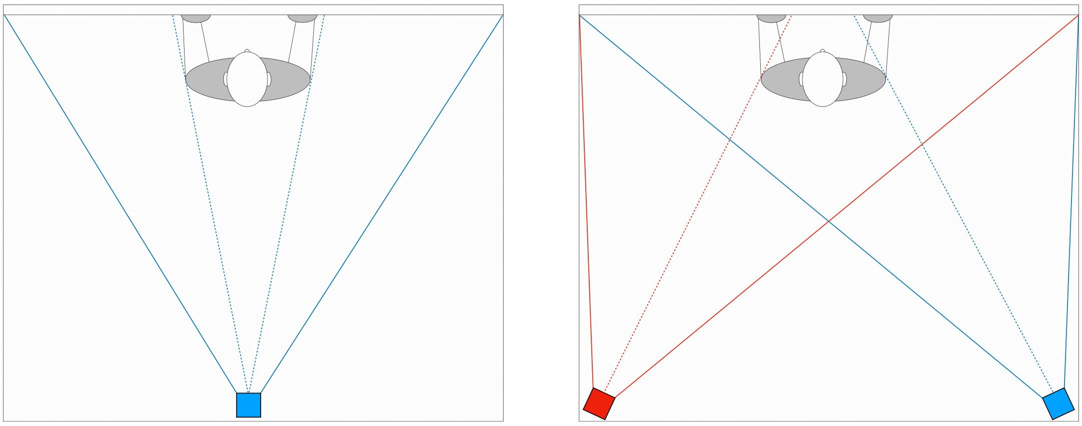
## 4.3 Optical Position Tracking Component

The *position tracking component* is based on an optical marker-based outside-in tracking system. Four cameras, acting as two stereoscopic cameras, are capturing a set of four active IR markers. The markers are attached to the climber's limbs and thereby tag these *targets* in the IR images. A computer vision algorithm detects the markers in the 2D views, calculates the 3D position of each marker and outputs the position data.

### 4.3.1 System Setup

As mentioned before, occlusions are a chief concern of optical tracking systems. Early designs showed that regardless of the set-up location, a single stereoscopic camera is not capable of facilitating a robust tracking scenario where all four markers are simultaneously detected. The biggest challenge is to maintain a clear line-of-sight between the cameras and the hands. The visual connection is lost as soon as the climber places the hands in front of the body *(see Fig. 4.3a)*. Therefore, the optical tracking component utilizes two cooperating camera sets. *Fig. 4.3b* illustrates the applied room setup. The left camera set has a clear line-of-sight to the left extremity markers while the right camera set can trace the right limbs without interruptions.

Due to performance issues that are discussed at a later point, each camera set runs on its own host computer. This means that two instances of the position tracking algorithm are executed independently on different machines. Therefore, the computer vision algorithm needs to differentiate between the left and right side of the body since the left camera set should only track the left extremities and vice versa.



(a) Single camera setup.

(b) Double camera setup.

Figure 4.3: Horizontal projection of the tracking area.

### 4.3.2 Hardware Components

#### Stereoscopic Camera System

The *stereoscopic camera system* provides the raw IR image data for the computer vision algorithm. Using a commercially available IR stereo camera poses two major issues. Firstly, high quality IR cameras are very expensive. Secondly, most manufacturers of stereoscopic cameras place the camera lenses very close to each other. While this narrow constellation has advantages in situations where the object is close to the camera, it is not suitable for large scale tracking scenarios.

Instead, in the present application each camera set is composed of two common 2D RGB webcams. This has the advantage that the distance between the cameras can be chosen freely. Obviously, the cameras need to be converted into IR cameras. This means that instead of being sensitive to visible light, the cameras need to react solely to IR light. When both cameras are interconnected firmly, they act as a single stereoscopic camera. This combination is henceforth referred to as *camera rig*.

#### Active Infrared Markers

The camera system detects four active IR markers that are placed on the climber's wrists and feet. Each active marker uses an IR LED that emits NIR light towards the cameras. A diffusing sphere on top of the LED guarantees a uniform light distribution in any direction. Naturally, the active marker needs a power supply to operate.

#### Camera Calibration

The optical tracking component needs to be calibrated before it can operate properly. As with any optical system, the calibration process consists of an intrinsic and an extrinsic calibration. The intrinsic calibration determines the camera's inner optical parameters and corrects possible inaccuracies in the camera's optical path. The extrinsic calibration calculates the camera's position and orientation relative to a specified point of reference. The intrinsic camera calibration only needs to be performed once for each camera since the concerned parameters do not change over time. In contrast, the extrinsic calibration needs to be repeated every time the cameras are repositioned. To ensure optimal tracking accuracy, it is best practice to re-calibrate the extrinsic parameters at the beginning of every tracking session.

### 4.3.3 Software Architecture

The computer vision algorithm responsible for calculating the 3D positions of the markers is composed of several modules. Each module fulfills a specific task in the computer vision pipeline. As a result, the software architecture is a highly flexible structure where every module can easily be replaced by a different implementation as long as the specifications are met. *Fig. 4.4* illustrates the modular design.

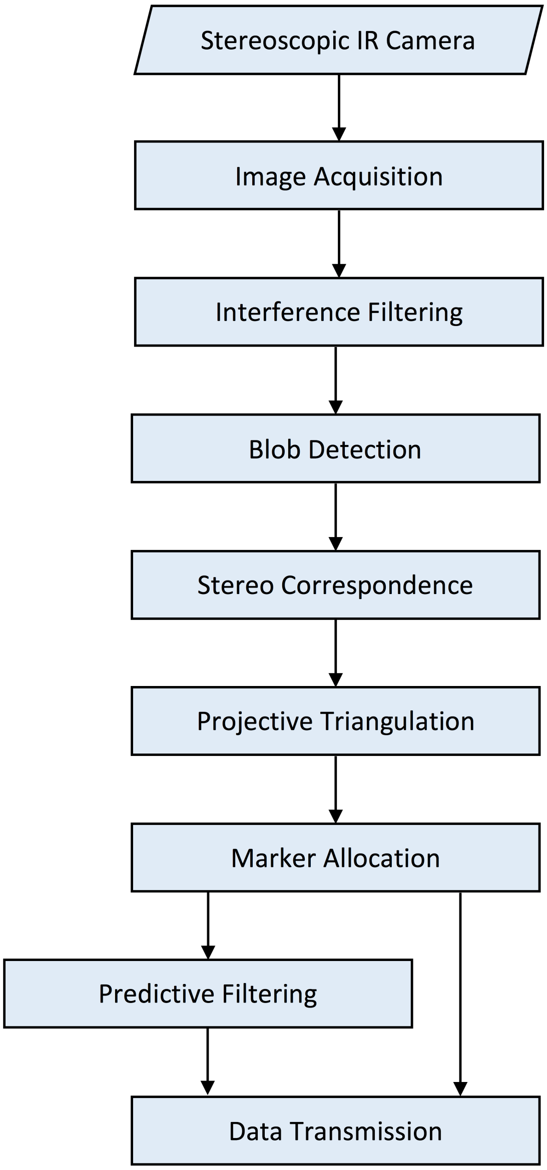


Figure 4.4: Software architecture of the optical position tracking component.

### Image Acquisition

The *image acquisition* module is responsible for receiving and forwarding a pair of IR images. It simultaneously collects the current image frame from both cameras and stores them in a data structure. In addition, it manages camera identification, maintains the camera connections, and conducts the extrinsic camera calibration.

### Interference Filtering

Ambient light usually contains a significant portion of IR radiation that interferes with the proper detection of the IR markers. The *interference filtering* module detects these interferences in the image frames and filters these regions accordingly.

### Blob Detection

The *blob detection* module receives two filtered images from the previous module and separates the markers from the dark background. This process is also known as *foreground segmentation*. The foreground objects are often called *blobs*. This module determines the centroid of every blob and stores their 2D positions in a list. Since the module operates on a stereoscopic set of images, two lists result.

### Stereo Correspondence

The sequential orders of the centroid positions do not necessarily match across the image sets. Ideally, every list entry of the first image relates to the corresponding list entry of the second image, i.e. both entries describe the 2D position of the same marker from two different perspectives. However, due to perspective distortion and partial occlusions, this best case scenario cannot be guaranteed. Therefore, the *stereo correspondence* module determines corresponding list entries and stores them in two new lists.

### Perspective Triangulation

Based on the corresponding centroid position pairs, the *perspective triangulation* module calculates the 3D positions of the markers and stores them in a list.

### Marker Allocation

After calculating the 3D position of the markers, their affiliation to a specific limb must be determined. In other words, each marker position needs to be allocated to the corresponding extremity. The *marker allocation* module addresses this task.

### Predictive Filtering

The frame rate of the IR cameras is not high enough to map fast movements smoothly. Therefore, the *predictive filtering* module interpolates the position of the markers in-between two consecutive frames to double the algorithm's update rate.

### Data Transmission

The *data transmission* module provides a continuous data stream of labeled position datasets. It alternately transmits the detected or the interpolated position data to the VR game engine and the hand pose estimation component.

## 4.4 Visual Hand Pose Estimation Component

The *visual hand pose estimation component* is based on an optical markerless outside-in tracking system. In contrast to the position tracking system, it does not rely on the detection of artificial IR landmarks. Instead, it searches for natural features that are characteristic for the object of interest.

### 4.4.1 Problem Definition

The hand pose is characterized by the palm and the fingertips. In an ideal scenario, the pose of the hand is determined by identifying each fingertip as well as the center of the palm. Based on these features, a virtual model of the hand could mimic the pose of its real counterpart precisely. However, the implementation of such a sophisticated markerless hand pose detection component exceeds the scope of a diploma thesis by far.[2] Two major issues are partial occlusions of the fingers and general image blurring during fast movements. In order to facilitate a robust pose tracking algorithm, the problem definition needs to be slightly adapted.

As a pre-condition, the hand pose estimation algorithm assumes that the climber only performs a grasping movement that ranges from a fully opened to a firmly closed hand. During this movement, all fingers carry out a mutual motion. Therefore, it is not necessary to map each finger individually. Instead, it is sufficient to determine the general opening width of the hand. In this context, the opening width is defined as the distance between the fingertips and the center of the palm.

### 4.4.2 System Setup

To minimize occlusions, each hand is captured from its own perspective. Therefore, a single RGB camera is integrated in each camera rig. Both cameras cover the full width of the VreeClimber but only recognize their designated extremity. Just like before, each perspective is managed by its own software instance running on its own host machine. Both RGB cameras are calibrated to the same reference point as the IR cameras using the same calibration algorithm.

### 4.4.3 Software Architecture

Similar to the optical position tracking component, the hand pose estimation component is comprised of independent modules that are interconnected via specified interfaces. *Fig. 4.5* depicts the algorithmic structure.

---

[2] Due to the complexity of optical markerless sensing, even well funded commercially oriented projects like [41, 30, 73] have been attempting to develop a robust markerless hand detection system for years.

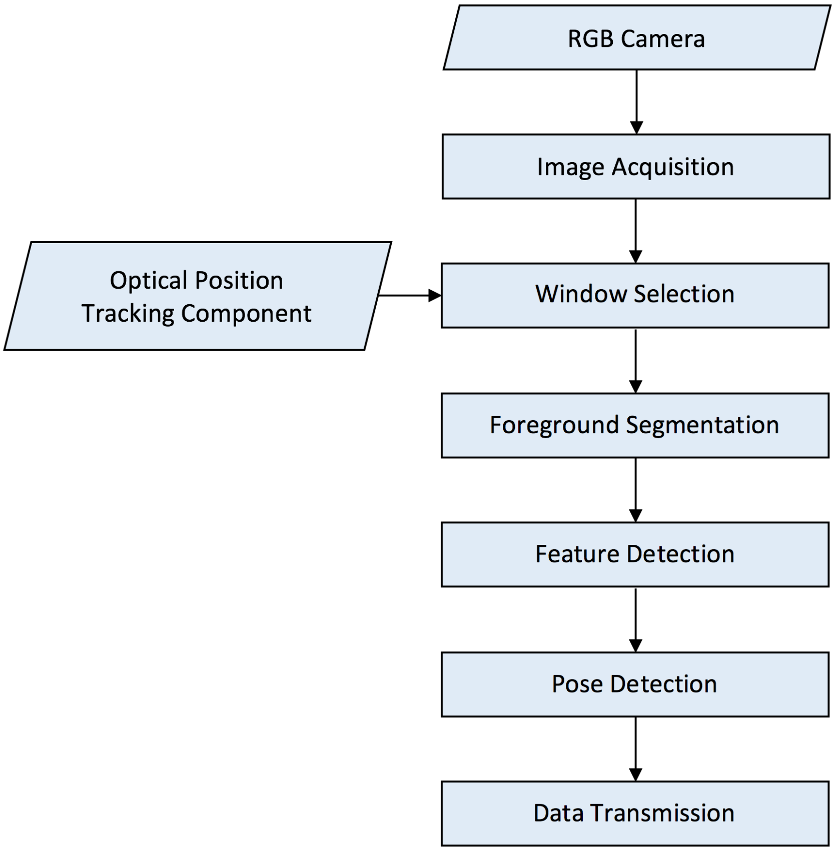


Figure 4.5: Software architecture of the visual hand pose estimation component.

### Image Acquisition

The *image acquisition* module handles all camera-related tasks of the hand pose estimation component. It interfaces with an RGB camera and provides the current image frame to subsequent modules. In addition, it is responsible for the extrinsic camera calibration. The RGB camera is calibrated with the same reference pattern as the IR camera rigs.

### Window Selection

Since the image processing algorithm runs in real-time, every task must be executed as efficiently as possible. A key factor is the number of pixels that the algorithm operates on. Therefore, the area in which the hand is searched needs to be reduced to a small region.

The *window selection* module receives the 3D position of the concerning hand marker from the position tracking component. Since all cameras are calibrated to the same reference point, this 3D position can be mapped directly to a pixel in the RGB image. The algorithm selects a search area around this pixel and discards the remaining image.

### Pose Estimation

The *pose estimation* module contains the image processing algorithm that performs the markerless hand pose detection. It is comprised of the following tasks.

1. *Foreground Segmentation:* Separate the hand from the background.
2. *Feature Detection:* Search for the fingertips and the center of the hand's back, and store the image coordinates of these landmarks in a list.
3. *Pose Detection:* Calculate the hand's opening width based on the detected features.

### Data Transmission

The *data transmission* module sends the hand pose data to the VR game engine.

## 4.5 Inertial Orientation Tracking Component

The *orientation tracking component* uses inertial sensors that are attached to the extremities to determine their absolute orientation. The sensor unit is connected to a micro-controller that fuses the raw sensor readings into a 3D orientation value and transmits this data wirelessly to the VR game engine.

### 4.5.1 Hardware Architecture

The orientation tracking component has a very simple hardware design *(see Fig. 4.6)*. It uses an inertial sensor array to measure the inertial forces acting on the device. The sensor array consists of a 3D accelerometer, a 3D gyroscope and a 3D magnetometer. It is connected to a micro-controller that calculates the 3D orientation based on the raw sensor readings. Afterwards, the micro-controller broadcasts the orientation data wirelessly to the VR game engine.

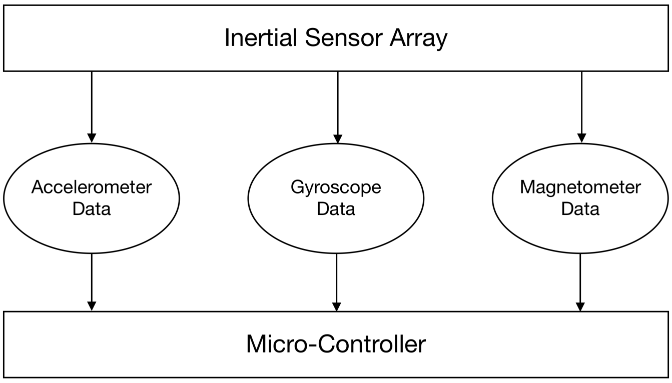


Figure 4.6: Hardware architecture of the inertial orientation tracking component.

### 4.5.2 Software Architecture

The software architecture is equally straightforward. At first, the raw sensor data is read and forwarded to a sensor fusion algorithm which converts the sensor readings into a 3D orientation value. This value is then transmitted to the VR game engine.

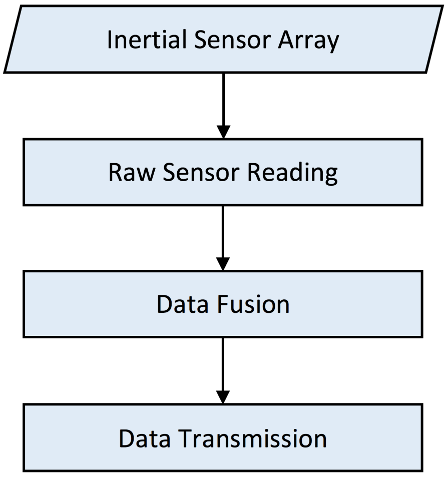


Figure 4.7: Software architecture of the inertial orientation tracking component.

## 4.6 Conclusion

This chapter presented the design aspects of the VreeTracker. It can be concluded that the VreeTracker aims at being a high-quality yet cost-effective motion capture system. Its modular architecture guarantees flexible operating capabilities. The use of off-the-shelf components ensures an easy replacement of malfunctioning equipment. The double camera setup facilitates a robust tracking operation.

## CHAPTER 5

# Implementation

The following chapter describes the implementation of the VreeTracker. It is divided into three distinct sections. The first segment discusses the *hardware development* of the camera rig and the wearable markers. The second part presents the *software implementation* of the prototype. The third part briefly introduces the *VR game engine* and illustrates the interface between the tracking system and the virtual content.

## 5.1 Hardware Development

### 5.1.1 Camera Rig

The camera rig consists of multiple optical sensors that capture the user on the VreeClimber. It contains two IR cameras that operate as a single stereoscopic group. They deliver the necessary images for the 3D position tracking. Additionally, the camera rig includes an RGB camera which is used by the visual hand pose estimation component.

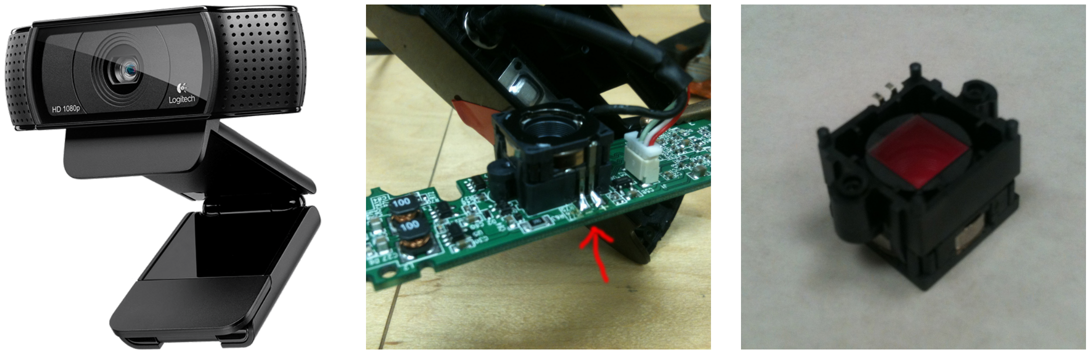
#### IR Cameras

The VreeTracker uses the Logitech C920 webcam [45] as an IR camera. Obviously, a common RGB webcam does not seem to be an ideal choice. Admittedly, it would be easier to use a native IR camera. However, aside of the far higher costs, high quality IR cameras are relatively hard to come by. In contrast, the Logitech C920 can be bought at almost any electronics store and is offered at a reasonable price. Furthermore, this camera model performs admirably in terms of quality. It provides high resolution images at an acceptable frame rate and allows control over many camera parameters. Ultimately, this webcam is an ideal candidate for a prototypical implementation. *Table 5.1* summarizes its technical specification.

| | |
|---:|:---|
| Video Resolution | 360p, 480p, 720p, 1080p |
| Frame Rate | 1080p@30fps |
| Diagonal Field-of-View | 78° |
| Camera Parameters | Exposure, Gain, Brightness,<br>Contrast, Focus, Saturation |
| Focus Type | Auto/Manual |
| Connection Type | USB 2.0 |

Table 5.1: Logitech C920's technical specification [46].

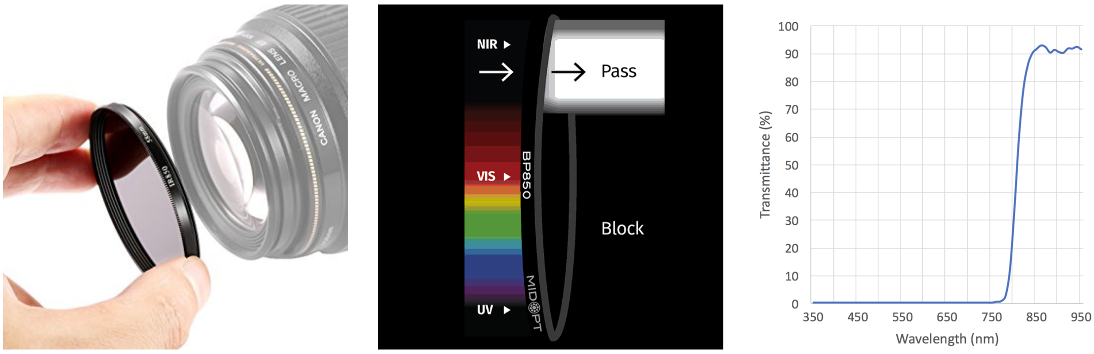
The RGB webcam needs to be transformed into an IR camera. As a first step, its *IR suppression filter* must be removed. The filter of the C920 is placed directly in front of the camera sensor behind the camera lens. After carefully removing all screws from the camera enclosure and cautiously disassembling the webcam piece by piece, the camera lens needs to be unsoldered from the circuit board. Now, the filter can be cut out gently with a very fine blade. It is crucial not to damage the image sensor in any way. Even a small dust particle on the sensor would impair the image quality drastically. Once the filter is successfully removed, the camera lens can be re-soldered on the circuit board and the webcam can be re-assembled. *Fig. 5.1* illustrates the process.



(a) Original webcam [45]. (b) Circuit board [5]. (c) IR suppression filter [5].

Figure 5.1: Disassembly of the webcam.

At this stage, the image sensor is sensitive to visible and IR light. As a result, ambient light sources are captured by the cameras which interferes with the correct detection of the IR markers. Therefore, the visible range of the light spectrum needs to be blocked with a *daylight suppression filter*. *Fig. 5.2b* demonstrates the operation principle. The VreeTracker uses simple *IR pass-filters* [6] that are usually attached to SLR cameras *(see Fig. 5.2a)*. The filter has a cut-off wavelength of approximately 850$nm$. *Fig. 5.2c* depicts a typical *filter transmittance curve*.

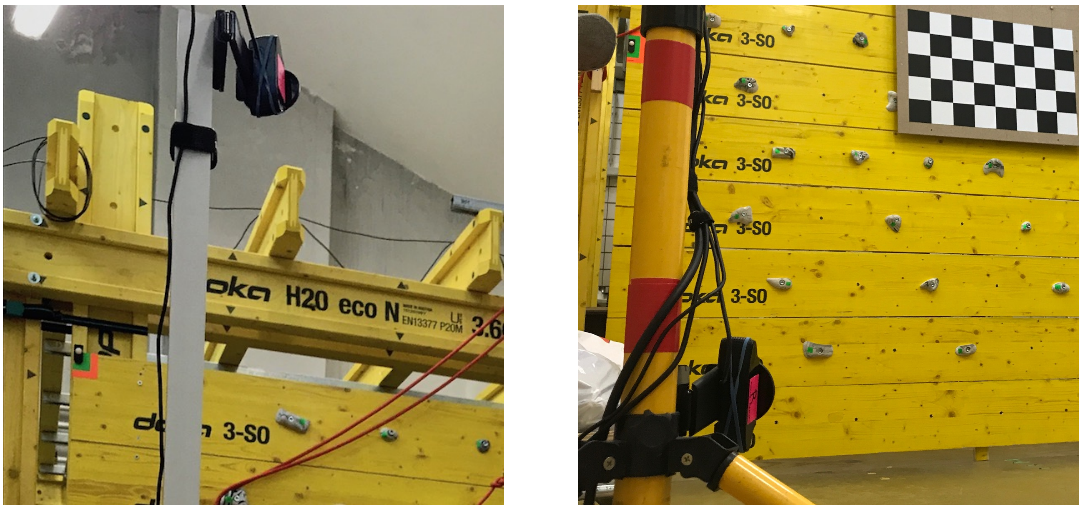


(a) IR pass-filter [6]. (b) Operation prinicple [77]. (c) Filter transmittance.

Figure 5.2: Characteristics of a daylight suppression filter.

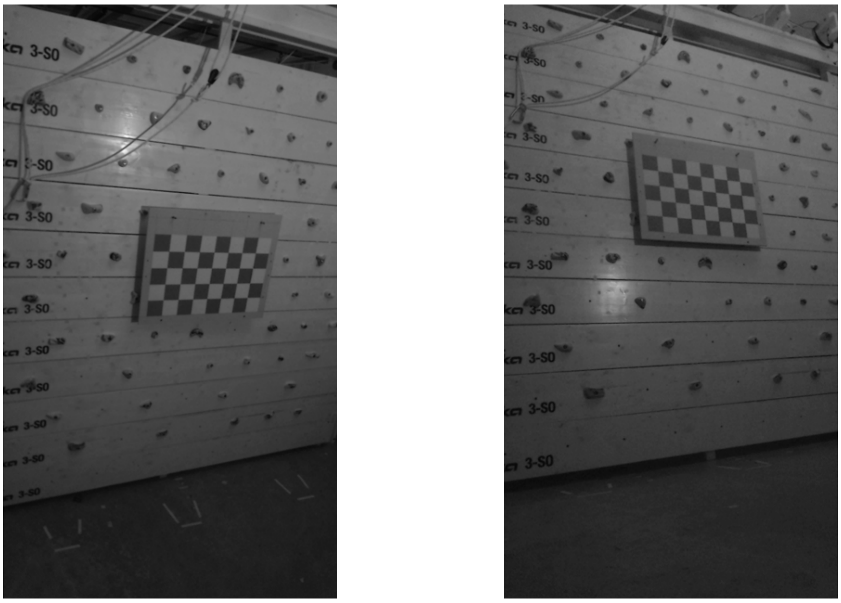
### Stereoscopic Camera Group

Two IR cameras are firmly interconnected on an aluminum rail. Together, they form a *stereoscopic camera group*. The distance between the cameras defines the baseline of the projective triangulation. Consequently, the chosen value influences the quality of the position measurements. The VreeTracker places the cameras at a distance of approximately two meters to maximize the disparity between both camera views. The cameras are tilted inwards to capture the same area from two different perspectives. The camera rig films the VreeClimber from a high and low angle. Both cameras are intentionally rotated 90° to maximize the coverage of the climbing wall. *Fig. 5.3* shows the camera rig. *Fig. 5.4* depicts the resulting image views from the IR cameras.



(a) High angle camera. (b) Low angle camera.

Figure 5.3: Stereoscopic camera group.



(a) High angle view.

(b) Low angle view.

Figure 5.4: Stereoscopic camera views.

### RGB Camera

In addition to the IR cameras, the camera rig incorporates an unmodified C920 webcam. This camera is installed at a height of approximately 140$cm$ and covers the whole width of the climbing wall. Since the camera is only used to capture the user's hands, the floor area is not covered by the camera's field of view. *Fig. 5.5* presents the RGB camera's perspective.

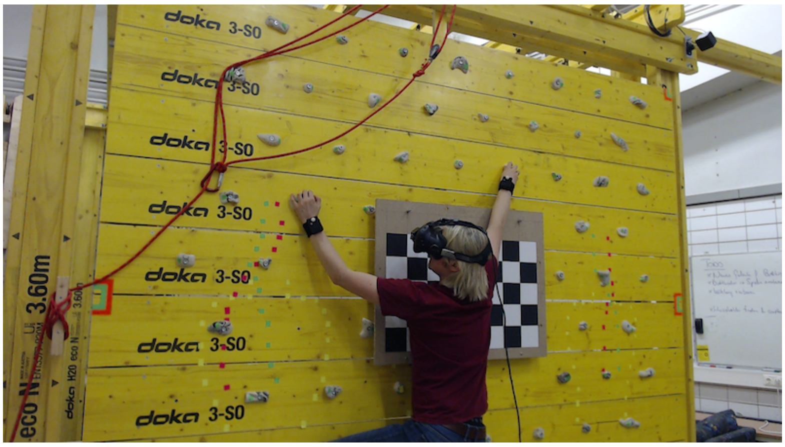


Figure 5.5: RGB camera view.

### Room Setup

The VreeTracker deploys two mirrored configurations of the camera rig to maximize the robustness of the optical position tracking component. Both rigs are placed in front of the climbing wall at a distance of approximately 2.5$m$. The camera placement corresponds to the horizontal projection presented in *Fig. 4.3b*. *Fig. 5.6* illustrates the camera rigs in front of the prototypical test wall.

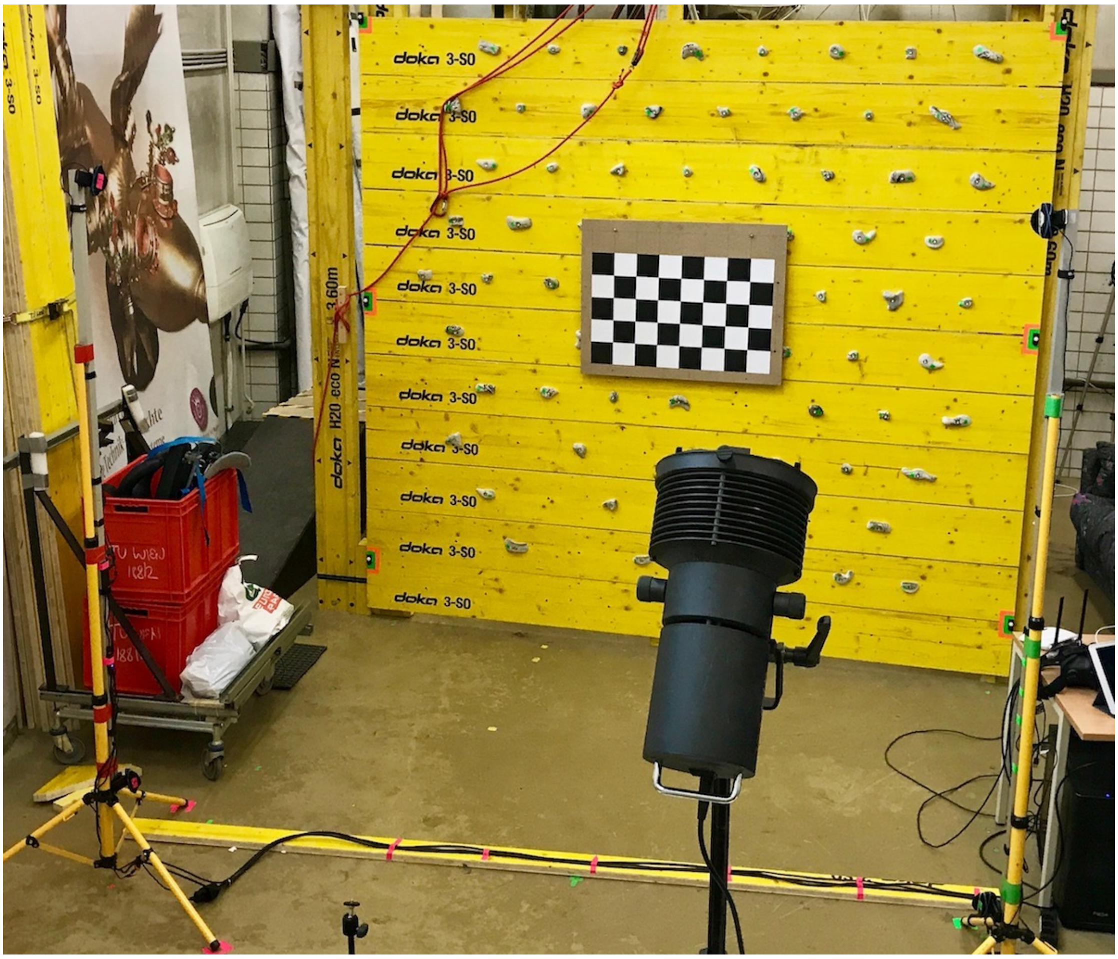
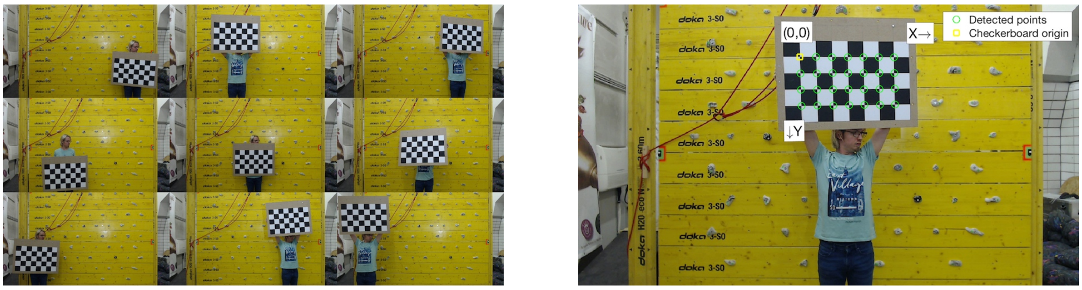


Figure 5.6: Prototypical setup of the camera rigs.

### Intrinsic Camera Calibration

After re-assembling the modified IR cameras and fitting daylight suppression filters to the camera lenses, the intrinsic camera parameters need to be calibrated. Needless to say, the same applies to the unmodified RGB cameras. In both cases, Matlab's *Single Camera Calibration App* is used to perform the intrinsic camera calibration. The application guides the user through a series of steps [56] which are described in the following.

1. *Preparation:* The camera calibration app uses a checkerboard pattern to calculate the intrinsic parameters of a connected camera. The application accesses the live camera feed and takes snapshots at a constant interval. The user needs to present the pattern to the camera and change the position and orientation of the checkerboard after every iteration. For optimal results, the algorithm requires around 20 images. *Fig. 5.7a* depicts a selection of the captured images.
2. *Loading:* The user chooses the images that are used in the calibration step. After specifying the size of the checkerboard elements, the algorithm detects the pattern in the loaded images and presents the results to the user *(see Fig. 5.7b)*.
3. *Calibration:* The application calculates the intrinsic camera parameters based on Zhang's technique [81].
4. *Evaluation:* After successfully completing the previous step, the user can evaluate the results of the calibration. A graph displaying the *reprojection errors*[1] helps identifying images that deteriorate the quality of the calibration *(see Fig. 5.8a)*.
5. *Adjustment:* Images with a *mean reprojection error* higher than one pixel should be removed from the selection and the calibration step should be repeated without them. *Fig. 5.8b* shows the reprojection error graph after refining the image selection.
6. *Export:* If the calibration passes the evaluation criteria, the computed *camera calibration matrix* can be exported and saved as a file.



(a) Calibration images.

(b) Detected feature points.

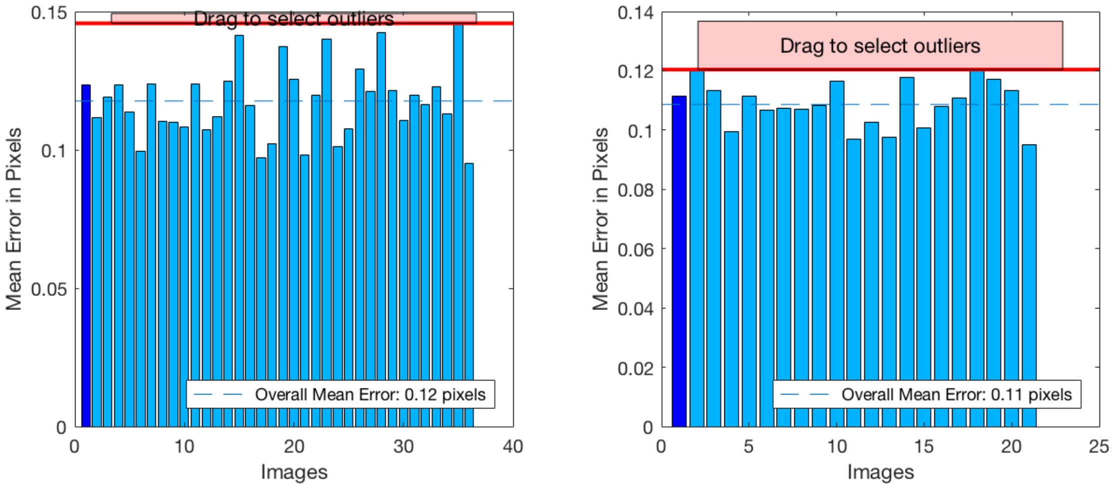
Figure 5.7: Snapshots of the intrinsic camera calibration.

### 5.1.2 Wearable Tracker

The optical position tracking and inertial orientation tracking components rely on wearable devices that need to be attached to the user's wrists and feet. Firstly, the position of the limbs is tagged with active IR markers. Secondly, the orientation of the extremities is calculated by inertial sensors. Both features are implemented in one single appliance.

---

[1] The *reprojection error* has been introduced in *Ch. 3.1.2.* Refer to [56] for additional information on the reprojection error in connection with intrinsic camera calibration.



(a) Graph before image exclusion.

(b) Graph after image exclusion.

Figure 5.8: Reprojection error graphs.

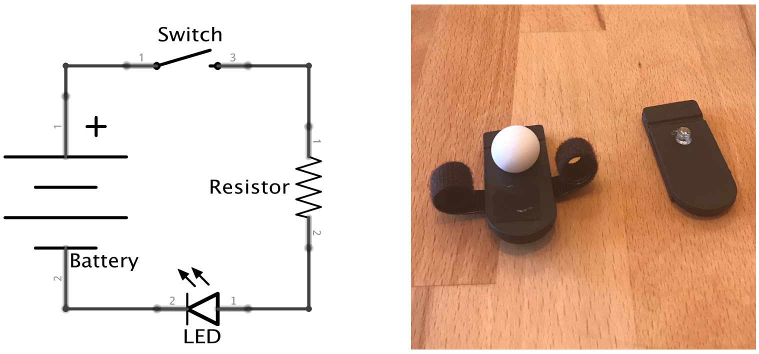
### Active IR Marker

Essentially, the *active IR marker* consists of an *IR LED* and a *battery*. The chosen LED emits IR light at a wavelength of 940$nm$. This value is slightly higher than the cut-off wavelength of the camera's daylight suppression filter. Consequently, the LEDs are fully visible to the camera's image sensor but invisible to the human eye. A *series resistor* compensates the voltage mismatch between the LED's permissible operating voltage and the supply voltage provided by the battery. A *switch* interrupts the electric circuit if required. *Fig. 5.9a* illustrates the circuit diagram of the active IR marker.

The LED has a viewing angle of approximately 90°. To ensure that the diode is visible from any direction, the emitted light needs to be scattered evenly with a *diffuser*. The company *Staedtler* produces a white translucent modeling clay called *Fimo Effect*. It can be shaped into a small sphere with a silicone form and hardened in an oven at 110°$C$. When the LED is inserted into the sphere, the clay gets illuminated from the inside and uniformly glows in all directions. *Fig. 5.9b* shows a prototypical IR marker with and without the diffusor sphere that was developed in the course of a preceding project.

### Inertial Orientation Tracker

The inertial orientation tracker is based on Adafruit's *NXP Precision 9DoF breakout board* [4]. It incorporates the *FXOS8700*, which is a combined triple-axis accelerometer and magnetometer, and the *FXAS21002* triple-axis gyroscope. The raw sensor data is sent to the *Adafruit Feather ESP8266 WiFi* micro-controller [3] which performs the AHRS calculations and wirelessly transmits the orientation data to the VR game engine. An 800$mAh$ *Li-Ion battery* powers the sensor module and the micro-controller.



(a) Circuit diagram. (b) Diffuser sphere.

Figure 5.9: Active IR marker.

In addition to the presented hardware components, a *rotary encoder* determines the ID of the orientation tracker. The selected value corresponds to the limb that the tracker is attached to. Consequently, the VR game engine is able to identify the origins of the four concurrent data streams and allocates the orientation data to the virtual hand or foot models accordingly. A *power switch* disconnects the battery from the micro-controller if needed. *Fig. 5.10* illustrates a schematic overview of the inertial orientation tracker.

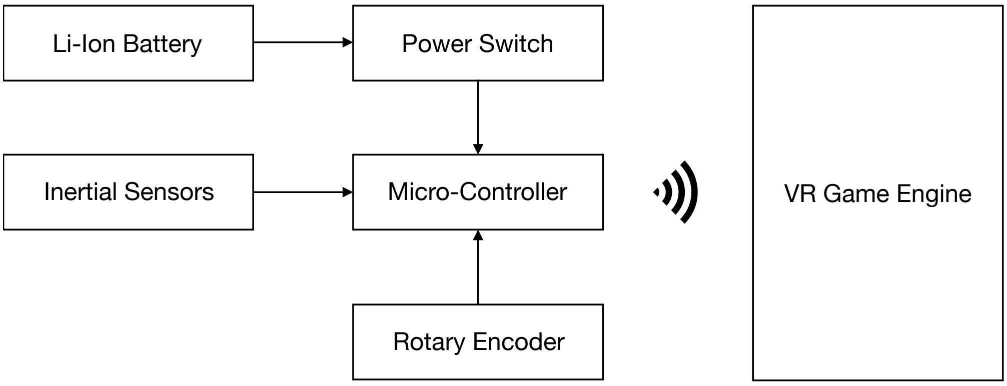


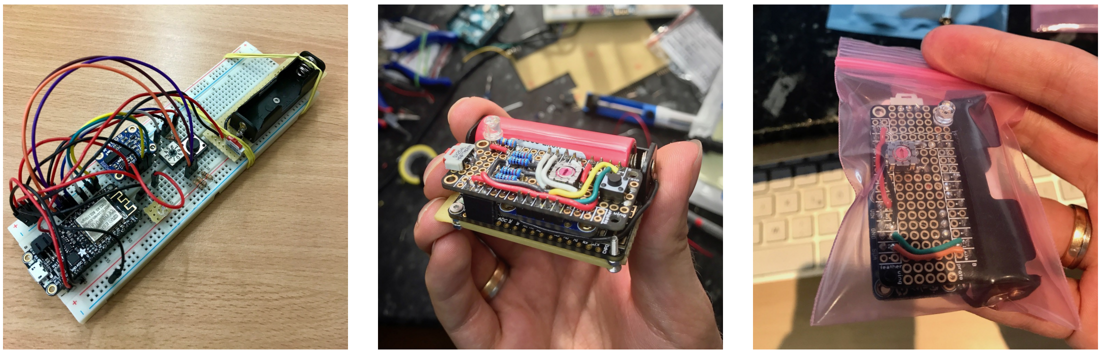
Figure 5.10: Schematic overview of the inertial orientation tracker.

### Six-Degrees-of-Freedom (6DoF) Tracker

The features of the active IR marker and the inertial orientation tracker are combined into one single *six-degrees-of-freedom (6DoF)*[2] *tracker*. In accordance with the requirements, this appliance needs to be small, light and easy to operate. As a result, minimizing the outer measurements of the device and choosing very small components are chief concerns. *Fig. 5.11* documents the evolutionary stages of the wearable 6DoF tracker.

---

[2] The six-degrees-of-freedom relate to the 3D position and the 3D orientation of the wearable device.



(a) Proof of concept.

(b) First prototype.

(c) Final prototype.

Figure 5.11: Evolutionary stages of the wearable 6DoF tracker.

The first stage was an initial proof of concept that helped to evaluate the suitability of the applied components. The second stage took several findings into account. For instance, some components were still too large and needed to be replaced. The final stage streamlined the unit's complexity. Some features were omitted during this step because they exceeded the requirements. *Fig. 5.12* depicts the circuit diagram of the final prototype.

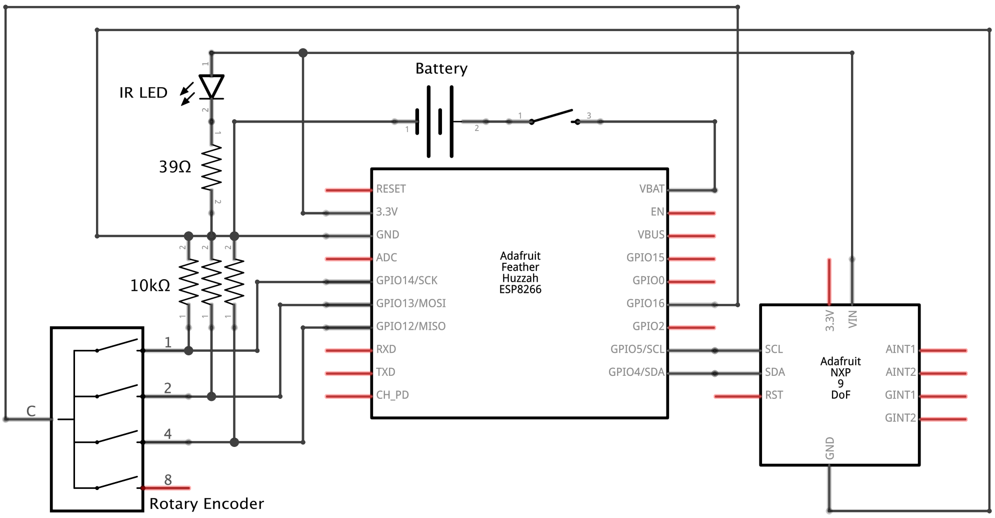
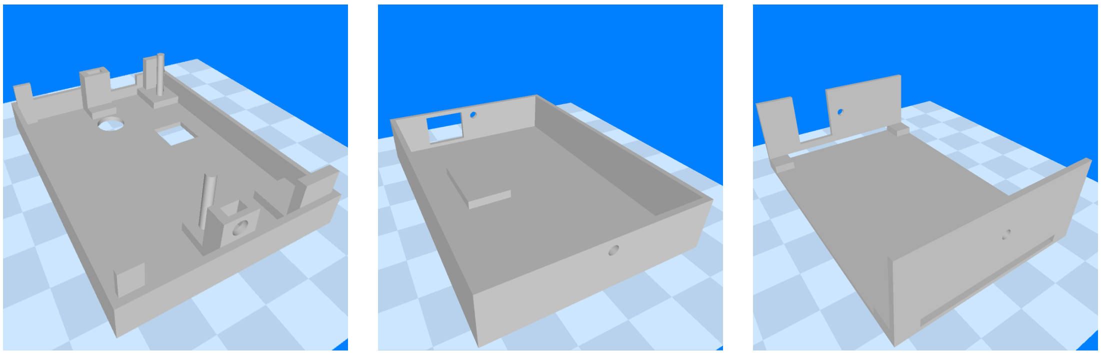


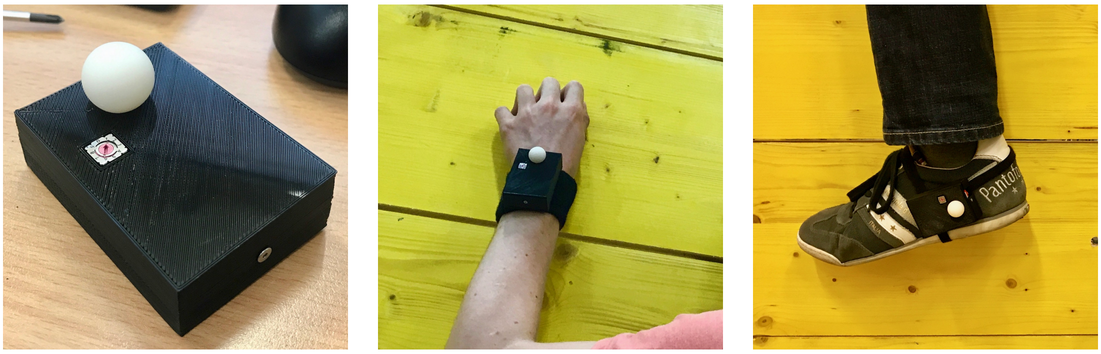
Figure 5.12: Circuit diagram of the 6DoF tracker.

As a next step, a 3D model of the tracker's enclosure was developed. It consists of a top and bottom element, as well as a universal bracket. *Fig. 5.13* illustrates the designs. *Fig. 5.14* shows the final enclosure after 3D printing and assembling. For convenience, the brackets of the hand trackers are sewed directly into sweatbands. The foot markers are attached to the user with velcro straps.



(a) Top element. (b) Bottom element. (c) Bracket element.

Figure 5.13: 3D models of the tracker's enclosure.



(a) Tracker enclosure. (b) Hand tracker. (c) Foot tracker.

Figure 5.14: Final 6DoF tracker.

### Inertial Sensor Calibration

Inertial sensors need to be calibrated to provide reliable measurements. The sensor model in use applies factory calibrated gyroscopes and accelerometers. Only the magnetometer needs to be calibrated by the operator. Since the magnetometer is affected by varying disruptive factors in close proximity, it needs to be re-calibrated before each tracking session.

The VreeTracker uses Stoffregen's motion sensor calibration tool called *MotionCal* [74].[3] The 6DoF tracker's ID needs to be set to channel 0. At this configuration, the wearable tracker capsules the raw sensor data in UDP packets and transmits them wirelessly over the local network to a specifically defined UDP port. The orientation data is encoded in a simple message that has the following form:

$$"Raw :< accel_{raw} >, < gyro_{raw} >, < mag_{raw} >; "$$

---

[3] The calibration process has been described in *Ch. 3.2.3*.

It is important to notice that the sensor calibration tool expects the sensor data as a serial data stream. Therefore, the network packets are sent to an additional microcontroller that receives the UDP packets, translates them into serial data and forwards the information to MotionCal. *Fig. 5.15* illustrates the data pipeline.

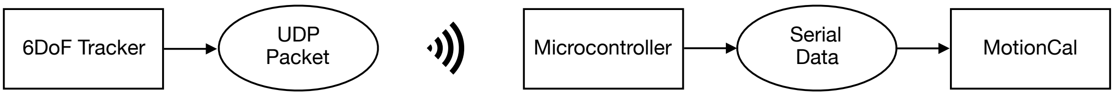


Figure 5.15: Data pipeline of the inertial sensor calibration.

## 5.2 Software Development

### 5.2.1 Position Tracking

The position tracking component calculates the markers' 3D position from the stereoscopic camera views and broadcasts this information over the local network. The source code is completely written in *Matlab*, a proprietary programming language licensed by *MathWorks* [57]. Matlab provides an *image processing toolbox* that facilitates the development of the prototypical software component. Furthermore, Matlab supplies a *support package for USB webcams* [58] that allows the algorithm to control the webcams' camera parameters and to access the live camera feeds.

#### Class Diagram

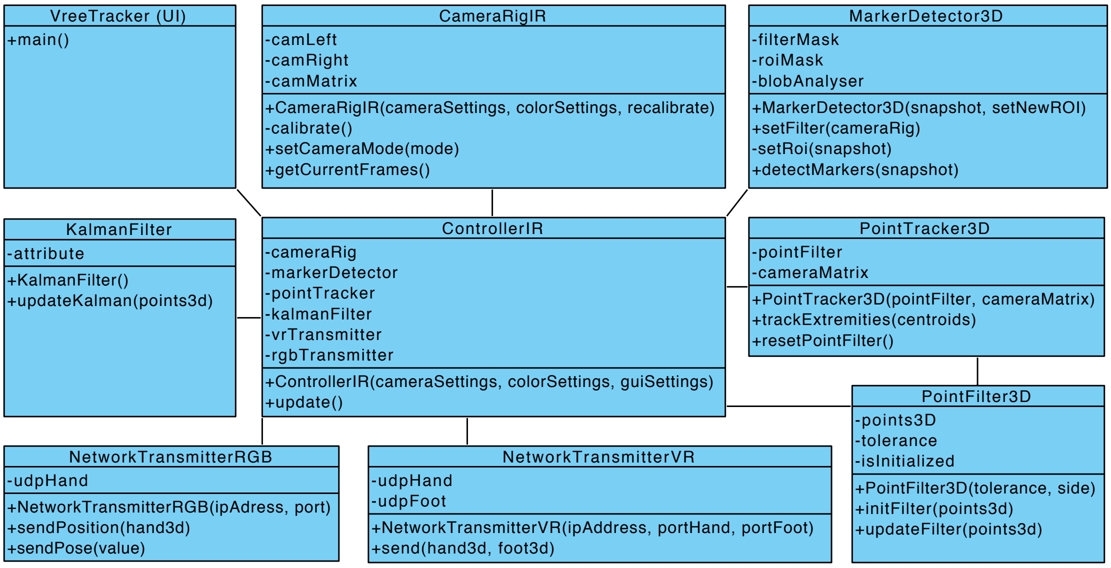


Figure 5.16: Class diagram of the position tracking component.

*Fig. 5.16* illustrates the class diagram of the position tracking component. The starting point of the application is the *VreeTracker (UI)* class. It calls the *ControllerIR* class for every iteration of the tracking algorithm. The ControllerIR is responsible for the algorithmic process. It requests the current camera frames from the *CameraRigIR* class and forwards them to the *MarkerDetector3D* class which returns a list of detected 2D centroid positions. The controller then hands this list to the *PointTracker3D* class which calculates the 3D positions of the markers and keeps track of them individually with the help of the *PointFilter3D* class. The labeled 3D positions are passed to the *NetworkTransmitterVR* and *NetworkTransmitterRGB* classes which transmit the position data to their designated receivers. The position data is also handed to the *KalmanFilter* class which interpolates a new set of position values. Like before, this estimated dataset is then forwarded to the transmitter classes.

### User Interface

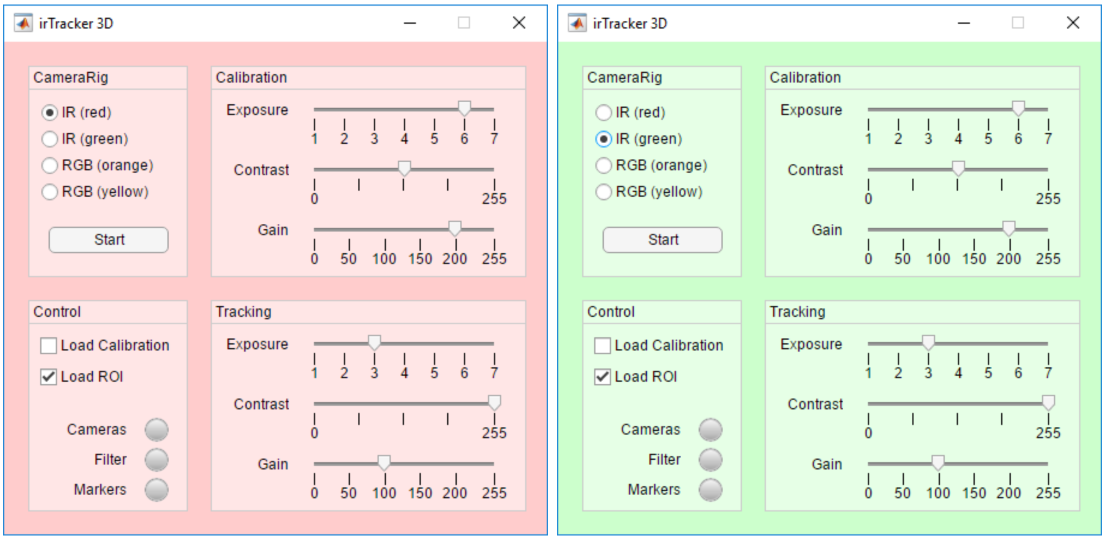


Figure 5.17: Graphical User Interface (GUI) of the optical position tracking component.

VreeTracker's position tracking component can be controlled via two different user interfaces, a *GUI* and a *script-based user interface*. As shown in *Fig. 5.17*, the GUI offers an intuitive way to interact with the application. It implements the following four features:

- *Camera Rig:* As mentioned before, each camera rig runs on its own host computer. The *Camera Rig* section lets the operator specify the connected camera rig. As a result, certain parameters that vary between camera rigs are loaded by the application.

- *Camera Mode:* The tracking algorithm consists of two stages. In the *calibration* stage, the extrinsic parameters of the camera rig are calculated. In the *tracking* stage, the tracking algorithm is executed until the user stops the operation. Both stages apply different camera settings that can be set independently by the operator in the calibration and tracking sections.
- *Calibration Control:* The *control* section provides additional options for the calibration process. A checkbox determines if the calibration settings are loaded from a file or if the application should re-calibrate the optical tracking system. Similarly, a second checkbox decides if the *Region of Interest (ROI)* is loaded from file or should be re-selected by the operator. The ROI determines the area in front of the VreeClimber in which the user should be detected.
- *Status Signals:* Three signal elements indicate the status of the tracking algorithm *(see Fig. 5.18).* An orange signal suggests that the algorithm is currently performing the corresponding step. A green signal indicates that the relating step has been completed successfully. Accordingly, a red signal reports an error. Steps that have not been started yet are marked by a grey signal. If all three elements show green signals, the algorithm has reached the tracking stage.

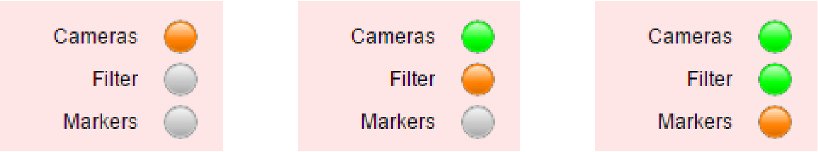


Figure 5.18: Status signals of the position tracking algorithm.

The script-based user interface provides the same functionality as the GUI. The application applies predefined parameters and runs automatically after starting the script. If necessary, these parameters can be adjusted directly in the script. Instead of showing graphical signals, the script prints the current status to the console.

### Calibration Stage

After starting the application, the algorithm calibrates the position tracking system. This stage consist of the *extrinsic camera calibration*, the *ROI selection* and the *IR interference filtering*.

***Extrinsic Camera Calibration:*** As a first step, the algorithm calculates the extrinsic parameters of the IR cameras. Similar to the intrinsic calibration, a checkerboard pattern is used to determine the relative position and orientation of the cameras. In contrast to the intrinsic calibration, neither the pattern nor the cameras are moved during the calibration process. Consequently, a single image from each camera does suffice. It is important to notice that both camera rigs need to be calibrated to the identical checkerboard and that the pattern must not be moved until both camera rigs are fully calibrated. As a result, the position of the checkerboard's upper left corner defines the spatial origin for both camera rigs *(see Fig. 5.7b)*.

The calibration pattern only reflects a small amount of surrounding IR light and does not emit any IR light directly. Therefore, the camera parameters need to be adjusted. The exposure, contrast and gain settings are set to a level where the cameras can capture the pattern clearly. *Fig. 5.19* illustrates the resulting images.

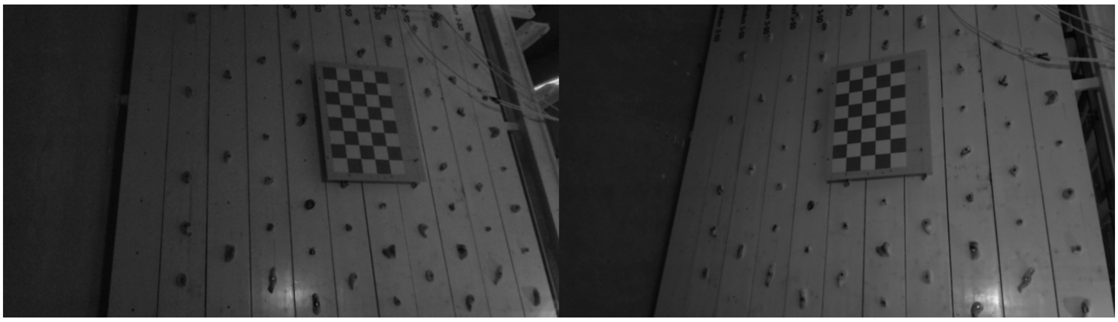


Figure 5.19: Stereoscopic IR camera views during the calibration stage.

***ROI Selection:*** The images from the extrinsic camera calibration are sequentially presented to the operator, so that the ROI can be selected *(see Fig. 5.20)*. The operator simply marks the corners of the designated tracking area with the mouse cursor. Every image pixel outside the ROI is set to black. Thereby, potentially disruptive factors outside the designated tracking area are removed.

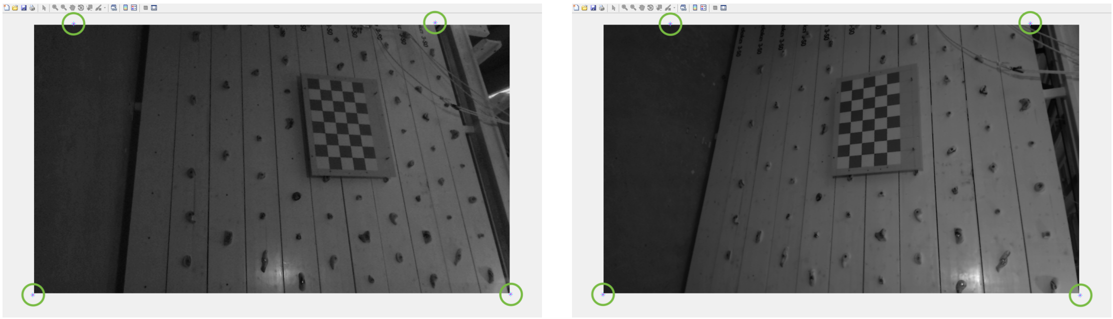


Figure 5.20: ROI selection.

***IR Interference Filtering:*** IR interferences within the tracking area also need to be filtered. The camera parameters are set to a level where only strong ambient IR light and IR emitting light sources are visible to the image sensors. The calibration algorithm automatically detects IR interferences by assuming that every detection of IR light during the calibration stage constitutes an interference.

The empty tracking area is captured for several seconds and a stereoscopic binary image is calculated from every camera frame. The foreground pixels in the binary image correspond to detections of IR light. The foreground pixels are dilated slightly and summed up across all image frames. By inverting the resulting stereoscopic binary image, the detected interferences are expressed as small black spots on an otherwise white canvas. *Fig. 5.21* depicts the stereoscopic filter mask that combines the ROI selection with the IR interference filter.

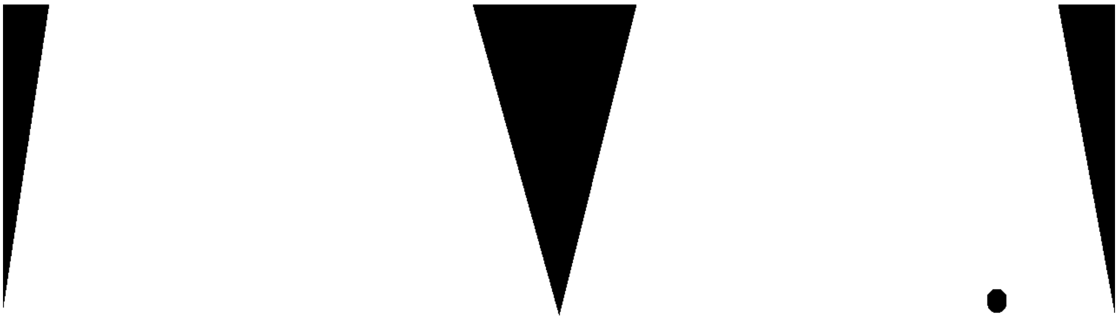


Figure 5.21: Filter mask that combines the ROI selection with the IR interference filter.

During the tracking stage, every stereoscopic image frame is multiplied with the filter mask. As a result, the area outside the ROI and the static IR interferences are filtered successfully.

### Tracking Stage

If the system calibration was successful, the application enters the tracking stage. The following section describes the algorithmic process which is orchestrated by the *IR Controller*. The controller class acts as a link between the user interface and the business logic.

***Marker Detection:*** After requesting the current image frames from the stereoscopic camera group, the controller instructs the *Marker Detector* to perform the blob detection. The detector binarizes the IR images at a very low threshold to separate the markers from the dark background. In addition, it filters the previously detected IR interferences and removes all blobs that lie outside the ROI. Consequently, all remaining blobs can be safely classified as markers. The marker detector calculates the centroid of every remaining blob in the stereoscopic image and stores these 2D position values as *stereoscopic list entries*.

***Projective Triangulation:*** As a next step, the projective triangulation needs to be performed by the *Point Tracker*. In this context, solving the stereo correspondence problem is essential. Based on the fact that each camera rig only tracks one side of the user's extremities, this challenge can be handled with a *brute force approach*.

Depending on the camera rig's side, the point tracker selects the two left (or right) centroids from both camera frames and calculates the projective triangulation for all four combinations. Wrong combinations can then be excluded by detecting two types of defects:

- *High Reprojection Error:* A wrong combination of stereoscopic centroids typically shows a high reprojection error. The point tracker excludes all combinations that exhibit a reprojection error of more than a few millimeters.
- *Invalid 3D Position:* If two stereoscopic centroids lie on the same epipolar line by coincidence, they exhibit a negligible reprojection error despite being unrelated. However, their projective triangulation results in a 3D position that lies outside the tracking area. For instance, the calculated 3D position might lie behind the climbing wall. These combinations can also be excluded.

***Marker Allocation:*** The calculated 3D positions need to be allocated to their corresponding limbs. The *Point Filter* distinguishes two possible scenarios. If the markers have been detected for the first time, then the point filter assumes that the upper point relates to the hand and the lower point corresponds to the foot. These position values are stored for future reference. In subsequent iterations, the point filter compares the new 3D points to their last known positions. If the distance between the old and the new point falls below a tolerance threshold, the 3D point can be allocated and the stored value is updated. Consequently, if a specific marker is temporarily occluded during tracking, the point filter expects it to reappear in close proximity.

***Predictive Filtering:*** Every calculated 3D position value is transmitted to a *Kalman Filter* which predicts a new position value in-between two consecutive camera frames. This *motion-compensated frame interpolation* doubles the update rate of the optical tracking system and guarantees smooth motions.

***Data Transmission:*** The labeled position data is broadcasted over the local network by the *Network Transmitters*. Similar to the inertial sensor calibration process, each 3D position is encoded in a message and encapsulated in a UDP packet. The *VR Transmitter* sends the position data of the hand and foot to the VR game engine. By using pre-assigned UDP ports, the game engine can easily allocate the data to the corresponding extremity.[4] The *RGB Transmitter* sends the hand position data to the hand pose estimation component.

---

[4] As an example, the left hand position data is transmitted over UDP port 2005 and the left foot position data is transmitted over UDP port 2006.

### 5.2.2 Hand Pose Estimation

The *hand pose estimation component* determines the opening width of the climber's hand. Initially, it was intended to integrate this feature in the position tracking component. Due to performance issues, the modules needed to be separated. As a result, both components show many similarities. For instance, the hand pose estimation component is also written in Matlab, it consists of a calibration and a tracking stage, and has an analogous algorithmic structure. In addition, it also offers a GUI as well as a script-based user interface.

#### Class Diagram

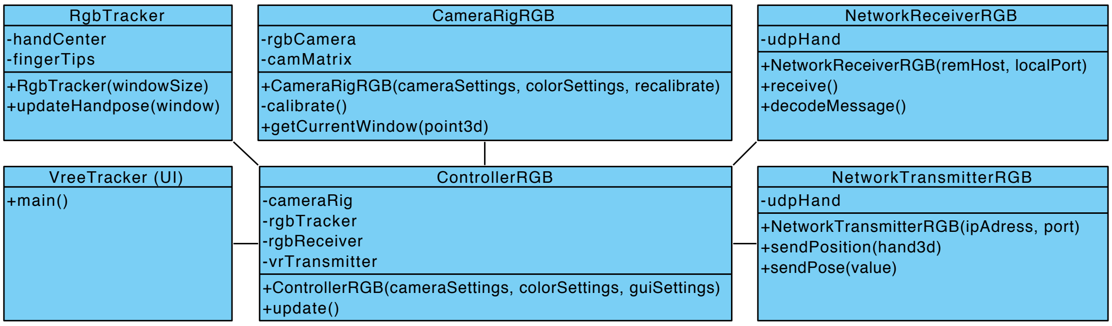


Figure 5.22: Class diagram of the hand pose estimation component.

*Fig. 5.22* illustrates the class diagram of the hand pose estimation component. Similarly to the position tracking component, the entry point of the application is the *VreeTracker (UI)* class. At startup, it initializes the *ControllerRGB* class, which then instructs the *NetworkReceiverRGB* class to wait for incoming UDP messages. As soon as a new packet arrives, the receiver extracts the position data from the message and hands the information to the controller. The controller requests the RGB window around the received 3D point from the *CameraRigRGB* class and forwards the image to the *RgbTracker* class. The RGB tracker processes the image and returns a numeric *hand opening width* parameter to the controller. The controller forwards the value to the *NetworkTransmitterRGB* class which sends the pose information to the VR game engine.

#### Calibration Stage

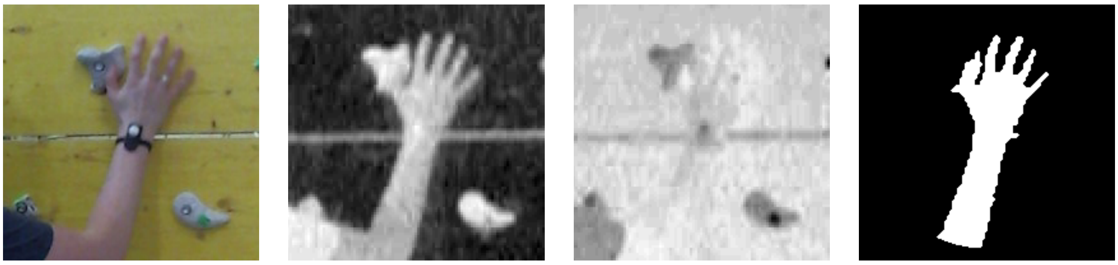
In the calibration stage, the application performs the extrinsic calibration of the RGB camera. The webcam is calibrated to the identical pattern as the IR cameras. This fact becomes relevant in the tracking stage. It is important that the checkerboard is not moved between the calibration of both components. Despite them running on different host computers, they should be considered a single unit.

### Tracking Stage

***Data Reception:*** When the application enters the tracking stage, it waits for incoming UDP packets from the position tracking component. The expected packets contain the 3D position of a single hand. As soon as a packet arrives, the *Network Receiver* extracts the position data from the message and forwards the information.

***Window Selection:*** The *RGB Controller* uses the 3D hand marker position to narrow down the search area for the hand. As mentioned before, all cameras are calibrated to the same checkerboard pattern. Therefore, the 3D position of the hand marker can be translated directly to an image pixel in the RGB image. The controller requests a small image section around the received 3D position from the *Camera Rig* and sends it to the *RGB Tracker*. This image detail is henceforth called window.

***Skin Segmentation:*** The RGB tracker is responsible for estimating the hand's opening width. The implemented algorithm is largely inspired by the work of *Yeo et al.* [79]. As a first step, the hand must be detected in the color image. The algorithm transforms the RGB window into the $YC_bC_r$ color model to detect the skin of the climber's hand. As *Fig. 5.23* shows, the $C_b$ color channel successfully separates the hand from the climbing wall, but also highlights the climbing grips and the user's clothing. In contrast, the $C_r$ color channel highlights everything except the climbing grips and the clothing. After combining both color channels and performing some minor morphological operations, the hand can be separated from the background efficiently.



(a) RGB image. (b) $C_b$ color channel. (c) $C_r$ color channel. (d) Binary image.

Figure 5.23: Comparison of the original image with the $YC_bC_r$ color channels.

***Palm Detection:*** Several natural landmarks, also known as feature points, are detected in the binary image. First of all, the center of the palm acts as a base point. It can be determined by selecting the foreground pixel with the highest Euclidean distance to any background pixel.[5] The palm radius, which is defined as the Euclidean distance from the palm center to the nearest background pixel, is the

---

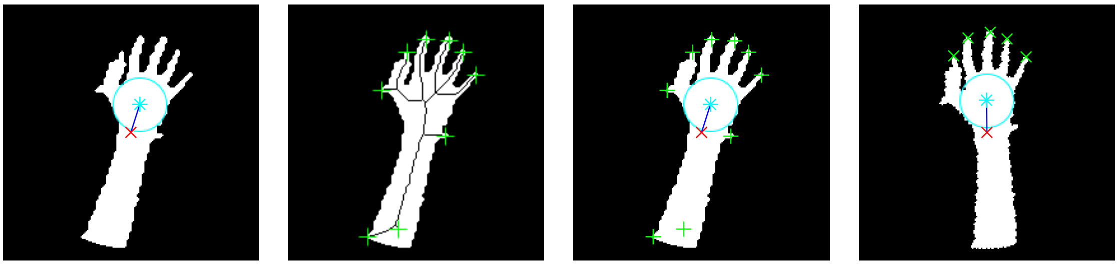
[5] This assumption might be incorrect if the arm runs through a corner of the image. Therefore, a binary circular mask is used to remove the corners from the image. When inspecting *Fig. 5.23d* closely, the edge of the circular mask can be discovered at the bottom of the image.

second important feature point. The line segment between the palm center and the IR marker[6] defines the orientation of the arm. This detail will be important in the next step. *Fig. 5.24a* illustrates these palm features.

***Fingertip Detection:*** The fingertips need to be detected in the window. The algorithm uses the morphological *thinning* operation to reduce the hand to a network of single-pixel lines *(see Fig. 5.24b)*. The endpoints of this network represent the humps of the foreground object *(see Fig. 5.24c)*. To exclude false positive detections, the hand is rotated counter clockwise by the orientation of the arm.[7] As a result, all fingertips lie above the palm center regardless of the original orientation of the arm. By eliminating all endpoints below the palm center, only the fingertips remain. *Fig. 5.24d* shows the final result.

***Hand Pose Estimation:*** The VreeTracker expresses the hand pose as the opening width of the hand. In the context of this thesis, the opening width is defined as the median distance from the palm center to the fingertips minus the palm radius. Thereby, a closed fist has an opening width of almost zero. The maximum opening width is stated in pixels.

***Data Transmission:*** Like the 3D position data, the hand pose value is transmitted to the VR game engine in form of UDP packets.



(a) Palm features. (b) Morph. thinning. (c) Endpoints. (d) Fingertips.

Figure 5.24: Feature points detection.

### 5.2.3 Orientation Tracking

The *orientation tracking component* reads the raw sensor data from the inertial sensor array, fuses the readings into a 3D orientation value and transmits this information wirelessly over the local network to the VR game engine. Since the 6DoF tracker uses an Arduino-compatible micro-controller, the source code of the component is written in the

---

[6] Since the window is selected around the hand marker, it is per definition located at the center of the binary image.

[7] In fact, only the feature points are rotated by the orientation of the arm. Rotating the binary image would be computationally expensive and is performed for demonstration purposes only.

Arduino language [7]. The orientation tracking algorithm is very simple. It is composed of the following stages:

***Initialization:*** After switching on the 6DoF tracker, several hardware components are initialized. The magnetometer is calibrated based on the values from the *inertial sensor calibration*. The micro-controller connects itself to the local wireless network. Finally, the tracker's ID is read from the rotary encoder.

***Tracking:*** As soon as the hardware components are set up, the algorithm starts reading raw sensor values from the inertial sensors. The data is forwarded to the *Mahony Filter* which calculates the 3D orientation from the raw sensor readings and outputs the information as quaternions.

***Data Transmission:*** Like the 3D position data and the hand pose, the 3D orientation data is encoded in a message, encapsulated in a UDP packet, and sent to the VR game engine over the local network. The tracker ID determines the UDP port of the transmission.

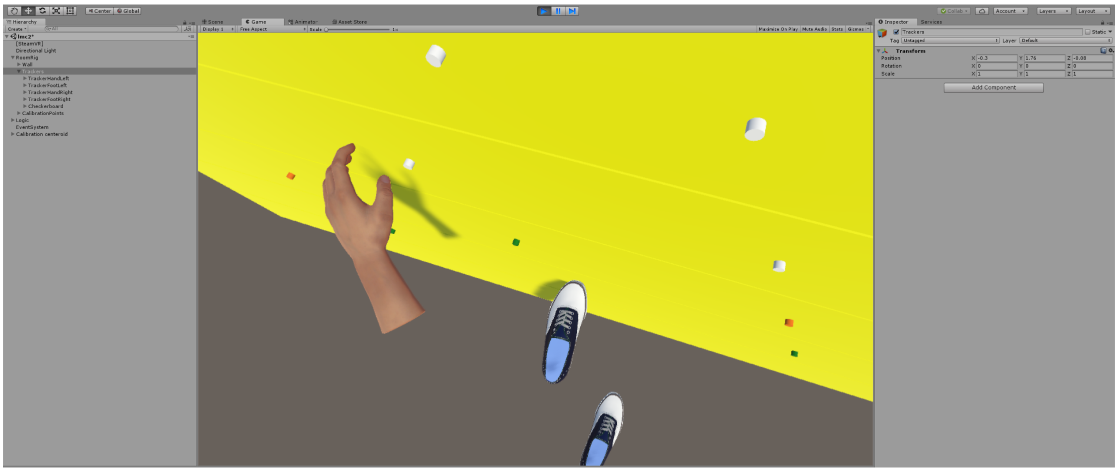
## 5.3 Virtual Environment

The VreeClimber utilizes *Unity* [76] as its VR game engine. Generally speaking, the game engine forms the link between the real world and the virtual environment and is the central component of every VR application. It serves as a tool to create the virtual content and lets the climber immerse in the artificial world. Furthermore, it applies the VreeTracker's motion data to control the virtual avatar. *Fig. 5.25* illustrates the GUI.

In the course of implementing the VreeTracker, a rudimentary environment was developed to demonstrate the tracking system. Simple hand and shoe models mimic the movements of the climber. The following features are realized by the game engine:

***Data Reception:*** A multi-threaded UDP receiver decodes incoming messages from the VreeTracker and translates the information into position, orientation, and hand pose values. Every data stream runs in its own thread and is expected on an individual UDP port. As a result, ten concurrent data streams can be processed simultaneously.

***Model Control:*** Every data stream manages one aspect of a single object. Each shoe model is operated by two data streams. One controls the position of the model, the other one determines its orientation. In addition, each hand model receives the hand pose information via a third data stream.



☑ Trackers ☐ Static
Tag Untagged Layer Default
Transform
Position X -0.3 Y 1.76 Z -0.08
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Figure 5.25: Unity's GUI.

## 5.4 Conclusion

This chapter discussed the implementation of the VreeTracker. It elaborated on the hardware modifications that converted four common RGB webcams into a set of two stereoscopic IR cameras rigs and described the development of the wearable trackers. Furthermore, the algorithmic interplay of the tracking processes was illustrated and the interface between the VreeTracker and the VR game engine was outlined.

## CHAPTER 6

# Evaluation

After concluding the implementation of the VreeTracker, the quality of the motion capture system is evaluated. To ensure an objective assessment of the results, the *HTC Vive* is used as a state-of-the-art reference system. Common spatial and temporal performance metrics form the core of the analysis. Furthermore, the robustness of the VreeTracker is inspected. Finally, the performance of the hand pose estimation component is discussed.

## 6.1 Setup and Methodology

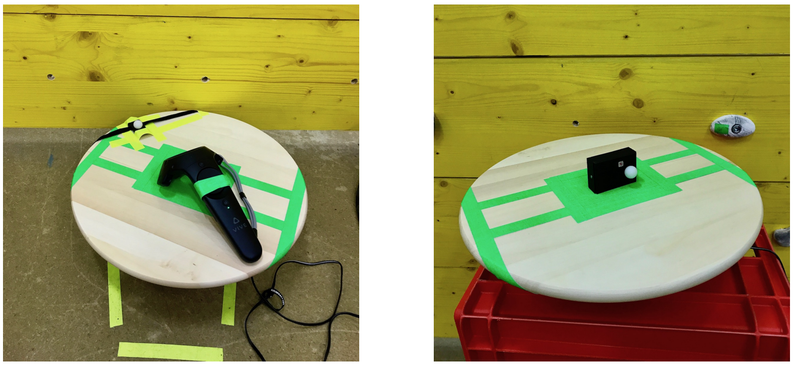
### 6.1.1 Spatial Performance Metrics

*Accuracy* and *precision* form two major performance metrics of the tracking system. The positional accuracy is determined by placing an IR marker in a resting position and comparing the calculated 3D positions from both camera rigs.[1] The positional precision is verified by tracking a resting marker for the duration of one minute and analyzing the recorded data. The orientational tracking component is evaluated similarly. The 6DoF tracker is placed on a resting platform and the angular output over time is inspected. Since there is no technique at hand that is capable of measuring the angular accuracy within the sub-degree range, this metric should be evaluated qualitatively in a subsequent project.

Furthermore, the behavior of moving tracking components is evaluated. A rotating platform, which the author of this thesis developed as part of a side project, ensures a consistent and repeatable movement. By placing the IR marker on the edge of the platform *(see Fig. 6.1a)*, a circular movement is performed and the resulting position data can be recorded. If the 6DoF tracker is placed at the center of the platform *(see Fig. 6.1b)*, the continuous rotation around its own axis is registered.

---

[1] By definition, the output of both stereoscopic camera systems should be identical since they are calibrated simultaneously to the same checkerboard pattern.



(a) Positional setup.

(b) Orientational setup.

Figure 6.1: Marker setup during spatial performance evaluation.

### 6.1.2 Temporal Performance Metrics

The *update rate* and the *latency* are important temporal metrics that need to be evaluated. The update rates of the tracking components are predetermined by the update rates of the position tracking and sensor fusion algorithms. The latency is calculated by taking timestamp measurements in-between individual steps of the corresponding algorithm.

### 6.1.3 Tracking Robustness

The tracking robustness is affected by two potential issues. Firstly, occlusions are a chief concern with any optical tracking system. In the case of the VreeTracker, the climber's hands might be obscured by the torso. Therefore, the borders of this *dead zone* are experimentally established and marked at multiple positions of the climbing wall *(see Fig. 6.2)*. Secondly, potential packet loss during the network-based data transmission is analyzed by sending a predefined number of data packets over the wireless network and counting the number of received packets at the second endpoint.

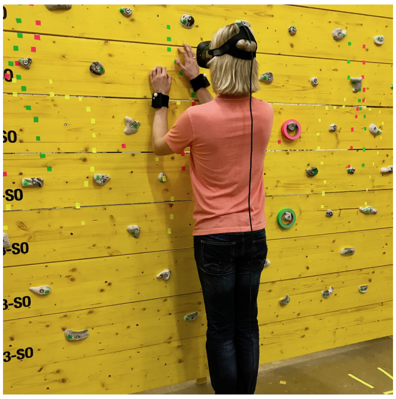


Figure 6.2: Subject determining the area of occlusion in front of the body.

## 6.2 Spatial Performance

### 6.2.1 Positional Accuracy

The mean distance between the calculated 3D positions of both camera rigs is $3.2mm$. This value corresponds to an accuracy of $\pm 1.6mm$. Further analysis reveals that the mean horizontal and vertical distances between both data sets are only $1.4mm$. However, the distance along the depth axis amounts to $2.6mm$. These values indicate that the depth calculation of the position tracking component operates slightly worse than the horizontal and vertical position tracking. In comparison, the HTC Vive shows an accuracy of $\pm 1mm$ [37].

### 6.2.2 Positional Precision

The jitter along the horizontal, vertical and depth axes amount to $\pm 0.4mm$, $\pm 0.26mm$ and $\pm 0.75mm$. These values result in a 3D jitter of $\pm 0.8mm$. Again, the depth calculation of the optical tracking system marginally deteriorates the overall precision. According to [37], the Vive shows a precision of $\pm 0.15mm$. However, tested under the same condition as the VreeTracker, the HTC Vive controller appears to have a jitter of $\pm 0.88mm$, $\pm 0.68mm$ and $\pm 0.42mm$ along the horizontal, vertical and depth axes. Puzzled by these contradicting statements, the distribution of the individual measurements is investigated. As *Fig. 6.3* shows, most measurements of the VreeTracker's position tracking component accumulate in 2-3 discrete values. This surprising effect is probably caused by a rounding error during the blob detection when the 2D centroid positions of the blobs in the 2D image views are calculated. In contrast, the Vive's measurements are distributed continuously *(see Fig. 6.4)*. As the second figure further shows, most measurements of the Vive's tracking system are accumulated in a small neighborhood. Still, based on the available data, the precision value stated by [37] cannot be re-created.

*Fig. 6.5* illustrates the position data of the IR marker on the rotating platform. *Fig. 6.5a* depicts the data recorded during a horizontal rotation. It corresponds to a horizontal projection which omits the vertical axis by definition. *Fig. 6.5b* illustrates the data recorded on a vertically rotating platform. Therefore, the second image conforms to a vertical projection that ignores the depth axis. As a result, the figure removes the depth component from the illustration. The first image confirms the VreeTracker's elevated jitter values along the depth axis. When comparing both circles, the line in *Fig. 6.5b* shows a sharper contour.

A close examination of the Vive's horizontal rotation *(see Fig. 6.6)* reveals two facts. On the one hand, the Vive provides a higher update rate than the VreeTracker which lets the data points appear as a solid line. Furthermore, the blue circle drawn by the Vive appears to suffer from less jitter than the VreeTracker. An overlay of both circles[2] proves the Vive's superiority in terms of precision while moving *(see Fig. 6.7)*. The reason for the Vive's poor accuracy results while resting remains unclear.

---

[2] To allow a direct comparison, both circles have been scaled to a radius of 1.

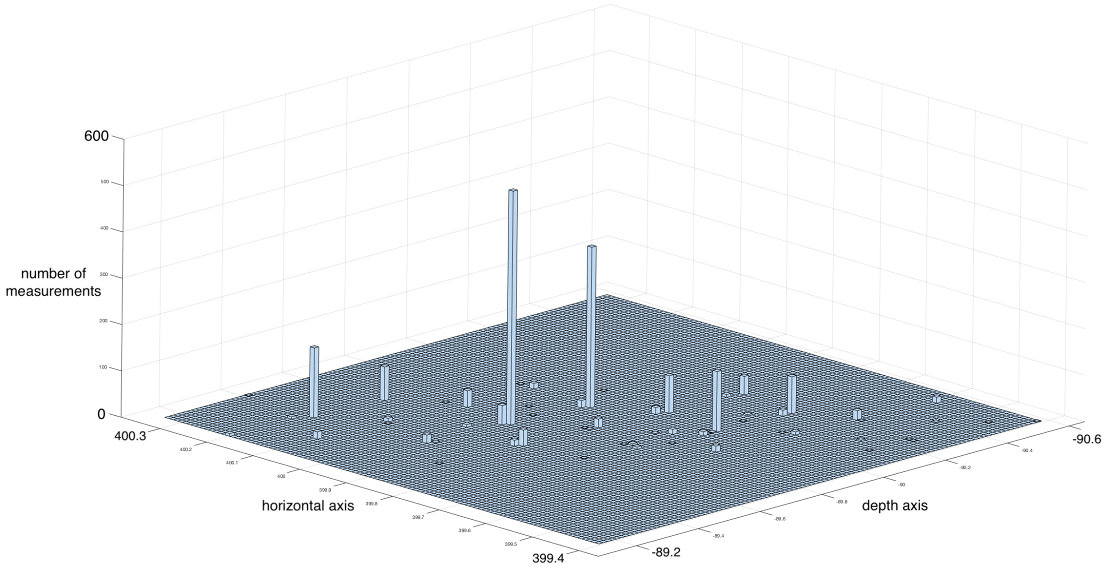


Figure 6.3: 3D histogram of VreeTracker's positional tracking data.

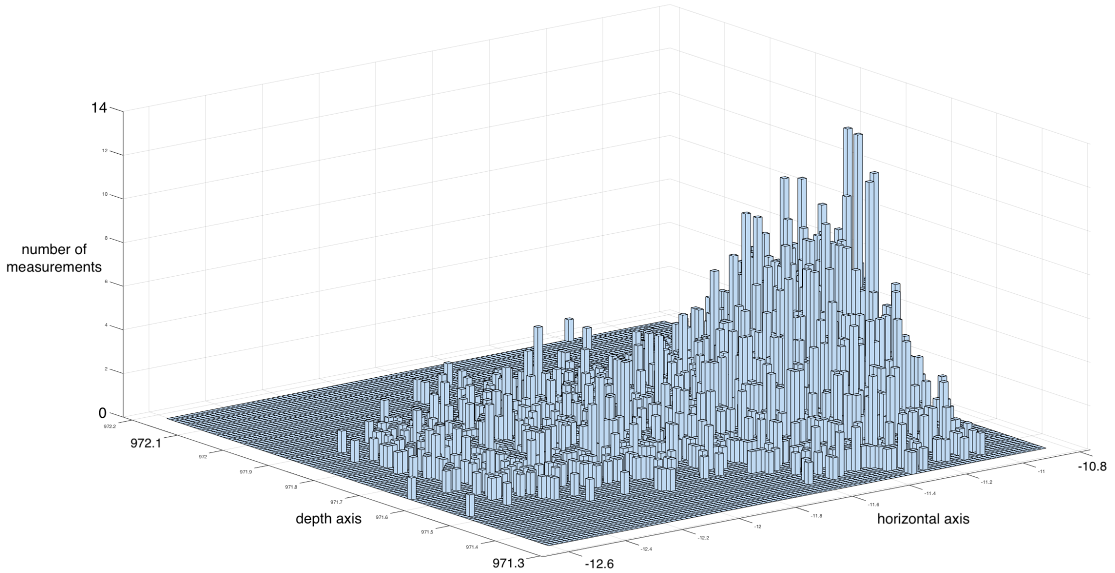
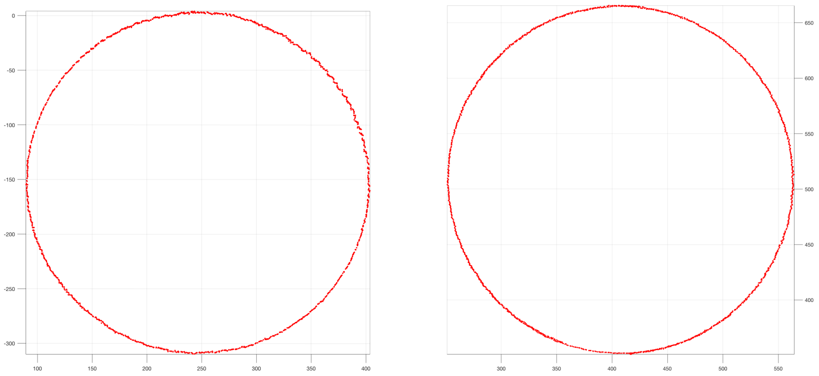


Figure 6.4: 3D histogram of Vive's positional tracking data.



(a) Horizontal rotation.

(b) Vertical rotation.

Figure 6.5: VreeTracker's positional data while moving on a rotating platform.

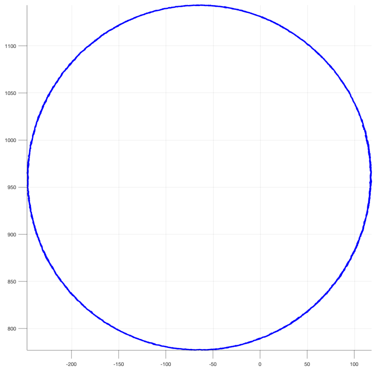
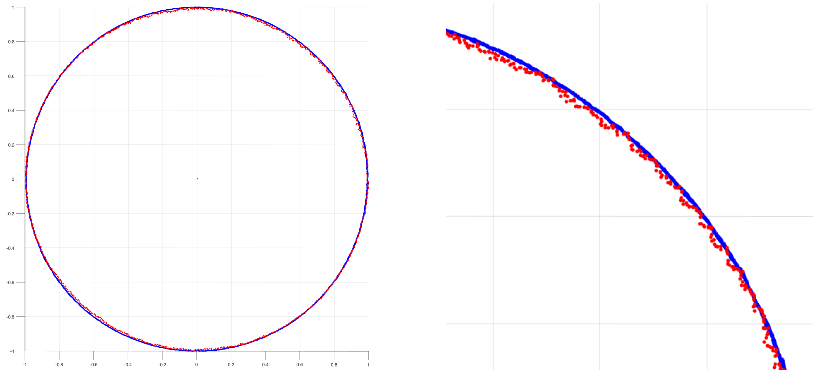


Figure 6.6: Vive's positional data while moving on a horizontally rotating platform.



(a) Overview.

(b) Section.

Figure 6.7: Comparison of the VreeTracker's and Vive's positional data.

### 6.2.3 Orientational Precision

The inertial orientation tracking component of the VreeTracker shows a precision of ±0.25°, ±1.15° and ±0.44° along the $x$-, $y$- and $z$-axes. In contrast, the Vive exhibits a precision of ±0.18°, ±0.12° and ±0.2°. As *Fig. 6.8* indicates, the VreeTracker's orientational jitter is more or less uniform along the time axis. The high jitter value illustrated by the green line might be caused by an electromagnetic field that interferes with the magnetometer readings. When analyzing the Vive's measurements over time *(see Fig. 6.9)*, it becomes clear that the precision result is influenced by a continuous drift along all three axes. The actual jitter appears to be even better than the measurement over the course of one minute indicates.

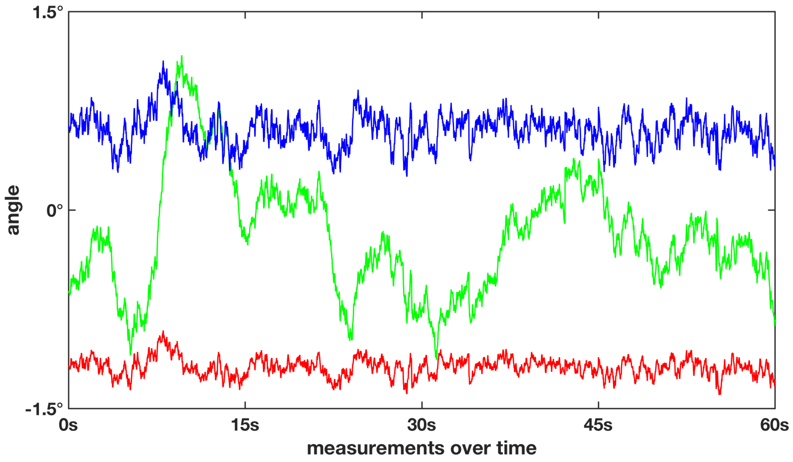


Figure 6.8: VreeTracker's orientational tracking data while resting still.

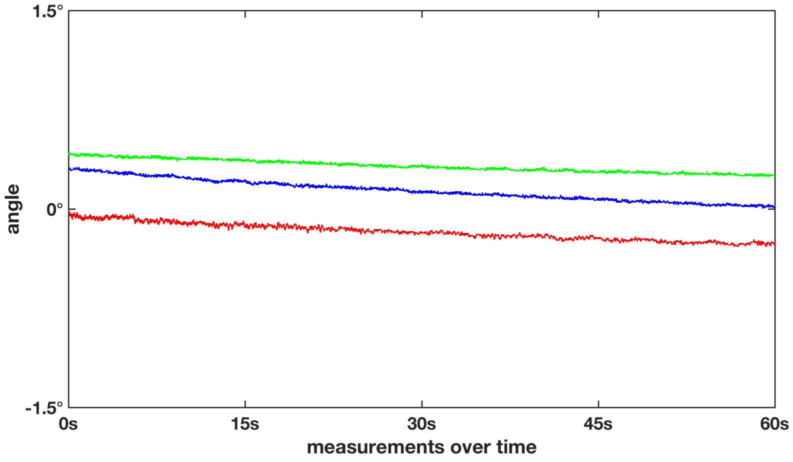


Figure 6.9: Vive's orientational tracking data while resting still.

The orientational tracking data of the moving, i.e. rotating, 6DoF tracker supports the hypothesis of the previous experiment. While the Vive's data shows a continuous rotation around the controller's axis *(see Fig. 6.11)*, *Fig. 6.10* indicates that the orientation tracking component of the VreeTracker might be influenced by an electromagnetic field. As the image illustrates, the tracking data suddenly drifts away from the expected course for several measurements before eventually returning to a regular state.

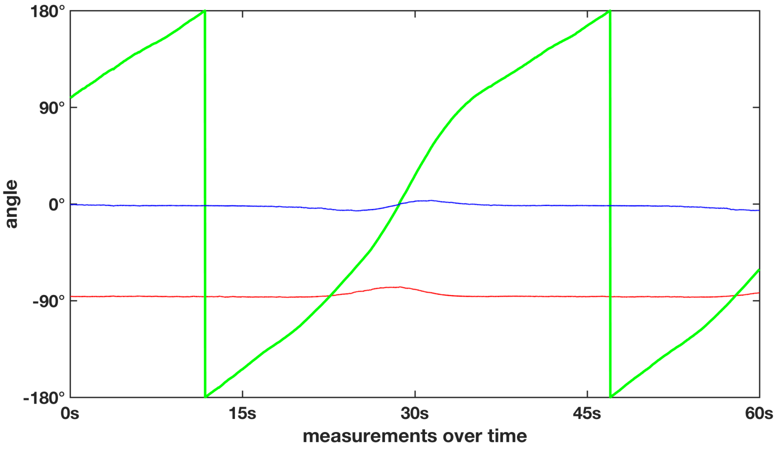


Figure 6.10: VreeTracker's orientational tracking data while rotating around an axis.

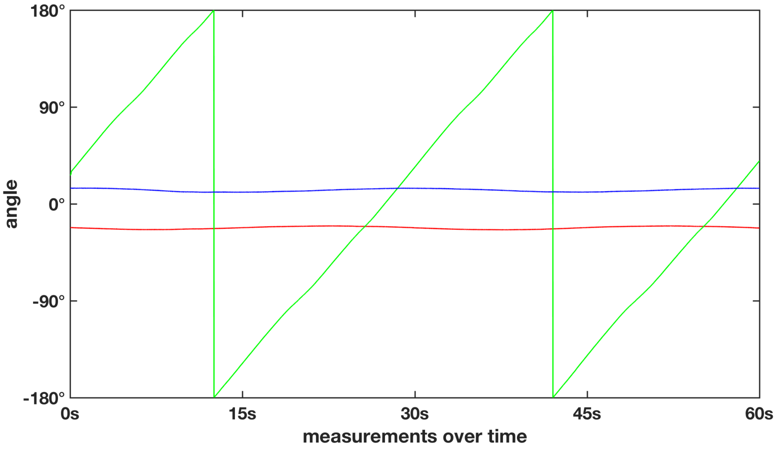


Figure 6.11: Vive's orientational tracking data while rotating around an axis.

## 6.3 Temporal Performance

### 6.3.1 Update Rate

The IR cameras used by the optical position tracking component deliver a frame rate of $30fps$. With the help of predictive filtering, the VreeTracker outputs 60 new position values per second. The orientation tracking component calculates approximately 75 values per second. In comparison, the controller of the HTC Vive delivers an update rate of $366Hz$ [37]. However, the attentive reader needs to bear in mind that the Vive's HMD and the VR game engine have a frame rate of only $90Hz$. Therefore, the high update rate of the controller primarily offers advantages in terms of data stabilization.

### 6.3.2 Latency

The VreeTracker's position tracking component has an algorithmic latency of approximately $7.5ms$. A closer examination reveals that the *blob detection* takes $6ms$ to calculate. Consequently, the remaining steps of the algorithm require only $1.5ms$ of time. However, since the cameras' frame rate is only $30Hz$, the algorithm waits for approximately $24ms$ before it receives new images from the cameras. The orientation tracking component requires $2ms$ to process the raw sensor data.

The second source of latency is the network transmission itself. The duration of a transmission roundtrip over the localhost is essentially zero. The roundtrip between two hosts connected over a wired network connection takes $1ms$ on a state-of-the-art gigabit router and $3ms$ on an entry-level 100-mbit device. A roundtrip over a wireless connection takes approximately $7ms$. Therefore, the transmission of the position data from the VreeTracker to the VR game engine is free of any additional latency as long as both applications run on the same host. If the game engine runs on a different host that is connected over wired ethernet, a latency of approximately $0.5ms$ is added due to the transmission over the network. Since the orientation data is transmitted wirelessly between the 6DoF trackers and the VR game engine, a latency of $3.5ms$ is added to the overall latency.

## 6.4 Robustness

### 6.4.1 Occlusions

If the climber places the hand in front of the torso, the line-of-sight between the IR marker and the IR cameras might be blocked. As *Fig. 6.12* illustrates, the area in which the marker is no longer detected depends on the user's position on the climbing wall. The yellow outline marks the contour of the climber's body. The green area indicates that a marker is detected by the right camera rig. The red area states that a marker is visible to the left camera rig. Consequently, markers that lie in the orange area are detected by both camera rigs. The black area marks the *dead zone* where both camera rigs lost their line-of-sight. When standing centrally in front of the wall, the dead zone is symmetrically in front of the torso and amounts to approximately one third of the shoulder width. When moving to the sides of the climbing wall, the dead zone shifts outwards. While the width of the dead zone decreases significantly on the right side, it remains unchanged on the left side. This might be caused by a slightly asymmetrical positioning of the camera rigs.

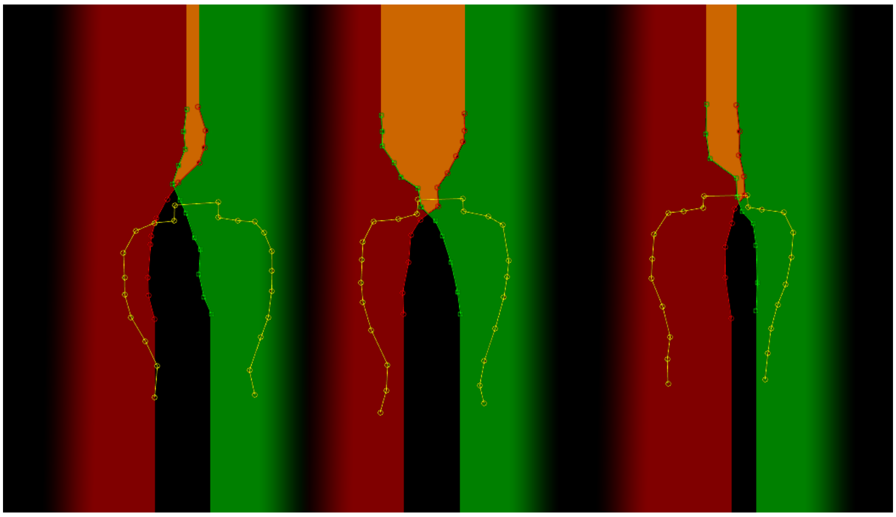


Figure 6.12: Areas of occlusion in front of the climbing wall.

As with any optical tracking system, Vive's tracking robustness strongly depends on the positioning of the lighthouses as well as the concrete application site. Extensive experimentation, during wich any feasible combination of lighthouse positions is tested[3], reveals one major drawback of the motion capture system. Due to the fact that the Vive currently only supports the simultaneous use of two lighthouses, maintaining a direct line-of-sight to all four extremities and the HMD is not possible. Even when performing regular and simple climbing motions, either the HMD or at least one of the limb trackers looses the direct line-of-sight at some point during the evaluation.

### 6.4.2 Packet Loss

Packet loss, especially when transmitting UDP packets wirelessly over the network, is one final source of error. In contrast to TCP, the communication over UDP does not involve any error control. 60 UDP packets per second are sent wirelessly from a 6DoF tracker to the VR game engine for ten minutes. In total, 36.000 data packets are transmitted without any packet loss.

---

[3] Tested configurations include multiple setups where both lighthouses are above, below, or laterally next to the climber, a diagonal line-up where one lighthose is above and the other one is below the user as well as a constellation where both lighthouses are situated behind the participant.

## 6.5 Hand Pose Estimation Component

Although the hand pose estimation component showed promising results during development and early testing, it turned out that the image processing algorithm is not efficient enough to run in real-time. Consequently, the hand pose estimation component cannot handle the workload dictated by the position tracking component and rapidly falls behind. As a result, the latency of the hand pose data increases with every frame.[4] After one minute of tracking, the latency already amounts to over a second. Therefore, the hand pose estimation component is not operational at this point of development.

However, aside of this temporal constraint, the hand pose estimation algorithm performs admirably in terms of skin detection as well as the subsequent feature point identification. At this point, the color-based skin detection is only tested with caucasian participants. Deviating results are to be expected with different skin tones and should be investigated in future works.

With the help of the binary circular mask, the palm center is steadily detected by the algorithm and proves to be a stable anchor point. The palm radius successfully subtracts the size of a closed fist from the calculation. As a result, the difference between a closed fist and a fully opened hand is maximized. At least three different endpoints, i.e. fingers, are successfully recognized most of the time. Problematic situations which deteriorate the success rate occur during fast movements[5] and partial hand occlusions by foreign objects.

## 6.6 Conclusion

This chapter presented the evaluation of the VreeTracker. It objectively compared the VreeTracker's key metrics with a state-of-the-art motion capture system. As a conclusion, it can be stated that the VreeTracker shows satisfactory results in many concerns. Especially the quality of the optical position tracking component must be emphasized. As an unresolved issue, the temporal constraint of the hand pose estimation component has been portrayed.

---

[4] The increasing latency is caused by the fact that the pose estimation component attempts to determine a new hand pose for every hand position received from the position tracking component. Naturally, the pose estimation component could only process the latest data entry from the position tracker and discard the rest. However, the update rate would still be far too low for reasonably fast pose adaptions.

[5] Based on minor testing with different webcams, affordable cameras generally appear to suffer from image blurring during fast object movements.

CHAPTER 7

# Conclusions and Future Work

This thesis discussed the development of the VreeTracker, a hybrid motion capture system that detects a user's movements on a revolving climbing wall. Following the latest state-of-the-art, a novel solution was presented that not only combines optical position tracking with inertial orientation sensing, but also attempts to determine the climber's hand poses by a markerless approach. The resulting prototype is embedded in a VR climbing application that fuses the opportunities of a virtual climbing adventure with the haptic interaction on a real climbing wall.

All hardware components are based on highly affordable, easily available, off-the-shelf devices. Nevertheless, the evaluated performance metrics demonstrate that the VreeTracker is able to keep up with commercially available products in many regards. Especially the performance of the optical position tracking system exceeded the author's expectations. Understandable hardware deficiencies are successfully compensated by well-established algorithms. An independent double camera setup minimizes the area in which occlusions affect the robustness of the tracking application.

During the development phase, noteworthy restrictions induced by the applied consumer hardware emerged as multiple cameras were connected to the same host computer. Despite extensive testing, all efforts to run all four IR cameras on one computer proved futile. As soon as three or more cameras were connected via USB, the frame rate delivered by the webcams fluctuated increasingly. As a consequence, the 2D image views from the individual cameras did no longer match which rendered the precise 3D position calculation impossible. In the spirit of evolutionary prototyping, the system architecture was adapted and a distributed system was developed.

The greatest difficulties during hardware development were caused by the magnetometer. The initial proof-of-concept used a different motion sensor that outputted highly stable and very precise orientation data. However, as soon as all necessary hardware components were arranged in close proximity, the magnetometer was strongly affected by electromagnetic interferences. Since said sensor performed all calibrations autonomously, this issue could not be resolved satisfactorily. The AHRS that was ultimately used could not compete in terms of precision or output stability but was less vulnerable for interfering influences.

Unfortunately, the implementation of the markerless hand pose estimation component reached its limits when the software module was integrated in the overall process. The developed feature point detection algorithm was not efficient enough to achieve the expected update rate. Apart from this temporal performance constraint, the approach showed promising results and should be pursued further.

Due to the complexity of the chosen topic, this thesis can only establish a baseline within a state-of-the-art computer vision field. Especially the development of a stable markerless hand pose detection solution presents an interesting problem definition that will be the focus of many future projects. Without any claim of completeness, the following research questions come to mind:

**Color-based skin detection:** Does the color-based skin detection deviate among different skin tones? If so, can the skin detection algorithm autonomously calibrate itself to the current participant?

**Performance optimization:** How can the general processing performance of the hand pose estimation component be improved? Would a shape-based hand detection algorithm offer advantages regarding partial occlusion and motion blur handling?

**3D hand pose detection:** Is it feasible to estimate the hand poses with the help of stereoscopic depth cameras?

# List of Figures

1.1 Evolutionary stages of the VreeClimber. . . . . . . . . . . . . . . . . . . . . 2

2.1 Comparison of corresponding RGB and IR images *(different camera angles).* 6
2.2 Components of the ioTracker [68]. . . . . . . . . . . . . . . . . . . . . . . . 10
2.3 Constellation pattern on the Oculus Rift [29]. . . . . . . . . . . . . . . . . . 11
2.4 HTC Vive's Lighthouse. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.5 HTC Vive's HMD [28]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.1 The electromagnetic spectrum [24]. . . . . . . . . . . . . . . . . . . . . . . 15
3.2 Emission spectra of common light sources [71]. . . . . . . . . . . . . . . . 16
3.3 The pinhole camera model [59]. . . . . . . . . . . . . . . . . . . . . . . . . 17
3.4 Types of radial lens distortion [59] . . . . . . . . . . . . . . . . . . . . . . 19
3.5 Calibration pattern showing detected feature points. . . . . . . . . . . . . 20
3.6 The epipolar geometry [59]. . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.7 The Kalman filter's feedback loop (based on [11, 80]). . . . . . . . . . . . 22
3.8 Occlusion handling using the Kalman filter [55]. . . . . . . . . . . . . . . . 22
3.9 Magnetometer calibration. . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.10 Euler angles demonstrated on an triaxial gimbal [23]. . . . . . . . . . . . . 26
3.11 An *RGB* image splitted into individual color channels (based on [78]). . . 27
3.12 An $YC_bC_r$ image composed of the luminance and chroma channels [78]. . 27
3.13 Overview of structuring elements [72]. . . . . . . . . . . . . . . . . . . . . 28
3.14 Primitive operations of image morphology [72]. . . . . . . . . . . . . . . . 29

4.1 Comparison of the real world and the virtual view. . . . . . . . . . . . . . 32
4.2 System overview. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.3 Horizontal projection of the tracking area. . . . . . . . . . . . . . . . . . . 36
4.4 Software architecture of the optical position tracking component. . . . . . 38
4.5 Software architecture of the visual hand pose estimation component. . . . . 41
4.6 Hardware architecture of the inertial orientation tracking component. . . 42
4.7 Software architecture of the inertial orientation tracking component. . . . 43

5.1 Disassembly of the webcam. . . . . . . . . . . . . . . . . . . . . . . . . . . 46
5.2 Characteristics of a daylight suppression filter. . . . . . . . . . . . . . . . 47
5.3 Stereoscopic camera group. . . . . . . . . . . . . . . . . . . . . . . . . . . 47

5.4 Stereoscopic camera views. . . . . . . . . . . . . . . . . . . . . . . . . 48
5.5 RGB camera view. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.6 Prototypical setup of the camera rigs. . . . . . . . . . . . . . . . . . . . . 49
5.7 Snapshots of the intrinsic camera calibration. . . . . . . . . . . . . . . . 50
5.8 Reprojection error graphs. . . . . . . . . . . . . . . . . . . . . . . . . . 51
5.9 Active IR marker. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
5.10 Schematic overview of the inertial orientation tracker. . . . . . . . . . . . 52
5.11 Evolutionary stages of the wearable 6DoF tracker. . . . . . . . . . . . . . 53
5.12 Circuit diagram of the 6DoF tracker. . . . . . . . . . . . . . . . . . . . . 53
5.13 3D models of the tracker's enclosure. . . . . . . . . . . . . . . . . . . . . 54
5.14 Final 6DoF tracker. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
5.15 Data pipeline of the inertial sensor calibration. . . . . . . . . . . . . . . . 55
5.16 Class diagram of the position tracking component. . . . . . . . . . . . . . 55
5.17 GUI of the optical position tracking component. . . . . . . . . . . . . . . 56
5.18 Status signals of the position tracking algorithm. . . . . . . . . . . . . . . 57
5.19 Stereoscopic IR camera views during the calibration stage. . . . . . . . . . 58
5.20 ROI selection. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
5.21 Filter mask that combines the ROI selection with the IR interference filter. 59
5.22 Class diagram of the hand pose estimation component. . . . . . . . . . . . 61
5.23 Comparison of the original image with the $YC_bC_r$ color channels. . . . . . 62
5.24 Feature points detection. . . . . . . . . . . . . . . . . . . . . . . . . . . 63
5.25 Unity's GUI. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

6.1 Marker setup during spatial performance evaluation. . . . . . . . . . . . . 68
6.2 Subject determining the area of occlusion in front of the body. . . . . . . 68
6.3 3D histogram of VreeTracker's positional tracking data. . . . . . . . . . . 70
6.4 3D histogram of Vive's positional tracking data. . . . . . . . . . . . . . . 70
6.5 VreeTracker's positional data while moving on a rotating platform. . . . . 71
6.6 Vive's positional data while moving on a horizontally rotating platform. . . 71
6.7 Comparison of the VreeTracker's and Vive's positional data. . . . . . . . . 71
6.8 VreeTracker's orientational tracking data while resting still. . . . . . . . . 72
6.9 Vive's orientational tracking data while resting still. . . . . . . . . . . . . 72
6.10 VreeTracker's orientational tracking data while rotating around an axis. . 73
6.11 Vive's orientational tracking data while rotating around an axis. . . . . . 73
6.12 Areas of occlusion in front of the climbing wall. . . . . . . . . . . . . . . 75

# List of Tables

5.1 Logitech C920's technical specification [46]. . . . . . . . . . . . . . . . . . . . 46

# Acronyms

**3D** three-dimensional. 3, 17, 20, 21, 23, 26, 35–37, 39, 41–43, 45, 52, 53, 55, 56, 60–64, 67, 69, 77, 78

**6DoF** six-degrees-of-freedom. 52, 54, 63, 64, 67, 73–75

**AHRS** Attitude and Heading Reference System. 8, 24, 26, 51, 78

**DoF** Degree of Freedom. 2

**GUI** Graphical User Interface. 56, 57, 61, 64, 65, 80

**HMD** Head-Mounted Display. 2, 10–13, 31, 74, 75

**IMU** Inertial Measurement Unit. 8, 11, 13, 24

**IR** infrared. 5–7, 9, 11, 12, 15, 16, 35–41, 45–52, 57–59, 61, 63, 65, 67, 69, 74, 77, 80

**NIR** Near Infrared. 5, 15, 16, 37

**ROI** Region of Interest. 57–59, 80

**UV** ultraviolet. 15, 16

**VE** Virtual Environment. 1

**VR** Virtual Reality. 1, 2, 4, 5, 8–10, 31, 35, 39, 42, 43, 45, 51, 52, 60, 61, 63–65, 74, 75, 77

# Bibliography

[1] A. Davies. Oculus Rift Vs. HTC Vive Vs. PlayStation VR. `http://www.tomshardware.co.uk/vive-rift-playstation-vr-comparison,review-33556-6.html`, Accessed: 2017-12-15.

[2] Adafruit. Magnetometer Calibration. `https://learn.adafruit.com/ahrs-for-adafruits-9-dof-10-dof-breakout/magnetometer-calibration`, Accessed: 2018-01-06.

[3] Adafruit. Adafruit Feather HUZZAH with ESP8266 WiFi. `https://www.adafruit.com/product/2821`, Accessed: 2018-02-25.

[4] Adafruit. Adafruit Precision NXP 9-DOF Breakout Board. `https://www.adafruit.com/product/3463`, Accessed: 2018-02-25.

[5] ALCS. Logitech C910 and C920 IR (Infrared) conversion for nightvision. `http://www.alcs.ch/logitech-c910-infrared-conversion-for-nightvision.html`, Accessed: 2018-02-18.

[6] Amazon. Neewer 58mm Infrared IR 850nm Filter For Canon Digital Rebel. `https://www.amazon.co.uk/Neewer-Infrared-850Nm-Filter-Digital/dp/B003TY3CQS`, Accessed: 2018-02-19.

[7] Arduino. Arduino IDE. `https://www.arduino.cc/en/Main/Software`, Accessed: 2018-03-17.

[8] D. Bartlett. *Essentials of positioning and location technology.* Cambridge University Press, 2013.

[9] S. J. Biggs and M. A. Srinivasan. Haptic interfaces. In *Handbook of Virtual Environments*, pages 93–116. Taylor & Francis, 2002.

[10] BiPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, and OIML. International vocabulary of metrology – basic and general concepts and associated terms, 2008. *JCGM*, 200, 2008.

[11] G. Bishop and G. Welch. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, 2001.

[12] D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice.* Addison Wesley Longman Publishing Co., Inc., 2004.

[13] G. Burdea and P. Coiffet. *Virtual Reality Technology.* John Wiley & Sons, Inc., second edition, 2003.

[14] W. Burger. Zhang's camera calibration algorithm: In-depth tutorial and implementation. Technical Report HGB16-05, University of Applied Sciences Upper Austria,School of Informatics, Communications and Media, Dept. of Digital Media, Hagenberg, Austria, may 2016.

[15] W. Burger and M. J. Burge. *Digitale Bildverarbeitung: Eine Einführung mit Java und ImageJ.* Springer Berlin Heidelberg, 2005.

[16] A. Cavallo, A. Cirillo, P. Cirillo, G. De Maria, P. Falco, C. Natale, and S. Pirozzi. Experimental comparison of sensor fusion algorithms for attitude estimation. *IFAC Proceedings Volumes*, 47(3):7585–7591, 2014.

[17] S. R. Chapala, G. S. Pirati, and U. R. Nelakuditi. Determination of coordinate transformations in uavs. In *Cognitive Computing and Information Processing (CCIP), 2016 Second International Conference on*, pages 1–5. IEEE, 2016.

[18] T. Cloete and C. Scheffer. Benchmarking of a full-body inertial motion capture system for clinical gait analysis. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, pages 4579–4582. IEEE, 2008.

[19] Crytek GmbH. The Climb – A VR Climbing Game. `http://www.theclimbgame.com`, Accessed: 2017-10-16.

[20] S. Davis, K. Nesbitt, and E. Nalivaiko. Comparing the onset of cybersickness using the oculus rift and two virtual roller coasters. In *11th Australasian Conference on Interactive Entertainment (IE 2015)*, volume 167, pages 3–14. ACS, 2015.

[21] H. Eidenberger and A. Mossel. Indoor skydiving in immersive virtual reality with embedded storytelling. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 9–12. Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology, 2015. talk: ACM Symposium on Virtual Reality Software and Technology (VRST), Beijing, China; 2015-11-13 – 2015-11-15.

[22] M. S. Grewal and A. P. Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems*, 30(3):69–78, 2010.

[23] M. S. Grewal, L. R. Weill, and A. P. Andrews. *Global positioning systems, inertial navigation, and integration.* John Wiley & Sons, 2007.

[24] P. R. Gringer. Image of EM Spectrum. `https://commons.wikimedia.org/w/index.php?title=File:EM_spectrumrevised.png&oldid=258493882`, Accessed: 2017-12-30.

[25] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, second edition, 2000.

[26] H. G. Hoffmann. Physically touching virtual objects using tactile augmentation enhances the realism of virtual environments. In *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)*, pages 59–63, Atlanta, GA, USA, 1998.

[27] HTC Corporation. HTC Vive. `https://www.vive.com/eu/product/`, Accessed: 2017-12-15.

[28] iFixit. HTC Vive Teardown. `https://de.ifixit.com/Teardown/HTC+Vive+Teardown/62213?lang=en`, Accessed: 2017-12-15.

[29] iFixit. Oculus Rift CV1 Teardown. `https://de.ifixit.com/Teardown/Oculus+Rift+CV1+Teardown/60612`, Accessed: 2017-12-15.

[30] Intel. RealSense. `https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html`, Accessed: 2018-03-18.

[31] S. Islam, B. Ionescu, C. Gadea, and D. Ionescu. Full-body tracking using a sensor array system and laser-based sweeps. In *2016 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 71–80, 2016.

[32] D. Jagneaux. HTC Vive vs. Oculus Rift With Touch – Which Is The Better Roomscale Experience? `https://uploadvr.com/vive-vs-oculus-rift-touch-roomscale/`, Accessed: 2017-12-15.

[33] I. Jahr. Lighting in machine vision. In *Handbook of Machine Vision*, chapter 3, pages 73–203. Wiley Online Library, 2007.

[34] R. E. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[35] E. Kolasinski. *Simulator sickness in virtual environments.* Number Bd. 4, Nr. 1027 in Technical report (U.S. Army Research Institute for the Behavioral and Social Sciences). U.S. Army Research Institute for the Behavioral and Social Sciences, 1995.

[36] F. Kosmalla, A. Zenner, M. Speicher, D. Daiber, N. Herbig, and A. Krüger. Exploring rock climbing in mixed reality environments. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 1787–1793, New York, NY, USA, 2017. ACM.

[37] O. Kreylos. Lighthouse tracking examined. `http://www.doc-ok.org/?p=1478`, Accessed: 2017-12-15.

[38] O. Kreylos. Oculus Rift DK2's tracking update rate. `http://www.doc-ok.org/?p=1405`, Accessed: 2017-12-15.

[39] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality.* Princeton University Press, 2002.

[40] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov. Head tracking for the oculus rift. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 187–194, 2014.

[41] Leap Motion Inc. Leap Motion. `https://www.leapmotion.com`, Accessed: 2018-03-18.

[42] K. Lenhardt. Optical systems in machine vision. In *Handbook of Machine Vision*, chapter 3, pages 73–203. Wiley Online Library, 2007.

[43] Q. Li, R. Li, K. Ji, and W. Dai. Kalman filter and its application. In *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 74–77, 2015.

[44] H. Liesker, E. Brenner, and J. B. J. Smeets. Combining eye and hand in search is suboptimal. *Experimental Brain Research*, 197(4):395–401, Aug 2009.

[45] Logitech. Logitech HD Pro Webcam C920 Product Page. `https://www.logitech.com/de-at/product/hd-pro-webcam-c920`, Accessed: 2018-02-18.

[46] Logitech. Logitech HD Pro Webcam C920 Technical Specification. `http://support.logitech.com/en_us/product/hd-pro-webcam-c920/specs`, Accessed: 2018-02-18.

[47] S. O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25, 2010.

[48] S. O.H. Madgwick, A. J.L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–7. IEEE, 2011.

[49] R. Mahony, T. Hamel, and J.-M. Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, 53(5):1203–1218, 2008.

[50] A. Malventano. SteamVR HTC Vive In-depth – Lighthouse Tracking System Dissected and Explored. `https://www.pcper.com/reviews/General-Tech/SteamVR-HTC-Vive-depth-Lighthouse-Tracking-System-Dissected-and-Explored/SteamV`, Accessed: 2017-12-15.

[51] Materialise NV. 3D Scanning Tutorial for Microsoft Kinect and 3D Builder. `https://i.materialise.com/blog/3d-scanning-tutorial-microsoft-kinect-and-3d-builder`, Accessed: 2017-10-16.

[52] Mathworks. What Is Camera Calibration? `https://de.mathworks.com/help/vision/ug/camera-calibration.html`, Accessed: 2018-01-01.

[53] Mathworks. Design and use Kalman filters in MATLAB and Simulink. `https://www.mathworks.com/discovery/kalman-filter.html`, Accessed: 2018-01-04.

[54] Mathworks. Understanding Kalman Filters. `https://www.mathworks.com/videos/series/understanding-kalman-filters.html`, Accessed: 2018-01-04.

[55] Mathworks. Using Kalman Filter for Object Tracking. `https://de.mathworks.com/help/vision/examples/using-kalman-filter-for-object-tracking.html`, Accessed: 2018-01-04.

[56] Mathworks. Matlab – Single Camera Calibration App. `https://de.mathworks.com/help/vision/ug/single-camera-calibrator-app.html?s_tid=gn_loc_drop`, Accessed: 2018-03-03.

[57] Mathworks. Matlab. `https://www.mathworks.com/products/matlab.html`, Accessed: 2018-03-10.

[58] Mathworks. Matlab Support Package for USB Webcams. `https://de.mathworks.com/help/supportpkg/usbwebcams/index.html`, Accessed: 2018-03-10.

[59] A. Mossel. *Robust Wide-Area Tracking and Intuitive 3D Interaction for Mixed Reality Environments*. PhD thesis, Institute of Software Technology and Interactive Systems, 2014.

[60] Oculus VR, LLC. Oculus Rift. `https://www.oculus.com/rift/`, Accessed: 2017-10-16.

[61] Oculus VR, LLC. Oculus Roomscale—Tips for Setting Up a Killer VR Room. `https://www.oculus.com/blog/oculus-roomscale-tips-for-setting-up-a-killer-vr-room/`, Accessed: 2017-12-15.

[62] Oculus VR, LLC. Simulator Sickness. `https://developer.oculus.com/design/latest/concepts/bp_app_simulator_sickness/`, Accessed: 2017-12-15.

[63] OptiTrack. Costs of a small VR Motion Capture Setup. `https://www.optitrack.com/systems/#virtual-reality/prime-41/6`, Accessed: 2017-10-16.

[64] OptiTrack. OptiTrack for Animation. `https://www.optitrack.com/motion-capture-animation`, Accessed: 2017-10-16.

[65] K. Orland. Facebook purchases VR headset maker Oculus for 2 billion dollars. `https://arstechnica.com/gaming/2014/03/facebook-purchases-vr-headset-maker-oculus-for-2-billion/`, Accessed: 2017-12-15.

[66] T. Pintaric and H. Kaufmann. Affordable infrared-optical pose tracking for virtual and augmented reality. In G. Zachmann, editor, *IEEE VR Workshop on Trends and Issues in Tracking for Virtual Environments*, pages 44–51. Shaker Verlag, 2007.

[67] T. Pintaric and H. Kaufmann. A rigid-body target design methodology for optical pose-tracking systems. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, pages 73–76. ACM Press, 2008.

[68] T. Pintaric and H. Kaufmann. ioTracker. `http://www.iotracker.com`, Accessed: 2017-12-15.

[69] PJRC. Prop Shield With Motion Sensors. `https://www.pjrc.com/store/prop_shield.html`, Accessed: 2018-01-06.

[70] M. Ribo, A. Pinz, and A. L. Fuhrmann. A new optical tracking system for virtual and augmented reality applications. In *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, volume 3, pages 1932–1936, 2001.

[71] D. Smith. Calculating the Emission Spectra from Common Light Sources. `https://www.comsol.com/blogs/calculating-the-emission-spectra-from-common-light-sources/`, Accessed: 2017-12-30.

[72] C. Solomon and T. Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab.* John Wiley & Sons, Inc., 2011.

[73] StereoLabs. ZED 2K Stereo Camera. `https://www.stereolabs.com`, Accessed: 2018-03-18.

[74] P. Stoffregen. MotionCal – Motion Sensor Calibration Tool. `https://github.com/PaulStoffregen/MotionCal`, Accessed: 2018-03-04.

[75] R. Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[76] Unity Technologies. Unity 3D. `https://unity3d.com/de`, Accessed: 2018-03-17.

[77] Unmanned Systems Source. BP850 Near-IR Bandpass Filter. `https://www.unmannedsystemssource.com/shop/aerial-imaging/bp850-near-ir-bandpass-filter/`, Accessed: 2018-02-19.

[78] Wikimedia Commons. Barns grand tetons YCbCr separation. `https://commons.wikimedia.org/w/index.php?title=File:Barns_grand_tetons_YCbCr_separation.jpg&oldid=151926874`, Accessed: 2018-01-04.

[79] H. S. Yeo, B. G. Lee, and H. Lim. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. *Multimedia Tools and Applications*, 74(8):2687–2715, 2015.

[80] P. Yonak. How To Determine Location If You have A Roof Over Your Head. `http://blog.lemberg.co.uk/how-determine-location-if-you-have-roof-over-your-head`, Accessed: 2018-01-04.

[81] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.