



# Mimikry

## Ein Low-Cost Motion-Tracking System

### Bachelorthesis

Computermodelle mit natürlichen Bewegungen zum Leben zu erwecken ist für moderne Videospiele und Filme, wie auch in der Medizinbranche, essentiell wichtig und ist grundsätzlich nur durch das Aufnehmen der Bewegungen eines Schauspielers möglich. Motion Capturing Systeme existieren, kosten aber im Minimum mehrere tausend Franken. Ziel dieser Arbeit ist es, ein kostengünstiges System zu erstellen, welches Bewegungen eines Menschen auf ein Computermodell übertragen kann. Dieses Ziel wurde erreicht. Unser System hat gegenüber existierenden Systemen Einschränkungen im Bezug auf den Aufnahmeraum und der Geschwindigkeit der verfolgbaren Bewegungen. Diese Probleme sind jedoch lösbar. Das System kann als Grundlage für eine Weiterentwicklung dienen oder, in der jetzigen Form, als kostengünstige Alternative für einige Animationen und Prototypen.

Studiengang: BSc Informatik

Autoren: Dellasperger Jan, Sheppard David

Betreuer: Hudritsch Marcus

Experte: Dr. Studer Harald

Datum: 17.01.2019

## Versionen

| Version | Date       | Status | Remarks                               |
|---------|------------|--------|---------------------------------------|
| 0.1     | 04.01.2019 | Draft  | Erster Entwurf                        |
| 0.2     | 12.01.2019 | Draft  | Entwurf zur Korrekturlesung geschickt |
| 1.0     | 17.01.2019 | Final  | Finale Version                        |

# **Management Summary**

## **Hintergrund**

Animierte, virtuelle Modelle sind aus modernen Videospielen und Spielfilmen nicht mehr wegzudenken. Die Bewegungen der Modelle müssen möglichst natürlich sein, damit sie glaubwürdig wirken. Um dies zu erreichen werden die Bewegungen entweder von einem Künstler von Hand in einem Animationstool erstellt, was schwierig, zeitaufwändig und dementsprechend teuer ist, oder ein Schauspieler wird als Referenz verwendet, dessen Bewegungen werden aufgezeichnet und auf das Modell angewandt. Diesen Prozess nennt man Motion Capturing oder Motion Tracking. Bewegungen, welche durch Motion Capturing aufgenommen wurden, finden neben der eingangs erwähnten Unterhaltungsindustrie auch in der Medizinbranche Anwendung. Dort werden sie zur Bewegungsanalyse von Spitzensportlern oder Rehabilitationspatienten eingesetzt. Professionelle Motion Capturing Systeme existieren, kosten aber im Minimum mehrere tausend Franken, wobei dem Preis gegen oben fast keine Grenzen gesetzt sind.

## **Ziele**

Ziel dieser Arbeit ist es, ein kostengünstiges Motion Capturing System zu erstellen, welches Bewegungen eines Menschen auf ein Computermodell übertragen kann. Die verfolgte Bewegung soll dabei auf wenige Zentimeter genau übernommen werden. Die Visualisierung des animierten Modells soll in Echtzeit geschehen.

## **Vorgehen**

Damit die Bewegungen eines Schauspielers verfolgt und aufgezeichnet werden können, muss das System in der Lage sein, die Position der Gelenke des Schauspielers im dreidimensionalen Raum zu erkennen. Als Grundlage dazu dienen Raspberry Pis mit angeschlossenen Kameras. Die Kameras sind handelsüblich, außer dass sie keinen IR (Infrarot) Filter eingebaut haben. Der Schauspieler trägt bei der Aufnahme einen Anzug, an dem 13 retroreflektive Kugeln, sogenannte Marker, angebracht wurden. Um die Kameras wurde ein Ring aus IR-LEDs angebracht, deren Licht von den Markern reflektiert wird. Die Kameras sind in der Lage dieses zu detektieren. Da jeder Marker von mehreren Kameras gesehen wird, kann deren Position trianguliert werden. Dies geschieht durch das Berechnen von Strahlenschnittpunkten. Die positionierten Marker werden dann Modellpunkten zugeordnet. Damit ist die Haltung des Schauspielers jederzeit definiert und kann auf ein Modell übertragen werden. Die Darstellung des Modells geschieht schlussendlich in der Spiel-Engine Unity.

## **Fazit**

Das System erfüllt die gestellten Anforderungen. Marker können relativ präzise (Fehler < 1cm auf jeder Achse) positioniert werden und das virtuelle Modell wird in Echtzeit animiert. Das System hat gegenüber professionellen Systemen jedoch einige Einschränkungen; Der Aufnahmehbereich ist auf einen am Boden stehenden Kegel von 2m Durchmesser und etwas über 1.70m Höhe beschränkt und es darf keine direkte Beleuchtung des Aufnahmerraumes geben. Diese Probleme sind jedoch durch den Einsatz entsprechender Hardware lösbar.



# Inhaltsverzeichnis

|   |    |
|---|----|
| <b>Management Summary</b>                           | i  |
| <b>1. Einleitung</b>                                | 1  |
| 1.1. Mathematische Notation . . . . .               | 2  |
| 1.2. Diagrammlegende . . . . .                      | 2  |
| <b>2. Ziele</b>                                     | 5  |
| <b>3. Projektmanagement</b>                         | 7  |
| 3.1. Zeitplanung . . . . .                          | 7  |
| 3.2. Versionskontrolle . . . . .                    | 7  |
| 3.3. Issue Tracking . . . . .                       | 7  |
| 3.4. Auswertung . . . . .                           | 8  |
| <b>4. Konzept</b>                                   | 11 |
| 4.1. Hardware . . . . .                             | 11 |
| 4.2. Netzwerkkomponente . . . . .                   | 13 |
| 4.3. Markerkonfiguration . . . . .                  | 14 |
| 4.4. Kamerasoftware Spotter . . . . .               | 15 |
| 4.5. Serversoftware Beholder . . . . .              | 15 |
| 4.6. Visualisierungssoftware Mimic . . . . .        | 16 |
| <b>5. Umsetzung</b>                                 | 17 |
| 5.1. Hardware . . . . .                             | 17 |
| 5.2. Spotter . . . . .                              | 19 |
| 5.3. Beholder . . . . .                             | 30 |
| 5.4. Mimic . . . . .                                | 38 |
| <b>6. Systemanalyse</b>                             | 43 |
| 6.1. Laufzeitanalyse . . . . .                      | 43 |
| 6.2. Präzisionsanalyse . . . . .                    | 44 |
| 6.3. Kostenvergleich mit Referenzsystem . . . . .   | 46 |
| 6.4. Zielerfüllung . . . . .                        | 47 |
| 6.5. Aktuelle Probleme und Lösungsansätze . . . . . | 47 |
| 6.6. Verbesserungspotential . . . . .               | 48 |
| <b>7. Fazit</b>                                     | 49 |
| <b>Selbstständigkeitserklärung</b>                  | 51 |
| <b>Glossar</b>                                      | 53 |
| <b>Bibliographie</b>                                | 57 |
| <b>Abbildungsverzeichnis</b>                        | 59 |
| <b>Tabellenverzeichnis</b>                          | 61 |
| <b>APPENDIX</b>                                     | 63 |
| <b>A. Spotter Installationsanleitung</b>            | 65 |

|   |           |
|---|-----------|
| <b>B. Spotter Frontplatte</b>             | <b>67</b> |
| <b>C. Messprotokoll Präzisionsanalyse</b> | <b>69</b> |
| <b>D. Quartile Laufzeitanalyse</b>        | <b>71</b> |
| <b>E. Inhalt des USB-Stick</b>            | <b>73</b> |

# 1. Einleitung

Eine virtuelle Welt, sei es in einem Spiel, einer Simulation oder einem Film mit animierten Komponenten lebt von der Glaubwürdigkeit seiner Darstellung. Im Zusammenhang mit virtueller Realität (VR) wird auch der Grad an erzeugter Immersion (vom lateinischen *immersio* für "Eintauchen", "Einbetten") als massgebend angesehen. Ein hoher Grad an Immersion bedeutet ein starkes "Mittendrin-Gefühl", also ein tiefes Eintauchen in die virtuelle Realität und ein akzeptieren jener als "Wahrheit".

Es gibt mehrere Faktoren, welche eine künstliche Welt glaubwürdig scheinen lassen. Der Hauptfaktor ist die möglichst korrekte Berechnung und Darstellung der Welt im Bezug auf die Beleuchtung. Dies bedeutet insbesondere die korrekte Darstellung von Licht und Schatten. Ein weiterer wichtiger Faktor ist das möglichst plastische Aussehen von Figuren und Objekten. Diese Faktoren können als optische Faktoren zusammengefasst werden. Es gibt aber noch andere Bestandteile einer hohen Glaubwürdigkeit von virtuellen Welten. Zum Beispiel korrekt berechnete physikalische Eigenschaften oder menschliche Mimik, Gestik und Bewegung die möglichst natürlich wirkt.

Eine Computerfigur (auch Modell genannt) mit Bewegungen zu versehen nennt man animieren. Solche Animationen können in geeigneten Tools von Hand gebaut werden. Dies bedeutet an den Schlüsselstellen der Bewegung die Position jedes Knochens zu definieren. Auch schon kleine Bewegungen zu animieren kann einen immensen Aufwand bedeuten. Dieser Aufwand wächst ungleich mit dem Anspruch auf Natürlichkeit. Neben den Körperbewegungen und Bewegung der Extremitäten an sich können auch feine Bewegungen im Gesicht (Mimik) oder zum Beispiel der Finger (Gestik) animiert werden.

Unsere Bachelorthesis widmet sich dem Thema Bewegung und Animation. Das Ziel soll sein, die Bewegungen eines Menschen auf eine virtuelle Figur zu legen. Diese Technik wird Motion Capturing genannt. Mittels Motion Capturing können die Bewegungen eines Menschen aufgezeichnet werden. Aus diesen Aufzeichnungen können dann Animationen generiert und auf ein Computermodell gelegt werden. Mit dieser Technik kann man komplexe Animationen verhältnismässig einfach erzeugen. Die Methode ist also schnell und eliminiert das Problem der Künstlichkeit der animierten Bewegung. Sowohl Körperbewegung, aber auch Mimik und Gestik, lassen sich mit entsprechenden Systemen aufzeichnen. Motion Capturing Systeme finden vor allem in der Spiel- und Filmindustrie Anwendung. Jedoch sind sie auch für die Medizinbranche interessant, wo sie zum Beispiel zur Bewegungsanalyse von Spitzensportlern oder Rehabilitationspatienten verwendet werden.

Um Bewegungen aus der Realität in eine virtuelle Welt zu projizieren sind mehrere Komponenten nötig. Einerseits sind Kameras notwendig, um die Bewegungen überhaupt aufzuzeichnen. Dann muss eine Komponente die Kameradaten auswerten und die reellen Punkte in die virtuelle Welt übersetzen. Zuletzt muss das Resultat visualisiert oder aber in geeigneter Form gespeichert werden.

Das Ziel der Arbeit ist die Erstellung eines umfassenden Motion Capturing Systems. Dieses soll in der Lage sein, eine Bewegung in Echtzeit auf ein Computermodell zu übertragen und dieses korrekt zu animieren. Wir wollen dazu auf die Zuhilfenahme von fertigen Systemen oder Teilsystemen verzichten, nicht zuletzt weil professionelle Systeme extrem teuer sind. Dies bedeutet, dass weder spezialisierte Kamerasytsemes noch Software beschafft oder benutzt werden. Diese Komponenten sollen selber geschaffen werden. Um nicht das komplette Gerüst für die Visualisierung selber erstellen zu müssen, wird die Spiel-Engine "Unity" zur Darstellung verwendet.

Wir nennen unsere Arbeit "Mimikry". Mimikry ist ein Verteidigungsmechanismus aus der Tierwelt, bei der ein eigentlich harmloses Tier einen gefährlichen Artgenossen nachahmt. Ein Beispiel dafür sind Holzwespen, welche selber nicht stechen können, aber bei flüchtigem Betrachten durchaus wie normale Wespen aussehen können. Genau wie eine harmlose Holzwespe eine normale Wespe nachahmt, will unser System so tun, als ob es ein hochprofessionelles System wäre.

## 1.1. Mathematische Notation

Bei Formeln und anderen mathematischen Definitionen gilt folgende Notation:

**Realzahlen** werden mit einem Kleinbuchstaben geschrieben.

$$t \in \mathbb{R} \quad (1.1)$$

**Vektoren** werden mit einem Kleinbuchstaben dargestellt und sind mit einem Pfeil versehen. Vektoren sind als Kolonnenvektoren zu interpretieren.

$$\vec{v} \in \mathbb{R}^N \quad (1.2)$$

**Punkte** werden mit einem Kleinbuchstaben und Fett dargestellt.

$$\mathbf{p} \in \mathbb{R}^N \quad (1.3)$$

**Matrizen** werden mit einem Grossbuchstaben und Fett geschrieben dargestellt. Wir verwenden Matrizen im Column-Major-Format.

$$\mathbf{M} \in \mathbb{R}^N \cdot \mathbb{R}^M \quad (1.4)$$

Wo anwendbar hat eine **Transformationsmatrixe** ausserdem ein hochgestelltes Präfix, welches das Zielkoordinatensystem identifiziert und ein tiefgestelltes Postfix, welches das Ursprungskoordinatensystem identifiziert.  ${}^cT_w$  transformiert also einen Punkt vom Koordinatensystem  $w$  ins Koordinatensystem  $c$ .

## 1.2. Diagrammlegende

Bei **Komponentendiagrammen** werden Komponenten farblich hinterlegt und in derselben Farbe beschriftet. Subkomponenten sind umrahmt, wobei Subkomponenten von Dritten (Libraries) gestrichelt umrahmt sind. Abhängigkeiten werden durch Verbindungslinien dargestellt.

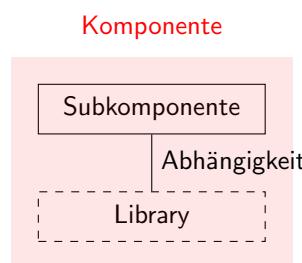


Abbildung 1.1.: Legende für Komponentendiagramme

**Zustandsdiagramme** haben einen Startknoten, welcher als ausgefüllter Punkt dargestellt wird, und einen Endknoten, welcher als ausgefüllter Punkt mit einem Kreis dargestellt wird. Zustände werden als Ovale dargestellt, Zustandübergänge als Pfeile.

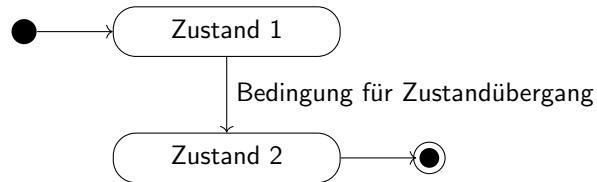


Abbildung 1.2.: Legende für Zustandsdiagramme

**Sequenzdiagramme** stellen den Programmfluss aus Komponentensicht dar. Funktionsaufrufe werden als Pfeile dargestellt. Asynchrone Abläufe werden als gestrichelte Linien dargestellt.

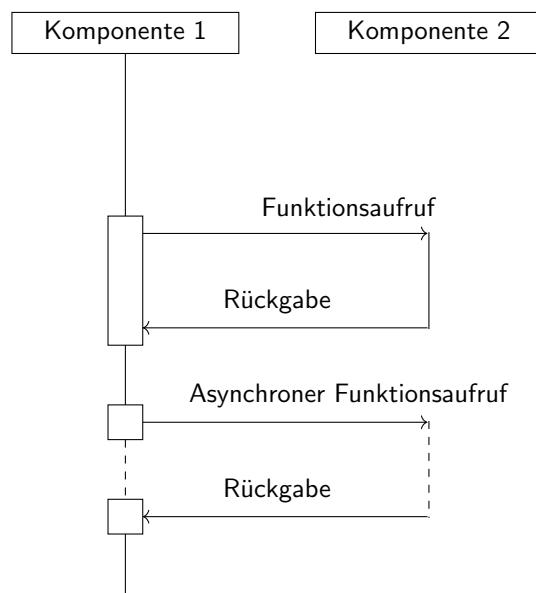


Abbildung 1.3.: Legende für Sequenzdiagramme



## 2. Ziele

Es soll ein System geschaffen werden, mit dem Bewegungen eines Menschen in Echtzeit auf ein Computermodell übertragen werden können. Dies soll dazu dienen, das Computermodell realistisch zu bewegen ohne aufwändige Animationen zu erstellen.

Je grösser der aufzuzeichnende Bereich ist, desto mehr Kameras werden benötigt. Da wir auf ein funktionierendes Minimalsystem abzielen, beschränken wir uns auf einen kleinen Bereich. Deshalb wird es nicht möglich sein, platzintensive Bewegungsabläufe aufzuzeichnen. Der abzudeckende Bereich ergibt sich aus der Armspannweite und der Körpergrösse eines ausgewachsenen Menschen, mit etwas Platz zu jeder Seite. Um eine nachvollziehbare und realistische Bewegung zu generieren, muss das System eine hohe Genauigkeit aufweisen. Um zwei aneinanderliegenden Handgelenke als separate Punkte identifizieren zu können, darf der Fehler beim Tracking die halbe Dicke eines Handgelenks nicht überschreiten. Die Framerate bei der Aufnahme und Verarbeitung definiert, wie schnell eine Bewegung sein darf, damit sie bei der Visualisierung noch als flüssig wahrgenommen wird. Unser Ziel orientiert sich in diesem Punkt am Rendering, bei welchem 60 Frames pro Sekunde als flüssig gelten. Wir wollen diese Framerate auf jeder Komponente unseres Systems erreichen.

Die Evaluation eines geeigneten Kamerasytems ist Teil dieses Projekts. Professionelle Hersteller solcher Systeme, wie OptiTrack oder Vicon bieten optimierte Kamerasyteme an. Die günstigste Kamera, welche OptiTrack fürs Motion Capturing anbietet, kostet 599\$ [1]. In Anbetracht dessen, dass Profisysteme mit mindestens vier Kameras arbeiten, ist dies eine beachtliche Investition. Für das Projekt soll eine preiswerte Alternative für solche Profisysteme gefunden werden. Ziel ist es, eine Alternative für weniger als 200 Franken pro Kamera zu finden.

Da das Hauptaugenmerk bei diesem Projekt aber nicht auf der Hardware, sondern auf der Software liegen soll, wird die Evaluierung, Beschaffung und Herstellung der Hardware nach dem Minimalprinzip erfolgen. Dies wird vermutlich einige Einschränkungen nach sich ziehen. Wir beschränken uns aber im Projektverlauf darauf, diese zu erkennen und zu dokumentieren. Sie werden nur im Ausnahmefall ausgemerzt.

Aus obigen Ausführungen lassen sich folgende qualitativen Ziele für das Gesamtsystem ableiten:

- Das System soll einen zylinderförmigen Bereich von 2 Metern Durchmesser und 2 Metern Höhe tracken können
- Jede der Komponenten soll eine Framerate von 60 erfüllen
- Der kumulierte Fehler bei der Positionsbestimmung der Marker soll <2cm sein
- Die Kosten pro Kamera sollen 200 Franken nicht übersteigen



# 3. Projektmanagement

Da wir uns während der Thesis möglichst auf die praktische Arbeit konzentrieren wollten, war es unser Ziel möglichst ohne Management-Overhead auszukommen. Aus diesem Grund wurde vorgängig nur eine sehr grobe Zeitplanung gemacht. Um eine kontinuierliche Kontrolle über die geleistete Arbeit zu haben, wurde außerdem ein Arbeitslog in einem Online-Spreadsheet geführt.

## 3.1. Zeitplanung

Aufgrund der Ziele wurden mehrere Arbeitspakete definiert. Für die Thesis wird mit 720h gerechnet (360h pro Person), was dem Zeitaufwand pro ECTS im Bologna-System entspricht [2]. Diese wurden auf die Arbeitspakete verteilt wie in Tabelle 3.1 ersichtlich ist.

| Arbeitspaket             | Zeitbudget [h] |
|--------------------------|----------------|
| Bau/Evaluierung Hardware | 80             |
| Markerdetektion          | 80             |
| Netzwerk                 | 60             |
| 3D Rekonstruktion        | 300            |
| Optimierung / Reserve    | 100            |
| Dokumentation            | 100            |

Tabelle 3.1.: Zeitplanung

Zusätzlich zu den für die Thesis vorgesehenen Stunden wurde etwas Vorarbeit bereits im Modul "Game Development" bei Marcus Hudritsch geleistet. Hier entstand ein POC (Proof of Concept) für die Kamera- und Systemarchitektur.

## 3.2. Versionskontrolle

Damit beide Entwickler immer Zugriff auf die Projekt-Sourcen haben wurde ein Git-Repository auf Github erstellt. Damit man sich bei der Entwicklung nicht in die Quere kommt, wurde pro Feature, an welchem gearbeitet wird, ein einzelner Branch erstellt. Sobald ein Feature kompletiert wurde, wurde der Branch zurück in den Master-Branch geführt. Da Git ein verteiltes Versionierungssystem ist, fungiert es zusätzlich auch als Backup [3].

## 3.3. Issue Tracking

Um ob der zweifellos vielen auftretenden Probleme die Übersicht nicht zu verlieren, wurde entschieden ein Issue Tracking Tool zu verwenden. Da wir bereits ein Github Repository verwenden, bot es sich an das dort mitgelieferte Tool zu verwenden. Im Verlauf des Projektes wurden auftretende Probleme dort erfasst, beschrieben und sobald sie abgehandelt wurden, geschlossen.

### 3.4. Auswertung

Folgende Auflistung gibt Aufschluss über die effektiv aufgewendete Zeit pro Arbeitspaket. Unter Optimierung/Reserve fällt neben Offensichtlichem auch alles was nicht direkt einem der anderen Arbeitspakete zugeordnet werden kann (Genauigkeitsmessung, Vorträge usw.). Unter Dokumentation ist neben effektiver Dokumentationsarbeit auch die Produktionszeit fürs Video gebucht. Im Paket Markerdetektion sind alle Arbeiten verbucht, welche im Zusammenhang mit den Spottern stehen. Diese Arbeiten erfolgten ganz am Anfang des Projekts. Dies bedeutet, dass wahrscheinlich ein valabler Anteil anderer Arbeiten (Netzwerk, Konzeptarbeit, usw.) ebenfalls unter Markerdetektion gebucht wurden.

Zu guter Letzt gilt es zudem zu beachten, dass wir effektiv 100 Stunden über soll sind. Dies ist damit zu begründen, dass wir auch beim Vorprojekt bereits Zeit gebucht haben, dies in der Planung aber nicht berücksichtigt haben. Dies erklärt auch den Offset der Soll-Stunden im Diagramm unten.

Die Arbeitsverteilung gliederte sich auch grob nach den geplanten Arbeitspaketen. David Sheppard kümmerte sich hauptsächlich um die Entwicklung der Netzwerkkomponente sowie des GUI und der Visualisierung. Jan Dellspenger war vor allem mit der Markerdetektion und der 3D Rekonstruktion beschäftigt. Die Systemanalyse sowie die Dokumentation wurden gemeinsam realisiert.

| Arbeitspaket             | Zeitbudget [h] | Effektiv aufgewendet [h] |
|--------------------------|----------------|--------------------------|
| Bau/Evaluierung Hardware | 80             | 60                       |
| Markerdetektion          | 80             | 146                      |
| Netzwerk                 | 60             | 38                       |
| 3D Rekonstruktion        | 300            | 203                      |
| Optimierung / Reserve    | 100            | 177.5                    |
| Dokumentation            | 100            | 197.5                    |
| <b>Gesamt</b>            |                | <b>822</b>               |

Tabelle 3.2.: Projekt Controlling

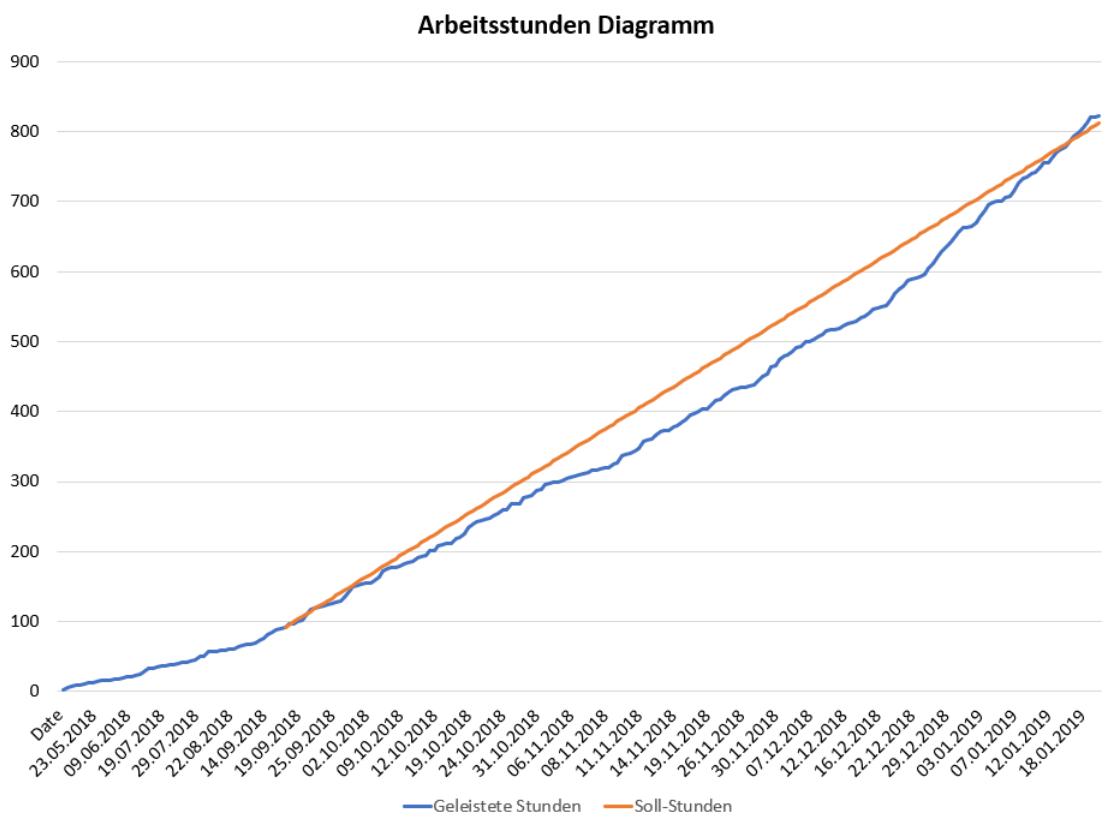


Abbildung 3.1.: Arbeitsstunden Diagramm



# 4. Konzept

Aufgrund der im Kapitel 2 definierten Ziele lässt sich folgende hochlevelige Systemübersicht ableiten:

Als Grundlage des Systems dienen Kameras, mit welchen Marker erkannt werden können. Das System muss in der Lage sein, die Positionen der erkannten Marker auf der zweidimensionalen Bildebene zu bestimmen. Es kann anschliessend ein Strahl vom Kameraursprung durch den Mittelpunkt des erkannten Markers berechnet werden, auf dem der Marker liegen muss. Um die Position der Marker im dreidimensionalen Raum zu bestimmen, müssen die Schnittpunkte der Strahlen mehrerer Kameras berechnet werden. Sobald die Position der Marker bestimmt ist, muss eine Zuweisung der Marker auf Modellpunkte stattfinden. Anhand der so positionierten und identifizierten Marker kann schlussendlich ein Modell in korrekter Stellung visualisiert werden.

Es bietet sich folgende dreiteilige Systemarchitektur an:

Auf jeder Kamera läuft ein Stück Software, genannt **Spotter**, welches die Marker auf den Kamerabildern detektiert und die entsprechenden Strahlen berechnet. Dies setzt voraus, dass jede Kamera ihre Position in einem gemeinsamen Koordinatensystem kennt. Die so berechneten Strahlen werden anschliessend an eine zentrale Serversoftware gesendet. Nur die Strahlen, und nicht etwa die ganzen Bilder, werden versendet, damit die Last auf dem Netzwerk möglichst gering bleibt.

Die Serversoftware, **Beholder**, erhält die Strahlen aller angeschlossenen Spotter und führt die Schnittpunktberechnung sowie die Zuweisung an die Modellpunkte aus. Die zugewiesenen Positionen werden dann an eine Visualisierungssoftware gesandt.

Die Visualisierungssoftware, **Mimic**, stellt ein Modell anhand der Markerpositionen dar. Außerdem dient sie als GUI um das System zu steuern, sowie Visualisierungshilfen für Debug-Zwecke ein und auszuschalten.

Abbildung 4.1 visualisiert das Konzept mit vier angeschlossenen Spottern.

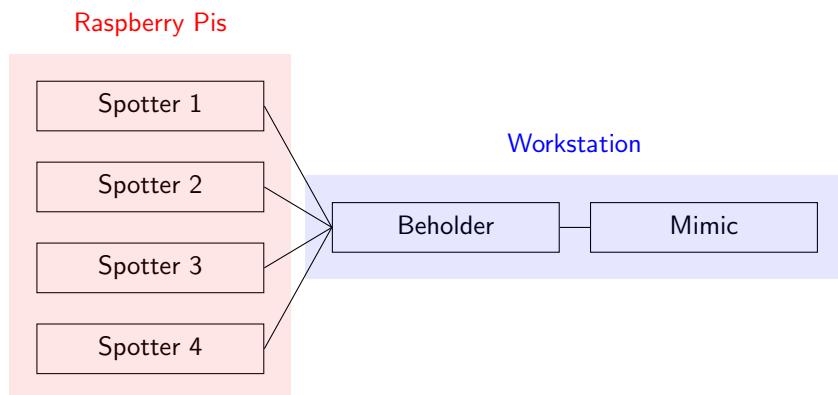


Abbildung 4.1.: Konzept für die Softwarearchitektur

## 4.1. Hardware

Um einen Marker im dreidimensionalen Raum zu triangulieren ist die Koordinatenangabe relativ zu mindestens zwei bekannten Punkten, in unserem Fall Kameras, nötig. Optimal wäre es aber, wenn 3 Messpunkte zur Verfügung stehen würden. Um von jedem Punkt eines Menschen zwei Bilder zu erhalten, sind also mindestens vier Kameras nötig (Abbildung 4.2), welche gleichmäßig in genügendem Abstand um die Person herum stehen. Aus diesem Grund

wurden vier Kameras als Startkonfiguration definiert. Sollten sich Probleme ergeben, zum Beispiel wenn Marker auf einem Bild zu oft verdeckt werden, werden zwei zusätzliche Kameras beschafft.

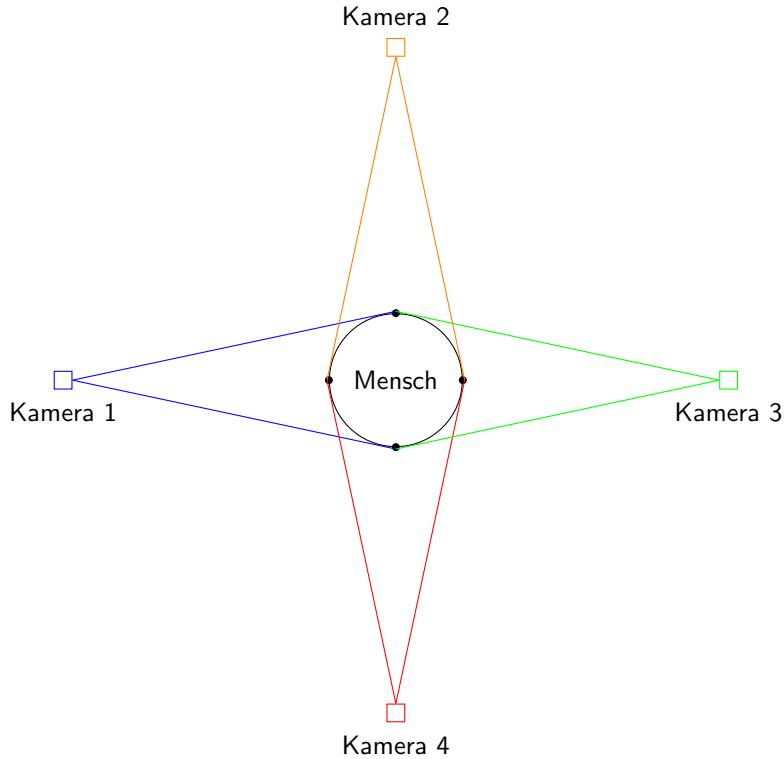


Abbildung 4.2.: Damit jeder Punkt eines Menschen von mindestens zwei Kameras gesehen wird, sind mindestens vier Kameras nötig.

#### 4.1.1. Marker

Um auf einem Bild die Marker sauber erkennen zu können müssen sie eindeutig segmentierbar sein. Eine Möglichkeit wäre es LEDs als Marker zu verwenden. Dies wären so genannte "aktive Marker". Je nach gewählter Lichtfarbe und der möglichen Intensität wären sie auf einem Bild sehr einfach zu segmentieren. Der grosse Nachteil der aktiven Marker ist die nötige Elektronik. Es würde bedeuten, dass der Schauspieler, der getrackt wird, etliche Kabel und eine Stromversorgung am Körper tragen muss. Die andere Möglichkeit sind passive Marker, also ohne elektronische Komponente. Es bedeutet, dass sie auf Beleuchtung von aussen angewiesen sind. Die einfachste Möglichkeit ist die Verwendung von Retroreflektoren als Marker. Retroreflektoren sind in der Lage, einfallende elektromagnetische Wellen grösstenteils in die Ursprungsrichtung zurückzuwerfen. Dieses Prinzip wird vielfach im Alltag verwendet, zum Beispiel bei "Katzenaugen" an Velos [4]. Die Verwendung von Retroreflektoren zwingt uns dazu, eine Lichtquelle zu verwenden, die möglichst nahe an der Kameralinse ist. Der Vorteil ist aber, dass sie autark funktionieren.

Um möglichst unabhängig vom Umgebungslicht zu sein, werden wir Licht im nahen Infrarotbereich verwenden [5]. So haben wir die Möglichkeit den Aufnahmefeld auszuleuchten ohne allzu viele Störungen bei der Segmentierung der Marker zu generieren.

#### 4.1.2. Kamera

Wir haben bereits im Vorprojekt unser Kamerasystem evaluiert und getestet. Da der Hauptteil der Arbeit nicht um Bau der Hardware sondern in der Programmierung liegen sollte, haben wir uns für ein sehr einfaches Kamerakonzept entschieden. Als Basis soll ein Raspberry Pi dienen. Darauf aufbauend wird das NoIR Kamera Modul in der Version 2 verwendet und eine Bright Pi LED-Komponente wird als Lichtquelle dienen. Diese Komponenten ergeben schlussendlich unser Kamerasystem. Dieses System wurde im Vorprojekt bereits als Prototyp gebaut und mit einem

Retroreflektor getestet. Das Ergebnis dieses Vorprojekts war vielversprechend. Auf dieser Basis wurde entschieden, dass dieses System so für die Thesis verwendet wird und nur noch im Notfall Komponenten ausgetauscht werden. Die Komponenten und deren Zusammenspiel soll so weit es die Zeit erlaubt, optimiert werden. Die Optimierung der Hardware hat aber wie gesagt keine Priorität in diesem Projekt.

## 4.2. Netzwerkkomponente

Um volle Kontrolle über die Netzwerkkomponente zu behalten und eine möglichst schlanke Architektur zu behalten wurde entschieden, dass die Daten als roher Bytestream übertragen werden. Dazu wurde uns die Library ZeroMQ (ZMQ, 0MQ oder ØMQ) empfohlen. ZMQ ist eine Library zum asynchronen Nachrichtenaustausch. Es bietet eine Abstraktionsschicht, so dass nicht mit RAW Sockets und deren Problemen gearbeitet werden muss. Um bestimmte Patterns möglichst effizient zu realisieren hat ZMQ built-in Sockettypen. Diese Sockets haben beispielsweise je eigene Routinen, welche socket flooding oder multiple connected peers handhaben [6]. ZMQ gilt gemeinhin als sehr performant und verlässlich [7]. Es wird von fast allen gängigen Sprachen unterstützt. Für Dotnet Sprachen existiert eine direkte Portierung namens NetMQ, welche wir im Mimic einbinden können.

Um nicht für jeden Verwendungszweck einen eigenen Socket eröffnen zu müssen, muss ein simples Netzwerkprotokoll entwickelt werden. Dieses Protokoll soll es erlauben, jegliche Nachrichtentypen (Handshake, Payload, Commands, usw.) über den gleichen Socket zu versenden, respektive auf der Gegenseite korrekt zu empfangen.

Die ZMQ Funktion zum versenden von Nachrichten über einen entsprechenden Socket sieht folgendermassen aus:

```

1 int zmq_send(void* socket ,
2             const void* buffer ,
3             size_t length ,
4             int flags);

```

Listing 4.1: Funktionsheader von zmq\_send

Die Parameterliste der Empfangsfunktion ist exakt dieselbe.

Die Daten liegen nach dem Empfangen lediglich als Pointer vor und es ist bekannt, dass es sich um einen Bytearray handelt. Um die Daten weiter zu verarbeiten müssen sie in die entsprechenden Datentypen, gecastet werden. Da der Inhalt der Nachrichten aber variabel ist, ist ein willkürlicher Cast nicht empfehlenswert. Es muss also zuerst eruiert werden, welche Daten, respektive Datentypen in der Nachricht enthalten sind. Dazu wird ein Nachrichtenheader verwendet. Dieser ist jeder Nachricht vorangestellt und ist immer gleich aufgebaut, so kann er auch sicher aus dem Nachrichtenanfang gecastet werden. Der Nachrichtenheader gibt dann Aufschluss darüber, wie die Payload der Nachricht gecastet werden muss.

Das von uns vorgeschlagene Protokoll sieht folgendermassen aus:

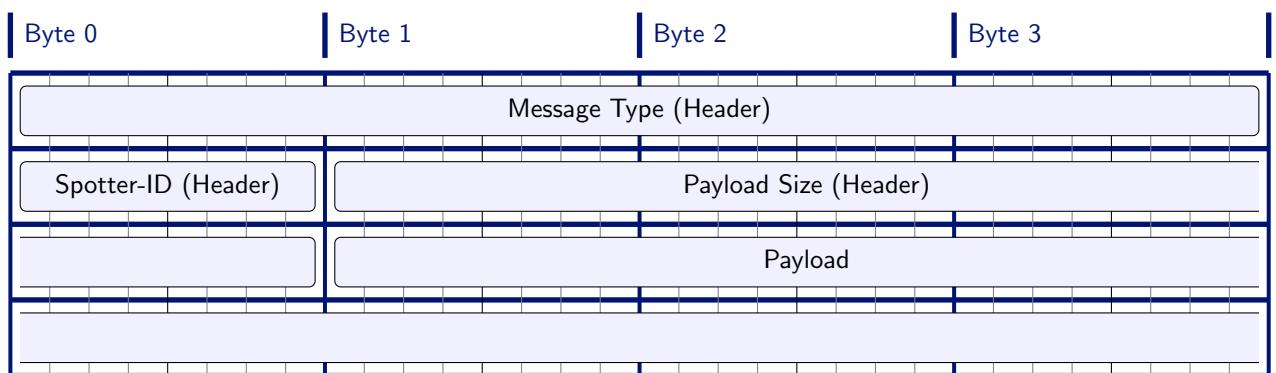


Abbildung 4.3.: Aufbau des Mimikry Data Protocol

|                                       |   |
|---------------------------------------|---|
| <b>Messagetype</b>                    | Der Messagetype gibt an, was für Daten sich in der Payload befinden.  |
| <b>Spotter-ID</b>                     | Die Spotter-ID gibt an, von welchem Spotter die Nachricht kommt (Nachricht von einem Spotter) oder an welchen Spotter die Nachricht gerichtet ist (Nachricht an einen Spotter). Ist die Spotter-ID 0, hat der Spotter keine ID oder die Nachricht ist an alle Spotter gerichtet. Wenn der Nachrichtentyp keine Spotter-ID verlangt, ist die ID ebenfalls 0. |
| <b>Payload Size</b>                   | Die Payload Size gibt an, wieviel Byte Payload in der Nachricht enthalten sind. Aus diesem Wert und dem Message Type lässt sich zum Beispiel die Anzahl Elemente errechnen.   |
| <b>Payload</b>                        | Die eigentlichen Daten gemäss Message Type.   |
| Angedacht sind folgende Messagetypen: |   |
| <b>Hello</b>                          | Wird zum Anmelden der Spotter beim Beholder verwendet.  |
| <b>Payload</b>                        | Wird für die Übertragung der Strahlen von den Spottern zum Beholder und der Modellpunkte vom Beholder zum Mimic verwendet.  |
| <b>Command</b>                        | Wird zur Befehlsgabe vom Mimic an den Beholder und an die Spotter verwendet.  |

ZMQ bietet mit den verschiedenen Sockets die Möglichkeit unterschiedliche Netzwerkpatterns optimal abzubilden. Die angedachte Architektur ist ein Client-Server Pattern (Spotter - Beholder). Die Verbindung wird jedoch nicht klassisch zum Server hin aufgebaut und dieser gibt entsprechend Antwort, sondern es wird ein konstanter Datenstrom von den Clients zum Server geschickt. Es handelt sich also eher um ein Pipeline Pattern. Die Daten werden danach vom Beholder zum Mimic weitergeschickt und zwar ebenfalls als kontinuierlicher Datenstrom, deshalb ist auch hier eine Art Pipeline Pattern zu erwarten. Um die Architektur optimal zu unterstützen haben wir uns für Push-Pull Sockets für alle Datenübertragungen entschieden. Dieses Socketpaar unterstützt nur Einwegkommunikation. Dies bedeutet, dass wir für Befehlsübertragung zu den Spottern ebenfalls ein Socketpaar in die andere Richtung brauchen. Für die Befehlsübertragung haben wir uns für Publish-Subscriber Sockets entschieden. So können Befehle sehr einfach an alle Spotter aber auch an einzelne gesendet werden. Da die Befehle im Mimic ausgelöst werden, wird auch vom Mimic zum Beholder ein Socketpaar initialisiert, jedoch der Einfachheit halber Push-Pull Sockets.

## 4.3. Markerkonfiguration

Um das Testen des Systems möglichst einfach zu halten, haben wir eine Minimalkonfiguration der Marker definiert, mit welcher ein humanoides Modell animiert werden kann. Diese besteht aus 13 Markern, welche, wie in Abbildung 4.4 ersichtlich, platziert sind.

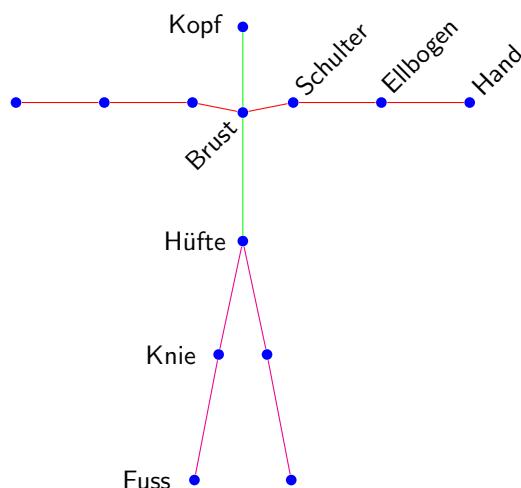


Abbildung 4.4.: Markerkonfiguration

## 4.4. Kamerasoftware Spotter

Auf jedem Raspberry Pi soll eine Software laufen, welche für die Markerdetektion zuständig ist. Dazu sollen Kamerabilder abgegriffen und darauf helle Ansammlungen von Pixeln (Blobs) segmentiert werden. Durch das Zentrum dieser Blobs soll vom Kameraursprung aus ein Strahl geschickt werden. Diese Strahlen werden dann zur Weiterverarbeitung an den Beholder gesandt.

Das Programm wird in C++ geschrieben sein. Für die Bildaufnahme wollen wir v4l2 (Video for Linux 2), für die Bildverarbeitung OpenCV verwenden. Für die Netzwerkübertragung soll, wie im Kapitel 4.2 beschrieben, ZMQ verwendet werden.

## 4.5. Serversoftware Beholder

Der Beholder soll von den auf den angeschlossenen Raspberry Pis laufenden Spottern Strahlen erhalten und anhand dieser Schnittpunkte berechnen. Die Schnittpunkte ergeben die Markerpositionen im dreidimensionalen Raum. Die so positionierten Marker soll der Beholder dann Modellpunkten zuweisen (siehe Kapitel 4.3). Die identifizierten und positionierten Modellpunkte sollen dann zur Visualisierung an den Mimic gesandt werden.

Der Beholder kann grundsätzlich auf einem eigenen Rechner laufen. Damit etwas Hardware gespart werden kann, soll er aber vorläufig auf demselben Rechner wie der Mimic laufen.

Das Programm wird in C++ geschrieben, wobei ZMQ, wie im Kapitel 4.2 bereits erwähnt, zur Netzwerkübertragung verwendet werden soll.

Der Name Beholder kommt von einer Kreatur aus dem pen-and-paper Rollenspiel Dungeons & Dragons. Da diese Kreatur einen grossen Kopf und viele kleine Augen hat, schien uns der Name passend (Abbildung 4.5).



Abbildung 4.5.: Ein Beholder [8]

## 4.6. Visualisierungssoftware Mimic

Für die Visualisierung der Ergebnisse soll Unity verwendet werden. Unity soll die vom Beholder berechneten Punkte erhalten und diese visualisieren. Das schlussendliche Ziel soll sein, ein beliebiges Modell aus dem "Asset Store" mit humanoidem Rig mit unserem Tracking-System animieren zu können. Optimalerweise würde ein eingepasstes Standard-Rig zum Beispiel aus Blender nicht angepasst werden müssen.

Es soll ebenfalls ein Interface im Unity erstellt werden, um das ganze System zu steuern. Es werden mehrere Konfigurationsschritte nötig sein um das ganze System zu starten. Deshalb sollen Steuerungselemente im GUI eingebaut werden um das System zu starten, zu beenden und die Pose Estimation durchzuführen.

Zusätzlich soll eine Möglichkeit gefunden werden Debuginformationen in die Visualisierung mit einzubringen. So soll nicht nur das Rig oder das Modell angezeigt werden, sondern auch die Position der Kameras, die erkannten Strahlen sowie die berechneten Schnittpunkte sichtbar gemacht werden können.

# 5. Umsetzung

Anhand des im Kapitel 4 vorgestellten Konzepts wurde die Umsetzung des Systems angegangen. Zunächst wurde geeignete Hardware evaluiert und anschliessend die im Konzept angesprochene Software umgesetzt.

## 5.1. Hardware

Wie bereits erwähnt wurde ein Prototyp der Kamera im Vorprojekt getestet. Wir verwenden folgende Hardwarekomponenten für unser System.

**Raspberry Pi Model 3 B+** Grundplatine

**NoIR v2 Kamera** Dies ist die Kamera. Es handelt sich um ein Äquivalent zur Originalkamera vom Raspberry Pi, jedoch ohne den IR Filter.

**Bright Pi** Dies ist ein Bausatz mit einem PCB, 8 IR LEDs, 4 weissen LEDs und einem Pinheader und muss selber zusammengelötet werden.

**Case** Für das Case wurde eine modulare Kamerabox verwendet. Die Seitenverkleidung wurde weg gelassen und die Frontplatte ist selber designed und hergestellt.

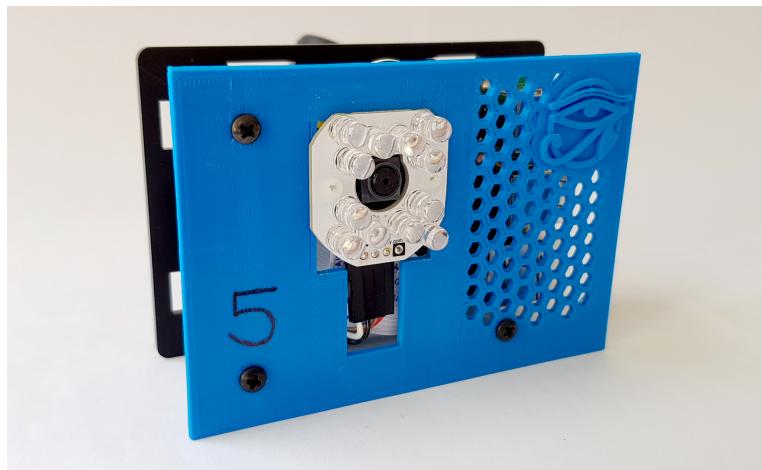


Abbildung 5.1.: Eine der sechs Kameras

Die Kamera wird über das mitgelieferte Flachbandkabel am entsprechenden Interface auf dem Raspberry Pi angeschlossen. Die Kamera kann mittels v4l2 (Video for Linux 2) angesteuert werden. Es sind mehrere Kameramodi verfügbar. Sie unterscheiden sich in Auflösung, Framerate und FOV. Tabelle 5.1 gibt Aufschluss über die Modi, welche die Kamera zur Verfügung stellt, und Abbildung 5.2 stellt die FOVs der Modis graphisch dar.

Die wichtigste Bedingung für die Aufnahme ist, dass das grösstmögliche Bild aufgezeichnet wird. Dies bedeutet, dass alle Modi, die nur ein partielles Field of View bieten, für unsere Zwecke nicht interessant sind. Die nächste Beschränkung ist das Erreichen der angepeilten 60 Frames auf jedem Komponenten des Systems. Dies ist in der aktuellen Konfiguration zwar nicht möglich, wir wählen aber trotzdem den bestmöglichen verfügbaren Wert. Wir haben uns schlussendlich für die Auflösung 1640x1232 mit 40 FPS und vollem FOV entschieden.

| Size      | Aspect Ratio | max. Framerate | FOV     | Binning |
|-----------|--------------|----------------|---------|---------|
| 1920x1080 | 16:9         | 30             | Partial | None    |
| 3280x2464 | 4:3          | 15             | Full    | None    |
| 3280x2464 | 4:3          | 15             | Full    | None    |
| 1640x1232 | 4:3          | 40             | Full    | 2x2     |
| 1640x922  | 16:9         | 40             | Full    | 2x2     |
| 1280x720  | 16:9         | 90             | Partial | 2x2     |
| 640x480   | 4:3          | 200            | Partial | 2x2     |

Tabelle 5.1.: Capture Modi der Raspberry Pi NoIR Kamera v2 [9]



Abbildung 5.2.: Die FOVs der verschiedenen Kameramodi [9]

Das Bright Pi wird über die GPIO Pins am Raspberry Pi angeschlossen und ist über I2C ansteuerbar. I2C ist ein Bussystem, das die Kommunikation von integrierten Schaltkreisen ermöglicht. I2C hat sich weitestgehend als Standard etabliert. [10]

Mit folgendem Befehl können zum Beispiel von der Shell aus alle LEDs eingeschaltet werden.

```
1 sudo i2cset -y 1 0x70 0x00 0xff
```

Listing 5.1: Shell command um alle LEDs auf dem Bright Pi einzuschalten

I2C muss vor der Verwendung auf dem Raspberry Pi eingeschaltet werden. Das genaue Setup der Raspberry Pi's ist im Anhang A beschrieben.

Die Frontabdeckung fürs Case ist eine Eigenproduktion. Designed wurde sie in Blender und mittels 3D-Druck hergestellt. Die genaue Vermassung ist ebenfalls im Anhang B zu finden.

Zur Erhöhung der Mobilität sind die Kameras aktuell mittels Kugelköpfen auf Stativen aufgestellt. Die Stromversorgung erfolgt mittels eines POE-Adapters, so müssen nicht Kabel für Strom und Netzwerk zu jeder Kamera verlegt werden.

### 5.1.1. Marker

Für die Marker haben wir verschiedene Herangehensweisen in Betracht gezogen. Retroreflektoren gibt es sowohl als Farbstoff mit Beimischung (Glaskugeln) oder als gestanzte Folie. Wir haben uns aufgrund der Verfügbarkeit für die Folienform entschieden. Die Firma OptiTrack verwendet für ihre Marker die Folie 7610 von 3M [11]. Da diese Folie in der Schweiz von keinem Fachhändler lieferbar ist, haben wir mehrere Alternativen getestet.



Abbildung 5.3.: Prototypen mit verschiedenen Folien. Links: 3M 983-10, Mitte: 3M 3210, Rechts: 3M 3150

Die 3M 983-10 Folie ist zu unflexibel und lässt sich nur schlecht um die Kugel legen. Die 3M 3210 reflektiert sehr viel schlechter als die anderen beiden Folien. Wir haben uns schlussendlich für die 3M 3150 entschieden. Diese reflektiert annähernd gleich gut wie die 3M 983-10, ist aber einiges flexibler.

## 5.2. Spotter

Die Spotter-Software, welche auf jedem Raspberry Pi läuft, nimmt folgende Aufgaben wahr:

- Abgreifen der Kamerabilder
- Ein- und Ausschalten der Bright-Pi LEDs
- Bestimmung der eigenen Kameramatrix anhand eines auf einem Kamerabild detektierten Schachbretts
- Detektion der Marker auf einem Kamerabild und Berechnen der Strahlen durch die Markermittelpunkte
- Übermitteln der Strahlen an die Serversoftware Beholder

Es bietet sich an, die Software in Komponenten, welche den einzelnen Verantwortlichkeiten entsprechen, zu gliedern. Diese Komponenten wären:

Eine **Core**-Komponente, welche systemabhängige sowie mathematische Funktionen zur Verfügung stellt. Auch die main-Funktion ist hier angesiedelt.

Eine **Capture**-Komponente, welche für die Aufnahme der Bilder verantwortlich ist. Dazu wird das v4l2-API (Video for Linux 2) benutzt.

Eine **Analyzation**-Komponente, welche Videobilder als Input erhält und darauf Operationen für die Systemkalibrierung und die Markerdetektion ausführt. Dazu wird die OpenCV-Library eingesetzt.

Eine **Transmission**-Komponente, welche für das Empfangen und Senden von Netzwerkpaketen verantwortlich ist. Das Konzept für die Netzwerkkomponente wurde bereits im Kapitel 4.2 vorgestellt. Wie dort erwähnt, wird hier die ZMQ-Library verwendet.

Eine **LED Control**-Komponente, welche für die Steuerung der BrightPI LEDs verantwortlich ist. Dies geschieht über das i2c GPIO Interface.

Die Spottersoftware wurde aufgrund dieser Komponenten gegliedert. Pro Komponente wurde ein Source- und ein Headerfile erstellt. Abbildung 5.4 stellt die Gliederung graphisch dar.

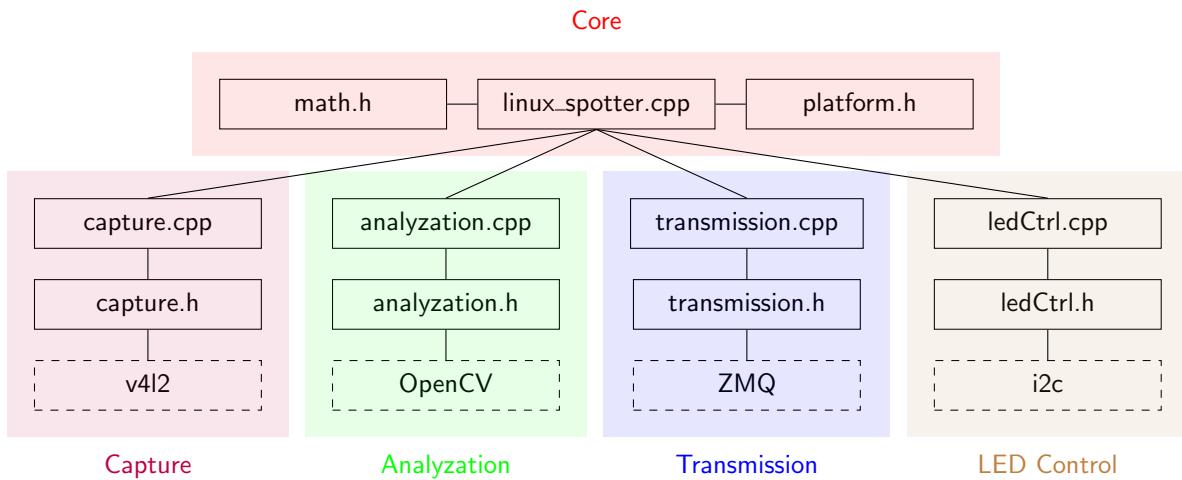


Abbildung 5.4.: Spotter Komponenten

### 5.2.1. Programmablauf

Der Programmablauf wurde in eine State machine modelliert (Abbildung 5.5). Diese startet im Status "Idle". Über das GUI in der Mimic Visualisierungssoftware (siehe Kapitel 5.4) kann der Befehl zum Starten der Systemkalibrierung ausgelöst werden. Dies versetzt die State machine in den Status "Estimating Pose". In diesem Status wird auf den Videobildern nach einem Schachbrett gesucht. Falls dieses gefunden wird, wird anhand der Schachbrettecken eine "Pose Estimation" (Positionsbestimmung) der Kamera vorgenommen (siehe Kapitel 5.2.5). Sobald die Pose Estimation erfolgreich abgehandelt wurde, geht die Applikation in den Status "Detecting" über. In diesem Status werden Markerpositionen extrahiert. Anhand dieser werden Strahlen bestimmt und an die Serverapplikation Beholder gesandt (siehe Kapitel 5.2.6). Über das GUI kann wiederum ein Befehl für die Systemkalibrierung gegeben werden, was zu einer Neukalibrierung führt. Außerdem ist es in allen Stati möglich durch einen Schliessbefehl in den Status "Exiting" zu gelangen. In diesem Status werden alle offenen Ressourcen (Files, Netzwerksockets) geschlossen und die Applikation beendet.

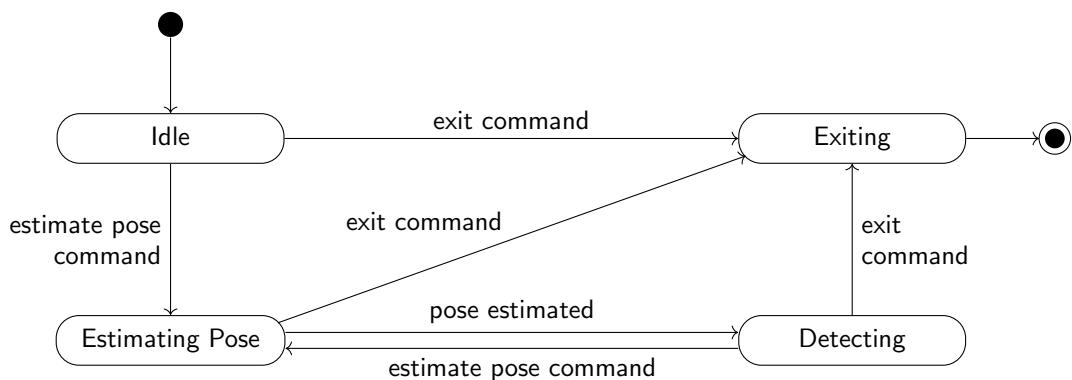


Abbildung 5.5.: Spotter State machine

### 5.2.2. Bildaufnahme

Die Bildaufnahme geschieht über das v4l2-API. Um dies zu verwenden muss das bcm-v4l2 Kernel Modul geladen sein, wie im Anhang A erklärt wird.

Angesteuert wird das v4l2-API, indem ein File geöffnet wird, welches im Linux Fielssystem einem Video Device entspricht. In unserem Fall ist dies immer /dev/video0.

```

1 int openVideoDevice(const char* deviceName) { // deviceName = "/dev/video0"
2     int result = open(deviceName, O_RDWR | O_NONBLOCK);
3     return result;
4 }
```

Listing 5.2: Öffnen eines Video Devices

Anschliessend kann der ioctl System Call für die Steuerung von v4l2 genutzt werden. v4l2 unterstützt eine ganze Reihe verschiedener Aufnahme- und Wiedergabemethoden sowie verschiedenste Framegrössen und Pixelformate [12]. Wir verwenden v4l2 im "video capture" Modus mit einer Framegröße von 1640x1232 Pixel (siehe Kapitel 5.1). Die Frames erhalten wir als Graustufenbild (Pixelformat "grey"). Die entsprechenden Einstellungen werden in einem v4l2\_format struct über ioctl gesetzt.

```

1 v4l2_format configureV4L2(int fd) { // fd = openVideoDevice("/dev/video0")
2     v4l2_format fmt = {};
3
4     fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
5     fmt.fmt.pix.width = 1640;
6     fmt.fmt.pix.height = 1232;
7     fmt.fmt.pix.pixelFormat = V4L2_PIX_FMT_GREY;
8     fmt.fmt.pix.field = V4L2_FIELD_ANY;
9
10    ioctl(fd, VIDIOC_S_FMT, &fmt);
11
12    return fmt;
13 }
```

Listing 5.3: v4l2 Einstellungen

v4l2 unterstützt verschiedene Methoden zur Speicher-Allokation [13]. Wir verwenden die "userptr" Methode. Bei dieser wird Speicher von der Client-Anwendung, also dem Spotter, alloziert und v4l2 Pointer auf diese Speicherbereiche übergeben. Dies hat den Vorteil, dass nur sehr wenig Daten zwischen v4l2 und dem Spotter ausgetauscht werden müssen, nämlich die Pointer auf die entsprechenden Speicherbereiche.

```

1 struct FrameBuffer {
2     void* memory;
3     int width, height;
4     int bytesPerPixel;
5     int pitch;
6     int size;
7 };
8
9 void allocateFrameBuffers(int fd,
10                         FrameBuffer* frameBuffer,
11                         int bufferCount,
12                         v4l2_format fmt) {
13     for (int i = 0; i < bufferCount; i++) {
14         v4l2_buffer buf = {};
15
16         frameBuffer->width = 1640;
17         frameBuffer->height = 1232;
18         frameBuffer->bytesPerPixel = 1;
19         frameBuffer->pitch = fmt.fmt.pix.bytesperline;
20         frameBuffer->size = fmt.fmt.pix.sizeimage;
21         frameBuffer->memory = malloc(frameBuffer->size);
22 }
```

```

23     buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
24     buf.index = i;
25     buf.memory = V4L2_MEMORY_USERPTR;
26     buf.m.userptr = frameBuffer->memory;
27     buf.length = frameBuffer->size;
28
29     frameBuffer++;
30 }
31 }
```

Listing 5.4: Allokation von v4l2 Buffern

Die Buffer müssen anschliessend für die Kamera zugänglich gemacht werden, indem sie auf die so genannte Bufferqueue gelegt werden.

```

1 void queueFrameBuffer(int fd,
2                         FrameBuffer* frameBuffer) {
3     v4l2_buffer buf = {};
4
5     buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
6     buf.memory = V4L2_MEMORY_USERPTR;
7     buf.m.userptr = frameBuffer->memory;
8     buf.length = frameBuffer->size;
9
10    ioctl(fd, VIDIOC_QBUF, &buf);
11 }
```

Listing 5.5: Queue Buffer

Um einen von der Kamera gefüllten Buffer auszulesen, muss dieser dequeued werden. Mit dem entsprechenden ioctl Aufruf wird der älteste gefüllte Buffer zurückgegeben. Idealerweise würde immer der neuste Buffer verwendet. Dazu könnte man alle Buffer von der Queue auslesen und nur den neusten behalten. Es hat sich allerdings gezeigt, dass dies einen negativen Einfluss auf die Performance hat und wurde aus diesem Grund wieder ausgebaut.

```

1 void dequeueFrameBuffer(int fd,
2                         FrameBuffer* frameBuffer) {
3     v4l2_buffer buf = {};
4
5     buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
6     buf.memory = V4L2_MEMORY_USERPTR;
7
8     ioctl(fd, VIDIOC_DQBUF, &buf);
9
10    frameBuffer->memory = buf.m.userptr;
11 }
```

Listing 5.6: Dequeue Buffer

Nach dem Verarbeiten des Frames muss der Buffer wieder auf die Queue gelegt werden. Der gesamte Ablauf sieht vereinfacht also folgendermassen aus:

```

1 #define BUFFER_COUNT 3
2
3 int main() {
4     int fd = openVideoDevice("/dev/video0");
5     v4l2_format fmt = configureV4L2(fd);
```

```

6     FrameBuffer buffers[BUFFER_COUNT];
7     allocateFrameBuffers(fd,
8                           &buffers[0],
9                           BUFFER_COUNT,
10                          fmt);
11
12
13    for (int i = 0; i < BUFFER_COUNT; i++) {
14        queueFrameBuffer(fd, &buffers[i]);
15    }
16
17    int bufferIndex = 0;
18
19    for (;;) {
20        FrameBuffer* buffer = &buffers[bufferIndex];
21        dequeueFrameBuffer(fd, &buffer);
22
23        // ...
24        // frame verarbeitung
25        // ...
26
27        queueFrameBuffer(fd, &buffer);
28        bufferIndex = (bufferIndex + 1) % BUFFER_COUNT;
29    }
30}

```

Listing 5.7: Ablauf Bildaufnahme

### 5.2.3. Synchronisierung der Bildaufnahme

Ein Problem bei der Bildaufnahme ist die zeitliche Differenz zwischen den Spottern. Es muss gewährleistet werden, dass die Spotter zu möglichst der gleichen Zeit ihre Bilder aufnehmen, damit die Marker an den selben Positionen detektiert werden. Dies wurde durch eine Netzwerksynchronisation erreicht. Der Beholder sendet den Befehl zur Bildaufnahme und die Spotter nehmen erst nach Erhalt dieses Befehls ein Bild auf (Abbildung 5.6).

Diese Art Synchronisation hat den Nachteil, dass immer eine Verzögerung, welche zumindest die Netzwerklatenz dauert, zwischen dem Versenden und der Bildaufnahme stattfindet. Da wir jedoch sowieso durch die Framerate der Kameras limitiert sind, ist der Einfluss dieser Verzögerung minimal. Bei Verwendung besserer Kameras müsste dieses Problem neu angegangen werden. Denkbar wäre dabei eine Synchronisation über GPIO oder über einen visuellen Hinweis, wie beispielsweise ein blinkendes Licht.

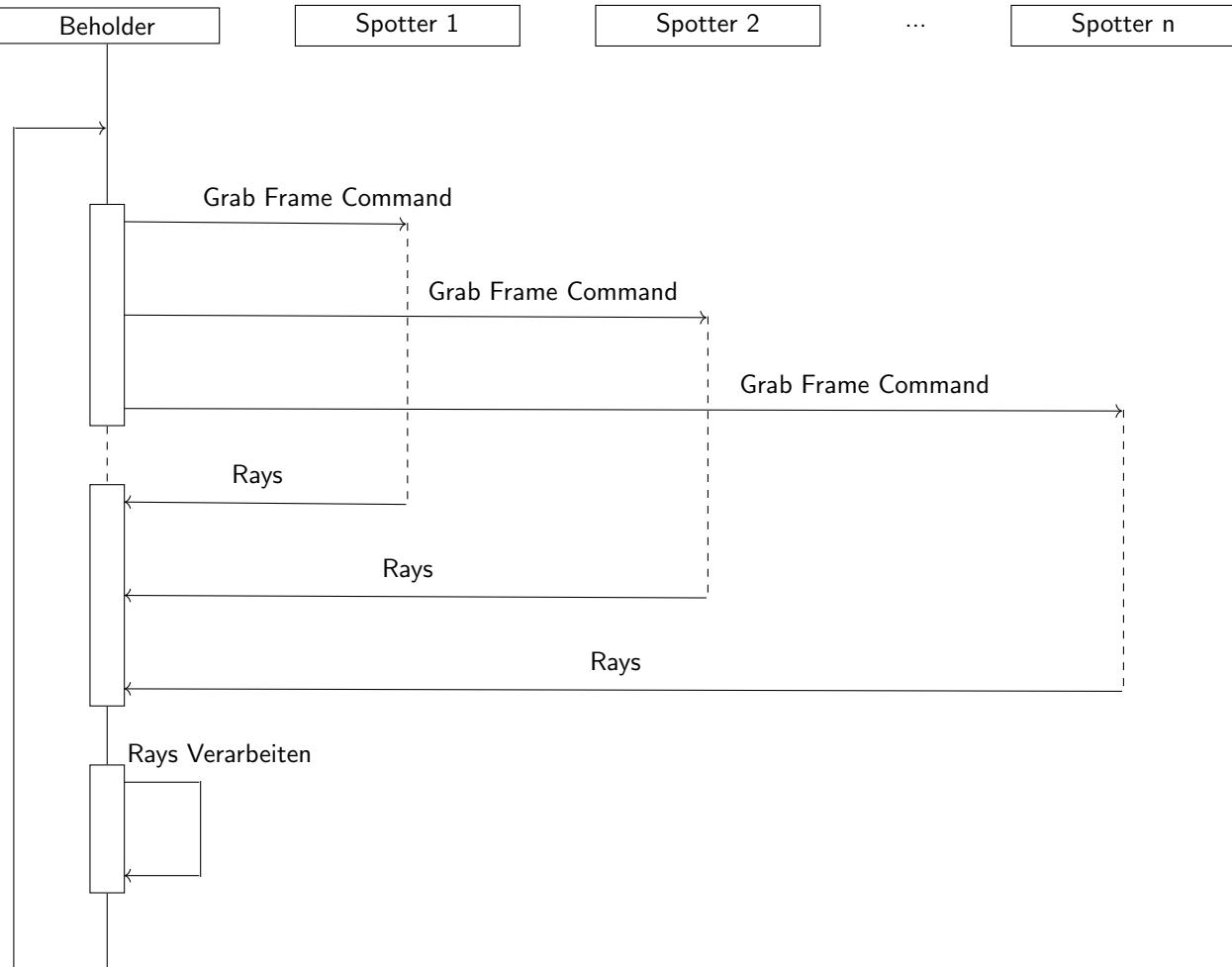


Abbildung 5.6.: Synchronisierung der Bildaufnahme

#### 5.2.4. Kontrolle der Bright Pi LEDs

Für die Steuerung der LEDs ist in C++ unter Linux keine zusätzliche Library nötig. Sie wird folgendermassen initialisiert.

```

1 void initI2C(I2CBus* brightPi) {
2     const char *filename = "/dev/i2c-1";
3     brightPi->file_i2c = open(filename, O_RDWR);
4
5     int addr = 0x70; // I2C Adresse
6     ioctl(brightPi->file_i2c, I2C_SLAVE, addr);
7 }
```

Listing 5.8: Initialisierung I2C

Beim Datentyp I2CBus handelt es sich um ein Struct, welches lediglich ein i32 beinhaltet.

Die Variable "brightPi" repräsentiert gewissermassen die I2C Adresse vom BrightPI. So kann nun mittels eines "write" die Subadresse und die Daten auf den Bus geschrieben werden.

```

1 void switchIROn(I2CBus* brightPi) {
2     i32 length = 0;
3     unsigned char buffer[2] = {};
4
5     buffer[0] = 0x00;
6     buffer[1] = 0xa5;
7     length = 2;
8     write(brightPi->file_i2c, buffer, length);
9 }
```

Listing 5.9: Snippet zum einschalten der IR LEDs

Wir haben in diesem Stil Funktionen implementiert um die LEDs ein- und auszuschalten. Weisse und IR LEDs können getrennt eingeschaltet werden. Bei der Pose Estimation werden die weissen LEDs eingeschaltet als eine Art Statusindikator. Die Lichtstärke der weissen LEDs ist zu gering, um einen merklichen Einfluss auf den Prozess zu haben. Sobald die Pose Estimation auf einem Spotter abgeschlossen ist, werden mit dem Wechsel in den "Detecting-State" die weissen LEDs ausgeschaltet und die IR LEDs werden eingeschaltet.

### 5.2.5. Kamerakalibrierung

Damit die Strahlen, welche von den Spottern durch die Marker geschickt werden, zur Schnittpunktberechnung verwendet werden können, müssen sie alle in einem gemeinsamen Koordinatensystem sein. Um dies zu erreichen muss jeder Spotter seine eigene Position in diesem Koordinatensystem kennen. Der Vorgang, diese Information herauszufinden, wird "Pose Estimation" genannt.

Die Pose Estimation resultiert in einer Transformationsmatrix  ${}^cT_w$ , genannt Kameramatrix. Durch eine Multiplikation mit dieser kann ein Punkt im Weltkoordinatensystem in das Kamerakoordinatensystem transformiert werden und durch die Multiplikation mit der inversen  ${}^cT_w^{-1} = {}^wT_c$  kann ein Punkt im Kamerakoordinatensystem in das Weltkoordinatensystem transformiert werden.  ${}^cT_w$  kann als Multiplikation zweier Matrizen ausgedrückt werden: einer intrinsischen  $K$ , mit der Fehler im Kamerasytem ausgedrückt werden, sowie einer extrinsischen  $T$ , welche die effektive Translation und Rotation der Kamera im Bezug zum Ursprung des Weltkoordinatensystems beschreibt. Formel 5.1 beschreibt die gesamte Kameratransformation eines Punktes  $p_w$  in Weltkoordinaten in einen Punkt  $p_c$  in Kamerakoordinaten.

$$p_c = {}^cT_w \cdot p_w = K \cdot T \cdot p_w = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_x \\ w_y \\ w_z \\ 1 \end{bmatrix} \quad (5.1)$$

|                 |                        |
|-----------------|------------------------|
| $f_x, f_y$      | Brennweite [px]        |
| $c_x, c_y$      | Optisches Zentrum [px] |
| $r_{ij}$        | Rotationskomponente    |
| $t_x, t_y, t_z$ | Translationskomponente |
| $w_x, w_y, w_z$ | Weltkoordinaten        |

Bei einer echten Kamera ergibt sich durch den Schliff der Linse ausserdem ein Fehler, die Distortion. Dieser kann jedoch durch die Anwendung eines Distortionsmodells, wie das von OpenCV angewandte Modell von Jean-Yves Bouguet, korrigiert werden. [14]

Die intrinsische Komponente  $K$  sowie die Distortionskoeffizienten werden vorgängig durch ein eigens entwickeltes Tool bestimmt. Dabei werden 30 Bilder eines Schachbretts mit der zu kalibrierenden Kamera geschossen (Abbildung 5.7).

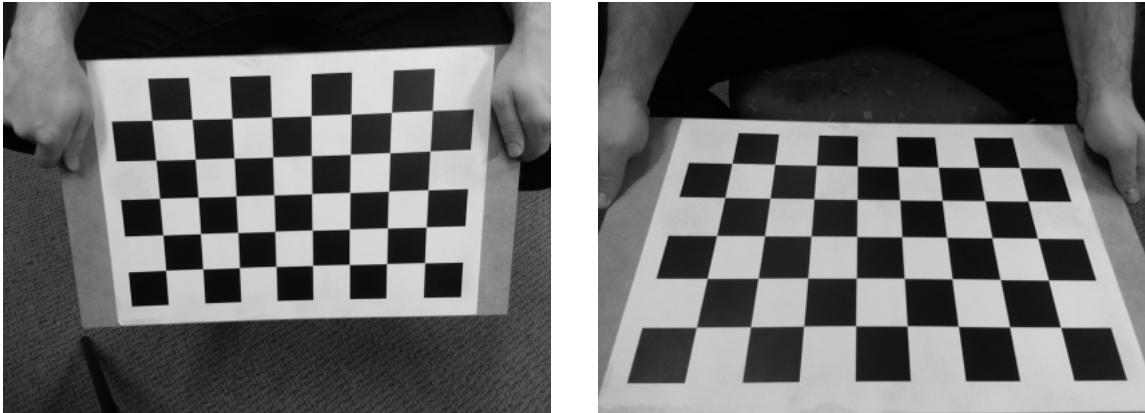


Abbildung 5.7.: Beispiele für Bilder, welche für die intrinsische Kalibrierung verwendet wurden.

Die Schachbrettecken werden mit OpenCV detektiert und es wird eine Korrespondenz zu einer definierten Weltkoordinate hergestellt. Dies ist möglich, da das Schachbrett nicht symmetrisch ist und jede Schachbrettecke so eindeutig identifiziert werden kann. Um eine möglichst genaue Kalibrierung zu erreichen wird für jede Schachbrettecke die subpixelgenaue Koordinate bestimmt.

```

1 #define CALIBRATION_FRAME_COUNT 30
2 #define CHESSBOARD_COLUMNS 8
3 #define CHESSBOARD_ROWS 5
4
5 std::vector allCameraPoints, allWorldPoints;
6 std::vector worldPoints =
7     calculateWorldPoints(); // Weltkoordinaten der Schachbrettecken
8
9 int frameIndex = 0;
10 while (frameIndex < CALIBRATION_FRAME_COUNT) {
11     cv::Mat img = readFrameFromCamera(); // Neuestes Kamerabild
12
13     // Detektieren der Schachbrettecken
14     bool cornersFound =
15         cv::findChessboardCorners(img,
16             cv::Size(CHESSBOARD_COLUMNS, CHESSBOARD_ROWS),
17             cameraPoints,
18             cv::CALIB_CB_ADAPTIVE_THRESH |
19             cv::CALIB_CB_NORMALIZE_IMAGE |
20             cv::CALIB_CB_FAST_CHECK);
21
22     if (cornersFound) {
23         // Subpixelgenaue Koordinaten
24         cv::cornerSubPix(img,
25             cameraPoints,
26             cv::Size(11, 11),
27             cv::Size(-1, -1),
28             cv::TermCriteria(cv::TermCriteria::EPS +
29                             cv::TermCriteria::COUNT,
30                             30000,
31                             0.01));
32
33         allCameraPoints.push_back(cameraPoints);
34         allWorldPoints.push_back(worldPoints);
35
36         frameIndex++;
37     }
38 }
```

```

37     }
38 }
```

Listing 5.10: Aufnahme der Schachbrettkoordinaten für die intrinsische Kamerakalibrierung

Durch die bekannte Korrespondenz von Kamera- und Weltpunkt wird ein Gleichungssystem aufgestellt. So können die unbekannten Parameter bestimmt werden. Dies wird für jede Kamera einmal gemacht. Dies ist in der OpenCV Funktion calibrateCamera umgesetzt. Die resultierenden Parameter werden anschliessend in einem File gespeichert und können so immer wieder geladen werden.

```

1 #define FRAME_WIDTH 1640
2 #define FRAME_HEIGHT 1232
3
4 cv::Mat cameraMat = cv::Mat::zeros(3, 3, CV_64F);
5 cv::Mat distortionMat = cv::Mat::zeros(5, 1, CV_64F);
6 std::vector<cv::Mat> rvecs, tvecs;
7
8 cv::calibrateCamera(allWorldPoints,
9                     allCameraPoints,
10                    cv::Size(FRAME_WIDTH, FRAME_HEIGHT),
11                    cameraMat,
12                    distortionMat,
13                    rvecs, tvecs);
```

Listing 5.11: Bestimmen der Kameramatrix

Um die extrinsische Matrix  $T$  zu bestimmen, müssen alle Kameras gleichzeitig auf ein Schachbrett sehen, dem sogenannten Kalibrierungsziel. Dies dient der Festlegung eines gemeinsamen, globalen Koordinatensystems (siehe Abbildung 5.8).

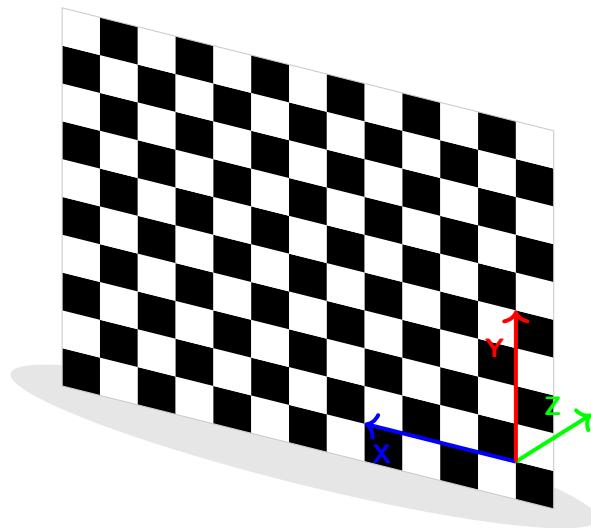


Abbildung 5.8.: Koordinatensystem definiert durch ein Schachbrett. Wenn das Schachbrett auf dem Boden liegt, zeigt die Z-Achse nach unten.

Wiederum wird jeder Schachbrettkantecke eine Koordinate im Weltkoordinatensystem zugewiesen. Aufgrund der Korrespondenzen der detektierten Schachbrettkanten in Kamerakoordinaten und der bekannten Weltkoordinaten wird ein Gleichungssystem aufgestellt und gelöst. Damit das Kalibrierungstarget auch auf mehrere Meter Distanz gut sichtbar bleibt, wurden zwei Schachbretter mit einer Kantenlänge von 12.1cm in DIN-A0 Grösse (841mm x 1189mm) ausgedruckt und zusammengeklebt (Abbildung 5.9). So resultiert ein Schachbrett mit 11 Reihen und 8 Kolonnen.



Abbildung 5.9.: Das System mit dem Kalibriertarget

Die Weltkoordinaten der Schachbrettecken wurden so definiert, dass die X-Achse den Reihen entspricht und die Y-Achse den Kolonnen. Die Z-Achse zeigt in den Boden. Die Weltkoordinaten sind in cm angegeben.

Die Bestimmung der extrinsischen Kameramatrix geschieht mit der OpenCV Funktion solvePnP.

```

1 // ...
2 // Detektierung der Schachbrettecken analog zur intrinsischen Kalibrierung
3 // ...
4
5 cv::Mat rvec, tvec;
6 cv::solvePnP(worldPoints,
7             cameraPoints,
8             cameraMat,
9             distortionMat,
10            rvec, tvec);

```

Listing 5.12: Pose Estimation

solvePnP gibt die Rotation als Euler-Rodrigues Parameter in einem dreidimensionalen Vektor zurück [15]. Um den Vektor in eine Rotationsmatrix zu transformieren, muss man die OpenCV Funktion Rodrigues anwenden.

Es ist wichtig für die Pose Estimation ein möglichst hochauflösendes Bild zu verwenden. Der Fehler, welcher entsteht, wenn man ein Bild mit kleinerer Auflösung verwendet, hat einen sehr negativen Einfluss auf die Genauigkeit der Markerdetektion. Dies ist damit zu begründen, dass die Positionen der Schachbrettecken nicht mehr so präzise detektiert werden können. Der Fehler kann mit Formel 5.2 berechnet werden. Die horizontale Sensorbreite  $s_x$  der von uns verwendeten Kamera ist 3.68mm und die Fokuslänge  $f$  beträgt 3.04mm [16].

$$\Delta_x = \frac{(s_x/p_x) \cdot d}{f} \quad (5.2)$$

- $\Delta_x$  Horizontaler Fehler [m]
- $s_x$  Horizontale Sensorbreite [m]
- $p_x$  Horizontale Pixelanzahl [px]
- $d$  Distanz von der Kamera zur Schachbrettecke [m]
- $f$  Fokuslänge der Kamera [m]

Bei einem Bild mit der von uns verwendeten Auflösung von 1640x1232 Pixeln und einer Distanz von der Kamera zur Schachbrettecke von 2.1m ist der Fehler bei einer Abweichung von einem Pixel für jede Schachbrettecke 1.55mm. Bei einem Bild mit einer Auflösung von 410x308 ist der Fehler bei gleichbleibender Abweichung 6.2mm, also um Faktor 4 schlechter.

Sobald die Kameramatrize bestimmt wurde, wird diese im Applikationsstatus gespeichert. Um im Falle eines Neustartes des Systems die Positionsbestimmung nicht neu machen zu müssen, wird die Kameramatrize außerdem in einem File gespeichert. Beim Programmstart kann über eine Kommandozeilenoption angegeben werden, dass die Kameramatrize von diesem File geladen werden soll. Anschliessend geht die Applikation in den Status "Detecting" über.

### 5.2.6. Markerdetektierung

Im Status "Detecting" werden Markerpositionen bestimmt. Dazu wird ein Videoframe zunächst binarisiert, sprich alle Pixel mit einer Helligkeit über einem bestimmten Schwellwert werden weiß, alle anderen schwarz (Abbildung 5.10).

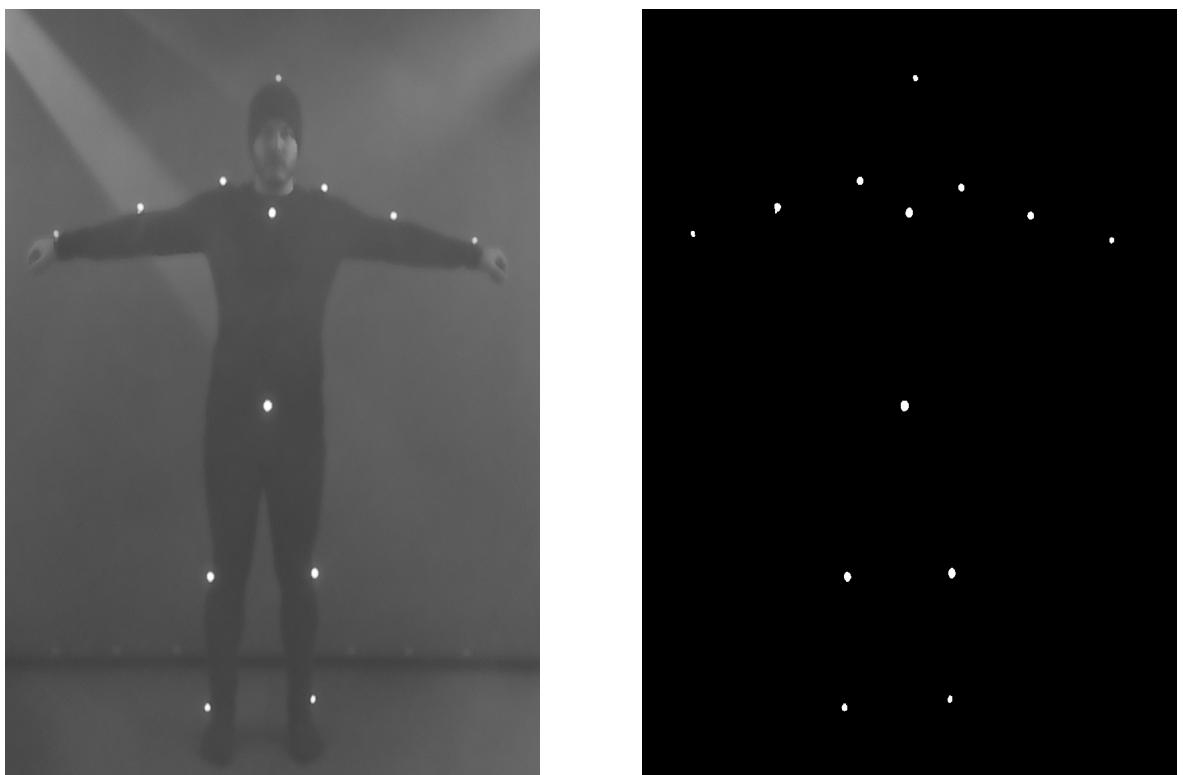


Abbildung 5.10.: Links das Graustufenbild, rechts dasselbe Bild binarisiert mit einem Schwellwert von 160.

Anschliessend werden die Konturen von Blobs (zusammenhängende weiße Pixel) gefunden und deren Zentrum bestimmt. Nahe zusammenliegende Zentren werden gruppiert und ein finales Zentrum bestimmt. Dies wird im Simple Blob Detector der OpenCV Library umgesetzt [17].

```
1 cv::SimpleBlobDetector::Params params;  
2  
3 // Maximum und Minimum Graustufenwert (Werte zwischen 0 und 255)  
4 params.minThreshold = 120;  
5 params.maxThreshold = 121;  
6 params.minRepeatability = 1;  
7  
8 // Wir detektieren helle Flächen
```

```

9  params.filterByColor = 1;
10 params.blobColor = 255;
11
12 // Ausschliessen sehr grosser Flächen
13 params.filterByArea = 1;
14 params.minArea = 1;
15 params.maxArea = 500;
16
17 cv::Ptr<cv::SimpleBlobDetector> detector =
18   cv::SimpleBlobDetector::create(params);
19
20 // Output ist ein Vektor von Bildkoordinaten
21 std::vector<cv::KeyPoint> keypoints;
22 detector->detect(img, keypoints);

```

Listing 5.13: Markerdetektion mit cv::SimpleBlobDetector

Jeder so detektierte Punkt  $p_c$  wird anschliessend mit einer Multiplikation mit der inversen Kameramatrix  ${}^wT_c$  ins Weltkoordinatensystem transformiert.

$$p_w = {}^wT_c \cdot p_c \quad (5.3)$$

Aus diesen Punkten wird ein Strahl (Ray) generiert. Ein Strahl ist eine halbe Gerade mit einem Ursprung  $o_w$  und einer Richtung  $\vec{d}$  [18]. Die von uns generierten Strahlen haben ihren Ursprung beim Kameraursprung, welcher durch die Multiplikation des Nullpunkts im Kamerakoordinatensystem mit der inversen Kameramatrix bestimmt wird.

$$o_w = {}^wT_c \cdot o_c = {}^wT_c \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (5.4)$$

Die Richtung wird durch das Abziehen des Ursprungs  $o$  vom Weltkoordinatenpunkt  $p_w$  bestimmt. Der resultierende Vektor wird noch normalisiert.

$$\vec{d} = \frac{p_w - o_w}{\|p_w - o_w\|} \quad (5.5)$$

### 5.2.7. Übermittlung der generierten Strahlen

Die generierten Strahlen werden dann an die Serversoftware Beholder gesandt. Dies geschieht über das im Kapitel 4.2 definierte Netzwerkprotokol. Abbildung 5.11 zeigt den Aufbau eines entsprechenden Netzwerkframes.

## 5.3. Beholder

Der Beholder erhält von allen angeschlossenen Spottern Strahlen zu detektierten Markern. Er ist dafür verantwortlich, Schnittpunkte zu berechnen und so die Markerpositionen im dreidimensionalen Raum zu bestimmen. Die so bestimmten Markerpositionen werden anschliessend vom Beholder auf ein humanoides Modell gepasst. Zusätzlich gibt der Beholder den Takt für die Bilddurchsuche der Spotter vor (siehe Kapitel 5.2.2).

Das Programm wurde in folgende Komponenten strukturiert:

Eine **Core**-Komponente, welche systemabhängige und mathematische Funktionen zur Verfügung stellt. Einige der Funktionen, insbesondere die mathematischen, sind identisch mit den Funktionen, welche in der Spotter Software verwendet werden (siehe Kapitel 5.2). Diese liegen in einem gemeinsamen Sourcefile.

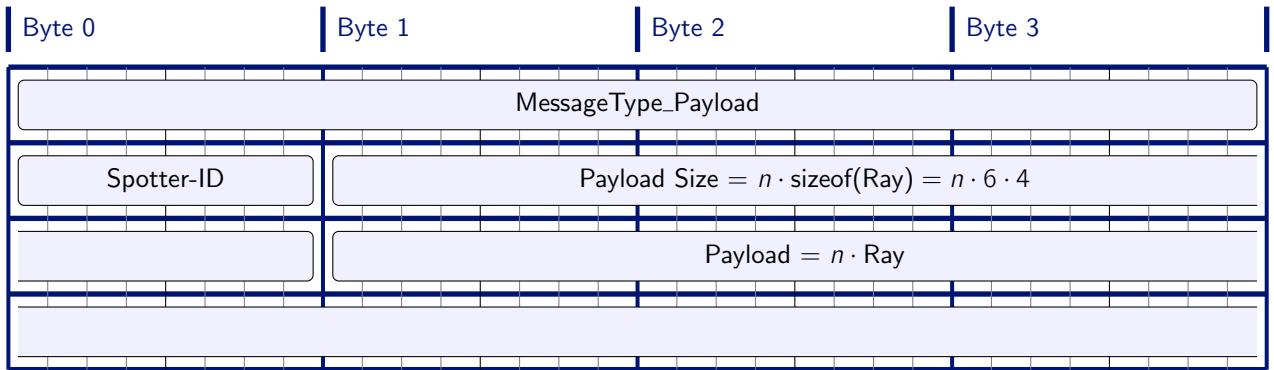


Abbildung 5.11.: Aufbau einer Spotter Payload Message für  $n$  Strahlen

Eine **Messagehandler**-Komponente, welche eingehende Nachrichten der Spotter und des Mimics erhält und verarbeitet, sowie ausgehende Nachrichten versendet.

Eine **Marker-Extraction**-Komponente, welche für die Bestimmung der Markerpositionen anhand der eingehenden Strahlen verantwortlich ist.

Eine **Rigging**-Komponente, welche die Marker an Modellpunkte zuweist.

Die Komponenten sind in Abbildung 5.12 graphisch dargestellt.

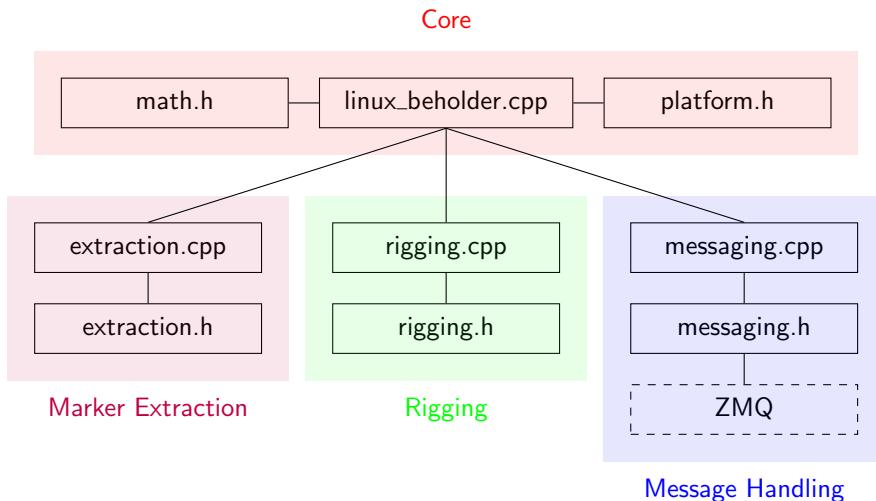


Abbildung 5.12.: Beholder Komponenten

### 5.3.1. Programmablauf

Damit die Verarbeitung der Rays möglichst zeitnah geschieht, läuft das Programm parallel in zwei Threads. Ein Thread, der **Nachrichtenthread**, ist nur für die Verarbeitung der eingehenden Netzwerknachrichten verantwortlich, der andere, der **Verarbeitungsthread**, für die Bestimmung der Markerposition und das Zuweisen der Marker auf die Modellpunkte.

Der Nachrichtenthread verarbeitet alle eingehenden Netzwerkpakete. Vom Mimic erhält er Steuerungsbefehle um die Pose Estimation und das Modelmatching zu starten oder das System zu stoppen. Befehle, welche den Beholder nicht betreffen, wie zum Beispiel der Befehl zum Starten der Pose Estimation, werden an die Spotter weitergeleitet. Von den Spottern erhält der Nachrichtenthread die Rays. Diese werden in einem Buffer abgelegt, bis sie der Verarbeitungsthread abholt.

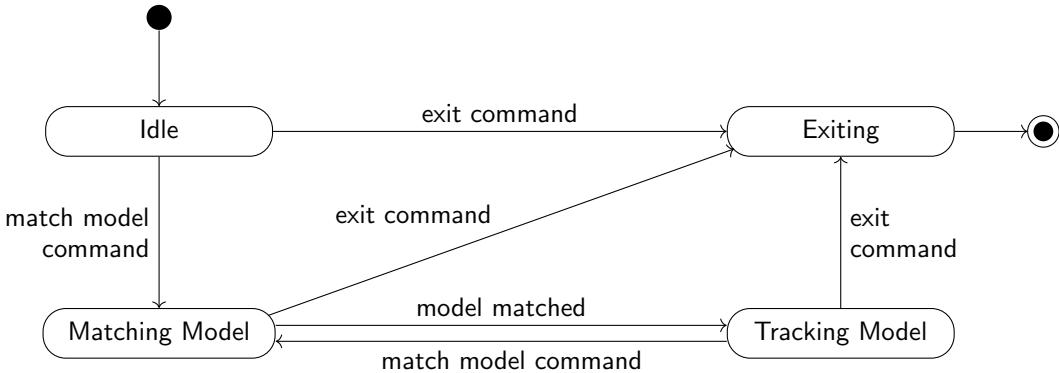


Abbildung 5.13.: Beholder Verarbeitungsthread Statemachine

Der Verarbeitungsthread ist "Idle", solange das System nicht kalibriert, sprich die Pose Estimation nicht auf allen Spottern durchgeführt ist. Sobald das System kalibriert ist, kann vom Mimic der Befehl für das Modelmatching ausgelöst werden. Zu diesem Zeitpunkt erhält der Beholder bereits Rays von den Spottern. Sobald der Befehl ausgelöst wurde, beginnt der Beholder Schnittpunkte zu berechnen und diese den Modellpunkten zuzuweisen (Kapitel 5.3.3). Wenn für alle Modellpunkte ein geeigneter Kandidat gefunden wurde, geht der Beholder in den Modeltracking Modus über (Kapitel 5.3.4).

### 5.3.2. Schnittpunktberechnung

Um die Position der Marker im Raum festzustellen, berechnen wir die Schnittpunkte der von den Spottern generierten Strahlen. Das grösste von uns festgestellte Problem bei dieser Methode ist, dass es viele false-positives gibt, also Stellen, an denen sich zwar Strahlen schneiden, aber kein Marker vorhanden ist. Dies hat hauptsächlich damit zu tun, dass wir, wegen der Ungenauigkeit beim Generieren der Strahlen und Fliesskommazahlfehlern (floating point imprecision), nicht davon ausgehen können, dass sich Strahlen genau schneiden. Deshalb müssen wir einen Schwellwert definieren, welcher angibt, wie fest sich Strahlen annähern müssen, damit sie als "sich schneidend" gelten. Um weniger false-positives zu generieren kann man nur die Schnittpunkte berücksichtigen, an denen sich Strahlen von drei oder mehr Spottern treffen. Allerdings ist nicht immer jeder Punkt von mindestens drei Spottern sichtbar.

Wir unterscheiden deshalb zwischen zwei Arten von Schnittpunkten: Schnittpunkte, an denen sich Strahlen von drei oder mehr Spottern kreuzen, nennen wir **Threeway-Intersections**. Diese Art Schnittpunkt weist nur einen geringen false-positive Wert auf. Wir verwenden ausschliesslich diese Art Schnittpunkt für das initiale Modelmatching (siehe Kapitel 5.3.3). Sie werden auch als initiale Kandidaten beim Modeltracking verwendet (siehe Kapitel 5.3.4).

Die sogenannten **Twoway-Intersections** sind solche Schnittpunkte, an denen sich Strahlen von nur zwei Spottern kreuzen. Diese werden beim Modeltracking als Kandidaten für Punkte verwendet, bei denen sich kein geeigneter Threeway-Intersection Kandidat finden lässt.

Twoway-Intersections werden generiert, indem die Rays aller Spotter, welche noch keine Threeway-Intersection generiert haben, mit allen Rays der anderen Spotter, welche ebenfalls noch keine Threeway-Intersection generiert haben, auf Schnittpunkte geprüft werden. Der Schwellwert für die Distanz zwischen zwei Strahlen, damit sie als "sich schneidend" gelten, liegt dabei bei 7mm. Da sich, wie eingangs erwähnt, die Strahlen nicht genau schneiden müssen, müssen wir die Punkte auf den Strahlen finden, die einander am nächsten sind. Für die Berechnung eines beliebigen Punktes  $p_t$  auf dem Strahl genügt es, die um den Wert  $t$  skalierte Richtung  $\vec{d}$  zum Ursprung  $o$  hinzuzufügen, wobei  $t \in \mathbb{R} \geq 0$ .

$$p_t = o + t \cdot \vec{d} \quad (5.6)$$

Um die t-Werte der am nächsten zusammenliegenden Punkte  $p_1$  und  $p_2$  zu erhalten, verwendet man die Formeln 5.7 und 5.8 [19].

$$t_1 = \frac{|\mathbf{M}_1|}{\|\vec{d}_1 \times \vec{d}_2\|^2} = \frac{|[\mathbf{o}_1 - \mathbf{o}_2 \quad \vec{d}_2 \quad \vec{d}_1 \times \vec{d}_2]|}{\|\vec{d}_1 \times \vec{d}_2\|^2} \quad (5.7)$$

$$t_2 = \frac{|\mathbf{M}_2|}{\|\vec{d}_1 \times \vec{d}_2\|^2} = \frac{|[\mathbf{o}_1 - \mathbf{o}_2 \quad \vec{d}_1 \quad \vec{d}_1 \times \vec{d}_2]|}{\|\vec{d}_1 \times \vec{d}_2\|^2} \quad (5.8)$$

Falls die Strahlen sich nicht kreuzen, sind die Punkte  $p_1$  und  $p_2$  die Punkte, an denen sich die Strahlen am nächsten sind. Falls die Strahlen parallel sind, gilt  $\vec{d}_1 \times \vec{d}_2 = 0$ . Dies muss vorgängig geprüft werden um eine Division durch null zu vermeiden.

Sobald die Punkte  $p_1$  und  $p_2$  bestimmt sind, kann deren Distanz berechnet werden. Falls diese über dem gegebenen Schwellwert, hier 7mm, liegt, kreuzen sich die Strahlen nicht und es wird kein Schnittpunkt generiert.

Unser Algorithmus für die Berechnung von Threeway-Intersections basiert auf der Feststellung, dass dort, wo eine Threeway-Intersection auftritt, drei Schnittpunkte, generiert durch drei Strahlen von drei verschiedenen Spottern, nahe beieinanderliegen müssen (Abbildung 5.14a). Für die Generierung von Threeway-Intersections werden zunächst die Schnittpunkte der Strahlen eines Spotters mit den Strahlen aller anderen Spotter berechnet (Abbildung 5.14b und 5.14c). Pro Strahl wird dabei eine Liste aller Schnittpunkte mitgeführt, die auf diesem Strahl generiert wurden. Diese Liste ist nach der Distanz vom Ursprung bis zum Schnittpunkt, also nach dem t-Wert, sortiert. Anschliessend werden die Listen aller Strahlen durchgegangen und Schnittpunkte gesucht, welche nicht weiter als eine gewisse Distanz auseinanderliegen. Ein Schwellwert von 1cm ergab gute Resultate. Wenn zwei solche Schnittpunkte gefunden wurden, wird der Schnittpunkt der beiden Strahlen, welche diese Schnittpunkte mit generiert haben, berechnet (Abbildung 5.14d). Die Distanz der ersten beiden Schnittpunkte zu diesem dritten Schnittpunkt muss ebenfalls unter dem Schwellwert liegen. Falls dies der Fall ist, liegt eine Threeway-Intersection vor. Zur Bestimmung der finalen Position wird der Schwerpunkt der drei Schnittpunkte berechnet (Abbildung 5.14e).

Auch bei Threeway-Intersections ist es möglich, dass false-positive Kandidaten auftreten, also Positionen, an denen sich drei Strahlen kreuzen, aber kein Marker vorhanden ist. Diese sind jedoch selten und werden nicht gesondert behandelt, da sich gezeigt hat, dass sie im Modelmatching oder Modeltracking keine Probleme bereiten.

Nach dem Generieren der Threeway-Intersections werden jene Schnittpunkte, welche nahe beieinanderliegen (Distanz kleiner als 1cm), zusammengefasst. Der finale Schnittpunkt hat die Position an dem Schwerpunkt der ursprünglichen Schnittpunkte. Der Fall, dass mehrere Threeway-Intersections sehr nahe beieinanderliegen, tritt auf, falls mehr als drei Spotter einen Punkt sehen.

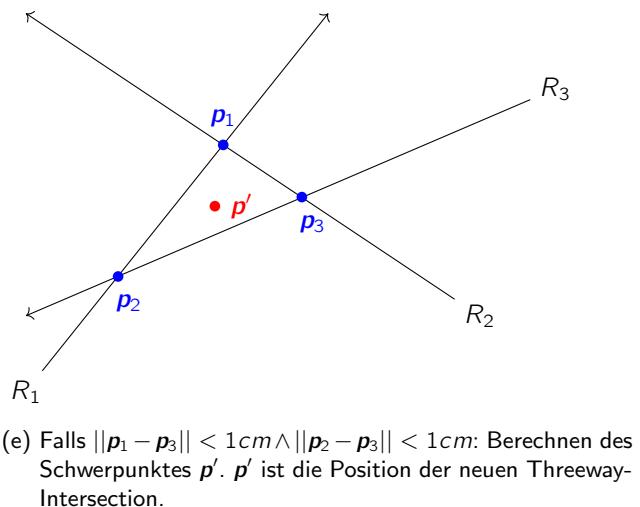
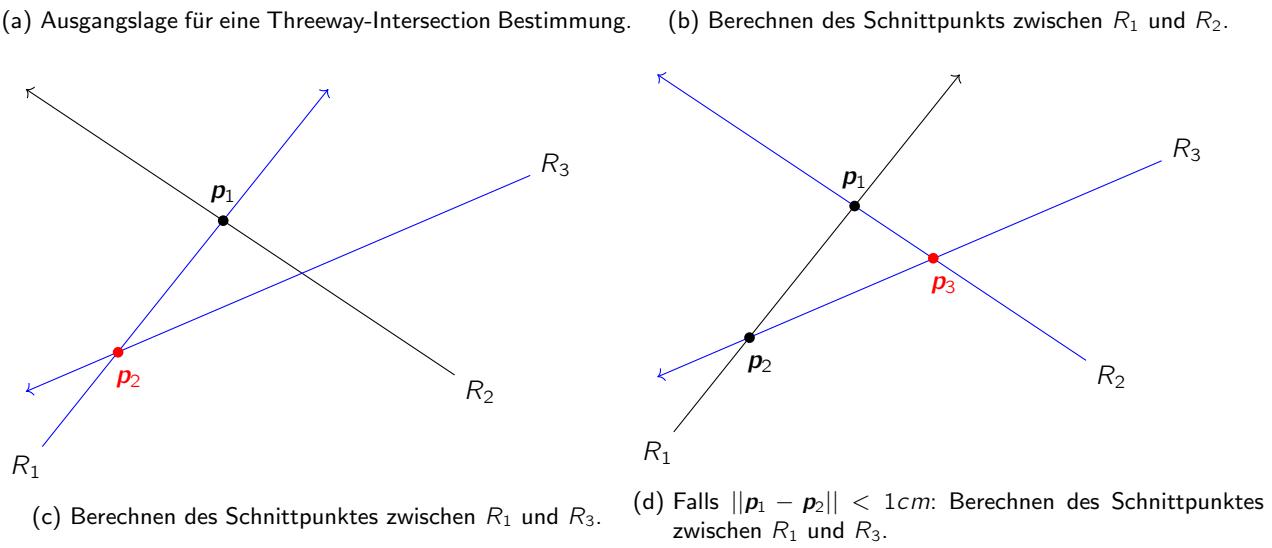
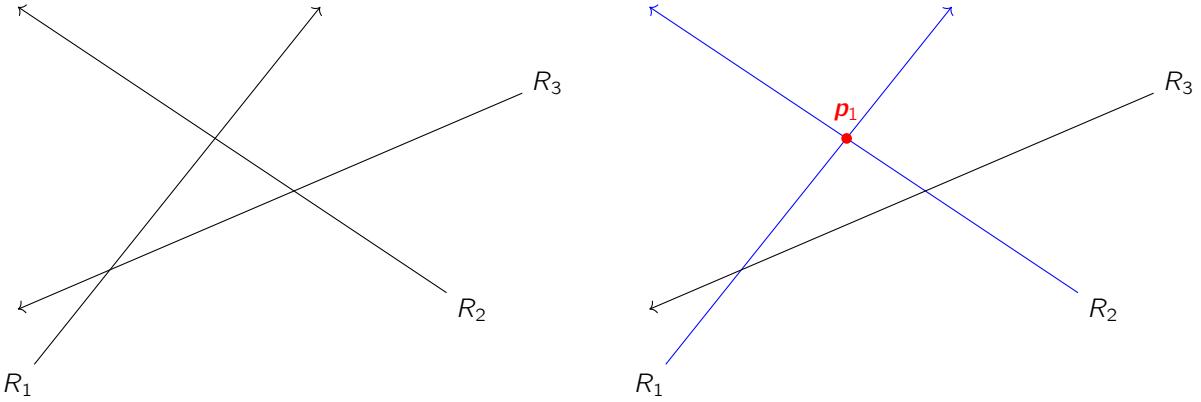


Abbildung 5.14.: Generierung von Threeway-Intersections

### 5.3.3. Modelmatching

Um das virtuelle Modell animieren zu können, müssen die detektierten Marker initial auf die Modellpunkte gepasst werden. Dazu muss der Schauspieler eine eindeutig identifizierbare Pose einnehmen. Die sogenannte T-Pose, welche in der Abbildung 4.4 zu sehen ist, ist dazu besonders geeignet, da alle Marker gut sichtbar sind.

Die Zuweisung der Modellpunkte geschieht mit dem Modelmatching Algorithmus. Um eine möglichst hohe Stabilität zu gewährleisten, und nicht fälschlicherweise false-positive Schnittpunkte zu verwenden, brauchen wir zu diesem Zeitpunkt nur die Threeway-Intersections (siehe Kapitel 5.3.2). Der Algorithmus funktioniert folgendermassen:

Zunächst werden die Fussmarker gefunden. Dazu werden die Schnittpunkte zuerst nach ihrer Höhe sortiert. Danach werden zwei Schnittpunkte gesucht, welche maximal einen Höhenunterschied von 5cm aufweisen und zwischen 25cm und 50cm weit auseinanderliegen<sup>1</sup>. Die Schnittpunkte, welche diese Bedingungen erfüllen und am tiefsten beim Boden liegen, werden als Fussmarker interpretiert. Was wir zu diesem Zeitpunkt noch nicht feststellen können, ist, welches der linke und welches der rechte Fuss ist. Diese Zuordnung geschieht in einem späteren Schritt.

Sobald die Fussmarker zugewiesen sind, wird ein Zylinder aufgespannt, in dem sich die Hüft-, Brust-, Kopf- und Schultermarker befinden müssen (Abbildung 5.15). Das Zentrum des Zylinders befindet sich auf dem Boden zwischen den beiden Fussmarkern und hat einen Radius von 25cm.

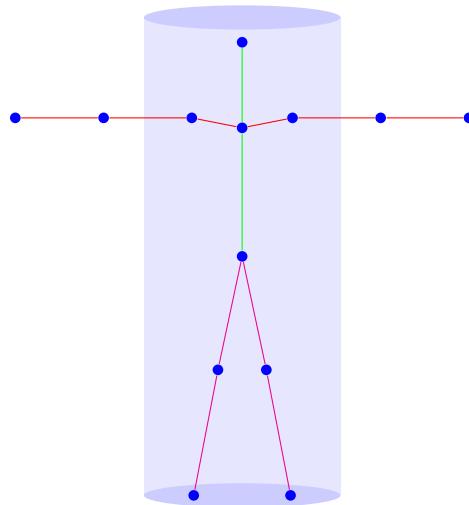


Abbildung 5.15.: Der Zylinder beinhaltet alle Körpermarker, sowie die Schulter- und Knie- und Fussmarker

Dem Kopfmarker wird jenem Schnittpunkt zugewiesen, welcher in diesem Zylinder am höchsten liegt.

Als nächstes werden die Knie zugewiesen. Dazu werden im Zylinder Kandidaten gesucht, welche höher als die Fussmarker und maximal 60cm hoch sind. Die Kandidaten dürfen eine maximale Höhendifferenz von 5cm haben. Ausserdem wird geprüft, dass der Winkel zwischen dem Vektor, welcher die beiden Kniekandidaten verbindet, und dem Vektor, welcher die beiden Fussmarker verbindet, nicht mehr als 30 Grad beträgt. Die tiefsten Kandidaten, welche all diese Bedingungen erfüllen, werden den Kniemarkern zugewiesen.

Kandidaten für die Schulter-, Ellbogen- und Handmarker werden gemeinsam gesucht. Zunächst werden im Zylinder Kandidaten für die Schultern gesucht, welche höher liegen als die Knie, maximal 5cm Höhenunterschied haben und deren Verbindungsvektor maximal 30 Grad von der Fussachse abweicht. Wenn solche Schulterkandidaten gefunden wurden, werden Kandidaten für die Ellbogenmarker gesucht. Ein Ellbogenkandidat darf maximal 10cm Höhenunterschied zu seinem Schulterkandidat aufweisen. Ausserdem darf der Verbindungsvektor zwischen dem Ellbogenkandidat und dem Schulterkandidat um nicht mehr als 30 Grad von der Fussachse abweichen. Dasselbe wird für das Eruieren von Handkandidaten wiederholt. Nur wenn alle Armmarker einen Kandidaten haben, werden sie entsprechend zugewiesen.

<sup>1</sup>Diese Werte basieren auf der DIN Norm 33402 über Körpergrössen des Menschen und wurden so gewählt, dass >95 Perzentile aller erwachsenen Menschen abgedeckt sind.

Dann werden noch Kandidaten für die Brust- und Hüftmarker gesucht. Dabei wird jeweils geprüft, dass die Kandidaten in der entsprechenden Höhe (unter den Schultern und über den Knien für die Brust und unter der Brust und über den Knien für die Hüfte) liegen.

Schlussendlich muss noch die korrekte Zuweisung an die linke respektive rechte Modellseite erfolgen. Wir verwenden hierzu die Flächenbilanz. Es wird ausgenutzt, dass der Brustumker leicht vor den Schultern liegt (Abbildung 5.16).

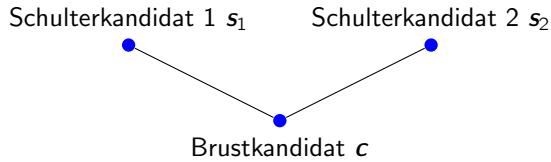


Abbildung 5.16.: Ausgangslage für die Zuweisung an die linke und rechte Modellseite (von oben betrachtet)

Zuerst wird ein Punkt bestimmt, welcher vor der Brust in der Blickrichtung des Modells liegt. Dazu wird der Vektor, welcher vom Schulterkandidaten 1 zur Brust zeigt, zum Vektor, welcher vom Schulterkandidaten 2 zur Brust zeigt, addiert und der resultierende Vektor normalisiert. Dieser Vektor wird dem Brustkandidaten addiert um den vorliegenden Punkt zu bestimmen (Abbildung 5.17).

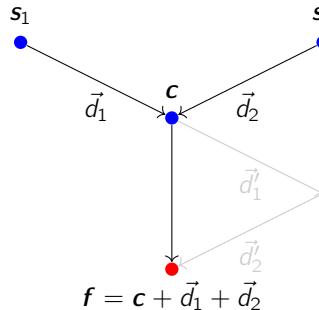


Abbildung 5.17.: Bestimmung eines vorliegenden Punktes

Anschliessend wird ein Dreieck zwischen dem Brustkandidaten, dem vorgelegten Punkt und dem zuzuordnenden Kandidaten aufgespannt.

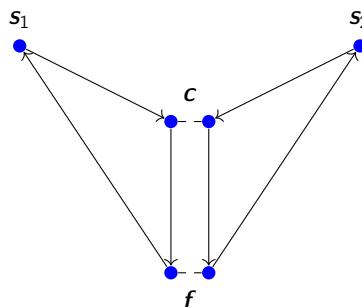


Abbildung 5.18.: Dreiecke zwischen Brust  $c$ , vorliegendem Punkt  $f$  und Kandidaten  $s_1$  und  $s_2$ . Man bemerke die entgegengesetzten Laufrichtungen der beiden Dreiecke.

Daraufhin werden zwei Flächen mit den Formeln 5.9 und 5.10 berechnet. Um die Flächenbilanz zu bestimmen wird die Differenz der beiden Flächen bestimmt und das Resultat halbiert (Formel 5.11).

$$a_1 = \vec{f} \cdot \vec{s}_1.x \cdot \vec{c} \cdot \vec{s}_1.y \quad (5.9)$$

$$a_2 = c\vec{s}_1.x \cdot f\vec{s}_1.y \quad (5.10)$$

$$a_s = \frac{a_2 - a_1}{2} \quad (5.11)$$

Die Flächenbilanz ist positiv, falls die Laufrichtung im Uhrzeigersinn (CW) geht, und negativ, falls sie im Gegen-uhrzeigersinn (CCW) geht (Formel 5.12) [20]. Anhand der Laufrichtung lässt sich dann sagen, ob ein Punkt zur linken oder rechten Modellhälfte gehört. Derselbe Algorithmus wird in OpenGL fürs Face-Culling verwendet [21].

$$\forall(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in \mathbb{R}^2 \exists a_s \Rightarrow a_s = \begin{cases} > 0, & \text{falls } \mathbf{a}, \mathbf{b}, \mathbf{c} \text{ in CW Laufrichtung} \\ 0, & \text{falls } \mathbf{a}, \mathbf{b}, \mathbf{c} \text{ auf Gerade} \\ < 0, & \text{falls } \mathbf{a}, \mathbf{b}, \mathbf{c} \text{ in CCW Laufrichtung} \end{cases} \quad (5.12)$$

Es folgt also, dass der Kandidat zur linken Modellseite gehört, falls die Flächenbilanz positiv ist und zur rechten, falls sie negativ ist. Mit dieser Methode werden die Fuss-, Knie-, Schulter-, Ellbogen- und Handkandidaten der korrekten Modellseite zugewiesen.

### 5.3.4. Modeltracking

Nach der initialen Zuweisung der Marker auf die Modellpunkte geht die Applikation in den Tracking Modus über. In diesem Modus wird versucht, den Modellpunkten Markerkandidaten aufgrund der vorangegangenen Position zuzuweisen. Als Basis dazu dient der Nearest-Neighbor Algorithmus [22].

Als Kandidaten werden zuerst Threeway-Intersections verwendet. Damit nicht alle Threeway-Intersections geprüft werden müssen, werden gewisse Restriktionen angewandt:

- Die Distanzen zwischen den Ellbogen und der jeweiligen Hand, den Schultern und dem jeweiligen Ellbogen sowie zwischen den Knien und dem jeweiligen Fuss werden als unveränderbar angeschaut. Nur Kandidaten, welche innerhalb von 5cm der entsprechenden Distanz liegen, werden überhaupt berücksichtigt.
- Es wird davon ausgegangen, dass sich ein Marker in jedem Frame nicht mehr als eine gewisse Distanz bewegen kann. Dabei wird berücksichtigt, dass sich der Marker auf den Extremitäten (z. Bsp. auf den Händen) pro Frame weiter bewegen kann als Marker am Körper (z. Bsp. auf der Brust). Eine maximale Distanz von 10cm pro Frame hat gute Resultate ergeben.

Falls sich keine geeignete Threeway-Intersection finden lässt, wird in den Twoway-Intersections nach einem Kandidaten gesucht. Falls auch dort kein geeigneter Kandidat vorhanden ist, bleibt die Position des entsprechenden Modellpunktes für das Frame unverändert. Falls für einen Modellpunkt in mehreren aufeinanderfolgenden Frames kein Kandidat gefunden wird, schlägt das Tracking fehl und es muss erneut ein Modelmatching stattfinden. Ein fehlgeschlagenes Tracking resultiert in der falschen Darstellung des Modells.

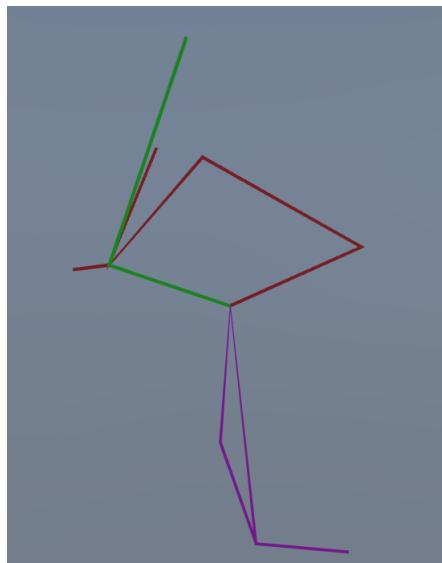


Abbildung 5.19.: Ein Modell bei fehlgeschlagenem Tracking

### 5.3.5. Übermittlung der Markerpositionen

Der Beholder übermittelt eine Liste von Markerpositionen zur Visualisierung an den Mimic. Dies geschieht über das in Kapitel 4.2 definierte Netzwerkprotokoll. Für unser Test-Rig werden dreizehn (Anzahl Punkte im Test-Rig) mal drei (Anzahl Dimensionen, dreidimensionaler Vektor) Fließkommazahlen (floats) übermittelt. Ein entsprechendes Netzwerkframe wird in Abbildung 5.20 dargestellt.

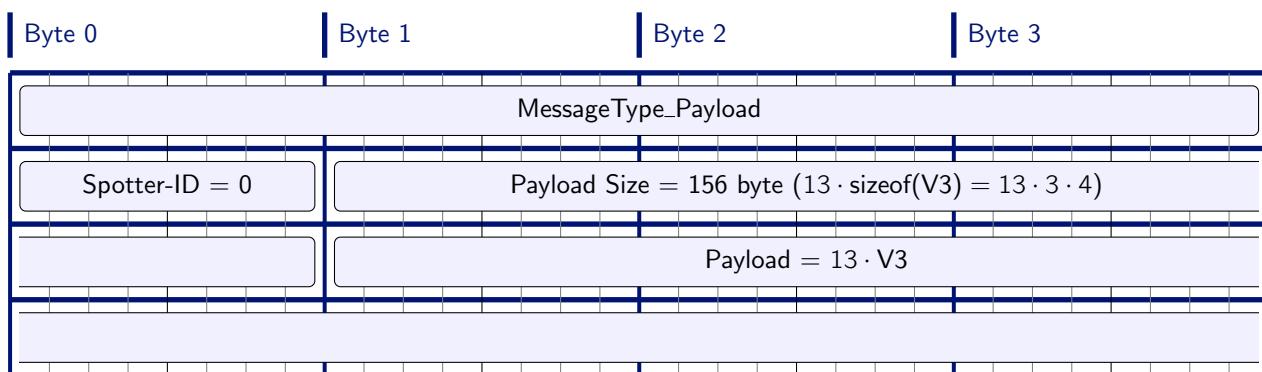


Abbildung 5.20.: Aufbau einer Beholder Payload Message

## 5.4. Mimic

Der Mimic erhält vom Beholder alle detektierten Marker als Intersections. Nach dem Modelmatching liefert der Beholder auch die Information, welcher Marker welchem Modellpunkt entspricht. Vom Mimic aus können, sobald das System läuft, Befehle an Beholder und Spotter gesendet werden. Dies dient einerseits dem Steuern des Systems, andererseits sind einige Hilfsfunktionen im Mimic eingebaut.

### 5.4.1. Interface

Alle Funktionen des Systems, bis auf die Startup-Prozedur und die intrinsische Kalibrierung der Kameras, können vom Interface aus gesteuert werden. Als Steuerfunktionen kann die Pose Estimation ausgelöst oder der Schwellwert

für die Binarisierung der Bilder auf dem Spotter angepasst werden. Ebenfalls kann das System heruntergefahren werden. Die weiteren Steuerungselemente beinhalten Hilfsfunktionen. Die Kamerabilder können als stark komprimierter JPG-Bildstream empfangen werden. Dies dient am Anfang zur Überprüfung des Sichtfelds jeder Kamera. Es können wahlweise Graustufenbilder oder bereits binarisierte Bilder empfangen werden. Die binarisierten Bilder dienen vor allem der Fehlersuche. Da diese Funktion Bandbreite wie auch nicht wenig Rechenleistung auf Spotter und Mimic erfordert, kann diese Funktion nach der Ausrichtung der Kameras abgeschaltet werden. Auf dem Spotter wird so erhebliche Rechenleistung eingespart, da dann pro Frame nicht zusätzlich noch ein JPG erstellt werden muss.

Um Bandbreite zu sparen kann das versenden der Debug-Informationen auch komplett ausgeschaltet werden. Dann werden nur noch die erkannten Markerpositionen sowie das erkannte Rig an den Mimic gesendet. Wahlweise können bei eingeschalteten Debuginformationen auch Rays und/oder Intersections ausgeblendet werden.

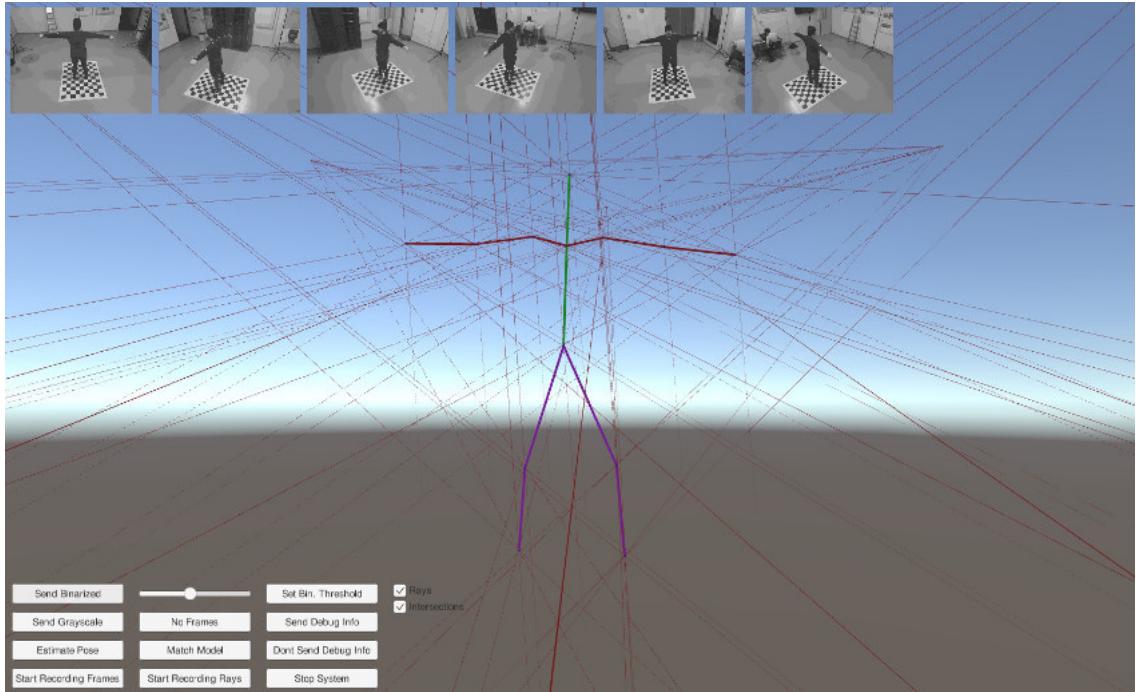


Abbildung 5.21.: Screenshot des GUI

### 5.4.2. Aufzeichnen von Strahlen und Kameraframes

Einzelne Kameraframes oder erkannte Strahlen können aufgezeichnet werden. Mit den aufgezeichneten Kameraframes kann die Spotter-Applikation auf einem beliebigen Gerät gestartet werden und die Frames zur Verarbeitung kommen dann nicht von der Kamera, sondern aus dem aufgezeichneten File. Die Möglichkeit Rays aufzuzeichnen verhilft uns dazu, nicht für jeden Test das gesamte System aufzustellen und kalibrieren zu müssen. So können neue oder verbesserte Funktionen auf dem Beholder und im Mimic einfach getestet werden.

### 5.4.3. Simple Rig

Für die erste Visualisierung haben wir ein einfaches Rig erstellt (Abbildung 4.4). Es besteht aus 13 Punkten welche eins zu eins unseren Markern am Körper entsprechen. Dieses Rig dient vor allem dazu, dass Modelmatching zu testen und eine erste Visualisierung des Trackings zu erhalten.

Das Rig besteht aus einer hierarchischen Struktur von Kugeln. Für die Visualisierung der Extremitäten werden LineRenderer Objekt von Unity verwendet. Es werden Linien von der linken zur rechten Hand, vom Kopf bis zur Hüfte und vom linken Fuss über die Hüfte zum rechten Fuss gezeichnet. Diese Linien schliessen jeweils die Zwischenknoten mit ein und werden in verschiedenen Farben gerendert. So ergibt sich eine nachvollziehbare humanoide Figur. Der



Abbildung 5.22.: Hierarchische Struktur des Rig

Mimic erhält vom Beholder jedes Frame ein Array mit allen 13 Punkten des Rigs. Diese 13 Punkte werden vom Beholder berechnet. Ist einer der 13 Punkte nicht fehlerfrei berechenbar, wird der letzte bekannte Wert übermittelt. Die Koordinaten aus dem übermittelten Array werden nun den Transformkomponenten vom Rig zugewiesen.

#### 5.4.4. Complex Rig

Nach der Visualisierung mit dem einfachen Rig in Abbildung 4.4 wurde ein Versuch unternommen, ein echtes Computermodell mit einem Skelett anhand der verfolgten Bewegung zu animieren. Dazu wurde auf das Modell von Sintel [23] ein Skelett gelegt, welches unseren Markerpositionen entspricht. Dies ist nicht ideal, weil so die Rotationspunkte der einzelnen Knochen nicht bei den eigentlichen Gelenken sind. Allerdings ist dies mit den wenigen Markern, welche wir verwenden, nicht besser möglich.

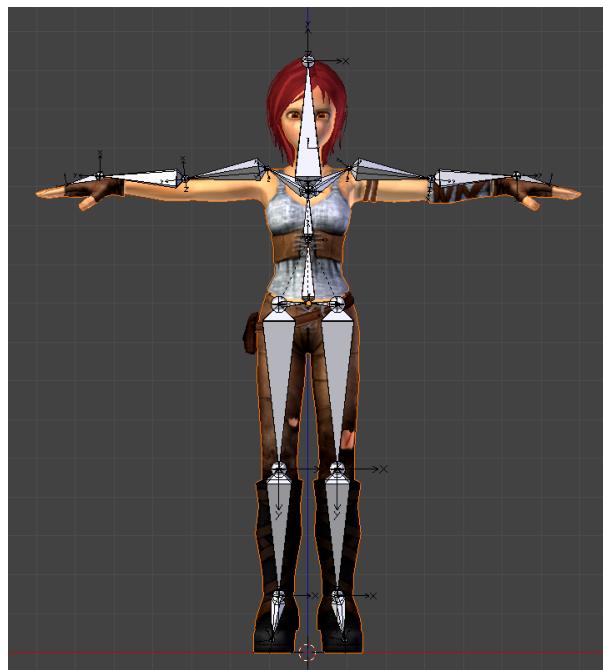


Abbildung 5.23.: Sintel mit dem den Markern entsprechenden Skelett

Das Skelett besteht aus einzelnen Knochen, welche hierarchisch miteinander verbunden sind. Jeder Knochen hat ein eigenes Koordinatensystem mit dem die Translation und Rotation zum Elternknochen ausgedrückt wird. Damit die Rotationen von den aufgenommenen Markerpositionen auf das Sintelmodell übertragen werden können, müssen die Koordinatensysteme anhand der Markerpositionen berechnet und analog zum Sintelmodell ausgerichtet werden.

Dazu wurde ein Root-Knochen definiert, welcher zwischen den Schultern liegt und Richtung Brust zeigt. Mit der momentanen Markerkonfiguration ist das Abgleichen der Koordinatensysteme nur annäherungsweise lösbar, da gewisse Marker fehlen um eine genaue Übertragung vorzunehmen.

Nachdem die Koordinatensysteme aufeinander abgeglichen wurden, können die Rotationen, vom Root-Knochen ausgehend rekursiv, auf die Kinderknochen angewandt werden. Da jeder Knochen die Rotation relativ zu seinem Elternknochen angibt, können die Rotationen dann direkt auf das Sintelmodell übertragen werden.

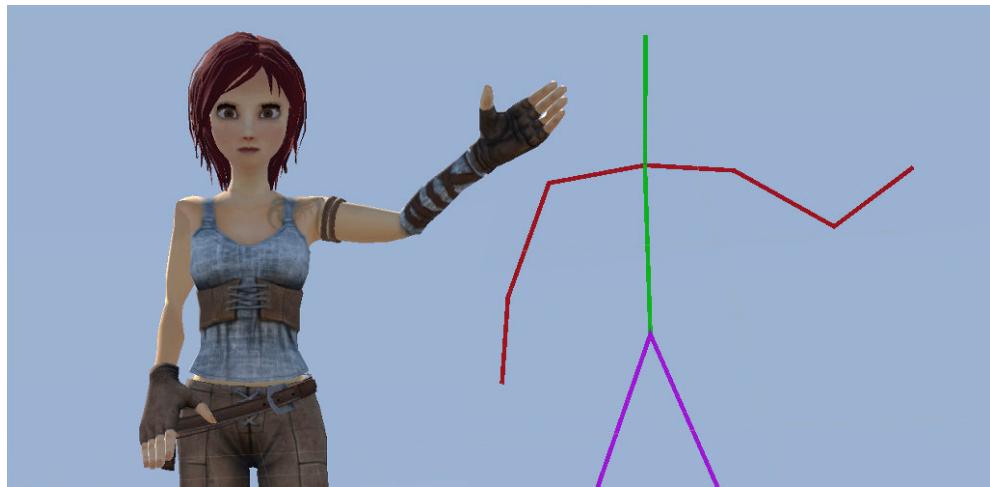


Abbildung 5.24.: Eine winkende Sintel neben einem winkenden Strichmann



# 6. Systemanalyse

Wir wollen in diesem Kapitel unser System mit einem professionellen System vergleichen. Dies vor allem um die Systematik und die Kosten zu vergleichen. Um die Zielerreichung zu verifizieren haben wir zudem die Genauigkeit unseres Systems gemessen.

## 6.1. Laufzeitanalyse

Durch die Wahl der Bildauflösung von 1640x1232 Pixel sind wir auf den Kameras auf 40 FPS beschränkt. Nichtsdestotrotz wollten wir die Laufzeit unserer Algorithmen, sprich die Markerextraktion auf den Spottern sowie die Schnittpunktberechnung auf dem Beholder, testen.

Um die Laufzeiten zu messen wurde das System mit sechs Kameras gestartet und ins Zentrum eine Anzahl Marker gelegt, so dass sie von allen Kameras sichtbar waren. Es wurden drei Messungen mit 10, 20 und 30 Markern durchgeführt.

### 6.1.1. Markerextraktion

Es stellte sich heraus, dass die Anzahl Marker keinen Einfluss auf die Laufzeit der Markerextraktion hat. Obwohl der Flaschenhals nach wie vor die FPS der Kamera sind, ist die Markerextraktion nah an der maximalen Dauer, welche gewartet werden muss. Bei einer Verbesserung des Systems wäre demnach eine Beschleunigung der Markerextraktion ein wichtiger Schritt. Die Resultate sind in Abbildung 6.1 als Boxplot dargestellt. Der untere "Whisker" ist der minimale gemessene Wert, der obere der maximale. Das untere und obere Quartil sind jeweils der tiefste, respektive höchste, Wert im unteren und oberen Viertel der gemessenen Werte. Die Zentrumsmarkierung entspricht dem Median. Die Werte sind im Anhang D zu finden.

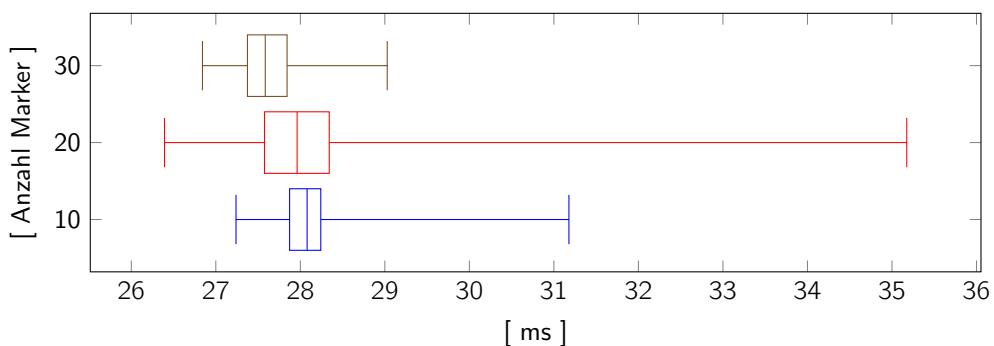


Abbildung 6.1.: Laufzeit der Markerextraktion in Millisekunden bei 10, 20 und 30 Markern.

### 6.1.2. Schnittpunktberechnung

Die erwartete Laufzeit für die Schnittpunktberechnung mit unserem naiven Algorithmus ist  $O(n^2)$  wobei  $n$  die maximale Anzahl Strahlen pro Kamera sind. Diese Vermutung bestätigte sich durch unsere Messungen: Die Laufzeit nahm mit der Anzahl Markern mehr als Linear zu.

Die Resultate sind in Abbildung 6.2 als Boxplot dargestellt. Die Werte sind im Anhang D zu finden.

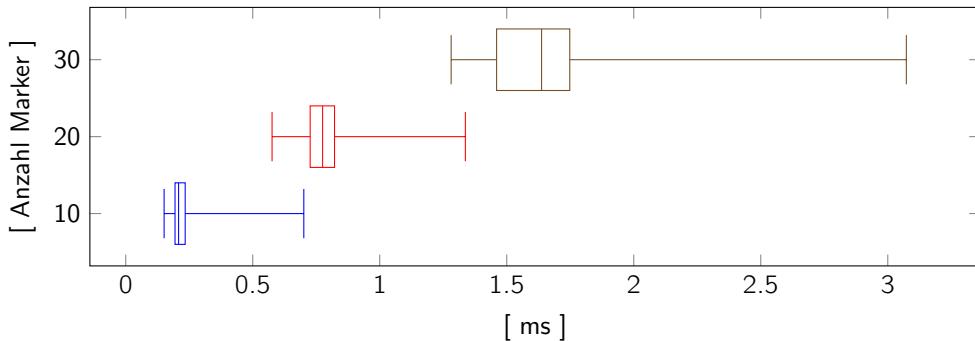


Abbildung 6.2.: Laufzeit der Schnittpunktberechnung in Millisekunden bei 10, 20 und 30 Markern.

## 6.2. Präzisionsanalyse

Um die Ungenauigkeit bei der Detektion zu bestimmen, haben wir uns dazu entschieden einen einzelnen Marker auf bekannten Referenzpunkten zu platzieren und die Position zu bestimmen. Die detektierte Position jedes Markers kann im Unity im Pausenmodus aus den Positionseigenschaften ausgelesen werden. Dieses Feature haben wir uns zunutze gemacht um die Genauigkeitsmessung durchzuführen. Da die Messungen sehr aufwändig sind, können wir nur eine beschränkte Anzahl Messungen durchführen.

### 6.2.1. Versuchsaufbau

Als Referenzpunkte bedienen wir uns der Schnittpunkte auf dem Schachbrett, welches wir als Kalibriertarget verwenden. Von diesen Punkten wissen wir die absoluten Weltkoordinaten. Wir messen jeweils die vier äussersten Eckpunkte auf jeweils drei verschiedenen Höhen. Die erste Messhöhe erfolgt auf Bodenhöhe, was faktisch der Höhe von einem Zentimeter entspricht. Dies röhrt daher, dass die Marker einen Durchmesser von zwei Zentimetern haben. Der Mittelpunkt der detektierten Marker sollte folglich auf der Höhe von einem Zentimeter zu liegen kommen. Die weiteren Höhen sind 105cm und 170cm. Ursprünglich sollten die Messungen auf 1.05m und 2.10m stattfinden. Die Messungen auf 2.10m sind jedoch nicht möglich, da der Kegel des IR Lichts nicht an diese Messpunkte hin reicht. So wurde 1.70m als höchstmöglicher Messpunkt an den Ecken des Schachbretts eruiert. Die mittlere Messhöhe wurde bei 1.05m belassen.

Die Höhe des Markers wurde bei jeder Messung mit einem handelsüblichen Massstab verifiziert. Die genaue Positionierung auf der X und Y Achse bei den beiden oberen Messhöhen wurde mit einem Maurerlot verifiziert (Abbildung 6.3). So kann sichergestellt werden, dass der Marker bei jedem Messpunkt möglichst exakt auf der Sollkoordinate liegt.

Es wurden zwei Messdurchgänge durchgeführt, was in insgesamt 24 Messpunkten resultiert. Dies reicht bei weitem nicht um eine genügend grosse Stichprobe zu erhalten. Es gibt aber eine Idee, wie genau unser System ist. Ebenfalls muss ein relativ hoher Messfehler angenommen werden, da wir die Höhe mit einem normalen Massstab gemessen haben. Ebenfalls wurde der Mittelpunkt der Marker nur optisch bestimmt. Aus diesem Grund gehen wir von einem Messfehler von 2mm aus. Auf den höheren Messpunkten ist davon in der Horizontalen von einem Fehler von 4mm auszugehen (Markerpositionierung und Platzierung des Lots).

### 6.2.2. Ergebnisse

Die höchste von der Sollposition gemessene Abweichung beträgt 10mm. Im Durchschnitt beträgt die Abweichung auf der X und Y Achse 2mm, beziehungsweise 3mm. In der Höhe beträgt die Abweichung 6mm. Auf Bodenhöhe beträgt die höchste Abweichung 3mm, im Durchschnitt auf jeder Achse nur maximal 1mm. Auf den oberen Messhöhen betragen die Abweichungen auf der X und Y Achse im Schnitt bis zu 5mm, in der Höhe beträgt die durchschnittliche Abweichung maximal 9mm.



Abbildung 6.3.: Messaufbau mit Maurerlot und retroreflektivem Marker

Aus Tabelle 6.1 sind die durchschnittlichen Abweichungen auf den drei Messhöhen über alle auf dieser Höhe gemessenen Punkte ersichtlich.

| Messhöhe [mm] | X [mm] | Y [mm] | Z [mm] |
|---------------|--------|--------|--------|
| 10            | 1      | 1      | 1      |
| 105           | 1      | 5      | 8      |
| 170           | 4      | 3      | 9      |

Tabelle 6.1.: Abweichungen Markerdetektion

Allgemein ist auf Bodenhöhe praktisch keine Abweichung erkennbar. Auf den oberen Messhöhen steigen die Abweichungen aber markant an. Das Messprotokoll ist im Anhang C zu finden.

### 6.2.3. Beurteilung

Grundsätzlich sind die Resultate der Genauigkeitsmessung sehr erfreulich. Sie sind aber wegen der kleinen Stichprobe mit Vorsicht zu genießen.

Die grössere Ungenauigkeit bei den oberen Messhöhen ist wahrscheinlich der Tatsache geschuldet, dass wir lediglich auf Bodenhöhe kalibriert haben. Ein dreidimensionales Kalibrierungstarget oder mehrere Durchgänge der Pose Estimation mit dem Target auf verschiedenen Höhen würde dieses Problem vermutlich reduzieren. Die selbe Ursache ist höchstwahrscheinlich für die höheren Abweichungen auf der Z Achse verantwortlich. Eine weitere Erklärung für die steigende Ungenauigkeit in grösseren Höhen ist der immer flachere Winkel, in dem Strahlen aufeinandertreffen.

Der Unterschied in den durchschnittlichen Ungenauigkeiten zwischen der X und der Y Achse ist nicht systembedingt zu erklären. Einerseits kann die Umgebung, in welcher die Messungen stattgefunden haben, eine Rolle spielen. Andererseits kann die unterschiedliche Kalibrierungsqualität der Kameras eine Rolle spielen. Nicht zuletzt sind Messungenauigkeiten in diesem Bereich durchaus möglich.

### 6.3. Kostenvergleich mit Referenzsystem

Um unser System mit einem System aus der Industrie vergleichen zu können, haben wir uns online ein Vergleichssystem von OptiTrack zusammengestellt. Es handelt sich um ein preiswertes Komplettsystem von OptiTrack<sup>1</sup>, welches jedoch auf USB und nicht auf Ethernet basiert. Das Referenzsystem wurde so vom OptiTrack Konfigurator vorgeschlagen und beinhaltet neben den Kameras auch Kabel, Software und sonstige Komponenten (Synchronisation, usw.), welche benötigt werden, damit das System funktionsfähig ist. Da unsere Software eine Einzelanfertigung ist, werden wir die Softwarekosten nicht in die Vergleichsrechnung mit einbringen. Ebenfalls nicht enthalten sind die Stativen.

| Produkt                   | Funktion | Partnumber          | Einzelpreis | Anzahl | Gesamtpreis        |
|---------------------------|----------|---------------------|-------------|--------|--------------------|
| Flex 3                    | Kamera   | #FL3.RD.LENS0003.LP | CHF 587.00  | 6      | CHF 3522.10        |
| OptiHub 2                 | Hub      | #OPTHUB             | CHF 293.00  | 2      | CHF 586.05         |
| CS-200 Calibration Square | Tool     | #CS-200             | CHF 146.00  | 1      | CHF 146.00         |
| CW-500 Calib. Wand Kit    | Tool     | #CW-500             | CHF 293.00  | 1      | CHF 293.00         |
| Sync Cable                | Kabel    | #CBL0013            | CHF 9.80    | 1      | CHF 9.80           |
| USB Active Ext. Cable     | Kabel    | #CBL1015            | CHF 19.60   | 2      | CHF 39.20          |
| USB Cable, Down Angle     | Kabel    | #CBL0018            | CHF 9.80    | 6      | CHF 58.80          |
| USB Uplink Cable: 16ft    | Kabel    | #CBL1021            | CHF 4.90    | 2      | CHF 9.80           |
| <b>Gesamtpreis</b>        |          |                     |             |        | <b>CHF 4664.80</b> |

Tabelle 6.2.: Referenzsystem von Optitrack

In der folgenden Tabelle sind die Kosten für unser System ersichtlich<sup>2</sup>. Wir haben nur Komponenten berücksichtigt, welche im Referenzsystem ein Pendant haben, Stativen, Rechner, usw. wurden nicht in die Berechnung mit einbezogen. Ebenfalls ist unser Kalibrierungstarget (ausgedrucktes Schachbrett auf Doppel A0) trotz Pendant nicht aufgeführt, da die Kosten schwer zu beziffern sind.

| Produkt            | Funktion | Einzelpreis | Anzahl | Gesamtpreis      |
|--------------------|----------|-------------|--------|------------------|
| Spotter            | Kamera   | CHF 134.70  | 6      | CHF 808.20       |
| Zyxel GS1900-10HP  | Switch   | CHF 137.00  | 1      | CHF 137.00       |
| Ethernet Cable     | Kabel    | CHF 24.40   | 7      | CHF 170.80       |
| <b>Gesamtpreis</b> |          |             |        | <b>\$1134.60</b> |

Tabelle 6.3.: Unser System

Der Spotter besteht aus mehreren einzeln beschafften Komponenten.

- Raspberry Pi 3 Model B+ (CHF 39.00)
- NoIR Camera Modul V2 (CHF 32.90)
- Bright Pi (CHF 24.90)
- Pi Camera Box (CHF 37.90)

Daraus ergibt sich ein Gesamtpreis von CHF 134.70.

<sup>1</sup>Wechselkurs CHF-USD 1:0.98, Stand 28.12.2018 18:07 UTC

<sup>2</sup>Alle Preise für Kamerakomponenten von pi-shop.ch, alle Preise für sonstige Komponenten von digitec.ch (Stand 28.12.2018)

## 6.4. Zielerfüllung

Im Folgenden wird die Zielerfüllung von jedem im Kapitel 2 gestellten Ziele überprüft.

### Bereich

Das System sollte einen Bereich von 2 Metern Durchmesser und 2 Metern Höhe (Zylinderform) tracken können. Dieses Ziel konnten wir nicht erreichen. Wir erreichen lediglich eine Höhe von etwas über 1.70m, wobei der Trackingbereich die Form eines auf dem Boden stehenden Kegels hat.

### Framerate

Das Ziel, dass das System 60 FPS erreichen soll, konnte auch nicht erreicht werden. Die Kamera ist in der Auflösung 1640x1232 lediglich in der Lage 40 Bilder pro Sekunde zu liefern. Rein die Bildverarbeitung wäre aber schneller machbar.

Der Beholder braucht bei unserer Markerconfiguration für die Verarbeitung nicht einmal eine Millisekunde und der Mimic erreicht auch ohne Probleme 60 FPS. Bei diesen beiden Komponenten wäre das Ziel daher erreicht. Der Beholder erhält im Optimalfall 13 Strahlen pro Kamera, bei sechs Kameras also 78 Strahlen insgesamt. Unser Intersection-Algorithmus hat eine quadratische Laufzeit. Wenn wir mit mehr Markern arbeiten würden, müssten wir diesen Algorithmus sicher optimieren. Bei unserer Anzahl Markern ist dies jedoch bis jetzt nicht nötig gewesen.

### Fehler bei der Bestimmung der Markerposition

Der Fehler sollte weniger als zwei Zentimeter betragen. Dieses Ziel wurde erreicht, die grösste gemessene Ungenauigkeit beträgt genau einen Zentimeter.

### Kosten

Die Kosten für die Kamera sollten weniger als 200 Franken betragen. Dieses Ziel wurde erreicht, unsere Kameras kosten CHF 134.70 pro Stück.

## 6.5. Aktuelle Probleme und Lösungsansätze

Die grössten noch bestehenden Probleme sind die Beschränkungen, welche die verwendete Kamerahardware mit sich bringt. Hauptproblem dabei ist der schmale Abstrahlwinkel der IR LEDs. Der Abstrahlwinkel beträgt lediglich 28 Grad. Dies resultiert in einem sehr kleinen trackbaren Bereich. Die Kamera hat momentan einen viel grösseren FOV, als wir mit den LEDs überhaupt ausleuchten können. Durch die Verwendung besserer LEDs mit einem grösseren Abstrahlwinkel und einer höheren Intensität wäre dieses Problem vermutlich gelöst.

Eine weitere Einschränkung basiert auf der Wahl der Bildauflösung auf den Spottern. Die Wahl der hohen Auflösung zieht eine maximale Framerate von 40 FPS nach sich, hat aber einen enorm positiven Einfluss auf die Genauigkeit des Trackings. Ausserdem erhalten wir nur so die volle FOV der Kamera (siehe Tabelle 5.1). Deshalb haben wir diese Auflösung gewählt, obwohl wir damit die geforderten 60 FPS nicht erreichen. Durch den Einsatz einer professionellen Kamera oder eines eigens entwickelten FPGA, welches die Markerdetektion direkt in der Hardware löst, könnte auch dieses Problem gelöst werden.

Unser System funktioniert momentan zudem ausschliesslich bei nahezu kompletter Dunkelheit. Nur nicht sehr intensives, nicht direktes Licht hat keinen Einfluss auf die Funktionsfähigkeit. Ein Hochpassfilter, oder IR-Pass-Filter, würde es uns erlauben unser System in einer künstlich beleuchteten Umgebung zu betreiben. Von gängigen Herstellern für Zubehör gibt es allerdings keinen entsprechenden Filter, und mit vertretbarem Aufwand wäre ein solcher nicht auf die Kamera einzupassen gewesen.

Im Endeffekt wäre ein höherer Aufwand für die Hardware nötig, um diese konkurrenzfähig zu machen. Als Prototyp hat die gebaute Kamera aber die Erwartungen erfüllt.

## 6.6. Verbesserungspotential

Obwohl das System lauffähig ist, gibt es noch verschiedene Punkte, welche verbessert werden können. Einige der Probleme sind bereits in Kapitel 6.5 erläutert und mögliche Lösungsansätze vorgestellt worden. Es gibt allerdings auch bei den erreichten Zielen noch Möglichkeiten zur Verbesserung.

Um die Präzision bei der Positionierung der Marker zu erhöhen wäre insbesondere eine Verbesserung des Kalibriervorgangs geeignet. Ein 3D Kalibriertarget würde vermutlich die Präzision in der Z-Achse erhöhen. Auch könnten mehrere Kalibrierungen durchgeführt werden, wobei die Kameraposition dann immer relativ zu einer "Master"-Kamera ausgedrückt wird. Somit müssten nicht immer alle Kameras gleichzeitig auf das Kalibriertarget sehen. Nach unseren Recherchen verwenden professionelle Systeme einen solchen Ansatz mit einem dynamischen Target und mehreren Aufnahmen zur Kalibrierung.

Die entwickelte Software könnte weiter verbessert werden. Insbesondere im Tracking-Modus werden noch ab und zu die Modellpunkte falsch zugeordnet, was zu falschen (wenn auch sehr amüsanten) Resultaten führt. Das Tracking kann durch eine schnellere Kamera stabiler gemacht werden, da die momentan sehr stringenten Restriktionen betreffend Bewegungsgeschwindigkeit so fallen würden. Aber auch die Wahl besserer softwareseitigen Restriktionen kann zu besseren Resultaten führen.

Ein weiterer Punkt sind die Anzahl der verfolgbaren Bewegungen. Durch den Einsatz von mehr Markern könnte auch die Rotation der einzelnen Extremitäten getrackt werden. Auch denkbar wäre der Einsatz für ein Gesichts- und Handtracking. Weiter würden mehrere Marker pro Gelenk (z.B. links und rechts vom Knie) die Chance erhöhen, dass zumindest einer der Marker erkannt wird und somit das Gelenk nicht aus dem Tracking fällt. Professionelle Systeme arbeiten mit einer weit höheren Anzahl an Markern, was aber auch einen grossen Mehraufwand beim Berechnen des Modells mit sich bringt. Damit das System in der Industrie anwendbar wird, müssten die resultierenden Animationen zusätzlich in einem gängigen Fileformat gespeichert werden können.

## 7. Fazit

Professionelle Motion Capturing Systeme sind teuer und die Animation von Hand ist aufwändig. Im Verlauf dieser Arbeit wurde ein System erstellt, welches die Bewegungen eines Schauspielers auf ein virtuelles Modell überträgt. Im momentanen Zustand ist unser System wenig mehr als ein Proof of Concept. Es ist aber davon auszugehen, dass durch mehr Aufwand, vor allem im Bereich der Hardware, das System so verbessert werden könnte, dass es konkurrenzfähig wäre.

Die Genauigkeit, die unser System erreicht, hat uns positiv überrascht. Wir haben die gesteckten Ziele in diesem Bereich als eher sportlich angesehen. Die Genauigkeit würde sich mit geringem finanziellen Mehraufwand auch noch weiter erhöhen lassen. Die Hardware ist eher zweckmäßig und hat nicht alle Ziele erreicht. Dennoch konnten wir vernünftige Ergebnisse damit erzielen. Mit entsprechendem Know-How und der Zeit ein eigenes Kamerasytem von Grund auf selber zu bauen, wäre aber auch diese Hürde mit nur geringem finanziellen Mehraufwand zu bewältigen.

Die Anzahl der Systeme und Komponenten haben die Entwicklung erschwert. Fortschritt war nur in kleinen Schritten zu erzielen. Mit der Middleware haben wir ein weiteres, uns unbekanntes Feld betreten, was ebenfalls eine Menge Zeit in Anspruch genommen hat.

Schlussendlich sind wir mit dem Ergebnis, trotz den Einschränkungen welche die Hardware mit sich bringt, sehr zufrieden. Wir haben das Hauptziel, ein funktionierendes Motion Capturing System zu bauen, erreicht. Dadurch, dass die Arbeit in der Entwicklung eines Gesamtsystems bestand, konnten wir unsere Fähigkeiten in einer Vielzahl von Bereichen unter Beweis stellen. Wir waren mit Themen wie Bildverarbeitungsalgorithmen, Middleware, Einbindung von Kernelmodulen sowie Evaluation und Bau von Hardware stets gefordert. Die Umsetzung eines (zumindest annähernd) einsatzbereiten Produktes war sehr spannend. Dies war auch einer der Anreize, uns dieser anspruchsvollen Aufgabe zu stellen. Das Resultat, eine von unserem System aufgezeichnete Bewegung wiederzugeben, erfüllt uns auch mit einem gewissen Mass an Stolz.



# **Selbständigkeitserklärung**

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt haben. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Biel, 17.01.2019

Namen Vornamen: Dellspurger Jan Sheppard David

Unterschriften: ..... ....







# Glossar

**Animation** Die Beschreibung eines Bewegungsablaufs, anhand dessen ein Computermodell transformiert werden kann um diesen Bewegungsablauf graphisch darzustellen.

**Application Programming Interface, API** Die für einen das Programm verwendenden Entwickler zugängliche Schnittstelle [24].

**Array** Eine Liste von zusammenhängenden Daten desselben Typs.

**Branch (GIT)** Eine Kopie der abgelegten Daten mit unabhängigen Veränderungen.

**Buffer** Ein dedizierter Speicherbereich.

**Bus** Ein Datenübertragungsleitung.

**Bytestream** Eine Sequenz aus Bits.

**casten** Das uminterpretieren eines Datenzeigers.

**Clockwise, CW** Im Uhrzeigersinn.

**Clockwise, CW** Im Gegenuhrzeigersinn.

**Computermodell** Die Beschreibung eines Objektes, welche es erlaubt dieses Darzustellen oder weiterzuverarbeiten.

**Feature** Eine Eigenschaft eines Computerprogramms.

**Field of View, FOV** Aufnahmewinkel einer Kamera, normalerweise in horizontaler und vertikaler Richtung angegeben.

**Field-programmable gate array, FPGA** Ein integrierter Schaltkreis mit programmierbarer Schaltstruktur [25].

**Frame** Das Bild einer Kamera.

**Framerate** Anzahl Bilder in einem gewissen Zeitabschnitt.

**Git** Ein verteiltes Versionierungssystem, entwickelt von Linus Torvalds.

**Github** Eine Website welche Git-Repositories verwaltet.

**Infrarot, IR** Licht mit einer Wellenlänge zwischen 700nm und 1mm [5].

**Issue** Ein noch zu lösendes Problem bei einem Computerprogramm.

**JPG** Ein verlustbehaftetes Bildkompressionsverfahren.

**Leuchtdiode, LED** Ein lichtemittierendes Halbleiter-Bauelement [26].

**Library** Ein Stück wiederverwendbare Software.

**Marker** Für einen Computer gut segmentierbare Objekte, welche zum Tracking verwendet werden.

**Middleware** Software zum Datenaustausch zwischen Anwendungsprogrammen [wiki:middleware](#).

**Modell** Siehe Computermodell.

**Modelmatching** Ein Algorithmus welcher Marker an Modellpunkte zuweist.

**Modeltracking** Ein Algorithmus Modellpunkte aus vorgängig bekannten Positionen zuweist.

**Motion Capturing** Der Prozess, welcher Bewegungen eines Menschen auf ein Computermodell überträgt.

**Motion Tracking** Siehe Motion Capturing.

**Payload** Die Daten eines Netzwerkpaketes.

**Pose Estimation** Bestimmung der Position der Kamera.

**Power over Ethernet, POE** Eine Technologie welche es erlaubt eine Stromversorgung über ein Ethernetkabel zu gewährleisten [27].

**Printed Circuit Board, PCB** Ein Träger für elektronische Bauteile [28].

**Queue** Eine nach einem Prioritätsmaß sortierte Liste.

**Raspberry Pi** Ein günstiger Single-Board-Computer.

**Repository** Eine Datenablage auf einem Server.

**retroreflektiv** Ein Material, welches einfallendes Licht in die Ursprungsrichtung zurückreflektiert.

**Rig** Die Definition eines Gerüstes welche als Grundlage für Animationen dient.

**segmentieren** Das Lösen eines Objektes auf einem Bild von seinem Hintergrund.

**Socket** Ein Endpunkt zum erhalten oder versenden von Netzwerkpaketen.

**Spiel-Engine** Eine Software-Bibliothek, welche die Basiskomponenten für das Programmieren von Computerspielen wie Rendering, Physikberechnungen und AI implementiert.

**System Call** Ein vom Betriebssystem ausgeführter Programmaufruf um auf Systemressourcen zuzugreifen.

**Thread** Ein Subprozess, welcher parallel ausgeführt werden kann.

**triangulieren** Das Berechnen der Position eines Objektes im dreidimensionalen Raum anhand mehrerer bekannter Beobachtungspunkte.

# Literatur

- [1] *OptiTrack - Flex3*. URL: <https://optitrack.com/products/flex-3/>.
- [2] *Rahmenreglement für Kompetenznachweise an der Berner Fachhochschule (KFN)*. 2005.
- [3] Jonas Nydegger. *Git Good*. 2018.
- [4] *Wikipedia - Retroreflector*. URL: <https://en.wikipedia.org/wiki/Retroreflector>.
- [5] *Wikipedia - Near Infrared*. URL: [https://en.wikipedia.org/wiki/Infrared#Regions\\_within\\_the\\_infrared](https://en.wikipedia.org/wiki/Infrared#Regions_within_the_infrared).
- [6] *ZMQ API - zmq\_socket(3)*. URL: [api.zeromq.org/4-2:zmq-socket](http://api.zeromq.org/4-2:zmq-socket).
- [7] Cern. *Middleware trends and market leaders 2011*. URL: <http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/frbhmult05.pdf>.
- [8] Wizards RPG Team. *Dungeons & Dragons Monster Manual*. 5th Edition. Wizards of the Coast, 2014.
- [9] *Picamera - Sensor Modes*. URL: <https://picamera.readthedocs.io/en/latest/fov.html#sensor-modes>.
- [10] NXP Semiconductors. *I2C - bus specification and user manual*. 2014. URL: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>.
- [11] *OptiTrack - Marker*. URL: <https://optitrack.com/products/motion-capture-markers/>.
- [12] *Linux Media System Subsystem - Image Formats*. URL: <https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/pixfmt.html>.
- [13] *Linux Media System Subsystem - Input/Output*. URL: <https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/io.html>.
- [14] Jean-Yves Bouguet. *Camera calibration tool box for matlab [eb/ol]*. 2004.
- [15] *Wikipedia - Euler–Rodrigues parameters*. URL: [https://en.wikipedia.org/wiki/Euler%20%93Rodrigues\\_parameters](https://en.wikipedia.org/wiki/Euler%20%93Rodrigues_parameters).
- [16] *Raspberry Pi - Camera Module*. URL: <https://www.raspberrypi.org/documentation/hardware/camera/>.
- [17] *OpenCV Simple Blob Detector Class Reference*. URL: [https://docs.opencv.org/3.4.3/d0/d7a/classcv\\_1\\_1SimpleBlobDetector.html](https://docs.opencv.org/3.4.3/d0/d7a/classcv_1_1SimpleBlobDetector.html).
- [18] *Wikipedia - Ray*. URL: [https://en.wikipedia.org/wiki/Line\\_\(geometry\)#Ray](https://en.wikipedia.org/wiki/Line_(geometry)#Ray).
- [19] Andrew S. Glassner et al. *Graphics Gems*. 1st Edition. 1990.
- [20] *Wikipedia - Signed area*. URL: [https://en.wikipedia.org/wiki/Signed\\_area](https://en.wikipedia.org/wiki/Signed_area).
- [21] *OpenGL Wiki - Face Culling*. URL: [https://www.khronos.org/opengl/wiki/Face\\_Culling](https://www.khronos.org/opengl/wiki/Face_Culling).
- [22] *Wikipedia - Nearest neighbor search*. URL: [https://en.wikipedia.org/wiki/Nearest\\_neighbor\\_search](https://en.wikipedia.org/wiki/Nearest_neighbor_search).
- [23] *Sintel*. 2010.
- [24] *Wikipedia - Application Programming Interface*. URL: [https://en.wikipedia.org/wiki/Application\\_programming\\_interface](https://en.wikipedia.org/wiki/Application_programming_interface).
- [25] *Wikipedia - Field-programmable gate array*. URL: [https://en.wikipedia.org/wiki/Field-programmable\\_gate\\_array](https://en.wikipedia.org/wiki/Field-programmable_gate_array).
- [26] *Wikipedia - Light-emitting diode*. URL: [https://en.wikipedia.org/wiki/Light-emitting\\_diode](https://en.wikipedia.org/wiki/Light-emitting_diode).
- [27] *Wikipedia - Power over Ethernet*. URL: [https://en.wikipedia.org/wiki/Printed\\_circuit\\_board](https://en.wikipedia.org/wiki/Printed_circuit_board).
- [28] *Wikipedia - Printed Circuit Board*. URL: [https://en.wikipedia.org/wiki/Printed\\_circuit\\_board](https://en.wikipedia.org/wiki/Printed_circuit_board).



# Abbildungsverzeichnis

|       |   |    |
|-------|---|----|
| 1.1.  | Legende für Komponentendiagramme . . . . .  | 2  |
| 1.2.  | Legende für Zustandsdiagramme . . . . .   | 3  |
| 1.3.  | Legende für Sequenzdiagramme . . . . .  | 3  |
| 3.1.  | Arbeitsstunden Diagramm . . . . .   | 9  |
| 4.1.  | Konzept für die Softwarearchitektur . . . . .   | 11 |
| 4.2.  | Damit jeder Punkt eines Menschen von mindestens zwei Kameras gesehen wird, sind mindestens vier Kameras nötig. . . . .  | 12 |
| 4.3.  | Aufbau des Mimikry Data Protocol . . . . .  | 13 |
| 4.4.  | Markerkonfiguration . . . . .   | 14 |
| 4.5.  | Ein Beholder [8] . . . . .  | 15 |
| 5.1.  | Eine der sechs Kameras . . . . .  | 17 |
| 5.2.  | Die FOVs der verschiedenen Kameramodi [9] . . . . .   | 18 |
| 5.3.  | Prototypen mit verschiedenen Folien. Links: 3M 983-10, Mitte: 3M 3210, Rechts: 3M 3150 . . . . .  | 19 |
| 5.4.  | Spotter Komponenten . . . . .   | 20 |
| 5.5.  | Spotter Statemachine . . . . .  | 20 |
| 5.6.  | Synchronisierung der Bildaufnahme . . . . .   | 24 |
| 5.7.  | Beispiele für Bilder, welche für die intrinsische Kalibrierung verwendet wurden. . . . .  | 26 |
| 5.8.  | Koordinatensystem definiert durch ein Schachbrett. Wenn das Schachbrett auf dem Boden liegt, zeigt die Z-Achse nach unten. . . . .                                  | 27 |
| 5.9.  | Das System mit dem Kalibriertarget . . . . .  | 28 |
| 5.10. | Links das Graustufenbild, rechts dasselbe Bild binarisiert mit einem Schwellwert von 160. . . . .   | 29 |
| 5.11. | Aufbau einer Spotter Payload Message für $n$ Strahlen . . . . .   | 31 |
| 5.12. | Beholder Komponenten . . . . .  | 31 |
| 5.13. | Beholder Verarbeitungsthread Statemachine . . . . .   | 32 |
| 5.14. | Generierung von Threeway-Intersections . . . . .  | 34 |
| 5.15. | Der Zylinder beinhaltet alle Körpermarker, sowie die Schulter- und Knie- und Fussmarker . . . . .   | 35 |
| 5.16. | Ausgangslage für die Zuweisung an die linke und rechte Modellseite (von oben betrachtet) . . . . .  | 36 |
| 5.17. | Bestimmung eines vorliegenden Punktes . . . . .   | 36 |
| 5.18. | Dreiecke zwischen Brust $c$ , vorliegendem Punkt $f$ und Kandidaten $s_1$ und $s_2$ . Man bemerke die entgegengesetzten Laufrichtungen der beiden Dreiecke. . . . . | 36 |
| 5.19. | Ein Modell bei fehlgeschlagenem Tracking . . . . .  | 38 |
| 5.20. | Aufbau einer Beholder Payload Message . . . . .   | 38 |
| 5.21. | Screenshot des GUI . . . . .  | 39 |
| 5.22. | Hierarchische Struktur des Rig . . . . .  | 40 |
| 5.23. | Sintel mit den den Markern entsprechenden Skelett . . . . .   | 40 |
| 5.24. | Eine winkende Sintel neben einem winkenden Strichmann . . . . .   | 41 |
| 6.1.  | Laufzeit der Markerextraktion in Millisekunden bei 10, 20 und 30 Markern. . . . .   | 43 |
| 6.2.  | Laufzeit der Schnittpunktberechnung in Millisekunden bei 10, 20 und 30 Markern. . . . .   | 44 |
| 6.3.  | Messaufbau mit Maurerlot und retroreflektivem Marker . . . . .  | 45 |



# Tabellenverzeichnis

|   |    |
|---|----|
| 3.1. Zeitplanung . . . . .                                      | 7  |
| 3.2. Projekt Controlling . . . . .                              | 8  |
| 5.1. Capture Modi der Raspberry Pi NoIR Kamera v2 [9] . . . . . | 18 |
| 6.1. Abweichungen Markerdetektion . . . . .                     | 45 |
| 6.2. Referenzsystem von Optitrack . . . . .                     | 46 |
| 6.3. Unser System . . . . .                                     | 46 |
| D.1. Quartile der Markerextraktion . . . . .                    | 71 |
| D.2. Quartile der Schnittpunktberechnung . . . . .              | 71 |



# **APPENDIX**



# A. Spotter Installationsanleitung

Nachdem der Spotter zusammengebaut wurde, müssen mehrere Schritte für die korrekte Konfiguration ausgeführt werden.

Zuerst muss dem Raspberry Pi eine IP zugewiesen werden. Danach muss im Menu unter "Application Menu - Preferences - Raspberry Pi Configuration" unter dem Reiter "Interfaces" folgende Optionen aktiviert werden.

- Camera
- SSH
- I2C
- VNC

Anschliessend muss die Kamera konfiguriert werden. Dies kann mit folgenden Befehlen über die Linux Shell bewerkstelligt werden.

```
1 cd /etc/modprobe.d
2 sudo -s
3 touch v4l2.conf
4
5 echo "options bcm2835-v4l2 max_video_width=1640" >> v4l2.conf
6 echo "options bcm2835-v4l2 max_video_height=1232" >> v4l2.conf
7
8 cd /etc
9 echo "bcm2835-v4l2" >> modules
10
11 exit
```

Daraufhin muss das der Raspberry Pi neu gestartet werden.

Optional kann nun noch ein zuvor generierter SSH-Key auf den Spotter kopiert werden, damit eine Verbindung ohne Passworteingabe möglich ist. In unserem Fall verwenden wir einen SSH-Key für alle Spotter, dieser liegt im Projektverzeichnis im Ordner "keys".

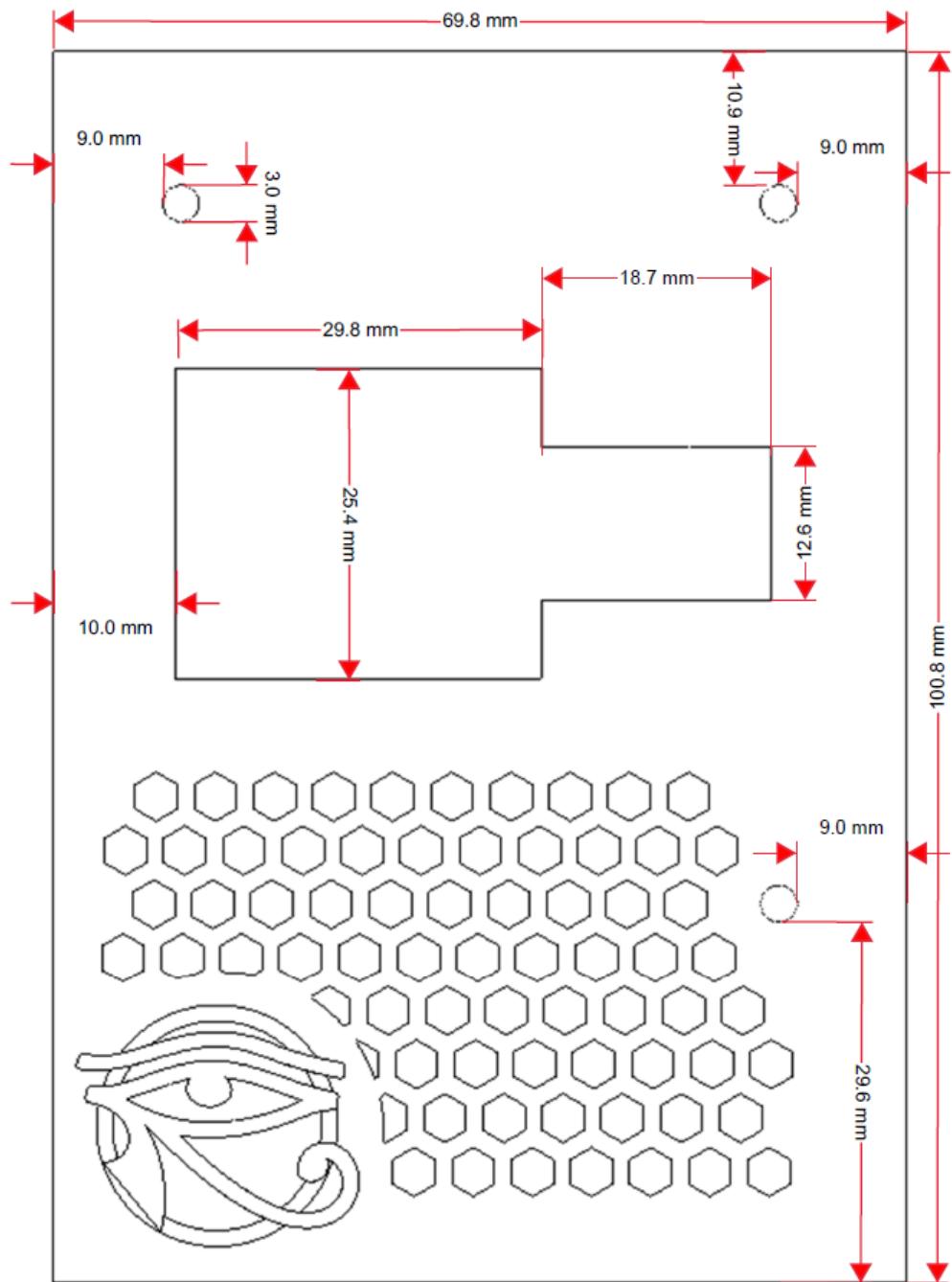
Folgender Befehl kann von einem Computer ausgeführt werden. Der Benutzername und die IP können je Konfiguration des Raspberry Pi variieren

```
1 ssh-copy-id -i mimikry/keys/spotter_rsa.pub pi@192.168.1.100
```

Anschliessend muss das System nochmal neu gestartet werden.



## B. Spotter Frontplatte





## C. Messprotokoll Präzisionsanalyse

| Erwartete Position [cm] |          |          | Gemessene Position [cm] |          |          |
|-------------------------|----------|----------|-------------------------|----------|----------|
| <b>X</b>                | <b>Y</b> | <b>Z</b> | <b>X</b>                | <b>Y</b> | <b>Z</b> |
| 0.0                     | 0.0      | 1.0      | -0.1                    | 0.0      | 1.2      |
| 0.0                     | 121.0    | 1.0      | -0.1                    | 121.3    | 0.9      |
| 84.7                    | 0.0      | 1.0      | 84.8                    | -0.1     | 1.1      |
| 84.7                    | 121.0    | 1.0      | 84.7                    | 121.1    | 1.0      |
| 0.0                     | 0.0      | 105.0    | 0.1                     | 0.6      | 106.0    |
| 0.0                     | 121.0    | 105.0    | -0.1                    | 121.3    | 105.5    |
| 84.7                    | 0.0      | 105.0    | 84.9                    | 0.5      | 106.0    |
| 84.7                    | 121.0    | 105.0    | 84.6                    | 121.6    | 105.7    |
| 0.0                     | 0.0      | 170.0    | 0.7                     | 0.0      | 170.9    |
| 0.0                     | 121.0    | 170.0    | 0.6                     | 121.4    | 170.7    |
| 84.7                    | 0.0      | 170.0    | 85.0                    | 0.1      | 171.0    |
| 84.7                    | 121.0    | 170.0    | 84.3                    | 121.3    | 170.7    |
| 0.0                     | 0.0      | 1.0      | 0.0                     | 0.0      | 1.2      |
| 0.0                     | 121.0    | 1.0      | 0.2                     | 121.0    | 1.0      |
| 84.7                    | 0.0      | 1.0      | 84.7                    | -0.1     | 1.0      |
| 84.7                    | 121.0    | 1.0      | 84.8                    | 121.0    | 1.1      |
| 0.0                     | 0.0      | 105.0    | -0.1                    | 0.5      | 106.0    |
| 0.0                     | 121.0    | 105.0    | 0.1                     | 121.5    | 105.6    |
| 84.7                    | 0.0      | 105.0    | 84.7                    | 0.3      | 105.8    |
| 84.7                    | 121.0    | 105.0    | 85.0                    | 121.4    | 105.6    |
| 0.0                     | 0.0      | 170.0    | 0.7                     | 0.4      | 171.0    |
| 0.0                     | 121.0    | 169.8    | 0.5                     | 121.6    | 170.7    |
| 84.7                    | 0.0      | 170.0    | 84.8                    | 0.3      | 170.9    |
| 84.7                    | 121.0    | 169.8    | 84.6                    | 121.4    | 170.8    |



## D. Quartile Laufzeitanalyse

Q0 ist der minimale gemessene Wert, Q4 der maximale. Q1 und Q3 sind jeweils der tiefste, respektive höchste, Wert im unteren und oberen Viertel der gemessenen Werte. Q2 entspricht dem Median.

| Anzahl Marker | Q0 [ms] | Q1 [ms] | Q2 [ms] | Q3 [ms] | Q4 [ms] |
|---------------|---------|---------|---------|---------|---------|
| 10            | 27.239  | 27.873  | 28.080  | 28.242  | 31.178  |
| 20            | 26.393  | 27.576  | 27.962  | 28.342  | 35.175  |
| 30            | 26.842  | 27.374  | 27.585  | 27.843  | 29.028  |

Tabelle D.1.: Quartile der Markerextraktion

| Anzahl Marker | Q0 [ms] | Q1 [ms] | Q2 [ms] | Q3 [ms] | Q4 [ms] |
|---------------|---------|---------|---------|---------|---------|
| 10            | 0.151   | 0.194   | 0.208   | 0.234   | 0.701   |
| 20            | 0.576   | 0.726   | 0.775   | 0.822   | 1.337   |
| 30            | 1.281   | 1.460   | 1.637   | 1.748   | 3.073   |

Tabelle D.2.: Quartile der Schnittpunktberechnung



## E. Inhalt des USB-Stick

Auf dem beigelegten USB-Stick befindet sich folgender Inhalt:

- build.sh (Script zum kompilieren der Anwendungen)
- sources (Der Source-Code für die verschiedenen Teilanwendungen)
  - beholder
  - spotter
  - mimic
  - include
- doc (Dokumentation)
- scripts (Shell-Scripts zum Aufsetzen und Deployen der Spotter)
- models (Das Blender-Modell für die Spotter Frontplatte)
- externals (Software-Sourcen von Drittanbietern)
  - sources (Source-Code der verwendeten Libraries)
    - \* glfw-3.2.1
    - \* libzmq-4.2.5
    - \* opencv-3.4.3
  - raspberrypi (Tools für das Cross-Compilen auf Raspberry Pi)
    - \* tools-master
    - \* raspbian\_rootfs.tar.gz