Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik
Arbeitsgruppe für Autonome intelligente Systeme

Augmented-Reality-Spiele mit 3D-Welterfassung und Ganzkörper-Bewegungserkennung

Bachelor-Arbeit

vorgelegt von:	Daniel Kuhner
Matrikelnummer:	2561119
E-Mail:	kuhnerd@informatik.uni-freiburg.de
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Matthias Teschner
betreut von:	Jürgen Sturm
Datum der Einreichung:	02. September 2010

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Bachelor-Arbeit mit dem Titel

Augmented-Reality-Spiele mit 3D-Welterfassung und Ganzkörper-Bewegungserkennung

selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet wurden und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, als solche kenntlich gemacht wurden. Darüber hinaus erkläre ich, dass diese Projektarbeit nicht, auch nicht auszugsweise, bereits anderweitig verwendet wurde.

Daniel Kuhner

Freiburg im Breisgau, den 02. September 2010

Zusammenfassung

Neue technische Möglichkeiten im Bereich von Ganzkörper-Bewegungserfassung und 3D-Welterfassung erlauben die Umsetzung neuartiger Spielkonzepte. In dieser Arbeit wurde ein tragbares Augmented-Reality-Spiel entwickelt, das es erlaubt, in der wirklichen Welt eine Minigolf-Simulation zu spielen. Im Laufe dieser Arbeit sind eine Reihe neuer Fragestellungen entstanden, zum Beispiel, wie die Spielsimulation, zusammen mit einer Videobrille, 3D-Kamera und der echten Umgebung robust in Übereinstimmung gebracht werden kann und wie die Steuerung des Spiels automatisch an die Körpereigenschaften des Spielers angepasst werden können. Hierzu wird zunächst aus den Kameradaten ein 3D-Mesh generiert, und der Fußboden mittels eines robusten Schätzverfahrens detektiert. Daraus wird die Kameraneigung korrigiert, um Helm, Kamera und Anzug gegenseitig in Übereinstimmung zu bringen. Dieses so erzeugte Mesh wird dann als Weltmodell für eine Physiksimulation verwendet. Zusätzlich simuliere ich einen Minigolfschläger in der Hand des Spielers und einen Ball. Beides zusammen wird dann mit den Bildern der realen Welt überlagert, sodass der Spieler simuliertes Minigolf in einer echten Umgebung spielen kann. Da verschiedene Spieler mit unterschiedlicher Körpergröße, Alter oder Spielstil verschiedene Spielparameter benötigen, stelle ich im zweiten Teil der Arbeit ein Lernverfahren vor, das es ermöglicht, sowohl die Schlägerlänge und -orientierung, als auch die Links/Rechtshändigkeit aus Schlagbewegungen des Spielers zu schätzen. In meinen Experimenten zeige ich, dass mittels, des in dieser Arbeit vorgestellten Verfahrens, die benötigte Hardware automatisch aufeinander abgestimmt werden kann, sodass das simulierte Spiel sich in guter Übereinstimmung mit der wirklichen Welt befindet. Außerdem zeige ich, dass die Anpassung des Spieles an den Spieler zu Verbesserungen in der Bedienbarkeit führen.

Inhaltsverzeichnis

1 Einleitung

1.1. Motivation

Bislang werden viele Spiele hauptsächlich durch konventionelle Eingabegeräte, wie Maus, Tastatur und Joystick, bedient. Ebenso erfolgt die Anzeige häufig auf normalen Monitoren oder Displays. Allerdings wurde in den letzten Jahren eine neue Generation von Eingabegeräten entwickelt, die es ermöglichen, dass der Spieler durch den Einsatz eines Körperteiles oder sogar des ganzen Körpers das Spielgeschehen beeinflussen kann. Als Beispiele sind hier die neuen Spielkonsolen, wie die *Nintendo Wii* oder *Microsoft Kinect*, zu nennen. Die bekannteste Konsole hierbei dürfte die *Nintendo Wii* sein, die ein System verwendet, das unter anderem die dreidimensionale Lage des Controllers ermittelt. Auf diese Weise ist ein neues Spielvergnügen möglich, da der Nutzer körperlich ins Spielgeschehen eingreifen kann. Microsoft entwickelt dieses Prinzip momentan mit *Microsoft Kinect* weiter. Das Verfahren verzichtet gänzlich auf Controller und nutzt stattdessen eine Kamera zur Bewegungserfassung. Es ist also eine Entwicklung hin zu Systemen zu beobachten, die den Spieler direkt ins Spielgeschehen integriert.

Des weiteren wird an neuen Spielen gearbeitet, die die reale Welt als Grundlage verwenden und in diese virtuelle Objekte integrieren. Ein solches Spiel wird als *Augmented-Reality-Spiel* bezeichnet. Diese stellen ein zunehmend wichtiges Thema in der Forschung und Spieleentwicklung dar. Mit diesem Ansatz können Spiele entworfen werden, die grundsätzlich überall verwendet werden können. So ist der Einsatz eines virtuellen Brettspieles als Augmented-Reality-Inhalt denkbar, bei dem mehrere Menschen miteinander spielen. Dieses könnte, bei Verwendung mobiler Hardware, überall eingesetzt werden, ohne dass man das eigentliche (reale) Spiel benötigt. Ein weiteres Beispiel stellen Spiele dar, die eine Wohnung als Level verwenden. So ist es möglich, dass man durch die Wohnung laufen kann, um Aufgaben zu erledigen oder (virtuelle) Gegner zu besiegen. Für die Anzeige solcher Spielinhalte ist ein mobiles Gerät von Vorteil. Hierbei stellen Head mounted displays oder Mini-Projektoren eine vielversprechende Möglichkeit dar.

Die Kombination von Bewegungserfassung und Augmented-Reality erlaubt eine ganz neue Art von Spielen. Vorteilhaft hierbei ist es, wenn die verwendete Hardware so klein und leicht wie möglich ist und schnell in Betrieb genommen werden kann. Es gibt zwar Hardware, die die geforderten Aufgaben, wie Bewegungs- oder Umgebungserfassung, bewerkstelligen können, doch ist diese häufig noch nicht für den mobilen Einsatz geeignet.

Marktfähige Anwendungen im Augmented-Reality-Sektor sollten so ausgelegt werden, dass sie mit kompakter und damit tragbarer Hardware auskommen. Noch besser wäre es natürlich, wenn der Spieler keinerlei Hardware am Körper tragen müsste. Zudem wäre gut, wenn die erweiterten Inhalte dreidimensional dargestellt werden würden und für andere Menschen sichtbar wären. Auf diese Weise wären Spiele möglich, deren Inhalte vollständig in die reale Welt integriert werden. Allerdings fehlt dafür noch die nötige Technik.

1.2. Problemstellung

Diese Bachelorarbeit untersucht die Grundlagen eines Augmented-Reality-Spieles. Ein einfaches Minigolf-Spiel dient zur Demonstration der einzelnen Komponenten. Dabei wird auf folgende Fragen eingegangen:

- Welche Hardwarevoraussetzungen müssen erfüllt werden?
- Wie kann das Kamerabild, der Körper des Spielers, die reale Umgebung und das simulierte Spiel zusammengeführt werden?
- Wie könnte ein Lernverfahren für die Schätzung der Aktionsparameter aussehen?

Für das Projekt wird ein Motion-Capture-Anzug verwendet. Außerdem wird eine 3D-Kamera zur Erfassung der Umwelt und eine Videobrille benutzt. Ein Motion-Capture-Anzug erlaubt eine genaue und einfache Erfassung der Bewegung eines Spielers. Diese genaue Erfassung ermöglicht es, Spiele zu entwickeln, die ein hohes Maß an Genauigkeit erfordern. Außerdem kann ein tragbarer Motion-Capture-Anzug überall verwendet werden, wodurch Spiele nicht auf den Innenbereich beschränkt bleiben. Des weiteren erlaubt eine 3D-Kamera eine zuverlässige und unkomplizierte Erfassung der Umwelt. Ein Minigolfspiel fordert gerade diese Eigenschaften. Die Bewegungserfassung muss möglichst genau erfolgen, da sonst das Schlagen des Balles nicht möglich wäre. Außerdem muss die Umgebung mit ausreichender Qualität erfasst werden, damit der Eindruck entstehen kann, dass der virtuelle Ball mit der realen Welt kollidiert. Die Anzeige des Bildes auf einer Videobrille fördert die Mobilität des Systems und lässt dem Spieler die Freiheit, sich beliebig zu bewegen.

Da jeder Mensch unterschiedliche Körpermaße hat, ist es sinnvoll, die Aktionsparameter aus den Demonstrationen des Spielers zu ermitteln. Bei Einsatz fester Parameter wäre die Menge der möglichen Spieler eingeschränkt, da beispielsweise Kinder andere Parameter benötigen,

als ein Erwachsener. Aus diesem Grund sollte auch unterschieden werden, ob der Spieler Links- oder Rechtshänder ist.

Das implementierte Spiel zeigt dem Spieler einen virtuellen Schläger, den er zum Schlagen des virtuellen Balles verwenden kann. Das aus der 3D-Kamera gewonnene Modell der Umgebung wird in einer Physikengine benutzt, die die Kollision des Balles mit der Umwelt simuliert. Die Erfassung der Umgebung gliedert sich in verschiedene Teilprobleme, nämlich der Erzeugung eines Meshs aus einer Punktwolke und die richtige Transformation vor die virtuelle Kamera. Das Verfahren erlaubt eine robuste Schätzung der nötigen Transformationen, wie die Transformation des Meshs vor die virtuelle Kamera oder die Transformation des 2D-Bildes auf eine virtuelle Leinwand, wodurch die automatisierte Überlagerung des 2D-Bildes und des 3D-Modelles ermöglicht wird.

Neben der Erfassung der Umwelt wurde ein robustes Verfahren entwickelt, das die Parameterschätzung des Minigolfschlägers übernimmt. Die zwei geschätzten Parameter sind die Länge und Orientierung des Schlägers, da diese die wichtigsten Informationen des Schlägers beschreiben. Außerdem schätzt der Ansatz, welche Hand der Spieler für die Schlagbewegung einsetzt. Auf diese Weise kann der Schläger an den Spieler angepasst werden.

1.3. Gliederung der Arbeit

Zunächst folgt in Unterkapitel 1.4 ein Überblick über die themenverwandte Forschung. Danach stellt Kapitel 2 die Grundlagen vor. Hier wird der Begriff „Augmented-Reality“ genauer beleuchtet. Dabei wird erklärt, was Augmented-Reality ist, wo es Anwendung findet und welche Hardware dafür verwendet werden kann. Der zweite Teil dieses Kapitels stellt die für dieses Projekt verwendete Hardware vor, wobei hier auf die 3D-Kamera, den Motion-Capture-Anzug und die Videobrille eingegangen wird. Abschließend wird die Notation eingeführt, die in den folgenden Kapiteln verwendet wird. Auf die Welterfassung wird in Kapitel 3 eingegangen. Es wird erläutert, wie die Stereo-Kamera dazu verwendet wird, um ein Modell der Umgebung zu erzeugen und wie dieses 3D-Modell vor die virtuelle Kamera transformiert wird, die durch die Position des Kopfes bestimmt sei. Anschließend folgt die Beschreibung der 2D-Erfassung. Sie wird dazu verwendet, um die reale Umgebung im Spiel sichtbar zu machen. Kapitel 4 erläutert, wie die Parameter des Minigolfschlägers geschätzt werden. Danach geht Kapitel 5 auf die Implementierung ein. Hier werden die technischen Details der Software und der Aufbau der Hardware erläutert. In Kapitel 6 werden die Ergebnisse der Experimente präsentiert. Abschließend folgen in Kapitel 7 die Zusammenfassung der Arbeit, eine Diskussion und der Ausblick. Anhang A befasst sich mit den mathematischen Grundlagen.

Abbildung 1.1.: Links: Augmented-Reality unter Wasser: Taucher mit virtuellen Muscheln, Fischen und Wasserpflanzen - **Rechts:** SixthSense: Anzeige eines Videos in auf einer Zeitung. Die Marker der Finger sind zu sehen.

1.4. Themenverwandte Arbeiten

AR-Anwendungen sind ein aktuelles Forschungsthema an Universitäten. So werden in vielen Forschungsarbeiten Systeme verwendet, die mittels spezieller Marker und Kameras gewisse Formen und Strukturen erkennen können. Einsatz findet hier häufig das ARToolkit [4], welches in einem Bild nach fixen Schwarz-Weiß-Markern sucht und die Position und Orientierung dieser zurückgibt. So verwendet Oda et al. [33] ein markerbasiertes System, um die Erfassung der Spielfläche und der Gegenstände zu erleichtern. Einsatz findet das System in einem AR-Spiel, in dem ein virtuelles Auto durch einen Fahrradlenker gesteuert wird. Der Fahrradlenker ist hierbei ebenfalls mit Markern versehen. Der Spieler sieht das Auto in einer Videobrille. Dabei können Hindernisse und Wegpunkte mittels weiterer Marker gesetzt und verschoben werden. Zusätzlich werden in [25] ein Datenhandschuh und ein Nintendo Wiimote Controller verwendet, um ein Auto zu steuern. Der Handschuh ermittelt hierbei die Bewegung der Finger. Definierte Fingerbewegungen führen dann zum Beispiel dazu, dass man einen virtuellen Gegenstand hochheben kann. Oda und Feiner stellen in [32] weitere Beispiele für AR-Spiele vor, wie zum Beispiel ein Murmelspiel in einem virtuellen Labyrinth.

Die AR-Ansätze müssen nicht auf trockene Umgebungen beschränkt sein, wie in [7] gezeigt wird. Hier werden Inertial- und Magnetfeldsensoren in Kombination mit Markern eingesetzt, um die Position und Orientierung eines Tauchers zu ermitteln, der über eine wasserdichte Videobrille unter anderem virtuelle Muscheln, Fische und Wasserpflanzen sehen kann (Abbildung 1.1, links).

Ein anderer Ansatz besteht darin, auf Marker zu verzichten und die Position und Orientierung von Gegenständen direkt aus dem Bild zu ermitteln. Löchtefeld et al. [26] hat ein Spiel implementiert, dass ohne Marker auskommt. Dabei wird ein Handy und ein kleiner Projektor eingesetzt, der das Bild direkt auf die Umgebung projiziert. Die Kamera und der Projektor

werden hierzu beispielsweise auf eine Tafel gerichtet. Zeichnet man auf die Tafel einige Linien oder bringt Objekte an der Tafel an, so wird dies von der Kamera erfasst und vom Handy verarbeitet. Ein virtueller Ball, der auf die Tafel projiziert wird, prallt dann an diesen Linien und Objekten ab.

Ebenfalls verwendet Pranav Mistry [29, 30] einen Projektor, um die erweiterten Informationen direkt auf die Umwelt zu projizieren. Eine Kamera erfasst zusätzlich die farblich markierten Finger der Hand, wodurch die Erkennung von Gesten ermöglicht wird. Es wurden zahlreiche Beispielsanwendungen implementiert, die die Funktion des Systems demonstrieren. So sorgt das Malen eines Kreises mit dem Zeigefinger auf den Arm dafür, dass eine virtuelle, analoge Uhr auf den Arm projiziert wird. Außerdem sind beispielsweise das virtuelle Zeichnen auf Papier oder die Anzeige eines Videos in einer Zeitung möglich (Abbildung 1.1, rechts).

Chekhlov et al. wählt in [12] einen auf SLAM (Simultaneous localisation and mapping [24]) basierten Ansatz, der die Umgebung mit einer Kamera erfasst und Ebenen ermittelt. Diese Ebenen werden dann dazu verwendet, einem simulierten Agenten in Form eines Ninjas die Fortbewegung von einer Ebene zur nächsten zu ermöglichen. Dabei werden die Ebenen in Echtzeit ermittelt und aktualisiert. Zur Bestimmung der Ebenen wird der RANSAC-Algorithmus eingesetzt.

Reitinger et al. [39] nutzen ein System, mit dem man die Umgebung erfassen kann. Dabei wird mittels GPS die Position der Person ermittelt. Diese macht mehrere Bilder von Gebäuden, aus denen dann Punktwolken berechnet werden. Auf diese Weise entsteht ein dreidimensionales Modell der Umgebung, welches nach und nach aktualisiert wird. Für die Berechnung der Punktwolken wird die Grafikkarte verwendet, wodurch eine höhere Performance erreicht werden kann. Zunächst werden die relativen Positionen der Kameras für jedes Bild bestimmt. Danach wird mit einem „Plane-Sweep-Ansatz“ das 3D-Modell in Form einer Punktwolke erzeugt. In [45] wird, auf gleichem Ansatz basierend, ein geschlossenes Mesh berechnet.

Die Erfassung der Bewegung kann auch durch Kameras geschehen. In [19] werden die Bewegungsdaten aus den Bildern mehrerer Kameras erfasst. Dabei wird auf Marker und Sensoren verzichtet.

In dieser Arbeit wird ein Lernverfahren verwendet, das aus einer Demonstration Aktionsparameter schätzt. Dieser „Learning by Demonstration“-Ansatz wird häufig in der Robotik verwendet. So nutzt Argall [2] ein Verfahren, dass neben den Demonstrationen auch die Kritik des Lehrers berücksichtigt. In [13] wird ein Ansatz beschrieben, der aus mehreren Demonstrationen ein entsprechendes Modell extrahiert.

2 Grundlagen

Dieses Kapitel dient zur Erläuterung der Grundlagen. Zunächst wird in Unterkapitel 2.1 erklärt, was Augmented Reality ist. In Sektion 2.2 folgt eine Einführung in die verwendete Hardware.

2.1. Augmented Reality

In diesem Unterkapitel werden die verschiedene Aspekte von Augmented Reality erläutert. Es wird erklärt, was man unter Augmented-Reality versteht. Danach folgt ein Überblick über die wichtigsten Einsatzgebiete. Abschließend werden die verschiedenen Arten von Hardware vorgestellt, die zur Anzeige der Augmented-Reality-Inhalte verwendet werden.

2.1.1. Was ist Augmented Reality?

Augmented Reality (AR, deutsch: „erweiterte Realität“) stellt ein Unterbereich von Mixed Reality dar. Abbildung 2.1 zeigt, wie Milgram [28] „Mixed Reality“ unterteilt. Dabei wird der Bereich zwischen der realen und virtuellen Welt als „Mixed Reality“ bezeichnet. AR liegt dabei näher an der Realität, während Augmented Virtuality hauptsächlich die Nutzung virtueller Techniken vorsieht.

Bei AR handelt es sich also um die Erweiterung der realen Welt um virtuelle Elemente, die typischerweise durch einen Computer erstellt werden. Azuma et al. [5] fordern folgende Eigenschaften, die von einer AR-Anwendung erfüllt sein müssen:

- Kombination von realen und virtuellen Objekten in einer realen Umgebung,
- die Anwendung ist interaktiv und in Echtzeit nutzbar und
- das gegenseitige Abgleichen der realen und virtuellen Objekte muss gewährleistet sein.

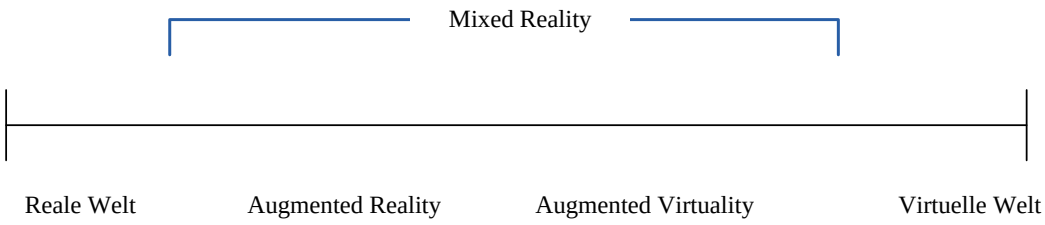

Abbildung 2.1.: Milgram's Überblick über „Mixed Reality"

Abbildung 2.2.: Links: AR bei Sportübertragungen (Football): Die gelbe Linie wurde zusätzlich eingefügt - **Rechts:** Sony EyePet

2.1.2. Einsatzgebiete

Es liegt nahe, dass AR in vielen Gebieten Einsatz findet. Die folgende Liste gibt einen Überblick über die wichtigsten Applikationen:

- *Spiele:* In Spielen setzt man AR dazu ein, um das Spielgeschehen in die reale Welt zu transferieren. Für die Sony PlayStation 3 wurde das Spiel „EyePet" [15] veröffentlicht. Dieses zeigt neben der - mit einer Kamera erfassten - Umwelt auch virtuelle Haustiere, die sich auf dem Boden bewegen und mit denen der Nutzer interagieren kann. So kann das Tier beispielsweise gefüttert werden. In Abbildung 2.2 ist ein Bild von EyePet zu sehen. Ein weiteres Beispiel ist die Entwicklung des auf „Quake" basierenden AR-Spieles namens „ARQuake" [3]. Dieser First-Person-Shooter nutzt unter anderem GPS zur Positionsbestimmung, wodurch der Spieler auch in Außenumgebungen spielen kann. Die Anzeige erfolgt über ein Head mounted display.
- *Sonstige Unterhaltung:* Neben Spielen findet AR in vielen Bereichen der Unterhaltungsindustrie Anklang. So wird AR schon seit einigen Jahren in Sportübertragungen eingesetzt, um zusätzliche Informationen anzeigen zu können. Als Beispiel werden beim Football zusätzliche, virtuelle Linien auf dem Spielfeld angezeigt, damit die Erkennung von Details vereinfacht wird (Abbildung 2.2, links).
- *Navigation:* AR wird hier dazu verwendet, um die Navigationsinformationen direkt mit der echten Umgebung zu überlagern. Dies geschieht beispielsweise auf einem kleinen

Abbildung 2.3.: Links: Navigationssystem Wikitude-Drive auf Android-Handy - **Rechts:** Überlagerung eines Fußes mit einem Bild aus dem Computertomografen (TU München)

Monitor. Auf diese Weise ist es für den Fahrer einfacher, sich zu orientieren, da er auf dem Bildschirm beides, also die echte und virtuelle Welt, sieht. Abbildung 2.3 zeigt ein Navigationssystem, welches neben der tatsächlichen Straße auch die virtuelle Route anzeigt. Des weiteren können auch zusätzliche Informationen zu der Umgebung angezeigt werden - der Name eines Gebäudes ist ein Beispiel.

- *Medizin:* Medizinische Anwendungen dienen vor allem der Simulation von Operationen, beziehungsweise der Visualisierung gewisser Informationen, die von außen (ohne Operation) nicht sichtbar sind. Die TU München hat ein System entwickelt, das eine Stelle des Körpers mit den Daten eines Computertomographen überlagert. So kann beispielsweise ein Knochen sichtbar gemacht werden, wie in Abbildung 2.3 dargestellt wird.
- *Wartung/Reparatur:* Auch in der Wartung und Reparatur komplexer Systeme kann AR eine Erleichterung darstellen, da ein derartiges System dem Menschen helfen kann, schneller und effizienter zu arbeiten.
- *Militär:* Eine militärische AR-Anwendung wird unter anderem dazu verwendet, um den Soldaten zusätzliche Informationen zur Umgebung zu bieten. So werden beispielsweise strategische Punkte markiert, Karten angezeigt und navigiert.

2.1.3. Hardware

Prinzipiell lassen sich AR-Informationen auf drei verschiedene Arten anzeigen [5]:

- *Head-mounted displays* (HMD)
- *normale Monitore*
- *Projektion*

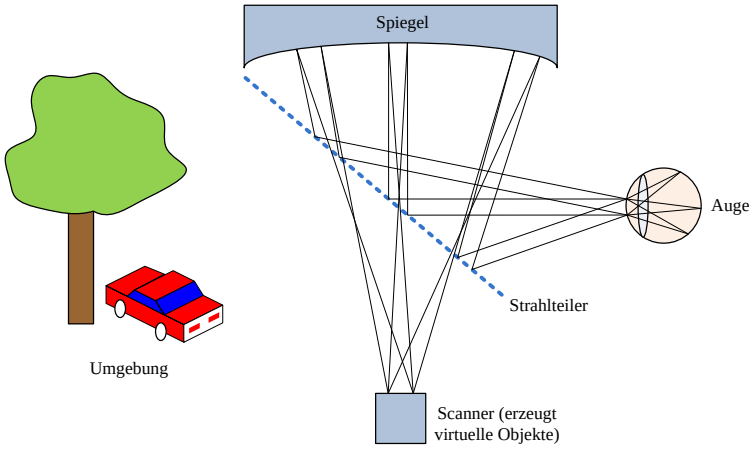

Abbildung 2.4.: Schematische Zeichnung eines Virtual Retinal Displays mit Strahlteiler: Das Bild setzt sich aus den Lichtstrahlen der realen Welt und der projizierten Laserstrahlen zusammen.

Die wichtigste Art stellen die *HMD's* dar. Dabei wird eine Brille verwendet, die die Bilder vor den Augen anzeigt. Es existieren hierbei drei verschiedene Arten. Einerseits kann eine undurchsichtige Brille benutzt werden, bei der man die Bilder der realen Umgebung mit einer Kamera aufnimmt und zusammen mit den virtuellen Informationen auf den Displays der Brille anzeigt. Nachteilhaft an diesen Brillen ist im Allgemeinen der geringe Öffnungswinkel, da die Anzeigefläche sehr begrenzt ist. Ein anderer Ansatz, der dieses Problem umgeht, besteht darin, eine halbtransparente Brille zu verwenden, auf der nur die virtuellen Informationen angezeigt werden - das reale Bild sieht man direkt.

Virtual Retinal Displays (VRD) [21] stellen die dritte Art der HMD's dar. Hier werden die virtuellen Bilder direkt auf die Retina abgebildet. Zum Einsatz kommt hier ein schwacher Laser, der das Bild erzeugt, indem der Strahl durch eine Linse aufs Auge fällt. Abbildung 2.4 zeigt eine schematische Zeichnung des Funktionsprinzips einer halbtransparenten Brille. Vorteile dieses Verfahrens liegen in einer hohen Helligkeit und hohem Kontrast. Außerdem ist der Stromverbrauch eines solchen System gering und die Schärfentiefe groß. Das „Human Interface Technology Laboratory" der Universität Washington, die dieses System entwickelt haben, arbeiten an einer Brille, die auf dem gleichen Prinzip beruht, aber dreidimensionale Bilder liefern kann [41].

Bei *Projektionsdisplays* kommt ein kleiner Projektor zum Einsatz, der die virtuellen Informationen direkt auf die tatsächliche Umgebung projiziert. Somit sind weder Brillen noch Monitore nötig. Allerdings lassen sich momentan noch keine virtuellen 3D-Objekte erzeugen, wodurch dieses Verfahren auch nur eingeschränkt verwendet werden kann.

Auch *normale Monitore* und *Handheld-Displays* eignen sich für AR-Anwendungen. Beispiele hierfür sind Fernsehgeräte (Sportübertragungen), Handys (Spiele) und Navigationssysteme (AR-Navigation).

HERSTELLER	PRODUKT	ART
Microvision	SHOWWX™ Laser Pico Projector [27]	Projektion
Brother	AiRScouter™ (Prototyp) [8]	VRD
Vuzix	WRAP 920AR [14]	HMD

Tabelle 2.1.: Einige Beispiele für die verschiedenen Anzeigemöglichkeiten in AR-Anwendungen

Abbildung 2.5.: Bumblebee2 Stereo-Kamera von Point-Grey

Die Tabelle 2.1 stellt einige Geräte vor, die für den Einsatz in AR-Anwendungen geeignet sind.

2.2. Verwendete Hardware

Dieses Kapitel dient zur Beschreibung der verwendeten Hardware. Es wurde ein Motion-Capture-Anzug zur Bewegungserfassung verwendet. Kombiniert mit einer 3D-Kamera ist es somit möglich, sich in der realen Welt zu bewegen und gleichzeitig durch eine Videobrille virtuelle Inhalte zu sehen, die mit der realen Welt interagieren können. Dabei wird die Kamera zur Erzeugung eines 3D-Modelles der Umgebung verwendet, das dann für die Kollisionserkennung der virtuellen Objekte mit der realen Welt verwendet wird.

2.2.1. 3D-Kamera

Bei der verwendeten Kamera handelt es sich um eine Bumblebee2-Stereo-Kamera von Point-Grey [38]. Das Modell ist mit zwei Kameras ausgestattet, die es ermöglichen, ein Szene aus zwei verschiedenen Blickrichtungen aufzunehmen. Aus diesen Bildern lassen sich dann die Tiefeninformationen ermitteln, wodurch eine 3D-Punktwolke entsteht. In Abbildung 2.5 ist die Kamera dargestellt.

Zudem wird die Kamera für die Erfassung der 2D-Bilder verwendet. Hierbei muss das Bild rektifiziert werden, da der Öffnungswinkel von $97°$ der Kamera recht groß ist (Fish-Eye-

Abbildung 2.6.: Links: Rohbild mit tonnenförmiger Störung, die durch die Kameralinsen verursacht werden - **Rechts:** Rektifiziertes Bild mit geraden Kanten

Effekt). Für die Berechnung der Punktwolke wird das Triclops SDK [36] verwendet, welches von Point-Grey angeboten wird. Das Verfahren zur Erzeugung einer Punktwolke untergliedert sich in mehrere Einzelschritte. Die Bilder der Kamera werden zunächst *rektifiziert*. Aus den runden Kanten des Originalbildes entstehen hierdurch wieder gerade Kanten. Abbildung 2.6 zeigt ein Rohbild und das Ergebnis nach der Rektifizierung.

Danach wird in beiden Bildern nach Übereinstimmungen gesucht, wofür ein *Block-Matching*-Ansatz [35, S. 115ff] verwendet wird. Block-Matching betrachtet in beiden Bildern immer ganze Blöcke und prüft diese auf Übereinstimmung. Hierfür wird die *Summe der absoluten Differenzen* (SAD) gebildet, die angibt, wie groß der Unterschied zweier Bilder oder Bildbereiche ist. Allgemein sieht diese Summe (für gleich große Bilder) wie folgt aus [10]:

$$\mathrm{SAD} = \sum_{x,y} \left| I_{\text{rechts}}(x,y) - I_{\text{links}}(y+d,y) \right|, \tag{2.1}$$

wobei I_{rechts} und I_{links} die zwei betrachteten Bilder sind. x und y geben die Koordinaten und d die Verschiebung an, in der im linken Bild gesucht wird. Eine große Übereinstimmung der Bilder bedeutet, dass der SAD-Wert klein ist; entsprechend wird der Wert groß, wenn die Übereinstimmung gering ist.

Triclops setzt die Suchgrenzen, beziehungsweise die *Disparaty*-Grenzen $d_{\min}$ und $d_{\max}$ fest. Außerdem wird jeweils nur ein Block um den betrachteten Pixel auf Übereinstimmung geprüft. Sei dieser Block ein Quadrat mit der Seitenlänge m. Für die tatsächliche Bestimmung der Übereinstimmung wird in Triclops

$$\min_{d \in [d_{\min}, d_{\max}]} \left[\sum_{i=-\frac{m}{2}}^{\frac{m}{2}} \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} \left| I_{\text{rechts}}(x+i,y+j) - I_{\text{left}}(x+i+d,y+j) \right| \right] \tag{2.2}$$

verwendet. Die Rektifizierung des Bildes ist für diesen Schritt wichtig, da der Ansatz bei der Suche eines übereinstimmenden Bereiches jeweils die gleiche Zeile (für horizontal ausgerich-

tete Kameras) oder Spalte (für vertikal ausgerichtete Kameras) betrachtet.

Durch das *Block-Matching* erhält man den relativen Abstand d zwischen entsprechenden Punkten im linken und rechten Bild, indem man die x-Werte voneinander abzieht. Sei der *Disparaty-Wert* $D(\mathbf{a})$ eines Punktes $\mathbf{a}$ durch

$$D(\mathbf{a}) = \mathbf{a}_{\text{links},x} - \mathbf{a}_{\text{rechts},x} \tag{2.3}$$

gegeben, wobei $\mathbf{a}_{\text{links},x}$ und $\mathbf{a}_{\text{rechts},x}$ den x-Wert des im linken und rechten Bild gefundenen Punktes $\mathbf{a}$ angibt.

Aus den gewonnen *Disparaty-Werten* kann nun der Tiefenwert [10, S. 994]

$$z = f\frac{T}{D(\mathbf{a})} \tag{2.4}$$

ermittelt werden. Hierfür wird die *Baseline* T (Abstand der beiden Kameras), die Brennweite f und der Disparaty-Wert des Punktes verwendet.

Die Tabelle B.1 im Anhang beschreibt die wichtigsten technischen Daten der Kamera.

2.2.2. Motion-Capture-Anzug

Der eingesetzte Motion-Capture-Anzug ist ein MVN-Anzug von Xsens [40] und ist für die Ganzkörper-Bewegungserkennung gedacht. Es ist allerdings auch möglich nur gewisse Körperbereiche zu erfassen. Als Sensoren kommen 17 sogenannte MTx-Tracker zum Einsatz, die Drehratensensoren, Beschleunigungssensoren und Hall-Sonden beinhalten, um die Lage und Orientierung der Körperteile zu bestimmen. Die anschließende Übertragung der Daten erfolgt über ein Bussystem, wodurch nur ein Kabel nötig ist (jeder Sensor hat einen Eingang und einen Ausgang). Das Bussystem befördert die Daten in die 2 Xbus-Master, die die Stromversorgung und die Synchronisierung der Daten sichern. Außerdem werden die Daten von hier über Bluetooth an einen Computer übertragen. Das MVN-Studio, eine dem Anzug beiliegende Software, die unter anderem die Kalibrierung des Anzuges und die Visualisierung der Bewegungsdaten übernimmt, verarbeitet die Daten der Sensoren am Computer. Von hier aus können die Bewegungsdaten (Position und Beschleunigung) über Ethernet an einen anderen Computer übertragen werden. Tabelle B.2 im Anhang fasst die wichtigsten Eigenschaften des Anzugs zusammen.

Abbildung 2.7 zeigt einerseits die Anordnung der 17 *MTx-Tracker* (links) und die Datenpunkte, die letztendlich ausgewertet werden können (rechts). Dabei erhält man die globalen Transformationsmatrizen von insgesamt 23 verschiedenen Punkten.

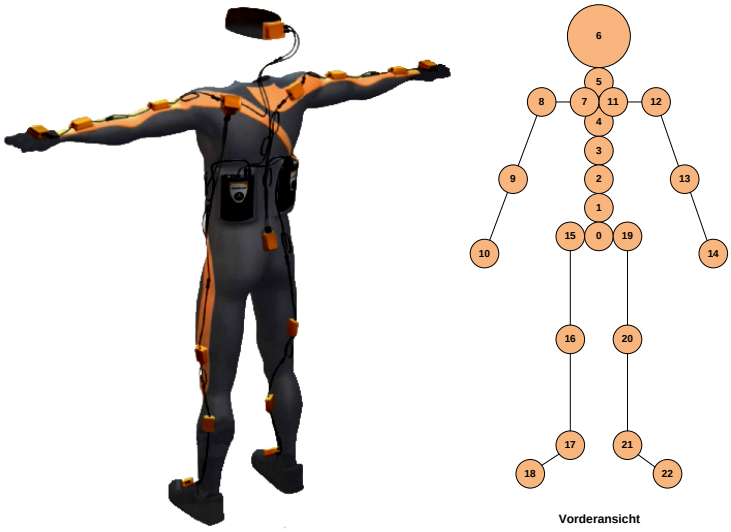

Abbildung 2.7.: Links: MVN-Anzug von Xsens für die Ganzkörper-Bewegungserkennung: Anordnung der 17 Sensoren - **Rechts:** Anordnung der 23 Datenpunkte für die Auswertung

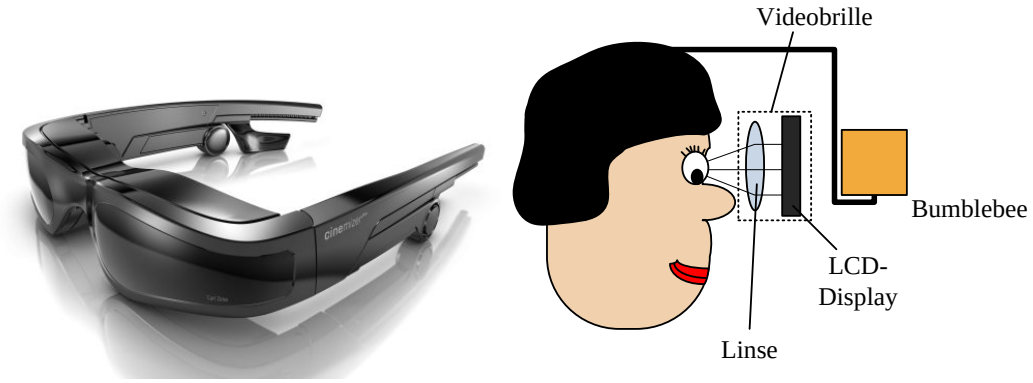

Abbildung 2.8.: Linkes Bild: Videobrille „Cinemizer Plus“ von Zeiss - **Rechtes Bild:** Schematische Darstellung der Brille, kombiniert mit der, vor der Brille befestigten, Bumblebee-Kamera

Die Vorteile des Anzuges liegen darin, dass keine Kameras oder Marker nötig sind, um die Bewegung zu erfassen. Das und die Übertragung der Daten mit Bluetooth sorgt dafür, dass der Anzug auch im Außenbereich eingesetzt werden kann. Weitere Informationen zum Anzug sind in [23, 40] und auf der Xsens-Homepage [44] zu finden.

2.2.3. Head Mounted Display

Für die Anzeige der Bilddaten kommt die Videobrille „Cinemizer plus“ von Zeiss [46] (Abbildung 2.8) zum Einsatz. Die Brille ist zwar nicht für AR-Anwendungen gedacht; sie liefert allerdings eine Basis, mit der man erste Tests machen konnte. Für spätere Ansätze sollte eine andere Brille eingesetzt werden.

Für die Anzeige werden zwei LCD-Displays verwendet, die sich jeweils vor einem Auge befinden. Durch eine Linse lässt sich die Schärfe an das Auge anpassen. Abbildung 2.8 zeigt die Funktionsweise der Brille. Ihr Öffnungswinkel ist mit $32°$ sehr gering und die Auflösung von 640×480 ist für diese Art von Brillen durchschnittlich. Dies sorgt für eine gefühlte 115cm-Leinwand, die sich 2m vor dem Auge befindet. Die Anzeige der Bilddaten muss mit einer Projektionsmatrix auf den Öffnungswinkel der Brille angepasst werden.

2.3. Notation

Die *3D-Kamera* liefert eine Punktwolke, die im folgenden mit $\mathcal{C}$ (für cloud) bezeichnet wird, wobei $\mathcal{C} \in \mathbb{R}^{3\times N}$. Diese stellt eine Menge von N Punkten $\mathbf{c} \in \mathbb{R}^3$ dar. $\mathcal{C}$ sei so ausgerichtet, dass der Ursprung der Punktwolke im globalen Koordinatenursprung liegt und die Punktwolke sich in die positive z-Richtung erstreckt. Es wird ein rechtshändiges Koordinatensystem verwendet.

Der *Motion-Capture-Anzug* liefert

1. die Positionen von 23 Datenpunkten, beschrieben durch die Spaltenvektoren $\mathbf{x}_i^t \in \mathbb{R}^3$ und
2. die Orientierungen der Datenpunkte $\mathbf{R}_i^t \in \mathbb{R}^{3\times3}$

für $1 \leq i \leq 23$ zu einem Zeitpunkt t. Kombiniert ergibt sich die Transformationsmatrix

$$\mathbf{T}_i^t = \begin{pmatrix} & \mathbf{R}_i^t & & \mathbf{x}_i^t \\ 0 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times4}. \tag{2.5}$$

Für das bessere Verständnis werden die einzelnen Körperteile, die von Relevanz sind, durch entsprechende Indizes in Textform gekennzeichnet. Dies seien im speziellen:

- $i = 6$: Kopf, gekennzeichnet durch $i = \mathrm{Kopf}$
- $i = 10$: rechte Hand, gekennzeichnet durch $i = \mathrm{HandR}$
- $i = 14$: linke Hand, gekennzeichnet durch $i = \mathrm{HandL}$

Wenn der Zeitpunkt nicht berücksichtigt werden muss, wird der Index t weggelassen. Außerdem wird die Position des Balls durch $\mathbf{b}$ definiert.

Bei Vektoren bezeichnet die Schreibweise $\mathbf{v}_{i,a}$, wobei $a \in \{x, y, z\}$, den Zugriff auf das entsprechende Element $a \in \mathbb{R}$ des Vektors. Des weiteren stellen Skript-Großbuchstaben Mengen dar (zum Beispiel $\mathcal{M}$).

2.4. Zusammenfassung

Zu Beginn des Kapitels wurde definiert, was man unter Augmented-Reality versteht, wo die Einsatzgebiete liegen und welche Hardware für die Anzeige verwendet werden kann. Anschließend folgte ein Überblick über die Hardware, die in dieser Arbeit eingesetzt wird. Hierbei wurde erläutert, wie die 3D-Kamera funktioniert, wie der Anzug die Bewegungsdaten erfasst und verarbeitet und welche Brille eingesetzt wird.

3 Welterfassung

Dieses Kapitel beschreibt die Erfassung der Umgebung. Zunächst wird die 3D-Erfassung mittels Stereo-Kamera erläutert. Der zweite Teil der 3D-Erfassung beschäftigt sich mit der Transformation des Weltmodelles vor die virtuelle Kamera. Die 2D-Erfassung wird anschließend in Unterkapitel 3.2 vorgestellt.

Abbildung 3.1 zeigt die einzelnen Projekt-Komponenten. Die grünen Boxen zeigen dabei die Eingaben, die das Spiel bekommt. Dabei liefert die Bumblebee-Kamera die 3D-Punktwolke und das 2D-Bild. Der Anzug erfasst die Bewegungsdaten und der Ball wird durch das Spiel simuliert. Als Resultat erhält man die fertigen Bilder für die Anzeigegeräte und die neue Pose des Balles. Das Lernverfahren wird im nächsten Kapitel präsentiert.

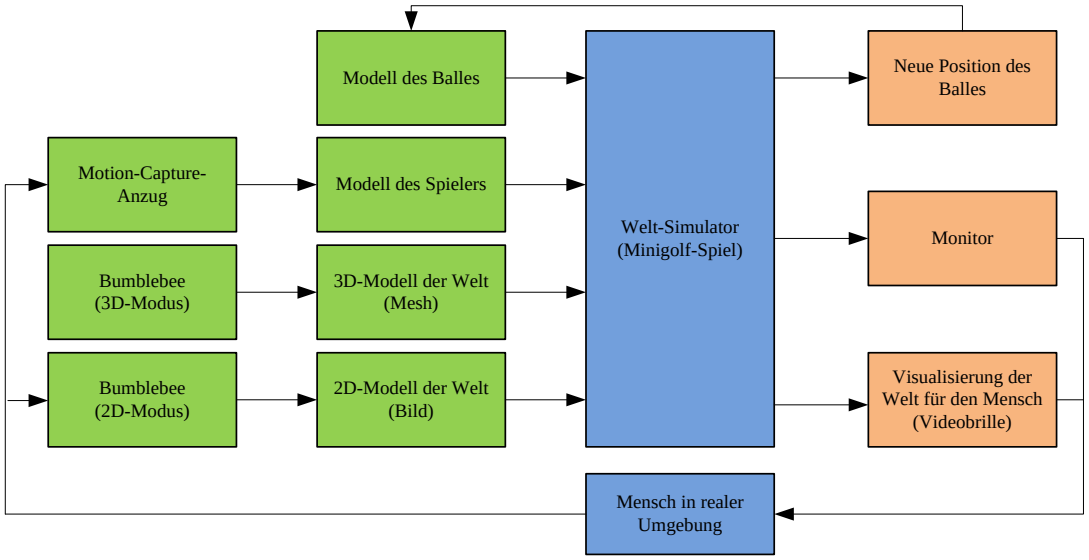

Abbildung 3.1.: Überblick über die einzelnen Komponenten

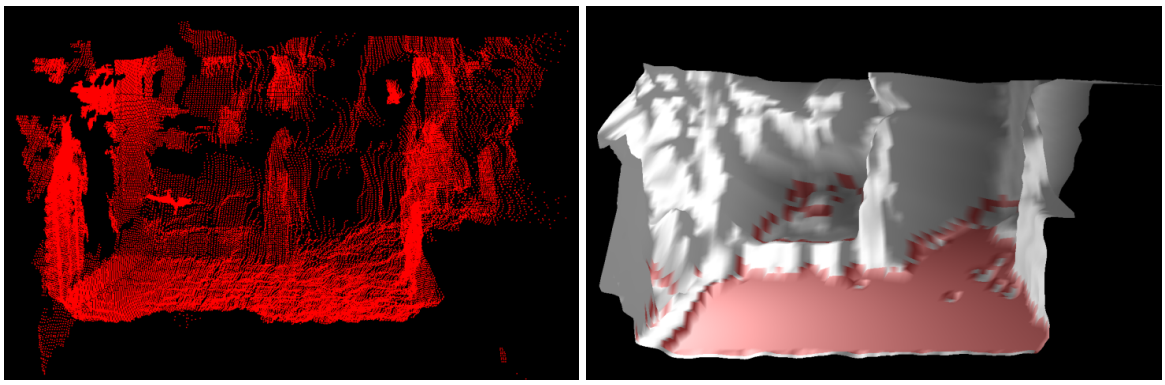

Abbildung 3.2.: Links: Punktwolke **Rechts:** Erzeugtes, geschlossenes Mesh. Der ermittelte Boden ist rot eingefärbt

3.1. 3D-Welterfassung

Die 3D-Erfassung der Umgebung gliedert sich in folgende Teilprobleme:

1. *Berechnen der Punktwolke aus den zwei Bildern der Bumblebee-Kamera:* Hierfür wird das Triclops-SDK [36] von Point-Grey verwendet.
2. *Verbinden der Punkte zu einem entsprechenden Mesh* (Unterkapitel 3.1.1)
3. *Ermitteln der Fußboden-Orientierung* (Unterkapitel 3.1.2)
4. *Transformation des Meshs vor die virtuelle Kamera* (Unterkapitel 3.1.3)

3.1.1. Robustes Meshing

Der Meshing-Algorithmus erzeugt aus einer Punktwolke ein Mesh. Hierfür wird ein Gitter aus Quadraten erzeugt, in welchem die Punkte aus der Punktwolke einem Quadrat zugeordnet werden. Um Punkte einer Gitterzelle zuzuordnen, wird eine Projektion der 3D-Punkte auf eine 2D-Ebene durchgeführt. Der Tiefenwert eines Quadrates ergibt sich dann durch die durchschnittlichen Tiefenwerte der zugeordneten Punkte. Abschließend wird das Mesh geschlossen, indem der dreidimensionale Mittelwert von benachbarten Eckpunkte gebildet wird. Diese Mittelwerte sind dann die neuen Eckpunktkoordinaten. Abbildung 3.2 zeigt ein Beispiel einer Punktwolke und das daraus gewonnene Mesh. Die Punktwolke ist verrauscht und beinhaltet Outlier. Durch Verwendung eines Grids werden diese Werte gefiltert.

Als Eingabe erhält der Meshing-Algorithmus eine Punktwolke $\mathcal{C} = \{\mathbf{c}_1, ..., \mathbf{c}_n\}, \mathbf{c}_i \in \mathbb{R}^3, n = |\mathcal{C}|, 1 \leq i \leq n$. Abbildung 3.3 skizziert den Algorithmus.

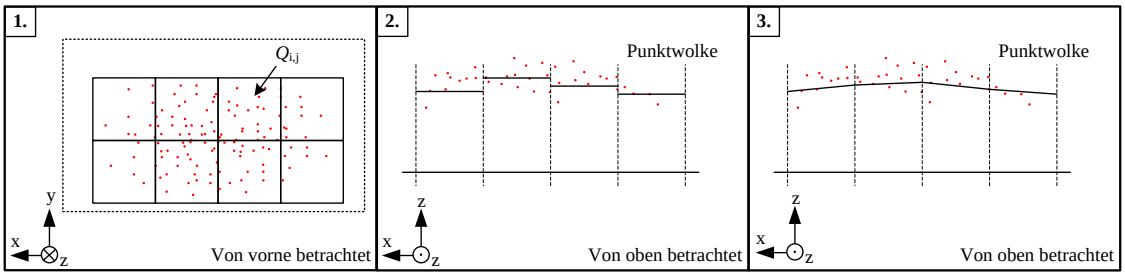

Abbildung 3.3.: In drei Schritten wird aus einer Punktwolke ein Mesh generiert. **Links:** Einteilung der Punkte in Zellen - **Mitte:** Mittlere Tiefe für jede Zelle berechnen - **Rechts:** Schließen des Meshs durch Bildung des Mittelwertes an den Eckpunkten der Zellen

Zunächst wird die Zuordnung der Punkte in die entsprechenden Quadrate erläutert. Dabei wird eine Projektion der $\mathbf{c}_n$ auf die Gitterebene $z = 1$ durchgeführt. Dieser Schritt macht die Zuordnung der Punkte einfacher, da das Problem auf zwei Dimensionen beschränkt wird.

Alle Punkte $\mathbf{c}_i \in \mathcal{C}$ werden auf diese Ebene projiziert. Der projizierte Punkt $\tilde{\mathbf{c}}_i$ ergibt sich somit durch

$$\tilde{\mathbf{c}}_i = \frac{\mathbf{c}_i}{\mathbf{c}_{i,z}}. \tag{3.1}$$

Die Seitenlänge q eines Quadrates sei eine vom Benutzer definierte Konstante. Vergrößert man q, dann beschleunigt sich die Berechnung des Meshs, da weniger Quadrate erzeugt werden müssen. Dadurch, dass der Einfluss eines Quadrates steigt, sinkt die Qualität des Meshs. Bei kleinerem q hat man mehr Details - die Berechnung des Meshs dauert aber länger. Mittels q kann nun bestimmt werden, in wie viele Quadrate $\mathcal{Q}_{i,j}$ die Ebene unterteilt werden soll. Dabei berechnet man das Indexpaar (i, j) durch folgende Regel (siehe Abbildung 3.3, links)

$$(i, j) = (i_{\mathbf{c}}, j_{\mathbf{c}}) = \left(\left\lfloor \frac{\tilde{\mathbf{c}}_{i,x}}{q} \right\rfloor, \left\lfloor \frac{\tilde{\mathbf{c}}_{i,y}}{q} \right\rfloor \right), \tag{3.2}$$

wobei $\lfloor \cdot \rfloor$ die Abrundung darstellt. Ein Quadrat wird dann durch die Menge

$$\mathcal{Q}_{i,j} = \{\mathbf{c} | (i_{\mathbf{c}}, j_{\mathbf{c}}) = (i, j), \mathbf{c} \in \mathcal{C}\} \tag{3.3}$$

beschrieben.

Nach der Zuordnung der Punkte werden die z-Werte der Quadrate berechnet. Für jedes $Q_{i,j}$ wird dieser Wert durch

$$d_{i,j} = \frac{1}{|\mathcal{Q}_{i,j}|} \sum_{\mathbf{c} \in \mathcal{Q}_{i,j}} \mathbf{c}_z \tag{3.4}$$

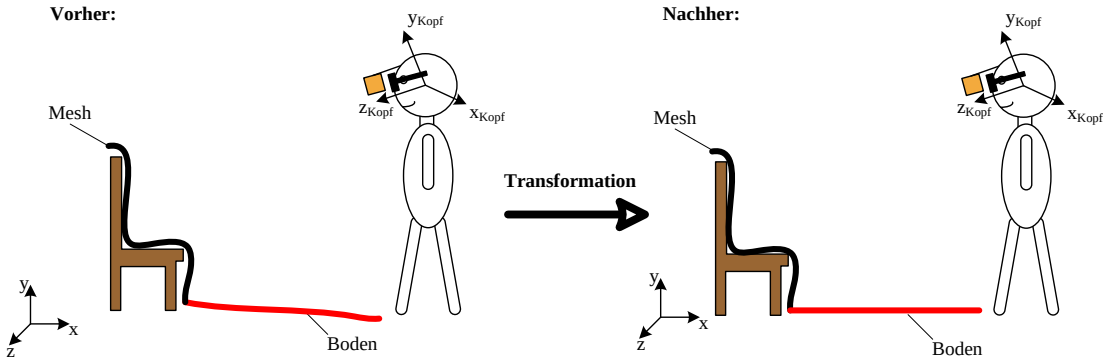

Abbildung 3.4.: Die zu ermittelnde Transformation soll den Boden waagrecht ausrichten

berechnet, wobei $\mathbf{c}_z$ den z-Wert von $\mathbf{c}_z$ angibt (siehe Abbildung 3.3, Mitte). Die Tiefe ergibt sich also aus der mittleren Tiefe der, dem Quadrat zugeordneten, Punkte. Jedem $\mathcal{Q}_{i,j}$ ordnet man die Tiefe $d_{i,j}$ zu, was zu folgender Koordinate $\mathbf{q}_{i,j}$ des Quadrates führt:

$$\mathbf{q}_{i,j} = (i \cdot q, j \cdot q, d_{i,j})^T \tag{3.5}$$

Die ermittelte Menge an Quadraten beschreibt nun zwar die Punktwolke, es ist allerdings nicht garantiert, dass das Mesh tatsächlich geschlossen ist. Hierfür betrachtet man die Eckpunkte der Quadrate. Bildet man den dreidimensionalen Mittelwert benachbarter Eckpunkte und setzt diese Werte dann entsprechend, so wird das Mesh geschlossen (siehe Abbildung 3.3, rechts). Dieses Mesh kann nun in der Physikengine zur Kollisionserkennung verwendet werden.

Ein Vorteil der Unterteilung in Quadrate besteht darin, dass einzelne Ausreißer der Punktwolke durch die Durchschnittsbildung nicht zu sehr ins Gewicht fallen. Ein Problem stellen allerdings die Quadrate dar, die keine Punkte beinhalten, da für diese keine Tiefe ermittelt werden kann. Der gegenwärtige Ansatz versucht aus angrenzenden Quadraten den Tiefenwert zu ermitteln. Ein besserer Lösungsansatz für die Schließung der Löcher besteht in der Verwendung von *Random Markov Fields*. Pérez et al. stellen in [34] ein derartiges Verfahren vor.

3.1.2. Robuste Erkennung des Bodens

Der Boden im Mesh ist im Allgemeinen, aufgrund von Ungenauigkeiten der Kamera und des Stereo-Algorithmus, nicht waagrecht. Ein waagrechter Boden ist aber nötig, da der Ball sonst nicht kontrollierbar wäre. Für die Drehung wird der Boden und dessen Orientierung geschätzt. Mit Hilfe der Orientierung kann das Mesh so gedreht werden, dass der Boden waagrecht wird. Abbildung 3.4 stellt die Problematik dar.

Es wird ein Verfahren benötigt, welches in einer Menge von Punkten $\mathcal{D}$ nach der Hauptebene sucht und deren Orientierung ermittelt. Die Ebene wird durch ein Modell M beschrieben. $\theta \in \mathbb{R}^4$ sei der Modellparametervektor. Das Verfahren soll also

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(P \left(\mathcal{D} \mid M, \theta \right) \right) \tag{3.6}$$

ermitteln, wobei $\hat{\theta}$ die Parameter der gesuchten Hauptebene beinhaltet. Hierfür wird der RANSAC-Algorithmus verwendet, der in Anhang A.4 beschrieben wird.

$\mathcal{D}$ sei hier die Menge aller Quadratkoordinaten $\mathbf{q}_{i,j}$. Außerdem sei

$$\mathcal{D}_{\text{minimal}} = \{\mathbf{d}_i, \mathbf{d}_j, \mathbf{d}_k\},\ i, j, k \in [1, ..., |\mathcal{D}|]. \tag{3.7}$$

Es werden also drei Punkte aus $\mathcal{D}$ ausgewählt, da diese für die Beschreibung einer Ebene im dreidimensionalen Raum ausreichend sind. θ kann aus diesen drei Punkten geschätzt werden.

Das Modell M einer Ebene basiert auf einer Normalen, die senkrecht auf der Ebene steht. Seien hierzu

$$\mathbf{r}_x = \mathbf{d}_k - \mathbf{d}_i \text{ und } \mathbf{r}_z = \mathbf{d}_j - \mathbf{d}_i, \tag{3.8}$$

zwei Richtungsvektoren, die für die Bestimmung der Normalen verwendet wird. M ist durch die Ebenengleichung

$$\langle (\theta_1, \theta_2, \theta_3), (x, y, z) \rangle + \theta_4 = 0 \tag{3.9}$$

gegeben, wobei $\langle \cdot, \cdot \rangle$ das Skalarprodukt zweier Vektoren in $\mathbb{R}^3$ darstellt. Hierbei bestimmt der Vektor

$$(\theta_1, \theta_2, \theta_3) = \mathbf{r}_x \times \mathbf{r}_z = \mathbf{r}_y \tag{3.10}$$

die Normale der Ebene und

$$\theta_4 = - \langle (\theta_1, \theta_2, \theta_3), \mathbf{d}_i \rangle ; \tag{3.11}$$

den Abstand zum Ursprung. Abbildung 3.5 zeigt, wie die Ebene definiert wird.

Die Fehlerfunktion

$$e_M(d, \theta) = \langle (\theta_1, \theta_2, \theta_3), \mathbf{d} \rangle + \theta_4 \tag{3.12}$$

bestimmt die Distanz zwischen Ebene und einem Punkt $\mathbf{d}$. Somit lässt sich die Menge der

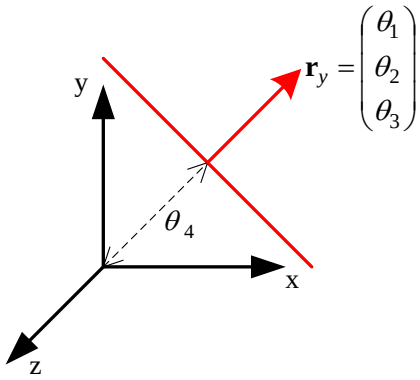

Abbildung 3.5.: Das verwendete Modell einer Ebene: Diese wird durch eine Normale $\mathbf{r}_y$ und den Abstand θ_4 vom Ursprung definiert

Inlier folgendermaßen berechnen:

$$I(\theta) = \{\mathbf{d} \in \mathcal{D} : e_M(d, \theta) \leq \epsilon\} \tag{3.13}$$

$I(\theta)$ beinhaltet nun alle Punkte $\mathbf{d} \in \mathcal{D}$, die in der durch M und θ definierten Ebene liegen. Durch mehrfache Auswahl erhält man die Ebene, für die die Kardinalität von $I(\theta)$ maximal wird.

3.1.3. Transformationen

Das zuvor gewonnene Ebenenmodell wird nun dazu verwendet, das Mesh so vor die virtuelle Kamera zu transformieren, dass der Boden waagrecht bleibt. Die Transformation des Meshs vor die Kamera erfolgt somit durch die drei folgenden Transformationen:

1. *Transformation des Weltkoordinatensystems in das des Kopfes*
2. *Transformation des Kopfkoordinatensystems in das der Kamera*
3. *Transformation des Kamerakoordinatensystems in das des Meshs*

Abbildung 3.6 stellt die einzelnen Transformationen visuell dar.

Die **Transformation des Weltkoordinatensystems in das des Kopfes** liefert der Motion-Capture-Anzug in Form der Transformation des Kopfes. Diese sei durch

$$\mathbf{T}_{\text{Welt}\to\text{Kopf}} \tag{3.14}$$

definiert.

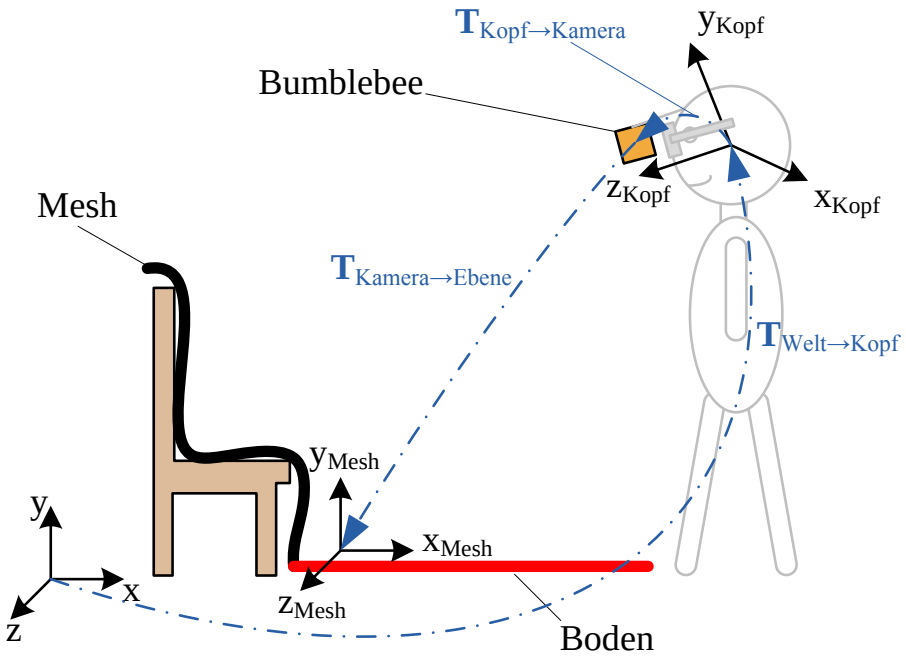

Abbildung 3.6.: Die einzelnen Schritte der Transformation des Meshs vor den Spieler - Das Koordinatensystem der Bumbleebee-Kamera wurde aus Platzgründen weggelassen

Des weiteren ist die zweite **Transformation des Kopfkoordinatensystems in das der Kamera** eine Matrix, die sich nicht automatisch bestimmen lässt. Diese muss von Hand ermittelt werden, da diese Transformation im Wesentlichen den Abstand zwischen Kopfsensor des Anzuges und der Kamera angibt. Sei diese Transformation durch

$$\mathbf{T}_{\text{Kopf}\rightarrow\text{Kamera}} \tag{3.15}$$

gegeben.

Die **Transformation des Kamerakoordinatensystems in das des Meshs** muss aus dem ermittelten Modell der Ebene M (siehe vorhergehendes Unterkapitel 3.1.2) bestimmt werden. Hierzu werden die drei Vektoren $\mathbf{r}_x$, $\mathbf{r}_y$ und $\mathbf{r}_z$ verwendet, die zur Bestimmung von $\hat{\theta}$ benutzt wurden. Aus diesen wird eine Rotationsmatrix [16, S. 203ff] ermittelt, die das Mesh so dreht, dass der Boden waagrecht wird. Zunächst werden $\mathbf{r}_x$, $\mathbf{r}_y$ und $\mathbf{r}_z$ orthogonalisiert.

Wie definiert, erstreckt sich die Punktwolke zu Beginn in die positive z-Richtung. Der Vektor $\mathbf{e}_x = (1,0,0)^T$ dient somit als x-Achse. Die drei Vektoren $\mathbf{r}_x$, $\mathbf{r}_y$ und $\mathbf{r}_z$ sind die Basisvektoren der Rotation und müssen orthonormal sein, wodurch $\mathbf{e}_x$ entsprechend angepasst werden muss. Hierfür kann das Gram-Schmidtsche Orthonormalisierungsverfahren [22, S. 164] ver-

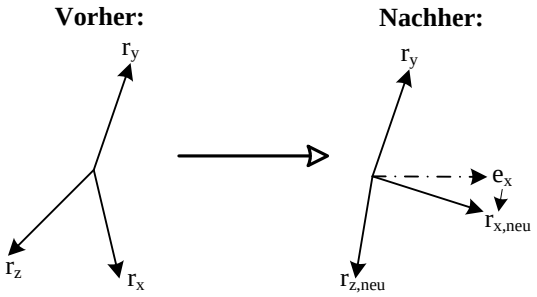

Abbildung 3.7.: Anpassung des Koordinatensystems der Hauptebene mit Hilfe des Gram-Schmidtschen Orthonormalisierungsverfahren

wendet werden (Abbildung 3.7):

$$\mathbf{w} = \mathbf{e}_x - \langle \mathbf{r}_y, \mathbf{e}_x \rangle \, \mathbf{r}_y \tag{3.16}$$

$$\mathbf{r}_{x,\text{neu}} = \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \tag{3.17}$$

Der Vektor $\mathbf{r}_{x,\text{neu}}$ ist nach diesem Schritt orthonormal zu $\mathbf{r}_y$ und beinhaltet keine Drehung um die y-Achse. $\mathbf{r}_{z,\text{neu}}$ kann mittels Kreuzprodukt aus $\mathbf{r}_x$ und $\mathbf{r}_y$ bestimmt werden.

Die Transformationsmatrix hat also folgende Form:

$$\mathbf{T}_{\text{Kamera}\to\text{Mesh}} = \begin{pmatrix} \mathbf{r}_{x,\text{neu}} & \mathbf{r}_y & \mathbf{r}_{z,\text{neu}} & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.18}$$

$\mathbf{0}$ bezeichnet den Vektor $(0,0,0)^T$.

Die drei Transformationsmatrizen werden nun hintereinander angewandt. Die **gesamte Transformation** ergibt sich also aus

$$\mathbf{T}_1 = \mathbf{T}_{\text{Welt}\to\text{Kopf}} \cdot \mathbf{T}_{\text{Kopf}\to\text{Kamera}} \cdot \mathbf{T}_{\text{Kamera}\to\text{Mesh}}. \tag{3.19}$$

Die Matrix $\mathbf{T_1}$ muss um eine weitere Transformation erweitert werden, da das Mesh zunächst waagrecht gedreht wird und dann durch die Transformation vor die Kamera wieder verdreht wird. Die Hauptebene ist somit im Allgemeinen nicht mehr waagrecht. Diese Transformation muss also dafür sorgen, dass der Boden, nach allen Transformationen, waagrecht bleibt. Für diese konnte keine geschlossene Form gefunden werden, weshalb zwei Rotationsmatrizen verwendet werden, wie im folgenden Abschnitt erläutert wird.

Für die Bestimmung der **fehlenden Transformation** benötigt man also eine Fehlerfunktion, die minimiert werden kann. Als Bedingung wird hier die Ausrichtung des Bodens herange-

zogen, der nach allen Transformationen waagrecht sein muss. Das bedeutet, dass der Normalenvektor des Bodens nach der Berechnung $(0,1,0)^T$ sein soll. Deshalb müssen noch zwei Rotationen um die x- und z-Achse berücksichtigt werden, die dieses Problem beheben.

Die Fehlerfunktion

$$f(\alpha,\beta) = \left\langle \left(\mathbf{T}_{\text{Welt}\to\text{Kopf}} \cdot \mathbf{T}_{\mathbf{x}}(\alpha) \cdot \mathbf{T}_{\mathbf{z}}(\beta) \cdot \mathbf{T}_{\text{Kopf}\to\text{Kamera}} \cdot \mathbf{T}_{\text{Kamera}\to\text{Mesh}} \right)_Y , (0,1,0)^T \right\rangle - 1 \tag{3.20}$$

wird dazu verwendet, um diese zwei fehlenden Rotationen zu ermitteln. Dabei steht $(\cdot)_Y$ für den zweiten Spaltenvektor des Rotationsteils der Matrix. Die Matrizen $\mathbf{T}_x(\alpha)$ und $\mathbf{T}_z(\beta)$ sind reine Rotationsmatrizen [16, S. 203ff], die jeweils eine Rotation um die x-, beziehungsweise z-Achse mit den Winkeln α, respektive β beschreiben.

Die Winkel $\tilde{\alpha}$ und $\tilde{\beta}$ ergeben sich dann durch:

$$(\tilde{\alpha},\tilde{\beta}) = \underset{\alpha,\beta}{\operatorname{argmin}} \left(f(\alpha,\beta) \right) \tag{3.21}$$

Die **vollständige Matrix**

$$\mathbf{T} = \mathbf{T}_{\text{Welt}\to\text{Kopf}} \cdot \mathbf{T}_x(\tilde{\alpha}) \cdot \mathbf{T}_z(\tilde{\beta})$$

beschreibt nun die richtige Transformation des Meshs.

Abschließend muss der Boden noch „planarisiert“ werden, damit der Ball nicht unkontrolliert herum rollt. Dies geschieht dadurch, dass die y-Werte des Bodens auf null gesetzt werden.

3.2. 2D-Welterfassung

Die 2D-Bilderfassung erfolgt ebenfalls durch die Bumblebee-Kamera. Hierbei wird nur das Bild der linken Kamera verwendet, welches dann auf eine planare Leinwand projiziert wird, die sich im Hintergrund der virtuellen Objekte befindet. Somit wird die reale Umgebung in der Videobrille sichtbar.

Es muss beachtet werden, dass das Bild, welches von der Kamera aufgenommen wird, aufgrund des Öffnungswinkel, stark verzerrt ist. Es muss also eine Rektifizierung durchgeführt werden, die die Verzerrungen entfernt. Danach wird mit der spezifischen Kameramatrix eine Projektion des Bildes auf eine Leinwand ermittelt.

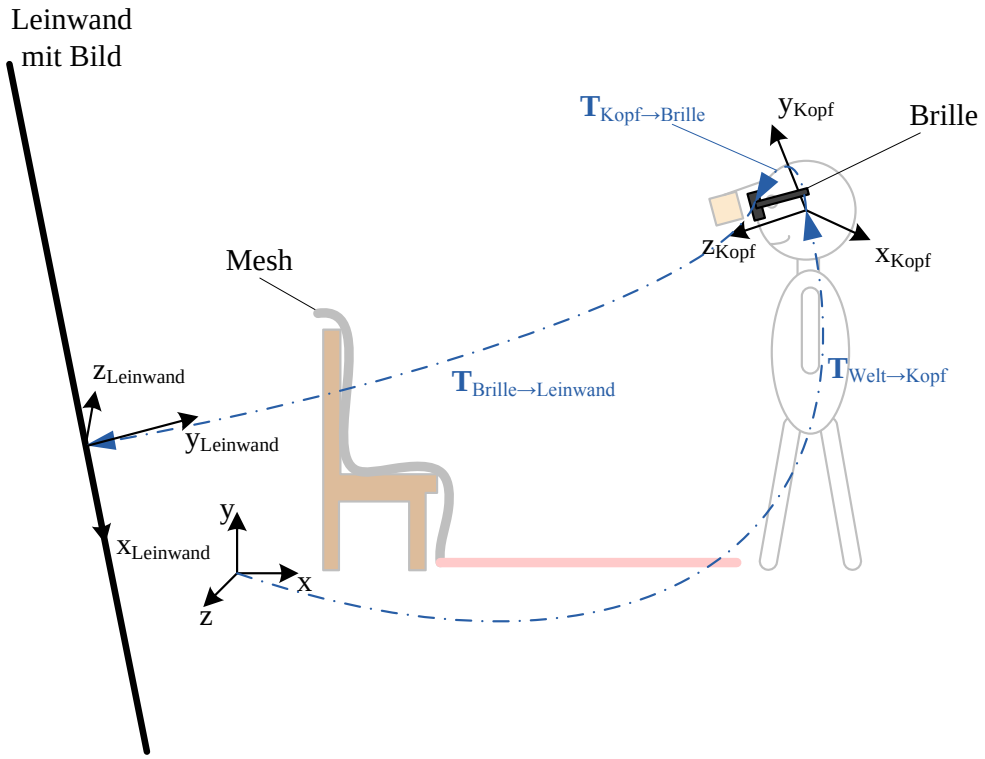

Abbildung 3.8.: Die einzelnen Schritte der Transformation der Leinwand vor den Spieler - Das Koordinatensystem der Videobrille wurde aus Platzgründen weggelassen

3.2.1. Transformation der Leinwand vor die virtuelle Kamera

Die Projektion des Bildes vor die virtuelle Kamera setzt sich aus folgenden Transformationen zusammen:

1. *Transformation des Weltkoordinatensystems in das des Kopfes:* $\mathbf{T}_{\text{Welt}\to\text{Kopf}}$ (ermittelt in Kapitel 3.1.3)
2. *Transformation des Kopfkoordinatensystems in das der Brille*
3. *Transformation des Brillenkoordinatensystems in das der Leinwand*

Abbildung 3.8 stellt die einzelnen Transformationen visuell dar.

Die **Transformation des Kopfkoordinatensystems in das der Brille**

$$\mathbf{T}_{\text{Kopf}\to\text{Brille}} \tag{3.22}$$

ist eine Matrix, die manuell ermittelt werden muss. Sie beschreibt im Wesentlichen die Translation zwischen Kopfsensor des Anzuges und der Brille.

Des weiteren beschreibt die **Transformation des Brillenkoordinatensystems in das der Leinwand**

$$\mathbf{T}_{\text{Brille}\rightarrow\text{Leinwand}} \tag{3.23}$$

den Abstand zwischen Brille und Leinwand. Dieser sollte groß genug gewählt werden, so dass die Leinwand hinter den virtuellen Objekten angezeigt wird.

Die Matrix

$$\mathbf{T}_{\text{Leinwand}} = \mathbf{T}_{\text{Welt}\rightarrow\text{Kopf}} \cdot \mathbf{T}_{\text{Kopf}\rightarrow\text{Brille}} \cdot \mathbf{T}_{\text{Brille}\rightarrow\text{Leinwand}} \tag{3.24}$$

gibt somit die gesamte Transformation der Leinwand vor die Kamera an.

3.2.2. Rückprojektion auf eine virtuelle Leinwand

Die Rückprojektion sorgt für die korrekte Abbildung des Kamerabildes auf eine Leinwand, ohne die ein zu großer Bildbereich dargestellt werden würde, da der Öffnungswinkel der Kamera größer ist, als der der Brille. Des weiteren wird das Bild in diesem Schritt rektifiziert.

Zunächst wird eine planare Leinwand in Form einer Gridfläche erstellt. Die Punkte des Grids seien durch $\mathbf{g}_{i,j} \in \mathbb{R}^3$ (Abbildung 3.9, rechts) definiert. Für jedes $\mathbf{g}_{i,j}$ wird die Bildkoordinate (Texturkoordinate) ermittelt, die dann durch die Triclops-Pixel-Rektifizierung entsprechend angepasst und für die Anzeige auf der Leinwand verwendet werden kann. Die Anzahl der verwendeten Zellen beeinflusst, an wie vielen Stellen die Texturkoordinaten rektifiziert werden.

Die intrinsische Kameramatrix [17, S. 11-14]

$$\mathbf{M}_{\text{intrinsisch}} = \begin{pmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.25}$$

berücksichtigt bei der Projektion die kameraspezifischen Daten. u_0 und v_0 geben das Zentrum des Bildes an und γ stellt ein Scherungsfaktor dar, der für die meisten Kameras Null ist. α_u und β_v ergeben sich aus

$$\alpha_v = \frac{f}{w} \text{ und } \beta_v = \frac{f}{h}, \tag{3.26}$$

wobei w und h die Breite und Höhe eines Pixels und f die Brennweite der Kamera ist. Abbildung 3.9 (links) zeigt eine Skizze der wichtigsten Parameter.

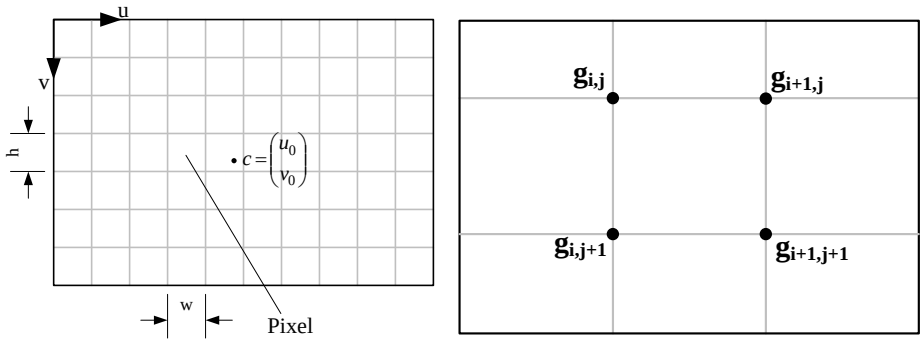

Abbildung 3.9.: Links: Skizzierung der wichtigsten Parameter der Projektion im Bild - **Rechts:** Die Leinwand wird in Rechtecke R_j eingeteilt, die jeweils vier Eckpunkte $\mathbf{r}_{j,k}$ haben.

Jeder Punkt $\mathbf{g}_{i,j}$ wird nun mit der obigen Matrix multipliziert, was zu den korrekten Texturkoordinaten führt. Somit erhält man

$$\mathbf{t}_{j,k} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{g}_{i,j}, \tag{3.27}$$

wobei $\mathbf{t}_{j,k} \in \mathbb{R}^3$ die homogene Texturkoordinate angibt. Es muss somit beachtet werden, dass u und v noch durch w geteilt wird, bevor man sie als Texturkoordinate verwenden kann.

Die Rechtecke müssen abschließend vor die Kamera transformiert werden. Hierfür wird die in Kapitel 3.2.1 beschriebene Transformation $\mathbf{T}_{\text{Leinwand}}$ verwendet:

$$\tilde{\mathbf{g}}_{j,k} = \mathbf{T}_{\text{Leinwand}} \cdot \mathbf{g}_{i,j}. \tag{3.28}$$

3.3. Zusammenfassung

Dieses Kapitel diente dazu, die Welterfassung zu erläutern. Zunächst wurde gezeigt, wie aus der Punktwolke ein geschlossenes Mesh erzeugt wird. Anschließend folgte die Schätzung des Fußbodens mittels RANSAC-Algorithmus. Die Orientierung des Bodens wurde dann zusammen mit weiteren Transformationen genutzt, um das Mesh korrekt vor der Kamera platzieren zu können. Um die Kollision des Balles mit der Umgebung zuzulassen, wird eine Physikengine verwendet.

4 Lernverfahren

Das Programm ermittelt die Länge und Orientierung des Minigolf-Schlägers. Dadurch soll sich der Schläger an den Spieler anpassen. Dies geschieht durch Demonstration mehrerer Schlagbewegungen mit sichtbarem Ball aber ohne Schläger. Aus diesen Daten werden dann die nötigen Informationen gewonnen. Außerdem wird ermittelt, mit welcher Hand gespielt wird.

Die Schlägerlänge richtet sich nach dem minimalen Abstand zwischen Ball und Hand während eines Schlages. Hierfür muss also ein Sequenz d^t ermittelt werden, die den Verlauf der Distanz zwischen Hand und Ball angibt. Da die Daten verrauscht sind, muss zunächst ein Gauß-Smoother angewandt werden, der die Werte glättet. In dieser Sequenz werden anschließend die lokalen Minimalstellen gesucht. Alle Minimalstellen werden gemittelt und ergeben die Schlägerlänge. Die Orientierung wird an den entsprechenden Minimalstellen ebenfalls gemittelt und anschließend zur Bestimmung der Schlägerorientierung verwendet. Die Durchschnittsrotation wird mittels Singulärwertzerlegung und Löwdin-Orthogonalisierung, die in Anhang A beschrieben sind, bestimmt.

4.1. Ermitteln der Länge und Orientierung

Wie bereits in der Einführung erläutert, wird jeweils eine Distanz-Sequenz zwischen Ball und Hand bestimmt, die dann für die Schätzung der Aktionsparameter verwendet wird. Für die Werte der linken Hand sei dies die Sequenz d^t_{links} - rechts entsprechend d^t_{rechts}. Außerdem werden die Transformationen der Hände durch die Sequenzen m^t_{links} und m^t_{lechts} beschrieben. Die Distanz-Sequenzen ergeben sich dann folgendermaßen:

$$d^t_{\text{links}} = \mathbf{b} - \mathbf{x}^t_{\text{HandL}} \tag{4.1}$$

$$d^t_{\text{rechts}} = \mathbf{b} - \mathbf{x}^t_{\text{HandR}}, \tag{4.2}$$

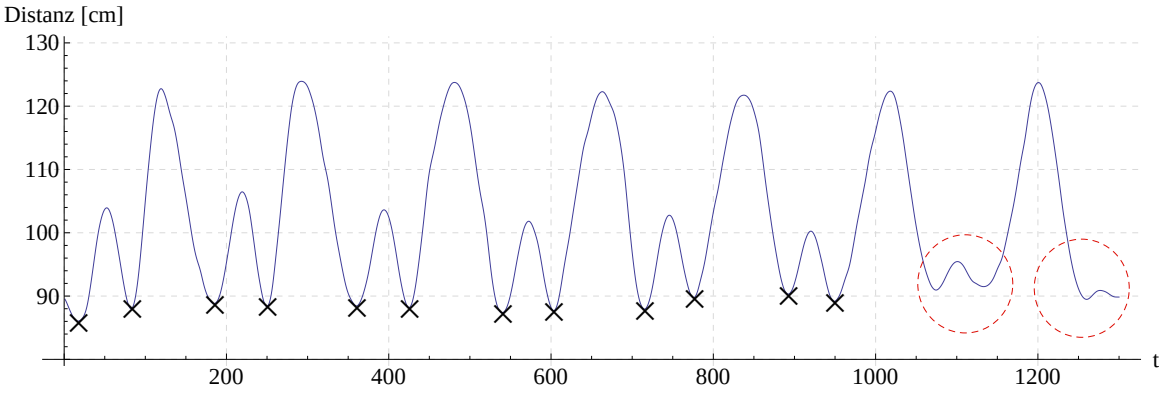

Abbildung 4.1.: Distanz-Sequenz der rechten Hand (d^t_{rechts}): Die rot eingekreisten Minimalstellen wurden aufgrund des Thresholds ausgelassen. Die verwendeten Minimalstellen sind durch die Kreuze gekennzeichnet. Es werden immer zwei Minimalstellen erfasst, da das Verfahren nicht unterscheidet, in welcher Richtung der Ball geschlagen wird.

wobei $\mathbf{b}$ die Position des Balles, $0 \leq t \leq T$ und T die Dauer der Lernphase ist. Entsprechend seien die Sequenzen der Transformationen durch

$$m^t_{\text{links}} = \mathbf{T}^t_{\text{HandL}} \text{ und} \tag{4.3}$$

$$m^t_{\text{rechts}} = \mathbf{T}^t_{\text{HandR}} \tag{4.4}$$

bestimmt.

Zunächst werden die Varianzen der beiden Distanz-Sequenzen ermittelt. Über die Varianz wird geschätzt, welche Hand der Spieler verwendet. Das Verfahren entscheidet sich für die Hand, die die größere Varianz aufweist. Die Sequenz der spielenden Hand sollte aufgrund der Bewegung eine größere Varianz aufweisen, als die andere Hand. Sei im folgenden die Sequenz d^t diejenige, deren Varianz $\text{Var}\left(d^t_{\text{links}}\right)$, beziehungsweise $\text{Var}\left(d^t_{\text{rechts}}\right)$ größer ist. Entsprechend wird die Transformations-Sequenz ausgewählt. Wenn sich das Verfahren für die linke Hand entscheidet, sei $m^t = m^t_{\text{links}}$, ansonsten gilt $m^t = m^t_{\text{rechts}}$.

Die Distanz-Sequenz d^t wird mittels Gauß-Smoother geglättet. Dieser Schritt ist für die anschließende Extremwertbestimmung notwendig, da diese sonst, aufgrund des Rauschens, nicht richtig funktionieren würde. Die Minimalstellen sind hierbei definiert durch die Menge

$$\mathcal{D}_{\text{min}} = \{t \,|\, d^{t-1} > d^t < d^{t+1}\}. \tag{4.5}$$

Entsprechend werden die Maximalstellen bestimmt:

$$\mathcal{D}_{\text{max}} = \{t \,|\, d^{t-1} < d^t > d^{t+1}\}. \tag{4.6}$$

Die Anzahl der Extremwerte ist ein Maß für die durchgeführten Schläge. Zeichnet man die Werte der Mengen $\mathcal{D}_{\min}$ und $\mathcal{D}_{\max}$ und der Sequenz d^t als Funktion der Zeit in ein Diagramm, so erwartet man eine sinusartige Funktion, wenn der Spieler wiederholt gleichmäßig schlägt. Allerdings können aufeinanderfolgende Extremstellen entstehen, deren y-Werte zu nahe beieinander liegen (siehe Abbildung 4.1). Diese Extremstellen treten beispielsweise dann auf, wenn der Spieler unruhig schlägt. Da diese Stellen das Ergebnis negativ beeinflussen können, werden diese ignoriert. Hierfür wird ein Threshold eingeführt, der dafür sorgt, dass der y-Abstand zweier aufeinanderfolgender Extremstellen genügend groß bleibt, wodurch diese Extremstellen nicht mehr betrachtet werden. Dieser Schritt entfernt aus der Menge $\mathcal{D}_{\min}$ diese Minimalstellen.

Die Bestimmung der Schlägerlänge l erfolgt abschließend durch Bildung der durchschnittlichen Distanz an den Minimalstellen:

$$l = \frac{1}{|\mathcal{D}_{\min}|} \sum_{t \in \mathcal{D}_{\min}} d^t \tag{4.7}$$

Nach der Bestimmung der Länge, erfolgt die Berechnung der *Orientierung* des Schlägers. Dieser Schritt sorgt dafür, dass der Schläger so orientiert wird, dass der Spieler den Ball optimal treffen kann. Für die Berechnung der Orientierung wird die Durchschnittstransformation des aktiven Handgelenkes bestimmt. Dabei werden die bei minimalem Hand-Ball-Abstand eingenommenen Transformationen berücksichtigt. Diese Transformation wird dann optimiert, so dass der Schläger im Falle eines Minimums den Ball trifft.

Die Bestimmung der Durchschnittstransformation erfolgt durch

$$\mathbf{T}_{\varnothing} = \frac{1}{|\mathcal{D}_{\min}|} \sum_{t \in \mathcal{D}_{\min}} m^t. \tag{4.8}$$

Sei $\mathbf{T}_{\varnothing,\mathrm{rot}} \in \mathbb{R}^{3\times3}$ der Rotationsteil der Matrix $\mathbf{T}_{\varnothing}$ und $\mathbf{t}_{\varnothing} \in \mathbb{R}^3$ der Translationsvektor. Die Matrix $\mathbf{T}_{\varnothing,\mathrm{rot}}$ ist nach der Mittelwertbildung nicht mehr orthogonal und muss deshalb mittels Löwdin-Orthogonalisierung wieder in eine reine Rotationsmatrix umgeformt werden:

$$\mathbf{R}_{\varnothing} = \mathrm{Löwdin}(\mathbf{T}_{\varnothing,\mathrm{rot}}) \tag{4.9}$$

Hierbei bezeichnet $\mathrm{Löwdin}(\cdot)$ die Löwdin-Orthogonalisierung, die deshalb eingesetzt wird, da sie die Orthogonalisierung symmetrisch auf alle drei Achsen verteilt. Die Durchschnittstransformation ergibt sich also aus $\mathbf{R}_{\varnothing}$ und $\mathbf{t}_{\varnothing}$:

$$\tilde{\mathbf{T}}_{\varnothing} = \begin{pmatrix} & \mathbf{R}_{\varnothing} & & \mathbf{t}_{\varnothing} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.10}$$

Nachdem die durchschnittliche Transformation der Hand ermittelt wurde, muss der Schläger noch so gedreht werden, dass der Abstand zwischen Ball und Schlägerspitze zu den Zeitpunkten $t \in \mathcal{D}_{\min}$ minimal wird. Hierfür werden zwei Rotationsmatrizen eingesetzt. Es müssen also zwei Winkel $\tilde{\alpha}$ und $\tilde{\beta}$ bestimmt werden:

$$(\tilde{\alpha}, \tilde{\beta}) = \underset{\alpha,\beta}{\operatorname{argmin}} \left(\tilde{\mathbf{T}}_{\varnothing} \mathbf{T}_x(\alpha) \mathbf{T}_z(\beta) \mathbf{s} \right) \tag{4.11}$$

Die Matrizen $\mathbf{T}_x(\alpha)$ und $\mathbf{T}_z(\beta)$ stellen reine Rotationsmatrizen dar, die jeweils um den angegebenen Betrag um die x-, beziehungsweise z-Achse rotieren. $\mathbf{s}$ bezeichnet die Position der Schlägerspitze, die durch die ermittelte Länge gegeben ist. Als Transformation des Schlägers erhält man dann, je nach Hand

$$\mathbf{T}_{\text{Schläger}} = \mathbf{T}^t_{\text{HandL}} \mathbf{T}_x(\tilde{\alpha}) \mathbf{T}_z(\tilde{\beta}) \mathbf{s}, \tag{4.12}$$

beziehungsweise

$$\mathbf{T}_{\text{Schläger}} = \mathbf{T}^t_{\text{HandR}} \mathbf{T}_x(\tilde{\alpha}) \mathbf{T}_z(\tilde{\beta}) \mathbf{s}. \tag{4.13}$$

4.2. Zusammenfassung

Dieses Kapitel erläuterte, wie die Länge und Orientierung des Schlägers bestimmt wird. Außerdem wurde ein einfaches Verfahren gezeigt, dass die aktive über die Varianz schätzt.

5 Implementierung

Als Grundlage für die Implementierung wurde ein eigenes Framework erstellt, dass alle nötigen Aufgaben übernimmt. Es bietet einen OpenGL-Viewer, die Anbindungen des Motion-Capture-Anzugs, der Bumblebee-Kamera und der Videobrille, diverse Fileloader (Punktwolken, Bewegungsdaten) und Steuerungselemente für die Benutzung der einzelnen Komponenten. Abbildung 5.1 zeigt die Oberfläche des Programms. Als Basis wurde wxWidgets eingesetzt, wodurch eine einfache Implementierung der Oberfläche möglich war. Außerdem bietet wxWidgets einfache Wege, um beispielsweise Daten über das Netzwerk zu empfangen, die Anzahl der angeschlossenen Monitore zu bestimmen (für die Video-Brille) oder Threads einzusetzen.

Wie bereits erwähnt, wird für die Erfassung der Bewegung ein *MVN-Anzug* von Xsens eingesetzt. Hinzu kommt die 3D-Kamera *Bumblebee2* von Point-Grey und die Videobrille *cinemizer plus* von Zeiss. Die Bumblebee-Kamera wurde dabei so an einem Fahrradhelm befestigt, dass sie sich etwa 5cm vor den Augen befindet. Das komplette System ist in Abbildung 5.2 zu sehen. Ein Problem an dieser Konfiguration ist das hohe Gewicht der Kamera und des Helmes. Für kurze Einsätze lässt sich dieses System trotzdem verwenden. Nach der Anzugskalibrierung, die durch das Xsens-Programm durchgeführt wird, werden die Bewegungsdaten über Ethernet an das Spiel übermittelt. Verarbeitet werden die Daten von zwei Notebooks. Das erste Notebook, auf dem Windows XP läuft, erhält die Daten vom Anzug und wertet diese entsprechend aus (MVN-Studio). Auf dem zweiten Notebook läuft Ubuntu und das Spiel.

Die Kamera wird über die Firewire-Bibliothek „libdc1394“ angesprochen. Die übermittelten Bilder werden mittels Triclops-SDK [36] in eine 3D-Punktwolke umgerechnet. Das Programm erstellt anschließend ein geschlossenes Mesh, welches für die Kollisionserkennung verwendet werdet wird. Neben dem 3D-Modus wird die Kamera auch für den 2D-Betrieb eingesetzt. Das Bild der linken Kamera wird dabei rektifiziert (mittels Triclops) und anschließend auf eine Fläche projiziert, die vor der Kamera ausgerichtet ist.

Abbildung 5.3 zeigt eine Skizze des Gesamtsystems. Die Laufzeit des Systems beträgt circa zwei Stunden, wenn die Notebooks im Akkumodus laufen. Werden diese mit Strom versorgt,

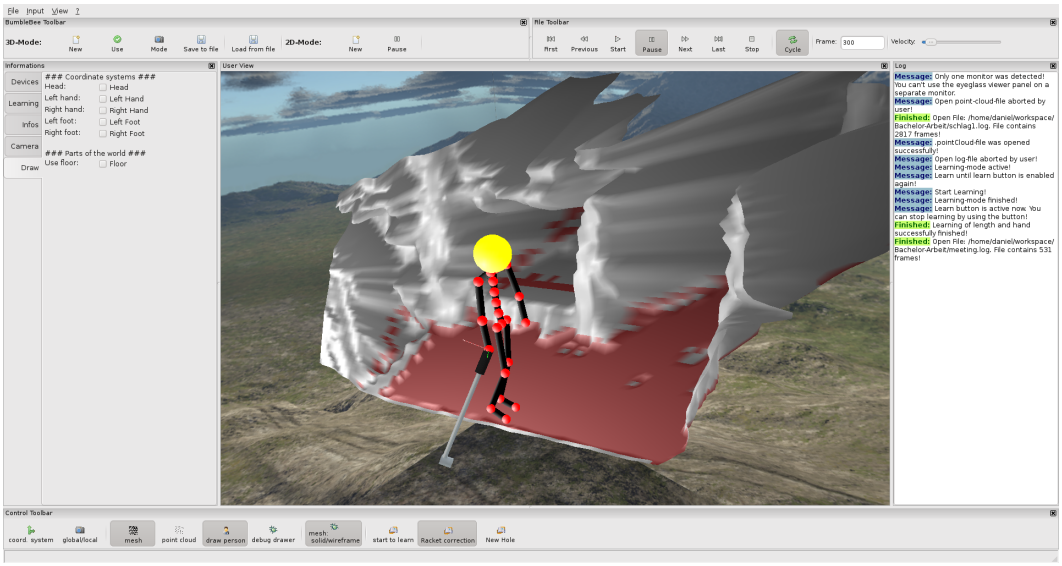

Abbildung 5.1.: Die Benutzeroberfläche der Implementierung mit Mesh und Spieler

Abbildung 5.2.: Links: Der gesamte Anzug mit Brille und Kamera **Rechts:** Helm mit Kamera

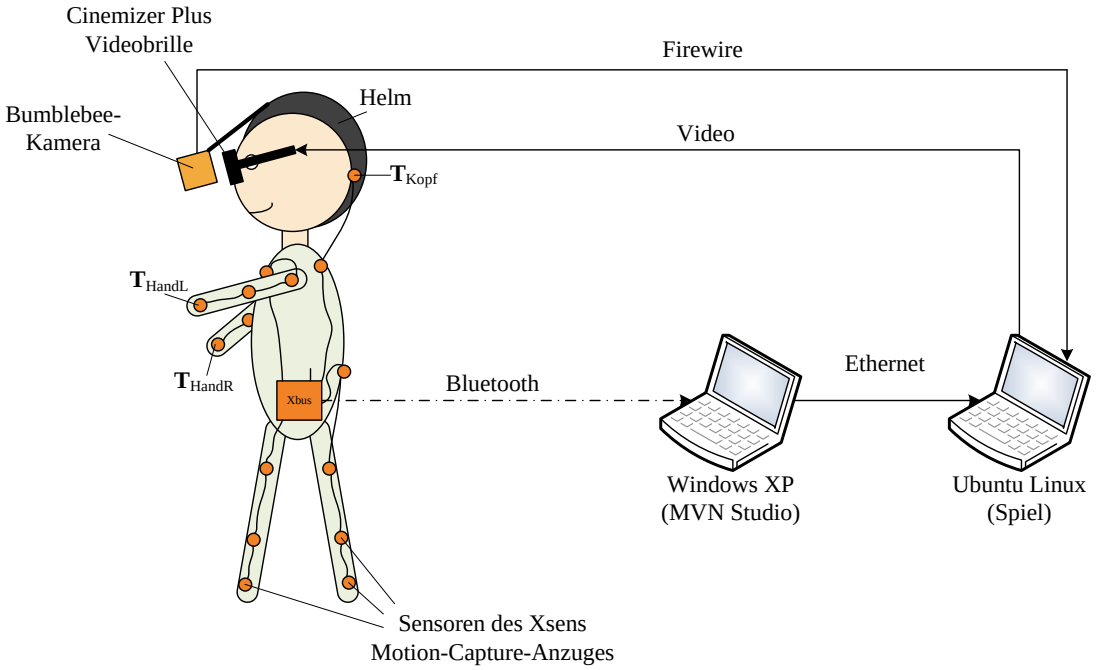

Abbildung 5.3.: Das Gesamtsystem im Überblick

so ist das System etwa vier Stunden einsetzbar (die Akkus des Anzugs halten ungefähr vier Stunden).

Für die Physiksimulation wird die Physikengine *Bullet* [11, 1] verwendet, die es ermöglicht, die nötige Kollisionserkennung durchzuführen. Dabei wird das Mesh als statische Umgebung initialisiert, wodurch die Kollision mit der „echten“ Welt entsteht.

An zwei Stellen der Implementierung wird ein Minimizer benutzt, der versucht eine gegebene Fehlerfunktion zu minimieren. Er liefert die Parameter, für die die Funktion minimal wird. Zum Einsatz kommt der multidimensionale Minimizer aus GSL (**G**NU **S**cientific **L**ibrary) [20]. Dieser verwendet eine vom Benutzer definierte, skalare Funktion

$$f(x_1, ..., x_n) \tag{5.1}$$

und bestimmt die Werte $x_1, ..., x_n$, für die die Funktion minimal wird. Verwendet wird der Minimizer für die Bestimmung der zwei Rotationen $\mathbf{T}_x(\alpha)$ und $\mathbf{T}_z(\beta)$ in den Kapiteln 3.1.3 und 4.1.

Die Transformationen $\mathbf{T}_{\text{Kopf}\to\text{Kamera}}$ und $\mathbf{T}_{\text{Kopf}\to\text{Brille}}$ werden mit der Identität angenähert.

6 Ergebnis

Hier sollen die Experimente erläutert und die Ergebnisse diskutiert werden. Zunächst werden die Versuche vorgestellt, die zur 3D-Welterfassung gemacht wurden. Anschließend werden die Ergebnisse der Aktionsparameter-Schätzung präsentiert.

6.1. 3D-Welterfassung

Dieses Unterkapitel soll die Güte der Welterfassung zeigen. Unter anderem wird geprüft, wie gut der RANSAC-Algorithmus funktioniert und wie sich die Kamera verhält, wenn man sie auf dem Helm befestigt.

6.1.1. Übereinstimmung der Ebenenschätzung mit der Wirklichkeit

Die Übereinstimmung der RANSAC-Orientierung mit der realen Orientierung der Kamera soll in diesem Versuch diskutiert werden. Dabei sollte die Differenz zwischen ermittelter und realer Orientierung so gering wie möglich sein.

Der Versuch wurde so gestaltet, dass die Bumblebee-Kamera auf einem Stativ angebracht und jeweils in $10°$-Schritten von der horizontalen ($0°$) in die vertikale ($90°$) Stellung gedreht wurde. Für jede Kamerastellung wurden anschließend zehn Punktwolken ermittelt. Dabei wurde die Bumblebee zunächst so aufgeschraubt, dass sie nach vorne drehbar war (*Versuch 1*). Danach wurde die Kamera und das Stativ um $90°$ gedreht, damit die Kamera zur Seite gekippt werden konnte (*Versuch 2*).

In der Abbildung 6.1 ist das Ergebnis dieses Experimentes dargestellt. Neben den Diagrammen ist jeweils der Versuchsaufbau skizziert. Das obere Diagramm zeigt hierbei die Resultate für das Kippen der Kamera nach vorne; unten sind die Werte für das seitliche Kippen zu sehen. Beide Diagramme fassen jeweils beide Rotationen zusammen.

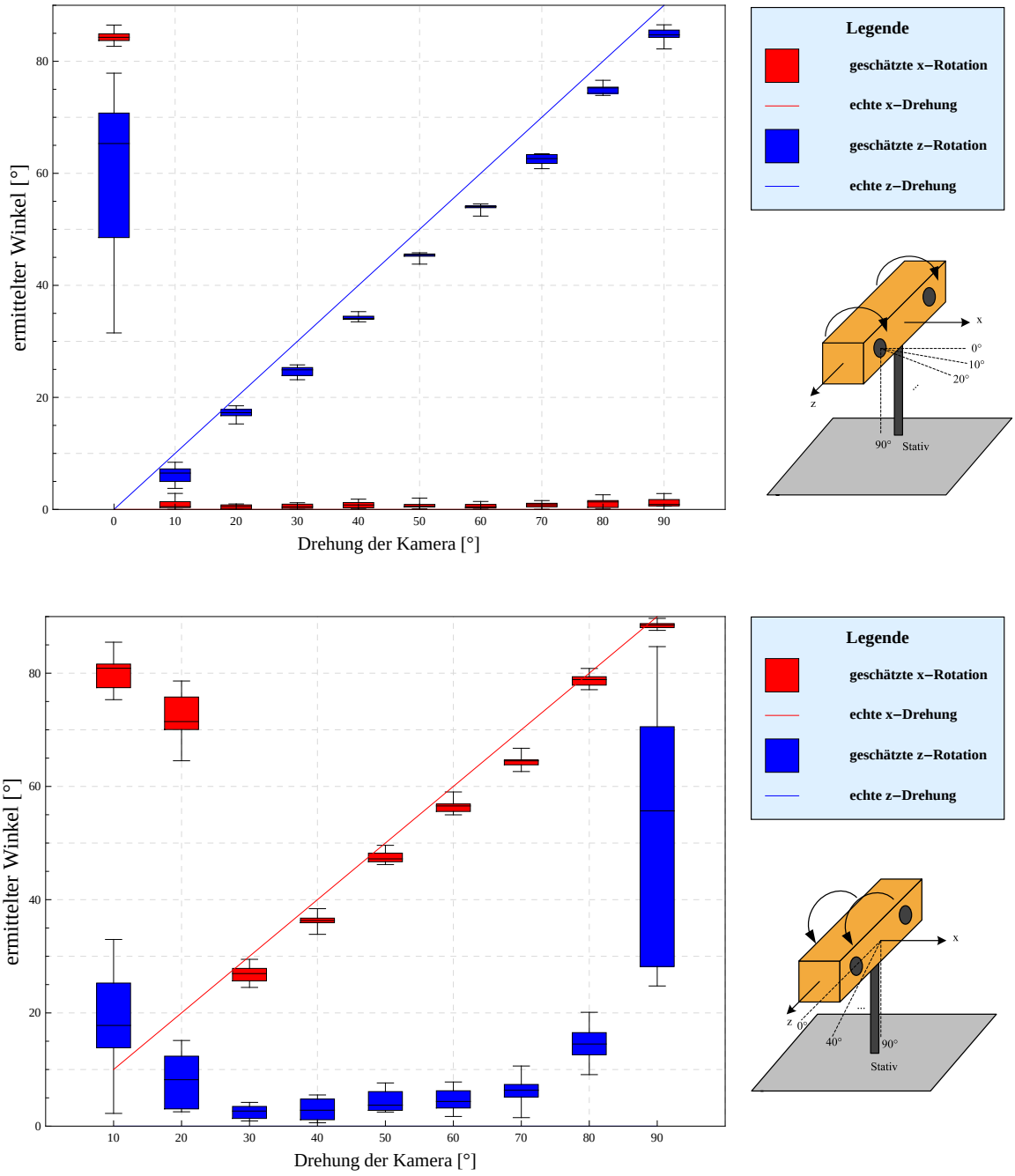

Abbildung 6.1.: Die vom RANSAC-Algorithmus bestimmten Winkel sind im Diagramm zu sehen. Die Box-Plots beinhalten den Median und die Quantile (0.25 und 0.75). Zu sehen sind die Diagramme für die Rotation nach vorne (**oben**) und zur Seite (**unten**), jeweils für beide Winkel. Die blaue und rote Linie demonstriert jeweils die optimalen Werte. Neben jedem Diagramm ist der Versuchsaufbau skizziert.

Zunächst folgt die Auswertung für das Kippen der Kamera nach vorne. Es ist zu sehen, dass die ermittelten Winkel des RANSAC-Algorithmus gut mit den tatsächlichen Winkeln übereinstimmen. Auch die Varianzen der einzelnen Winkel fällt klein aus. Die Hauptrotationsachse der Kamera bei dieser Kippbewegung ist die z-Achse. Die Werte sind in den blauen Plots zu sehen. Es fällt auf, dass die Werte für den $0°$-Fall stark von den gewünschten Werten abweichen. Die Kamera ist in dieser Stellung waagrecht und der RANSAC-Algorithmus findet den Boden im Mesh nicht mehr zuverlässig. Dadurch wird eine andere Ebene gewählt, wodurch es zu falschen Ergebnissen kommt. Außerdem variiert die ermittelte Ebene, was zu dem großen Abstand der beiden Quantile führt. Betrachtet man die zweite Rotationsachse (x-Achse), so erkennt man (bis auf den $0°$-Fall) ebenfalls eine gute Annäherung an den tatsächlichen Wert von $0°$. Die Erkennung des Bodens und dessen Ausrichtung funktioniert also zwischen $10°$ und $90°$ zuverlässig.

Der zweite Plot beschreibt das Resultat bei Drehung der Kamera zur Seite. Die Hauptrotationsachse ist in diesem Fall die x-Achse, deren Diagramme rot dargestellt sind. Die Ergebnisse für diesen Fall sind schlechter, als in Versuch 1. Der brauchbare Bereich beschränkt sich hier auf $30°$ bis $80°$. Das liegt daran, dass die Kamera immer direkt nach vorne gerichtet ist und so nur wenig Boden erfassen kann.

Der Algorithmus ist in der Lage den Boden und dessen Orientierung korrekt zu erkennen, sofern die Kamera genügend Boden erfassen kann. Der Algorithmus ist also für die Erkennung und Drehung des Bodens geeignet, um eine waagrechte Ausrichtung garantieren zu können. Es muss allerdings darauf geachtet werden, dass die zulässigen Winkel nicht verlassen werden, da es dann zu falschen Ergebnissen kommen kann.

6.1.2. Anzahl der ermittelten Inlier mit Stativ und Helm

Hier wird das Verhältnis zwischen den Punkten in der ermittelten Ebene und der insgesamt betrachteten Punkte untersucht. Abbildung 6.2 stellt die Kurven für 80 Versuche in drei verschiedenen Umgebungen dar.

Zuerst fällt auf, dass die Anzahl der Punkte in der ermittelten Hauptebene größer ist, wenn man ein Stativ verwendet. So sind hier zwischen 5% und 10% mehr Inlier vorhanden, als bei Benutzung der Helm-Variante. Das liegt daran, dass die Orientierungen der Kamera zwischen Helm und Stativ nicht vollständig übereinstimmen, da der Mensch seinen Kopf immer leicht bewegt. Aus dem vorgehenden Experiment folgt ebenfalls, dass sich die Anzahl der Inlier mit der Orientierung ändert, da dort in den Randfällen die falschen Ebenen erkannt wurden. In diesen Fällen ist die Anzahl der Inlier im Boden also zu gering, wodurch die Erkennung des Bodens fehlschlägt.

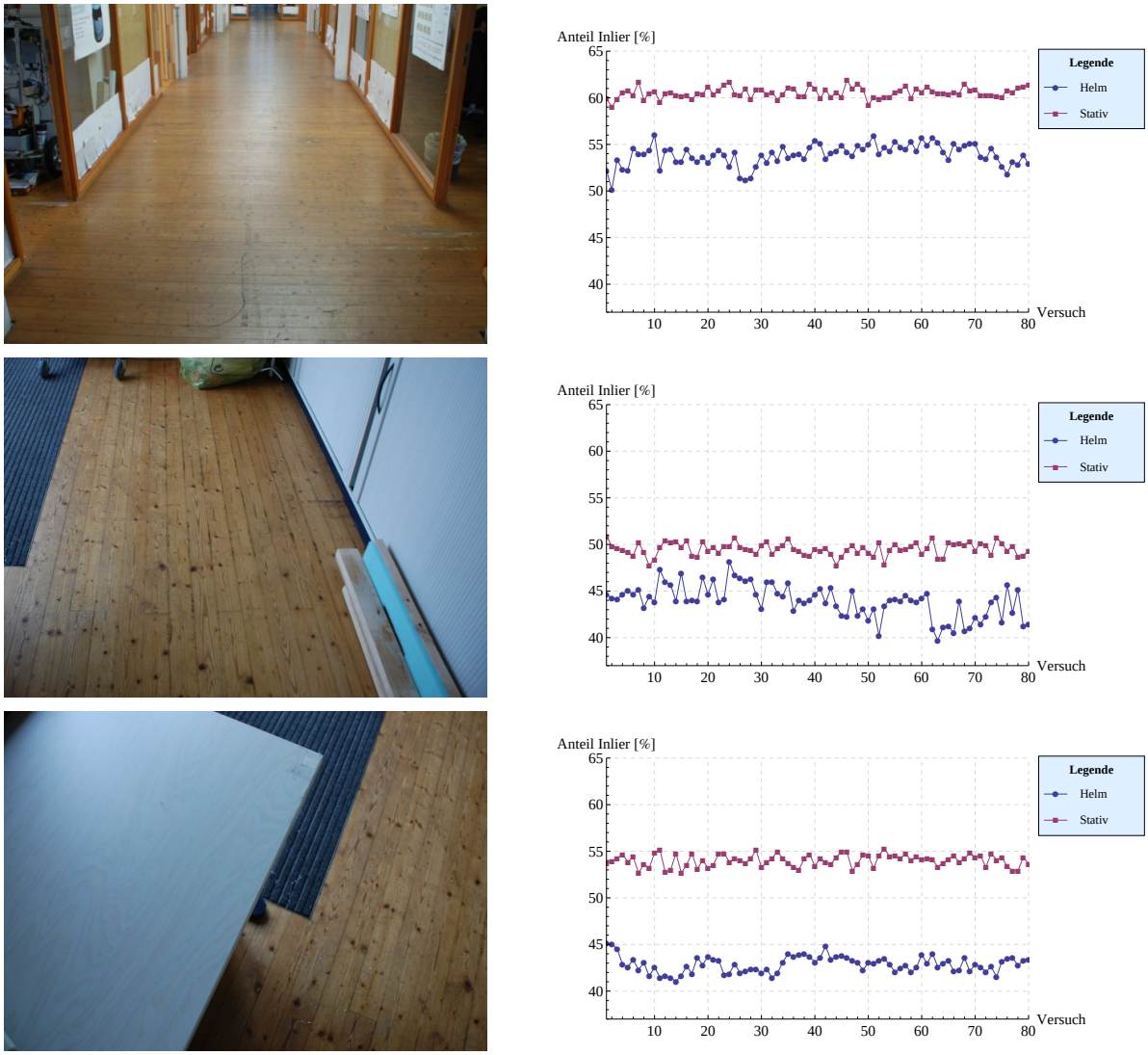

Abbildung 6.2.: Verhältnis der Inlier zu der Gesamtanzahl an betrachteten Punkten für Umgebung 1 (**oben**), Umgebung 2 (**Mitte**) und Umgebung 3 (**unten**)

Die Bewegung des Menschen schlägt sich auch in den Werten nieder. So ist die Varianz mit Stativ kleiner, als mit Helm. Die durchschnittliche Standardabweichung bei Verwendung des Stativs beträgt circa 0.65%, während die Standardabweichung mit Helm bei 1.3% liegt.

Eine weitere Beobachtung besteht darin, dass die Anzahl der Inlier abnimmt, wenn die Umgebung komplexer wird. In Umgebung 1 ist viel Boden sichtbar, was die Anzahl der Inlier auf 55% - 60% steigen lässt. Die beiden anderen Umgebungen haben weniger Inlier in der Hauptfläche, da hier der Anteil des Bodens geringer ist.

Allerdings haben in diesem Experiment, sowohl für Stativ, als auch für den Helm, die Anzahl der Inlier ausgereicht, um den Boden richtig zu identifizieren. Somit ist der Helm grundsätzlich als Befestigungskonstruktion für die 3D-Kamera geeignet, auch wenn eine feste Kamera

auf einem Stativ besser geeignet ist.

6.1.3. Qualität des ermittelten Meshs

Die Abbildungen 6.3 und 6.4 zeigen Bilder, die verdeutlichen, wie gut die Übereinstimmung des Meshs mit der realen Umgebung ist. In der ersten Abbildung ist ein Sessel zu sehen, auf dem herunterfallende Bälle abprallen oder zur Ruhe kommen. Das obere, linke Bild zeigt das ermittelte Mesh. Der RANSAC-Algorithmus hat dabei den Boden, in der Grafik hervorgehoben durch die rote Stelle, korrekt bestimmt. Der Sessel ist außerdem gut zu erkennen. Oben rechts wurde das reale Bild mit dem Mesh transparent überlagert. Hier sieht man, dass die Kanten des ermittelten Bodens entlang des Flurs bündig sind. Des weiteren stimmt die Position des Sessels mit der Position im Mesh überein. Nur bei weiter hinten liegenden Bereichen lässt die Qualität des Meshs nach. Unter anderem erkennt der RANSAC-Algorithmus den Boden nicht mehr vollständig.

In der zweiten Abbildung 6.4 ist eine Person zu sehen, die am Boden sitzt und als Kollisionsobjekt dient. Oben links ist wieder das Mesh und oben rechts die Überlagerung zu sehen. Die zwei unteren Bilder zeigen, wie ein Ball am Kopf abprallt. Beim überlagerten Bild sieht man, dass ein Mensch als Form erkannt wird, aber die Details fehlen. Wie aber zu sehen ist, fällt die Übereinstimmung, vor allem für vorne liegende Bereiche, gut aus.

6.2. Lernen

Die folgenden Experimente zeigen, dass die Schätzung der Schlägerparameter zu besseren Spielergebnissen führen. Außerdem wird gezeigt, wie gut die Bestimmung der aktiven Hand erfolgt.

6.2.1. Verbesserung durch die Schätzung der Parameter

Dieses Experiment wurde so gestaltet, dass der Spieler zunächst 30 Schläge mit einem Schläger ausführt, der eine feste Länge (95cm) hat und die Orientierung der Hand verwendet. Der Spieler konnte sich den Schläger anschauen und etwas trainieren, um sich an das System zu gewöhnen. Anschließend wurde der virtuelle Ball 25cm vor die Füße gelegt. Die Aufgabe der Testperson war es, den Ball zu treffen, ohne hinzuschauen. Wenn der Spieler den Ball getroffen hat, wurde er wieder vor die Füße gelegt. Abbildung 6.5 zeigt zwei Bilder aus dem Spiel.

Im zweiten Teil des Versuches wurden die optimalen Schlägerparameter durch Lernen geschätzt und die 30 Schläge wiederholt.

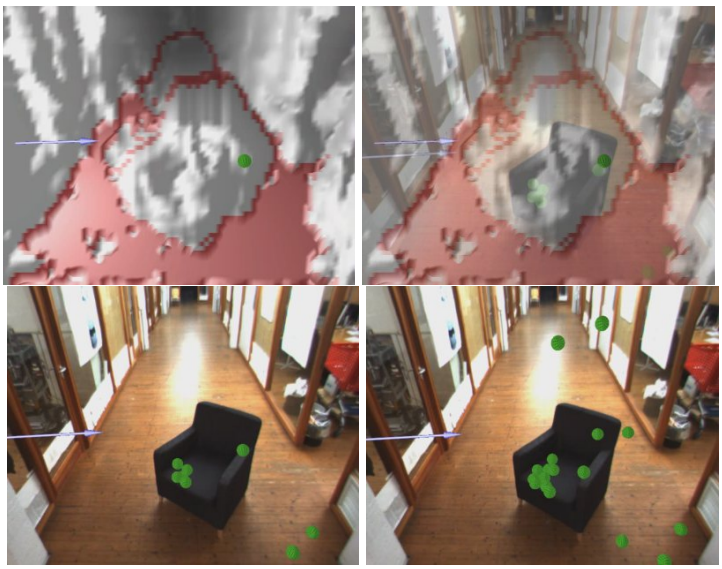

Abbildung 6.3.: Es wurde ein Mesh erzeugt (**oben links** als geschlossenes Mesh mit rotem Boden, bestimmt durch RANSAC, **oben rechts** als Überlagerung beider Bilder). Die **unteren zwei Bilder** zeigen, wie die Bälle auf dem Sessel und am Boden liegen bleiben.

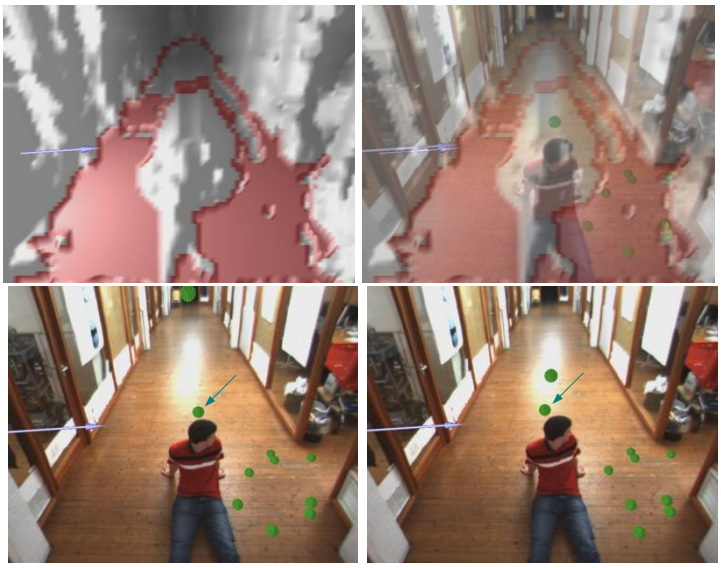

Abbildung 6.4.: Das Mesh einer Person ist **oben links** zu sehen (die rote Fläche entspricht dem Boden). **Oben rechts** ist die Überlagerung des Meshs mit der Umgebung dargestellt. Die **unteren zwei Bilder** zeigen, wie ein Ball auf dem Kopf abprallt.

Abbildung 6.5.: Bilder des Lernvorgangs: **Links:** Spieler während des Lernens; der Ball ist sichtbar, der Schläger nicht - **Rechts:** Spieler mit Schläger

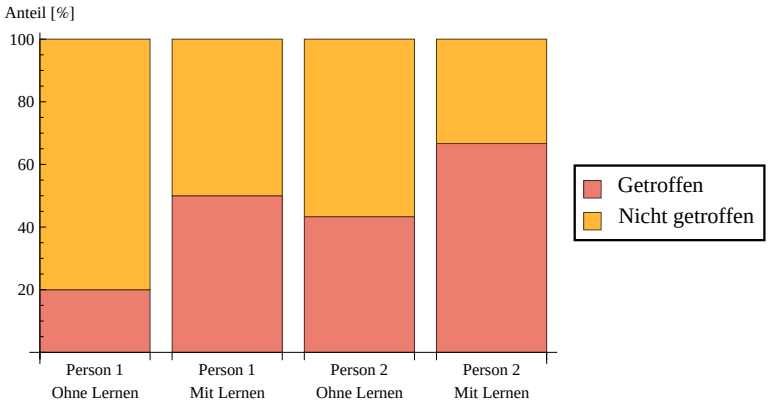

Abbildung 6.6.: Vergleich der getroffenen Schläge mit und ohne Schätzung der Schlägerparameter

Das Diagramm 6.6 zeigt das Verhältnis der getroffenen / nicht getroffenen Bälle der zwei Versuchspersonen. Bei beiden Personen war eine deutliche Steigerung feststellbar.

6.2.2. Schätzen der richtigen Hand

Der Versuch prüft die Qualität der Erkennung der spielenden Hand. Abwechselnd wurde dabei mit der linken, beziehungsweise rechten Hand gelernt. Diese Schritte wurde für beide Hände drei Mal wiederholt. Die nicht-spielende Hand wurde hierbei nicht bewegt. Vier weitere Lernvorgänge folgten, in denen beide Hände bewegt wurden. Zunächst wurde die rechte Hand für das Schlagen verwendet, wobei die linke Hand ebenfalls bewegt wurde (nach vorne und hinten). Danach folgten je ein Schlag mit der linken und rechten Hand, bei denen beide Hände am virtuellen Schläger waren. Abschließend wurden beide Hände identisch bewegt.

Tabelle 6.1 stellt die Ergebnisse des Versuchs dar. Es zeigt sich, dass die Schätzung sehr gut funktioniert. Für den normalen Einsatz reicht dieses einfache Verfahren der Varianzschätzung aus, um 90% der Schläge korrekt zu erkennen.

real \ *gesch.*	LINKS	RECHTS
LINKS	89%	11%
RECHTS	10%	90%

Tabelle 6.1.: Schätzen der linken und rechten Hand (gesch.: geschätzter Wert)

6.3. Kombiniertes Ergebnis

Abschließend soll die Funktionalität des gesamten Systems vorgestellt werden. Wie in den bisherigen Ergebniskapiteln beschrieben, funktionieren die einzelnen Komponenten des Programms. Für diesen Versuch wurde ein 3D-Mesh erzeugt. Abbildung 6.7 zeigt einige Bilder, wobei links die virtuelle und rechts die reale Welt zu sehen ist. Dabei sind die Bilder der virtuellen Welt aus Sicht der globalen Kamera dargestellt. Es ist zu sehen, dass der Minigolfschläger die richtige Länge und Orientierung hat. Rechts ist der Spieler dabei in der Realität zu sehen.

In Abbildung 6.8 sind Bilder aus dem Spiel zu sehen. Das linke Bild zeigt dabei den Spieler mit dem Ball vor den Füßen. Rechts ist die Überlagerung von Mesh und 2D-Bild zu sehen.

Insgesamt funktioniert das gesamte System. Allerdings hat sich herausgestellt, dass die Hardware in ihrer gegenwärtigen Form nicht vollständig für eine Augmented-Reality-Anwendung geeignet ist. Einerseits ist das Gewicht der Kamera und des Helmes zu hoch, als dass man ihn lange tragen könnte. Des weiteren beeinflusst der Helm den Kopfsensor des Motion-Capture-Anzuges, da dieser den Sensor berührt und ihn so verschiebt. Dadurch erhält man fortlaufend schlechtere Daten vom Anzug. Außerdem rutscht der Helm nach vorne, was dazu führte, dass die Kamera ihren Winkel änderte und die Übereinstimmung des 2D-Bildes mit dem 3D-Mesh nicht mehr passte. Für eine AR-Anwendung wäre also ein System besser geeignet, welches die Brille und die 3D-Kamera vereinigt. Auf diese Weise wäre das System kompakter und leichter.

6.4. Zusammenfassung

Das Kapitel stellte die durchgeführten Experimente dar. Zunächst wurden die Experimente präsentiert, die zur 3D-Welterfassung durchgeführt wurden. Es wurde gezeigt, dass die Mesherzeugung funktioniert und dass der RANSAC-Fußbodenschätzer die richtigen Werte liefert, sofern keine Extremfälle betrachtet werden (waagrechte/senkrechte Kamerastellung). Danach wurden die Versuche für das Schätzen der optimalen Schlägerlänge und -orientierung vorgestellt, die zeigten, dass das implementierte Lernverfahren sehr gut arbeitet. Abschließend war der Versuch zu sehen, der alle Komponenten vereinigte. Das Resultat ist mit der

Abbildung 6.7.: Links sind einige Bilder der virtuellen Welt aus der globalen Kamera zu sehen. Der rote Stab mit der grünen Kugel markiert das Ziel, zu dem der Ball geschossen werden muss. **Rechts** sind die realen Bilder dargestellt.

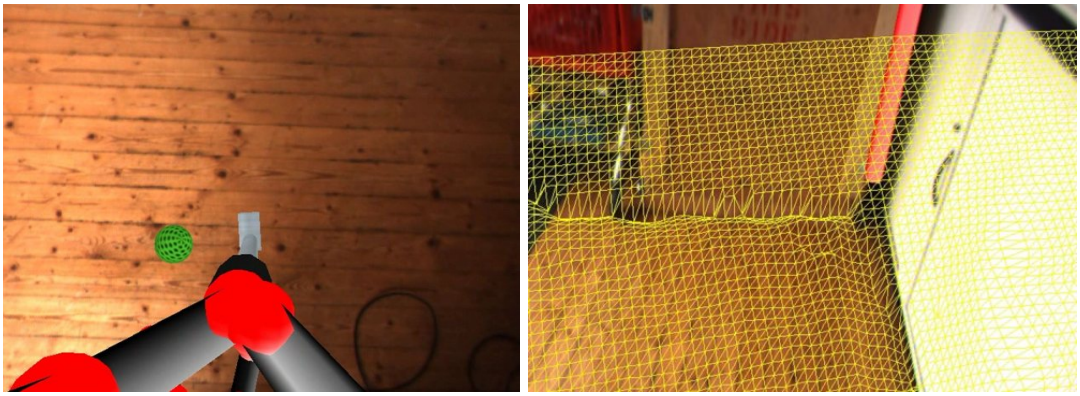

Abbildung 6.8.: Links: Spieler mit Ball und Schläger - **Rechts:** Überlagerung Mesh und Kamera-Bild

vorliegenden Hardware aber noch nicht gut genug, um ein angenehmes Spielen zu erlauben. Hierfür benötigt man ein kompakteres System.

7 Zusammenfassung und Diskussion

7.1. Zusammenfassung

Diese Arbeit beschäftigte sich mit den Einzelaspekten eines Augmented-Reality-Spieles. Es wurde ein Minigolf-Spiel entworfen, mit dem die einzelnen Punkte untersucht wurden. Zum Einsatz kamen ein Motion-Capture-Anzug, eine 3D-Kamera und eine Videobrille, die dem Spieler die Möglichkeit geben, ein Augmented-Reality-Spiel zu spielen.

Die Kamera liefert dabei ein 3D-Modell der Umgebung, aus dem ein Mesh erstellt wird. Hierfür wurde ein Meshing-Algorithmus vorgestellt. Das Mesh wird anschließend vor die virtuelle Kamera transformiert. Die dafür nötigen Transformationen wurden ausführlich behandelt. Die Kamera liefert außerdem ein 2D-Bild, welches mit dem Mesh in Übereinstimmung gebracht wurde. Eine Physikengine wurde zu Simulation der Kollisionen eingesetzt.

Neben der Welterfassung wurde ein Lernverfahren für die Bestimmung der Aktionsparameter vorgestellt. Dieses erlaubt die Schätzung der Parameter eines Minigolfschlägers aus den Demonstrationen des Spielers. Hierbei wurde die Länge und Orientierung des Schlägers ermittelt. Außerdem wurde geschätzt, welche Hand der Spieler für das Schlagen des Balles einsetzt.

In den Versuchen wurde gezeigt, dass das implementierte Verfahren funktioniert. Hierfür wurden Experimente durchgeführt, die die Qualität der 3D-Welterfassung prüfen. Es zeigte sich, dass das ermittelte Modell der Umwelt gut mit der Wirklichkeit übereinstimmt. Des Weiteren war zu sehen, dass das Lernverfahren die Bedienbarkeit des Spieles verbessert. Abschließend konnte gezeigt werden, dass das Gesamtsystem als Augmented-Reality-Spiel benutzt werden kann. Allerdings sind noch zu lösende Hardware-Probleme vorhanden.

7.2. Diskussion und Ausblick

Wie bereits im Ergebnisteil erklärt, bestehen noch Probleme mit der verwendeten Hardware. Der Helm ist zu schwer und rutscht daher, wodurch sich die Position der Kamera ändert. Ebenso verschiebt der Helm den Kopfsensor des Anzuges. Für die Transformation des Kopfsensors zur Kamera konnte daher keine geschlossene Form gefunden werden. Auch die Einstellung von Hand funktionierte nur kurz. Daher wäre eine andere Art von Hardware für die Verwendung in einem AR-Spiel nötig. Dabei sollten die Geräte so kompakt wie möglich sein und sich gegenseitig nicht beeinflussen. Eine Brille mit integrierter 3D-Kamera wäre beispielsweise eine Lösung, da diese den Kopfsensor des Motion-Capture-Anzugs nicht beeinflusst.

Ebenso ist ein System erstrebenswert, das einen mobilen Einsatz zulässt. Hierfür könnte anstatt der zwei Notebooks nur eines verwendetet werden. Dieses sorgt für die Kommunikation mit dem Anzug und überträgt die Daten kabellos an einen mobilen Computer (zum Beispiel Handys). Auf diesem wiederum könnte das Spiel laufen und die Brille angesteuert werden.

Interessant wäre ein dreidimensionales Spiel, da dieses einfacher zu bedienen wäre. Der aktuelle Ansatz nutzt keine 3D-Technik, wodurch der Spieler Schwierigkeiten hat, den Ball zu treffen, da die optischen Tiefeninformationen nicht vorhanden sind.

Augmented-Reality-Spiele werden in den nächsten Jahren an Bedeutung gewinnen. Diese Spiele erlauben neue Spielprinzipien, wodurch der Nutzer nicht mehr direkt an einem Computer gebunden ist. Er kann sich frei bewegen und mit der Umwelt in spielerischer Form interagieren.

A Mathematische Grundlagen

Dieses Kapitel führt einige mathematische Instrumente ein. Zunächst wird erklärt, was Eigenvektoren und Eigenwerte sind. Darauf aufbauend folgt eine Einführung in die Singulärwertzerlegung, die dann für die Löwdin-Orthogonalisierung verwendet wird.

A.1. Eigenwerte und Eigenvektoren

Die *Eigenvektoren* [16, S. 121ff] einer Matrix sind die Vektoren, die durch eine Multiplikation mit der Matrix nicht gedreht werden - sie werden lediglich um einen Faktor gestreckt. Dieser Faktor ist der zum *Eigenvektor* gehörende *Eigenwert*.

Die *Eigenwerte* errechnen sich mit Hilfe der folgenden Gleichung:

$$\det(\mathbf{A} - \lambda \cdot \mathbf{I}) = 0, \tag{A.1}$$

wobei $A \in \mathbb{R}^{n \times n}$ die Matrix ist, von der man die *Eigenwerte* bestimmen möchte. Dabei ist $\lambda \in \mathbb{R}^n$ und $\mathbf{I} \in \mathbb{R}^{n \times n}$ die Einheitsmatrix. Die Eigenwerte sind folglich die Nullstellen des charakteristischen Polynoms

$$\chi_{\mathbf{A}} = \det(\mathbf{A} - \lambda \cdot \mathbf{I}). \tag{A.2}$$

Die *Eigenvektoren* sind dann folgendermaßen definiert:

$$(\mathbf{A} - \lambda_i \cdot \mathbf{I}) \cdot \mathbf{e}_i = \mathbf{0} \tag{A.3}$$

Hier steht λ_i für den i-ten Eigenwert. Die Gleichung wird nach $\mathbf{e}_i$ gelöst, wobei $\mathbf{e}_i$ der entsprechende Eigenvektor ist.

A.2. Singulärwertzerlegung

Jede Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ kann folgendermaßen faktorisiert werden [31, S. 123ff]:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{A.4}$$

Die Spalten der orthogonalen Matrix $\mathbf{U} \in \mathbb{R}^{m \times m}$ sind die Eigenvektoren von $\mathbf{A}\mathbf{A}^T$ und die Spalten der orthogonalen Matrix $\mathbf{V} \in \mathbb{R}^{n \times n}$ sind die Eigenvektoren von $\mathbf{A}^T\mathbf{A}$. Die Matrix $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ ist eine Diagonalmatrix, deren Diagonalelemente *singuläre Werte* heißen. Sie sind die positiven Quadratwurzeln der Eigenwerte von $\mathbf{A}\mathbf{A}^T$ und $\mathbf{A}^T\mathbf{A}$.

Die obige Faktorisierung wird Singulärwertzerlegung (SVD, **S**ingular **V**alue **D**ecomposition) genannt. SVD wird in dieser Arbeit zur Orthogonalisierung von Matrizen verwendet, wie sie im nächsten Abschnitt erläutert wird.

A.3. Löwdin-Orthogonalisierung mit SVD

Möchte man eine Matrix $\mathbf{A}$ orthogonalisieren, so dass das Ergebnis die optimale Annäherung an $\mathbf{A}$ ist, so kann das *Löwdin-Orthogonalisierungsverfahren*, das auf einer Singulärwertzerlegung basiert, eingesetzt werden. Sei die Singulärwertzerlegung von $\mathbf{A} \in \mathbb{R}^{n \times n}$ gegeben durch

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{A.5}$$

dann lässt sich die optimale Annäherung an $\mathbf{A}$ [6, S. 83ff] durch

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \tag{A.6}$$

berechnen. $\mathbf{R}$ ist dabei orthogonal und der Fehler, der durch die Rechnung entsteht, wird gleichmäßig auf jeden Spaltenvektor verteilt. Das ist auch einer der Gründe, warum hier kein Gram-Schmidt-Verfahren verwendet wird. Dort ist die Fehleraufteilung nicht symmetrisch.

A.4. RANSAC-Algorithmus

Der RANSAC-Algorithmus (**Ran**dom **Sa**mple **C**onsensus) dient in der Computergrafik zur Bestimmung von Modellparametern. Einsatz findet das Verfahren beim „Feature-Matching“ (beispielsweise bei der Erstellung von Panoramabilder aus Einzelbilder [9]) oder beim Erkennen von geometrischen Primitiven. Eingeführt wurde der Algorithmus von Fischler und

Bolles [18]. An dieser Stelle soll der Algorithmus allgemein erläutert werden. In dieser Arbeit wird der Algorithmus beim Ermitteln der Hauptebene in einer 3D-Punktwolke eingesetzt.

Sei $\mathcal{D} = \{\mathbf{d}_1, ..., \mathbf{d}_N\}, \mathbf{d}_i \in \mathbb{R}^d, 1 \leq i \leq N$ eine Menge von N Wahrnehmungen. Der Algorithmus wählt eine Teilmenge $\mathcal{D}_{\text{minimal}} \subset \mathcal{D}$ aus. Dabei hängt die die Mächtigkeit $|\mathcal{D}_{\text{minimal}}|$ vom Modell M ab. Der Parametervektor θ richtet sich nach der minimalen Anzahl n an Werten, die zur eindeutigen Bestimmung von M nötig sind. Diese Parameter seien durch den Vektor $\theta \in \mathbb{R}^n$ definiert. Die Auswahl von $\mathcal{D}_{\text{minimal}}$ erfolgt zufällig und gleichverteilt aus $\mathcal{D}$. Basierend auf $\mathcal{D}_{\text{minimal}}$ wird θ geschätzt. Die Schätzfunktion

$$\theta = f_M(\mathcal{D}_{\text{minimal}}) \tag{A.7}$$

hängt dabei vom jeweiligen Modell M ab.

Im nächsten Schritt werden diese Modellparameter in Bezug auf die gesamte Menge $\mathcal{D}$ ausgewertet. Eine Kostenfunktion bestimmt dabei die Qualität von θ. Als typisches Kostenmaß kommt das Verhältnis von Inlier zu Outlier zum Einsatz. Inlier sind Punkte, die unter Annahme einer Fehlertoleranz ϵ, mit θ harmonieren. Die Fehlerfunktion ist dabei durch

$$e_M(d, \theta) = \text{dist}(M_\theta, d) \tag{A.8}$$

gegeben, wobei $\text{dist}(\cdot, \cdot)$ eine entsprechende Distanzfunktion ist, die den Abstand von d zum Modell M unter Verwendung von θ angibt. Die Menge der Inlier

$$I(\theta) = \{\mathbf{d} \in \mathcal{D} : e_M(d, \theta) \leq \epsilon\} \tag{A.9}$$

ergibt sich dann aus der Fehlerfunktion.

Durch wiederholte Auswahl von $|\mathcal{D}_{\text{minimal}}|$ und Bestimmung von θ erhält man ein

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \left(P\left(\mathcal{D} \mid M, \theta\right)\right), \tag{A.10}$$

für das die Anzahl der Inlier maximal ist.

Ein wichtiger Parameter ist die Fehlertoleranz, die angibt, um wie viel ein Inlier von den Modellparametern abweichen darf. Des weiteren ist die Anzahl der Wiederholungen ein weiterer relevanter Parameter. Dieser beeinflusst einerseits die Qualität (zu wenige Versuche führen zu schlechten Ergebnissen) und andererseits die Laufzeit (zu viele Versuche brauchen Zeit).

Der Algorithmus A.1 beschreibt das Verfahren als Pseudocode. Weitere Informationen sind im Originalpaper [18] oder in [43] zu finden.

```
Eingabe : 𝒟 - Menge von Wahrnehmungen
          Modell - Modell für Ebene
          maxIt - maximale Anzahl an Iterationen
          ϵ - erlaubter Fehler
Ausgabe : Bestes Modell, entsprechende Inlier, Fehler
1  bestesModell ← NULL
2  besteInlier ← NULL
3  Güte ← ∞
4  for i ← 0 to maxIt do
5      testWerte ← drei zufällige Punkte aus 𝒟
6      möglichesModell ← zu testWerte passende Modellparameter
7      möglicheInlier ← NULL
8      forall the d ∈ 𝒟 do
9          if d harmoniert mit möglichesModell mit Fehler ϵ then
10             Füge Wert in möglicheInlier ein
11     if Anzahl Elemente in möglicheInlier größer als in besteInlier then
12         besseresModell ← auf möglicheInlier angepasstes Modell
13         fehler ← Güte von besseresModell
14         if fehler < Güte then
15             bestesModell ← besseresModell
16             Güte ← fehler
17             besteInlier ← möglicheInlier
18 return bestesModell, Güte, besteInlier
```

Algorithmus A.1: RANSAC-Algorithmus zur Flächenerkennung als Pseudocode

A.5. Eindimensionaler Gauß-Smoother

Der Gauß-Smoother hat seinen Namen von der Gauß-Funktion, die für die Glättung verwendet wird. Diese hat die allgemeine Form

$$f_G(\mathbf{x}) = a \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mu\|^2\right), \tag{A.11}$$

wobei a die Streckung in y-Richtung, σ die Bandbreite und μ die Verschiebung in x-Richtung beeinflusst. Die entstehenden Glocken-Funktionen sind für verschiedene Parameter in Abbildung A.1 dargestellt.

Für die Glättung einer Sequenz an Werten wird eine eindimensionale Faltung verwendet [42,

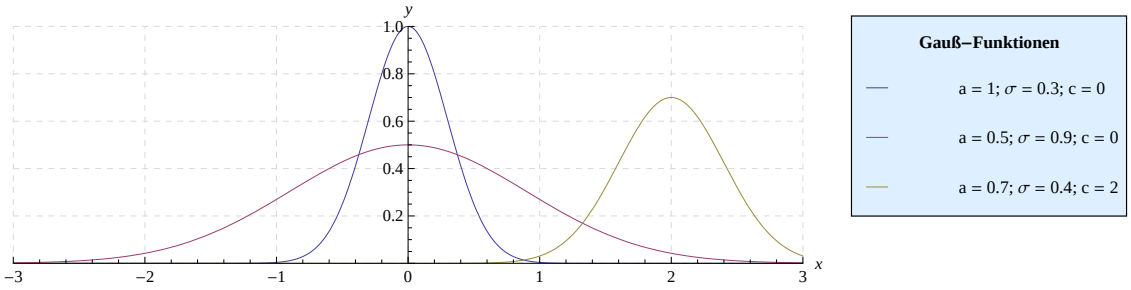

Abbildung A.1.: Beispiel für einige Gauß-Funktionen

S. 53ff] und

$$\phi(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{A.12}$$

als Kernelfunktion eingesetzt. μ wird nicht benötigt, da die Funktion nicht verschoben wird.

Der Kernel wird dazu verwendet, jeden Wert der Eingabesequenz so anzupassen, dass die benachbarten Werte ebenfalls berücksichtigt werden. Der Kernel gibt dabei an, wie stark die benachbarten Werte gewichtet werden sollen.

Die Fläche der Funktion $\phi(x)$ ist Eins, also

$$\int_{-\infty}^{\infty} \phi(x)dx = 1, \tag{A.13}$$

da es sich um eine Wahrscheinlichkeitsdichte handelt. So ändert der Gauß-Smoother nicht die eigentliche Funktion (da die summierten Gewichte Eins ergeben), sondern glättet nur die Werte und entfernt das Rauschen.

Die Gauß-Verteilung ist für jedes $\phi(x)$ ungleich Null, was zu einem unendlich großen Kernel führen würde. Da die Gauß-Funktion aber schnell kleiner wird, reicht es, nur einen gewissen Bereich um die Maximalstelle von $\phi(x)$ zu betrachten. Wertet man $\phi(x)$ an $2k + 1$, $k \in \mathbb{N}$ Stellen aus (symmetrisch zu $x = 0$), dann erhält man einen Kernel

$$K = (K_{-k}, ..., K_{-1}, K_0, K_1, ..., K_k). \tag{A.14}$$

Die Stärke der Glättung lässt sich durch σ beeinflussen. Je größer σ ist, desto stärker fällt die Glättung aus, da die Gaußfunktion flacher und breiter wird.

Bei Glättung der Sequenz D ergeben sich an der Stelle i folgende neue Werte:

$$\tilde{\mathcal{D}}_i = \sum_{j=i-k}^{i+k} K_{-j} \cdot \mathcal{D}_j \tag{A.15}$$

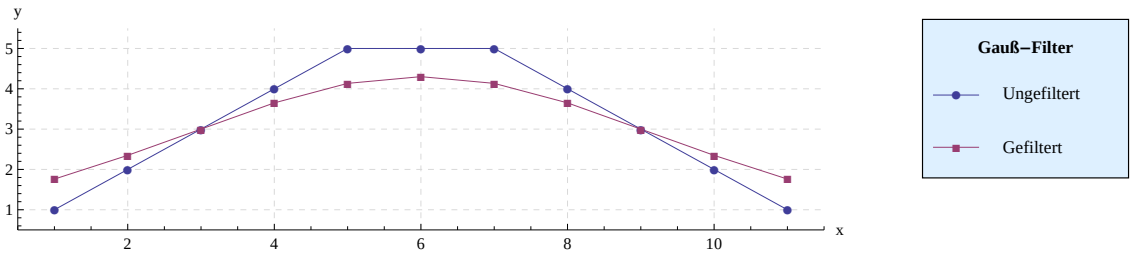

Abbildung A.2.: Funktion vor und nach Anwendung eines Gauß-Filters mit $k = 3$

Abbildung A.2 zeigt ein einfaches Beispiel für eine gefilterte Datenreihe.

B Tabellen

B.1. Hardware

EIGENSCHAFT	
Sensoren	CCD-Bildsensor
Kameraabstand	12cm
Frame-Rate	20FPS
Anschluss	6-pin IEEE-1394a Firewire
Auflösung	maximal 1032×776px
Gewicht	342g
Abmessungen	157×36×47,4mm

Tabelle B.1.: Die wichtigsten Eigenschaften der Bumblebee2-Stereo-Kamera von Point-Grey. Weitere Informationen sind dem Datenblatt der Kamera zu entnehmen [37]

EIGENSCHAFT	
Sensoren	17 MTx-Tracker, 6 Freiheitsgrade
Einsatzzeit	3 Stunden
Kommunikation	USB 1.1 oder 2.0 / Bluetooth 2.0
Reichweite (kabellos)	zwischen 50m (innen) und 150m (außen)
Gewicht	1930g (mit Batterien und Kabel)
Aufnahmefrequenz	bis zu 120Hz

Tabelle B.2.: Die wichtigsten Eigenschaften des Motion-Tracking-Anzugs von Xsens

B.2. Tabellen und Diagramme

WINKEL	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
Optimum [°]	0	10	20	30	40	50	60	70	80	90
Mittelwert [°]	61.1	6.27	17.2	24.6	34.3	45.3	53.9	62.5	75.0	84.5
Abweichung [%]	-	37.3	13.8	17.9	14.4	9.46	10.1	10.7	6.19	6.07
Varianz [°]	221	2.1	0.88	0.72	0.28	0.37	0.34	0.82	0.61	1.9

WINKEL	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
Optimum [°]	0	0	0	0	0	0	0	0	0	0
Mittelwert [°]	84.4	0.89	0.47	0.56	0.89	0.72	0.59	0.75	1.22	1.30
Varianz [°]	1.13	0.84	0.15	0.17	0.41	0.32	0.18	0.25	0.59	0.78

Tabelle B.3.: Mittelwert, Abweichung vom Optimum und Varianz bei Kippen der Kamera um die z-Achse (oben) und x-Achse (unten) für Versuch 1

WINKEL	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
Optimum [°]	0	0	0	0	0	0	0	0	0	0
Mittelwert [°]	-	19.0	8.37	2.61	3.01	4.52	4.71	6.08	14.6	51.4
Varianz [°]	-	77.	21.	1.3	3.3	3.6	4.0	7.7	9.8	481

WINKEL	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
Optimum [°]	0	10	20	30	40	50	60	70	80	90
Mittelwert [°]	-	80.1	72.3	27.1	36.3	47.5	56.6	64.4	78.9	88.4
Abweichung [%]	-	701	262	9.8	9.34	5.03	5.73	8.03	1.42	1.77
Varianz [°]	-	9.1	17	2.3	1.3	1.1	1.3	1.2	1.4	0.39

Tabelle B.4.: Mittelwert, Abweichung vom Optimum und Varianz bei Kippen der Kamera um die z-Achse (oben) und x-Achse (unten) für Versuch 2

Literaturverzeichnis

[1] API, BulletPhysics: *Homepage*. online. http://bulletphysics.com/Bullet/BulletFull/. – Version: 09.08.2010

[2] ARGALL, Brenna ; BROWNING, Brett ; VELOSO, Manuela: Learning by demonstration with critique from a human teacher. In: *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*. New York, NY, USA : ACM, 2007. – ISBN 978–1–59593–617–2, S. 57–64

[3] ARQUAKE: *ARQuake*. online. http://wearables.unisa.edu.au/projects/arquake. – Version: 09.08.2010

[4] ARTOOLKIT: *ARToolkit - Homepage*. online. http://www.hitl.washington.edu/artoolkit/. – Version: 15.08.2010

[5] AZUMA, R. ; BAILLOT, Y. ; BEHRINGER, R. ; FEINER, S. ; JULIER, S. ; MACINTYRE, B.: Recent advances in augmented reality. In: *Computer Graphics and Applications, IEEE* 21 (2001), nov., Nr. 6, S. 34 –47

[6] BEAVER, Scott: *Banach Algebras of Integral Operators, Off-Diagonal Decay, and Applications in Wireless Communications*, University of California, Diss., 2004

[7] BLUM, Lisa ; BROLL, Wolfgang ; MÜLLER, Stefan: Augmented reality under water. In: *SIGGRAPH '09: SIGGRAPH '09: Posters*. New York, NY, USA : ACM, 2009, S. 1–1

[8] BROTHER INDUSTRIES, Ltd: *Brother officially named Retinal Imaging Display AiRScouter™*. online. http://www.brother.com/en/news/2010/airscouter/. – Version: 27.08.2010

[9] BROWN, Matthew ; LOWE, David G.: Automatic Panoramic Image Stitching using Invariant Features. In: *International Journal of Computer Vision* 74 (2007), Nr. 1, S. 59–73

[10] BROWN, Myron Z. ; BURSCHKA, Darius ; HAGER, Gregory D.: Advances in Computational Stereo. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 25 (2003), Nr. 8, S. 993–1008

[11] BULLETPHYSICS: *Homepage*. online. http://bulletphysics.org/. – Version: 09.08.2010

[12] CHEKHLOV, Denis ; GEE, Andrew P. ; CALWAY, Andrew ; MAYOL-CUEVAS, Walterio: Ninja on a Plane: Automatic Discovery of Physical Planes for Augmented Reality Using Visual SLAM. In: *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 978–1–4244–1749–0, S. 1–4

[13] COATES, Adam ; ABBEEL, Pieter ; NG, Andrew Y.: Learning for control from multiple demonstrations. In: *ICML '08: Proceedings of the 25th international conference on Machine learning*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–205–4, S. 144–151

[14] CORP., Vuzix: *Wrap 920AR*. online. http://www.vuzix.com/iwear/products_wrap920ar.html. – Version: 29.08.2010

[15] EUROPE, Sony Computer E.: *EyePet*. online. http://www.eyepet.com. – Version: 15.08.2010

[16] FARIN, Gerald ; HANSFORD, Diane: *Lineare Algebra: Ein geometrischer Zugang (Springer-Lehrbuch) (German Edition)*. Berlin, Deutschland : Springer, 2003. – ISBN 3540418547

[17] FAUGERAS, Olivier ; LUONG, Quang-Tuan: *The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications*. The MIT Press, 2004. – ISBN 0262562049

[18] FISCHLER, Martin ; BOLLES, Robert: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM* 24 (1981), Nr. 6, S. 381–395

[19] GALL, J. ; STOLL, C. ; AGUIAR, E. de ; THEOBALT, C. ; ROSENHAHN, B. ; SEIDEL, H.-P.: Motion capture using joint skeleton tracking and surface estimation, 2009, S. 1746 –1753

[20] GSL: *Homepage: GSL Multidimensional Minimization*. online. http://www.gnu.org/software/gsl/manual/html_node/Multidimensional-Minimization.html. – Version: 18.08.2010

[21] JOHNSTON, R.S. ; WILLEY, S.R.: Development of a commercial retinal scanning display. In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* Bd. 2465, 1995, S. 2–13

[22] KOWALSKY, Hans-Joachim ; MICHLER, Gerhard O.: *Lineare Algebra*. Berlin, Deutschland : de Gruyter, 2003. – ISBN 3110179636

[23] KUHNER, Daniel ; RUCHTI, Philipp ; MAIER, Christian: *Computergestütztes Bewegungstraining: Erkennung und Korrektur von Körperhaltung und Bewegungsabläufen*. Freiburg, 2010. – Bachelor-Projekt

[24] LEONARD, J.J. ; DURRANT-WHYTE, H.F.: Simultaneous map building and localization for an autonomous mobile robot, 1991, S. 1442 –1447 vol.3

[25] LIAROKAPIS, Fotis ; MACAN, Louis ; MALONE, Gary ; REBOLLEDO-MENDEZ, Genaro ; FREITAS, Sara d.: A Pervasive Augmented Reality Serious Game. In: *VS-GAMES '09: Proceedings of the 2009 Conference in Games and Virtual Worlds for Serious Applications*. Washington, DC, USA : IEEE Computer Society, 2009. – ISBN 978–0–7695–3588–3, S. 148–155

[26] LÖCHTEFELD, Markus ; ROHS, Michael ; SCHÖNING, Johannes ; KRÜGER, Antonio: LittleProjectedPlanet: An Augmented Reality Game for Camera Projector Phones. In: *Proceedings of MRIW* (2009)

[27] MICROVISION: *Pico Projector Displays*. online. http://www.microvision.com/pico_projector_displays/index.html. – Version: 29.08.2010

[28] MILGRAM, P. ; COLQUHOUN, H.W.: A framework for relating head-mounted displays to mixed reality displays. In: *Human Factors and Ergonomics Society Annual Meeting Proceedings* Bd. 43. Houston, Texas, USA, 1999, S. 1177–1181

[29] MISTRY, Pranav ; MAES, Pattie: SixthSense: a wearable gestural interface. In: *SIGGRAPH ASIA '09: ACM SIGGRAPH ASIA 2009 Sketches*. New York, NY, USA : ACM, 2009, S. 1–1

[30] MISTRY, Pranav ; MAES, Pattie ; CHANG, Liyan: WUW - wear Ur world: a wearable gestural interface. In: *CHI '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2009. – ISBN 978–1–60558–247–4, S. 4111–4116

[31] MONAHAN, John F.: *Numerical Methods of Statistics*. Cambridge University Press, 2001. – ISBN 0521791685

[32] ODA, Ohan ; FEINER, Steven: Rolling and shooting: two augmented reality games. In: *CHI EA '10: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2010. – ISBN 978–1–60558–930–5, S. 3041–3044

[33] ODA, Ohan ; LISTER, Levi J. ; WHITE, Sean ; FEINER, Steven: Developing an augmented reality racing game. In: *INTETAIN '08: Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. – ISBN 978–963–9799–13–4, S. 1–8

[34] PÉREZ, Emiliano ; SALAMANCA, Santiago ; MERCHÁN, Pilar ; ADÁN, Antonio ; CERRADA, Carlos ; CAMBERO, Inocente: A Robust Method for Filling Holes in 3D Meshes Based on Image Restoration. In: *ACIVS '08: Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 978–3–540–88457–6, S. 742–751

[35] POINT-GREY: *Triclops Software Development Kit (SDK)*. 3.1. Richmond, Canada: Point-Grey, 2003

[36] POINT GREY RESEARCH, Inc.: *Tripclops*. online. http://www.ptgrey.com/products/triclopsSDK/index.asp. – Version: 08.08.2010

[37] POINT GREY RESEARCH, Inc.: *Bumblebee - Stereo Vision Camera System (Datasheet)*. Feb 2009. Richmond, Canada, 02 2009. http://www.ptgrey.com/products/bumblebee2/bumblebee2_xb3_datasheet.pdf

[38] POINT GREY RESEARCH, Inc.: Stereo Vision Introduction and Applications / Point Grey Research, Inc. 2010 (TAN2008005). – Forschungsbericht. – Technical Application Note TAN2008005

[39] REITINGER, B. ; ZACH, C. ; SCHMALSTIEG, D.: Augmented Reality Scouting for Interactive 3D Reconstruction, 2007, S. 219 –222

[40] ROETENBERG, Daniel ; LUINGE, Henk ; SLYCKE, Per: *Xsens MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors*. 8. April 2009. Enschede, Niederlande: XSENS TECHNOLOGIES, 2009. http://www.xsens.com/images/stories/PDF/MVN_white_paper.pdf

[41] SCHOWENGERDT, B.T. ; SEIBEL, E.J.: True 3-D scanned voxel displays using single or multiple light sources. In: *Journal of the Society for Information Display* 14 (2006), S. 135

[42] SHIH, Frank Y.: *Image Processing and Pattern Recognition: Fundamentals and Techniques*. New York, USA : Wiley-IEEE Press, 2010. – ISBN 0470404612

[43] TORR, P. H. S. ; ZISSERMAN, A.: MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. In: *Computer Vision and Image Understanding* 78 (2000), S. 2000

[44] XSENS: *Homepage*. online. http://www.xsens.com/en/general/mvn. – Version: 08.08.2010

[45] ZACH, Christopher ; SORMANN, Mario ; KARNER, Konrad: High-Performance Multi-View Reconstruction. In: *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2825–2, S. 113–120

[46] ZEISS: *cinemizer plus*. online. http://www.zeiss.de/cinemizer. – Version: 10.08.2010

Abbildungsverzeichnis

Abbildungsverzeichnis