Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Christian Gutjahr

Modellbasierte Sensorfusion von kamerabasierter Bewegungsverfolgung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Gutjahr

Modellbasierte Sensorfusion von kamerabasierter Bewegungsverfolgung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke

Eingereicht am: 3. März 2015

Christian Gutjahr

Thema der Arbeit
Modellbasierte Sensorfusion von kamerabasierter Bewegungsverfolgung

Stichworte
HCI, Gestensteuerung, Modellgestaltung, Sensorfusion, Bewegungsverfolgung

Kurzzusammenfassung
Diese Arbeit befasst sich mit der Realisierung einer Fusion von mehreren Sensoren an einem einheitlichem Modell. Anhand eines genauen Szenarios soll damit eine Erhöhung des Gestenrepertoires sowie eine Verringerung von Fehlinformationen erreicht werden.

Christian Gutjahr

Title of the paper
Model based sensor fusion with camera based motion tracking

Keywords
HCI, gestures, motion capture, sensor fusion, model design

Abstract
This work thoroughly takes on the realisation of sensor fusion through multiple sensors on a single model. With the help of a precise scenario, the repertoire of gestures will rise and a reduction of misinformation will be coherent.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Jeder hat im heutigen Zeitalter seine eigene Vorstellung von einem „Smart Home" (deutsch: Intelligentes Zuhause). Sei es das Einschalten der Deckenlampe durch Klatschen in die Hände, die automatische Regelung von Heizung oder Rolläden anhand von Uhrzeit oder Raumtemperatur, bis hin zum Steuern aller Geräte durch Bewegung oder Sprache - wie man sie aus Science-Fiction-Filmen kennt. Eng verwandt zu dieser Art ist das „ambient assisted living", das auf Deutsch etwa in „umgebungsgestütztes Wohnen" übersetzt werden kann. Bei sogenannten „AAL's" liegt das Hauptaugenmerk auf der Unterstützung älterer sowie benachteiligter Menschen. So werden etwa durch Sensoren die Bewegungen der Nutzer verfolgt und automatisch Türen geöffnet oder ein Hinfallen der Nutzer festgestellt.

Bei dem „Smart Home" hingegen liegt das Hauptaugenmerk auf der Erhöhung des Komforts durch Geräte und soll vor allem die alltäglichen Aufgaben unterstützen und vereinfachen. Die Grundidee von „ubiquitous computing" (deutsch: Rechnerallgegenwart) ist dabei, Geräte von überall aus, ohne hohen Aufwand zu bedienen. So kann der Nutzer an kalten Tagen, auf dem Weg nach Hause per Smartphone-Applikation die Heizung einstellen oder Konfigurationen verschiedener Geräte innerhalb der Wohnung für den anstehenden Winter anpassen. Dies kann auch genutzt werden, um Voreinstellungen für die morgendliche Prozedur des Aufstehens einzurichten, indem, abhängig von der Tiefe des Schlafes des Nutzers sowie der aktuellen Uhrzeit, ein Aufwachszenario gestaltet wird. Der Nutzer kann dann zum Beispiel durch warmes Licht geweckt werden. Gleichzeitig wird die Heizung im Bad eingeschaltet. Durch Sensoren im Boden oder im Bett kann festgestellt werden, ob der Nutzer aufgestanden ist. Sobald dies der Fall ist, können aktuelle Verkehrsnachrichten sowie die neusten Mails auf dem Smartphone oder anderen Bildschirmen innerhalb der Wohnung angezeigt werden. Im Rahmen von „disappeared computing", was auf Deutsch sinngemäß in „das Verschwinden der Computer" übersetzt werden kann, soll die Verwendung von Computern dem Nutzer nicht auffallen. Dafür wird eine natürliche und einfache Bedienung der einzelnen Geräte per

Bewegungen oder Sprache angestrebt.

In dieser Arbeit wird darauf hingearbeitet eine Möglichkeit zu entwickeln, mit der mehrere Sensoren miteinander verbunden werden können, sodass nach außen hin ein einzelner großer Sensor entsteht. Mit dessen Hilfe können dann Bewegungen genauer sowie vielfältiger ausgeübt und die jeweiligen Geräte intuitiver bedient werden. Dafür wird in der Arbeit auf den aktuellen Stand der Technologie eingegangen und die verschiedenen Möglichkeiten aufgezeigt und miteinander verglichen.

1.2 Gliederung der Arbeit

In Kapitel 2 wird auf die Grundlagen der Technologie, genauer auf die der Mensch-Computer-Interaktion eingegangen. Darunter fällt die Begriffserklärung der Bewegungsverfolgung (2.1.1) sowie die der Gestenerkennung (2.1.2). In Folge dessen wird auf die verschiedenen Arten von Gesten eingegangen (2.1.3 & 2.1.4) und grundlegende Probleme der Interaktion von Computern mit Hilfe von Gesten aufgezeigt. Anschließend wird auf die Gestaltung eines Modells und dessen Grundverfahren eingegangen (2.2). Zuletzt wird in Kapitel 2 auf das daraus resultierende Ziel der Arbeit (2.3) eingangen und dieses anhand eines genauen Szenarios (2.3.1) eingegrenzt.

Kapitel 3 bezieht sich dann auf die eigentliche Realisierung der SensorFusion. Zuerst wird auf das Gesamtkonzept eingegangen (3.1), in welchem sich das System befindet. In Kapitel 3.2 wird auf die gewählte Systemarchitektur eingangen. Darunter fällt die Wahl der Kommunikation (3.2.1) der einzelnen Module sowie die Art der Kopplung (3.2.2). Kapitel 3.3 bezieht sich dann auf die Wahl der Sensoren, mit denen die Bewegungen des Nutzers aufgezeichnet werden. Danach wird die Wahl des Modells in Kapitel 3.4 beschrieben und anschließend auf das Mapping der Daten auf dieses Modell (3.5) eingegangen.

Kapitel 4 bezieht sich dann auf die Beschreibung der Testumgebung (4.1) sowie die Usability des Systems (4.2).

2 Analyse

Dieses Kapitel befasst sich mit dem Thema der Bewegungsverfolgung sowie der daraus erfolgenden Gestenerkennung und -deutung. Desweiteren wird auf die Modellgestaltung und Grundlagen der Interaktion mit dem Computer eingangen.

2.1 Human-Computer-Interaction

Die Geschichte dieses Teilgebietes der Informatik lässt sich auf das Industriezeitalter zurückführen, denn die reibungslose Interaktion zwischen Mensch und Maschine wurde dort das erste Mal als notwendig angesehen und diente zur Verbesserung der Bedienbarkeit der Maschinen.

Unter Human-Computer-Interaction (Deutsch: Mensch-Computer-Interaktion) versteht man die Interaktion zwischen einem Menschen und einem Computersystem. Eine Interaktion besteht dann, wenn eine Handlung des Benutzers den Status des Systems oder Programmes verändert (vgl. [Heinecke 2004]). Dies bedeutet, dass schon die Bedienung eines Autos oder des Fernsehers eine Interaktion zwischen dem Benutzer und des Systems darstellt. Die benutzerfreundliche Bedienung eines Systems steht in diesem Bereich der Informatik stark im Vordergrund.

Interaktionen werden in der Informatik durch Schnittstellen realisiert. Solche Schnittstellen können für den Menschen Maus, Tastatur und auch Spracheingaben sein. Es lassen sich auch zwischen 2D Interaktionen, wie zum Beispiel durch die Eingabe via Touchpad und 3D Interaktionen unterscheiden. Diese 3D Interaktionen finden heutzutage größtenteils in der Spieleindustrie kommerzielle Verwendung und werden häufig durch kamerabasierte Bewegungsverfolung umgesetzt.

2.1.1 Bewegungsverfolgung

Unter Bewegungsverfolgung versteht man im Allgemeinen die Aufzeichnung von menschlicher Bewegung. Einfache Bewegungsmelder und Lichtschranken, für etwa Alarmanlagen, fallen so-

mit im weiteren Sinne auch unter diese Definition. Typische Bewegungsmelder arbeiten mittels Infrarotaufnahmen oder elektromagnetischer Wellen, um Objekte in bestimmten Bereichen zu erfassen. Lichtschranken hingegen senden ein Signal, sobald ein Lichtstrahl unterbrochen wurde (vgl. [Hillebrand & Tschakert 2008]).

Da diese beiden Arten der Bewegungserkennung jedoch nur die bloße Anwesenheit von Menschen registrieren, eignen sie sich nicht für die durchgehende Verfolgung von Bewegungsabläufen. Die bei Bewegungsmeldern verwendete Technologie aber lässt sich bei ”Motion Capture” verwenden. So werden bei elektromagnetischen Systemen elektromagnetische Impulse niedriger Frequenz ausgestrahlt und von Sensoren, welche an der zu verfolgenden Person angebracht sind, gemessen. Eine weitere Möglichkeit besteht darin, Ultraschallwellen auszustrahlen, welche von Empfängern am Körper ausgewertet werden. Diese Technologie ist im Vergleich zu elektromagnetischen Systemen preiswert, jedoch empfindlich gegenüber Hintergrundgeräuschen. Im Folgenden wird näher auf einen anderen Ansatz der Bewegungsverfolgung eingegangen.

Kamerabasierte Ansätze

Rein auf Kameras basierende Bewegungsverfolgung bietet die Möglichkeit, durchgehend menschliche Bewegungen präzise auszuwerten, ohne dabei Sensoren, welche die Bewegungsfreiheit einschränken können, an dem Körper befestigen zu müssen. Dazu stellen sie eine kostengünstige Alternative zu den markerbasierenden Sensorsystemen dar, welche vor allem in der Filmindustrie für möglichst realistische Bewegungsabläufe in animierten Sequenzen verwendet werden. Nachfolgend werden häufig verwendete Arten der kameragestützten Bewegungsverfolgung kurz beschrieben.

- **Stereo Kamera**
  Zwei Kameras werden in einem bestimmten Abstand zueinander aufgestellt und die entstehenden Bilder werden zu einem Raumbild zusammengesetzt.

- **Time of Flight Kamera**
  Hierbei wird die Dauer berechnet, die das Licht braucht, um zu einem bestimmten Punkt zu gelangen. Mit der daraus resultierenden Zeit lässt sich die Distanz des Punktes zur Kamera berechnen, was eine Generierung des Tiefenbildes ermöglicht.

- **Markerbasierte Aufzeichnung**
  Bei der markerbasierten Bewegungsverfolgung werden mehrere optische Marker an dem

gesamten Körper befestigt. Anschließend werden mit einer normalen Kamera die Bewegungen gefilmt und im Nachhinein die Positionen der Marker mit dem aufgenommenen Bild verrechnet.

- **Infrarot-Tiefenkamera**
  Bei der Infrarotkamera werden im Sichtfeld dieser Kamera Punkte, die sich im Infrarotspektrum befinden und nicht für das menschliche Auge sichtbar sind, projiziert. Mit Hilfe dieser Punkte werden Informationen über die Tiefe der aufgenommenen Objekte gewonnen.

False Positive & False Negative

Wie bei allen Geräten kann es auch bei Kameras zur Fehlauswertung der aufgenommenen Bilder kommen. Die dabei auftretenden Fehler können mitunter „False Positives" und „False Negatives" sein. False Positive steht für die Erkennung einer Bewegung, obwohl keine ausgeübt wurde. Diese Fehler können zum Beispiel bei Bewegungsmeldern durch Tiere ausgelöst werden oder bei Kameras durch die Auswertung von vermeintlichen Bewegungen.

False Negatives hingegen beschreiben die Nichterkennung der ausgeübten Bewegung. Diese Art von Fehler können bei Kameras mit geringer Framerate in Verbindung mit schnellen Bewegungen entstehen. Die Framerate steht dabei für die Anzahl der Bilder pro Sekunde einer Kamera.

2.1.2 Gestenerkennung

Die Bewegungsverfolgung ist, wie oben erwähnt, ein Hauptbestandteil der 3D Interaktion zwischen dem Menschen und einem Computer. Bewegungen alleine reichen jedoch nicht aus, um mit einem System interagieren zu können. Um dies zu erreichen, müssen bestimmte Bewegungen von dem System als Gesten wahrgenommen und korrekt gedeutet werden.

Gesten sind laut McNeill „spontane oder bewusst eingesetzte Bewegung des Körpers, besonders der Hände und des Kopfes, die jemandes Worte begleitet oder ersetzt und eine bestimmte innere Haltung ausdrückt"[McNeill 1992]. McNeill unterscheidet Gesten in „nebensprachliche" Gesten sowie in „lexikalisitierte" oder „emblem" Gesten (vgl. [Martin 2014]). Die nebensprachlichen Gesten sind dabei Bewegungen, welche von der sprechenden Person unbewusst gemacht werden. „Emblem"-Gesten jedoch sind Bewegungen, welche Informationen, abhängig von der kulturellen Herkunft, übertragen.

Bei der Gestenerkennung filtert man nun durch Anwendung von Algorithmen an den Daten der Bewegungsverfolgung, die nebensprachlichen Gesten, welche keine Informationen enthalten, von den Emblem-Gesten heraus. Um dies zu ermöglichen, werden die Gesten, mit denen das System gesteuert werden kann, im Vorfeld definiert. So könnte ein Kopfnicken oder ein „Daumen hoch" zur Bestätigung einer Aktion genutzt werden.

Taxonomie

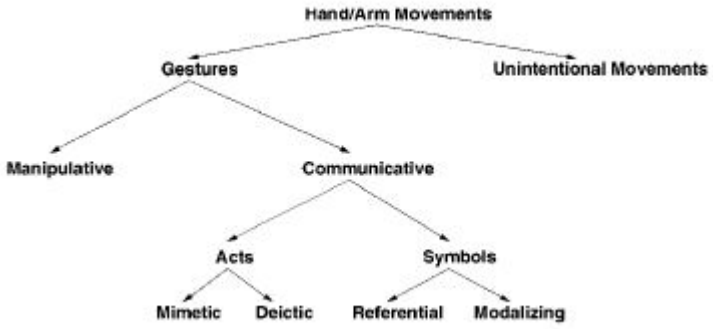


Abbildung 2.1: Taxonomie von Pavlovic, Sharma, Huang
Quelle: [Bowie 1997]

Vladimir I. Pavlovic, Rajeev Sharma und Thomas S. Huang haben in ihrer Arbeit ein Klassifizierungsschema beschrieben, welches die gewollten Bewegungen weiter unterscheidet (siehe Abbildung 2.1). Demnach gibt es zwei unterschiedliche Arten von Gesten, die „manipulativen Gesten" und die „kommunikativen Gesten". Manipulative Gesten werden dort als Gesten beschrieben, die sich auf Objekte der realen Welt beziehen. Diese Art von Gesten werden dort nicht weiter unterteilt, jedoch wurden diese von einer Arbeitsgruppe an den Wismarer Wirtschaftsinformatik-Tagen analysiert. Genaueres dazu findet man in der Arbeit von Sobin Ghose (vgl. [Ghose 2014]).

Die kommunikativen Gesten werden in der Arbeit von Pavlovic, Sharma und Huang in „Acts" (deutsch: Handlungen) und „Symbols" (deutsch: Symbole) unterteilt (vgl. [Bowie 1997]). Handlungen werden als Bewegungen beschrieben, die ihre Bedeutung in den eigentlichen Bewegungen finden. So sind „mimetic" (deutsch: nachahmend) Acts, Bewegungen die Aktionen imitieren und „deitic" (deutsch: hinweisend) Acts wiederum Bewegungen, die auf etwas deuten (vgl. [Bowie 1997]). Symbolische Gesten werden als Gesten beschrieben, die Objekte referenzieren

(zum Beispiel das Rotieren des Fingers, das an ein Rad erinnert) oder die gesprochenen Worte verdeutlichen sollen. Desweiteren werden diese symbolischen Gesten als die Wichtigsten im Kontext der HCI definiert.

2.1.3 2D-Gesten

Gesten im zweidimensionalem Feld werden bei berührungsempfindlichen Displays, wie zum Beispiel Tablets sowie Smartphones, eingesetzt. Entscheidend ist dabei, dass sich die Geste auf eine zweidimensionale Ebene beschränkt. Die generelle Problematik der Start-End-Geste ist für diese Art von Gesten irrelevant, da davon ausgegangen werden kann, dass eine Geste startet, sobald ein oder mehrere Finger das Display berühren. Die Geste endet dann mit dem Abheben des Fingers von dem Monitor beziehungsweise des Displays. Auch die Dauer der Geste kann unterschiedliche Aktionen hervorrufen. Mögliche Gesten können sein:

- **Tippen**
  Dies ist eine grundlegende Geste, bei der ein Finger auf einen Punkt des Displays gedrückt wird. Meist werden damit Anwendungen gestartet oder auch eine simple Navigation innerhalb einer Anwendung vollbracht.

  Analog dazu existiert die Geste des langen Tippen: Ein Halten des Fingers an einer bestimmten Stelle. Ein Anwendungsbereich kann etwa der Aufruf von extra Informationen einer Anwendung sein.

- **Wischen**
  Das Wischen, auch „Swipe" genannt, ist ein Ziehen des Fingers über das Display. Bei Smartphones kann diese Geste zum Wechseln von Reitern innerhalb einer Applikation verwendet werden.

- **Zwei-Finger-ziehen**
  Bei dieser Geste werden zwei Finger aufgelegt und voneinander weggezogen. Diese Geste entspricht der typischen Zoom-Geste. Sie wird verwendet, um etwa Bilder zu vergrößern. Zum Verkleinern des Bildes werden die beiden Finger zueinander hingezogen.

2.1.4 3D-Gesten

Anders als 2D-Gesten findet diese Art von Gesten in einem dreidimensionalem Raum statt. Somit kommt neben den zwei vorhandenen Achsen noch eine dritte hinzu, was die Ausführung und die Auswertung einer Geste erschwert. Joachim Boetzer beschreibt in seiner Arbeit, dass diese 3D-Gesten aufgrund von unterschiedlichen Körperproportionen verschiedener Nutzer

eine hohe Toleranz aufweisen sollten [Boetzer 2008]. Eine Studie, welche die Benutzung eines Assistenzsystems mit 2D- und 3D-Gesten geprüft hat, hat ergeben, dass viele Nutzer die 2D-Gesten bevorzugen, da diese als einheitlich und intuitiv angesehen werden (vgl. [Martin 2014]). Die Verwendung von 3D-Gesten im Zusammenhang mit einer zweidimensionalen Oberfläche wurde als unnatürlich empfunden. Im Folgenden wird auf häufig verwendete Gesten im dreidimensionalem Raum eingegangen.

- **Tippen**
  Diese Geste ist vergleichbar mit der Benutzung an einem berührungsempfindlichen Display. Der Nutzer streckt einen Finger nach vorne aus und bewegt die Hand als wolle er ein Element auf dem Display drücken.

- **Wischen**
  Anders als bei der Geste im zweidimensionalem Bereich, wird hierbei die komplette Hand benutzt. Der Nutzer streckt die Hand offen nach vorne und bewegt diese dann nach links oder rechts um zum Beispiel einen anderen Reiter aufzurufen.

- **Greifen**
  Bei der Greifgeste wird die zunächst offene Hand über ein Element gehalten, das greifbar ist. Das Zusammenziehen der Finger zur Faust stellt dann das Greifen dieses Elementes dar. Anschließend kann dann das Element über den Bildschirm gezogen werden. Sobald der Nutzer die Hand öffnet wird das Element wieder „losgelassen".

Start-End-Problematik

Die Problematik, wann eine Geste beginnt und wann sie abgeschlossen ist, ist in dem dreidimensionalen Raum ein zentrales Problem. So kann nach der erfolgreichen Verwendung einer Wisch-Geste zum Wechseln des Reiters innerhalb eines Programmes die Bewegung der Hand zurück zur Ursprungsposition erneut als Wisch-Geste gedeutet werden. Da ein Mensch nie völlige Bewegungslosigkeit erreicht, muss dem Computersystem eine Hilfestellung geboten werden, um erkennen zu können, wann eine Geste startet, und wann sie endet. Diese Hilfestellung ist in der zweidimensionalen Ebene das Auflegen des Fingers. In der dreidimensionalen Ebene könnte dies das Drücken eines Schalters oder Knopfes erreichen. Jedoch wird dadurch der Interaktionsbereich des Nutzers eingeschränkt. Alternativ dazu wäre eine Nutzung von Sprachbefehlen möglich, die dem System signalisieren, dass eine Geste begonnen wird.

Das Latenzproblem

Ein weiteres Problem innerhalb der Gestenerkennung und Bewegungsverfolgung ist die der Latenz. Eine Verzögerung der Wahrnehmung einer Geste kann im falschen Kontext zu Fehldeutungen und somit zu falschen Handlungen des Systems führen. Sobald ein Nutzer eine Geste durchführt, jedoch das System nicht reagiert und die gewollte Aktion nicht durchführt, kann der Nutzer, in der Annahme, dass die erste Geste vom System nicht registriert wurde, erneut diese Geste ausführen. Das System hat beide Gesten registriert und führt nach kurzer Reaktionslosigkeit beide Gesten aus. Dadurch entsteht ungewolltes Verhalten der Software, welches vom einfachen Überspringen von Reitern oder Seiten auch zum unwiderruflichen Löschen von Daten führen kann.

Ein Grund dafür kann die Art der Kommunikation der verschiedenen Module innerhalb des Systems sein. Dies kann zu Nachrichtenstaus führen und somit die Verarbeitung von anstehenden Nachrichten deutlich verlangsamen. Auch die Verarbeitung der Daten kann zu einem Nachrichtenstau oder gar Nachrichtenverlust führen. Falls die gewählten Operationen einen zu großen Aufwand darstellen, werden die Nachrichten nicht schnell genug verarbeitet.

2.2 Modellgestaltung

Ein Modell ist allgemein betrachtet ein vereinfachtes, in manchen Fällen auch ein beschränktes Abbild eines Objekt aus der Realität. Häufig werden diese zum Beispiel in Form von Miniaturen zur Veranschaulichung und Darstellung in verschiedenen Arten von Projekten benutzt. So kann ein Architekt neben dem Entwurf eines Gebäudes auch ein Modell von diesem entwerfen. Zum einen kann er dadurch seinen Entwurf verdeutlichen und die Idee besser präsentieren, zum anderen kann er mit Hilfe des Modells erste Validierungen des Entwurfs vornehmen.

In dem Kontext der Bewegungsverfolgung ist ein Modell ein vereinfachtes Abbild eines Menschen. Diese Modelle können sich, abhängig von den Projekten, stark voneinander unterscheiden. Bei Projekten, die sich auf die Bewegungen des Gesichtes spezialisieren, würde ein dafür verwendetes Modell nicht auf die Detailierung der Hände eingehen. Wie auch in der Architektur dient ein solches Modell der Validierung und der Veranschaulichung der aufgenommenen Daten. Das Modell wird mit Hilfe von Computerprogrammen erstellt. Dazu wird die Haut, „Mesh” genannt, durch ein Polygonnetz dargestellt (vgl. Abbildung 2.2 links).

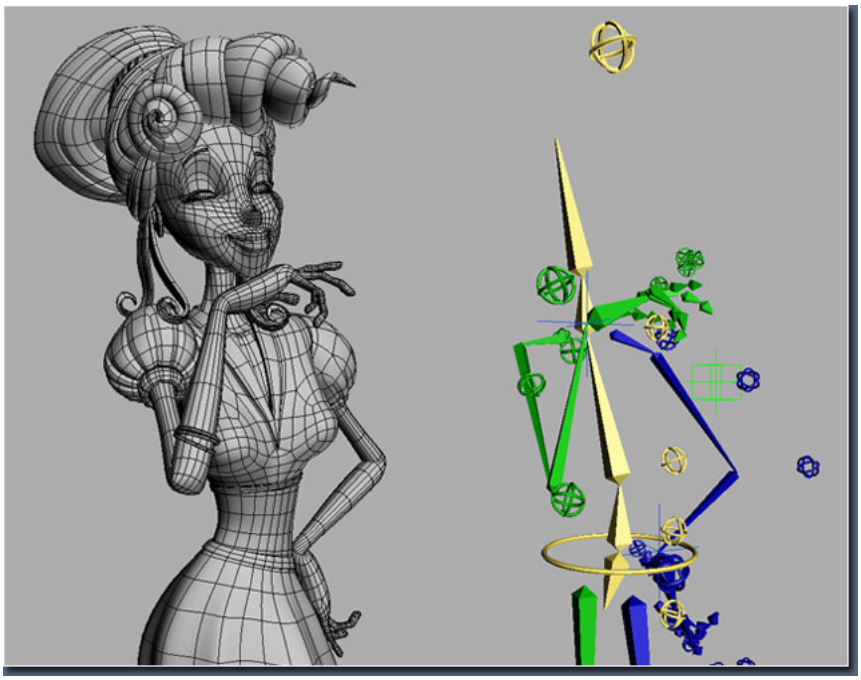


Abbildung 2.2: Darstellung von Mesh und Rig.
Quelle: http://www.3dtotal.com/index_tutorial_detailed.php?id=123&page=2#.VOctw_mG-2Z

2.2.1 Rigging

Um das Modell bewegen zu können, muss es mit einem Skelett beziehungsweise „Rig" versehen werden. Das „Rigging" beschreibt in der 3D-Animation das Erstellen eines aus Knochen und Gelenken bestehendes, bewegliches Skelettkonstrukts. Dieses ähnelt, je nach Abhängigkeit der auszuwertenden Bewegungen, zu einem bestimmten Grad dem menschlichen Skelett. Abbildung 2.2 veranschaulicht, wie ein solches „Rig" aussehen kann. Während dem Erstellen des Skelettes wird desweiteren definiert, wie die einzelnen Knochen in Verbindung zueinander stehen. Damit können Bewegungsabläufe des Menschens natürlicher dargestellt werden. So beeinflusst zum Beispiel eine Bewegung des Oberarmknochens die Position des Unterarmknochens.

2.2.2 Skinning

Die Verbindung des Körpers und der Skelettkonstruktion wird als „skinning" (deutsch: Häutung) beschrieben. Ziel dieses Arbeitsablaufes ist es, zuzuordnen, welche Polygone im Netz von Bewegungen bestimmter Knochen beeinflusst werden. Dies hat zur Folge, dass sich die

Haut verformt, sobald, ein oder mehrere Knochen bewegt werden. Um bestimmen zu können, wie stark bestimmte Polygone abhängig von den Bewegungen der Knochen sind, spielt das „Weight-Painting" (deutsch: Aufmalen der Gewichtungen) eine starke Rolle. Diese Darstellung zeigt die Gewichtung für einen oder mehrere ausgewählte Knochen (vgl. Abbildung 2.3).

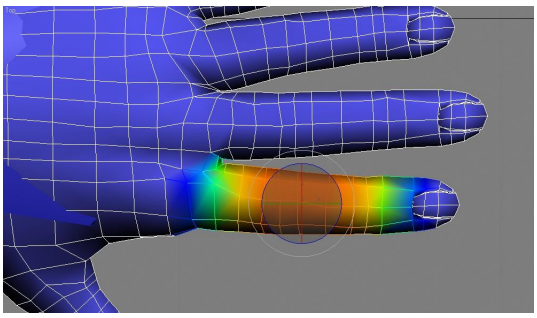


Abbildung 2.3: Darstellung des WeightPaintings von Blender
Quelle: http://forum.3d-plugin.com/viewtopic.php?f=41&t=415

Die Berechnung des Gewichtes für einen Punkt im Mesh erfolgt durch folgende simple Zuweisung: Für jeden Punkt im Netz werden gleichmäßig alle darunterliegende Knochen zugeordnet (vgl. [Bridson 2008]). Liegt der Punkt über dem Fuß, so hat dieser Knochen die Gewichtung 1 und alle anderen Knochen 0. Sollte der Meshpunkt über zwei Knochen stehen, so wird für diese Knochen die Gewichtung 0.5 und für alle anderen Knochen der Wert 0 zugeordnet.

$$\sum_{n=0}^{N} w_n = 1 \tag{2.1}$$

Der „Linear Blend Skinning"-Algorithmus ist aufgrund seiner Einfachheit einer der häufigsten verwendeten Algorithmen zur Berechnung der Deformierung.

$$v' = \sum_{n=0}^{N} w_n T(n) B(n)^{-1} v \tag{2.2}$$

$w(n)$ steht für die Gewichtung des Knochens, der den Vertex v im Polygonnetz beeinflusst. Die Transformation des Knochens wird durch $T(n)$ bestimmt und die Matrix $B(n)^{-1}$ transformiert den Vertex v in das lokale Koordinatensystem des Knochens (vgl. [Bauerdiek 2014]). Der Vertex v' stellt dann die Position des Meshpunktes nach einer Bewegung eines oder mehrerer Knochen dar.

2.3 Ziel der Arbeit

Das Ziel der Arbeit ist es, eine Erweiterung des Gestenrepertoirs durch Verbindung mehrerer Sensoren zu erreichen und dies für die Nutzung innerhalb anderer Projekte weiterzugeben. Dazu soll der gesamte Körper einer Person mit Hilfe mehrerer Sensoren verfolgt und die gesammelten Daten an einem gemeinsamen Modell dargestellt werden. Durch die Verwendung mehrerer kamerabasierter Sensoren, die sich auf unterschiedliche Körperregionen spezialisieren, erreicht man ein Ganzkörpermodell. Zusätzlich erhöht sich die Präzision des Trackings, da mögliche Überschneidungen miteinander verrechnet werden, um somit Fehlinformationen eines Sensors auszugleichen.

Die Validierung anhand eines Modelles dient in diesem Fall der Überprüfung, ob die gesammelten Daten physikalisch Möglich sind. So kann sichergestellt werden, dass eine vermeintliche Drehung des Kopfes um 180 Grad, eine Fehlinformation darstellt und im weiteren Verlauf nicht berücksichtigt wird.

2.3.1 Szenario

Der von Sobin Ghose in seiner Arbeit „Konzeption und Evaluation eines interaktiven Badezimmerspiegels" konzipierte „Smart-Mirror", dient hierbei als Szenario, um einen genaueren Kontext der Bewegungsverfolgung zu erreichen [Ghose 2014]. Ghose hat dort beschrieben, wie eine mögliche Nutzung dieses Spiegels am Morgen aussehen kann. Die Interaktion mit dem Spiegel wurde durch Gesten- und Sprachsteuerung umgesetzt. So kann mit Handgesten, während dem Abtrocknen, die E-Mail- oder Terminübersicht geöffnet werden, um sich für anstehende Meetings vorzubereiten.

Das gleiche Anwendungsfeld stellen Flugzeugkabinen dar. So besteht im Vergleich zu einem üblichen Badezimmer nicht viel Platz und man könnte mit Hand- oder Körpergesten bestimmte Elemente bedienen. Dadurch müssten die Armaturen nicht mehr angefasst werden, um sie zu bedienen.

Anhand dieser Szenarien kann nun eine Einschränkung auf das zu verwendende Modell geschlossen werden. So stellt der wichtigste Bereich den Oberkörper und speziell die Hände dar. Dies bedeutet, dass Informationen zu den Gesichtsausdrücken vernachlässigt werden können und im Modell nicht erforderlich sind.

2.3.2 Verwandte Arbeiten

Im Rahmen des Living Places der HAW Hamburg wird seit Jahren mit aktuellen Technologien an Themen der Mensch-Computer-Interaktion geforscht. Im Folgenden wird auf solche Projekte eingegangen und Schnittstellen zu dieser Arbeit gezogen.

Olaf Potratz: Ein Framework für physikbasierte 3D Interaktion mit großen Displays
In der Arbeit von Olaf Potratz [Potratz 2014] wird ein Framework beschrieben, welches mit Hilfe einer Physiksimulation Gesten interpretiert. In Folge dessen werden auf die verschiedenen 2D und 3D Gesten und deren Probleme eingegangen.
Die Architektur des entwickelten Frameworks ist an das MVC-Muster angelehnt und mit einer Observer-Komponente erweitert. Die Gesten werden durch ein ARTtracker-System aufgezeichnet. Desweiteren verwendet Potratz den ActiveMQ als Message-Broker.

Sobin Ghose: Konzeption und Evaluation eines interaktiven Badezimmerspiegels
Sobin Ghose beschreibt in seiner Arbeit die Konzeption eines „Smart Mirrors", welcher durch Sprach- sowie Gestensteuerung bedienbar ist [Ghose 2014]. Ghose geht dort auf die verschiedene Arten der Gesten ein und beschreibt die Steuerung des Spiegels mit diesen. Um die Steuerung des Spiegels durch Spracheingaben zu gewährleisten, wurde eine Grammatik entworfen. Ghose hat bei der Entwicklung auf den Einsatz mehrerer Sensoren gesetzt, welche eigenständig Daten liefern und getrennt bearbeitet werden.
Die gewählte Architektur ähnelt dem MVC-Entwurfsmuster und ein Message-Broker dient zur Kommunikation der verschiedene Module. Die gewählten Sensoren sind zum Einen die LeapMotion um Fingergesten zu erkennen und zum Anderen die Kinect von Microsoft um Hand- sowie Armbewegungen zu registrieren.

Hillebrand & Tschackert: Evaluierung kamerabasierter Tracking-Ansätze für Motion Capture Applikationen
Thorsten Hillebrand und Patrick Tschackert vergleichen in ihrer Arbeit verschiedene kamerabasierte Ansätze, um Bewegungen aufzuzeichnen (vgl. [Hillebrand & Tschakert 2008]). Dabei nehmen sie das Szenario des Motion Capture gestützten Muskelaufbautrainings, um ein effizientes System zu entwickeln. Es wird zum Einen die Microsoft Kinect und zum Anderen ein ARTtracking-System genauer betrachtet. Desweiteren beschreiben sie eine Möglichkeit, die beiden Messergebnisse miteinander zu verrechnen und somit die Vorteile der Nutzung von mehreren Sensoren in ihr System mit einzubauen.

Johann Heitsch: Ein Framework zur Erkennung von dreidimensionalen Gesten
In der Bachelorthesis von Johann Heitsch wird ein Framework erarbeitet, auf dessen Basis Gestenerkennungsmaschinen realisiert werden können (vgl. [Heitsch 2008]). Es wird auf die verschiedenen Klassen der Gesten und auf verschiedene Arten des Motion Trackings eingegangen. Daraus spezifizieren sich Anforderungen an ein Framework, welches eine Gestendatenbank durch erkannte Gesten erstellt und diese dadurch wiedererkennbar gemacht werden sollen.

3 Design & Realisierung

In diesem Kapitel wird auf das Grundkonzept der Software eingegangen und aufgezeigt, weshalb man sich für die verwendete Architektur entschieden hat. Bei dem Design und der Realisierung wurde darauf geachtet, dass die jeweiligen Komponenten möglichst unabhängig voneinander und selbstständig arbeiten können. Es wurde auf eine lose Kopplung gesetzt, um die Austauschbarkeit und Erweiterbarkeit verschiedener Komponenten zu gewährleisten.

3.1 Das Gesamtkonzept

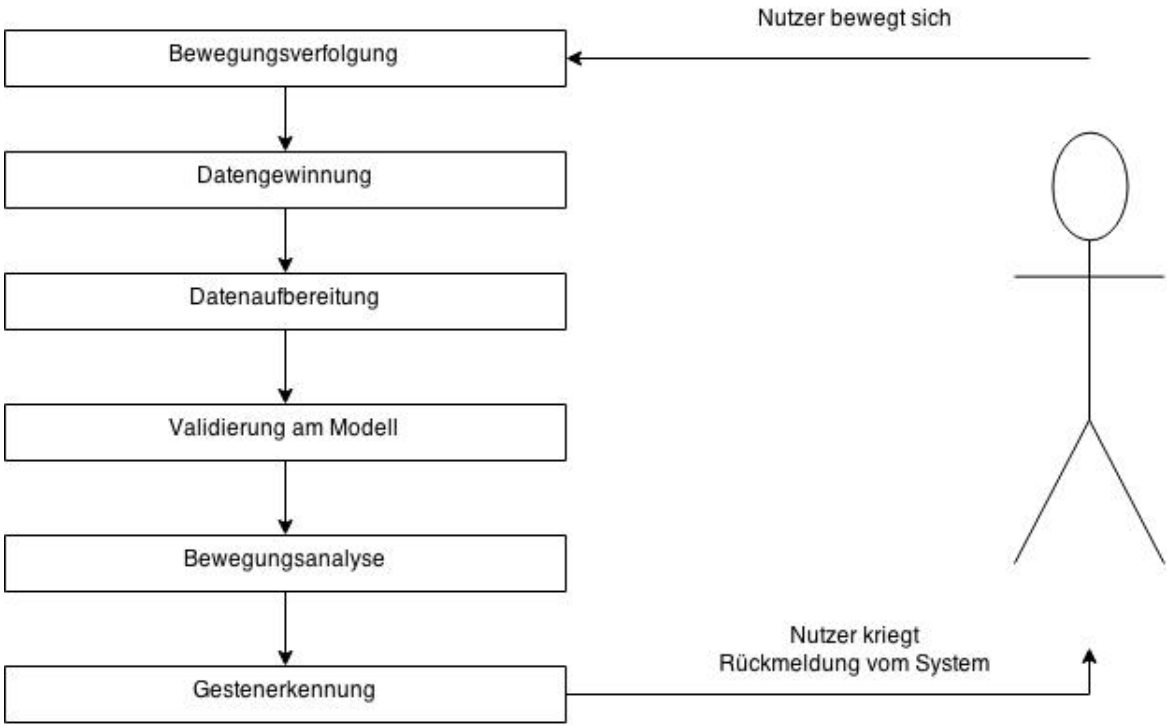


Abbildung 3.1: Gesamtübersicht

Das Gesamtkonzept besteht aus allen Abläufen, die zur Gestenerkennung benötigt werden. Wie im Analysekapitel beschrieben ist werden mit Hilfe von Kameras Bewegungen aufgezeichnet, diese dann von den jeweiligen Sensoren vor verarbeitet und von einem System nach weiterer Aufbereitung an einem Modell verbunden und abgebildet. Im letzten Schritt wertet dann eine weitere Software die fusionierten Daten aus, um Gesten erkennen und deuten zu können.

Bewegungsverfolgung: Bei der Datenerfassung, werden von Kameras die zu verfolgende Person erfasst und kontinuierlich aufgenommen. Dabei wird eine simple Vorverarbeitung der Daten durch die Treiber der Kameras vorgenommen. So werden etwa Bilder des Körpers von Hintergrundinformationen getrennt.

Datengewinnung: Die Datengewinnung beschreibt die erstmalige Aufbereitung von Daten. Es werden die API's der Kameras angesprochen und die Rohdaten, die diese bereitstellen, für die weitere Verarbeitung bereitgestellt.

Datenaufbereitung: Die bereitgestellten Rohdaten werden verarbeitet und zur kommenden Validierung am Modell vereinheitlicht. Hier werden erste Informationen über die Bewegungen gewonnen.

Validierung am Modell: Die Daten werden in das lokale Koordinatensystem des Modells umgewandelt und an einem Modell abgebildet. Die Verwendung einer Physik-Engine ermöglicht eine erste Validierung der aufgezeichneten Daten. So werden „False Positives", die physikalisch nicht möglich sind, gefunden und im weiteren Verlauf ignoriert. Nach der Darstellung werden die Daten über die momentane Position aller Knochen zusammengeführt und zur weiteren Analyse bereitgestellt.

Bewegungsanalyse: Dieser Schritt beschreibt die Analyse der validierten Daten der Bewegung einer Person. Es wird auf bestimmte Muster von Bewegungen geachtet, welche Gesten beschreiben.

Gestenerkennung: Sobald die Bewegungsanalyse ein Muster von Bewegungen gefunden hat, wird in einem bestimmten Kontext korrekt von dem System reagiert. Die Gestenerkennung führt somit zur Steuerung eines Systems oder Progamms.

Als Metapher eignet sich hierbei die Interaktion mit einem Badezimmerspiegel. Der feste Abstand zu den Sensoren erleichtert die Eingrenzung des Interaktionsbereiches erheblich, da der Nutzer nicht von den Kameras gefunden werden muss, wenn dieser sich direkt vor dem Spiegel befindet. Aufgrund des festen Interaktionsbereiches kann eine Eingrenzung des Gestenrepertoires erfolgen, denn die Interaktion erfolgt nur mit Hilfe des Oberkörpers. Ein ähnliches System hat Sobin Ghose in seiner Arbeit „Konzeption und Evaluation eines interaktiven Badezimmerspiegels" (vgl. [Ghose 2014]) realisiert. Der folgende Abschnitt bezieht sich auf diese Arbeit.

Ein interaktiver Spiegel

Ghose stellt in seiner Bachelorthesis einen interaktiven Spiegel vor, welcher mit Hilfe verschiedener Sensoren, die kontinuierlich Bewegungen aufzeichnen, bedient werden kann. Der Spiegel unterstützt Gestensteuerung sowie auch eine Sprachsteuerung. Um dies zu ermöglichen, wurden folgende Komponenten implementiert:

- Kinect-Adapter: Der Kinect-Adapter nimmt die Daten der Kinect auf, führt eine Gestenerkennung durch und sendet die Daten in einem vordefinierten Format weiter. Zusätzlich wird über das Mikrofon auf Spracheingaben gewartet. Diese werden dann auf Grundlage einer von Ghose entwickelten Grammatik erkannt und an den Sprachinterpreter gesendet.
- Leap-Motion-Adapter: Der Leap-Motion-Adapter nimmt die Daten der Leap-Motion auf und sendet, wie der Kinect-Adapter, die Daten nach einer Gestenerkennung in einem bestimmten Format weiter.
- Sprachinterpreter: Diese Anwendung verfügt über einen Befehlssatz. Die Daten der Spracheingabe werden zu eindeutigen Befehlkommandos umgewandelt und an die GUI geschickt.
- Grafische Benutzeroberfläche (GUI): Die grafische Oberfläche ist für die Darstellung der Inhalte zuständig. Es werden Befehle im aktuellen Kontext überprüft und der Nutzer erhält über diese Oberfläche eine visuelle Rückmeldung des aktuellen Standes des Systems.

Als architektonisches Muster des Systems wurde das Blackboard-System (vgl. [Ellenberg u. a. 2011]) gewählt. Mit Hilfe des Blackboards können verschiedene Dienste Daten anbieten und abonnieren, ohne dabei die anderen Dienste zu kennen. Ghose implementierte die verschiedenen

Komponenten des Spiegels als Dienste, die untereinander über das Blackboard kommunizieren (vgl. [Ghose 2014]).

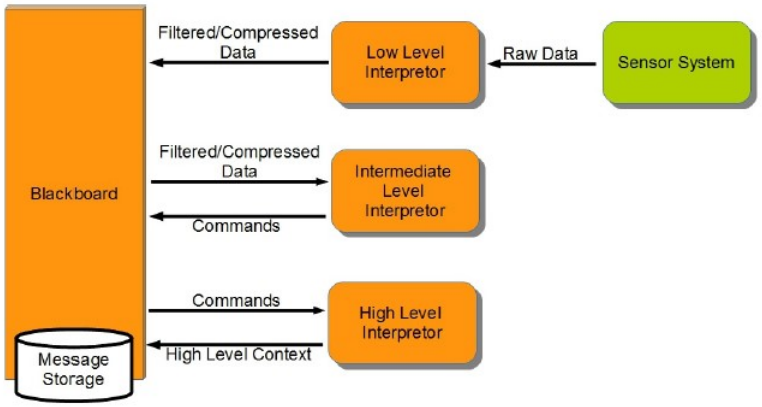


Abbildung 3.2: Blackboard - Interpretationslevel von Sobin Ghose
Quelle: [Ghose 2014]

In Abbildung 3.2 erkennt man, dass Ghose verschiedene Stufen der Interpretation implementierte, um das System als weiteres System innerhalb des Living Places hinzuzufügen. Konkret bedeutet dies, dass die Sensorauswertungen von dem Low-Level-Interpretor auf das Blackboard geschrieben werden, welche dann nach weiterer Intepretation als Befehle ans Blackboard geliefert werden. Die Benutzeroberfläche greift dann auf diese Befehle zu und wertet diese bezüglich des aktuellen Anwendungsfalles aus (vgl. [Ghose 2014]).

Wie im Komponentendiagramm (Abbildung 3.3) sichtbar ist, werden die beiden Sensoren getrennt verarbeitet und in einen gemeinsamen Data Cache gesendet. Somit arbeiten die beiden Adapter nur für sich und es findet keine Auswertung der Gemeinsamkeiten oder Abweichungen statt. Es wäre effektiver, wenn die verschiedenen Sensoren miteinander verrechnet und validiert werden. Die folgenden Kapitel beziehen sich nun auf eine Lösung, mit welcher dies erreicht werden kann.

3.2 Systemarchitektur

In der Gesamtübersicht wurde der gesamte Kontext, in welchen die Software integriert wird, dargestellt und auf ein solches System auch am Beispiel von Ghose's entwickelten Badezimmerspiegels genauer eingangen. Dabei stellen „Bewegungsverfolgung", „Datengewinnung",

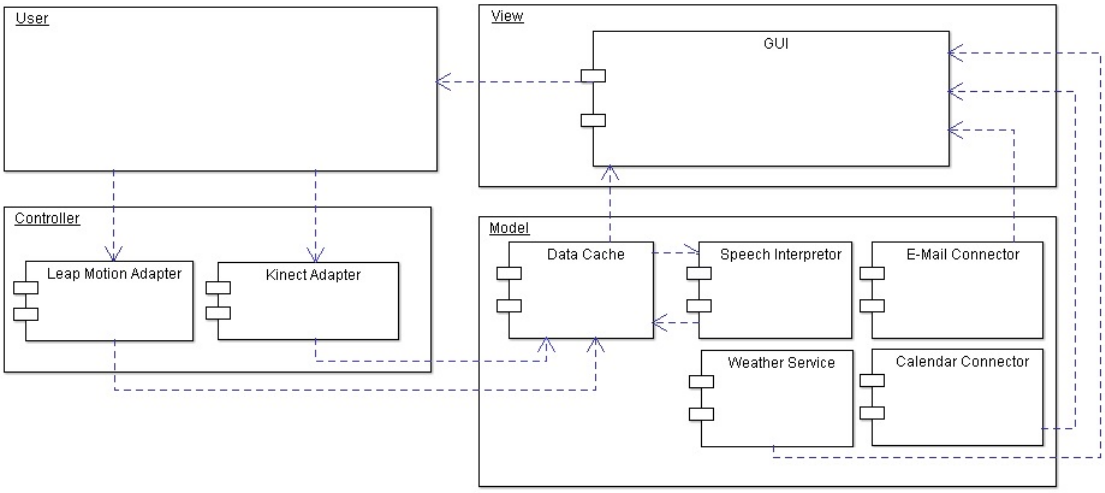


Abbildung 3.3: Komponentendiagramm - Interaktiver Spiegel von Sobin Ghose
Quelle: [Ghose 2014]

„Datenaufbereitung" sowie „Validierung am Modell" die relevanten Punkte bei der Sensorfusion dar. Genauer kann von zwei Modulen unterschieden werden (Abbildung 3.4):
Das erste Modul beschafft sich Informationen zu den Bewegungen des Nutzers und übergibt diese nach kurzer Vorbearbeitung der Middleware. Das zweite Modul greift sich diese Informationen ab und verbindet sie an einem gemeinsamen Modell miteinander. Desweiteren werden diese gebündelten Informationen dann zur Weiterverarbeitung für weitere Systeme bereitgestellt. Daraus lassen sich folgende Komponenten ableiten:

- SensorStarter: Diese Komponente stellt den Einstiegspunkt der Software dar. Dort werden alle Module und Komponenten initialisiert sowie gestartet.
- Sensor1-Adapter und Sensor2-Adapter: Die Adapter sprechen die jeweiligen Sensoren an und erhalten Informationen über Bewegungsabläufe des Nutzers. Danach grenzen sie notwendige Daten von, in diesem Kontext, nicht brauchbaren ab und senden diese verpackt zum Middleware-Adapter.
- Middleware-Adapter: Dies ist die Schnittstelle zur Middleware.
- Middleware: Die Middleware hat die Funktion, Daten zuverlässig für die verschiedenen Module bereitzustellen und dabei die Arbeit auf verteilten Systemen zu gewährleisten.

- CalculationModule: Diese Komponenten dient der Verarbeitung der Daten, die mit Hilfe des Middleware-Adapters bereitgestellt werden. Die Daten werden dort aufeinander kalibriert und validiert.
- BodyControl: BodyControl verwendet die berechneten Daten und stellt ein Körpermodell dar, welche die Bewegungen des Nutzers nachahmt.

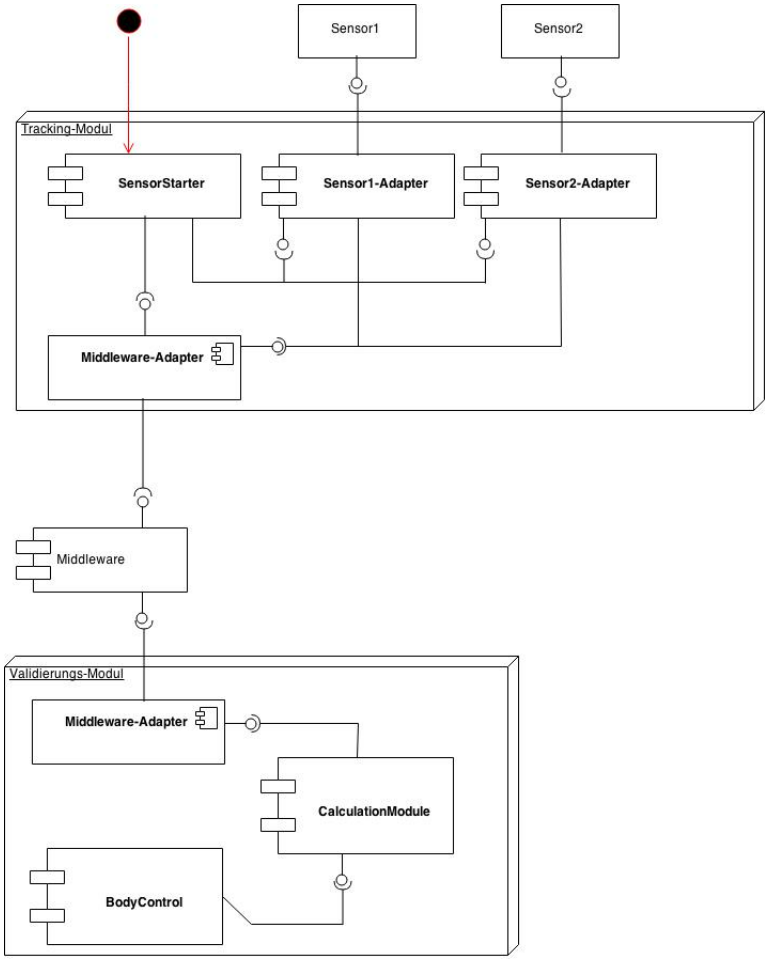


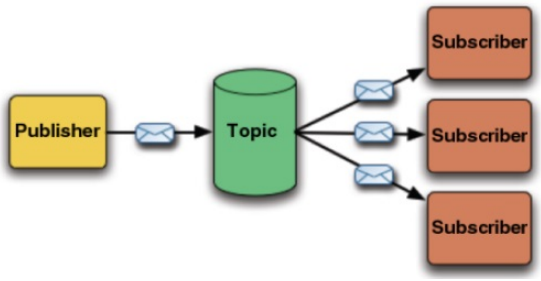
Abbildung 3.4: Systemarchitektur

3.2.1 Kommunikation

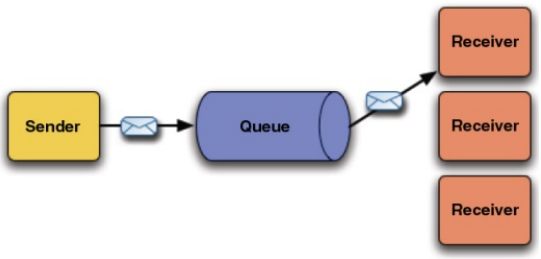
Um die verteilte Arbeit zu gewährleisten, wird ein zentraler Nachrichtenpunkt benötigt, welcher von allen Modulen angesprochen werden kann. Hierfür bietet sich ein Message-Broker-System

an, mit dem Nachrichten zuverlässig untereinander austauschbar gemacht werden. Ein System dafür stellt ein ActiveMQ-Server dar, mit dem über verschiedene Arten eine Kommunikation verschiedener Dienste oder Module erreicht wird:

- Publish-Subscriber: Bei dem Publish-Subscriber-Muster abonnieren Clients ein bestimmtes Thema (Topic). Sobald eine Nachricht auf das Thema publiziert wird, erhalten alle Abonnenten diese Nachricht und können diese verarbeiten.
- Producer-Consumer: Anders als zum Publish-Subscriber-Muster wird bei dem Producer-Consumer-Muster von einem Sender, Nachrichten zu einem Empfänger, mittels Nachrichtenqueue, gesendet. Diese Nachricht kann nur von einem gezielten Empfänger gelesen werden („Point-to-Point Messaging").



(a) Publish-Subscribe-Muster



(b) Producer-Consumer-Muster

Abbildung 3.5: ActiveMQ Kommunikationsmuster
Quelle: http://www.straub.as/java/jms/basic.html

Um zu ermöglichen, dass verschiedene Validierungsmodule gleichzeitig die erfassten Daten berechnen, wird auf ein Publish-Subscribe-Muster zurückgegriffen. Innerhalb des „Tracking-Moduls" (vgl Abbildung 3.4) veröffentlicht der Middleware-Adapter die erhaltenen Daten der Sensoren auf jeweils ein Topic. Auf Seiten des Validierungs-Moduls wird dann die Nachricht erhalten und kann weiter verarbeitet werden. Ein gemeinsames Nachrichtenformat dient zur korrekten Auswertung der Nachrichten.

```
{
        Attribut1: Wert,
        Attribut2: Wert
}
```

Abbildung 3.6: Beispielcode JSON

Konfiguration des ActiveMQ-Servers

Ein Problem, das ActiveMQ aufweist, ist, dass intern mit Nachrichtenschlangen die Nachrichten behandelt werden. Dies ist ein Problem, sobald der Empfänger mit einer geringeren Arbeitsrate als der Sender arbeitet. Ein Nachrichtenstau hat zur Folge, dass das komplette System einen starken Delay aufweist und nicht zeitnah reagieren kann. Um dies zu umgehen, kann mit bestimmten Konfigurationen innerhalb des ActiveMQ-Servers dazu geführt werden, dass keine Nachrichten anstehen.

Dazu bietet Apache die „constantPendingMessageLimitStrategy" an, die aussagt, wieviele Nachrichten maximal anstehen können. Durch die manuelle Anpassung dieses Wertes auf „1" wird somit nur maximal eine Nachricht in der Queue des Servers gespeichert. Sobald eine neue Nachricht von dem Sender publiziert wird, wird die offene Nachricht mit der neuen ersetzt. Weitere Informationen dazu kann von der dazugehörigen Seite der Dokumentation entnommen werden [1] .

Das Nachrichtenformat JSON

Die JavaScript Object Notation (kurz: JSON) ist ein einfach lesbares und kompaktes Datenformat, welches zum Datenaustausch verwendet wird. Es werden dabei einfache Attribute mit Werten versehen und verpackt (vgl. Abbildung 3.6). Als weitere Option wurde XML in die nähere Auswahl gezogen. XML ist im Vergleich zum JSON-Format eine Auszeichnungssprache, hat einen weitaus höheren Einsatzbereich und ist vielseitiger. Jedoch ist vorab ein höherer Aufwand notwendig, um Attribute und Werte zu spezifizieren, welcher sich nur bei sehr großen Datenmengen bezahlt macht und daher für dieses Projekt ungeeignet ist.

3.2.2 Kopplung

Die Stärke der Kopplung der einzelnen Module untereinander spielt eine wichtige Rolle für die Wiederverwendbarkeit und Austauschbarkeit der Komponenten im System. Jedes System

[1] http://activemq.apache.org/slow-consumer-handling.html

strebt eine lose Kopplung an, um dies zu ermöglichen. Ein klassisches Architekturmuster stellt das Model-View-Controller-Muster (MVC-Muster) dar (vgl. Abbildung 3.7). Hierbei handelt es sich um eine Unterteilung der Komponenten in drei logische Bereiche:

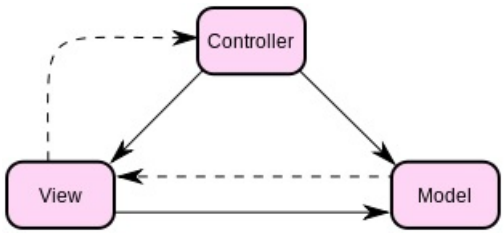


Abbildung 3.7: Model-View-Controller Muster
Quelle: http://de.wikipedia.org/wiki/Model_View_Controller

- Model: Das Model (Deutsch: Modell) beinhaltet alle Komponenten, die sich um die Bearbeitung der Daten kümmern. Dort werden Berechnungen durchgeführt, um die Daten für die Präsentation vorzubereiten. Die Daten werden vom Controller bereitgestellt.
- View: Die View (Deutsch: Präsentation) dient der Darstellung von Daten. Sie greift dabei auf die Model-Komponenten zu (z.B. auf eine Datenbank) um diese zu erhalten. Hauptfunktion ist es, dem Nutzer ein Feedback geben zu können.
- Controller: Der Controller (Deutsch: Steuerung) verarbeitet Nutzereingaben und gibt diese an das Model zur Verarbeitung weiter. Es sollte darauf geachtet werden, dass im Controller nur einfache Vorverarbeitungen stattfinden, sodass die Weiterleitung der Daten kein Latenzproblem erzeugt.

Wie in Abbildung 3.8 zu sehen ist, wurde an ein MVC-Muster angenähert, um eine lose Kopplung der einzelnen Komponenten zu erhalten. Entscheidend dabei ist, dass die jeweiligen Sensoren voneinander unabhängig arbeiten.

Controller: Im Controller befinden sich die Adapter der Sensoren, um die Bewegungen des Nutzers aufzuzeichnen. Die beiden Adapter wissen nichts voneinander und senden mittels des Middleware-Adapters (vgl. Abbildung 3.4) die vorverarbeiteten Daten weiter.

Model: Im Model werden diese Daten dann aufgegriffen und der DataCache wird mit den aktuellen Daten befüllt. Es wird eine Kalibrierung gestartet, um die beiden Sensoren aufeinander

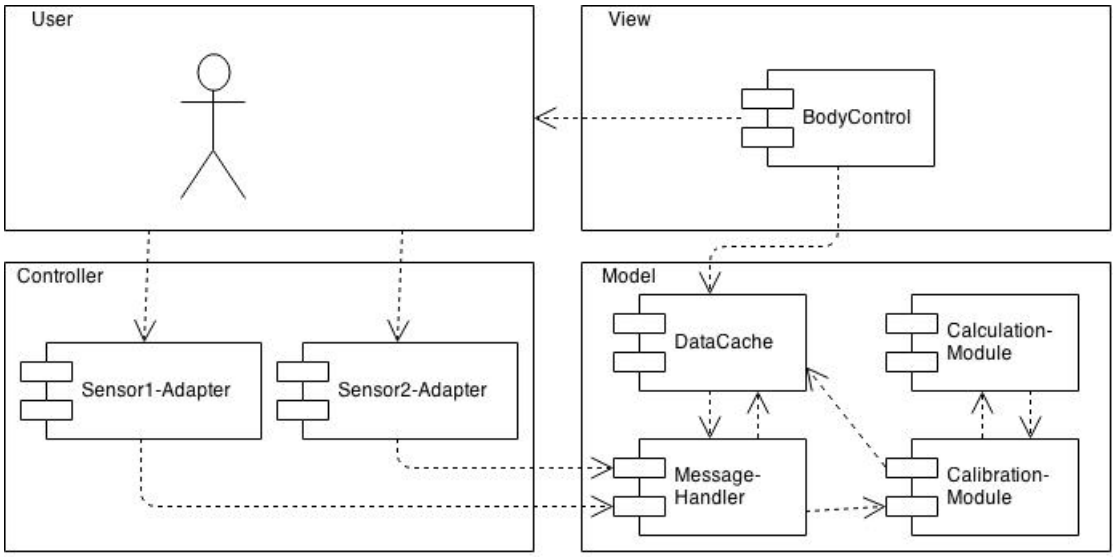


Abbildung 3.8: Komponentendiagramm

abzustimmen und sie somit zu einem Sensor zu vereinigen.

View: In der View wird nun auf den DataCache zugegriffen und die dort vorhandenen Daten für den Nutzer in einem Modell abgebildet. Optional kann die Darstellung bei der Verwendung ausgelassen werden, um Ressourcen für andere, weitere Berechnungen freizugeben.

3.3 Motiontracking

Die Datenerfassung beschäftigt sich mit der Erlangung der Daten über die Bewegungen einer getrackten Person. Im Analysekapitel wurde dabei schon auf die verschiedenen Arten zur Bewegungsverfolgung eingegangen. Aufgrund des in dem Szenario beschriebenen Platzmangels im Badezimmer wird auf eine Verwendung von optisch markerbasierten Sensoren verzichtet, da man diese vor der Benutzung des Smart Mirrors an seinen Körper anbringen müsste und somit das Konzept dieses Systems zerstört wäre.

Da man auf die Bewegung von Körper und speziell der Hände eingeht, wurden unterschiedliche Sensoren verwendet, die im Folgenden genauer betrachtet werden. Es wurde darauf geachtet, dass diese sich gegenseitig nicht beeinflussen, um die Effizienz der Software zu gewährleisten.

3.3.1 LeapMotion

Die LeapMotion ist ein kleines USB-Gerät, welches mit 2 Infrarot Kameras und 3 Infrarot LED's arbeitet. Sie ist darauf auslegt, Finger- sowie Handbewegungen zu erkennen, um innerhalb einer Software zu navigieren. Dafür werden beide Hände und alle Finger in einem Bereich von 30*30*30 Zentimeter erfasst. Mit einer Framerate von bis zu 130 Frames pro Sekunde werden die Bilder an den Computer gesendet und dort von der LeapMotion-Software ausgewertet. Ein starker Vorteil bei der Benutzung der LeapMotion ist es, dass über mehrere Frames hinweg nach vordefinierten Gesten gesucht wird und somit erkannt werden können.

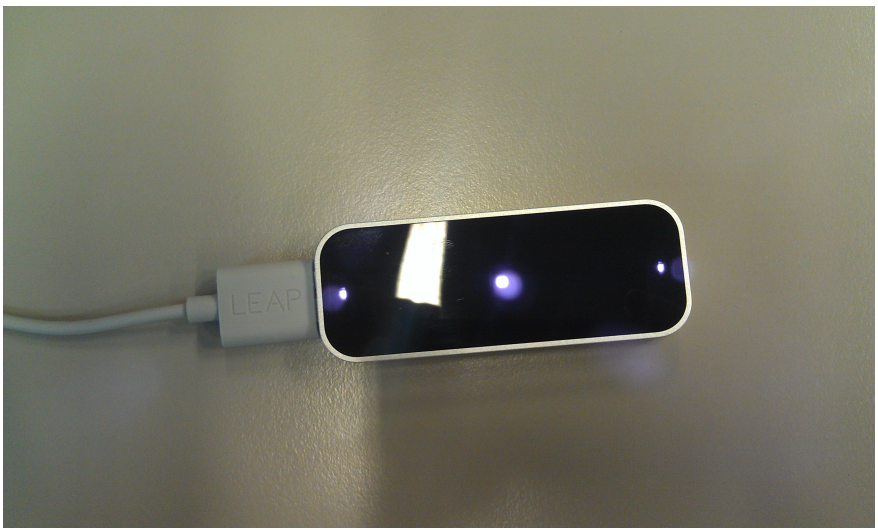


Abbildung 3.9: LeapMotion

LeapMotion SDK

Die Benutzung des Software-Development-Kits der LeapMotion wurde von den Herstellern einfach gestaltet. Der Sensor liefert einen Frame, in dem alle erkannten Daten vorhanden sind. Darunter befinden sich alle getrackten Hände mit sogenannten „Pointables". Unter „Pointables" versteht man dabei alle gefundenen Finger sowie mögliche Werkzeuge, die als Eingabehilfe verwendet werden können (z.B. Stifte). Da jedoch ein Körpermodell geschaffen werden soll und Eingabehilfen nicht Teil eines Körpers sind, werden nur gefundene Finger beachtet.
Seit Version 2 des SDK's bietet LeapMotion auch eine Unterscheidung der Fingertypen (Daumen, Zeigefinger, Mittelfinger etc.).

```
[
  {
    "handName": "LeftHand",
    "handPosition": [
      -93.947464,
      151.43161,
      21.23683
    ],
    "handDirection": [
      0.02455736,
      0.262532562,
      -0.9646106
    ],
    "handGrabStrength": 0.0,
    "fingerList": [
      {
        "jointName": "TYPE_THUMB_L",
        "jointPosition": [
          -44.7153664,
          128.745758,
          -15.31006
        ],
        "jointOrientation": [
          0.258961529,
          -0.04584273,
          -0.964799166
        ]
      },
        ...
```

Abbildung 3.10: LeapMotion Grundstruktur der JSON-Nachricht

Json Nachricht der LeapMotion-Daten

Die verwendete Struktur zur Übermittlung der Daten baut sich wie folgt auf: Jede gefundene Hand hat einen Namen, eine Ausrichtung sowie die Information über die Position der Handfläche. Desweiteren gibt eine Dezimalzahl zwischen 0.0 und 1.0 an, wie weit eine Hand geöffnet ist (vgl. „Grabstrength"). Zu jeder Hand gehören Finger, welche in einer Liste in der Hand referenziert werden. Jeder Finger hat Informationen über den Namen (bzw. Fingertyp) sowie über die Ausrichtung und Position der Fingerkuppe.

3.3.2 Microsoft Kinect

Microsoft hat mit der Kinect in Verbindung mit der XBOX 360 eine Infrarotsensor-Kamera auf den Markt gebracht, die Körperbewegungen zur Navigation innerhalb von Spielen verfolgt. Die Microsoft Kinect 1 arbeitet mit Infrarotkameras sowie LEDs und ist in der Lage, mehrere Körper komplett zu tracken (mit einschränkung der Hände). Um eine Aussage über Position und Ausrichtung von Gelenken treffen zu können, werden dabei verschiedene Bilddaten miteinander verrechnet. Bei der gemeinsamen Nutzung von LeapMotion und der Microsoft Kinect wurde festgestellt, dass diese sich untereinander beeinflussen und keine genauen Datensätze liefern.

Mit der Kinect 2, die im Sommer 2014 offiziell erhältlich wurde, veröffentlichte Microsoft eine verbesserte Version der Kinect 1, welche mit einer Time of Flight-Kamera arbeitet und somit die gemeinsame Nutzung von LeapMotion und Kinect ermöglicht. Zusätzlich werden nun auch genauere Informationen zu den Händen bereitgestellt (Daumen, Fingerspitzen sowie die Handfläche).

Abbildung 3.11: Kinect 2 Pre-Release

Kinect SDK

Das SDK der Kinect1 bietet verschiedene „Streams" an, welche unterschiedliche Daten bereitstellen. Seit der Einführung der Kinect 2 und dessen SDK-Version werden sogenannte „Frames" angeboten, die die gleiche Funktion der „Streams" der Kinect 1 SDK haben. Der „BodyFrame" liefert dabei alle Informationen, die über getrackte Körper vorhanden sind. Darunter Positionsdaten zu allen erkannten Gelenken, Handposen (vgl. 'HandState') oder auch Informationen, ob die Augen geschlossen oder offen sind. Da dies, eine riesige Menge von Informationen

sein kann, wurde eine Struktur erstellt, die für das Modell relevante Daten enthält. Durch das Szenario kann davon ausgegangen werden, dass es nur einen Benutzer des Systems gibt. Um aber, sofern sich zwei Personen vor der Kinect befinden, valide Daten zu erzeugen, werden die Daten des ersten erkannten Nutzers verwendet und die Daten der restlichen Nutzer verworfen.

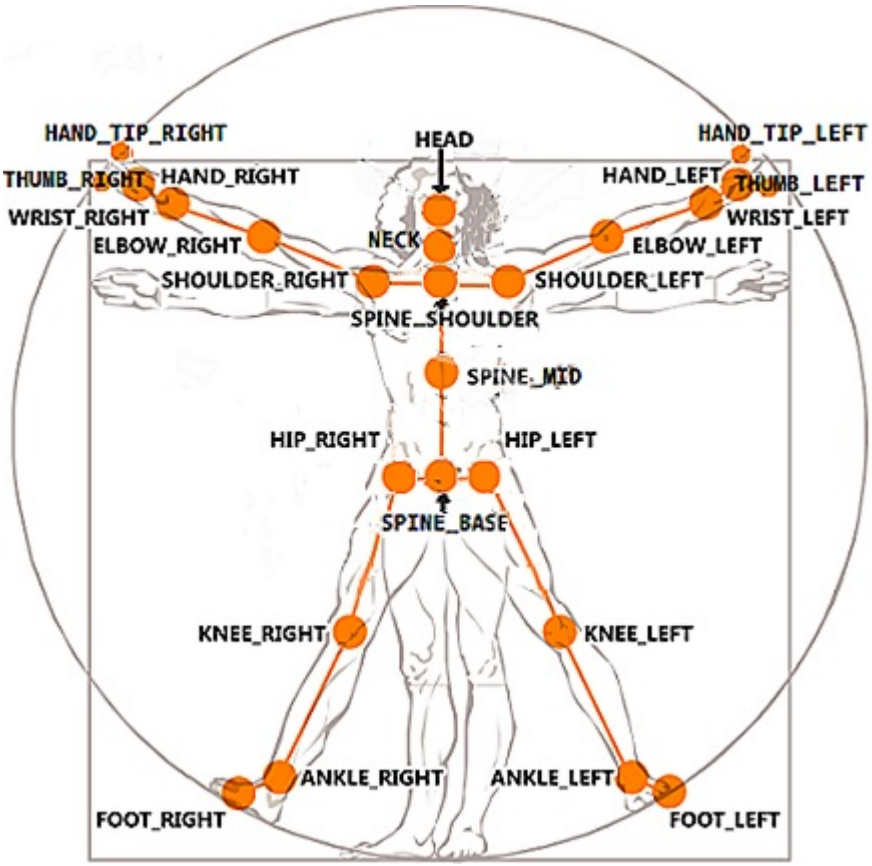


Abbildung 3.12: Kinect 2 Skelett-Struktur
Quelle: https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

Json Nachricht der Kinect-Daten

Um auf Modellseite losgelöst von Kinect-eigenen Datentypen zu sein, wurde eine Struktur erstellt, die möglichst kompakt alle notwendigen Körperdaten beinhalten kann. Abbildung 3.12 zeigt die von Microsoft Kinect entwickelte Skelett-Struktur, an die sich angelehnt wurde. Zu dem ersten getrackten Körper wird eine Liste von Gelenkinformationen bereitgestellt, welche

die Position und Ausrichtung des Gelenks enthält. Desweiteren werden Informationen zu den Handposen mitgeliefert. (vergleiche Abbildung 3.13).

```
[
  {
    "handLeftState": "Unknown",
    "handRightState": "Open",
    "bodyJoints": [
      {
        "jointName": "SpineBase",
        "jointPosition": [
          0.0883123949,
          -0.5621422,
          1.2445972
        ],
        "jointOrientation": [
          -0.0342487022,
          0.9600914,
          0.153577566,
          -0.2312259
        ]
      },
        ...
```

Abbildung 3.13: Kinect 2 Grundstruktur der JSON-Nachricht

3.4 Validierung am Modell

Im nächsten Schritt wird nun ein Modell benötigt, das die Informationen über die Körperbewegung sammelt und darstellt. Zusätzlich sollen bestimmte Vorraussetzungen erfüllt werden. Es soll:

- die Daten deutlich darstellen.
- performant sein.
- keine hohe Latenz aufweisen.

Um die Latenz gering zu halten, sollte das Modell desweiteren nicht weit von der Struktur der Daten entfernt sein, sodass zusätzliche Umrechnungen und somit Zeit gespart werden können.

Vorerst gilt es aber, eine geeignete Plattform zu finden, auf der eine Sensorfusion sowie die Darstellung leicht und effizient realisiert werden kann. Um Modelle darzustellen, gibt es mehrere Möglichkeiten. Zum einen bietet sich eine rein optische Darstellung via OpenGL an. OpenGL ist eine plattformunabhängige und performante Variante, 3D Computergrafiken zu erstellen. Mit OpenGL lassen sich einfache Objekte darstellen und verarbeiten, jedoch steht die teilweise veraltete API und die nicht vorhandene Möglichkeit, einfache Knochenabhängigkeiten zu modellieren, dieser Variante im Weg.

Eine weitere Möglichkeit ist es, eine Darstellung mit Hilfe einer Spielengine zu realisieren. Eine Spielengine ist eine Entwicklungsumgebung, die zur Erstellung von Computerspielen verwendet wird und daher auf den Umgang mit 3D-Objekten ausgerichtet ist. Solche Plattformen verfügen darüber hinaus über eine implementierte Physikengine, welche physikalische Berechnungen für Objekte anbietet. Unity3D ist eine solche Spielengine und sticht heraus, indem die Plattform mit anderen Programmen, die zur Modellgestaltung verwendet werden, verbunden werden kann.

3.4.1 Versuch 1: Blendermodell

Der erste Versuch, beide Sensoren miteinander zu verbinden, wurde anhand eines Blendermodells umgesetzt. Das Programm „Blender" bietet die Möglichkeit, Körpermodelle zu gestalten und direkt in Unity zu importieren. Dafür wird, wie in dem Analysekapitel beschrieben, ein Polygonnetz erstellt und durch Rigging sowie Skinning eine Grundlage geschaffen, dieses Modell beweglich zu machen.

Es wurde eine körpernahe Struktur gewählt, die alle Knochen beinhaltet, die von der Microsoft Kinect sowie der Leap Motion angesprochen werden. Desweiteren, kann innerhalb von Blender bestimmte Restriktionen angegeben werden. So kann ein Ellenbogen nicht überdehnt werden oder andere Knochen nicht zu stark gedreht werden. Auch wurden bestimmte Knochenabhängigkeiten festgelegt, die sicherstellen sollen, dass physikalische Gegebenheiten eines Menschens (zum Beispiel die Tatsache, dass ein Arm immer mit der Schulter verbunden bleibt) nicht aufgelöst werden.

Zu Beginn eines jeden Trackings, muss der Körper des Modells auf den zu trackenden Körper skaliert werden. Da es auch zum Fehlerfall der Sensoren kommen kann und somit bestimmte Knochen in einem Frame nicht erkannt werden, muss darüber hinaus ein Algorithmus geschrieben werden, welcher ungefähre Positionen für nicht getrackte Knochen berechnet. Dies stellt dann sicher, dass bestimmte Abhängigkeiten von Knochen (wie zum Beispiel die des

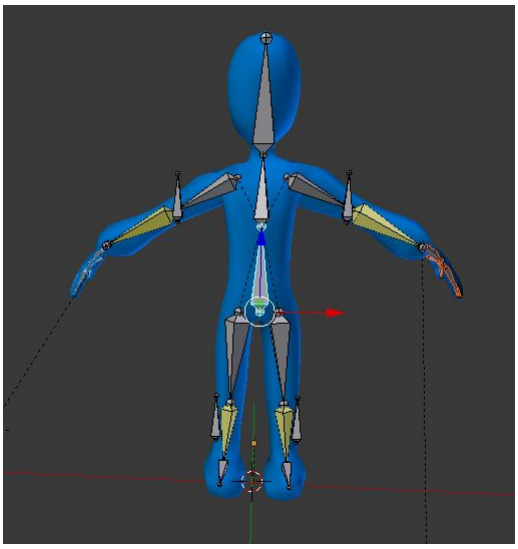


Abbildung 3.14: Blendermodell

Unterams und des Oberarms) im Falle eines Nichterkennens nicht verletzt werden.

Kinect sowie LeapMotion liefern Gelenkdaten. In dem Blendermodell werden jedoch Anfangspunkte von Knochen sowie ihre Länge verwendet. Damit die Daten der Sensoren benutzt werden können, müssen zusätzliche Berechnungen eine Knochenlänge ermitteln. Diese Berechnungen kosten Zeit und Aufwand, die rein für die Darstellung keinen Mehrwert ergeben. Somit entschied man sich gegen diese Form der Darstellung.

3.4.2 Versuch 2: Jointdarstellung

Eine weitere Möglichkeit, Gelenke und Verbindungen innerhalb von Unity darzustellen, ist es, einfache Objekte wie Würfel zu verwenden und diese mit einem „LineRenderer" zu verbinden. Das gewählte Datenmodell, um die Informationen zu sammeln, orientiert sich dabei an der Vorgehensweise von der Microsoft Kinect. Es besteht eine Liste von Gelenken, welche alle beinhaltet, die von der Kinect sowie der LeapMotion erkannt werden können.

Abbildung 3.15 zeigt einen Auszug aus dieser Liste. Die Liste wird durchnummeriert, um sie ordentlich durchlaufen zu können. Jetzt muss eine Verbindung der Gelenke erreicht werden. Dazu wird ein Verzeichnis erstellt, welches die Abhängigkeiten der Gelenke festlegt (vgl. Abbildung 3.16). Jeder Eintrag stellt somit einen Knochen im Modell dar. Durch die Nähe dieser Variante zu den gesammelten Daten der Sensoren erübrigen sich große Umrechnungen.

```
public enum JointType
{
        SpineBase = 0,
        SpineMid = 1,
        Neck = 2,
        ...
        ...
        TYPE_INDEX_R = 29,
        TYPE_THUMB_R = 30
}
```

Abbildung 3.15: Auszug aus der Gelenkliste

Somit ist diese Variante zeitsparender als die Variante von Versuch 1. Es müssen lediglich die Informationen der Sensoren zu den passenden Gelenken gemappt werden.

```
public Dictionary<JointType, JointType> _BoneMap
{
        { JointType.FootLeft, JointType.AnkleLeft },
        { JointType.AnkleLeft, JointType.KneeLeft },
        { JointType.KneeLeft, JointType.HipLeft },
        { JointType.HipLeft, JointType.SpineBase },
        ...
        ...
}
```

Abbildung 3.16: Auszug aus der Knochenliste

Ein Algorithmus durchläuft dann diese Liste, ruft zu jedem Gelenk den aktuellen Stand ab und passt die Position sowie die Ausrichtung neu an. Ein Vorteil dabei ist, dass keinerlei Skalierungen der Knochenlängen vor Benutzung benötigt werden. Bei dieser Variante werden die Positionen der Gelenke in Unity projiziert, weshalb die physikalische Machbarkeit nicht auf Weiteres getestet werden kann.

3.5 Mapping am Modell

Als nächster Schritt müssen nun die vorhandenen Daten auf das obige Modell gemappt werden. Durch Verwendung einer Struktur, die auf den voneinander getrennten Modulen bekannt ist, vereinfacht sich die Zuweisung der jeweiligen Gelenke. Die Anpassung und Verarbeitung der beiden Sensoren unterlaufen einigen Schritten, die gewährleisten, dass die resultierenden Daten valide sind.

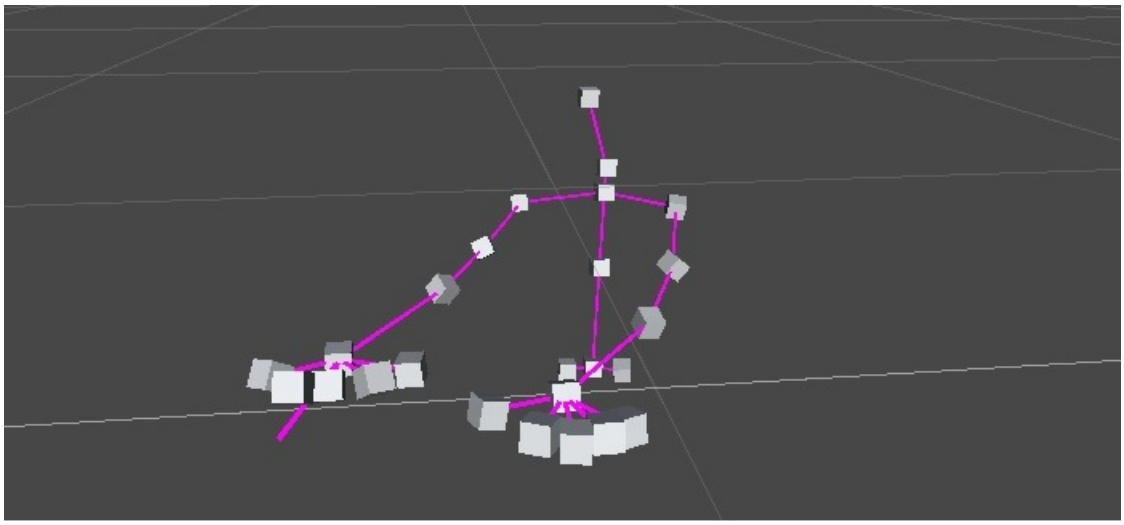


Abbildung 3.17: Jointdarstellung innerhalb des Programmes Unity

Zuerst müssen die Sensoren aufeinander skaliert und kalibriert werden. Danach erfolgt eine Verrechnung der gemeinsamen Daten, die die Sensoren liefern. Dabei werden durch einen gewählten Filter bestimmte Kriterien berücksichtigt. Zuletzt werden dann die resultierenden Daten validiert und zur Darstellung und Weiterverarbeitung freigegeben.

3.5.1 Kalibrierung der Sensoren

Damit mehrere Sensoren nach außen hin als ein einziger Sensor erscheint müssen die Sensoren aufeinander abgestimmt werden. Diesen Vorgang nennt man Kalibrierung. Bei der Kalibrierung zweier Sensoren wird eine Abweichung des einen Sensors zum anderen gemessen. Ein Sensor stellt dabei ein „Normal" dar, während der andere auf dieses Normal angepasst wird. In den Szenarien wird deutlich, dass der Kinect Sensor der primäre Sensor ist, da dieser dauerhaft Daten über die Körperbewegungen sammelt. Die LeapMotion hingegen kann nur bei einer bestimmten Stellung der Hände Informationen liefern. Somit wurde die Kinect als „Normal" gewählt und Daten der LeapMotion entsprechend angepasst.

Eine Kalibrierung zweier frei beweglicher Sensoren erwartet bestimmte Voraussetzungen. Da die beiden Sensoren im Normalfall keinen bestimmten Abstand zueinander haben, wird vorausgesetzt, dass von beiden Sensoren Informationen zu gemeinsamen Gelenken geliefert werden. Nach einer Kalibrierung sollte kein Sensor verschoben werden, um Fehldaten

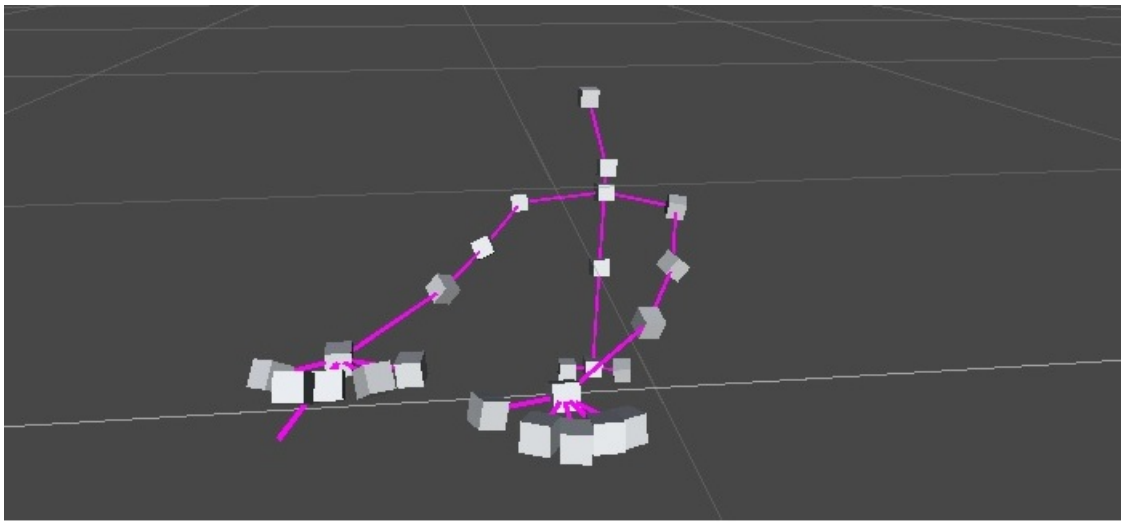


Abbildung 3.18: Jointdarstellung innerhalb des Programmes Unity

zu vermeiden. Sollte dies jedoch passieren, so muss zuerst eine neue Kalibrierung per Programmneustart erfolgen. Um die Kalibrierung zu starten, muss der Nutzer sich vor der Kinect positionieren und beide Hände ausgestreckt, mit der Handfläche nach unten, über die LeapMotion halten. Sobald beide Sensoren die Handflächen gefunden haben, startet nach einigen Sekunden die Kalibrierung. Der Delay von 60 Frames wurde gewählt, um sicher zu stellen, dass beide Sensoren korrekte Werte zu den Positionen der Handflächen liefern.

Bei der gewählten Kalibierungsmethode werden von beiden Sensoren Informationen zu beiden Handflächen ausgewertet und miteinander verrechnet. Zunächst müssen die Sensoren aufeinander skaliert werden, da die Kinect ihre Daten in Metern und die LeapMotion in Millimetern liefert. Nach der Skalierung werden von beiden Sensoren die Punkte der Handflächen ermittelt und voneinander subtrahiert. Daraus entstehen zwei Vektoren, einen für jede Hand, die die Differenz der beiden Sensoren darstellen und auf die Daten der LeapMotion zur Kalibrierung angewendet werden. Zu berücksichtigen ist, dass die Skalierung der LeapMotion-Daten vor der Verrechnung des Vektors stattfinden muss.

Diese Art der Kalibrierung setzt jedoch voraus, dass die Koordinatensysteme beider Sensoren parallel zueinander stehen. Eine Möglichkeit dies zu umgehen ist, eine Kalibrierung über hunderte Frames auszuwerten. Innerhalb dieser Frames sollten beide Hände in alle möglichen Richtungen bewegt werden, um Informationen zur Ausrichtung der Sensoren zu gewinnen und anhand dieser die Kalibrierung zu berechnen.

3.5.2 Kalman Filter

Nach der Kalibrierung der Daten befinden sich alle getrackten Gelenke in einem einheitlichen Koordinatensystem. Jetzt kann auf die Vorteile zugegriffen werden, die mehrere Sensoren bieten:

- Verrechnung zweier gemeinsamer Gelenkdaten
- Erstellung eines gesamten Körpers

Durch die Verrechnung der gemeinsamen Daten ergeben sich Mittelwerte der Positionen. Zusätzlich verschafft es Sicherheit, dass, falls ein Sensor ausfällt, die Daten des Körpers für die gemeinsamen Punkte bestehen bleiben. Dies gewährleistet die weitere Benutzung der Software. Um zwei Punkte miteinander zu verrechnen, bieten sich bestimmte Verfahren an, die dies zuverlässig und einfach gestalten. Ein solches Verfahren ist der Kalman Filter, welcher die jeweiligen Standartabweichungen unter Berücksichtigung zieht. Der von Rudolph E. Kalman entwickelte und nach ihm benannte Kalman-Filter, berechnet von mehreren verrauschten Daten einen gemeinsamen geschätzten Ausgangspunkt [Kalman 1960]. Dabei wird von der Annahme ausgegangen, dass Messwerte immer eine bestimmte Ungenauigkeit besitzen. Die Gleichung 3.1 zeigt, wie mit den Daten ein geschätzter Ausgangswert x berechnet wird. Dabei stehen x_1 und x_2 für die Positionsdaten und σ_1 sowie σ_2 für deren Standardabweichung.

$$x = x_1 * (\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}) + x_2 * (\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}) \tag{3.1}$$

Zusätzlich können Gewichtungen für die jeweiligen Sensoren angegeben werden. Mit diesen Gewichtungen, welche in einer Config-File angegeben werden, lassen sich Präferenzen einstellen, um einen Sensor zu bevorzugen. Da die LeapMotion im Vergleich zur Microsoft Kinect für das Handtracking ausgelegt ist und daher ein genaueres Tracking für diese Gelenke bietet, lässt sich mit den Gewichtungen ein stärkeres Vertrauen zu den Daten der LeapMotion aufbauen.

Der Kalman-Filter berrechnet dann den Mittelpunkt unter Berücksichtigung der Standardabweichung und Gewichtung (w_1 und w_2") von zwei Positionsdaten mit folgender Gleichung:

$$x = w_1 * x_1 * (\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}) + w_2 * x_2 * (\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}) \tag{3.2}$$

3.5.3 Validierung der neuen Daten

Eine Validierung der neuen, verrechneten Daten hilft, um mögliche Fehldaten der Sensoren zu erkennen und auszublenden. Sobald die Verrechnung erfolgt ist, wird ein gesamtes Körperdatenmodell aufgebaut. Wie bei der Jointdarstellung des Modells aufgezeigt ist, gibt es eine Liste von Wertzuweisungen zu den jeweiligen Gelenken. Diese Liste wird mit den verrechneten Daten befüllt, sobald die Überprüfung, welche Strecke ein Gelenk seit dem letzten Frame zurückgelegt hat, positiv ausfällt.

Für die Überprüfung wird nun die Liste des vergangenen Frames genutzt und die Positionsdaten des jeweiligen Gelenkes aufgerufen. Anschließend wird die Distanz der beiden Vektoren berechnet. Sollte die Distanz höher als der in der Config-File angegebene Wert, standartmäßig 1 Meter, ausfallen, kann davon ausgegangen werden, dass es sich bei den neuen Daten des Gelenkes um fehlerhafte Informationen handelt. Es werden dann die Daten des vergangenen Frames beibehalten und die Validierung geht zum nächsten Gelenk über.

3.5.4 Weitergabe & Darstellung der Daten

Sobald ein Datenmodell des gesamten Körpers vorhanden ist, greift ein weiteres Script ein, um die Daten für den Nutzer darzustellen. Für jedes Gelenk wird die Liste der Daten durchlaufen und die Position dann dem dazugehörigen GameObject innerhalb von Unity zugewiesen. Anhand der Abhängigkeiten der Gelenke (Ellenbogen -> Schulter) kann nun, falls das nachfolgende Gelenk auch einen Wert besitzt und somit getrackt wurde, eine sichtbare Verbindung der Gelenke erstellt werden. Diese Linie stellt dann einen Knochen dar.

Zusätzlich wird das gesamte Datenmodell zur Weiterverarbeitung von anderen Systemen freigegeben. Dafür wird ein Objekt erstellt, welches an die Struktur der Daten von der Microsoft Kinect angelehnt ist. Neben den Positionsdaten aller Gelenke werden noch andere, wichtige Informationen mitgeliefert, die beide Sensoren anbieten. Darunter fallen:

- die „Grabstrength" beider Hände, welche angibt, zu welchem Grad die Hände geöffnet sind (geliefert von LeapMotion).
- die „HandStates" beider Hände, welche die Grundgesten der Hände angeben, die von der Kinect eigenständig erkannt werden können (geliefert von Kinect).

Abbildung 3.19 zeigt einen Ausschnitt der Nachricht, die nach allen Verarbeitungen an den ActiveMQ-Server auf ein bestimmtes Topic gesendet wird. Das Topic kann in der Config-File konfiguriert werden.

3.6 Fazit

Es wurde eine Möglichkeit aufbereitet mit der verschiedene Sensoren miteinander verknüpft und an einem gemeinsamen Modell validiert werden können. Durch die Wahl der Systemarchitektur, welche an das MVC-Entwurfsmuster angelehnt ist, wurden die Komponenten nach ihrer Logik unterteilt. Informationen zu den jeweiligen Sensoren sind daher nur auf der Controller-Ebene vorhanden und bleiben somit für nachstehende Bearbeitung verdeckt.

Die gewählte Kalibrierungsmethode weist Lücken auf und kann somit nur unter bestimmten Bedingungen, die in Kapitel 3.5.1 erläutert wurden, optimal genutzt werden. Durch die Trennung der Datenverarbeitung von der Datendarstellung kann optional die Darstellung durch eine Abkapslung der Skripte ausgestellt werden. Nach jedem Durchlauf entsteht ein einheitliches Datenmodell, welches alle Informationen beinhaltet, die von den Sensoren erfasst wurden. Das fertige Datenmodell liegt dann in Form einer JSON-Nachricht auf dem ActiveMQ-Server zur Weiterverarbeitung bereit.

```
[
  {
    "bodyJoints": [
      {
        "jointName": "SpineBase",
        "jointPosition": [
          0.0247626714,
          -0.45310533,
          1.12902451
        ],
        "jointOrientation": [
          -0.0264170114,
          0.970452845,
          0.111358091,
          -0.212421179
        ]
      },
      {
        "jointName": "SpineMid",
        "jointPosition": [
          0.02387992,
          -0.236207962,
          1.17966461
        ],
        "jointOrientation": [
          -0.0234840084,
          0.9756957,
          0.112013325,
          -0.186867088
        ]
      },
                ...
                ...
                ...
      }
    ],
    "handLeftState": "Open",
    "handRightState": "Open",
    "handLeftGrabStrength": 0.0,
    "handRightGrabStrength": 0.0
  }
]
```


Abbildung 3.19: Ausgabe auf ActiveMQ

4 Evaluation

4.1 Testumgebung

Durch das gewählte Szenario des Badezimmerspiegels kann auf eine bestimmte Einsatzumgebung geschlossen werden. Man geht somit davon aus, dass sich der Nutzer in nahem Abstand zur Kinect und die LeapMotion in Reichweite der ausgestreckten Arme befindet, zum Beispiel in der Nähe des Wasserhahnes. Der Interaktionsbereich kann dadurch auf den Bereich hinter dem Waschbecken eingegrenzt werden. Auch eine direkte Sonnenlichteinstrahlung, wodurch die Sensoren gestört werden können, kann in einem Badezimmer ausgeschlossen werden.

4.2 Usability

Die spätere Verwendbarkeit der entwickelten Software wurde nach jeder Entwicklungsphase durch kleine Tests sichergestellt, sodass erste Latenzprobleme aufgedeckt und behoben werden konnten.

4.2.1 Alphatests

Es wurden eine Reihe von Tests durchlaufen, die besonders auf die Latenzprobleme, die durch die Verknüpfung der Module via ActiveMQ auftreten können, eingehen sollen.

Sensoren

Die Sensortests waren darauf ausgelegt zu testen, wie stark die Vorverarbeitungen, auf Seiten des Tracking-Moduls Auswirkungen auf die Latenz der jeweiligen Sensoren haben wird. Damit wurde die performante Arbeit der Adapter geprüft.

Der LeapMotion-Adapter beziehungsweise die LeapMotion liefert trotz einigen String-Operationen und Anpassungen der gesammelten Daten eine Framerate zwischen 80 und 130. Der Kinect-Sensor liefert nach allen Operationen stetig 30 Frames pro Sekunde. Das Tracking-Modul

sendet somit genügend Daten, um eine stetige Auswertung dieser auf dem Validierungs-Modul zu ermöglichen.

ActiveMQ

Die Schnittstellen zum ActiveMQ-Server sowie die performante Arbeit des Servers wurde mit Hilfe von Timestamps und Reaktionszeiten des Systems ausgewertet.

Ein großer Faktor spielt die Verwendung von Queues innerhalb von ActiveMQ. Das Prinzip von Queues ähnelt dem eines Stacks. Nachrichten werden vorne herausgeholt aber hinten angehängt. Somit kann, sofern der Empfänger mit einer niedrigeren Framerate als der Sender arbeitet, ein Stau von alten Informationen entstehen. Dieser Test hat dazu geführt, den verwendeten Server für dieses System durch spezielle Konfiguration (vgl Kapitel 3.2.1) anzupassen.

Bearbeitung der Daten

Da die Bearbeitung der Daten die zeitaufwendigste Berechnung innerhalb des Systems ist, wurde mit vollständigen Bewegungen über einen längeren Zeitraum getestet, ob diese sich aufstauen oder das System starken Delay aufweist.

Es wurde festgestellt, dass die Operationen ohne stark sichtbaren Verzug ausgeführt werden und ein gesamtes Datenmodell innerhalb von Unity mit knapp unter 30 Frames pro Sekunde erzeugt wird.

Darstellung des Modells

Durch die Verwendung eines Modells entstehen wieder bestimmte Zeitaufwände, die zwischen nicht bemerkbar, bis hin zu stark auffällig gehen. So wurde anhand des Blender-Modells festgestellt, dass die zusätzlichen Operationen sich stark auf die Anzahl der Frames in Unity auswirken würde (vgl. Kapitel 3.4).

4.2.2 Fazit der Tests

Durch die Auswahl der Tests wurde gewährleistet, dass eine Nutzbarkeit des Systems besteht und die daraus entstehenden Daten in weiteren Systemen verwendet werden können. Die Daten werden performant verarbeitet und das Modell reagiert ohne sichtbaren Verzug. Der Nutzer erhält somit Feedback zu den ausgeübten Bewegungen.

4.3 Erweiterungsmöglichkeiten

Durch die lose Kopplung der einzelnen Module und der Adapter ist es möglich, einen weiteren Sensor einzubinden, sowie einen der beiden Sensoren auszutauschen, solange die definierte Datenstruktur eingehalten oder für die Benutzung geändert wird. Die Verwendung einer anderen Middleware kann durch eine Anpassung der Middleware-Adapter erreicht werden.

Die Darstellung der Daten am Modell wurde desweiteren unabhängig von den Berechnungen getätigt, weshalb diese auch durch eine andere Umgebungen realisiert werden kann. Der komplette Code wurde innerhalb von Unity in C# programmiert, wodurch die Berechnungen auch in andere Prozesse ausgelagert werden können und man somit unabhängig von Unity arbeiten kann.

Imaginary Interfaces (deutsch: imaginäre Oberflächen) sind ein weiterer Ansatz, um eine natürliche Bedienung zu erreichen. Diese Benutzeroberflächen sind rein in der Vorstellung existent und somit „überall". Der Nutzer interagiert zum Beispiel mit Objekten, die er selbst in der Vorstellung erschaffen hat, anstelle von Objekten die, wie bei herkömmlichen Computersystemen, eine vorgegebene Position auf einem Bildschirm besitzen. Wie Gustafson, Bierwirth und Baudisch in ihrer Arbeit beschreiben, ist eine räumliche Interaktion ohne feste Oberfläche zu einem bestimmten Anteil möglich (vgl. [Gustafson u. a. 2010]). So können einfache Graphen oder Objekte in der Vorstellung dargestellt und benutzt werden. In Zusammenhang mit der SensorFusion können dafür genauere Informationen gesammelt werden.

5 Schluss

5.1 Zusammenfassung

In dieser Arbeit wurde beschrieben, wie man anhand eines genauen Szenarios eine Sensorfusion von zwei Sensoren realisiert und die entstandenen Daten an einem gemeinsamen Modell validiert und darstellt. Dabei wurde in Kapitel 2 auf die Grundlagen eingangen. Innerhalb dieser wurde speziell auf die Interaktion von Mensch und Computer eingangen (2.1), sowie auf die generelle Idee der Bewegungsverfolgung (2.1.1) und die der Gestenerkennung (2.1.2). Es wurden verschiedene kamerabasierte Ansätze des „Motion Capture" aufgezeigt und mögliche Fehlerfälle erklärt. Die Gestenerkennung beschäftigt sich mit der Erkennung von Mustern innerhalb einer Bewegung. Dabei werden Gesten in verschiedene Kategorien unterteilt, um mit diesen korrekt umzugehen. Es wurde dann von Gesten im zweidimensionalen (2.1.3) sowie im dreidimensionalen Raum (2.1.4) unterschieden und jeweils Beispiele für mögliche Gesten gegeben und erläutert. Zusätzlich wurde auf weit verbreitete Probleme innerhalb der Gestenerkennung eingangen.

Um die Daten an einem Modell zu validieren, wurde innerhalb des Analyse-Kapitels auf die Modellgestaltung eingegangen (2.2). Darunter fällt die allgemeine Definition eines Modells sowie die Grundlagen der Modellgestaltung am Computer. Aus den gewonnenen Grundlagen wurde dann auf ein eindeutiges Ziel der Arbeit geschlossen (2.3) und dieses mit Hilfe des Szenarios (2.3.1) genauer definiert.

In Kapitel 3 wird auf die Realisierung der Sensorfusion eingangen. Es wird zuerst das Gesamtkonzept aufgezeigt (3.1), in welchem die jeweiligen Schritte eines Systems, das mit dem Nutzer interagiert, erklärt werden. Daraus wird auf die gewählte Systemarchitektur (3.2) geschlossen und diese genauer erläutert. Darunter fällt die Art der Kommunikation zwischen den jeweiligen Modulen (3.2.1) sowie die Stärke der Kopplung (3.2.2). In Kapitel 3.3 wird dann auf die benutzten Sensoren, LeapMotion (3.3.1) und Microsoft Kinect (3.3.2) eingangen. Dafür wird die Benutzung der jeweiligen Software-Development-Kit's (kurz: SDK) beschrieben und

die daraus entstehende Nachrichtenstruktur erklärt.

Der nachfolgende Abschnitt (3.4) handelt von der Wahl des Modells, an welchem die Sensorfusion und die Darstellung ausgeführt wird. Dabei werden Voraussetzungen an das Modell gestellt und zwei Varianten der Modellgestaltung genauer betrachtet. Die erste Variante (3.4.1) beschreibt ein Blendermodell, welches ein vollwertiges Modell darstellt und dem menschlichem Körper stark ähnelt. Die zweite Variante (3.4.2) beschreibt eine Darstellung mit Hilfe von einfachen Objekten. Es werden Vor- und Nachteile abgedeckt und die Entscheidung für die gewählte Variante begründet. Im letzten Abschnitt des dritten Kapitels wird auf die Zuweisung der Daten auf das Modell genauer eingegangen (3.5). Darunter fällt die Kalibrierung der beiden Sensoren (3.5.1), das gewählte Verfahren zur Verrechnung der Datenknoten, die von beiden Sensoren geliefert werden (3.5.2), sowie die Validierung der neuen Daten (3.5.3). Desweiteren wird in Kapitel 3.5.4 auf die Weitergabe sowie die Darstellung der finalen Daten eingegangen.

Kapitel 4 beschäftigt sich mit der Testumgebung sowie der Usability des entstandenen Systems. Es werden die durchgeführten Tests beschrieben und daraus entstehende Informationen erläutert.

5.2 Ausblick

Durch den technischen Fortschritt der letzten Jahre kann gesagt werden, dass eine Interaktion mit einem System durch Bewegungen des eigenen Körpers immer mehr beansprucht werden wird. Die intuitive Steuerung durch Gesten wird in den kommenden Jahren weiter Fokus, auch von großen Unternehmen, wie Apple [1], bekommen. So meldet Apple ein Patent an, um eine Steuerung des Betriebssystem mit einfachen Gesten zu ermöglichen.

Es kann gesagt werden, dass die Computer in der heutigen Form immer weiter durch interaktivere Systeme ersetzt werden. Bei diesen Systeme gilt: „Body is Control", der Körper steuert. Es sollen Geräte, die andere Geräte bedienen, verschwinden. Dabei ist „ubiquitous computing" im Rahmen von „disappeared computing" immer mehr im Vordergrund für zukünftiges, computergestütztes Leben. Im Zusammenhang damit spielt die Akzeptanz sowie die Genauigkeit von 3D-Gesten eine große Rolle.

Man geht davon aus, dass die Benutzung von Gegenständen mit Hilfe von 3D-Gesten in

[1] http://www.golem.de/news/apple-patent-macs-und-apple-tv-mit-gesten-steuern-1501-111672.html

Zukunft als natürlich empfunden wird. Die Akzeptanz von „imaginary interfaces" muss erst noch genauer geprüft werden, jedoch kann durch diese Art von „greifbaren" Benutzeroberflächen ein weiterer Schritt zur natürlichen Bedienung von Geräten, ohne dabei zusätzliche Geräte als Hilfe zu benutzen, gemacht werden.

Literaturverzeichnis

[Aigner u. a. 2012] Roland Aigner, Daniel Wigdor, Hrvoje Benko, Michael Haller u. a. *Understanding Mid-Air Hand Gestures: A Study of Human Preferences in Usage of Gesture Types for HCI*. Microsoft Research, 2012. Erreichbar unter: http://research.microsoft.com/pubs/175454/GesturesTR-20121107-RoA.pdf - besucht am: 05.01.2015

[Apache Software Foundation] The Apache Software Foundation. *ActiveMQ*. 2013. Erreichbar unter: http://activemq.apache.org/ - besucht am: 21.12.2014

[Bauerdiek 2014] Judith Bauerdiek. *Skinning und dessen Optimierungsverfahren*. Universität Koblenz, 2014. Erreichbar unter: http://kola.opus.hbz-nrw.de/volltexte/2014/1029/pdf/Skinning.pdf - besucht am: 08.11.2014

[Boetzer 2008] Joachim Boetzer. *Bewegungs- und Gestenbasierte Applikationssteuerung auf Basis eines Motion Trackers*. HAW Hamburg, 2008. Erreichbar unter: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/boetzer.pdf - besucht am: 10.01.2015

[Boustani 2008] Christian El Boustani. *Human-Computer-Interaction im Seminar: Informationsfluss - Dokumentation und Kommunikation im Unternehmen*. 2008. Erreichbar unter: http://www.se.uni-hannover.de/priv/lehre_2008winter_seminar/Christian_El_Boustani-Human-Computer-Interaction-Folien.pdf - besucht am: 01.11.2014

[Bowie 1997] Vladimir I. Pavlovic, Rajeev Sharma, Thomas S. Huang. *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review*. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 1997

[Bridson 2008] Robert Bridson. *CPSC 426 Computer Animation, Chapter 3B Skinning*. 2008. Erreichbar unter: http://www.ugrad.cs.ubc.ca/~cs426/426-fall-2011/chapter3b.pdf - besucht am: 15.10.2014

[Ellenberg u. a. 2011] Jens Ellenberg, Bastian Karstaedt, Soren Voskuhl, Kai von Luck, Birgit Wendholt. *An Environment for Context-Aware Applications in Smart Homes*. HAW Hamburg, 2011. Erreichbar unter: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/IPIN2011.pdf - besucht am: 03.12.2014

[Ghose 2014] Sobin Ghose. *Konzeption und Evaluation eines interaktiven Badezimmerspiegels*. HAW Hamburg, 2014. Erreichbar unter: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/ghose.pdf - besucht am: 15.02.2015

[Gustafson u. a. 2010] Sean Gustafson, Daniel Bierwirth, Patrick Baudisch. *Imaginary interfaces: spatial interaction with empty hands and without visual feedback*. 2010. Erreichbar unter: http://patrickbaudisch.com/publications/2010-GustafsonBaudisch-UIST10-ImaginaryInterfaces.pdf - besucht am: 09.02.2015

[Hillebrand & Tschakert 2008] Thorsten Hillebrand, Patrick Tschackert. *Evaluierung kamerabasierter Tracking-Ansätze für Motion Capture Applikationen*. HAW Hamburg, 2008. Erreichbar unter: https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/hillebrand.pdf - besucht am: 29.12.2014

[Heinecke 2004] Andreas M. Heinecke. *Mensch-Computer-Interaktion*. Springer Verlag, Germany, 2004.

[Heitsch 2008] Johann Heitsch. *Ein Framework zur Erkennung von dreidimensionalen Gesten*. HAW Hamburg, 2008. Erreichbar unter: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/heitsch.pdf - besucht am: 03.02.2015

[Kalman 1960] Rudolph E. Kalman. *A New Approach to Linear Filtering and Prediction Problems*. Research Institute for Advanced Study, Baltimore, Md., 1960. Erreichbar unter: https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf - besucht am: 28.11.2014

[LeapMotion 2014] LeapMotion Inc. LeapMotion Skeletal Tracking 2014. Erreichbar unter: https://developer.leapmotion.com/documentation/csharp/api/Leap_Classes.html - besucht am: 30.10.2014

[Martin 2014] Alexander Martin. *2D und 3D Gesten-Interaktion mit einem Assistenzsystem am Arbeitsplatz für leistungsgeminderte Arbeiter*. Universität Stuttgart, 2014. Erreichbar unter: http://elib.uni-stuttgart.de/opus/volltexte/2014/9198/pdf/DIP_3559.pdf - besucht am: 20.10.2014

[McNeill 1992] David MacNeill. *Hand and Mind: What Gestures Reveal about Thought*. University of Chicago Press, 1992. Erreichbar unter: http://www.cogsci.ucsd.edu/~nunez/COGS160/McNeill_CH4_PS.pdf - besucht am: 23.11.2014

[Microsoft Corporation 2014] Microsoft Corporation. *Body tracking*. 2010. Erreichbar unter: https://msdn.microsoft.com/en-us/library/dn799273.aspx - besucht am: 15.01.2015

[Newton-King 2013] James Newton-King. *Json.Net*. 2013. Erreichbar unter: http://www.newtonsoft.com/json - besucht am: 13.10.2014

[Potratz 2014] Olaf Potratz. *Ein Framework für physikbasierte 3D Interaktion mit großen Displays*. HAW Hamburg, 2014. Erreichbar unter: https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/potratz.pdf - besucht am: 09.11.2014

[Tang u. a. 2011] Jeff K. T. Tang, Jacky C. P. Chan, Howard Leung. *Interactive Dancing Game with Real-time Recognition of Continuous Dance Moves from 3D Human Motion Capture*. ACM, 2011. Erreichbar unter: http://dl.acm.org/citation.cfm?id=1968613.1968674 - besucht am: 23.10.2014

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. März 2015 Christian Gutjahr