Universität des Saarlandes
Naturwissenschaftlich-Technische Fakultät I
Fachrichtung Informatik

Bachelorarbeit

Implementierung einer gestenbasierten Steuerung des PentAI in sechs Freiheitsgraden mit dem Microsoft Kinect Sensor

vorgelegt von

Patrick Rump

am 20. Juni 2012

Angefertigt unter der Leitung von

Prof. Dr. Jörg Siekmann

Begutachtet von

Prof. Dr. Jörg Siekmann

PD Dr. Christoph Igel

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum / Date) (Unterschrift / Signature)

"I believe we will look back on 2010 as the year we expanded beyond the mouse and keyboard and started incorporating more natural forms of interaction such as touch, speech, gestures, handwriting, and vision — what computer scientists call the 'NUI' or natural user interface."

Steve Ballmer, CEO Microsoft

Danksagung

An dieser Stelle möchte Ich meinen Dank an Herrn Prof. Dr. Jörg Siekmann und Herrn PD Dr. Christoph Igel ausdrücken, die mir die Möglichkeit gaben diese Bachelorarbeit unter ihrer Leitung am Centre for e-Learning Technologies zu verfassen. Ein Großer Dank gilt außerdem Yecheng Gu, der mir mit seinem kompetenten Fachwissen stets zur Seite stand und helfen konnte.

Weiterhin möchte ich mich bei einigen Personen bedanken, die alle einen wertvollen Beitrag zu dieser Arbeit geleistet haben: Pascal danke ich für die wirklich sehr hilfreichen Anmerkungen zur Struktur der Arbeit. Anja danke ich besonders für die sprachlichen Korrekturen. Ganz herzlich möchte ich mich auch bei meiner Freundin Nina bedanken, die mir während der gesamten Arbeit hilfreich zur Seite stand.

Schließlich gilt ein großer Dank meinem Vater für den Rückhalt und die Unterstützung.

Zusammenfassung

Die VR-Umgebung *PentAI* nutzt Techniken der *Virtuellen Realität* (VR) zur Darstellung stereoskopischer und immersiver virtueller Welten. Sie wird dabei gezielt zur Entwicklung und Erforschung von VR-Anwendungen im Bereich der Aus- und Weiterbildung eingesetzt.

Im Rahmen der vorliegenden Arbeit wird die Entwicklung eines gestenbasierten Interaktionsmodells zur Navigation im virtuellen Raum und in *sechs Freiheitsgraden* beschrieben. Hierzu wird zuerst ein Konzept zur Erfassung von Ganzkörpergesten erstellt, mit denen eine solche Navigation realisiert werden kann. Anschließend wird die bestehende Hard- und Softwarekonfiguration des *PentAI* um ein weiteres 3D-Eingabegerät in Form des Microsoft *Kinect Sensors* erweitert und das neue Interaktionsmodell integriert.

Abbildungsverzeichnis

Abbildung 1.1 Dreidimensionale Darstellung der VR-Umgebung PentAI 1

Abbildung 2.1 Die sechs Freiheitsgrade 5
Abbildung 2.2 3D-Eingabegeräte 7
Abbildung 2.3 Virtual Space Environment Simulation (VSESS) 9
Abbildung 2.4 Aufbau des Medical Readiness Trainer (MRT) 10
Abbildung 2.5 CyberMath 10

Abbildung 3.1 Komponenten des Kinect Sensors 11
Abbildung 3.2 Das Referenzdesign The PrimeSensor 12
Abbildung 3.3 Messung der räumlichen Auflösung des Kinect Sensors 13
Abbildung 3.4 Die VR-Umgebung PentAI 15
Abbildung 3.5 Ein Eventgraph mit Modulen und Sensoren, verbunden über Routen 16
Abbildung 3.6 Lightning Systemkonfiguration 17
Abbildung 3.7 GUI des VRfx Frameworks 18
Abbildung 3.8 OpenNI Modell 19
Abbildung 3.9 NITE Modell 20
Abbildung 3.10 Verfügbare Körperpunkte in NITE 20

Abbildung 4.1 Offset-Ball 22
Abbildung 4.2 Kalibrierungsgeste zum Start der Navigation 23
Abbildung 4.3 Vorwärts-/Rückwärts- und Links-/Rechtsbewegung 24
Abbildung 4.4 Auf-/Abwärtsbewegung 25
Abbildung 4.5 Pitch- und Roll-Bewegung 27
Abbildung 4.6 Yaw-Bewegung (Lenkrad) 28
Abbildung 4.7 Stoppgeste zur Beendigung der Navigation 29
Abbildung 4.8 Beziehung zwischen Socketserver, Lightningmodul und VRfx-Plugin 30
Abbildung 4.9 Vereinfachtes Klassenmodell des Socketservers 30
Abbildung 4.10 Sequenzdiagramm einer erfolgreichen Client-Server-Kommunikation 32
Abbildung 4.11 Zustände des Serverplugins 33
Abbildung 4.12 Das Lightningmodul ltDeviceKinect mit Ein- und Ausgängen . . . 33
Abbildung 4.13 Einbindung des Moduls ltDeviceKinect im Lightning Eventgraphen 34
Abbildung 4.14 2D-Benutzeroberfläche des VRfx-Plugins 35
Abbildung 4.15 Visuelle Rückmeldung der fünf Interaktionszustände 36

Abbildung A.1 Klassendiagramm des Socketservers 45

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziele der Arbeit . . . 2
- 1.3 Organisation der Arbeit . . . 2

2 Theoretischer Hintergrund **3**
- 2.1 Virtuelle Realität . . . 3
  - 2.1.1 Immersion . . . 4
  - 2.1.2 Bewegung in einer virtuellen Welt . . . 4
- 2.2 3D-Eingabegeräte . . . 5
- 2.3 Potential von VR in der Bildung . . . 7
- 2.4 Beispielanwendungen . . . 8
  - 2.4.1 VSESS . . . 9
  - 2.4.2 MRT . . . 9
  - 2.4.3 CyberMath . . . 10

3 Technische Rahmenbedingungen **11**
- 3.1 Microsoft Kinect Sensor . . . 11
- 3.2 PentAI . . . 14
  - 3.2.1 Eingabegeräte . . . 14
  - 3.2.2 Visuelle Ausgabeumgebung . . . 14
  - 3.2.3 Lightning . . . 15
  - 3.2.4 VRfx . . . 17
- 3.3 OpenNI . . . 18
- 3.4 NITE . . . 19

4 Implementierung **21**
- 4.1 Konzept zur Erfassung von Ganzkörpergesten . . . 21
  - 4.1.1 Offset-Bereiche . . . 21
  - 4.1.2 Kalibrierungsgeste . . . 22
  - 4.1.3 Translative Bewegungen . . . 23
  - 4.1.4 Rotationsbewegungen . . . 26
  - 4.1.5 Stoppgeste . . . 28
- 4.2 Software . . . 28
  - 4.2.1 Architektur . . . 29

4.2.2 Socketserver 29
4.2.3 Lightningmodul 33
4.2.4 VRfx-Plugin 34
4.2.5 Anwendung im PentAI 35

5 Schlussbetrachtung **37**
5.1 Zusammenfassung 37
5.2 Ausblick 38
5.3 Fazit 39

A Anhang **40**
A.1 Gestendefinitionen 40
A.1.1 Links-/Rechtsbewegung 40
A.1.2 Vorwärts-/Rückwärtsbewegung 41
A.1.3 Auf-/Abwärtsbewegung 41
A.1.4 Pitch-Bewegung 42
A.1.5 Roll-Bewegung 43
A.1.6 Yaw-Bewegung 43
A.1.7 Stoppgeste 44
A.2 Klassendiagramm des Socketservers 45

Literaturverzeichnis **46**

1. Einleitung

Diese Arbeit präsentiert die Implementierung einer gestenbasierten Steuerung der VR-Umgebung *PentAI* (Pentagon for Artifcial Intelligence, vgl. Abbildung 1.1) des Centre for e-Learning Technology (CelTech). Mithilfe des Microsoft *Kinect Sensors* wird ein Interaktionsmodell entwickelt, das eine Navigation im virtuellen Raum und in *sechs Freiheitsgraden* bereitstellt. Hierzu werden intuitive Ganzkörpergesten definiert und die nötige Software entwickelt, um die neue Steuerung des *PentAI* in dessen bestehende Hard- und Softwarekonfiguration zu integrieren.

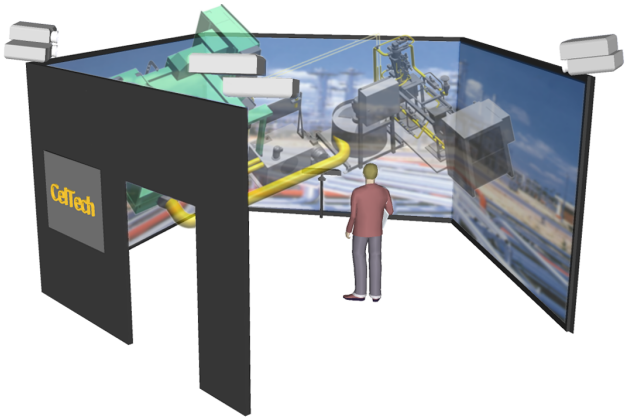


Abbildung 1.1.: Dreidimensionale Darstellung der VR-Umgebung *PentAI*

1.1. Motivation

Virtuelle Welten sind computergenerierte und dreidimensionale Szenarien, in die ein Betrachter durch Technologien der *Virtuellen Realität* (VR) interaktiv eintauchen kann (Boytscheff, 2006). CAVE-Systeme (Cave Automatic Virtual Environment), wie das *PentAI*, sind immersive VR-Umgebungen zur Projektion virtueller Welten. Sie erreichen durch ihren Aufbau und die Struktur ihrer Mensch-Maschine-Schnittstelle einen hoch immersiven Effekt und bieten zudem ein hohes didaktisches Potential (Boytscheff, 2006). Ein wesentlicher Faktor für den Grad des Immersionsempfindens ist dabei die eingesetzte Interaktionsform und damit verbunden auch die Wahl des Eingabegerätes. Vielmehr noch bestimmt das Eingabegerät durch seine Möglichkeiten und seine Grenzen ganz entscheidend die gesamten Fähigkeiten eines VR-Systems (Henning, 2007).

Die Möglichkeit, Information in unterschiedlicher Art und Weise zu visualisieren und erfahrbar zu machen, ist ein Grund dafür, weshalb VR-Systeme heute immer häufiger im Bereich der Aus- und Weiterbildung eingesetzt werden. Lerninhalte können dabei in abstrahierter Form präsentiert werden. Kleine Objekte wie Atome oder Moleküle können darin genauso visualisiert werden wie Planeten oder ganze Universen. Zudem bietet die VR Möglichkeiten zur Manipulation von Zeit und Distanz innerhalb einer virtuellen Welt an. Darauf aufbauende Lernszenarien unterstützen den Ansatz, dass Lernende das präsentierte Wissen einfacher und besser verstehen, wenn sie es interaktiv erfahren können (Youngblut, 1998). In Abschnitt 2.4 dieser Arbeit werden drei existierende VR-Systeme als Beispiele hierfür vorgestellt.

Durch das, in dieser Arbeit vorgestellte, Interaktionsmodell für das *PentAI* zur Navigation im virtuellen Raum soll eine Grundlage für die Entwicklung interaktiver und immersiver VR-Lernszenarien geschaffen werden.

1.2. Ziele der Arbeit

In dieser Arbeit soll eine Möglichkeit aufgezeigt werden, wie mithilfe des *Kinect Sensors* von Microsoft eine gestenbasierte Steuerung des *PentAI*, basierend auf intuitiven Ganzkörpergesten, erreicht werden kann. Konkret soll hierzu ein Interaktionsmodell entwickelt werden, das einem Benutzer des *PentAI* die Navigation im dreidimensionalen Raum und in *sechs Freiheitsgraden* ermöglicht. Es lassen sich hierzu die folgenden Teilziele definieren:

- Die Definition intuitiver Ganzkörpergesten, die eine Bewegung im virtuellen Raum und in *sechs Freiheitsgraden* ermöglichen.
- Die Entwicklung einer Software, die die definierten Gesten unter Verwendung des *Kinect Sensors* erkennen und interpretieren kann.
- Die Integration der neu entwickelten Steuerung in die bestehende Hard- und Softwarekonfiguration des *PentAI*. Dabei soll für die Navigation insbesondere die bisherige Benutzerschnittstelle mittels 3D-Maus ersetzt werden.

1.3. Organisation der Arbeit

Die Arbeit ist wie folgt gegliedert: In **Kapitel 2** wird der theoretische Hintergrund erläutert und die grundlegenden Begriffe und Zusammenhänge zum tieferen Verständnis der Arbeit geklärt. Die technischen Rahmenbedingungen für die Implementierung der gestenbasierten Steuerung für das *PentAI* werden in **Kapitel 3** beschrieben. Das **vierte Kapitel** umfasst die Implementierung und beschreibt hierzu ein Konzept zur Gestenerfassung, die Entwicklung der benötigten Softwarekomponenten und deren Einsatz in der VR-Umgebung *PentAI*. Abschließend wird in **Kapitel 5** eine Zusammenfassung der Arbeit und ein Ausblick auf mögliche Erweiterungen der Ergebnisse gegeben, die aus dieser Arbeit hervorgehen.

2. Theoretischer Hintergrund

In diesem Kapitel werden die theoretischen Grundlagen für die vorliegende Arbeit geschaffen und die wichtigsten Begriffe und Zusammenhänge erläutert. Zunächst werden die Begriffe *Virtuelle Realität* und *Immersion* definiert. Zudem wird die Interaktionsform der Bewegung in einer virtuellen Welt näher beleuchtet und insbesondere der Begriff *Freiheitsgrad* erklärt. Des Weiteren wird eine Übersicht über existierende 3D-Eingabegeräte gegeben. Abschließend soll das Potential von VR-Systemen im Bereich der Aus- und Weiterbildung aufgezeigt und Beispiele existierender VR-Systeme vorgestellt werden.

2.1. Virtuelle Realität

Der Begriff *Virtuelle Realität* ist in der Literatur nicht einheitlich definiert. Vielmehr existieren teils gegensätzliche Begriffsdefinitionen. Beispielsweise beschreibt Bryson (1996) VR als die Verwendung von Computern und Mensch-Maschine-Schnittstellen zur Erzeugung des Effekts einer dreidimensionalen Welt mit interaktiven Objekten und einem starken Gefühl der dreidimensionalen Wahrnehmung. Dabei stellt Bryson (1996) insbesondere heraus, dass VR einen Effekt und keine Illusion einer realen Welt beschreibt. Im Gegensatz dazu verwendet VR laut Rogers, Sharp und Preece (2007) computergenerierte, grafische Simulationen, um die Illusion zu erzeugen, sich unmittelbar in einer synthetischen Umgebung zu befinden. Burdea und Coiffet (2003) wiederum geben in ihrer Arbeit eine übergreifende Definition, wonach VR eine hochqualitative Mensch-Maschine-Schnittstelle ist, die eine Echtzeitsimulation und -interaktion über mehrere Sinne beinhaltet. Diese Definition beschreibt VR in einer abstrakten Weise, die spezifische Definitionen des Begriffs, wie die zuvor genannten, umfasst. In dieser Arbeit soll jedoch die Definition von Carlson (2003) verwendet werden. So wird im Folgenden VR als eine dreidimensionale, computergenerierte Simulation verstanden, bei der man in eine virtuelle Umgebung eintaucht, durch diese navigieren und mit ihr interagieren kann. Andere Begriffe wie *Artificial Reality*, *Augmented Reality* oder *Cyberspace* wurden in der Vergangenheit zwar oft als Synonyme für VR benutzt, sollten aber heute vom Begriff VR getrennt betrachtet werden (Burdea & Coiffet, 2003).

Nach Burdea und Coiffet (2003) werden der VR drei wichtige Aspekte zugesprochen, die sie in ihrer Arbeit als die drei I's der VR bezeichnen. Im Einzelnen sind das *Interaction* (Interaktion), *Immersion* (vgl. Abschnitt 2.1.1) und *Human Imagination* (die menschliche Vorstellungskraft). Interaktion und Immersion werden für gemeinhin als fester Bestandteil von VR in fast allen Definitionen berücksichtigt. Mit dem dritten Aspekt, der menschlichen Vorstellungskraft, betonen die Autoren jedoch zusätzlich die Fähigkeit der Benutzer eines *VR-Systems*, die dargestellten Objekte bewusst wahrzunehmen (Burdea & Coiffet, 2003).

Mit einer *VR-Umgebung* wird eine Kombination aus diversen Ein- und Ausgabegeräten bezeichnet, die eine virtuelle Welt visualisiert. Ein *VR-System* wiederum beschreibt ein System, bestehend aus einer *VR-Umgebung* und einer *VR-Anwendung* (Gu, 2011).

2.1.1. Immersion

Immersion entsteht, wenn ein Individuum die eigene Interaktion mit der Umwelt stärker in einer virtuellen als in der realen Welt wahrnimmt (Guadagno, Blascovich, Bailenson & McCall, 2007). Damit lässt sich das Eintauchen in eine virtuelle Welt, wie es in der Definition von VR nach Carlson (2003) erwähnt wird, mit dem Begriff *Immersion* weiter spezifizieren. Passend dazu definiert Boytscheff (2006) *Immersion* auch als „das Eintauchen bzw. den Grad des Eintauchens in eine Virtual Reality-Szene“. Das Immersionsempfinden hängt dabei, neben der Darstellung der virtuellen Umgebung, auch von der Wahl und dem Design der Mensch-Maschine-Schnittstelle ab. Darunter zählt insbesondere auch die Auswahl des Eingabegerätes eines VR-Systems. In einem Experiment von Asai, Osawa, Sugimoto und Tanaka (2002) konnte gezeigt werden, dass die Verwendung eines optischen Tracking-Systems gegenüber einem Joystick für die Navigation in virtuellen Welten einen positiven Effekt auf das Immersionsempfinden des Benutzers hat. Auch deshalb soll in der Implementierung dieser Arbeit als Eingabegerät ein solches optisches Tracking-System in Form des Microsoft *Kinect Sensors* genutzt werden.

2.1.2. Bewegung in einer virtuellen Welt

Gegenüber zweidimensionalen Benutzerschnittstellen bietet die Verwendung von VR-Systemen neue Möglichkeiten zur Erfahrung virtueller Welten, indem einem Betrachter die Möglichkeit gegeben wird, sich frei durch solche Welten zu bewegen und mit den dargestellten Objekten zu interagieren (Rogers et al., 2007). Mine (1995) nennt hierzu vier grundlegende Interaktionsformen: *Selektion*, *Manipulation*, *Skalierung* und *Bewegung*. Da die Implementierung dieser Arbeit die Integration einer gestenbasierten Steuerung des *Pent-AI* zum Zwecke der Navigation in einer virtuellen Welt vorsieht, wird im Folgenden lediglich der Begriff *Bewegung* näher definiert und beleuchtet.

Eine *Bewegung* besteht aus einer Richtungsangabe und einer Geschwindigkeit, mit der die *Bewegung* ausgeführt wird. In einem VR-System ist dabei die direkte Umsetzung einer realen in eine entsprechende *Bewegung* innerhalb der virtuellen Welt die einfachste und intuitivste Interaktionsmöglichkeit. Ein Vorteil dieser direkten Umsetzung ist, dass ein Betrachter keine besonderen Gesten erlernen muss und ein besseres Verständnis seiner eigenen Präsenz in der virtuellen Welt erfährt (Mine, 1995). Je nach Wahl des Eingabegerätes können Richtung und Geschwindigkeit einer *Bewegung* über unterschiedliche Methoden bestimmt werden. So kann bei der Verwendung eines *Datenhandschuhs* etwa die Handstellung zur Richtungsanzeige genutzt werden oder, unter Verwendung optischer Tracking Systeme, die Position des Betrachters im Raum. Auch indirekte Eingaben über Menüoptionen oder Eingaberäte mit Knöpfen, wie z.B. einem Joystick, sind denkbar (Mine, 1995). Die Bestimmung der Geschwindigkeit einer *Bewegung* lässt sich ebenfalls auf unterschied-

liche Weise erfassen. So unterscheidet Mine (1995) hierbei insbesondere zwischen einer *Bewegung* mit konstanter Geschwindigkeit, bei der ein Betrachter lediglich die Richtung der *Bewegung* beeinflussen kann und einer beschleunigten *Bewegung*, bei der ein Betrachter in der Lage ist, auch die Geschwindigkeit der *Bewegung* direkt zu beeinflussen. Welche konkreten *Bewegungen* in einer virtuellen Welt letztendlich möglich sind hängt somit in der Gesamtheit von der Auswahl und der Implementierung der Interaktionsformen eines VR-Systems ab.

Sechs Freiheitsgrade

Mit dem Begriff *sechs Freiheitsgrade* (englisch: Six degrees of freedom, kurz *6 DOF*) werden die sechs Möglichkeiten zur Positionierung und Orientierung eines Objektes in einem dreidimensionalen Raum beschrieben (Stork, 2001). Sie umfassen die drei in Abbildung 2.1 gezeigten translativen Bewegungen entlang der Achsen und die drei möglichen Rotationsbewegungen um die Achsen eines kartesischen Koordinatensystems. Konkret sind das die sechs folgenden Bewegungen: vorwärts/rückwärts, links/rechts, hoch/runter, yaw (Gierung), pitch (Neigung) und roll (Rolle).

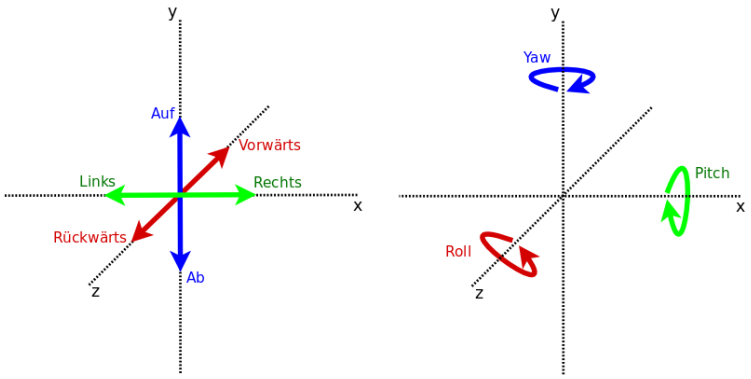


Abbildung 2.1.: Die *sechs Freiheitsgrade* - Translationen (links) und Rotationen (rechts)

2.2. 3D-Eingabegeräte

In den vorherigen Abschnitten wurde bereits erwähnt, dass die Wahl des Eingabegerätes in einem VR-System Auswirkungen auf das Immersionsempfinden des Benutzers hat und entscheidend dazu beiträgt, eine möglichst intuitive Mensch-Maschine-Schnittstelle zu implementieren. Abstrahiert kann ein Eingabegerät als ein System verstanden werden, das Eigenschaften der realen Welt in logische Informationen übersetzt, die von einer Anwendung interpretiert werden können (Card, Mackinlay & Robertson, 1990). Konkret erlaubt ein Eingabegerät dem Benutzer eines VR-Systems die Kommunikation mit einer virtuellen Welt (Gu, 2011). Insbesondere unterstützen *3D-Eingabegeräte* dabei i.d.R. die Eingabe in *sechs Freiheitsgraden* (Stork, 2001).

Aufgrund der Vielzahl an existierenden *3D-Eingabegeräten* sollen nachfolgend nur vier Vertreter der gängigsten, in VR-Systemen eingesetzten, Geräte betrachtet werden, nämlich die *SpaceMouse*, der *Polhemus-Sensor*, der *Datenhandschuh* und der *Kinect Sensor*. Neben einer Beschreibung der Geräte wird zudem ihre Eignung für die Implementierung einer freien Bewegung in einer dreidimensionalen und virtuellen Welt herausgestellt.

1. **SpaceMouse**
   Bei der *SpaceMouse* handelt es sich um ein tisch-gebundenes *3D-Eingabegerät* (Stork, 2001). Sie zählt zur Gruppe der 3D-Mäuse und ermöglicht dem Benutzer die Eingabe in *sechs Freiheitsgraden* über einen freibeweglichen Puck in der Mitte des Gerätes. Durch das Kippen, Schieben und Drehen dieses Pucks kann der Benutzer sich somit in einer virtuellen Welt bewegen. Über zusätzlich am Gerät angebrachte Knöpfe können zudem weitere Interaktionsmöglichkeiten definiert werden.

   Obwohl die *SpaceMouse* als tisch-gebundenes Eingabegerät vor allem für Desktopanwendungen verwendet wird, nutzt beispielsweise das VR-System *PentAI* (vgl. Abschnitt 3.2) den *SpaceNavigator* (vgl. Abbildung 2.2 [a]) zur Navigation in stereoskopischen 3D-Welten. Er zählt ebenfalls zur Gruppe der 3D-Mäuse.

2. **Polhemus-Sensor**
   Der *Polhemus-Sensor* (vgl. Abbildung 2.2 [b]) gehört zur Gruppe der elektromagnetischen Tracker. Es handelt sich also um ein Gerät, das aus einer Kombination von „Sensor und Empfänger besteht, die über ein elektromagnetisches Feld miteinander gekoppelt sind“ (Paes, 2002). Dabei erzeugt der Sender das elektromagnetische Feld und erkennt darin die Lage des Empfängers. Somit lassen sich Position und Orientierung des Empfängers im Raum bestimmen und verfolgen.

3. **Datenhandschuh**
   Ein *Datenhandschuh* (vgl. Abbildung 2.2 [c]) dient der Identifikation der Handstellung. Dazu erkennt der *Datenhandschuh* über angebrachte Sensoren die Krümmung einzelner Gelenke oder ganzer Finger. Er ist allerdings nicht in der Lage, die Handposition im Raum zu bestimmen. Als Sensoren werden meist Glasfasern genutzt, die die unterschiedlichen Lichtverhältnisse beim Verändern der Handstellung erfassen können (Henning, 2007).

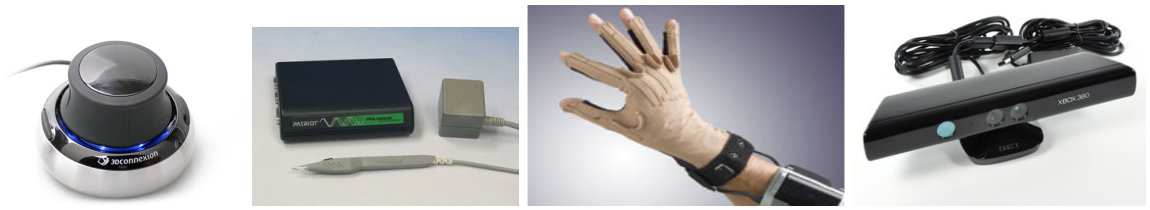
   In Kombination mit einem *Polhemus-Sensor* kann die Interaktion über den *Datenhandschuh* durch die Bestimmung der Handposition erweitert werden (Paes, 2002).

4. **Kinect Sensor**
   Der Microsoft *Kinect Sensor* (vgl. Abbildung 2.2 [d]) ist ein markerloses und optisches Tracking-System, bestehend aus einem Tiefensensor und einer Farbkamera. Er kann bis zu 24 Körperpunkte von zwei Benutzern gleichzeitig erfassen und verfolgen. Für die Bestimmung der Position und Orientierung einer Person im Raum ist hierzu kein weiteres Gerät am Benutzer nötig und es können Hand- sowie Ganzkörpergesten gleichermaßen definiert und genutzt werden. Eine detaillierte Beschreibung des *Kinect Sensors* wird in Abschnitt 3.1 gegeben.

Grundsätzlich können alle genannten *3D-Eingabegeräte* in einem VR-System zur Navigation in einer virtuellen Welt genutzt werden. Markerlose optische Tracking-Systeme, wie der *Kinect Sensor*, weisen jedoch drei wesentliche Vorteile gegenüber anderen Eingabegeräten auf.

- Sie benötigen keine zusätzlichen physischen Objekte, die am Körper des Benutzers angebracht werden müssen und somit zur Ermüdung bei längerer Nutzung beitragen.
- Sie erlauben die Eingabe über flexibel definierbare Hand- und Ganzkörpergesten.
- Sie sind im Vergleich zu professionellen optischen Tracking-Systemen oder vergleichbaren Eingabegeräten kostengünstig zu erwerben[1].

Datenhandschuhe, *SpaceMouse* und *Polhemus-Sensor* übertreffen markerlose optische Tracking Systeme im Bezug auf die erreichbare Genauigkeit der erzeugten Eingabeinformation sowie der Möglichkeit des haptischen Feedbacks. In Abschnitt 3.1 wird jedoch gezeigt, dass auch Systeme wie der *Kinect Sensor* über eine Genauigkeit der Positionsbestimmung verfügen, die zur Entwicklung intuitiver Interaktionsmodelle ausreichend ist.



(a) SpaceNavigator (Schade, 2008) (b) Polhemus-Sensor (Polhemus.com) (c) Datenhandschuh (Abs-tech.com, 2011) (d) Kinect Sensor (Ifixit.com, 2010)

Abbildung 2.2.: 3D-Eingabegeräte

2.3. Potential von VR in der Bildung

Mit VR werden „Wissensinhalte nicht mehr linear vermittelt, sondern vielmehr als ein interaktiv erfahrbares Erlebnis“ (Boytscheff, 2006). In Psotka (1995) wird dieser Paradigmenwechsel mit der Ergänzung einer zusätzlich erlebten Erfahrung zum bisherigen Lernen beschrieben. Daraus resultiert für VR-Systeme ein enormes didaktisches Potential (Boytscheff, 2006). Aktuelle VR-Systeme bieten dabei neue Möglichkeiten zur Wahrnehmungserweiterung, zur Steigerung der Kreativität sowie zu neuartigen sozialen Interaktionen. Diese Eigenschaften sind im Feld der Lernpsychologie zum Beispiel für die Bereiche des Experimentellen Lernens, des Konstruktivismus und des Sozialen Lernens relevant (Bricken & Byrne, 1992).

[1] Am 8. Juli 2012 ist der Kinect Sensor unter www.amazon.de zu einem Preis von 109,99 Euro erhältlich

Heute werden VR-Systeme zu Trainingszwecken meist in Bereichen eingesetzt, in denen das Training der dargestellten virtuellen Situation im realen Leben ein Risiko für den Betrachter darstellen könnte. Somit lassen sich etwa interaktive Trainingsszenarien für Piloten oder angehende Mediziner erstellen, die vom Betrachter von einem „sicheren Platz“ aus erlebt werden können (Rogers et al., 2007). In solchen Szenarien werden meist realitätsnahe Simulationen der realen Welt eingesetzt, die im Hinblick auf ihr Verhalten und ihr Aussehen einer korrespondierenden Situation aus der realen Welt entsprechen.

Neben diesen realistischen Simulationen besteht zudem die Möglichkeit, von der Wirklichkeit abstrahierende Szenarien zu erstellen. Je nach Anwendung kann damit eine Vereinfachung des Lerninhaltes präsentiert werden, wobei sich der Grad einer solchen Abstraktion z.B. nach dem Wissensstand des Betrachters ausrichten kann. Winn (1993) nennt in seiner Arbeit hierzu drei Möglichkeiten der Abstraktion. Eine vierte gibt Gu (2011).

- Durch die *Veränderung von Größenverhältnissen* (Winn, 1993) der virtuellen Welt ist es einem Betrachter u.a. erlaubt, kleinste Atome in einer für ihn nutzbaren Größe zu studieren. Im Gegensatz dazu lassen sich große Objekte, wie unser Sonnensystem, in einer Größe darstellen, in der es dem Betrachter möglich ist, astronomische Distanzen schnell und einfach hinter sich zu lassen.

- Die zweite Abstraktionsmöglichkeit ist die *Transduktion* (Winn, 1993). Sie beschreibt das Sichtbarmachen von Dingen, die im realen Leben nicht über die menschlichen Sinnesorgane wahrgenommen werden können. Hierzu zählen z.B. Lichtwellen, geruchlose Gase oder magnetische Felder.

- Eine dritte Art der Abstraktion bildet die *Reifikation* (Winn, 1993), womit eine Vergegenständlichung von Objekten und Ereignissen gemeint ist, welche in der realen Welt keine physikalische Form besitzen, wie z.B. mathematische Formeln oder der Bevölkerungswachstum eines Landes.

- Die letzte Abstraktionsmöglichkeit ist die Veränderung der Zeitdimension einer virtuellen Welt (Gu, 2011). Hierdurch lässt sich z.B. Vergangenes wieder erleben oder ein zukünftiges Ereignis simulieren.

Ein weiterer wichtiger Punkt bei der Beurteilung des Potentials von VR-Systemen ist die Möglichkeit der sozialen Interaktion. Nach Bricken und Byrne (1992) kann das Austauschen von Blickwinkeln in virtuellen Welten mit mehreren Benutzern zu einer Intensivierung der sozialen Lernerfahrung beitragen. Durch den schnell fortschreitenden Ausbau der Kommunikationsnetze scheint eine Vernetzung von VR-Systemen über das Internet möglich. Zudem werden VR-Systeme durch den Fortschritt in Bereichen wie der 3D-Technologie zunehmend kostengünstiger.

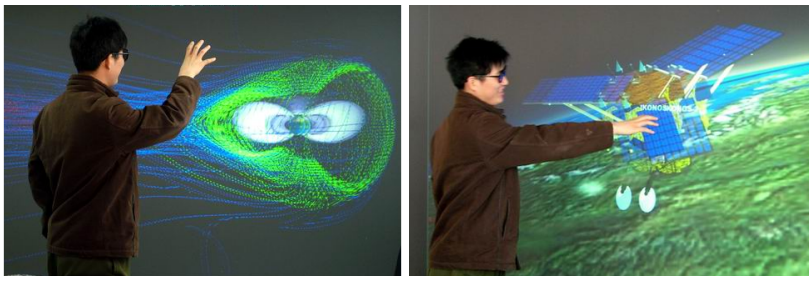
2.4. Beispielanwendungen

In diesem Abschnitt sollen nun drei existierende Beispiele von VR-Systemen vorgestellt werden, die im Bereich der Aus- und Weiterbildung Anwendung finden.

2.4.1. VSESS

Bei VSESS (Virtual Space Environment Simulation System) handelt es sich um eine immersive VR-Umgebung zur Darstellung erdnaher Weltraumobjekte und physikalischer Phänomene (Lan, Xu, Li & Zhou, 2007). Auf der Grundlage realer Daten kann ein Benutzer des VSESS dabei verschiedene Szenarien simulieren. In Abbildung 2.3 [a] sieht man einen Benutzer, der mit dem visualisierten Magnetfeld der Erde interagiert. Demnach setzt VSESS *Transduktion* zur Darstellung physikalischer Phänomene ein. In Abbildung 2.3 [b] ist hingegen die Interaktion mit einem wirklichkeitsgetreuen virtuellen Raumfahrzeug im Orbit der Erde dargestellt. Das System bietet außerdem eine Abstraktion der Zeitdimension an, wodurch Simulationen auf bestimmte Zeiträume beschränkt werden können oder die Ausführungsgeschwindigkeiten der Simulationen veränderbar ist.

VSESS findet vornehmlich Anwendung im Bereich der Raumfahrt und kann nach Lan et al. (2007) u.a. im Zuge von Missionsplanungen sowie zur Durchführung von Probedurchläufen bestimmter Raumfahrtprojekte genutzt werden.



(a) Visualisiertes Erdmagnetfeld (b) Ein Raumfahrzeug im Erdorbit

Abbildung 2.3.: Virtual Space Environment Simulation System (VSESS) (Lan et al., 2007)

2.4.2. MRT

Der MRT (Medical Readiness Trainer) ist eine VR-Umgebung zur realistischen Simulation komplexer medizinischer Szenarien (Pletcher, Bier & Lubitz, 2000). Der Aufbau des MRT besteht aus zwei Kernkomponenten: einem CAVE, in dem der OP-Saal eines Krankenhauses visualisiert wird, und einer menschenähnlichen Simulationspuppe (englisch: Human Patient Simulator, kurz *HPS*) (vgl. Abbildung 2.4). Darüber hinaus werden zur Steigerung der Immersion Audiosignale einer belebten OP-Szene in die Simulation integriert, wodurch zusätzlich psychische Stresssituationen erzeugt werden können. Durch die Vernetzung mehrerer MRT werden außerdem teamorientierte Lernszenarien ermöglicht (Pletcher et al., 2000).

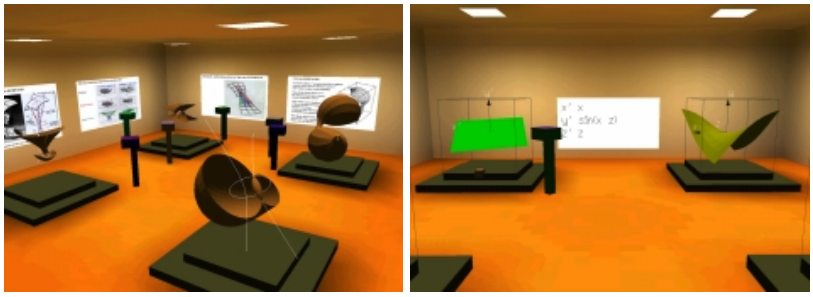
Das MRT ist ein Beispiel für ein sehr komplexes, immersives VR-System, das eine medizinische Situation möglichst real im Bezug auf ihre visuelle Darstellung, das haptische Empfinden und psychische Stressfaktoren simuliert.

Abbildung 2.4.: Aufbau des Medical Readiness Trainer (MRT) (Pletcher et al., 2000)

2.4.3. CyberMath

CyberMath ist eine virtuelle Umgebung, in der Schüler mathematische Lerninhalte erleben können. Die virtuelle Welt stellt eine Art Museum dar (vgl. Abbildung 2.5 [a]), in dem Schüler unterschiedliche Ausstellungsbereiche erforschen können (Taxén & Naeve, 2001). Jeder dieser Bereiche präsentiert dabei einen bestimmten mathematischen Lerninhalt und kann entweder alleine, gemeinsam mit anderen Schülern oder in Begleitung eines Lehrers erkundet werden. Zusätzlich können in speziellen Vorlesungsräumen Powerpoint-Präsentationen verfolgt werden. Durch unterschiedliche Interaktionsmöglichkeiten kann ein Schüler mithilfe eines virtuellen Avatars die Lerninhalte dabei auch interaktiv erleben, indem er z.B. 3D-Graphen in Echtzeit manipuliert (vgl. Abbildung 2.5 [b]).

CyberMath ist ein Beispiel für ein VR-System, das soziales aber auch individuelles und experimentelles Lernen unterstützt. Bei der Vermittlung von Lerninhalten nutzt *CyberMath* z.B. die Reifikation zur Darstellung mathematischer Formeln.



(a) Ein Ausstellungsbereich (b) Echtzeit-Interaktion mit Graphen

Abbildung 2.5.: CyberMath (Taxén & Naeve, 2001)

3. Technische Rahmenbedingungen

In diesem Kapitel werden die technischen Rahmenbedingungen für die Implementierung einer gestenbasierten Steuerung der VR-Umgebung *PentAI* mit dem Microsoft *Kinect Sensor* beschrieben. Zu Beginn des Kapitels wird der *Kinect Sensor* vorgestellt. Daran schließt sich eine Beschreibung der VR-Umgebung *PentAI* an, worin die beiden Softwareprodukte *VRfx* und *Lightning* gesondert betrachtet werden. Zum Abschluss des Kapitels werden die beiden Softwareprodukte *OpenNI* und *NITE* näher beschrieben.

3.1. Microsoft Kinect Sensor

NUI-Systeme (Natural User Interface) sind Systeme, die über eine natürliche Mensch-Maschine-Schnittstelle verfügen und damit einem Benutzer erlauben, auf natürliche Weise mit ihm zu interagieren, etwa durch Spracheingaben, Hand- oder Ganzkörpergesten (Rogers et al., 2007). Der *Kinect Sensor* ist ein 3D-Eingabegerät für NUI-Systeme und zählt zur Gruppe der optischen Tracking-Systeme, die eine markerlose und berührungsfreie Interaktion erlauben. Er wurde im November 2010 von Microsoft als Eingabegerät für deren Spielkonsole Xbox 360 veröffentlicht und ermöglicht einem Benutzer die Steuerung von Spielen über Hand- und Ganzkörpergesten. Kurz nach der Erscheinung waren bereits erste Frameworks verfügbar, die eine Integration des Sensors auch auf anderen Plattformen ermöglichen. Die bekanntesten sind heute *libfreenect*[1] von der Open Source Community OpenKinect, die beiden Frameworks *OpenNI* und *NITE*[2] von PrimeSense sowie das 2011 veröffentlichte *Kinect for Windows SDK* von Microsoft[3].

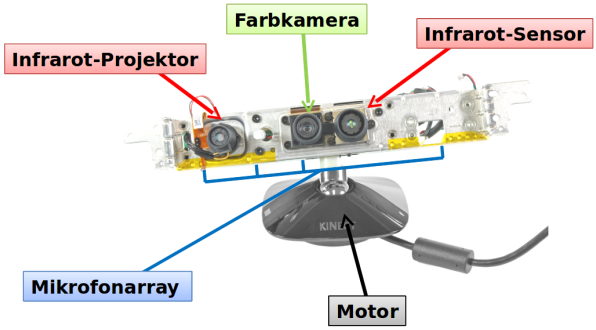


Abbildung 3.1.: Komponenten des *Kinect Sensors* (modifiziert nach Ifixit.com, 2010)

[1] Nähere Informationen unter: http://openkinect.org
[2] Nähere Informationen unter: http://www.openni.org
[3] Nähere Informationen unter: http://www.microsoft.com/en-us/kinectforwindows

Die Hardwarekonfiguration des *Kinect Sensors* basiert auf dem Referenzdesign „The PrimeSensor“ (PrimeSense, 2010) und beschreibt eine Kombination von audiovisuellen Sensoren um den Mikrochip PS1080 (vgl. Abb. 3.2). Die Kernkomponenten (vgl. Abbildung 3.1) des Microsoft *Kinect Sensors* sind nachfolgend aufgelistet:

- Eine RGB-Farbkamera, die Farbbilder in VGA-Auflösung mit 640x480 Pixeln und einer Farbtiefe von 8 Bit aufzeichnet (Freitag, 2011).
- Eine Kombination aus einem Infrarot-Projektor und einem Infrarot-Sensor, die gemeinsam als Tiefensensor fungieren (PrimeSense, 2010). Dabei zeichnet der Infrarot-Sensor monochrome Bilder mit einer VGA-Auflösung von 640x480 und einer Pixeltiefe von 11 Bit auf (Freitag, 2011).
- Ein Mikrofonarray bestehend aus insgesamt 4 Mikrofonen zur Spracheingabe und Lokalisierung von Geräuschquellen im Raum (Microsoft, 2011).
- Ein Motor zur vertikalen und mechanischen Neigung des Eingabegerätes.

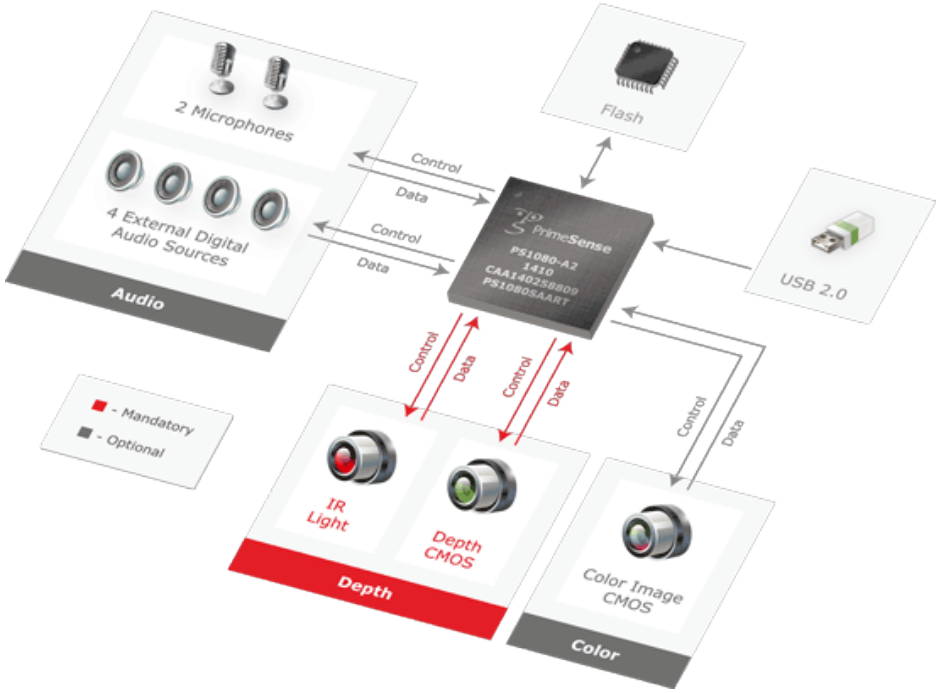
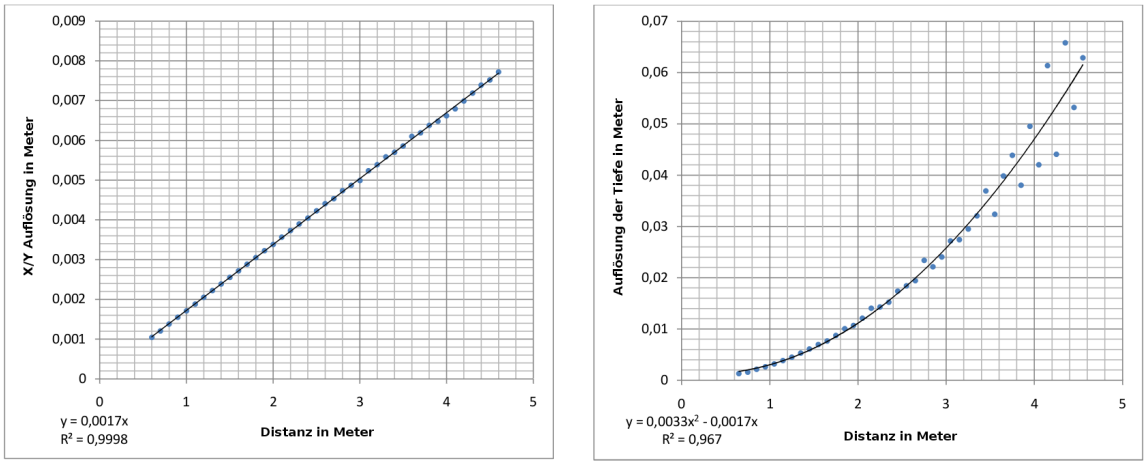


Abbildung 3.2.: Das Referenzmodell „The PrimeSensor“ (PrimeSense, 2010)

Zusammen bilden die Komponenten ein Eingabegerät, das über die Erzeugung von Tiefenbildern mehrere Personen in einem Raum erkennen kann. Offiziell existieren keine Angaben über die Anzahl der Personen, die vom *Kinect Sensor* gleichzeitig verfolgt werden können. Nach inoffiziellen Angaben jedoch geht man in vielen Fällen von zwei Personen aus, die der Sensor aktiv verfolgen kann. Eine aktive Verfolgung bedeutet, dass Hand- und

Ganzkörpergesten vom Sensor identifiziert und verarbeitet werden können. Nach dem Referenzmodell von PrimeSense (2010) wird hierzu die Kombination des Infrarot-Projektors und des Infrarot-Sensors in Verbindung mit der proprietären *Light Coding* Technologie von PrimeSense verwendet (PrimeSense, 2010). Bei diesem *Light Coding* Verfahren wird ein vom PS1080 kodiertes Punktmuster über den Infrarot-Projektor in den Raum emittiert. Die Informationen der reflektierten Lichtpunkte werden anschließend vom Infrarot-Sensor aufgezeichnet und vom PS1080 dazu genutzt, ein resultierendes Tiefenbild zu erzeugen. Durch die Anwendung des sogenannten *Registration* Prozesses wird zudem jeder Pixel aus dem Tiefenbild einem korrespondierenden Pixel im aufgezeichneten Farbbild zugeordnet, wodurch eine Steigerung der Genauigkeit der Sensordaten erreicht wird (PrimeSense, 2010). Die somit generierten Farb- und Tiefeninformationen sowie die aufgezeichneten Audiosignale werden gemeinsam über eine USB 2.0 Schnittstelle an einen angeschlossenen Konsumenten übertragen. Ein großer Vorteil dieses Designs ist, dass bereits ganze Tiefenbilder auf dem Eingabegerät erzeugt werden und somit direkt von einer Anwendungssoftware genutzt werden können.

Im oben genannten Referenzdesign wird die Genauigkeit der Sensordaten bei einer Entfernung von zwei Metern zum Sensor mit einer räumlichen Auflösung von 3 mm (x/y-Auflösung) sowie mit einer Auflösung der Tiefe von 1 cm (z-Auflösung) angegeben (PrimeSense, 2010). Detailliertere Informationen werden von PrimeSense und Microsoft nicht bereitgestellt. In seiner Arbeit hat Viager (2011) jedoch einige dieser Aspekte detaillierter evaluieren können. Seine Ergebnisse (vgl. Abb. 3.3) verifizieren die zuvor genannten Werte und zeigen außerdem, dass der im Referenzdesign beschriebene Interaktionsbereich von 0,8 m bis 3,5 m die genauesten Daten liefert.



(a) Räumliche Auflösung (x/y-Achse)

(b) Auflösung der Tiefe (z-Achse)

Abbildung 3.3.: Messung der räumlichen Auflösung des *Kinect Sensors* (Viager, 2011)

3.2. PentAI

Das *PentAI* (Pentagon for Artificial Intelligence) des Centre for e-Learning Technology (CelTech) ist eine VR-Umgebung in Form eines Pentagons und eine kostengünstige Spezifizierung des in Cruz-Neira, Sandin, DeFanti, Kenyon und Hart (1992) vorgestellten CAVE. Sein Aufbau besteht aus der Kombination einer visuellen Ausgabeumgebung, verschiedener Eingabemöglichkeiten und einer speziellen Softwarekonfiguration, welche auf insgesamt drei miteinander vernetzten Computern ausgeführt wird. Diese drei Computer bilden ein Cluster und dienen der Erzeugung und der Ausgabe von VR-Objekten. Alle Computer weisen zudem die gleiche Hardwarekonfiguration auf, welche sich in ihren Grundkomponenten wie folgt zusammensetzt: Als Prozessor wird die Quadcore CPU Intel Xeon E5520 mit je 2,26 GHz pro Prozessorkern eingesetzt. Die Grafikleistung wird über zwei Grafikkarten vom Typ PNY Verto GeForce 295 GTX bereitgestellt. Hinzu kommt ein Arbeitsspeicher von 4 GB und 2 SATA II Festplatten mit einer Gesamtkapazität von 1 TB.
Die im *PentAI* eingesetzten Softwareprodukte *Lightning* und *VRfx* des Fraunhofer-Instituts für Arbeitswirtschaft und Organisation IAO in Stuttgart werden nachfolgend in den Abschnitten 3.2.3 und 3.2.4 im Einzelnen betrachtet.

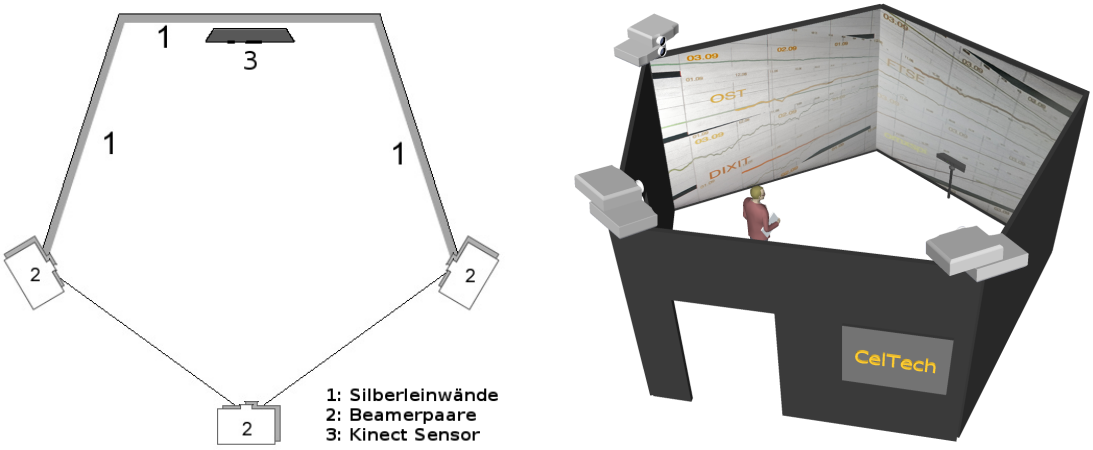
3.2.1. Eingabegeräte

Als Eingabegerät für des *PentAI* wird aktuell, neben Tastatur und Maus, zusätzlich auch der *SpaceNavigator* von 3Dconnexion (vgl. Abschnitt 2.2) eingesetzt. Somit ist bereits in der verfügbaren Konfiguration des *PentAI* eine Eingabe in *sechs Freiheitsgraden* möglich. Mit der Implementierung dieser Arbeit wird dem gegebenen Repertoire durch den *Kinect Sensor* eine weitere Eingabemöglichkeit hinzugefügt. Gegenüber der Eingabe mit dem vorhandenen *SpaceNavigator* handelt es sich bei der neuen Eingabemöglichkeit um eine natürliche Mensch-Maschine-Schnittstelle, deren Ziel die Steigerung des Immersionsgefühls ist.

3.2.2. Visuelle Ausgabeumgebung

Die Ausgabe visueller Information geschieht im *PentAI* mittels Wandprojektion über insgesamt sechs Beamer (vgl. Abbildung 3.4 [a] und [b]), wobei jeweils zwei Beamer in einem Beamerpaar zusammengefasst werden. Drei der insgesamt fünf ca. 3x3 Meter großen Wände des *PentAI* sind mit einer Silberfolie überzogen und bilden die Projektionsflächen des Aufbaus. Sie ermöglichen einen Betrachtungswinkel von insgesamt 180°. Jeweils senkrecht zu den Projektionsflächen ist ein Beamerpaar montiert, so dass aufgrund der geometrischen Anordnung eine nahezu senkrechte Aufprojektion erreicht wird. Zur Erzeugung stereoskopischer 3D-Bildinformationen sind vor den Linsen der Beamer passive Polarisationsfilter angebracht. Dadurch überträgt jedes Beamerpaar zwei unterschiedlich polarisierte Bilder auf die Projektionsfläche, wobei deren Ausrichtung derart kalibriert ist, dass beide Bilder exakt übereinander liegen. Die Silberleinwände sorgen dafür, dass auch nach der Reflexion von den Projektionsflächen die Polarisation der Bildinformation beibehalten wird. Durch die Verwendung einer entsprechenden Polarisationsbrille nimmt ein Benutzer dann die unterschiedlichen Bilder mit jeweils einem Auge wahr. Die in dieser Weise wahrgenommenen

Einzelbilder werden im Gehirn wieder zu einem dreidimensionalen Gesamtbild zusammengefügt.



(a) Schematische Darstellung (b) 3D-Modell des PentAI

Abbildung 3.4.: Die VR-Umgebung *PentAI*. Schematische Darstellung nach Gu (2011).

3.2.3. Lightning

Lightning ist ein modulares Entwicklungssystem des Fraunhofer-Instituts für Arbeitswirtschaft und Organisation IAO in Stuttgart. Basierend auf der Skriptsprache *Tcl* stellt *Lightning* datenflussorientierte Kommunikationsmechanismen zur Verfügung, um verschiedene Objekte einer VR-Anwendung logisch miteinander zu verknüpfen. In *Lightning* werden Objekte als Module bezeichnet und über *Ein- und Ausgabefelder* sowie eine Funktion zur Aktualisierung des Zustandes des Moduls definiert. Die Auswirkungen solcher Zustandsänderungen werden von den Ausgabefeldern über sogenannte *Routen* an daran anknüpfende Eingabefelder anderer Module weitergegeben (vgl. Abbildung 3.5). Zur Entwicklung derartiger Module bietet *Lightning* eine zweischichtige API für die Skriptsprache *Tcl* und die Hochsprache C++ an. Die Gesamtheit aller Objekte einer VR-Anwendung im Zusammenhang mit den definierten Routen bilden einen gerichteten *Eventgraphen*, der das Konzept einer VR-Anwendung widerspiegelt. Module ohne Eingabefelder werden im Speziellen *Sensoren* genannt und bilden die Anfangspunkte eines *Eventgraphen*. In ihnen werden die Informationen angeschlossener Eingabegeräte interpretiert und verarbeitet (Bues, Gleue, Haas & Sulzmann, 2009).

Die wichtigsten Systemkomponenten in *Lightning* bilden ein *Renderer*, ein *Skript Interpreter* und ein *Device Server*, wobei jede dieser Komponenten eigenständig gekapselt und somit austauschbar ist (vgl. Abbildung 3.6 [a]). In der im *PentAI* eingesetzten Version von *Lightning* wird *OpenSceneGraph*[1] als Renderer genutzt.

[1]Nähere Informationen unter: http://www.openscenegraph.org

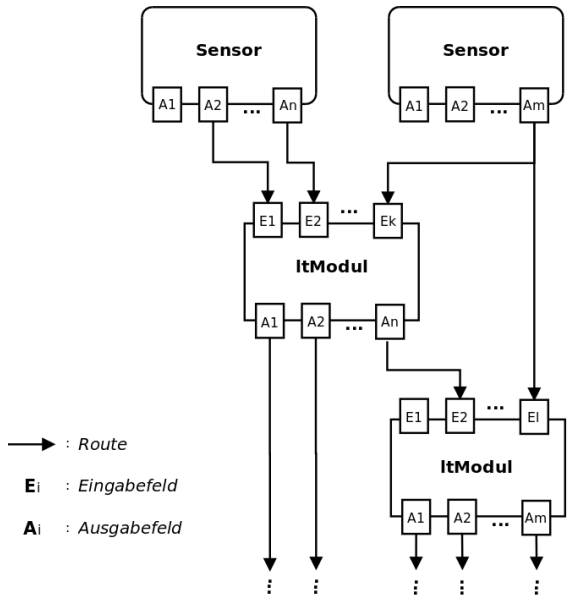
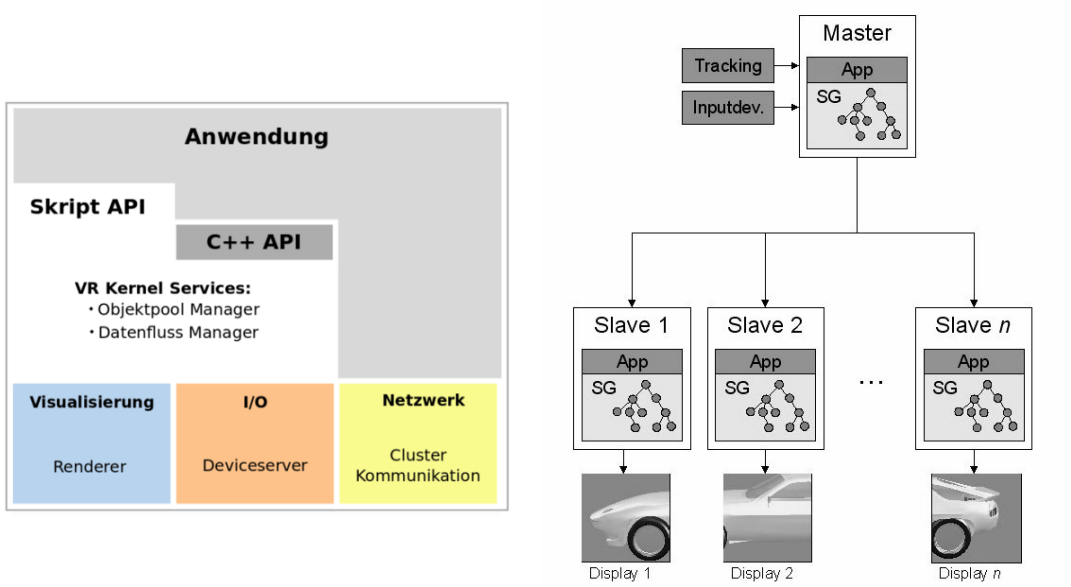


Abbildung 3.5.: Ein Eventgraph mit Modulen und Sensoren, verbunden über Routen

Zur Laufzeit wird der *Eventgraph* einer VR-Anwendung in einer sogenannten *Simulationsschleife* und gemäß einer Breitensuche traversiert, beginnend bei vorhandenen Sensoren. Während dieser Traversierung werden die Zustände der Module auf Grundlage der anliegenden Informationen an den Eingabefeldern verändert, sofern diese sich im Vergleich zur vorangegangenen Traversierung geändert haben. Neben dieser Simulationsschleife führt *Lightning* eine zweite Schleife aus, die zum Rendern der visuellen Objekte einer VR-Anwendung genutzt wird. Die Synchronisierung der beiden Schleifen wird von *Lightning* selbst durchgeführt und sorgt dafür, dass ausgegebene Bildinformationen immer auch die Gesamtheit der aktuellen Modulzustände eines *Eventgraphen* darstellen (Bues et al., 2008).

Ein wesentlicher Vorteil von *Lightning* ist die Unterstützung von Clusterkonfigurationen. Erst hierdurch ist eine erweiterte Projektion auf mehreren Leinwänden wie im *PentAI* möglich. In einem solchen Cluster übernimmt einer der über ein LAN miteinander vernetzten Computer die Rolle eines dedizierten *Masters*, während die restlichen Computer als *Slaves* bezeichnet werden. Auf jedem der Computer existiert zur Laufzeit eine exakte Kopie der ausführenden Lightninganwendung, insbesondere des *Eventgraphen*. Der *Master* berechnet während der Simulationsschleife und vor dem Beginn jeder Traversierung des *Eventgraphen* die Informationen aller angeschlossenen Eingabegeräte und propagiert diese Information weiter an die angeschlossenen *Slaves*. Die anschließende Traversierung des Graphen geschieht auf allen Computern und auf Grundlage identischer Eingabewerte. Einzig bei der Konfiguration des Renderers unterscheiden sich die verbundenen Computer voneinander, da jeder einzelne von ihnen nur einen bestimmten Bereich der visuellen Ausgabe erzeugt. Der zuvor beschriebene Prozess wird in Abbildung 3.6 [b] schematisch veranschaulicht.



(a) Übersicht der Systemkomponenten (b) Clusterkonzept

Abbildung 3.6.: *Lightning* Systemkonfiguration (Bues et al., 2008)

3.2.4. VRfx

Bei *VRfx* handelt es sich ebenfalls um ein VR-Framework, das eine direkte Manipulation einer visualisierten VR-Szenerie erlaubt. Durch die Verwendung von *VRfx* wird der Entwicklungsprozess einer VR-Anwendung derart modifiziert, dass bereits während der Modellierungsphase von VR-Objekten eine Visualisierung der Ergebnisse am Desktop, die der Ausgabe im Zielmedium (z.B. einem CAVE) entspricht, möglich wird. Dadurch ist eine gezielte Trennung zwischen der Entwicklungsphase von VR-Komponenten und der Integration dieser Komponenten in eine VR-Umgebung möglich (Bues, Wenzel, Dangelmaier & Blach, 2007).

Auch für die Visualisierung von 3D-Objekten im *PentAI* wird *VRfx* eingesetzt, wodurch die Generierung von komplexen *Sessions* über die mitgelieferte GUI (Graphical User Interface) möglich ist. In einer persistenten Form beschreibt eine Session eine Menge von Referenzen auf existierende VR-Komponenten wie etwa Geometrien, Texturen oder Shader-Effekte, die dabei in unterschiedlichen Formaten vorliegen können (Bues et al., 2007). Einmal geladene Objekte können über die GUI in Echtzeit modifiziert und die resultierenden Änderungen im Ausgabemedium verfolgt werden (vgl. Abbildung 3.7). Spezielle Funktionen, die nicht direkt über die GUI des *VRfx* Frameworks unterstützt werden, können manuell mit der Skriptsprache Tcl entwickelt und als *VRfx*-Plugin in eine Session geladen werden. Auch die Implementierung dieser Arbeit umfasst die Entwicklung eines solchen Plugins zur einfachen Modifizierung des ebenfalls neu implementierten Lightningmoduls für eine gestenbasierte Steuerung des *PentAI*.

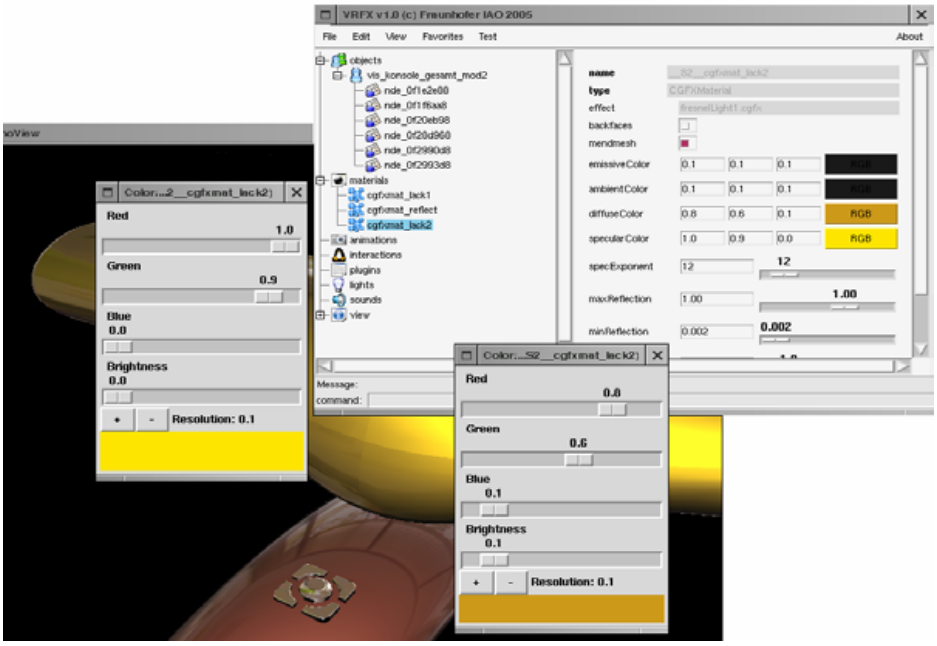


Abbildung 3.7.: GUI des *VRfx* Frameworks (Bues et al., 2007)

3.3. OpenNI

OpenNI (Open Natural Interaction) ist ein Framework, das eine Menge von Opensource APIs zur Erstellung von NUI Anwendungen bereitstellt. Die Intention des *OpenNI* Frameworks ist es, einen Standard bereitzustellen, an dem sich Anwendungssoftware und Hardwarekonfigurationen ausrichten können, um modulare Systeme zu erzeugen, deren Kommunikation über die APIs des *OpenNI* Frameworks gesteuert wird (Kinamon & Gold Hadar, 2011).

Im Einzelnen bietet das *OpenNI* Framework Schnittstellen zur Kommunikation mit audiovisuellen Sensoren sowie Schnittstellen zur Kommunikation mit speziellen Softwareprodukten, sogenannter *Middleware*, an. Sensoren und eingesetzte *Middleware* können im fertigen System aufgrund der modularen Struktur des Frameworks einfach ausgetauscht werden (vgl. Abbildung 3.8). Dadurch erreichen Anwendungen, die auf dem *OpenNI* Framework basieren, einen hohen Grad an Flexibilität und Portabilität.

Neben integrierten Funktionalitäten, wie dem Einlesen von Sensordaten oder der Erkennung von Personen im Raum, bietet *OpenNI* auch Schnittstellen für bestimmte Komponenten einer *Middleware* an, welche die Grundfunktionen des *OpenNI* Frameworks ergänzen können. Diese optionalen Erweiterungen werden *Capabilities* genannt und umfassen u.a. die Erkennung bestimmter Gesten oder die Möglichkeit, einem erkannten Benutzer ein virtuelles Skelett zuzuordnen (Kinamon & Gold Hadar, 2011).

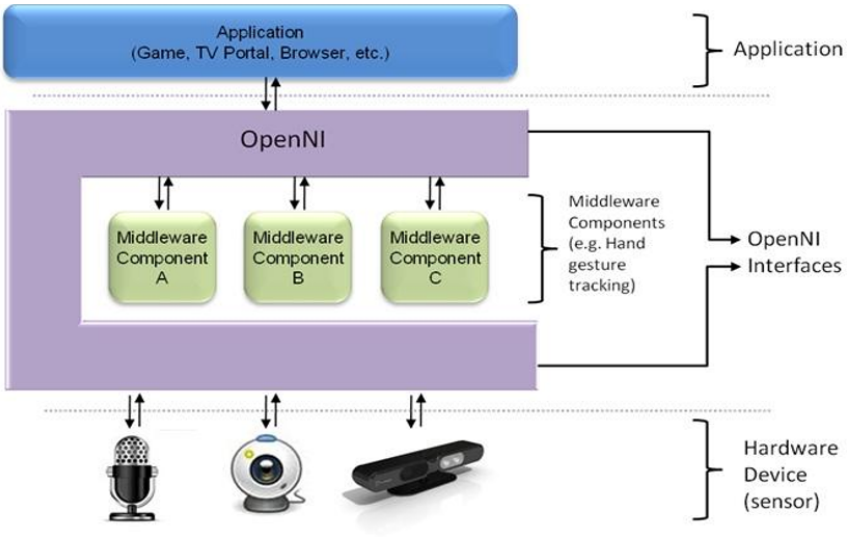


Abbildung 3.8.: *OpenNI* Modell (Kinamon & Gold Hadar, 2011)

Die Implementierung dieser Arbeit basiert ebenfalls auf dem *OpenNI* Framework. Als Sensor wird dabei der *Kinect Sensor* von Microsoft (vgl. Abschnitt 3.1) eingesetzt und als datenverarbeitende Middleware das *NITE* Framework von PrimeSense, welches im nachfolgenden Abschnitt beschrieben wird.

3.4. NITE

Bei *NITE* (Natural Interaction Technology for End-user) handelt es sich um eine Middleware, die in das zuvor beschriebene *OpenNI* Framework integriert werden kann und dessen Grundfunktionalität durch zusätzliche Funktionen erweitert. Hierzu stellt *NITE* neben der eigentlichen Schnittstelle zum *OpenNI* Framework insbesondere auch Module und Algorithmen zur Verfügung, die zur Erkennung und Verfolgung von Hand- und Ganzkörpergesten sowie zur Erstellung eines virtuellen Skelettes dienen. Die Software greift dazu auf Funktionen der freien Programmbibliothek *OpenCV*[1] sowie der Physiksimulationsbibliothek *PhysBAM*[4] zurück (PrimeSense, 2011).

Der Aufbau von *NITE* lässt sich in zwei Ebenen gliedern, die jeweils eine Menge an Implementierungen bestimmter *OpenNI* Schnittstellen umfassen (vgl. Abbildung 3.9). Die untere Ebene, namens *NITE Algorithmus*, interpretiert und verarbeitet empfangene Tiefeninformationen angeschlossener Sensoren. Als Resultat bietet die Ebene Funktionen zur Segmentierung einer Szene, zur Handverfolgung und zur Verfolgung des gesamten Körpers an (PrimeSense, 2011). In der Anwendungsebene *NITE Controls* werden die in der *NITE Algorithms* Ebene generierten Informationen aufbereitet und schließlich über eine weitere Schnittstelle den angeschlossenen Anwendungen bereitgestellt.

[1] Nähere Informationen unter: http://opencv.willowgarage.com
[4] Nähere Informationen unter: http://physbam.stanford.edu

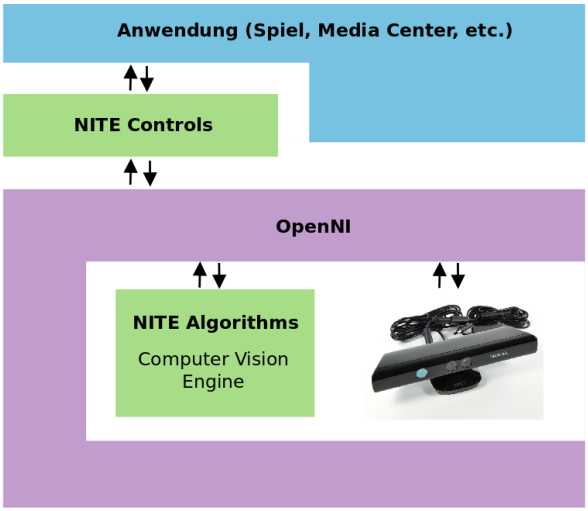


Abbildung 3.9.: *NITE* Modell (modifiziert nach PrimeSense, 2011)

Für die Implementierung dieser Arbeit sind die vordefinierten Gesten des *NITE* Frameworks nicht ausreichend, weshalb eigene Gesten definiert werden müssen (siehe Abschnitt 4.1). Dazu werden die von *NITE* bereitgestellten Informationen über die Positionsdaten der insgesamt 24 Körperpunkte eines Benutzers verwendet, die das Framework erfassen und verfolgen kann (vgl. Abbildung 3.10). Bringt man ihre Positionen miteinander in Verbindung, so erlaubt dies die Erkennung und Verfolgung bestimmter Körperhaltungen.

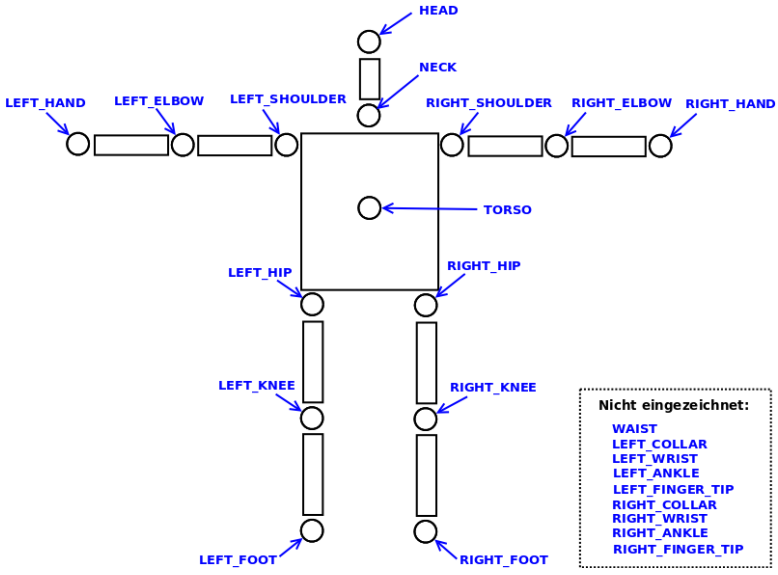


Abbildung 3.10.: Verfügbare Körperpunkte in *NITE* (modifiziert nach LaBelle, 2011)

4. Implementierung

In diesem Kapitel wird die Entwicklung der gestenbasierten Steuerung für die VR-Umgebung *PentAI* durch die Verwendung des Microsoft *Kinect Sensors* vorgestellt. Zu Beginn wird ein Konzept zur Erfassung von Ganzkörpergesten vorgestellt. Das Konzept definiert insgesamt acht Gesten, die die Grundlage für eine intuitive Bewegung im virtuellen Raum bilden sollen. Im zweiten Teil des Kapitels wird die Entwicklung der Software beschrieben, die zur Erkennung und zur Verarbeitung der Gesten nötig ist. Im Einzelnen wird darin auf die Architektur der Softwarekomponenten, auf die Integration in die VR-Umgebung *PentAI* und auf die Anwendung der neuen Steuerung innerhalb des *PentAI* eingegangen.

4.1. Konzept zur Erfassung von Ganzkörpergesten

Beim Design einer erfolgreichen Mensch-Maschine-Schnittstelle gilt es darauf zu achten, dass während der gesamten Entwicklungsphase bestimmte Usability-Ziele verfolgt werden. Hierzu zählen u.a. *Effizienz*, *Effektivität*, eine einfache *Erlernbarkeit* und ein guter *Wiedererkennungswert* (Rogers et al., 2007). Auch bei der Entwicklung des nachfolgend beschriebenen Interaktionsmodells waren diese Ziele zu jeder Zeit wichtige Orientierungspunkte. Einige Gesten zum Beispiel nutzen während der Steuerung konstante Längenangaben, um u.a. die Ausmaße eines bestimmten Offset-Bereichs zu definieren. Eine Kalibrierungsphase vor dem Start der Steuerung ermöglicht es diese konstanten Werte individuell an einen Benutzer anzupassen. Dazu wird die Länge des Unterarms eines Benutzers gemessen und zur Bestimmung der zuvor genannten Größenangaben verwendet. Somit wird die Grundlage für eine intuitive Steuerung und damit verbunden eine Steigerung des Immersionsgefühls im *PentAI* geschaffen.

Die in diesem Kapitel vorgestellten Gesten definieren die Eingabemöglichkeiten für Bewegungen in *sechs Freiheitsgraden*. Zusätzlich dazu wird eine vom verwendeten *OpenNI* Framework vordefinierte Kalibrierungsgeste sowie eine Geste zur Beendigung der Eingabe präsentiert.

4.1.1. Offset-Bereiche

Die in den folgenden Abschnitten beschriebenen Ganzkörpergesten definieren die Eingabemöglichkeiten, die von der implementierten Software dieser Arbeit identifiziert und verarbeitet werden können. Dabei wurde darauf geachtet, dass jede Geste eindeutig erkannt wird und somit mögliche Zweideutigkeiten bei der Erkennung ausgeschlossen werden. Nur so kann eine exakte Übertragung der intendierten Bewegung eines Benutzers in die ent-

sprechende Bewegung im virtuellen Raum erfolgen. Die Ausführung von Gesten, die nicht das Ziel einer Bewegungsänderung im virtuellen Raum haben, sollen möglichst vermieden werden. Trotzdem soll dem Benutzer weiterhin die Möglichkeit zur Kommunikation mit Mitmenschen in der realen Welt gegeben werden, d.h. dass der Benutzer sich im virtuellen Raum bewegen kann, während er beispielsweise durch das Zeigen auf ein Objekt mit seiner Umwelt kommuniziert.

Zur Vermeidung nicht gewollter Bewegungen werden für jede Geste sogenannte *Offset-Bereiche* definiert. Hierbei handelt es sich um kleine Regionen, in denen eine definierte Geste von der interpretierenden Software zwar als solche erkannt, ihre intendierte Bewegung aber nicht in eine virtuelle Bewegung übersetzt wird.

Offset-Ball

Eine spezielle Form des *Offset-Bereichs* bildet der implementierte *Offset-Ball*. Dabei handelt es sich um einen kugelförmigen *Offset-Bereich* um den Mittelpunkt des Rumpfes eines Benutzers (vgl. Abbildung 4.1). Der Radius des *Offset-Balls* entspricht der Länge des Unterarms eines Benutzers, welche während der Kalibrierungsphase gespeichert wird. Bei einigen Gesten sorgt der *Offset-Ball* dafür, dass ungewollte Bewegungen im realen Raum aus der Eingabe herausgefiltert werden, sofern sich mindestens eine Hand des Benutzers innerhalb des *Offset-Balls* befindet.

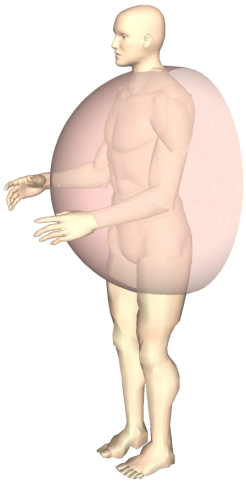


Abbildung 4.1.: Offset-Ball

4.1.2. Kalibrierungsgeste

Zum Start der Navigation in einer virtuellen Welt mithilfe der nachfolgend definierten Ganzkörpergesten bedarf es einer bestimmten Kalibrierungsgeste, die vom Benutzer ausgeführt werden muss. Hierzu bietet das eingesetzte *OpenNI* Framework eine vordefinierte „Psi“-Geste an (vgl. Abbildung 4.2). In der verwendeten Version des Frameworks besteht

ebenfalls die Möglichkeit, die Kalibrierung eines Benutzers ohne die Verwendung dieser besonderen Geste durchzuführen. Hierzu beginnt das Framework die Kalibrierungsphase sobald ein Benutzer im Sichtfeld des Sensors erkannt wurde. In der vorliegenden Arbeit soll jedoch ein erfolgreicher Kalibrierungsprozess auch zum Start der Steuerung genutzt werden und somit der Zeitpunkt der Kalibrierung vom Benutzer frei wählbar sein. Daher wurde auf eine automatische Kalibrierung ohne Geste verzichtet.

Zur Durchführung der oben erwähnten Kalibrierungsgeste muss ein Benutzer eine bestimmte Körperhaltung einnehmen und diese über einen Zeitraum von ca. zwei Sekunden beibehalten. Hierzu stellt er sich in einem Abstand von etwa zwei Metern zum *Kinect Sensor* auf und winkelt beiden Arme senkrecht nach oben an. Der Erfolg der Kalibrierung wird dem Benutzer von der Anwendung visuell mitgeteilt (vgl. Abschnitt 4.2.5).

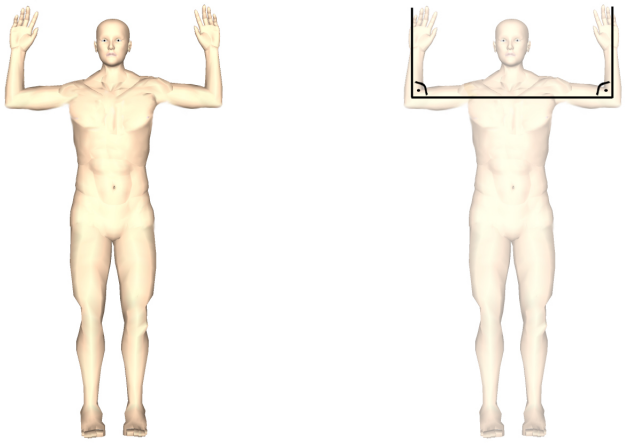


Abbildung 4.2.: Kalibrierungsgeste zum Start der Navigation

4.1.3. Translative Bewegungen

Gemäß dem in Abbildung 2.1 dargestellten Koordinatensystem ergeben sich drei verschiedene translative Bewegungen im Raum: eine Vorwärts- und Rückwärtsbewegung entlang der z-Achse, eine Links- und Rechtsbewegung entlang der x-Achse sowie eine Auf- und Abwärtsbewegung entlang der y-Achse.

Vorwärts-/Rückwärts- und Links-/Rechtsbewegung

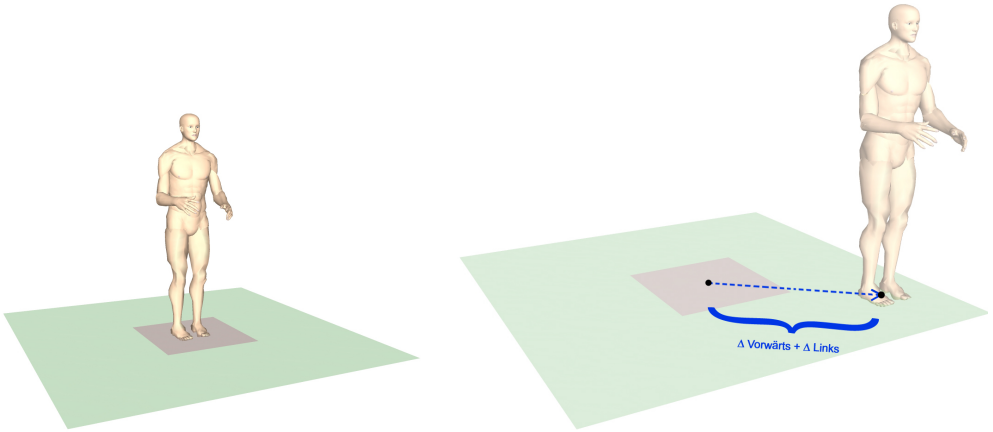
Eine virtuelle Bewegung entlang der x- und der z-Achse des Raumes wird durch eine identische Positionsveränderung in der realen Welt erzeugt. Damit Geschwindigkeit und Richtung bestimmt werden können, bedarf es hierzu zweier Informationen: zum einen wird ein initialer Standort des Benutzers benötigt und zum anderen seine aktuelle Position. Der Abstand zwischen diesen beiden Punkten ist dann ein Maß für die Geschwindigkeit der Bewegung, während der entsprechende Vektor die Richtung der Bewegung beschreibt. Die

initiale Position des Benutzers wird nach einer erfolgreichen Kalibrierungsphase zu Beginn der Navigation als globaler Wert gespeichert und über den Mittelpunkt des Rumpfes bestimmt.

Bewegt sich der Benutzer im realen Raum also beispielsweise auf den *Kinect Sensor* zu, so wird daraus eine entsprechende Vorwärtsbewegung im virtuellen Raum abgeleitet. Analog dazu wird bei einer Bewegung nach links oder rechts verfahren. Beide Bewegungen lassen sich zudem kombinieren und resultieren dann in einer überlagerten Bewegung (vgl. Abbildung 4.3 [b]). Die direkte Umsetzung von der realen in eine virtuelle Bewegung sorgt dafür, dass die beschriebene Geste intuitiv zu benutzen und leicht zu erlernen ist.

Um ungewollte Bewegungen zu vermeiden, wird bei der Erkennung der Gesten ein Offset-Bereich verwendet. Er wird durch einen konstanten Wert beschrieben, der eine Entfernung zur initialen Position des Benutzers definiert. Da für die Vorwärts- und Rückwärtsbewegung sowie für die Links- und Rechtsbewegung der gleiche Wert genutzt wird, bildet der Offset-Bereich eine quadratische Fläche um die initiale Position des Benutzers (vgl. Abbildung 4.3 [a] und [b]).

In den Abschnitten A.1.1 (Links- und Rechtsbewegung) und A.1.2 (Vorwärts- und Rückwärtsbewegung) des Anhangs wird die Berechnung der, von dieser Geste ausgehenden, Bewegungen anhand von Pseudocode näher beschrieben.



(a) Benutzer im Offset-Bereich (b) Bewegungsänderung nach vorne links

Abbildung 4.3.: Vorwärts-/Rückwärts- und Links-/Rechtsbewegung

Auf-/Abwärtsbewegung

Die Auf- und Abwärtsbewegung ist eine translative Bewegung entlang der y-Achse. Eine direkte Umsetzung, wie im Abschnitt zuvor, wäre auch hier denkbar. Ein Benutzer könnte dazu zum Beispiel eine Bück- oder Streckbewegung ausführen. Dies jedoch würde zu einer unkomfortablen Kombination der translativen Bewegungen führen. Möchte sich ein

Benutzer in der virtuellen Welt etwa schnell zurück und gleichzeitig nach unten bewegen, so müsste er im realen Raum in gebückter Haltung ein paar schnelle Schritte nach hinten gehen. Eine solche Geste würde womöglich eine starke Ermüdung des Benutzers nach sich ziehen und wäre zudem unkomfortabel zu benutzen.

Aus diesem Grund wird für die Auf- und Abwärtsbewegung in dieser Arbeit eine abstraktere Geste definiert. Der Benutzer streckt hierzu beide Hände zur Seite aus, sodass sie sich außerhalb des *Offset-Balls* befinden. Durch das parallele Anheben oder Senken der Hände erreicht der Benutzer anschließend eine entsprechende Auf- bzw. Abwärtsbewegung in der virtuellen Welt (vgl. Abbildung 4.4 [a]). Zur Vermeidung ungewollter Bewegungen ist neben dem *Offset-Ball* noch ein weiterer Offset-Bereich definiert. Teilt man den Raum neben dem Benutzer in drei gleichgroße Ebenen, bei denen der höchste Punkt auf Schulterhöhe und der tiefste Punkt auf Hüfthöhe des Benutzers liegt, so beschreibt die mittlere dieser Ebenen den zusätzlichen Offset-Bereich. Die obere der drei Ebenen beschreibt den Interaktionsbereich für die Aufwärtsbewegung und die untere Ebene den Interaktionsbereich für die Abwärtsbewegung. Die Geschwindigkeit der Bewegung ergibt sich dabei entsprechend aus dem Mittel der Tiefe, in welche der Benutzer beide Hände in einen der beiden Interaktionsbereiche „eingetaucht“ hat (vgl. Abbildung 4.4 [b]).

Die Idee hinter dieser Geste ist das Anheben, bzw. das Absenken eines physischen Objektes. In diesem Fall kann die Idee mit dem Anheben oder Absenken der Kamera in der virtuellen Welt verglichen werden. Es handelt sich also auch hierbei um eine bekannte Geste aus der realen Welt. Daher kann auch hierbei angenommen werden, dass die Geste einfach zu erlernen und anschließend intuitiv zu benutzen ist. Im Anhang wird die Erkennung der Geste in Abschnitt A.1.3 näher beschrieben.

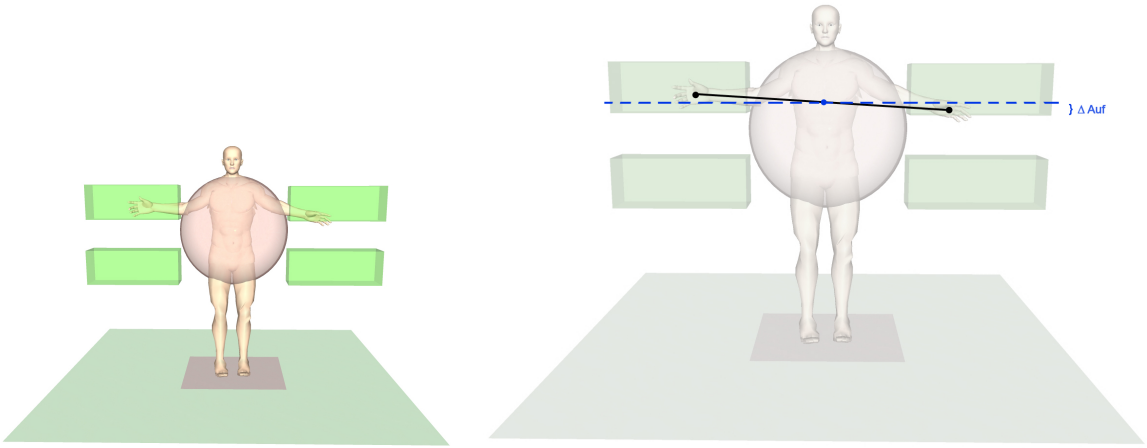


Abbildung 4.4.: Auf-/Abwärtsbewegung

4.1.4. Rotationsbewegungen

Im dreidimensionalen Raum existieren neben den in Abschnitt 4.1.3 beschriebenen translativen Bewegungen, auch die drei Rotationen *yaw*, *pitch* und *roll* (vgl. Abschnitt 2.1.2). *Yaw* beschreibt hierbei die Rotation um die y-Achse, *pitch* die Rotation um die x-Achse und *roll* die Rotation um die z-Achse.

Pitch & Roll

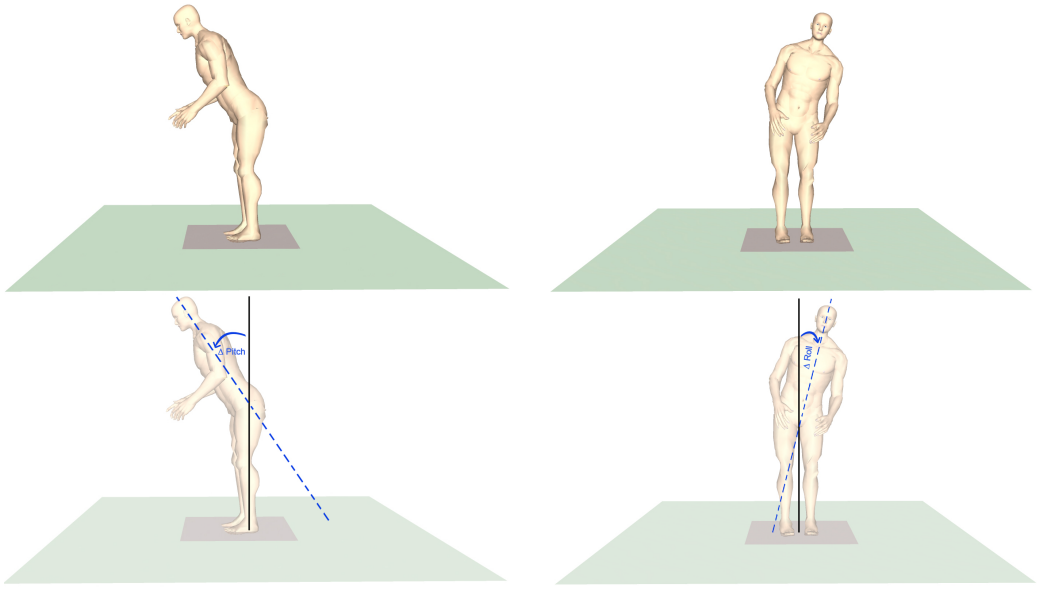
Für die *Pitch*- und *Roll*-Bewegung wird eine gemeinsame Geste definiert, ähnlich der Gestendefinition für die translativen Vorwärts-, Rückwärts- Links- und Rechtsbewegungen. *Pitch* stellt eine Neigung der virtuellen Kamera nach vorne oder nach hinten dar, während *Roll* eine Rollbewegung der Kamera zur Seite beschreibt. Im realen Leben entspricht dies einer Änderung des Blickwinkels durch die Neigung des Kopfes. Eine Geste, die auf der direkten Umsetzung dieser Neigung basiert, wäre für die Navigation im *PentAI* zwar intuitiv aber auch ungeeignet. Wegen der Verwendung einer beschleunigten Bewegung würde die Neigung des Kopfes zu Latenzen zwischen der realen und der Bewegung im virtuellen Raum führen. Damit wäre eine intuitive Bedienung nicht möglich. Zudem ist der *Kinect Sensor* bezüglich der Erkennung der Orientierung eines Kopfes für eine solche Geste noch nicht ausreichend genau. Aus diesem Grund wurde für die Bewegungen *pitch* und *roll* eine vergleichbare Geste verwendet.

Durch die Neigung seines Oberkörpers nach links, rechts, vorne oder nach hinten kann der Benutzer eine gleichwertige *Roll*- oder *Pitch*-Bewegung in der virtuellen Welt ausführen. Beide Bewegungen können zur gleichen Zeit ausgeführt werden und resultieren dann in einer entsprechend überlagerten Gesamtbewegung. Grundlage für die Geschwindigkeit und die Richtung der Bewegung ist der Grad der Körperneigung. Für die *Pitch*-Bewegung ist das der Winkel zwischen der geneigten Oberkörperachse und der x/y-Ebene des Raumes (vgl. Abbildung 4.5 [a]), für die *Roll*-Bewegung entsprechend der Winkel zwischen der Oberkörperachse und der y/z-Ebene des Raumes (vgl. Abbildung 4.5 [b]).

Der Offset-Bereich dieser Bewegungen ist über eine konstante Winkelgröße festgesetzt. Dabei werden verschiedene Werte für den Offset-Bereich nach vorne, nach hinten und zu den Seiten definiert. Eine reale Bewegung wird somit erst in eine virtuelle Bewegung umgesetzt, sofern die Körperneigung größer als die jeweilige Offset-Größe ist. In den Abschnitten A.1.4 (pitch) und A.1.5 (roll) des Anhangs werden die Berechnungen zur Erkennung der Geste anhand von Pseudocode näher beschrieben.

Yaw

Die letzte der drei Rotationen bildet die sogenannte *Yaw*-Bewegung. Dabei handelt es sich um eine Rotation um die y-Achse, was in der realen Welt einer Drehung um die eigene Achse entspricht. Wie schon bei der *Pitch*- und *Roll*-Bewegung wäre auch bei der *Yaw*-Bewegung eine direkte Umsetzung der realen Bewegung für die Navigation im *PentAI* ungeeignet. Zwar wäre die Geste intuitiv, doch der Benutzer könnte durch eine Drehung um



(a) Pitch (b) Roll

Abbildung 4.5.: *Pitch*- und *Roll*-Bewegung

die Körperachse eventuell den Blick zur Leinwand verlieren oder ihn nur durch ein starkes Verdrehen des Kopfes aufrecht erhalten. Speziell bei der Verwendung im *PentAI* ist somit eine intuitive Navigation innerhalb einer virtuellen Welt nicht gewährleistet.

Eine abstrakte Alternative und die hier implementierte Geste stellt die Nutzung eines virtuellen Lenkrades bereit (vgl. Abbildung 4.6). Die Funktion eines realen Lenkrades ist hinlänglich bekannt und kann daher als intuitiv bedienbar betrachtet werden.

Wie schon bei der translativen Auf- und Abwärtsbewegung müssen sich auch bei dieser Geste beide Hände des Benutzers außerhalb des *Offset-Balls* befinden. Erst dann kann der Benutzer eine *Yaw*-Bewegung ausführen, indem er ein unsichtbares Lenkrad vor dem Körper bedient. Die Richtung der Bewegung ergibt sich gemäß der Verwendung eines realen Lenkrades in einem PKW. Die Erkennung der Richtung und der Geschwindigkeit erfolgt dabei über den Winkel zwischen der x/z-Ebene des Raumes und einer Geraden, die durch beide Hände des Benutzers führt. Im Einzelnen wird die Geschwindigkeit der Bewegung über die Größe des Winkels bestimmt und die Richtung anhand der beiden Handpositionen. Befindet sich die linke Hand höher als die rechte, so resultiert daraus eine Rotation nach rechts und umgekehrt.

Der Offset-Bereich der *Yaw*-Bewegung wird, wie bei der *Pitch*- und *Roll*-Bewegung, über eine konstante Winkelgröße definiert. Erst wenn die erzeugte Lenkbewegung einen größeren Winkel beschreibt als die Offset-Größe resultiert daraus eine Bewegung im virtuellen Raum. Zu diesem Offset-Bereich wird außerdem noch eine weitere Einschränkung gegeben, nach der der Abstand zwischen beiden Händen kleiner sein muss als der dynamische

Wert, der sich aus dem Produkt aus *1,7 x Unterarmlänge des Benutzers* ergibt. Auch die Erkennung dieser Geste wird im Anhang (Abschnitt A.1.6) anhand von Pseudocode näher beschrieben.

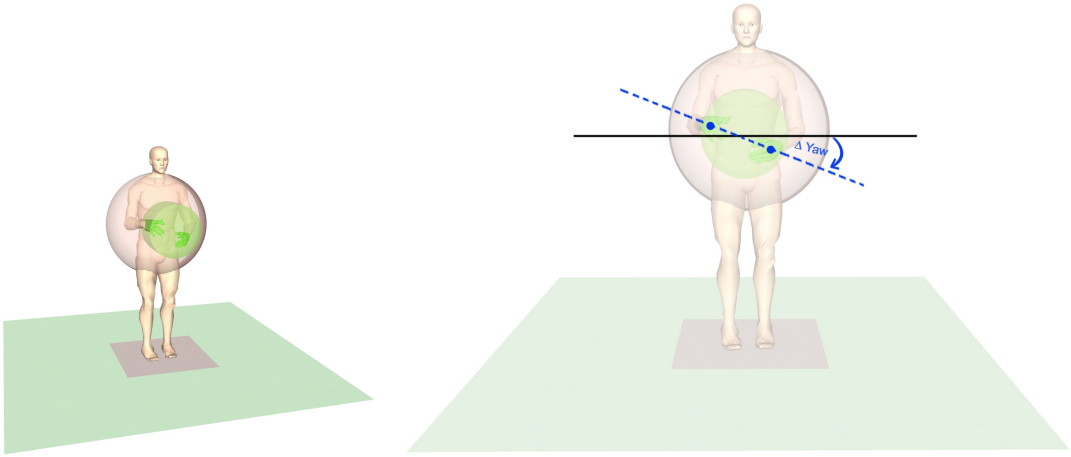


Abbildung 4.6.: *Yaw*-Bewegung (Lenkrad) mit Offset-Ball

4.1.5. Stoppgeste

Zum Abschluss der Gestendefinitionen wird die Stoppgeste vorgestellt. Sie erlaubt dem Benutzer die aktive Gestensteuerung des *PentAI* zu beenden. Ähnlich der weiter oben beschriebenen Kalibrierungsgeste muss die Stoppgeste über eine Zeitdauer von ca. drei Sekunden beibehalten werden.

Hierzu hebt der Benutzer einen seiner beiden Arme derart nach oben an, dass sich die Hand des Armes oberhalb des Kopfes befindet. Gleichzeitig muss er die andere Hand unter den Ellenbogen des angehobenen Arms halten, so dass sich beide Hände auf der gleichen Körperhälfte befinden (vgl. Abbildung 4.7). Es spielt dabei keine Rolle, welchen seiner beiden Arme der Benutzer anhebt, denn die Geste ist symmetrisch für beide Möglichkeiten definiert. Diese Stoppgeste muss anschließend für eine Dauer von ca. drei Sekunden gehalten werden. Nach erfolgreicher Durchführung wird die aktive Steuerung beendet und der Benutzer wird visuell darüber informiert (vgl. Abschnitt 4.2.5). Eine konkrete Beschreibung zur Erkennung der Stoppgeste wird in Abschnitt A.1.7 des Anhangs gegeben.

4.2. Software

In diesem Abschnitt wird die Software vorgestellt, die zur Erkennung der Gesten und zur Integration der gestenbasierten Steuerung in die Konfiguration des *PentAI* entwickelt wurde. Es wird zuerst ein Konzept vorgestellt, in dem die Relation der entwickelten Softwarekomponenten beschrieben wird. Anschließend wird jede Komponente im Detail betrachtet.

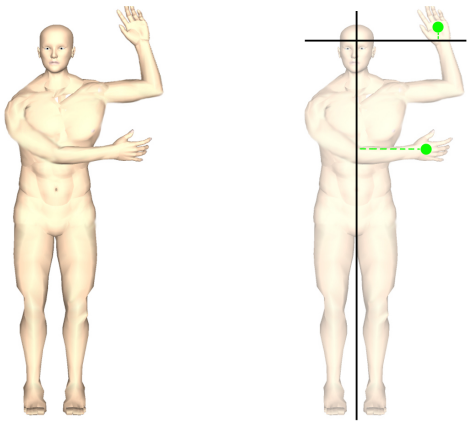


Abbildung 4.7.: Stoppgeste zur Beendigung der Navigation

4.2.1. Architektur

Die Entwicklung der Software zur Integration einer gestenbastierten Steuerung des *PentAI* umfasst drei Teile:

- Ein **Socketserver**, welcher die Informationen des eingesetzten *Kinect Sensors* verarbeitet und bei Anfrage eines Clients an diesen weiterleitet. Zur Verarbeitung der Informationen zählt dabei u.a. die Identifikation der in Abschnitt 4.1 definierten Ganzkörpergesten und die daraus resultierende Bewegungsänderung für die virtuelle Welt.
- Ein **Lightningmodul**, das als Client den Kommunikationspartner für den zuvor genannten Socketserver bildet.
- Ein **VRfx-Plugin**, durch das die Konfiguration der gestenbasierten Steuerung zur Laufzeit über eine grafische 2D-Benutzeroberfläche ermöglicht wird.

In Abbildung 4.8 wird die Beziehung zwischen diesen drei Komponenten schematisch dargestellt. Die Abbildung beschreibt, wie die Sensordaten vom *Socketserver* verarbeitet und über eine UDP-Verbindung an das *Lightningmodul* übertragen werden. Das *VRfx/Lightning* Framework transformiert die empfangenen Informationen in eine Positions- und Orientierungsveränderung der Kamera im virtuellen Raum und passt entsprechend die visuelle Ausgabe im *PentAI* an.

4.2.2. Socketserver

Der Socketserver wurde mit der Programmiersprache Java entwickelt und wird auf dem *Master* des *PentAI* ausgeführt. Er dient zur Verarbeitung der vom *Kinect Sensor* erfassten Positionsdaten eines Benutzers und zur Identifikation der in Abschnitt 4.1 beschriebenen Gesten. Zur Verarbeitung der Sensordaten werden die beiden Frameworks *OpenNI* und *NITE* von PrimeSense eingesetzt. Der Socketserver bildet zudem das Bindeglied zwischen dem eingesetzten 3D-Eingabegerät und der VR-Umgebung *PentAI*.

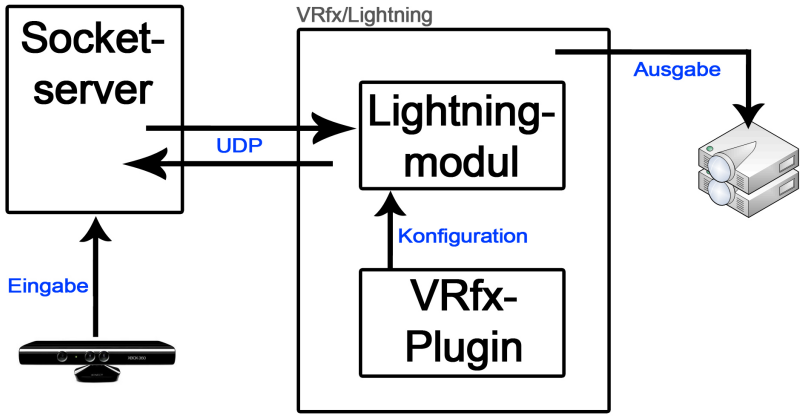


Abbildung 4.8.: Beziehung zwischen *Socketserver*, *Lightningmodul* und *VRfx*-Plugin

Die Entscheidung, Java als Programmiersprache für die Implementierung des Socketservers zu benutzen, gründet in der Tatsache, dass ein zusätzliches visuelles Feedback für den Benutzer hiermit schnell und einfach produziert werden kann ohne dabei Rücksicht auf den darunterliegenden Fenstermanager der eingesetzten Ubuntu Unix Version zu nehmen. Ein weiterer Vorteil von Java liegt in der Portabilität des fertigen Produkts. Ein einmal kompilierter Socketserver kann somit einfach auf ein anderes System kopiert werden. Zudem unterstützen die beiden Frameworks *OpenNI* und *NITE* die Entwicklung mit Java.

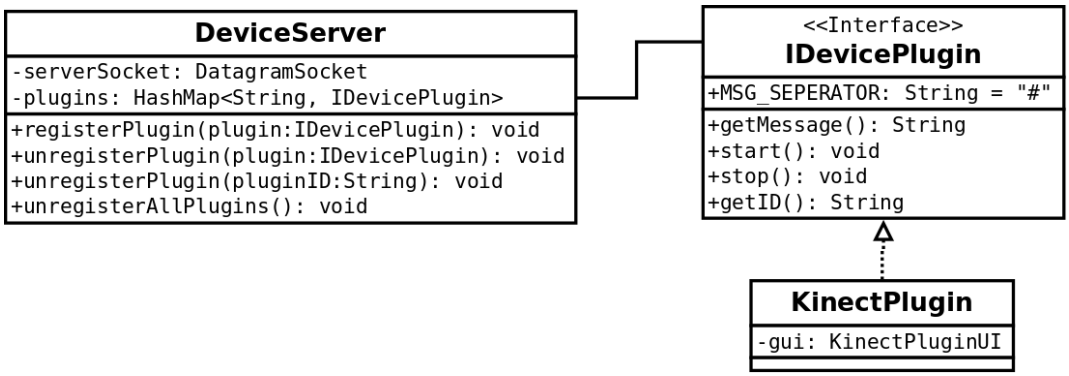


Abbildung 4.9.: Vereinfachtes Klassenmodell des Socketservers

Das Konzept des Socketservers basiert auf einem Plugin-basierten Modell (vgl. Abbildung 4.9). Der Server repräsentiert somit einen Container für Plugins, dessen Hauptaufgabe in der Verwaltung der Kommunikation mit den Clients besteht. Plugins hingegen können unterschiedliche Funktionalitäten anbieten und sind eine Realisierung der Schnittstelle *IDevicePlugin*. Sie können zur Laufzeit am Socketserver registriert werden und stellen auf Anfrage eines Clients Informationen über ihren aktuellen Zustand bereit. Dieses Konzept erlaubt eine einfache Erweiterung des Socketservers durch die Entwicklung und Integration neuer Plugins. Ein Beispiel hierfür wäre die Entwicklung eines zusätzlichen Plugins

zur Erkennung neuer Gestendefinitionen oder der Kommunikation mit einem anderen Eingabegerät als dem *Kinect Sensor*.

Ein Plugin ist durch eine eindeutige Kennung definiert, die während einer Kommunikation zwischen Client und Socketserver vom Client genutzt wird, um eine gezielte Datenabfrage an den Server zu stellen. Für die Implementierung dieser Arbeit wurde ein solches Plugin mit der Kennung *PENTAI2KINECT* entwickelt. Es liefert einem anfragenden Client Informationen zu aktuellen Bewegungsänderungen gemäß der in Abschnitt 4.1 beschriebenen Gesten. Im folgenden Abschnitt wird genauer auf die Funktion und den Aufbau dieses Plugins eingegangen.

Ein ausführlicheres Klassendiagramm des Socketservers wird in Abschnitt A.2 des Anhangs gegeben.

Serverplugin

Das Plugin mit der Kennung *PENTAI2KINECT* übernimmt zur Laufzeit des Socketservers die Identifikation von Ganzkörpergesten sowie die Berechnung der daraus resultierenden Bewegungsänderungen in der virtuellen Welt. Hierzu werden die beiden Frameworks *OpenNI* und *NITE* eingesetzt, welche die Daten des *Kinect Sensors* verarbeiten.

Nachdem ein Benutzer die gestenbasierte Steuerung des *PentAI* durch Ausübung der Kalibrierungsgeste (vgl. Abschnitt 4.1.2) gestartet hat, registriert das Plugin fortwährend jede Bewegung. Dabei werden wichtige Schlüsselpunkte am Körper des Benutzers (vgl. Abbildung 3.10) kontinuierlich verfolgt, da deren Position im Raum Grundlage für die Erkennung der benutzten Ganzkörpergesten ist. Immer dann, wenn ein Client eine Datenabfrage für das Plugin mit der Kennung *PENTAI2KINECT* an den Socketserver richtet, analysiert das Plugin die aktuellen Positionen dieser Schlüsselpunkte und generiert daraus ein Datenpaket, das Geschwindigkeit und Richtung aller sechs Gesten beinhaltet, die zur Bewegung im virtuellen Raum dienen. Dieses Datenpaket wird in Form einer Zeichenkette an den Server übermittelt. Der Server wiederum leitet das Datenpaket an den anfragenden Client weiter.

Die Struktur des zuvor beschriebenen Datenpakets setzt sich dabei wie folgt zusammen:
Jede der sechs möglichen Bewegungen wird durch eine Zahl mit einem ganzzahligen Wert zwischen -100 und 100 definiert, wobei 0 für *keine Bewegung* steht. Das Vorzeichen definiert dabei die Richtung der Bewegung und der Betrag der Zahl ihre Geschwindigkeit. Der Geschwindigkeitswert ist frei gewählt und kann über das *VRfx*-Plugin (vgl. 4.2.4) zur Laufzeit vom Benutzer skaliert werden, um die Geschwindigkeit der Bewegungen an eine bestimmte virtuelle Welt anzupassen. Zwischen den sechs Zahlenwerten wird das „#“-Zeichen als Trennung benutzt. Die Zeichenkette **"65#-10#0#0#30#0"** identifiziert also zum Beispiel eine Bewegungsänderung nach vorne mit einer Geschwindigkeit von 65, nach links mit einer Geschwindigkeit von 10, keine Auf- oder Abwärtsbewegung, keine *Yaw*-Bewegung, eine *Pitch*-Bewegung mit der Geschwindigkeit 30 und keine *Roll*-Bewegung. Kann keine Bewegung festgestellt werden oder wurde die Eingabe durch den Benutzer beendet, so wird

auf eine Anfrage mit der Zeichenkette **"0#0#0#0#0#0"** geantwortet. In Abbildung 4.10 wird eine erfolgreiche Kommunikation zwischen einem anfragenden Client, dem Socketserver und dem Serverplugin in Form eines Sequenzdiagramms dargestellt.

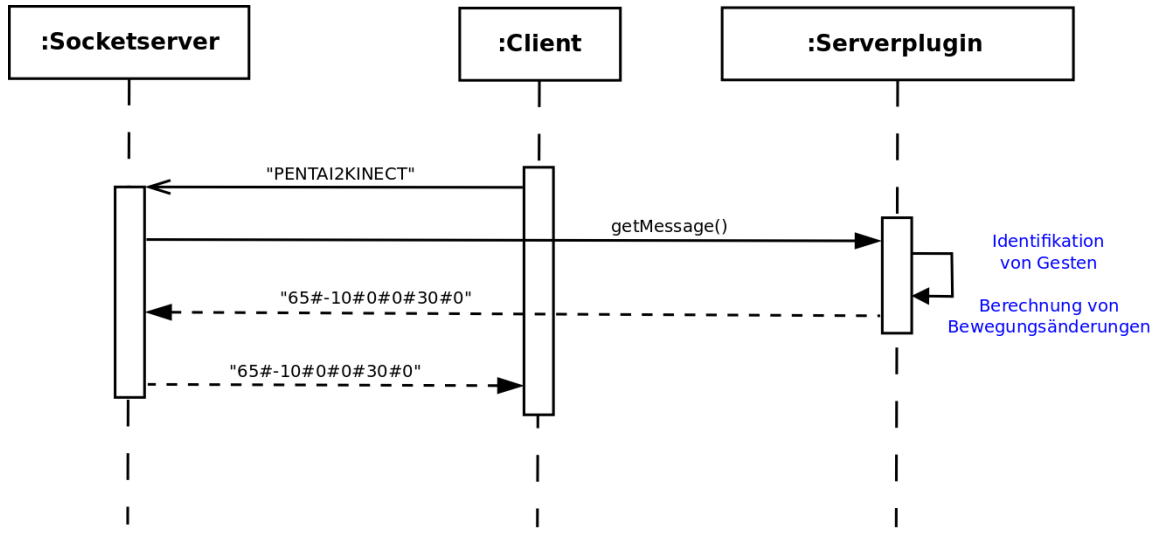


Abbildung 4.10.: Sequenzdiagramm einer erfolgreichen Client-Server-Kommunikation

Das Plugin kann nach der Initialisierung der Frameworks *OpenNI* und *NITE* fünf logische Zustände einnehmen (vgl. Abbildung 4.11). Im Einzelnen sind das die Zustände *Suche Benutzer*, *Suche Kalibrierungsgeste*, *Kalibrierung*, *Weitere Kalibrierung* und *Verfolgung*. Nach seiner Initialisierung befindet sich das Plugin im Zustand *Suche Benutzer* und wartet darauf, dass das *OpenNI* Framework eine Person im Sichtfeld des *Kinect Sensors* erkennt. Im Zustand *Suche Kalibrierungsgeste* verharrt das Plugin solange, bis eine erkannte Person die in Abschnitt 4.1.2 beschriebene Kalibrierungsgeste ausführt. Die Kalibrierung selbst wird innerhalb des Plugins durch zwei Zustände beschrieben. Im Zustand *Kalibrierung* wird zunächst der Kalibrierungsprozess des eingesetzten *OpenNI* Frameworks durchgeführt. Nach einer erfolgreichen Kalibrierung wird eine zusätzliche Kalibrierungsphase durch den Zustand *Weitere Kalibrierung* eingeführt. Hierzu sei erwähnt, dass in der eingesetzten Version des *OpenNI* Frameworks die Kalibrierung entgegen älterer Versionen in zwei Teile unterteilt wurde: eine erste, grobe und eine zweite, feinere Kalibrierungsphase. Allerdings ist eine Unterscheidung dieser beiden Phasen mit der eingesetzten Version des Frameworks nicht möglich. Nach der ersten, groben Kalibrierungsphase sind die empfangenen Sensordaten jedoch instabil und deshalb ungeeignet für eine präzise Navigation. Auf Grundlage einfacher Messungen kann die Zeit für die zweite Phase im Durchschnitt mit zehn Sekunden beziffert werden. Diesem Faktum wird im Zustand *Weitere Kalibrierung* Rechnung getragen, indem eine Verzögerung des Kalibrierungsprozesses simuliert wird. Nach dieser Verzögerung befindet sich das Plugin im Zustand *Verfolgung*. Nur wenn sich das Plugin in diesem Zustand befindet, werden erkannte Bewegungsänderungen nach Anfrage an den Server übermittelt.

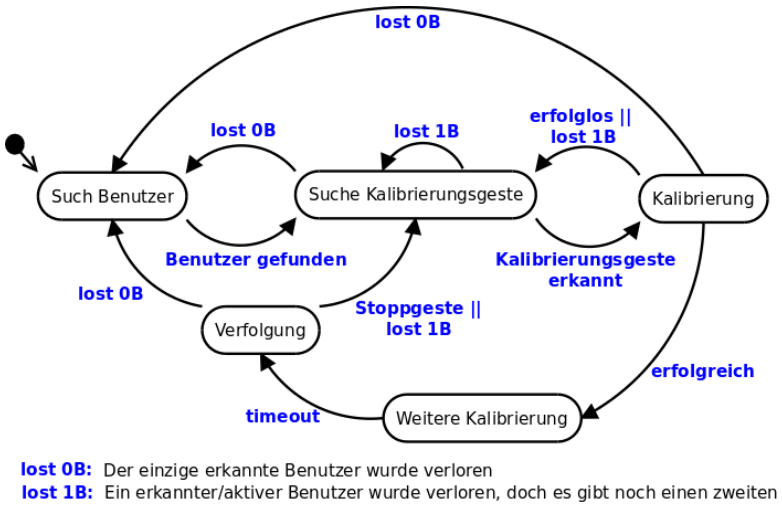


Abbildung 4.11.: Zustände des Serverplugins

4.2.3. Lightningmodul

Der zweite Teil der Implementierung umfasst die Entwicklung eines Lightningmoduls wie es in Abschnitt 3.2.3 vorgestellt wurde. Das entwickelte Lightningmodul mit dem Namen *ltDeviceKinect* dient dabei der Umsetzung der vom Socketserver stammenden Bewegungsänderungen in Datenstrukturen, die das *VRfx/Lightning* Framework interpretieren kann. Bei der Kommunikation mit dem Socketserver nimmt das Modul dabei die Rolle des anfragenden Clients ein. Es wurde als Erweiterung eines der mit *Lightning* ausgelieferten Beispiele in der Programmiersprache C++ entwickelt. Dieses soll im Folgenden beschrieben werden.

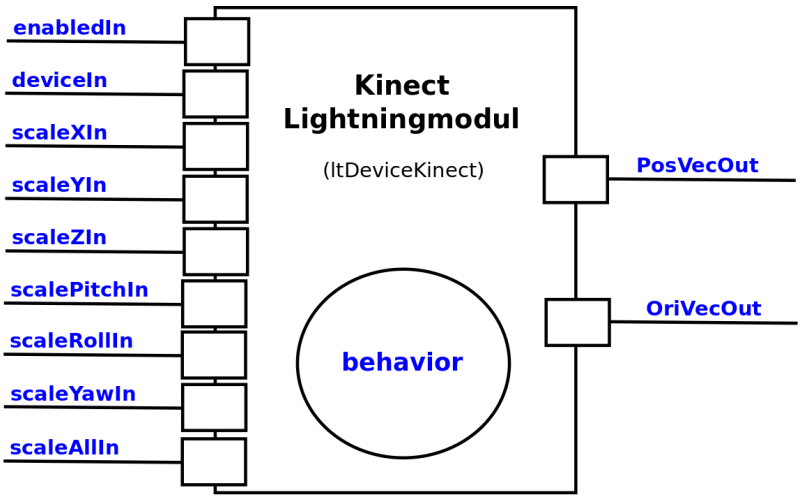


Abbildung 4.12.: Das Lightningmodul *ltDeviceKinect* mit Ein- und Ausgängen

Das Modul besitzt insgesamt neun Eingänge sowie zwei Ausgänge (vgl. Abbildung 4.12) und eine Updatemethode, genannt „*behavior*“. Der Eingang *enabledIn* dient zur Aktivierung bzw. Deaktivierung des Moduls. An den Eingang *deviceIn* kann ein Name für das

Modul in Form einer Seriennummer angelegt werden. Die Eingänge *scaleXIn*, *scaleYIn*, *scaleZIn*, *scaleYawIn*, *scalePitchIn*, *scaleRollIn* sowie *scaleAllIn* werden genutzt, um die Geschwindigkeit der Bewegungsänderung so zu skalieren, dass eine perfekte Anpassung des Interaktionsmodells an verschiedene virtuelle Szenarien möglich ist. Innerhalb der Updatemethode werden hierzu die Werte der vom Server ankommenden Bewegungsänderungen mit den anliegenden Werten zur Skalierung multipliziert. Zum Beispiel kann die Geschwindigkeit einer translativen Vorwärtsbewegung entlang der z-Achse durch den Wert 0,5 am Eingang *scaleZIn* halbiert werden. Die Konfiguration dieser Werte kann vom Benutzer im grafischen Visualisierungseditor *VRfx* über das im nächsten Abschnitt 4.2.4 vorgestellte *VRfx*-Plugin vorgenommen werden. Ein am Eingang *scaleAllIn* anliegender Wert skaliert entsprechend die Geschwindigkeit aller Bewegungen gleichermaßen. Die beiden Ausgänge *PosVecOut* und *OriVecOut* liefern einem daran angeschlossenen Modul die skalierten Bewegungsänderungen. Die Werte an den Ausgängen liegen in Form eines Vektors mit drei Elementen vor. *PosVecOut* beschreibt dabei eine Translation (entlang der x-, y- und z-Achse) und *OriVecOut* entsprechend eine Rotation (um die x-, y- bzw. z-Achse).

Im *Eventgraphen* von *Lightning* nimmt das Modul *ltDeviceKinect* die Rolle einer Art *Sensor*-Objekt (vgl. Abschnitt 3.2.3) ein. Das eigentliche *Sensor*-Objekt allerdings wird vom Objekt *clKinect* beschrieben, welches *ltDeviceKinect* in sich kapselt. *ClKinect* gibt die von *ltDeviceKinect* skalierten Bewegungsänderungen an das globale *Motion*-Objekt des Eventgraphen weiter, welches den Namen *clMotionKinect* trägt. Es ist eine Spezialisierung des *Lightning*-Objektes *clMotion3DTracker* und sorgt für die Berechnung der nötigen Positions- und Orientierungsänderung der an ihm angeschlossenen *Kamera*-Objekte. Diese Änderungen werden dann auch über entsprechenden Routen an die *Kamera*-Objekte weitergegeben. Abbildung 4.13 stellt diese Beziehung noch einmal schematisch dar.

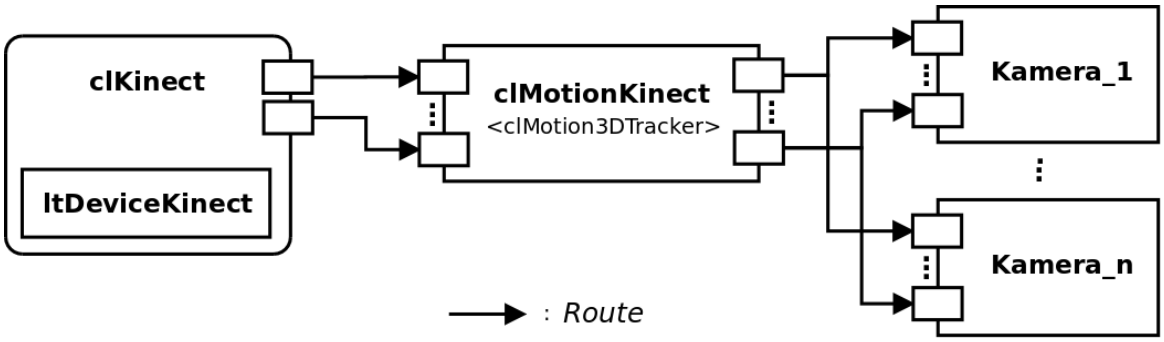


Abbildung 4.13.: Einbindung des Moduls *ltDeviceKinect* im *Lightning* Eventgraphen

4.2.4. VRfx-Plugin

Als dritter Teil der Implementierung wurde neben dem Socketserver und dem Lightningmodul auch ein grafisches 2D-Konfigurationsmenü in Form eines Plugins für den Visualisierungseditor *VRfx* entwickelt (vgl. Abbildung 4.14).

Über das Konfigurationsmenü des Plugins werden dem Benutzer folgende Interaktionsmöglichkeiten angeboten: Das Starten und Stoppen der Kommunikation zwischen Socketserver und dem Lightningmodul, indem dieses aktiviert oder deaktiviert wird und die Skalierung der Bewegungsgeschwindigkeiten. Hierzu wurde die Datei *KinectNI.ltfxscr* erstellt, in welcher die nötigen visuellen Komponenten des 2D-Menüs definiert wurden. Darunter zählen sieben *Slider*-Komponenten zur Skalierung der Geschwindigkeiten sowie zwei *Button*-Komponenten zum Aktivieren und Deaktivieren des Lightningmoduls. Änderungen innerhalb dieses Konfigurationsmenüs haben eine sofortige Auswirkung auf das Modul *clKinect*, das wiederum die Konfigurationen an die entsprechende Eingänge des Lightningmoduls setzt.

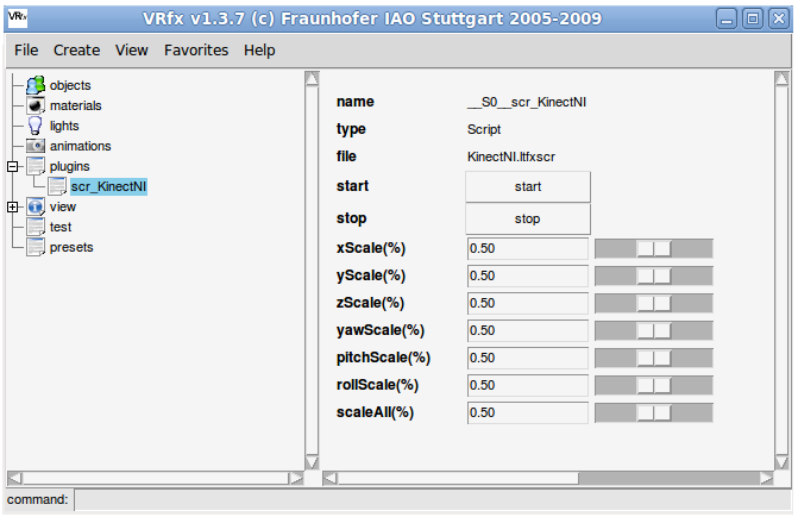


Abbildung 4.14.: 2D-Benutzeroberfläche des *VRfx*-Plugins

4.2.5. Anwendung im PentAI

In diesem Abschnitt wird die konkrete Anwendung der gestenbasierten Steuerung im *PentAI* beschrieben. Dabei wird gezeigt, wie ein Benutzer die Steuerung startet, eine Bewegung ausführen und deren Geschwindigkeit mit Hilfe des *VRfx*-Plugins beeinflussen kann.

Zu Beginn startet der Benutzer dazu das Skript *startClusterCeltechKinect* aus seiner *VRfx/-Lightning* Installation. Damit bewirkt er zum einen die Initialisierung einer clusterbasierten Konfiguration des *PentAI* und zum anderen den Start des Socketservers auf dem *Master* innerhalb des Clusters. Im Anschluss kann der Benutzer eine beliebige *VRfx*-Session oder einzelne 3D-Modelle laden, so dass diese im *PentAI* visualisiert werden. Um nun die gestenbasierte Steuerung über den *Kinect Sensor* zu aktivieren, lädt der Benutzer das *VRfx*-Plugin *KinectNI.ltxsrc* und wechselt die Ansicht zum 2D-Konfigurationsmenü des geladenen Plugins (vgl. Abbildung 4.14). Durch Betätigung der Schaltfläche *Start* beginnt die Steuerung.

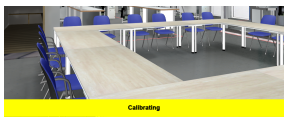
Gemäß der in Abschnitt 4.2.2 beschriebenen Zustände des Serverplugins befindet sich die Interaktion zu Beginn im Zustand *Suche Benutzer*, was dem Benutzer über eine Grafik innerhalb der virtuellen Welt angezeigt wird (vgl. Abbildung 4.15 [a]). Der Benutzer muss sich nun, der Aufforderung entsprechend, in das Sichtfeld des Sensors begeben. Die Grafik ändert sich daraufhin und fordert den erkannten Benutzer zur Ausführung der Kalibrierungsgeste auf. Nach einer erfolgreichen Kalibrierung verschwindet die Grafik und der Benutzer ist in der Lage, mithilfe der in Abschnitt 4.1 definierten Ganzkörpergesten, Bewegungen in der virtuellen Welt auszuführen. Anfangs ist die Steuerung für den Zeitraum von etwa zehn Sekunden jedoch spürbar instabil und unpräzise, was dem Benutzer über einen orangen Balken am unteren Rand der mittleren Leinwand angezeigt wird. Der orange Balken weist den Benutzer auf den Zustand *Weitere Kalibrierung* des Serverplugins hin und ändert seine Farbe nach Ablauf der zehn Sekunden automatisch zu grün. Das Serverplugin befindet sich dann im Zustand *Verfolgen* und erlaubt eine stabile und gestenbasierte Navigation in der virtuellen Welt. Über das 2D-Konfigurationsmenü kann der Benutzer jederzeit die Geschwindigkeit der Bewegungen durch Verändern der Skalierung beeinflussen. Somit kann er die Steuerung für jede virtuelle Welt individuell anpassen.

Durch Ausführung der Stoppgeste kann ein Benutzer die Interaktion wieder in den Zustand *Suche Benutzer* bzw. *Suche Kalibrierungsgeste* zurücksetzen. Verliert das Serverplugin während der Navigation den erkannten Benutzer, so wird die zuvor genannte Zustandsänderung automatisch vollzogen.

Durch die Betätigung der Schaltfläche *Stop* im 2D-Konfigurationsmenü des *VRfx*-Plugins trennt der Benutzer die Verbindung zwischen Lightningmodul und Socketserver und setzt somit die gestenbasierte Steuerung des *PentAI* aus.

(a) Suche Benutzer

(b) Suche Geste



(c) Kalibrierung

(d) Weitere Kalibrierung

(e) Verfolgung

Abbildung 4.15.: Visuelle Rückmeldung der fünf Interaktionszustände

5. Schlussbetrachtung

In diesem Kapitel werden die Kernkompetenzen dieser Arbeit noch einmal zusammengefasst und ein Fazit gezogen. Im Anschluss daran wird ein Ausblick über mögliche Erweiterungen der vorgestellten Implementierung gegeben.

5.1. Zusammenfassung

Das Ziel dieser Arbeit war die Implementierung einer gestenbasierten Steuerung der VR-Umgebung *PentAI* in *sechs Freiheitsgraden* mit dem Microsoft *Kinect Sensor*.

Zunächst wurde zum besseren Verständnis des theoretischen Hintergrunds der Begriff *Virtuelle Realität* detailliert betrachtet. Dabei wurde mit Bezug zur Implementierung auch auf die Begriffe *Immersion* und *sechs Freiheitsgrade* eingegangen und das Interaktionsmodell der Bewegung in einer virtuellen Welt näher beleuchtet. Darauf aufbauend wurden vier Vertreter gängiger 3D-Eingabegeräte vorgestellt und verglichen. Es wurde außerdem herausgestellt, dass ein hoher Grad der Immersion einen wesentlichen Einfluss auf das didaktische Potential immersiver VR-Umgebungen hat, was anschließend mit existierenden Beispielanwendungen veranschaulicht wurde.

Im dritten Kapitel wurden die technischen Rahmenbedingungen für die Implementierung der vorliegenden Arbeit aufgezeigt und die eingesetzten Hard- und Softwareprodukte im Einzelnen beschrieben. Dabei wurde die VR-Umgebung *PentAI*, der *Kinect Sensor* und die beiden Frameworks *OpenNI* und *NITE* vorgestellt.

Mit dem Wissen über den Einfluss der Mensch-Maschine-Schnittstelle auf den Grad der Immersion eines VR-Systems wurde dann im vierten Kapitel die Implementierung der neuen und gestenbasierten Steuerung des *PentAI* beschrieben. Hierzu wurde zunächst ein Konzept vorgestellt zur Erfassung von Ganzkörpergesten, die eine Navigation innerhalb einer virtuellen Welt ermöglichen. Es wurden insgesamt acht Gesten definiert, die zur eigentlichen Bewegung im virtuellen Raum und zum Starten bzw. zum Beenden der Steuerung verwendet werden. Mit dem Ziel der Steigerung des Immersionsempfindens gegenüber der bisherigen Eingabemethode mit dem *SpaceNavigator* (vgl. Abschnitt 2.2) wurden diese Gesten dabei so definiert, dass sie für einen Benutzer des *PentAI* intuitiv zu benutzen und leicht erlernbar sind. Damit dies erreicht werden konnte, wurde auf eine starke Abstraktion entsprechender Bewegungsmuster der realen Welt verzichtet und eine möglichst direkte Umsetzung natürlicher Bewegungen in entsprechende virtuelle Bewegung verwendet. Nach der Definition der Gesten wurde eine Beschreibung der Software gegeben, die zur Verarbei-

tung der Sensordaten und zur Integration der neuen Interaktionsform in der VR-Umgebung *PentAI* nötig waren. Hierzu zählten im Einzelnen:

- Die Entwicklung eines plugin-basierten *Socketservers*, der zur Identifikation der Gesten und zur Bereitstellung erkannter Bewegungsänderungen innerhalb der Clusterkonfiguration des *PentAI* dient.
- Ein *Lightningmodul*, das als Client die erkannten Bewegungsänderungen vom Socketserver empfängt und in entsprechender Form in die Simulationsschleife des *Lightning* Frameworks integriert.
- Ein *VRfx*-Plugin mit einer grafischen 2D-Benutzeroberfläche zur Steuerung des Lightningmoduls. Dazu zählt der Aufbau der Kommunikation zwischen Lightningmodul und Socketserver sowie die Skalierung der Bewegungsgeschwindigkeit in der virtuellen Welt.

5.2. Ausblick

Diese Arbeit stellt die Grundlage für eine Reihe gestenbasierter Interaktionsmodelle für das *PentAI* bereit. Neben der hier vorgestellten Möglichkeit einer Bewegung im virtuellen Raum und in *sechs Freiheitsgraden* kann die gestenbasierte Interaktion auch durch weitere Interaktionsformen ergänzt werden. So könnten beispielsweise neue universelle Gesten zur Selektion und Manipulation von Eigenschaften virtueller Objekte integriert werden. Des Weiteren könnten individuell angepasste Interaktionsmodelle für konkrete virtuelle Welten entwickelt werden. Ein Beispiel dafür ist die virtuelle Welt von *CyberMath* (vgl. Abschnitt 2.4.3). In ihr könnte zum einen eine gestenbasierte Interaktion dazu genutzt werden, einen Avatar durch die virtuellen Ausstellungsräume zu bewegen, zum anderen könnten gezielte Gesten etwa zum Greifen und Manipulieren von Objekten integriert werden.

Ähnlich der in Pletcher et al. (2000) vorgestellten VR-Umgebung MRT könnte eine kostengünstige immersive VR-Umgebung für Medizinstudenten entwickelt werden, in der die Studenten medizinische Lerninhalte virtuell erfahren, indem sie etwa durch Blutbahnen „fliegen“ oder an virtuellen Organen experimentieren können. Auch hierzu wäre, neben der Entwicklung der virtuellen Welt, lediglich eine Erweiterung der Gestendefinitionen aus Kapitel 4 nötig.

Im Bereich der medizinischen Diagnostik könnten Interaktionsmodelle erstellt werden, die bei der Identifikation bestimmter Krankheiten des Bewegungsapparates helfen. Dazu würde man die bisherige Konfiguration des *PentAI* mit einem Laufband in der Mitte des Raumes erweitern. Man könnte dann etwa Fehlstellungen im Gang der Patienten anhand entsprechender Gestendefinitionen erkennen. Mit seiner Implementierung der Johansson Experimente in einer VR-Umgebung hat Gu (2011) in seiner Arbeit bereits ein Beispiel dafür gegeben, wie sich bestimmte physische Eigenschaften oder Gefühlszustände im Gang einer Person widerspiegeln und nur anhand weniger Punkte visualisiert werden können.

Für Kinder ließen sich interaktive und immersive Lernwelten erstellen, in denen abstrakte Lerninhalte aus unterschiedlichsten Bereichen in einer spielerischen Weise vermittelt werden. Winkelgrößen könnten zum Beispiel durch die Nachbildung eines dargestellten Winkels mit den eigenen Armen direkt am eigenen Körper erfahren werden. Zum anderen ließen sich interaktive Trainer in Form intelligenter Avatare in die virtuelle Welt integrieren.

Man erkennt schnell, dass die flexible Definition von Hand- und Ganzkörpergesten grundsätzlich eine Vielzahl an Interaktionsmöglichkeiten in den verschiedensten Anwendungsbereichen bietet.

Es bleibt zu zeigen, in welchem Maß sich das vorgestellte Interaktionsmodell dieser Arbeit letztendlich auf den Grad des Immersionsempfindens eines Benutzers im *PentAI* auswirkt. Hierzu muss das beschriebene Interaktionsmodell zur freien Bewegung in *sechs Freiheitsgraden* anhand einer weiterführenden Studie evaluiert werden. Eine solche Evaluierung könnte dann konkrete Hinweise auf die erreichte Steigerung der Immersion bei der Verwendung des *Kinect Sensors* gegenüber des *SpaceNavigators* geben. Sie könnte zudem Aufschluss darüber geben, inwieweit die definierten Gesten für den Benutzer des *PentAI* wirklich intuitiv zu handhaben sind.

5.3. Fazit

Die vorgestellte Implementierung einer gestenbasierten Steuerung des *PentAI* mit dem *Kinect Sensor* hat gezeigt, dass eine natürliche Mensch-Maschine-Schnittstelle gegenüber einer 3D-Maus eine neue Art der Erfahrung bei der Interaktion mit dem *PentAI* bietet. Während bei einer 3D-Maus wie dem *SpaceNavigator* Bewegungen in der virtuellen Welt lediglich über einen beweglichen Puck am Eingabegerät erfahren werden, so kann dies bei der Eingabe über intuitive Ganzkörpergesten über den gesamten Körper geschehen. Der damit verbundene Anstieg des Immersionsempfindens hat wiederum einen positiven Effekt auf das didaktische Potential der VR-Umgebung, in der eine solche gestenbasierte Steuerung benutzt wird (Boytscheff, 2006). Die Beispiele in Abschnitt 2.4 zeigen, dass VR-Systeme bereits erfolgreich zur Vermittlung von Lerninhalten eingesetzt werden. Bisher sind solche Systeme jedoch meist noch sehr kostspielig und daher nicht weit verbreitet. Das *PentAI* stellt eine kostengünstige Alternative zu teuren immersiven VR-Systemen dar und in Kombination mit der Implementierung dieser Arbeit unterstützt das *PentAI* zudem eine flexible und intuitive sowie natürliche Mensch-Maschine-Schnittstelle.

Die Implementierung der vorliegenden Arbeit kann also als Grundlage genutzt werden, um angepasste und inhaltsbezogene Interaktionsmodelle für das *PentAI* zu entwickeln, die auf intuitiven Hand- und Ganzkörpergesten basieren.

A. Anhang

Im Folgenden sollen einige wichtige Aspekte der Implementierung dieser Arbeit noch einmal näher beschrieben werden. Dazu werden zum einen die Definitionen der Ganzkörpergesten aus Abschnitt 4.1 in Form von Pseudocode erklärt und zum anderen ein detailliertes Klassendiagramm für den Socketserver und das Serverplugin aus Abschnitt 4.2.2 gegeben.

A.1. Gestendefinitionen

Nachfolgend wird die Erkennung der in Abschnitt 4.1 beschriebenen Gesten anhand von Pseudocode erklärt. Es soll herausgestellt werden, welche Informationen nötig sind und wie die Berechnung von Geschwindigkeit und Richtung der virtuellen Bewegung erfolgt. Um die Algorithmen zu vereinfachen und den Fokus auf deren Kernaussage zu legen, werden die Eingabewerte dabei jeweils als gegeben angesehen.

A.1.1. Links-/Rechtsbewegung

Eine Links- bzw. Rechtsbewegung wird wie folgt erkannt und berechnet:

Algorithmus 1 Erkennung und Bestimmung einer Links- bzw. Rechtsbewegung

Eingabe: *pX*: Initiale Position des Benutzers auf der x-Achse; *qX*: Aktuelle Position des Benutzers auf der x-Achse; *t0*: Offset; *tMax*: Maximaler Bewegungsradius

Ausgabe: *out*: Richtung und Geschwindigkeit einer erkannten Links- bzw. Rechtsbewegung ($-100 \leq out \leq 100$). Ein negativer Wert indiziert eine Linksbewegung, ein positiver Wert eine Rechtsbewegung

```
1:  distance = qX - pX;
2:  if abs( distance ) > t0 then
3:      out = min ( ( abs ( distance ) - t0 ) / ( tMax - t0 ) * 100, 100 );
4:      if dist < 0 then
5:          out = -out;
6:      end if
7:  else
8:      out = 0;
9:  end if
10: return  out;
```

In Zeile 3 hat der Benutzer den Offset-Bereich verlassen und die Geschwindigkeit der Bewegung wird über den Abstand zwischen der aktuellen Position und der initialen Position des Benutzers minus dem definierten Offset *t0* bestimmt. Über das Verhältnis zum maximalen Bewegungsradius *tMax* wird die Geschwindigkeit an die verwendete Skala angepasst.

A.1.2. Vorwärts-/Rückwärtsbewegung

Die Berechnung der virtuellen Vorwärts- bzw. Rückwärtsbewegung ergibt sich analog zur Bestimmung der Links-/Rechtsbewegung. Es wird hierbei allerdings nicht die Position des Benutzers auf der x-Achse, sondern auf der z-Achse verwendet.

A.1.3. Auf-/Abwärtsbewegung

Eine Auf- bzw. Abwärtsbewegung wird wie folgt erkannt und berechnet:

Algorithmus 2 Erkennung und Bestimmung einer Auf- bzw. Abwärtsbewegung

Eingabe: *inActionArea*: Sind beide Hände außerhalb des Offset-Balls? (true|false); *distHands*: Distanz zwischen den beiden Händen eines Benutzers; *distHips*: Distanz zwischen linker und rechter Hüfte; *lowerArmLength*: Die Länge des Unterarms des Benutzers; *mHandsY*: Position des Mittelpunkts der Strecke zwischen beiden Händen des Benutzers auf der y-Achse; *mSchouldersY*: Position des Mittelpunkts der Strecke zwischen linker und rechter Schulter auf der y-Achse; *mHipsY*: Position des Mittelpunkts der Strecke zwischen linker und rechter Hüfte auf der y-Achse;

Ausgabe: *out*: Richtung und Geschwindigkeit einer erkannten Auf- oder Abwärtsbewegung ($-100 \leq out \leq 100$). Ein negativer Wert indiziert eine Abwärtsbewegung, ein positiver Wert eine Aufwärtsbewegung

```
 1: if inActionArea and distHands > ( distHips + lowerArmLength * 2.8 ) then
 2:    height = ( mShoulderY - mHipsY ) * 0.2;
 3:    if mHandsY > ( mHipsY + height ) and mHandsY < ( mShouldersY - height )
           or mHandsY > ( mSchoulersY + lowerArmLength * 0.5 ) then
 4:       out = 0;
 5:    else if mHandsY < ( mHipsY + height ) then
 6:       out = max( -100, -( 100 - ( mHandsY - mHipsY ) / height * 100 ) );
 7:    else if mHandsY > ( mShouldersY - height ) then
 8:       out = min( 100, ( mHandsY - ( mShouldersY - interHeight ) ) / height * 100);
 9:    end if
10: else
11:    out = 0;
12: end if
13: return out;
```

In Zeile 1 wird geprüft, ob beide Hände des Benutzers außerhalb des Offset-Balls sind. Sie müssen außerdem einen gewissen Abstand voneinander haben, um eine Überschneidung

mit der *Yaw*-Bewegung zu vermeiden. In Zeile 2 wird die Höhe der Interaktionsbereiche bei der Auf- bzw. Abwärtsbewegung bestimmt. Sie wird in diesem Fall mit einer Höhe von $\frac{1}{5}$ der Strecke von den Hüften bis zu den Schultern definiert. Zeile 3 überprüft, ob sich die Hände des Benutzers zwischen dem oberen und dem unteren Interaktionsbereich befinden, also im Offset-Bereich der Geste. Zeile 5 überprüft, ob sich die Hände im oberen Interaktionsbereich befinden und Zeile 7 entsprechend, ob sich die Hände im unteren Interaktionsbereich befinden. In den beiden zuletzt genannten Fällen wird die Geschwindigkeit der virtuellen Bewegung anschließend durch die Tiefe des Eintauchens beider Hände in den jeweiligen Interaktionsbereich bestimmt.

A.1.4. Pitch-Bewegung

Eine *Pitch*-Bewegung wird wie folgt erkannt und berechnet:

Algorithmus 3 Erkennung und Bestimmung einer Pitch-Bewegung

Eingabe: *mSchouldersZ*: Position des Mittelpunkts der Strecke zwischen linker und rechter Schulter auf der z-Achse; *mHipsZ*: Position des Mittelpunkts der Strecke zwischen linker und rechter Hüfte auf der z-Achse; *distShouldersHips*: Länge der Strecke zwischen mSchouldersZ und mHipsZ; *offsetBack*: Eine Winkelgröße zur Bestimmung des Offset-Bereichs bei der Neigung nach hinten; *offsetForward*: Eine Winkelgröße zur Bestimmung des Offset-Bereichs bei der Neigung nach vorne;

Ausgabe: *out*: Richtung und Geschwindigkeit einer erkannten *Pitch*-Bewegung ($-100 \leq out \leq 100$). Ein negativer Wert indiziert eine Neigung nach vorne, ein positiver Wert indiziert eine Neigung nach hinten.

```
 1: out = asin( ( mShouldersZ - mHipsZ ) / distShouldersHips ) * 100;
 2: if out > 0 and out > offsetBack then
 3:     out = min( 90, out );
 4:     out = ( out - offsetBack ) / ( 90 - offsetBack ) * 100;
 5: else if out < 0 and abs( out ) > offsetForward then
 6:     out = max( -90, out );
 7:     out = -( ( abs( out ) - offsetForward ) / ( 90 - offsetForward ) * 100 );
 8: else
 9:     out = 0;
10: end if
11: return out;
```

Zu Beginn wird in Zeile 1 der Winkel der Oberkörperneigung eines Benutzers berechnet. Ist der Wert positiv, so wird in Zeile 2 überprüft, ob die Neigung größer als die zugehörige Winkelgröße des Offset-Bereichs ist. Ist dies der Fall, wird eine Neigung nach hinten berechnet. Analog wird ab Zeile 5 verfahren, wenn es sich um eine Neigung nach vorne handelt und der gemessene Winkelwert negativ ist.

A.1.5. Roll-Bewegung

Die Berechnung der virtuellen *Roll*-Bewegung ergibt sich analog zur Bestimmung einer *Pitch*-Bewegung. Es werden hierbei allerdings nicht die Position der Schultern und Hüften auf der z-Achse, sondern auf der x-Achse verwendet.

A.1.6. Yaw-Bewegung

Eine *Yaw*-Bewegung wird wie folgt erkannt und berechnet:

Algorithmus 4 Erkennung und Bestimmung einer Yaw-Bewegung

Eingabe: *inActionArea*: Sind beide Hände außerhalb des Offset-Balls? (true|false); *distHands*: Distanz zwischen den beiden Händen eines Benutzers; *lowerArmLength*: Die Länge des Unterarms des Benutzers; *nHandsVectorX*: Der x-Wert des Vektors zur Beschreibung der Strecke zwischen beiden Händen in normalisierter Form; *offset*: Eine Winkelgröße zur Beschreibung des Offset-Bereichs einer *Yaw*-Bewegung; *lHandX und lHandY*: Die Position der linken Hand auf der x-Achse bzw. auf der y-Achse; *rHandX und rHandY*: Die Position der rechten Hand auf der x-Achse bzw. auf der y-Achse;

Ausgabe: *out*: Richtung und Geschwindigkeit einer erkannten *Yaw*-Bewegung ($-100 \leq out \leq 100$). Ein negativer Wert indiziert eine Gierung nach links, ein positiver Wert indiziert eine Gierung nach rechts.

```
 1: if inActionArea and distHands ≤ lowerArmLength * 1.7 then
 2:    out = acos( nHandsVectorX ) * 180 / π;
 3:    if abs( out ) ≤ offset then
 4:       out = 0;
 5:    else
 6:       if lHandY > rHandY and lHandX < rHandX then
 7:          out = min( 90, abs( out ) );
 8:       else if lHandY > rHandY then
 9:          out = 90;
10:       else if lHandY < rHandY and lHandX < rHandX then
11:          out = -( min( 90, abs( out ) ) );
12:       else
13:          out = -90;
14:       end if
15:    end if
16:    out = ( abs( out ) - offset ) / ( 90 - offset ) * 100 * [ ( out > 0 ) ? 1 : (-1) ];
17: else
18:    out = 0;
19: end if
20: return out;
```

In Zeile 1 wird überprüft, ob sich beide Hände des Benutzers außerhalb des Offset-Balls befinden und ob die Distanz zwischen den Händen kleiner ist als 1,7 mal die Unterarmlänge des Benutzers. Die zweite Überprüfung ist dabei nötig, um eine Überschneidung mit der Auf- und Abwärtsgeste zu vermeiden. Zeile 6 prüft, ob die linke Hand höher als die rechte positioniert ist und ob sie sich auf der linken Körperhälfte des Benutzers befindet. Ist dies der Fall, so wird in Zeile 7 eine Gierung nach rechts angenommen. In Zeile 8 wird vermieden, dass ein größerer Winkel als 90° bei der Gierung nach rechts beschrieben wird. Analog hierzu wird in den Zeilen 10 und 12 der Fall betrachtet, bei dem die rechte Hand höher gelegen ist als die linke. Abschließend wird in Zeile 16 der Wert an die verwendete Skala zur Beschreibung von Bewegungsgeschwindigkeiten angepasst.

A.1.7. Stoppgeste

Die Stoppgeste zur Beendigung einer gestenbasierten Steuerung wird wie folgt erkannt:

Algorithmus 5 Erkennung der Stoppgeste

Eingabe: *lHandX und lHandY*: Die Position der linken Hand auf der x-Achse bzw. auf der y-Achse; *rHandX und rHandY*: Die Position der rechten Hand auf der x-Achse bzw. auf der y-Achse; *headX und headY*: Die Position des Kopfes auf der x-Achse bzw. auf der y-Achse; *cancelCounter*: Ein Thread, der eine eine Zeitdauer von drei Sekunden wartet und die Steuerung nach Ablauf der drei Sekunden beendet;

Ausgabe: -

```
1:  if (lHandY > headY and rHandX < headX)
        or (rHandY > headY and lHandX > headX) then
2:    if cancelCounter is null or cancelCounter is not running then
3:      Create new instance of cancelCounter;
4:      Start cancelCounter;
5:    end if
6:  else
7:    if cancelCounter is not null and cancelCounter is running then
8:      Interrupt cancelCounter;
9:    end if
10: end if
```

In Zeile 1 wird geprüft, ob eine der beiden Ausführungsmöglichkeiten der Stoppgeste angewendet wurde. Falls die Stoppgeste erkannt wurde, so wird in Zeile 2 geprüft, ob ein Thread zur Beendigung der Gestensteuerung schon existiert und in Zeile 3 und 4 eine Instanz eines solchen Threads bei Bedarf erstellt und gestartet. Wurde die Stoppgeste nicht erkannt, aber ein Thread zur Beendigung der Steuerung ist noch aktiv, so wird dieser in Zeile 8 in seiner Ausführung gestoppt.

A.2. Klassendiagramm des Socketservers

An dieser Stelle soll ein ausführlicheres Klassendiagramm des Socketservers gegeben werden. Es umfasst auch die Realisierung der Schnittstelle *IDevicePlugin*, die in dieser Arbeit zur Erkennung und Verarbeitung der Ganzkörpergesten eingesetzt wird.

Abbildung A.1.: Klassendiagramm des Socketservers

Literaturverzeichnis

Abs-tech.com. (2011). *CyberGlove II.* Zugriff am 01.06.2012 auf http://www.abs-tech.com/produto.php?id_produto=563

Asai, K., Osawa, N., Sugimoto, Y. Y. & Tanaka, Y. (2002). Viewpoint motion control by body position in immersive projection display. In *Proceedings of the 2002 acm symposium on applied computing* (S. 1074–1079). New York, NY, USA: ACM.

Boytscheff, C. (2006). *Immersive Virtual Reality an der HTWG Konstanz.* Zugriff am 11.05.2012 auf http://www.medien.ag.fh-konstanz.de/VR_Kulturzentrum.pdf

Bricken, M. & Byrne, C. M. (1992). *Summer Students in Virtual Reality: A Pilot Study on Educational Applications of Virtual Reality Technology.* Zugriff am 31.05.2012 auf http://www.hitl.washington.edu/publications/r-92-1

Bryson, S. (1996, Mai). Virtual reality in scientific visualization. *Commun. ACM*, *39* (5), 62–71.

Bues, M., Gleue, T. & Blach, R. (2008). Lightning. Dataflow in motion. In *Software engineering and architectures for realtime interactive systems (SEARIS), proceedings of the IEEE Virtual Reality 2008 workshop* (S. 7-11). Aachen: Shaker.

Bues, M., Gleue, T., Haas, A. & Sulzmann, F. (2009, Juni). *LIGHTNING - User's Guide.*

Bues, M., Wenzel, G., Dangelmaier, M. & Blach, R. (2007). VRfx – A User Friendly Tool for the Creation of Photorealistic Virtual Environments. In C. Stephanidis (Hrsg.), *Universal access in human-computer interaction. ambient interaction* (Bd. 4555, S. 624-632). Springer Berlin / Heidelberg.

Burdea, G. C. & Coiffet, P. (2003). *Virtual Reality Technology* (2. Aufl.). New York, NY, USA: John Wiley & Sons, Inc.

Card, S. K., Mackinlay, J. D. & Robertson, G. G. (1990). The design space of input devices. In *Proceedings of the sigchi conference on human factors in computing systems: Empowering people* (S. 117–124). New York, NY, USA: ACM.

Carlson, W. (2003). *A Critical History of Computer Graphics and Animation.* Zugriff am 31.05.2012 auf http://design.osu.edu/carlson/history/lesson17.html

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V. & Hart, J. C. (1992, Juni). The CAVE: audio visual experience automatic virtual environment. *Commun. ACM*, *35* (6), 64–72.

Freitag, L. (2011). *Eine natürliche Interaktionsschnittstelle zur Steuerung von humanoiden fußballspielenden Robotern.*

Gu, Y. (2011). *VR Anwendungsentwicklung für das PentAI am Beispiel einer Anwendung zur Durchführung der Johansson Experimente zur Bewegungswahrnehmung in stereoskopischem 3D.*

Guadagno, R. E., Blascovich, J., Bailenson, J. N. & McCall, C. (2007, Juni). Virtual Humans and Persuasion: The Effects of Agency and Behavioral Realism. *Media Psychology*, *10* (1), 1-22.

Henning, P. A. (2007). *Taschenbuch Multimedia*. Leipzig: Carl Hanser Verlag.

Ifixit.com. (2010). *Microsoft Kinect Teardown.* Zugriff am 01.06.2012 auf `http://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066`

Kinamon, E. & Gold Hadar, D. (2011). *OpenNI User Guide.* Zugriff am 14.05.2012 auf `http://www.openni.org/images/stories/pdf/OpenNI_UserGuide_v4.pdf`

LaBelle, K. (2011). *Kinect Rehabilitation Project.* Zugriff am 31.05.2012 auf `http://netscale.cse.nd.edu/twiki/bin/view/Edu/KinectRehabilitation`

Lan, C., Xu, Q., Li, J. & Zhou, Y. (2007). A virtual space environment simulation system. In *Proceedings of the 2nd international conference on virtual reality* (S. 497–503). Berlin, Heidelberg: Springer-Verlag.

Microsoft. (2011). *MicArrayEchoCancellation Walkthrough: C++.* Zugriff am 02.05.2012 auf `http://download.microsoft.com/download/F/9/9/F99791F2-D5BE-478A-B77A-830AD14950C3/MicArrayEchoCancellation_Walkthrough.pdf`

Mine, M. R. (1995). *Virtual Environment Interaction Techniques* (Bericht). Chapel Hill, NC, USA.

Paes, T. (2002). *Navigations- und Interaktionstechniken*. Unveröffentlichte Diplomarbeit.

Pletcher, T., Bier, K. & Lubitz, D. von. (2000). An Immersive Virtual Reality Platform for Medical Education: Introduction to the Medical Readiness Trainer. *Hawaii International Conference on System Sciences*, *5*, 5025.

Polhemus.com. *PATRIOT™*. Zugriff am 01.06.2012 auf `http://www.polhemus.com/?page=Motion_Patriot`

PrimeSense. (2010). *The PrimeSensor™ Reference Design 1.08.* Zugriff am 02.05.2012 auf http://primesense.360.co.il/files/FMF_2.PDF

PrimeSense. (2011). *PrimeSense™ NITE Controls User Guide.* Zugriff am 14.05.2012 auf http://www.openni.org/downloads/nite-bin-linux-x64-v1.5.2.21.tar.bz2

Psotka, J. (1995, November). Immersive training systems: Virtual reality and education and training. *Instructional Science*, *23* (5), 405-431.

Rogers, Y., Sharp, H. & Preece, J. (2007). *Interaction Design: Beyond Human-Computer Interaction* (2. Aufl.). Chichester: Wiley.

Schade, C. (2008). *3Dconnexion SpaceNavigator im Kurztest.* Zugriff am 01.06.2012 auf http://www.hardware-aktuell.com/artikel/63/3dconnexion_spacenavigator_im_kurztest

Stork, A. (2001). *Effiziente 3D-Interaktions- und Visualisierungstechniken für benutzerzentrierte Modellierungssysteme*. Unveröffentlichte Dissertation, TU Darmstadt.

Taxén, G. & Naeve, A. (2001). *CyberMath: A Shared Virtual Environment for Mathematics Exploration* (Bericht).

Viager, M. (2011). *Analysis of Kinect for mobile robots.* Zugriff am 01.05.2012 auf http://www.scribd.com/doc/56470872/Analysis-of-Kinect-for-Mobile-Robots-unofficial-Kinect-data-sheet-on-page-27

Winn, W. (1993, August). *A Conceptual Basis for Educational Applications of Virtual Reality* (Report No. TR-93-9). Seattle, WA: Human Interface Technologies Laboratory.

Youngblut, C. (1998). *Educational Uses of Virtual Reality Technology*. Alexandria, VA.: Institute for Defense Analyses.