Gesten- und Sprachsteuerung für einen mobilen Roboter mittels Kinect for Windows unter Java

Studienarbeit

für die Prüfung zum

Bachelor of Science

der Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Simon Ebner, Volker Werling

Januar 2013

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	5837963 7012192, TAI10B2
Gutachter	Prof. Hans-Jörg Haubner

Erklärung

Wir erklären hiermit ehrenwörtlich:

1. dass wir unsere Studienarbeit mit dem Thema *Gesten- und Sprachsteuerung fuer einen mobilen Roboter mittels Kinect for Windows unter Java* ohne fremde Hilfe angefertigt haben;
2. dass wir die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet haben;
3. dass wir unsere Studienarbeit bei keiner anderen Prüfung vorgelegt haben;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Wir sind uns bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Karlsruhe, Januar 2013

Simon Ebner, Volker Werling

Zusammenfassung

Verwendet man heutzutage einen Computer oder eine Spielekonsole, so benutzt man in den meisten Fällen noch immer einen Controller, Maus und Tastatur, oder irgendein anderes Eingabegerät. Doch immer häufiger werden moderne Formen integriert, wie die Bewegungsdetektion verwendet. Dabei wird mittels Videokamera der Benutzer erkannt und dieser ist in der Lage, mit vermindertem Einsatz von herkömmlichen Eingabegeräten oder ganz ohne diesen eine Anwendung, oder gar Spiele zu steuern. Kinect for Windows ist ein populärer Vertreter dieses modern anmaßendnen, aber in Wahrheit doch recht alten Verfahrens der Eingabesteuerung.

In den folgenden Seiten wird erläutert, wie mit einer Kinect for Windows eine Steuerung für einen mobiler Roboter entworfen werden, und aussehen kann. Dabei werden Techniken und Modelle für die Gestenerkennung, ebenso Sprachbefehle und die Umsetzung einer solchen Anwendung erläutert.

Die Autoren Ebner und Werling haben diese Arbeit in Zusammenarbeit verfasst, wobei die Kapitel jeweils von einem der Autoren geschrieben wurden. Auf Herrn Ebner entfallen Kapitel 1, 3, 5, 6 und 7 und auf Herrn Werling die Kapitel 2, 4, 8, 9 und 10.

Inhaltsverzeichnis

1 Einleitung

Gestik im Sinne von kommunikativen Bewegungen stellt einen wesentlichen Teil der nonverbalen Kommunikation dar. Die Definition nach Kurtenbach und Hulteen [KH90] besagt:

> A gesture is a motion of the body that contains information. Waving goodbye is a gesture.Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key on a keyboard is neither observed nor significant. All that matters is which key was pressed.

Als solches bildet eine Geste abstrakte Strukturen und bildhafte Vorstellungen unmittelbar ab. Sie ist somit eine Körperbewegung, die Informationen enthält. Informationen, die herkömmliche Eingabegeräte, darunter Tastatur und Maus, nicht in dieser Form wiedergeben können. Ein Umstand, der sich besonders im Bereich der Mensch-Computer-Interaktion auswirkt.
Das Ziel der Mensch-Computer-Interaktion ist es, die Kommunikation intuitiv und unmittelbar zu gestalten. Genau an dieser Stelle setzen Gesten an, in dem sie eine bessere Schnittstelle zwischen Mensch und Maschine darstellen.
Die Spannweite von Realisierungen von Mensch-Maschine-Schnittstelle ist groß. Frühe Ansätze zeigen bereits das Potential, Gesten zur Steuerung und Nutzung von diversen Geräten zu verwenden. Das *Theremin* - ein 1919 erfundenes elektronisches Musikinstrument, das mit Händen berührungslos durch Beeinflussung eines elektromagnetischen Feldes gesteuert wird und dadurch Töne erzeugt - ist eine der ersten technischen Lösungen für eine Gestensteuerung. Eine weitere From, der *Datenhandschuh*, 1977 von Electronic

Visualization Laboratory Labs entwickelt [1], erregte großes Interesse und wurde unter anderem als *Power Glove* von Mattel vertrieben. Datenhandschuhe werden bis heute in den verschiedensten Bereichen eingesetzt.
Eine weitere Variante der Gestenerkennung ist die Bewegungsdetektion. Dabei wird über optische Sensoren der Körper oder einzelne Körperteile des Nutzers erkannt und somit die Steuerung durch Gesten ermöglicht. Eine frühe Lösung ist der sogenannte *Videoplace* [2], entwickelt von Myron Krueger Mitte der 70'er Jahre [3]. Ausgelegt als ein *Labor für künstliche Realität* war eine Person, durch ihn umgebende Projektoren und Videokameras, in der Lage seine Umgebung zu beeinflussen. Moderne Lösungen haben das Konzept der Detektion durch Kameratechnik weiterentwickelt und einer weiten Öffentlichkeit zugänglich gemacht. Darunter auch das *PLAYSTATIONEye* [4]. Der populärste Vertreter ist *Kinect for XBOX* mit über 18 Millionen verkauften Exemplaren [5].
Aufgrund des geringen Einkaufspreises, von derzeit etwa 195 €[6], wird die Kinect in zahlreichen Anwendungen verwendet. Ein aktuelles Beispiel, ist ein Projekt von Disney Research, „Playing Catch and Juggling with a Humanoid Robot" von Kober et al. [KM12], wobei ein Roboter mittels Kinect auf den Wurf eines Balles von einer Person reagiert, diesen versucht zu fangen und zurückwirft.
Aufgrund der Verfügbarkeit der Kinect und deren hoher Verbreitungsgrad wird im folgenden mit jener Kinect gearbeitet.
In dieser Arbeit wird mit Hilfe der Kinect eine Gesten- und Spracherkennung entwickelt um eine Steuerung für einen mobilen Roboter zu implementieren. Aufgrund des Umfangs dieser Studienarbeit wird die Erarbeitung dieser Anwendung in zwei Teile aufgeteilt.
Der erste Teil der Arbeit beschäftigt sich vorrangig mit der Realisierung der Gesten- und Spracherkennung. Die eigentliche Steuerung für einen mobilen Roboter wird vorerst

[1] Sturman, D.J., Zeltzer, D. (January 1994). „A survey of glove-based input". IEEE Computer Graphics and Applications 14 (1): 30–39
[2] Beschreibung der Einrichtung „VIDEOPLACE". medienkunstnetz.de. Abgerufen Dezember 22, 2012
[3] Myron Krueger. Artificial Reality 2, Addison-Wesley Professional, 1991. ISBN 0-201-52260-8
[4] „PLAYSTATIONEye Brings Next-Generation Communication to PLAYSTATION3". us.playstation.com. Sony Computer Entertainment America. Abgerufen Dezember 21, 2012
[5] Takahashi, Dean (Januar 9, 2012). „Xbox 360 surpasses 66M sold and Kinect passes 18M units". venturebeat. Abgerufen Dezember 20, 2012
[6] Amazon.de - Microsoft Kinect Sensor for Windows. Amazon.de. Abgerufen Januar 14, 2013

durch eine Schnittstelle und einer Testumgebung ersetzt. Der Fokus liegt hierbei in erster Linie auf der verwendeten Technik, in diesem Fall der Kinect zur Bewegungsdetektion, den verwendeten Gesten- und Sprachbefehlen, sowie der Umsetzung der Software und den darin verwendeten Technologien. Der zweite Teil wird an diese Arbeit anknüpfen und im weiteren die Umsetzung der Steuerung für und den Einsatz eines mobilen Roboters beschreiben.
Die folgende Ausarbeitung spiegelt die Umsetzung des ersten Teils dieser Arbeit wieder.

2 Aufgabenstellung

2.1 Problemstellung und Ziel der Arbeit

Das Ziel des ersten Teils dieser Arbeit ist die Entwicklung einer Schnittstelle zwischen Mensch und Roboter. Über dieser Schnittstelle soll es dem Benutzer möglich sein, mittels Sprache und Gesten einen mobilen Roboter fernzusteuern. Die Entwicklung der Ansteuerung des mobilen Roboters wird im zweiten Teil der Arbeit erläutert.
Zur Erfassung von Sprache und Bewegung des Benutzers soll im Rahmen dieser Arbeit die *Kinect* von Microsoft und das zugehörige *SDK* [1] eingesetzt werden. Die Entwicklung dieser Anwendung soll auf Basis der Programmiersprache Java erfolgen. Hierzu ist es erforderlich eine Schnittstelle zum Kinect zu verwenden, da hier native Entwicklung nur mit C++, C# oder Visual Basic möglich ist. Daher muss an dieser Stelle ein Framework eingesetzt werden, das den Zugriff auf das mittels Java ermöglicht. In diesem Rahmen soll das Framework OpenNI und etwaige Alternativen evaluiert werden.

2.2 Geplantes Vorgehen

Es müssen die Grundlagen zur Entwicklung der zuvor beschriebenen Anwendung geschaffen werden. Hierzu ist es zunächst erforderlich die notwendigen theoretischen Grundlagen

[1] Microsoft Corporation (2012), *Kinect for Windows* microsoft.com, Abgerufen Dezember 28, 2012

zur Sprach- und Gestensteuerung, sowie deren Erkennung und Verarbeitung zu erarbeiten. Ebenso müssen zur Entwicklung mit dem Kinect in Kombination mit der Programmiersprache Java möglichen Optionen ermittelt und evaluiert werden. Auf Basis der hierdurch gewonnenen Erkenntnisse soll im ersten Schritt eine Anwendung entwickelt werden, die es ermöglicht durch Sprache und Gesten definierte Aktionen zu erkennen. Diese Aktionen werden die Grundlage der Ansteuerung des mobilen Roboters darstellen.
Nach Möglichkeit sollen darüber hinaus weitere Anwendungsszenarien zur Steuerung durch die Kinect erdacht und evaluiert werden.

2.3 Ausblick

Im zweiten Teil dieser Arbeit soll die Schnittstelle von der bisher entwickelten Sprach- und Gestenerkennung zu einem bisher nicht näher spezifizierten mobilen Roboter geschaffen werden. Hierzu wird es notwendig die entsprechenden Grundlagen zu erwerben und die Anwendung um die Implementierung etwaiger Roboter-Aktionen zu erweitern. Darüber hinaus soll das Repertoire an zu erkennenden Aktionen um im Zusammenhang einer Roboter-Steuerung sinnvollen Kommandos erweitert werden.

3 Stand der Technik

3.1 Kinect

Kinect ist eine Gerät zur Detektion von Bewegungen. Entwickelt wurde es von PrimeSense als Hardware zur Steuerung der Videokonsole XBOX 360 von Microsoft Corp. Nach der Ankündigung im März 2010 [1] war die Erweiterung mit dem Erscheinungsdatum 10. November 2010 in der Ausführung *Kinect for XBOX 360* in Europa erhältlich [2]. Nach einem sehr erfolgreichem Verkaufsstart [3] und anhaltender Nachfrage[4]. kündigte Microsoft am 9. Januar 2012 ein weiteres Modell der Kinect, die sogenannte *Kinect for Windows* für den 1. Februar 2012 an [5].
Darüber hinaus veröffentlichte Microsoft bereits am 16. Juni 2011 eine erste Version ihrer *Kinect for Windows SDK*[6]. Mit diesem Software Development Kit ist es möglich auf einer Windows 7 Plattform Anwendungen zu entwickeln, die eine Kinect als Eingabegerät verwenden.
Mit der Verfügbarkeit einer Kinect-Variante, die für den Einsatz am PC ausgelegt ist und der frei zugänglichen einer breiten Öffentlichkeit als Forschungsgegenstand zugänglich,

[1] Pressemitteilung, der Veröffentlichung. Siehe Link. Microsoft.com. Abgerufen Dezember 20, 2012
[2] Erscheinungsdatum der *Kinect for XBOX 360*. Siehe Link. BBC UK. Abgerufen Dezember 20, 2012
[3] „Am schnellsten verkauftes Perepheriegerät für Spiele“. Guinnessworldrecords.com. Abgerufen Dezemeber 22, 2012
[4] Takahashi, Dean (Januar 9, 2012). „Xbox 360 surpasses 66M sold and Kinect passes 18M units“. venturebeat. Abgerufen Dezember 20, 2012
[5] „Ankündigung der *Kinect for Windows*“. blogs.msdn.com. Abgerufen Dezember 22, 2012
[6] „Microsoft Releases Kinect for Windows SDK Beta for Academics and Enthusiasts“. Microsoft.com. Abgerufen Dezember 21, 2012

was auch der ausschlaggebende Punkt für den Einsatz der Kinect in dieser Studienarbeit ist.

3.1.1 Hardware

Die beiden Kinect-Modelle unterscheiden sich in einigen Details[7]. Der bedeutendste Faktor ist das sogenannte *Near Mode* Feature:

> Near Mode enables the depth sensor to see objects as close as 40 centimeters and also communicates more information about depth values outside the range than was previously available. There is also improved synchronization between color and depth, mapping depth to color, and a full frame API. [7]

Neben dem aktualisierten Tiefensensor unterscheiden sich die beiden Varianten auch in einer höheren Auflösung der RGB-Kamera, die für eine Gestenerkennung relevant sein kann. Aus diesem Grund wird für diese Arbeit die *Kinect for Windows* genutzt.

Technische Daten

Eine Kinect entält innerhalb des Gehäuses, folgende Sensoren [8]:

- Eine RGB-Kamera mit einer Auflösung von 1280x960. Dies ermöglicht eine Farbbilderfassung
- Ein Infrarot (IR) Emitter und ein IR Tiefensensor. Der Emitter emittiert infrarote Lichtstrahlen und der Tiefensensor erfasst die an den Sensor reflektierten Strahlen. Die reflektierten Strahlen werden in Tiefeninformation umgewandelt, in dem der

[7] „Informationsseite über Unterschiede der Kinect Versionen“. Microsoft.com. Abgerufen Dezember 22, 2012

[8] „Kinect for Windows Sensor Components and Specifications“. msdn.microsoft.com. Abgerufen Dezember 21, 2012

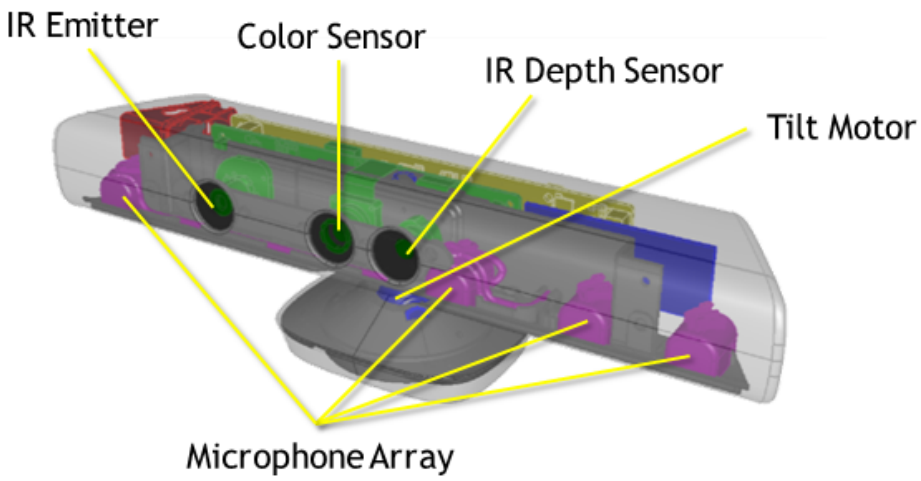


Abbildung 3.1: Schematische Ansicht der Sensoren einer *Kinect for Windows* (Quelle: Microsoft Corp.[8])

Abstand zwischen Objekt und Sensor bestimmt wird. Dies ermöglicht die Erfassung von Tiefenbildern

- Ein Mikrofonarray, das vier Mikrofone zur Soundaufnahme enthält. Die Anzahl der Mikrofone ermöglicht nicht nur die Aufzeichnung von Audiodaten, sondern auch die Lokalisierung der Soundquelle und die Richtung des Audiosignals
- Ein 3-Achsen-Beschleunigungssensor, konfiguriert für einen Bereich der zweifachen Erdbeschleunigung, um die gegenwärtige Ausrichtung der Kinect zu bestimmen
- Ein Kippmotor, zur automatisierten Justierung der Sensoren

Weitere Details der technischen Spezifikation einer *Kinect for Windows* sind in Tabelle 3.1 aufgelistet:

Tabelle 3.1: Kinect for Windows – technische Spezifikation [8]

Kinect	**Spezifikation**
Blickwinkel	43° vertikales, 57° horizontales Blickfeld
Vertikaler Neigebereich	±27°
Bildwiederholrate (Farb- und Tiefensignal)	30 Bilder pro Sekunde (FPS)
Audioformat	16-kHz, 24-bit mono pulse code modulation (PCM)
Audioeingang	Ein Vier-Mikrofonarray mit 24-Bit Analog-Digital-Wandler (ADC)
Datensignal–Tiefensensor	640x480 16-bit, 30 Bilder pro Sekunde
Datensignal–RGB-Kamera	1280x960 16-Bit, 12 Bilder pro Sekunde 640x480 16-Bit, 30 Bilder pro Sekunde
Tiefensensorreichweite	0,4 – 4 m
Skelett Tracking System	Erkennung von bis zu sechs Benutzern, zwei davon trackbar/verfolgbar Verfolgung von 20 Gelenken pro aktivem Nutzer

[8] „Kinect for Windows Sensor Components and Specifications“. msdn.microsoft.com. Abgerufen Dezember 21, 2012

3.1.2 Software

Durch die Veröffentlichung eines Software Development Kits ist es möglich, die Kinect in eigene Programme einzubinden und neue Anwendungsfälle zu bearbeiten. Dazu hat Microsoft ebenfalls ein Handbuch veröffentlicht [Cor12], das Informationen und Ratschläge zum Entwurf von Anwendungen liefert.

Kinect for Windows SDK

Das *Kinect for Windows SDK* steht aktuell in der Version 1.6 [9] bereit. Dabei kann direkt in den Programmiersprachen C++, C# , und Visual Basic auf einer Windows 7 Plattform entwickelt werden.
Das SDK umfasst dabei folgende Funktionen [10]:

- Treiber und technische Dokumentation zur Implementierung von Anwendungen, die Kinect for Windows Sensoren nutzen
- Referenz APIs und Dokumentation zur Programmierung von *managed* und *unmanaged* Code. Die APIs stellen mehrere Mediensignale bereit
- Beispiele und Best-Practice-Lösungen

Weitere Frameworks

Da Microsoft die Nutzungsmöglichkeiten seines SDK hinsichtlich verwendeter Programmiersprache und Plattform einschränkt, begannen Forscher eigene Frameworks und Treiber zur Nutzung der Kinect zu entwickeln[11]. PrimeSense selbst veröffentlichte Treiber

[9] Downloadseite des SDK. Microsoft.com. Abgerufen Dezember 21, 2012
[10] „Kinect for Windows Programming Guide“. msdn.microsoft.com. Abgerufen Dezember 23, 2012
[11] „Open Source Kinect contest has been won“. hackaday.com. November 11, 2010. Abgerufen Dezember 21, 2012

und Middleware für die Kinect [12], als das offizielle SDK noch nicht zur Verfügung stand. Das Ziel dieser Frameworks war es zu meist, Kinect-Steuerung unter Unix-Plattformen und Programmiersprachen, wie Java, nutzbar zu machen. Ein Teil dieser Ausarbeitung ist es, ein für die Aufgabenstellung und Zielsetzung der Studienarbeit passendes Framework zu bestimmen.
Verbreitete Lösungen, die im Kapitel 4 näher betrachtet werden, sind:

- OpenNI
- OpenKinect
- jnect

3.2 Java – Eclipse

Die Anwendung, die im Rahmen dieser Arbeit entsteht, wird auf Basis der Programmiersprache Java und der Entwicklungsumgebung Eclipse erstellt. Die Nutzung der Kinect und des Kinect SDK unterstützt nativ nicht die Entwicklung auf Basis von Java, wie in Abschnitt 3.1.2 festgestellt wurde. Da jedoch die Expertise der Autoren dieser Arbeit im Java-Umfeld liegt, wird die Anwendung dennoch in der Programmiersprache Java umgesetzt.
Als Entwicklungsumgebung kommt die Eclipse IDE zum Einsatz. Diese bietet alle Voraussetzungen zur Entwicklung von flexiblen Java-Anwendungen und bieten eine ausführliche Support-Plattform für Entwickler. Zusätzlich setzt es auf eine dynamische Supportplattform zur Modularisierung von Anwendungen[13].

[12] Mitchell, Richard (Dezember 10, 2010). „PrimeSense releases open source drivers, middleware for Kinect“. Joystiq. Abgerufen Dezember 22, 2012
[13] „OSGi“. eclipse.org. Abgerufen Januar 1, 2013

3.2.1 OSGi

Zur Entwicklung einer komplexen Anwendung mit vielen Abhängigkeiten und Schnittstellen, wie die hier zu erarbeitende, braucht es eine entsprechend *mächtige* Software-Architektur. Dabei wird hier auf OSGi-Plattform gesetzt. OSGi[14] ist ein Framework, dass Spezifikationen zur Definition eines dynamischen *Komponentenmodells* unter Java. Diese Komponenten werden *Bundles* genannt und stehen anderen Komponenten als sogenannte *Services* bereit. Die OSGi Alliance[1] selbst spezifiziert lediglich die Programmierschnittstelle. Die Umsetzung für die Eclipse IDE lautet Equinox[15].

3.2.2 Grafische Oberfläche

Das Graphical User Interface (GUI) ist ein wesentlicher Bestandteil der Anwendung, da hierüber der gesamte Informationsaustausch mit dem Nutzer der Kinect stattfindet. Dabei wird auf verschiedene Techniken gesetzt, die im folgenden kurz beschrieben werden. Bei der grafischen Darstellung muss man zwischen Informationen der Anwendung und Interaktion des Nutzers mit der Software unterscheiden. Dazu werden nämlich unterschiedliche Techniken verwendet.

Rich client platform

Die Rich client platform (RCP) ist ein Werkzeug zur Entwicklung von unabhängigen Software-Komponenten, die nicht nur auf GUI-Komponenten beschränkt ist [16]. Dabei liegt der Fokus auf Client-Applikation, worunter auch die Anwendung dieser Arbeit fällt. Daher wird RCP im Rahmen dieser Arbeit zur Erstellung der Oberfäche der Anwendung verwendet.

[14] „OSGi Alliance“. osgi.org. Abgerufen Januar 1, 2013
[15] „equinox OSGi“. eclipse.org. Abgerufen Januar 1, 2013
[16] „Rich Client Platform“. wiki.eclipse.org. Abgerufen Januar 1, 2013

Die Lightweight Java Game Library

Die grafische Darstellung der Interaktion zwischen Akteur und Kinect ist ein wichtiger Bestandteil dieser Anwendung. Zur Anzeige aufwändiger grafischer Objekte wird auf die Lightweight Java Game Library (LWJGL)[17] gesetzt. Damit können einfach zweidimensionale Grafiken, zum Beispiel Kreise, oder komplexe dreidimensionale Ansichten in einer Java-Anwendung angezeigt werden. Diese Java-Bibliothek stellt eine API zum Zugriff und zur Verwendung der bekannten Open Graphics Library[18] bereit.

Eclipse Modeling Framework

Das Eclipse Modeling Framework (EMF)[19] ist ein Framework zur Modellierung von Java-Anwendungen. Es ermöglicht aus strukturierten Datenmodellen, Java-Code zu generieren. Für die Kinect-Anwendung sind lediglich die Bereiche des Frameworks relevant, die diese *Modelle* beschreiben.

Graphical Editing Framework

Zur grafischen Darstellung und grafischen Manipulation der zuvor vorgestellten EMF-Modelle existiert das sogenannte Graphical Editing Framework (GEF)[20]. Es stellt Methoden zur Verfügung, um Grafikeditoren und weitere Ansichten fr Eclipse Anwendungen zu erstellen. Für die Studienarbeit relevante Bereiche, sind die Funktionen, die eine vereinfachte Darstellung der eben genannten EMF-Modelle ermöglichen.

[17] „LWJGL Lightweight Java Game Library“. lwjgl.org. Abgerufen Januar 1, 2013
[18] „The Industry's Foundation for High Performance Graphics“. opengl.org. Abgerufen Januar 1, 2013
[19] „Eclipse Modeling Framework Project (EMF)“. eclipse.org. Abgerufen Januar 1, 2013
[20] „GEF (Graphical Editing Framework)“. eclipse.org. Abgerufen Januar 1, 2013

3.3 Roboter – Ausblick

Ziel und Aufgabenstellung dieser Arbeit ist, wie bereits in Kapitel 2 erläutert, der Entwurf einer Steuerung für einen mobilen Roboter. Der erste Teil, der zweigeteilten Studienarbeit befasst sich dabei nicht direkt mit der eigentlichen Robotersteuerung oder der Beschreibung und Auswahl eines entsprechenden Modells. Dies ist Aufgabe und Fokus des zweiten Teils.
Für die Zwecke der Arbeiten an diesem Teil und der dabei zu erstellenden Anwendung, wird lediglich ein Grundgerürst und eine Schnittstelle erarbeitet, um auf die Kinect-Steuerung zuzugreifen. Siehe hierzu Abschnitt 9.2.4.

3.4 Mustererkennung

Die *Kinect for Windows* und die dazu gehöhrende *Kinect for Windows SDK* besitzen bereits Algorithmen zur Detektion von Bewegungen einzelner Körperteile und der Erkennung von Sprache. Dabei muss festgehalten werden, dass zwischen der Detektion der Bewegung eines Gelenkes und der Erkennung einer Geste eine große Diskrepanz besteht. Die Bewegung an sich macht noch keine Geste aus. Zur *Gestenerkennung* sind Verfahren und Algorithmen notwendig, die über den Funktionsumfang der Kinect hinausgehen.
Hierzu sind Verfahren der Mustererkennung notwendig, die auf mathematischen Modellen beruhen. Dabei haben sich zwei Richtungen herausgebildet. Eine Variante nutzt hierbei sogenannte künstliche neuronale Netze. Beschrieben und umgesetzt wurde dieser Ansatz unter anderem von Chun Zhu and Weihua Sheng [ZS09].
Eine weitere Variante setzt das sogenannte Hidden Markov Model (HMM) ein, das von Leonard E. Baum [BP66] postuliert wurde. Dabei handelt es sich um ein stochastisches Modell, indem ein System durch eine Markov-Kette (Siehe Abschnitt 6.5) modelliert wird. Eine der ersten Arbeiten zum Einsatz in der Gestenerkennung schrieb Lawrence R. Rabiner [Rab89]. Dieser Ansatz findet häufige Verwendung in Bereichen der Spracherkennung, Gestenerkennung und in der Mensch-Maschine-Kommunikation. Aus die-

sem Grund wurde entschieden, ein HMM zu erstellen. Das stochastische Modell dahinter wird in Kapitel 6 und die Umsetzung in der Implementierung in Kapitel 9 beschrieben.

4 Kinect–Framework

4.1 Microsoft Kinect SDK

Das Kinect SDK von Microsoft stellt, wie zuvor bereits erwähnt, die technische Schnittstelle zur Kinect zur Verfügung. Dies sind zum einen die Programmierschnittstellen in den Sprachen C++, C# und VB .net also auch die benötigten Treiber und Runtime Libraries, die die Kommunikation mit dem Gerät ermöglichen. In der nachfolgenden Abbildung ist der Aufbau des SDK abgebildet.

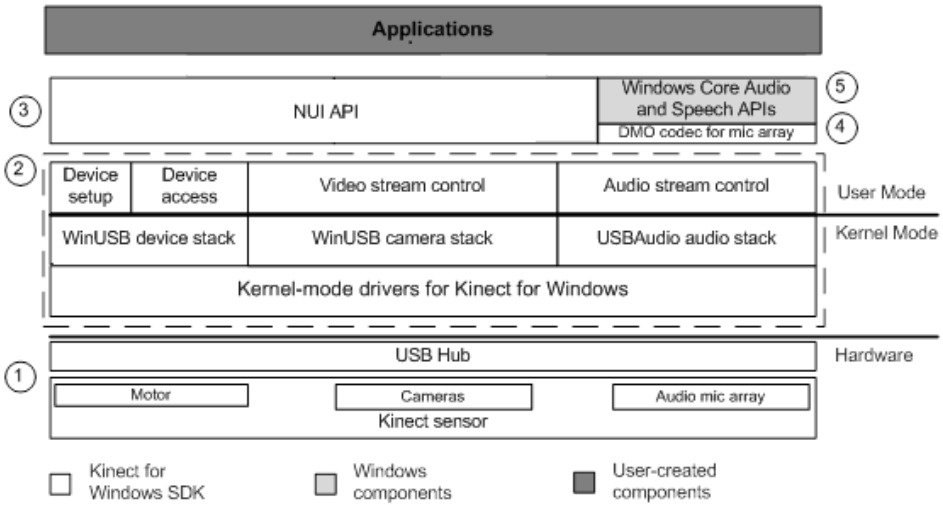


Abbildung 4.1: SDK Architektur (Quelle: Microsoft Corp.[1])

[1]Microsoft Corp. (2012) *SDK Architektur* msdn.microsoft.com, Abgerufen Januar 03, 2013

Die Ebene 3 enthält die Programmierschnittstelle. Mittels der NUI API kann auf die Datenströme, die die Kinect liefert zugegriffen werden. Von der Kinect werden hier vier Datenströme[2] geliefert:

- AudioStrom
- Farbstrom
- Tiefenstrom
- Infrarotstrom

Auf Basis dieser Datensröme kann eine Erkennung von Personen, Gesten und Sprachkommandos realisiert werden. Dies ist jedoch nicht zwingend notwendig, da hier durch die Möglichkeit des Zugriffs auf einen Skelettstrom bereits, ohne diesen Transformationsschritt von Rohdaten zu einem Model, auf die Koordinaten eines Modells des menschlichen Körpers zugegriffen werden kann. Diese Komponente kann bis zu 2 Personen erkennen und die Modelle dieser Personen liefern[3]. Das Modell basiert auf 20 Punkten, die bestimmt Körperteile und -regionen abbilden. Dieses Modell ist in Abbildung 4.2 zu sehen.

[2]Microsoft Corp. (2012) *DataStreams* msdn.microsoft.com, Abgerufen Januar 03, 2013
[3]Microsoft Corp. (2012) *Tracking Users with Kinect Skeletal Tracking* msdn.microsoft.com, Abgerufen Januar 03, 2013

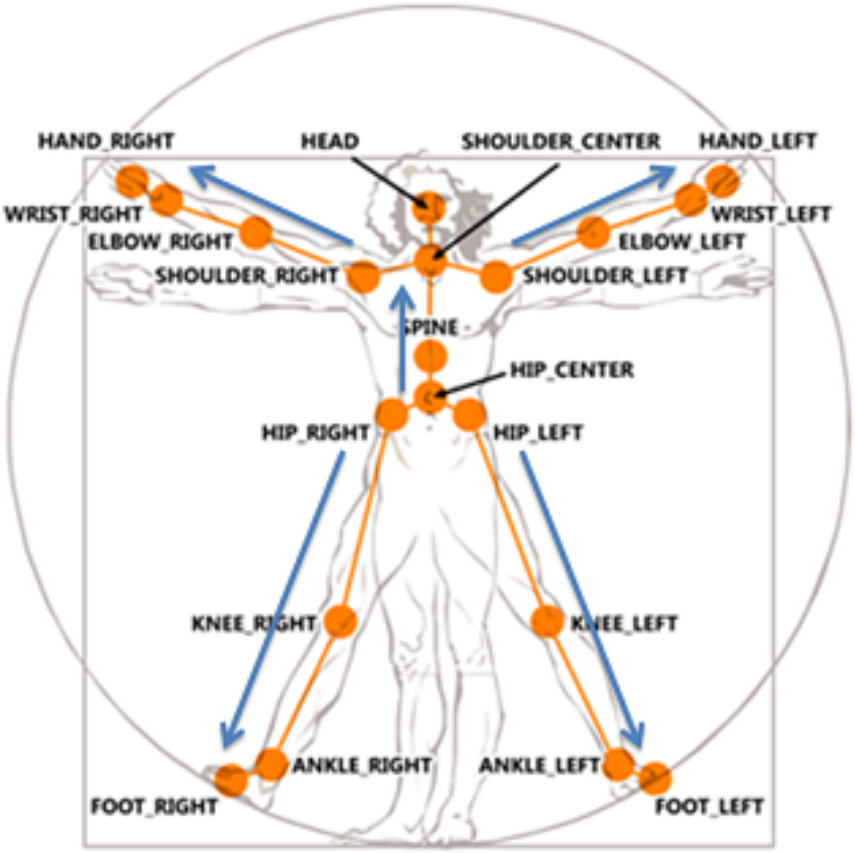


Abbildung 4.2: Skeleton (Quelle: Microsoft Corp.[3])

Auf dieser Basis wird es schon relativ leicht möglich eine Gestensteuerung zu entwickeln. Weitere Beispiele für das Tracking bilden die Ausgabe des Kinect Explorer[4], einem Beispiel welches Teil des SDK ist und einer Visualisierung mittels GEF in der Anwendung die im Rahmen dieser Arbeit entwickelt wird. In Abbildung 4.3 ist die Visualisierung der Punkte im Kinect Explorer zu sehen, in Abbildung 4.4 die Darstellung in mit dem Graphical Editing Framework.

[4]Microsoft Corp. (2012) *Kinect Explorer* msdn.microsoft.com, Abgerufen Januar 03, 2013

Abbildung 4.3: Kinect Explorer[4]

Außerdem ist es auch unter Verwendung des SDK in mindestens Version 1.5 und einer *Kinect for Windows* möglich ein Gesicht und dessen Bewegungen zu erfassen[5]. Diese Funktion wird im ersten Teil dieser Arbeit jedoch noch keine Anwendung finden.

Zur Spracherkennung muss ebenfalls nicht notwendigerweiße direkt der Audiostrom der Kinect analysiert werden. Mittels einer Speechengine kann auf Basis einer durch den Entwickler definierten Grammatik Sprache erkannt werden. Der Entwickler wird dann über erkannte Kommandos mittels Events benachrichtigt.

[5]Microsoft Corp. (2012) *Face Tracking* msdn.microsoft.com, Abgerufen Januar 03, 2013

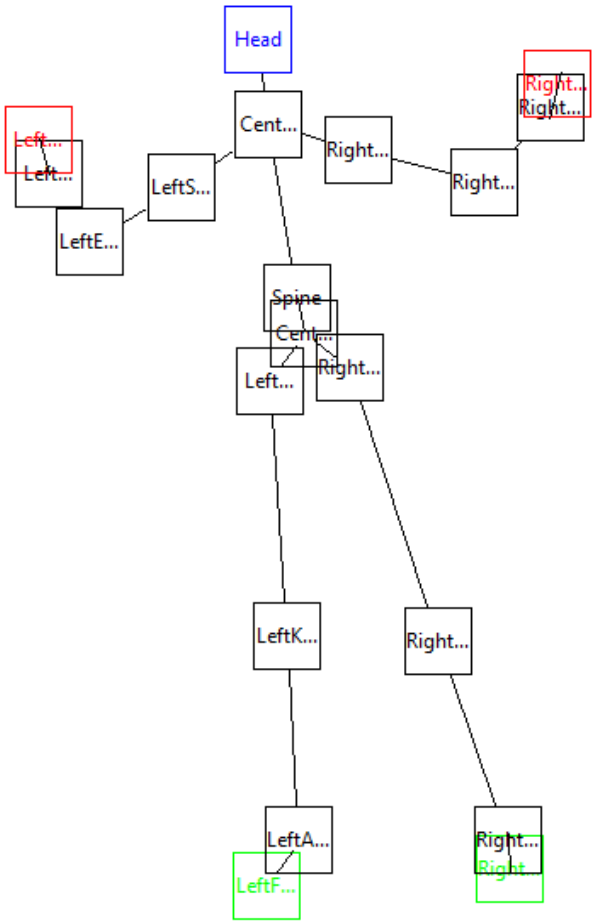


Abbildung 4.4: GEF Editor

4.2 Java Frameworks

Da die Entwicklung mit Java durchgeführt werden soll ist es notwendig einen Wrapper für das Kinect SDK zu verwenden. Hierzu existieren bereits diverse Frameworks die genau das ermöglichen. Nachfolgend werden drei dieser Frameworks betrachtet und deren Nutzbarkeit im Rahmen dieser Arbeit evaluiert. Abschließend wird ein Framework ausgewählt auf dessen Basis die Umsetzung dieser Arbeit erfolgen soll.

Ziel dieser Arbeit ist die Umsetzung einer Gesten- und Sprachsteuerung. Daher sollte ein möglichst einfacher Zugang zum Skelettstrom und der Spracherkennungskomponennte des Kinect SDK vorhanden sein. Eine Eigenentwicklung dieser Funktionen würde den Rahmen dieser Arbeit bei weitem überschreiten.

4.2.1 OpenNI

Beschreibung

OpenNI[7] ist der Name eines Industriekonsortiums dessen Ziel es ist, die Entwicklung und Standardisierung von Software und Hardware im Bereich der Natural User Interfaces. Es wurde im November 2010 ins Leben gerufen und setzt sich aus verschiedenen großen Unternehmen zusammen. Dazu gehört beispielsweise Primesense, welches selbst solche Geräte entwickelt und vertreibt oder auch ASUS, welche in Kooperation mit Primesense die Xtion Pro[9] entwickelt haben und vertreiben. Zusammen formen diese unter dem Namen OpenNI eine Non-Profit-Organisation.

[6]OpenNI *Home* openni.org, Abgerufen Januar 04, 2013
[7]OpenNI *Organization* openni.org, Abgerufen Januar 04, 2013
[8]OpenNI *About* openni.org, Abgerufen Januar 04, 2013
[9]Asus (Mai 2011) *Xtion Pro* aus.de, Abgerufen Januar 04, 2013

Technische Daten

Durch OpenNI wird ein SDK und eine Plattform zur Verfügung gestellt, mit der eine Verbindung mit verschiedenen Geräten möglich ist. Zu sehen ist der Aufbau in Abbildung 4.5. Das SDK stellt eine Schnittstelle zu den grundsätzlichen Sensoren der Hardware und den davon generierten Datenströmen dar. Einzig der Audiostrom wird zum jetzigen Zeitpunkt noch nicht nutzbar. Durch den Einsatz von *Middleware Libraries* können abstraktere Funktionen, wie das Tracking eines Skelettes abgebildet werden. Ein Beispiel hierfür ist die Middleware Nite 2[10] von Primesense. Das SDK eignet sich grundsätzlich zur Nutzung verschiedener Geräte, sofern Treiber verfügbar sind, ist also nicht auf die Kinect festgelegt. Eigene Anwendungen lassen sich mittels der API in C++ entwickeln. Außerdem werden Wrapper Java und C# bereitgestellt. Zugriff auf das Mikrofonarray ist jedoch nicht gegeben und muss vom Benutzer selbst implementiert werden. Somit steht hier auch keine nutzbare Engine zur Spracherkennung zur Verfügung. Auch durch Middleware wird hier keine Lösung geboten.

Analyse der Anwendbarkeit

Eine Java-Schnittstelle ist vorhanden. Das Skeleton-Tracking wird durch das SDK direkt nicht geboten, wird jedoch von einer Middleware zur Verfügung gestellt, welche ebenfalls eine Java-Schnittstelle besitzt. Jedoch verfügt dieses SDK nicht über die Möglichkeit die Spracheingabe auf Muster zu untersuchen. Eine Spracherkennung ist also nicht geboten.

[10] OpenNI *Nite 2* openni.org, Abgerufen Januar 04, 2013

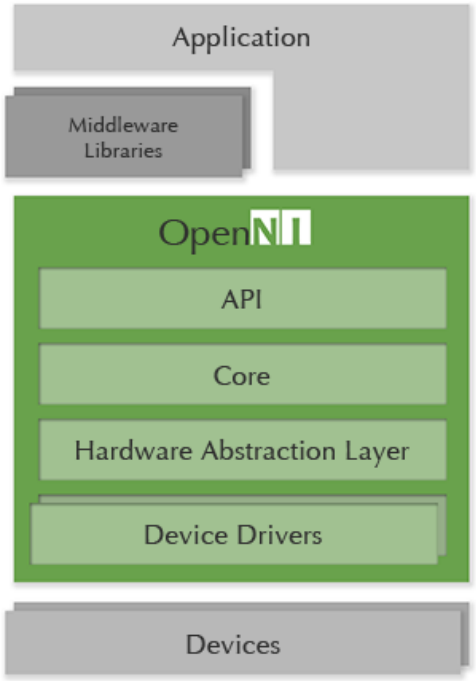


Abbildung 4.5: Architektur des OpenNI SDK[8]

4.2.2 OpenKinect

Beschreibung

OpenKinect[11] ist eine Open Source Community, die sich aus Personen zusammensetzt deren Interesse in der Entwicklung rund um die Kinect und ähnlichen Geräten liegt. Diese Community entwickelt derzeitig die Software libfreenect. Diese ist die Schnittstelle zur Kinect. Das Projekt ist allein durch seine Community getragen.

Technische Daten

OpenKinect ist in C geschrieben, stellt jedoch Wrapper für viele weitere Sprachen bereit[12], darunter auch Java. Außerdem sind Treiber für die Kinect enthalten. Hiermit besteht Zugriff auf die, durch die Kinect gelieferten, Rohdaten.

Analyse der Anwendbarkeit

Dieses Projekt bietet ebenfalls die Möglichkeit mittels Java auf die Kinect zuzugreifen. Jedoch sind auch hier bislang keine Erweiterungen verfügbar. Hier stellt sich ebenfalls das Problem der nicht vorhanden Spracherkennung.

4.2.3 jnect

Beschreibung

jnect[13] ist ein Plugin für die Eclipse-Plattform. Es macht Funktionen der Kinect für Eclipse-basierte Anwendungen und Plugins nutzbar. Im Gegensatz zu OpenNI und

[11] OpenKinect *OpenKinect Mainpage* openkinect.org, Abgerufen Januar 05, 2013
[12] Abschnitt Wrapper *OpenKinect Mainpage* openkinect.org, Abgerufen Januar 05, 2013

OpenKinect stellt jnect keine eigene Schnittstelle zur Kinect zur Verfügung. Es ist lediglich ein Wrapper für die Skeleton-Tracking und Speech Recognition Engine des Kinect for Windows SDK[14].

Technische Daten

Das Projekt besitzt einen Wrapper für die genutzten Funktionen des Kinect SDK. Die durch jnect genutzten Funktionen sind das Skeleton-Tracking und die Sprachengine. Das Skeleton der Kinect wird innerhalb von jnect in einem Modell des EMF gespeichert. Die Software wird in Form von OSGi-Bundles ausgeliefert und ist in erster Linie dazu gedacht in einer Eclipse-basierten Anwendung eingesetzt zu werden.

Analyse der Anwendbarkeit

Auch dieses Projekt ist durch die Implementierung in Java schon einmal grundsätzlich als nutzbar einzustufen. Die Datenströme der Sensoren werden jedoch nicht zur Verfügung gestellt. Das stellt aber kein Problem dar, da für diese Arbeit Gesten und Sprachbefehle erfasst werden sollen, was durch die Verfügbarkeit des Skelett-Modells und den Zugang zur Sprachengine gegeben sind.

4.2.4 Nutzwertanalyse

Zur Auswahl eines geeigneten Frameworks wurde eine, in Abbildung 4.6 zu sehende, Nutzwertanalyse durchgeführt. Hierfür werden die folgenden 5 Kriterien herangezogen:

- **Java Schnittstelle**, beziehungsweise die Verfügbarkeit dieser. Dies ist ein wichtiger Punkt, da das KnowHow der Autoren dieser Arbeit vor allem im Bereich der Java Programmierung liegt.

[13]jnect *jnect Mainpage* code.google.com, Abgerufen Januar 05, 2013
[14]jnect *jnect Wiki Features* openkinect.org, Abgerufen Januar 05, 2013

- **Abdeckung SDK**, die Abdeckung der Sensoren und Funktionen der Kinect. Notwendigerweiße werden hier nur die nachfolgenden 2 Funktionen benötigt, jedoch wäre eine ABdeckung der weiteren Sensoren für zukünftige Erweiterungen wünschenswert.
- **Skeleton Stream**, die Verfügbarkeit eines Skeleton-Modell, wie es auch auch durch das Kinect SDK zur Verfügung gestellt wird. Dies ist wichtig für diese Arbeit, da das Erfassen von Gesten das Thema ist, und das Aufstellen eines eigenen Modells auf Basis der Rohdaten sehr aufwendig wÄre.
- **Speech Recognition**, eine Engine, die die Analyse des Audiostroms vornimmt und daraus gewünschte Sprachbefehle erkennen kann. Ebenfalls absolut wichtig für das Thema dieser Arbeit.
- **Dokumentation**, Dokumentation der API

Entsprechend wurden die Kriterien gewichtet.

		Jnect		OpenNI		OpenKinect	
Kriterien	Gewicht	Wertung	Gewichtete Wertung	Wertung	Gewichtete Wertung	Wertung	Gewichtete Wertung
Java Schnittstelle	20	5	100	5	100	5	100
Abdeckung SDK	15	3	45	4	60	2	30
Skeleton Stream	25	5	125	4	100	0	0
Speech Recognition	25	3	75	0	0	0	0
Dokumentation	15	1	15	2	30	2	30
Score			350		290		160

Abbildung 4.6: Nutzwertanalyse der verwendbaren Frameworks

Da weder bei OpenKinect noch bei OpenNI eine Spracherkennung enthalten ist, diese sehr schlecht abschneiden. Zwar hÄtte man sich im Notfall einer Umgehungslösung, wie das entwickeln eigener Treiber, widmen können, jedoch hÄtte dies die Durchführbarkeit

des Projekts gefÄhrdet. Ein Problem mit allen Frameworks ist es, dass sie nur sehr schlecht dokumentiert sind.

4.2.5 Analyseergebnis

Aufgrund der guten Eignung wird für die Umsetzung dieser Arbeit auf das Projekt jnect gesetzt, wie im Ergebnis der Analyse in Abbildung 4.6 zu sehen. Zwar werden die Anforderungen nicht vollumfanglich erfüllt, jedoch in jeweils ausreichendem Maße.

5 Konzeption

In diesem Kapitel wird kurz die Konzeption der Anwendung und der dazu gehörigen Komponenten beleuchtet und die Restriktionen aufgelistet, die aufgrund der Vorgaben durch die genutzten Techniken und Softwarekomponenten zustande kommen.

5.1 Technische Vorgaben

Hierunter fallen die Abhängigkeiten zu der Robotertechnik, die in den weiteren Arbeiten rund um diese Ausarbeitung entstehen, die Maßgaben, der Verwendung der Kinect und der Vorgaben der Software–Architektur.

5.1.1 Vorgaben durch Kinect

Die Kinect for Windows ist ein Produkt von Microsoft, so auch die Kinect for Windows . Microsoft vertreibt beides mit der Vorgabe, dass diese nur unter Windows-Betriebssystemen, vorrangig Windows 7, lauffähig sind, und vollständig unterstützt werden, sowie eine Installation des .Net Framework 4.0 auf dem System vorhanden sein muss [1] [2].

[1] Downloadseite des SDK. Microsoft.com. Abgerufen Dezember 21, 2012

[2] „Kinect for Windows Sensor Components and Specifications“. msdn.microsoft.com. Abgerufen Dezember 21, 2012

5.1.2 Vorgaben durch Java—Eclipse

Vorgaben durch das jnect-Framework

Das Framework jnect wurde bereits im Abschnitt 4.2.3. Um das Framework in der Eclipse Umgebung zu nutzen, muss lediglich das jnect-Plugin, sowie die beiden Module

`org.eclipse.emf.core`

Service–Architektur

Durch die Nutzung der OSGi-Plattform und der Eclipse IDE wird für die Service-Architektur auf Equinox gesetzt.

5.1.3 Abhängigkeit zu Robotertechnik

Der Betrieb eines Roboters und die Verwendung innerhalb der Anwendung, die im Rahmen dieser Arbeit entsteht, ist bisher nicht entgültig geklährt, da noch keine Auswahl eines speziellen Modells getroffen wurden ist.
Für die Implementierung in dieser Arbeit wurde ein hypothetisches Modell eines abstrakten Roboters angenommen, der in der Lage ist, sich in alle Richtungen eines drei dimensionalen Koordinatensystems zu bewegen. Es wird weiterhin davon ausgegangen, dass ein solcher Roboter in der Lage ist, sich in einem Kreis zu drehen und binnen weniger Sekunden anzuhalten.

5.2 Fachliche Vorgaben

Da bereits alle technischen Abhängigkeiten aufgezeigt wurden, werden nun die Bedingungen aufgelistet, die eigentlicher Fokus dieser Arbeit, nämlich der Analyse und Vorgaben der Gestensteuerung.

5.2.1 Vorgaben durch Mensch-Computer-Interaktion

Baudel und Beaudouin-Lafon [BBL93] beschreiben die Vorteile der Gestensteuerung gegenüber der, herkömmlicher Eingabegeräte. Dazu müssen aber einige Voraussetzungen erfüllt sein. Nach Baudel und Beaudouin-Lafon, sind dies:

Handanspannung: Eine angespannte Hand erleichtert die Wahrnehmung der Intention des Bedieners in einer Startposition eine Geste auszuführen. Endpositionen sollten hingegen frei von solch einer Anspannung sein

Schnelle, aufbauende, revidierbare Aktionen: Geschwindigkeit ist ein grundlegender Faktor in der Anwendung von Gesten, denn nur wenn Gesten einen Geschwindigkeitsvorteil bringen, werden diese auch eingesetzt. Zudem macht der Aufbau einer Geste aus kleinen Teilgesten, die Anwendung überschaubarer und rücknahme von Gesten bieten einen Komfortvorteil für den Nutzer

Favorisiere eine einfache Nutzung: Es muss stets ein Kompromiss zwischen einfachen und natürlichen Gesten, die einfach zu lernen sind, und komplexen Gesten, die mehr Kontrolle bieten, eingegangen werden. Dabei sollte stets die einfache Nutzung bevorzugt werden

Nutze Gesten, wo sinnvoll: Obgleich Gesten klare Vorteile bieten, muss stets berücksichtigt werden, dass diese auch ihre Grenzen haben. Gesten sind bisweilen unter anderem ungeeignet, um präzise Interaktionen durchzuführen. Hierfür wird immer noch ein physischer Kontakt benötigt (z.B., Steuerung des Cursors weiterhin per Maus)

5.3 Das resultierende Konzept

In Abschnitt 5.1.3 wurde bereits der hier verwendete hypothetische Roboter kurz vorgestellt. Dabei ergab sich folgende Liste an Funktionen:

- Kreisdrehung
- Geradeaus fahren
- In einem Winkel fahren
- Anhalten

Darüber hinaus kann noch das Blockieren der Gesteneingabe hinzugefügt werden. Setzt man nun all diese Funktionen und Möglichkeiten in Gesten und Bedienungsanweisungen um, so erhält man folgende Anweisungen:

- Kreisbewegung
- Vorwärtsbewegung
- Haltesignal
- Blockieren–Entriegeln

Diese Liste wird in Kapitel 7 verwendet, um die in der Anwendung eingesetzten Gesten zu modellieren.
Die aus der Liste entworfenen Sprachbefehle werden in Kapitel 8 erläutert.

6 Modelle zur Gesten- und Spracherkennung

Es existieren diverese Verfahren zur Gesten- und Spracherkennung. Die bekanntesten Vertreter sind künstliche neuronale Netze, Dynamic time warping (DTW) und HMM. Im Folgenden werden kurz die diversen Eigenschaften und Algorithmen hinter den weiteren Modell erörtert, um eine fundierte Entscheidung über die Auswahl eines dieser Modelle für diese Arbeit und das zu erarbeitende Programm zu treffen.

6.1 Dynamic Time Warping

DTW ist ein Algorithmus zur Messung von Ähnlichkeiten zwischen einer Eingabesequenz und einem gegebenen Muster, die sich im zeitlichen Ablauf oder der Geschwindigkeit unterscheiden. Dabei würden zum Beispiel die Gemeinsamkeiten im Laufmuster erkannt werden, selbst wenn eine Person in einem Video langsam läuft und diese Person in einer weiteren Aufnahme schneller laufen würde, oder gar eine Beschleunigung oder Verlangsamung während des Zeitraumes der Beobachtung stattfindet.
Formalisiert versteht man unter DTW eine vorlagenbasierte Erkennungstechnik auf Grundlage von dynamischer Programmierung [LK99, S. 963]. Trotz der Erfolge bei Aufgaben mit kleinem Vokabular, braucht DTW eine große Anzahl an Vorlagen für ein grösseres Umfeld an Variation. Weiterhin kann es keine undefinierte Muster verarbeiten. Takahashi et al. [NO96] haben einen *Erkundungsalgorithmus* vorgeschlagen, um Gesten

des Körpers und der Arme zu erkennen. Lee beschreibt die Funktionalität wie folgt: Ein Eingabemuster wird von einer Bildeingabesequenz zu jeder Zeit t generiert. Dabei wird der Zeitpunkt t als möglicher Endpunkt der Geste betrachtet und das Eingabemuster mit allen vordefinierten Mustern verglichen. Dynamische Programmierung wird dabei genutzt, um die Entfernung zwischen zwei Mustern zu berechnen, ungeachtet der zeitlichen Differenz.
Der Nachteil dieses Ansatzes ist die geringe Robustheit gegenüber Variationen der Form und Gestalt.

6.2 Künstliche neueronale Netze

Ein weiterer Ansatz ist die Nutzung von Künstliche neuronale Netze. Beale und Edwards [BE90] nutzten diese trainierbare Erkennungssysteme zur Erkennung von Handgesten. Eine weitere Arbeit von Beale und Finlay [FB93] nutzten diese Technik, denn sobald große Datenmengen verfügbar werden, um so mehr Relevanz erhält der Einsatz von künstlichen neuronalen Netzen.[1]
Ein Problem der neuronalen Netze im Kontext der Gestenerkennung ist jedoch die Modellierung von Mustern von *Nichtgesten*. Trotz der Effektivität bei der Erkennung von statischen Mustern, sind Künstliche neuronale Netze nach Vaananen und Boehm [VB93] nicht für dynamische Muster (Gesten) geeignet.

6.3 Hidden Markov Model

Das HMM ist ein stochastisches Modell, in dem ein System durch eine Markov-Kette mit unbeobachteten Zuständen modelliert wird. Die Theorie dazu wurde von Baum [BP66], 1966 veröffentlicht.
Es wird dazu angewendet, um Zeitreihen mit zeitlicher und räumliche Variabilität zu

[1] Vergleiche Lee und Kim [LK99].

analysieren. Dabei kann es auch mit undefinierten Mustern umgehen. Ein HMM besteht dabei aus verschiedenen Komponenten.
Der versteckte (englisch *hidden*) Prozess ist eine Markov-Kette und besteht aus Zuständen und Übergangswahrscheinlichkeiten. Das Modell ist wie folgt definiert:

Definition

Ein HMM $\lambda = (S, A, O, B, \pi)$ ist gegeben durch

- $S = S_1, \ldots, S_n$ – Menge aller Zustände,
- $A = a_{ij}$ – Übergangsmatrix zwischen den Zuständen, wobei a_{ij} die Wahrscheinlichkeit angibt, dass Zustand S_i in Zustand S_j gewechselt wird,
- $O = O_1, \ldots, O_m$ – Menge der möglichen Beobachtungen (*Emissionen*),
- $B = b_{ij}$ – Beobachtungsmatrix, wobei b_{ij} die Wahrscheinlichkeit angibt, im Zustand S_i die Beobachtung $O_j \in O$
- π – Anfangswahrscheinlichkeitsverteilung mit $\pi(i)$ Wahrscheinlichkeit, dass S_i der Startzustand ist.

6.4 Auswahl des Modells für die Arbeit

Rabiner et al. [Rab89, S. 257f], sowie Lee und Kim [LK99, S. 961] sprechen von den guten Erfolgen des HMM als stochastisches Werkzeug zur Modellierung von Gesten. Die Autoren stimmen dieser Auffassung zu. Daher wird für diese Arbeit eine Implementierung mittels HMM erarbeitet.

6.5 Diskrete Markov-Prozesse

Eine Markov-Kette ist ein Stochastischer Prozess. Dabei unterscheidet man zwischen Markov-Ketten in diskreter und in stetiger Zeit. Weiter spricht man von Markov-Ketten, sofern ein diskreter Zustandsraum vorhanden ist, von *Markov-Prozessen* im Allgemeinen wenn ein stetiger Zustandsraum vorliegt.
Bei einem HMM werden ausschließlich diskrete Markov-Ketten verwendet, die wie folgt definiert sind.

Definition

Gegeben sei ein System, das zu jeder gegebenen Zeit t beschrieben werden kann, als sich in einer Menge von N verschiedenen Zuständen $S_1, S_2, \ldots, S_N$ befindend. Abbildung 6.1 zeigt eine solche Markov-Kette mit fünf Zuständen S_1 bis S_5. In regelmäßigen diskreten

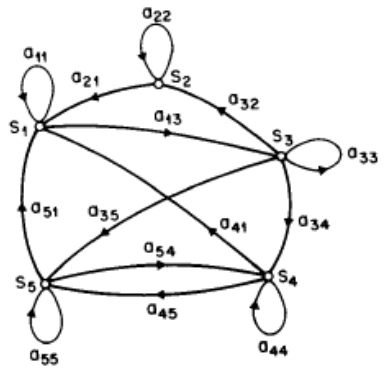


Abbildung 6.1: Markov-Kette mit fünf Zuständen (S_1 bis S_5) mit Verbindungen (Quelle: [Rab89])

Zeitabständen nimmt das System eine Zustandsänderung vor, gemäßeiner Wahrscheinlichkeitsmenge, die mit dem Zustand verknüpft ist. Die Zeitinstanzen, die mit den Zustandsänderungen verknüpft sind, werden durch $t = 1, 2, \cdots$ gekennzeichnet und der

aktuelle Zustand zum Zeitpunkt t als q_t. Für den Fall, einer diskreten Markov-Kette erster Ordnung ist diese Charakterisierung ausreichend, da

$$P[q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \cdots] = P[q_t = S_j | q_{t-1} = S_i]. \tag{6.1}$$

Weiterhin werden ausschließlich die Prozesse betrachtet, in denen die rechte Seite der Gleichung 6.1 zeitunabhängig ist, was somit zur Menge der Übergangswahrscheinlichkeiten a_{ij} der Form

$$a_{ij} = P[q_t = S_j | q_{t-1} = S_i], 1 \leq i, j \leq N \tag{6.2}$$

führt, wobei die Zustandsübergangskoeffizienten folgende Eigenschaften erfüllen:

$$a_{ij} \geq 0 \tag{6.3a}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{6.3b}$$

Ein solcher Stochastischer Prozess stellt aber noch kein HMM dar.
Um ein besseres Verständnis für diese Prozesse und deren Eigenschaften zu erlangen, wird im folgenden anhand eines Beispiels aus einer Anleitung von Lawrence Rabiner [Rab89] näher auf das Thema eingegangen.
Man betrachte ein einfaches Markov Modell, dass mittels drei Stati das Wetter beschreibt. Diese drei Zustände sind folgende:

Status 1: Regnerisch

Status 2: Bewölkt

Status 3: Sonnig

Es wird gefordert, dass das Wetter am Tag t durch einen der drei oberen Zustände beschrieben wird. Weiter wird die Matrix A der Wahrscheinlichkeiten der Zustandsübergänge wie folgt definiert:

$$\mathbf{A} = a_{ij} = \begin{pmatrix} 0,4 & 0,3 & 0,3 \\ 0,2 & 0,6 & 0,2 \\ 0,1 & 0,1 & 0,8 \end{pmatrix}$$

Das Wetter am ersten Tag ($t = 1$) sei als sonnig (Status 3) gegeben. Stellt man nun die Frage, nach der Wahrscheinlichkeit für eine Folge von Wetterzuständen, oder anders formuliert, für einen Wetterverlauf der nächsten sieben Tage, wie „sonnig, sonnig, regnerisch, regnerisch, sonnig, bewölkt, sonnig,...“, so kann man hierzu formal eine Sequenz aus mehreren Zustänen betrachten. Hierzu sei die Beobachtungssequenz O definiert als $O = S_3, S_3, S_1, S_1, S_3, S_2, S_3$, was mit $t = 1, 2, \cdots, 8$ übereinstimmt. Die Wahrscheinlichkeit kann ausgedrückt werden als,

$$\begin{aligned} P(O|Modell) &= P[S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3|Modell] \\ &= P[S_3] \cdot P[S_3|S_3] \cdot P[S_3|S_3] \cdot P[S_1|S_3] \cdot P[S_1|S_1] \cdot P[S_3|S_1] \cdot P[S_2|S_3] \cdot P[S_3|S_2] \\ &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{23} \\ &= 1 \cdot (0,8)(0,8)(0,1)(0,4)(0,3)(0,1)(0,2) \\ &= 1,536 \times 10^{-4}, \end{aligned}$$

wobei

$$\pi_i = P[q_1 = S_i], \qquad 1 \leq i \leq N \tag{6.4}$$

die Wahrscheinlichkeit für den initialen Status darstellt.
Auch die Frage nach der Wahrscheinlichkeit wie lange das Modell in einem gegebenen

und bekannten Status genau d Tage verbleibt, kann mittels HMM beantwortet werden. Dabei kann die Wahrscheinlichkeit durch die Beobachtungssequenz

$$O = \underset{1}{S_i}, \underset{2}{S_i}, \underset{3}{S_i}, \cdots, \underset{d}{S_i}, \underset{d+1}{S_j} \neq S_i$$

ermittelt werden:

$$P(O|Modell,\, q_1 = S_i) = (a_{ii})^{d-1}(1 - a_{ii}) = p_i(d) \tag{6.5}$$

Die Größe $p_i(d)$ ist die (diskrete) Wahrscheinlichkeitsdichtefunktion der Dauer d im Status i. Diese ist charakteristisch für die Dauer eines Status in einer Markov-Kette. Basierend auf $p_i(d)$ kann man die erwartete Zahl der Betrachtungen (Dauer) in einem Status berechnen, unter der Voraussetzung, dass in diesem Status gestartet wird, mit

$$\overline{d_i} = \sum_{d=1}^{\infty} d p_i(d) \tag{6.6a}$$

$$= \sum_{d=1}^{\infty} d(a_{ii})^{d-1}(1 - a_{ii}) = \frac{1}{1 - a_{ii}}\,. \tag{6.6b}$$

Daraus ergeben sich die Anzahl aufeinander folgender Sonnentage, gemäß dem Modell $\frac{1}{0,2} = 5$, die Anzahl aufeinanderfolgender bewölkter Tage entsprechend mit $2,5$ und die Anzahl aufeinanderfolgender Regentage mit $1,67$.

6.5.1 Erweiterung auf Hidden Markov Model

Um Markov Prozesse auf Probleme anwenden zu können, in denen sich die Beobachtungen nicht auf Zustand beziehen, muss das bisher dargestellte Konzept erweitert werden. Dazu wird der Fall hinzugefügt, dass eine Beobachtung durch eine Wahrscheinlichkeitsfunktion des Zustandes ausgedrückt wird. Nach Rabiner [Rab89] besteht das resultierende Modell aus einem doppelt eingebetteten stochastischen Prozess mit einem darunter liegenden, nicht beobachtbaren(versteckten, daher *hidden*), stochastischen Prozess und

wird als HMM bezeichnet.
Ein HMM ist wie folgt charakterisiert:

1. Sei N die Anzahl aller Zustände im Modell, dann sind die verschiedenen Zustände definiert als $S = S_1, S_2, \cdots, S_N$ und der Zustand zum Zeitpunkt t ist definiert als q_t.

2. Sei M die Anzahl aller verschiedenen Beobachtungssymbole pro Zustand, dann sind die individuellen Symbole definiert als $V = v_1, v_2, \cdots, v_M$.

3. Sei $A = a_{ij}$ die Übergangsmatrix, wobei

$$a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]\,, \qquad 1 \leq i, j \leq N\,. \tag{6.7}$$

   Für den speziellen Fall, dass jeder beliebige Zustand jeden beliebigen anderen Zustand in einem Schritt erreichen kann, ist $a_{ij} > 0$ für alle i, j. Für alle weiteren Typen eines HMM ist $a_{ij} = 0$ für mindestens ein (i, j) Paar.

4. Sei $B = b_j(k)$ die Beobachtungsmatrix im Zustand j, wobei

$$b_j(k) = P[v_k \,\text{zum Zeitpunkt}\, t \mid q_t = S_j]\,, \qquad \begin{aligned} &1 \leq j \leq N \\ &1 \leq k \leq M\,. \end{aligned} \tag{6.8}$$

5. Sei $\pi = \pi_i$ die Anfangswahrscheinlichkeitsverteilung, wobei

$$\pi_i = P[q_1 = S_i]\,, \qquad 1 \leq i \leq N\,. \tag{6.9}$$

Sind entsprechende Werte für N, M, A, B und π gesetzt, so kann das HMM als Generator verwendet werden, um eine Beobachtungssequenz

$$O = O_1\, O_2\, \cdots\, O_T \tag{6.10}$$

wie folgt zu erhalten:

1. Wähle einen Startzustand $q_1 = S_i$ entsprechend der Anfangswahrscheinlichkeitsverteilung π .
2. Setze $t = 1$.
3. Wähle $O_t = v_k$ entsprechend der Beobachtungsmatrix für den Zustand S_i, das heißt, $b_i(k)$.
4. Gehe in in einen neuen Zustand $q_{t+1} = S_j$ entsprechend der Beobachtungsmatrix für den Zustand S_i über, das heißt, a_{ij}.
5. Setze $t = t + 1$; kehre zu Schritt 3 zurück, falls $t < T$; beende andernfalls die Prozedur.

Nach Rabiner [Rab89, S. 5] kann diese Prozedur sowohl dazu verwendet werden, Beobachtungen zu generieren und als Modell, wie eine gegebene Beobachtungssequenz von einem entsprechendem HMM erstellt wurde.
Eine vollständige Spezifikation eines HMM benötigt die Beschreibung zweier Modellparameter (N und M), die Spezifikation von Beobachtungssymbolen und die Beschreibung der drei Wahrscheinlichkeitsgrößen A, B und π. Meist wird hierfür die kompakte Notation

$$\lambda = (A, B, \pi) \tag{6.11}$$

verwendet, um die Vollständigkeit aller Parameter des Modells anzuzeigen.

6.5.2 Die drei grundlegenden Probleme eines Hidden Markov Model [2]

Verwendet man die oben Form eines HMM, die in Abschnitt refsubsec:HMM beschrieben ist, so ergeben sich drei grundlegende Probleme, die vor einem Einsatz von HMM in einer Anwendung gelöst werden müssen. Diese lauten wie folgt:

[2]Die Ausführungen basieren auf den Vorträgen von Jack Ferguson von IDA an den Bell Laboratories

Problem 1: Sind die Beobachtungsmenge $O = O_1\, O_2\, \cdots\, O_T$ und ein Modell $\lambda = (A, B, \pi)$ gegeben, wie kann dann die Wahrscheinlichkeit der Beobachtungsmenge, $P(O \mid \lambda)$, effizient berechnet werden, sofern das Modell gegeben ist?

Problem 2: Sind die Beobachtungsmenge $O = O_1\, O_2\, \cdots\, O_T$ und ein Modell λ gegeben, wie w ählt man dann eine entsprechende Zustandsfolge $Q = q_1\, q_2\, \cdots\, q_T$ die für das gewünschte Resultat optimal ist?

Problem 3: Wie sind die Parameter des Modells $\lambda = (A, B, \pi)$ anzupassen, um $P(O \mid \lambda)$ zu maximieren?

Im folgenden sind Verfahren und Techniken aufgeführt, wie die soeben genannten Probleme gelöst werden können.

6.5.3 Das Vorwärts–Rückwärtsverfahren

Das Vorwärts–Rückwärtsverfahren [BE67] [BS68] [Rab89] ermöglicht eine Lösung des ersten Problems aus Abschnitt 6.5.2 [3]: Betrachtet man die Vorwärtsvariable $\alpha_t(i)$, definiert als

$$\alpha_t(i) = P(O_1\, O_2\, \cdots\, O_t\,,\; q_t = S_i \mid \lambda) \tag{6.12}$$

d.h., die Wahrscheinlichkeit der partiellen Menge der Beobachtungen, $O_1\, O_2\, \cdots\, O_t$, (bis Zeitpunkt t) und Zustand S_i zum Zeitpunkt t, bei gegebenem Modell λ. Induktiv ist dies wie folgt lösbar:

1) Induktionsanfang:

$$\alpha_1(i) = \pi_i b_i(O_1)\,, \qquad\qquad 1 \le i \le N\,. \tag{6.13}$$

[3] Zur Lösung des Problems 1 wird lediglich der Verwärtsteil des Verfahrens benötigt, der Rückwärtsteil wird im Problem 3 benötigt, aber dennoch hier behandelt

2) Induktionsschritt:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \,, \qquad 1 \leq t \leq T-1$$
$$1 \leq j \leq N \,. \tag{6.14}$$

3) Induktionsschluss:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) \,. \tag{6.15}$$

Der Induktionsanfang initialisiert die Vorwärtswahrscheinlichkeit als die kombinierte Wahrscheinlichkeit des Zustands S_i und der Anfangsbeobachtung O_1. Der Induktionsschritt ist Kern des Vorwärtsalgorithmus und ist dargestellt in Abbildung 6.2(a). Die Ab-

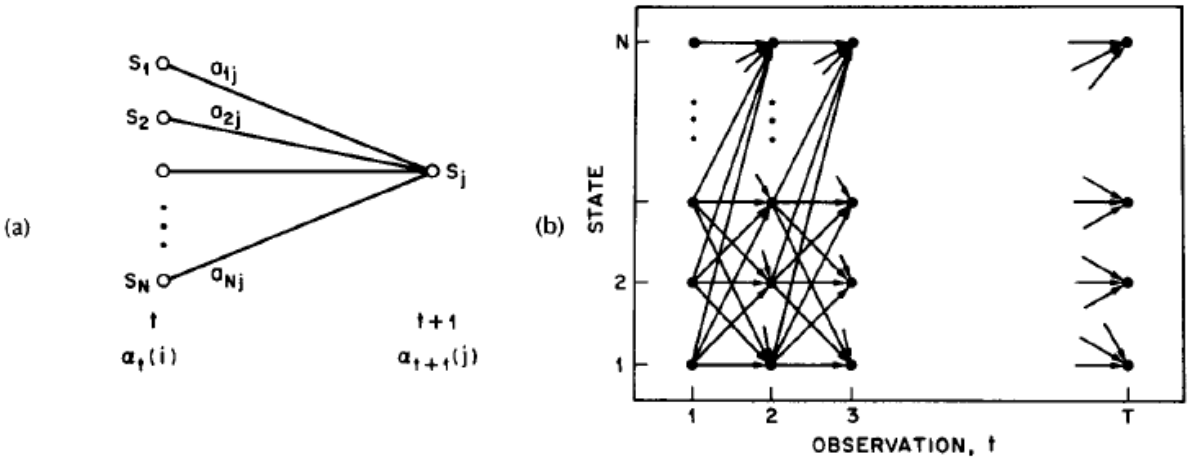


Abbildung 6.2: (a) Illustration der Oerpationsfolge zur Berechnung der Vorwärtsvariable $\alpha_{t+1}(j)$. (b) Implementierung der Berechung von $\alpha_t(i)$ im Netz der Beobachtungen t und Zustände i. (Quelle: [Rab89])

bildung zeigt wie der Zustand S_j zum Zeitpunkt t+1 von den N möglichen ZuständenS_i , $i \leq i \leq N$ vom Zeitpunkt t aus erreicht werden kann. Das Produkt $\alpha_t(i)a_{ij}$ stellt die Wahrscheinlichkeit des gemeinsamen Ereignisses dar, dass ($O_1\, O_2 \cdots O_t$ beobachtet wurden

und Zustand S_j von S_i aus erreicht wurde. Das Aufsummieren des Produkts ergibt die Wahrscheinlichkeit für S_j mit allen früheren partiellen Beobachtungen. Die Berechnung von 6.14 wird für alle Zustände j durchgeführt. Abschließend ergibt der Induktionsschluss die gewünschte Lösung von $P(O \mid \lambda)$ als die Summe der terminalen Vorwärtsvariablen $\alpha_T(i)$. Das ist der Fall, da laut Definition,

$$\alpha_T(i) = P(O_1\, O_2 \cdots O_T\,,\, q_T = S_i \mid \lambda) \tag{6.16}$$

und somit $P(O \mid \lambda)$ lediglich die Summe der $\alpha_T(i)$'s ist.
Die Vorwärtswahrscheinlichkeitsberechnung basiert auf der Gitterstruktur, die in Abbildung 6.2(b) dargestellt ist. Der Schlüssel liegt darin, da nur N Zustände (Knoten zu jedem Zeitintervall im Gitter) liegen, alle möglichen Zustandsfolgen, wieder in diese N Knoten fallen, ungeachtet der Länge der Beobachtungsfolge.
In änhlicher Weise kann eine R uckwärtsvariable $\beta_t(i)$ definert werden, als

$$\beta_t(i) = P(O_{t+1}\, O_{t+2} \cdots O_T \mid q_t = S_i\,,\, \lambda) \tag{6.17}$$

d.h., die Wahrscheinlichkeit der partiellen Beobachtungsfolgen von $t+1$ bis zum Ende, gegeben dem Zustand S_i zum Zeitpunkt t und dem Modell λ.
Ebenfalls induktiv wird $\beta_t(i)$ wie folgt gelöst:

Induktionsanfang:

$$\beta_T(i) = 1\,, \qquad\qquad 1 \le i \le N\,. \tag{6.18}$$

Induktionsschritt:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)\,, \qquad t = T-1, T-2, \cdots, 1$$
$$1 \le i \le N\,. \tag{6.19}$$

Der Induktionsschritt definiert $\beta_T(i)$ willkürlich als 1 für alle i. Der Induktionsschritt, dargestellt in Abbildung 6.3, zeigt, dass man um im Zustand S_i zum Zeitpunkt t gewesen zu sein, man alle möglichen Zustände S_j zum Zeitpunkt $t+1$ berücksichtungen muss. Die Berechnung von $\beta_t(i)$ kann ebenfalls in einer Gitterstruktur, ähnlich der in Abbildung 6.2(b) durchgeführt werden.

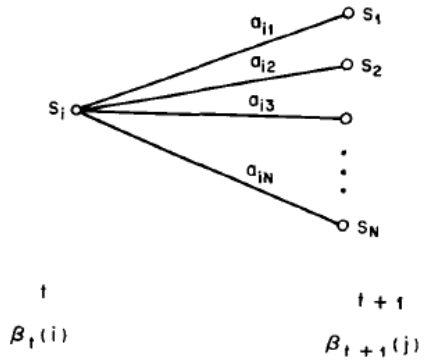


Abbildung 6.3: Illustration der Oerpationsfolge zur Berechnung der Rückwärtsvariable $\beta_t(i)$. (Quelle: [Rab89])

6.5.4 Anmerkungen zu Problem 2

Anders als für Problem 1, kann für Problem 2 aus Abschnitt 6.5.2 keine exakte Lösung angegeben werden. Dennoch gibt es mehrere Ansätze das Problem 2 hinsichtlich einzelner Kriterien zu lösen. Zum Einen kann nach Rabiner [Rab89] zum einen die „optimale“ Zustandsfolge verbunden mit der gegebenen Beobachtungsfolge gefunden werden. Die Schwierigkeit liegt bei der Definition der optimalen Zustandsfolge;das heißt, es gibt verschiedene mögliche Optimierungskriterien. Ein mögliches Optimierungskriterium ist, die Zustände q_t zu wählen, die *individuell* am wahrscheinlichsten sind. Um dies umzusetzen, wird die Variable

$$\gamma_t(i) = P(q_t = S_i \mid O\,,\, \lambda) \tag{6.20}$$

definiert, das heißt, die Wahrscheinlichkeit zum Zeitpunkt t im Zustand S_i zu sein, bei gegebener Beobachtungsfolge O und Modell λ. Gleichung 6.20 kann mittels Vorwärts–Rückwärts Variablen ausgedrückt werden, d. h.,

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{6.21}$$

wobei $\alpha_t(i)$ für die partielle Beobachtungsfolge ($O_1\, O_2 \cdots O_t$ und und Zustand S_i zum Zeitpunkt t steht, während $\beta_t(i)$ für die restliche Beobachtungsfolge $O_{t+1}\, O_{t+2} \cdots O_T$ steht, sofern Zustand S_i bei t gegeben ist. Der Normierungsfaktor $P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)$ macht $\gamma_t(i)$ ĕiner Wahrscheinlichkeitsgrö ße, so dass

$$\sum_{i=1}^{N} \gamma_t(i) = 1\,. \tag{6.22}$$

Benutzt man $\gamma_t(i)$, so kann man für den individuell wahrscheinlichsten Zustand q_t zum Zeitpunkt t lösen, als

$$q_t = \underset{1 \le i \le N}{\operatorname{argmax}}[\gamma_t(i)] \qquad\qquad 1 \le t \le T\,. \tag{6.23}$$

Trotz der Tatsache, dass Gleichung 6.23 die erwartete Anzahl an korrekten Zuständen maximiert, können Probleme in den resultierenden Zustandsfolgen auftreten, da lediglich der wahrscheinlichste Zustand in jedem Moment bestimmt wird, ungeachtet der Wahrscheinlichkeit des Auftretens von Zustandsfolgen.
Daher wird meist ein anderes Optimierungskriterium verwendet, das Auffinden der *einzelnen* besten Zustandesfolge, d. h., die Maximierung von $P(Q \mid O\,, \lambda)$. Die Technik, diese beste Zustandsfolge zu finden, basiert auf dynamischer Programmierung und ist der sogenannte Viterbi Algorithmus [Vit67] [FJ73].
Dieser spielt für diese Arbeit aber keine Rolle und wird daher nicht weiter betrachtet. Für weitere Informationen wird auf die Literatur verwiesen.

6.5.5 Baum-Welch-Algorithmus

Das dritte und bei weitem schwierigste Problem eines HMM, ist die Bestimmung einer Methode zur Anpassung der Modellparameter (A, B, π), um die Wahrscheinlichkeit der Beobachtungssequenz eines gegebenen Modells zu maximieren.
Es ist laut Rabiner [Rab89, S. 264] kein analytisches Verfahren zur Lösung des Problems bekannt. Es ist jedoch möglich, so Rabiner weiter, ein $\lambda = (A, B, \pi)$ so zu wählen, dass $P(O \mid \lambda)$ lokal maximiert wird. Das bekannteste Verfahren hierzu ist der sogenannte Bauch-Welch-Algorithmus [BPSW70] [Wel03].
Rabiner [Rab89, S. 264ff] stellt hierzu eine detaillierte Beschreibung des Algorithmus bereit.

6.5.6 Typen des Hidden Markov Model

Bislang wurde lediglich der Spezialfall eines sogenannten *ergodischen* oder vollständig verbundenen HMMs betrachtet, wobei jeder Zustand des Modells von jedem anderen Zustand des Modells aus (in einem Schritt) erreicht werden. (Streng mathematisch gesprochen: in einer finiten Anzahl von Schritten). Wie in Abbildung 6.4(a) zu sehen, hat dieses $N = 4$ Zustandsmodell, die Eigenschaft, dass jeder a_{ij} Koeffizient positiv ist. Aus dem Beispiel in Abbildung 6.4(a) erhalten wir

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} . \tag{6.24}$$

Andere Typen von HMMs bieten bessere Beobachtungseigenschaften des Signals und stellen sich für einige Anwendungen daher als besser modelliert heraus, als das ergodi-

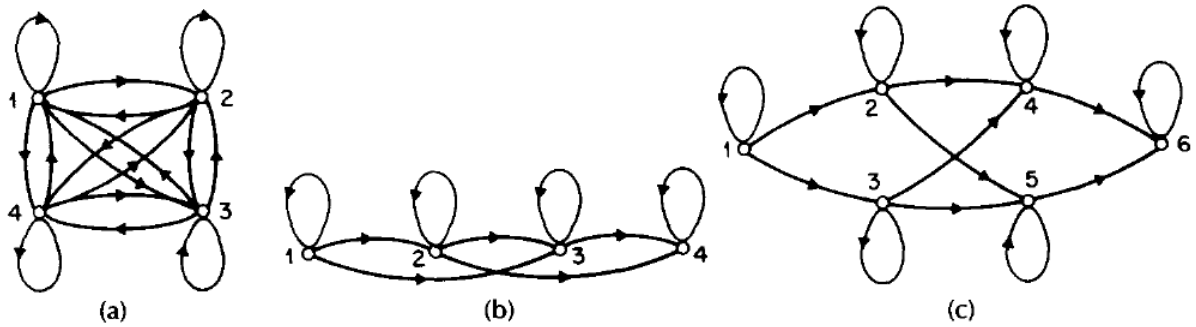


Abbildung 6.4: Darstellung dreier verschiedener Typen eines HMMs. (a) 4-Zustand ergodisches Modell. (b) 4-Zustand links-rechts Modell. (c) 6-Zustand Parallelpfad links-rechts Modell. (Quelle: [Rab89])

sche Modell. Eines dieser Modelle ist das aus Abbildung 6.4(b). Dieses Modell ist ein sogenanntes links-rechts Model oder Bakis Model [Jel76] [Bak76], da die dahinter liegende Zustandsfolge, die mit dem Modell verbunden ist, die Eigenschaft hat, dass sofern die Zeit zunimmt, der Zustandsindex zunimmt (oder gleich bleibt), d. h., die Zustände verlaufen von links nach rechts. Die fundamentale Eigenschaft aller links-rechts HMMs ist, dass die Zustandsübergangskoeffizienten die Eigenschaft

$$a_{ij} = 0\,, \qquad\qquad j \leq i \tag{6.25}$$

besitzen, d. h., keine Übergänge zu Zustände, deren Indizes kleiner als der gegenwärtige sind, sind zugelassen. Weiter haben die Anfangswertwahrscheinlichkeiten folgende Eigenschaft

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases}$$

nachdem die Zustandsfolge im Zustand 1 starten (und im Zustand N enden) muss. Bei links-rechts Modellen werden oft weitere Bedingungen an die Zustandsübergangskoeffizienten

geknüpft, um sicher zustellen, dass große Änderungen in den Zustandsindizes nicht auftreten; daher wird folgende Bedingung häufing verwendet:

$$a_{ij} = 0\,, \qquad\qquad j > i + \Delta\,. \tag{6.26}$$

Insbesondere für das Beispiel aus Abbildung 6.4(b), ist Δ gleich 2, das heißt, keine Sprünge über mehr als zwei Zustände sind zulässig. Die Übergangsmatrix für dieses Beispiel sieht demnach wie folgt aus:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \tag{6.27}$$

Dabei sollte klar sein, dass für den letzten Zustand in einem links-rechts Modell, die Zustandsübergangskoeffizienten definiert sind, als

$$a_{NN} = 0 \tag{6.28a}$$

$$a_{Ni} = 0\,, \qquad\qquad i < N\,. \tag{6.28b}$$

Daneben existieren natürlich noch zahlreiche weitere Variationen und Kombinationen von Modelltypen. Als Beispiel hierfür zeigt Abbildung 6.4(c) eine doppelt überkreuzt Verbindung zweier paralleler links-rechts HMMs.

7 Gesten

Gestik spiegelt einen wesentlichen Teil unseres Alltags dar. Als Teil der nonverbalen Kommunikation dienen sie zum Ausdruck von Emotionen, zur Übermittlung von Einstellungen, zur Darstellung von Persönlichkeitseigenschaften oder Modulation einer verbalen Nachricht, so Archer und Akert [AA84].
Nachdem bereits die wissenschaftliche Definition einer Gesten nach Kurtenbach und Hulteen in Kapitel 1 aufgeführt wurde und einführende Beispiele erläutert wurden, werden hier Gesten tiefgehender betrachtet.

7.1 Kategorisierung

Gesten können isoliert, oder im Zusammenhang mit einem externen Einfluss oder Objekt existieren. Ein bekanntes Beispiel für isolierte oder auch freigestellte Gesten, ist die Zeichensprache. In Bezug auf Objekte gibt es eine große Bandbreite an möglichen Gesten. Daher wurden Gesten in Klassen eingeteilt, um diese besser unterscheiden zu können. Die Klassifikation nach Cadoz [Cad94], die Gesten nach deren Funktion gruppiert, defniniert drei Typen:

- *semiotisch*: Diejenigen Gesten, die aussagekräftige Informationen kommunizieren
- *ergodisch*: Diejenigen Gesten, die genutzt werden, um das reale Umfeld zu manipulieren und Gegenstände zu erstellen

- *epistemisch*: Diejenigen Gesten, die genutzt werden, um von der Umwelt durch taktile und haptische Erkundung zu lernen

Darüber hinaus werden diese Klassen weiter unterteilt. Das Augenmerk dieser Arbeit liegt auf den semiotischen Gesten. Innerhalb dieser Kategorie liefert Mulder [Mul96] eine weitere Sammlung an verschiedenen Klassifikationen.
Da der Fokus dieser Arbeit auf der Mensch-Computer-Interaktion liegt, werden vorrangig sogenannte *empty handed* semiotische Gesten betrachtet. Diese können entsprechend ihrer Funktionalität weiter unterteilt werden. Rime und Schiaratura [RS91] beschreiben folgende Taxonomie:

- *symbolische Gesten*: Dabei handelt es sich um Gesten, die innerhalb eines bestimmten Kulturkreises, eine allgemeingültige Bedeutung erlangt haben. Das „Daumen Hoch"-Zeichen, währe ein solches Beispiel

- *deiktische Gesten*: Unter dieser Kategorie fallen Gesten des Zeigens oder der Richtungsweisung. Diese stehen meist in einem Kontext, wie „Lege das hier hin"

- *ikonische Gesten*: Darunter fallen Gesten, die Informationen über Form, Gestalt und Ausprägung eines bestimmten Gegenstandes oder einer Handlung

- *pantomimische Gesten*: Dies sind Gesten, die typischer Weise die Nutzung eines nichtgegenwärtigen Gegenstandes oder Werkzeugs wiederspiegeln. Während der Mimik einer Aktion wird dabei eine pantomimische Geste gemacht

Baudel und Beaudouin-Lafon [BBL93] zeigen einige Vorteile der Nutzung von symbolischen Gesten zur Interaktion auf:

- *Natürliche Interaktion*: Gesten sind eine selbstverständliche Form der Interaktion und einfach zu verwenden

- *Knapp und Ausdrucksstark*: Eine Geste übermittelt genug Information, um Befehl und Paramter zu liefern

- *Direkte Interaktion*: Einzelne Körperteile, insbesondere die Hand, als Eingabegeräte machen weitere Signalgeber zwischen Eingabe und Empfänger obsolet

Natürlich bergen diese auch Nachteile. So können symbolische Gesten auf Dauer anstrengend und aufwänding werden. Ein Nutzer benötigt in aller Regel Training und ein detailiertes Wissen über Anwendung und Funktion der Gesten, bevor er in der Lage ist, die Anwendung mittels symbolischen Gesten zu bedienen. Je weiter die Anzahl der Gesten und deren Komplexität steigt, desto schwieriger wird es, sich an bestimmte Gesten zu erinnern.
Weiter stellt sich ein Segmentierungsproblem, dahingehend, dass eine Bewegungsdetektion in aller Regel, den gesamten Verlauf einer Handbewegung, oder die eines anderen Körperteils erfasst und die eigentliche Geste nur einen Ausschnitt dieses kontinuierlichen Signalstroms darstellt.
Es ist daher wichtig, Gesten, die in einer Anwendung verwendet werden sollen, so zu modellieren, dass sie einfach auszuführen sind, eine angemessene Unterscheidung zwischen diversen Gesten vorhanden ist und diese sich möglichst nahe an natürlichen Bewegungen orientieren.

7.2 Gesten in der Anwendung

Nachdem bereits in Abschnitt 5.3 die Anforderungen an Gesten ausgeführt wurden, werden hier die daraus entwickelten Gesten aufgef'ührt. Dabei werden alle grundlegenden Vorgaben für ein Mensch-Computer-Interaktion aus Abschnitt 5.2.1 berücksichtigt und die zu entwerfenden Gesten auf Basis der oben beschriebenen Kategorie der semiotischen Gesten modelliert.

7.2.1 Kreisbewegung

Ein abstrakter Roboter, ist in aller Regel in der Lage, sich im Stand vollständig um seine eigene Achse zu drehen. Dies ist eine hilfreiche Funktion zur Steuerung eines solchen Geräts. Betrachtet man diese Drehung, so findet im Grunde eine Kreisbewegung statt. Um eine intuitive Assoziation zwischen dieser Drehung und der dafür zu verwendeten Geste herzustellen, wird zur Durchführung dieser Geste, eine Kreisbewegung verlangt. Das heißt bildlich gesprochen, dass der Bediener mit seinen Händen einen Kreis in die Luft zeichnen muss, um die Geste auszuführen.
Die Ausführung dieser Bewegung ist intuitiv und ohne nennenswerten Trainingsaufwand anwendbar.

Fachliche Aspekte

Die Kreisbewegung ist eine ikonische Geste, da sie die Form eines Kreises wiedergibt.

Technische Aspekte

In einem HMM wird diese Bewegung durch acht Zustände modelliert. Abbildung 7.1 zeigt ein ideales Bild dieser Geste mit ihren acht Zuständen.

Diese acht Zustände bilden ein Mittelmaßdar, um auf der einen Seite kein unterdimensioniertes HMM aufzustellen, auf der anderen Seite, aber genügend Spielraum für die Akzeptanz von Kreisbewegungen einer etwas ungenauen Form zuzulassen. Abbildung 7.2 zeigt mehrere solcher Kreise, die aus Trainingsdaten ermittelt worden sind.

[1] Quelle: „How to Do Gesture Recognition With Kinect Using Hidden Markov Models (HMMs)“. creativedistraction.com. Abgerufen Januar 14, 2013

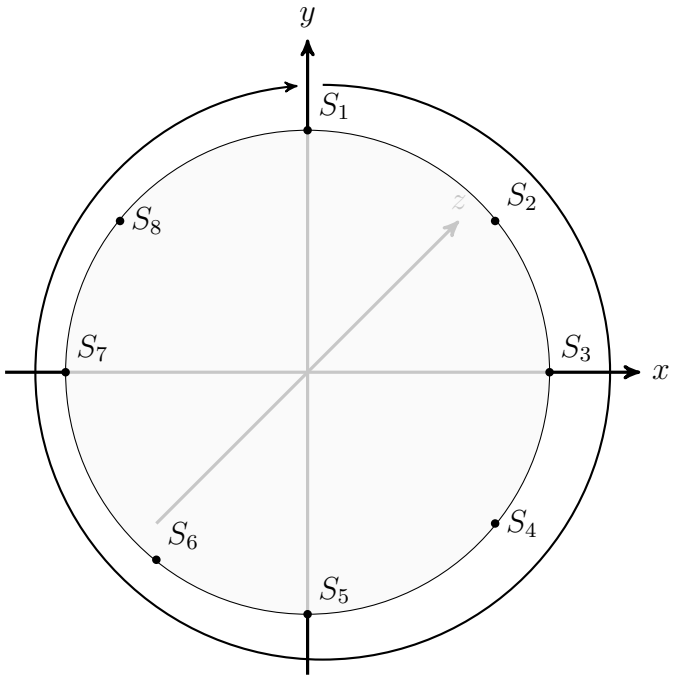


Abbildung 7.1: Abstrakte Darstellung einer Kreisbewegung im Koordinatensystem inklusiver ihrer 8 Zustandspunkte

Die Übergangsmatrix A, für die Geste einer Kreisbewegung wird definiert, als

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{66} & a_{67} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{77} & a_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{88} \end{pmatrix}. \tag{7.1}$$

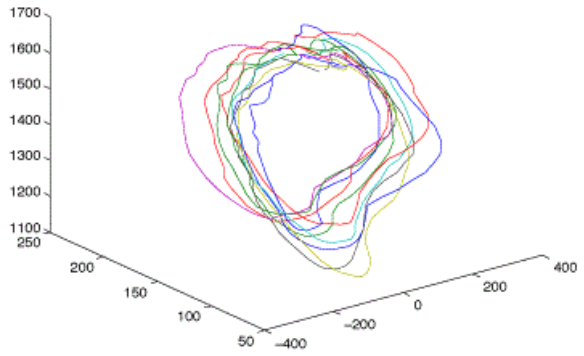


Abbildung 7.2: Darstellung von Kreisbewegungen im Koordinatensystem die aus realen Trainingsdaten stammen[1]

Dabei ist der Übergangskoeffizient a_{88} gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden.

7.2.2 Vorwärtsbewegung

Die Fortbewegung und die grundlegende Bewegung nach vorne ist die Voraussetzung, damit ein Roboter überhaupt bedient werden kann. Doch diese Bewegung als Befehl zu übergeben, ist wiederum nicht so simpel. Es mehrere Möglichkeiten, eine Geste hierfür umzusetzen. Die Lösung, die hier verwendet wird, basiert auf der Idee der Richtungsanweisung. Das heißt, dem Roboter wird die Anweisung gegeben, sich geradeaus nach vorne zu bewegen.
Hierbei wird wird ausgehend von der Hand modelliert. Dabei wird ausgehend, von einer Körperhaltung, in der die überwachte Hand in Nähe des Köpers auf Höhe des Schulterblatts liegt, der Ausgangspunkt festgelegt. Die Ausührung und Verständlichkeit dieser Geste ist relativ einfach, da sie einfach eine fallende Handbewegung dargestellt. Abbildung 7.3 zeigt eine schematische Darstellung mit vier Zuständen.

Fachliche Aspekte

Die Vorwärtsbewegung ist eine deiktische Geste, da sie eine Richtungsanweisung wiedergibt.

Technische Aspekte

In einem HMM wird diese Bewegung durch vier Zustände modelliert. Diese vier Zustände bilden ähnlich, wie bei der Kreisbewegung, ein Mittelmaßdar, ausreichend Spielraum für die Akzeptanz einer Vorwärtsbewegung einer etwas ungenauen Form zu besitzen.

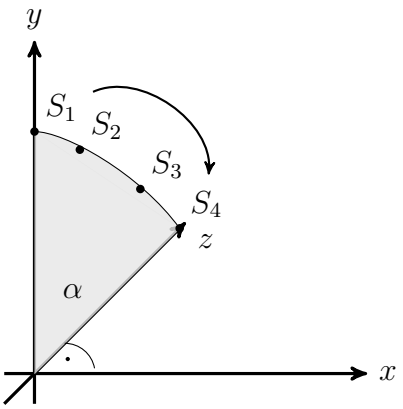


Abbildung 7.3: Abstrakte Darstellung einer Vorwärtsbewegung im Koordinatensystem inklusive ihrer 4 Zustandspunkte

Die Übergangsmatrix A, für diese Geste wird definiert, als

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}. \tag{7.2}$$

Die selben Bedingungen für die Übergangskoeffizienten gelten für die Vorwärtsbewegung, als für die Kreisbewegung definiert wurden.
DerÜbergangskoeffizient a_{44} ist gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptdiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden.

7.2.3 Erweiterte Vorwärtsbewegung

Die oben beschriebene Vorwärtsbewegung beinhaltet keine Information über einen Winkel, in dem sich der Roboter fortbewegen soll. Um diese Information zu erhalten, muss die Geste und das darunter liegende HMM angepasst und erweitert werden.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}. \tag{7.3}$$

Die selben Bedinungen für die Übergangskoeffizienten gelten für die Vorwärtsbewegung, als für die Kreisbewegung definiert wurden.
DerÜbergangskoeffizient a_{44} ist gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptdiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$

gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}. \tag{7.4}$$

Die selben Bedingungen für die Übergangskoeffizienten gelten für die Vorwärtsbewegung, als für die Kreisbewegung definiert wurden.
DerÜbergangskoeffizient a_{44} ist gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptdiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden.

Dabei gibt es zwei Varianten, wie diese Information extrahiert werden kann. Variante 1, siehe Abbildung 7.4, zeigt, wie an die bestehende Geste eine weitere Bewegung angefügt wird, um die Winkelinformation θ zu erhalten, Variante 2, siehe Abbildung 7.5 , eine überarbeitete Geste, in der der Winkel θ direkt über das Koordinatensystem extrapoliert wird.
Variante 2 stellt sich unter realen Testbedingungen als Problemanfällig und ungenau heraus, da bereits bei der Gestenerkennung eine gewisse Unschärfe vorherrscht. Der Winkel θ wird dabei also verfälscht wiedergegeben und es kann nicht sichergestellt werden, dass die Intention des Bedieners vollständig in die Informationen, die die Geste an den Roboter liefert einfließt. Daher wird diese Variante nicht weiter verfolgt.
Variante 1 kann anschaulich in zwei Teile zerlegt werden. Neben Abbildung 7.4(a), die den gesamten Ablauf zeigt, weist Abbildung 7.4(b) auf den zweiten Teil hin, der die Geste dahingehend erweitert, dass mit einer relativen Genauigkeit, der Winkel θ aus der Geste ausgelesen werden kann. Die Größe des Winkels, oder die Geschwindigkeit, mit der die Geste ausgeführt wird, spielt keine besondere Rolle, denn sobald das HMM entsprechend trainiert wurde, wird die Geste in aller Regel, verlässlich erkannt.

Technische Aspekte der Variante 1

Ausgehend von der vorhandenen Vorwärtsbewegung und kann das HMM der erweiterten Vorwärtsbewegung und deren Übergangsmatrix A definiert werden, als

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & a_{66} \end{pmatrix}. \tag{7.5}$$

Dabei ist der Übergangskoeffizient a_{66} gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptdiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden.

7.2.4 Haltesignal

Die Möglichkeit den Roboter anzuhalten ist die wichtigste Funktion, da hierbei die Sicherheit vor Unfällen eine wesentliche Rolle spielt. Der Roboter muss möglichst schnell und einfach zu stoppen sein. Sei es einfach aus dem Grund, dass der Bediener seine Aufgabe abgeschlossen hat, oder der nicht unerhebliche Fall der Unfallvermeidung oder Sicherstellung des Roboters eintritt.
Auf Basis dieses Sicherheitsgedanken wurde das Haltesignal modelliert und hierzu die menschliche Haltung bei Gefahr betrachtet. Der Mensch tritt bei drohender Gefahr in eine defensive Stellung, tritt einen Schritt zurück, oder zieht Gelenke, wie Arme oder

Beine näher zum Körperschwerpunkt.
Um die Erkennung der Geste zu optimieren und auch in den verschiedenen Startzuständen zu erkennen zeigt die Abbildung 7.6, zwei abstrakte Modelle mit entsprechenden Zuständen.
Abbildung 7.6(a) zeigt, wie ein Haltesignal aus einer Vorwärtsbewegung heraus erzeugt werden. Dabei spielt der Winkel θ nur dann eine Rolle, wenn er sich während der laufenden Geste stark ändert, da dann nicht mehr sichergestellt ist, ob diese Geste überhaubt beabsichtigt war.
In Abbildung 7.6(b) is dargestellt, wie aus einer Kreisbewegung in das Haltesignal übergegangen wird. Dabei ist der Winkel θ ebenfalls nur dann relevant, sofern zweifeln an der Intention der Geste besteht, da dieser sich ändert.

Fachliche Aspekte

Das Haltesignal ist eine pantomimische Geste, da sie den Schutzgedanken wiedergibt und hierbei die Emotion in eine defensive Bewegung oder Haltung resultiert.

Technische Aspekte

Die Geste für das Haltesignal daran angelehnt, so ausgeführt, dass der Bediener mindestens eine Hand zum Körper zurückziehen muss. Eine abstrakte Ansicht hierzu liefert die Abbildung 7.6. Mittels vier Zuständen wird hier ein HMM modelliert.
Wichtig hierbei ist, dass der Bediener sich zum Start des Haltesignals gerade in einer Vorwärtsbewegung, oder einer Kreisbewegung befinden kann. Daher werden, wie in Ab-

bildung 7.6 zu sehen, zwei Varianten erkannt und akzeptiert. Für die Übergangsmatrix A hat das keine Auswirkung, diese ist für beide Fälle definiert, als

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}. \tag{7.6}$$

Die selben Bedingungen für die Übergangskoeffizienten, wie zuvor, gelten auch hier, jedoch mit einer wichtigen Ausnahme: DerÜbergangskoeffizient a_{44} ist gleich 1. Weiter wird zur Initialisierung für die weiteren Indizes i, j auf der Hauptdiagonalen und der rechten Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden. Die Beobachtungsfolge $O = O_1\, O_2\, O_3\, O_4$ besteht bei den zwei Varianten, jedoch aus verschiedenen Zuständen S_1, S_2, S_3, S_4.

7.2.5 Entriegeln — Blockieren

Eine zur Steuerung eines Roboters nicht notwendige, dennoch recht nützliche Geste zur Entriegelung und Blockierung weiterer Gesteneingaben. Dem Bediener, auf der einen Seite, wird ein gewisser Komfort dadurch geboten, dass er nicht ständig gegenwärtig und bewusst darauf achten muss, natürliche Bewegungen nicht mit Gestenbefehlen zu vermischen. Auf der anderen Seite erlangt das gesamte System der Anwendung dabei an Stabilität, da Fehleingaben des Nutzers minimiert werden.
Angelehnt an ein Schiebeschloss muss hierbei lediglich entweder eine Handbewegung von links nach rechts (zum Blockieren), wie in Abbildung 7.7 dargestellt, als auch eine Handbewegung von rechts nach links (zum Entriegeln) ausgeführt werden.

Fachliche Aspekte

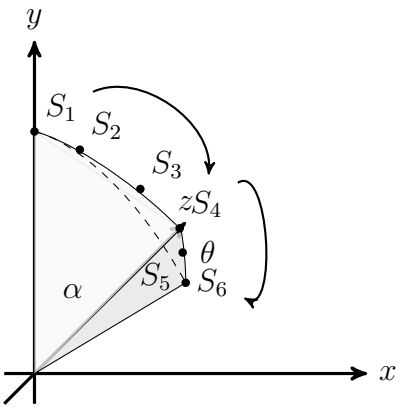
Das Entriegeln — Blockieren ist eine pantomimische Geste, da sie das Werkzeug *Schloss* nachmiemt.

Technische Aspekte

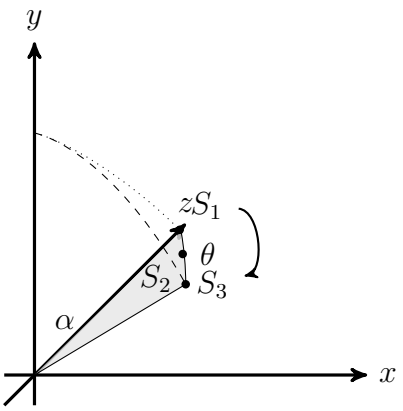
Das besondere an dieser Geste, ist die Tatsache, dass sie von beiden Richtungen aus durchgeführt werden kann und dabei unterschiedliche Aktionen ausführt.
Daher verhält sich die Übergangsmatrix A anders, als dies bisher der Fall ist. Zusätzlich zur rechten Nebendiagonalen ist hier auch die linke Nebendiagonale mit Koeffizienten a_{ij} besetzt. Demnach ist die Matrix definiert, als

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix}. \tag{7.7}$$

DerÜbergangskoeffizient a_{44} ist hier ungleich 1, da er nicht nur Endpunkt, sondern auch Start der Geste sein kann. Daher werden zur Initialisierung für alle Indizes i, j auf der Hauptdiagonalen und den Nebendiagonalen, $a_{ij} = 0,5$ gesetzt. In Trainingsdurchläufen müssen diese Wahrscheinlichkeiten eventuell vereinzelt angepasst werden, um insbesondere a_{11} und a_{44} optimal zu bestimmen. Die Beobachtungsfolge $O = O_1\, O_2\, O_3\, O_4$ besteht bei den zwei Varianten $(\Leftarrow, \Rightarrow)$, jedoch aus den umgedrehten Zustandensfolgen $S_1\, S_2\, S_3\, S_4(\Rightarrow)$ und $S_4\, S_3\, S_2\, S_1(\Leftarrow)$.



(a) Darstellung der gesamten erweiterten Vorwärtsbewegung mit 6 Zuständen



(b) Detaildarstellung der Seitwärtsbewegung der erweiterten Vorwärtsbewegung und ihrer 2 Zustände

Abbildung 7.4: Idealisierte Darstellung der 1. Variante einer erweiterten Vorwärtsbewegung mit 6 Zuständen

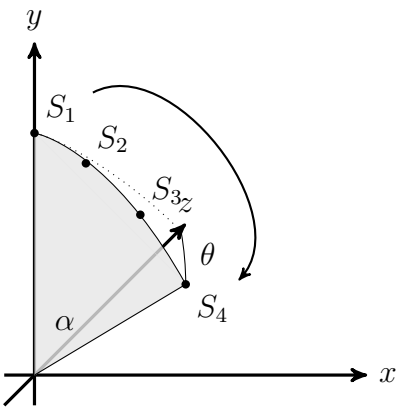
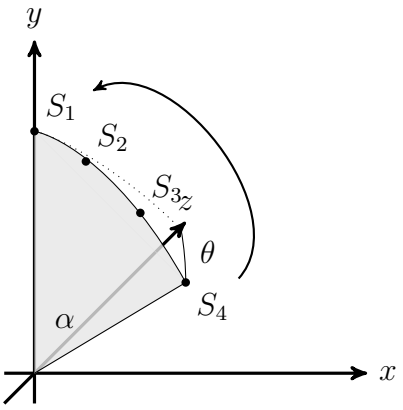
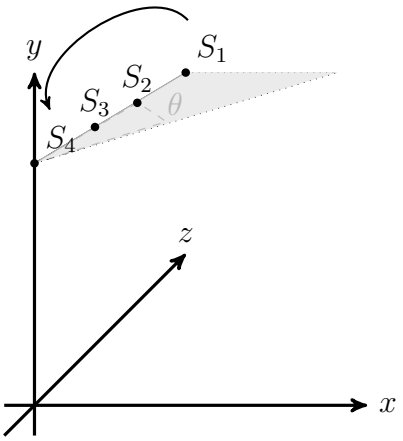


Abbildung 7.5: Idealisierte Darstellung der 2. Variante einer erweiterten Vorwärtsbewegung mit 4 Zuständen



(a) Abstrakte Darstellung des Haltesignals aus einer möglichen Vorwzärtsbewegung heraus, mit ihren 4 Zuständen



(b) Abstrakte Darstellung des Haltesignals aus einer möglichen Kreisbewegung heraus, mit ihren 4 Zuständen

Abbildung 7.6: Abstrakte Darstellung der Modelle für die Geste des Haltesignals mit jeweils 4 Zuständen

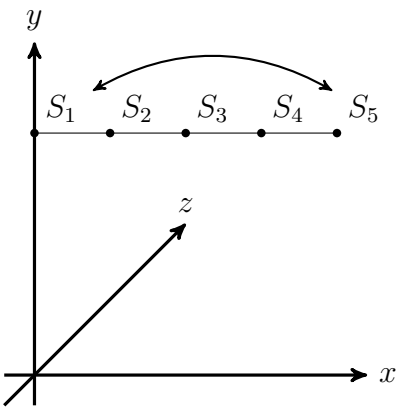


Abbildung 7.7: Idealisierte Darstellung der Geste zum Blockieren und Freigeben der Gesteneingabe mit 4 Zuständen

8 Sprachbefehle

8.1 Sprache

Sprache ist für Menschen die natürlichste Form der der Verständigung. So definiert Edward Sapir (1921):

> Sprache ist eine ausschließlich dem Menschen eigene, nicht im Instinkt wurzelnde Methode zur übermittlung von Gedanken, Gefühlen und Wünschen mittels eines Systems von frei geschaffenen Symbolen[1]

Menschen nutzen Sprache zum Ausdruck ihrer selbst, sowie für Beschreibung und Manipulation ihrer Umgebung.

8.2 Sprache und Gesten

Da die menschliche Sprache sehr komplex und auch unpräzise ist, ist die die korrekte Deutung der erkannten Sprache für eine Maschine sehr komplex. Stellt man jedoch den Bezug zu weiteren menschlichen Interaktionen her, wie die Gestik, so wird die Deutung der Bedeutung erheblich vereinfacht. Umgekehrt kann durch Sprache beschrieben werden, was durch eine Geste ungenau ausgedrückt wurde. Cohen [Coh92, CT89] zeigt hierzu, dass Sprache sich besser zur Beschreibung eignet als Gesten und das umgekehrt

[1] Zitiert nach John Lyons, 4.Auflage, 1992, S.13

Gesten besser für eine direkte Manipulation geeignet sind. Dass eine Kombination dieser beiden Modalitäten für Nutzern bei grafischen Aufgaben bevorzugt werden zeigen auch Hauptman und McAvinney [HM93].

8.3 Kinect for Windows SDK und Spracherkennung

Die Spracherkennung erfolgt in diesem Fall durch die Sprachengine des Kinect for Windows SDK.

Zur Verwendung von Sprachbefehlen mit jnect wird das Interface *SpeechListener* implementiert. Eine Implementierung ist in Listing 8.1 zu sehen. Diese Implementierung wird dem *KinectManager* übergeben. Erkennt das Kinect SDK eines der Worte, auf das es mittels der Methode *getWords()* Zugriff hat, wird die Methode *notifySpeech(...)* mit dem erkannten Wort als String aufgerufen. Umgesetzt ist der *SpeechListener* derzeit im Action Bundle (Abschnitt 9.2.3).

```
1  public class SpeechlistenerImpl implements SpeechListener{
2  {
3
4    Set<String> words = new HashSet<String>();
5
6    @Override
7    public void notifySpeech(String speech) {
8      System.out.println("Detected :" +speech);
9    }
10
11   @Override
12   public Set<String> getWords() {
13     words.add("Hello");
14     words.add("World");
15     words.add("Hello World");
16
17     return words;
```

```
18     }
19 };
```

Listing 8.1: jnect SpeechListener Implementation

Wörter und Phrasen

Bei Auswahl der verwendbaren Befehle sollte neben ihrer Zweckmäßigkeit auch darauf geachtet werden, dass diese für die Sprachengine des Kinect SDK gut erkennbar bleiben [Cor12]. Probleme können hier komplexe oder exotische Worte bereiten. Aber auch einfache Worte können schwer erkennbar sein, wenn es viele ähnliche oder gleich klingende Worte in der Zielsprache gibt. Wichtig ist es hier die richtige Kombination aus passenden und mögichst klaren Worten zu finden.

User Interfaces für Spracheingabe

Zur besseren Nutzbarkeit von Sprachkommandos ist es sinnvoll dem Nutzer ein entsprechendes Feedback zu geben [Cor12]. Sinnvoll hierfür erweist es sich dem Nutzer visuell dar zu bieten, welchen aktuell nutzbaren Sprachbefehle verfügbar sind. Als ebenfalls hilfreich erweist sich die Ausgabe des erkannten Befehls, und eventuellen Nachfragen oder Verbesserungen, falls ein Befehl nicht eindeutig erkannt werden konnte.

8.4 Sprachkommandos für Roboter

Da bisher keine näheren Informationen zum Roboter, welcher im zweiten Teil der Arbeit angesteuert werden soll, vorliegen, ist es an dieser Stelle schwer konkrete Sprachbefehle zu bestimmen, mit denen der Roboter später ferngesteuert werden soll. Die einzige konkrete Information die vorliegt ist, dass es sich um einen mobilen Roboter handeln wird. Jedoch lässt auch dies offen, ob der Roboter sich nur zweidimensional auf einer Ebene bewegen

wird oder ob es sich möglicherweise um eine Drohne handeln wird, welche ferngesteuert wird. Diese kann sich auch dreidimensional fortbewegen.

Bewegung

Da es sich um einen mobilen Roboter handelt wird, können insofern schon einmal Annahmen getroffen werden. Auf jeden Fall wird dieser sich vorwärts und rückwarts bewegen können. Hieraus ergeben sich die ersten Befehle: “Forward” und “Backward”. Der Roboter bewegt sich nun. Davon ausgehend ist der nächste Befehl das Anhalten, also “Stop”. Hierauf beschränken sich aber schon die Möglichkeiten, die man an dieser Stelle hat um Befehle zu definieren. Eine Drehung oder ein Abbiegen führt hier schon zu Problemen.

Doch zunächst noch zu grundsätzlichen Befehlen die abhängig von aktuellen Zustand des Roboters sind. Einer weitere Möglichkeit besteht darin die Bewegung des Roboters zu beschleunigen oder zu verlangsamen. Hieraus ergeben sich die Befehle “Faster” und “‘Slower”.

Richtungsveränderungen

Wie bereits erwähnt ergeben sich bei Richtungsveränderungen schon die ersten Probleme. Es stellt sich die Schwierigkeit, wie man einen Richtungswechsel definiert. Soll sich der Roboter um 90 Grad drehen oder eine bestimmte, gesprochene Gradzahl. Auch stellt sich die Frage wie der Roboter sich fortbewegt. Fährt dieser auf Ketten so dreht er sich auf der Stelle, das ist relativ unproblematisch. Fährt er jedoch auf Rändern, oder läuft gar auf Beinen muss die Richtungsveränderung detaillierter beschrieben werden.

Geste und Sprache in Kombination

Bei dem zuvor beschriebenen Problem bietet es sich an Sprache und Geste zu kombinieren. Während es durch Sprache schwierig ist, den Winkel in welchem die Rich-

tungsänderung stattfinden soll zu benennen, so wird dies erheblich vereinfacht durch eine weisende Geste, wie in 7.2.3 beschrieben.

Je nach Art des Roboters kann es auch noch weitere sinnvolle Kombinationen geben. Da dies an dieser Stelle jedoch zu weit in den Bereich der Spekulation geht, wird sich dieser Problematik im zweiten Teil dieser Arbeit gewidmet.

9 Implementierung

9.1 Architektur

Nachfolgend wird der derzeitige Stand der Architektur der Anwendung beschrieben. Da dies lediglich der erste Teil einer größeren Arbeit ist, können infolge des zweiten Teils der Arbeit Veränderungen an der Architektur auftreten.

Wie in Abschnitt 3.2.1 bereits angesprochen, wird die Anwendung unter Verwendung von OSGi [1] umgesetzt. Als Implementierung des OSGi-Standard wurde im Rahmen dieser Arbeit das Framework Equinox [2] gewählt. Die Eclipse IDE und die von jnect verwendeten Frameworks basieren ebenfalls auf dieser Plattform, sie ist daher optimal geeignet. Durch Implementierung der Gesten, Sprachbefehle und Aktionen durch OSGi-Bundles [3] ist es möglich zur Laufzeit der Anwendung neue Gesten nachzuladen oder Aktionen neu zu definieren. Geschaffen wird diese lose Kopplung der Komponenten durch die Nutzung der Service-Funktionalität der OSGi-Plattform.

In Abbildung 9.1 ist die aktuelle Architektur der Anwendung zu sehen. In Abschnitt 9.2 werden die dargestellten Komponenten näher beschrieben. Die Komponenten stellen ihre Funktionalität in Form von Services [3] bereit. Ein Service wird mittels eines Interface am Framework registriert. Somit können andere Bundles einen Service nutzen, ohne dessen konkrete Implementierung kennen zu müssen.

[1] Osgi Alliance (2013) *OSGi Alliance* osgi.org, Abgerufen Januar 07, 2013
[2] The Eclipse Foundation (2013) *Eclipse Equinox* eclipse.org, Abgerufen Januar 07, 2013
[3] Osgi Alliance (2013) *OSGi Technology* osgi.org, Abgerufen Januar 07, 2013

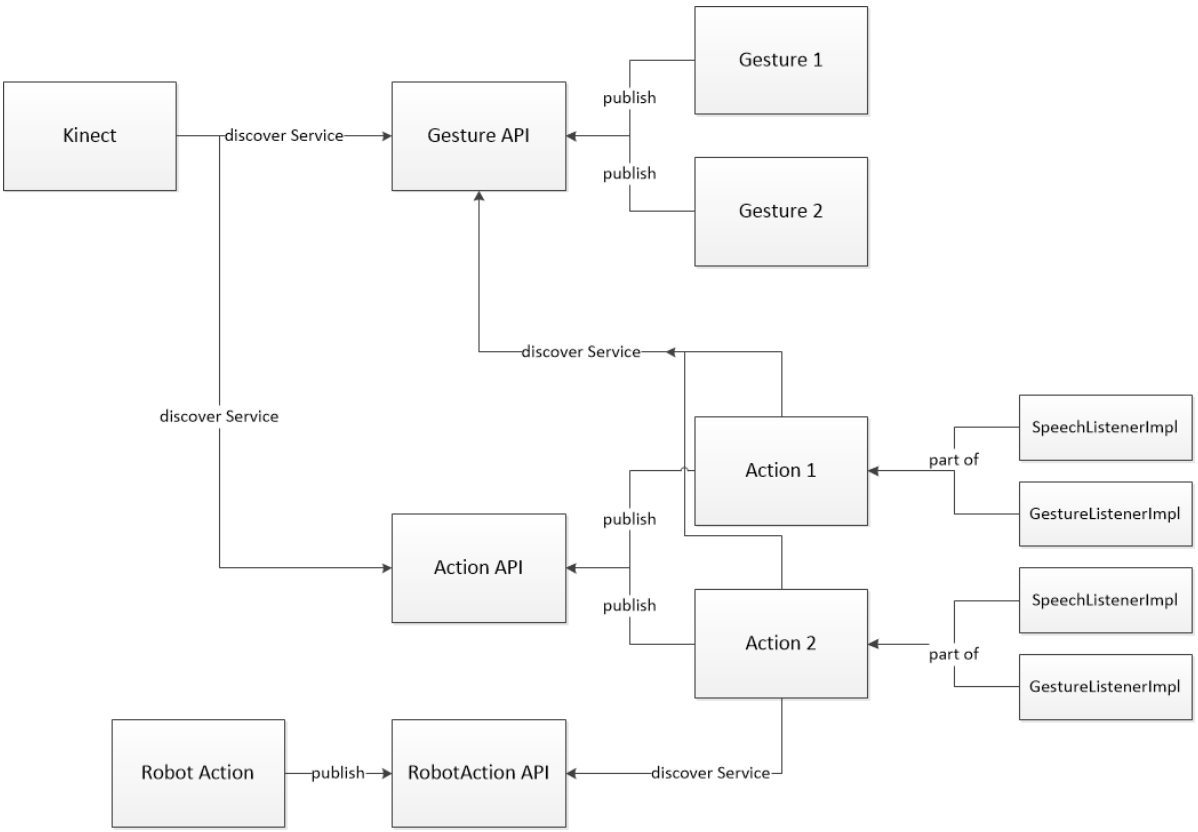


Abbildung 9.1: Architektur der Anwendung

9.2 Bundles

Bundles stellen ihre Funktionalität mittels Services zur Verfügung. Dies geschieht mittels eines Interface. Das Interface wird in einem eigenen Bundle definiert. Hierdurch entsteht eine statische Abhängigkeit zwischen einem API-Bundle und seinen implementierenden Bundles. Dieser wird geschaffen mittels der Deklaration von importierten und exportierten Bundles im Manifest des Bundles (Abbildung 9.1). Die OSGi-Runtime liest beim Start eines Bundles das Manifest ein und löst die Abhängigkeiten auf. Ist dies nicht möglich, kann ein Bundle nicht gestartet werden. Ebenso können laufende Bundles wieder beendet werden. Hierfür existiert das Konzept des Bundle-Lifecycle [3].

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Rocovomo Gesture API
Bundle-SymbolicName: de.rocovomo.jnect.gesture
Bundle-Version: 0.0.1.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: org.eclipse.emf.common,
 org.eclipse.emf.ecore,
 org.jnect.gesture
Export-Package: de.rocovomo.jnect.gesture.api,
 de.rocovomo.jnect.gesture.provider.api
```

Listing 9.1: Manifest der Gesture API

9.2.1 Kinect

Das Kinect-Bundle hat Abhängigkeiten zu den jnect-Bundles. In dieser Komponente wird die Anbindung zur Kinect implementiert. Dieses Bundle sucht nach Services, welche durch die Gesture- und Action-Bundles zur Verfügung gestellt werden. Es registriert Gesten, sowie die Gesten-Listener und Sprach-Listener der Action-Bundles an der Kinect. Durch die Listener erhalten die Action-Bundles die Information ob sie ausgelöst wurden. Das Bundle stellt die effiziente Verwaltung von Gesten und Listenern sicher, sowie die Verwaltung der angeschlossenen Kinect. Ohne das Kinect for Windows kann dieses Bundle nicht gestartet werden.

9.2.2 Gesture

In den Gesture-Bundles werden Gesten implementiert. Finden Veränderungen im Modell statt, so wird dies den Gesture-Bundles mitgeteilt und diese analysieren mittels

HMMs ob sie ausgelöst wurden. Gesten stellen sich als Service zur Verfügung. Sie werden vom Kinect-Bundle zur Erkennung verwendet. Auch die Action-Bundles müssen ihre entsprechenden Gesten kennen. Gesten werden als unabhängige Bundles von Aktionen realisiert. Hierdurch wird eine Codeduplikation bei Aktionen die gleiche Gesten verwenden vermieden.

Die Implementierung einer Geste muss von der jnect-Klasse *Gesture* (Listing 9.2) erben. In der Methode *isGestureDetected(...)* wir das HMM zur Erkennung der Geste implementiert. Ist die Gesten am Gesten-Proxy der Kinect angemeldet, so wird der *GestureListener*, welcher diese Geste nutzt aufgerufen.

```
1  public abstract class Gesture extends EContentAdapter {
2
3    private GestureProxyCallback gestureProxy;
4
5    /**
6     * DO NOT CALL THIS METHOD, IT WILL BE CALLED BY THE GESTUREPROXY
7     *
8     * @param gestureProxy
9     *            the proxy to notify when a gesture is detected
10    */
11   public void setGestureProxy(GestureProxyCallback gestureProxy) {
12     this.gestureProxy = gestureProxy;
13   }
14
15   @Override
16   public void notifyChanged(Notification notification) {
17     if (gestureProxy != null && isGestureDetected(notification)) {
18       this.gestureProxy.notifyGestureDetected(this.getClass());
19     }
20   }
21
22   /**
23    * checks whether the searched gesture is detected
24    *
25    * @param notification
```

```
26      *              the notification containing the model changes
27      * @return true if the gesture was detected
28      */
29     protected abstract boolean isGestureDetected(Notification notification);
30 }
```

Listing 9.2: Klasse Gesture

9.2.3 Action

Die Action-Bundles bilden einen Befehl ab. Dieser Befehl kann entweder eine Geste, ein Sprachbefehl oder eine Kombination aus beidem sein. Je nachdem verfügt die Action über einen *Speech-* oder einen *GestureListener*. Eine Kombination aus beidem ist hier auch möglich. Die Integration dieser beiden Eingabeoptionen wird von Buxton [BBG+11] [4] in Form von Frames besprochen. Eine Aktion ist abhängig von einer Geste die von einem Gesture-Bundle bereitgestellt wird.

Wird eine Geste erkannt wird die Methode *notifyGestureDectected(...)* der entsprechenden Implementierung der abstrakten Klasse *GestureListener* (Listing 9.3) aufgerufen. Von hier kann dann die entsprechende Roboter-Aktion aufgerufen werden.

Das Vorgehen ist beim *SpeechListener* (9.4) prinzipiell ähnlich. Wird ein Sprach-String der Implementierung des *SpeechListener* erkannt, so wird die entsprechende Methode, *notifySpeech(...)* aufgerufen. Im Gegensatz zur Geste ist die Sprache weniger komplex und wird nur mittels Strings definiert. Diese erhält der *KinectManager* mittels *getWords(...)*.

Bislang stellt sich das Action-Bundle selbst als Service zur Verfügung. Das Kinect-Bundle erhält so Zugriff auf die *Speech-* und *GestureListener*.

[4] Kapitel 14 Multimodal Integration

```java
public abstract class GestureListener {

  /**
   * callback method, that gets called when a {@link Gesture} is detected
   *
   * @param gesture
   *            - the class of the {@link Gesture} that was detected
   */
  public abstract void notifyGestureDetected(Class<? extends Gesture>
      gesture);

  /**
   * {@link Set} of {@link Gesture}s this {@link GestureListener} listens
       to
   *
   * @return {@link Set} of {@link Gesture}s to be notified about, can be
   *         empty but not null
   */
  public Set<Gesture> getGestures() {
    return Collections.emptySet();
  }

  /**
   * Whether the {@link GestureListener} listens only to special
   * {@link Gesture}s.
   *
   * @return true if only the {@link Gesture}s provided in
   *         {@link #getGestures()} should be provided to the listener,
       false
   *         otherwise
   */
  public boolean isFiltered() {
    return false;
  }
```

Listing 9.3: Klasse GestureListener

```
1  public abstract class SpeechListener {
2    /**
3     * callback when a relevant speech is recognized
4     *
5     * @param speech
6     *            - the speech recognized
7     */
8    public abstract void notifySpeech(String speech);
9
10   /**
11    * a set of words that this {@link SpeechListener} wants to be
          recognized
12    * and notified about
13    *
14    * @return a {@link Set} of {@link String}s building the relevant words
15    */
16   public abstract Set<String> getWords();
17
18   /**
19    * if a {@link SpeechListener} is filtered than it will be only notified
20    * about recognized words that are contained in the {@link Set} provided
          by
21    * {@link #getWords()}
22    *
23    * @return true if should be filtered, false otherwise
24    */
25   public boolean isFiltered() {
26     return true;
27   }
28 }
```

Listing 9.4: Klasse SpeechListener

9.2.4 Robot Action

Bundles, welche die Robot Action implementieren, stellen sich ebenfalls als Services zur Verfügung. Action-Bundles stellen auf dieser Basis eine Verbindung zwischen Geste/Sprachbefehl und Roboter-Aktion her. Grundsätzlich könnte man dies auch umgekehrt konstruieren, dass sich die Roboter-Aktion ein passendes Action-Bundle sucht. Da Roboter-Interaktion Teil des zweiten Teils der Arbeit ist, ist dieses Bundle bisher noch ohne jede Funktionalität.

9.3 User Interface

Als User-Interface ist eine Anwendung auf Basis der RCP von Eclipse vorgesehen. Zur Visualisierung der der Eingaben des Nutzers war eine dreidimensionale Umgebung mit Hilfe der LWJGL vorgesehen. Optisch orientiert sich eine Anwendung auf dieser Plattform am Aussehen der Eclipse IDE, zu sehen in Abbildung 9.2. Bislang kann das Skeleton-Modell in einem GEF-Editor dargestellt werden, dies wurde in Abbildung 4.4 bereits gezeigt. Erkannte Gesten und Sprachbefehle werden bislang auf einer Logging-Konsole ausgegeben.

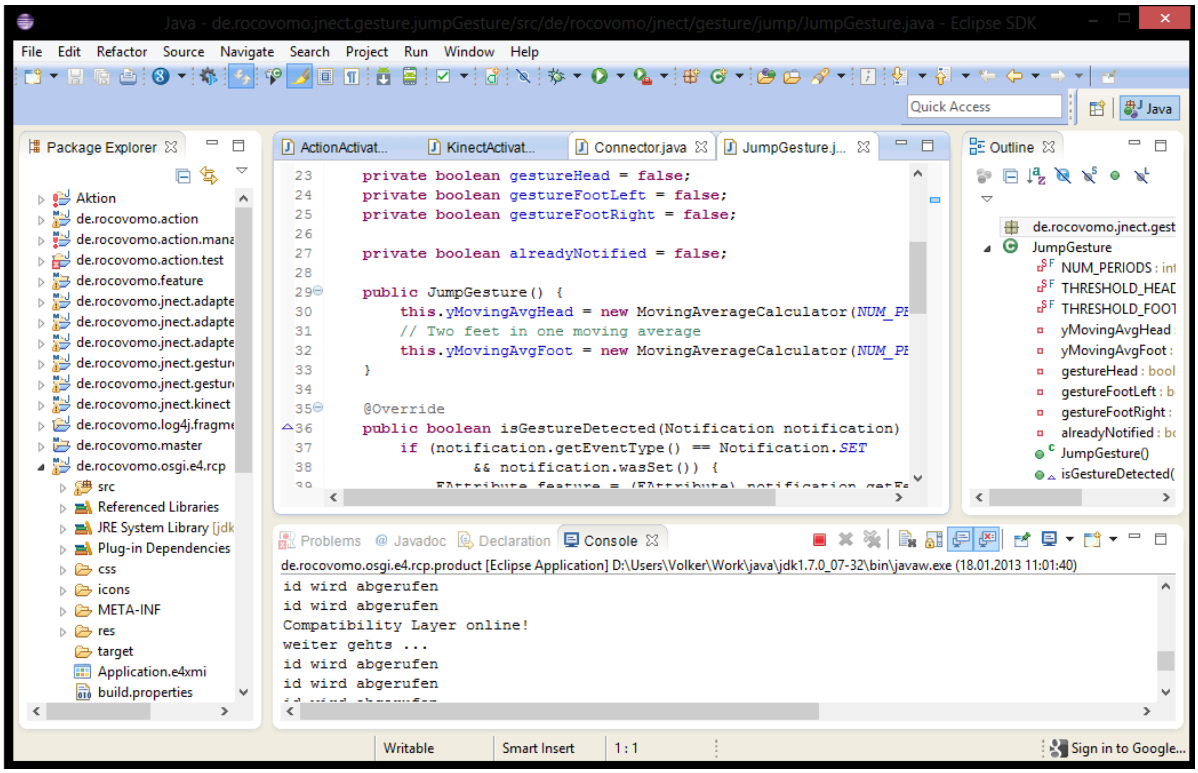


Abbildung 9.2: Eclipse IDE
b

10 Ausblick - weitere Arbeiten

Mit dieser Arbeit wurden die nöetigen Grundlagen geschaffen auf deren Basis in der nachfolgenden Arbeit die Ansteuerung eines mobilen Roboters erfolgen kann. Durch den Einsatz der in Kapitel 6 erarbeiteten Modelle wurde die Möglichkeit geschaffen Gesten auf Grund der Eingangsdaten der Kinect zu ermitteln. Die Hidden Markov Modells bilden auch die Grundlage der Erkennung von Worten aus dem Audiostrom.

Welche Bedeutung der Einsatz von Gesten und Sprachen für ein User Interface hat wurden in den Kapiteln 7 und 8 besprochen, sowie auch erste Entwürfe für Befehle, mit denen man einen Roboter steuern kann. Festgestellt wurde, dass sich nicht jede Geste und jeder Sprachbefehl für eine Schnittstelle zwischen Mensch und Maschine eignen und das vor allem auch die Natürlichkeit für den Anwender wichtig ist.

Auf Basis des in Kapitel 4 gewählten Frameworks zur Ansteuerung der Kinect, jnect, wurde eine, der in Kapitel 5 vorgestellten Konzeption folgende, Anwendung zur Erfassung von Gesten und Sprachbefehlen entwickelt. Diese bietet eine Schnittstelle zur leichten Integration der, in der zweiten Arbeit folgenden, Roboterschnittstelle.

Im zweiten Teil der Arbeit werden die bisher erdachten Gesten und Sprachbefehle noch weiter spezifiziert und auch mit Hilfe eines Roboters umgesetzt werden. In diesem Rahmen wird auch die Anwendung um die Roboterschnittstelle erweitert, sowie die bestehende Implementierung nochmals betrachtet und überarbeitet. Zudem werden die notwendigen Konzepte zur Interaktion zwischen Mensch und Roboter, sowie die Grundlagen zur Steuerung des gewählten Roboters beleuchtet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[AA84] D Archer and R. M. Akert. Problems of context and criterion in nonverbal communication: A new look at the accuracy issue. In *Issues in person perception*, pages 114–144, Methuen, New york, 1984.

[Bak76] R. Bakis. Continuous speech recognition via centisecond acoustic states. *The Journal of the Acoustical Society of America*, 59(S1):S97–S97, 1976.

[BBG+11] W. Buxton, M. Billinghurst, Y. Guiard, A. Sellen, and S. Zhai. Human input to computer systems: theories, techniques and technology, 2011.

[BBL93] T. Baudel and M. Beaudouin-Lafon. Charade: remote control of objects using free-hand gestures. *Communications of the ACM*, 36(7):28–35, 1993.

[BE67] L.E. Baum and J.A. Egon. An inequality with applications to statistical estimation for probabilistic functions of a markov process and to a model for ecology. *Bull. Amer. Meteorology Soc.*, vol. 73:360–363, 1967.

[BE90] Russel Beale and Alistair D. N. Edwards. Gestures and neural networks in human-computer interaction. *IEE Colloquium on Neural nets in human-computer interaction*, pages 5/1–5/4, 1990.

[BP66] E. Leonard Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, Vol. 37, No. 6:1554–1563, 1966.

[BPSW70] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[BS68] Leonard E. Baum and George R. Sell. Growth transformations for functions on manifolds. *Source: Pacific J. Math.*, vol. 27(2):211–227, 1968.

[Cad94] C Cadoz. Le geste, canal de communication homme/machine: la communication instrumentale. In *Technique et Science de l'Information*, volume 13, pages 31–61, 1994.

[Coh92] P. R. Cohen. The role of natural language in a multimodal interface. In *7th Annual ACM Symposium on User Interface Technology*, 1992.

[Cor12] Microsoft Corp., editor. *HUMAN INTERFACE GUIDELINES*, volume v1.5.0. Microsoft Corp., 2012.

[CT89] M.; Pereira F.C.N.; Sullivan J.W.; Gargan Jr. R.A.; Schlossberg J.L. Cohen, P. R.; Dalrymple and S.W. Tyler. Synergistic use of direct manipulation and natural language. In *Conference on Human Factors in Computing Systems*, pages 227–233, Austin, Texas, 1989.

[FB93] Janet Finlay and Russell Beale. Neural networks and pattern recognition in human-computer interaction. *SIGCHI Bull.*, 25(2):25–35, April 1993.

[FJ73] G.D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[HM93] Alexander G. Hauptmann and Paul McAvinney. Gestures with speech for graphic manipulation. *Int. J. Man-Mach. Stud.*, 38(2):231–249, February 1993.

[Jel76] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[KH90] G. Kurtenbach and E. A. Hulteen. Gestures in human-computer communication. In *The Art of Human-Computer Interface Design*. Addison-Wesley Publishing Co., May 1990.

[KM12] Matthew Kober, Jens; Glisson and Michael Mistry. Playing catch and juggling with a humanoid robot. In *12th IEEE-RAS International Conference on Humanoid Robots*, Osaka, Japan, 2012.

[LK99] Hyeon-Kyu Lee and Jin H. Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(10):961–973, 1999.

[Mul96] Axel Mulder. Hand gestures for hci, research on human movement behaviour reviewed in the context of hand centred input. In *Hand Centered Studies of Human Movement Project*. School of Kinesiology, Simon Fraser University, 1996.

[NO96] T. Nishimura and R. Oka. Spotting recognition of human gestures from time-varying images. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 318–322, oct 1996.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Vol. 77, No. 2:257–286, 1989.

[RS91] B. Rimé and L. Schiaratura. *Gesture and speech.* New York, NY, US: Cambridge University Press; Paris, France: Editions de la Maison des Sciences de l'Homme, 1991.

[VB93] K. Vaananen and K. Bohm. Gesture driven interaction as a human factor in virtual environments-an approach with neural networks. *Virtual reality systems*, pages 93–106, 1993.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260 –269, april 1967.

[Wel03] Lloyd R. Welch. Hidden markov models and the baum–welch algorithm. *IEEE Information Theory Society Newsletter*, Dec. 2003.

[ZS09] Chun Zhu and Weihua Sheng. Online hand gesture recognition using neural network based segmentation. *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 1:2415–2420, 2009.

Abkürzungen

DTW Dynamic time warping. 28

EMF Eclipse Modeling Framework. 19

GEF Graphical Editing Framework. 19

GUI Graphical User Interface. 18

HMM Hidden Markov Model. 20, 21, 28–32, 34–36, 42, 48, 51–55

IDE Integrated development environment. 17, 18, 25

LWJGL Lightweight Java Game Library. 19

RCP Rich client platform. 18

Glossar

Bewegungsdetektion

Unter Bewegungsdetektion versteht man Methoden des Maschinellen Sehens, die auf die Erkennung von Fremdbewegung im Erfassungsbereich eines optischen Detektors (also dem Blickfeld der Maschine) abzielen. 3, 8, 9

Datenhandschuh

Der Datenhandschuh ist ein Eingabegerät in Form eines Handschuhs. Durch Bewegungen der Hand und Finger erfolgt eine Orientierung im virtuellen Raum. 7, 8

Deixis

Deixis, auch indexikalische Semantik, ist ein Fachbegriff aus der Sprachwissenschaft. Er bezeichnet die Bezugnahme auf Personen, Orte und Zeiten im Kontext, die mit Hilfe von deiktischen oder indexikalischen Ausdrücken wie *ich, du, dort, hier, morgen, heute ...* erfolgt. 56, 61

Dynamische Programmierung

Dynamische Programmierung ist eine Methode zum algorithmischen Lösen von Optimierungsproblemen. Der Begriff wurde in den 1940er Jahren von dem amerikanischen Mathematiker Richard Bellman eingeführt, der diese Methode auf dem

Gebiet der Regelungstheorie anwandte. In diesem Zusammenhang wird auch oft von Bellmans Prinzip der dynamischen Programmierung gesprochen. 38, 39, 51

Eclipse

Eclipse (von englisch eclipse ‚Sonnenfinsternis', ‚Finsternis', ‚Verdunkelung') ist ein quelloffenes Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Ursprünglich wurde Eclipse als integrierte Entwicklungsumgebung (IDE) für die Programmiersprache Java genutzt, aber mittlerweile wird es wegen seiner Erweiterbarkeit auch für viele andere Entwicklungsaufgaben eingesetzt. Für Eclipse gibt es eine Vielzahl sowohl quelloffener als auch kommerzieller Erweiterungen. 17, 35

Epistemik

Die epistemische Logik, auch Wissenslogik befasst sich mit Glauben und Wissen bei Individuen sowie Gruppen. Ziel von Untersuchungen mittels epistemischer Logik ist oft ein dynamisches oder flexibles Modell von Meinungs- und Wissenszuständen. 56

Equinox

Equinox (von englisch Tag- und Nachtgleiche) ist ein von der Eclipse Foundation entwickeltes Java-basiertes Framework, welches die OSGi-Kernspezifikation implementiert und das Gerüst der integrierten Entwicklungsumgebung Eclipse bildet. 18, 35

Ergodizität

Ergodizität ist eine Eigenschaft dynamischer Systeme. Sie bezieht sich auf das mittlere Verhalten eines Systems. Ein solches System wird durch eine Musterfunktion beschrieben, die die zeitliche Entwicklung des Systems abhängig von seinem aktuellen Zustand bestimmt. 52, 55

Framework

Ein Framework (englisch für Rahmenstruktur) ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird. 18, 35

Ikonizität

Ikonizität ist ein mehrdeutiger linguistischer Fachbegriff, der sich auf den Begriff des Ikons im Sinne von Charles Sanders Peirce bezieht. Er bezeichnet unter anderem, die Beziehung zwischen dem Ausdruck und Inhalt ikonischer Zeichen. Ikonizität wird oftmals mit Abbildungsverhältnis sprachlicher Ausdrücke übersetzt. 56, 58

Künstliche neuronale Netze

Künstliche neuronale Netze basieren meist auf der Vernetzung vieler McCulloch-Pitts-Neuronen oder leichter Abwandlungen davon. Grundsätzlich können auch andere künstliche Neuronen Anwendung in KNNen finden, z.B. das High-Order-Neuron. Die Topologie eines Netzes (die Zuordnung von Verbindungen zu Knoten) muss abhängig von seiner Aufgabe gut durchdacht sein. Nach der Konstruktion eines Netzes folgt die Trainingsphase, in der das Netz „lernt“. 39

Künstliches neuronales Netz

Künstliche neuronale Netze (selten auch künstliche neuronale Netzwerke, kurz: KNN, engl. artificial neural network – ANN) sind Netze aus künstlichen Neuronen. Sie sind ein Zweig der künstlichen Intelligenz und prinzipieller Forschungsgegenstand der Neuroinformatik. 20, 38

Kinect

Kinect ist eine Plattform zur Steuerung der Videospielkonsole Xbox 360, die seit Anfang November 2010 verkauft wird. 8, 10

Mensch-Computer-Interaktion

Die Mensch-Computer-Interaktion (englisch *Human-Computer Interaction, HCI*) als Teilgebiet der Informatik beschäftigt sich mit der benutzergerechten Gestaltung von interaktiven Systemen und ihren Mensch-Maschine-Schnittstellen. 7, 56, 57

Mensch-Maschine-Schnittstelle

Die Benutzerschnittstelle (nach Gesellschaft für Informatik, Fachbereich Mensch-Computer-Interaktion auch Benutzungsschnittstelle) ist die Stelle oder Handlung, mit der ein Mensch mit einer Maschine in Kontakt tritt. 7, 20

Mikrofonarray

Beamforming ist ein Verfahren zur Positionsbestimmung von Quellen in Wellenfeldern (z. B. Schallfeldern). Entsprechende Vorrichtungen werden auch akustische Kamera, Mikrofonarray oder akustische Antenne genannt. 14, 15

Open Graphics Library

OpenGL (Open Graphics Library) ist eine Spezifikation für eine plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafik. Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben. 19

OpenNI

OpenNI oder Open Natural Interaction ist eine von der Industrie geführte gemeinnützige Organisation deren Ziel die Zertifizierung und Verbesserung von natürlichen

und organischen Benutzeroberflächen , zugehörigen Geräten und deren Anwendugen, sowie auch Anwendungen zum Zugriff auf solche Geräte, ist. 27

Pantomime

Pantomime bezeichnet sowohl eine Form der darstellenden Kunst, deren Darsteller in den meisten Fällen ohne gesprochenes Wort auskommen, als auch den Künstler selbst, der diese Form der Darstellung praktiziert. 56, 65, 67

Programmierschnittstelle

Eine Programmierschnittstelle (englisch application programming interface, API; deutsch Schnittstelle zur Anwendungsprogrammierung) ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird, plural=API. 18, 23

Runtime Library

Der Begriff Laufzeitbibliothek (engl. runtime library) wird in der Informatik verwendet. Er bezeichnet eine spezielle Programmbibliothek, eine Sammlung von Softwarefunktionen, die benutzt wird, um innerhalb eines Computerprogramms die in eine Programmiersprache eingebauten Funktionen zur Zeit der Ausführung des Programms (Laufzeit) zu realisieren. Dazu gehören oft z. B. Funktionen zur Ein- und Ausgabe, Speicherverwaltung oder mathematische Funktionen. 22

Semiotik

Semiotik ist die Wissenschaft, die sich mit Zeichensystemen aller Art (zum Beispiel: Bilderschrift, Gestik, Formeln, Sprache, Verkehrszeichen) befasst. Sie ist die allgemeine Theorie vom Wesen, der Entstehung (Semiose) und dem Gebrauch von Zeichen. 55, 57

Skelett Tracking System

Unter Motion Capture, oder auch Skelett Tracking System, wörtlich Bewegungs-Erfassung, versteht man ein Tracking-Verfahren, das es ermöglicht, jede Art von Bewegungen so zu erfassen und in ein von Computern lesbares Format umzuwandeln, dass diese die Bewegungen analysieren, aufzeichnen, weiterverarbeiten und zur Steuerung von Anwendungen verwenden können. 15

Software Development Kit

Ein Software Development Kit (SDK) ist eine Sammlung von Werkzeugen und Anwendungen, um eine Software zu erstellen, meist inklusive Dokumentation. Mit diesem ist es Softwareentwicklern möglich, eigene darauf basierende Anwendungen zu erstellen. 10, 12, 16, 20, 34

Stochastischer Prozess

Ein stochastischer Prozess ist die mathematische Beschreibung von zeitlich geordneten, zufälligen Vorgängen. 41, 42, 44

Wrapper

Als Wrapper oder Adapter bezeichnet man in der Informationstechnik ein Stück Software, welches ein anderes Stück Software umgibt. Dies kann sich sowohl auf ganze Programme, als auch nur auf einzelne Programmteile- bis Klassen beziehen. Die “Umüllung” kann sowohl visueller als auch technischer Natur sein. 27