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

Inhaltsverzeichnis

1 Einleitung

Gestik im Sinne von kommunikativen Bewegungen stellt einen wesentlichen Teil der nonverbalen Kommunikation dar. Die Definition nach Kurtenbach und Hulteen (1990) 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 elektornisches 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 vorranging 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 wid 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

3 Stand der Technik

3.1 Kinect

Kinect ist eine Gerät zur Dedektion 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. Novmeber 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 Forschungsgegenschaft 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

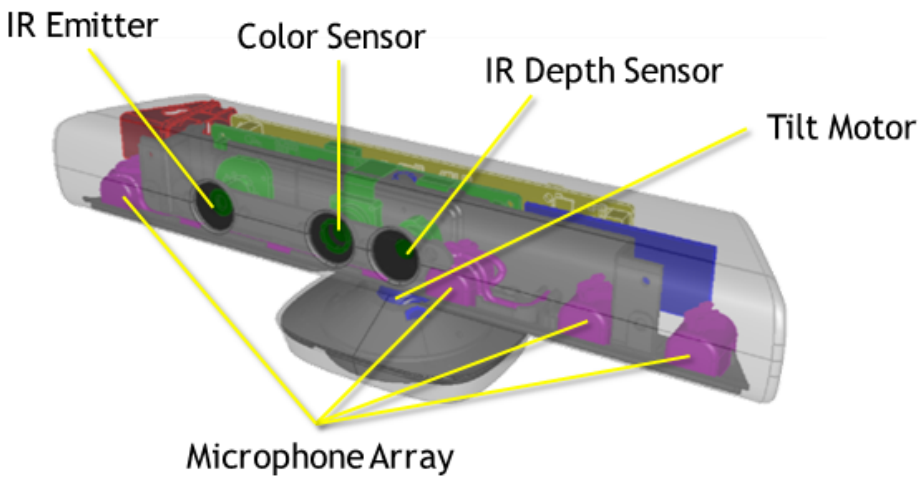

Abbildung 3.1: Schematische Ansicht der Sensoren einer *Kinect for Windows* (Quelle: Microsoft Corp.[1])

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 Pogramme 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 Pogrammiersprache 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 Andwendung 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 9.1.3.

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

asdf

4.2 Java Frameworks

adsf

4.2.1 OpenNI

Beschreibung

adsf

Technische Daten

SWOT–Analyse

4.2.2 OpenKinect

Beschreibung

Technische Daten

SWOT–Analyse

4.2.3 jnect

Beschreibung

Technische Daten

SWOT–Analyse

4.2.4 Scoring–Modell

4.2.5 Analyseergebnis

adsasfd

5 Konzeption

5.1 Technische Vorgaben

5.1.1 Vorgaben durch Kinect

5.1.2 Vorgaben durch jnect–Framework

5.1.3 Service–Architektur

5.1.4 Abhängigkeit zu Robotertechnik

5.2 Fachliche Vorgaben

5.2.1 Vorgaben durch Mensch-Computer-Interaktion

5.2.2 Klassifikation der Anwendung

5.2.3 Analyse der Gesten- und Sprachsteuerung

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 Bespiel 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 From 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

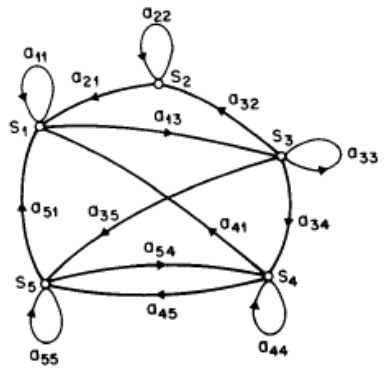

Abbildung 6.1: Markov-Kette mit fünf Zuständen (S_1 bis S_5) mit Verbindungen (Quelle: [Rab89])

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 beantwortert 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 aufeinanderfolgender 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 darunterliegenden, 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 Paramter 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:

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

[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-

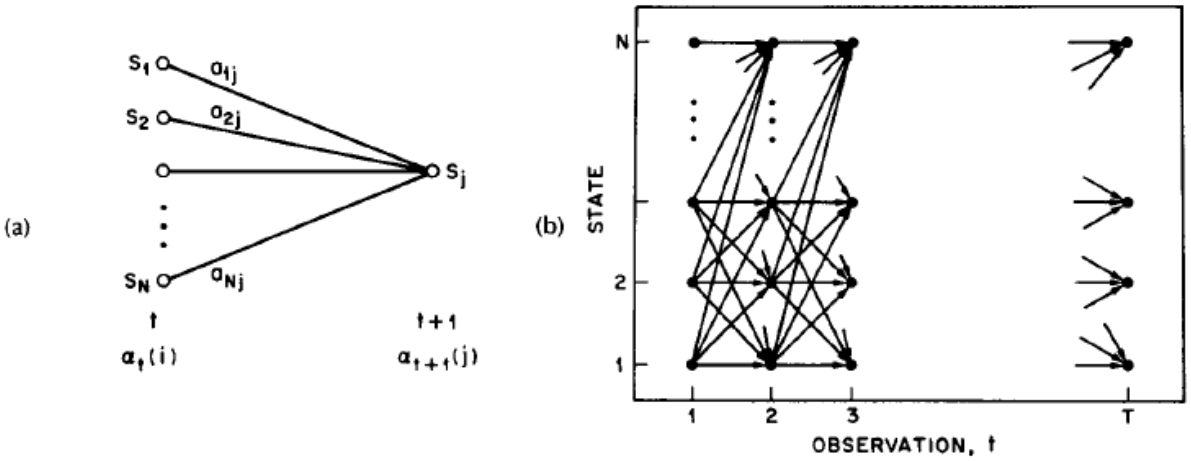

Abbildung 6.2: (a) Illustration der Oerpationsfolge zur Berechnung der Vorwärtsvariable $\alpha_{t+1}(j)$. (b) Implementierung der Berechung von $\alpha_t(i)$ im Netz der Beobachtungen t und Zustände i. (Quelle: [Rab89])

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 Berechung von $\beta_t(i)$ kann ebenfalls in einer Gitterstruktur, ähnlich der in Abbildung 6.2(b) durchgeführt werden.

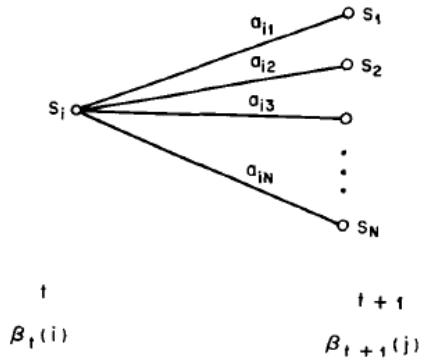

Abbildung 6.3: Illustration der Oerpationsfolge zur Berechnung der Rückwärtsvariable $\beta_t(i)$. (Quelle: [Rab89])

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 Zeitpunt 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 Literaturreferenz 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 Beoboachtungssequenz eines gebenen 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 [**?**, 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 ergodische Modell. Eines dieser Modelle ist das aus Abbildung 6.4(b). Dieses Modell ist ein

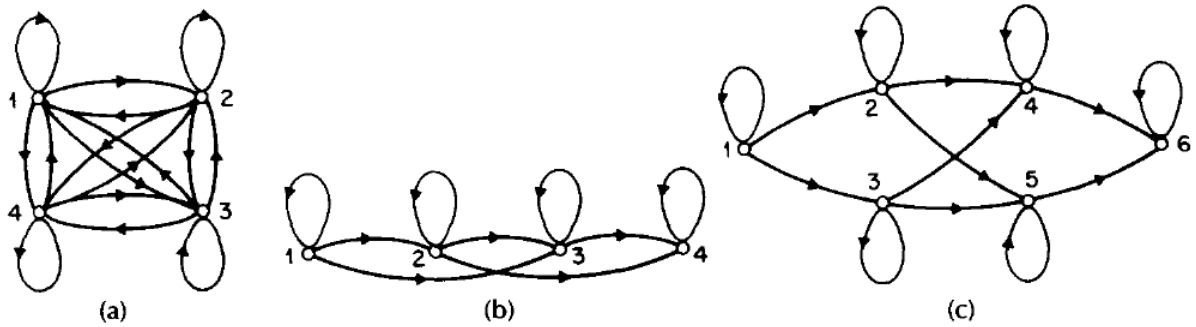

Abbildung 6.4: Darstellung dreier verschiedener Typen eines HMMs. (a) 4-Zustand ergodisches Modell. (b) 4-Zustand links-rechts Modell. (c) 6-Zustand Parallelpfad links-rechts Modell. (Quelle: [Rab89])

sogenanntes links-rechts Model oder Bakis Model [Jel76] [**?**], da die dahinterliegende 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

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

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

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

7.1 Kreisbewegung

7.1.1 ToDo

7.2 Vorwärtsbewegung

7.2.1 ToDo

7.3 Haltesignal

7.3.1 ToDo

8 Sprachbefehle

8.1 ToDo

9 Implementierung

9.1 OSGI–Bundles

9.1.1 Gesten

9.1.2 Sprachbefehle

9.1.3 Roboterschnittstelle

9.1.4 Ausführbare Aktionen

9.2 Oberfläche

9.2.1 RCP–Anwendung

9.2.2 Grafische Darstellung

9.3 Ablaufbeschreibung

10 Ausblick - weitere Arbeiten

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[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. 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. vol. 27(2):211–227, 1968.

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

[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.

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

[KM12] Matthew Kober, Jens; Glisson and Michael Mistry. Playing catch and juggling with a humanoid robot. 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.

[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.

[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*, 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.

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. 8

Datenhandschuh

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

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. 16

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. 17

Kinect

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

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

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

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. 13, 14

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. 18

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. 17

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. 14

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. 11, 15