Karlsruhe Institute of Technology

Fraunhofer IOSB

Gestensteuerung mit Microsoft Kinect

Studienarbeit
von

Stefan Bürger

Institut für Anthropomatik
Fraunhofer IOSB

Erstbetreuer:	Prof. Dr.-Ing. J. Beyerer
Betreuende Mitarbeiter:	Dipl.-Inf. Alexander Streicher, Fraunhofer IOSB
	Dipl.-Inf. Anton Berger, Fraunhofer IOSB

Bearbeitungszeit: Mai 2011 – Juli 2011

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 27. Juli 2011

Zusammenfassung

Im Bereich der Mensch-Maschine-Interaktion werden neue Bedienkonzepte erforscht, die dem Menschen eine natürliche und intuitive Interaktion mit Computersystemen ermöglichen. Diese führen im Vergleich zu traditionellen Eingabegeräten, wie Maus, Tastatur und Touchscreen, zu einer dem Menschen eingängigen Bedienung.
Für das 3D-Simulationsprogramm ViSAR für die Radar-Bildauswertung, welches am Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung IOSB entwickelt wird, wurde eine gestenbasierte Interaktion realisiert. Die Gestenerkennung basiert auf dem Kinect for Windows Software Developement Kit (SDK).
Im Rahmen dieser Studienarbeit wurden in einem ersten Schritt bekannte Freiform-Gesten im Hinblick auf ihre Eignung zur Steuerung von ViSAR analysiert. Darauf aufbauend wurden Gesten identifiziert, die sich einerseits sinnvoll für ViSAR verwenden lassen und andererseits eine robuste Erkennungsrate bei der Skelettberechnung des Kinect SDK aufweisen. Die wichtigsten Interaktionsmöglichkeiten für ViSAR wurden implementiert. Die Realisierung ermöglicht eine personeninvariante Steuerung von ViSAR mittels Gesten. Die erzielten Ergebnisse und die generisch gehaltene Implementierung lassen sich auf andere Anwendungsgebiete übertragen.

Abstract

In the field of human-machine interaction new operational concepts are being researched in order to provide people a intuitive interaction with computer systems. In relation to traditional input devices like mouse, keyboard and touchscreen this enables the operator to use natural movements.
For the 3D simulation program ViSAR which is used for radar image interpretation and is being developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, a gesture-based interaction has been realized. The gesture recognition is based on the Kinect for Windows Software Development Kit (SDK).
This work analyzes well-known free-form gestures regarding their suitability to control ViSAR. On this basis, gestures have been identified that on the one hand make sense for their usage with ViSAR and on the other hand show a robust detection rate using the skeleton calculation of the SDK. The most important possibilities to interact with ViSAR have been implemented. The implementation offers a people-invariant control of ViSAR using gestures. The results and the generic implementation can be transferred to other application areas.

Inhaltsverzeichnis

1. Einleitung

Auf der Suche nach neuen intuitiven Interaktionsmöglichkeiten werden Bedienkonzepte erforscht, die ohne Maus, Tastatur und Touchscreens auskommen. Eingabegeräte sollen für den Benutzer unsichtbar werden, sodass er auf natürliche Art und Weise mit Computersystemen interagieren kann. Ein Teil dieser Natural-User-Interfaces stellen Gesten-Interfaces dar, die eine Mensch-Maschine-Schnittstelle ermöglichen, die für den Menschen leicht verständlich ist.

Das am Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung IOSB entwickelte 3D-Simulationsprogramm ViSAR dient zur Visualisierung von geometrischen Effekten bei der Radar-Bildauswertung. Hierzu wird von einer 3D Szene ein simuliertes Radarbild erzeugt. Ein wichtiges Anwendungsszenario von ViSAR ist das Erlernen und Erweitern von Fähigkeiten zum Lesen von Radarbildern. Zu diesem Zweck steht ein Lehrer mit einem oder mehreren Schülern vor einem großen Display oder einer Leinwand, auf die das Programm projeziert wird. Die Steuerung von ViSAR basiert auf der Verwendung von Maus und Tastatur. Dies wird dem Anwendungszweck nicht gerecht, da der Lehrer während seiner Erläuterungen immer wieder zu Maus und Tastatur greifen muss. Die Entwicklung eines Gesten-Interface auf Basis der Kinect-Kamera ermöglicht den Verzicht auf Maus und Tastatur, sodass ViSAR mit Bewegungen der Arme bedient werden kann.

1.1 Motivation

Maus und Tastatur sind die am Weitesten verbreiteten Eingabegeräte für Computersysteme. Obwohl sich der Benutzer an sie gewöhnt hat, entsprechen die Bewegungen, die zur Bedienung notwendig sind, keinen natürlichen Bewegungen, die der Mensch aus dem Alltag kennt. So muss ihre Verwendung von Menschen, die wenig Erfahrung mit der Verwendung von Maus und Tastatur haben, nach wie vor mit hohem Aufwand erlernt werden. Dennoch sind die Interaktionsmöglichkeiten, die diese Eingabegeräte bieten, beschränkt. Auf einer Tastatur können ausschließlich Tasten gedrückt werden und eine Maus besitzt neben der Information über gedrückte Tasten lediglich Informationen über die Bewegung der Maus auf einer zweidimensionalen Fläche.

Eine Verbesserung stellen Geräte mit berührungsempfindlicher Oberfläche dar. Auf Touchscreens kann die Steuerung mit Gesten vorgenommen werden, indem der Benutzer eine Bewegung ausführt, während er die Oberfläche berührt. Sie sind im Vergleich zur Verwendung von Maus und Tastatur flexibler einsetzbar, da der Benutzer keine zusätzlichen

Eingabegeräte benötigt. Touchscreens werden seit den 1970er Jahren entwickelt und feierten den endgültigen Durchbruch mit der Einführung des iPhones im Jahr 2007. Neben Smartphones wird diese Technologie zum Beispiel für Touchtables oder bei Ticketautomaten eingesetzt. Wie bei der Verwendung von Maus und Tastatur ist allerdings auch diese Technologie auf eine zweidimensionale Fläche beschränkt, da nur hier Berührungen erkannt werden können.

Auf Grund der Einschränkungen, die bei Maus, Tastatur und Touchscreen zu finden sind, wird im Bereich der Mensch-Maschine-Interaktion (MMI) nach neuen Interaktionsmöglichkeiten geforscht. Der menschliche Körper eignet sich auf Grund seiner hohen Anzahl an Freiheitsgraden perfekt für die Ausführung von Bewegungen, die als Gesten interpretiert werden können. Gesten im dreidimensionalen Raum werden als Freiform-Gesten bezeichnet. Sie sind eine der wichtigsten und ausdrucksstärksten Formen der Kommunikation zwischen Menschen. Für die Steuerung eines Computersystems können Gesten übertragen werden, die bereits aus der zwischenmenschlichen Kommunikation bekannt sind. Dies sorgt für eine intuitive Bedienung dieser Systeme. Es sind keine langen Lern- und Eingewöhnungsphasen nötig. Erste Ansätze zur Erkennung von Freiform-Gesten arbeiteten mit zusätzlicher Hardware, wie Datenhandschuhen, die vom Benutzer getragen werden mussten. Mittlerweile ist dies nicht mehr nötig, da mit Hilfe von Kamerasystemen die menschliche Bewegung wahrgenommen werden kann. Eine dieser Kameras ist die Kinect-Kamera von Microsoft. Zusammen mit dem Kinect for Windows Software Developement Kit (SDK) ergeben sich weitreichende Möglichkeiten zur Erkennung von Freiform-Gesten, die zur Steuerung von Programmen eingesetzt werden können.

1.2 Ziel der Arbeit

Ziel der Arbeit ist der Entwurf einer Gestensteuerung für ViSAR unter Einsatz der Kinect-Kamera und dem zugehörigen Software Developement Kit von Microsoft. Zu diesem Zweck soll eine Reihe von Freiform-Gesten vorgestellt und deren Eignung für die Steuerung von ViSAR analysiert werden. Hierauf aufbauend soll ein Konzept zur Steuerung von ViSAR entworfen und exemplarisch implementiert werden.

1.3 Verwandte Arbeiten

Die Gestenerkennung ist seit vielen Jahren Teil der Forschung. Erste Ansätze zur Gestenerkennung wurden, wie in der Veröffentlichung von Baudel und Beaudouin-Lafon [1] beschrieben, mit Datenhandschuhen durchgeführt. Auf Grund des Fortschritts bei der Weiterentwicklung von visuellen Sensoren und Computer Vision Algorithmen kann, wie bei der Gestenerkennung mit der Kinect-Kamera, auf die Verwendung von Hardware, die vom Benutzer getragen werden muss, verzichtet werden. Voraussetzung für die Erkennung von Gesten ist zunächst die Berechnung der Pose des Körpers oder von Teilen des Körpers auf der Grundlage von Kamerabildern. Agarwal und Triggs [2] zeigen, dass es möglich ist, aus einem monokularen Bild die dreidimensionale Körperpose zu berechnen. Wesentlich weiter verbreitet ist die Verwendung von Tiefenbildern ([3], [4], [5]), die von Stereo- oder Time-of-flight-Kameras stammen. Die Möglichkeit Informationen aus 2D und 3D Bilder zu diesem Zweck zu fusionieren stellen Ghobadi *et al.* [6] vor. Bei der Kinect-Kamera stehen sowohl Tiefen- als auch Farbbilder zur Verfügung. Das SDK verwendet zur Berechnung der Körperpose jedoch lediglich Tiefenbilder.

Einen Überblick über die Gestenerkennung in der Mensch-Maschine Interaktion auf Grundlage von visuellen Sensoren liefert die Arbeit von Pavlovic *et al.* [7]. Zur Erkennung von Gesten werden meist die aus der Spracherkennung bekannten Hidden Markov Modelle (HMMs) [8] eingesetzt. Als Gesten können sowohl Bewegungen des ganzen Körpers als auch

Bewegungen von Teilen des Körpers interpretiert werden. Auf Basis von HMMs beschreiben Starner und Pentland [9] die Erkennung der American Sign Language. Schwerpunkt der Arbeit von Lee [10] ist die Erkennung von Ganzkörpergesten mit Hilfe von HMMs. Dass Bewegungen des Kopfes ebenfalls als Geste eingesetzt werden kann, ist in der Arbeit von Fujie *et al.* [11] nachzulesen. Hierbei wird der optische Fluss des Kopfes und HMMs zur Erkennung der Gesten eingesetzt. Als Alternative zu HMMs veröffentlichten Wang *et al.* [12] eine Gestenerkennung, die mit Hidden Conditional Random Fields arbeitet. In dieser Arbeit werden zur Gestenerkennung Körperposen, die anhand von geometrischen Berechnungen bestimmt werden, und Folgen von Körperposen verwendet.

Gestenerkennung kann zu Unterhaltungszwecken in Spielen eingesetzt werden, wie zum Beispiel bei Bannach *et al.* [13] als Einparkhilfe für ein virtuelles Auto. Bei Schick *et al.* [14] werden Gesten zur Bedienung von großen Displays, wie in Lagezentren beim Krisenmanagement, verwendet. Medizinische Anwendungen profitieren ebenfalls von der Entwicklung von sterilen Steuerungen mit Gesten. In der Arbeit von Johnson *et al.* [15] wird ein System zur sterilen Interaktion mit digitalen Bildern vorgestellt. Bei der Kommunikation mit Robotern nehmen Gesten ebenfalls eine wichtige Rolle ein. Der Einsatz von Zeigegesten in der Mensch-Roboter Interaktion wird in der Arbeit von Nickel und Stiefelhagen [16] beschrieben.

Die vorliegende Studienarbeit beschäftigt sich mit der Entwicklung eines Gesten-Interfaces auf Basis der Kinect-Kamera für das PC-Programm ViSAR, dessen Bedienung auf die Verwendung von Maus und Tastatur optimiert ist. Vor dem Erscheinen des in dieser Arbeit verwendeten Kinect for Windows SDK von Microsoft wurde bereits von PrimeSense das OpenNI [17] Framework veröffentlicht, welches die Nutzung der Kinect-Kamera an einem PC ermöglicht. Die NITE Middleware [18] erkennt Gesten auf Basis von OpenNI. Im Gegensatz zum offiziellen Kinect-SDK benötigt OpenNI eine initiale Kalibrierpose um den Körper einer Person zu tracken. Suma *et al.* [19] ermöglichen mit ihrer FAAST Middlware die Erstellung von Gesten-Interfaces für bestehene Anwendungen, indem Körperposen, die von der Kinect-Kamera erkannt werden, mit virtuellen Maus- und Tastaturkommandos verknüpft werden. FAAST enthält lediglich eine feste Anzahl an einfachen Gesten, die verwendet werden können. Um ViSAR mit intuitiven Gesten steuern zu können, sind Gesten notwendig, die von FAAST nicht erkannt werden.

1.4 Aufbau der Arbeit

In Kapitel 2 werden zunächst die Grundlagen der Gestensteuerung behandelt. Hierzu gehören neben der Definition und Klassifizierung von Gesten die Anforderungen, die an ein Gesten-Interface gestellt werden. Kapitel 2 schließt mit der Vorstellung der wichtigsten Einsatzmöglichkeiten von Gestensteuerungen ab.

Kapitel 3 beschreibt eine Sammlung von Freiform-Gesten, die für die grundlegenden Funktionen eines Programms benötigt werden und sich bereits in der Praxis bei anderen Systemen bewährt haben.

Die Beschreibung der Kinect-Kamera und des Algorithmus zur Körperposenberechnung, der von Microsoft eingesetzt wird, folgt in Kapitel 4.

Kapitel 5 beinhaltet die Beschreibung des Programms ViSAR und die Vorstellung des Konzepts zur Bedienung von ViSAR mit Gesten.

Kapitel 6 ist der Implementierung gewidmet.

Abschließend fasst Kapitel 7 diese Arbeit zusammen und gibt einen Ausblick.

2. Grundlagen der Gestensteuerung

Bei der Entwicklung eines Gesten-Interfaces müssen zunächst die theoretischen Grundlagen betrachtet werden. Hierzu werden Gesten definiert, sowie eine Klassifizierung von Gesten vorgestellt. Für eine qualitativ hochwertige Gestensteuerung müssen wichtige Anforderungen eingehalten werden. Das Kapitel schließt mit einer Betrachtung von Einsatzgebieten von Gesten-Interfaces.

2.1 Mensch-Computer-Interaktion

Ein User Interface erlaubt Nutzern mit einem System zu interagieren. Die Mensch-Computer-Interaktion (eng. Human Computer Interaction, HCI) begann, wie in Abbildung 2.1 zu sehen, mit der Verwendung von Command-Line Interfaces (CLI). Hierbei findet die Interaktion mit Hilfe von geschriebenen Anweisungen statt, die auf einer Tastatur eingetippt werden. Größter Nachteil dieses User Interface ist die Tatsache, dass der Benutzer die Anweisungen kennen muss, um sie einsetzen zu können. Eine Weiterentwicklung stellen die Graphical User Interfaces (GUI) dar. Diese basieren auf einer graphischen Oberfläche, die mit Hilfe eines Zeigegerätes, zum Beispiel einer Maus, bedient werden kann. Der Benutzer muss die Anweisungen nicht kennen, er kann sie mit der Maus aktivieren, indem er auf den entsprechenden Button oder Menüeintrag klickt. Bei Natural User Interfaces (NUI) wird auf die Verwendung von klassischen Eingabegeräten wie Maus und Tastatur verzichtet. Die Interaktion mit einem System findet über natürliche Bewegungen, Gesten und Sprache statt. Dies führt zu einer verständlichen und intuitiven Bedienungsoberfläche, sodass der Benutzer alltägliche Handlungsmuster anwenden kann.

Abbildung 2.1: Die Evolutionsstufen der User Interfaces: Command-Line Interfaces, Graphical User Interfaces und Natural User Interfaces. In Anlehnung an [20]

Die Bedienung eines Systems mit Hilfe von Gesten gehört zum Bereich der Natural User Interfaces. Im Wesentlichen gibt es zwei verschiedene Typen von Gesten-Interfaces. Sie lassen sich unterteilen in Touchscreen-Gesten-Interfaces und Freiform-Gesten-Interfaces [21].

Bei Touchscreens, wie zum Beispiel dem Apple iPhone oder dem Microsoft Surface, findet die Interaktion ausschließlich über Berührungen des Displays statt. Hierdurch ergeben sich Einschränkungen bei den Gesten, die von dem Gerät erkannt werden können, da die berührungsempfindliche Fläche zweidimensional ist und eine beschränkte Größe aufweist. Diesen Nachteil gibt es bei Freiform-Gesten-Interfaces nicht, da sie unabhängig von einer berührungsempfindlichen Fläche arbeiten. Beliebige Körperbewegungen im Raum können als Gesten eingesetzt werden und bieten somit im Vergleich zu Touchscreens erweiterte Möglichkeiten. Zum Steuern des Systems ist keine direkte Berührung notwendig, sodass der Benutzer das System über eine größere Entfernung bedienen kann. Die Anzahl der möglichen Gesten erweitert sich, da jeder Freiheitsgrad des menschlichen Körpers zum Ausführen der Gesten eingesetzt werden kann.

2.2 Definition einer Geste

Gesten lassen sich nicht eindeutig definieren. Je nach Kontext ergeben sich unterschiedliche Schwerpunkte der Definitionen. Bei der Kommunikation zwischen Menschen unterstützen Gesten das gesprochene Wort. Die Gesten sind in diesem Fall ohne Sprache nicht zu verstehen. Bei der Gebärdensprache ersetzen Gesten das gesprochene Wort. Dieser kommunikative Aspekt von Gesten wird in der Definition des Dudens deutlich: Eine Geste ist eine „*spontane oder bewusst eingesetzte Bewegung des Körpers, besonders der Hände und des Kopfes, die jemandes Worte begleiten oder ersetzen.*“ [22]

Im Bereich der Mensch-Maschine-Interaktion werden Gesten allerdings nicht zur Verdeutlichung oder als Ersatz von Sprache eingesetzt. Der Schwerpunkt liegt hierbei auf der Interpretation der Geste als Eingabe für ein Computersystem. Aus diesem Grund liefert Saffer für den Kontext dieser Arbeit eine bessere Definition: „*Eine Geste ist eine körperliche Bewegung, die von einem digitalen System erfasst wird und dieses darauf reagiert ohne die zu Hilfenahme von traditionellen Zeigegeräten wie Maus oder Eingabestift.*“ [21]

2.3 Klassifizierung von Gesten

Gesten lassen sich in unterschiedliche Kategorien einordnen. Pavlovic *et al.* [7] haben hierzu ein auf einer Arbeit von Quek *et al.* [23] basierendes Klassifikationsschema von Arm- und Handbewegungen im Kontext der Mensch-Maschine-Interaktion vorgestellt (Abbildung 2.2):

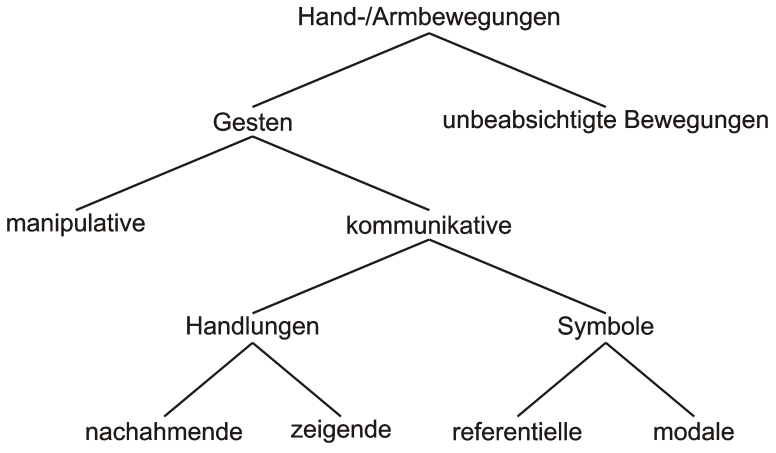


Abbildung 2.2: Das Klassifikationsschema für Gesten von Pavlovic *et al.*. In Anlehnung an [7]

Zunächst werden alle Bewegungen des Körpers in unbeabsichtigte Bewegungen und Gesten unterteilt. Unbeabsichtigte Bewegungen haben als Geste keine Bedeutung und werden infolgedessen nicht weiter betrachtet. Die Gruppe der Gesten wird in zwei Unterklassen aufgeteilt. Die erste Klasse beinhaltet manipulative Gesten, welche die direkte Interaktion mit Objekten, wie zum Beispiel das Verschieben eines Objekts, beschreibt. Die zweite Unterklasse beinhaltet die kommunikativen Gesten, die wiederum in Handlungen und Symbole aufgespalten werden. Bei Handlungen wird die Bewegung interpretiert. Sie können nachahmende Gesten sein, bei denen eine Handlung imitiert wird, oder zeigende Gesten. Bei den Symbolen gibt es referentielle Gesten, die eine Bewegung mit einem Objekt verknüpfen, und modale Gesten, die sich auf sprachliche Äußerungen beziehen.

Bei der Gestenerkennung wird weiterhin zwischen statischen und dynamischen Gesten unterschieden [24]. Statische Gesten beschreiben eine Körperhaltung zu einem Zeitpunkt. Sie lassen sich dementsprechend in einem Einzelbild erkennen. Dynamische Gesten beziehen sich hingegen auf eine Bewegung, sodass neben der Körperpose auch temporale Informationen eine Rolle spielen. Für ihre Erkennung ist eine Folge von Bildern notwendig.

2.4 Anforderungen an ein Gesten-Interface

Ziel beim Entwurf eines Gesten-Interface ist es bisherige Interfaces zu verbessern. Der Erfolg eines solchen Interface hängt von der Akzeptanz der Benutzer ab. Hierzu muss das Gesten-Interface ein Maß an Qualität erreichen, dass durch die Einhaltung von Anforderungen gewährleistet wird. Im Folgenden werden die wichtigsten Aspekte aus den Veröffentlichungen von Saffer [21] und Wachs *et al.* [25] zusammengefasst. Die Anforderungen können je nach Anwendung variieren. Je nach Einsatzgebiet des Interface kann auf einzelne Anforderungen weniger Wert gelegt werden, wenn diese für den speziellen Anwendungsfall nicht relevant sind.

Reaktionsfähigkeit des Systems und Feedback auf eine Geste

Die Fähigkeit, Gesten in Echtzeit zu erkennen, ist eine elementare Voraussetzung. Ein System mit hoher Latenz bei der Gestenerkennung wird in praktischen Anwendungen nicht akzeptiert werden. Der Benutzer muss das Gefühl haben, dass das System ohne Verzögerung auf seine Eingabe reagiert. Um dies zu gewährleisten, muss das System dem Benutzer als Reaktion auf eine Geste in Echtzeit Feedback geben, um ihm zu signalisieren, dass die Geste richtig ausgeführt und erkannt wurde. Reagiert das System nicht in Echtzeit, kann es vorkommen, dass ein Benutzer Funktionen doppelt aktiviert. Kommt das Feedback zu spät, führt er die Geste erneut aus, weil er das Gefühl hat, dass sie nicht erkannt wurde.

Intuitive Gesten und eine einheitliche Gestensprache für alle Systeme

Intuitive Gesten vereinfachen die Benutzung eines Gesten-Interface. Die Bewegung einer Geste muss einfach auszuführen und sinnvoll mit der dahinterstehenden Funktion verknüpft sein. Es darf keine große Einweisung oder Erläuterung zur Ausführung der Geste notwendig sein. Im Optimalfall benötigt ein Benutzer, der zum ersten Mal das System verwendet, keine Einlernphase. Eine einheitliche Gestensprache ist anzustreben. Gesten, die sich bereits auf Systemen für bestimmte Funktionen durchgesetzt haben, sollten auch auf neuen Geräten für dieselbe Funktion eingesetzt werden. Die Verwendung von Standardgesten auf allen Geräte erleichtert dem Anwender die Bedienung. Viele Benutzer sind überfordert, wenn sie für jedes System neue, komplexe Gesten lernen und behalten müssen.

Angemessene Gesten

Gesten werden von Menschen mit unterschiedlichem Hintergrund abweichend interpretiert. Bei der Entwicklung von Gesten-Interfaces, die in verschiedenen Kulturen eingesetzt werden sollen, muss darauf geachtet werden, dass verwendete Gesten in diesen Kulturen keine ungleichen Bedeutungen besitzen oder diese als anstößig gelten. Zum Beispiel gilt die in den USA geläufige „okay“-Geste in Griechenland, der Türkei, Russland und dem Mittleren Osten als beleidigend. Darüber hinaus gibt es Situationen, in denen bestimmte Gesten unangemessen sind. Gesten mit ausfallenden Armbewegungen und Tanzgesten sind zum Beispiel an öffentlichen Orten weniger üblich als in Discos.

Abbildung 2.3: Die „Okay“-Geste

Komfort bei der Gestenausführung

Bei der Auswahl der Gesten, die für ein System verwendet werden, soll darauf geachtet werden, dass das Ausführen der Gesten nicht zu körperlichen Anstrengungen führt. Extreme Belastungen für die Muskeln sind zu vermeiden. Zu unnötigen Belastungen führen unter anderem extreme Körperhaltungen, die Notwendigkeit, eine Körperpose für eine bestimmte Zeit halten zu müssen, und das Ausführen einer Geste mit einer vorgeschriebenen, hohen Geschwindigkeit. Führt die Ausführung der Gesten zu Anstrengungen und Ermüdung des Benutzers kann dieser das System nicht über einen längeren Zeitraum bedienen.

Keine Verwendung von Hilfsmitteln

Die Erkennung von Gesten kann durch Hilfsmittel, die vom Benutzer getragen werden, wie zum Beispiel einem Datenhandschuh oder Markern, erleichtert werden. Diese müssen jedoch vom Anwender vor der Benutzung angelegt werden. Für den Benutzer ist es angenehmer, lediglich vor das System zu treten und direkt mit der Interaktion beginnnen zu können. Um eine hohe Benutzerfreundlichkeit zu gewährleisten, muss auf die Verwendung von Hilfsmitteln verzichtet werden.

Personeninvarianz

Menschen unterscheiden sich bezüglich Körpergröße, Form, Extremitätenlänge und Kleidung voneinander. Beim Entwerfen eines Gesten-Interface muss darauf geachtet werden, dass das System nicht auf einen spezifischen Körpertyp bei der Erkennung der Gesten angewiesen ist, sondern mit Personen unterschiedlicher Ausprägung zurecht kommt. Jeder Mensch sollte in der Lage sein, das System bedienen zu können.

Vertrauenswürdigkeit

Ein Benutzer sträubt sich ein System zu verwenden, das ihm nicht als sicher erscheint. Er darf keine Angst vor der Interaktion haben. Das System muss dem Benutzer das Gefühl geben, dass es ihm keinen materiellen Schaden zufügt und mit seinen persönlichen Daten wie Adressen und Photos vertrauensvoll umgeht.

Verspieltheit

Gesten-Interfaces sollen den Benutzer animieren, unbekannte Bestandteile des Systems spielerisch zu erlernen. Voraussetzung für das Learning by Doing ist ein System, das möglichst wenig Fehler bei der Verwendung zulässt und die Möglichkeit beinhaltet, Fehler korrigieren zu können.

Geringe Kosten

Damit ein Gesten-Interface kommerziell erfolgreich sein kann, müssen die Kosten möglichst gering gehalten werden. Durch den Einsatz von zusätzlichen oder exakteren Sensoren kann die Erkennung der Gesten verbessert oder durch die Verwendung von Hardware mit höherer Rechenleistung die Latenz verringert werden. Diese Maßnahmen sorgen zwar für positive Effekte bei der Gestenerkennung, verursachen im Gegenzug aber eine Erhöhung der Kosten. Hierbei muss ein Kompromiss gefunden werden, sodass auf der einen Seite die Gestenerkennung mit all ihren Anforderungen gewährleistet ist, auf der anderen Seite der Preis des Systems für den Kunden attraktiv bleibt.

2.5 Einsatzgebiete von Gestensteuerungen

Gesten-Interfaces lassen sich in vielen Bereichen des menschlichen Lebens einsetzen. Hierzu gehören unter anderem Spiele und Entertainment, medizinische Anwendungen, das Kriesenmanagement und die Mensch-Roboter-Interaktion.

Weit verbreitet ist der Einsatz der Gestensteuerung im Bereich der Videospiele. Nintendo veröffentlichte mit seiner Wii (Abbildung 2.4a) im Jahr 2006 erstmals eine Spielekonsole mit Gestensteuerung, die sich am Massenmarkt durchsetzte. Mittlerweile gibt es mit der PlayStation Move von Sony (Abbildung 2.4b) und der Kinect von Microsoft (Abbildung 2.4c) weitere Videospielkonsolen, die das Interagieren mit der Spielewelt durch Freiform-Gesten ermöglichen [26]. Nintendo und Sony setzen zum Erkennen der Gesten auf Controller, die vom Spieler in der Hand gehalten werden müssen. Die Kinect hingegen ermöglicht das Erkennen von Spieleingaben ohne getragenen Controller. Auch jenseits der Videospiele ergeben sich Möglichkeiten, die Gestensteuerung im Entertainmentbereich einzusetzen. Sie machen zum Beispiel Fernbedienungen überflüssig. Freeman und Weissman [27] zeigen dies, indem sie einen Fernseher mit Handgesten bedienen.

Abbildung 2.4: Videospielekonsolen, die mit einer Gestensteuerung arbeiten: a) Der Controller der Wii von Nintendo b) der Controller der PlayStation Move von Sony und c) die Kinect von Microsoft [26]

Im medizinischen Umfeld ist steriles Arbeiten eines der wichtigsten Aspekte. Bakterien und Keime sammeln sich auf Eíngabegeräten wie Maus und Tastatur an. Jede Form der berührungslosen Interaktion vermeidet den Kontakt mit diesen Keimherden. Ein Beispiel für eine Gestensteuerung, die steriles Arbeiten ermöglicht, ist das von Wachs *et al.* [28]

beschriebene System Gestix (Abbildung 2.5). Dieses Gesten-Interface ermöglicht einem Arzt in einem Operationssaal die Navigation und Manipulation von Bildern in einer elektronischens Patientenakte mit Hilfe von Gesten. Die Verwendung von Maus und Tastatur entfällt, sodass die Hände des Arztes steril bleiben und die Aufmerksamkeit des Arztes während der Operation nicht verloren geht.

Abbildung 2.5: Das Gestix System dient zur sterilen Navigation und Manipulation von Bildern im Operationssaal [25]

Das Bedienen von technischen Geräten stellt für Menschen mit körperlicher Behinderung oder eingeschränkter Mobilität eine Herausforderung dar. In Abhängigkeit der Einschränkungen können individuelle Lösungen entwickelt werden, sodass das Steuern von Geräten mit Hilfe von Gesten ermöglicht oder vereinfacht werden kann. Dies führt für Menschen mit Handicap zu einer Erweiterung ihrer Fähigkeiten und einer Erleichterung des täglichen Lebens. Kuno *et al.* [29] beschreiben die Steuerung eines Rollstuhls mit Hilfe von Gesten, die mit dem Kopf ausgeführt werden.

Bei der Bewältigung von Krisen, wie zum Beispiel Naturkatastrophen, ist der Zugriff auf eine große Menge komplexer Daten notwendig. Diese müssen gleichzeitig einer Gruppe von Menschen zugänglich gemacht werden, die den Einsatz zur Bewältigung der Katastrophe planen und koordinieren. Dies führt zur Notwendigkeit großer Videoleinwände, auf denen die Daten visualisiert werden und eine Mehrbenutzerinteraktion möglich ist. Die Steuerung dieser Displays mit Hilfe von Berührungsgesten ist nur möglich, wenn die Benutzer alle Stellen des Displays mit der Hand erreichen können. Für den Fall, dass dies auf Grund der Größe des Displays nicht gewährleistet ist, werden, wie in Abbildung 2.6, Freiform-Gesten zur Interaktion eingesetzt [14],[30].

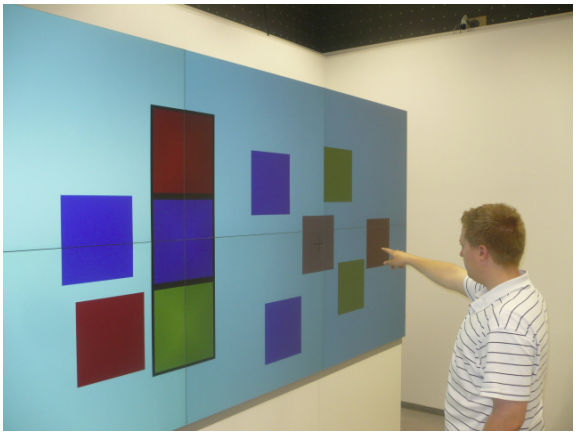


Abbildung 2.6: Interaktion mit einer großen Videoleinwand, wie sie beim Krisenmanagement eingesetzt wird [14]

Die Kommunikation mit einem Roboter wird durch den Einsatz von Gesten einfacher und intuitiver [31], [32], [33], [16]. In Kombination mit gesprochenen Anweisungen ergeben sie wertvolle Werkzeuge um dem Roboter geometrische Informationen mitzuteilen. Der Roboter kann auf Anweisungen wie zum Beispiel „Fahr dort hin“ oder „Bring mir diese Flasche“, die zusammen mit einer Zeigegeste auf die entsprechende Position artikuliert werden, reagieren. Mit dieser Kommunikationsform nähert sich die Mensch-Roboter-Interaktion der Kommunikation zwischen Menschen an und sorgt für eine größere Akzeptanz von Robotern.

3. Praktische Gesten für ein Gesten-Interface

Die Auswahl der Gesten stellt einen entscheidenden Schritt bei der Entwicklung eines Gesten-Interface dar. Im Prinzip können sämtliche Bewegungen des menschlichen Körpers oder Teile des Körpers als Gesten für die Steuerung eines Systems eingesetzt werden. Doch nur wenige Gesten machen Sinn, wenn man sie mit einer Funktion verknüpft. Im Folgenden wird eine Auswahl an praktischen Freiform-Gesten vorgestellt, die für die Bedienung der grundlegenden Funktionen eines Gesten-Interface eingesetzt werden können. Sie werden mit Hilfe der Arme und der Finger ausgeführt. Ein Großteil der Gesten basiert auf Bewegungsmustern, die von Touchscreens bekannt und somit bereits weit verbreitet sind.

3.1 Aktivieren eines Systems

Die einfachste aller Gesten, die zum Ein- und Ausschalten eines Gerätes oder zum Aktivieren einer Funktion eingesetzt werden kann, ist die **Anwesenheit** einer Person (eng. proximity activates/deactivates) [21]. Sie wird vor allem zum Sparen von Energie eingesetzt. So können das Licht oder andere elektrische Geräte an die Anwesenheit einer oder mehrerer Personen in einem Raum gekoppelt werden. Befindet sich mindestens eine Person in einem Raum werden die Geräte aktiviert. Nachdem die letzte Person den Raum wieder verlassen hat, werden sie ausgeschaltet. Somit wird der Energieverbrauch minimiert, indem Geräte oder das Licht nur dann eingeschalten sind, wenn auch jemand davon profitiert. Dies spart Geld und schont die Umwelt. Die Aktivierung einer Gestensteuerung eines Systems kann auch an die Anwesenheit einer Person in einem bestimmten Bereich gekoppelt sein.

3.2 Einrichten und Aufheben einer Bediensperre

Es gibt Anwendungsfälle, bei denen die Kopplung der Aktivierung der Gestensteuerung mit der Anwesenheit einer Person nicht ausreicht. Aus diesem Grund wird eine Geste benötigt, die eine Bediensperre einrichten und entfernen kann [34]. Hierzu kann die **Winken**-Geste (eng. wave to activate) eingesetzt werden. Sie ermöglicht dem Benutzer die gezielte Deaktivierung der Gestensteuerung, damit das System Bewegungen vom Benutzer, die keine Eingabe für das System sein sollen, nicht fehlinterpretiert. Anschließend kann die Gestensteureung durch erneute Anwendung der Winken-Geste wieder reaktiviert werden. Winken ist eine intuitive Geste, die in der Kommunikation zwischen Menschen als Begrüßung weit

verbreitet ist und auf Grund ihres Bewegungsmusters vom Benutzer nicht ungewollt ausgeführt wird [21], [35].

3.3 Annulieren von Fehleingaben

Die Benutzerfreundlichkeit eines Systems steigt mit der Möglichkeit, Fehler korrigieren zu können. Vor allem bei Geräten mit Gestenerkennung kommt es zu Fehlinterpretationen von Bewegungen, da diese schwieriger zu erkennen sind als das Drücken einer Taste. Üblicherweise werden hierfür symbolische Gesten verwendet. Zum Annulieren einer Fehlbedienung wird die **Schleife nach links** Geste verwendet. Hierzu wird, wie in Abbildung 3.1 zu sehen ist, mit dem Arm eine Schleifenbewegung nach links durchgeführt, um den Zustand des Systems einen Schritt zurückzusetzen. Mit der komplementären **Schleife nach rechts** Geste kann der Zustand schrittweise wiederhergestellt werden [34].

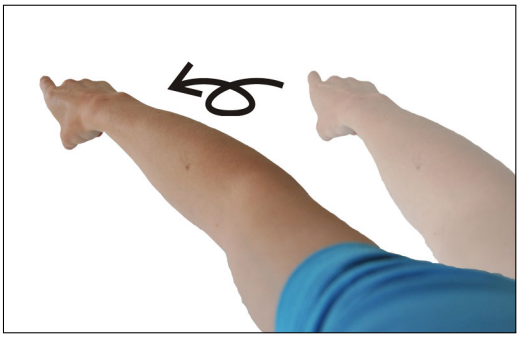
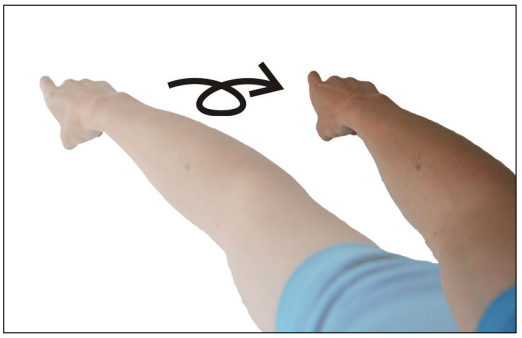
a)  b) 

Abbildung 3.1: Die Gesten zur Annulierung von Fehleingaben: a) Die Schleife nach links Geste für die Funktion „*Rückgänging*“ und b) die Schleife nach rechts Geste für die Funktion „*Wiederherstellen*“. In Anlehnung an [34]

3.4 Aktivieren einer Funktion

Zum Aktivieren einer Funktion wird auf einem Touchscreen die Antippen-Geste (eng. tab to activate) verwendet. Hierzu berührt der Benutzer eine definierte Fläche oder ein Objekt, wie zum Beispiel einen Button, auf dem Bildschirm. Es wird zwischen *active on press* und *active on release* unterschieden. Bei *active on press* wird die Funktion aktiviert, sobald der Benutzer den Bildschirm an der entsprechenden Stelle berührt. Alternativ kann die Funktion erst aktiviert werden, wenn der Finger des Anwenders den Touchscreen verlässt. Dies wird als *active on release* bezeichnet. Hiermit ist eine Korrektur der Eingabe möglich, falls sich der Benutzer während der Ausführung der Geste entscheidet, die Funktion doch nicht aktivieren zu wollen. Eine entsprechende Freiform-Geste stellt die **Zeigegeste** (eng. point to activate) dar, die ohne Berührung des Bildschirms auskommt [21]. Hierfür wird durch die Haltung des Arms ein Punkt auf dem Display beschrieben. Auf die Verwendung von *active on press* und *active on release* wird verzichtet. An Stelle dessen wird eine Funktion aktiviert, wenn der Punkt auf dem Display, der von der Zeigegeste beschrieben wird, eine vorgeschriebene Zeit den Button nicht verlässt. Dies lässt sich durch einen Kreis visualisieren, der sich währenddessen um den Button schließt [35]. Alternativ kann ein Tap als Freiform-Geste ausgeführt werden. Der Benutzer bewegt dabei seine Handfläche auf einen Button zu und zieht diese gleich wieder zurück. Bei Systemen, die in der Lage sind, die Handkonfiguration zu erkennen, kann eine Funktion aktiviert werden, indem der Benutzer mit der Handfläche auf den Button zeigt und ihn anschließend aktiviert, indem er eine **Klick**-Geste ausführt. Hierzu knickt der Benutzer, wie in Abbildung 3.2 zu sehen, den Zeigefinger wie beim Betätigen der linken Maustaste kurz ab. Dies kann sowohl als *active on press* oder auch als *active on release* realisiert werden.

Abbildung 3.2: Die Klick-Geste wird zum Aktivieren einer Funktion oder zum Selektieren eines Objekts eingesetzt.

3.5 Auswählen eines Objekts

Um Objekte manipuliueren zu können, müssen diese zunächst ausgewählt werden. Dies ist mit den aus Abschnitt 3.4 bekannten Gesten zum Aktivieren einer Funktion möglich. Hierbei wird jedoch keine Funktion aktiviert, sondern das entsprechende Objekt selektiert. Neben der **Zeigegeste** (eng. point to select) [21] kann auch die **Klick**-Geste verwendet werden.

3.6 Verschieben von Objekten

Objekte in der realen Welt werden verschoben, indem man sie greift, an die gewünschte Position zieht und wieder loslässt. Auf Touchscreens wird das **Greifen und Ziehen** (eng. drag to move) ebenfalls als Geste zum Verschieben von Objekten angewendet [21]. Da man die Objekte hier nicht richtig greifen kann, reicht eine Berührung aus, um sie mit dem Finger zu verankern. Bei Freiform-Gesten wird die Verankerung realisiert, indem die offene Hand auf das Objekt zeigt und anschließend zu einer Faust geschlossen wird. Das Objekt folgt nun der Bewegung des Armes, bis die Faust wieder geöffnet wird (siehe Abbildung 3.3).

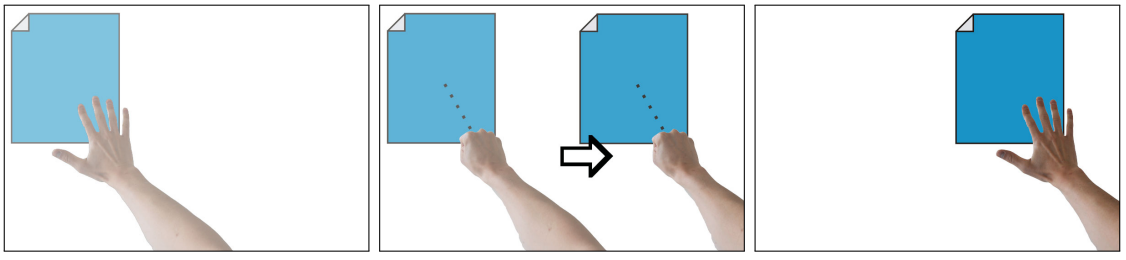


Abbildung 3.3: Objekte werden mit der Ziehen-Geste verschoben.

3.7 Scrollen und Blättern

Anwendungen bestehen aus einer Fülle von Informationen, die dem Anwender präsentiert werden. Auf Grund von Beschränkungen der Displaygröße vieler Geräte stehen dem Benutzer immer nur kleine Ausschnitte dieser Informationsmenge zur Verfügung. Somit werden

Gesten benötigt, um ein einfaches Navigieren durch diese Inhalte zu ermöglichen. Typische Darstellungsformen, durch die navigiert werden muss, sind Bilder oder Karten, die nur in Ausschnitten auf dem Bildschirm dargestellt werden können, vertikale und horizontale Listen und Folgen von Seiten, durch die geblättert wird [34].

Auf Touchscreens wird durch eine **Verschieben**-Geste (eng. slide to scroll) durch den Inhalt des Bildschirms navigiert [21]. Das Bewegungsmuster dieser Geste entspricht der Bewegung zum Verschieben von Objekten (Vergleich Abschnitt 3.6). Hierbei folgt allerdings nicht nur ein Objekt der Bewegung des Fingers sondern der gesamte Inhalt des scrollbaren Bereichs.

Entsprechend wird als Freiform-Geste der Inhalt durch Bilden einer Faust an die Hand gekoppelt. Der Inhalt folgt der Bewegung nach dem Prinzip der Verankerung. Öffnet der Anwender seine Faust, stoppt auch die Scrollbewegung des Inhalts. Da diese Geste ebenfalls für das Verschieben von Objekten verwendet wird, muss das System entscheiden, für welche Geste sie der Benutzer einsetzen möchte. Als Grundlage für diese Entscheidung wird die Art des Inhalt verwendet, auf den der Benutzer zeigt.

Eine Erweiterung stellt die **Schnippen**-Geste (eng. flick to nudge) dar [21]. Hierbei besitzt der Inhalt eine definierte Trägheit, sodass er sich nach dem Lösen der Verankerung mit der Endgeschwindigkeit der Geste weiterbewegt und langsam abbremst, bis er stoppt. Der Scrollvorgang kann durch **Antippen** (eng. tap to stop) unterbrochen werden. Hierzu bewegt der Benutzer seine Hand kurz in Richtung des Bildschirms und zieht sie anschließend wieder zurück. Zur Unterscheidung zwischen der Verschieben- und der Schnippen-Geste wird die Endgeschwindigkeit beim Loslassen des Objekts verwendet. Ist diese ausreichend schnell, wird die Geste als Schnippen gewertet. Andernfalls wird sie als Ziehen interpretiert, sodass ein gezieltes Positionieren des Inhalts möglich wird.

Eine einfache Version des Scrollens kann realisiert werden indem ein andauernder Scrollvorgang durch eine Geste gestartet und anschließend von einer weiteren Geste beendet wird (eng. slide and hold for continuous scroll) [21]. Der Scrollvorgang kann gestartet werden, indem der Anwender eine **Wischbewegung** (Abbildung 3.4) in die entsprechende Richtung ausführt. Dabei wird lediglich die komplette Bewegung ohne genaue Informationen wie Geschwindigkeit, Startpunkt oder Endpunkt interpretiert. Um den Scrollvorgang zu beenden kommt wiederum die **Antippen**-Geste zum Einsatz. Für das schrittweise Blättern durch Seiten wird ebenfalls die Wischen-Geste eingesetzt.

Abbildung 3.4: Die Wischen-Geste, die zum Blättern durch Seiten verwendet wird.

3.8 Zoomen und Skalieren

Die meisten Multitouchgeräte arbeiten beim Zoomen der Ansicht oder beim Skalieren von Objekten mit der **Zwicken**-Geste zum Herauszoomen (eng. pinch to shrink) und der

Spreizen-Geste zum Heranzoomen (eng. spread to enlarge) [21], [34]. Hierzu werden zwei Berührungspunkte ausgewertet. Auf kleinen Touchscreens werden hierzu Daumen und Zeigefinger verwendet. Bei größeren multitouchfähigen Geräten bieten sich Finger von jeweils einer Hand an. Um ein Objekt zu skalieren oder die Ansichtsgröße anzupassen, wird der Abstand zwischen den zwei Kontaktpunkten verändert. Bewegen sie sich aufeinander zu, wird das Objekt verkleinert beziehungsweise herausgezoomt. Entfernen sie sich voneinander, wird das Objekt vergrößert oder herangezoomt. Diese intuitive Geste ist für den Benutzer verständlich, da sie dem Prinzip der Verankerung folgt. Jeder der beiden Berührungspunkte heftet sich an jeweils eine Bildstelle. Durch die Verankerung folgen diese Bildpunkte den Bewegungen der Finger, sodass das Objekt oder die Ansicht dementsprechend skaliert wird. Dies ermöglicht ein proportionales, stufenloses Skalieren und Zoomen. Zoomen mit Hilfe der Zwicken- und Spreizen-Geste kann nicht nur auf Multitouchscreens eingesetzt werden. Sie lassen sich ebenfalls als Freiform-Gesten umsetzen. Hierzu zeigen wie in Abbildung 3.5 zu sehen linker und rechter Arm auf die gewünschten Positionen, die durch Bilden einer Faust zu Verankerungspunkten werden. Skalieren und Zoomen ist somit durch Bewegung der Arme aufeinander zu oder voneinander weg möglich. Die Verankerung wird gelöst, indem der Benutzer die Faust öffnet.

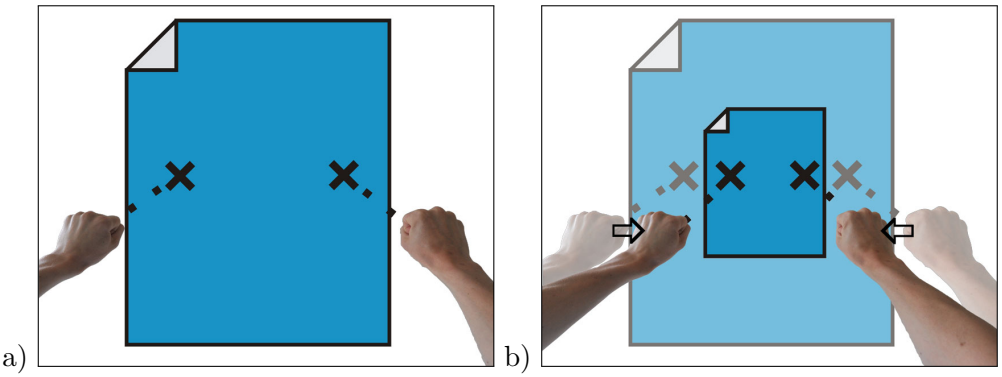


Abbildung 3.5: Die Zwicken-Geste zum Verkleinern von Objekten als Freiform-Geste: a) Zwei Verankerungspunkte werden durch das Bilden einer Faust der linken und rechten Hand erzeugt. b) Das Objekt wird verkleinert, wenn sich die Hände aufeinander zu bewegen.

Als Alternative kann die Zoomstufe abhängig vom Abstand des Körpers oder eines Körperteils zum Display gemacht werden. Menschen lehnen sich automatisch nach vorne oder bewegen sich auf ein Objekt zu, wenn sie Details nicht erkennen können. Diese intuitive Bewegung kann als Geste für eine Anpassung der Zoomstufe interpretiert werden [36].

Je nach Anwendung sind zusätzliche Zoomfeatures denkbar, die beim Ausführen einer Geste eine voreingestellte Zoomstufe aktivieren. Auf dem iPhone kann zum Beispiel durch das Ausführen eines doppelten Antippens (eng. double tap) auf ein vergrößertes Bild die Zoomstufe auf 100% eingestellt werden, sodass das gesamte Bild auf dem Display zu sehen ist.

3.9 Drehen

Gesten zur Rotation von Objekten oder der Ansicht in der Bildebene sind eng mit den Gesten zum Zoomen und Skalieren verbunden. Auch hier kommt das Prinzip der Verankerung zum Einsatz, das eine intuitive Bedienung ermöglicht. Zwei Punkte werden markiert, indem die linke und rechte Hand eine Faust bilden. Wie in Abbildung 3.6 zu sehen, beschreibt dabei eine Faust den Rotationsmittelpunkt. Das Objekt folgt der Bewegung der anderen Hand, sodass sich der Drehwinkel aus dem Winkel der virtuellen Linie zwischen

den beiden Punkten ergibt. Es gibt Objekte, die nur um einen festen Rotationspunkt gedreht werden können, der vom Benutzer nicht beeinflusst werden kann. Ein Drehregler kann zum Beispiel nur um seinen eigenen Mittelpunkt gedreht werden. Hier reicht die Verwendung eines Verankerungspunktes aus, um das Objekt zu rotieren [34].

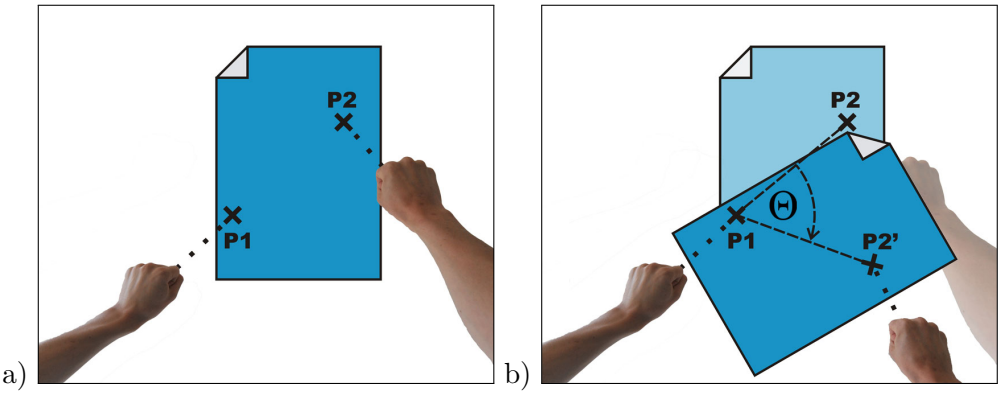


Abbildung 3.6: Rotation von Objekten: a) Linke und rechte Faust beschreiben die Verankerungspunkte P1 und P2 auf dem Objekt. Die linke Hand gibt dabei den Rotationsmittelpunkt P1 an. b) Das Objekt wird um P1 gedreht, indem P2 der Bewegung des Arms folgt. In Anlehnung an [37]

3.10 Kombination von Verschieben, Zoomen und Drehen

Die Gesten zum Verschiebung, Zoomen und Drehen lassen sich kombinieren. Hierzu werden zwei Verankerungspunkte benötigt. Die Anpassung der Ansicht oder die Veränderung der Objektgröße und -lage kann somit mit einer flüssigen Bewegung durchgeführt werden, ohne das jeweils eine neue Gesten ausgeführt werden muss [37].

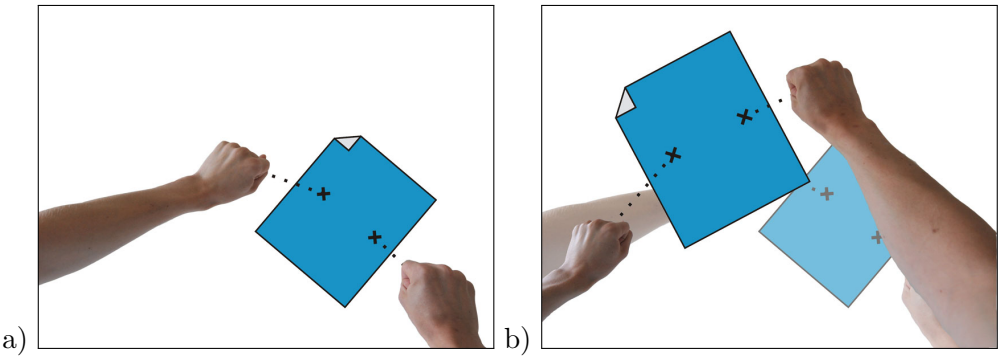


Abbildung 3.7: Verschieben, Zoomen und Drehen lassen sich kombinieren. Hierzu werden in a) zwei Verankerungspunkte festgelegt, denen in b) das Objekt folgt.

4. Die Microsoft Kinect-Kamera

Die Kinect-Kamera wurde von Microsoft zur Steuerung der Videospielekonsole Xbox 360 entwickelt. Als Eingabe wird auf handgetragene Controller verzichtet. Die Steuerung der Spiele ist einzig über Körperbewegungen und gesprochene Kommandos möglich. Jenseits der Videospiele ergeben sich neue Einsatzgebiete für die Kinect-Kamera.

4.1 Hardwareausstattung der Kinect-Kamera

Die Kinect-Kamera (Abbildung 4.1) besitzt eine RGB-Kamera, einen Tiefensensor sowie ein zur Erkennung von Bewegung irrelevantes Array aus vier Mikrofonen. Der Sichtwinkel der Kameras beträgt vertikal 43° und horizontal 57°. Um den Sichtbereich der Kameras auf ein gewünschtes Sichtfeld anzupassen, lässt sich der Neigungswinkel der Kinect-Kamera mit Hilfe eines Motors mechanisch im Bereich ± 28° anpassen.

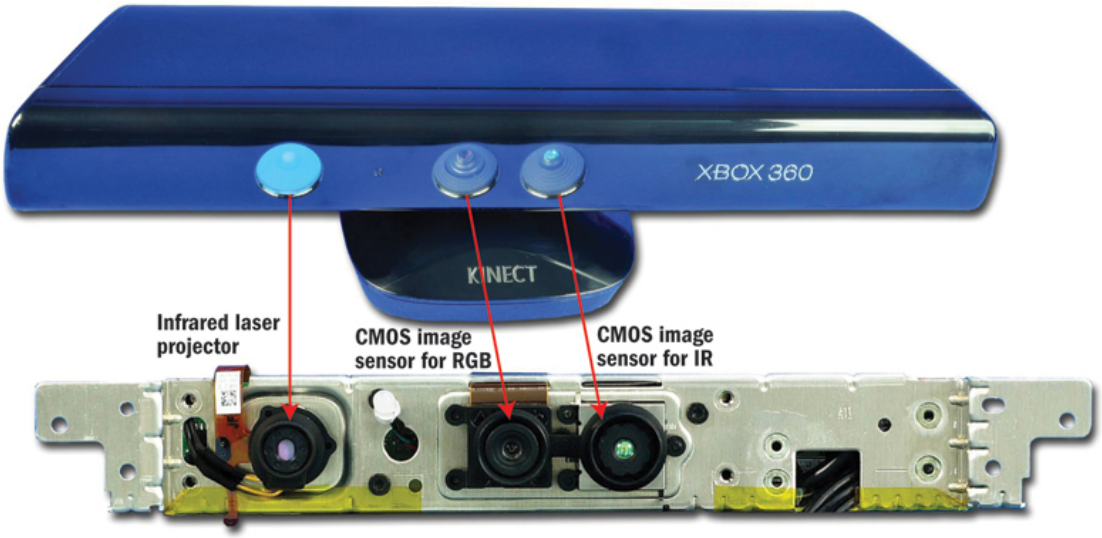


Abbildung 4.1: Die Kinect-Kamera von Microsoft [38]

Der RGB-Sensor liefert einen Stream an Farbbildern mit 32-bit Farbtiefe und VGA-Auflösung (640 x 480 Pixel). Der von der israelischen Firma PrimeSense entwickelte Tiefensensor berechnet das Tiefenbild nach einem mit Light Coding benannten Verfahren

[39]. Hierzu wird ein Infrarot-Punktmuster auf die Umgebung geworfen, das wiederum von einem Standard CMOS-Bildsensor aufgenommen wird. Ein System-on-a-Chip berechnet anhand des Bildes der CMOS-Kamera das Tiefenbild der Kamera. Dieses gibt für jeden Pixel einen Wert an, der die Entfernung des Objekts, das vom Pixel abgedeckt wird, zur Kamera beschreibt. Durch die Verwendung von Infrarotlicht ist dieses Verfahren für das menschliche Auge unsichtbar. Der Sensor liefert einen Stream an 16Bit Tiefenbildern mit VGA-Auflösung (640 x 480 Pixel). Sowohl Farb- als auch Tiefenstream arbeiten mit einer Bildwiederholrate von 30 Bildern pro Sekunde. Die Kinect-Kamera wird per USB an einen PC angeschlossen und benötigt zusätzlich eine externe Energieversorgung [38], [40].

4.2 Vom Tiefenbild zur Körperpose

Für die Erkennung von Gesten ist es nötig, die aktuelle Haltung des Körpers, die Körperpose, zu kennen. Sie beschreibt die Lage der Körperteile im Raum und wird auch als Konfiguration des Körpers bezeichnet. Als Grundlage zur Berechnung der Körperpose wird das von der Kinect-Kamera erzeugte Tiefenbild verwendet. Die Lage eines Körperteils lässt sich durch die dreidimensionalen Koordinaten der Gelenke beschreiben. Zur Berechnung der Körperpose anhand des Tiefenbildes der Kinect-Kamera in Echtzeit hat Microsoft Research den Algorithmus *Real-Time Human Pose Recognition in Parts from Single Depth Images* [41] vorgestellt. Der Algorithmus verzichtet bei der Bestimmung der Koordinaten der Gelenke auf Informationen vergangener Frames. Die Körperpose wird für jedes einzelne Tiefenbild unabhängig berechnet.

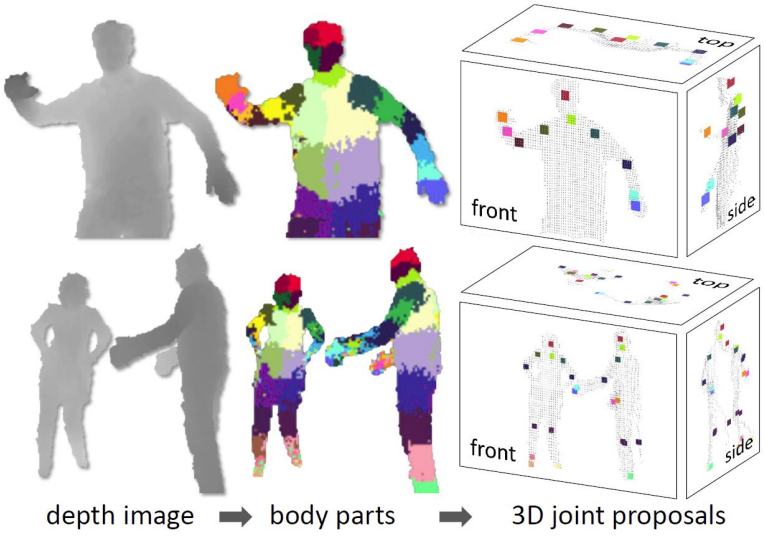


Abbildung 4.2: Die Zwischenschritte bei der Berechnung der Körperpose: Im Tiefenbild wird für jeden Pixel des Menschen bestimmt, zu welchem Körperteil er gehört. Aus den Positionen der Pixel, die ein Körperteil beschreiben, werden die 3D-Koordinaten des Körperteils berechnet. [41]

Kernidee des Algorithmus ist es, die Schätzung der Körperpose vorzunehmen, indem mit Hilfe eines Objekterkennungsansatzes das Tiefenbild in Körperteile segmentiert wird. Für jeden Pixel wird die Wahrscheinlichkeitsdichte berechnet, die angibt, wie wahrscheinlich der Pixel zu allen möglichen Körperteilen gehört. Zur Klassifikation der Pixel verwendet Microsoft Features $f_\Theta(I, x)$, die Tiefenvergleiche in dem Bild I an der Position x durchführen.

$$f_\Theta(I, x) = d_I\left(x + \frac{u}{d_l(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right)$$

Ein Feature wird durch den Parameter $\Theta = (u, v)$ mit den Offsetwerten u und v beschrieben. $d_I(x)$ gibt die Tiefe im Bild an der Position x an. Damit die Features tiefeninvariant sind, werden die Offsets mit $\frac{1}{d_I(x)}$ normalisiert. Je nach Wahl von u und v ergeben sich unterschiedliche Tiefenfeatures die sich zum Auffinden unterschiedlicher Körperteile eignen. In Abbildung 4.3 sind zwei Beispielfeatures visualisiert. Für das Auffinden von Pixeln, die sich sehr weit oben am Körper befinden, eignet sich Feature f_{Θ_1}, da es nach oben zeigt. Feature f_{Θ_2} liefert hingegen bei dünnen, vertikalen Strukturen positive Resultate.

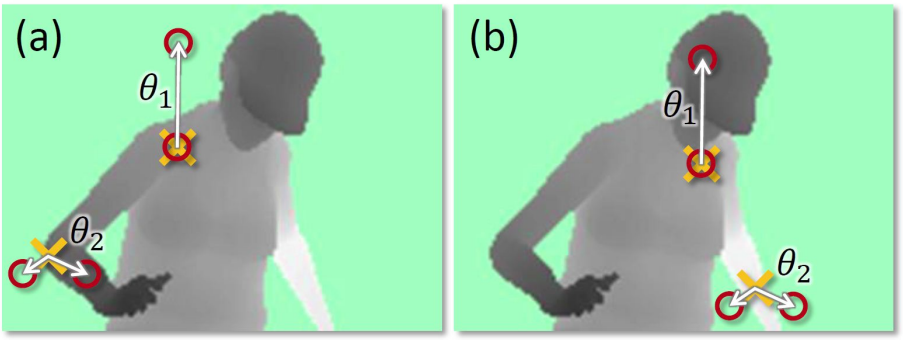


Abbildung 4.3: Visualisierung von zwei Tiefenfeatures: Ein gelbes Kreuz gibt die Position x des Pixels an, das klassifiziert wird. Die Position der roten Kreise ergeben sich durch die Offsets u und v des verwendeten Features. f_{Θ_1} und f_{Θ_2} ergeben in a) eine große, in b) eine kleine Tiefendifferenz. [41]

Einzelne Features können keine Aussage darüber treffen, zu welchem Körperteil ein Pixel gehört. Zur Klassifikation wird deswegen ein randomisierter Entscheidungswald (eng. Randomized Decision Forest) verwendet. Dieser setzt sich aus T Entscheidungsbäumen zusammen, die jeweils aus einer Menge an Aufteilungs- und Blattknoten bestehen. Jeder Aufteilungsknoten entspricht einem Feature f_Θ mit einem Schwellenwert τ. Zur Klassifikation eines Pixels durchläuft dieser, wie in Abbildung 4.4 zu sehen, jeden Entscheidungsbaum. Für einen Baum t ergibt sich anhand der Entscheidungen an den Aufteilungsknoten, die auf Grundlage des Features und des Schwellenwerts getroffen werden, ein Blattknoten und somit eine gelernte Wahrscheinlichkeitsverteilung über alle Körperteile $P_t(c|I, x)$.

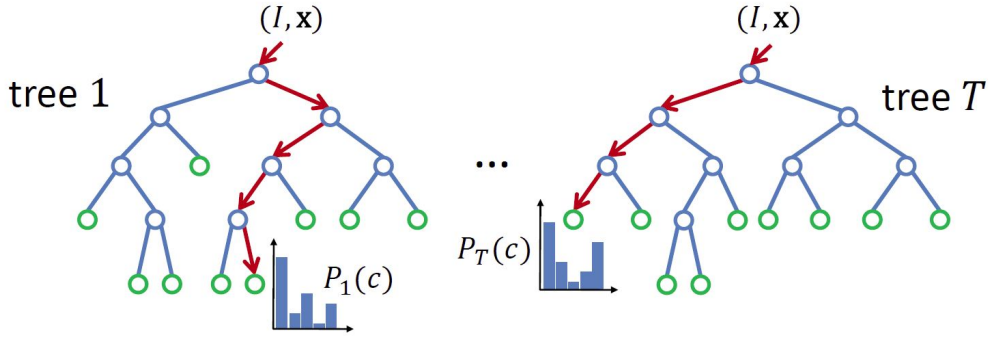


Abbildung 4.4: Ein randomisierter Entscheidungswald, der aus mehreren Entscheidungsbäumen besteht. [41]

Um die endgültige Wahrscheinlichkeitsverteilung zu erhalten, wird über die Ergebnisse aller Entscheidungsbäume gemittelt.

$$P(c|I, x) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|I, x)$$

Microsoft verwendet zum Parametrisieren jedes Entscheidungsbaums unterschiedliche synthetische Trainingsbilder. Von diesen Bildern werden jeweils 2000 Pixel zum Trainieren der Entscheidungsbäume zufällig ausgewählt. Zunächst wird eine Menge an Aufteilungskandidaten $\phi = (\Theta, \tau)$ zufällig erzeugt. Diese bestehen aus dem Parameter des Features Θ und dem Schwellenwert τ. Die Menge der Trainingspixel $Q = \{(I, x)\}$ wird nun jeweils mit Hilfe eines Aufteilungskandidaten ϕ in zwei Untermengen klassifiziert. Hieraus ergibt sich eine linke und eine rechte Teilmenge.

$$Q_{links}(\phi) = \{(I, x) | f_\Theta(I, x) < \tau\}$$

$$Q_{rechts}(\phi) = Q \backslash Q_{links}(\phi)$$

Anschließend wird für jeden Aufteilungskandidaten θ der Informationsgewinn $G(\phi)$ berechnet.

$$G(\phi) = H(Q) - \sum_{s \in \{l,r\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi))$$

$H(Q)$ ist hierbei die Shannonentropie, die auf dem normalisierten Histogramm der Körperteilbeschriftungen berechnet wird. Ausgewählt wird der Aufteilungskandidat mit dem größten Informationsgewinn.

$$\phi^* = \arg\max_\phi G(\phi)$$

Wenn der Informationsgewinn dieses Aufteilungsknotens über einem Schwellenwert liegt und die maximale Tiefe des Entscheidungsbaums noch nicht erreicht ist, wird für die Teilmengen Q_{links} und Q_{rechts} rekursiv fortgefahren. Microsoft verwendet zum Trainieren von 3 Entscheidungsbäumen mit jeweils 20 Ebenen 1 Million Bilder.

Abschließend wird aus den klassifizierten Pixeln die Position der Körperteile berechnet. Um den Einfluss von falsch klassifizierten Ausreißern zu minimieren, wird die wahrscheinlichste Position nicht über eine Mittelung der zugehörigen Pixel berechnet, sondern durch Anwendung eines Mean Shift Algorithmus mit gewichtetem Gauss Kernel. Hierzu wird für jedes Körperteil c der Dichteschätzer $f_c(\hat{x})$ ausgewertet.

$$f_c(\hat{x}) \propto \sum_{i=1}^{N} w_{ic} \exp\left(-\left\|\frac{\hat{x} - \hat{x}_i}{b_c}\right\|^2\right)$$

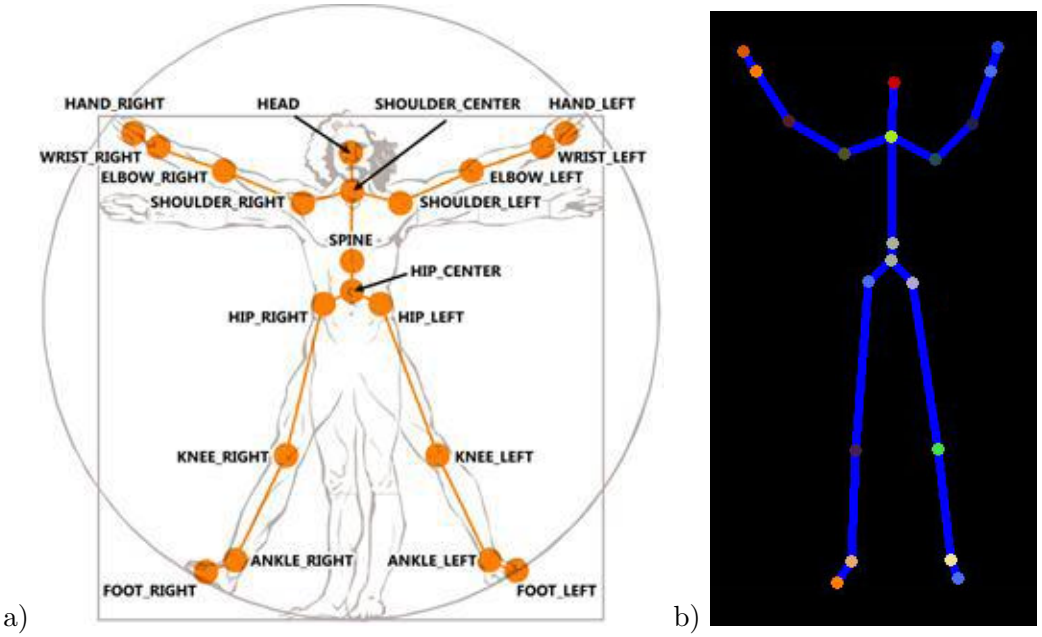
Dieser beinhaltet $\hat{x}$ eine dreidimensionale Weltkorrdinate, N die Anzahl der Bildpixel, ω_{ic} eine Pixelgewichtung und $\hat{x}_i$ die Rückprojektion des Pixels x_i mit Hilfe der Tiefe $d_I(x)$ in Weltkoordinaten. b_c beschreibt eine gelernet Bandbreite für jedes Körperteil c. Die Pixelgewichtung w_{ic} setzt sich aus der Körperteilwahrscheinlichkeit und des Welt Flächeninhalts des Pixels zusammen.

$$w_{ic} = P(c|I, x_i) \cdot d_I(x_i)^2$$

Der Mean Shift Algorithmus bestimmt für diese Wahrscheinlichkeitsverteilung den Modalwert und liefert somit eine Position für das Körperteil. Zur Berechnung der endgültigen Positionen müssen die Werte noch um einen Offset in Z-Richtung zurückverschoben werden, da der berechnete Modalwert lediglich die Position auf der Oberfläche des Körpers beschreibt. Der Algorithmus verhält sich invariant gegenüber Körperhaltung, Körperform und Bekleidung. Die Klassifizierung der Pixel mit Hilfe von randomisierten Entscheidungswäldern kann auf einer GPU implementiert werden. Der Algorithmus stellt somit einen robusten und recheneffizienten Ansatz zur Bestimmung der Körperpose aus Tiefenbildern dar.

4.3 Das Kinect for Windows Software Development Kit (SDK)

Das von Microsoft Reseach entwickelte Kinect for Windows SDK [40] wurde Mitte Juni 2011 als Beta Version veröffentlicht. Es enthält Treiber für die Verwendung an einem Windows 7 PC und eine Sammlung an Programmierschnittstellen (eng. application programming interfaces, APIs), die Tiefenbildverarbeitung, das Erkennen von menschlicher Bewegung und Spracherkennung ermöglichen. Neben dem Zugriff auf die unverarbeiteten Datenstreams von Farb- und Tiefensensor ermöglicht die NUI API Skelettracking von Personen, die sich innerhalb des Blickfeldes der Kamera bewegen, das als Grundlage zur Gestenerkennung eingesetzt werden kann. Ein Skelett besteht dabei aus 20 Gelenken, deren Verteilung in Abbildung 4.5a) beschrieben wird.



a) b)

Abbildung 4.5: a) Schematische Darstellung des Skeletts mit seinen 20 Gelenken, das vom Kinect for Windows SDK berechnet wird und b) Visualisierung des berechneten Skeletts von einer aktiv getrackten Person. [40]

In jedem Frame sind für jedes Gelenk die dreidimensionalen Positionsdaten abrufbar. Die Menge aller Gelenkpositionen repräsentiert die Körperpose der getrackten Person. Informationen über einzelne Finger oder die Handkonfiguration stehen bei der Skelettberechnung des SDKs nicht zur Verfügung. Somit ist es mit den Skelettdaten nicht möglich, Gesten zu erkennen, die in Abhängigkeit der Fingerkonfiguration stehen. Standardmäßig werden die ersten beiden Personen, die vom System gefunden werden, aktiv getrackt. Für diese Skelette stehen vollständige Skelettdaten zur Verfügung. Für bis zu vier weitere Personen im Sichtfeld wird passives Tracking angewendet. Für diese Skellete sind nur eingeschränkte Informationen über die Position der Person und keine Informationen über die Körperpose vorhanden. Die Positionen der Gelenke werden als x-,y- und z-Koordinaten im Skelettkoordinatensystem (Abbildung 4.6) angegeben. Der Tiefensensor der Kinect liegt im Ursprung des Koordinatensystems. Die z-Achse, deren positive Werte in Blickrichtung der Kamera zeigen, die waagerechte x-Achse und die senkrechte y-Achse bilden ein rechtshändiges Koordinatensystem. Die positive y-Achse zeigt nach oben und die positive x-Achse aus Sicht der Kinect-Kamera nach links.

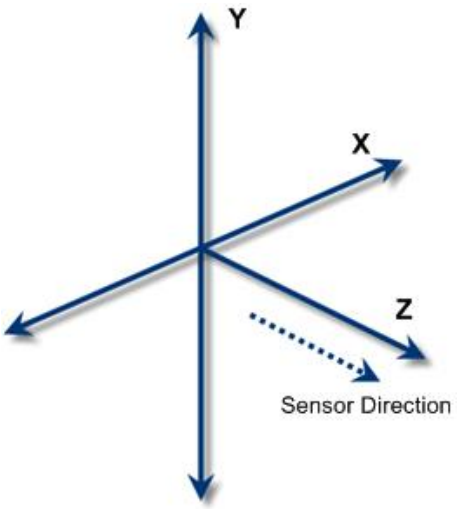


Abbildung 4.6: Das rechtshändige Skelettkoordinatensystem, in dem die Koordinaten der Skelettgelenke angegeben werden. [40]

Lizenz des SDK

Die Lizenz des Kinect for Windows SDK erlaubt lediglich eine Verwendung des SDKs zu nicht kommerziellen Zwecken. Hierzu gehören die wissenschaftliche Forschung, die Lehre und forschende Schüler- und Studentenprojekte. Eine Veröffentlichung von Demoanwendungen ist erlaubt. Ein SDK, das kommerzielle Nutzung ermöglicht, will Microsoft zu einem späteren Zeitpunkt veröffentlichen.

4.4 Charakteristika der Kinect-Kamera und des SDKs

Je nach Einsatzgebiet eines Gesten-Interfaces muss das Kamerasystem, das zur Erkennung der Gesten verwendet wird, unterschiedlichen Anforderungen gerecht werden. Es gibt keine Kamera, die sich für sämtliche Anwendungsszenarien gleich gut eignet. Die Verwendung der Kinect-Kamera und des zugehörigen SDKs bringt ebenfalls eine Reihe von Vor- und Nachteilen mit sich. Im Folgenden werden die positiven und negativen Eigenschaften der Kinect-Kamera und des SDKs erläutert.

Stärken der Kinect

Vorteil eines Tiefenbildes Die Tiefenbilder der Kinect-Kamera bieten Vorteile gegenüber den Bildern normaler Kameras. Sie sind farb-, textur- und helligkeitsinvariant, sodass sie auch bei schwachen Lichtverhältnissen erzeugt werden können. Sie vereinfachen die Hintergrundsubtraktion, sodass Objekten oder Personen, die frei im Raum stehen, ohne Probleme erkannt werden können. [41]

Posenberechnung ohne getragene Hardware Mit der Kinect-Kamera ist die Berechnung von Körperposen möglich, ohne dass der Benutzer zusätzliche Hardware am Körper tragen muss.

Keine Notwendigkeit einer Kalibrierpose Das Skeletttracking des Kinect-SDKs kommt ohne initiale Kalibrierpose aus. Somit ist ein Verlassen und Wiederbetreten des Sichtbereichs der Kinect-Kamera möglich.

Günstiger Preis der Kinect-Kamera Kamerasysteme, die Tiefenbilder erzeugen können, gibt es schon lange zu kaufen. Deren Erwerb ist jedoch meist mit einem hohen Preis verbunden, sodass ihre Verwendung wenig attraktiv erscheint. Die Kinect-Kamera ist das erste System, das einen günstigen Verkaufspreis bietet. Aus diesem Grund ist sie bereits weit verbreitet, sodass Entwicklungen auf Basis der Kinect von vielen Menschen genutzt werden können.

Schwächen der Kinect

Fehler bei der Berechnung des Tiefenbildes Die Berechnung des Tiefenbildes kann durch externe Infrarotlichtquellen gestört werden. Eine Verwendung im Freien ist durch die Infrarotlichtanteile des Sonnenlichts nicht möglich. Schmale Gegenstände, die direkt auf die Kamera zeigen, und spiegelnde Flächen führen ebenfalls zu falschen oder fehlenden Tiefeninformationen.

Sichtbereich der Kinect-Kamera Der Sichtbereich der Kinect-Kamera ist eingeschränkt. Als Folge dessen kann eine Person mit nach oben ausgestrecktem Arm sich nicht beliebig der Kamera nähern, wenn ihr komplettes Skelett erkannt werden soll. Zusätzlich gelten für die Berechnung des Tiefenbildes in z-Richtung Einschränkungen des SDKs. Das Tiefenbild enthält lediglich Werte für Objekte, die sich im Abstand von 0,8m bis 4,0m zur Kinect-Kamera befinden.

Einschränkungen des Skeletttrackings Das Tracking des SDKs funktioniert nur für den gesamten Körper. Für viele Anwendungen würde es jedoch ausreichen, ausschließlich die Arme oder die Hände zu verfolgen. Die Skelettdaten enthalten keine Informationen über die Finger. Handgestenerkennung ist somit nicht möglich.

Keine kommerzielle Nutzung Die Lizenz des SDKs erlaubt lediglich nichtkommerzielle Nutzung.

Betriebssystembeschränkung Eine Verwendung des SDKs ist nur bei Nutzung von Windows 7 möglich. Windows XP oder microsoftfremde Betriebssysteme werden nicht unterstützt.

5. ViSAR - Visualisierung geometrischer SAR-Effekte

Im Folgenden wird das Programm ViSAR vorgestellt und ein Konzept zur Steuerung von ViSAR mit Gesten beschrieben. Hierzu werden Gesten ausgewählt, um die Funktionen von ViSAR mit Hilfe von Bewegungen der Arme im Raum steuern zu können.

5.1 Funktionen und Einsatzgebiete von ViSAR

Das am Fraunhofer IOSB entwickelte Programm ViSAR dient zur Visualisierung von geometrischen Radar-Effekten. Bei der Verwendung eines Radars mit synthetischer Apertur (eng. Synthetic Aperture Radar; Abkürzung SAR) treten spezifische Effekte, wie zum Beispiel Layover, Foreshortening und Radarschatten auf. Diese erschweren das Interpretieren eines Radarbildes. ViSAR wird zum Erlernen oder Verbessern von Fähigkeiten zum Lesen von Radarbildern eingesetzt.

Basis des Programms bildet eine 3D-Szene. Diese setzt sich aus einem oder mehreren 3D Modellen von Gebäuden oder Gegenständen zusammen, die zum Beispiel mit Google Sketchup modelliert werden. Der Benutzer hat die Möglichkeit, beliebige Modelle zu kombinieren und die 3D-Szene nach seinen Wünschen zu gestalten. Hierzu stehen Funktionen zur Verfügung, mit denen die 3D Objekte in der Szene modifiziert werden können. In die Szene können 3D-Objekte eingefügt, verschoben und gedreht werden. Zusätzlich kann die Höhe, Breite und Tiefe angepasst werden.

Die Ansicht lässt sich als Blick auf eine Szene beschreiben, die sich innerhalb einer Glaskugel befindet. Zur Ausrichtung der Ansicht stehen die Funktionen Kreisen, Wischen und Skalieren zur Verfügung. Durch die Funktion Kreisen werden Aspekt- und Depressionswinkel angepasst. Dies bewirkt eine Rotation der Ansicht auf die Szene. Die Funktion Skalieren ermöglicht die Anpassung der Zoomstufe. Zusätzlich kann der Mittelpunkt der Szene und damit der Rotationspunkt mit Hilfe der Wischen-Funktion verschoben werden. Oft benötigte Ansichten (siehe Tabelle 5.1) sind vordefiniert und können über Buttons ausgewählt werden.

Die aktuelle Ansicht entspricht dem Sichtwinkel des virtuellen Radarsensors, der zur Simulation verwendet wird. Vor dem Start der Simulation können unterschiedliche Effekte ausgewählt werden, die simuliert werden sollen. Hierbei stehen Speckle, Lambert und Streuung zur Verfügung, die mit Hilfe eines Parameters ausgewählt und konfiguriert werden können.

Ansichtsoption	Aspektwinkel	Depressionswinkel
Schräg	-30°	45°
Aufsicht	0°	90°
von rechts	90°	0°
Vorderansicht	0°	0°
Rückseite	180°	0°
von links	-90°	0°

Tabelle 5.1: Überblick über oft benötigte Ansichtswinkel in ViSAR

Wird Speckle ausgewählt, werden die geometrischen Radareffekte Schatten und Layover mit dem im SAR typischen Speckle-Rauschen berechnet. Die Einstellung Lambert sorgt für eine Berechnung der Radareffekte aus den materialabhängigen Reflektionseigenschaften der Oberflächen. Nach Ausführung der Simulation erscheint, wie in Abbildung 5.1 zu sehen, das simulierte, zweidimensionale Radarbild in der Grundebene der 3D Szene.

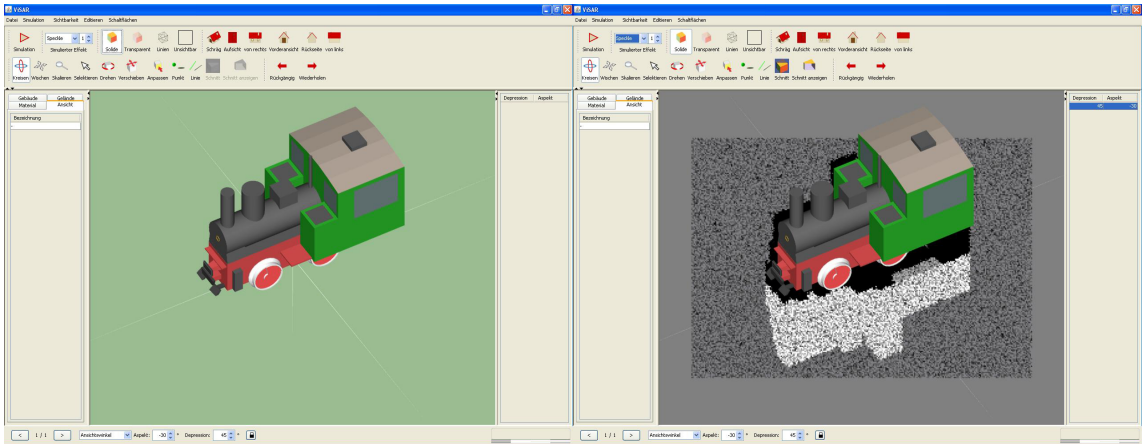


Abbildung 5.1: Screenshots von ViSAR vor und nach der Simulation. Das Radarbild erscheint in der Grundebene der 3D Szene.

Um eine bessere Sicht auf das Radarbild zu erhalten, können die Sichtbarkeiten der 3D-Objekte angepasst werden. Neben der Standardeinstellung Solide stehen Transparent, Linien und Unsichtbar zur Verfügung (Vergleich Abbildung 5.2).

Die Erklärungskomponenten Punkt und Linie dienen der Analyse der Radeffekte. Das Punkt-Werkzeug visualisiert den Radarstrahl, den Layover-Strahl und den Schattenstrahl. Das Linien-Werkzeug projiziert Objektkanten ins Layover und in den Schatten. Das Schnitt-Werkzeug dient der Analyse der Strahlen in einer Ebene. Hierzu wird eine Schnittebene, die parallel zu den Radarstrahlen und senkrecht zur Grundebene liegt, verschoben. Nach Ausführung des Schnittes wird die 3D-Szene in zwei Teile aufgeteilt, wobei ein Teil die 3D-Objekte anzeigt, während der andere Teil einen freien Blick auf das Radarbild ermöglicht. Zur Erhöhung der Benutzerfreundlichkeit können Aktionen rückgängig gemacht und wiederholt werden.

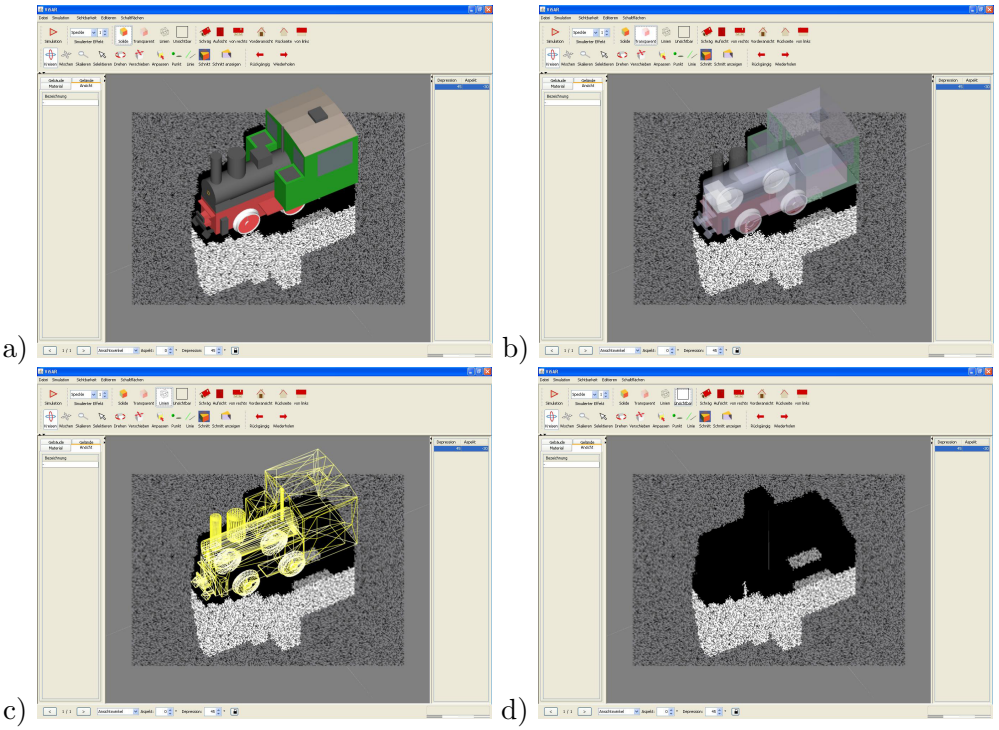


Abbildung 5.2: Mögliche Einstellungen für die Sichtbarkeiten der 3D Objekte: a) Solide, b) Transparent, c) Linien und d) Unsichtbar

5.2 Konzept für die Steuerung von ViSAR mit Gesten

Im Folgenden wird ein Konzept zur Steuerung von ViSAR mit Gesten vorgestellt. Hierbei werden die einzelnen Interaktionsmöglichkeiten mit Gesten verknüpft. Die hier verwendeten Gesten beachten die Besonderheiten des Kinect for Windows SDK. Eine Geste kann somit lediglich von der Armkonfiguration und nicht von der Handkonfiguration abhängen.

Übernahme der Rechte zum Steuern von ViSAR

Im Blickfeld der Kamera können sich mehrere Personen befinden. Wenn die Kinect Kamera mehr als eine Person trackt, kann immer nur eine Person die Rechte zum Steuern von ViSAR besitzen. Zur Übernahme der Rechte kann die Person, die aktuell die Steuerung nicht vornehmen darf, diese durch Ausführung der **Winken**-Geste auf sich übertragen.

Einrichten einer Bediensperre

Da ViSAR im Unterricht eingesetzt wird, gibt es Phasen, in denen der Lehrer das Programm nicht bedienen, sondern lediglich den Schülern Informationen auf dem Bildschirm zeigen möchte. Die Gefahr, dass Bewegungen des Lehrers fehlinterpretiert werden wird eleminiert, indem es eine Möglichkeit gibt eine Bediensperre einrichten zu können. Hierzu wird ebenfalls die **Winken**-Geste (Vergleich Abschnitt 3.2) eingesetzt. Wenn die Bediensperre deaktiviert ist, kann sie durch Winken aktiviert werden. Ist sie aktiviert, wird sie dementsprechende deaktiviert. Die Winken-Geste stellt somit die einzige Geste dar, auf die zu jedem Zeitpunkt reagiert wird.

Wischen

Zum Verschieben der Ansicht auf die 3D-Szene wird das Prinzip der Verankerung eingesetzt. Die Verankerung wird aktiviert, sobald der **rechte Arm ausgestreckt** wird. Der Szenenmittelpunkt, der zugleich Rotationszentrum ist, folgt nun solange der Bewegung des Armes, bis dieser zurückgezogen wird.

Kreisen

Zur Anpassung des Aspekt- und Depressionswinkels wird, wie bei der Wischen-Funktion, eine Verankerung verwendet. Diese wird durch das **Ausstrecken des linken Armes** in Richtung Bildschirm aktiviert. Horizontale Bewegungen bestimmen den Aspektwinkel, vertikale Bewegungen den Depressionswinkel. Die Verankerung wird gelöst, indem der Arm zurückgezogen wird.

Skalieren

Skaliert wird mit der **Zwicken- und Spreizen-Geste** (Vergleich Abschnitt 3.8). Hierzu werden rechter und linker Arm nach vorne ausgestreckt und der Abstand der Hände variiert. Um heranzuzoomen bewegt man die Arme voneinander weg. Die entgegengesetzte Richtung bewirkt eine Verkleinerung der Ansicht. Die Zoombewegung stoppt, sobald einer der Arme nicht mehr ausgestreckt ist.

Simulation Ausführen

Nachdem mit Hilfe der Funktionen Kreisen, Wischen und Skalieren der gewünschte Ansichtswinkel eingestellt ist, kann die Simulation gestartet werden. Hierzu führt der Benutzer eine **Klatschbewegung** aus. Zunächst entfernt er die Arme links und rechts vom Körper und bewegt sie vor seinem Körper aufeinander zu.

Schnitt durch die 3D-Szene

Die Schnittebene durch die 3D-Szene wird angezeigt, indem der Benutzer eine **Teilen**-Geste ausführt. Hierzu streckt der Benutzer seinen rechten Arm nach oben aus und bewegt ihn in einer flüssigen Bewegung nach unten, als ob er die Luft vor sich mit dem rechten Arm teilen würde. Während der gesamten Bewegung bleibt der Arm gestreckt. Falls gewünscht, kann der Benutzer die Schnittebene verschieben, indem er den rechten Arm ausstreckt und somit eine Verankerung auslöst. Die Schnittebene folgt nun der horizontalen Bewegung des Armes, bis dieser zurückgezogen wird. Um den Schnitt anzuzeigen beziehungsweise bei sichtbarem Schnitt wieder die gesamte Szene sichtbar zu machen, wird erneut die Teilen-Geste angewendet.

Menü für die Einstellung des zu simulierenden Effekts, der Sichtbarkeit und der festen Ansichtswinkel

Aufgrund der Tatsache, dass es eine Vielzahl an Einstellungsmöglichkeiten für den zu simulierenden Effekt, die Sichtbarkeit der Objekte und voreingestellte Ansichtswinkel gibt, macht es keinen Sinn, jede Einstellung mit einer einzelnen Geste zu verknüpfen. Besser ist die Verwendung eines Menüs, in dem die Einstellungen vorgenommen werden können. Dieses besteht aus einem Hauptmenü und Untermenüs. Zunächst wählt der Benutzer aus, ob er den zu simulierenden Effekt, die Sichtbarkeit oder eine feste Ansichtsoption ändern möchte. Durch Auswahl eines dieser Punke öffnet sich das entsprechende Untermenü, indem die eigentliche Einstellung vorgenommen wird. Um das Hauptmenü zu öffnen, hält der Benutzer eine **45° Pose mit dem linken Arm** für einen bestimmten Zeitraum. Hierzu streckt der Benutzer seinen linken Arm in einem 45° Winkel in der X-Ebene aus und hält diese Pose. Die einzelnen Menüs bestehen aus horizontalen Listen. Der Vorgang zum Aktivieren eines Menüeintrags teilt sich in zwei Teilaufgaben. Zunächst wird ein Menüeintrag selektiert und anschließend die Auswahl aktiviert. Beim Eintritt in eines der Menüs ist ein Eintrag vorselektiert. Durch **Wischen nach links oder rechts** wird der Eintrag, der sich auf der entsprechenden Seite neben der aktuellen Auswahl befindet, markiert. Befindet sich die Auswahl auf dem gewünschten Menüeintrag, wird dieser durch einen **Tap**, ein kurzes

Ausstrecken des rechten Arms in Richtung des Displays, aktiviert. Die Position des Taps spielt dabei keine Rolle, da der selektierte Menüeintrag aktiviert wird. Das Einstellungsmenü kann jederzeit beendet werden, ohne dass neue Einstellungen vorgenommen werden müssen. Zum Abbrechen hält der Benutzer eine **45° Pose mit dem rechten Arm** für einen definierten Zeitraum.

Objektmanipulation in der 3D-Ansicht

Um Objekte in der 3D-Ansicht von ViSAR manipulieren zu können, stehen die Funktionen Objekt drehen, Objekt verschieben und Objekt anpassen zur Verfügung. Zur Auswahl eines der Funktionen bietet sich die Erweiterung des Menüs an, das bereits zum Vornehmen von Einstellungen verwendet wird. Nachdem eine der Funktionen zur Objektmanipulation ausgewählt wurde, muss als nächster Schritt das Objekt, das manipuliert werden soll, selektiert werden. Dies ist möglich, indem der Mauszeiger der linken Hand folgt, solange der linke Arm ausgestreckt ist. Befindet sich der Mauszeiger auf dem gewünschten Objekt, kann mit dem rechten Arm die eigentliche Manipulation durchgeführt werden, indem er ausgestreckt und eine Bewegung in die gewünschte Richtung durchgeführt wird.

Markierungen setzen

In ViSAR können sowohl Punktmarken, als auch Linienmarken gesetzt werden. Auch hier wird zunächst die gewünschte Funktion in einem Menü ausgewählt. Um ein Detail der Szene markieren zu können, ist die Steuerung der Maus nötig. Da der Mauszeiger exakt auf ein Detail der Szene ausgerichtet werden muss, werden hierzu beide Hände verwendet. Die Bewegung der Maus wird mit dem linken Arm und das Setzen der Punkt- oder Linienmarkierung mit dem rechten Arm durchgeführt. Nach dem Ausstrecken des linken Arms nach vorne folgt der Mauszeiger der Bewegung der linken Hand. Damit beim Zurückziehen der Hand keine Verschiebung des Mauszeigers stattfindet, wird durch Ausführen eines Taps mit dem rechten Arm die Markierung gesetzt, während der linke Arm sich weiter in ausgestrecktem Zustand befindet. Anschließend können beide Arme zurückgezogen werden, ohne dass dies Einfluss auf die Markierung nimmt.

Rückgängig und Wiederherstellen

Diese Funktionen werden wie in Abbschnitt 3.3 beschrieben mit der **Schleife nach links** und **Schleife nach links** Geste ausgeführt.

6. Realisierung der Gestensteuerung für ViSAR

Ziel der Arbeit ist eine Implementierung der Gestensteuerung für die grundlegenden Funktionen zur Steuerung der 3D-Ansicht von ViSAR. In den folgenden Abschnitten wird die Umsetzung beschrieben und abschließend kritisch betrachtet.

6.1 Umgesetzte Gesten

Bei der Realisierung der Gestensteuerung für ViSAR wurden die wichtigsten Funktionalitäten, die zum Bedienen von ViSAR nötig sind, umgesetzt:

- Einrichten und Aufheben einer Bediensperre
- Rotation der Ansicht
- Verschieben der Szene
- Skalieren der Szene
- Starten einer Simulation
- Anzeigen eines Schnitts durch die Szene
- Einstellungsmöglichkeiten für die Sichtbarkeit und feste Ansichtswinkel

Die Gestensteuerung reagiert nicht zu jedem Zeitpunkt auf alle Gesten. Der Zustandsautomat in Abbildung 6.1 beschreibt die möglichen Zustände, in denen sich die Gestensteuerung befinden kann. Je nach Zustand werden unterschiedliche Gesten vom System akzeptiert. Nach dem Start des Programms ist zunächst die Bediensperre aktiviert. Die Winken-Geste stellt die einzige Geste dar, die in diesem Zustand eine Änderung bewirkt. Ihre Ausführung führt zum Übergang in den *Wartezustand*, in dem sich das Programm befindet, wenn momentan keine Geste ausgeführt wird. Von diesem Zustand ausgehend kann die Ansicht rotiert, verschoben oder skaliert werden, die Simulation ausgeführt, das Hauptmenü geöffnet oder die Schnittebene angezeigt werden. Der Übergang zu den Zuständen *Kreisen* und *Wischen* geschieht durch das Ausstrecken des linken beziehungsweise rechten Arms. Der Wartezustand wird wieder erreicht, sobald der entsprechende Arm zurückgezogen wird. Werden im Grundzustand beide Arme ausgestreckt, findet ein Übergang zum Zustand *Skalieren* statt. Eine Simulation kann gestartet werden, indem der Benutzer mit beiden Händen klatscht. Das Öffnen des *Hauptmenüs* geschieht durch eine 45° Pose des linken

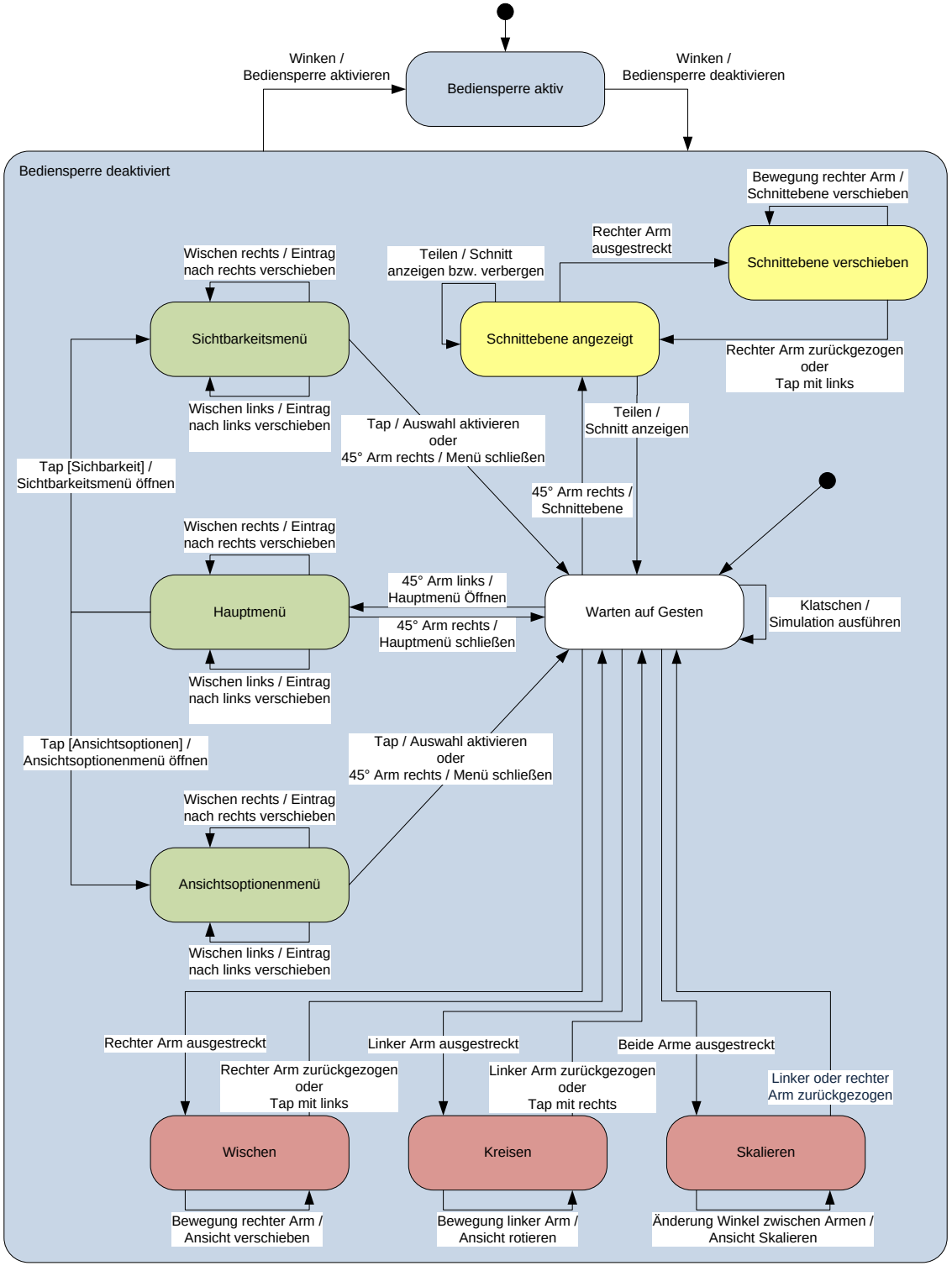


Abbildung 6.1: Das Zustandsdiagramm der Gestensteuerung von ViSAR. Rote Zustände sind Teil der Anpassung der Ansicht, grüne Zustände beschreiben die Navigation durch die Menüs und gelbe Zustände begleiten den Vorgang zum Anzeigen eines Schnittes durch die Szene. Zur Umsetzung der Bediensperre dienen die blauen Zustände.

Arms, die für eine bestimmte Zeit gehalten wird. Sobald das Menü geöffnet ist, werden ausschließlich Gesten akzeptiert, die zur Navigation durch das Menü nötig sind. Hierzu

gehören das Wischen nach links und nach rechts, ein Tap zur Aktivierung des selektierten Eintrags oder die 45° Pose des rechten Arms zum Abbrechen der Menüauswahl. Andere Gesten werden erst wieder interpretiert, wenn das Programm in den Wartezustand zurückkehrt. Die Anzeige eines Schnittes erfolgt mit den beiden Zuständen *Schnittebene angezeigt* und *Schnittebene verschieben*. Befindet sich die Gestensteuerung im Wartezustand, kann durch eine Teilengeste zunächst die Schnittebene durch die Szene angezeigt werden. Durch Ausstrecken des rechten Arms wechselt die Gestensteuerung in den *Schnittebene verschieben* Zustand. Hier kann durch Bewegungen des Arms die Schnittebene verschoben werden. Zurückziehen des Arms beendet den Vorgang der Positionierung der Schnittebene. Angezeigt wird der Schnitt, indem erneut eine Teilen-Geste durchgeführt wird. Wenn momentan ein Schnitt angezeigt wird, kann mit dieser Geste wieder die komplette 3D-Szene angezeigt werden. Um zum *Wartezustand* zurückzukehren, wird die von dem Menü bekannte 45° Pose des rechten Arms zum Abbrechen des Vorgangs angewendet.

Gestenerkennung

Die vom SDK berechneten Skelletdaten bilden die Grundlage zur Erkennung der Gesten. Meistens sind nur ein kleiner Teil der 20 Gelenkpositionen nötig, um eine einzelne Geste zu erkennen. Die Arme stellen dabei die wichtigste Informationsquelle dar.

Im Wesentlichen werden zwei verschiedene Arten von Gesten zum Steuern von ViSAR eingesetzt. Die erste Gruppe bilden Gesten, die eine binäre Antwort liefern. Hierbei kommt es lediglich darauf an, dass eine Geste ausgeführt wurde. Im Gegensatz hierzu gibt es Gesten, deren Ergebnis Paramter darstellen, auf die ViSAR reagiert. Eine Geste mit binärer Antwort wird entweder durch das Halten einer bestimmten Pose oder durch eine Bewegung realisiert. Soll die Geste dem Programm Parameter zur Verfügung stellen, kommt das Prinzip der Verankerung zum Einsatz.

Erkennung einer Pose als Geste

Die Geste zum Öffnen eines Menüs stellt einen typischen Vertreter der Gesten dar, die durch das Halten einer Pose realisiert sind. Um zu erkennen, ob der linke Arm im 45° Winkel zum Körper steht, wird nur die Position der linken Hand, der linken Schulter und der linken Hüfte in der x- und y-Ebene benötigt. Anhand dieser drei Koordinaten wird der Winkel zwischen Arm und Körper berechnet. Die Position der Schulter dient als Scheitelpunkt des Winkels. Da es schwierig ist, den Arm exakt im 45° Winkel zu positionieren, wird eine Toleranzgrenze eingeführt, die eine Abweichung von $\pm$ 5° erlaubt. Hält der Benutzer seinen Arm für einen vordefinierten Zeitabschnitt in dieser Pose, gilt die Geste als ausgeführt.

Erkennung einer Bewegung als Geste

Neben einer Pose kann auch eine Bewegung als eine Geste mit binärer Antwort verwendet werden. Die Bewegung wird als Folge von Zuständen interpretiert. Ein Zustand wird durch das Einnehmen einer bestimmten Pose erreicht, wobei jede Pose einen Toleranzbereich besitzt, sodass die Bewegung auch bei leicht unterschiedlicher Ausführung erkannt wird. Als Beispiel dieser Gattung von Gesten wird die Teilen-Geste erläutert. Diese besteht aus drei Zuständen (Abbildung 6.2). Der erste Zustandsübergang wird ausgelöst, wenn der Benutzer seinen Arm nach oben ausstreckt, sodass er eine virtuelle Ebene über seinem Kopf schneidet, die senkrecht auf der y-Achse steht. Als nächstes wird der Schnitt einer virtuellen Ebene vor dem Körper detektiert, die senkrecht zur z-Achse steht. Zum Abschluss ist der Schnitt einer virtuellen Ebene unter der Hüfte nötig, die wiederum senkrecht zur y-Achse steht. Diese drei Zustände können nur in dieser Reihenfolge durchlaufen werden. Dabei ist für einen Zustandsübergang eine maximale Übergangsdauer definiert. Verfehlt

der Benutzer diese Zeitschranke gilt die Geste als nicht ausgeführt. Die Gesten Wischen nach links, Wischen nach rechts und der Tap werden ebenfalls als Bewegungen erkannt. Hierbei werden entsprechend andere Zustandsfolgen verwendet.

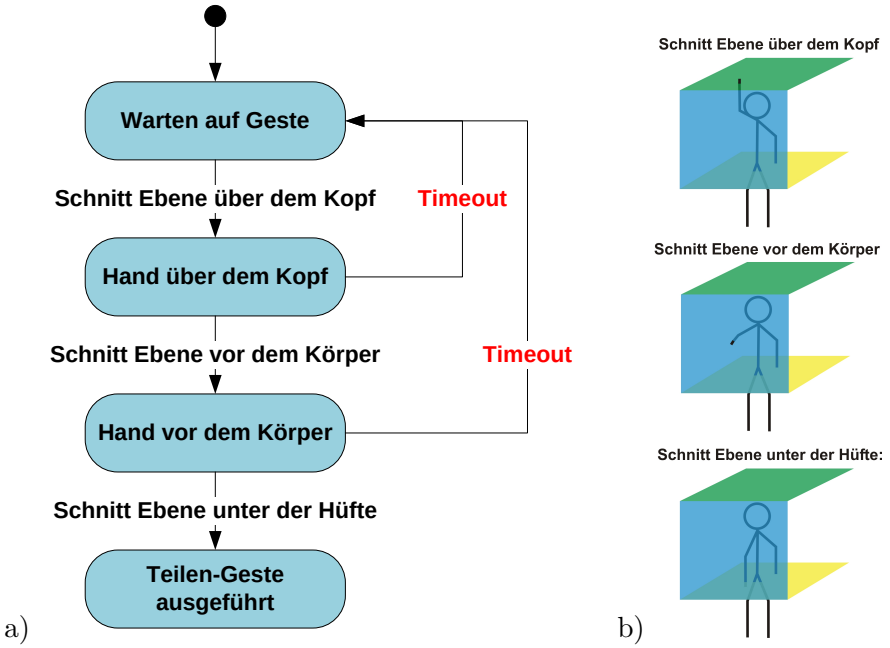


Abbildung 6.2: a) Das Zustandsdiagramm der Teilen-Geste und b) die Posen, die bei der Teilen-Geste zu Zustandsübergängen führen.

Gesten mit dem Prinzip der Verankerung

Bei den Funktionen Kreisen, Wischen, Skalieren und bei der Verschiebung der Schnittebene reicht eine binäre Antwort der Gestenerkennung nicht aus. Es sind Parameter der Bewegung nötig, da die Szene oder die Schnittebene mit Hilfe der Bewegung des Arms ausgerichtet wird. Um dies zu realisieren, ist eine Verankerung nötig. Dieses Prinzip wird mit Hilfe der Verschieben-Geste beschrieben. Zunächst muss die Verankerung aktiviert werden. Dies geschieht, indem der Benutzer seinen Arm in Richtung Display ausstreckt, sodass er eine virtuelle Ebene, die sich vor dem Körper befindet, schneidet. Sobald sich die Hand auf der anderen Seite der Ebene befindet, folgt die Szene der Bewegung der Hand. Die Verankerung wird gelöst sobald sich der Arm nicht mehr in ausgestrecktem Zustand befindet. Dies wird erkannt indem eine virtuelle Kugel um das Schultergelenk gelegt wird. Die Verankerung bleibt solange aktiv, bis sich die Hand innerhalb dieser Kugel befindet. Der Radius der Kugel entspricht der Entfernung der virtuellen Ebene, die beim Aktivieren der Verankerung eingesetzt wird.

Beim Lösen der Verankerung kann es zu einem unerwünschten Effekt kommen. Um die Verankerung zu entfernen, muss die Hand zurückgezogen werden. Während dieser Bewegung des Zurückziehens ist die Verankerung noch aktiv, sodass die Ansicht weiterhin angepasst wird. Sie stoppt erst, wenn sich die Hand vollständig innerhalb der virtuellen Kugel befindet. Dieser Effekt lässt sich umgehen, indem eine weitere Möglichkeit hinzugefügt wird, mit der die Verankerung gelöst werden kann. Hierzu wird mit dem linken Arm ein Tap ausgeführt, während die rechte Hand die Szene in der gewünschten Position hält. Anschließend können beide Arme zurückgezogen werden, ohne dass die Szene verschoben wird.

Positionierung der virtuellen Schnittebenen und Schnittkugeln

Eine wesentliche Voraussetzung für die Anwendbarkeit der Gestensteuerung für ViSAR ist eine Gestenerkennung, die unabhängig von äußeren Merkmalen der Person arbeitet. Das Kinect for Windows SDK liefert bereits eine Skelettberechnung, die unabhängig vom Aussehen der Person funktioniert. Je nach Körpergröße besitzen die einzelnen Segmente des Skeletts jedoch unterschiedliche Längen. Aus diesem Grund werden die virtuellen Schnittebenen und -kugeln in Abhängigkeit der Person positioniert. Hierzu wird die Länge des Arms berechnet, die der Benutzer erreichen kann, wenn der Arm maximal ausgestreckt ist. Zu diesem Zweck werden die Längen der einzelnen Armsegmente addiert. Dazu gehören die Abstände Hand-Handgelenk, Handgelenk-Ellbogen und Ellbogen-Schulter.

Zur Detektion, ob ein Arm in eine bestimmte Richtung ausgestreckt wird, verwendet man eine virtuelle Ebene, die senkrecht auf der Richtung steht und dessen Abstand zur Schulter ein proportionaler Anteil der berechneten maximalen Armlänge ist. Soll ein ausgestreckter Arm ohne vorgegebene Richtung erkannt werden, wird an Stelle einer Ebene eine virtuelle Kugel mit der Schulter als Zentrum festegelegt. Der Radius ist wiederum ein proportionaler Anteil der maximalen Armlänge.

Visuelles Feedback

Dem Anwender wird eine Statusanzeige zur Verfügung gestellt, die über dem ViSAR Fenster liegt und somit dauerhaft sichtbar ist. Sie dient dazu, dem Nutzer die Bedienung zu erleichtern, indem auf erkannte Gesten ein positives Feedback folgt. Neben der Information, ob die Bediensperre aktiv oder inaktiv ist, wird dem Nutzer nach dem Erkennen einer Geste die erfolgreiche Ausführung mit einem Bild visualisiert.

6.2 Anbindung der Kinect-Steuerung an ViSAR

Um die Kinect-Steuerung nicht nur auf ViSAR zu beschränken, wird bei der Anbindung der Gestensteuerung an ViSAR ein generischer Ansatz verwendet. Erkannte Gesten lösen das virtuelle Drücken von Tasten und Mausbewegungen aus, die mit Hilfe der user32.dll Bibliothek aus dem Windows Application Programming Interface (Windows-API), an das Fenster gesendet werden, auf das momentan der Fokus liegt. ViSAR wurde um entsprechende Hotkey-Funktionen erweitert, sodass es auf empfangene Tastenkombinationen reagiert. Der Benutzer erhält beim Start der Kinect-Steuerung die Möglichkeit, das geöffnete ViSAR-Fenster in einer Liste aller geöffneten Programme auszuwählen (Abbildung 6.3), sodass anschließend der Fokus auf dieses Fenster automatisch gesetzt werden kann.

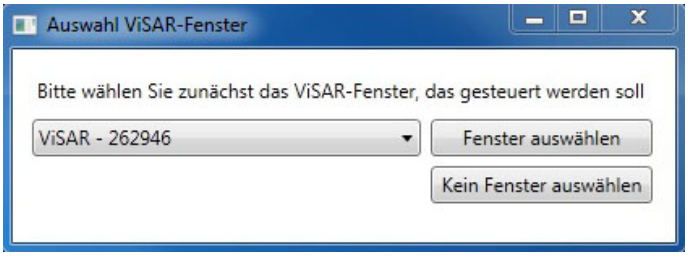


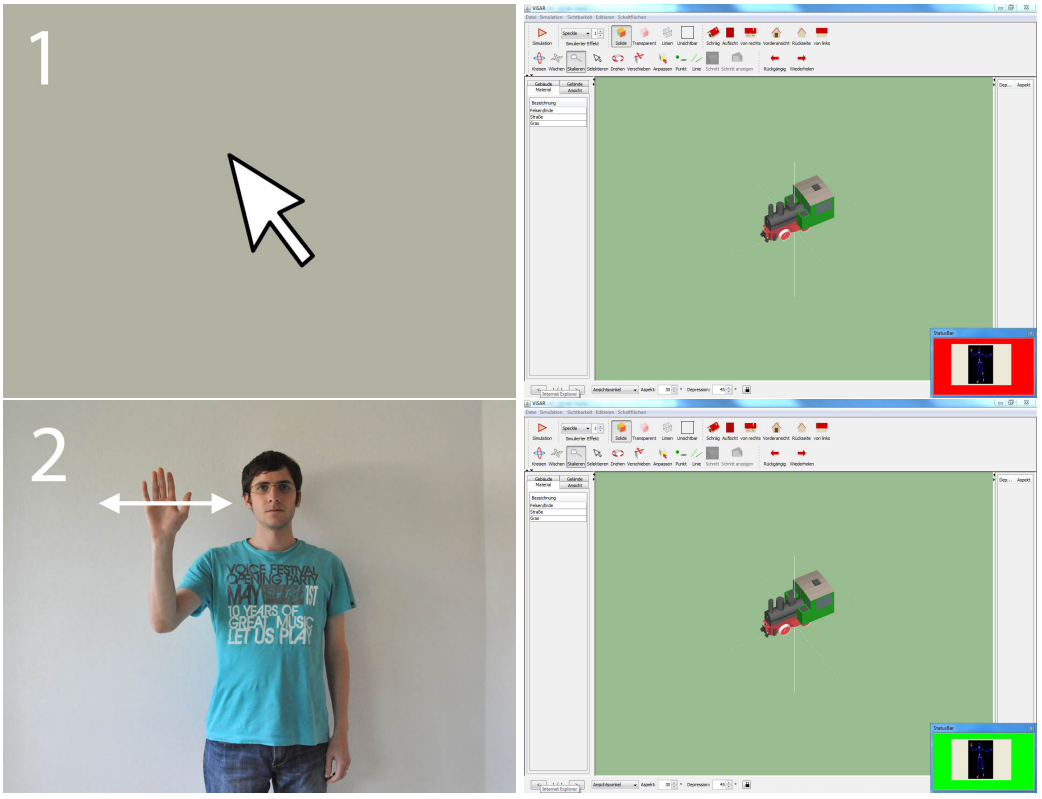
Abbildung 6.3: Das ViSAR-Fenster, das gesteuert werden soll, wird beim Start der Gestensteuerung ausgewählt.

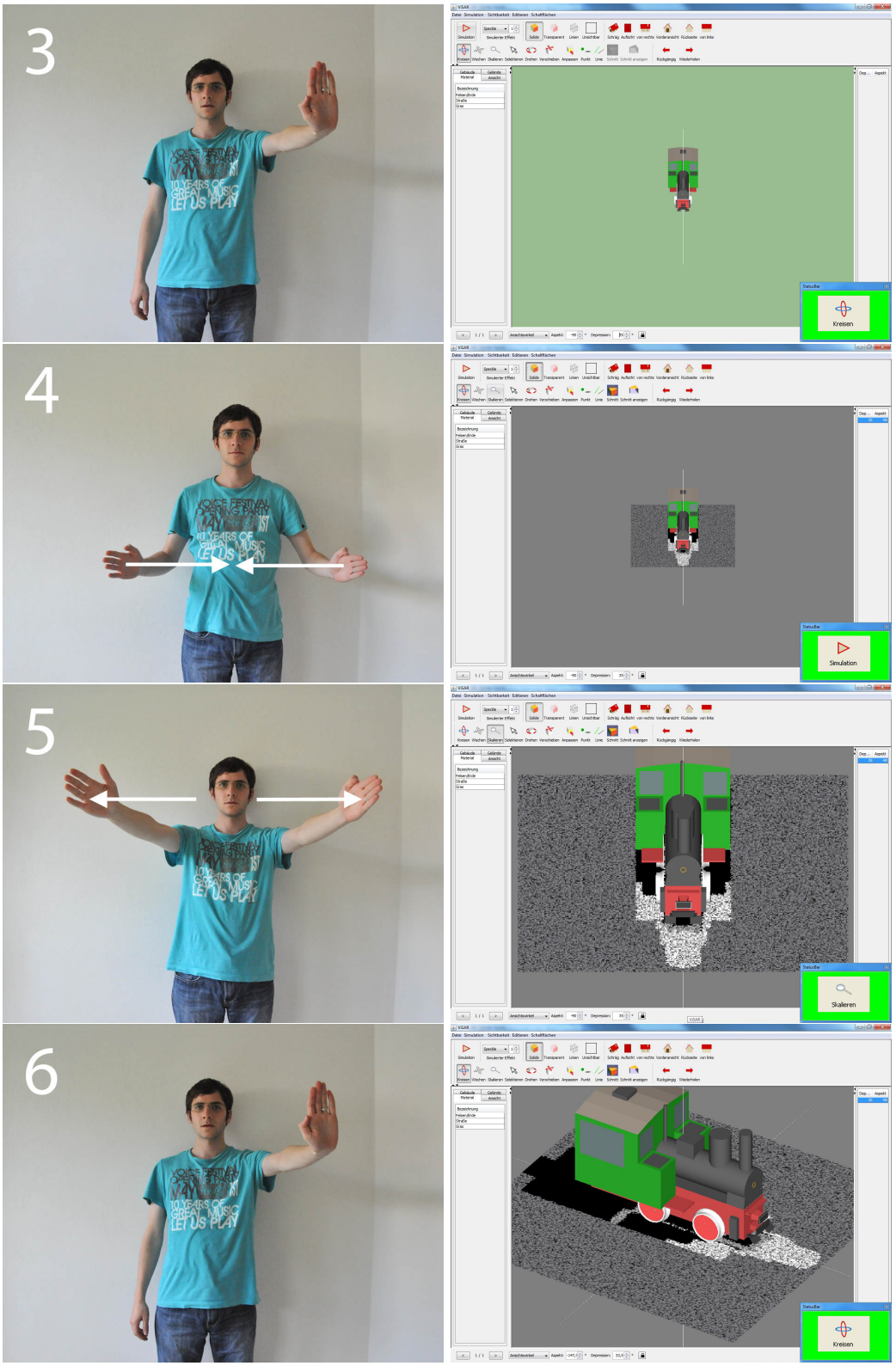
6.3 Experimente

Anwendungsfall

In Abbildung 6.4 wird ein typischer Anwendungsfall für die Steuerung von ViSAR mit Gesten beschrieben. Folgende Gesten werden hierzu nacheinander ausgeführt:

1. Nach dem Start von ViSAR lädt der Anwender die gewünschte 3D-Szene per Maus.
2. Der Benutzer deaktiviert die Bediensperre, indem er eine Winken-Geste durchführt.
3. Die Ansicht der Szene wird mit der Kreisen-Geste gedreht, bis der gewünschte Sichtwinkel des virtuellen Radarsensors erreicht ist.
4. Das simulierte Radarbild wird durch das Ausführen einer Klatsch-Bewegung erzeugt.
5. Beide Arme werden ausgestreckt und voneinander weg bewegt um heranzuzoomen.
6. Erneutes Anwenden der Kreisen-Geste führt zum Ausrichten der Ansicht.
7. Das Menü wird mit der 45° Pose des linken Arms geöffnet.
8. Im Hauptmenü wird das Menü mit den Sichtbarkeitsoptionen mit einer Tap-Geste geöffnet.
9. Durch dreifaches Wischen nach rechts wird die Option *Unsichtbar* markiert.
10. Nach dem Aktivieren des ausgewählten Eintrags mit der Tap-Geste wird die 3D-Szene unsichtbar, sodass ein freier Blick auf das simulierte Radarbild möglich ist.




3
Kreisen
4
Simulation
5
Skalieren
6
Kreisen

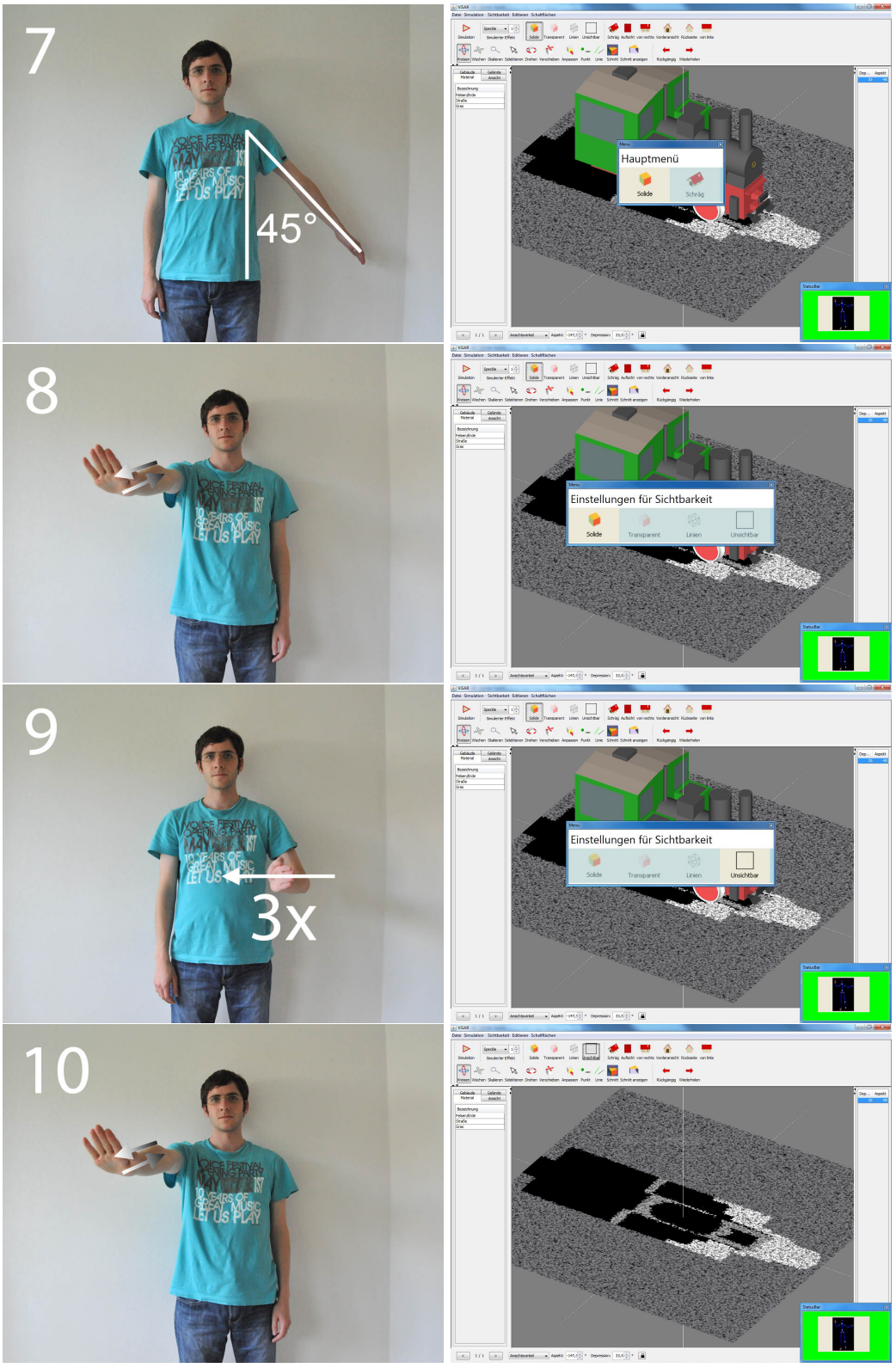


Abbildung 6.4: Ein typischer Anwendungsfall: Gesten und deren Effekt auf ViSAR

Prozessorauslastung

Zur Untersuchung der Rechenintensivität wurden Tests auf zwei verschiedenen PCs durchgeführt. Um unterscheiden zu können, welche der drei Komponenten Skelettberechnung, Gestenerkennung und ViSAR die größte Latenz mit sich bringt, wurde die Prozessorauslastung bei der Vewendung folgender Teilelemente der Gestensteuerung betrachtet.

- Test 1: Skelettberechnung des SDK
- Test 2: Skelettberechnung des SDK und Gestenerkennung ohne ViSAR
- Test 3: Skelettberechnung des SDK und Gestenerkennung mit ViSAR

Der Aufbau der Versuchs-PCs und die Ergebnisse sind in Tabelle 6.3 zu sehen.

	PC 1	PC 2
Prozessor	Intel Core2 6320 @ 1,86GHz	Intel Core i7 @ 2,67GHz
Systemtyp	Windows 7 64 Bit	Windows 7 64 Bit
Arbeitsspeicher	8 GB	8 GB
Grafikkarte	NVIDIA GeForce 7300 LE	NVIDIA NVS 3100M
Prozessorauslastung		
Test 1	60-70 %	18-22 %
Test 2	70-80 %	25-27 %
Test 3	85-100 %	30-65 %

Tabelle 6.1: Systeminformationen der PCs, die zur Auswertung der Rechenintensivität verwendet wurden und die Ergebnisse der Untersuchung

Diskussion

Zur Bewertung der Gestensteuerung wird im Folgenden die Implementierung mit den elementaren Anforderungen an ein Gesten-Interface aus Abschnitt 2.4 verglichen. Hierzu wurde die Implementierung von fünf Testpersonen unterschiedlicher Größe getestet. Die Körperlänge der größten Testperson beläuft sich auf 1,93m und bei der Kleinsten auf 1,53m. Auf Grund unterschiedlicher Körpergrößen ergeben sich auch unterschiedliche Armlängen. Zur Analyse der Gestenerkennung wurde der Anwendungsfall aus Abschnitt 6.3 mehrfach von den Testpersonen ausgeführt.

Die Tests haben ergeben, dass die Auswahl an Gesten sowohl selbsterklärende Gesten, als auch Gesten, die bei der ersten Ausführung ohne Anleitung nicht ausführbar sind, beinhaltet. Zu den intuitiven Gesten gehören unter anderem die Skalieren- und die Wischen-Geste. Im Gegensatz hierzu steht zum Beispiel das Lösen der Verankerung mit dem zweiten Arm bei der Verschieben- und Kreisen-Geste oder die Geste zum Öffnen des Menüs. Diese Gesten konnten von den Testpersonen ohne Erläuterung und Lernphase nicht ausgeführt werden.

Die Erfolgsrate der Gestenerkennung lag, nachdem die Testpersonen die korrekte Ausführung der Gesten erlernt haben, auf einem gleich hohen Niveau. Die Gestenerkennung funktionierte bei den Testpersonen personeninvariant. Die Positionierung der virtuellen Schnittebenen in Abhängigkeit der Armlänge des Benutzers sorgt für eine Gestenerkennung, die unabhängig von der Größe des Benutzers funktioniert.

Die Experimente zeigen darüber hinaus, dass kleine Testpersonen näher an die Kinect-Kamera herantreten können. Die Beschränkungen des vertikalen Öffnungswinkels wirken

sich hierbei nicht so stark aus wie bei großen Personen. Der Körper der kleinen Testpersonen passt auch dann noch ins Bild der Kamera wenn sie sich näher an der Kinect-Kamera befinden, auch bei nach oben ausgestrecktem Arm.

Ein weiteres Ergebnis der Tests ist, dass die Bedienung von ViSAR mit Gesten über einen längeren Zeitraum hinweg im Gegensatz zur Bedienung mit Maus und Tastatur zu körperlichen Anstrengungen führt. Es können keine Gesten verwendet werden, die mit den Fingern ausgeführt werden, da die Skelettberechnung des SDKs hierfür keine Informationen zur Verfügung stellt. Aus diesem Grund kommen Gesten zum Einsatz, bei denen die ganzen Arme bewegt werden müssen und nicht nur die Finger. Vor allem die Skalieren-Geste, bei der beide Arme in Aktion treten, ist bei ihrer Ausführung auf Dauer anstrengend. Gleiche Gesten, die mehrfach hintereinander ausgeführt werden, führen ebenfalls zu einer Belastung der körperlichen Verfassung des Anwenders. Ein Beispiel hierfür stellt das Wischen zur Navigation durch das Menü dar, das möglicherweise sechs Mal hintereinander verwendet werden muss.

Die Betrachtung der Prozessorauslastung der einzelnen Komponenten führt zu dem Ergebnis, dass die Skelettberechnung des SDKs sehr rechenaufwändig ist. Sie benötigt im Vergleich zur Gestenerkennung mehr Rechenleistung. Auf Systemen mit geringer Leistung kann es passieren, dass sich die Gestensteuerung und ViSAR gegenseitig behindern, da hier die Prozessorauslastung auf 100% steigt. Dies führt zu Verzögerungen bei der Anpassung der Ansicht von ViSAR und einer erhöhten Latenz bei der Berechnung des Skeletts, sodass die Gestenerkennung teilweise fehlschlägt. In diesem Fall ist die Steuerung von ViSAR mit Gesten für den Anwender unbefriedigend. Das visuelle Feedback in Form einer Statusanzeige erleichtert dem Anwender die Bedienung bei hoher Latenz. Er erfährt in Echtzeit, ob eine Geste richtig ausgeführt und erkannt wurde, auch wenn ViSAR auf Grund von einer beschränkten Rechenleistung des PC-Systems mit der Anpassung der Darstellung hinterherhinkt.

7. Zusammenfassung und Ausblick

Im Rahmen dieser Studienarbeit wurden zunächst Gesten definiert und klassifziert sowie Anforderungen, die an ein Gesten-Interface gestellt werden, aufgezeigt. Eine Auswahl an Freiform-Gesten für die wichtigsten Interaktionsmöglichkeiten eines Systems wurde vorgestellt. Ein Großteil der Gesten ist bereits von der Verwendung auf Touchscreens bekannt und konnte zur Verwendung auf Gesten-Interfaces, die mit Freiform-Gesten arbeiten, übertragen werden. Die Kinect-Kamera von Microsoft stellt eine Möglichkeit zum Erkennen menschlicher Bewegung dar. Neben der Hardwareausstattung wurde Microsofts Ansatz zur Berechnung der Körperpose anhand des Tiefenbildes der Kinect-Kamera beschrieben. Das Kinect for Windows SDK nutzt diesen Algorithmus, um die Positionsdaten von 20 Gelenken von bis zu zwei Personen, die sich im Blickfeld der Kamera befinden, zu berechnen. Für das 3D-Simulationsprogramm ViSAR vom Fraunhofer IOSB wurden Gesten identifiziert und ein Bedienkonzept für die Steuerung von ViSAR mit Gesten erstellt. Bei der Auswahl der Gesten wurde beachtet, dass sie einerseits sinnvoll mit den Funktionen von ViSAR verknüpft und einfach auszuführen sind, anderseits mit Hilfe der vom SDK berechneten Skelttdaten robust erkannt werden können. Ein Teil des Bedienkonzeptes für ViSAR wurde implementiert. Neben statischen Posen werden Bewegungen und Gesten mit Verankerung erkannt. Hierzu kommt ein Ansatz mit virtuellen Schnittebenen und -kugeln zum Einsatz. Die Positionierung der Schnittebenen geschieht in Abhängigkeit der Armlänge des Benutzers. Die Gestensteuerung wird mit Hilfe der Windows-API an ViSAR angebunden.

Für einen Großteil der Interaktionsmöglichkeiten konnten intuitive Gesten gefunden werden. Andere Gesten stehen dagegen in keinem direkten Zusammenhang mit ihrer Funktion. Tests der Gestensteuerung haben ergeben, dass das Ausführen der Gesten über einen längeren Zeitraum zu körperlichen Beschwerden führen kann. Bei den Testpersonen unterschiedlicher Größe funktionierte die Gestenerkennung personeninvariant. Der generische Ansatz bei der Anbindung der Gestensteuerung an ViSAR ermöglicht eine Verwendung für weitere Programme. Durch Anpassung der Hotkeys ist zum Beispiel die Steuerung der 3D-Ansicht von Google SketchUp möglich.

Ausblick

Im Moment gibt es nur wenige standardisierte Gesten. In der Zukunft werden immer mehr Freiform-Gesten-Interfaces entstehen, die eine Menge neuer Gesten hervorbringen werden. Die Entwicklung einer einheitlichen Gestensprache ist für die Akzeptanz zukünftiger Gesten-Interfaces von elementarer Bedeutung.

Bei der Entwicklung eines Gesten-Interface für ein bereits bestehendes Programm, werden lediglich die Funktionen, die bisher von Maus und Tastatur bedient werden, mit Gesten verknüpft. Wenn während der Konzeption und Entwicklung eines Programms bereits auf die Bedürfnisse einer Gestensteuerung eingegangen wird, können Gesten auf einem tieferen Level mit Programmfunktionen verknüpft werden. Hierdurch wird es möglich Funktionen zu erstellen, die eine höhere Anzahl an Freiheitsgraden des Programms gleichzeitig verändern, als es die Simulation von Mausbewegungen und Hotkeys zulässt.

In dieser Arbeit wurde ein einfacher geometrischer Ansatz zum Erkennen der Gesten gewählt. Hiermit kann nicht jede Art von Geste erkannt werden. Eine Gestenerkennung mit einem stochastischen Modell, wie dem Hidden Markov Model, erweitert Möglichkeit bei der Erkennung von Gesten. Zum Beispiel können neue Gesten eingelernt werden, indem man sie vorführt.

Das Skelett des SDKs deckt nicht alle Freiheitsgrade des menschlichen Körpers ab. Es fehlen zum Beispiel die Finger, die für die Ausführung intuitiver Gesten eine wichtige Rolle spielen. Der Mensch nutzt seine Finger im Alltag für eine große Zahl an Gesten. Der Zugriff auf das Tiefenbild der Kinect-Kamera gibt einem die Möglichkeit, mit eigenen Bildverarbeitungsalgorithmen zusätzliche Gelenke im Bild zu erkennen. Das Skelett kann somit um die Positionsdaten der Finger erweitert werden, sodass zukünftig Gesten in Abhängigkeit der Handkonfiguration erkannt werden können. Am Fraunhofer IOSB wird in der Abteilung Interaktive Analyse und Diagnose bereits an der Erkennung der Finger gearbeitet [42].

Gestensteuerungen beschreiben lediglich einen Bereich der Natural-User-Interfaces. Um die Bedienung noch intuitiver zu gestalten, kann Spracherkennung verwendet werden, auch in Kombination mit einem Gesten-Interface. Die Kinect-Kamera mit ihrem Array aus Mikrofonen und die Audio-Verarbeitungsmöglichkeiten des SDKs bringen bereits die grundlegenden Vorraussetzungen mit, um das Gesten-Interface um Spracheingaben zu erweitern. Bei der in dieser Arbeit vorgestellten Gestensteuerung für ViSAR bietet es sich zum Beispiel an, die Einstellungsmöglichkeiten, die bisher mit einer Folge von Gesten ausgewählt werden, zusätzlich mit gesprochenen Befehlen auswählen zu können.

Abbildungsverzeichnis

Literaturverzeichnis

[1] T. Baudel and M. Beaudouin-Lafon, "Charade: remote control of objects using free-hand gestures," *Commun. ACM*, vol. 36, pp. 28–35, July 1993.

[2] A. Agarwal and B. Triggs, "Recovering 3D human pose from monocular images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 44–58, Jan. 2006.

[3] D. Demirdjian and T. Darrell, "3-D articulated pose tracking for untethered diectic reference," *Multimodal Interfaces, IEEE International Conference on*, vol. 0, pp. 267+, 2002.

[4] S. Knoop, S. Vacek, and R. Dillmann, "Sensor fusion for 3d human body tracking with an articulated 3d body model," *Proceedings 2006 IEEE International Conference on Robotics and Automation 2006 ICRA 2006*, no. May, pp. 1686–1691, 2006.

[5] Zhu and Fujimura, "A bayesian framework for human body pose tracking from depth image sequences," *Sensors*, vol. 10, pp. 5280–5293, May 2010.

[6] S. E. Ghobadi, O. E. Loepprich, K. Hartmann, and O. Loffeld, "Hand segmentation using 2d/3d images," *IVCNZ 2007*, no. December, pp. 64– 69, 2007.

[7] V. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for Human-Computer interaction: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 677–695, 1997.

[8] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb. 1989.

[9] T. Starner and A. Pentland, "Real-Time american sign language recognition from video using hidden markov models," in *SCV95*, 1995.

[10] S.-W. Lee, "Automatic gesture recognition for intelligent human-robot interaction," in *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on*, pp. 645 –650, april 2006.

[11] S. Fujie, Y. Ejiri, K. Nakajima, Y. Matsusaka, and T. Kobayashi, "A conversation robot using head gesture recognition as para-linguistic information," in *Robot and Human Interactive Communication, 2004. ROMAN 2004. 13th IEEE International Workshop on*, pp. 159 – 164, sept. 2004.

[12] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, pp. 1521 – 1527, 2006.

[13] D. Bannach, O. Amft, K. Kunze, E. Heinz, G. Troster, and P. Lukowicz, "Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 32 –39, april 2007.

[14] A. Schick, F. van de Camp, J. Ijsselmuiden, and R. Stiefelhagen, "Extending touch: towards interaction with large-scale surfaces," in *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, (New York, NY, USA), pp. 117–124, ACM, 2009.

[15] R. Johnson, K. O'Hara, A. Sellen, C. Cousins, and A. Criminisi, "Exploring the potential for touchless interaction in image-guided interventional radiology," in *Proceedings of the 29th International Conference on Human Factors in Computing Systems*, April?Spring 2011.

[16] K. Nickel and R. Stiefelhagen, "Visual recognition of pointing gestures for human-robot interaction," *Image Vision Comput.*, vol. 25, pp. 1875–1884, December 2007.

[17] Prime Sense, "OpenNI." `http://www.openni.org/`. zuletzt abgerufen am 18.07.2011.

[18] Prime Sense, "NITE Middleware." `http://www.primesense.com/?p=515`. zuletzt abgerufen am 18.07.2011.

[19] E. A. Suma, B. Lange, A. Rizzo, D. Krum, and M. Bolas, "FAAST: the flexible action and articulated skeleton toolkit," in *IEEE Virtual Reality*, pp. 247–248, 2011.

[20] Wikipedia, "Natural user interface — wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Natural_user_interface&oldid=434138507`, 2011. zuletzt abgerufen am 13.06.2011.

[21] D. Saffer, *Designing Gestural Interfaces: Touchscreens and Interactive Devices*. O'Reilly Media, 1 ed., Nov. 2008.

[22] K. Razum and R. Osterwinter, *Duden - Deutsches Universalwörterbuch*. Library Information Portal, Mannheim: Brockhaus Duden Neue Medien, 6., überarb. und erw. aufl. ed., 2007.

[23] F. K. H. Quek, "Toward a vision-based hand gesture interface," in *Proceedings of the conference on Virtual reality software and technology*, (River Edge, NJ, USA), pp. 17–31, World Scientific Publishing Co., Inc., 1994.

[24] Y. Wu and T. S. Huang, "Vision-based gesture recognition: A review," in *Proceedings of the International Gesture Workshop on Gesture-Based Communication in Human-Computer Interaction*, GW '99, (London, UK), pp. 103–115, Springer-Verlag, 1999.

[25] J. P. Wachs, M. Kölsch, H. Stern, and Y. Edan, "Vision-based hand-gesture applications," *Commun. ACM*, vol. 54, pp. 60–71, February 2011.

[26] K. Sung, "Recent videogame console technologies," *Computer*, vol. 44, pp. 91 –93, feb. 2011.

[27] W. T. Freeman and C. Weissman, "Television control by hand gestures," in *IEEE International Conference on Automatic Face and Gesture Recognition*, 1995.

[28] J. P. Wachs, H. I. Stern, Y. Edan, M. Gillam, J. Handler, C. Feied, and M. Smith, "A gesture-based tool for sterile browsing of radiology images.," *Journal of the American Medical Informatics Association : JAMIA*, vol. 15, no. 3, pp. 321–323, 2008.

[29] Y. Kuno, T. Murashina, N. Shimada, and Y. Shirai, "Intelligent wheelchair remotely controlled by interactive gestures," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 4, pp. 672 –675 vol.4, 2000.

[30] I. Rauschert, P. Agrawal, R. Sharma, S. Fuhrmann, I. Brewer, and A. MacEachren, "Designing a human-centered, multimodal GIS interface to support emergency management," in *GIS '02: Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, (New York, NY, USA), pp. 119–124, ACM Press, 2002.

[31] D. Kortenkamp, E. Huber, and R. P. Bonasso, "Recognizing and interpreting gestures on a mobile robot," in *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, AAAI'96, pp. 915–921, AAAI Press, 1996.

[32] O. Rogalla, M. Ehrenmann, R. Zollner, R. Becher, and R. Dillmann, "Using gesture and speech control for commanding a robot assistant," in *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pp. 454 – 459, 2002.

[33] M. Hasanuzzaman, V. Ampornaramveth, T. Zhang, M. Bhuiyan, Y. Shirai, and H. Ueno, "Real-time vision-based gesture recognition for human robot interaction," in *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pp. 413 –418, aug. 2004.

[34] R. Dorau, *Emotionales Interaktionsdesign Gesten und Mimik interaktiver Systeme*. Springer-Verlag Berlin Heidelberg, 2011.

[35] Microsoft XBOX Support: Kinect Themen, "Verwenden von Gesten." `http://support.xbox.com/de-de/Pages/kinect/body-controller/default.aspx`. zuletzt abgerufen am 14.06.2011.

[36] C. Harrison and A. K. Dey, "Lean and zoom: proximity-aware user interface and content magnification," in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, (New York, NY, USA), pp. 507–510, ACM, 2008.

[37] M. S. Hancock, F. D. Vernier, D. Wigdor, S. Carpendale, and C. Shen, "Rotation and translation mechanisms for tabletop interaction," in *Proc. of Tabletop 2006*, pp. 79–86, 2006.

[38] B. Widenhofer, "Inside xbox 360's kinect controller." `http://www.eetimes.com/design/signal-processing-dsp/4211071/Inside-Xbox-360-s-Kinect-controller`, Okt 2010. zuletzt abgerufen am 14.06.2011.

[39] PrimeSense, "Reference Design." `http://www.primesense.com/?p=514`. zuletzt abgerufen am 14.06.2011.

[40] Microsoft Research, *Programming Guide: Getting Started with the Kinect for Windows SDK Beta*, June 2011. Beta 1 Draft Version 1.0.

[41] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," 2011.

[42] V. Henne, "Hand gesture recognition with a depth-sensing camera." Unveröffentlichte Bachelorarbeit.