



Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Fachgebiet Graphische Datenverarbeitung

Projektseminar

Gestensteuerung einer 3D-Anwendung mittels Kinect

Autoren: Mario Janke
Peter Lindner
Patrick Stäblein

Betreuer: M. Sc. Julian Meder

18. August 2017

Inhaltsverzeichnis

1 Rahmenbedingungen	2
1.1 Grundlagen und Motivation	2
1.2 Aufgabenstellung	3
1.3 Trackingsystem Microsoft Kinect	5
2 Vorüberlegungen	9
2.1 Hinzugekommene oder abgewandelte Anforderungen	11
2.2 Struktur der Arbeit	12
2.3 Werkzeuge	12
3 Entwurfsentscheidungen	15
3.1 Der Master	15
3.1.1 Möglichkeiten der Master-Identifikation	15
3.1.2 Fuzzy Skelettmatching	16
3.1.3 Robustheit	18
3.1.4 Umsetzung	19
3.2 Gesten und ihre Wirkung	21
3.3 Zustandsmaschine	27
3.4 Robustheit und Pufferung	30
3.4.1 Auftretende Probleme	30
3.4.2 Auswirkungen aus Nutzersicht	35
3.4.3 Behandlung	37
4 Bemerkungen zum Quellcode	41
4.1 Wichtige Datenstrukturen, Variablen und Funktionen	41
4.2 Ergänzungen zum Zusammenspiel und Ablaufskizze	47
4.3 Einbinden	51
5 Schlussbemerkungen	52
5.1 Eignung unserer Software	52
5.2 Eignung der Kinect	52
5.3 Verbesserungen	53
5.4 Fazit	54

1 Rahmenbedingungen

Die vorliegende Ausarbeitung entstand im Rahmen des Projektseminars im Master-Studiengang Informatik an der Technischen Universität Ilmenau. Ziel und Zweck des Projektseminars ist die teambasierte Auseinandersetzung mit einem Forschungsgegenstand unter Verwendung von Fachliteratur.

Im Folgenden werden die Grundgegebenheiten erläutert, die genaue Aufgabenstellung formuliert und das titelgebende Tracking-System Microsoft Kinect eingeführt.

1.1 Grundlagen und Motivation

Gegeben ist eine bereits vorhandene 3D-Anwendung, die vom Betreuer des Projektseminars, Herrn M. Sc. Julian Meder, erstellt wurde. Das Programm wird zu Demonstrationszwecken z. B. am Tag der offenen Tür in den Räumlichkeiten des Fachgebiets genutzt. Innerhalb der Anwendung ist es möglich,

- ein Objekt zu laden und anzeigen zu lassen sowie
- die Kamera (bzw. Kameras) zu bewegen, zu rotieren und zu zoomen.

In Erweiterung soll das Programm auch andere Aufgaben der Objektmanipulation wahrnehmen können, insbesondere das Laden von mehreren Objekten in die Szene und eine abwechselnde Manipulation dieser im Sinne von Translation, Rotation, Skalierung und Löschung.

Um den oben genannten Demonstrationszweck zu erfüllen, ist das Programm auf einem Rechner im Laborraum verfügbar. Es rendert zwei Ausgabefenster aus zwei verschiedenen, stereoskopisch angeordneten Kamerasichten. Der Laborrechner ist per HDMI an einen 3D-Kameraaufbau angeschlossen, der aus zwei Projektoren im Nebenraum besteht. Sie werfen die beiden Ansichten aus dem Programm von hinten auf einen Projektionsschirm, der in die Trennwand von Labor- und Nebenraum eingelassen ist. Mittels handelsüblicher aus 3D-Kinos bekannten Shutterbrillen können Benutzer und Zuschauer sodann den 3D-Eindruck wahrnehmen.

Der konkrete Präsentationsablauf, wie er vor dem Projektseminar stattfand, lässt sich wie folgt skizzieren. Ein Vorführender befindet sich mit einigen Personen Publikum im Laborraum. Während er etwa sich und das Fachgebiet vorstellt, werden das Programm auf dem Laborrechner und die Projektoren gestartet. Am Rechner kann der

Präsentierende dann z. B. ein Objekt laden. Diese Eingaben erfolgen per Maus und Tastatur.

Anschließend kann er wieder vor die Zuhörenden treten, da die Kameramanipulationen, die das Programm erlaubt, auch mittels eines Präsentationspointers durchgeführt werden können. So kann er das Programm, während er vorträgt, bedienen. An die Zuhörer werden die genannten Shutterbrillen verteilt, sodass sie im Laufe des Vortrages ein 3D-Bild der Szene und der Manipulationen des Vortragenden sehen können.

In dieser Form hat der Ablauf Schwächen, die die Motivation für das Projektseminar bilden:

- Die Steuerungen per Tastatur und Maus und noch stärker sogar die per Präsentationspointer bieten nur ein geringes Maß an Immersion.
- Es ist am Fachgebiet Technik (genauer Tracking-Systeme) vorhanden, mittels derer man eine Gestensteuerung für das genannte Programm realisiert werden kann. Eine solche ist für einen Vortrag wie etwa im Rahmen eines Tages der offenen Tür zusätzlich attraktiver für die Zuhörer und bietet außerdem dem Vortragenden ein weiteres Thema, auf das er eingehen und welches präsentiert werden kann.

Darüber hinaus motivierend ist auch die bloße Auseinandersetzung mit der genannten vorhandenen, aber bislang ungenutzten Technik und eine allgemeinere Untersuchung der Möglichkeiten, sie zielführend einzusetzen.

1.2 Aufgabenstellung

Aus der in Abschnitt 1.1 genannten Motivation heraus, ist es Ziel des Projektes, die Steuerung der gegebenen 3D-Anwendung hinsichtlich ihrer Präsentation vor einer Zuschauergruppe zu erleichtern und intuitiv zu gestalten. Dabei wird besonderer Wert auf das Eintauchen des Vorführenden in die 3D-Szene gelegt. Diese Anforderung soll durch die Implementierung einer Gestensteuerung erfüllt werden. Nebenbei findet Arbeit mit den noch nicht eingesetzten Tracking-Systemen des Fachgebiets statt, die zukünftige Themen für studentische Arbeiten oder Forschung am Fachgebiet stützen kann. Die vorhandenen Trackingsysteme sind

- ein professionelles Motion-Capture-Trackingsystem zum Tracken von Raumpunkten über angebrachte Marker und Wands und

- die Microsoft Kinect 2 zur optischen Gestenerkennung, i. W. mittels Kameraaufnahmen einer Infrarotprojektion.

Das so ausgewählte Trackingsystem ist also zur Implementierung einer Gesteuerung der gegebenen Anwendung zu verwenden. Die dafür entwickelte Software soll folgendes leisten:

- Es soll in der Lage sein, sämtliche Steuerung und Manipulation, die oben beschrieben wurde, durchzuführen, d. h. Kamera und Objekte verschieben und drehen können.
- Die Bedienung soll intuitiv und einfach sein, d. h. etwaige Gesten müssen bezüglich der ihnen zugeordneten Aktion einleuchtend und leicht ausführbar sein.
- Die Steuerung soll ihrem Zweck angemessen genau sein, bestenfalls als glatte 1-zu-1-Übertragung von Handbewegungen auf die Szene.
- Das entstehende Programm soll möglichst einfach eingebunden und wiederverwendet werden können.
- Darüber hinaus soll eine Mastererkennung bzw. -verwaltung implementiert sein, d. h. ein Verfahren, das garantiert, dass auch nur die dafür vorgesehene Person das Programm steuert und niemand sonst. Hiermit wird ausgeschlossen, dass ein Fremder die Kontrolle über das Programm gewinnen kann und die Vorführung damit – gewollt oder unbewusst – behindert. Die als Vorführender ausgezeichnete Person soll auch später wieder erkannt werden können und die Kontrolle wiedererlangen, etwa nachdem sie einen Augenblick lang nicht im von der Kamera abgedeckten Bereich war.

Damit ist die vollständige Liste der Anforderungen gegeben und wird hier zur Übersicht nochmals im Kern zusammengefasst:

Ziel ist die Entwicklung einer Software

- unter Verwendung vorhandener Technik (einem Trackingsystem),
- die eine Gesteuerung der gegebenen Anwendung ermöglicht und
- dabei nur dem Präsentierenden als ausgezeichneter Person die Steuerung erlaubt.

Zur Umsetzung dessen fiel die Wahl auf die Microsoft Kinect 2 als Trackingsystem. Dies hatte vielerlei Gründe:

- Wegen ihrer kommerziellen Herkunft aus der Spieleindustrie (vgl. Abschnitt 1.3) kann davon ausgegangen werden, dass sie bereits einem breiteren Publikum bekannt ist und gegebenenfalls auch besonders reizvoll erscheint.
- Vom Verfahren der 3D-Daten-Gewinnung her (Genaueres in Abschnitt 1.3) kann die Kinect vom Nutzer direkt und ohne weiterführende Vorbereitungsmaßnahmen verwendet werden. Markerbasierte Trackingsysteme haben diesen „Plug-and-Play“-Vorteil nicht.
- Ähnlich zum ersten Punkt liegt hinsichtlich der Kinect aufgrund ihres Bekanntheitsgrades eine sehr gute Quellenlage im Internet vor. Sie verfügt über eine offizielle (wenn auch in Teilen knappe) Online-Dokumentation und da sie mit SDK und API veröffentlicht wurde, finden sich auch in einem breiteren Rahmen zahlreiche Lösungsdiskussionen und -präsentationen von Nutzern im Netz.

1.3 Trackingsystem Microsoft Kinect



Abbildung 1: Frontalansicht der Kinect v2, Quelle: [3].

Die Kinect wurde in Version 1 das erste Mal auf der E3 2009 in Los Angeles präsentiert, damals noch unter dem Namen „Project Natal“ (siehe [13]). Ziel war es, für Spiele der Xbox-Konsole von Microsoft den herkömmlichen Controller durch den Einsatz des gesamten Spielerkörpers zu ersetzen, nicht zuletzt, um damit ein breiteres Zielpublikum anzusprechen. Bereits zu diesem Zeitpunkt war geplant, unabhängigen Entwicklern die Verwendung und Programmierung des Systems zu ermöglichen (s. ebd.).

Im Folgenden werden die technischen Spezifikationen der Kinect in ihrer aktuellen Version erläutert. Diese sind [8] entnommen. Die Kinect verfügt über eine Full-HD-Farbkamera und eine Infrarotkamera, die die Aufgabe des Tiefensensors erfüllt. Dieser ist mit 512 zu 424 Pixeln deutlich geringer aufgelöst. Der effektive Bereich ist laut Hersteller in einer Entfernung zwischen einem halben und 4,5 Metern. Die Kameras der Kinect liefern weiterhin maximal 30 Frames pro Sekunde. Weiterhin besitzt die Kinect ein Multi-Array-Mikrofon, mit dem Klang nicht nur aufgenommen, sondern auch hinsichtlich seiner Ausbreitung untersucht werden kann. Im Vergleich zu ihrer Vorgängerversion wurden vorwiegend Genauigkeitsverbesserungen vorgenommen, es kommt zu weniger Grundrauschen und die Objekterkennung ist generell stabiler und zuverlässiger. Mit der aktuellen Version können die Skelette von bis zu sechs Personen bei 25 Gelenkpunkten pro person vollständig getrackt werden (siehe Abbildung 2). Die zur Verfügung stehenden Gelenkpunkte lassen sich über die API des Kinect-SDK abfragen.

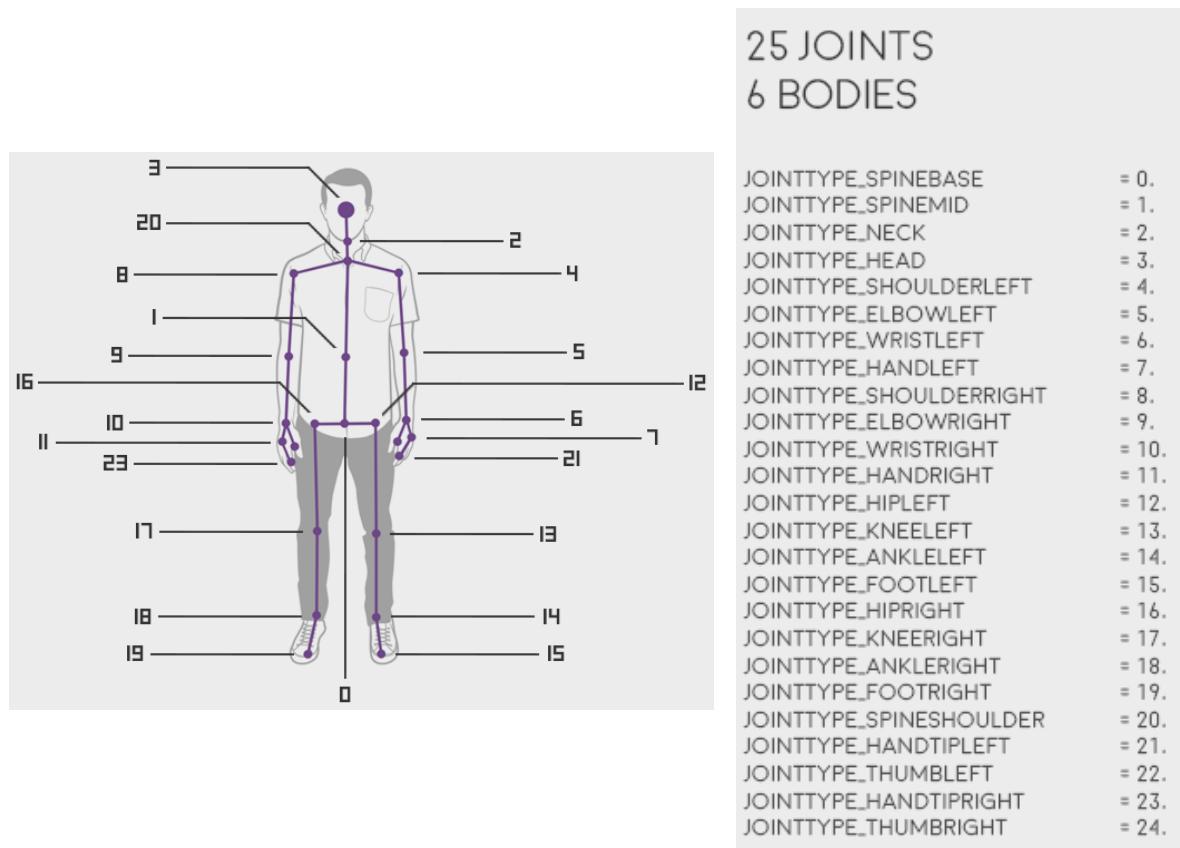


Abbildung 2: „Kinect V2 Joint ID Map“, Gelenkpunkte die mit der Kinect getrackt werden können. Quelle (Ausschnitt): [17]

Um die Tiefeninformation zu gewinnen, verwendet die Kinect in der zweiten Version ein sogenanntes Time-Of-Flight-Verfahren (hier und im Folgenden vgl. [4]). Grundlage ist die Gewinnung von 3D-Daten aus einem Kamerabild (in diesem Falle dem des Tiefensensors). Beim Time-Of-Flight-Verfahren wird für das ausgesendete Infrarotlicht gemessen, wie lange es braucht, um von der Objektoberfläche reflektiert zu werden und zurück zum Sensor zu gelangen. Beim Vorgänger, der Kinect-Version 1, wurde noch eine pseudorandomisiertes Muster auf die Szene projiziert und die Tiefeninformation daraus trianguliert (siehe [5]). Ein Problem dieses Verfahrens ist die Forderung, für einen Punkt des Musters auch stets eine kleine Umgebung zu sehen, in der genug andere Lichtpunkte liegen, um eine Identifikation vorzunehmen. Der Time-Of-Flight-Ansatz der Kinect 2 besitzt diese Abhängigkeit nicht mehr. Eine weitere Verbesserung gegenüber der Kinect 1 ist das Vorhandensein eines eingebauten Umgebungslichtfilters: Wird ein Pixel von zu viel Licht aus dem sichtbaren Spektrum, insbesondere dem infrarotnahen Teil, getroffen, kann er während der Bestimmung zurückgesetzt werden (vgl. ebd.).

Neben dem oben erwähnten, aus 25 Gelenkpunkten bestehenden und damit recht grob-gramularen Skelett, dass die Kinect den Körpern zuweist, gibt es weiterhin für beide Hände einer getrackten Person einen „Handzustand“, der „offen“, „geschlossen“, „unbekannt“ oder „Lasso“ sein kann. Die Lassogeste besteht dabei aus einer halbgeschlossenen Hand, etwa mit zwei ausgestreckten Fingern).

Über eigene Konfidenzmechanismen, die dem Programmierer gegenüber transparent sind, verfügt die API nicht. In diesem Zusammenhang unterscheidet die Kinect, wie der Dokumentation [11] auch zu entnehmen ist, nur zwischen drei Erkennungsgüten:

1. Ungetrackt: Das betroffene Gelenk wurde im Aufnahmebereich der Kinect nicht erkannt, es liegen keine Daten vor.
2. Gefolgert: Das betroffene Gelenk wurde nicht direkt erkannt, seine Daten jedoch aus den Restdaten angenähert. Über die Güte der Annäherung gibt es keinerlei Garantie.
3. Getrackt: Das Gelenk ist im Aufnahmebereich gefunden und getrackt und die Daten können als zuverlässig angesehen werden.

Die Kinect-Eigenschaften hinsichtlich der Datengüte werden im Rahmen der Besprechungen zur Robustifizierung des Projektcodes zusätzlich und präziser in Abschnitt

3.4 diskutiert. Die hier aufgeführten Daten können über die Kinect-API abgegriffen werden. Dazu muss auf dem entsprechenden Rechner das Kinect-SDK vorliegen und die Kinect mit vorhandenen Treibern per USB und HDMI angeschlossen sein.

In ihrer Erscheinungszeit kostete die Kinect v2 für Windows ca. 200 Euro, was verglichen mit professionellen 3D-Erfassungssensoren (die z. T. dasselbe Verfahren verwenden) ein niedriger Preis ist (vgl. [4]). Gerade aus diesem Grund und wegen der Verbreitung sind weitere Anwendungen aus Sicht der Forschung interessant, z. B. die Verwendung bei Assistenzrobotern (siehe [16] und [18]).

2 Vorüberlegungen

Für das Vorgehen zur Lösung der in Abschnitt 1.2 genannten Aufgaben zentral sind die folgenden Bereiche:

- (i) Die Interaktion mit dem Benutzer, der das Programm steuert, d. h.
 - das Entwerfen von Gesten für die verschiedenen Zwecke, die intuitiv und eingängig, leicht ausführbar und gut voneinander unterscheidbar sind sowie
 - das Auszeichnen einer getrackten Person als „Master“ (i. F. zum Teil auch so bezeichnet), der als einziger das Programm steuert.
- (ii) Die technische Umsetzung, d. h.
 - das konkrete Verfahren zur Auszeichnung einer Person als Master,
 - damit verbunden das korrekte Wiedererkennen der ausgezeichneten Person und die Verhinderung der Programmsteuerung durch andere,
 - das korrekte Erkennen und Werten von Gesten der ausgezeichneten Person und
 - die Einbindung der Software, die all dies leistet, in die bereits bestehende Applikation.

In diesem Abschnitt werden die unmittelbaren wichtigen Beobachtungen und Überlegungen vorgestellt, die sich aus der Aufgabenstellung und dem Versuchsaufbau ableiten lassen.

Von der formulierten Aufgabe ausgehend kann abstrahierend zwischen zwei primitiven Steuerungsmodi unterschieden werden:

- Einem Modus, in dem die Kamera verschoben und rotiert werden kann und
- einem Modus, in dem Objektmanipulationen möglich sind.

Der Benutzer sollte sich zu jedem Zeitpunkt der Programmausführung nur in maximal einem dieser Modi aufhalten, d. h. gleichzeitige Kamera- und Objektmanipulationen sind ausgeschlossen. Diese Vereinfachung gestattet einerseits die Verwendung von weniger komplexen Gesten, andererseits besitzt eine solche simultane Manipulation keine vordergründige praktische Relevanz und kann auch als mit dem Prinzip der Einfachheit in Konflikt stehend angesehen werden. Zusätzlich bietet sie als Ansatz bereits eine

geeignete Kapselung, da zum Beispiel Rotationsparameter berechnet werden können und dann nur noch entschieden werden muss, ob sie auch die Kamera oder ein Objekt anzuwenden sind, je nach Modus. Dies erlaubt es gegebenenfalls die Anzahl der verwendeten Gesten weiter zu reduzieren, wenn für die verschiedenen Modi Gesten wiederverwendet werden.

Die Kinect ermöglicht ein Tracking des gesamten Körpers für mehrere Personen (vgl. Abschnitt 1.3). Für die vorgesehene Anwendung ist jedoch nur ein Teil dieses Datenspektrums relevant:

- Der Hauptteil der Funktionalität „Steuerung durch eine bestimmte Person“ betrifft nur die Tracking-Daten eines Benutzers. Eine genaue Auswertung weiterer sich im Raum befindlicher Personen und ihrer Skelette ist zumeist unnötig.
- Die Gesten, die für den gegebenen Anwendungsfall infrage kommen, sind Gesten, die ausschließlich Oberkörperpartien, vor allem Arme und Hände, einbeziehen. Im Sinne der Masterfestlegung sind gegebenenfalls weitere Körpermerkmale in Form von Bedingungen zu verwenden, jedoch wird insgesamt nur ein Teil der zur Verfügung stehenden Skelett- und Gelenkdaten ausgewertet werden.
- Von den Daten, die die Kinect zurückliefert, erweisen sich einige für die Anwendung als gänzlich irrelevant, etwa die Aufnahmen vom Kinect-Mikrofon.

Eine erste primitive Erkennungsmöglichkeit eines Masters kann aus der Entfernung der getrackten Person zur Kamera und der Position der Personen im Raum gewonnen werden. In der oben angedeuteten Vergleichsvariante wird hingegen eine Sammlung von Körpermerkmalen einer Person zusammengestellt, die später dem Abgleich dienen kann. Genauere Erläuterungen hierzu finden sich in Abschnitt 3.

Eine intuitive Geste für Verschiebungen imitiert das Verschieben eines großen Gegenstands, etwa einer imaginären Box, sodass hier eine Bewegung der flachen Hand in der Luft naheliegt. Für eine intuitive Drehgeste eignet sich die Vorstellung eines imaginären Lenkrades. Dieses kann gedacht für eine Rotation in beliebige Richtungen zu einer Art Lenkkugel erweitert werden, bei der Rotation um eine beliebige Raumachse nach dem Lenkradprinzip erfolgen kann. Eine sehr naheliegende Geste zur Objektauswahl und damit -manipulation wäre eine Greifgeste.

Die genannten Gesten und Umsetzungen folgten den Überlegungen nach ersten Tests mit der Kinect und wurden im weiteren Verlauf überarbeitet.

2.1 Hinzugekommene oder abgewandelte Anforderungen

Wie bei solch praktisch orientierten Projekten gewöhnlich, fielen in den Tests und der Arbeit mit dem Programm bestimmte Features auf, die einen äußerst positiven Effekt auf die Bedienerfahrung des Programmes hätten. So kamen zu den in Abschnitt 1.2 genannten Aufgaben einerseits zusätzliche Anforderungen hinzu, andererseits wurden Vorstellungen von der Funktionalität, die Schwierigkeiten bei der Umsetzung oder Bedienung aufwiesen abgewandelt und durch sinnvollere Konzepte ersetzt.

In der original angedachten Variante der Masterfestlegung wurde der Einspeichervorgang (Sammlung von Daten über den vorgesehenen Master) per Tastendruck gestartet. Dies kann ein Partner des Vortragenden am Rechner vornehmen oder aber der Vortragendes selbst per Präsentationspointer. Eine Verbesserung dieses Prinzips ist eine Selbstauslösermechanik. Dazu soll der vorgesehene Master das Einspeichern am Rechner (wahlweise per Präsentationspointer) anstoßen können. Das Einspeichern beginnt dann jedoch nicht sofort, sondern erst, wenn eine Person (in einer eigens dafür definierten Standardpose) im Sichtbereich der Kamera ist. Dies gibt ihm die Möglichkeit sich ins Aufnahmefeld zu bewegen, um dort als Master festgelegt zu werden. Die Notwendigkeit einer zweiten Person, die den Rechner bedient (sofern kein Präsentationspointer oder ähnliche Eingabegeräte vorhanden sind) entfällt hiermit komplett. Hierbei fiel weiterhin auf, dass es für den Anwender nützlich ist, die Länge (und damit die Genauigkeit) der Einspeicherung beeinflussen zu können. Dafür eignet es sich etwa, einen Zusammenhang zwischen der Länge des Tastendrucks für Einspeicherung und der Einspeicherungszeit selbst herzustellen. D. h. je länger der Vorführende zu Beginn die Einspeichertaste drückt, desto mehr Frames werden – sobald die Sammlung beginnt – zur Festlegung verwendet. Eine Sekunde Tastendruck sollte etwa einer Sekunde Einspeicherungszeit entsprechen.

Ein weiterer Punkt ist das großräumigere Navigieren durch die Szene. Entsprechend der oben genannten Ideen zur Kamerabewegung ist für das Zurücklegen einer größeren Strecke ein ständiges Nachgreifen nötig. Dies ließe sich durch Erschaffen eines „Flug-Modus“ vermeiden. Konzeptuell soll sich die Kamera dabei solange nach vorne bewegen, wie eine bestimmte Geste präsentiert wird. Ein einfacher Ansatz hierfür sind nach vorne ausgestreckte Arme, wobei mit Kippbewegungen und der Zeigerichtung auch die Flugrichtung manipuliert werden kann.

Schließlich stellte sich bei der Arbeit schnell heraus, dass die Entwicklung einer solchen Anwendung sehr auf visuelles Debugging angewiesen ist. Im ursprünglichen Konzept gab es jedoch nach den A-priori-Überlegungen keinerlei Rückkopplung zum Hauptprogramm, die über den Erfolg oder Misserfolg von Einzelschritten Auskunft hätte geben können. So kam als Anforderung hinzu, ein Eventsystem in Form von Stubs zu implementieren, das später dazu ausgebaut werden kann, bestimmte Statusmeldungen von der Gesteuerung und Mastererkennung, wie etwa „Master festgelegt“, „Master verloren“ oder die Angabe des aktuellen Steuerungsmodus zurückzugeben.

2.2 Struktur der Arbeit

Als initiale Motivation konzentriert sich die Arbeit zunächst auf Gesteuerung. Hierbei können Prinzipien für die Steuerung der Kamera gewonnen werden, die sich sodann in Teilen auch auf die Objektmanipulation übertragen lassen. Für diese Erarbeitung ist es notwendig, genaue Gesten zu definieren und ein Rechenmodell zu entwerfen, das sich zur Steuerung eignet. Dies findet in den Abschnitten 3.2 und 3.3 statt. In 3.1 wird die Umsetzung der Masterfestlegung und -erkennung erläutert und diskutiert. Darüber hinaus bietet Abschnitt 4 einen tieferen Einblick in den im Rahmen des Projektes entstandenen Programmcode, geht auf Schlüsselemente ein und erläutert das Zusammenspiel. Schließlich wird in Abschnitt 5 die Eignung der entstandenen Software, aber auch die Eignung der Kinect für die gestellte Aufgabe besprochen und ein Ausblick auf Verbesserungen und Erweiterungen der Anwendung gegeben.

2.3 Werkzeuge

In diesem Abschnitt sollen die Tools genannt und erklärt werden, die für die Entwicklung vordergründig waren. Sie soll einen Einblick in die Mächtigkeit der verwendeten Werkzeuge geben und dokumentieren, welche Debugmöglichkeiten bestanden und wie sich eine solche Arbeit koordinieren lässt.

Das im Zentrum stehende Trackingsystem *Microsoft Kinect Version 2* war durch das Fachgebiet gegeben. Zur Arbeit damit stand ein Raum bereit, der über die Kinect und einen 3D-Kamera-Aufbau zur Projektion verfügte.

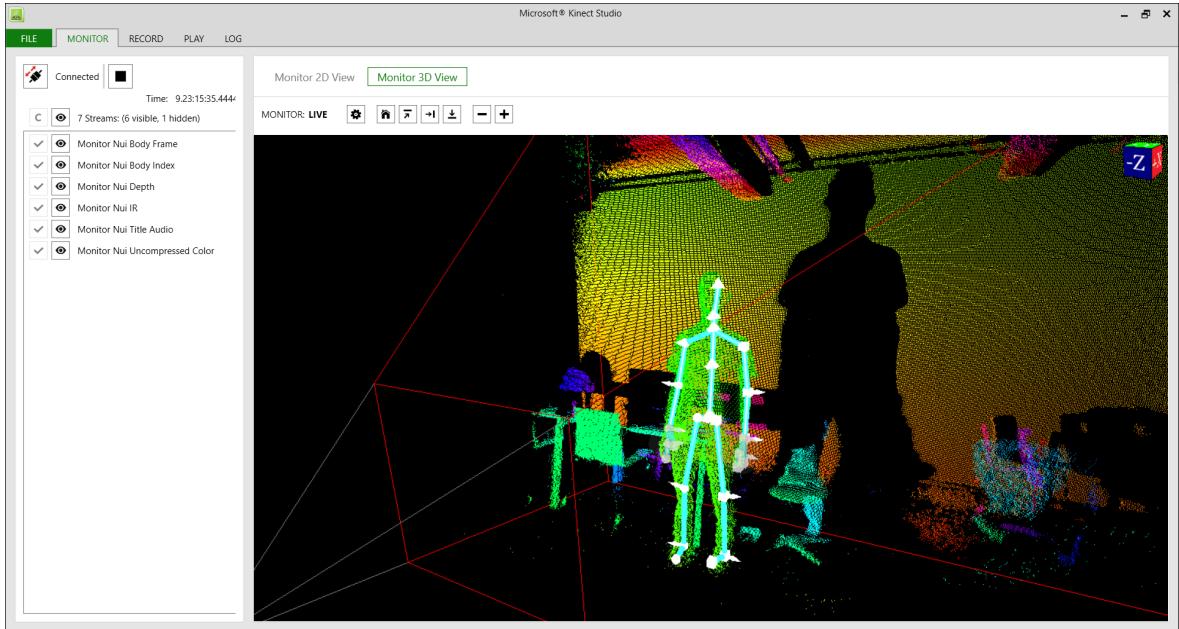


Abbildung 3: Kinect-Studio mit getracktem Skelett, 3D-Ansicht.

Mit dem Kinect SDK wird eine Softwarelösung namens *Kinect Studio* ausgeliefert, die sich bei der Kinect-Programmierung zum visuellen Debugging eignet (siehe Abbildung 3). Im Kinect Studio können die verschiedenen Sensoraufnahmen der Kinect nebst der interpretierten Skelette und Hand-States sowie der festgestellten Tiefensituation betrachtet werden. Diese Features können getoggelt oder umgeschaltet werden. Die Tiefenkarte wird per Falschfarbendarstellung und, sofern erwünscht, sogar dreidimensional präsentiert. Ebenso kann hier die Infrarot- und Farbkameraaufnahme betrachtet werden. Das Kinect Studio erweist sich als sehr hilfreich, um die Güte der Kinect-Daten zu überprüfen und zu erkennen, ob die Kinect einen Teil des Aufnahmebereiches fehlinterpretiert. Dies ist zum Teil notwendig, um bei der Programmierung schnell und einfach zwischen Fehlern des Programmes und fehlerhaften Kinect-Daten unterscheiden zu können.

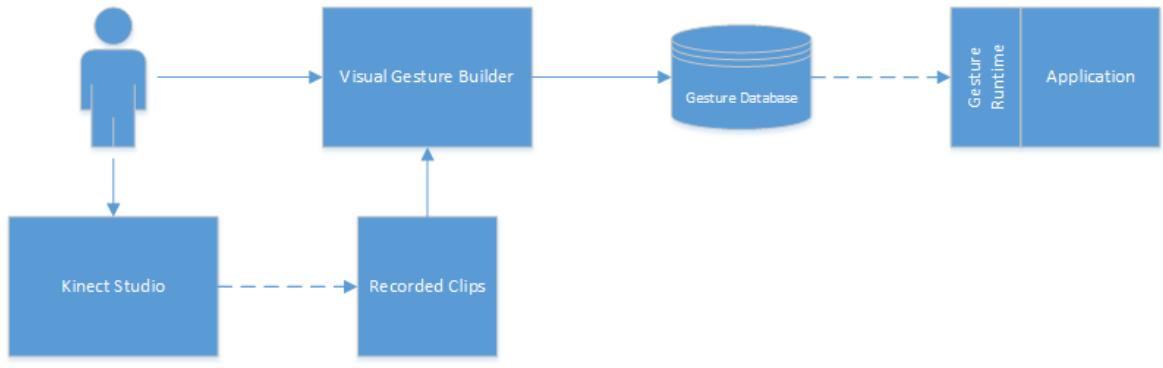


Abbildung 4: Schemadarstellung der Verwendung des *Visual Gesture Builders* im Kontext eines Kinect-Programmes. Quelle: [12]

Ebenfalls zum Kinect-SDK gehörig ist der sogenannte *Visual Gesture Builder*, mit dem Gesten(folgen) aufgenommen und in Programme eingespeist werden können (siehe Abbildung 4). Da die Entscheidung letztlich (siehe Abschnitt 3.2) auf einen anderen Weg der Gestenimplementierung fiel, fanden hiermit nur kleinere Tests in der Anfangsphase statt.

Die Kinect-API kann mit JavaScript, C++ oder C# verwendet werden. Da das im Rahmen der Aufgabenstellung übermittelte Programm, für das die entstehende Gestensteuerung gedacht ist, in C++ geschrieben war, wurde aus Kompatibilitäts- und Einheitlichkeitsgründen heraus ebenfalls C++ verwendet. Die Entwicklung und das Debugging fanden mit dem *Microsoft Visual Studio 2015* unter *Microsoft Windows 10*. Zur Versionsverwaltung fand zusätzlich das Kommandozeilentool git mit [GitHub](#) Verwendung.

3 Entwurfsentscheidungen

3.1 Der Master

Der Master ist die Person (unter den getrackten Personen), der es obliegt, die Anwendung zu steuern, d. h. im Anwendungsfall der Präsentation ist der Master der Präsentierende. Es muss gewährleistet werden, dass nur der Master das Programm steuert und dabei von weiteren Personen im Raum nicht (bzw. nicht ohne weiteres) gestört werden kann. Die Erkennung muss robust gegen Jittering der Kinectdaten sein.

3.1.1 Möglichkeiten der Master-Identifikation

Grundsätzlich kamen für die Festlegung des Masters zwei Ideen auf. Zunächst sollte bei jedem Frame die getrackte Person identifiziert werden, die der Kinect bzgl. der z-Koordinate am nächsten ist und diese als Master festgelegt werden. Der Master könnte hierbei bei jedem Frame zwischen den getrackten Personen wechseln.

Die zweite Möglichkeit war die Festlegung des Masters auf eine bestimmte Person, von der zunächst bestimmte Identifikations-Merkmale eingespeichert werden und die dann anhand dieser als Master reidentifiziert werden kann. Sofern diese Festlegung erst einmal geschehen ist, bleibt diese Person Master, selbst nachdem sich diese zwischenzeitlich in einem ungetrackten Zustand (beispielsweise beim Herausgehen aus dem getrackten Bereich) befunden hat und dann wieder als getrackt erkannt wird. Bei einer Recherche, welche Merkmale sich aus den von der Kinect gelieferten Daten extrahieren lassen ließen, um hierfür in Frage zu kommen, ergaben sich verschiedene Möglichkeiten, von denen einige jedoch aufgrund ihrer Unpraktikabilität ausschieden. Eine Erkennung anhand des Gangs ([15]) oder anhand der Stimme würde hier keinen Sinn ergeben, da die Master-Person während der Bedienung kaum umher läuft und diese hierfür nicht zu sprechen braucht). Schließlich gibt es Verfahren, die die Skelettdaten der Kinect zur Identifikation nutzen. Dies schien die praktikabelste Lösung zu sein, wenngleich der hier präsentierte Ansatz, die Skelettdaten zu nutzen im Vergleich zu denjenigen in den gefundenen Arbeiten ([2]) stark vereinfacht wurde.

Grundlegendstes Prinzip für die Identifizierung einer Person ist hierbei das Auslesen der Skelettkoordinatenpunkte mit Hilfe des Kinect SDKs und daraus der Ermittlung diverser Längen als Körperproportionen mittels der Berechnung des euklidischen Abstands zwischen den entsprechenden Skelettpunkten. Beispielsweise wird die rechte

Oberarmlänge als Abstand zwischen dem rechten Schulterpunkt und dem rechten Ellenbogenpunkt ermittelt. Weitere Körperproportionen die verwendet wurden sind unter anderem die Schulterbreite, Hüftbreite, Unterarmlänge, Abstand zwischen Hals und Kopf.

3.1.2 Fuzzy Skelettmaching

Eine notwendige Grundfunktion bei der Mastererkennung ist es, ein aufgenommenes Skelettprofil mit einem im Bild sichtbaren Skelett zu vergleichen. Bei jedem Skelettmerkmal, wie beispielsweise der Oberarmlänge oder der Torsolänge, sind Abweichungen zu erwarten. Es bestehen Freiheitsgrade in der Gewichtung dieser Abweichungen. Ziel ist es, eine Funktion zu entwerfen, welche die Ähnlichkeit zweier Skelette repräsentiert. Folgende Varianten sind denkbar: Für jedes Skelettmerkmal P des Masterprofils und jedes Merkmal K des Kandidaten:

1. Es wird das Maximum von P und K gebildet und durch das Minimum von P und K geteilt. Die so berechneten Werte werden durch Multiplikation akkumuliert.

Der Ergebniswert liegt zwischen 0 und 1. Je größer er ist, desto näher sind die Merkmale des Kandidaten denen des eingespeicherten Masters. Die Funktion besitzt die erstrebenswerte Eigenschaft, dass ein identisches Matching einen Fehlerwert von 1 repräsentiert. Beim experimentellen Testen der Funktion bestätigte sich der Verdacht, dass große Fehler zu schwach bestraft werden. Das genaue Verhalten wurde aus Zeitgründen nicht weiter untersucht, die Ergebnisse waren jedoch nicht zufriedenstellend.

2. Es wird angenommen, dass die Merkmale des Kandidaten Poisson-verteilt sind und die Erwartungswerte die realen Knochenlängen repräsentieren. Aus P werden der empirische Erwartungswert und die empirische Varianz der angenommenen Poisson-Verteilung berechnet. Für jedes Skelettmerkmal wird die Wahrscheinlichkeit des Ereignisses „Das Skelettmerkmal ist K “ berechnet. Diese Werte werden akkumuliert.

Der resultierende Wert würde der Wahrscheinlichkeit entsprechen, dass K tatsächlich P repräsentiert. Der Ansatz kann weiterhin für andere Verteilungen versucht werden. Es zeigte sich jedoch, dass die Fehler des Skelettprofils nicht gut als Poisson-verteilt angenähert werden können. Die Verteilung zeigte Sprünge

und multimodales Verhalten, sodass es großen Aufwands bedürfe, sie genau zu bestimmen.

3. Es wird $(P - K)^2$ berechnet und die entstandenen Werte summiert.

Dieser Ansatz kam aus der generellen Feststellung, dass das zweite zentrale Moment (die quadrierte Abweichung) häufig genutzt wird, um Ähnlichkeit (beispielweise zwischen Funktionen) zu bestimmen. Der Ansatz einspricht einem χ^2 -Hypothesentest. Auch dieser trifft eine implizite Annahme über die Verteilung der Fehler. In diesem Fall wird angenommen, die Fehler könnten gut als χ^2 -verteilt (Verteilung der quadrierten Summe normalverteilter unabhängiger Zufallsgrößen) genähert werden. Um diese Forderung zu testen, wäre es möglich gewesen, einen Anpassungstest auf χ^2 Verteilung zu machen. Hierauf wurde aus folgenden Gründen verzichtet:

- Der Aufwand, Fehlerwerte aufzunehmen und auszuwerten wäre nicht zu vernachlässigen.
- Um die Resultate nutzen zu können, wäre ein vielversprechender alternativer Ansatz zum Vergleich notwendig.

4. Diverse weitere Akkumulationsverfahren, welche ad hoc entstanden, wie beispielsweise die Summe der normierten Abweichungen oder das Maximum der normierten Abweichungen.

Diese Ansätze wurden nicht weiter verfolgt, da sie nicht lohnend für experimentelle Überprüfung schienen.

Die Entscheidung fiel daher nach ausgiebiger Diskussion und experimenteller Überprüfung einiger Variante auf Variante (3), da sie die besten Ergebnisse lieferte.

Ein weiterer Ansatz wäre, die verschiedenen Abweichungswerte mit dem Kehrwert der Standardabweichung ihres zugehörigen Profils zu wichten. Dies würde einem Merkmal mit geringer Standardabweichung eine sehr hohe Relevanz verleihen. Der Ansatz schien jedoch nur unzuverlässig zu funktionieren. Grund dafür könnte eine unbekannte Eigenschaft der (ebenfalls nicht bekannten) Fehlerverteilung sein. Bei der Implementierung wurde die Möglichkeit offen gehalten, die Standardabweichung des Profils einfließen zu lassen, eine schwächere Einwirkung als direkt multiplikativ könnte sinnvoll sein.

Ein Schwellwert für die Akzeptanz eines Kandidaten wurde experimentell festgelegt. Die Wahl bestimmt den Anteil von falsch-positiven sowie korrekt-positiven Akzeptanzen. Die Anpassung wurde so getroffen, dass zufriedenstellend wenige falsch-negative Erkennungen passieren, und ein Nutzer verlässlich den Masterstatus wiedererlangen kann.

3.1.3 Robustheit

Nach einigen Experimenten mit den Körperproportionen wurde festgestellt, dass einige Proportionen übermäßig große Abweichungen aufwiesen, wenn die gleiche Person in unterschiedlichen Posen mit der Kinect vermessen wurde bzw. dass sich einige Skelettpunkte „verschieben“, wenn der Benutzer mit einem Körperteil darüber fährt, beispielsweise die gemessene Schulterbreite, welche sich signifikant zwischen der Standard-Pose und einer Pose bei der die Arme über den Kopf gestreckt sind unterschied.) Aus

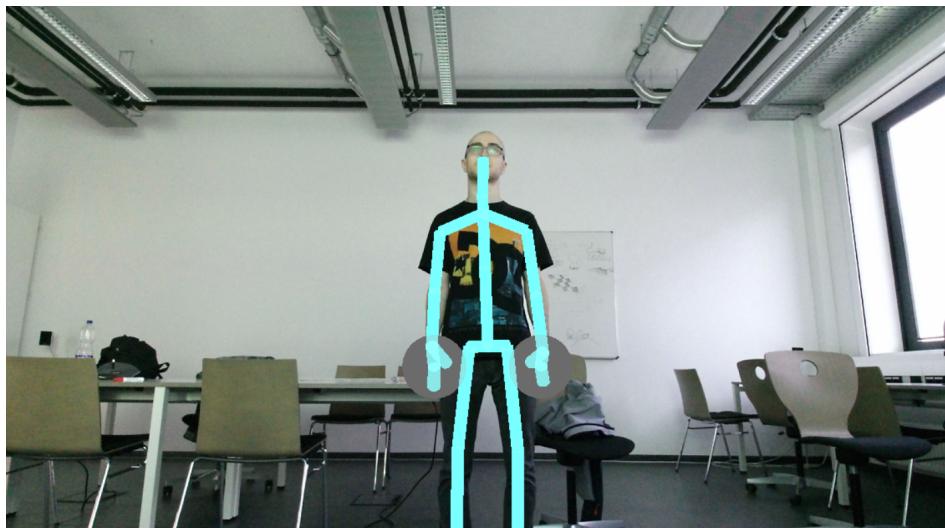


Abbildung 5: Die Standardpose, in der sich eine Person befinden muss, damit sie vermessen werden kann.

diesem Grund wird die Vermessung einer Person nur durchgeführt, wenn sich die entsprechende Person über eine bestimmte Anzahl von Frames (etwa 20) durchgehend in der festgelegten Standard-Pose befindet. Falls während der Sammlung das Einhalten der Standard-Pose unterbrochen wird, wird die Sammlung erneut begonnen.

Ein weiteres Problem, das zu lösen war, bestand darin, dass selbst beim Stillhalten, jedoch hauptsächlich bei der Bewegung einer getrackten Person, Skelettpunkte kurzzeitig

auf einen anderen weiter entfernten Raumpunkt springen konnten, was teils zu unrealistischen Messungen einer Körperproportion für einzelne Frames führte (z.B. Messung einer Oberarmlänge von 2 Metern). Um derartige Ausreißer zu eliminieren wird der während der Vermessung einer Person zur Einspeicherung als Master über mehrere Frames der Mittelwert sowie die Standardabweichung jeder Körperproportion berechnet und jeder Wert der nicht innerhalb des Intervalls um den Mittelwert m mit der Ausdehnung der Standardabweichung s liegt, d.h. des Intervalls $(c \cdot (m - s), c \cdot (m + s))$ ignoriert. Dabei ist c ein empirischer Faktor.

Weiterhin wird darauf geachtet, dass alle für die Vermessung relevanten Körperproportionen mit ausreichender Konfidenz von der Kinect getrackt werden. Die Kinect liefert hierfür für jeden Skelettpunkt einen von drei möglichen Konfidenzwerten, der angibt, wie wahrscheinlich die von der Kinect gelieferte Koordinatenwerte für diesen Punkt dem tatsächlichen Wert entsprechen. Die drei Konfidenzwerte heißen Tracked, Inferred, NotTracked; wobei Tracked die höchste Konfidenz und NotTracked die niedrigste Konfidenz für einen Skelettpunkt bezeichnet. Ist in einem Frame der Konfidenzwert von einem der beiden Skelettpunkte, aus denen eine Körperproportion berechnet wird, nicht Tracked, so wird diese Körperproportion für diesen Frame nicht berücksichtigt bzw. mit Null gewichtet.

3.1.4 Umsetzung

Die Master-Identifikation geht insgesamt folgendermaßen vonstatten: Beim Start des Programms ist zunächst einmal die Person Master, die bzgl. der Kinect die geringste z -Koordinate aufweist. Dies ist lediglich eine primitive Basisvariante, solange kein Master eingespeichert wurde. Hierfür wird bei jedem Schleifendurchlauf abgefragt, welche der 6 (potenziell) getrackten Personen, den geringsten z -Wert hat. Der Master wird somit bei jedem Frame neu bestimmt.

Durch die Betätigung einer vorher festgelegten Taste wird schließlich die Master-Festlegung auf eine bestimmte Person aktiviert. Hierzu werden die Körperproportionen der Person, die als erstes in Standardpose erkannt wird, über mehrere Frames aus den Skelettdaten extrahiert, gepuffert und schließlich aus diesen für jedes Körpermerkmal der Mittelwert sowie die Standardabweichung berechnet. Sofern beim Tastendruck noch keine Person im getrackten Bereich war, wird gewartet bis eine Person getrackt wird und diese die Standard-Pose einnimmt. Der Mittelwert und die Standardabweichung werden dazu verwendet Ausreißer zu identifizieren und zu entfernen, um schließlich die

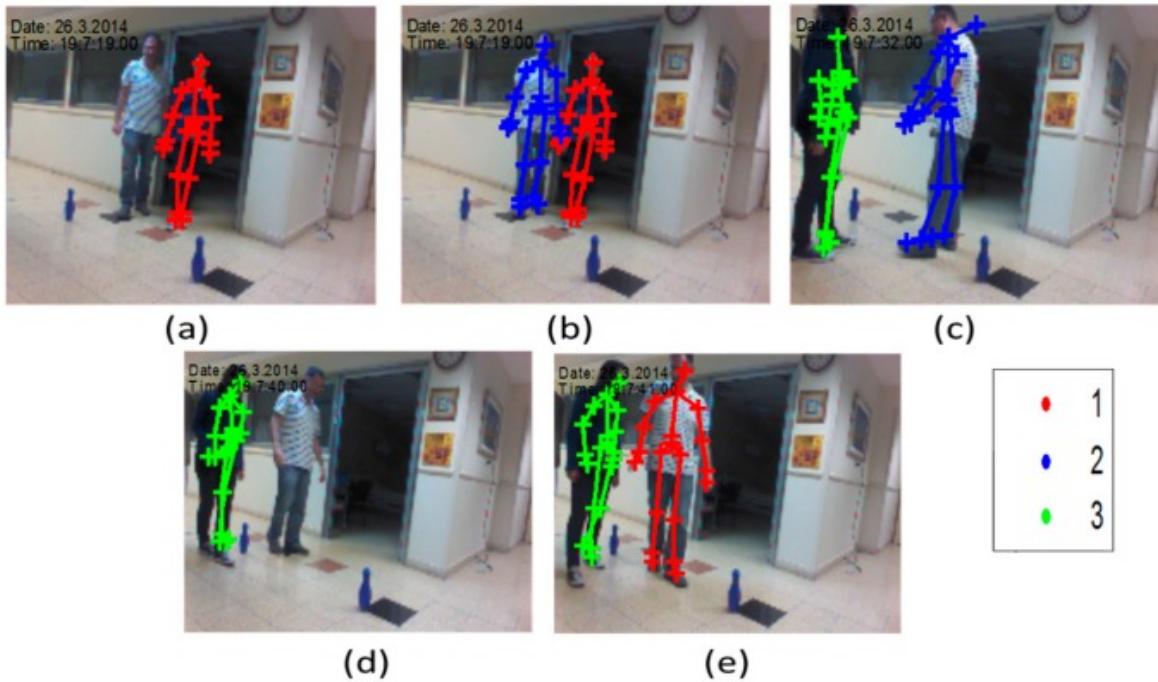


Abbildung 6: Neuzuordnung von IDs nach zwischenzeitlichem „Verlust“ von Skeletten,
Quelle:[2]

Mittelwerte erneut ohne die Ausreißer zu berechnen. Neben diesen Mittelwerten für die nun als Master festgelegte Person, wird außerdem die von der Kinect vergebene ID der Person eingespeichert. Diese bleibt dieser Person zugeordnet, solange sie durchgehend getrackt wird. Verlässt der Master den getrackten Bereich oder wird anderweitig vorübergehend nicht getrackt (beispielsweise wenn dieser von einer anderen Person verdeckt wird) und kommt danach wieder in den getrackten Bereich zurück, hat dieser eine neue ID. In diesem Fall muss der Master neu identifiziert werden.

Solange der Master noch nicht gefunden wurde, werden hierfür für jede Person, die in Standard-Pose erkannt wird, die Körperproportionen über eine gewisse Anzahl (etwa 20) an Frames mit denen der für den Master eingespeicherten Werte verglichen. Schließlich wird aus den frameweise berechneten Abweichungen der Durchschnitt berechnet. Ist diese durchschnittliche Abweichung kleiner als ein im Programm festgelegter Wert ist diese als Master identifiziert worden. Diese kann nun mit der Steuerung des Programms fortfahren.

3.2 Gesten und ihre Wirkung

Für die eingangs erwähnte Aufgabenstellung war es notwendig, bestimmte Programm-funktionalitäten mit Gesten zu verbinden. Einerseits hätte die Möglichkeit bestanden, mit dem Kinect-eigenen Visual Gesture Builder Gesten aufzunehmen und einzulernen. Diese Gesten werden dann als Datenbank ins Programm geladen und bei Vorführung erkannt. Dies erspart natürlich primitive aber umständliche Low-Level-Erkennungsmechanismen. Weiterhin sind hierdurch einige weiterführende Möglichkeiten gegeben wie etwa die Rückgabe, bis zu welchem Punkt eine Geste bereits ausgeführt wurde (in Bezug zur Gesamtgeste, d. h. beispielsweise wieviel Prozent einer Armbewegung vordefinierte Länge bereits ausgeführt wurde). Anderseits wiederum können durch die Kinect-Rohdaten auch eigene Erkennmechanismen implementiert werden. Dies bietet dem Programmierer die vollständige Kontrolle und Freiheit darüber, wie er Gesten definiert und auswertet, statt wie im Falle des Gesture Builders auf ein gewisses Rahmenwerk angewiesen zu sein. Änderungen können kurzfristig und schnell vorgenommen werden und für einfache Projekte ist die Zusatzfunktionalität, die der Visual Gesture Builder gestattet nicht vonnöten, der eher für komplexere Gestenfolgen ausgelegt zu sein scheint. Demgegenüber ist für diese Direktimplementierung von Gesten aber die bereits erwähnte Low-Level-Erkennung zu implementieren, d. h. ein Extrahieren von Bewegungen und Bewegungsrichtungen aus den Skelett- und Gelenkdaten, die die Kinect bestimmt. Aus dieser Gegenüberstellung heraus, ist für den gegebenen Einsatzzweck eine direkte Gestenerkennung die sinnvollere Alternative. Mit dem Visual Gesture Builder ist es schlicht nicht möglich, eine wie von uns beabsichtigte 1-zu-1-Abbildung zwischen Geste und Wirkung zu bewerkstelligen.

Auch bei der Low-Level-Erkennung gibt es jedoch verschiedene Ansätze bzw. Ausprägungen. Es ist sogar das Implementieren nicht ganz primitiver Gestenfolgen möglich, indem eine Geste zeitlich und räumlich in verschiedene Segmente unterteilt wird. Dies sei an einem Beispiel erläutert: Es soll eine Winkgeste der rechten Hand erkannt werden. Die Geste wird in zwei Segmente geteilt. Ein Wechsel zwischen den Segmenten findet statt, wenn die horizontale Position der Hand und des Ellenbogens wechselt. Wird dieser Übergang dreimal in Folge erkannt, so wurde die Winkgeste präsentiert. Eine genauere Auseinandersetzung mit der Aufgabenstellung und allgemeinen Vorstellungen von intuitiven Gesten für die zu realisierenden Funktionalitäten zeigte jedoch auf, dass auch eine Segmentinteilung von Gesten für das Projekt nicht notwendig ist.

Stattdessen sind die gegebenen Aufgaben (ein Verschieben oder Rotieren per Geste) in ihrer Struktur simpel genug, um die verschiedenen Wirkungen mit diskreten Gesten zu erzeugen, d. h. es genügt die Erkennung einer Geste durch bestimmte Zustände der Kinect-Rohdaten zu einem einzigen Zeitpunkt. Um die Wirkung jedoch zu erzielen, ist natürlich auch eine Betrachtung der Geste über mehrere Frames notwendig.

Im Folgenden ist der im Projekt Verwendung findende Gestenkatalog erklärt. Dabei wird darauf eingegangen, was der Benutzer vorführen muss, damit die Geste erkannt wird und wie die Geste genutzt wird, um in der Anwendung die Kamera oder Objekte zu manipulieren:

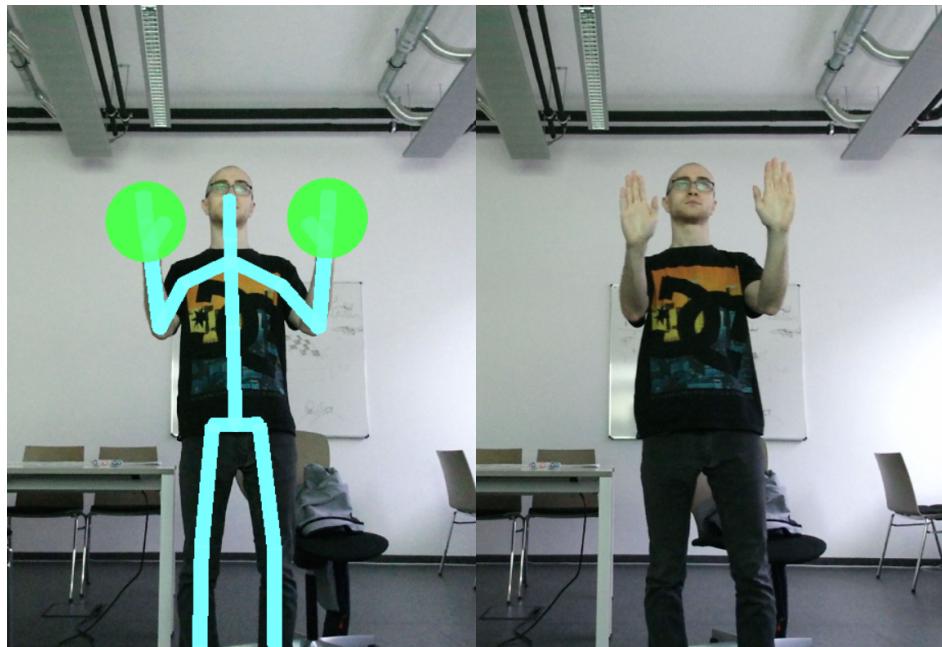


Abbildung 7: Die (Kamera-)Translations-Geste, mit und ohne eingezeichnetes Skelett und HandStates.

TRANSLATE_GESTURE (siehe Abb. 7)

Der Benutzer hat beide Hände geöffnet, mit den Handflächen zur Kamera (wichtig ist nur, dass die Kinect beide Hände als offen erkennt, die genaue Haltung ist dabei egal). Ein paralleles Verschieben der beiden Hände in eine Richtung bewirkt ein zur Bewegungsgeschwindigkeit proportionales Verschieben der Kamera in diese Richtung.

Wie in den Vorüberlegungen (Abschnitt 2) schon erwähnt ist dies eine einfache Übertragung des Prinzips nachdem eine Person in der Realität einen großen Gegenstand verschieben würde und wird daher der geforderten Intuitivität gerecht.

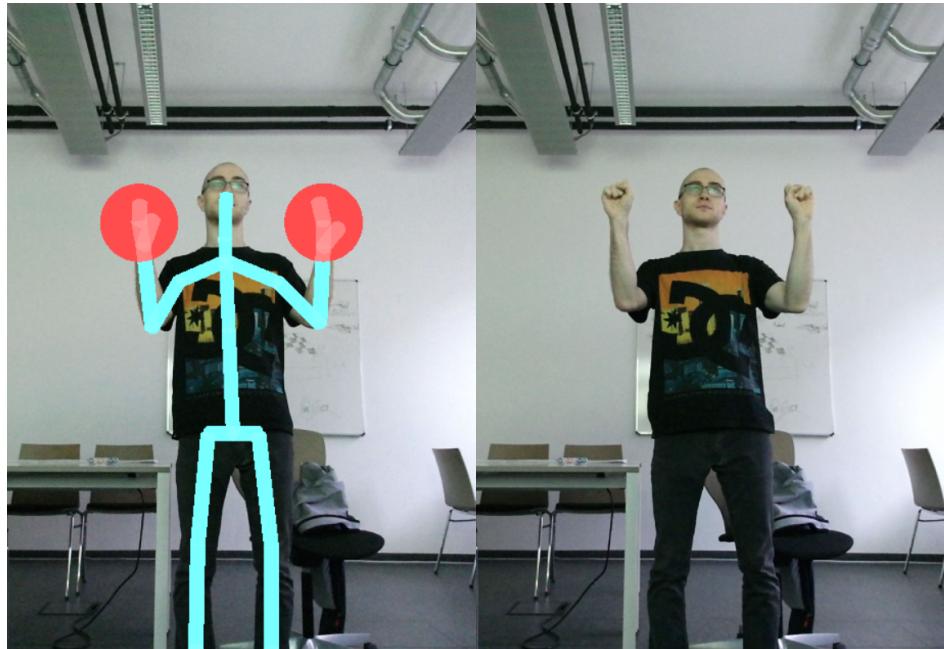


Abbildung 8: Die (Kamera-)Rotations-Geste, mit und ohne eingezeichnetes Skelett und HandStates.

ROTATE_GESTURE (siehe Abb. 8)

Der Benutzer hat beide Fäuste geballt. Dann bewirkt eine gleichzeitige Bewegung der Hände auf einer Kreisbahn eine Rotation der Kamera um die Senkrechte des zugehörigen Kreises. Analog zur Verschiebegeste fand diese Geste bereits in den Vorüberlegungen (Abschnitt 2) Erwähnung. Die Aufgabe, eine Rotation im Raum zu vollführen ist leider weniger alltäglich als die des Verschiebens von Gegenständen. Ein verwandtes Prinzip ist jedoch das des Lenkrades, das man sich – ins Dreidimensionale erweitert – als Lenkkugel vorstellen kann. Mittels der üblichen Lenkbewegung kann dann um beliebige Raumachsen rotiert werden. Ähnlich findet dies auch etwa bei der Bedienung von Münzfernrohren statt. Wegen dieser Analogien, kann davon ausgegangen werden, dass der Benutzer schnell ein intuitives Verständnis von der Wirkungsweise dieser Geste entwickelt.

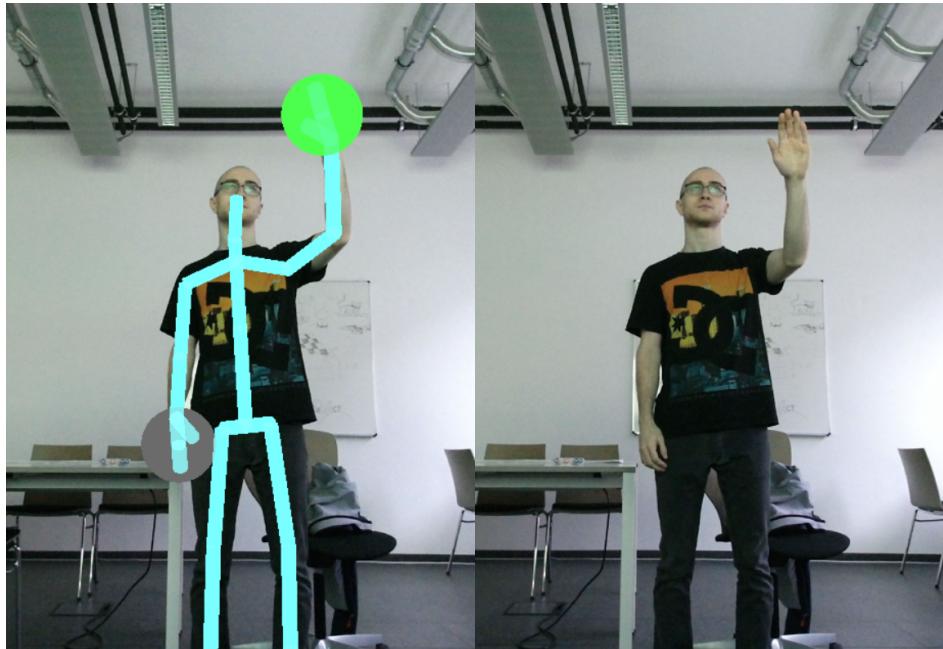


Abbildung 9: Die Objektmanipulations-Geste, mit und ohne eingezeichnetes Skelett und HandStates.

GRAB_GESTURE (siehe Abb. 9)

Zunächst war angedacht, dass die Objektmanipulation dieselben Gesten verwendet wie die Kameramanipulation und die Unterscheidung, was manipuliert wird durch einen globalen Zustand gefällt wird. Bei näherer Betrachtung dieses Ansatzes und ersten Tests dessen fiel auf, dass es so schwierig ist, zwischen Kamera- und Objektmanipulation zu wechseln. Weiterhin schien es während des Testens weniger intuitiv als zuvor angenommen, ein Objekt auf diese Art und Weise zu manipulieren. Es bedarf hier also anderer Ansätze.

In das Problem der Objektmanipulation eingeschlossen ist das Problem des Object-Pickings, d. h. die Auswahl des zu manipulierenden Objekts vom Bildschirm. Auch dies wäre mit der oben beschriebenen Methode, die die Gesten der Kameramanipulation verwendet, nur schwierig und umständlich realisierbar gewesen. Das Object-Picking bietet jedoch einen anderen Weg, einen Ansatz für eine Objektmanipulationsgeste zu finden. Ein alltägliches und dem Benutzer bekanntes Beispiel hierfür ist das einfache Greifen nach einem Gegenstand – dies motiviert auch den Namen „GRAB“-Geste. Übertragen in eine Geste ließe sich dies durch eine erhobene und geschlossene Hand definieren. In weiterer Analogie

zum Alltagsbeispiel sollte ein Hin- und Herbewegen der Hand, mit der gegriffen wurde (der „Kontroll-Hand“) auch das Objekt hin- und herbewegen. Die Rotation führt jedoch bei dieser Geste zu einem Problem: Mit der geschlossenen Hand ist die Erkennung der Orientierung des entsprechenden Gelenks durch die Kinect zu schlecht, um an dieser Stelle sinnvoll Verwendung zu finden. Im Programm äußerte sich dies einerseits durch ein Ausbleiben der Auswirkungen vorgeführter Rotationen, andererseits aber auch durch ein starkes Rauschen. Probleme dieser Art treten auch an anderer Stelle auf und sind behandelbar (dies wird in Abschnitt 3.4 deutlich), hier jedoch wurden die Ergebnisse so schlecht, dass eine andere Gestendefinition nötig war. Die Ergebnisse wurden direkt deutlich besser, wenn die GRAB-Geste durch eine gehobene und offene (!) Hand definiert wurde. Die größere Fläche bietet der Kinect mehr Anhaltspunkte und verbessert so die Genauigkeit für die Orientierung, etwa wenn die Handfläche gekippt bzw. gedreht ist.

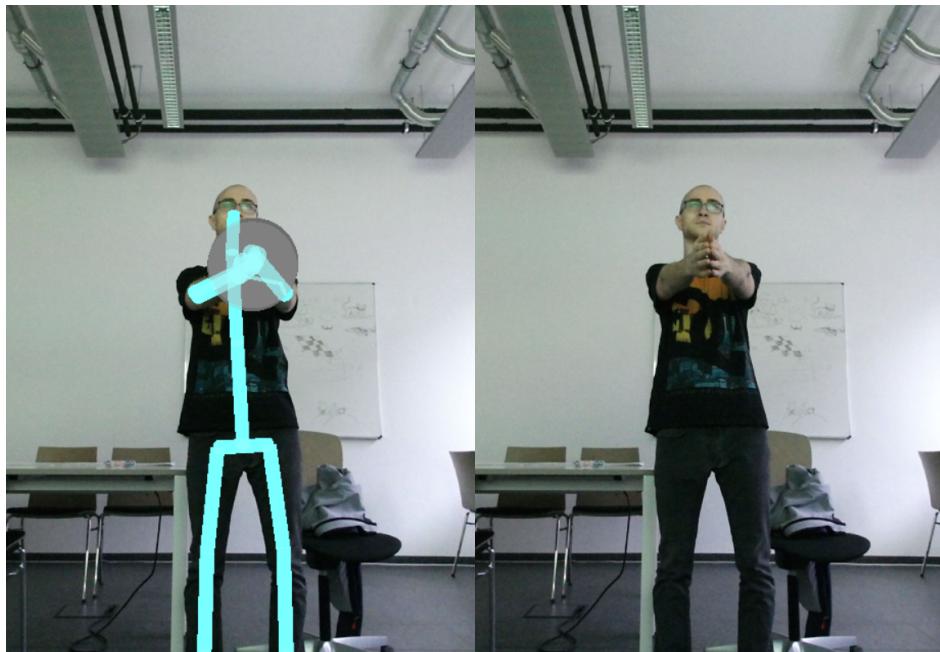


Abbildung 10: Die Flug-Geste, mit und ohne eingezeichnetes Skelett und HandStates.

FLY_GESTURE (siehe Abb. 10)

Im Rahmen der Tests mit einem Beispielobjekt wurde schnell deutlich, dass es auch eine einfache Möglichkeit geben sollte, Bewegungen über etwas weitere Stre-

cken durch den Raum zu vollführen, ohne dabei ständig zwischen dem Vorführen einer Geste und einem „Nachgreifen“ wechseln zu müssen. Eine übliche Lösung für eine solche Aufgabe ist ein Flugmodus. Im konkreten Anwendungsfall sollte dann das Vorführen einer besonderen Geste bewirken, dass die Kamera losfährt und erst anhält, wenn die Geste nicht mehr präsentiert wird.

Die FLY-Geste entspricht dem Ausstrecken beider Arme vor den Körper, sodass sich die Hände mehr oder weniger am selben Punkt im 3D-Raum befinden. Passiv findet bei dieser Geste im Programm eine Bewegung nach vorne statt. Durch Schwenken der Arme soll der Nutzer dabei die Richtung der Bewegung beeinflussen können, d. h. ein Zeigen der Arme nach oben bewirkt, dass die Bewegung immer weiter nach oben gezogen wird, während mit einem Zeigen nach links oder rechts eine Kurve geflogen werden kann. Dabei bestimmt der Ausschlag der Arme beim Zeigen (verglichen mit der Ausgangsposition, in der beide Arme genau nach vorne gerichtet sind) die Stärke der Richtungsänderung. Um eine sogenannte Fassrolle durchzuführen oder sich „in Kurven legen“ zu können, kann der Nutzer nebenbei seine Schulterpartie in die entsprechende Richtung kippen. Insgesamt ist die Steuerung des Flugmodus in ihrem Funktionsumfang damit ähnlich zu üblichen Steuerungen von Flugzeugen in Actionspielen oder Simulationen: Es findet eine automatische Bewegung nach vorne statt, die in verschiedene Achsen gekippt werden kann.

Diese erneute Prinzipübertragung macht die Geste intuitiver. Mit Hilfe dieser Fluggeste hat der Benutzer eine im Gegensatz zur Bewegung über Drehen und Schieben einfache Möglichkeit, sich etwa durch ein System von Gängen in einer 3D-Szene zu bewegen und generell Strecken zurückzulegen, statt Objekte zu betrachten. Werden die Strecken zu lang, kann diese Geste jedoch anstrengend für den Benutzer sein. Bei alternativen Anwendungen wäre dies zusätzlich zu bedenken und eine dem abweichenden Zweck besser angepasste Gestendefinition zu verwenden.

UNKNOWN Dies enthält alles, was als keine der anderen Gesten erkannt wird. Die Kamera und geladene Objekte sollen, solange diese Geste gezeigt wird, stillstehen. Neben der naheliegenden Motivation, dass der Nutzer die Szene gegebenenfalls auch bei Ruhe betrachten möchte, dient diese „Geste“ (oder besser „Nicht-Geste“) darüber hinaus noch einem anderen Zweck. Sie kann in andere Gesten, wie bei-

spielsweise Translationen, eingebaut werden, um diese aufzubrechen und „nachgreifen“ zu können. Erst dies gestattet dem Nutzer, während der Bedienung an Ort und Stelle stehen bleiben zu können.

Die Verwendung der Gesten hat ergeben, dass es notwendig ist, bei derartig selbst implementierten Gesten auch eigene Robustheitsmechanismen einzubauen, die die Gesteuerkennung gegen Schwankungen der Kinecterkennung (etwa des Status einer Hand) abhärten. Für genauere Informationen hierzu sei auf Abschnitt 3.4 verwiesen.

3.3 Zustandsmaschine

In der folgenden Abbildung ist die Zustandsmaschine des Programmes zu sehen. Im Weiteren wird auf die Zustände und ihre Übergänge eingegangen.

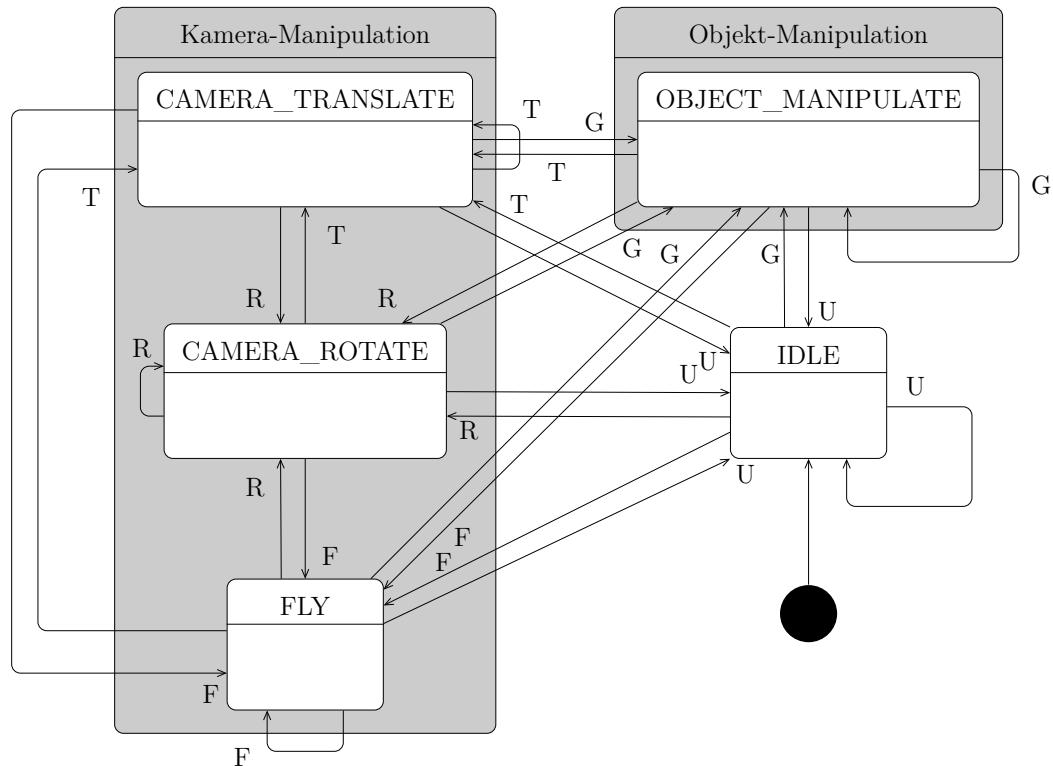


Abbildung 11: Die Zustandsmaschine. Die Beschriftungen T, R, G, F und U stehen für die verschiedenen Gesten aus Abschnitt 3.2: (T)RANSlate_GESTURE, (R)otate_GESTURE, (G)rab_GESTURE, (F)ly_GESTURE und schließlich (U)Nknown_GESTURE.

Das Programm besteht aus zwei Grundmodi der Manipulation: Einerseits der Manipulation der Kamera und andererseits jener des Objekts. Diese beiden Modi können als zwei Superzustände aufgefasst werden, innerhalb derer sich wiederum unterscheidet, auf welche Art und Weise manipuliert. Die Zustandsmaschine dient einerseits der Kapselung und Modularisierung der vom entstehenden Programm bereitgestellten Funktionen und bildet andererseits die Struktur der Manipulationsmodi abstrakt ab.

Die Zustandsmaschine befindet sich zu jedem Zeitpunkt in einem Zustand. In diesem Zustand findet eine Berechnung der Parameter statt, die das Programm zurückgibt. Diese Parameter beschreiben wiederum die Manipulation, die ausgeführt werden soll. Dies geschieht durch Auswertung der gesehenen Geste und die Berechnung entscheidender Größen, u. U. unter Einbeziehung der Werte vergangener Frames. Schließlich erfolgt basierend auf der präsentierten Geste ein Zustandswechsel am Ende eines Berechnungsschritts.

Im Folgenden sind die Zustände erklärt, ihre Semantik und die enthaltenen Berechnungen (vgl. dazu auch wieder Abbildung 11):

IDLE Dieser Zustand entspricht dem Initialzustand der Zustandsmaschine. Er ist eine Art Default-Zustand, in dem keine Kamera- und auch keine Objektmanipulation (genauer: keine Berechnung überhaupt) vorgenommen wird. Der Zustand wird betreten, wenn keine der vordefinierten Gesten sicher genug erkannt wurde. Durch Ausführung der entsprechenden Gesten gelangt der Benutzer zurück in die anderen Zustände.

CAMERA_TRANSLATE Dieser Zustand gehört zur Kameramanipulation. In ihm werden gemäß der oben erklärten Geste die Parameter zur Kamerabewegung bestimmt. Ziel ist die direkte Übertragung der Handbewegungen des Benutzers auf die Kamerabewegung, die im lokalen Raum der Kamera stattfindet. Dazu wird aus den gepufferten Positions値en von linker und rechter Hand die diskrete Ableitung berechnet, die uns ein Maß für die Geschwindigkeit der Bewegung liefert. Ebenso ergibt sich daraus die Richtung, in die die Hände bewegt wurden. Die Geschwindigkeit wird dann mit der vergangenen Zeit zwischen dem aktuellen und dem vorhergehenden Frame multipliziert, um wieder einen Entfernungswert zu erhalten. Daraus entstehen die Translationsparameter für die x -, y - und z -Richtungen, die für diesen Zustand die Rückgabeparameter definieren.

CAMERA_ROTATE Dieser Zustand gehört ebenfalls zur Kameramanipulation.

Analog zu oben wird hier die Rotation vorbereitet. Die Rotation ist konzeptionell nahe der Translation. Während die Geste aktiv ist, wird eine Achse zwischen linker und rechter Hand des Benutzers bestimmt. Bewegt der Benutzer die Hände, so verändert sich die Achse. Ein Quaternion, der die Achsen ineinander überführt, wird berechnet, und analog zur Translation als Geschwindigkeit interpretiert. Dieser wird proportional zur verstrichenen Zeit zum vorhergegangenen Frame verkleinert und auf die Kamerarotation angewandt.

OBJECT_MANIPULATE Die Objektmanipulation realisiert das einhändige Packen eines virtuellen Objektes mit einer einzigen Hand. Dabei werden Aspekte der Translation und Rotation abgewandelt verwendet. Die Bewegung der packenden Hand wird sehr ähnlich zur Kamera-Translation direkt übertragen. Im Gegensatz zu CAMERA_TRANSLATE wird diese allerdings auf das Objekt angewandt. Außerdem soll jede Rotation der packenden Hand auf das Objekt übertragen werden. Dazu wird Kinect::JointOrientation genutzt, welches die Orientierung der Hand im Raum als Quaternion bereitstellt. Es wird eine Art Differenz zwischen der Orientierung eines Frames und den gepufferten Vorgängerorientierungen gebildet. Dies findet statt, indem die Vorgängerorientierungen invertiert auf die aktuelle Orientierung angewandt werden. Wie zuvor kann diese Differenz als Geschwindigkeit interpretiert werden. Bei der Anwendung auf das Objekt muss jedoch beachtet werden, dass eine Drehung der Hand beispielsweise um ihre z -Achse nicht unweigerlich auch einer Drehung des Objektes um dessen z -Achse haben soll. Für natürliches Verhalten muss die Rotation selbst noch abhängig von Objekt- sowie Kamerarotation gedreht werden.

FLY Dieser Zustand wurde nachträglich eingeführt, als die Notwendigkeit eines Flug-Modus deutlich wurde. Er wird mittels Vorführung der FLY-Geste betreten und analog zu den anderen Zuständen verlassen. Der Flugmodus soll eine durchgehende Bewegung realisieren, ohne dass der Benutzer sich permanent bewegt. Sie bildet primär eine Alternative zum wiederholten Anwenden von CAMERA_TRANSLATE. Während die Geste aktiv ist, wird eine Translation mit festen Werten nach vorn vorgenommen. Außerdem ist es möglich, durch die Neigung der nach vorn gestreckten Arme zu lenken. Dazu wird die mittlere Handposition mit einem Referenzpunkt auf dem Körper des Benutzers verglichen. Die Abweichung

der Position von einer neutralen Ausgangsposition bestimmt die Lenkrichtung. Die Stärke der Auslenkung bestimmt dabei die Stärke der Rotation. Um schnellere Drehungen zu gewährleisten, werden starke Auslenkungen noch zusätzlich verstärkt. Analog zur CAMERA_ROTATE wird ein Quaternion berechnet, welcher die neutrale Position auf die gegebene Position abbildet. Er wird wie üblich angewandt. Außerdem wird die Neigung der Achse von linker Schulter zu rechter Schulter bestimmt. Diese wird in eine kontinuierliche Seitwärtsrolle übersetzt.

Zur Verdeutlichung sei darauf hingewiesen, dass von jedem Zustand zu jedem anderen übergegangen werden kann, wobei dieser Übergang lediglich anhand erkannter Gesten erfolgt: Wird eine unserer Gesten erkannt (Details siehe Abschnitt 3.4), so wird der zugehörige Zustand betreten. Da unser Programm darauf ausgelegt ist, während des Event-Loops einer Hauptanwendung zu laufen, besteht die Zustandsmaschine ab ihres Starts permanent (bzw. bis zum Ende der Hauptanwendung) und besitzt keinen Finalzustand. Genaueres zum Aussehen der State-Machine als Datenstruktur ist in Abschnitt 4.1 zu finden.

3.4 Robustheit und Pufferung

Wie vorangegangen festgestellt wurde, sind einige der zu implementierenden Mechanismen anfällig gegenüber qualitativ niedrigwertigen Kinectdaten. Die fortwährende Auseinandersetzung mit den verschiedenen Gesten in verschiedenen Situationen lieferte in Verbindung mit passend gewählten Testszenarien eine Liste kritischer Punkte, die unten aufgeführt ist. Dabei war entscheidend, wie sich die Handhabung des Programmes subjektiv für den Benutzer anfühlt – dies ist nicht gut mit Testdaten belegbar, sondern eher auf direktes Feedback des Steuernden angewiesen. Aus den so gewonnenen Erkenntnissen darüber, welche Kinect-Eigenschaften die Bedienbarkeit des Programmes aus Nutzersicht einschränken oder behindern, werden Mechanismen zu deren Behandlung entwickelt.

3.4.1 Auftretende Probleme

Nachfolgend sind zunächst die verschiedenen Probleme genannt. Dies sind Situationen und Szenarien, in denen die Kinect schlechtere Werte liefert. Es wird geschildert, wie der jeweilige Fehler entsteht und gegebenenfalls in welcher Hinsicht die resultierenden Daten schlecht sind.

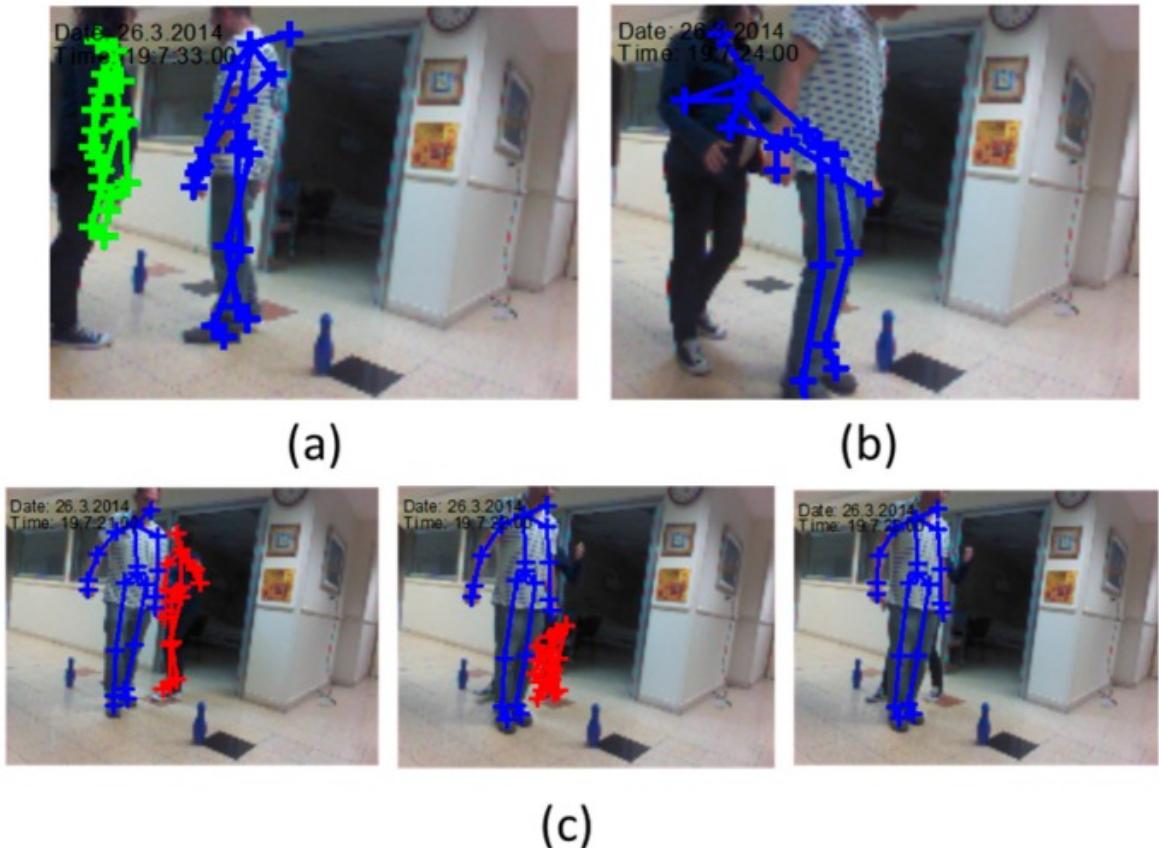


Abbildung 12: Diverse falsch erkannte Skelette: degenerierte Skelette (a), verschmolzene Skelette (b) und Skelettverlust bei Überdeckung (c). Quelle: [2]

- (i) **Gelenke in der Nähe von Objekten und anderen Körpern.** Diese können falsch oder verzerrt erkannt werden (siehe Abbildung 12). So kann etwa die erkannte Handposition zwischen zwei Kinectframes Raumunterschiede von mehreren Metern aufweisen und zurückspringen. Dies kann auch Körperteile betreffen, die vor oder hinter anderen Körperteilen liegen. Für hinter Körpern oder Objekten versteckte Teile ist klar, dass diese von der Kinect nur geraten werden können. Gliedmaßen, die sich vor Körpern (seltener Gegenständen) befinden können durch die Kinect-Optik zum Teil nicht hinreichend von den weiter hinten befindlichen Körperregionen differenziert werden. Das Problem verschärft sich mit zunehmender optischer und räumlicher Ähnlichkeit, d. h. es ist insbesondere vom Rückstrahlverhalten im Infrarotlicht und der Position von Personen im Raum (nah beieinander, verdeckend oder verstreut) abhängig. Ferner steigt die Unge-

nauigkeit, je weiter getrackte Personen vom „Abdeckbereich“ der Kinect (siehe 1.3) entfernt sind.

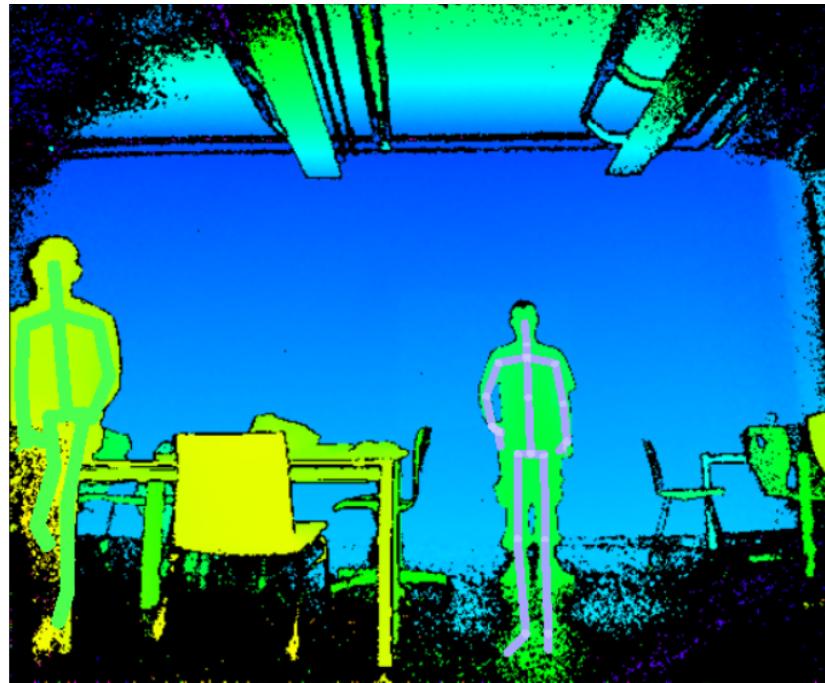


Abbildung 13: Im Sonderfall von Punkt eins beschriebene Situation eines Infrarotschatzens nebst zugewiesenum (fehlerhaften) Skelett.

Ein Sonderfall fehlerhafter Skelette erwächst aus der Lichtabhängigkeit der Kinect in Verbindung mit dem von ihr verwendeten Verfahren der Tiefenfeststellung, im Falle der Kinect 2 das Time-Of-Flight-Verfahren, das in Abschnitt 1.3 skizziert wurde. Zur Illustration siehe Abbildung 13. Die in den Raum gestrahlten Infrarotstrahlen können bei Spiegelungen unvorhergesehene Nebeneffekte bewirken. Nach Kenntnis dieser Problematik wurde diese Lichtabhängigkeit zusätzlich getestet, indem die Testperson Kleidung (hier Hosen) von hohem Infrarot-Rückstrahlvermögen trug. In dem in der Abbildung dargestellten Fall ist die Infrarotreflexion des Bodens und der Hose so groß, dass die Hose einen „Infrarotglanzfleck“ vor die Beine des Probanden wirft. Dies ist problematisch, da über die Zeitmessung festgestellt wird, dass dieser Bodenbereich eine ähnliche Tiefe aufweist, wie der Körper der Testperson. Daher wurden dem Skelett des Probanden beim Test unnatürlich lange Beine zugewiesen (vgl. erneut Abbildung

13 am unteren Bildrand). Der Boden in der Testsituation war dabei eine diffus reflektierende Fläche ohne Ähnlichkeit zu einem idealen Spiegel.

- (ii) **Status der Hände.** Auch bei durchgängiger Aufrechterhaltung eines Handzustands kann es passieren, dass die Kinect vereinzelt falsche Zuweisungen trifft (eine offene Hand wird etwa als Lasso erkannt) oder keine Zuweisung möglich ist (Handstatus unbekannt, obwohl augenscheinlich einer der anderen Status zutrifft). Besonders schlecht wird die Erkennung, wenn sich die Hände vor dem Körper befinden. Sind die Hände selbst vollständig oder auch nur teilweise verdeckt, ist selbstverständlich ebenfalls keine sinnvolle Erkennung des Handstatus möglich. Hier liegen weitestgehend dieselben Mechanismen zugrunde, die auch die Probleme von Punkt eins verursachen.
- (iii) **Jitter.** Die Kinectdaten sind verrauscht und weisen bspw. von Frame zu Frame kleine Ungenauigkeiten und Abweichungen der Gelenkpositionen in beliebige Richtungen auf. Diese Fehler nehmen ebenfalls umgekehrt proportional zur Kinect-Sicherheit zu, d. h. treten vermehrt auf, wenn die genaue Position nicht richtig erkannt wird, also besonders stark bei den Gelenken mit gefolgerten Positionsdaten (siehe Auflistung der Trackingstatus in 1.3). Insbesondere ist dies wieder bei Verdeckung (egal in welcher Reihenfolge) mit infrarot-optischer und räumlicher Nähe der Fall.

Die bereits in Punkt eins (Gelenke in der Nähe von Objekten / Personen) genannte Fehlersituation erzeugt teils ähnliche Effekte (ein „Zittern“ von Gelenkpunkten in verschiedenste Richtungen). Der hier unter „Jitterfehler“ aufgeführte Fehlerfall ist jedoch von Fehlern nach Punkt eins insofern abgegrenzt, dass hier es sich hierbei um ständig auftretende, von ihrer Natur her kleine Fehler handelt und die Probleme auch auftreten, wenn der betroffene Körper sich vollständig und unverdeckt im Kinect-Aufnahmebereich befindet.

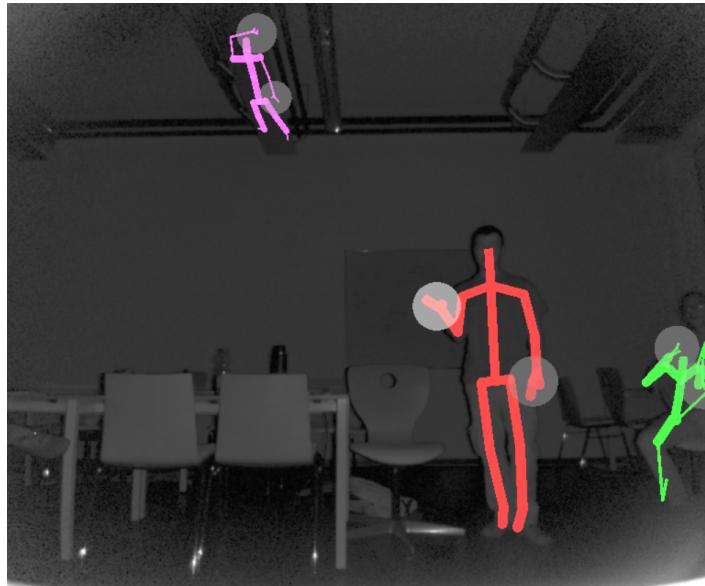


Abbildung 14: Infrarotbild der Kinect mit Phantomskelett (oben links, magentafarben, entlang der Lampe erkannt). Insbesondere werden Teile des Phantoms von der Kinect mit „hoher Konfidenz“ erkannt (verdeutlicht durch die dicken Linien, zu den „Konfidenzstufen“ siehe 1.3). Ferner verzerrtes Skelett am rechten Rand (verursacht durch Verlassen des Aufnahmebereichs).

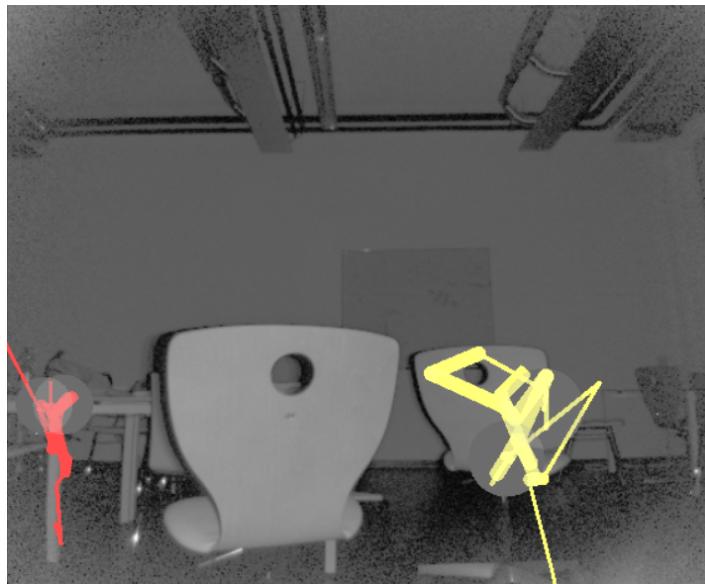


Abbildung 15: Infrarotbild der Kinect mit an Stühlen entstandenen Phantomskeletten.

- (iv) **Phantome.** Mitunter kann es passieren, dass ein ganzes Skelett an einem Gegenstand „hängenbleibt“, siehe Abbildungen 14 und 15. Dies geschieht, wenn eine getrackte Person einen Gegenstand passiert und dabei nicht korrekt erkannt wird – etwa weil sich die Situation sehr nahe am Rand des Aufnahmebereichs abspielt oder der Gegenstand eine nahezu humanoide Form hat. Sobald die Person wieder richtig erkannt wird, erhält sie ein neues Skelett. Die Skelettdaten dieser Phantome variieren über die Zeit sehr stark und willkürlich.
- (v) **Skelettzuweisung.** Besonders problematisch für Mastererkennungsmechanismen ist ein anderer Fall von Fehlerskeletten, der von der Problematik aus Punkt eins zu unterscheiden ist: In dem Augenblick, in dem eine neue Person im Aufnahmebereich erkannt wird, bekommt sie ein Skelett zugewiesen. Wie diese Zuweisung genau stattfindet, ist nicht bekannt, vermutlich fließen jedoch nichtdeterministische Faktoren ein. Vorstellbar wäre dies als Identifizieren des Körpers mit einem geschätzten Skelett. Das Problem hierbei ist, dass eine erneute Skelettzuweisung bei zwischenzeitlichem Verlust anders ausfallen kann und der Körper somit leicht andere Proportionen hat. Derartiges lässt sich während der Anwendung ohne die visuellen Debug-Möglichkeiten des Kinect Studios nur schwer verhindern. Der Effekt scheint sich jedoch mildern zu lassen, wenn die betroffenen Personen sich bewegen und der Kinect damit mehr Schätzungsmöglichkeiten und die Möglichkeit zur Korrektur bieten.

3.4.2 Auswirkungen aus Nutzersicht

Die in Abschnitt 3.4.1 genannten Punkte können gravierende Einschränkungen bezüglich der Programmbedienbarkeit mit sich ziehen.

- Eine fehlerhafte Erkennung von Positionen gemäß des ersten Punktes (Situation: Gelenke in der Objekt- / Personennähe) kann zu einem gänzlichen Verlust der gegenwärtigen Position im virtuellen Raum führen: Im naiven Ansatz der Direktauswertung der Daten wird ein hoher Differenzwert zwischen den die Bewegung (oder Drehung) bestimmenden Handpositionen festgestellt, der die Stärke der Manipulation bestimmt und demzufolge auch eine extrem starke Manipulation bewirkt. Bei der Arbeit mit der Objektsteuerung ist ein weiteres Fehlerszenario aufgefallen, welches als Sonderfall dieses Punktes betrachtet werden kann: Ist die Hand des Nutzers, mit welcher dieser das Objekt bewegt oder dreht, durch die Ki-

nect genau im Profil zu sehen, d. h. die Handfläche ist bezüglich der Höhenachse 90 Grad verdreht, so kann die Kinect nicht genau erkennen, in welche Richtung sie verdreht ist (m. a. W., ob sich der Daumen vorne oder hinten befindet). Solange dies der Fall ist, kann es passieren, dass die Kinect zwischen beiden Möglichkeiten hin- und herspringt, was sich in einer plötzlichen und äußerst starken Drehung widerspiegelt, der jedoch keinerlei bewusste Nutzereingabe zugrunde liegt.

- Das vorübergehende Verlieren (oder Missinterpretieren) des vorgeführten Hand-States (**Punkt zwei** von oben) äußert sich bei der Programmsteuerung dagegen in einem Stottern, d. h. dass die ursprünglich fortlaufend präsentierte Geste zu den Zeitpunkten der Fehlerkennung nicht wirkt und daher z. B. eine kontinuierlich angedachte Bewegung mehrfach abrupt unterbrochen wird. Siehe Abb. 12 für eine Illustration.
- Jitterfehler nach **Punkt drei** verursachen durch die vielen willkürlichen kleinen Bewegungen eine als „zittrig“ wahrgenommene Steuerung der Anwendung: So nimmt ein bewegtes Objekt etwa eine Vielzahl kleiner Bewegungen bzw. Drehungen in verschiedene Richtungen vor, ohne dass der Nutzer eine entsprechende Geste präsentiert hat.
- Die Erzeugung eines Phantoms (**Punkt vier**) ist vor allem dann kritisch, wenn das Phantom aus dem augenblicklichen Master hervorgeht. In diesem Falle übernimmt das Phantom die Programmsteuerung, wodurch einerseits der eigentliche Master die Kontrolle verliert, aber auch andererseits das Programm gänzlich chaotische Bewegungen und Drehungen vornehmen könnte. Letzteres ist wegen der Mechanismen zur Gestenerkennung jedoch unwahrscheinlich, da das Phantomskelett die entsprechenden, relativ strengen Constraints erfüllen müsste, um den Idle-Modus zu verlassen, nach seiner Erzeugung das Programm aber sehr schnell in den Idle-Modus bringt.
- Schließlich ist klar, dass Fehler nach **Punkt fünf** (Skelettneuzuweisung) die Mastererkennung erschweren, da der Master gegebenenfalls mit fehlerhaften Proportionen eingespeichert wird.

Diese Probleme üben einen negativen Einfluss auf die Erfahrung aus, die der Nutzer mit der Software macht. Insbesondere Fehler nach dem erstgenannten Schema können dem

Nutzer das Erreichen seines Ziels – etwa des Annavigierens eines Objektes – unmöglich machen. Die weiteren Punkte werden dagegen einfach als störend empfunden.

3.4.3 Behandlung

Die verschiedenen (und auch üblichen) von angewendeten Mechanismen, um diese Probleme zu beheben, sollen hier erklärt werden. Ein Großteil der genannten Schwierigkeiten ergeben sich aus den üblichen Problemen von Sensoren (korrekte Messung physikalischer Sachverhalte, hier zusätzlich mit Zeitanforderungen). Hinzu kommt, dass die Kinect als für ein breiteres Publikum ausgelegtes System über vergleichsweise preiswerte Sensoren verfügt (siehe 1.3). In diversen Arbeiten, die sich mit ähnlichen Problemstellungen beschäftigen (s. [1], [2], [14] und [16]), finden die Grenzen der Kinect häufig Erwähnung und ein wesentlicher Punkt in der Auseinandersetzung mit der Kinect und ihrer Anwendung in unserem und ähnlichen Szenarien widmet sich einer möglichst fehlerarmen Auswertung der Daten. In der Regel wird dabei auf Verzerrungen und schlechte Werte eingegangen, die sich durch den eingeschränkten „Abdeckbereich“ der Kinect und den Einfluss von Licht ergeben (siehe [2] und [14]). Ferner wird darauf hingewiesen, dass das Detektions- und Trackingproblem generell von Beleuchtung, Blickwinkel, Distanz und weiteren Faktoren abhängt (vgl. [16]). Es ist jedoch zu beachten, dass sich die Betrachtungen in den diversen wissenschaftlichen Arbeiten z. T. noch auf die Kinect-Version 1 beziehen, die über teilweise andere Hardwarespezifikationen und Verfahrenstechniken verfügte (siehe 1.3).

Ferner ist ein gerade für das vorliegende Projekt wichtiger Punkt die Abhängigkeit der Kinect-Daten von der Pose, was etwa bereits in [1] festgestellt wurde: Die Autoren haben dort Unterschiede bei Messungen verschiedener Posen festgestellt, die statistisch signifikant waren (s. ebd., S. 6). Generell ist es möglich, durch ungünstiges Verdecken von Körperpartien die durch die Kinect erkannten Gelenkpositionen zu verschieben. Im Test war es so möglich, eine Verschiebung des Genicks (bzw. des Gelenkpunktes zwischen Schultern und Kopf) um mehrere Zentimeter reproduzieren, indem die Hände vor dieser Stelle auf und ab bewegt werden. Besonders kritisch für Steuerungsaufgaben ist dies vor allem dann, wenn die Verdeckung nach dem Verschieben aufgehoben wird und der Gelenkpunkt an seine eigentliche Position „zurückschnappt“. Die zur Behandlung der diversen Fehler erfolgt nach zwei Grundmotiven. Dazu werden die Daten einerseits gemittelt und andererseits auf sinnvolle Bereiche beschränkt.

- Der ursprüngliche Ansatz, Pufferung und Mittelung, eliminiert die jitterartigen Fehler, mit denen die Kinect-Daten belastet sind. Hierzu wird ein Puffer vorher festgelegter Länge verwendet und während des Programmablaufs mit den für den Anwendungszweck wichtigen Daten, hier den Handpositionen des Nutzers gefüllt. Wenn schließlich die Rückgabeparameter für die Manipulationen bestimmt werden sollen, wird dieser Puffer ausgewertet. Dabei wird ein exponentiell gewichtetes Mittel der gepufferten Positionen gebildet. Die neuesten Puffereinträge werden am stärksten gewichtet. Die Pufferlänge wurde genau so angelegt, dass das dadurch erzeugte Delay dem Nutzer nicht unangenehm auffällt und gleichzeitig die Kontrolle über das Programm per Geste steuerung wesentlich glatter und angenehmer erfolgen kann.
- Die eben beschriebene Glättung mag zwar kleine Jitterfehler behandeln, versagt jedoch bei Kinect-Daten, die sehr stark von den eigentlichen Realdaten abweichen (vgl. hierzu erneut die Auflistung von Problemsituationen aus Abschnitt 3.4.1). Ein Beispiel für dieses immer wieder auftauchende Problem ist etwa ein weiterer Nutzer der sich im Hintergrund des steuernden Nutzers bewegt. In einem solchen Fall (und ähnlichen Fällen) kann es passieren, dass die Kinect Körperteile dieses zweiten Nutzers falsch interpretiert und dem Steuernden zuordnet. Dadurch können z. B. Positionsdaten entstehen, die um mehrere Meter von der Realität abweichen. Diese Fehler benötigen eine eigene Ausreißerbehandlung: Werte, die eine zu große Abweichung von den zuletzt ermittelten Werten aufweisen (etwa eine Änderung der Handposition um mehrere Meter in aufeinanderfolgenden Frames) und daher unplausibel sind, werden auf eine vordefinierte Maximalabweichung abgeschnitten. Ohne eine solche Behandlung hätten diese Ausreißer dazu führen können, dass der Nutzer seine aktuelle Position in der 3D-Welt ohne sein Zutun mit großer Geschwindigkeit verlässt (falls er sich etwa im Kamerabewegungsmodus befand).

Die genannten Methoden bieten Robustheit, was fehlerhafte Kinect-Daten hinsichtlich der Position von Joints (Gelenkpunkten) angeht. Dies ist jedoch nicht der einzige Aspekt der Anwendung, der fehleranfällig ist und solche Sonderbehandlungen verlangt. Es wurde im Vorfeld ein weiterer solcher Punkt genannt – die Erkennung der Handzustände. Der naive Vorgang, die Handzustände diskret zu erkennen und auszuwerten, würde bei der gegebenen Zustandsmaschine zu sofortigen Zustandswechseln führen.

Eine Verbesserung dessen ist, statt der diskreten Gestenerkennung Fuzzy-Gesten zu implementieren. Hierbei findet die Erkennung nicht nach dem Schema statt, dass eine konkrete Geste als erkannt zurückgegeben wird, sondern ein Tupel von Konfidenzwerten für die verschiedenen Gesten. Diese Konfidenzwerte ergeben sich aus dem Zustand, in dem sich die Zustandsmaschine befindet und den rohen Kinect-Daten für die Hände. So ist beispielsweise das folgende Szenario denkbar: Der Benutzer ist dabei, die Kamera zu verschieben, als eine seiner Hände fehlerhaft als geschlossen erkannt wird. Diese „Geste“ ist so nicht definiert und mehrdeutig, es kann jedoch aufgrund des State-Machine-Zustands davon ausgegangen werden, dass der Nutzer wahrscheinlich weiterhin dabei ist, die Kamera zu verschieben – es wird jedoch auch eine Restwahrscheinlichkeit für die Rückkehr in den IDLE-Zustand eingeräumt.

Die genannten Konfidenzwerte addieren sich über das Tupel zu 1 und können als Maß für das Vertrauen darin gesehen werden, dass die jeweilige Geste präsentiert wurde. Dies wird in Verbindung mit einem Puffer verwendet, der nach dem selben Prinzip wie oben funktioniert und kurzzeitige Hand-State-Fehler entfernen soll. Der Puffer wird genau wie oben ausgewertet (gewichtete Mittelwertbildung) und das erhaltene Mittelwerttupel gibt entsprechend seines Maximalenintrags die erkannte Geste wieder, die dann einen Zustandswechsel bewirken kann.

Abschließend sei noch ein Szenario genannt, gegen das die Robustheitsmechanismen keinen hinreichenden Schutz bieten: Das der gezielten Manipulation. Wie bereits bemerkt, sind die Kinectdaten z. T. ungenau, etwa bezüglich der Gelenkpositionen. Durch (gegebenenfalls bewusstes) Verdecken oder Unkenntlichmachen von Körperteilen ist es möglich, die von der Kinect erkannten Jointkoordinaten zu verschieben. Dies kann durch den Einsatz von der Hände und der Körperhaltung, aber auch z. B. durch weite Kleidung hervorgerufen werden. Ein Beispiel ist die im Rahmen der Mastererkennung verwendete Torsolänge: Ein Nutzer, der von der Kinect getrackt wird, kann die an ihm erkannte Torsolänge (genauer den Abstand zwischen den entsprechenden erkannten Gelenkpunkten) verändern, indem er sich beispielsweise streckt und „groß macht“ (dies verlängert die erkannte Torsolänge) oder aber sich leicht nach vorne beugt (was die erkannte Torsolänge staucht). Dies eröffnet ihm einen Spielraum bezüglich des genannten Merkmals, in dem er die vom eingespeicherten Master bekannte Torsolänge annähern kann. Ähnliches ist für andere Körperpartien reproduzierbar, etwa durch leichtes Beugen der Arme oder Anheben und Hängenlassen der Schultern. Eine weitere,

aber weniger relevante Möglichkeit ist auch das Ausnutzen von Kinect-Ungenauigkeiten am Rande ihres Aufnahmebereiches, z. B. weit weg von der Kamera. Ihre kleinere Relevanz liegt in der Schwierigkeit begründet, *bewusst* diverse Effekte hervorzurufen, da die Randungenauigkeiten aus Anwendersicht willkürlich und ohne Muster sind.

Für Nutzer, die sich ohnehin aufgrund ihrer Körpermerkmale recht ähnlich sind, ist es bei solchen wie eben beschriebenen Manipulation schließlich nicht mehr sicher möglich, eine korrekte Entscheidung zu fällen. Es sei jedoch noch einmal darauf hingewiesen, dass der Beeinflussungsspielraum relativ gering und damit nur für a priori ähnliche Skelette von Belang ist. Ferner ist es augenscheinlich unmöglich, die Manipulation ohne Debugausgaben der genauen Werte bewusst und gezielt durchzuführen.

4 Bemerkungen zum Quellcode

In diesem Abschnitt werden wesentliche Stellen bzw. Strukturen des Programmcodes erläutert. Dazu wird auf die verwendeten Datenstrukturen, Variablen und Funktionen eingegangen und ihr Zusammenspiel grob illustriert. Schließlich wird in diesem Abschnitt auch auf das Einbinden des Programmes zur Verwendung in fremder Software eingegangen.

4.1 Wichtige Datenstrukturen, Variablen und Funktionen

Auf dem folgenden Bild ist zunächst die Dateistruktur des Programmes zu sehen: Da-

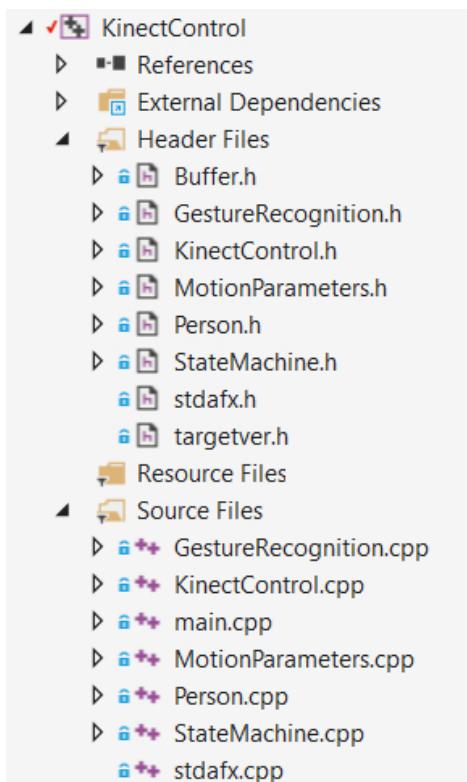


Abbildung 16: Ausschnitt aus dem Solution Explorer der Visual Studio IDE.

bei entsprechen die Header- und die ihnen zugehörigen .cpp-Dateien den wichtigsten Klassen. Diese sollen nun, sofern dies nicht in den vorangegangenen Kapiteln bereits geschehen ist, hinsichtlich ihrer Intention, ihrer Umsetzung und ihrer Verwendung erläutert werden. Die Reihenfolge, in der sie vorgestellt werden orientiert sich an ihren logischen Abhängigkeiten zueinander.

KinectControl. Dies ist eine Managerklasse, die sowohl als Einstiegspunkt für Aufrufe durch andere Programme dient, als auch zur Gesamtkoordination und Arbeit mit der Kinect Verwendung findet. Wegen ihrer zentralen Funktion, sind alle anderen Header-Dateien in dieser Klasse direkt oder indirekt eingebunden. Insbesondere ist `Kinect.h` aus dem Kinect-SDK eingebunden, um überhaupt mit der Kinect arbeiten zu können.

`KinectControl` stellt eine `init()`-Funktion bereit, die die Datenstrukturen, die für die Kinect-Kommunikation gebraucht werden initialisiert und bereitet diverse Puffer durch Speicherallokation und Füllen mit Default-Einträgen vor. Dazu wird der `KinectSensor` geholt und „geöffnet“, d. h. seine Nutzung ermöglicht. Anschließend kann über den `KinectSensor` auf die `BodyFrameSource` des Sensors zugegriffen werden, mittels derer man dann durch Öffnen eines Readers den Stream der von der Kinect interpretierten Körperinformationen abgreifen kann. Wir sparen weitere technische Details der Funktion an dieser Stelle aus.

Weiterhin spielt die `run()`-Funktion der `KinectControl`-Klasse eine wichtige Rolle: Sie entspricht dem „Main Loop“ unseres Programmes. Alle Berechnungen und Aufrufe haben ihren Ursprung in ihrem Rumpf. Im Wesentlichen werden die wichtigen Bestandteile des aktuellen Zustands bzw. bisheriger Berechnungen ausgelesen und mit den von der Kinect erhaltenen neuen Daten abgeglichen oder verrechnet. Einerseits ist die Mastererkennung implementiert, zum anderen findet auch das Puffern der für die Steuerung wichtigen Handpositionen hier statt und schließlich wird von hier aus auch die Zustandsmaschine gesteuert.

```

42     GetDefaultKinectSensor(&kinectSensor);
43     kinectSensor->Open();
44     kinectSensor->get_BodyFrameSource(&bodyFrameSource);
45     bodyFrameSource->get_BodyCount(&numberOfTrackedBodies);
46     bodyFrameSource->OpenReader(&bodyFrameReader);
    ↴

```

Abbildung 17: Ausschnitt aus der `init()`-Funktion.

Buffer. An dieser Stelle sei auf die Pufferklasse verwiesen. Sie folgt dem bekannten Schema und dient im Projekt lediglich als Werkzeug. Auf Details sei deshalb verzichtet. Es gibt folgende Funktionen:

- Einen Konstruktor, der einen Puffer fester Größe erzeugt; dazu einen Destruktor.

- Diverse Zeiger (`begin`, `end` und `next`).
- Eine Push-Funktion, sowie eine Funktion zum Leeren des Buffers.
- Abfragen auf Gefülltheit und Elemente an gegebenen Positionen.

Es sei angemekkt, dass die Push-Funktion eine Ringpufferfunktionalität implementiert, d. h. bei vollem Puffer wird wieder von vorne beginnend überschrieben.

MotionParameters. Dies ist eine Hilfsklasse, die die zentrale Sammlung an Informationen kapselt: Die Bewegungsparameter. Ein `MotionParameters`-Objekt besteht aus drei Floats, die eine Bewegung in die drei Achsenrichtungen beschreiben, einem Quaternion, der die Rotation enthält und einem „`MotionTarget`“ – einem booleschen Wert, der angibt, ob die Kamera oder ein Objekt manipuliert wird. Darüber hinaus extistieren eine Vielzahl an Gettern und Settern für einzelne oder auch mehrere Klassenvariablen, da es z. B. zweckmäßig ist, die Translationsparameter zusammen setzen zu können (vgl. Zeile 12 in Abb. 18).

```

1  #pragma once
2  #include "Eigen/Dense"
3
4  class MotionParameters {
5  public:
6      enum MotionTarget {
7          TARGET_OBJECT = false,
8          TARGET_CAMERA = true
9      };
10
11     void setMotion(float translateX, float translateY, float translateZ, Eigen::Quaternionf rotate, MotionTarget target);
12     void setTranslation(float translateX, float translateY, float translateZ);
13     void setRotation(Eigen::Quaternionf rotate);
14     void setTarget(MotionTarget target);
15     void resetMotion();
16     void resetTranslation();
17     void resetRotation();
18
19     float getTranslateX();
20     float getTranslateY();
21     float getTranslateZ();
22     Eigen::Quaternionf getRotation();
23     MotionTarget getTarget();
24
25     MotionParameters();
26 private:
27     float translateX;
28     float translateY;
29     float translateZ;
30     Eigen::Quaternionf rotate;
31     MotionTarget target; //0-verändere Model, 1-verändere Kamera
32 }
```

Abbildung 18: Inhalt der Header-Datei `MotionParameters.h`.

Person. Die Person-Klasse kapselt Informationen, die logisch zu einer von der Kinect erkannten Person gehören. Es werden so gut wie ausschließlich jene Informationen gespeichert, die wir im Laufe unseres Programmes auch benötigen. Zu einer Person gehören als wohl wichtigstes Element die Gelenkdaten der Kinect – ein Array von Joints. Ferner werden von diesen Gelenkdaten auch die Orientierungen, gespeichert in einem separaten Array, benötigt. Zentral sind weiter die Puffer für die Positionen der linken und rechten Hand, ein Puffer für die per Geste vorgeführten Rotationen, sowie die HandStates der beiden Hände und eine „ControlHand“ für die einhändige Objektmanipulation. Dazu wird eine ID für die Person verwendet und ihre z -Koordinate verfolgt.

```
--  
18  □ Person::Person()  
19  {  
20      id = -1;  
21  
22      leftHandCurrentPosition = { 0,0,0 };  
23      rightHandCurrentPosition = { 0,0,0 };  
24  
25      leftHandLastPosition = { 0,0,0 };  
26      rightHandLastPosition = { 0,0,0 };  
27  
28      leftHandState = HandState_Unknown;  
29      rightHandState = HandState_Unknown;  
30  
31      z = FLT_MAX;  
32  
33      // Handpositionenbuffer  
34      leftHandPositionBuffer = new Buffer<CameraSpacePoint>(POS_BUFFER_SIZE);  
35      rightHandPositionBuffer = new Buffer<CameraSpacePoint>(POS_BUFFER_SIZE);  
36  
37      // Rotationenbuffer  
38      rotationBuffer = new Buffer<Eigen::Quaternionf>(ROT_BUFFER_SIZE);  
39
```

Abbildung 19: Ausschnitt aus dem Person-Konstruktor mit Initialisierung der wesentlichen Merkmale.

Die weiteren Klassenvariablen dienen verschiedenen Zwecken, unter anderem stellen sie die Strukturen und Funktionen bereit, die später bei der Mastererkennung dienlich sind. Zentral für die Erkennung ist hierbei die Vermessung der gezeigten Körperproportionen.

```
65  □ enum BODY_PROPERTIES {  
66      LEFT_UPPER_ARM_LENGTH, RIGHT_UPPER_ARM_LENGTH, LEFT_LOWER_ARM_LENGTH, RIGHT_LOWER_ARM_LENGTH,  
67      SHOULDER_WIDTH, HIP_WIDTH, TORSO_LENGTH, NECK_TO_HEAD, NUMBER_OF_BODY_PROPERTIES  
68  };
```

Abbildung 20: Körperproportionen, die zur Erkennung einer Person gespeichert werden

GestureRecognition. Hierbei handelt es sich erneut um eine Hilfsklasse. Enthalten sind zunächst die Grundstrukturen für die Arbeit mit selbstdefinierten Gesten. Neben der Definition dieser Gesten selbst als Enumeration ist hier die Struktur „`GestureConfidence`“ gespeichert. Ein Behälter, der für die verschiedenen Gesten Floats enthält, die angeben werden, wie sicher eine Erkennung der zugehörigen Geste war. Die auf Abb. 21 ebenfalls zu sehende Enumeration `ControlHand` ist wieder für die einhändige Objektsteuerung nötig, als Angabe, welche Hand steuert.

```
- 4  class GestureRecognition {
5  public:
6  enum Gesture {
7      UNKNOWN,
8      TRANSLATE_GESTURE,
9      ROTATE_GESTURE,
10     GRAB_GESTURE,
11     FLY_GESTURE
12 };
13
14 struct GestureConfidence {
15     float unknownConfidence;
16     float translateCameraConfidence;
17     float rotateCameraConfidence;
18     float grabConfidence;
19     float flyConfidence;
20 };
21
22 enum ControlHand {
23     HAND_LEFT = 0,
24     HAND_RIGHT = 1
25 };
```

Abbildung 21: Ausschnitt aus dem `GestureRecognition`-Header.

Die Hauptfunktion dieser Klasse ist schließlich, neben dem Rahmen der Struktur `GestureConfidence`, für genau diese Konfidenzen einen Puffer nebst Auswertungsfunktion bereitzustellen, welche schließlich anhand der gepufferten Konfidenzen eine Endkonfidenz pro Geste bestimmt und so letztendlich die vermutlich vorgeführte Geste ermittelt, vergleiche Abb. 22. Dies ist die Geste mit der höchsten ermittelten Konfidenz. Die eben genannte „Ermittlung“ ist dabei einfach eine Glättung der Pufferdaten, Näheres siehe Abschnitt 3.4.

StateMachine. Die Zustandsmaschine ist das Muster, das die Gestensteuerung implementiert. Dem zugrunde liegt die natürliche Abbildung von Steuerungsmodus auf Zustand: Die Zustandsmaschine ist stets gemäß präsentierter Geste in genau einem

```

58     //Ergebniskonfidenz der Auswertung
59     GestureRecognition::GestureConfidence finalConfidence = { 0,0,0,0 };
60
61     //Gehe durch den Puffer und glätte alle Komponenten
62     for (int i = 0; i < GESTURE_BUFFER_SIZE; i++) {
63         float iSmoothFactor = gestureSmooth[i] / gestureSmoothSum;
64         currentConfidence = *confidenceBuffer->get(i);
65         finalConfidence.flyConfidence += currentConfidence.flyConfidence * iSmoothFactor;
66         finalConfidence.grabConfidence += currentConfidence.grabConfidence * iSmoothFactor;
67         finalConfidence.translateCameraConfidence += currentConfidence.translateCameraConfidence * iSmoothFactor;
68         finalConfidence.rotateCameraConfidence += currentConfidence.rotateCameraConfidence * iSmoothFactor;
69         finalConfidence.unknownConfidence += currentConfidence.unknownConfidence * iSmoothFactor;
70     }
71
72     //Werte aus, welche Komponente den höchsten Konfidenzwert hat
73     float maxConfidence = finalConfidence.unknownConfidence; Gesture maxConfidenceGesture = UNKNOWN;
74     if (maxConfidence < finalConfidence.flyConfidence) { maxConfidence = finalConfidence.flyConfidence; maxConfidenceGesture = FLY_GESTURE; }
75     if (maxConfidence < finalConfidence.grabConfidence) { maxConfidence = finalConfidence.grabConfidence; maxConfidenceGesture = GRAB_GESTURE; }
76     if (maxConfidence < finalConfidence.translateCameraConfidence) { maxConfidence = finalConfidence.translateCameraConfidence; maxConfidenceGesture = TRANSLATE_GESTURE; }
77     if (maxConfidence < finalConfidence.rotateCameraConfidence) { maxConfidence = finalConfidence.rotateCameraConfidence; maxConfidenceGesture = ROTATE_GESTURE; }
78
79     setRecognizedGesture(maxConfidenceGesture);

```

Abbildung 22: Ausschnitt aus der `evaluateGestureBuffer`-Funktion.

ihrer Zustände und wertet die gesehenen Daten zu Parametern aus. In der `run()`-Methode von Kinect-Control wird stets ein Berechnungsschritt, d. h. Berechnungen und Zustandswechsel, der State Machine durchgeführt.

Aus datenstruktureller Sicht ist das `State`-Struct zentral, das die bereits konzeptuell bekannten Zustände `IDLE`, `CAMERA_TRANSLATE`, `CAMERA_ROTATE`, `OBJECT_MANIPULATE` und `FLY` definiert. Wichtige Klassenvariablen sind der aktuelle Zustand „`state`“, die eingespeicherte Person „`master`“, sowie eine `GestureRecognition`-Instanz für die Gestenerkennung und die aktuellen berechneten Rückgabeparameter „`motionParameters`“. Darüber hinaus enthält sie zahlreiche Konstanten, die als Gewichte, Constraints oder empirisch ermittelte Verstärkungs- bzw. Abschwächungsfaktoren Verwendung finden. Für einen Arbeitsschritt der Zustandsmaschine ausschlaggebend sind die drei Funktionen `bufferGestureConfidence()`, `compute()` und `switchState()`. Diese Funktionen werden in genau dieser Reihenfolge im Main-Loop von KinectControl aufgerufen. In `bufferGestureConfidence()` wird aus den Kinectdaten unter Prüfung diverser Constraints eine Fuzzy-Geste in Form eines Konfidenztupels bestimmt und gepuffert. Dieser Puffer wird anschließend direkt ausgewertet und liefert die erkannte Geste, die in `recognizedGesture` gespeichert wird. Die `compute()`-Funktion errechnet daraufhin je nach aktuellem Zustand aus den gesehenen Skelettdaten und Hand-States entsprechend vorangegangener Erklärungen die Bewegungsparameter und legt sie gebündelt in `motionParameters` ab. Schließlich entscheidet `switchState()` anhand der `recognizedGesture` über den Folgezustand und nullt gegebenenfalls die `motionParameters` der `StateMachine`.

Das als Erweiterung der Aufgabenstellung eingeführte Eventsystem liegt aufgrund der Klassenhierarchie ebenfalls in der `StateMachine`-Klasse.

```
436     switch (recognizedGesture) {
437         case GestureRecognition::Gesture::ROTATE_GESTURE:
438             // Initialisiere den Rotationsbuffer mit (0,0,0,1)-Werten
439             for (int i = 0; i < Person::ROT_BUFFER_SIZE; i++) {
440                 rotationBuffer->push(Eigen::Quaternionf::Identity());
441             }
442             motionParameters.setTarget(MotionParameters::MotionTarget::TARGET_CAMERA);
443             newState = CAMERA_ROTATE;
444             break;
445         case GestureRecognition::Gesture::TRANSLATE_GESTURE:
446             motionParameters.setTarget(MotionParameters::MotionTarget::TARGET_CAMERA);
447             newState = CAMERA_TRANSLATE;
448             break;
449         case GestureRecognition::Gesture::GRAB_GESTURE:
450             if (currentState != OBJECT_MANIPULATE)
451                 master.setLastHandOrientationInitialized(false);
452             master.setControlHand(master.getRisenHand());
453             // Initialisiere den Rotationsbuffer mit (0,0,0,1)-Werten
454             for (int i = 0; i < Person::ROT_BUFFER_SIZE; i++) {
455                 rotationBuffer->push(Eigen::Quaternionf::Identity());
456             }
457             widget->pickModel(0, 0); // löst Ray cast im widget aus
458             motionParameters.setTarget(MotionParameters::MotionTarget::TARGET_OBJECT);
459             newState = OBJECT_MANIPULATE;
460             break;
461         case GestureRecognition::Gesture::FLY_GESTURE:
462             motionParameters.setTarget(MotionParameters::MotionTarget::TARGET_CAMERA);
463             newState = FLY;
464             break;
465         default: //Übergang zu IDLE, entspricht UNKNOWN
466             motionParameters.setTarget(MotionParameters::MotionTarget::TARGET_CAMERA);
467             newState = IDLE; //Zustand nicht wechseln
468             motionParameters.resetMotion();
469             break;
470     }
471 
472     setState(newState);
473     if (newState != currentState) motionParameters.resetMotion();
474 }
```

Abbildung 23: Ausschnitt aus der `switchState()`-Funktion.

Anmerkung: Die `pickModel()`-Funktion in Zeile 457 nutzt eine Stubfunktion, die in einer möglichen Erweiterung der Software zum Object Picking verwendet werden kann.

4.2 Ergänzungen zum Zusammenspiel und Ablaufskizze

In diesem Abschnitt wird der Ablauf des Programms skizziert. Dabei wird davon ausgegangen, dass KinectControl bereits instanziert und durch Aufruf der `init()`-Funktion initialisiert wurde. Weiterhin gilt die Vorstellung, dass die `run()`-Methode im Main-Loop der umgebenden Software ausgeführt wird. Genaueres zur Einbindung ist Abschnitt 4.3 zu entnehmen.

Sollte kein Master bestimmt sein, so werden für den Master zunächst eindeutige Defaultwerte für ID (der negative Wert -1) und z -Entfernung zur Kamera (der maximale Wert des Floatdatentyps) angenommen.

Dann wird versucht, über den `BodyFrameReader` der Kinect einen aktuellen Kinect-Frame abzugreifen. Sollte dies fehlschlagen, so werden die alten Bewegungsparameter einfach weiterverwendet. Im Falle des Erfolgs versuchen wir auf die Körpertracking-Daten der Kinect zuzugreifen, was im Fehlerfall analog behandelt wird. Gerade die erste Art von Fehlschlag (der Versuch, einen Kinect-Frame zu holen, obwohl kein solcher vorhanden ist) tritt dabei tatsächlich häufig ein, da die Kinect nur etwa 30 Frames pro Sekunde liefern, wohingegen der Tick des Main Loops üblicherweise deutlich darüber liegt.

Nachfolgend findet im Code die Masterbehandlung statt. Hier werden vier Fälle unterschieden, die gesondert erklärt werden sollen:

- Der weitere Programmablauf sei nun zunächst für den Fall geschildert, dass kein Master eingespeichert wird oder wurde. In diesem Falle ist der nächste relevante Schritt das Iterieren über die Körper, die die Kinect zurückgeliefert hat. Dabei holen wir uns die genauen Körperdaten, namentlich die Positionen und Orientierungen der Gelenke. In der genannten Situation (vor jeglicher Masterfestlegung) wird als Basislösung der primitive z -Test zur Festlegung einer Person, die das Programm steuert, verwendet. Dazu wird das Master-Objekt einfach mit den Werten des Körpers belegt, dessen Kopf den niedrigsten z -Wert aufweist.
- Das Programmverhalten ändert sich, wenn durch Betätigen der Einspeicher-taste X ein Master festgelegt werden soll. Eine entsprechende Abfrage im Main-Loop des unterliegenden Programmes ruft die `assignMaster()`-Funktion von `KinectControl` auf, die die für die Masterfestlegung wichtigen Parameter initialisiert. Vor allem werden die booleschen Variablen `masterDetermined` und `collectFrames` auf `true` gesetzt und je nach Länge des Tastendrucks eine Variable hochgezählt, die die Anzahl der Samples für die Vermessung bestimmt. Fall zwei bei der Masterbehandlung entspricht dann der Belegung der beiden Variablen `masterDetermined` und `collectFrames` mit `true`. In diesem Falle wird beim Iterieren über die Körper zunächst ein Skelett in Standardpose gesucht. Die ID dieses Skeletts wird gespeichert und die zugehörige Person soll als Master eingespeichert werden. Befinden sich mehrere Skelet-

te in Standardpose, so wird das indexmäßig erste im `trackedBodies`-Array für die Masterfestlegung verwendet. Nachdem die genannte ID und damit der künftige Master festgelegt ist, werden solange Körpermerkmale gesammelt, wie durch den Framezähler aus `assignMaster()` vorgegeben. Dazu wird die `collectBodyProperties()`-Funktion aus der `Person`-Klasse aufgerufen, die wie weiter oben ausgeführt ein festes Set an Körpermerkmalen puffert. Sollte der designierte Master dabei die Standardpose verlassen, wird die bisherige Sammlung mittels `deleteCollectedBodyProperties()` vollständig verworfen und die ID-Festlegung durch Standardposensuche vom Anfang erneut durchgeführt. War der Frame hingegen gut, wird die Anzahl noch zu sammelnder Frames dekrementiert. Sind genau so viele Frames gesammelt, wie durch `assignMaster()` vorgegeben wurde, so wird die Masterfestlegung durch Rücksetzen von `collectFrames` auf `false` und Aufruf von `calculateBodyProperties()` (ebenfalls aus der `Person`-Klasse) beendet. Konzeptuell erstellt letztere Funktion aus den vorher gesammelten Daten einen charakteristischen Vektor für die eingespeicherte Person, den sie im Master-Objekt ablegt.

- Der dritte Fall in Sachen Masterfestlegung schließt sich sodann an, wenn also ein Master bestimmt wurde (`masterDetermined` ist `true`), jedoch keine Samples mehr gesammelt werden müssen (`collectFrames` ist `false`), weil der charakteristischen Vektor bereits gebildet wurde und verwendet werden kann. Dann muss hinsichtlich des Masters nur noch etwas getan werden, falls der Master zwischenzeitlich aus dem Tracking verschwunden ist, was vor der Iteration über die Körper überprüft wird. In diesem Falle wird `searchForMaster` gesetzt. Für die 0-1-Flanke von `searchForMaster` wird ein spezielles `lostMaster`-Flag gesetzt. In der Iteration über die Körper wird dann nach Körpern in Standardpose Ausschau gehalten und für diese Körper eine Sammlung von Abweichungswerten zum charakteristischen Vektor des Masters aufgebaut. Auf Codeebene wird dies durch ein Array von Puffern, `deviationBuffer[]`, gelöst. Die Abweichungen werden durch Aufruf der `compareBodyProperties()`-Methode der `Person`-Klasse berechnet. Das korrekte Weitersammeln in einem angefangenen Puffer kann über die von der Kinect vergebene, eindeutige `trackingId` gesichert werden. Die Puffer haben eine feste Größe und werden, wenn sie voll – aktuell entspricht dies 20 Samples – sind, durch Aufruf von `evaluateDeviationBuffer()` gemittelt.

Ergebnis ist ein gemittelter Abweichungswert vom charakteristischen Vektor des Masters. Unterschreitet dieser eine empirisch festgelegte Grenze, nehmen wir an, das die präsentierte Person der Master ist und setzen das `master`-Objekt unserer Zustandsmaschine entsprechend.

- Als letzter Punkt in Sachen Masterbehandlung ist noch das Verhalten im Falle eines Masterverlusts (`lostMaster` ist dann `true`) zu klären. In diesem Falle werden sämtliche Bewegungsparameter und der Zustand der State-Machine zurückgesetzt. Außerdem wird `lostMaster` wieder auf `false` gesetzt, was den Effekt hat, dass die Variable tatsächlich genau die 0-1-Flanke von `searchForMaster` einfängt.

Im Programmtext und Ablauf folgt nach der Masterbehandlungsphase nun die Masterauslesung zur Steuerung des Programms. Sie findet statt, falls ein aktiver Master vorhanden ist. Von diesem werden sodann die Gelenke mit ihren Orientierungen und die Status der Hände ausgelesen. Die entsprechenden und wichtigen Merkmale werden im `master`-Objekt (z. T. nach Plausibilitätsüberprüfungen und gepuffert) abgelegt. Dann findet die Kernberechnung der Zustandsmaschine statt (siehe Abb. 24). In `bufferGestureConfidence()` wird aus den Joints und HandStates unter Verwendung eines Puffers ein Konfidenzvektor für die Gesten erstellt. In `compute()` findet die Berechnung der `motionParameters` entsprechend des aktuellen Zustands und der Gelenkdaten statt. Die Funktion `switchState()` nimmt schließlich anhand des Gestenkonfidenzvektors gegebenenfalls einen Zustandsübergang vor. Konzeptuelle Erläuterungen zur Funktionsweise dieser Funktionen sind in den vorangegangenen Abschnitten zu finden.

```
474  ****
475  * State-Machine-Berechnungsschritt
476  ****
477  stateMachine.bufferGestureConfidence();
478  stateMachine.compute();
479  stateMachine.switchState();
```

Abbildung 24: Der Code, der den Berechnungsschritt der State-Machine darstellt.

Am Ende der `run()`-Methode wird der Frame freigegeben und die bei `compute()` berechneten `motionParameters` zurückgegeben. Wie genau sie schließlich verwendet werden, entscheidet das unterliegende Programm.

4.3 Einbinden

Das vorliegende Projekt kompiliert eine statische Bibliothek (.lib). Diese ist (in Visual Studio) als „Additional Dependency“ für das Linking einzutragen.

Das Programm, das die Library verwendet, sollte mittels

```
KinectControl kinectControl;
```

ein KinectControl-Objekt anlegen und in seiner Initialisierung durch Aufruf seiner `init()`-Funktion mitinitialisieren:

```
kinectControl.init(this);
```

Dabei wird ein `ControlWidget` übergeben, für welches die Funktionen `pickModel()` für die Objektwahl sowie `sendEvent()` für das Versenden von Nachrichten implementiert sein müssen.

Im Main-Loop oder Event-Loop des Programms werden per

```
MotionParameters motionParameters = kinectControl.run();
```

die Parameter (Translation, Rotation und Ziel) geholt und können ausgewertet und entsprechend angewendet werden.

Weiterhin sollte `kinectControl.assignMaster()` irgendwie angestoßen werden können, z. B. über ein bestimmtes Tastendruck-Event.

5 Schlussbemerkungen

In diesem Abschnitt soll diskutiert werden, in welchem Umfang die entstandene Anwendung die an sie gestellten Anforderungen erfüllt und andererseits bemerken, wie die Kinect selbst überhaupt für eine solche Aufgabe geeignet ist.

5.1 Eignung der Software

Die erstellte Software erfüllt die Aufgabenstellung in einem zufriedenstellenden Maße: Es ist möglich, die 3D-Szene und enthaltene Objekte per Geste zu Steuerung und diese Steuerung technisch einfach durchzuführen. Nach Rückkopplung aus diversen Selbstversuchen hat sich gezeigt, dass das Steuerempfinden für den Nutzer in der subjektiven Wahrnehmung in einem guten Maße präzise ist, was die Übertragung der Handbewegungen auf Objekte oder die Kamera betrifft. Bei schlechten Daten (siehe Abschnitt 3.4.1) kann es dennoch passieren, dass die Steuerung zittert oder springt. Um dies zu minimieren sollte zusätzlich auf Kooperation des Nutzers gesetzt werden in dem Sinne, dass er z. B. nicht zu weite Kleidung wie offene Jacken trägt und die Hände bei der Steuerung vorzugsweise neben (und nicht vor) dem Körper hat. Dies kommt dann der Kinect bei der Erkennung entgegen. Bei der einhändigen Objektrotation ist der Einfluss der Kinectgüte am stärksten sprbar, vor allem beim Kippen nach vorne oder hinten. Die Mastererkennung funktioniert ebenfalls wie vorgesehen, es kann jedoch im ungünstigsten Falle passieren, dass für den Master ein fehlerhaftes Skelett mit falschen Proportionen beobachtet wird. Dann wird er gegebenenfalls nicht wiedererkannt und neu eingespeichert werden muss.

5.2 Eignung der Kinect

Wie sich durch das Projekt herausgestellt hat, ist die Kinect mit einigen Abstrichen bei der Datenqualität für 1-zu-1-Abbildungen geeignet. Demgegenüber steht aber ihre Verfügbarkeit, der einfache Zugang und der Preis (vgl. wieder 1.3).

Rückblickend wäre eine Steuerung durch diskrete Gestenerkennung ohne 1-zu-1-Zusammenhang einfacher zu implementieren gewesen, bspw. so, dass eine bestimmte Geste eine Bewegung vordefinierter Distanz in eine der Achsenrichtungen bewirkt. Dies würde zwar viele der Probleme aus Abschnitt 3.4 vermeiden, trotzdem bleibt eine Direktabbildung wegen der größeren Freiheiten angenehmer. Außerdem gibt es, während die

Bewegung einfach zu berwerkstelligen wäre, keinen natürlichen Ansatz einer solchen Gestenfindung für eine Rotation im Raum.

Trotz starker Glättung ist es nicht gelungen, den Jitter unter Erhalt der Echtzeitabbildung vollständig zu eliminieren, sodass in Resten noch für den Nutzer sichtbar ist, wenn auch nur in einem subtilen Maße.

Bei der Mastererkennung ist die größte Problematik die im vorangegangenen Abschnitt ?? genannte. Dazu kommt, dass es mit den hier entwickelten Methode schwierig ist, sehr ähnliche Personen zu differenzieren – wobei bessere Systeme z. B. bei der Präsentation eineriiger Zwillinge vermutlich auch an ihre Grenzen kommen. Hierzu bestand jedoch kein Zugriff auf passende Testpersonen. Mit größerem Aufwand lassen sich jedoch bessere Erkennungsquoten erzielen, vergleiche dazu etwa [18], [1] und [2]. Es gibt auch gänzlich andere Ansätze, wie etwa über den Gang (siehe [15]) oder das Gesicht (siehe [7]). In einem gewissen Rahmen findet dabei stets ein Abtausch zwischen Sicherheit der Erkennung und Laufzeit statt, der je nach Anwendung zu berücksichtigen ist.

An dieser Stelle sei damit geschlossen, dass die Kinect für den gegebenen Zweck verwendet werden kann, aber einiges an Nachbehandlung der Daten erfordert. Zu sicherheitsrelevanten Zwecken sollte der Einsatz höherwertiger Systeme in Betracht gezogen werden.

5.3 Verbesserungen

Die entstandene Software wird den Anforderungen aus Abschnitt 1.2 und diversen Erweiterungen gerecht. Dennoch soll hier Ausblick auf potenzielle Verbesserungen gegeben werden. Zunächst wäre es in der Zukunft natürlich wünschenswert, wenn sämtliche Stubfunktionalität und der geplante Funktionsumfang implementiert würden, sobald das einbindende Grundprogramm sie unterstützt.

Die Parameter des Programms ließen sich in der Zukunft weiter optimieren. Eventuell sind dafür auch Tests in einem größerem Rahmen – z. B. mit etwa 50 Testpersonen – nützlich. Diese würden den zusätzlichen Zweck erfüllen, einen besseren Einblick in die Unterschiede zwischen verschiedenen Skeletten zu gewinnen und so bspw. neue Körpermerkmale für die Erkennung zu gewinnen oder weniger aussagekräftige auszusondern. Darüber hinaus bieten derartige Tests noch mehr individuelle Erfahrungsberichte, die

genutzt werden können, um die Intuitivität und das Eintauchen des Nutzers in die virtuelle Umgebung zu verbessern.

In der Hinsicht auf Programmparameter wären für den Anwender auch zusätzliche Schnittstellen nach außen wünschenswert, z. B. um die Genauigkeit der Mastererkennung von außen vorgeben zu können. Während der Arbeit war hier etwa im Gespräch, die Einspeicherung so lange fortzusetzen, bis die Standardabweichung aller gesammelten Daten unter eine vorgegebene Konvergenzschanke gerät und die erlaubte Abweichung damit anstelle der empirischen Feststellung präzise vorgegeben werden kann. Dies ist jedoch ohne Feedback für den Nutzer nicht sinnvoll möglich, da dieser sonst über keinerlei Wissen verfügt, wie lange die Masterfestlegung (noch) dauern wird. Daher sollte im Vorhinein ein komplett funktionstüchtiges Eventsystem und Möglichkeiten der Ausgabe in der Grundanwendung bereitstehen. Weiterhin löst dies nicht das Problem schlechter Skelettzuweisungen und es kann weiterhin passieren, dass Skelette mit falschen Proportionen eingespeichert werden. Zusammenfassend war hier das erwartete Aufwand-Nutzen-Verhältnis zu niedrig und es wurde nicht zuletzt aus Zeitgründen auf eine Implementierung verzichtet. Darüber hinaus gibt es Erweiterungen der Funktionalität, die die Bedienerfahrung verbessern würden. Dies betrifft etwa ein Zurücksetzen von Kamera und Objekten, ggf. per Geste; die Skalierung und das Löschen von Objekten mittels Gesten und sicher viele weitere. Einige solcher Wunschfunktionalitäten ergeben sich erst im Laufe der tatsächlichen Verwendung der Software, wie es etwa bei der hinzugekommenen Aufgabe, einen Flugmodus zu implementieren der Fall war.

5.4 Fazit

Trotz der Probleme der Kinect stellt das hier entwickelte Programm mit vertretbarem Aufwand eine verwertbare Softwarelösung der Aufgabenstellung und insbesondere für Präsentationen wie an einem Tag der offenen Tür einen Mehrwert dar.

Sofern hohe Genauigkeit notwendig ist oder die Anwendung irgendwelchen Sicherheitskriterien genügen muss, sei jedoch die Nutzung eines höherwertigen Trackingsystems zuungunsten der Kinect empfohlen.

Literatur

- [1] R. M. Araujo, G. Graña und V. Andersson. “Towards skeleton biometric identification using the microsoft kinect sensor”. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Zugriff 12.6.17. ACM. 2013, S. 21–26. URL: https://www.researchgate.net/profile/Ricardo_Araujo2/publication/237064051_Towards_Skeleton_Biometric_Identification_Using_the_Microsoft_Kinect_Sensor/links/0046351b1d1cc4c5a0000000.pdf.
- [2] G. Blumrosen u.a. “A Real-Time Kinect Signature-Based Patient Home-Monitoring System”. In: *Sensors (Basel)* (2016). Zugriff 12.6.17. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5134624/>.
- [3] *Kinect for Windows v2 sensor*. <https://blogs.msdn.microsoft.com/kinectforwindows/2014/06/05/pre-order-your-kinect-for-windows-v2-sensor-starting-today/>. Zugriff 18.8.2017.
- [4] P. König. “Kinect v2 für Windows”. In: *c’t* (2014). Zugriff 17.8.17. URL: <https://www.heise.de/ct/ausgabe/2014-17-aktuell-Kinect-fuer-Windows-2264919.html>.
- [5] D. Lau. *The Science Behind Kinects or Kinect 1.0 versus 2.0*. http://gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php. Zugriff 17.8.2017.
- [6] Microsoft. *Kinect für Windows 2.0 SDK*. <https://developer.microsoft.com/de-de/windows/kinect/tools>. Zugriff 12.6.17.
- [7] A.-A. Nagât und C.-D. Căleanu. “Face identification using Kinect Technology”. In: *2013 IEEE 8th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. Zugriff 3.8.17. IEEE. 2013, S. 169–172. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6608961&isnumber=6608938>.
- [8] Microsoft Developer Network. *Kinect Hardware*. <https://developer.microsoft.com/de-de/windows/kinect/hardware>. Zugriff 17.8.2017.
- [9] Microsoft Developer Network. *Skeletal Tracking*. <https://msdn.microsoft.com/en-us/library/hh973074.aspx>. Bezieht sich noch auf die Kinect 1, Zugriff 25.4.2017. 2017.

- [10] Microsoft Developer Network. *Tracking Users with Kinect Skeletal Tracking*. <https://msdn.microsoft.com/en-us/library/jj131025.aspx>. Zugriff 25.4.2017. 2017.
- [11] Microsoft Developer Network. *TrackingState Enumeration*. <https://msdn.microsoft.com/en-us/library/microsoft.kinect.trackingstate.aspx>. Zugriff 12.6.17. 2017.
- [12] Microsoft Developer Network. *Visual Gesture Builder in Context*. <https://msdn.microsoft.com/en-us/library/dn785529.aspx>. Zugriff 17.8.2017.
- [13] A. Pham. “E3: Microsoft shows off gesture control technology for Xbox 360”. In: *Los Angeles Times* (2009). Zugriff 17.8.17. URL: <http://latimesblogs.latimes.com/technology/2009/06/microsofte3.html>.
- [14] R. Seggers. “People Tracking in Outdoor Environments: Evaluating the Kinect 2 Performance in Different Lighting Conditions”. Zugriff 12.6.17. Bachelorarbeit. University of Amsterdam, 2015. URL: <https://staff.fnwi.uva.nl/a.visser/education/bachelorAI/thesisSeggers.pdf>.
- [15] A. Sinha, K. Chakravarty und B. Bhowmick. “Person Identification using Skeleton Information from Kinect”. In: *ACHI 2013: The Sixth International Conference on Advances in Computer-Human Interactions*. Zugriff 3.8.17. 2013, S. 101–108. URL: https://www.thinkmind.org/download.php?articleid=achi_2013_4_50_20187.
- [16] L. Susperregi u. a. “On the Use of a Low-Cost Thermal Sensor to Improve Kinect People Detection in a Mobile Robot”. In: *Sensors (Basel)* (2013). Zugriff 12.6.17. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3871095/>.
- [17] J. Walters. *Kinect v2 Joint Map*. <http://glitchbeam.com/2015/04/02/kinect-v2-joint-map>. Original nicht mehr verfügbar, gecacht abgerufen von <http://web.archive.org/web/20151031033003/http://glitchbeam.com:80/2015/04/02/kinect-v2-joint-map/>, Zugriff 4.7.2017.
- [18] M.-T. Yang und S.-Y. Huang. “Appearance-Based Multimodal Human Tracking and Identification for Healthcare in the Digital Home”. In: *Sensors (Basel)* (2014). Zugriff 3.8.17. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4179041/>.