Projektseminar: Gestensteuerung einer 3D-Anwendung mittels Kinect

Mario Janke
Peter Lindner
Patrick Stäblein

Inhaltsverzeichnis

1 Rahmenbedingungen

1.1 Grundlagen & Technik

Gegeben ist eine bereits vorhandene 3D-Anwendung, die zu Demonstrationszwecken genutzt wird. Innerhalb der Anwendung ist es möglich,

- Objekte zu laden und damit anzeigen zu lassen sowie
- die Kamera (bzw. Kameras) zu manipulieren, d. h. zu bewegen, zu rotieren und zu zoomen,

zusätzlich geplant ist später

- geladene Objekte manipulieren, in diesem Falle skalieren oder löschen zu können.

Das Programm rendert dabei zwei Ausgabefenster, in denen die Szene dargestellt ist, wobei die Kameras 3D-Aufbau bilden.
Die so beschriebene Ausgabe wird über zwei Projektoren von hinten auf eine Projektionsfläche geworfen – ein Projektor für die linke Kamera und einer für die rechte. Wird die „Leinwand“ von vorne durch eine Shutterbrille betrachtet, entsteht der 3D-Eindruck.
Die Steuerung der Anwendung erfolgt über Tastatur und Maus bzw. Präsentationspointer.

1.2 Aufgabenstellung

Ziel des Projektseminars ist es, die Steuerung der Anwendung hinsichtlich einer Präsentation vor einer Zuschauergruppe zu erleichtern und intuitiv zu gestalten, sodass parallel an der Universität vorhandene (und bislang ungenutzte) Technik verwendet und präsentiert werden kann. In diesem Sinne geeignet und vorgeschlagen sind

- ein professionelles Trackingsystem zum Tracken von Raumpunkten und
- die Verwendung einer Microsoft Kinect 2 zur Gestenerkennung.

Das damit entwickelte Programm soll Folgendes leisten:

- Es soll in der Lage zu sein, sämtliche Steuerung und Manipulation, die oben beschrieben wurde durchzuführen.

- Die Bedienung soll sehr intuitiv und einfach sein, d. h. etwaige Gesten müssen bezüglich der ihnen zugeordneten Aktion einleuchtend und leicht auszuführen sein.
- Das Programm soll möglichst einfach eingebunden und wiederverwendet werden können.

1.3 Eigenschaften der Kinect

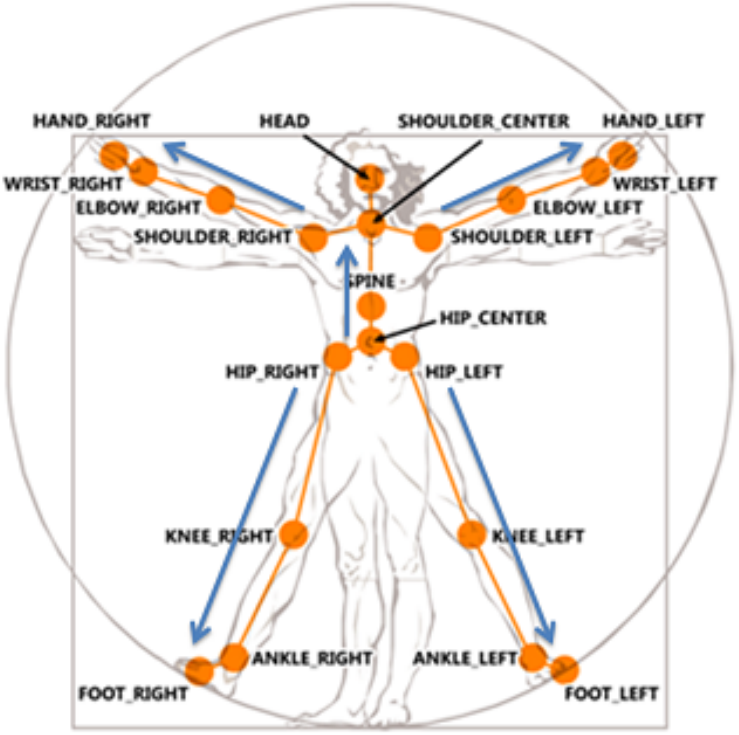

Abbildung 1: Gelenkpunkte die mit der Kinect getrackt werden können
Quelle: [1]

Wir stellen in diesem Abschnitt nur die für uns interessanten Eigenschaften und Möglichkeiten der Kinect vor (hinsichtlich unserer Aufgabe und der Rahmenbedingungen). Die Kinect erkennt visuell den 3D-Raum vor sich. Dabei werden Personen als solche detektiert und konfidenzbasiert mit einem primitiven und grobgranularen Skelett ausgestattet. Hierbei lassen sich die Koordinaten der oben abgebildeten Gelenkpunkte mit Hilfe des Kinect SDKs abfragen. Die praktische Reichweite für die Erkennung einer

Person beträgt etwa 1,2 bis 3,5 Meter. Dieses Tracking ist für bis zu sechs Personen zeitgleich möglich. Weiterhin wird für beide Hände einer getrackten Person ein „Handzustand“ erkannt, nämlich ob die Hand offen oder geschlossen ist, oder die sogenannte Lassogeste gebildet wird (etwa nur zwei Finger ausgestreckt). Kann einer Hand keiner dieser Zustände zugeordnet werden, ist ihr Status unbekannt. Diese Daten (Skelett und Status pro getrackter Person) können unter Verwendung der USB-Schnittstelle und des Kinect-SDKs abgegriffen werden. Sie werden dafür 30 mal in der Sekunde zur Verfügung gestellt.

2 Vorüberlegungen

Für unser Vorgehen zentral sind die folgenden beiden Bereiche:

1. die technische Umsetzung, d. h.
    - das korrekte Erkennen und Werten von Gesten einer ausgezeichneten getrackten Person
    - das korrekte Berechnen notwendiger Bewegungsparameter
    - die Einbindung in die bestehende Applikation
2. die Interaktion mit dem Benutzer, d. h.
    - das Entwerfen intuitiver und eingängiger Gesten für die verschiedenen Zwecke
    - das Auszeichnen einer getrackten Person als „Master“, der das Programm steuert

Wir stellen in diesem Abschnitt die zentralen unmittelbaren Beobachtungen vor, die sich aus der Aufgabenstellung und dem Versuchsaufbau ziehen lassen.

Ausgehend von der Aufgabenstellung kann man abstrahierend zwischen zwei primitiven Steuerungsmodi unterscheiden:

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

Der Benutzer sollte sich zu jedem Zeitpunkt nur in maximal einem dieser Modi aufhalten, d. h. gleichzeitige Kamera- und Objektmanipulation wird ausgeschlossen. Diese Vereinfachung treffen wir, da damit weniger komplexe Gesten benötigt werden und eine solche simultane Manipulation keine praktische Relevanz besitzt. Für Manipulationen, die man sowohl für die Kamera, als auch für Objekte haben will, bietet dies zudem eine geeignete Kapselung, da z. B. Rotationsparameter berechnet werden und dann nur entschieden werden muss, ob sie auf die Kamera oder ein Objekt angewendet werden, je nach Modus. Dies reduziert die Gesamtzahl nötiger Gesten.

Die Kinect ermöglicht ein Tracking des gesamten Körpers für mehrere (genauer sechs) Personen. Wir beschränken uns aus naheliegenden Gründen jedoch auf einen Teil dieses Spektrums:

- Wir benötigen nur eine Person, die die Anwendung (möglichst ungestört) steuert. Eine genauere Auswertung der restlichen Personen, ihrer Skelette etc. ist unnötig.
- Die in unserem Anwendungsfall intuitiven Gesten werden ausschließlich mit den Händen (bzw. Armen) durchgeführt.

Primitive Erkennungsmöglichkeiten eines Masters kann man etwa aus der Entfernung der getrackten Personen zur Kamera und der Position der Personen im Raum gewinnen. Genauere Erklärungen folgen weiter unten.
Intuitive Gesten für Verschiebungen imitieren das Verschieben eines großen Gegenstands, etwa einer imaginären Box, sodass hier etwa ein Verschieben der flachen Hand in der Luft naheliegt. Für eine intuitive Drehgeste eignet sich die Vorstellung eines imaginären Lenkrads, genauer gesagt einer Lenkkugel, bei der die Rotation um eine Raumachse nach dem Lenkradprinzip erfolgt. Eine intuitive Geste zur Objektauswahl ist offenbar eine Greifgeste.

2.1 Priorisierung der Aufgaben

2.2 Aufgabenverteilung im Team

Grob: Mario – Algorithmik und Berechnung; Peter – Backend, Struktur; Patrick – Master, Kinect-Basics; Anmerkung, dass wir zu viert angefangen haben.

3 Entwurfsentscheidungen

3.1 Der Master

Der Master ist die Person (unter den getrackten Personen), der es obliegt, die Anwendung zu steuern, d. h. in unserem 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.
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 nähesten 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, stießen wir hierbei auf verschiedene Möglichkeiten, von denen einige jedoch aufgrund ihrer Unpraktikabilität ausschieden (Erkennung anhand des Gangs oder anhand der Stimme würde bei unserer Anwendung keinen Sinn machen, da die Master-Person während der Bedienung kaum umher läuft und diese hierfür nicht zu sprechen braucht). Schließlich stiessen wir auch auf Verfahren die die Skelettdaten der Kinect zur Identifikation nutzen. Dies schien die für unsere Zwecke praktibalste Lösung zu sein, wenngleich unser Ansatz die Skelettdaten zu nutzen im Vergleich zu denjenigen in den gefundenen Arbeiten stark vereinfacht wurde.

3.2 Gesten und ihre Wirkung

Für die eingangs erwähnte Aufgabenstellung war es notwendig, bestimmte Programmfunktionalitäten mit Gesten zu verbinden. Einerseits hätte die Möglichkeit bestan-

den, 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 über seinen Gestenkatalog. 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. Dennoch entschieden wir schließlich uns kraft dieser Gegenüberstellung (nebst einigen Versuchen mit dem Visual Gesture Builder, die uns nicht von seinem Mehrwert in unserer konkreten Anwendungssituation überzeugen konnten) für eine direkte Gestenerkennung.
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 wechseln. Wird dieser Übergang dreimal in Folge erkannt, so wurde die Winkgeste präsentiert.
Eine genauere Auseinandersetzung mit der Aufgabenstellung und unseren Vorstellungen von intuitiven Gesten für die zu realisierenden Funktionalitäten zeigte jedoch auf, dass auch eine Segmenteinteilung von Gesten für das Projekt nicht notwendig ist. Stattdessen sind die gegebenen Aufgaben 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 erklären wir unseren Gestenkatalog und gehen dabei darauf ein, 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:

TRANSLATE_GESTURE 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.

Diese Geste war allen Projektteilnehmern unmittelbar einleuchtend und intuitiv und bedurfte keiner weiteren Diskussionen.

ROTATE_GESTURE 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. Wie die TRANSLATE-Geste war auch diese Geste von Anfang an unumstritten und alternativlos.

GRAB_GESTURE 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 mussten also andere Ansätze gefunden werden.

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. Wir näherten uns dem Finden eines neuen Weges diesmal auf einem anderen Weg, nämlich nicht über die Manipulation, sondern über das Picking des Objekts. Schnell einigten wir aus auf das Greifen eines Objekts (eine Hand ist erhoben und geschlossen – dies motiviert auch den Namen „GRAB“-Geste) als intuitivste Möglichkeit dafür. Von der Idee her sollte ein Hin- und Herbewegen dieser „Kontroll-Hand“ auch das Objekt hin- und herbewegen. Nachdem dies zufriedenstellend eingebaut war, widmeten wir uns der Objektrotation, was schnell

eine fundamentale Schwäche dieser Geste offenbarte: Die Rotation des Objekts sollte der Rotation der geschlossenen Hand folgen, jedoch ist die Erkennung der Rotation einer geschlossenen Hand durch die Kinect viel zu schlecht um an dieser Stelle sinnvoll Verwendung zu finden.

Die Ergebnisse wurden direkt sehr gut, als wir dazu übergingen, die GRAB-Geste durch eine gehobene und offene (!) Hand zu definieren, da die Kinect (wie auch naheliegend) viel besser erkennen kann, wie die Handfläche gekippt bzw. gedreht ist. Intern behielten wir jedoch den semantischen Namen „GRAB“-Geste bei.

FLY_GESTURE Im Rahmen der Tests mit einem Beispielobjekt wurde schnell deutlich, dass es auch eine einfache Möglichkeit geben sollte, sich über weitere Strecken durch den Raum zu bewegen, ohne dabei ständig zwischen dem Vorführen einer Geste und einem „Nachgreifen“ wechseln zu müssen. Als sinnvoll erschien hier, dass das Vorführen einer besonderen Geste bewirkt, 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. Durch Schwenken der Arme soll auch die Kamera während der Fahrt schwenken.

UNKNOWN Dies enthält alles, was als keine der anderen Gesten erkannt wird.

Tests mit der Kinect haben ergeben, dass es notwendig ist, bei derartig selbst implementierten Gesten auch eigene Robustheitsmechanismen einzubauen, die die Gestenerkennung gegen Schwankungen der Kinecterkennung (etwa des Status einer Hand) abhärten. Für genauere Informationen hierzu verweisen wir auf Abschnitt 3.4.

3.3 Zustandsmaschine

Das Programm besteht aus zwei Grundmodi der Manipulation: Einerseits der Manipulation der Kamera und andererseits jener des Objekts. Wir können diese beide Modi als zwei Superzustände auffassen, innerhalb derer sich wiederum unterscheidet, auf welche Art und Weise wir manipulieren. Die Zustandsmaschine dient einerseits der Kapselung und Modularisierung der von uns bereitgestellten Funktionen und bildet andererseits die Struktur unserer Manipulationsmodi abstrakt ab.
Die Zustandsmaschine befindet sich zu jedem Zeitpunkt in einem Zustand. In diesem Zustand findet eine Berechnung der Parameter statt, die unser Programm zurückgibt,

die wiederum die Manipulation beschreiben, 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. Die Zustandsmaschine ist in Abb. 3.3 zu sehen. Im Folgenden erklären wir die Zustände, ihre Semantik und die enthaltenen Berechnungen etwas näher:

IDLE Dieser Zustand entspricht dem Initialzustand unserer Zustandsmachine. 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 man 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. Wir berechnen dazu aus den gepufferten Positionswerten von linker und rechter Hand die diskrete Ableitung, die uns ein Maß für die Geschwindigkeit der Bewegung liefert. Ebenso erhält man daraus die Richtung, in die die Hände bewegt wurden. Aus diesen Größen berechnen wir Translationsparameter für die x-, y- und z-Richtungen, die für diesen Zustand unsere `motionParameters` definieren.

CAMERA_ROTATE Dieser Zustand gehört ebenfalls zur Kameramanipulation. Analog zu oben wird hier die Rotation vorbereitet.

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.

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.

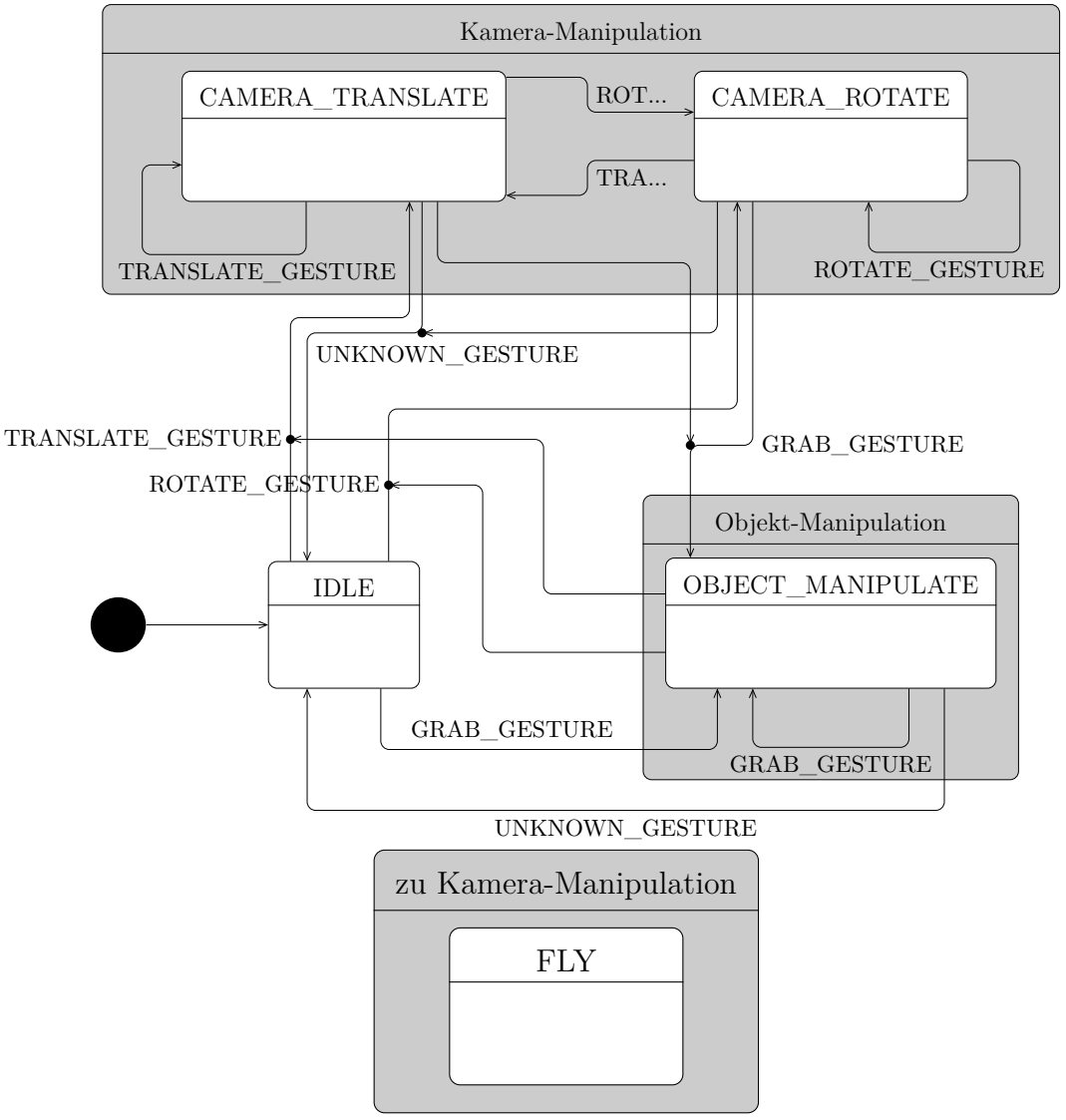

Abbildung 2: Die Zustandsmaschine. Der nachträglich eingefügte Zustand FLY ist mit allen anderen Zuständen über die entsprechende Geste verbunden und wird von allen Zuständen durch Präsentieren der FLY-Geste erreicht. Aus Gründen der Übersichtlichkeit wurde auf das Einzeichnen dieser Kanten verzichtet.

3.4 Robustheit und Pufferung

Wie wir vorangegangen festgestellt haben, sind einige der Mechanismen, die wir implementieren wollen anfällig gegenüber qualitativ niedrigwertigen Kinectdaten. Tests mit der Kinect haben folgende kritische Situtationen ergeben:

- Gelenke und Skelettbestandteile in der Nähe von Objekten und anderen Personen. Diese können falsch oder verzerrt erkannt werden. So kann etwa die erkannte Handposition zwischen zwei Kinectframes Raumunterschiede von mehreren Metern aufweisen und zurückspringen.
- Status der Hände. Auch bei durchgängiger Aufrechterhaltung eines Handzustands kann es passieren, dass die Kinect vereinzelt falsche Zuweisungen trifft oder keine Zuweisung möglich ist. Besonders schlecht wird die Erkennung, wenn sich die Hände vor dem Körper befinden.

Beide Punkte können gravierende Einschränkungen bezüglich der Programmbedienbarkeit mit sich ziehen. Beide Situationen lassen sich behandeln, indem Entscheidungen unseres Programms, nicht nur vom augenblicklichen Rückgabewert der Kinect abhängen, sondern auch einige vergangene Werte mit einbeziehen. So kann ermittelt werden, ob der aktuelle Wert (mit hoher Wahrscheinlichkeit) ein zu ignorierender Ausreißer ist. Dazu wird ein Ringpuffer verwendet und an den entsprechenden Stellen im Quellcode ein gewichtetes Mittel über den Pufferinhalt gebildet, wobei neuere Einträge mit deutlich größerem Gewicht eingehen. Für Gesten kann dann mit einer bestimmten Zuverlässigkeit eine Zuordnung getroffen werden, für Raumpositionen stellt dieses Mittel eine Glättung dar. Dies hat den positiven Nebeneffekt, dass die endgültige Anwendung der errechneten Parameter auf die Kamera bzw. das Objekt ebenfalls geschmeidiger werden.

4 Bemerkungen zum Quellcode

4.1 Wichtige Datenstrukturen, Variablen und Funktionen

4.2 Details zum Zusammenspiel

4.3 Einbinden

Literatur

[1] Microsoft Developer Network. *Tracking Users with Kinect Skeletal Tracking*. https://msdn.microsoft.com/en-us/library/jj131025.aspx. [Online; accessed 25-April-2017]. 2017.