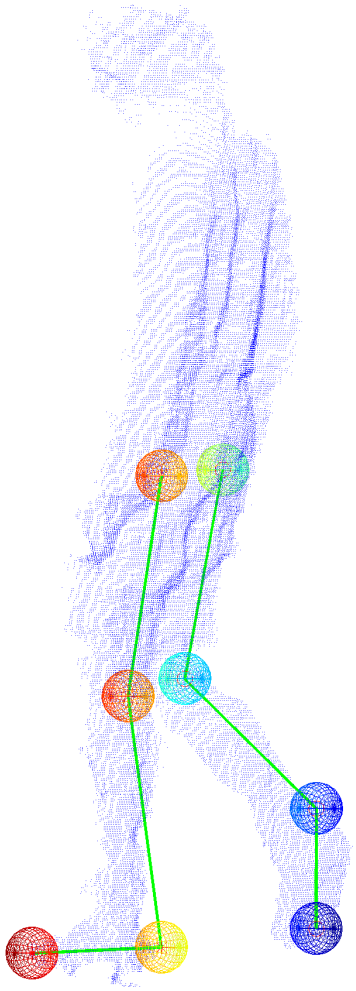
Bewegungsrekonstruktion der Beine zur Ganganalyse

anhand von Tiefenbildern eines verteilten Kinect Sensor Netzwerks



Autor Mathias Hunold
Betreuer Prof. Dr. Guido Schuster

Masterarbeit
HSR Hochschule für Technik Rapperswil
ICOM Institute for Communication Systems

2. Februar 2012

Dieses Dokument wurde mit LaTeX gesetzt.

Abstract

Ausgangssituation

Es gibt bereits viele Systeme die die Körperhaltung von Menschen mit Hilfe von Kameras analysieren. In den letzten Jahren sind TOF-Tiefenbildkameras aufgekommen. Diese haben die Aufgabe erheblich vereinfacht. Der Vorteil dieser Systeme ist, dass sie auch Informationen über die dritte Raumdimension bereitstellen.

Kinect

Der Hauptnachteil der Tiefenbildsensoren war deren hoher Preis, was sie unbrauchbar für Massenanwendungen machte. Der Kinect-Sensor von Microsoft hat dies schlagartig geändert. Er kam im Spätherbst 2010 zu einem Preis von 150 US-Dollar auf den Markt und wurde als Steuereinheit zu der Spielkonsole XBox 360 entwickelt.

Sensornetzwerk

In dieser Arbeit wurde versucht die Tiefendaten mehrerer Kinect-Sensoren zu kombinieren. Die Sicht aus verschiedenen Perspektiven hat den erheblichen Vorteil, dass es praktisch keine Abdeckungen von Körperteilen gibt. Dies vereinfacht die Aufgabe erheblich.

AGEX

In den Tiefendaten konnten mit einem eleganten Algorithmus die verschiedenen Körperteile gefunden werden. Er basiert auf der Tatsache, dass sich Distanzen auf der Körperoberfläche kaum ändern, egal aus welcher Perspektive die Person gesehen wird, oder in welcher Körperhaltung sich diese befindet. Der Algorithmus wurde an der Stanford University in den USA entwickelt [2]. Der Name AGEX, **A**ccumulative **G**eodesic **EX**trema, deutet an, dass nach den Extremalstellen auf der Körperoberfläche gesucht wird.

Resultat

Es konnte ein Algorithmus realisiert werden, der in den Tiefendaten eines Kinect-Sensors die unteren Gliedmassen eines Menschen findet und die Daten mehrerer Perspektiven kombiniert. Dies wurde mithilfe des AGEX-Algorithmus und eines Kalmanfilters, welches die Daten mehrerer Perspektiven kombiniert, erreicht. Weiter wurde ein Justierverfahren entwickelt, mit welchem die Positionen und Winkel aller Sensoren, mithilfe von Markierungen im Tiefenbild, mit guter Genauigkeit bestimmt werden können.

Inhaltsverzeichnis

1 Einleitung

Ausgangslage

Für Systeme zur automatischen Erfassung von menschlichen Bewegungen gibt es viele denkbare Anwendungsgebiete. Insbesondere in der Medizin, im Spitzensport, und in der Unterhaltungsbranche. Ein Sportler kann seine Bewegungsabläufe verbessern. In der Medizin kann zum Beispiel festgestellt werden, ob ein spezifisches Bewegungsmuster normal oder krankhaft ist. In der Unterhaltungsbranche können die erkannten Gesten zur Steuerung von Spielkonsolen verwendet werden.

Tiefensensoren

Menschliche Bewegungen können aus Kamerabildern geschätzt werden. Diese Methode hat den Vorteil, dass die Information einen absoluten Bezugspunkt, die Kamera, hat. Jedoch ist die Schätzung von dreidimensionalen Bewegungen aus einem gewöhnlichen zweidimensionalen Bild sehr schwierig. Dieses Problem stellt sich mit Tiefenbildern nicht. Bei diesen ist jedem Bildpunkt die Distanz zwischen Kameraobjektiv und beobachteter Raumposition zugeordnet.

Entwicklungen

In den letzten Jahren gab es grosse Fortschritte im Bereich von Tiefensensoren. Hier hat vor allem die TOF-Technik grosse Bekanntheit erlangt. Bei diesem Verfahren werden Lichtpulse ausgesandt und über die Lichtlaufzeit die Distanz zur Reflexionsstelle ermittelt, jedoch ist diese Technik sehr teuer. Der Preis für einen TOF-Sensor, zum Beispiel des SwissRanger 4000 von MESA Imaging, liegt im Bereich von 10'000 US-Dollar. Eine weitere Möglichkeit stellen Stereokameras dar. Dabei wird versucht, analog zum menschlichen Auge, aus zwei räumlich leicht versetzten Kamerabildern ein dreidimensionales Tiefenbild zu rekonstruieren. Preislich sind diese günstiger, haben aber das Problem, dass sie bei konturlosen, glatten Oberflächen nicht funktionieren.

Kinect

Im Spätherbst 2010 hat die Firma Microsoft den Sensor Kinect, welcher unter anderem einen Tiefensensor enthält, zu einem Preis von 150 US-Dollar auf den Markt gebracht. Bei Betrachtung der Preise und Nachteile der anderen Systeme sowie der vielfältigen Anwendungsmöglichkeiten einer günstigen Alternative, wird klar, dass die Markteinführung von Kinect einer Revolution glich.

Aufgabe

Um den menschlichen Bewegungsablauf möglichst vollständig erfassen zu können sind mehrere Sensoren, die aus verschiedenen Perspektiven das Geschehen beobachten, vorteilhaft. Der im Kinect Software Development Kit schon enthaltene Bewegungstracker verwendet nur einen Sensor und hat deshalb Mühe mit verdeckten Körperteilen umzugehen. Auch funktioniert er nur, wenn der ganze Körper im Sichtbereich des Gerätes liegt. Die Idee der vorliegenden Masterarbeit ist nun, menschliche Bewegungsabläufe mithilfe von Tiefenbildern mehrerer Kinect Sensoren zu erfassen. Dabei soll der Fokus zunächst auf die Beine gelegt werden.

2 Kinect Sensor

Einführung

Kinect wurde Ende 2010 von Microsoft auf den Markt gebracht. Die Technologie wurde von der Firma PrimeSense entwickelt und war ursprünglich unter dem Namen Project Natal bekannt[6]. Kinect ist als Erweiterung zu der Microsoft Spielkonsole XBox 360 gedacht und dient als neuartige Benutzerschnittstelle, mit welcher die Spielkonsole intuitiv durch Bewegungen gesteuert werden kann.

Abbildung 2.1: Kinect Sensor

Enthaltene Sensoren

Im Gerät ist ein Tiefenbildsensor, eine RGB-Kamera und ein Mikrofonarray enthalten. Weiter ist sie über über einen internen Motor von minus 28 Grad bis plus 28 Grad schwenkbar. Für die Steuerung der XBox wurde von der Firma Microsoft ein Echtzeit Skeleton-Tracker entwickelt [1]. Die Daten des Skeleton-Trackers stehen zur freien Verfügung, wurden jedoch nicht verwendet. Für diese Arbeit wurden hauptsächlich die Daten des Tiefenbildsensors und sporadisch die RGB-Daten benutzt. Auf das Mikrofonarray wird nicht eingegangen.

2.1 Grundprinzip Kinect Tiefenbildsensor

Funktionsprinzip

Der Kinect Tiefenbildsensor beruht auf dem Prinzip eines Structured-Light 3D Scanners [12]. Es wird ein Infrarotmuster auf die Umgebung projiziert. Das von Kinect projizierte Pseudozufallsmuster ist in Abbildung 2.2 zu sehen. Es wird von einer Infrarotkamera aufgenommen. Anhand des bekannten Musters und der Aufnahme der Infrarotkamera kann das dreidimensionale Tiefenbild rekonstruiert werden. Auf den Algorithmus wird hier nicht näher eingegangen. Das zugrunde liegende Prinzip,

wie es in [12] beschrieben ist, ist leicht verständlich. Infrarotmuster, welche auf einem weit entfernten Objekt abgebildet werden, erscheinen auf dem Infrarotsensor der Kamera kleiner als sensornahe Muster. Es ist einfach zu verstehen, dass anhand dieser Information die Distanz zum Objekt berechenbar ist.

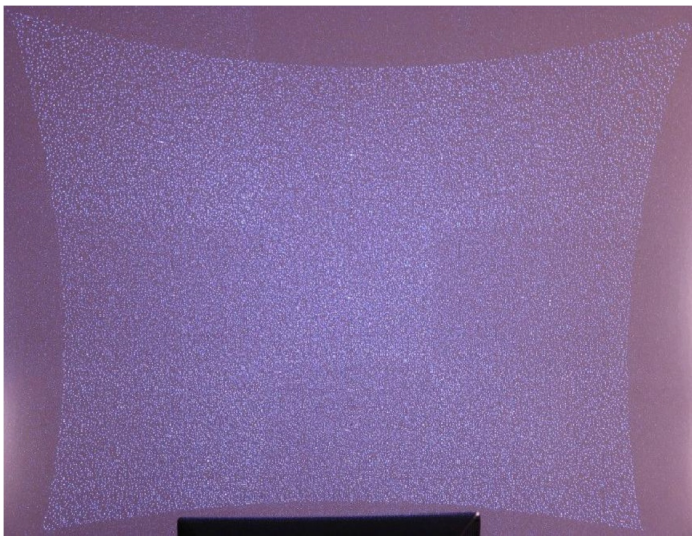


Abbildung 2.2: Von Kinect projiziertes Infrarotmuster [10]

2.1.1 Sensorkoordinatensystem

Distanz

Der Tiefenbildsensor stellt Distanzwerte in einem zweidimensionalen Array angeordnet zur Verfügung. Die errechneten Distanzen werden senkrecht zu der Sensorebene und nicht, wie angenommen werden könnte, radial zum Sensor angegeben. In der Abbildung 2.3 ist das Messprinzip des Tiefensensors ersichtlich.

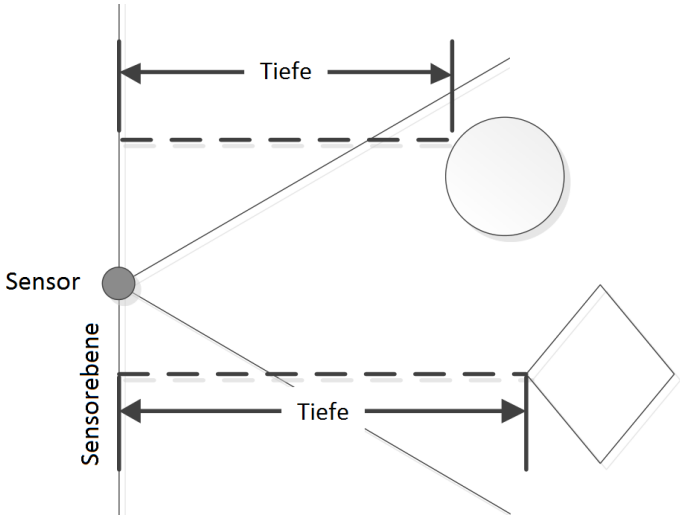


Abbildung 2.3: Messprinzip des Tiefensensors von Kinect [7]

Die Sensorebene ist durch die Blickrichtung des Sensors definiert. Diese entspricht der Normalen der Ebene. Jede Distanzangabe ist als die kürzeste Distanz vom registrierten Objekt zu dieser Ebene definiert. Die Masseinheit beim Microsoft SDK ist Millimeter.

Indizes

Als weitere Koordinaten können die Indizes betrachtet werden. Diese werden benötigt um die beiden noch fehlenden Raumrichtungen eines kartesischen Koordinatensystems berechnen zu können. Die Indizes entsprechen jedoch laut [7] nicht physikalischen Raumkoordinaten. Deren Interpretation ist von der Optik und dem bildgebenden Sensor abhängig. Für die Umrechnung sind Annahmen zum Sensor notwendig. Siehe dazu den Versuch, welcher in Kapitel 2.3 beschrieben ist.

2.1.2 Sensorfehler

Quantisierung

Die gemessene Distanz wird mit 12 Bit Auflösung bereitgestellt. Tatsächlich werden aber nur 345 Werte unterschieden. Siehe dazu Kapitel 2.4. Diese Quantisierungsfehler hat bei der Implementierung zu einigen Schwierigkeiten geführt.

Auflösung

Die maximale Auflösung des Sensorbildes ist auf 640 Pixel waagrecht und 480 Pixel senkrecht begrenzt. Auch dies führt zu Fehlern, welche vor allem bei relativ grossen Distanzen eine immer erheblichere Rolle spielen. Siehe dazu den Versuch in 2.3.

Wellen/Linien

Sehr deutlich sind zwei Fehlermöglichkeiten in der Abbildung 2.5 b sichtbar. Dies sind senkrechte Linien und kreisförmige Wellen. Die Fehler kommen zum Vorschein weil eine sehr starke Auflösung gewählt wurde. Der sichtbare Distanzbereich reicht von 985 mm bis 1025 mm. Dies sind 40 mm. Die Effekte, bei dieser Distanz, sind folglich deutlich kleiner. Die Kreise schwanken zwischen 995 mm und 1007 mm. Die Sprünge sind zwei bis drei Millimeter hoch, was genau einem Quantisierungsschritt entspricht.

Sonnenlicht

Der Tiefensensor ist sehr anfällig auf Sonnenlicht. Offenbar ist die Infrarotstrahlung der Sonne stark genug, damit sie den Sensor stören kann. Kinect wurde nicht für outdoor Anwendungen konzipiert.

2.2 Verwendung Kinect Sensoren

Treiber

Die Verwendung der Kinect Sensoren mit dem Computer und dem original Treiber ist recht einfach, es sind nur wenige Details zu beachten. Der original Microsoft Kinect Treiber funktioniert nur unter Microsoft Windows 7. Er ist gratis im Netz verfügbar [5] und steht für jegliche nicht kommerzielle Zwecke zur freien Verfügung.

USB-Controller

Jedem Kinect Sensor muss ein eigener USB-Controller zur Verfügung stehen. Ein

Sensor reserviert standardmässig 60% der Controllerkapazität. Falls mehrere Kameras an einem Computer verwendet werden sollen, dann müssen zwangsläufig gleichviele oder mehr Controller vorhanden sein.

Anschluss

Nach dem Anschliessen des Kinect-Sensors sollte dessen grüne LED zu blinken beginnen. Wenn die Treiber von Microsoft richtig installiert sind, wird jeder angeschlossene Sensor automatisch erkannt. Im Device Manager ist jeder Sensor als 'Microsoft Kinect' ersichtlich.

Microsoft SDK

Der Zugriff auf die Daten kann im Microsoft SDK aus einem c++ oder einem c# Programm geschehen. Für diese Arbeit wurde mit c# gearbeitet. Als Softwarewerkzeug kann Visual C# express, welches gratis erhältlich ist, verwendet werden. Eine einfache Anleitung, wie ein eigenes Projekt gemacht werden kann und ein kurzes Beispielprogramm, wie der Datenzugriff geschieht, ist im Anhang C und D zu finden.

Probleme

Bei der Verwendung von einem oder zwei Sensoren ergaben sich keine weiteren Schwierigkeiten. Die Labor PC's sind standardmässig mit zwei USB2 Controllern sowie einen USB3 Controller ausgerüstet. Bei drei Kinect jedoch schlug die Initialisierung fehl, obwohl der PC alle drei Einheiten erkannt hatte. Einen zusätzlichen USB2-Controller zu verwenden war nicht die Lösung. Dieser wurde als Karte über PCI angeschlossen. Die USB2 zu PCI Karte war zu langsam. Zudem scheiterte die Initialisierung weiterhin. Um nicht noch mehr Zeit für dieses Problem zu verlieren, wurde es umgangen. Statt alle Sensoren an einem PC anzuschliessen, wurden diese auf zwei Computer verteilt. Dies gab etwas Mehraufwand, da die Synchronisation der Computeruhren dazu kam. Ein einzelner Computer kann jedoch nicht beliebig viel Sensordaten verarbeiten. Zu einem späteren Zeitpunkt im Projektverlauf wären ohnehin mehrere PC notwendig geworden. Es zeigte sich, dass bereits mit zwei Sensoren viel mehr Frames verloren gehen als mit einer. Drei Kinect gleichzeitig an einem PC anzuschliessen wäre also grenzwertig gewesen.

2.3 Versuch zur Berechnung der Kartesischen X- und Y-Koordinaten

Einführung

Die Sensoren erzeugen ein Tiefenbild. Das heisst ein Array von Distanzwerten, welchen X- und Y-Indizes zugeordnet werden können. Die Tiefenwerte sind auf eine Ebene bezogen (Siehe Kapitel 2.1.1). Sie entsprechen folglich schon einer Koordinate eines kartesischen Systems und müssen nicht transformiert werden. Im folgenden werden die Tiefenwerte als Z-Koordinaten interpretiert. Die Indizes hingegen entsprechen nicht zwingend physikalischen Raumkoordinaten. Deren Interpretation ist von der Optik und dem bildgebenden Sensor abhängig [7]. Es müssen folglich einige Annahmen bezüglich Aufbau des Sensors gemacht werden, damit die Indizes verwendet werden können. Der Versuch hat gezeigt, dass eine Umrechnung in physikalische Koordinaten möglich und damit auch die Grundidee dieser Arbeit, mehrere Sensoren zu kombinieren, prinzipiell umsetzbar ist.

Projektion

Das Umrechnungsprinzip ist sehr einfach. Es wird davon ausgegangen, dass jeder Bildpunkt unverzerrt auf dem Infrarotsensor abgebildet wird. Das Prinzip wurde in der Abbildung 2.4 veranschaulicht.

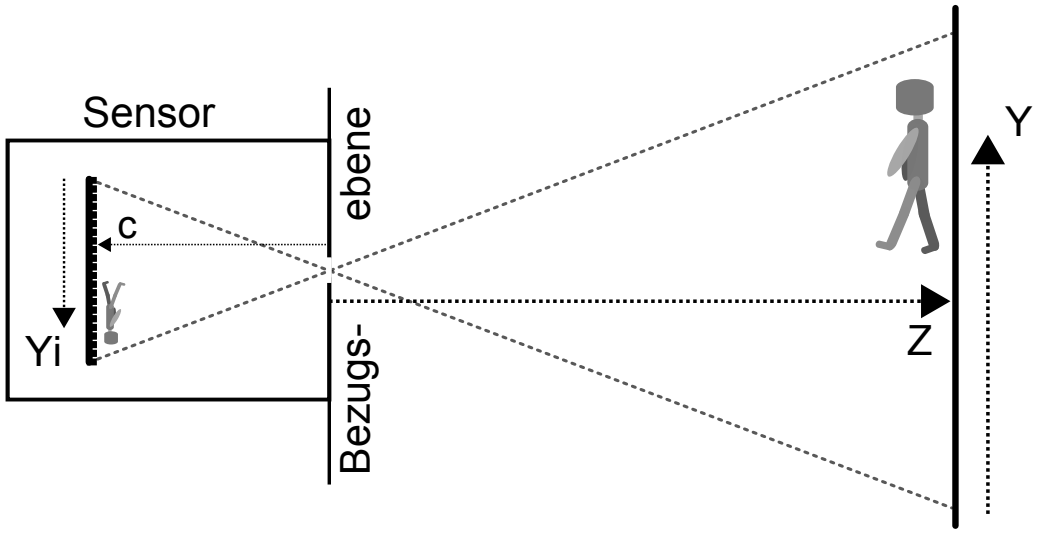


Abbildung 2.4: Projektion

Die Distanz zur Sensorebene spielt für die Umrechnung eine entscheidende Rolle. In der Abbildung 2.4 wurde diese mit Z bezeichnet. Je weiter ein Gegenstand vom Sensor entfernt ist, desto kleiner wird er auf dem Sensor abgebildet. Ist der Gegenstand doppelt so weit entfernt, erscheint er halb so gross auf dem Chip. Die Grösse auf dem Sensorchip ist also umgekehrt proportional zur Z Koordinate. Dass die X Indizes x_i und die Y Indizes y_i mit Z multipliziert werden müssen ist also offensichtlich. Aufgrund der Annahme, dass die Umgebung unverzerrt auf dem Chip abgebildet wird, muss der folgende Zusammenhang gelten:

$$\frac{c}{u \cdot y_i} = \frac{Z}{Y} \tag{2.1}$$

Aus Abbildung 2.4 wird klar, dass es sich um eine zentrischen Streckung handelt. Die Konstante u steht für die Umrechnung von der Masseinheit Pixel zur Masseinheit Millimeter.

Aus Gleichung 2.1 folgt

$$Y = \frac{u \cdot y_i \cdot Z}{c} = y_i \cdot Z \cdot a \tag{2.2}$$

analog gilt natürlich auch

$$X = x_i \cdot Z \cdot a \tag{2.3}$$

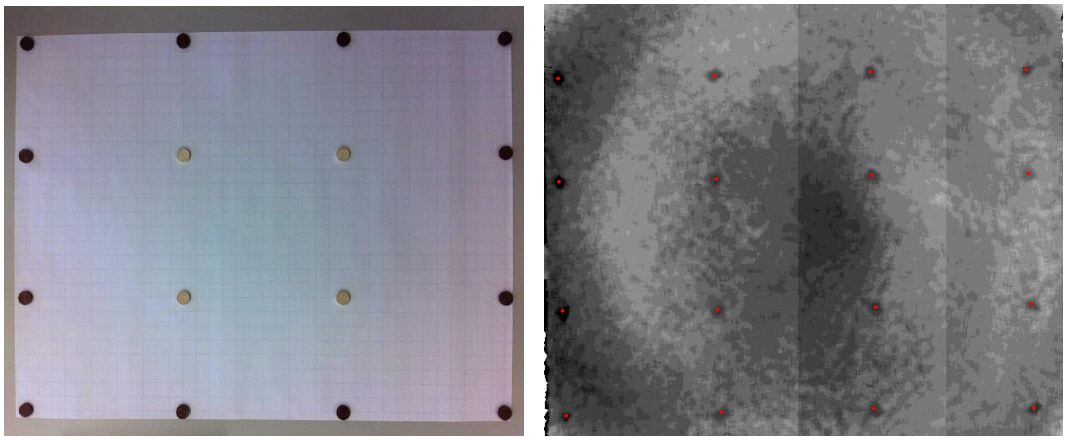
Die Unbekannte a wurde in folgendem Versuch ermittelt. Mit a und der Z-Komponente können die kartesischen Raumkoordinaten jedes Punktes im Tiefenbild berechnet werden.

Versuchszweck

Dieser Versuch überprüft, ob die Annahme, dass die Aufnahmen auch in den x_i und y_i unverzerrt sind, richtig ist. Stimmt diese Modellannahme mit genügender Genauigkeit, können die korrekten kartesischen Koordinaten berechnet werden. Einem Zusammenführen der Daten mehrerer Sensoren steht prinzipiell nichts mehr im Weg. Weiter kann anhand der Ergebnisse die Konstante a des Sensors mit guter Genauigkeit bestimmt werden.

Aufbau

Es wurde eine weisse Wand mit regelmässig verteilten runden Erhöhungen (zirka 5 mm hoch) vor einem Kinect Sensor platziert (Abbildung 2.5). Die Distanz zum Sensor betrug genau einen Meter. Die horizontalen Abstände zwischen den Erhöhungen sind 300 mm. Die Vertikalen sind 200 mm, 250 mm und wieder 200 mm. Die hellsten Stellen im Bild 2.5 b sind 1025 mm entfernt, die dunkelsten 985 mm.



(a) Farbbild (b) Tiefensensorbild

Abbildung 2.5: Versuch zur Bestimmung des Streckungsfaktors a

Vorgehen

Die Konstante a kann mit Hilfe der Formel 2.3 aus den Daten geschätzt werden. Da sich a in X- und Y-Richtung unterscheiden könnte, wurde zuerst $a_x = 1.7232 \frac{1}{px}$ und $a_y = 1.7306 \frac{1}{px}$ berechnet. Weiter wurden die Standardabweichungen in beide Richtungen ermittelt. Sie beträgt für die X-Richtung $std(a_x) = 0.0185 \frac{1}{px}$ und für die Y-Richtung $std(a_y) = 0.0195 \frac{1}{px}$. Die Differenz zwischen a_x und a_y beträgt weniger als ein Prozent. Vergleicht man diese mit den Standardabweichungen, wird klar, dass die Nullhypothese "a_x und a_y sind verschieden" verworfen werden muss. Bei Betrachtung der Histogramme ist ersichtlich, dass die implizite Annahme einer Normalverteilung, nicht abwegig ist. Weiter ist offensichtlich, dass sich die Messwerte beider Richtungen stark überlappen, was die These, a_x und a_y sind gleich, stützt. Die Umrechnung kann folglich mit einer einzigen Konstante a vorgenommen werden. Der Versuch wurde, mit dem Matlab Skript 'showResultsFinda.m' im Verzeichnis 'C_matlab', durchgeführt.

Resultat

Bei Betrachtung von Abbildung 2.5 b, ist offensichtlich, dass keine Verzerrungen vorliegen. Die rot markierten Punkte sind genau dort sichtbar, wo sie platziert wurden. Das schachbrettartige Muster ist unverzerrt. Die Konstante a beträgt 1.7267 $\frac{1}{px}$. Die Standardabweichung der Messungen ist 0.019 $\frac{1}{px}$. Der mittlere Messfehler ist somit ein Prozent. Es liegen 24 Distanzmessungen vor. Die maximale Abweichung vom Mittelwert beträgt 0.0427 $\frac{1}{px}$ was unterstreicht, dass keine Verzerrungen vorliegen.

Schwierigkeiten

Es war nicht von Beginn weg klar wie die X- und Y-Koordinaten berechnet werden können. Der Kinect SDK bietet dazu leider keine Hilfestellung. Am Anfang wurde vermutet, es könnte sich beim X- und beim Y-Index um Winkel handeln. Diese Idee entstand durch die Tatsache, dass der Sensor von einem Punkt aus misst, der Blende des Infrarotsensors. Dabei wurde aber die Tatsache ausser Acht gelassen, dass es sich beim Infrarotsensor um eine Ebene handelt. Das Heisst eine Fläche wird unverzerrt wieder auf eine Fläche abgebildet. Damit die Winkelannahme stimmen würde müsste der Sensor eine Kugelform, ähnlich dem menschlichen Auge, haben.

2.4 Versuch Quantisierungsschritte des Tiefensensors

Präzision

Dass die Präzision der Messwerte genügend gut ist, wird schon aus dem Versuch in Kapitel 2.3 ersichtlich. Es wurde eine Wand mit einer Distanz von genau einem Meter vor der Kamera platziert. Die Messwerte lagen innerhalb eines Bereiches von zirka einem Zentimeter. Siehe dazu auch das Unterkapitel 2.1.2 Sensorfehler. Auch in [10] wird die Genauigkeit untersucht. In diesem Versuch wurde nur die Quantisierung der Messwerte analysiert.

Quantisierung

Bei genauerer Betrachtung der Quantisierung der Distanzwerte des Sensors ist auffällig, dass diese bei grossen Distanzen sehr grob ist. Diese Quantisierung steht logischerweise in direktem Zusammenhang mit der Sensorgenauigkeit.

Vorgehen

In der Abbildung 2.6 sind die Quantisierungsschritte gegenüber der Distanz aufgetragen. Auf der Abszissenachse die Distanz zum Sensor und auf der Ordinatenachse der Quantisierungsschritt zur nächsten Quantisierungsstufe. Die Quantisierung wurde mit dem File 'findDepthResolution' (CD: Unterverzeichnis C_matlab) ermittelt. Es liest ein beliebiges Tiefenbildvideo ein und ermittelt alle darin vorkommenden Abstufungen. Dazu wird jeder Sensorwert des gesamten Tiefenbildvideos untersucht. Neue, vorher noch nicht festgestellte Distanzwerte werden gespeichert. Zuletzt werden alle ermittelten Sensorwerte den eruierten Quantisierungsschritten in einer Grafik gegenübergestellt. Die Auswertung einiger Videos hat ergeben, dass maximal 345 verschiedene Abstufungen existieren. Der kleinste Wert ist 801 mm und die grösste mögliche Sensordistanz ist 3930 mm.

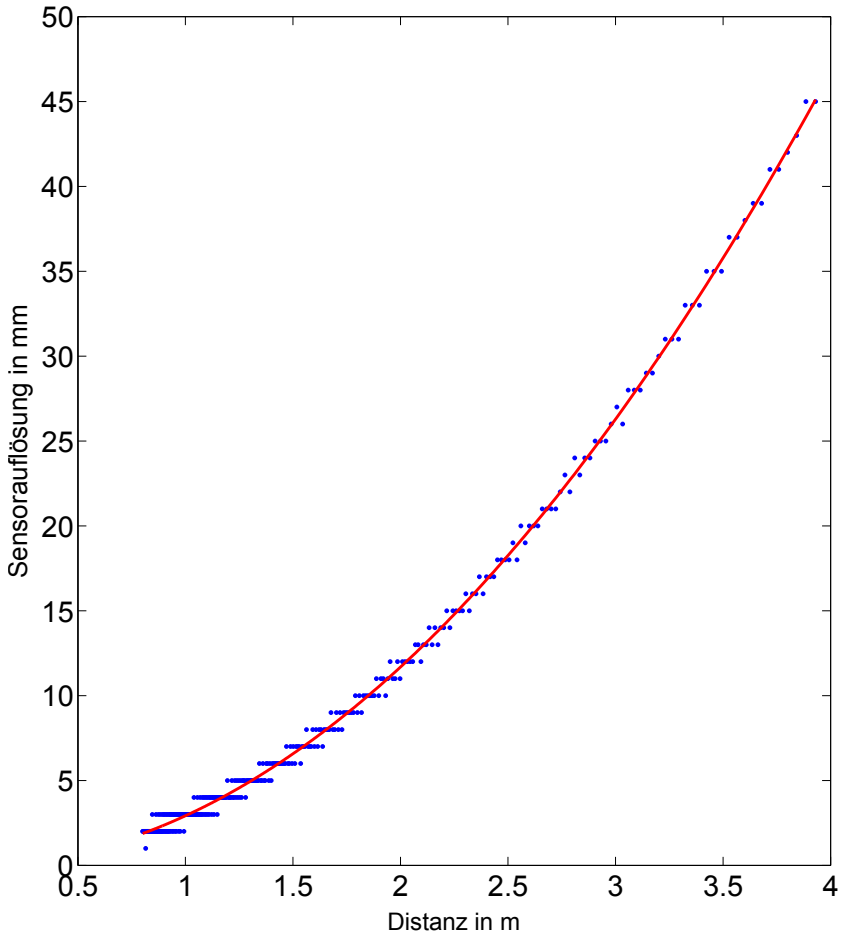


Abbildung 2.6: Quantisierungsschritte des Kinect-Tiefensensors

Approximation

Die Genauigkeit des Sensors nimmt, laut Daten, scheinbar quadratisch mit der Distanz ab. Die Auflösung kann mit der sehr einfachen Formel

$$y = 2.92 \cdot x^2 \ \frac{mm}{m^2} \tag{2.4}$$

abgeschätzt werden. Die quadratische Approximation ist rot in der Abbildung 2.6 ersichtlich. Dieser Zusammenhang ist relativ einfach durch das Funktionsprinzip des Sensors erklärbar. Das projizierte Infrarotmuster erscheint bei doppelter Distanz halb so gross. Dies bedeutet die abgebildete Fläche auf dem Bildgebenden Infrarot Sensor halbiert sich. Dem Algorithmus des Sensors stehen folglich vier mal weniger Pixel zur Verfügung. Die logische Konsequenz ist eine viermal schlechtere Distanzauflösung.

3 Berechnung der Sensorpositionen und Sensorwinkel

Allgemein

Die Lage der Kinect Sensoren, dass heisst deren Position und gegenseitige Ausrichtung, kann aus den Tiefenbilddaten geschätzt werden. Dazu wurden kugelförmige Marker, in einem für alle Sensoren erfassbaren Gebiet, positioniert.

Problem

Es müssen die Positionen und Winkel aller Sensoren, anhand der im Versuchsraum verteilten Marker, bestimmt werden. Die Markerposition aus Sicht der „Ausgangs-Kinect“ muss so gedreht und verschoben werden, dass sie der Position des gleichen Markers aus der Perspektive einer anderen Kamera („Ziel-Kinect“) entspricht.

$$\vec{x}_t = R \cdot \vec{x}_m + \vec{d} \tag{3.1}$$

$\vec{x}_t$ ist die Zielposition, $\vec{x}_m$ die Ausgangsposition. R ist eine Rotationsmatrix mit neun unbekannten Parametern. $\vec{d}$ ist ein Vektor mit drei Unbekannten. Zusammen sind das zwölf Unbekannte. Es braucht also vier Markierungen (es kommen pro Markierung drei Gleichungen dazu) um dieses lineare Gleichungssystem zu lösen.

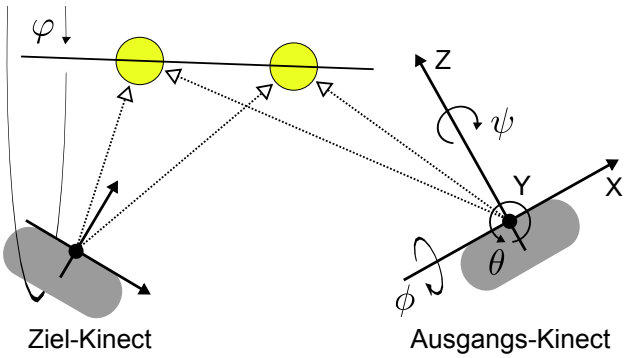


Abbildung 3.1: Sensor-Koordinatensysteme und Transformationswinkel

Anzahl Markierungen

Die Transformation ist mit sechs Parametern beschreibbar.

- Drei Verschiebungen: ΔX, ΔY und ΔZ
- Drei Drehungen: ϕ, θ und ψ (Siehe Abbildung 3.1).

Daraus könnte man schliessen, dass nur zwei Marker notwendig sind, um alle Parameter zu finden. Dem ist leider nicht so. Der Grund ist anhand Abbildung 3.1 intuitiv verständlich. Die gestrichelten Pfeile zeigen die Distanzvektoren von den Sensoren

zu den Markern. Diese Vektoren sind im Gleichungssystem bekannt. φ zeigt an aus welchem Winkel die Sensoren auf die Linie, welche die beiden Markierungen verbindet, blicken. Der Winkel φ ist aber offensichtlich nicht bestimmt. Bekanntlich lässt sich mit zwei Punkten eine Gerade beschreiben. Um alle Verdrehungen eindeutig zu bestimmen, ist jedoch eine Ebene notwendig. Es braucht also drei Markierungen.

Parametrisierung

Die Rotationsmatrix aus Gleichung 3.1 hat neun Parameter. Das sind neun Freiheitsgrade, eine Rotation ist jedoch mit drei Parametern realisierbar. Um die Anzahl unbekannter Variablen des Gleichungssystems zu reduzieren macht es Sinn eine bessere Parametrisierung für R zu finden. Es wurden zwei Möglichkeiten geprüft.

- Eulerwinkel — Die Rotation wird in Drehungen um die drei Koordinatenachsen des kartesischen Koordinatensystems zerlegt (siehe Unterkapitel 3.2).
- Quaternionen — Quaternionen sind ein vierdimensionales Zahlensystem. Eine Drehung mit diesen kann als Winkel um eine Drehachse verstanden werden (siehe Unterkapitel 3.3).

Nichtlinearität

Wenn man versucht die Anzahl Parameter von R zu reduzieren, wird das Gleichungssystem nichtlinear. Multiplikationen bei den Quaternionen und trigonometrische Funktionen bei den Eulerwinkeln lassen sich nicht vermeiden. Es ist schwierig diese Gleichungssysteme zu lösen.

Alternative

Die Lagebestimmung wäre auch direkt anhand der Versuchsdaten machbar. Zum Beispiel könnten die beiden Zehenspitzen und der Kopf einer Person gefunden werden. Diese drei Punkte sind relativ einfach zu detektieren und würden ausreichen um alle Positionen und Drehwinkel zu bestimmen. Es bliebe aber eine Unsicherheit darüber, ob sich die Punkte wirklich entsprechen. Je weiter auseinander die Markierungen positioniert werden, desto genauer wird die Lagebestimmung. Bei diesem alternativen Verfahren sind grosse Distanzen aber selten gegeben, weshalb es oft ungenauer ist.

3.1 Gauss-Newton-Algorithmus zum lösen des nichtlinearen Gleichungssystems

Grundidee

Zur Lösung des nichtlinearen Gleichungssystems wurde der Gauss-Newton-Algorithmus verwendet. Dessen Grundidee ist es mit den Residuen und einer Linearisierung des Gleichungssystems die gesuchten Parameter iterativ zu finden. Der Algorithmus ist hier anhand der Quaternionen-Parametrisierung beschrieben, das Prinzip ist jedoch unabhängig von der Parametrisierung anwendbar.

Parametervektor

Die Parameter werden in einem Vektor zusammengefasst. Bei q_1 bis q_4 handelt es sich um die vier Parameter des Quaternions. Δx, Δy und Δz sind die Verschiebungen in Richtung der jeweiligen Achsen.

$$\vec{p} = \begin{pmatrix} \Delta x & \Delta y & \Delta z & q_1 & q_2 & q_3 & q_4 \end{pmatrix}^T \tag{3.2}$$

Funktion

Der Vektor $\vec{F}(\vec{p})$ ergibt sich aus der Drehung und der Verschiebung der n Markerpositionen $\vec{x}_{m_1}$ bis $\vec{x}_{m_n}$. Die Bezeichnung der Drehmatix als $R_q = R(q_1, q_2, q_3, q_4)$ symbolisiert, dass diese mit Quaternionen parametrisiert wurde. Die Verschiebungen wurden im Vektor $\vec{d} = \begin{pmatrix} \Delta x & \Delta y & \Delta z \end{pmatrix}^T$ zusammengefasst.

$$\vec{F}(\vec{p}) = \begin{pmatrix} \vec{f}_1(\vec{p}) \\ \vec{f}_2(\vec{p}) \\ \vdots \\ \vec{f}_n(\vec{p}) \end{pmatrix} = \begin{pmatrix} \vec{f}(\vec{x}_{m_1}, \vec{p}) \\ \vec{f}(\vec{x}_{m_2}, \vec{p}) \\ \vdots \\ \vec{f}(\vec{x}_{m_n}, \vec{p}) \end{pmatrix} = \begin{pmatrix} R_q \cdot \vec{x}_{m_1} + \vec{d} \\ R_q \cdot \vec{x}_{m_2} + \vec{d} \\ \vdots \\ R_q \cdot \vec{x}_{m_n} + \vec{d} \end{pmatrix} \tag{3.3}$$

Zielvektor

Die n Positionsmessungen aus der Perspektive des Zielsensors werden im Vektor $\vec{T}$ zusammengefasst.

$$\vec{T} = \begin{pmatrix} \vec{x}_{t_1} \\ \vec{x}_{t_2} \\ \vdots \\ \vec{x}_{t_n} \end{pmatrix} \tag{3.4}$$

Gleichungssystem

Es ist das Ziel die Punkte $\vec{x}_{m_1}$ bis $\vec{x}_{m_n}$ in die Punkte $\vec{x}_{t_1}$ bis $\vec{x}_{t_n}$ zu transformieren. Es muss also $\vec{F}(\vec{p})$ mit $\vec{T}$ gleichgesetzt werden.

$$\vec{F}(\vec{p}) = \vec{T}. \tag{3.5}$$

Approximation

Man möchte nun das Gleichungssystem 3.5 nach $\vec{p} = \vec{F}^{-1}(\vec{T})$ auflösen. Das geht aber nicht direkt, da es nichtlinear ist. Es muss deshalb iterativ gelöst werden. Es wird mit den Parametern $\vec{p}_0$ gestartet. Die lineare Approximation von $F(\vec{p})$ durch den Punkt $\vec{F}(\vec{p}_0)$ ist

$$\vec{F}(\vec{p}) \approx A \cdot \Delta\vec{p} + \vec{F}(\vec{p}_0) \tag{3.6}$$

was äquivalent ist zu

$$\vec{T} \approx A \cdot (\vec{p} - \vec{p}_0) + \vec{F}(\vec{p}_0) \tag{3.7}$$

Jacobi-Matrix

Die Jacobi-Matrix A wird aus den Ableitungen nach den Parametern $\vec{p}$ erstellt.

$$A = \begin{pmatrix} \frac{\partial f_1}{\partial \Delta x} & \frac{\partial f_1}{\partial \Delta y} & \cdots & \frac{\partial f_1}{\partial q_4} \\ \frac{\partial f_2}{\partial \Delta x} & \frac{\partial f_2}{\partial \Delta y} & \cdots & \frac{\partial f_2}{\partial q_4} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial \Delta x} & \frac{\partial f_n}{\partial \Delta y} & \cdots & \frac{\partial f_n}{\partial q_4} \end{pmatrix} \tag{3.8}$$

Pseudoinverse

Die Approximation in 3.7 kann mit der Pseudoinversen $A^+ = (A^T A)^{-1} A^T$ nach dem Parametervektor $\vec{p}$ aufgelöst werden.

$$\vec{p} \approx A^+ \cdot (\vec{T} - \vec{F}(\vec{p_0})) + \vec{p_0} = \vec{p_1} \tag{3.9}$$

Iteration

Wenn die Funktion $\vec{F}(\vec{p})$ einen kontinuierlichen Verlauf hat, sollte $\vec{p_1}$ eine bessere Näherung an den Parametervektor $\vec{p}$ sein als $\vec{p_0}$. Für den nächsten Iterationsschritt wird $\vec{p_1}$ anstelle von $\vec{p_0}$ verwendet. Der Algorithmus wird beendet, wenn der Residuenvektor $\vec{r} = \vec{T} - \vec{F}(\vec{p_k})$ sich nur noch minimal ändert.

Implementation

Der Algorithmus wurde versuchsweise mit Eulerwinkeln und Quaternionen realisiert. Zur Anwendung mit anderen Algorithmen wurde die Version mit Quaternionen verwendet. Im Gegensatz zu Euler-Winkeln, kann es nur zwei Quaternionen geben, welche die gleiche Drehung darstellen. Ein weiterer Vorteil der Quaternionen ist, dass sie kein Gimbal Lock kennen. Die Realisierung ist in der Matlab Funktion 'findQuatAndTrans' implementiert (Unterverzeichnis 'turn'). Die symbolische Berechnung der A Matrix wurde mit Matlab durchgeführt (File 'QuaternionCalculations.m' im Unterverzeichnis 'turn').

3.1.1 Versuch zur Konvergenz des Algorithmus

Vorgehen

Der Algorithmus wurde mit Zufallswerten verifiziert. Die Zahlen wurden folgendermassen generiert.

1. Zufällige Markerpositionen $\vec{x}_{t_i}$ für den Zielsensor generieren: Die Zufallswerte für die X- und Y-Koordinaten liegen zwischen -2000 und 2000. Die Z-Komponente liegt zwischen null und 4000.

2. Zufällige Parameter für die Translation und die Drehung generieren: Die Zufallswerte für die Verschiebungen Δx, Δy und Δz liegen zwischen -2000 und 2000. Die Quaternionen wurden aus φ und $\vec{n}$ berechnet. Für φ wurden Zufallswerte zwischen null und zwei π generiert. Für $\vec{n}$ wurden Zufallszahlen zwischen minus eins und eins generiert, dann wurde $\vec{n}$ auf eins normiert.

3. Markerpositionen $\vec{x}_{m_i}$ aus $\vec{x}_{t_i}$ und den Parametern berechnen: Zu den berechneten wahren Positionen normalverteilte Zufallswerte addieren, um Messfehler zu simulieren. Die Standardabweichung der Zufallswerte war 50. Dies wäre also vergleichbar mit einem relativ grossen mittleren Messfehler von 50 mm.

Wenn nichts weiter erwähnt wurde, sind die Zufallswerte gleichverteilt.

Der Versuch wurde mit drei und mit sechs simulierten Markern, jeweils zehntausend mal, durchgeführt. Der Algorithmus wurde jeweils gestoppt, wenn sich die Norm des Residuenvektors $\vec{r}$ um weniger als 0.01 änderte. Es wurden die Anzahl benötigter Iterationen und die Norm des verbleibenden Residuen Vektors verglichen.

Auswertung

Beim Versuch mit sechs Markern konvergierte der Algorithmus immer innerhalb von vierzehn Zyklen. Er benötigte minimal vier Zyklen zum konvergieren. Weniger als 1% der Tests benötigten mehr als zehn Iterationen. Die Norm von $\vec{r}$ war im Mittel 160. Dieser Wert ist im wesentlichen vom Messfehler und den Anzahl Markern abhängig. Mehr Marker führen zu einem längeren residuen Vektor, was folglich auch zu einer grösseren Norm führt.

Auch mit drei Markierungen benötigte der Algorithmus nur in drei Prozent der Fälle mehr als zehn Iterationen. Er benötigt im schlechtesten Fall jedoch 69 Iterationsschritte. Die mittlere Norm des Residuenvektors ist 65. Die Standardabweichung beträgt 119. In genau sieben Fällen konnte der Algorithmus die Marker nur sehr schlecht zusammenführen. Der Residuenvektor hat in diesen Fällen eine Norm von mehr als 700. Für diese sieben Ausreisser wurden mehr als zwanzig Iterationen benötigt. Interessanterweise korrespondieren diese nicht mit den grössten Ausreissern bei den Anzahl Iterationen. Es gibt folglich Fälle, in welchen der Algorithmus nach langer Suche trotzdem noch geeignete Parameter findet.

Wieso es zu einzelnen schlechten Resultaten kommt, wurde weiter analysiert. Es hat sich gezeigt, dass sich die Markierungen in diesen Fällen ungefähr auf einer Gerade befanden. Mit drei Punkten auf einer Geraden kann keine Ebene beschrieben werden. Wie schon erwähnt ist eine Ebene notwendig, um alle Verdrehungen eindeutig zu bestimmen. Einer der drei Punkte enthält redundante Information.

Als Ergebnis kann gesagt werden, dass der Algorithmus mit zusätzlichen Markern zuverlässiger wird. Es hat sich zudem gezeigt, dass der Gauss-Newton-Algorithmus bei diesem Problem sehr zuverlässig funktioniert. Der Grund für die Verlässlichkeit liegt wohl in der Natur der Drehfunktion. Wenn die Drehmatrix mit Quaternionen parametrisiert wird, existieren nur Konstanten und Produkte von maximal zwei Variablen. Die Ableitungen sind lineare Funktionen. Die Funktion hat mit Sicherheit keine Sprungstellen und kann nur ein einziges, globales Minimum haben.

3.2 Eulerwinkel

Grundidee

Drehungen im dreidimensionalen Raum werden oft mit Eulerwinkel beschrieben. Die Drehung wird in drei Teile zerlegt. Eine Drehung um die X-, die Y- und die Z-Achse. In der Abbildung 3.2 sind diese drei Drehungen mit ϕ, θ und ψ bezeichnet.

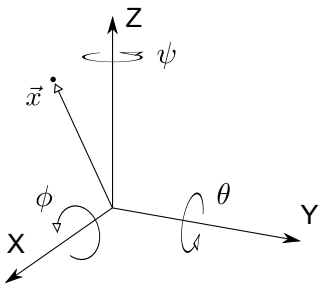


Abbildung 3.2: Eulerwinkel

Drehmatrix

Die drei folgenden Matrizen stellen die Parametrisierungen mit Eulerwinkeln jeweils um eine der drei Hauptachsen dar.

$$R_z(\phi) = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

$$R_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{3.11}$$

$$R_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{3.12}$$

Zum erreichen einer beliebigen Drehung können alle drei Matrizen miteinander multipliziert werden. Dabei ist die Reihenfolge zu beachten.

$$R(\phi, \theta, \psi) = R_x(\phi) \cdot R_y(\theta) \cdot R_z(\psi) \tag{3.13}$$

Probleme

Wenn die drei Drehungen in einer anderen Reihenfolge ausgeführt werden, führt dies zu einem anderen Endresultat. Neben dieser intuitiv nicht leicht verständlichen Schwierigkeit, kann es mehrere Eulerwinkelkombinationen geben, die zum gleichen Resultat führen. Eine weitere Schwierigkeit kann das sogenannte Gimbal Lock sein. Eine ausführliche Beschreibung dieses Phänomens ist in [4] zu finden.

3.3 Quaternionen

allgemein

Im folgenden wird kurz beschrieben, was unter Quaternionen und deren Interpretation als Drehoperatoren zu verstehen ist. Da Herleitungen den Rahmen dieser Arbeit sprengen würden, wird auf diese verzichtet. Der Inhalt dieses Kapitels wurde [4] entnommen, wo dieses Thema ausführlich beschrieben wird. Eine verständliche Zusammenfassung zu Quaternionen ist in [13] zu finden.

Grundidee

Quaternionen sind eine Erweiterung des Zahlensystems auf vier Dimensionen, analog den komplexen Zahlen. Bekanntlich können Multiplikationen mit komplexen Zahlen als Drehungen auf einer zweidimensionalen Ebene interpretiert werden. Daraus folgt die Idee, dass im dreidimensionalen Raum das selbe gemacht werden kann. Da es kein dreidimensionales Zahlensystem gibt, in dem Multiplikationen und Divisionen definiert sind, kann diese Idee nicht direkt in das dreidimensionale übernommen werden. Es ist jedoch möglich, Drehungen im Raum mit den vierdimensionalen Quaternionen zu beschreiben.

Darstellung

Es gibt viele Möglichkeiten Quaternionen darzustellen. Hier eine Auflistung üblicher Schreibweisen. q_0 ist der Skalarteil und $\vec{q}$ der Vektorteil vom Quaternion $\mathbf{q}$.

- Die „Klassische“ Darstellung $\mathbf{q_{ijk}} = q_1 + iq_2 + jq_3 + kq_4$
- Als Vektor $\mathbf{q} = \begin{pmatrix} q_0 & q_1 & q_2 & q_3 \end{pmatrix}^T = \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix}$
- Als Matrix $\mathbf{Q} = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}$

Multiplikation

Die Multiplikation zweier Quaternionen ist in Gleichung 3.14 definiert.

$$\mathbf{q} \star \mathbf{p} = \begin{pmatrix} q_0 p_0 - \vec{q} \cdot \vec{p} \\ q_0\vec{p} + p_0\vec{q} + \vec{q} \times \vec{p} \end{pmatrix} = \begin{pmatrix} q_0q_0 - q_1q_1 - q_2q_2 - q_3q_3 \\ q_1q_0 + q_0q_1 - q_3q_2 + q_2q_3 \\ q_2q_0 + q_3q_1 + q_0q_2 - q_1q_3 \\ q_3q_0 - q_2q_1 + q_1q_2 + q_0q_3 \end{pmatrix} = \mathbf{P} \cdot \mathbf{q} \qquad (3.14)$$

Die Quaternion Multiplikation ist nicht kommutativ ($\mathbf{q} \star \mathbf{p} \neq \mathbf{p} \star \mathbf{q}$).

konjugiert

Das konjugiert Komplexe eines Quaternions ist in Gleichung 3.15 definiert.

$$\mathbf{q}^* = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{pmatrix}^T \qquad (3.15)$$

Drehungen

Eine Drehung mit Quaternionen im dreidimensionalen Raum kann als eine Drehung um einen Vektor $\vec{n}$ mit einem Drehwinkel φ interpretiert werden.

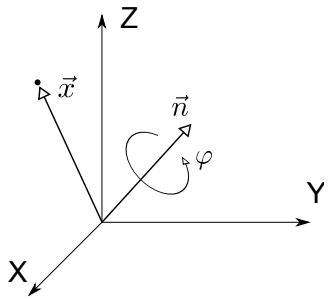


Abbildung 3.3: Drehung um die Drehachse $\vec{n}$

Quaternionen die Drehungen darstellen sind immer auf eins normiert. Die Umrechnung von den physikalisch interpretierbaren Parametern $\vec{n}$ und φ zum Quaternion geschieht folgendermassen.

$$\mathbf{q} = \begin{pmatrix} cos(\varphi) \\ \vec{n} \cdot sin(\varphi) \end{pmatrix} \tag{3.16}$$

$\vec{n}$ hat die Länge eins, folglich hat auch das Quaternion die Länge eins. $\vec{x}$ ist ein Vektor im Raum der gedreht werden soll. Eine Drehung dessen wird folgendermassen vorgenommen.

$$\begin{pmatrix} 0 \\ \vec{x}_t \end{pmatrix} = \mathbf{q} \star \begin{pmatrix} 0 \\ \vec{x} \end{pmatrix} \star \ \mathbf{q}^* \tag{3.17}$$

$\vec{x}_t$ ist der gedrehte Vektor $\vec{x}$.

Matrixform

Ein Rotationsquaternion kann in eine Rotationsmatrix R umgerechnet werden. Die Parametrisierung der Rotationsmatrix R mit Quaternionen ist in Gleichung 3.18 zu finden.

$$R(q_1, q_2, q_3, q_3) = 2 \cdot \begin{pmatrix} \frac{1}{2} - q_2^2 - q_3^2 & q_1 \cdot q_2 - q_0 \cdot q_3 & q_1 \cdot q_3 + q_0 \cdot q_2 \\ q_1 \cdot q_2 + q_0 \cdot q_3 & \frac{1}{2} - q_3^2 - q_1^2 & q_2 \cdot q_3 - q_0 \cdot q_1 \\ q_1 \cdot q_3 - q_0 \cdot q_2 & q_2 \cdot q_3 + q_0 \cdot q_1 & \frac{1}{2} - q_1^2 - q_2^2 \end{pmatrix} \tag{3.18}$$

4 Zeitsynchronisation

allgemein

Um die Frames verschiedener Sensoren kombinieren zu können, müssen die genauen Zeitpunkte deren Aufnahme bekannt sein. Geeignet wäre eine Zeitauflösung von einer Millisekunde. Mit der verwendeten Framerate von 30 Frames pro Sekunde steht alle 33 Millisekunden ein neues Tiefenbild an. Eine Zeitauflösung, welche etwa einem fünftel dieser Zeit entspricht wäre noch tragbar. Und ist auch realistisch. Die Gleichzeitigkeit der Aufnahmen ist von untergeordneter Bedeutung und wurde nicht erstrebt.

4.1 Zeitsynchronisation der Sensoren

Zeitstempel Kinect

Die Frames der Tiefensensoren tragen Zeitstempel. Das heisst der relative Zeitpunkt jeder Tiefensensor-Aufnahme im Vergleich zu früheren oder späteren Tiefenbildern des gleichen Sensors ist bekannt. Die relative Zeitverschiebung zwischen den Verschiedenen Kinect ist jedoch unbekannt. Die internen Uhren der Kinect werden zum Zeitpunkt der Initialisierung gestartet. Leider dauert der Initialisierungsbefehl mehrere hundert Millisekunden. Es ist nicht feststellbar wann genau die Uhr gesetzt wird. Die Synchronisation stellt sich als schwierig heraus, da der Kinect-Sensor nicht zu diesem Zweck gemacht ist, er wird in seinem ursprünglichen Verwendungszweck immer als Einzelgerät verwendet.

Kinect Abstimmung

Die Zeitasynchronität der verschiedenen Sensoren kann also nicht mit dem Abgleich der Computerzeit zu einem gewissen Zeitpunkt festgestellt werden. Aus diesem Grund werden nicht nur die Zeitstempel der Sensoren, sondern auch die Zeitpunkte der Frame-Events abgespeichert. Die Zeitsynchronisation wird in Matlab realisiert. Die Grundidee ist, die Minimale Zeitdifferenz $\Delta t_{\min}$ als Korrektur der verschiedenen Kinect Zeitstempel zu verwenden. Die Berechnung der Zeitunterschiede muss unter Annahmen durchgeführt werden, das Verfahren funktioniert in der Praxis jedoch zuverlässig genug. Es wurde mit einfachen Versuchen verifiziert. Zur Überprüfung wurden visuelle Ereignisse, wie beispielsweise das Zuschlagen eines Buches, verglichen.

Die Zeitdifferenzen Δt für alle Bilder eines Tiefenvideos, sind in der Abbildung 4.1 zu sehen. Sie werden aus der Computer Zeit $\mathrm{t_C}$ beim Eintreffen eines neuen Bildes und der Sensor Zeit $\mathrm{t_S}$ bei der Aufnahme des Bildes berechnet (Zeitstempel).

$$\Delta t = t_{\mathrm{C}} - t_{\mathrm{S}} \tag{4.1}$$

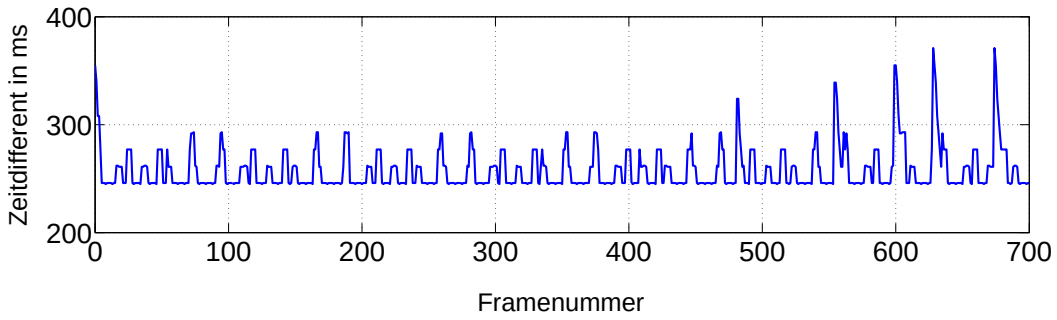


Abbildung 4.1: Zeitdifferenzen Δt zwischen Events und Zeitstempel aller Frames eines Tiefenvideos

Die Zeitdifferenzen Δt bestehen zudem aus $\Delta t_{\mathrm{S,\,C}}$, $t_{\ddot{\mathrm{U}}}$ und $\mathrm{t_B}$. Alles sind unbekannte Variablen. $\Delta t_{\mathrm{S,\,C}}$ ist der Zeitunterschied der beiden Uhren. Also um wie viele Millisekunden die Uhr des Sensors gegenüber der Computeruhr nachgeht. Das eigentliche Ziel wäre es die Konstante $\Delta t_{\mathrm{S,C}}$ zu bestimmen. Das ist nicht möglich, da $\mathrm{t_{\ddot{U}}}$ und $\mathrm{t_B}$ unbekannt sind. $\mathrm{t_{\ddot{U}}}$ ist die Übertragungszeit eines Frames. $\mathrm{t_B}$ ist die Zeit, die das Betriebssystem benötigt, bis ein Event ausgelöst wird.

$$\Delta t = \Delta t_{\mathrm{S,C}} + t_{\ddot{\mathrm{U}}} + t_{\mathrm{B}} \tag{4.2}$$

Bei Betrachtung der Abbildung 4.1 wird die Lösung des Problems verständlich. Die Minimale Übertragungszeit $\Delta t_{\mathrm{min}} = min(\Delta t)$ scheint konstant zu sein. Es gibt in der Abbildung 4.1 keine Zeitdifferenz, welche kleiner ist als ein $\Delta \mathrm{t_{min}}$ von 245 Millisekunden. Die Annahme ist nun, dass gilt

$$t_{\mathrm{B}} = 0 \quad falls \quad \Delta t = \Delta t_{\mathrm{min}} \tag{4.3}$$

dann gilt

$$\Delta t_{\mathrm{min}} = \Delta t_{\mathrm{S,C}} + t_{\ddot{\mathrm{U}}} \tag{4.4}$$

Weiter wird vorausgesetzt, dass $\mathrm{t_{\ddot{U}}}$ eine Konstante, und bei allen Kinect Kameras gleich ist. Unter diesen Voraussetzungen kann das Δt_{min} jedes Sensors i verwendet werden um dessen Uhrzeit zu korrigieren.

$$\Delta t_{\mathrm{Korrektur\ i}} = \Delta t_{\mathrm{min\ i}} \tag{4.5}$$

Die deutlich sichtbaren Abstufungen in Abbildung 4.1 sind ein Hinweis darauf, dass diese durch das Betriebssystem verursacht werden. Sie haben bei diesem Beispiel eine Höhe von genau 15 Millisekunden. Wahrscheinlich handelt es sich dabei um einen Zeitschlitz zugeteilt durch den Prozess-Scheduler des Betriebssystems.

4.2 Synchronisation der Computeruhren

Computerzeit

Die Synchronisation der Uhren der verschiedenen beteiligten Computer geschieht über einen NTP-Server. Die Genauigkeit ist besser als eine Millisekunde. Zu beginn jeder Aufnahme geschieht eine Synchronisation. Um Zeitsprünge zu vermeiden wurden während der Aufnahme keine Computerzeit-Anpassungen vorgenommen. Zur Sicherheit wurden die Computerzeiten über eine simple serielle Schnittstelle alle zwei Sekunden überprüft. Über eine Versuchszeit von 30 Sekunden, wurde eine relative Zeitdrift der beiden Computer von maximal einer Millisekunde festgestellt.

ntpdate

Die Zeitsynchronisation geschieht über den Befehl ntpdate. Der Befehl hat zwei Parameter. Der erste ist -b. Dieser besagt, dass die Computerzeit angepasst werden soll. Der zweite Parameter ist die IP-Adresse eines geeigneten NTP Servers. Es wurde ch.pool.ntp.org verwendet. Der Aufruf sieht so aus:

```
ntpdate -b ch.pool.ntp.org
```

Dieser Aufruf geschieht aus einem neuen Prozess heraus. Damit das Uhr umstellen funktioniert, muss das Hauptprogramm Administratorrechte haben. Deshalb wird es zu Beginn mit diesen nochmals neu gestartet. Einen Prozess mit Administrator-Rechten zu starten ist einfach. Man muss lediglich das Attribut der Startinformation ('StartInfo') des Prozesses auf 'runas' setzen.

Überprüfung

Zur Verifikation der Genauigkeit und zur Kontrolle des Zeit-Drift, wurde eine serielle Schnittstelle verwendet. Im Takt von zwei Sekunden wird ein sehr kurzes Triggersignal gesendet. Die Übertragung eines acht Bit Symbols bei einer Bitrate von 115'200 Bit pro Sekunde dauert circa 70 Mikrosekunden, ist also genügend schnell. Zu diesem Zeitpunkt werden die Uhrzeiten beider Computer in das File 'info.txt' gespeichert. Durch nachträgliches Auswerten kann überprüft werden ob keine zu grossen Zeitunterschiede bestehen.

5 Vorverarbeitung der Daten

Vordergrund

Als erstes wird der Hintergrund vom Vordergrund getrennt. Dieser Schritt ist bei Tiefenbildern relativ einfach, da Tiefeninformation vorhanden ist. Es kann immer davon ausgegangen werden, dass weit entfernte Objekte zum Hintergrund und nahe Objekte zum Vordergrund gehören. Ein Schwellwert ist die einfachste Lösung. Bei den durchgeführten Versuchen würde diese Methode meistens genügen. Um jedoch auch Objekte, die weiter in den Vordergrund ragen entfernen zu können, wird ein Hintergrundbild erzeugt. Jedes Distanzpixel im Tiefenbild, das zu nahe am Hintergrund liegt, wird ausgeblendet.

Hintergrund

Das Hintergrundbild wird beim 'INIT'-Vorgang aus dem Datenverzeichnis geladen. Falls es noch nicht vorhanden ist, wird ein neues erzeugt. Die Erzeugung des Hintergrunds ist einfach. Es wird ein Mittelwertbild aus dem gesamten Video erzeugt, wobei nur Sequenzen berücksichtigt werden in welchen sich keine Bewegungen abspielen. Es ist auch möglich einen Schwellwert zu verwenden. Dazu kann der gesamte Hintergrund auf einen konstanten Wert gesetzt werden.

Segmentierung

Damit Rauschen nicht als Vordergrund mitberücksichtigt wird, wird das Bild in zusammenhängende Segmente zerlegt. Das grösste Bildsegment wird als Vordergrund verwendet. Damit wird erreicht, dass nur ein einziges zusammenhängendes Gebiet weiterverarbeitet wird.

Boden

Am meisten Schwierigkeiten haben die Ausfransungen an den Füssen, wie sie auf allen drei Bildern in Abbildung 5.1 zu sehen sind, bereitet. Diese entstehen durch den direkten Bodenkontakt der Füsse. Da die Tiefensensoren in einem sehr flachen Winkel auf diesen ausgerichtet sind, können sie den Boden nicht erfassen,was grundsätzlich ein Vorteil ist. Leider ist es eine Eigenschaft von Kinect, in der Umgebung von Objekten diesen manchmal kurzzeitig erfassen zu können. Die Störungen zu entfernen ist sehr schwierig, da sie etwa die selbe Grösse haben wie Zehenspitzen und deshalb kaum von diesen zu unterscheiden sind. Die einzige Möglichkeit dazu ist die Fransen anhand älterer Aufnahmen von den Zehen zu unterscheiden, was nicht in der Vorverarbeitung geschieht, sondern direkt mit dem Kalman Tracker (siehe Kapitel 7).

Ungültige Frames

Ist die Anzahl Pixel des extrahierten Vordergrundes kleiner als ein gewisser Schwellwert, wird das betreffende Bild ignoriert, da es sich bei dieser Grösse und einer maximalen Distanz von vier Metern nicht um einen Menschen handeln kann. Bei der verwendeten Auflösung von 640 mal 480 Pixeln haben Versuche ergeben, dass dieser Schwellwert bei etwa 10'000 Pixeln liegt. Weiter werden Frames verworfen, bei

welchen sich Pixel des extrahierten Vordergrundes auf dem linken oder dem rechten Bildrand befinden.

Das Ergebnis der Vorverarbeitung ist in Abbildung 5.1 zu sehen. Es sind drei verschiedene Perspektiven auf die selbe Person abgebildet. Tiefenbilder

Abbildung 5.1: Tiefenbilder verschiedener Perspektiven. Schwarz bedeutet nahe, hell steht für weit entfernte Objekte.

Bis anhin wurde im Sensorkoordinatensystem, wie es im Kapitel 2.1.1 beschrieben ist, gearbeitet. Die X- und Y-Werte sind also Indizes und die Messwerte können auch als Bild interpretiert werden (siehe Abbildung 5.1). Einige der beschriebenen Algorithmen arbeiten im Sensorkoordinatensystem, andere benötigen ein lokales kartesischen Koordinatensystem. Die Umrechnung ist ein weiterer Vorverarbeitungsschritt und wird im Unterkapitel 5.1 beschrieben. Koordinaten

5.1 Kartesisches Sensorkoordinatensystem

Das lokale kartesische Koordinatensystem eines Sensors, wird direkt aus den Rohdaten berechnet. Dies geschieht auf gleiche Weise, wie bereits im Kapitel 2.3 beschrieben. Z_r kann direkt den Rohdaten entnommen werden. Die X_r- und die Y_r-Koordinaten werden aus den x_i, y_i Indizes und aus Z_r berechnet. Die Indizes werden verschoben, so dass sie nicht mehr vom oberen linken Bildrand gemessen, sondern vom Zentrum des Bildes, weshalb der Ursprung des kartesischen Systems genau bei der Linse der Infrarotkamera zu liegen kommt. w ist die Bildbreite und h die Bildhöhe. Weiter wird Y umgedreht, damit die Y-Koordinate von unten nach oben grösser wird, was mehr Sinn macht.

$$X_r = (x_i - w/2 - \frac{1}{2}) \cdot Z \cdot a \tag{5.1}$$

$$Y_r = -(y_i - h/2 - \frac{1}{2}) \cdot Z \cdot a \tag{5.2}$$

Das Koordinatensystem bezieht sich direkt auf den Sensor. Wenn sich die Lage des Sensors ändert, verschieben sich folglich alle Punkte im Koordinatensystem. Zu beachten ist, dass es sich um ein Linkssystem handelt.

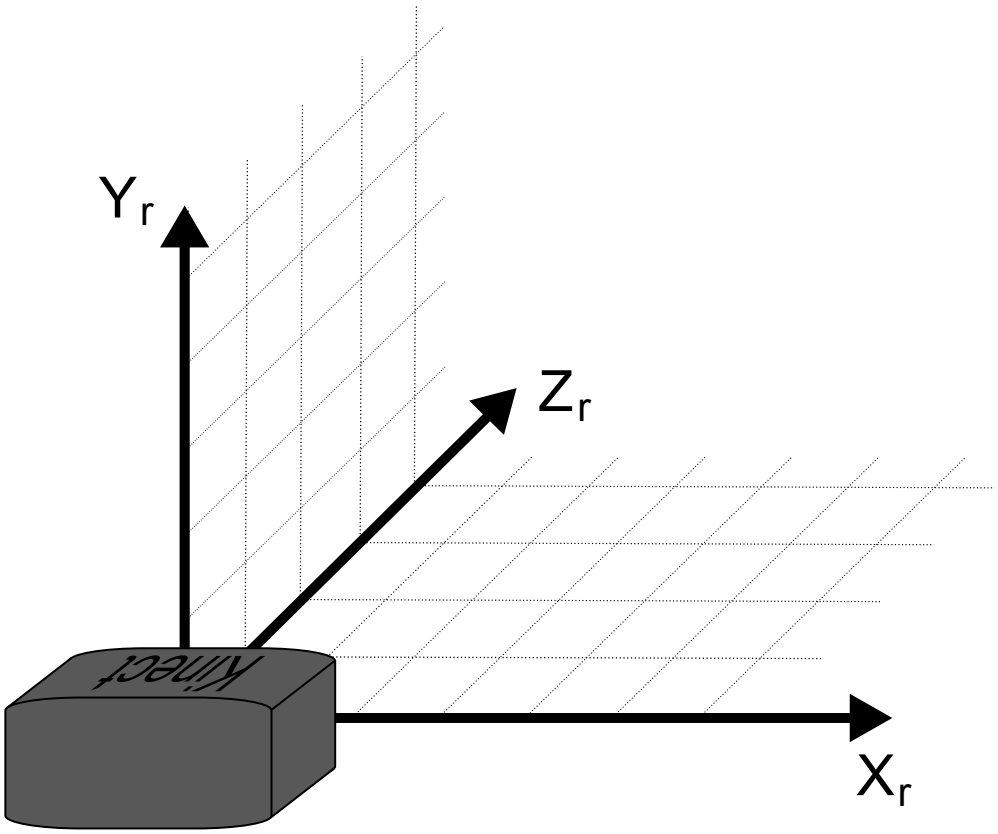


Abbildung 5.2: Lokales kartesisches Koordinatensystem eines Sensors

6 Körperteile erkennen und unterscheiden

Überblick

Dieser Verarbeitungsschritt ist auf dem AGEX-Algorithmus aufgebaut, wie er von [2] vorgeschlagen wird (Siehe Unterkapitel 6.1). Die Analyse eines Tiefenbildes geschieht in zwei Schritten. In einem ersten Schritt wird nach den fünf wichtigsten Extremitäten der Versuchsperson gesucht. Kopf, Arme und Beine. Im zweiten Schritt wird lokal gesucht.

6.1 AGEX Algorithmus

Überblick

Der Algorithmus AGEX wurde von [2] übernommen und bedeutet ausgesprochen **A**ccumulative **G**eodesic **EX**trema, auf deutsch **A**kkumulierende **G**eodätische **EX**trema. Der Name deutet an, dass nach Extremalstellen auf einer Oberfläche gesucht wird. Er basiert auf der Tatsache, dass sich Distanzen auf der Körperoberfläche kaum ändern, egal aus welcher Perspektive die Person gesehen wird, oder in welcher Körperhaltung sich diese befindet.

Vorteile

Zum Beispiel ändert sich der kürzeste Weg von einer Fingerspitze zur anderen kaum. Das ist ein grosser Vorteil, da bei der Bildverarbeitung genau die extreme Variabilität der Aufnahmen das grösste Problem darstellen. Kein Bild sieht aus wie das andere. Deshalb ist ein starkes allgemeingültiges Feature ein grosser Gewinn. Es werden Schlüsselpositionen ermittelt (in [2] „interest points“) und zugleich deren Orientierung bestimmt. Die Orientierung ist für die spätere Klassifizierung von entscheidender Bedeutung. Die „interest points“ nach einer Grobsuche sind in der Abbildung 6.1 zu sehen.

6.1.1 Prinzip

Vernetzung

Als erster Schritt werden die Punkte, welche die Körperoberfläche darstellen, miteinander vernetzt. Jeder Punkt bekommt Verbindungen zu dessen Nachbarpunkten. Damit entsteht ein Maschennetz, bestehend aus Knoten, welche den Punkten entsprechen, und Verbindungen.

Distanzen

Die Oberflächendistanzen werden vom Dijkstra Algorithmus gemessen. Dieser Algorithmus wird im Unterkapitel 6.2 genauer erläutert. Hier genügt es zu erwähnen, dass dieser alle kürzesten Distanzen von jedem Knoten auf einem Maschennetz zu einem einzigen Startknoten ermittelt.

Extremas

Auf dem Maschennetz interessieren nun die entferntesten aller kürzesten Distanzen des gesamten Netzes. Diese entferntesten Knoten liegen, falls der Algorithmus auf einen ganzen Körper angewendet wird, klarerweise auf den Extremitäten wie Zehen, Finger und Kopf. In Abbildung 6.1 sind es Hände und Zehenspitzen. Die längsten kürzesten Pfade sind schwarz eingezeichnet. Die roten Balken zeigen die Orientierung der Schlüsselknoten an.

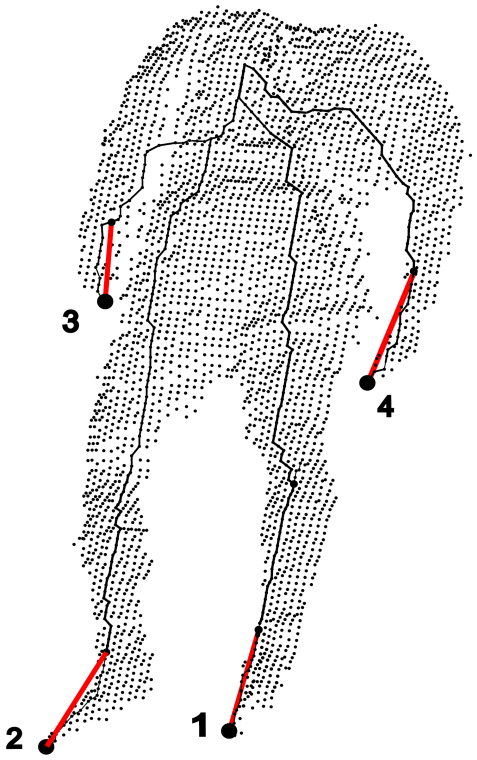


Abbildung 6.1: Schlüsselpositionen (in [2] als „interest points“ bezeichnet)

Ablauf

Gewöhnlich wird im Zentrum begonnen. Der Ort des ersten Startknotens ist jedoch relativ unwichtig. Dijkstra liefert alle kürzesten Pfade. Von diesen wird der Längste ausgewählt. Der Endpunkt ist die erste Schlüsselposition. Die Richtung, aus welcher der Pfad dorthin gelangt ist dessen Orientierung. Nun wird eine neue Verbindung zwischen dem Startknoten und dem gefundenen Schlüsselknoten eingeführt. Sie hat die Distanz Null. Der Punkt wird also virtuell ganz nah an den Startpunkt herangezogen. Damit rücken auch alle Nachbarknoten näher an die Startposition heran. Sie kommen als Schlüsselposition nicht mehr in Frage.

Iteration

Nun wird erneut mit dem Dijkstra auf dem erweiterten Maschennetz gesucht. Die neue Stelle wird erneut mit dem Startknoten verbunden und so wird fortgefahren bis genügend Position gefunden wurden. Normalerweise macht es wenig Sinn mehr als zwanzig Schlüsselstellen zu suchen.

Schwellwert

Sind Distanzen in der Tiefe zwischen zwei Nachbarpixeln grösser als ein Schwellwert, wird zwischen den beiden Knoten keine Verbindung eingetragen. Dies hat den Vorteil, dass starke Ausreisser keinen Einfluss auf den Algorithmus haben.

6.2 Dijkstra Algorithmus

Einführung

Der Dijkstra Algorithmus findet alle kürzesten Pfade eines Netzwerks. Ein Netzwerk besteht aus Knoten und Verbindungen. In Abbildung 6.2 sind die Knoten als grosse Punkte eingezeichnet. Die Nummern bei den Verbindungen sollen deren Länge angeben. Dijkstra liefert immer alle kürzesten Verbindungen. Es ist interessanterweise nicht möglich nur den kürzesten Pfad von A nach B zu finden. Egal welcher Algorithmus gewählt wird, es werden immer alle kürzesten gefunden. Man startet also bei bei A und findet schrittweise alle Pfade bis man den Punkt B erreicht. Der Dijkstra Algorithmus funktioniert nur mit positiven Distanzen.

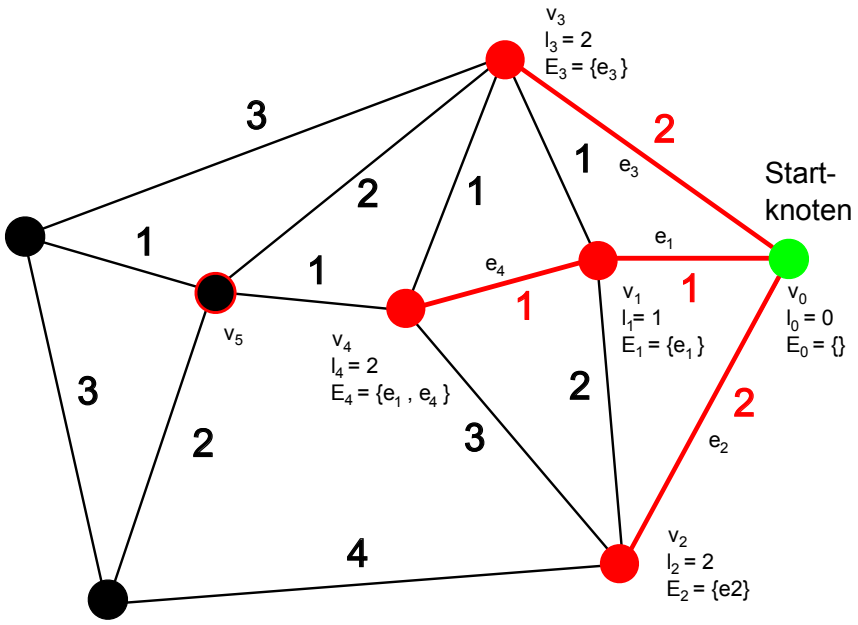


Abbildung 6.2: Suche nach den kürzesten Pfaden mit Dijkstra

Start

Alle Knoten, welche eine bekannte kürzeste Distanz haben werden nummeriert. Man beginnt beim Startknoten und bezeichnet ihn als v_0. Die Distanz zu ihm beträgt natürlich $l_0 = 0$. Die Verbindungen werden mit e_i durchnummeriert. Jeder neu nummerierte Knoten erhält ein Set von Verbindungen E_i, welche der kürzeste Weg zum Knoten v_0 sind. Beim Knoten v_0 ist dies natürlich die leere Menge $E_0 = \{\}$.

Schritt

Nun werden alle Verbindungen durchsucht, welche an einem nummeriert Knoten w beginnen und zu einem Knoten z führen der noch nicht nummeriert ist. $\omega(e)$ ist die Länge der Verbindung von w nach z. Es wird für jede Verbindung die Distanz zum Startknoten berechnet.

$$l_z = l_w + \omega(e_{zw}) \tag{6.1}$$

Der Knoten mit dem kürzeste Pfad zu v_0 wird als v_{i+1} bezeichnet. Dessen länge ist $l_{i+1} = l_z$. Nun ist auch dessen kürzester Pfad E_{i+1} zu v_0 bekannt.

Beispiel

Am Beispiel in Abbildung 6.2 kommt als nächstes der rot umrandete Knoten dazu. Er wird mit v_5 bezeichnet. Der Pfad von v_4 nach v_5 wird zu e_5. Die Distanz zum Knoten v_0 ist $l_{i+1} = l_4 + 1 = 3$. Das Set von v_5 wird zu : $E_5 = \{e_1, e_4, e_5\}$

Komplexität

Die klassischen Implementation hat eine Zeitkomplexität von $O(n^2)$. Es muss jeder der n Knoten einmal abgearbeitet werden. Zusätzlich muss in jedem Schritt der Knoten mit der minimalen Distanz gefunden werden, was einer weiteren linearen Suche entspricht. Wenn mit Vorrang Warteschlangen gearbeitet wird, ergibt sich ein Aufwand von nur noch $O(n \cdot log_2(n))$, da für die Suche nach der minimalen Distanz nur noch ein Aufwand von $O(log_2(n))$ nötig ist. Eine Vorrang-Warteschlangen kann zum Beispiel als AVL-Baum implementiert werden. Neue Einträge werden nach der Grösse links oder rechts an einem Ast angehängt. Das Minimum zu bestimmen ist damit sehr einfach. Der kleinste Eintrag ist immer ganz rechts oder ganz links zu finden.

Implementierung

Die Implementation wurde in c-Code geschrieben, damit die Abarbeitung effizienter wird. Die Geschwindigkeit des gesamten Algorithmus hängt entscheidend von der Dijkstra-Implementation ab. Da eine relativ grosse Zahl von Knotenpunkten vorkommen, wäre eine $O(n \cdot log_2(n))$ Implementierung sinnvoll, wurde aber nicht umgesetzt. Falls die Effizienz des in dieser Arbeit verwendeten Algorithmus gesteigert werden soll, dann ist eine schnellere Implementierung des Dijkstra, mit Vorrang-Warteschlange, sicher sinnvoll.

6.3 Grobsuche

Überblick

Mit Grobsuche ist die Suche nach den Gliedmassen gemeint. Das Ergebnis dieses Verarbeitungsschrittes mit vier AGEX Iterationen ist in Abbildung 6.1 zu sehen.

Subsampling

Bei der Grobsuche wird der AGEX-Algorithmus einmal auf den gesamten Körper angewandt. Da der Dijkstra Algorithmus, und damit auch der AGEX-Algorithmus, einen Aufwand von $O(n^2)$ hat, musste die Anzahl der Knotenpunkte reduziert werden. Dies geschieht mittels Subsampling des Tiefenbildes. Da die Grösse der Körperoberfläche stark von der Distanz abhängig ist, wird der Faktor des Subsamplings mit der mittleren Distanz der Oberfläche variiert.

$$s = (\frac{4000 - Z_O}{2000} + 1) \cdot 2 \tag{6.2}$$

Z_O ist die mittlere Distanz der detektierten Körperoberfläche. s ist der Subsamplingfaktor. Er soll zwei betragen, wenn die Distanz maximal, also vier Meter, ist. Bei abnehmender Distanz wächst s linear an. Damit ist die Anzahl Pixel immer etwa konstant.

Schwellwert

Der Schwellwert muss so eingestellt werden, dass er grösser als der maximale Quantisierungsschritt des aktuellen Frames ist (siehe Kapitel 2.4, Quantisierung). Mit einer

schlechten Auflösung steigt die Wahrscheinlichkeit, dass mehrere Quantisierungsstufen überwunden werden müssen. Deshalb wird für die Grobsuche der maximale Quantisierungsschritt mit zwei multipliziert.

Startpunkt

Der Startpunkt für den AGEX Algorithmus wird mit einer einfachen morphologischen Operation bestimmt. Als erster Schritt wird aus dem Tiefenbild ein Binärbild erzeugt. Es werden also alle Pixel, welche zu der Körperoberfläche gehören zu eins, alle anderen zu Null. Dieses Binärbild wird nun mehrmals erodiert, einen Schritt vor vollständigem verschwinden der letzten Pixel wird gestoppt. Alle diese Pixel liegen etwa im Zentrum und können als Startknoten verwendet werden. Es wurde das Pixel mit dem maximalen y_i Wert gewählt. Es wird also ein Pixel gewählt das möglichst weit oben und damit weit von den Füssen entfernt ist, was für spätere Verarbeitungsschritte von Vorteil ist. Das ist natürlich eine implizite Annahme über die Orientierung der Sensoren. Normalerweise ist diese aber korrekt, da der Fuss des Gerätes standardmässig so montiert ist, dass die y_i Indizes vertikal ausgerichtet werden.

Anzahl Iterationen

Es werden nur drei AGEX Iterationen vorgenommen. Es sind nicht mehr notwendig, da, wie im letzten Abschnitt erwähnt, der Startpunkt weit entfernt von den Füssen gewählt wird. Normalerweise werden deshalb bereits bei den ersten beiden Iterationen die beiden Zehenspitzen gefunden.

Klassifizierung

Die Klassifizierung ist sehr primitiv. Als Füsse werden die beiden Schlüsselstellen gewählt, welche sich zu unterst befinden. Solange sich die Versuchsperson aufrecht bewegt ist diese Klassifizierung sehr stark. An dieser Stelle müsste auch ein komplexerer Klassifizierer eingebaut werden. Es wäre zum Beispiel interessant, ob der Algorithmus mit einem relativ einfachen Klassifizierer, wie etwa mit 'PCA' oder 'Fishers Linear Discriminant' ([3], ab S. 117), verbessert werden könnte. Auf jeden Fall müsste die Grösse der Bildausschnitte unabhängig von der Distanz gemacht werden, was mit dem Subsampling schon geschieht. Weiter müssten diese immer in die gleiche Lage rotiert werden. Dank Orientierung der Schlüsselknoten ist dies auch kein Problem. Hier wird der Zweck der Orientierung offensichtlich. In [2] wurde ein Klassifizierer, mit auf diese Weise gedrehten Bildausschnitten, realisiert.

6.4 Detailsuche

Überblick

Die Grobsuche liefert bereits die Zehen. Im weiteren Verlauf müssen die Fersen-, Knie- und Hüftpositionen gefunden werden. Die Ferse wird mit einer erneuten AGEX Analyse gefunden. Zur Suche nach den Knien und Hüften wird der schon erwähnte Dijkstra Pfad verwendet, welcher bei der Ermittlung jedes AGEX Schlüsselknotens anfällt (Siehe schwarze Linien in der Abbildung 6.1).

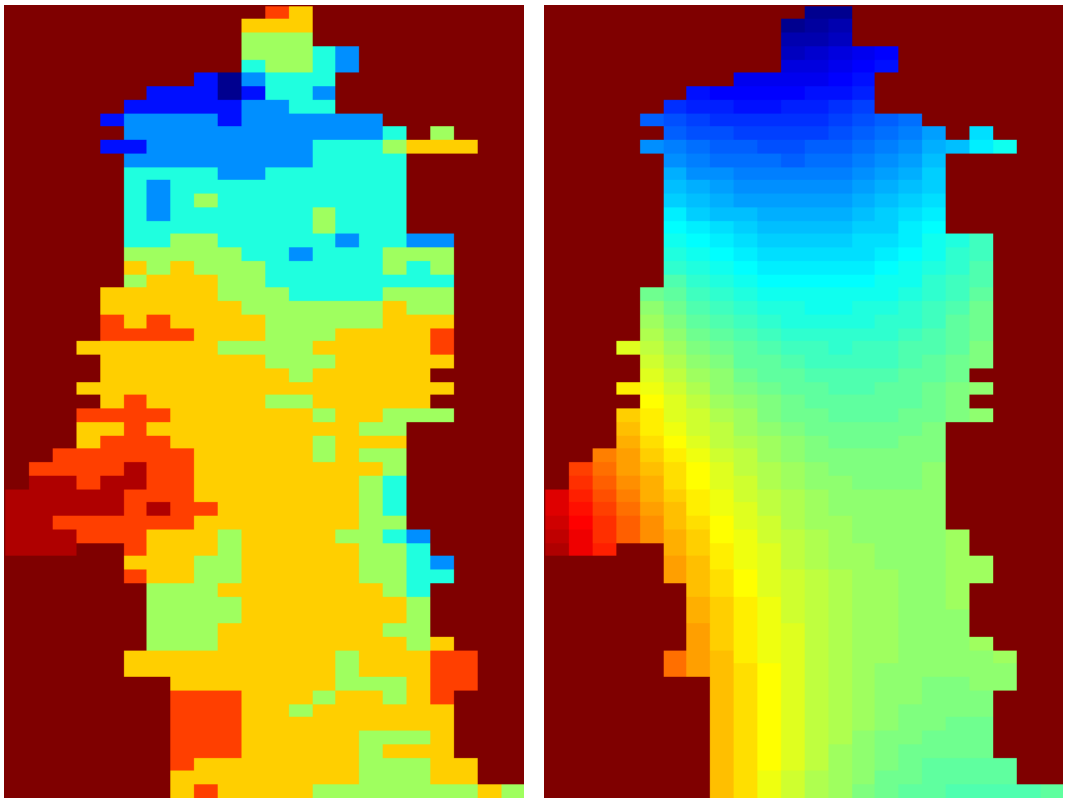
6.4.1 Zehen und Ferse

Auswahlfenster

Für die Suche nach den Fersen wird ein kleines Auswahlfenster um die Füsse gelegt. Dazu wird das „Zentrum der Füsse“ bestimmt, indem ein kleines Stück dem Dijkstra-Pfad des Schlüsselknotens/Zehens rückwärts gegangen wird. Diese Stelle wird zum Zentrum des Fensters. Die Weite w und die Höhe h wird von den Indizes des Fusses $x_{i,F},\ y_{i,F}$ und der Zehen $x_{i,Z},\ y_{i,Z}$ abhängig gemacht.

$$w = h = max(|x_{i,F} - x_{i,Z}|, |y_{i,F} - y_{i,Z}|) \tag{6.3}$$

Damit wird gewährleistet, dass sich immer der ganze Fuss im Bild befindet.



(a) Distanzdaten (b) gefittet

Abbildung 6.3: Auswahlfenster um einen Fuss

Region Growing

Im Auswahlfenster ist sehr oft nicht nur der selektierte Fuss ersichtlich, sondern auch noch andere Körperteile im Hintergrund. Um diese aus dem Bild zu entfernen wird ein einfacher Region Growing Algorithmus verwendet. Es wird von einem Startpunkt ausgegangen, von welchem bekannt ist, dass er auf dem Fuss liegt (Fenstermitte). Im folgenden werden alle Pixel als dem Fuss zugehörig klassifiziert, die ein Nachbarpixel haben, welches schon auf dem Fuss liegt und das gleichzeitig nicht zu weit entfernt ist.

Fitting

In der Abbildung 6.3 b sieht man das mit diesem Schritt erreichte Ergebnis. Es wurde ein Zweidimensionales Polynom zweiten Grades in die Punktwolke gefittet.

$$Z = a \cdot x^2 + b \cdot y^2 + c \cdot x \cdot y + d \cdot x + e \cdot y + f \tag{6.4}$$

Die Methode der kleinsten Quadrate ergibt ein zufriedenstellendes Ergebnis. Damit konnte das Rauschen sehr gut unterdrückt werden. Vor allem problematische Ausreisser, wie in der Abbildung 6.3 a ganz oben ersichtlich, konnten damit elegant entfernt werden. Solche Ausreisser kommen sehr häufig vor. Der Schlüsselknoten für den Zehen kommt in diesen zu liegen, was dessen Position stark verfälscht. Mit dem Modell konnte die Position der Zehen jeweils korrigiert werden.

Ferse

Die Position der Ferse wird mit einer weiteren AGEX Analyse, angewendet auf das Auswahlfenster, ermittelt. Der Startknoten wird auf der Zehenspitze gewählt. Die erste gefundene Position ist unbrauchbar. Sie liegt auf der anderen Seite des Bildes, bei der Stelle wo das Bein vom Auswahlfenster abgeschnitten wurde. Beim zweiten 'Interest Point' handelt es sich ziemlich zuverlässig um die Ferse. Bei Betrachtung der Abbildung 6.3 b ist das Prinzip leicht verständlich. Falls das Tracking ergeben sollte, dass dieser Wert sehr unrealistisch ist, wird die Position analog zu der Hüfte und dem Knie ermittelt (siehe Kapitel 6.4.2).

6.4.2 Oberflächendistanzen

Skalierung

Da nicht alle Personen gleich gross sind und deshalb die Oberflächendistanzen verschiedener Personen variieren, werden diese mit einem personenspezifischen Skalierungsfaktor angepasst. Der Skalierungsfaktor wird aus den Tiefendaten geschätzt. Er muss linear zur Körpergrösse der Versuchsperson sein. Es wird die Kovarianzmatrix Σ aus den Daten ermittelt.

$$\Sigma = \frac{1}{N-1} \sum_{k=1}^{N} (\vec{x}_k - \vec{m}) \cdot (\vec{x}_k - \vec{m})^T; \tag{6.5}$$

$\vec{x}_k$ sind die Punkte in der Punktwolke. $\vec{m}$ ist der Mittelwert der Punktwolke. Der maximale Eigenwert von Σ ist λ_{max}. Er wird zur Schätzung der Körpergrösse verwendet. λ_{max} ist die Varianz der Punktwolke in die Richtung ihrer maximalen Ausdehnung. Der Skalierungsfaktor s ist die Standardabweichung.

$$s = \sigma_{max} = \sqrt{\lambda_{max}} \tag{6.6}$$

Damit hat man ein Mass für die Grösse der Punktwolke in die Richtung seiner maximalen Ausdehnung und damit auch ein Schätzmass für die Körpergrösse. Leider kann s auch falsch sein. Zum Beispiel, wenn die Person in die Hocke geht. Es müssen die Schätzwerte für s mehrerer Aufnahmen kombiniert werden.

Falls s grösser ist als das neu dazukommende s_i, wird dem schon vorhandenen Schätzwert s viel Vertrauen geschenkt, s_i hat einen sehr kleinen Einfluss auf s.

$$s = 0.999 \cdot s + 0.001 \cdot s_i \tag{6.7}$$

das Gegenteil gilt, wenn s kleiner ist als s_i. Nun wird dem neuen Schätzwert s_i mehr vertraut.

$$s = 0.9 \cdot s + 0.1 \cdot s_i \tag{6.8}$$

Damit tendiert s zum Maximum aller s_i. Oberflächendistanzen können nun mit der personenunabhängigen Masseinheit s im Algorithmus verwendet werden. Für die Umrechnung in Millimeter werde diese mit s multipliziert.

Knie/Hüfte

Das Knie und die Hüfte sind einfach aufzuspüren. Vorausgesetzt die Körpergrösse ist bekannt, kann auch die Position der Hüfte und des Knies abgeschätzt werden. Man muss lediglich den Dijkstra Pfad von der Zehe aus zurückgehen. Wie bereits erwähnt sind Distanzen auf der Körperoberfläche immer etwa gleich lang. Das heisst auch, dass die kürzeste Distanz entlang der Oberfläche bis zum Knie oder irgend einem anderen Körperteil gleich lang bleibt. Es wurden folgende Distanzen verwendet:

- Knie $l_k = 0.625 \cdot s$
- Hüfte $l_h = 0.375 \cdot s$

Problem

Das Ausstrecken der Arme führt kurzzeitig zu einer Verfälschung von s. Da es hauptsächlich um eine Ganganalyse geht, wurde dieser Fakt im folgenden ignoriert.

7 Bewegungs Tracking

allgemein

Die Positionen der verschiedenen Körperpunkte wurden statisch ausgewertet, das heisst ohne Berücksichtigung zeitlich oder räumlich versetzter Aufnahmen. Diese können jedoch die Information enthalten, welche dem einzelnen Tiefenbild fehlt. Die dynamische Auswertung wird in diesem Kapitel beschrieben.

7.1 Globales Koordinatensystem

Um die Daten verschiedener Sensoren kombinieren zu können, muss ein einheitliches, globales Koordinatensystem definiert werden, welches für alle Sensoren gültig ist. Aus praktischen Gründen ist das lokale kartesische Koordinatensystem des Kinect mit der Nummer eins äquivalent mit dem globalen Koordinatensystem. Diese Wahl macht Sinn, da die Lageparameter aller Kinect-Sensoren standardmässig zur Kinect Nummer eins bezogen werden.

7.2 Kalman Tracker

Grundprinzip

Zuerst werden die gefundenen Punkte in das globale Koordinatensystem transformiert. Für jede Gelenkposition existiert ein eigenes Kalman-Filter. Das Kalman-Filter macht aufgrund früherer Messungen eine Vorhersage, wo die nächste Messung liegen sollte. Danach wird ein Gate um die vorhergesagte Position gelegt. Das Gate ist ein Raumvolumen, in welchem die Messungen mit hoher Wahrscheinlichkeit zu liegen kommen. Sind sie ausserhalb dieses Bereiches werden sie ignoriert. Sind sie innerhalb, fliessen die neuen Daten in den Kalman-Filter ein.

Initialisierung

Beim ersten Tiefenbild werden alle Parameter des Trackers mit den Messwerten aus dem ersten Tiefenbild initialisiert. Genaueren Angaben dazu finden sich im Kapitel 7.2.3 Kalman-Filter.

7.2.1 Zuordnung der Messungen zu den Kalman Vorhersagen

Allgemein

Die Messungen müssen den entsprechenden Körperteilen zugeordnet werden. Diese Information ist jedoch grösstenteils schon vorhanden, da spezifisch nach den Körperteilen gesucht wurde und diese bereits klassifiziert wurden. Jedoch müssen sämtliche Messungen dem linken oder dem rechten Bein zugeordnet werden.

Schwierigkeit

Es kann nicht einfach die Messung mit dem kleineren Abstand zugeordnet werden. Um dies zu verstehen kann die Abbildung 7.1 betrachtet werden. Die Kreuze stellen die Messungen dar, die den Beinen (als Kreise dargestellt) zugeordnet werden müssen. Zum Beispiel im oberen rechten Fall wäre die Zuordnung bei einer Entscheidung aufgrund der Distanz nicht eindeutig. Würden zuerst die Distanzen zum linken Kreis gemessen, würde die Entscheidung heissen: Das untere Kreuz gehört zum linken Bein. Umgekehrt würde das untere Kreuz dem rechten Bein zugeordnet.

Lösung

Als erstes muss entschieden werden, welcher der vier Fälle in der Abbildung 7.1 zutrifft. Danach kann die Zuordnung gemacht werden.

- In den beiden Fällen links oben und rechts unten ist die Zuordnung klar.
- Im Fall rechts oben liegen beide Messungen näher am rechten Bein. Es ist also wahrscheinlich, dass die Messung die näher am rechten Bein liegt auch die Messung des rechten Beines ist. Eine Zuordnung mit den Distanzen zum rechten Bein ist sinnvoller.
- In der linken unteren Ecke sind beide Messungen näher am linken Bein. Die Zuordnung geschieht also aufgrund der Distanzen zum linken Bein.

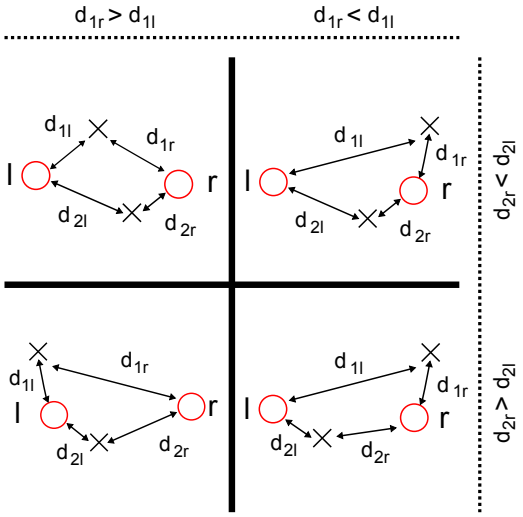


Abbildung 7.1: Fallunterscheidung

7.2.2 Gate

Allgemein

Dieser Ansatz wurde [8] entnommen. Darin wird ein GNN-Tracker (Global Nearest Neighbor Tracker) beschrieben. Es handelt sich hier nicht um einen GNN-Tracker, da keine komplizierten Zuordnungsprobleme existieren. Dem Paper wurde aber die Idee, die in diesem Unterkapitel beschrieben wird, entnommen.

Gewichtsmatrix

Die Grundidee ist, nicht in euklidischen Distanzen zu messen. Da die Vorhersagegenauigkeit der Positionen nicht in jede Richtung zu jeder Zeit gleich gut ist, macht es

Sinn das Gate in der Richtung mit der schlechteren Vorhersagegenauigkeit grösser zu machen. Deshalb werden die Distanzen wie in [8] beschrieben berechnet. Die Kovarianzmatrix des Schätzfehlers P und die Kovarianzmatrix des Messrauschens Q_v vom Kalman-Filter enthalten eine Schätzung der Vorhersageabweichung. Deshalb wird aus diesen eine Gewichtungsmatrix S errechnet.

$$S = H \cdot P \cdot H^T + Q_v \tag{7.1}$$

$$\vec{y} = \vec{z} - H \cdot \vec{\hat{x}} \tag{7.2}$$

In H enthält die Messinformation. Mit H wird also bestimmt, welche $\vec{x}$ gemessen werden (siehe Kalman-Filter [11]).

Weiter wird die Differenz aus den gemessenen und den vorhergesagten Werten genommen. $\vec{z}$ ist die momentane Messung und $\vec{\hat{x}}$ die Vorhersage des Kalman-Filters. Weiter wird die quadrierte Distanz d^2 berechnet.

$$d^2 = \vec{y}^{\,T} \cdot S^{-1} \cdot \vec{y} \tag{7.3}$$

Nun kann ein Gate G definiert werden, in welchem die Messungen liegen sollen. Ist d^2 kleiner als G ist die Messung gültig und fliesst in den Kalman-Filter ein. Im anderen Fall handelt es sich um eine Fehlmessung und sie wird verworfen.

7.2.3 Kalman-Filter

Allgemein

Der Trackingalgorithmus baut auf einem Kalman-Filter auf. Eine Einführung in den Kalman-Filter ist in [11] zu finden. In diesem Unterkapitel sollen nur die Parametereinstellungen, welche vorgenommen wurden, dokumentiert werden.

Modell

Als Modellannahme wurde eine konstante Geschwindigkeit gewählt. Obwohl diese Annahme offensichtlich nicht richtig ist, ergeben sich damit gute Resultate. Die A Matrix der Zeit-Update-Gleichung sieht folgendermassen aus:

$$A = \begin{pmatrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.4}$$

T ist die Zeit, welche verstrichen ist, seit dem letzten Zeit-Update. Die Zeit wird auf eine Framerate von 30 Frames pro Sekunde normiert ($T = T_{sec} \cdot 30$). T_{sec} ist die verstrichene Zeit in Sekunden.

Der Zustandsvektor $\vec{\hat{x}}_k$ enthält die Positionen und Geschwindigkeiten in den drei Raumrichtungen.

$$\vec{\hat{x}}_k = \begin{pmatrix} x_1 & v_1 & x_2 & v_2 & x_3 & v_3 \end{pmatrix}^T \tag{7.5}$$

Prozessrauschen

Die Kovarianzmatrix des Prozessrauschens Q_w ist ein reiner Erfahrungswert. Ist Q_w kleiner, wird mehr dem Modell vertraut, ist Q_w grösser werden die Messwerte stärker gewichtet. Bei grosser Beschleunigung dürfen sich die gefilterten Werte nicht zu träge verhalten. Auf der anderen Seite sollte das Messrauschen genügend unterdrückt werden.

$$Q_w = 4 \cdot I_6 \tag{7.6}$$

I_6 ist die Identitätsmatrix der Grösse sechs.

Messrauschen

Das Messrauschen wurde in einem Versuch ermittelt. Es wurde eine Sequenz ausgewertet, in der keine Bewegung vorkommt. Danach wurde die Standardabweichung der Messungen berechnet. In die drei Raumrichtungen konnten folgende Streuungen ermittelt werden:

- $\sigma_x = 21.7$
- $\sigma_y = 25.4$
- $\sigma_z = 49.6$

Es wurde die grösste der drei Standardabweichungen für die Kovarianzmatrix verwendet.

$$Q_v = 50 \cdot I_3 \tag{7.7}$$

Anpassung Q_w

Der Kalman Filter ist unter der Annahme von weissem gaussschem Rauschen optimal. Leider trifft dies beim Prozessrauschen offensichtlich nicht zu. Zwischen zwei Messungen vergeht nicht immer gleich viel Zeit. Es kommt vor, dass zwei Aufnahmen gleichzeitig gemacht werden. In dieser Zeit hat sich das Objekt nicht bewegt und deshalb kann es auch keinen Prozessfehler gegeben haben, der ja zum grossen Teil aus der vernachlässigten Beschleunigung entsteht. Deshalb wird Q_w mit T gewichtet. T ist null, wenn keine Zeit seit der letzten Aufnahme vergangen ist und eins, wenn die Zeit der normalen Periodendauer, also 33 Millisekunden entspricht.

$$P = A \cdot P \cdot A^T + Q_w \cdot T \tag{7.8}$$

Initiale Parameter

$\vec{\hat{x}}_0$ wird mit den entsprechenden Messungen des ersten Frames initialisiert. Die initialen Geschwindigkeiten sind null. $\vec{z}_0$ ist die Messung des ersten analysierten Tiefenbildes.

$$\vec{\hat{x}}_0 = H^T \cdot \vec{z}_0 \tag{7.9}$$

Das erste P wird als P_0 bezeichnet. P enthält die erwarteten Fehler der Kalman-Vorhersage.

$$P_0 = E\{(\vec{\hat{x}}_0 - \vec{x}_0) \cdot (\vec{\hat{x}}_0 - \vec{x}_0)^T\} = E\{(H^T \cdot \vec{z}_0 - \vec{x}_0) \cdot (H^T \cdot \vec{z}_0 - \vec{x}_0)^T\} \tag{7.10}$$

Da am Anfang die Messwerte zur Initialisierung der Position verwendet werden, macht es Sinn die entsprechenden Einträge von P_0 mit dem Messrauschen gleichzusetzen. Die zu der Geschwindigkeitsschätzung gehörenden Einträge wurden nicht statistisch ermittelt, sondern lediglich geschätzt. Zum Ermitteln dieser Parameter müsste eine Statistik über die Geschwindigkeiten gemacht werden.

$$P_0 = \begin{pmatrix} 50 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{pmatrix} \tag{7.11}$$

8 GUI

allgemein

Zu Darstellungszwecken und zur angenehmeren Bedienung wurde für den Hauptalgorithmus ein einfaches GUI programmiert. Es ist in der Abbildung 8.1 ersichtlich. Es wird automatisch gestartet, wenn das Matlab Programm 'main.m' aufgerufen wird. Es wird Bild für Bild in der richtigen zeitlichen Reihenfolge angezeigt.

Ansichten

Auf der Linken Seite ist eine dreidimensionale Darstellung der Punktwolke ersichtlich. In die Punktwolke sind grün die Beine eingezeichnet. Mit Kugeln wird der momentan erwartete Vorhersagefehler angezeigt. Auf der rechten Seite sieht man mehrere Ansichten. Oben ist eine 3D Skelettansicht in einem Raum mit absoluten Koordinaten ersichtlich. Darunter ist auf der linken Seite eine zweidimensionale Projektion des Skelettes auf die X-Y-Ebene dargestellt. Auf der rechten Seite ist das Tiefenbild des Kinect Sensors, wie es nach der Vorverarbeitung aussieht, ersichtlich.

Bedienung

Die Analyse kann mit dem 'play'/'pause' Button pausiert werden. In der Abbildung ist dieser mit 'play' angeschrieben, dies deutet an, dass das Programm im Pausen-Modus war. In diesem kann mit 'step back' und 'step forward' in den alten Auswertungsdaten navigiert werden. Die dreidimensionalen Darstellungen sind mit der Computermaus, wie in normalen dreidimensionalen Matlab-Plots, drehbar. Damit können die Grafiken aus allen erdenklichen Perspektiven ausgewertet werden. Wenn sich das Programm im 'play'-Modus befindet, wird ein Bild nach dem anderen analysiert und angezeigt. Mit Klick auf 'stop program' kann das Programm beendet werden.

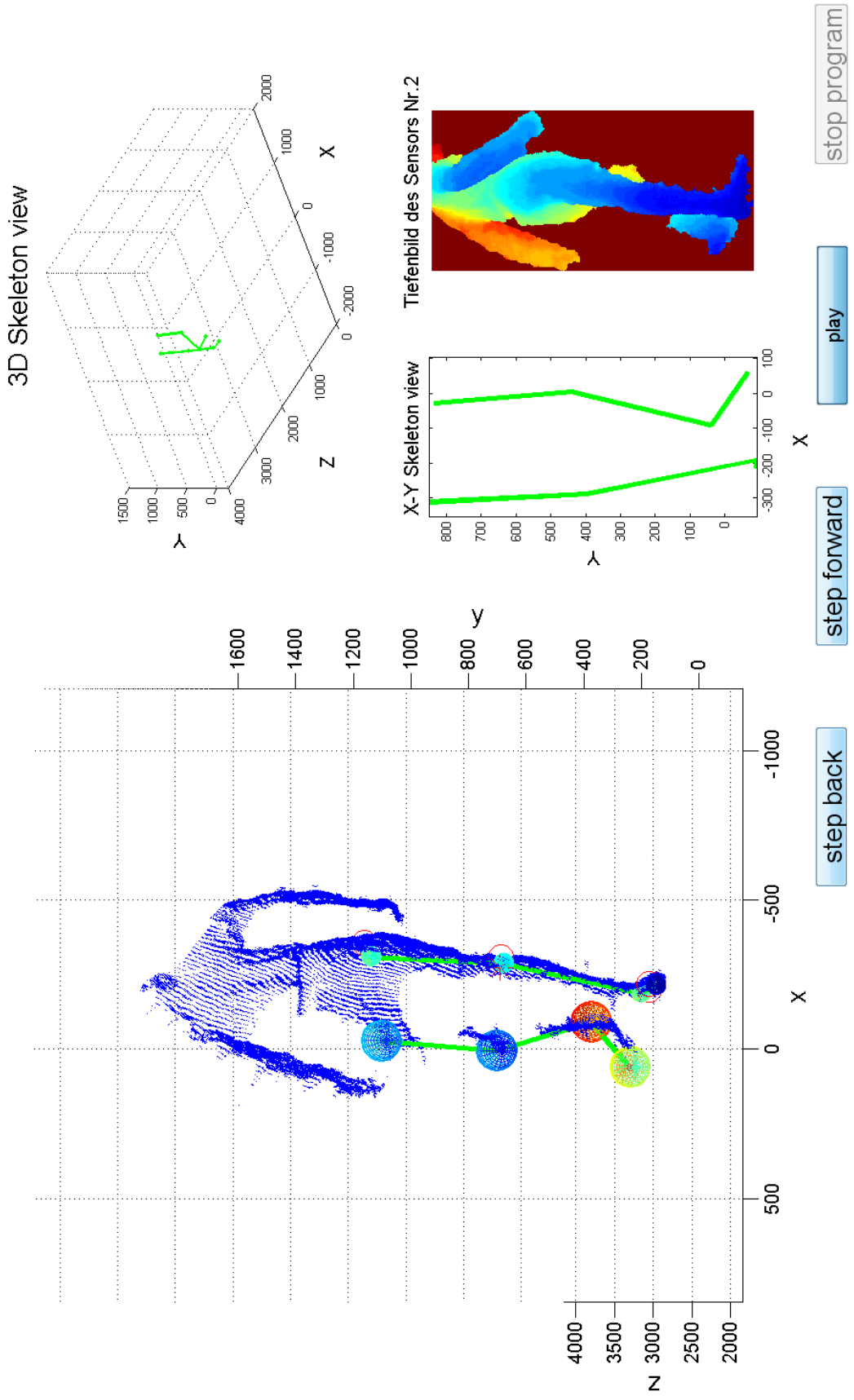


Abbildung 8.1: GUI, alle Achsen in Millimeter

9 Versuche

9.1 Übersicht

Verteilung

Die Anzahl Sensoren welche verwendet werden ist im Prinzip beliebig und für den in dieser Arbeit beschriebenen Algorithmus nicht von Belang. Für alle Versuche wurde die Anzahl Kinect auf maximal vier beschränkt. Die Kinect-Sensoren können beliebig im Raum verteilt werden. Es ist jedoch vorteilhaft möglichst viele Überschneidungen der Sensor Sichtfelder zu haben, damit die Versuchsperson auf mehreren Kameras aus verschiedenen Perspektiven sichtbar ist. Für die hier beschriebenen Versuche wurden drei Sensoren in einem Kreis angeordnet.

Justierung Sensorlage

Vor jedem Versuch müssen die Positionen und die Winkel aller für die Aufnahmen verwendeten Kinect-Sensoren bestimmt werden. Man könnte sie von Hand ausmessen. Diese Variante wäre aber sehr mühsam und, insbesondere für die Bestimmung der Sensorwinkel, ziemlich ungenau. Besser ist es diese anhand der Sensordaten mittels eines vorangehenden Versuches zu bestimmen. In Abbildung 9.1 entspricht dies dem oberen Ablauf. Es müssen kugelförmige Marker, von allen Sensoren gleichzeitig erfassbar, platziert werden. Nachdem die Aufnahmen gemacht wurden, werden diese mit dem Matlab-File 'findSensorPositions.m' ausgewertet. Genauer erläutert wird Vorgehen und Prinzip im Kapitel 9.2.

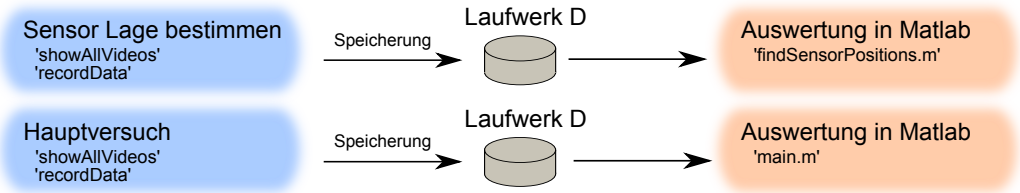


Abbildung 9.1: Schematischer Ablauf aller Versuche

Aufnahmen

Ist die Lage der Sensoren bekannt, können Aufnahmen für das Tracken der Körperteile gemacht werden. Diese werden lokal auf den Rechnern gespeichert. Eine genaue Versuchsbeschreibung ist im Kapitel 9.3 zu finden. Wichtig ist dabei zu beachten, dass sich die Person immer im Sichtfeld von möglichst vielen Sensoren befinden sollte. Zwischen den beiden Versuchen, Lagebestimmung und Tracken der Körperteile, darf keiner der Sensoren mehr bewegt werden. Die Auswertung aller Aufnahmen geschieht in Matlab und wurde, wie in Abbildung 9.1 dargestellt, nicht in Echtzeit realisiert.

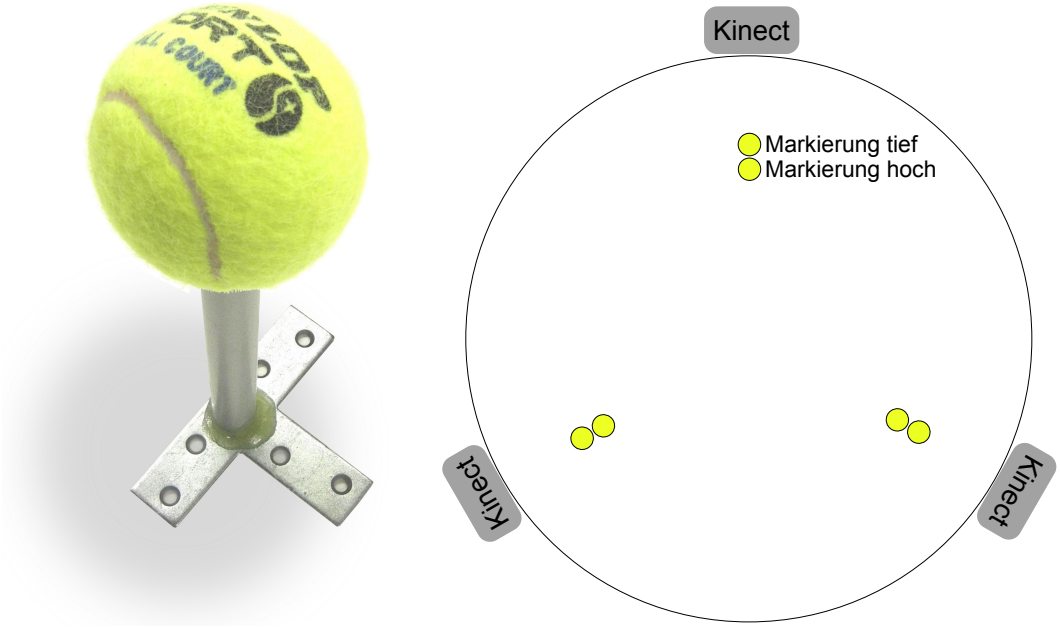
9.2 Versuch zur Lagebestimmung der Kinect Sensoren

Wie in Kapitel 3 beschrieben, kann die Position und gegenseitige Ausrichtung der Sensoren aus den Tiefenbilddaten geschätzt werden. Dazu müssen in einem vorausgehenden Versuch kugelförmige Marker, in einem für alle Sensoren erfassbaren Gebiet, angebracht werden. Nachdem alle Markierungen platziert wurden, kann eine Aufnahme gemacht werden, wie es im Kapitel 9.3.2 erläutert ist. Diese Aufnahme wird dann mit dem Matlab File 'findSensorPositions.m' ausgewertet (Kapitel 9.2.1).

9.2.1 Marker

Allgemein

Als Markierungen wurden Tennisbälle verwendet. In Abbildung 9.2 a zu sehen. Insgesamt wurden sechs Tennisbälle positioniert. Ein Objekt von der Grösse eines Tennisballs ist von den Sensoren aus maximal 2.5 Metern erfassbar. Grössere Bälle könnten weiter weg vom Sensor positioniert werden und wären deshalb etwas besser geeignet. Alle Markierungen müssen immer von allen Sensoren erfassbar sein. Dies macht deren Positionierung anspruchsvoll. Die Anzahl der Marker muss grösser oder gleich drei sein. Zur Auswertung eines Versuches zur Lage Bestimmung kann das Matlab File 'findSensorPositions.m' benutzt werden. Die Sensor Positionen werden in den Files 'turnPar1.mat' bis 'turnParN.mat', im Verzeichnis der Aufnahmen zur Lagebestimmung, abgelegt. Die Matlab Files können in das Verzeichnis der Aufnahmen für das Tracken kopiert werden, damit dem Hauptalgorithmus ('main.m') die noch fehlenden Parameter zur Verfügung stehen.



(a) Marker (b) Anordnung

Abbildung 9.2: Markierungen und deren Anordnung

Platzierung

In einem ersten Versuch wurden nur drei Markierungen verwendet. Diese wurden sämtliche direkt auf dem Boden platziert. In einem späteren Schritt wurden alle drei Tiefenbilder in einer dreidimensionalen Darstellung zusammengefügt. Die Tiefendaten konnten in Bodennähe mit guter Genauigkeit zusammengefügt werden, weiter oben nicht mehr. Deshalb wurden weitere Versuche mit sechs Markierungen unternommen. Die drei zusätzlichen Markierungen wurden auf Hüfthöhe platziert. Die Anordnung ist in Abbildung 9.2 b skizziert.

Mittelwertbild

Das Hintergrundbild in Abbildung 9.3 ist ein mittleres Bild aller Tiefenbilder einer Video Sequenz. Jeder Punkt, der weiter als drei Meter entfernt ist wird ausgeblendet, da die Tennisbälle in dieser Distanz nicht mehr erfassbar sind. Weiter wurden alle ungültigen Werte, als Pixel mit dem Wert Null, ignoriert. Auch ignoriert werden Pixel die in weniger als 10 Prozent der Tiefenbilder einen Messwert haben. Mit diesen Massnahmen konnte ein relativ fehlerfreies Tiefenbild produziert werden.

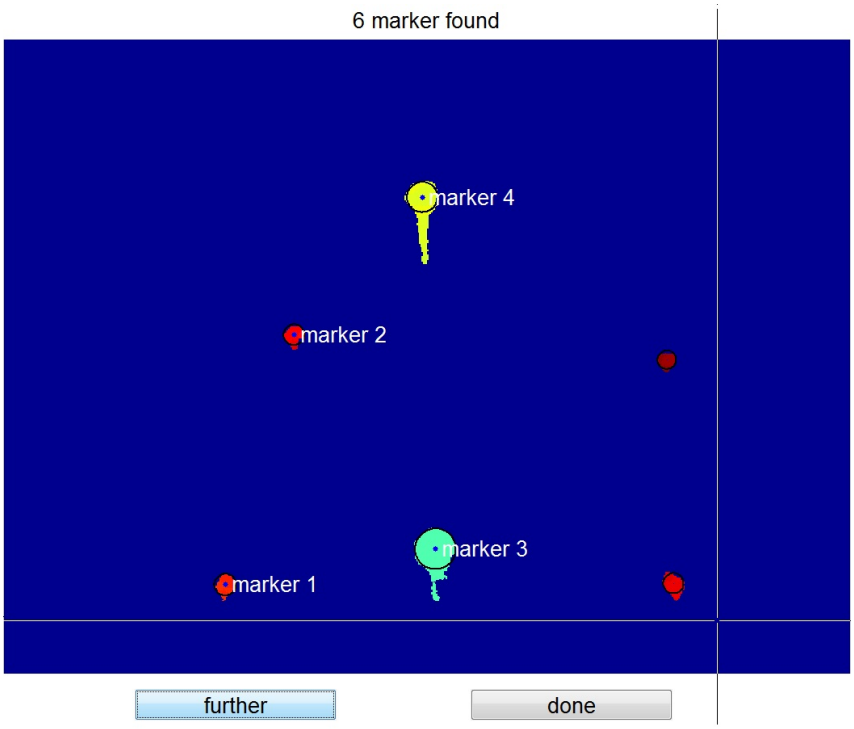


Abbildung 9.3: GUI zur manuellen Nummerierung der Marker

Marker finden

Der Algorithmus zum finden der Markierungen ist einfach. Es ist darauf zu achten, dass im Video möglichst keine anderen Gegenstände ersichtlich sein dürfen. Alles, was weniger als drei Meter vom Sensor entfernt ist, wird vom Programm als potentielle Markierung erkannt. Die Marker werden vom Sensor nicht als Kugelförmige Punktwolken erfasst. Deshalb kann nicht einfach der Mittelwert der Segmente als Position verwendet werden. Das Hauptproblem sind die Stäbchen, auf welchen die Bälle positioniert wurden. Wie in Abbildung 9.3 ersichtlich ist, sind diese in den Tiefenbildern teilweise sichtbar. Eine erste Idee war den Median zu verwenden, dies führte jedoch nicht direkt zum gewünschten Ergebnis. Es wird iterativ vorgegangen. Als erstes wird der Median vom gesamten Segment ermittelt. Danach wird ein

Kreis um das Resultat gezogen, welcher etwas grösser als ein Tennisball ist. Von den Punkten in diesem Gebiet wird wiederum der Median genommen, und so weiter. Dies führt dazu, dass sich die schwarzen Kreise, in Abbildung 9.3 sichtbar, zu einer vernünftigen Position bewegen.

Nummerierung

Die Marker werden von einem Algorithmus selbständig gefunden. Die Nummerierung der Marker muss jedoch manuell vorgenommen werden. Der Benutzter muss also wissen, welche Markierungen auf den Bildern der verschiedenen Sensoren die gleichen sind und diesen eine einheitliche Nummer zuweisen. Dazu dient das einfache GUI, welches in Abbildung 9.3 zu sehen ist. Das grosse Kreuz im Bild ist der Cursor mit welchem die Nummern zugeordnet werden können. Mit schwarzen Kreisen werden die gefundenen Marker angezeigt. Mit 'further' kann die nächst höhere Nummer zugeordnet werden, mit 'done' wird zur nächsten Kinect Perspektive gewechselt. Wurden zu viele Markierungen gefunden, kann einfach nur den richtigen Suchresultaten eine Nummer zugewiesen werden.

Versuche

In den im Kapitel 9.3.2 beschriebenen Versuchen, war die Anordnung der Marker und Sensoren wie in Abbildung 9.2 b ersichtlich. Die Nummerierung wurde jeweils im Uhrzeigersinn vorgenommen.

9.3 Versuchssequenzen

9.3.1 Versuchsbeschrieb

Allgemein

Es wurden viele Versuche gemacht. Zunächst wurden die Positionen noch von Hand ausgemessen. Es hatte sich schnell gezeigt, dass so die Winkelgenauigkeit nicht genügt. Es wurde darauf verzichtet diese ersten Aufnahmen weiter auszuwerten.

Abbildung 9.4: Versuch mit drei Kinect-Sensoren und zwei Computern

Nachdem der Justieralgorithmus entwickelt worden war, wurden Aufnahmen gemacht, welche mit drei Markern justiert wurden. Diese befanden sich auf dem Boden und waren nicht Kugelförmig. Nun wurde in Bodennähe eine gute Genauigkeit erreicht. Die Daten zu diesem Versuch sind im Verzeichnis

- *'Videos 03.12.2011 19.01.52 mit marker 1'*

zu finden. Der Algorithmus funktioniert bei diesen Aufnahmen noch nicht optimal.

Zuletzt wurden wurden Sechs Kugelförmige Tennisbälle verwendet, wie sie in der Abbildung 9.2 a im Kapitel 9.2.1 zu sehen sind. Die Daten zu diesen Versuchen sind in den Verzeichnissen

- *'Videos 22.12.2011 17.52.01 slow'*
- *'Videos 22.12.2011 17.47.47 slow and fast'*
- *'Videos 22.12.2011 17.53.34 fast'*

zu finden.

Anordnung

Die Anordnung wie sie in der Abbildung 9.2 b zu sehen ist, wurde in allen Versuchen verwendet. Das Ziel war die Versuchsperson aus möglichst vielen verschiedenen Perspektiven zu sehen. Diesen Vorteil bietet die Anordnung im Kreis. In der Abbildung 9.4 ist der Versuchsaufbau vom 22.12.2011 zu sehen. Der Markierte Kreis in der Mitte, zeigt das Gebiet an, in welchen sich die Sichtfelder aller drei Sensoren überschneiden.

9.3.2 Aufnahmen

Allgemein

In diesem Unterkapitel wird erläutert, wie eine mit dem Hauptalgorithmus kompatible Aufnahme gemacht werden kann. Als Hilfestellung ist zudem eine stichwortartige Anleitung im Anhang B zu finden. Zum Aufnehmen von Tiefenvideos wurden die beiden C#-Programme 'showAllVideos' und 'recordData' mit Visual Studio erstellt.

'showAllVideos'

Vor dem Aufnehmen einer Videosequenz, kann 'showAllVideos' gestartet werden. Dieses zeigt das Tiefen- und das Farbvideo von maximal zwei Kinect an. Damit kann die Ausrichtung der Sensoren überprüft werden.

'recordData'

Das Programm 'recordData' speichert die Tiefenbilder ab. Die Farbbilder werden als Grauwertbilder gespeichert. Es gibt keine Anzeige der Videos, damit möglichst wenig Bilder durch die grosse Auslastung der Computer verloren gehen. Das Programm 'recordData' ist für maximal vier Kinect gemacht, welche auf zwei Computer verteilt angeschlossen werden.

Aufbau

Wenn mehr als zwei Sensoren verwendet werden sollen, müssen diese auf zwei Computer verteilt angeschlossen werden. Beide müssen über eine Internet Verbindung verfügen, damit die Computerzeit mit einem NTP-Server synchronisiert werden kann. Die Computer werden mit einer einfachen RS232 Schnittstelle verbunden.

Über diese Verbindung werden die Aufnahmen zeitgleich gestartet. Weiter werden über die Schnittstelle die Computerzeiten synchron abgespeichert. Anhand dieser Timingdaten kann der NTP-Zeitabgleich verifiziert und zudem die Zeitdrift beider Computeruhren kontrolliert werden.

Speicherung

Alle Daten werden lokal auf dem jeweiligen PC in einem neu angelegten Verzeichnis auf das Laufwerk D gespeichert. Der Verzeichnisname besteht aus 'Videos "aktuelles Datum" "aktuelle Uhrzeit"'. Es werden die Unterverzeichnisse mit den Namen 'depthVideo' und 'grayVideo' erstellt. In 'depthVideo' werden alle Tiefenvideos gespeichert, in 'grayVideo' werde die Grauvideos gespeichert. Alle Zeitinformationen werden in das Textfile 'timestaps.txt' (respektive 'timestaps2.txt' für den zweiten PC) geloggt. Weitere Infos werden in 'info.txt' und 'info2.txt' gespeichert.

Matlab

Alle weiteren Analysen geschehen auf Matlab. Damit die Aufnahmen in Maltab ausgewertet werden können, müssen sich alle im gleichen Verzeichnis befinden. Das heisst es müssen die Aufnahmen, welche mit den zwei verschiedenen Computern gemacht wurden, zusammengeführt werden. Alle Tiefenvideos (beider Computer) müssen sich in im Unterverzeichnis 'depthVideo' befinden. Das selbe gilt natürlich auch für die Graustufenvideos. Auch 'timestaps.txt' und 'timestaps2.txt' müssen in das Verzeichnis kopiert werden. In Matlab kann der Verzeichnispfad jeweils am Anfang des Files angegeben werden (in den zwei wichtigsten Files 'main.m' und 'findSensorPositions.m', jeweils vor dem 'INIT' Aufruf).

10 Schlusswort

Schwierigkeiten

Bildverarbeitung ist im Allgemeinen eine sehr schwierige Aufgabe. In meiner Arbeit zeigte sich schnell, dass Bilder mit Tiefeninformation eine erhebliche Erleichterung sind. Das Problem, aus einem zweidimensionalen Bild auf etwas zu schliessen, was sich im dreidimensionalen Raum abspielt, entfällt. Es hat sich jedoch gezeigt, dass die Aufgabe trotzdem anspruchsvoll bleibt. Es liegen mit jedem Bild enorm viele Daten vor, die auf das Wesentliche reduziert werden müssen. Dabei können diese Daten fast beliebig verschiedene Formen annehmen. Und ist ein Algorithmus gefunden, ist dieser oft auf leichte Weise störbar. Es musste also ein robuster Algorithmus gefunden werden, der aus dem Daten-Urwald, aufgrund eines möglichst allgemein anwendbaren Verfahrens, die richtige Information herausfiltert. Der implementierte Algorithmus zeigt auf, dass mehrere Kameras eine Verbesserung bringen. Sobald nicht mehr genügend Perspektiven vorhanden sind, ist der Algorithmus nicht mehr robust. Natürlich könnte dieser durch einen Klassifizierer und ein Körpermodell verbessert werden, was aber nicht das primäre Ziel der Arbeit war. Während der Arbeit stellte sich zudem heraus, dass es wichtig ist, die Lage der Sensoren möglichst exakt zu kennen. Zunächst wurde versucht diese auszumessen. Jedoch resultieren aus kleinen Ungenauigkeiten im Winkel sehr grosse Fehler. Bereits eine Abweichung von einem Grad führt, bei einem in vier Metern Entfernung stehenden Objekt, zu einem Fehler von sieben Zentimetern. Es ist jedoch schwierig diese Genauigkeit von Hand zu erreichen. Dadurch entstand die Idee einen Algorithmus zu verwenden, welcher die Lage aller Sensoren aus Tiefendaten bestimmen kann. Die Umsetzung war jedoch schwieriger als erwartet, da sich ein nichtlineares Problem ergibt.

Ergebnisse

Ich wusste, dass es am erfolgsversprechensten ist, auf schon bekannte Verfahren zu setzen. Während der Recherche nach hilfreichen Algorithmen hat sich gezeigt, dass das Gebiet der Schätzung von Körperbewegungen aus Bildern sehr stark umforscht ist. Eindrücklich ist das zum Beispiel in [9] aufgezeigt. Dabei werden unzählige Verfahren vorgeschlagen. Ich entschloss auf dem in [2] beschriebenen AGEX-Algorithmus aufzubauen. Dieser schien geeignet, da er sehr allgemein anwendbar ist. Er ist weder auf gewisse Körperteile, noch auf den Mensch beschränkt und trotzdem liefert er ein starkes und zuverlässiges Feature. Es war für mich auch klar, dass die Zusammenführung der gefundenen Beobachtungen mit einem Kalman-Tracker sinnvoll ist. Da ich mich schon in einer vorangehenden Arbeit mit diesem beschäftigt habe, war der Einstieg leicht. Das primäre Ziel war einen Algorithmus für die Beine zu finden. Dieses Ziel wurde erreicht. Ein weiteres wichtiges Ergebnis meiner Arbeit ist ein Algorithmus, welcher die Lageparameter der Sensoren anhand von Markern im Tiefenbild findet.

Empfehlungen

Der für die Beine entwickelte Algorithmus kann ohne grösseren Entwicklungsaufwand auf den gesamten Körper ausgeweitet werden. Der AGEX Algorithmus und

auch der Kalman Tracker sind unabhängig vom Körperteil anwendbar. Es wäre jedoch Entwicklungsarbeit in einen guten Klassifizierer zu investieren. Zudem könnte ein Körpermodell den Algorithmus wesentlich verbessern. Interessant ist, dass der AGEX-Algorithmus auch für Echtzeitanwendungen geeignet ist. Dazu müsste der Dijkstra-Algorithmus möglichst effizient mit einer Vorrang-Warteschlange implementiert werden. Da immer nur eine oder zwei Kameras an einem Computer angeschlossen sind, könnte das finden der Körperteile auf den lokalen Rechnern ausgeführt werden. Die Messwerte müssten über Schnittstellen auf einen zentralen Computer gebracht werden. Dieser könnte mit dem Tracking-Verfahren die Daten kombinieren. Mit diesem Aufbau könnten praktisch unbegrenzt viele Kinect-Sensoren kombiniert werden. Eine weiterer Verbesserungsvorschlag betrifft die Lageparameter. Diese wären auch ohne Kalibrierung mit Markern direkt aus den Versuchsdaten ermittelbar. Dazu könnten die in den Tiefenbildern der einzelnen Kameras gefundenen Körperteile verwendet werden. Jedoch müsste beachtet werden, dass die Bilder nicht zur gleichen Zeit entstehen. Falls sich die Versuchsperson bewegt, befinden sich die Körperteile nicht mehr an der gleichen Stelle, was zu einer Verfälschung der Parameter führen würde.

Danksagung

Ich möchte mich herzlich bei Prof. Dr. Guido Schuster und Robert Hegner für deren wertvollen Tipps und Ratschläge bedanken. Weiter möchte ich mich bei meinem Studienkollegen Stefan Zollinger bedanken. Die Diskussionen mit ihm zu kniffligen Problemen führten oft zu mehr Klarheit.

11 Literaturverzeichnis

[1] al., Jamie Shotton et: *Real-Time Human Pose Recognition in Parts from Single Depth Images*. Computer Vision and Pattern Recognition (CVPR), Seiten 1297 – 1304, 2011.

[2] Christian Plagemann, Varun Ganapathi, Daphne Koller Sebastian Thrun: *Real-time Identification and Localization of Body Parts from Depth Images*. Robotics and Automation (ICRA), Seiten 3108 – 3113, 2010.

[3] Duda, Richard O.: *Pattern Classification*. Wiley-Interscience, 2001.

[4] Hanson, Andrew J.: *Visualizing Quaternions*. Elsevier, 2006.

[5] `http://kinectforwindows.org` Kinect for Windows, 04.01.2012.

[6] `http://www.microsoft.com/Presspass/press/2010/mar10/03-31PrimeSensePR.mspx` Microsoft News Center, 31.03.2010.

[7] Microsoft: *Programming Guide für Kinect Microsoft SDK*, 22. juli 2011.

[8] Pavlina Konstantinova, Alexander Udvarev, Tzvetan Semerdjiev: *A Study of a Target Tracking Algorithm Using Global Nearest Neighbor Approach*, 2003.

[9] Thomas B. Moeslund, Adrian Hilton, Volker Krüger: *A survey of advances in vision-based human motion capture and analysis*. Computer Vision and Image Understanding, 2006.

[10] Viager, Mikkel: *Analysis of Kinect for mobile robots*, 2011.

[11] Welch Greg, Bishop Gary: *An Introduction to the Kalman Filter*, 2006.

[12] `http://en.wikipedia.org/wiki/Structured-light_3D_scanner` Wikipedia, 17.11.2011.

[13] Zollinger, Stefan: *Attitude Representations of Strapdown Inertial Navigation Systems*. 2011.

12 Abbildungsverzeichnis

13 Akronyme und Definitionen

Akronyme

TOF Time Of Flight, Tiefenbildsensor, welcher mit der Lichtlaufzeit Distanzen misst.

NUI Natural User Interface, Schnittstelle, welche dem Nutzer natürliche Maschineninteraktionen durch Gesten oder Bewegungen ermöglicht.

MS Microsoft

mm Millimeter

m Meter

SDK Software Development Kit

RGB Rot, Grün und Blau sind die Grundfarben des RGB-Farbraums. Es sind die 3 Grundfarben die das menschliche Auge wahrnehmen kann. RGB-Kamera ist ein Synonym zu Farbkamera.

AGEX Accumulative Geodesic EXtrema, Algorithmus zum auffinden von Extremalstellen auf einer Oberfläche [2].

GUI Ein GUI, Graphical User Interface, ist eine grafische Benutzeroberfläche die dem Benutzer eines Computers die Interaktion mit der Maschine erlaubt.

Definitionen

Tiefenbild Wird in dieser Dokumentation synonym zu Tiefendaten, Tiefenbilddaten und Tiefensensorbild verwendet. Bild, in welchem die einzelnen Pixel keine Farbwerte oder Grauwerte darstellen, sondern Distanzwerte.

Tiefensensor Synonym wird auch Tiefenbildsensor verwendet. Sensor, welcher viele Distanzen zu einer definierten Ebene misst. Diese Distanzen können in einem zweidimensionalen Bild angeordnet werden.

Tiefenvideo Viele Tiefenbilder als Video abgespielt.

Schlüsselpositionen Vom AGEX-Algorithmus gefundene Positionen auf einer Oberfläche. In [2] werden diese Positionen als 'Interest Points' bezeichnet.

14 Beiliegender Datenträger

Beiliegender Datenträger

Der beiliegende Datenträger enthält folgende Verzeichnisse:

Doku	Dieses Dokument als pdf und LaTeX Quelltext
Code	Der C# Quellcode, unter anderem, zum Aufnehmen einer Sequenz mit den Kinect-Sensoren
Matlab	Algorithmen zur Interpretation der Tiefendaten
Literatur	Quellen als pdf (sofern verfügbar)
Unterlagen	Software- und Hardwareunterlagen
Daten	Datenbank aller verwendeten Tiefen- und Graustufenvideos

15 Zusammenstellung der wichtigsten Files

Hier sind die wichtigsten Code-Files, nach Relevanz sortiert, zusammengestellt.

Matlab-Code

main	Hauptalgorithmus
findSensorPositions	Findet die kugelförmigen Marker in den Tiefenbildern aller Sensoren und berechnet daraus deren relativen Winkel und Positionen zu einander.
INIT	Hier werden die Zeitstempel, Video-Streams und Hintergründe geladen oder erzeugt und wichtige Konstanten gesetzt und in eine Struktur abgelegt.
showResultsFinda	Auswertung des Versuches zur Koordinaten Berechnung, beschrieben im Kapitel 2.3.
findDepthResolution	Ermittelt alle Quantisierungsschritte eines Tiefenvideos (siehe Kapitel 2.4).

C#-Code

recordData	Aufnehmen und speichern einer Videosequenz
showAllVideos	Zeigt die Farb- und Tiefenvideos von maximal zwei Kinect-Sensoren an.

16 Erklärung zur Urheberschaft

Ich erkläre hiermit, Erklärung

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt zitiert habe.

Rapperswil, 2. Februar 2012 Ort, Datum

Mathias Hunold Unterschrift

A Technische Details Kinect-Sensor

Allgemein

Zum Kinect-Sensor sind manche technische Details nur sehr schwierig in Erfahrung zu bringen. Einige Angaben konnten aus [7] und [10] übernommen werden, andere mussten anhand eigener Versuche ermittelt werden. Alle Angaben basieren auf Daten, welche mit dem original Microsoft Kinect Hardware-Treiber ermittelt wurden. Hier folgt eine Zusammenstellung der für diese Arbeit wichtigsten technischen Daten.

Kinect

Technische Details:

- Mechanisch schwenkbar zwischen $-28°$ und $+28°$
- Enthält einen Tiefenbildsensor, eine Farbkamera und ein Mikrofonarray

Tiefensensor

Technische Details:

- Einheit der Rohdaten: In Millimeter
- Minimale erfassbare Sensordistanz: 801 mm (Kapitel 2.4)
- Maximale erfassbare Sensordistanz: 3930 mm (Kapitel 2.4)
- Ungültige Messungen haben den Wert 0
- 12 Bit (4096 theoretisch mögliche Werte)
- Anzahl Quantisierungsstufen: 345 (Kapitel 2.4)
- Quantisierung der Sensordaten bei 801 mm Abstand: 2 mm
- Quantisierung der Sensordaten bei 3930 mm Abstand: 45 mm
- Maximale Auflösung: 640x480 Pixel
- Weitere verfügbare Auflösungen: 320x240, 80x60
- Streckungskonstante: $a = 1.7267 \frac{1}{px}$ (siehe Versuch Kapitel 2.3)

Farbkamera

Technische Details:

- Maximale Auflösung: 640x480 Pixel
- Weitere verfügbare Auflösungen: 320x240, 80x60
- Sichtwinkel: 43° vertikal, 57° Grad horizontal

B Anleitung zum machen eigener Aufnahmen

Hier wird stichwortartig beschrieben, wie eine Aufnahme gemacht werden kann, damit sie für den Hauptalgorithmus ('main.m') verwendbar ist. Siehe dazu auch das Kapitel 9.3.2. Dazu dienen die beiden C#-Programme 'showAllVideos' und 'recordData'. Die Programme wurden mit Microsoft Visual Studio erstellt. Sie können mit diesem oder mit dem gratis erhältlichen Microsoft Visual C#-Express gestartet werden.

1. Kinect über USB anschliessen
2. Pro Kinect braucht es einen USB Controller.
3. pro Computer maximal zwei Kinect
4. maximal zwei Computer → maximal vier Kinect
5. 'showAllVideos' starten und Kinect wie gewünscht ausrichten
6. alle Computer brauchen Internet Anbindung für NTP Zeitsynchronisation
7. mit RS232-Kabel Computer verbinden
8. Auf beiden Computern 'Control Panel' → 'System and Security' → 'Administrative Tools' → 'Services' → 'Windows Time' auf disabled stellen damit NTP time update funktioniert.
9. 'recordData' auf dem ersten Computer starten
10. Anzahl Computer angeben (1 oder 2)
11. Aufforderung "Pleas switch on x Computer" mit 'OK' bestätigen → PC wartet, falls zwei Computer angegeben wurden
12. falls zwei Computer angegeben wurden, 'recordData' auf dem zweiten PC starten. Aufnahme beginnt
13. Zum Aufnahme beenden: "Stop Recording?" mit 'OK' bestätigen (auf jedem PC)
14. Aufnahmen von verschiedenen PC in ein gemeinsames Verzeichnis kopieren, damit sie auf Matlab weiterverwendet werden können.

C Verwendung Kinect mit Visual Studio

C.1 Verwendung Kinect

- Je Kinect Sensor ein eigener USB-Controller (reserviert 60% der Kapazität)
- Nach dem Anschliessen des Kinect-Sensors sollte dessen Grüne LED zu blinken beginnen
- Im Device Manager ist jeder Sensor als 'Microsoft Kinect' ersichtlich
- Treiber für Microsoft Windows 7 verfügbar unter [5]
- Zugriff auf die Daten aus C++ oder C# Programm

Siehe dazu auch Kapitel 2.2.

C.2 Eigenes Projekt erstellen

Diese Kurzanleitung zeigt, wie mit Microsoft Visual Studio oder oder Microsoft Visual C#-Express ein eigenes C#-Projekt gemacht werden kann. (WPF Application)

1. 'file' → 'new' → 'project...'
2. 'WPF Application' anwählen und 'Name' und 'Location' bestimmen
3. Rechtsklick auf 'References' → 'add References...' → '.NET' → 'Microsoft.Research.Kinect' Doppelklick
4. 'Coding4Fun.Kinect.Wpf' aus Verzeichnis B_code kopieren und in das Projektverzeichnis kopieren (Coding4Fun ist hilfreich aber nicht notwendig).
5. Rechtsklick auf 'References' → 'add References...' → 'Browse' → 'Coding4Fun.Kinect.Wpf' Doppelklick

Wie ein Programm erstellt und auf die Daten einer oder mehrerer Kinect Sensoren zugegriffen wird ist im Anhang D, Code Beispiele, ersichtlich.

D Code Beispiele

Alle gezeigten Code-Beispiele sind auch auf der CD unter 'B_Code' → 'sampel code' als Visual Studio Projekte vorhanden.

D.1 Minimalbeispiel, Daten eines Sensors visualisieren

XAML-Code:

```
<Window x:Class="showAllVideos.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        WindowState="Maximized"
        Title="Color and depth data"
        Height="1191" Width="1647"
        Loaded="Window_Loaded"
        Closed="Window_Closed">
    <Grid Background="black" Width="1800">
        <Image Height="480"
               HorizontalAlignment="Left"
               Margin="688,48,0,0"
               Name="image"
               Stretch="Fill"
               VerticalAlignment="Top"
               Width="640" />

        <Image Height="480"
               HorizontalAlignment="Left"
               Margin="42,48,0,0"
               Name="depth"
               Stretch="Fill"
               VerticalAlignment="Top"
               Width="640" />
    </Grid>
</Window>
```

```
using System;
using System.Collections.Generic;
using System.Windows;
using Microsoft.Research.Kinect.Nui;
using Coding4Fun.Kinect.Wpf;

namespace showAllVideos
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        Runtime nui;

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            nui = new Runtime();

            // initialice all kinect devices
            nui.Initialize(RuntimeOptions.UseColor | RuntimeOptions.UseDepth);

            // Add event to handler
            nui.DepthFrameReady +=
                new EventHandler<ImageFrameReadyEventArgs>(nui_DepthFrameReady);
            nui.VideoFrameReady +=
                new EventHandler<ImageFrameReadyEventArgs>(nui_VideoFrameReady);

            // Elevate angle
            nui.NuiCamera.ElevationAngle = 20;

            nui.VideoStream.Open(ImageStreamType.Video, 2,
                ImageResolution.Resolution640x480, ImageType.Color);
            nui.DepthStream.Open(ImageStreamType.Depth, 2,
                ImageResolution.Resolution640x480, ImageType.Depth);
        }

        // Wird beim schliessen des Fensters aufgerufen. Beendet das Programm.
        private void Window_Closed(object sender, EventArgs e)
        {
            nui.Uninitialize();
            Environment.Exit(0);
        }

        // Wird aufgerufen wenn ein neues Tiefenbild zur Verfügung steht.
        void nui_DepthFrameReady(object sender, ImageFrameReadyEventArgs e)
        {
            depth.Source = e.ImageFrame.ToBitmapSource();
        }
```

```
        // Wird aufgerufen wenn ein neues Farbbild zur Verfügung steht.
        void nui_VideoFrameReady(object sender, ImageFrameReadyEventArgs e)
        {
            image.Source = e.ImageFrame.ToBitmapSource();
        }
    }
}
```

D.2 Minimalbeispiel Datenzugriff

Hier wird nur die Funktion 'nui_DepthFrameReady' gezeigt. Alles andere ist gleich wie im Anhang D.1, und kann dort entnommen werden.

```
void nui_DepthFrameReady(object sender, ImageFrameReadyEventArgs e)
{
    uint height = (uint)e.ImageFrame.Image.Height;
    uint width = (uint)e.ImageFrame.Image.Width;
    Byte[] depthData = e.ImageFrame.Image.Bits;
    uint distance;

    uint i = 0;
    for (uint y = 0; y < height; y++)
    {
        for (uint x = 0; x < width; x++)
        {
            // berechne pixel distanz in mm:
            // falls RuntimeOptions.UseDepthAndPlayerIndex, dann:
            // (uint)(depthData[i] >> 3 | depthData[i + 1] << 5)
            distance = (uint)(depthData[i] | depthData[i + 1] << 8);

            // Auf Pixel kan hier zugegrifen werden
            if(y > height/2)
            {
                distance = distance/2;
            }

            // Wieder zurückschreiben für die Anzeige
            depthData[i] = (Byte)distance;
            depthData[i + 1] = (Byte)(distance >> 8);

            i += 2;
        }
    }
    e.ImageFrame.Image.Bits = depthData;

    // Bild anzeigen
    depth.Source = e.ImageFrame.ToBitmapSource();
}
```

D.3 Minimalbeispiel mehrere Sensoren

Es sind nur wenige Änderungen nötig:

1. Anzahl Kinect bestimmen:

```
numOfCams = (new Device()).Count;
```

2. Runtimes können beispielweise in einer Liste gespeichert werden:

```
// List of Runtime objects
nui = new List<Runtime>();

// Add Runtimes to List
for (int i = 0; i < numOfKinect; i++)
    nui.Add(new Runtime(i));
```

3. Mit 'foreach' können nun alle Initialisierungen, etc. ausgeführt werden.
4. In den Funktionen können die verschiedenen Instanzen folgendermassen unterschieden werden:

```
int camIndex = ((Runtime)sender).InstanceIndex;
```