Multimodale Bedienung eines Roboters

Created by
Lars Engel
lars@l-engel.de
7. April 2016

M.Sc. Information Technology
Faculty of Computer Science and Electrical Engineering
Kiel University of Applied Sciences

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Motivation . . . 3
1.2 Aufbau des Berichts . . . 5

2 Hintergrund **6**
2.1 KUKA LBR iiwaa 7 R800 . . . 6
2.2 Microsoft Kinect . . . 6
2.2.1 Infrarotsensor . . . 7
2.2.2 Tiefenkamera . . . 7
2.2.3 Farbkamera . . . 7
2.2.4 4-Mikrofon-Array . . . 8
2.3 Aufbau . . . 9

3 Implementierung **10**
3.1 Struktur der Anwendung . . . 10
3.2 Kommunikation zwischen Kamera und Roboter . . . 10
3.2.1 Bidirektionale Kommunikation . . . 12
3.2.2 Übertragung von Gleitkommazahlen . . . 12
3.3 Hand-Augen-Kalibirierung . . . 14
3.3.1 Unterschiede der Koordinatensysteme . . . 14
3.3.2 Kalibrierung und Transformation der Koordinatensysteme 14
3.4 Microsoft Kinect . . . 19
3.4.1 Wahl der Softwareschnittstelle . . . 19
3.4.2 Frames . . . 20
3.4.3 Personenerkennung mit Hilfe des Body Frames . . . 20
3.4.4 Spracherkennung . . . 23
3.5 Programmierung des Roboters . . . 25
3.5.1 Roboteranwendung . . . 25
3.5.2 Klassenbeschreibung . . . 25
3.5.3 Sicherheitskonfiguration . . . 26
3.6 Projektdurchführung . . . 26

4 Schluss **27**
4.1 Ergebnisse 27
4.2 Ausblick 29

Kapitel 1

Einleitung

In diesem Kapitel wird die Motivation für das Projekt genannt und es wird der Aufbau des Berichts beschrieben.

1.1 Motivation

Waren Roboter zu Beginn ihrer Entwicklung dazu ausgelegt den Menschen bei mühsamen, sich wiederholenden Aufgaben zu ersetzen, geht der Trend in den letzten Jahren mehr und mehr in Richtung einer engen Interaktion zwischen Mensch und Roboter[4]. Menschen interagieren mit Robotern, um diesen Befehle oder Anweisungen zu erteilen, oder um Aufgabenabläufe festzulegen.
Dabei passt sich der Mensch meist den Interaktionsmöglichkeiten an, die ihm vom Roboter angeboten werden, um diesem explizite Anweisungen und Befehle, meist über eine Tastatur oder ein Smartpad, zu geben. Die Art, wie die Interaktion zwischen Mensch und Maschine funktioniert, ist dabei abhängig von dem verwendeten Roboter und wie die Schnittstelle umgesetzt wurde. Dadurch, dass die Kommunikation mit einem Roboter sehr unterschiedlich sein kann, müssen die entsprechenden Befehle von jedem Anwender erst gelernt werden, bevor eine effektive Verwendung des Roboters möglich ist[8]. Zusätzlich ist die Bedienung eines Roboters durch Eingabe von Befehlen über eine Tastatur oder ein Smartpad meist aufwendig und kostet viel Zeit. Oft müssen Koordinaten für Positionen per Tastatur eingegeben werden oder vor Start der Anwendung eingeteached werden [2]. Muss ein Roboter mit Hilfe eines Smartpads bedient werden, hat der Operator außerdem die Hände nicht mehr frei für eventuelle weitere Operationen oder Interaktionen.

Da in der zwischenmenschlichen Kommunikation nicht nur Sprache, sondern auch Gestik und Mimik benutzt wird, erwartet der Mensch von einer intuitiven Interaktion mit einem Roboter, dass auch hier verschiedene Kommunikationswege benutzt werden können. Ein Roboter sollte also auf Sprache reagieren, aber auch auf die Gesten achten oder die Mimik eines Gegenübers erkennen. Diese Art der Kommunikation über verschiedene Kommunikationswege wird als

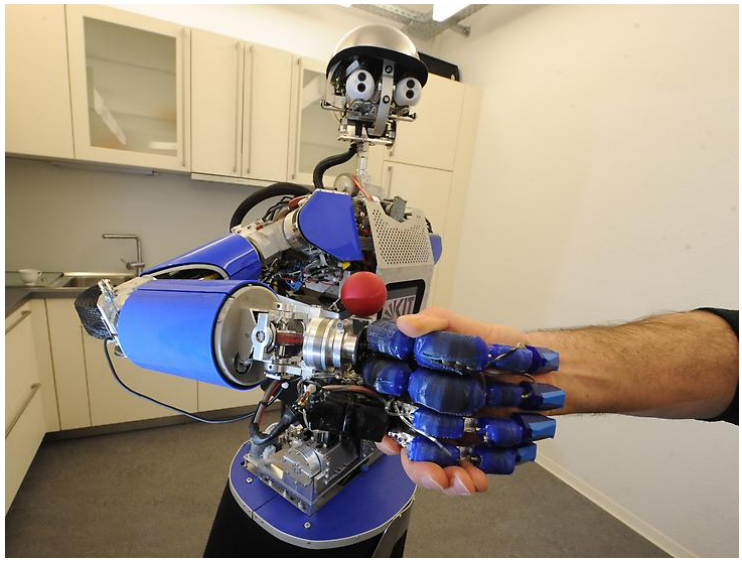


Abbildung 1.1: ARMAR 3, ein humanoider Roboter entwickelt am Karlsruher Institut für Technologie [http://bilder3.n-tv.de/img/incoming/origs10286266/6098257657-w778-h550/29756021.jpg]

multimodale Kommunikation bezeichnet.
Besonders bei humanoiden Robotern wie dem ARMAR[11], der vom Karlsruher Institut für Technologie entwickelt wird oder dem ASIMO[9], entwickelt von Honda, ist es wichtig dass die Interaktion zwischen Mensch und Roboter so intuitiv wie möglich erfolgt, da diese Art von Roboter genau für die Interaktion mit Menschen ausgelegt sind, um zum Beispiel ein Ambient Assisent Living zu ermöglichen[6]. Aber auch bei Industrierobotern, deren Hauptaufgabe das Handhaben von Werkstücken in einem industriellen Umfeld (z.B. Automobilfertigung) ist, kann eine multimodale Steuerung von Vorteil sein, um Aufgaben effizient erledigen zu können

Die oben genannten Beispiele [11] und [9] beschäftigen sich mit der Entwicklung von Robotern, deren Hauptaufgabe die Kommunikation zwischen Mensch und Roboter ist. Es handelt sich dabei um Maschinen, die extra für diesen Einsatz gebaut worden sind. Diese humanoiden Roboter sind durch ihre Bauweise nicht dazu geeignet industrielle Aufgaben zu erledigen. Die Sensoren, die der Roboter zur Erkennung seiner Umgebung benötigt (z.B. Kamera und Mikrofon) sind fest verbaut und speziell für den Roboter hergestellt worden.
In diesem Projekt wird die Frage gestellt, ob es möglich ist eine multimodale Roboteranwendung mit einem industriellen Roboter wie dem KUKA LBR iiwaa 7 R800 zu entwickeln. Als Sensoren zur Erkennung von Bild und Ton soll dem Roboter dabei eine handelsübliche Microsoft Kinect 3D - Kamera dienen. Es

soll geprüft werden, welche Funktionalitäten auf diese Weise umgesetzt werden können. Zudem soll die Anwendung flexibel einsetzbar sein und keinen großen Kalibrierungsaufwand erfordern, da der verwendete Roboter durch seine Bauweise leicht an unterschiedlichen Orten eingesetzt werden kann (mehr dazu in Abschnitt 2.1).

1.2 Aufbau des Berichts

Der Rest dieses Berichts ist wie folgt strukturiert: In Kapitel zwei werden Hintergrundinformationen über die benutzte Hardware beschrieben, in Kapitel drei wird die Implementierung beschrieben und abschließend wird in Kapitel 4 ein Fazit gezogen und ein Ausblick gegeben.

Kapitel 2

Hintergrund

In diesem Kapitel wird der Hintergrund des Projekts beschrieben. Es werden die Komponenten und der Aufbau des Projekts erläutert.

2.1 KUKA LBR iiwaa 7 R800

Es wird ein Roboter des Modells KUKA LBR iiwaa 7 R800[12] benutzt. Dabei handelt es sich um einen 7-achsigen Gelenkarmroboter mit nur 23,9 kg Gewicht und einer Traglast von 7 kg. Der Roboter kann für intellligente industrielle Arbeiten benutzt werden, die ein hohes Maß an Beweglichkeit erfordern. Da der Roboter sieben Achsen besitzt, ist er kinematisch redundant, was bedeutet, dass ein Punkt im Arbeitsraum theoretisch von unendlich vielen Achsenkonfigurationen angefahren werden könnte. Dadurch kann der Roboter auch an schwer zugänglichen Orten eingesetzt werden. Welche Achsenkonfiguration gewählt werden soll, um einen Punkt anzufahren, kann programmatisch festgelegt werden.
Der Roboter wird mit Hilfe der Entwicklungsumgebung KUKA Sunrise Workbench programmiert und benutzt die Programmiersprache Java. Die Software KUKA Sunrise Workbench kann zudem zum Debuggen eines Programmes verwendet werden und ermöglicht es weitere Funktionen des Roboters, wie zum Beispiel die Sicherheitskonfiguration, einzustellen.

2.2 Microsoft Kinect

Eine Microsoft Kinect V2[13] dient dem Roboter als Augen und Ohren. Die Kinect wird mit Hilfe des Kinect for Windows SDK 2.0 programmiert. Als Programmiersprache wird C# und als Entwicklungsumgebung Microsoft Visual Studio 2013 benutzt.
Die Kinect besteht aus einem Infrarotsensor, einer Tiefenkamera, einer Farbkamera und einem 4-Mikrofon-Array. Dadurch kann sie sowohl zur Bilderkennung sowie zur Tonerkennung verwendet werden.

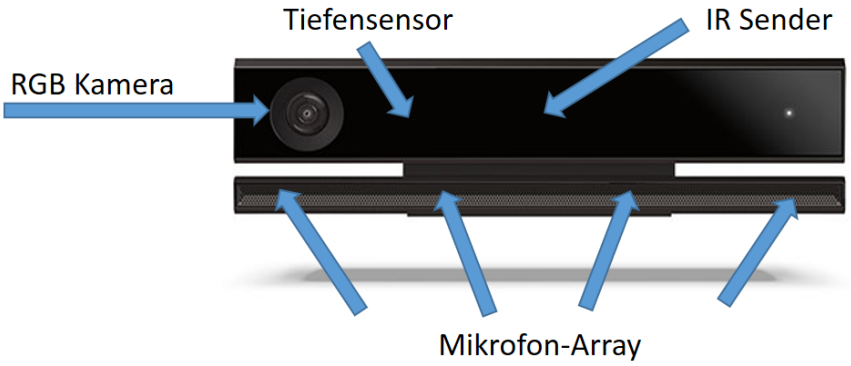


Abbildung 2.1: Aufbau der Microsoft Kinect V2

2.2.1 Infrarotsensor

Der Infrarotsensor hat eine Auflösung von 512 x 424 und arbeitet mit 30 Hz. Er dient zur Erkennung der Umgebung bei schwacher oder sogar gar keiner Beleuchtung. Dies ermöglicht den Betrieb der Kinect auch an Arbeitsplätzen, die nicht ausreichend beleuchtet sind für eine RGB Kamera, oder die unregelmäßige Lichtverhältnisse haben, die zum Beispiel durch direkte Sonneneinstrahlung auftreten könnten.

2.2.2 Tiefenkamera

Die Infrarotkamera ist auch ein Teil der Tiefenkamera der Kinect. Zur Erkennung der Tiefeninformation wird die *Time-of-Flight* Technologie benutzt.
Die Beleuchtungseinheit sendet einen Infrarot-Puls ab. Die Infrarot-Strahlen werden von der Umgebung reflektiert und von der Optik der Kinect wieder aufgefangen. Der Sensor der Tiefenkamera misst für jeden der 512 x 424 Pixel einzeln, wie viel Zeit verstrichen ist, bis ein gesendeter Puls wieder angekommen ist. Da bekannt ist, wie die Geschwindigkeit des ausgestrahlten Puls ist, kann nun gemessen werden, wie weit jeder Pixel von der Kinect entfernt ist[5]. (Siehe Abbildung 2.2)

2.2.3 Farbkamera

Die Farbkamera, die mit 1920 x 1080 Pixel bei 30 Bildern pro Sekunde filmt, kann zur Aufzeichnung von Videos bzw. Bildern und zur sofortigen Bildverarbeitung verwendet werden.

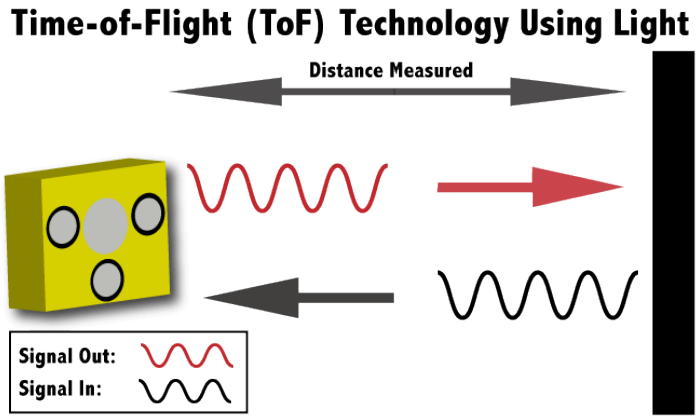


Abbildung 2.2: Prinzip von Time-of-Flight [`http://www.teraranger.com/wp-content/uploads/2015/04/principle.png`]

2.2.4 4-Mikrofon-Array

Das Mikrofon der Kinect kann zur Aufzeichnung und Auswertung der Geräusche der Umgebung verwendet werden. Mit dem Microsoft Speech Platform SDK bietet Microsoft ein Framework zur Spracherkennung an, das auf die Hardware der Kinect abgestimmt ist.
Die Kinect enthält ein Mikrofon-Array, das aus 4 nebeneinander gelegenen Mikrofonen besteht. Dadurch kann die Position eines Geräusches ermittelt werden (*Beamforming*). Dabei wird die Zeitdifferenz gemessen, mit der ein Signal bei den unterschiedlichen Mikrofonen ankommt[18]. (Siehe Abbildung 2.3)

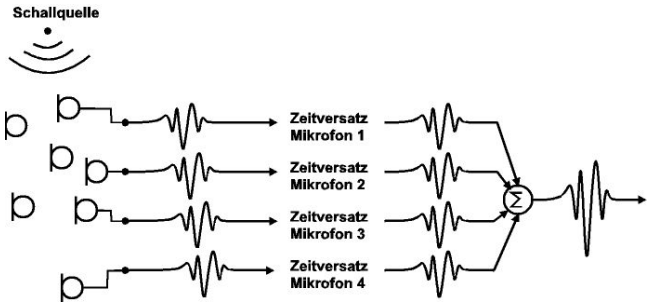


Abbildung 2.3: Prinzip von Beamforming [`https://upload.wikimedia.org/wikipedia/de/6/61/Funktion_mikrofonarray.jpg`]

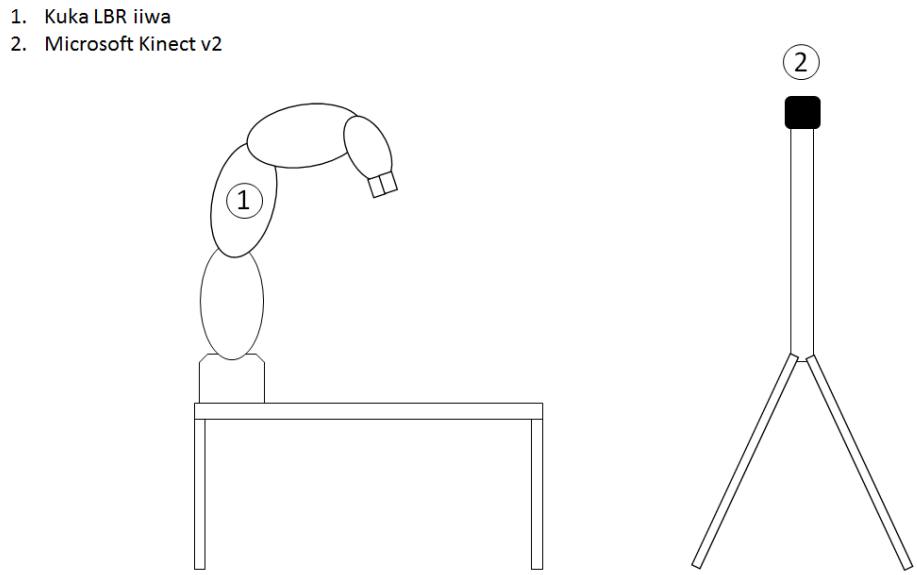


Abbildung 2.4: Aufbau des Projekts

2.3 Aufbau

Zur Durchführung des Projekts werden drei Komponenten benötigt:

1. KUKA LBR iiwaa 7 R800
2. Microsoft Kinect v2 und einen Kinect Adapter for Windows
3. PC oder Laptop mit Microsoft Windows 8, 8.1 oder 10, einem LAN Kabel Anschluss und einem USB 3.0 Eingang

Der Roboter ist auf einem Tisch montiert und per LAN Kabel an einen Switch angeschlossen.
Die Kinect ist auf einem verstellbaren Stativ montiert und über den Kinect Adapter for Windows an den USB 3.0 Eingang des Computers verbunden.
Der Computer ist per LAN Kabel an den Switch angebunden und erhält eine statische IP Adresse.

Kapitel 3

Implementierung

In diesem Kapitel wird die Implementierung der Kinectanwendung und der Roboteranwendung beschrieben.

3.1 Struktur der Anwendung

Die Implementierung ist in zwei Komponenten unterteilt.

1. Die Kinectanwendung
   Die Kinectanwendung steuert die Microsoft Kinect und ist zuständig für die Bild- und Spracherkennung. Die Anwendung läuft auf dem PC mit Windows und basiert auf dem .NET - Framework, welches mit C# programmiert wird.

2. Die Roboteranwendung
   Die Roboteranwendung steuert die Bewegungen des Roboters. Sie läuft direkt auf der Rechnereinheit des Roboters und wurde mit der KUKA Sunrise Workbench entwickelt, welche Java als Programmiersprache benutzt.

3.2 Kommunikation zwischen Kamera und Roboter

Die Kommunikation zwischen Kamera und Roboter wird über das Modbus Protokoll per TCP / IP geregelt. Dabei ist der Roboter ein Modbus Master (TCP Client) und die Kamera ein Modbus Slave (TCP Server).
Der Modbus Master schickt einen ReadMultipleRegistersRequest ab und gibt dabei das Startregister zum Lesen und die Anzahl zu lesende Register an. Der Modbus Slave horcht auf kommende Anfragen, wertet diese aus und schickt eine entsprechende Antwort.

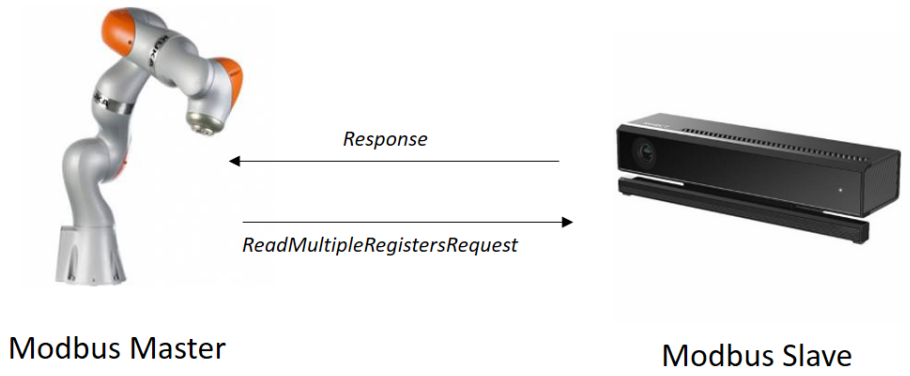


Abbildung 3.1: Aufbau der Kommunikation

Über das vom Modbus Master übermittelte Startregister kann der Modbus Slave, erkennen, welche Informationen der Modbus Master erwartet. So wurde zum Beispiel das Register 30010 verwendet, um die aktuelle Position des Fingers des menschlichen Akteurs zu übermitteln (siehe Listing 3.1).

Listing 3.1: Auszug aus der Modbus Slave Methode zur Evaluierung der Modbus Master Anfrage anhand des Startregisters

```
switch(startRegister) {
        ...
        case 30000:
                getAndReturnIfAnyActionToMaster();
                break;
        case 30010:
                getAndReturnCurrentFingerPositionToMaster();
                break;
        ...
}
```

In der Roboter Applikation wurde das Java Paket jamod[10] verwendet und in der Kinect Applikation NModbus[14].

3.2.1 Bidirektionale Kommunikation

Ein Modbus Slave kann nur Nachrichten (Response) an einen Modbus Master schicken, wenn zuvor der Modbus Master eine Anfrage (Request) an den Modbus Slave geschickt hat. Dies ist allerdings in diesem Projekt ein Problem, da die Microsoft Kinect, der Modbus Slave, genutzt wird, um den Roboter, den Modbus Master, per Sprache und Gesten zu steuern. Das bedeutet, dass die Kinect Informationen an den Roboter schicken können muss, ohne dass der Roboter diese Informationen erwartet oder anfordert. Um dies zu realisieren wird in der Roboter Applikation ein Thread erstellt, der alle 100 Millisekunden eine generische Anfrage an das Register 30000 an die Kinect schickt (Polling). Innerhalb der Kinect Applikation wird ermittelt, ob eine Aktion per Sprache oder Geste an den Roboter geschickt werden soll. Die Kinect übermittelt dem Roboter anschließend einen *action_code*, der die gewünschte Aktion beschreibt.

Der *action_code* kann dabei folgende Werte haben:

action_code	**Bedeutung**
0	keine Aktion
2	Bewege den Greifer des Roboters auf der übermittelten Achse um die übermittelte Millimeter Zahl
3	Rotiere den Greifer des Roboters auf der übermittelten Achse um die übermittelte Grad Zahl
10	Stoppe die Roboter Bewegung
21	Öffne den Greifer des Roboters
20	Schließe den Greifer des Roboters

3.2.2 Übertragung von Gleitkommazahlen

Das Register, im dem die Daten zwischen Master und Slave übermittelt werden, kann nur ganze positive 16 Bit Zahlen (Wertebereich 0 - 65.535) enthalten (ushort). Da allerdings die Koordinaten der Kinect Gleitkommazahlen sind, die auch negativ sein können (in Meter angegeben), musste eine Funktion zur Konvertierung der Zahlen geschrieben werden.
Der Algorithmus des Slaves evaluiert das Vorzeichen eines Wertes und übergibt dieses in einem extra Register an den Master (0 = negativ, 1 = positiv). Die zu übermittelnde Zahl wird als Betrag mit 1.000 multipliziert und die übrigen Nachkommastellen werden abgeschnitten. Nach Übertragung zu dem Master wird in einer Konvertierungsmethode das Vorzeichen ausgewertet und auf die Zahl entsprechend zugerechnet. Die Multiplikation mit 1.000 muss nicht rückgängig gemacht werden, da das Weltkoordinatensystem des Roboters in Millimeter ist, während das Koordinatensystem der Kinect in Metern ist. Somit ist die Skalierung der Koordinatentransformation schon durchgeführt. Auf die Koordinatentransformation wird in Kapitel 3.3.2 näher eingegangen. Der Ablauf der Konvertierung ist in Abbildung 3.2 dargestellt.
Durch die gewählte Berechnung ergibt sie für die Anwendung eine mathematische Genauigkeit im Millimeter Bereich. Die höchste Zahl, die mit diesem Al-

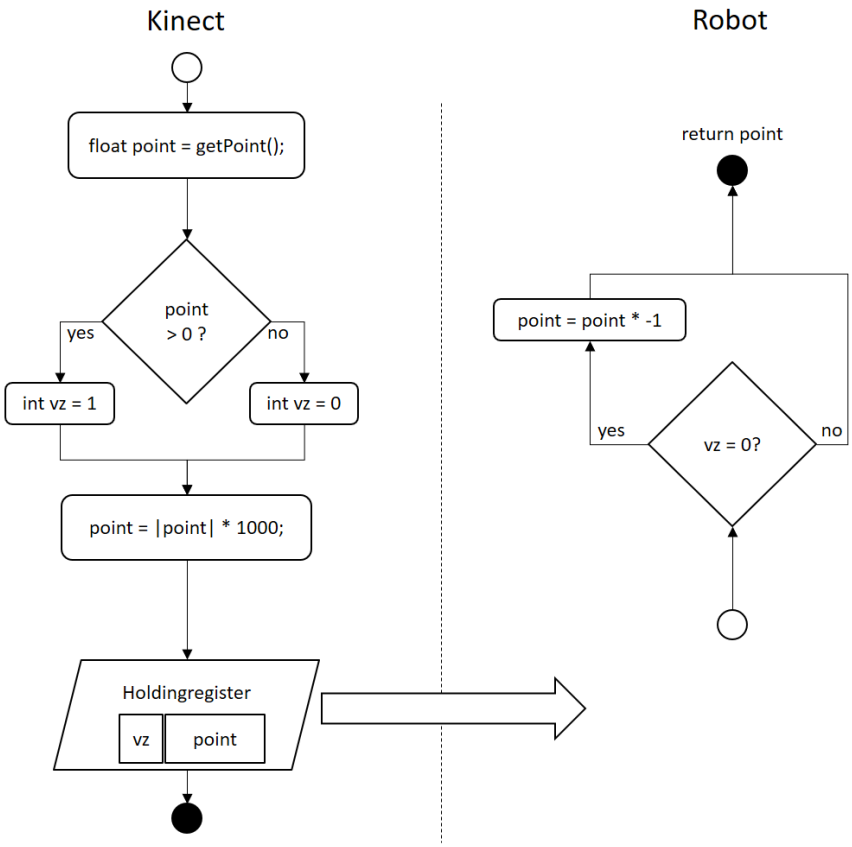


Abbildung 3.2: Konvertierung und Übertragung von Gleitkommazahlen

gorithmus übermittelt werden kann, ist 65,535, da sonst der Wertebereich von ushort überschritten werden würde. Da der Arbeitsbereich der Kinect allerdings nur 8 Meter beträgt ist, stellt dies kein Problem dar.

Abbildung 3.3: Vergleich der Koordinatensysteme von Roboter und Kinect

3.3 Hand-Augen-Kalibirierung

3.3.1 Unterschiede der Koordinatensysteme

Das Koordinatensystem der Tiefenkamera der Kinect ist in Meter angegeben, der Ursprung befindet sich im Zentrum des Infrarotsensors, die x-Achse wächst entlang des Sensors in linke Richtung, die y-Achse aufwärts des Sensors, wobei die Rotation der Kinect beachtet werden muss und die z-Achse wächst in Blickrichtung des Sensors (siehe Abbildung 3.3). Das Weltkoodinatensystem des Roboters ist in Milimetern angegeben. Der Ursprung liegt im Roboterfuß, die x-Achse wächst in Blickrichtung des Roboters, die y-Achse wächst zur Linken des Roboters und die z-Achse wächst aufwärts des Roboters (siehe Abbildung 3.3). Dies bedeutet also, dass sich die Achsenbeschriftungen der Koordinatensysteme unterscheiden. Um dies Problem zu umgehen werden die Beschriftungen der Koordinaten, die die Kinect an den Roboter schickt umgewandelt, sodass aus der z-Achse der Kinect die x-Achse des Roboters, aus der x-Achse der Kinect die y-Achse des Roboters und aus der y-Achse der Kinect die z-Achse des Roboters wird. Im weiteren Verlauf dieses Berichts werden die Achsen nur noch mit den Achsenbeschriftungen des Weltkoordinatensystems des Roboters bezeichnet.

3.3.2 Kalibrierung und Transformation der Koordinatensysteme

Die Kinect und der Roboter haben jeweils ein eigenes Koordinatensystem (bzw. im Fall des Roboters mehrere Koordinatensysteme), die den Arbeitsraum definieren. Damit es möglich ist Positionsdaten, die die Kinect ermittelt innerhalb der Roboterapplikation verwenden zu können, müssen die Koordinatensysteme

kalibriert werden. Das bedeutet, dass die Skalierung (Unterschiede der Größe der Koordinatensysteme), die Translation (Verschiebung der Koordinatensysteme) und die Rotation der Koordinatensysteme zueinander ermittelt werden müssen. Zur Vereinfachung wird in diesem Projekt davon ausgegangen, dass die Rotation sich auf die z-Achse beschränkt und somit die x-, und y-Achse nicht rotieren. Das bedeutet, dass die Kinect auf ihrer x- und y-Achse waagerecht bzw. senkrecht ausgerichtet werden muss. Da das verwendete Stativ eine Wasserwaage enthält, ist diese Ausrichtung allerdings einfach zu ermöglichen.

Synchronisationsprozess

Zu Beginn der Roboterapplikation wird der Benutzer dazu aufgefordert eine Synchronisation der Koordinatensysteme von Roboter und Kinect durchzuführen. In diesem Prozess werden die Rotation und Translation der Kinect Kamera zum Roboterfuß, wo das Weltkoordinatensystem des Roboters seinen Ursprung hat, ermittelt. Um diese Parameter zu finden, werden Berechnungen an Dreiecken durchgeführt. Dazu müssen die Koordinaten von festgelegten Punkten des Koordinatensystem des Roboters im Koordinatensystem der Kinect gefunden werden.
Im ersten Schritt legt der Nutzer seine Hand an den Roboter Fuß. Im Weltkoordinatensystem des Roboters handelt es sich dabei um den Punkt (0, 0, 0). Nun liest die Kinect die Position der Hand ein und erhält somit die Koordinaten des Ursprungs des Koordinatensystems des Roboters im eigenen Koordinatensystem. Anschließend fährt der Roboter einen Punkt auf der eigenen y-Achse an zum Beispiel (0, -300, 0). Der Nutzer legt nun seine Hand an den Greifer des Roboters und erneut ermittelt die Kinect die Position der Hand. Als dritten Punkt wird der Ursprung des Koordinatensystems der Kinect genommen (0,0,0). Somit sind drei Punkte innerhalb des Koordinatensystems der Kinect bekannt und die Berechnung der Rotation und Translation kann durchgeführt werden. (Siehe Abbildung 3.4)

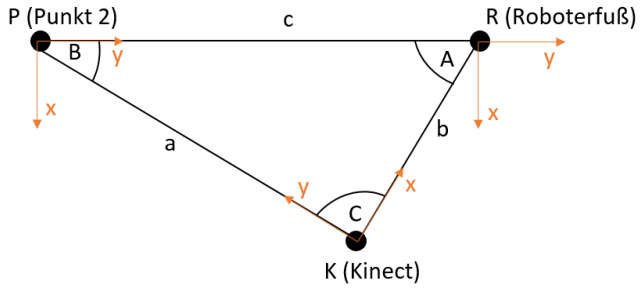


Abbildung 3.4: Dreieck aufgespannt zwischen Roboter, einem weiteren Punkt und Kinect

Ermitteln der Rotation um die z-Achse

Durch Berechnung der Winkel des aufgespannten Dreiecks ist es möglich die Rotation der Kinect zum Roboter zu ermitteln. Um die Winkel des Dreiecks zu ermitteln, müssen zunächst die Seitenlängen des Dreiecks berechnet werden. Durch Anwendung des Satz des Pythagoras (siehe Abbildung 3.5) kann die Länge der Seiten a,b und c berechnet werden.

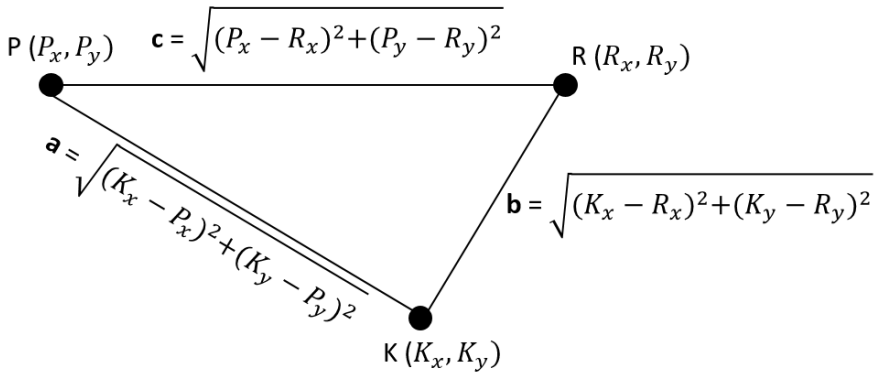
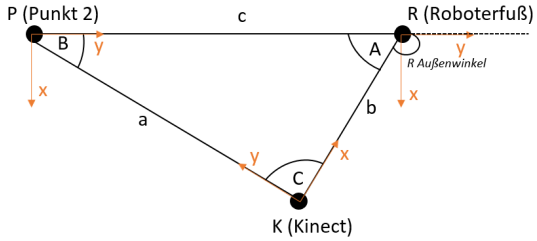


Abbildung 3.5: Distanzformel

Nachdem die Seitenlängen des Dreiecks berechnet wurden, können die Winkel des Dreiecks mithilfe des Kosinussatzes (siehe Abbildung 3.6) berechnet werden. Der Außenwinkel des Punktes des Roboterfußes zum Ursprungspunkt der Kinect ergibt die Rotation der Kinect zum Roboter.

Die Rotation um die x- und y-Achse könnte auf die gleiche Weise ermittelt werden. Dazu müssten zwei weitere Punkte im Synchronisationsprozess eingelesen und zwei weitere Dreiecke aufgespannt werden.



$$c^2 = a^2 + b^2 - 2ab * \cos A$$

$$\cos A = \frac{a^2+b^2-c^2}{2ab}$$

$$A = \arccos(\frac{a^2+b^2-c^2}{2ab})$$

Abbildung 3.6: Außenwinkel des Dreiecks bestimmt die Rotation der Kinect (links), Kosinussatz für den Winkel A (rechts)

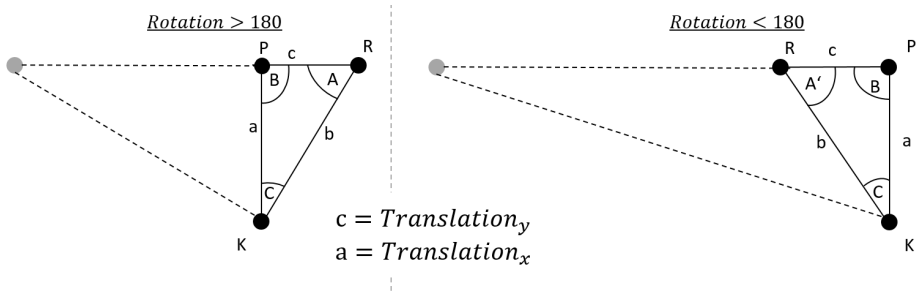


Abbildung 3.7: Berechnung der x- und y- Translation

Ermitteln der Translation

- x- und y-Achse:
  Zur Berechnung der Translation auf der x- und y-Achse wird ein rechtwinkliges Dreieck zwischen Roboterfuß, Kinect-Ursprung und einem zu berechnenden Punkt P gezogen. Da der Winkel A aus den vorigen Berechnungen der Rotation bekannt ist und es sich um ein rechtwinkliges Dreieck handelt, lassen sich auch die anderen Winkel des Dreiecks berechnen:

  $$B = 90^\circ; C = 90^\circ - A$$

  Da zudem die Punkte R und K und die somit die Länge der Seite b bekannt sind, können mit Hilfe des Sinussatzes die Längen der Seiten c (y-Translation) und a (x-Translation) berechnet werden. Dabei ist zu beachten, dass bei einer Rotation kleiner als 180° der Außenwinkel von A, also A' benutzt werden muss. (Siehe Abbildung 3.7)

  $$\frac{b}{\sin B} = \frac{Translation_y}{\sin C} \qquad \frac{Translation_y}{\sin C} = \frac{Translation_x}{\sin A}$$

  $$Translation_y = \frac{b \times \sin C}{\sin B} \qquad Translation_x = \frac{Translation_y \times \sin A}{\sin C}$$

- z-Achse:
  Die Translation auf der z-Achse, das heißt der Höhenunterschied zwischen Roboterfuß und Kinect, lässt sich aus dem ermittelten Punkt des Roboterfußes im Koordinatensystem der Kinect einfach bestimmen:

  $$Translation_z = Roboterfuss_z \times -1$$

  Da der Ursprung der Kinect im eigenen Koordinatensystem bei (0,0,0) liegt, ist klar, dass der z-Wert der Koordinate, die der Roboterfuß im Koordinatensystem der Kinect hat, der Höhenunterschied zwischen Kinect und Roboterfuß sein muss.

Ermitteln der Skalierung

Die Ermittlung der Skalierung der Koordinatensysteme wurde in diesem Projekt nicht behandelt, da die Skalierung der Koordinatensysteme von Kinect und Roboter bei einem festen Wert von 1000 liegt (Meter im Kinect Koordinatensystem zu Millimeter im Roboter Koordinatensystem). Allerdings könnte durch Vergleichen der Längen der Seiten des aufgespannten Dreiecks jeweils im Bezug des Roboter Koordinatensystems und des Kinect Koordinatensystems der Skalierungsfaktor berechnet werden.

Berechnung der Koordinatentransformation

Nach erfolgreichem ermitteln der Faktoren werden diese wie folgt verrechnet:

$$\text{Punkt Kinect} = P_k = \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \qquad \text{Punkt Roboter} = P_r = \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix}$$

- Skalierung (S): Multiplikation eines Punktes mit einem Skalierungsvektor

$$S = \begin{pmatrix} S_x \\ S_y \\ S_z \end{pmatrix} \qquad \begin{aligned} x_r &= x_k * S_x \\ y_r &= y_k * S_y \\ z_r &= z_k * S_z \end{aligned}$$

- Translation (T): Addition eines Punktes mit einem Translationsvektor

$$T = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} \qquad \begin{aligned} x_r &= x_k + T_x \\ y_r &= y_k + T_y \\ z_r &= z_k + T_z \end{aligned}$$

- Rotation (R): Multiplikation eines Punktes mit der Rotationsmatrix um die z-Achse (Rotation im Uhrzeigersinn)

$$R_z = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{aligned} x_r &= \cos\phi * x_k + \sin\phi * x_k \\ y_r &= -\sin\phi * y_k + \cos\phi * y_k \\ z_r &= z_k \end{aligned}$$

3.4 Microsoft Kinect

3.4.1 Wahl der Softwareschnittstelle

Zu Beginn der Implementierung der Kinectanwendung musste analysiert werden, welche Softwareschnittstelle benutzt wird. Zur Auswahl standen folgende Softwaredevelopment Kits:

1. Microsoft Kinect SDK 2.0
   Das Microsoft Kinect SDK 2.0 wird direkt von Microsoft entwickelt und ist speziell auf die Hardware der Kinect ausgelegt. Es lässt sich nur mit Windows 8, 8.1 oder 10 betreiben und ist für die Programmiersprachen C++ und C# verfügbar. Da das SDK speziell für die Kinect entwickelt wurde, unterstützt es alle Funktionen, die die Kinect von Haus aus unterstützen soll. Dazu gehören Personenerkennung, Gestenerkennung, Spracherkennung oder Bildtiefenerkennung.
   Durch zahlreiche Beispielanwendungen, die in dem SDK enthalten sind, fällt der Einstieg in die Programmierung sehr leicht.

2. OpenNI 2 SDK
   Das OpenNI 2 SDK wurde von der Firma occipital entwickelt und bildet die Grundlage für das Structure SDK, welches für den firmeneigenen Structure Sensor verwendet wird[15]. Der Structure Sensor ist eine 3D Kamera für Apples iPad Serie.
   Das SDK kann mit Mac OS X, Windows, Linux und Android betrieben werden und vereinfacht das Zugreifen auf den Infrarot, RGB und Tiefensensor der Kamera. Allerdings müssen zusätzlich spezielle Treiber installiert werden, damit die Kinect unterstützt wird. Zudem enthält das OpenNI 2 SDK keine Unterstützung zur Spracherkennung und zum einfachen Erkennen von Körpern. Zur Programmierung kann C++ oder Java verwendet werden.
   Da occipital für den Structure Sensor nun das Structure SDK benutzt, wird das OpenNI 2 SDK nicht mehr regelmäßig gepflegt. So liegt das letzte größere Update ein dreiviertel Jahr zurück (Stand Januar 2016).

3. OpenKinect libfreenect2
   Libfreenect2 wird von der OpenKinect Gemeinschaft betrieben. Es handelt sich dabei um eine Sammlung von Open Source Bibliotheken, die mit Windows, Mac und Linux betrieben werden können. Libfreenect2 stellt Wrapper für zahlreiche Programmiersprachen zur Verfügung. Dazu gehören Java, C++, C#, C, Phyton, Actionscript.
   Libfreenect2 stellt zudem auch alle Treiber zur Verfügung, um die Kinect betreiben zu können. Allerdings enthält das SDK keine Unterstützung zur Spracherkennung und zur Erkennung von Körpern.

Nach Analyse aller Möglichkeiten wurde sich für das Microsoft Kinect SDK 2.0 entscheiden, da es die meisten Funktionen von Haus aus mitbringt. Besonders die Unterstützung der Spracherkennung ist ein großer Vorteil gegenüber der

Konkurrenz. Außerdem ist die Dokumentation des SDKs am umfangreichsten. Dadurch fiel der Einstieg in die Programmierung sehr leicht. Ein Nachteil des Microsoft Kinect SDKs ist allerdings, dass es nur mit den neuesten Windows Betriebssystemen betrieben werden kann.

3.4.2 Frames

Die Kinect fasst die Ergebnisse der Sensoren in Frames zusammen. Dabei gibt es folgende Typen:

- Infrared: Infrarotbild der Kamera
- Depth: Tiefeninformation für jeden Pixel der Infrarotkamera
- Color: RGB-Bild
- Body: Echtzeitinformation über Körper, die sich in der Szene befinden.

Die Klasse *MultiSourceFrameReader* kann benutzt werden, um die Frames auszulesen. Die Klasse enthält einen Eventhandler (MutliSourceFrameArrived), der gefeuert wird, sobald ein neuer Frame vorhanden ist. Eine Methode an diesen Eventhandler gebunden ermöglicht es zum Beispiel Personenerkennung durchzuführen.

3.4.3 Personenerkennung mit Hilfe des Body Frames

Der Body Frame enthält Informationen über erkannte Körper im Blickfeld der Kamera. Es können bis zu sechs Personen gleichzeitig erkannt werden. Die Kamera ist außerdem im Stande dazu ein Skelett für jede Person zu erstellen, mit Informationen über die Ausrichtung und Position der Gelenke in allen drei Dimensionen und den Status der Hände der Personen. (Die Kinect kann erkennen, ob eine Person die Hand geschlossen hat, oder geöffnet.)
Mit

```
var frame = e.FrameReference.AcquireFrame().
        BodyFrameReference.AcquireFrame();
```

kann der Body Frame innerhalb der MultiSourceFrameArrived - Methode ausgelesen werden.
Mit

```
IList<Body> _bodies =
  new Body[frame.BodyFrameSource.BodyCount];
frame.GetAndRefreshBodyData(_bodies);
```

können anschließend die Informationen über die vorhandenen Körper in eine Variable gespeichert werden.

Abbildung 3.8: Das Skelett einer Person erkannt von der Kinect

Erkennung des Operators Da die Kinect bis zu sechs Personen gleichzeitig erkennen kann, und in dieser Anwendung ein einziger Operator die Anwendung benutzt, muss eine Auswahl des richtigen Körpers durchgeführt werden. Hierzu wird der Status der Hand der Körper überprüft. Der Operator der Applikation wird seine Hand geschlossen halten, um sich so von eventuellen anderen Körpern im Bild unterscheiden zu können (siehe Abbildung 3.9):

```
foreach (var body in _bodies) {
  if (body != null) {
    if (body.IsTracked &&
        body.handRightState == Handstate.closed) {
                Joint handRight =
                  body.Joints[JointType.HandTipRight];
        }
  }
}
```

Die Variable handRight vom Typ Joint enthält nun die Koordinaten der rechten Hand.

Algorithmus zur Personenerkennung Der genaue Algorithmus, den die Kinect v2 zur Erkennung von Personen verwendet, ist nicht von Microsoft veröffentlich worden. Es wird allerdings davon ausgegangen, dass es ähnlich

Abbildung 3.9: Personenerkennung anhand des Status der Hand

wie bei der ersten Kinect Version funktioniert. Hier basierte der Algorithmus auf Machine Learning. Die Kinect verfügte über mehrere Millionen Trainingsdaten und kann mithilfe eines *Random Decision Forrest* ermitteln, ob es sich bei Objekten im Bild um Körper handelt und wo sich welche Gelenke befinden[17].

3.4.4 Spracherkennung

Zur Spracherkennung wurde das mit dem Microsoft Kinect SDK 2.0 kombinierbare Microsoft Speech Platform SDK benutzt. Bei der Installation des SDKs kann ausgewählt werden, welche Sprache die Spracherkennung unterstützen soll. In diesem Projekt wird en-US als Sprache verwendet. Die Spracherkennung mit der Kinect und dem SDK erfolgt in zwei Schritten. Zunächst muss die Sprachengine initialisiert werden, anschließend muss auf die eingegangene Sprache reagiert werden.

Initialisierung der Spracherkennung Zur Initialisierung der Spracherkennung wird ein Objekt der Klasse *SpeechRecognitionEngine* erstellt. Als Parameter wird dabei die ID des Kinect Speech Recognizers übergeben, damit der Sprachengine das richtige Spracherkennungsmodul zugewiesen werden kann. Anschließend können die möglichen Spracheingaben definiert werden. Einem Objekt der Klasse *Choices* werden alle möglichen Spracheingaben angehängt. Diese werden daraufhin einem Objekt der Klasse *GrammarBuilder* zugewiesen, welches von einem Objekt der Klasse *Grammar* benutzt wird, um die möglichen Spracheingaben in die Sprachengine zu laden. Anschließend wird dem Event-Handler der Sprachengine eine Methode übergeben, welche aufgerufen wird, wenn die Sprachengine Spracheingabe erkennt. (Siehe Listing 3.2)

Listing 3.2: Initialierung der Spracherkennung

```
this.speechEngine = new SpeechRecognitionEngine(ri.Id);
var speech_input = new Choices();
speech_input.Add(
  new SemanticResultValue("move gripper", "MOVE"));
speech_input.Add(
  new SemanticResultValue("rotate gripper", "ROTATE"));
speech_input.Add(
  new SemanticResultValue("gripper open", "GRIPPEROPEN"));
speech_input.Add(
  new SemanticResultValue("gripper close", "GRIPPERCLOSE"));
...
var gb = new GrammarBuilder { Culture = ri.Culture };
gb.Append(speech_input);
var g = new Grammar(gb);
this.speechEngine.LoadGrammar(g);
this.speechEngine.SpeechRecognized += this.SpeechRecognized;
```

Behandlung der Spracheingabe In der Methode, die dem EventHandler der Sprachengine zugewiesen wurde, wird die Spracheingabe behandelt und entschieden, was bei erkannter Sprache getan werden soll. Ein confidenceThreshold Wert bestimmt, wie exakt ein Wort bzw. ein definierter semantischer Wert übereinstimmen muss, damit die Sprache erkannt wird. In einer Switch - Bedin-

Listing 3.3: Eventhandler der Spracherkennungseingine

```
private void SpeechRecognized(object sender,
  SpeechRecognizedEventArgs e) {
    const double ConfidenceThreshold = 0.3;
    if (e.Result.Confidence >= ConfidenceThreshold) {
          switch (e.Result.Semantics.Value.ToString()) {
            case "MOVE":
                  this.action_from_speech = 2;
                  goto case "RESETTIMER";
                case "ROTATE":
                  this.action_from_speech = 3;
                  goto case "RESETTIMER";
                case "GRIPPEROPEN":
                  this.action_from_speech = 21;
                  break;
                case "GRIPPERCLOSE":
                  this.action_from_speech = 20;
                  break;
                ...
          }
   }
}
```

gung kann nun auf die einzelnen möglichen Spracheingaben eingegangen werden und eine entsprechende Aktion ausgeführt werden. (Siehe Listing 3.3)

3.5 Programmierung des Roboters

3.5.1 Roboteranwendung

Die jetzige Roboteranwendung dient nur zur Präsentation der möglichen Operationen und hat keinen übergreifenden Zweck. Nach der Synchronisation der Koordinatensysteme wird dem Nutzer ein Dialog auf dem Smartpad angezeigt. Dieser Dialog gibt dem Operator die Möglichkeit alle implementierten Funktionalitäten auszuführen. Dazu gehört das Anfahren der Fingerposition, das Greifen einen Objektes von der Fingerposition, das erneute synchronisieren der Koordinatensysteme sowie das Öffnen und Schließen des Greifers. Außerdem lauscht die Roboteranwendung durchgehend auf Sprachkommandos, die von der Kinect kommen und führt diese aus, sobald sie übermittelt worden sind.

3.5.2 Klassenbeschreibung

Die Roboteranwendung setzt sich aus den folgenden Klassen zusammen:

Bewegungsablauf Die Klasse Bewegungsablauf ist die Hauptklasse der Anwendung. Sie erbt von der Klasse RoboticsAPIAppliaction und initialisiert und startet die Roboter Anwendung.

Logger Die statische Klasse Logger enthält eine statische Methode zum Ausgeben von Nachrichten. Mit ihrer Hilfe kann aus jeder Klasse der Applikation eine Nachricht auf das Smartpad des Roboters ausgegeben werden.

Points Die Klasse Points enthält die Klassenvariablen für die Rotation und Translationswerte der beiden Koordinatensysteme. Zudem enthält sie Methoden um die Translation und Rotation der Koordinatensysteme auf einen Punkt zu verrechnen.

RobotMovements In der Klasse RobotMovements sind Methoden zur Steuerung des Roboters hinterlegt. Es gibt Methoden um den Greifer des Roboters relativ zur vorherigen Position zu bewegen oder zu rotieren, sowie eine Methode, um zur Fingerposition des Operators zu fahren.

RobotInteractions Die Klasse RobotInteractions benutzt die Klasse RobotMovements um Aktionen durchzuführen. In RobotInteractions finden sich unter Anderem Methoden zum öffnen und schließen des Greifers des Roboters und zum Aufnehmen eines Gegenstandes von einer bestimmten Position.

RestServer Die Klasse RestServer ist für die Kommunikation zur Kinect nötig. Hier wird der Modbus Master instanziiert und es werden die verschiedenen action_codes, die der Modbus Slave übermittelt, evaluiert und eine entsprechende Aktion durchgeführt.

3.5.3 Sicherheitskonfiguration

Die richtige Sicherheitskonfiguration des Roboters hilft dabei Unfälle zu vermeiden, da der Operator in Bereichen agiert, wo auch der Roboterarm hinfährt (besonders beim Zeigen auf Punkte, die der Roboter anfahren soll), ist es essenziell, dass die Kollisionserkennung richtig eingestellt ist. Die Roboterapplikation wird automatisch gestoppt, sollte ein Drehmoment (Kollision) mit mehr als 25 Nm an einer der Achsen gemessen werden. Weiterhin wird die Geschwindigkeit der Achsenbewegung überwacht, da eine Abbruchbedingung für die Kollisionserkennung nichts bringt, wenn der Roboterarm mit sehr hoher Geschwindigkeit auf ein Zeil auftritt. Die Kollisionserkennung wird den Roboterarm zwar stoppen, allerdings ist dann die Kollision schon zum Teil aufgetreten. Deshalb wurde die Geschwindigkeit für die Achsenbewegung auf 500 mm/s begrenzt.

3.6 Projektdurchführung

Versionsverwaltung Zur Versionsverwaltung der Anwendungen wurde ein SVN Archiv angelegt, das auf dem Teamlaufwerk des Robotik Labors abgelegt wurde. Mittels WebDAV konnte das Archiv auch außerhalb des Netzes der Hochschule erreich werden.

Arbeitsablauf Während die Kinectanwendung von unterschiedlichen Orten getestet werden konnte, da die Kinect transportabel ist, konnte die Roboteranwendung nur im Institut für CIM-Technologietransfer der FH Kiel getestet werden.
Zum Testen von Funktionalitäten, die nicht unmittelbar den Roboter benötigen, wurde eine weitere Java Anwendung geschrieben. Hier konnte zum Beispiel die Kommunikation mit Modbus getestet werden, ohne dass Zugang zum Roboter nötig war.
In regelmäßigen Abständen wurden die Kinect- und die Roboteranwendung kombiniert und das Zusammenspiel getestet.

Kapitel 4

Schluss

4.1 Ergebnisse

Die programmierte Anwendung gibt dem Nutzer die Möglichkeit den Roboter auf intuitive Weise zu bedienen.
Die einfache Synchronisation zwischen Kinect und Roboter sorgt dafür, dass die Anwendung flexibel einsetzbar ist. Der Einrichtungsaufwand ist minimal: Ein beweglicher Roboter kann an einen Arbeitsplatz gefahren werden, die Kinect wird so platziert, dass der Arbeitsbereich im Bild ist und nach kurzer Synchronisation ist die Anwendung einsatzbereit.
Durch die Verwendung einer Microsoft Kinect hat der Roboter die Fähigkeit des räumlichen Sehens bekommen. Es können Punkte im Sichtbereich der Kamera angefahren werden. So kann der Roboter zum Beispiel zur Position der Hand des Operator fahren und von dieser Position einen Gegenstand aufnehmen. Ein einteachen von Punkten vor dem Starten der Anwendung ist somit nicht mehr nötig. Die Anwendung gibt dem Nutzer die Möglichkeit den Roboter per Sprache zu steuern, er kann die Position und Rotation des Grippers bestimmen, oder den Gripper schließen und öffnen. Dadurch hat der Operator des Roboters die Hände für etwaige andere Tätigkeiten frei.

Trotz der überwiegend positiven Ergebnisse des Projekts sind auch die Limitierungen der Verwendung einer Microsoft Kinect als Sensor auffällig geworden:

Genauigkeit der Bilderkennung Die Erkennung von Körperteilen mit der Kinect ist nicht hundertprozentig genau. Bei still gehaltenem Arm springt der Punkt, der die Position der Hand anzeigen soll im Bereich von +/- 5 Zentimetern. Bei gleichem Aufbau ist es bei der Synchronisation zwischen Kinect und Roboter zu unterschiedlichen Werten der berechneten Rotation der Kinect zum Roboter gekommen. Die Unterschiede lagen im Bereich von 5° - 10 °.
Das kann auch daran liegen, dass mit der Kinect nur die generelle Position einer Hand erkannt werden kann und nicht zum Beispiel die Position des Zeigefingers. Dadurch ist das präzise Einlesen eines genauen Punktes nicht leicht möglich.

Zudem kann die Kinect Körper nur erkennen, wenn das gesamte Skelett (ab Kniehöhe) im Bild zu sehen ist. Daher muss die Position der Kinect gut gewählt werden, sodass der Operator nur selten von etwas verdeckt wird.

Funktionalität der Sprachsteuerung Dass der Arbeitsort von Industrierobotern in der Regel durch laufende Maschinen einer starken Lautstärke ausgesetzt ist, kann zu Problemen mit der Spracherkennung führen. Das Mikrofon der Kinect befindet sich an der Kamera und es kann sein, dass Operator und Mikrofon weit auseinander sind. Dadurch kann die Sprache unter Umständen nicht mehr korrekt erkannt werden.
Besser wäre für eine Sprachsteuerung des Roboters über ein Headset, das der Operator sich anlegen kann. Dadurch könnte die Qualität der Spracherkennung deutlich verbessert werden.

Schlussfolgerung

Die Ergebnisse des Projekts zeigen, dass eine gewöhnliche 3D Kamera wie die Microsoft Kinect dazu benutzt werden kann einen industriellen Roboter wie den KUKA LBR iiwaa 7 R800 auf multimodale Weise, per Sprache und Gesten, zu steuern. Der Vorteil der Verwendung einer Microsoft Kinect ist, dass es sich dabei um ein Konsumentenprodukt handelt, das für jeden frei erhältlich ist. Es muss keine spezielle Kamera für den Roboter angefertigt werden. Dadurch lässt sich die Kamera auch flexibel für unterschiedliche Robotertypen verwenden. Dadurch, dass die Roboteranwendung und die Kinectanwendung komplett voneinander getrennt sind und nur über eine Schnittstelle kommunizieren, lässt sich die Kinectanwendung auch ohne Weiteres für andere Robotertypen anwenden, wenn die gleichen Schnittstellen benutzt werden.
Die Ergebnisse zeigen allerdings auch, dass eine Kinect Kamera in der aktuellen Version nicht dazu in der Lage ist präzise Angaben über Koordinaten im Raum anzugeben. Die Kamera erkennt Punkte nicht hundertprozentig genau, was bedeutet, dass die Anwendung nicht für Präzisionsaufgaben verwendet werden kann. Man kann allerdings davon ausgehen, dass sich die Qualität und Leistung von zukünftigen Versionen von 3D - Kameras weiter verbessern wird, so wie es auch schon beim Upgrade der Microsoft Kinect v1 zur Microsoft Kinect v2 der Fall war. Mit einer neuen Versionen könnten eventuell genauere Ergebnisse erzielt werden.

4.2 Ausblick

Die in diesem Projekt erzielten Ergebnisse bilden eine gute Grundlage für weitere Entwicklungen, die in unterschiedliche Richtungen gehen könnten:

Synchronisation der Koordinatensysteme Die Synchronisation der Koordinatensysteme von Kinect und Roboter lässt sich noch verbessern. Erstens sollte für eine vollständige Unabhängigkeit der Position der Kinect zum Roboter die Rotation auf x- und y-Achse im Synchronisationsprozess bestimmt werden. Dazu müssten zwei weitere Dreiecke auf den jeweiligen Achsen aufgespannt werden.
Zusätzlich kann der Synchronisationsprozess dahingehend vereinfacht werden, dass kein Eingriff eines Menschen nötig wäre. Dazu müsste an dem Greifer des Roboters ein Gegenstand befestigt werden, der von der Kinect erkannt werden kann. Zum Beispiel ein markantes Muster oder Ähnliches. Nun kann der Roboter die Synchronisationspunkte anfahren und eigenständig von der Kinect die Koordinaten des erkannten Gegenstands anfahren.

Objekterkennung Da Microsoft die Kinect als Gerät zur Erkennung von Personen ausgelegt hat, enthält das Microsoft Kinect SDK keine Methoden, die bei der Erkennung von Gegenständen helfen. Allerdings kann die Erkennung von Objekten mit weiteren Bibliotheken, wie zum Beispiel EmguCV[7], realisiert werden. Könnte die Kinect die Position von Gegenständen erkennen, könnte dies für eine Vielzahl von Anwendungsfällen benutzt werden, bei denen der Roboter mit einem Objekt interagieren muss.

Steuerung des Roboters mit Gesten Die Kinect kann Gesten von Personen erkennen und darauf reagieren. Dieses könnte benutzt werden, um den Roboter mit Gesten zu steuern. So könnten zum Beispiel vorher definierte Bewegungsabläufe des Roboters durch bestimmte Gesten gesteuert werden.

Erweiterung der Spracherkennung Die Sprachsteuerung des Roboters kann durch zahlreiche weitere Kommandos erweitert werden. Ein mögliches weiteres Kommando wäre zum Beispiel die Bewegung des Roboters zu einem vorher benannten Punkt. Man könnte also bestimmte Punkte mit einem Kommando speichern und einem Namen geben. „Save current Position to Point Lincoln“. Mit dem Kommando „Move to Point Lincoln“ könnte der Roboter anschließend zu diesem Punkt gefahren werden.

Verwendung von anderen Kamerasystemen In einem folgenden Projekt könnten die Ergebnisse, die mit der Verwendung einer Microsoft Kinect erzielt wurden mit anderen 3D - Kamerasystemen verglichen werden. Mögliche alternative Konsumerprodukte wären zum Beispiel der Structure Sensor von

Occipital[15] oder der von Asus entwickelte Xtion PRO bzw. Xtion Sensor[3]. Zudem könnte überprüft werden welche Ergebnisse mit eigens für die Robotik entwickelten Sensoren (z.B. MobileRanger - StereoCamera[1] oder Roboception[16]) erzielt werden können.

Abbildungsverzeichnis

1.1 ARMAR 3, ein humanoider Roboter entwickelt am Karlsruher Institut für Technologie [http://bilder3.n-tv.de/img/incoming/origs10286266/6098257657-w778-h550/29756021.jpg] 4

2.1 Aufbau der Microsoft Kinect V2 7
2.2 Prinzip von Time-of-Flight [http://www.teraranger.com/wp-content/uploads/2015/04/principle.png] 8
2.3 Prinzip von Beamforming [https://upload.wikimedia.org/wikipedia/de/6/61/Funktion_mikrofonarray.jpg] 8
2.4 Aufbau des Projekts 9

3.1 Aufbau der Kommunikation 11
3.2 Konvertierung und Übertragung von Gleitkommazahlen 13
3.3 Vergleich der Koordinatensysteme von Roboter und Kinect . . . 14
3.4 Dreieck aufgespannt zwischen Roboter, einem weiteren Punkt und Kinect 15
3.5 Distanzformel 16
3.6 Außenwinkel des Dreiecks bestimmt die Rotation der Kinect (links), Kosinussatz für den Winkel A (rechts) 16
3.7 Berechnung der x- und y- Translation 17
3.8 Das Skelett einer Person erkannt von der Kinect 21
3.9 Personenerkennung anhand des Status der Hand 22

Programm-Listings

3.1 Auszug aus der Modbus Slave Methode zur Evaluierung der Modbus Master Anfrage anhand des Startregisters 11
3.2 Initialierung der Spracherkennung 23
3.3 Eventhandler der Spracherkennungseingine 24

Literaturverzeichnis

[1] Adept MobileRobots. (2016) Mobileranger - stereocamera. http://www.mobilerobots.com/Accessories/MobileRanger.aspx (17.03.2016).

[2] B. Akan, A. Ameri, B. Çürüklü, and L. Asplund, "Intuitive industrial robot programming through incremental multimodal language and augmented reality," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3934–3939.

[3] ASUSTek Computer Inc. (2016) Xtion pro. http://www.asus.com/3D-Sensor/Xtion_PRO/ (17.03.2016).

[4] J. M. Azorín and J. L. Pons, "Guest editorial: Special issue on multimodal human–robot interfaces," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 6, pp. 1140–1141, 2012.

[5] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo, *Time-of-flight cameras and Microsoft Kinect*. Springer Science & Business Media, 2012, ch. 1, pp. 3–4.

[6] K. Doelling, J. Shin, and D. O. Popa, "Service robotics for the home: a state of the art review," in *Proceedings of the 7th International Conference on Pervasive Technologies Related to Assistive Environments*. ACM, 2014, p. 35.

[7] Emgu CV. (2016, feb) Emgu cv. http://www.emgu.com/wiki/index.php/Main_Page (17.03.2016).

[8] N. Hofemann, "Videobasierte handlungserkennung für die natürliche mensch-maschine-interaktion," p. 1, 2006.

[9] Honda. (2016, mar) Asimo by honda. http://asimo.honda.com/ (07.03.2016).

[10] Java Modbus Library (jamod). (2010, feb) Java modbus library (jamod). http://jamod.sourceforge.net/index.html (17.03.2016).

[11] Karlsruher Institut für Technologie. (2012, may) Armar-familie. http://his.anthropomatik.kit.edu/241.php (07.03.2016).

[12] KUKA AG. (2016) Lbr iiwa 7 r800. http://www.kuka-robotics.com/germany/de/products/industrial_robots/sensitiv/lbr_iiwa_7_r800/start.htm (17.03.2016).

[13] Microsoft. (2016) Kinect hardware. https://dev.windows.com/en-us/kinect/hardware (17.03.2016).

[14] NModbus4. (2016, feb) Nmodbus4. https://github.com/NModbus4/NModbus4 (17.03.2016).

[15] Occipital, Inc. (2016) Structure sensor. http://structure.io/ (17.03.2016).

[16] Roboception GmbH. (2016) Roboception. http://roboception.com/ (17.03.2016).

[17] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[18] M. Woelfel and J. McDonough, *Distant Speech Recognition*. John Wiley & Sons, Ltd, apr 2009, ch. 13, pp. 409–411.