Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Stephan Kleborn

Evaluierung einer Microsoft Kinect v2 zur gesten- und sprachbasierten Steuerung eines Roboterarmes

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Stephan Kleborn

Evaluierung einer Microsoft Kinect v2 zur gesten- und sprachbasierten Steuerung eines Roboterarmes

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung im Studiengang Informations- und Elektrotechnik am Department Informations- und Elektrotechnik der Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. J. Maaß
Zweitgutachter: Prof. Dr.-Ing. J. Dahlkemper

Abgegeben am: 03.08.2016

Stephan Kleborn

Thema der Bachelorarbeit
Evaluierung einer Microsoft Kinect v2 zur gesten- und sprachbasierten Steuerung eines Roboterarmes

Stichworte
Microsoft Kinect, Kinect v2, Tiefensensor, *Kinect for Windows* SDK 2.0, ToF, Gestensteuerung, Sprachsteuerung, Roboterarm, UR5-Roboter

Kurzzusammenfassung
Diese Abschlussarbeit untersucht die Beschaffenheit und Anwendungsmöglichkeiten eines Kinect v2-Sensors von Microsoft für die Gesten- und Sprachsteuerung eines Roboterarmes. Grundlage ist die Untersuchung der technischen Beschaffenheit und Funktionen der Kinect für diese Anwendung. Der Schwerpunkt des Projekts bildet die Entwicklung einer Applikation auf Basis erfasster Körperdaten im dreidimensionalen Raum. Die Programmierung dieser Applikation erfolgt unter Windows.

Stephan Kleborn

Title of the paper
Evaluation of a Microsoft Kinect v2 for a gesture and speech control of a robotic arm

Keywords
Microsoft Kinect, Kinect v2, depth sensor, *Kinect for Windows* SDK 2.0, ToF, gesture control, speech control, robotarm, UR5-robot

Abstract
This thesis examines the configuration and application possibilities of a Microsoft Kinect v2 sensor for gesture and speech control of a robotic arm. Basis is the examination of the technical state and the features of the Kinect for this application. The focus of this project constitutes the development of a software based on tracked body data in the three-dimensional space. The coding of this software occures within Windows.

Inhaltsverzeichnis

Abkürzungsverzeichnis **6**

1 Einleitung **7**
1.1 Motivation . . . 7
1.2 Aufgabenstellung . . . 7
1.3 Aufbau der Arbeit . . . 8

2 Grundlagen **9**
2.1 UR5-Roboter . . . 9
2.2 Microsoft Kinect . . . 11
2.2.1 Vergleich Kinect v1 und Kinect v2 . . . 11
2.2.2 Tiefenmessverfahren mit *Time of Flight* . . . 14
2.2.3 Systemanforderungen und Schnittstelle . . . 18
2.2.4 *Kinect for Windows* SDK und API . . . 20
2.3 Mathematische Berechnungen . . . 30
2.3.1 Geometrische Transformationen . . . 30
2.3.2 Homogene Koordinaten und Transformation . . . 32
2.3.3 Aufeinanderfolgende Transformation . . . 33
2.3.4 Bezugssysteme und Koordinatentransformation . . . 34
2.4 Entwicklungsumgebung und Programmiersprache . . . 35
2.5 *OpenCV* und *Qt* . . . 35

3 Implementierung **37**
3.1 Aufbau . . . 37
3.1.1 Bezugssysteme . . . 38
3.1.2 Positionierung der Kinect v2 . . . 39
3.1.3 Bewegungsbereich des Roboters . . . 39
3.2 Realisierung der Gestensteuerung . . . 40
3.2.1 Roboterscript *kinectComm_movej* . . . 41
3.2.2 Gestensteuerung *KinectRobotControl* . . . 42

4 Funktionstest **51**
4.1 Abrufen der Kinect-Frames . . . 51
4.2 Interaktion mit dem UR5-Roboterarm . . . 53

5 Fazit und Ausblick **54**

Literaturverzeichnis **56**

Abbildungsverzeichnis **59**

Tabellenverzeichnis **61**

Anhang **62**

Abkürzungsverzeichnis

API	Application Programming Interface
dB	Dezibel
fps	frames per second
GUI	Graphical User Interface
HCI	Human Computer Interaction
HD	High Definition
ID	Identifikator
IR	Infrarot
OpenCV	Open Source Computer Vision
QML	Qt Meta-Object Language
RGB	Rot-Grün-Blau
SDK	Software Development Kit
SL	Structured Light
SoC	System on a Chip
SQL	Structured Query Language
TCP	Tool Center Point
ToF	Time of Flight
USB	Universal Serial Bus
VGB	Visual Gesture Builder
XEF	eXtended Event File

1 Einleitung

1.1 Motivation

Obwohl die Bedienung grafischer Oberflächen mit Tastatur und Maus die beliebteste Kommunikationsmethode in der *Human Computer Interaction (HCI)* ist, hat sie ihre Grenzen. Bei der Interaktion mit dreidimensionalen Objekten ist die Maus mit ihren zwei Freiheitsgraden nicht in der Lage, die drei Dimensionen des Raumes abzudecken [Kau14, S. 633]. Die Steuerung eines Roboters, wie z.B. dem UR5, ist eine solche Interaktion. Auch er lässt sich über ein *Graphical User Interface (GUI)* bedienen, was sich aber nicht besonders natürlich anfühlt. Eine Alternative ist die Steuerung mit Gesten und Sprache. Erste Applikationen für sprachliche und gestische Interaktion wurden schon in den 80er Jahren entwickelt wie etwa *Videoplace* und *Put-That-There* [Pre15, S. 473 f.]. Doch ist es die Spieleindustrie, die in den letzten Jahren bei der Entwicklung und Einführung massentauglicher Konzepte für die Gesten- und Sprachsteuerung große und schnelle Fortschritte erzielt hat. Sehr erfolgreiche Beispiele sind die Wii von Nintendo und die Kinect von Microsoft. Beide basieren auf unterschiedlichen Konzepten. Während die Wii mit einem Controller bedient wird, ist der Controller bei der Kinect der Spieler selbst [Pre15, S. 517]. Der Tiefenmesssensor Kinect besitzt eine Reihe von Funktionen für die Bildverarbeitung, ist kostengünstig und lässt sich auch ohne die Spielekonsole XBox verwenden, für die er eigentlich entwickelt wurde. Als Kommunikationsgerät für die Interaktion mit dem Roboterarm UR5 ist der Kinect-Sensor somit durchaus vielversprechend.

1.2 Aufgabenstellung

Im Rahmen dieser Arbeit werden die Funktionen des Kinect-Sensors der zweiten Generation untersucht. Neben den Datenquellen wird auch der Umfang des *Kinect for Windows* Software Development Kits (SDKs) 2.0 betrachtet. Dieses ist frei zu erwerben und für die Anwendung des Sensors ohne die Spielekonsole XBox One zwingend erforderlich.

Für die Evaluierung der Kinect wird eine Applikation entwickelt, mit der sich der Roboterarm UR5 mit Hilfe von Benutzergesten steuern lässt. Es muss ermittelt werden, welche Datenquellen der Kinect für die Gestensteuerung geeignet sind und wie sich die erforderlichen Daten verarbeiten und anwenden lassen. Nach erfolgreicher Implementierung mit anschließenden Anwendungstests wird eine Bewertung des Kinect-Sensors für den Gebrauch als Eingabegerät für die Gestensteuerung des UR5-Roboterarmes durchgeführt.

1.3 Aufbau der Arbeit

Im Anschluss an die Einleitung werden die für die Gestensteuerung erforderlichen zwei Komponenten, der UR5-Roboterarm und der Kinect-Sensor vorgestellt. Der Schwerpunkt liegt im technischen Aufbau, der Technologie der Tiefenmessung sowie den hard- und softwaretechnischen Features. Es folgt ein Einblick in die für die Implementierung der Robotersteuerung relevanten mathematischen Berechnungen und eine Übersicht der verwendeten Softwaretools.
Im dritten Kapitel wird der Aufbau der Testumgebung und die Funktionsweise der zu implementierenden Gestensteuerung des UR5 vorgestellt. Einzelnen Programmelemente und -abläufe werden beschrieben.
Nach Implementierung der Gestenapplikation wird diese getestet und bewertet.
Im Fazit und Ausblick werden die wesentlichen Ergebnisse zusammengefasst und der Einsatz des Kinect v2-Sensors für eine gesten- und sprachbasierte Steuerung beurteilt. Weitere Einsatzmöglichkeiten mit dem Kinect-Sensor werden vorgeschlagen.

Die Quelldateien des Roboterscripts *kinectComm_movej* und der gesamte Quellordner mit den Quell- und Projektdateien der Robotersteuerung *KinectRobotControl* sind dem Anhang auf DVD beigefügt.

2 Grundlagen

Zunächst erfolgt ein technischer Überblick des hier genutzten Roboterarmes und des für die Erfassung von Bild- und Koordinatendaten verwendeten Tiefensensors Microsoft Kinect. Weiterhin werden die mathematischen Beschreibungen für die Implementierung einer Testapplikation präsentiert und eine Einführung in die für die Umsetzung hilfreichen Programmierbibliotheken *OpenCV* und *Qt* gegeben.

2.1 UR5-Roboter

Als Manipulator für dieses Projekt wird der Industrieroboter UR5 der Firma Universal Robots verwendet. Abbildung 1 zeigt den UR5 Roboterarm.

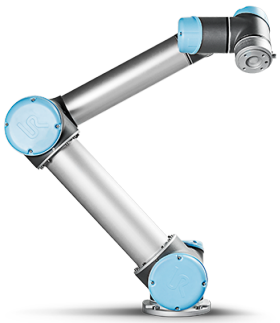


Abbildung 1: UR5-Roboter von Universal Robots [Uni15b]

Der Roboter besitzt einen Arbeitsradius von bis zu 850 mm und eine maximale Traglast von 5 kg. Er besteht aus sechs Drehachsen, die ihm sechs Freiheitsgrade ermöglichen. Drei Freiheitsgrade entsprechen der Position im Raum (x, y, z) und drei weitere der Orientierung (ϕ_x, ϕ_y, ϕ_z). Jede Rotationsachse hat eine Reichweite von $\pm 360°$.
Über Ethernet lässt sich eine TCP/IP Kommunikationsverbindung zwischen dem UR5 und externen Rechenmaschinen aufbauen [Uni15a, S. I-69].

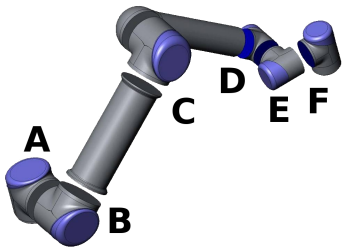
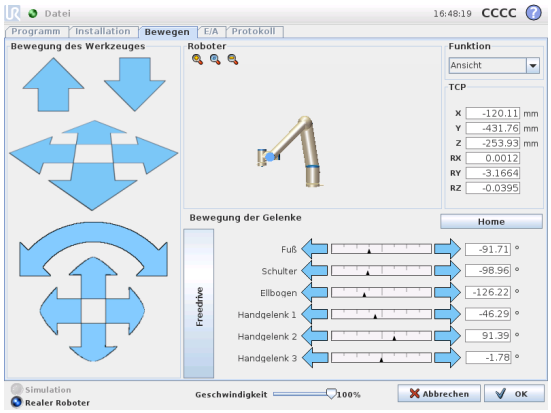
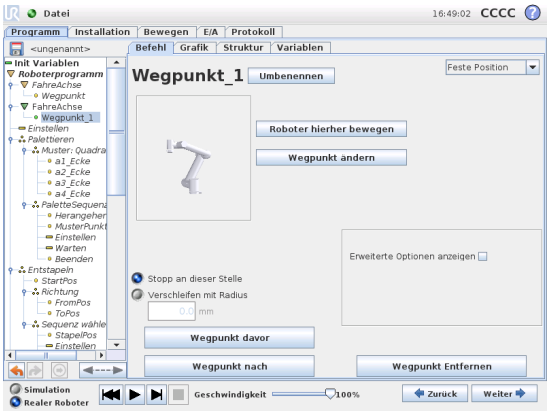


Abbildung 2: UR5-Gelenke. A: Fußflansch, B: Schulter, C: Ellenbogen, D,E,F: Handgelenk 1, 2, 3 [Uni15a, S. II-3]

Der Abbildung 2 lässt sich die Anordnung der Gelenke des UR5 entnehmen. Am Fußflansch (s. A) ist der Roboter auf der Werkbank montiert. Werkzeuge können am Handgelenk 3 (s. F) befestigt werden.



(a) „Move“-Tab [Uni15a, S. II-19]



(b) „Programm“-Tab [Uni15a, S. II-55]

Abbildung 3: Bedienung UR5

Der Roboterarm wird über ein GUI mit Namen *PolyScope* bedient. Mit der Bedienoberfläche in Abbildung 3a lässt sich der Roboter direkt bewegen. Dies geschieht entweder durch die Bewegung des Roboterwerkzeugs oder durch die Vorgabe der Winkel der einzelnen Robotergelenken. In Abbildung 3b ist ein Auszug der Programmieroberfläche zu sehen. In ihr lässt sich ein Roboterprogramm erzeugen, verändern oder ausführen. Die Programmierung erfolgt mit der Scriptsprache *URScript*. Die detaillierte Beschreibung der Strukturen, Variablen und Anweisungen kann im Scriptmanual [Uni15c] eingesehen werden.

2.2 Microsoft Kinect

Die Microsoft Kinect ist ein Bild und Ton erfassendes Eingabegerät für die von Microsoft entwickelten Spielekonsolen Xbox 360 und Xbox One. Durch den Sensor wird der traditionelle Controller bei der Interaktion mit den Konsolen ersetzt. Die aufgenommenen Daten lassen sich außerdem am PC weiterverarbeiten und für verschiedenste Anwendungen nutzen. Mit der Kinect können Rot-Grün-Blau (RGB)-, Infrarot (IR)- und Tiefenbilder mit einer Bildfrequenz von bis zu 30 frames per second (fps) aufgenommen werden. Es lassen sich außerdem Audiodaten erfassen und verarbeiten.
Der Sensor wurde von Microsoft in Kooperation mit der israelischen Firma PrimeSense entwickelt. Am 4. November 2010 begann der offizielle Verkaufsstart der Kinect als Accessoire für die Xbox 360 in Nordamerika. Am 16. Juni 2011 veröffentlichte Microsoft sein offizielles *Kinect for Windows SDK*, mit dem es nun möglich war den Sensor auch unter Windows zu nutzen. Aufgrund seines niedrigen Preises und der großen Verfügbarkeit ist das Modul, neben der Spieleindustrie, auf vielen verschiedenen Gebieten in der Forschung beliebt. Dabei sind vor allem die Robotik, Biomedical Engineering und Computer Vision hervorzuheben [Pag15, S. 27569 f.].
Mit 8 Million verkauften Einheiten in den ersten 60 Tagen nach Veröffentlichung ist es das bis dahin am schnellsten verkaufte elektronische Unterhaltungsgerät und wurde in das Guinness Buch der Rekorde aufgenommen [Bil11].

2.2.1 Vergleich Kinect v1 und Kinect v2

2013 wurde eine zweite Version der Kinect vorgestellt und ist für Forscher und Entwickler seit Juli 2014 erhältlich [Fan15, S. 1]. Für die eindeutige Unterscheidung werden die verschiedenen Sensorgenerationen im folgenden Vergleich mit Kinect v1 bzw. v2 bezeichnet.

Kinect v1 Der Kinect-Sensor der ersten Generation wurde für die Spieleindustrie entwickelt und erlaubt die Interaktion mit der Konsole Xbox 360 ohne ein Zusatzgerät. Die Steuerung erfolgt allein mittels Sprache und Gesten des Nutzers [Pag15, S. 27570 f.]. Die Kinect v1 nutzt für die Tiefenmessung ein aktives *Structured Light (SL)*-Verfahren. Eine Laserdiode erzeugt Licht mit einer Wellenlänge von 850 nm im Infrarotbereich. Ein Muster des strukturierten Infrarotlichts wird in Form einer Punktewolke auf ein Objekt projiziert. Das Muster wird von der geometrischen Form des Objektes verzerrt und mit dem verbauten IR-Sensor der Kinect wieder erfasst. Die eigentliche Berechnung der Tiefendaten

erfolgt dann über die Methode der Triangulation. Dabei wird das vom Sensor erfasste Muster mit den bekannten Daten des vom Projektor ausgesendeten Musters verglichen [Sar15, S. 5-7].

Abbildung 4: Kinect der zweiten Generation [Mic14, S. 5]

Kinect v2 Der Sensor in Abbildung 4 zeigt die zweite Generation der Kinect. Er wurde für die Xbox One entwickelt. Anders als sein Vorgänger arbeitet dieses System mit dem *Time of Flight (ToF)*-Prinzip, welches in Unterabschnitt 2.2.2 näher ausgeführt wird. Bis auf den fehlenden Neigungsmotor besteht das Modul wie das Vorgängermodell aus einer RGB-Kamera, einem IR-Projektor, einem IR-Sensor und einem Array aus vier einzelnen Mikrofonen. In Abbildung 5 sind die verbauten Komponenten der Kinect v2 zu sehen.

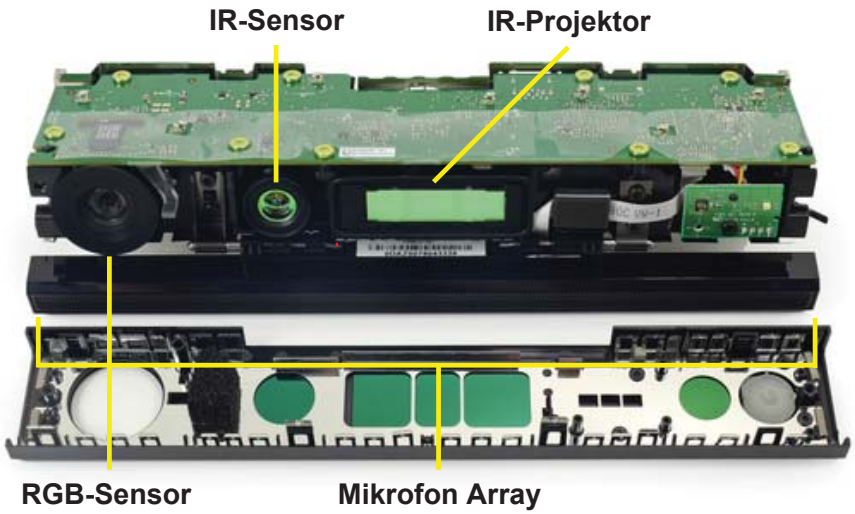


Abbildung 5: Komponenten der Kinect v2 [iFi16]

Die Kinect v2 führt eine präzisere Tiefenmessung durch als ihr Vorgänger. Die berechneten Punktewolken zeigen eine bessere Auflösung, da die Distanzmessung mit dem ToF-Prinzip für jedes einzelne Pixel der aufgenommenen Tiefenbilder durchgeführt wird. Durch die bessere Auflösung lassen sich kleine Objekte besser abbilden [Lac15, S. 99]. Ein weiterer Vorteil der neuen Version ist die Aufnahme von Farbbildern in *High Definition (HD)*. Die Auflösung der RGB-Kamera der Kinect v1 beträgt nur 640×480 px. Weiterhin hat sich das horizontale Sichtfeld der RGB-Kamera und das vertikale Sichtfeld der IR-Kamera vergrößert [Pag15, S. 27571]. Die Abbildung 6 zeigt das Sichtfeld des Sensors für Infrarot und Tiefenmessung.

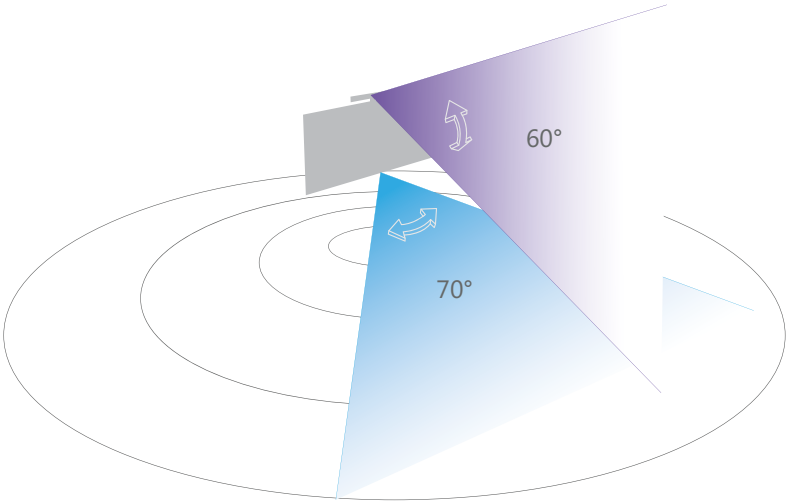


Abbildung 6: Sichtfeld des Kinect v2 für Infrarot und Tiefemessung [Mic14, S. 7]

Eine Übersicht der wichtigsten Kenndaten der beiden Kinectgenerationen und der Vergleich dieser ist der Tabelle 1 zu entnehmen.

Merkmal	*Kinect v1*	*Kinect v2*
Auflösung RGB-Kamera	640×480 px	1920×1080 px
Sichtfeld RGB-Kamera (h x v)	57.5° x 43.5°	84° x 53°
Auflösung IR-Kamera	bis zu 640×480 px	512×424 px
Sichtfeld IR-Kamera (h x v)	57.5° x 43.5°	70° x 60°
Neigungsmotor	ja	nein
vertikaler Neigungswinkel	±27°	-
Bildwechselfrequenz	30 fps	30 fps
Arbeitsbereich	0.8 m - 4 m	0.5 m - 4.5 m
Anzahl Skelett-Gelenkpunkte	20	25
Anzahl erfasster Personen	2	6
USB	2.0	3.0

Tabelle 1: Vergleich der beiden Kinect-Versionen

Es konnte gezeigt werden, dass die RGB- und IR-Bilder bei der Kinect v2 stabiler und weniger verschwommen sind. Es werden außerdem präzisere Objektkanten dargestellt und in den Tiefenbildern sind mehr sichtbare Details zu verzeichnen [Pag15, S. 27576].

Aufgrund der besseren Eigenschaften und Leistung wird für dieses Projekt die Kinect v2 gewählt.

2.2.2 Tiefenmessverfahren mit *Time of Flight*

Da in diesem Projekt die Kinect v2 zum Einsatz kommt, wird im Folgenden das Tiefenmessverfahren mit ToF genauer beschrieben. Sell *et al.* geben einen Überblick über die Systemkomponenten der Kinect v2 und erklären die Ermittlung der erforderlichen Tiefendaten mit dem ToF-Pinzip.

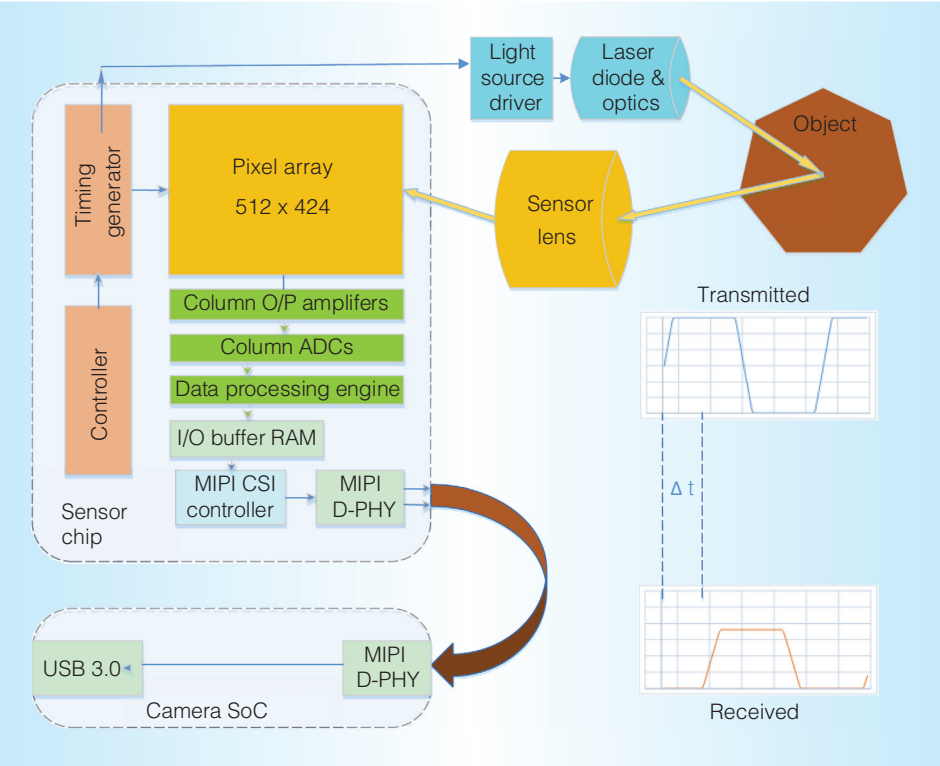


Abbildung 7: 3D Sensorsystem [Sel14, S. 49]

In Abbildung 7 ist das 3D Sensorsystem der Kinect v2 dargestellt. Es besteht aus einem Sensorchip, einem Kamera-*System on a Chip (SoC)* und einer Belichtungseinheit mit Laserdiode. Das SoC verwaltet den Sensor und kommuniziert über die USB 3.0 Schnittstelle mit dem Betriebssystem der Haupteinheit. Die Haupteinheit entspricht der Xbox One oder wie in dieser Arbeit dem Windows-PC [Sel14, S. 49].

Arbeitsprinzip ToF-Sensor Ein Taktgeber erzeugt ein moduliertes Rechtecksignal. Dieses Signal wird verwendet, um die Lichtquelle im Sender und die Pixel des Pixel-Arrays im Empfänger abzustimmen. Das von der Quelle ausgesendete Licht bewegt sich in einer Zeit Δt zu einem Objekt und zurück zum Empfänger. Das System nutzt die Phasenerkennung, um diese Zeit zu ermitteln. Mit der Kenntnis der Modulationsfrequenz f_{mod} und der gemessenen Phasenverschiebung $\Delta\varphi_{rad}$ zwischen gesendetem und empfangenem Licht errechnet das System die Zeit Δt. Zu beachten ist, dass die Phasenverschiebung im Bogenmaß gemessen wird. Diese Operation erfolgt für jedes einzelne Pixel des Arrays. Die Berechnung der Tiefe erfolgt dann über die Lichtgeschwindigkeit $c = \frac{1}{33}\frac{cm}{psec}$ mit der Formel:

$$2d = \frac{\Delta\varphi_{rad}}{2\pi} \cdot \frac{c}{f_{mod}} \tag{1}$$

Mit dem Zusammenhang zwischen dem Phasenwinkel $\Delta\varphi_{rad}$ und der Laufzeit Δt

$$\Delta t = \frac{\Delta\varphi_{rad}}{2\pi \cdot f_{mod}} \tag{2}$$

ergibt sich für die Tiefe d

$$d = \frac{c \cdot \Delta t}{2} \tag{3}$$

[Sel14, S. 49]

Differenzierende Pixel Was den ToF-Sensor von einem klassischen Kamerasensor unterscheidet, ist das differenzierende Pixel-Array. Jedes einzelne Pixel besitzt zwei unabhängige Ausgänge, die von einem *Clock*-Signal gesteuert werden.
Abbildung 8 zeigt den ToF-Sensor und die für die Tiefenmessung erforderlichen Signale. *Light* und *Return* stellen den Verlauf des gesendeten und empfangenen Lichtsignals dar. An den Kanälen *A out* und *B out* sind die Ausgangsspannungen der unabhängigen Ausgänge eines Pixels zu erkennen. *Clock* ist dabei der Ausblendtakt am Pixel.

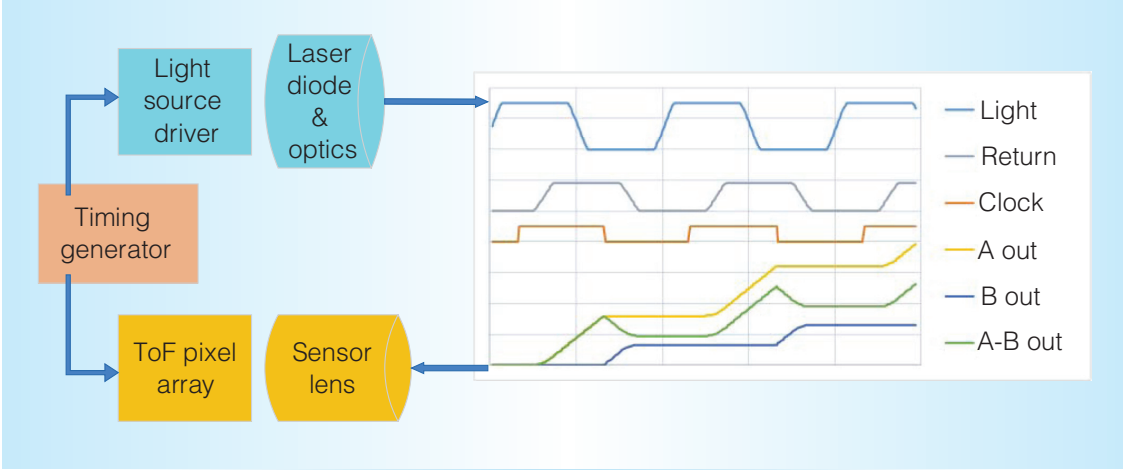


Abbildung 8: ToF-Sensor und dessen Signalformen [Sel14, S. 50]

Der Taktgeber erzeugt Taktsignale, um das Pixel-Array zu steuern und ein Synchronsignal für die Modulation der Lichtquelle. Das modulierte Licht wird von einer Laserdiode ausgesendet. Das Signal verlässt die Kinect, wird von Objekten im Sichtfeld reflektiert und gelangt mit einer Verzögerung und einer Abschwächung zur Sensorlinse zurück. Die Linse bündelt das Licht auf die Sensorpixel. Das Taktsignal stimmt den Pixelempfänger ab.

Das auf die Pixel fallende Licht wird in ein Spannungssignal umgewandelt und lädt die beiden Ausgänge des Pixels, ähnlich wie ein Akkumulator auf. Wie in den Signalverläufen von Abbildung 8 zu erkennen, laden die einfallenden Photonen bei einem „*high*“ des *Clocks* den *A out*-Kanal und bei einem „*low*“ den *B out*-Kanal des Pixels.
Eine $(A - B)$-Berechnung am Pixelausgang liefert dann ein Signal, dessen Wert sich aus der Intensität des zurückkehrenden Lichtes und der Ankunftszeit des Lichtes, in Abhängigkeit des Pixeltakts, zusammensetzt. Diese Berechnung ist das Kernelement der ToF-Phasenerkennung.
Im Folgenden werden einige mathematische Beziehungen der Pixelausgänge vorgestellt, die zu einer Reihe unterschiedlicher Darstellungen führen:

- $(A + B)$ erzeugt ein Graustufenbild bei normaler Umgebungsbeleuchtung („Umgebungsbild“).
- $(A - B)$ liefert Phaseninformationen nach einer arctan-Berechnung („Tiefenbild“).
- $\sqrt{\left(\sum (A - B)^2\right)}$ erzeugt ein vom Umgebungslicht unabhängiges Graustufenbild („Aktivbild“).

[Sel14, S. 50]

Abtastung über große Bereiche mit feiner Auflösung Um eindeutige Tiefeninformationen bei der Berechnung mittels Phasenerkennung zu erhalten, nutzt die Kinect v2 mehrere Modulationsfrequenzen. Diese Frequenzen liegen bei 120 MHz, 80 MHz und 16 MHz.

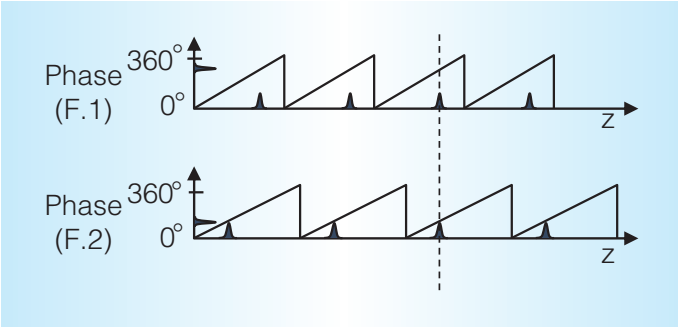


Abbildung 9: Unterschiedliche Modulationsfrequenzen [Sel14, S. 51]

Wie in Abbildung 9 zu sehen, ergibt jede Einzelfrequenzphase mehrdeutige Tiefenmesswerte. Die Kombination mehrerer Frequenzen macht diese Messwerte eindeutig. Die Phasenmesswerte der Frequenz F sind auf der vertikalen und die der Tiefendaten auf der horizontalen Achse abgebildet [Sel14, S. 51].

Um genauere Tiefenmessergebnisse zu erhalten, sollte die Aufwärmzeit des Sensors berücksichtigt werden. Die Messgenauigkeit als Funktion der Aufwärmzeit des Sensors ist in Abbildung 10 zu sehen. Der gemessene Abstand variiert um 5 mm. Erst nach 30 Minuten bleiben die Werte mit einer Abweichung von ±1 mm konstant [Lac15, S. 96].

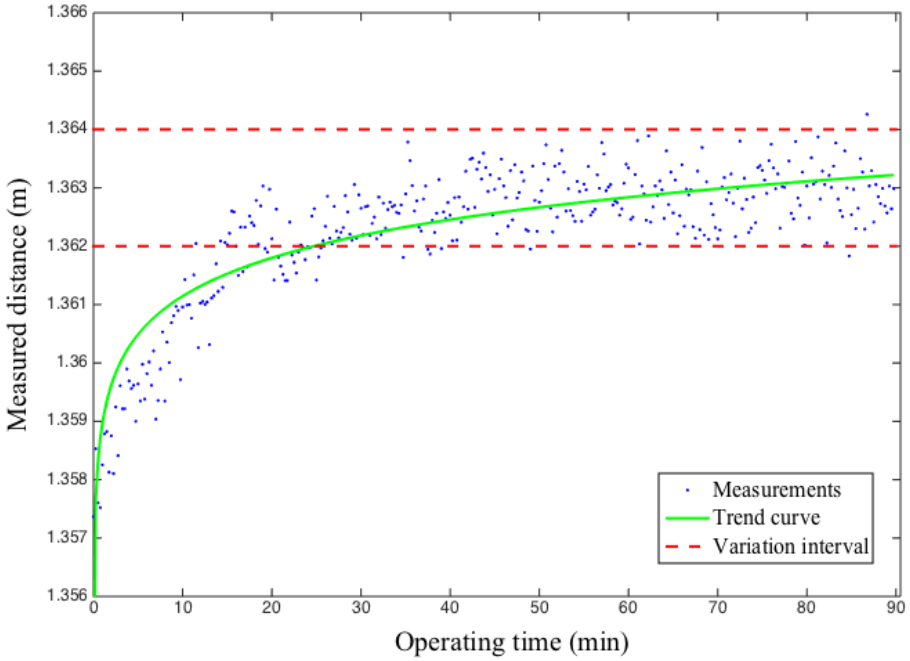


Abbildung 10: Messgenauigkeit als Funktion der Aufwärmzeit [Lac15, S. 96]

2.2.3 Systemanforderungen und Schnittstelle

Für die Entwicklung von Anwendungen mit dem *Kinect for Windows* SDK gibt Windows die folgenden Systemvoraussetzungen an:

- Betriebssystem: Windows 8 (x64), Windows 8.1 (x64), Windows 8 Embedded Standard (x64), Windows 8.1 Embedded Standard (x64), Windows 10 (x64)
- 64-Bit Prozessor, physikalische Doppel-Kern-CPU mit 3.1 GHz oder schneller
- USB 3.0 Controller
- mindestens 4 GB RAM
- DirectX 11 fähige Graphikkarte

Es gibt zwei verschiedene Varianten der Kinect v2, die *Kinect for Xbox One* und die *Kinect for Windows*. In dieser Arbeit wurde die *Kinect for Xbox One* verwendet. Für das Arbeiten unter Windows ist ein Kinect-Adapter erforderlich (s. Abbildung 11).

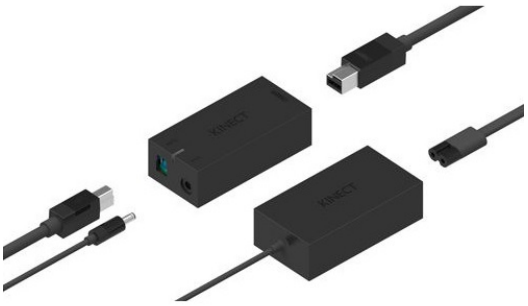


Abbildung 11: Kinect-Adapter [Mic16e]

Die Kompatibilität der Kinect mit dem Computer lässt sich mit dem *Kinect Configuration Verifier* testen. Das Programm untersucht die relevante Hardware auf dem Rechner und testet die Verbindung mit der Kinect. Sind die Vorraussetzungen erfüllt, erscheint ein grüner Haken. Treten eventuell Unstimmigkeiten auf, wie z.B. beim *Universal Serial Bus (USB)*-Controller in Abbildung 12, erscheint ein oranges Ausrufezeichen. Das Bild zeigt das Ergebnis des Kompatibilitätstests des im Projekt verwendeten Rechners. Trotz der angezeigten Einschränkung beim USB-Controller, war die Kompatibilität der Kinect ausreichend gewährleistet.

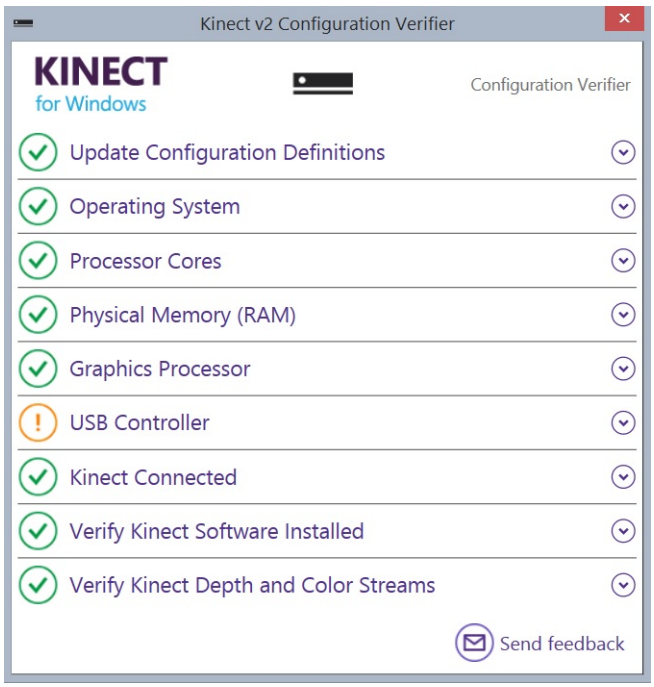


Abbildung 12: Auszug des Kinect-Konfigurationsverifizierers

2.2.4 *Kinect for Windows* SDK und API

Bis zum jetzigen Zeitpunkt ist die Verwendung des Sensors neben der XBox One nur unter Windows möglich. Das *Kinect for Windows* SDK wird benötigt, um Anwendungen mit dem Kinect-Sensor unter Windows zu entwickeln und lässt sich kostenlos von Microsoft erwerben. Es stellt die Treiber, die *Application Programming Interfaces (APIs)*, einige Beispielprogramme und nützliche Software-Tools zur Verfügung. Mit der Verwendung dieses SDKs wird der Zugriff auf alle Funktionen des Kinect-Sensors gewährleistet [Mic16d].

Datenquellen

Die Kinect besitzt zwei physikalische Bildsensoren und ein Mikrofonarray (s. Abbildung 5). Der Farbsensor hat eine Full-HD-Auflösung von 1920×1080 px. Je nach Lichtverhältnissen werden Bilder mit 30 oder 15 fps erfasst. Der IR-Sensor hat eine Auflösung von 512×424 px. Die Erfassung von Objekten ist unabhängig von Bedingungen des Umgebungslichts mit 30 fps möglich. Das Mikrofonarray besteht aus vier Mikrofonen. Die Erfassung aller Datenquellen des Sensors erfolgt unter Verwendung der API.

Zugriff auf Sensordaten Das Vorgehen für den Zugriff auf die Sensordaten ist für jede Datenquelle ähnlich. Der folgende Beispielcode für die einzelnen Schritte der Datenerfassung ist in c++ dargestellt. Die Rückgabewerte der Funktionen werden in die Variable *hr* vom Windowsdatentyp *HRESULT* gespeichert. *hr* hat den Wert *S_OK*, wenn die Funktion erfolgreich war. Andernfalls wird ein Fehlercode zurückgegeben. Als Datenquelle für den Beispielcode werden die Farbdaten gewählt. Ein Ablaufschema für den Zugriff auf die Datenquellen mit der API ist Abbildung 13 zu entnehmen.

Abbildung 13: Zugriff auf Kinect-Sensordaten

Die Bedeutung der einzelnen Datenstrukturen aus der obigen Abbildung und der programmiertechnische Zugriff auf diese werden im Folgenden aufgeführt.

Der **Sensor** entspricht dem physikalischen Kinect-Sensor.
Es wird ein Zeiger auf das *Sensor*-Interface erzeugt:

```
IKinectSensor* pSensor;
```

Der Sensor wird dem Zeiger zugewiesen:

```
hr = GetDefaultKinectSensor(&pSensor);
```

Der Sensor wird geöffnet und das *Streamen* der Sensordaten gestartet:

```
hr = pSensor->Open();
```

Mit der **Source** wird die Auswahl der Datenquelle getroffen und die zur Quelle zugehörigen Metadaten freigelegt.
Ein Zeiger wird auf das Interface der Datenquelle erzeugt:

```
IColorFrameSource* pColorSource;
```

Die Datenquelle wird dem Zeiger zugewiesen:

```
hr = pSensor->get_ColorFrameSource(&pColorSource);
```

Der **Reader** erlaubt den Zugriff auf die Frames (durch *Polling* oder ereignisgesteuert) und unterstützt nur eine einzige Datenquelle.
Ein Zeiger wird auf das *Frame Reader*-Interface erzeugt:

```
IColorFrameReader* pColorReader;
```

Der *Frame Reader* wird mit der *Source* geöffnet:

```
hr = pColorSource->OpenReader(&pColorReader);
```

Der **Frame** erlaubt den Zugriff auf die *Frame*-Metadaten und die Bearbeitung nur eines Frames zur selben Zeit.
Ein Zeiger wird auf das *Frame*-Interface erzeugt:

```
IColorFrame* pColorFramer;
```

Der jüngste Frame wird erworben:

```
hr = pColorReader->AcquireLatestFrame(&pColorFrame);
```

Die **Daten** stellen die Metadaten für die Weiterverarbeitung in der Anwendung dar. Die Verarbeitung der Daten erfolgt direkt oder nach Speicherung in einem Array.

Werden Interfaces nicht mehr benötigt, müssen die jeweiligen Zeiger in umgekehrter Reihenfolge ihres Aufrufs wieder freigegeben werden:

```
pColorFrame->Release();
pColorFrameReader->Release();
pColorSource->Release();
```

Vor dem Beenden der Anwendung sollte auch der Sensor geschlossen werden:

```
pSensor->Close();
pSensor->Release();
```

Die folgenden Datenquellen lassen sich mit der Kinect verarbeiten. Die entsprechenden Sensorbilder wurden mit der jeweiligen Beispielanwendung der SDK aufgenommen.

ColorFrameSource Die Datenquelle *ColorFrameSource* lässt sich in drei verschiedenen Farbformaten (RGBA, GBRA, YUV2) verarbeiten. Das erfasste Bild wird gespiegelt. In Abbildung 14 ist ein Farbbild mit Full-HD-Auflösung zu sehen. Die Bildfrequenzen sind 15 fps bei schlechter und 30 fps bei guter Belichtung [Mic16c].

Abbildung 14: Farbbild in Full HD

InfraredFrameSource Abbildung 15 zeigt eine Aufnahme mit dem IR-Sensor. Sie hat eine Auflösung von 512×424 px. Das Umgebungslicht wird bei den Aufnahmen entfernt. Wärmestrahlung wird nicht erkannt. Die Infrarotdaten werden als 16-Bit *unsigned Integer* gespeichert [Mic16c].

Abbildung 15: Infrarotbild

DepthFrameSource Das Prinzip der Tiefenmessung der Kinect v2 wurde in Unterabschnitt 2.2.2 beschrieben. Tiefenbilder wie in Abbildung 16 haben ebenfalls eine Auflösung von 512×424 px und eine Bildfrequenz von 30 fps. Der stabile Messbereich liegt zwischen 0.5 m und 4.5 m. Distanzmessungen bis 8 m sind möglich, aber die Zuverlässigkeit des Messergebnisses nimmt ab ungefähr 4.5 m ab. Die gemessenen Entfernungen werden als 16-Bit *unsigned Integer* in mm bereitgestellt. Die Messung beginnt ab der Brennebene des Sensors. Da Tiefen- und Infrarotmessung mit dem gleichen Sensor durchgeführt werden, sind die Bilder identisch ausgerichtet. Ein Pixel im Infrarotbild entspricht demselben Pixel im Tiefenbild [Mic16c].

Abbildung 16: Tiefenbild

BodyFrameSource Die Datenquelle für die Körpererkennung enthält alle in Echtzeit berechneten Informationen von Körpern, die im Sichtfeld des Sensors liegen. Der Erkennungsbereich für diese Daten liegt, wie Abbildung 17 zeigt, bei 0.5 m bis 4.5 m. Der effektivste Bereich für die Personenerkennung ist zwischen 1 m und 4 m [Mic16c].

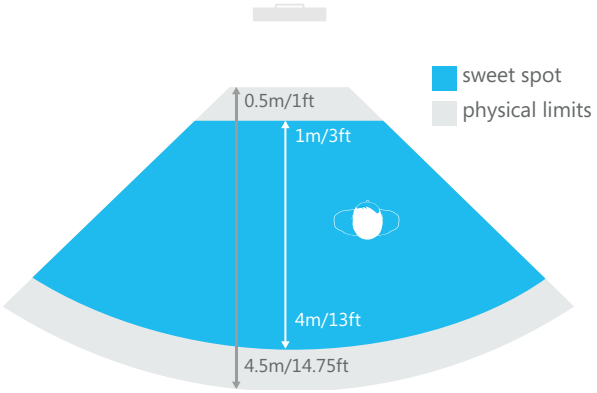


Abbildung 17: Personenerkennungsbereich [Mic14, S.7]

Es können bis zu sechs Personen mit je 25 Körperpunkten bei einer Bildfrequenz von 30 fps gleichzeitig erfasst werden. Abbildung 18 gibt eine Übersicht der möglichen Körperpunkte und ihrer Bezeichnungen. Jeder Punkt hat eine dreidimensionale Position und eine Ausrichtung.

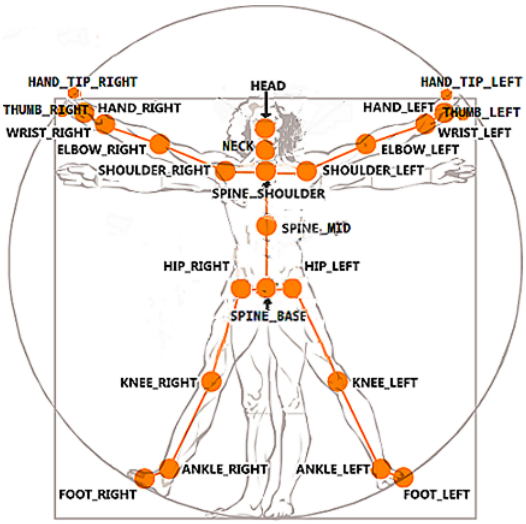


Abbildung 18: Körperpunkte (Joints) [Mic16b]

Von zwei erkannten Körpern kann der jeweilige Handzustand erfasst werden. Es werden fünf Zustände unterschieden (unbekannt, nicht erfasst, offen, geschlossen und „Lasso“) (s. Abbildung 19). Abbildung 20 zeigt die erfassten Körperdaten einer Person.

(a) offen (b) geschlossen (c) „Lasso“

Abbildung 19: Handzustände

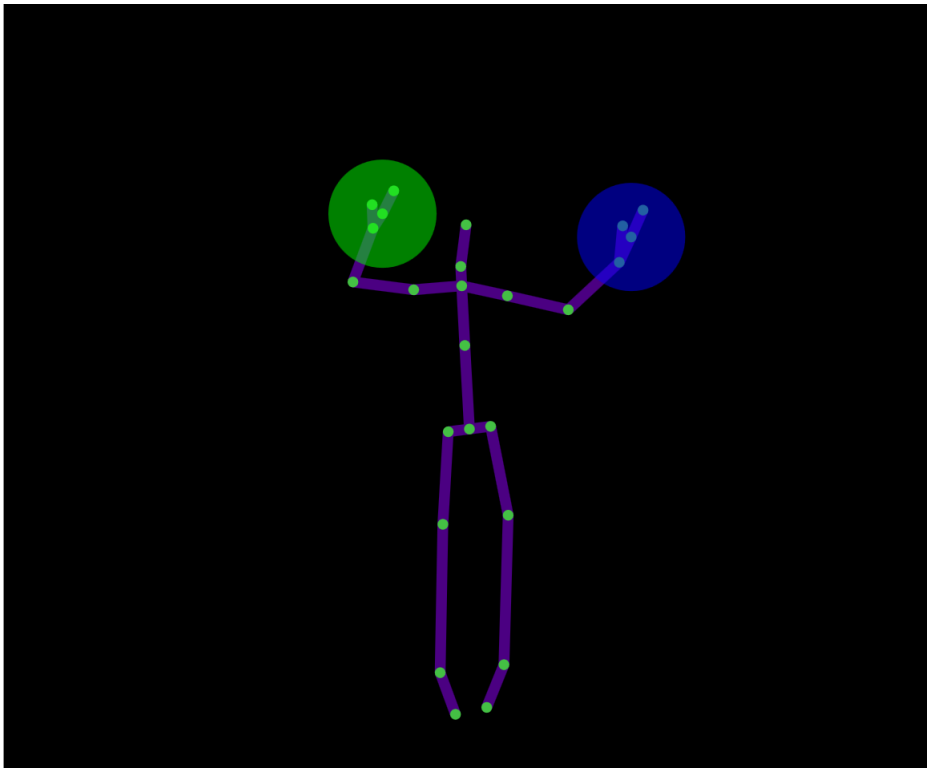


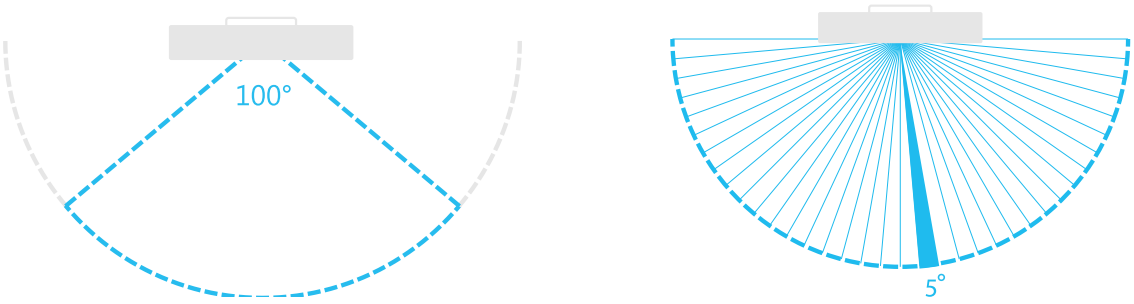
Abbildung 20: Bodytracking

Weiterhin können Informationen für Gesichtsausdrücke von Personen erkannt und verarbeitet werden. Auf diese Funktion wird nicht weiter eingegangen.

BodyIndexFrameSource Die Informationen dieser Quelle basieren auf den berechneten Tiefendaten des Sensors und geben Auskunft über die Beziehung der erfassten Pixel zu Körpern und Hintergrund. Die Pixel sind als 8-Bit *unsigned Integer* abrufbar. Eine Indexvariable wird verwendet, um die Pixel den erfassten Körpern (Index zwischen 0 und 5) oder dem Hintergrund (Index > 5) zuzuordnen. In Abbildung 21 ist ein Body Index dargestellt [Mic16c].

Abbildung 21: Body Index

AudioSource Neben den Bild- können auch Audioquellen mit dem Sensor verarbeitet und für eigene Anwendungen genutzt werden. Abbildung 22a veranschaulicht den Erfassungsbereich für Audioquellen an der Vorderseite des Sensors. Er beträgt ausgehend von der Sensormitte +50° bis −50°. Das Mikrofonarray kann die Richtung einer Audioquelle mit einer Auflösung von 5°-Schritten lokalisieren (s. Abbildung 22b).



(a) Audioeingang Front [Mic14, S. 9] (b) Microfon-Auflösung [Mic14, S. 9]

Abbildung 22: Microfon-Array

Das Mikrofon beinhaltet einen Filter für Umgebungsgeräusche. Eine Reduzierung von bis zu 20 Dezibel (dB) sind möglich und verbessert die Qualität des zu erfassenden Audiosignals. Durch die Bauart des Sensorgehäuses ergibt sich bei Geräuschen, die der Rückseite des Sensors entspringen eine zusätzliche Unterdrückung von 6 dB. Bei der Wahl des Einsatzortes für den Sensor ist zu beachten, dass mit den Voreinstellungen die lauteste Eingangsquelle erfasst wird. Sind die Umgebungsgeräusche lauter als das relevante Signal, so wird der Filter auch auf dieses Signal angewendet. Programmiertechnisch lässt sich die Audioerkennung mit der Personenerfassung kombinieren [Mic14, S. 9 f.]. Dieses Vorgehen ist vorteilhaft für das Setzen von Start- und Endpunkten der Audioerfassung. Auch lassen sich etwaige Sprachbefehle den Personen zuordnen, die sie getätigt haben. Für die

Spracherkennung sollten kurze Befehlsausdrücke mit maximal fünf einsilbigen Worten gewählt werden. Weitere Informationen zum Vorgehen bei der Spracherkennung können der *Human Interface Guideline v2.0* entnommen werden [Mic14, S. 47 ff.].

Die Audiodaten werden in der Applikation für die Robotersteuerung dieser Arbeit nicht verwendet.

Weitere APIs

Neben den Datenquellen beinhaltet das SDK zwei weitere wichtige APIs für die Entwicklung eigener Anwendungen, den *MultiSourceFrameReader* und den *CoordinateMapper*.

MultiSourceFrameReader Alle Datenframes werden vom Sensor zu unterschiedlichen Zeiten zur Verfügung gestellt. Mit der Verwendung des *MultiSourceFrameReaders* lassen sich die gewünschten Frames gleichzeitig abrufen.

Der *MultiSourceFrameReader* wird mit dem Sensor geöffnet. Im folgenden Beispiel werden die Farb- und Bodydaten als Quelltypen freigegeben:

```
hr = pSensor->OpenMultiSourceFrameReader(FrameSourceTypes_Color | FrameSourceTypes_Body,
&pMultiSourceFrameReader);
```

Es ist darauf zu achten, dass bei der Verwendung von Farbframes die Bildfrequenz bei wenig Licht auf 15 fps fallen kann. Das hat zur Folge, dass die gesamte Bildfrequenz des *MultiSourceFrameReaders* ebenfalls auf 15 fps fällt.
Audiodaten werden vom *MultiSourceFrameReader* nicht unterstützt. Wird die Audioquelle dennoch hinzugefügt, schlägt die Funktion *OpenMultiSourceFrameReader()* fehl [Mic16c].

CoordinateMapper Für die Datenquellen des Sensors gibt es drei verschiedene Koordinatensysteme, *ColorSpace*, *DepthSpace* und *CameraSpace*. Der Tabelle 2 sind die Eigenschaften der Koordinatensysteme zu entnehmen.

Name	*gilt für*	*Dimension*	*Einheit*	*Bereich*	*Ursprung*
ColorSpace	Color	2	px	1920×1080	oben links
DepthSpace	Depth, Infrared, Body Index	2	px	512×424	oben links
CameraSpace	Body	3	m	-	Infrarotsensor

Tabelle 2: Vergleich der Kinect-Koordinatensysteme [Mic16a]

In Abbildung 23 ist die Ausrichtung des dreidimensionalen *CameraSpace*-Koordinatensystems zu erkennen. Das Systems ist als rechtshändig definiert und hat seinen Ursprung im Infrarotsensor.

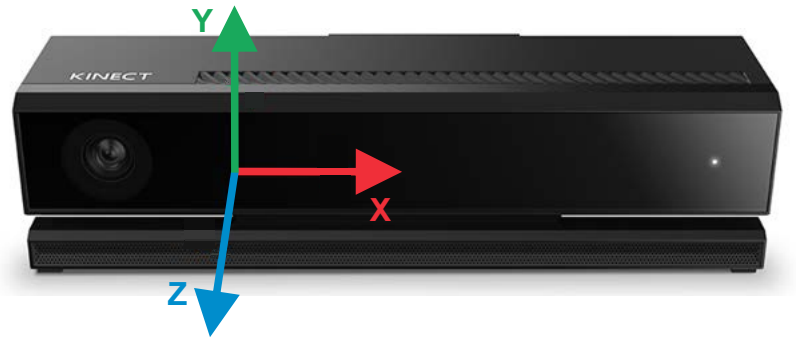


Abbildung 23: CameraSpace-Koordinatensystem der Kinect v2 [Mic16a]

Mit dem *CoordinateMapper* lassen sich einzelne oder mehrere Punkte gleichzeitig aus einem in ein anderes Koordinatensystem umwandeln. Mit der Anwendung der Umwandlungsfunktionen werden z.B. die Bodyjoints einer erfassten Person in ein Farbbild überführt:

```
pCoordinateMapper->MapCameraPointToColorSpace(joint.Position, &colorSpacePoint);
```

[Mic16c]

Kinect-Tools

Das Kinect SDK stellt zwei Tools zur Verfügung, das *Kinect Studio* und den *Visual Gesture Builder (VGB)*.

Kinect Studio Diese Software dient der Ansicht der vom Sensor erfassten Datenströme und zum Aufnehmen und Wiedergeben von Videoclips im *eXtended Event File (XEF)*-Format. Die Auswahl von 2D oder 3D Ansichten ist möglich. Alle verfügbaren Datenquellen können aufgenommen werden. Der Nutzer kann eine Auswahl der gewünschten Quellen im Programm vornehmen. Durch die Verwendung des *Kinect Studios* werden Test- und Debugging-Vorgänge vereinfacht. Videoclips im XEF-Format werden vor allem bei der Entwicklung von Anwendungen mit dem VGB gebraucht. Die Verwendung des *Kinect Studios* ist dabei unverzichtbar [Mic16d].

Visual Gesture Builder Mit dem *Visual Gesture Builder* können Gestendatenbanken entwickelt werden, die in einer Applikation für die Gestenerkennung in Echtzeit Anwendung finden. In den mit *Kinect Studio* aufgenommenen XEF-Videoclips lassen sich Gesten durch Markierung bestimmen. Die XEF-Daten müssen dafür mindestens Tiefen- und Skelettinformationen enthalten. Der VGB nutzt zur Gestenerkennung Algorithmen für maschinelles Lernen. In Abbildung 24 ist der Entwicklungsprozess für einen Gestendetektor mit dem VGB zu sehen [Mic16d], [Mic13].

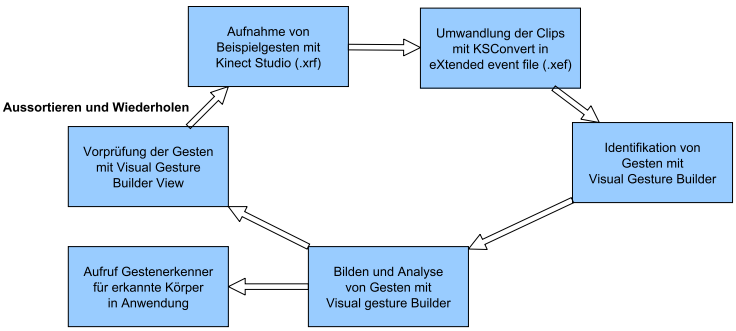


Abbildung 24: Datengesteuerter Entwicklungsprozess für einen Gestendetektor mit VGB

Der *Visual Gesture Builder* findet in diesem Projekt keine Anwendung und wird nicht näher beschrieben.

2.3 Mathematische Berechnungen

Für die Berechnung der Roboterkoordinaten sind Transformationen im dreidimensionalen Raum notwendig. Im Folgendem wird ein mathematischer Überblick über die Problemstellung und Berechnungsmethoden gegeben. Bei den betrachteten Systemen handelt es sich um Rechtssysteme bzw. rechtshändige Koordinatensysteme. Die Notation erfolgt in kartesischen Koordinaten.

2.3.1 Geometrische Transformationen

Punkte eines Koordinatensystems lassen sich durch geometrische Transformationen in ein anderes Koordinatensystem überführen. Die wesentlichen Transformationen sind:

- Rotation
- Skalierung
- Spiegelungen
- Translation
- Scherungen
- Projektionen

Bis auf die Translation sind alle Transformationen linear. Sie lassen sich durch die Multiplikation eines Vektors $\boldsymbol{x} \in \mathbb{R}^n$ mit einer Matrix $A \in \mathbb{R}^{n \times n}$ mit

$$\boldsymbol{x}' = A\boldsymbol{x} \tag{4}$$

darstellen. In der Robotik sind nur die Rotation (Drehung) und die Translation (Verschiebung) von Interesse [Glo16, S. 4].

Rotation

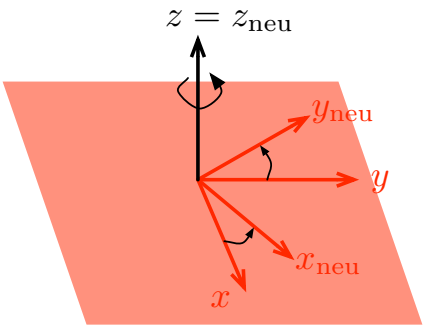


Abbildung 25: Rotation um die z-Achse [Glo16, S. 6]

In Abbildung 25 wird eine Rotation um die z-Achse im $\mathbb{R}^3$ veranschaulicht. Die Drehachse bleibt bei der Drehung konstant. Sie steht senkrecht auf der Drehebene, die in diesem Fall von der x- und der y-Achse aufgespannt wird.
Rotationen um die Koordinatenachsen werden im $\mathbb{R}^3$ durch die folgenden Matrizen dargestellt:

$$Rot_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{pmatrix} \quad Rot_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix}$$

$$Rot_z(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Mit ihnen lässt sich ein Vektor im euklidischen Raum um einen Winkel θ drehen. Bei einem positiven Winkel erfolgt die Drehung gegen den Uhrzeigersinn [Wüs16, S. 52 f.].

Translation

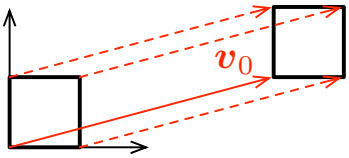


Abbildung 26: Translation mit $\boldsymbol{v}_0$ [Glo16, S. 19]

Eine Translation mit einem Verschiebungsvektor $\boldsymbol{v}_0$ ist in Abbildung 26 dargestellt. Sie ist nicht linear und durch eine Vektoraddition

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{v}_0$$

gegeben. Eine 3×3-Matrix kann eine Translation im $\mathbb{R}^3$ nicht beschreiben. Es werden homogene Koordinaten benötigt [Glo16, S. 19].

2.3.2 Homogene Koordinaten und Transformation

Bei homogenen Koordinaten wird ein n-dimensionaler Raum mit $n+1$ Dimensionen dargestellt. Aus (x, y, z) wird für $n = 3$ (hx, hy, hz, h), wobei h eine beliebige Zahl ist. In der Robotik wird $h = 1$ gewählt. Das entspricht einer direkten Projektion bei der Abbildung vom n- in den $n+1$-dimensionalen Raum.

Im $\mathbb{R}^3$ wird eine 4×4-Transformationsmatrix T als Blockmatrix mit folgenden Dimensionen definiert:

$$T = \left(\begin{array}{ccc:c} 3 & & & 3 \\ & \times & & \times \\ & & 3 & 1 \\ \hdashline 1 & \times & 3 & 1 \times 1 \end{array}\right) \tag{6}$$

Die Rotation und Translation lassen sich den Blöcken zuordnen:

$$T = \left(\begin{array}{ccc:c} & \text{Rotation} & & \text{Translation} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right) \tag{7}$$

Eine Verschiebung durch den Vektor $\boldsymbol{v} = (v_x, v_y, v_z)^T$ lässt sich dann darstellen mit:

$$Trans(v_x, v_y, v_z) = \begin{pmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

Die Rotationen um die Koordinatenachsen aus Gleichung 5 werden durch folgende homogene Matrizen beschrieben:

$$Rot_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Rot_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

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

[Zha16, S. 321 ff.]

2.3.3 Aufeinanderfolgende Transformation

Wird ein Vektor $\boldsymbol{v}_1$ mehrfach transformiert, wiederholt sich der Vorgang wie folgt:

$$\boldsymbol{v}_2 = T_1 \boldsymbol{v}_1 \tag{10}$$

$$\boldsymbol{v}_3 = T_2 \boldsymbol{v}_2 = T_2 T_1 \boldsymbol{v}_1 \tag{11}$$

Die Zusammenwirkung der beiden Transformationen T_1 und T_2 lässt sich als Matrizenprodukt

$$T_{ges} = T_2 T_1 \tag{12}$$

darstellen. Matrizen sind nicht kommutativ. Folglich ist zu beachten, welche Transformation zuerst angewendet werden soll. In diesem Fall wird erst T_1 und dann T_2 ausgeführt [Wüs16, S. 54].

2.3.4 Bezugssysteme und Koordinatentransformation

In der Robotik ist die Verwendung von Bezugssystemen üblich. Unter Bezugssystemen versteht man die Verwendung mehrerer Koordinatensysteme für die Beschreibung von Roboterteilen, Werkstücken und anderen Objekten. Mit der Verwendung von Bezugssystemen können Zeitabhängigkeiten beseitigt, an das Problem angepasste Koordinatenachsen gelegt und somit viele Probleme einfacher formuliert werden [Wüs16, S. 62].
Auch bei der Positionsbeschreibung von Objekten durch die Verwendung mehrerer Koordinatensysteme spricht man von Bezugssystemen. Die Bezugssysteme gehen aus der Transformation anderer Koordinatensysteme hervor.
Ein links hochgestellter Index gibt an, auf welches Bezugssystem sich eine Koordinatenangabe bezieht, z.B. ist ${}^{A}P$ der Punkt P in Koordinaten des Bezugssystems A und ${}^{B}P$ der Punkt P in Koordinaten des Bezugssystems B.
Eine Überführung eines Koordinatensystems A in ein Koordinatensystem B geschieht durch die Transformationsmatrix ${}^{A}T_{B}$. Durch die Transformationsmatrix werden ebenfalls die Punkte aus der Darstellung in B-Koordinaten in die Darstellung in A-Koordinaten transformiert. Es gilt:

$$K_B = {}^{A}T_{B} K_A \tag{13}$$

$${}^{A}P = {}^{A}T_{B} {}^{B}P \tag{14}$$

Die inverse Transformationsmatrix $\left({}^{A}T_{B}\right)^{-1}$ transformiert Koordinatensystem B in Koordinatensystem A und die Darstellung der Punkte von A-Koordinaten in B-Koordinaten:

$$K_A = \left({}^{A}T_{B}\right)^{-1} K_B \tag{15}$$

$${}^{B}P = \left({}^{A}T_{B}\right)^{-1} {}^{A}P \tag{16}$$

Aus Gleichung 16 ist die Koordinatentransformation ersichtlich. Es gilt:

$$\left({}^{A}T_{B}\right)^{-1} = {}^{B}T_{A} \tag{17}$$

[Wüs16, S. 65 f.]

2.4 Entwicklungsumgebung und Programmiersprache

Das Programm der Robotersteuerung mit der Kinect wird unter Windows entwickelt. Als Entwicklungsumgebung wird das ebenfalls von Microsoft entworfene *Visual Studio 2013 Professional* gewählt.

Als Programmiersprache wird c++ verwendet. Sie lässt sich auf einer vielseitigen und hohen Abstraktionsebene verwenden und ist dennoch effizient und maschinennah. Weiterhin ist die Verwendung von anderen offenen Klassenbibliotheken für das plattformübergreifende Programmieren von großem Vorteil. Die beiden in diesem Projekt verwendeten Klassenbibliotheken werden im nächsten Abschnitt vorgestellt.

2.5 *OpenCV* und *Qt*

Für die Entwicklung der Robotersteuerung werden zwei externe Bibliotheken verwendet, *Open Source Computer Vision (OpenCV)* und *Qt*.

OpenCV *OpenCV* ist eine offene und plattformunabhängige Programmbibliothek. Ihre Stärke liegt in der Verarbeitungsgeschwindigkeit, insbesondere bei Anwendungen in Echtzeit. Mit der verwendeten Version 2.4.13 kommt eine c++ API zum Einsatz, die Bibliotheken für die Bildverarbeitung und Videoanalyse, Kamerakalibrierung, maschinelles Sehen, Objekt- und Bewegungserkennung, Filterung und viele weitere Anwendungen enthält. Eine detaillierte Onlinedokumentation beinhaltet eine Übersicht über die API-Referenz, ein Benutzerhandbuch und eine Reihe von Tutorien für die Anwendung grundlegender Funktionen von *OpenCV* [Ope16].

Qt Die plattformübergreifende c++ Klassenbibliothek *Qt* kann mit den gängigen Betriebssystemen verwendet werden. Mit ihr lassen sich z.B. grafische Benutzeroberflächen erstellen, Scriptsprachen wie *Qt Meta-Object Language (QML)* und *JavaScript* verarbeiten, Datenbanken mit *Structured Query Language (SQL)* integrieren und Netzwerkanwendungen programmieren. Auch für die in Abschnitt 2.3 vorgestellten geometrischen Transformationen gibt es Klassenelemente, die die Implementierung der Berechnungen deutlich vereinfachen können.

Qt-Anwendungen können mit dem *Qt Creator*, einer von *Qt* entwickelten, plattformübergreifenden und vollständig integrierten Entwicklungsumgebung entworfen werden. Bei der Verwendung von *Qt* mit *Visual Studio* wird ein *Qt Visual Studio Add-in* zur Verfügung gestellt. Das Add-in funktioniert ab *Visual Studio* 2008, aber nicht mit den *Express*-Editionen. Das Add-ins enthält den *Qt Designer*, mit dem sich grafische Benutzeroberfläche erstellen lassen.
Ein wichtiges und zentrales Element von *Qt* ist das *Signal-Slot*-Konzept. Mit der Anwendung von Signalen und Slots lässt sich eine ereignisgesteuerte Kommunikation zwischen Objekten realisieren. Ein Signal wird von einem Objekt gesendet, wenn eine interne Statusänderung dieses Objektes erfolgt, die für ein anderes Objekt von Interesse ist. Ein Slot wird gerufen, wenn ein mit ihm verbundenes Signal gesendet wurde. Bis auf die Verbindung mit Signalen sind Slots normale c++ Funktionen und können auch standardmäßig aufgerufen werden. Es können beliebig viele Signale mit einem einzigen Slot und ein Signal mit beliebig vielen Slots gekoppelt werden. Verglichen mit der Verwendung von Funktionszeigern bzw. *Callbacks*, sind Signale und Slots geringfügig langsamer. In der Praxis ist dieser Aspekt aber kaum relevant. Die Vorteile des Konzeptes liegen bei der einfachen und flexiblen Verwendung und seiner Typ- und Threadsicherheit. [Qt 16b]. Auch für *Qt* gibt es eine sehr ausführliche Onlinedokumentation [Qt 16a].

3 Implementierung

Für die Evaluierung der Kinect zur Gestensteuerung des UR5-Roboterarmes wird eine Applikation entwickelt. Das Ziel dieser Applikation ist die Bewegung des Roboterarmes an eine vom Nutzer vorgegebene Postion. Diese Zielposition wird durch die linke Hand des Nutzers mit einer Faustbewegung markiert. Die markierten Zielkoordinaten werden vom Kinect-Sensor erfasst und über geometrische Transformationen in Roboterbasiskoordinaten überführt.
Auf dem roboterinternen Rechner wird ein Script ausgeführt, welches eine Anfrage für die Roboterzielkoordinaten an die Testapplikation sendet. Bei erfolgreicher Anfrage übermittelt ein Server der Applikation die angefragten Koordinaten zurück an das Script. Mit den vorhandenen Zielkoordinaten wird der Roboter über ein Bewegungsbefehl im Script an die Zielposition gefahren. Hat der Roboter seine Zielposition erreicht, kann der Nutzer neue Roboterzielkoordinaten mit der linken Hand vorgeben.
Der Aufbau der Testumgebung sowie die Realisierung der Applikation werden im Folgenden beschrieben.

3.1 Aufbau

Für die Robotersteuerung werden der Kinect v2-Sensor für die Erfassung der Bilddaten und der UR5-Roboterarm für die Ausführung der Bewegungsbefehle verwendet. Der Roboter ist auf einer Werkbank befestigt und seine Postion unveränderbar. Die Kinect wird auf ein Kamerastativ montiert und lässt sich beliebig innerhalb der Testumgebung positionieren. Für den erfolgreichen Ablauf der Anwendung muss aber eine feste Position für den Sensor definiert werden. Weitere Informationen dazu sind in Unterabschnitt 3.1.2 gegeben.
Aus Sicherheitsgründen wird der Bewegungsspielraum des Roboters vordefiniert. Der Erfassungsbereich der linken Hand des Nutzers muss somit am Rand dieses Bewegungsraumes begrenzt werden. In Abbildung 27 ist eine abstrakte Darstellung des Aufbaus der Testumgebung zu sehen.

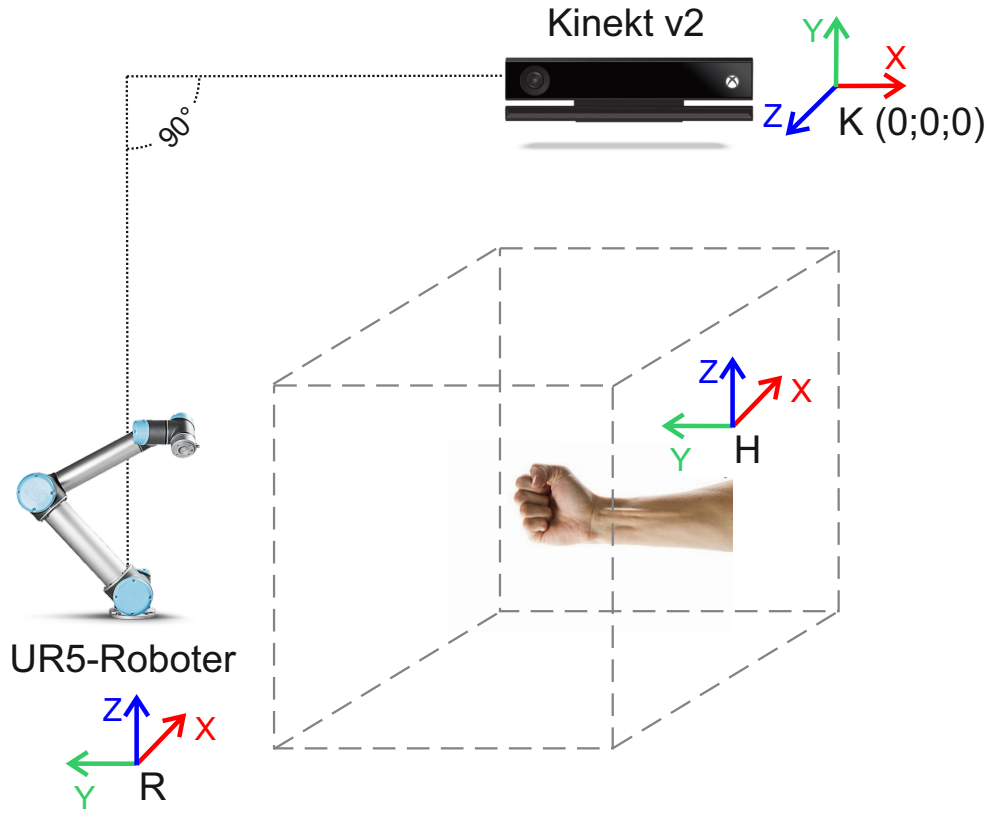


Abbildung 27: Testaufbau der Robotersteuerung

3.1.1 Bezugssysteme

Der Roboter und die Kinect besitzen individuelle räumliche Koordinatensysteme, die als geometrische Rechtssysteme definiert sind. Diese Koordinatensysteme sind Bezugssysteme zu den jeweils anderen Koordinatensystemen. Die Position des Fußflansches (s. A in Abbildung 2) wird relativ zu den Weltkoordinaten als Roboterbasiskoordinatensystem R definiert. Die Position des Handgelenkes 3 (s. F in Abbildung 2) wird relativ zu den Roboterbasiskoordinaten als Handkoordinatensystem H definiert. Die Koordinatenachsen des Handkoordinatensystems haben die gleiche Ausrichtung wie die Achsen des Roboterbasiskoordinatensystems. Die Berechnung der Handkoordinaten in Bezug auf die Roboterbasis erfolgt über die kinematische Kette der Robotergelenke. Die Koordinaten des Handkoordinatensystems H entsprechen den mit der Kinect erfassten Roboterzielkoordinaten der linken Hand des Nutzers aus Sicht der Roboterbasis.
Der Kinect-Sensor wird als Datenquelle für die Gestensteuerung genutzt. Das *CameraSpace*-Koordinatensystem der Kinect wird deshalb mit dem Namen Kamerakoordinatensystem K als Weltkoordinatensystem definiert. Die Position des IR-Sensors ist

dann der definierte Ursprung des gesamten Systems mit den Koordinaten $(0, 0, 0)$. Der Roboter bewegt sich an Raumpunkte, dessen Bezugssystem die Roboterbasis ist. Die vorgegebene Zielposition des Nutzers beinhaltet Koordinaten, die dem Kamerakoordinatensystem zugeordnet sind. Durch die unterschiedliche Definition der beiden Systeme, muss eine geometrische Transformation dieser Zielposition durchgeführt werden. Die Transformation muss vom Kamera- in das Roboterbasiskoordinatensystem erfolgen. Für die Berechnung der Transformation ist die Bestimmung der Roboterbasisposition erforderlich. Die Positionsbestimmung geschieht mit Hilfe des Nutzers. Vor der Erfassung der Zielposition, wird der Benutzer aufgefordert durch eine Faustbewegung der linken Hand die Position des Fußflansches zu markieren.
Damit das Gesamtsystem einen festen Ursprung besitzt, muss der Kinect-Sensor fest positioniert werden.

3.1.2 Positionierung der Kinect v2

Wie schon erwähnt wurde, ist die Position der Kinect innerhalb der Testumgebung frei wählbar. In den geometrischen Berechnungen der Zielkoordinaten werden für die Rotation der Bezugssysteme feste Werte der Winkel vorausgesetzt. Die Entfernung der für die Berechnung benötigten Position der Roboterbasis muss ebenfalls eindeutig und stabil sein. Eine Positionierung der Kinect im rechten Winkel zum Roboter wie in Abbildung 27 zu sehen, wird angestrebt. Da der Nutzer die linke Hand für die Interaktion benutzt, fällt die Sicht der Kamera auf die Körpervorderseite. Probleme bei der Erfassung der Körperpunkte werden somit minimiert. Die Höhe der Kamera orientiert sich an der Höhe der Roboterbasis. Auch der Neigungswinkel des Kamerastativs muss beachtet werden. Das Stativ muss waagerecht ausgerichtet sein.

3.1.3 Bewegungsbereich des Roboters

Der Interaktionsbereich mit dem Roboterarm wird aus sicherheitstechnischen Gründen relativ klein gehalten. Aber auch für die Ausführung des Bewegungsbefehls macht die Begrenzung Sinn. Eine inverse Kinematik muss angewendet werden, um die Winkelstellungen der Robotergelenke zu berechnen. Wird der Bewegungsspielraum nicht begrenzt und eine Zielpose vorgegeben, die außerhalb der Erreichbarkeit des Roboterarmes liegt, wird für die inverse Kinematik keine Lösung gefunden. Das Programm für die Steuerung bricht ab. Diesem Problem lässt sich mit der Vorgabe von räumlichen Grenzen vorbeugen. Der Bewegungsbereich für die jeweiligen Achsen im Roboterbasiskoordinatensystem sind:

- x: $-0.3\,\mathrm{m}$ bis $0.3\,\mathrm{m}$
- y: $-0.82\,\mathrm{m}$ bis $-0.31\,\mathrm{m}$
- z: $0.39\,\mathrm{m}$ bis $0.82\,\mathrm{m}$

3.2 Realisierung der Gestensteuerung

Für die Umsetzung der Gestensteuerung des Robterarmes UR5 werden zwei parallel laufende Programme benötigt. Das erste Programm ist ein Script (*kinectComm_movej*), welches auf dem roboterinternen Rechner läuft. Dieses Script erfragt die benötigten Koordinaten für die Zielpose des Roboters und führt nach Erhalt dieser Koordinaten den Bewegungsbefehl für den Roboter an diese Zielpose aus. Das zweite Programm ist eine Anwendung (*KinectRobotControl*), die die Bilddaten mit dem Kinect-Sensor erfasst, die gewonnenen Daten geometrisch transformiert und die neu berechneten Daten über einen Server an das Roboterscript versendet.

3.2.1 Roboterscript *kinectComm_movej*

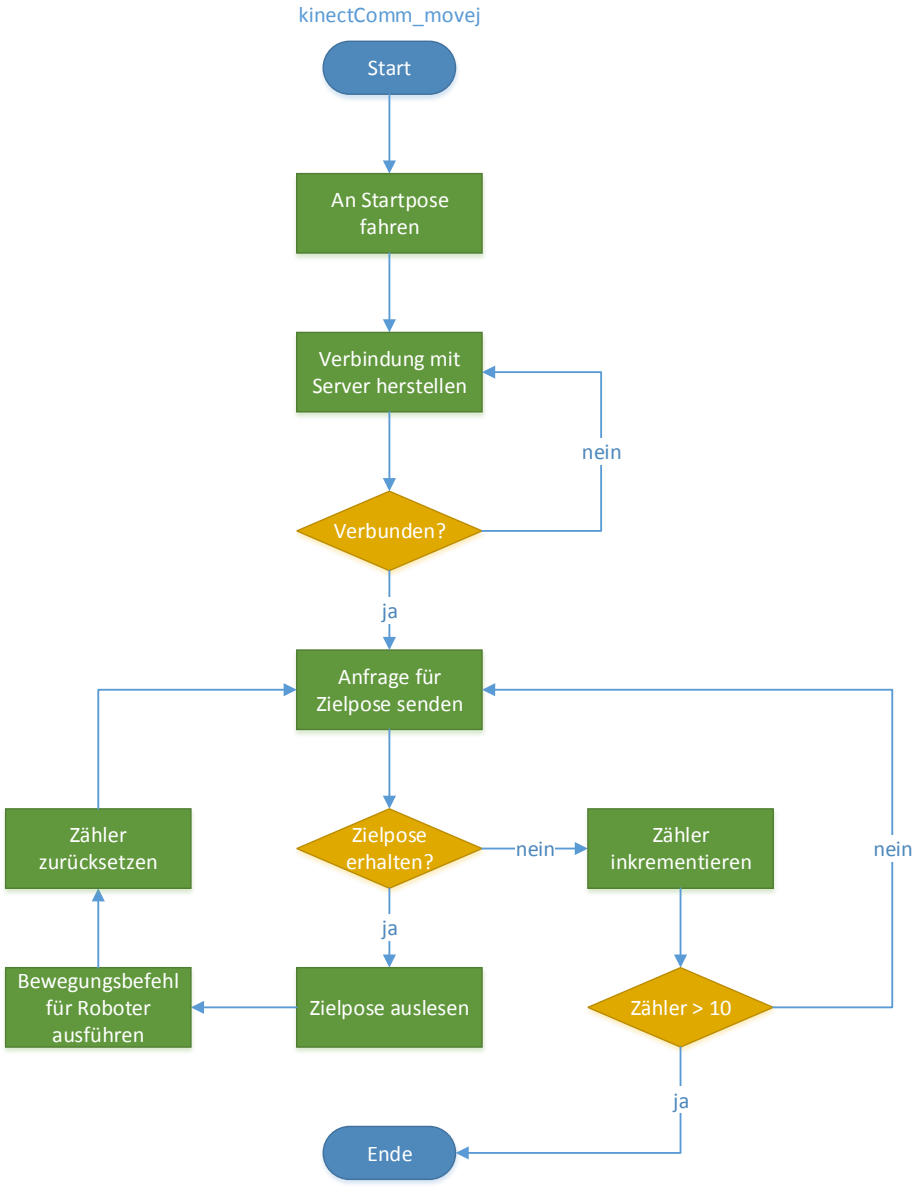


Abbildung 28: Programmablauf Roboterscript

Das Roboterscript *kinectComm_movej* wird in der Programmiersprache *URScript* geschrieben. In Abbildung 28 ist der Programmablauf zu sehen. Das Programm wird aus der Programmieroberfläche der *PolyScope*-Umgebung gestartet. Nach dem Start wird der Roboter an eine definierte Startposition gefahren. Ist die Startposition erreicht, wird versucht eine Verbindung zum Server der *KinectRobotControl*-Anwendung aufzubauen. Wenn die Verbindung steht, sendet das Programm eine Anfrage für eine Zielpose.
Ist die Anfrage nicht erfolgreich, wird ein Zähler inkrementiert. Das Programm wird beendet, wenn mehr als zehn aufeinanderfolgende Anfragen ohne Erhalt der angefor-

derten Zielpose abgeschlossen wurden. Bei erfolgreicher Anfrage, wird die Zielpose in Form eines Strings vom Server der *KinectRobotControl* an das Script gesendet. Der String enthält sechs durch Kommata getrennte *Float*-Werte. Die Werte beschreiben eine Pose mit den Positionskoordinaten (x, y, z) und der Orientierung $(\phi x, \phi y, \phi z)$ des *Tool Center Points (TCPs)*. Da es sich bei der Zielpose um kartesische Koordinaten handelt, wird innerhalb des Bewegungsbefehls *movej* die Funktion *get_inverse_kin* aufgerufen. Mit dieser Funktion werden die Winkelstellungen der einzelnen Robotergelenke für die Zielpose berechnet. Die Funktion für die Berechnung erhält als Parameter neben der eigentlichen Zielpose eine weitere Pose, anhand derer sich die Lösung der inversen Kinematik orientieren soll (s. [Uni15c, S. 14, 23]). Bei erfolgreicher Lösung, wird der Roboter an die Zielposition gefahren und der Zähler zurückgesetzt. Ist die Zielposition mit dem Roboterarm erreicht, startet die Anfrage einer neuen Zielpose erneut.

3.2.2 Gestensteuerung *KinectRobotControl*

Das Programm *KinectRobotControl* ist eine *Qt* Application. Die Entwicklung dieser Applikation umfasst das Design einer Benutzeroberfläche und die Programmierung des Kinect-Interfaces, einer Ablaufsteuerung für die Berechnung der geometrisch korrekten Zielpose sowie eines Servers für die Übermittlung der Zielpose an das Roboterscript. Die Anwendung ist ereignisgesteuert und die Bearbeitung dieser Ereignisse wird mit dem *Signal-Slot*-Konzept von *Qt* umgesetzt. In der Anwendung laufen drei Threads parallel, die *QDialog::exec()*-Funktion, der Gerätetreiber der Kinect und der *Qt*-Server. Die Funktion *QDialog::exec()* startet ein modales Dialogfenster für die Kommunikation mit dem Benutzer. Typischerweise wird die Beendigung des Dialogfensters mit dem Drücken eines Buttons verknüpft [Qt 16a]. In dieser Anwendung ist das der *Exit Application*-Button. Es werden weitere Buttons für den Dialog mit dem Benutzer verwendet. Der Aufbau des GUIs wird im Folgenden beschrieben.

Graphical User Interface

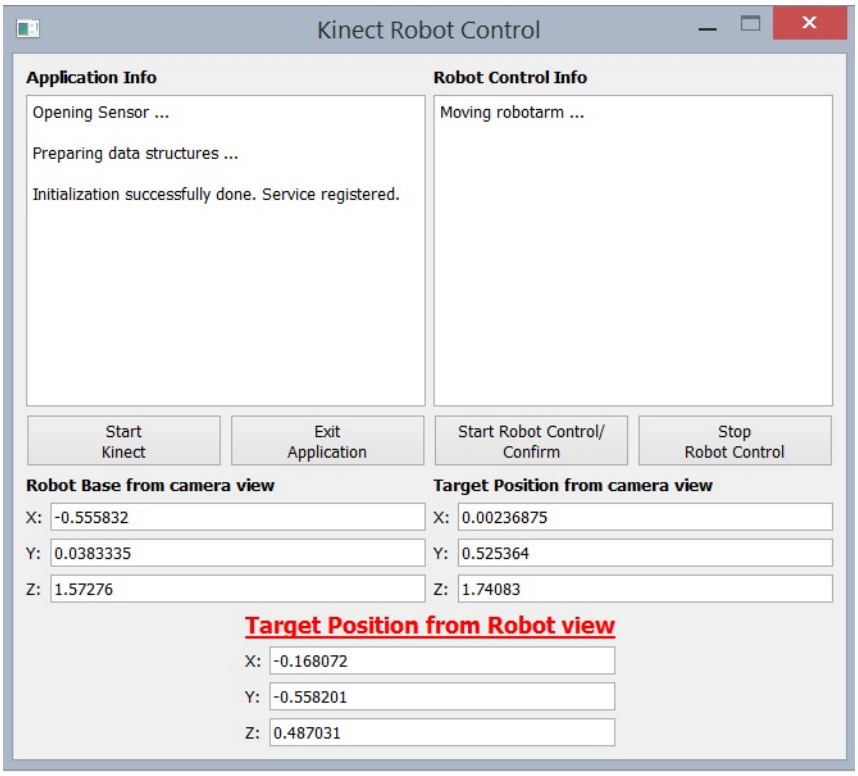


Abbildung 29: Oberfläche des *Graphical User Interfaces*

Das *Graphical User Interface (GUI)* unterstützt den Nutzer der Roboter-Gestensteuerung bei der Bedienung des Programms. In Abbildung 29 ist das GUI der Steuerung zu sehen. Durch den Button *Start Kinect* wird die Kinect gestartet, initialisiert und die Frames werden in einem weiteren Fenster dargestellt. Informationen und Fehlermeldungen bei der Initialisierung oder der Erwerbung der Frames werden im Textfeld *Application Info* ausgegeben. Ist die Kinect betriebsbereit, kann durch Drücken des Buttons *Start Robot Control* der Automat und somit die eigentliche Robotersteuerung gestartet werden. Der Button dient weiterhin als Eingabe für Bestätigungsanfragen während der Steuerung. Die Anfragen und andere Informationen erscheinen auf dem Textfeld *Robot Control Info*. Nach der Erfassung der Roboterbasis werden die Koordinaten unter *Robot Base from camera view* ausgegeben. Die mit der Kinect erfassten Zielkoordinaten sind unter *Target Position from camera view* und die transformierten Koordinaten, die letztendlich das Roboterziel markieren, unter *Target Position from Robot view* zu sehen. Der Automat lässt sich jederzeit mit dem Knopf *Stop Robot Control* beenden. Die Aufnahme und Ausgabe der Kinect-Frames läuft weiter und der Automat kann erneut gestartet werden. Erst durch Betätigung des Buttons *Exit Application* wird der Server gestoppt, alle Interfaces der Kinect wieder freigegeben und die Anwendung sauber geschlossen.

Auswahl der Datenquellen und Erwerb der Kinect-Frames

Nachdem das Dialogfenster mit dem GUI erscheint, wird die Kinect vom Benutzer durch einen Knopfdruck gestartet. Bevor die Framedaten der Kinect abgerufen werden können, muss der Sensor geöffnet und die gewünschten Datenquellen ausgewählt werden. Abbildung 30 zeigt den Programmablauf nach dem Start der Kinect über die grafische Bedienoberfläche.

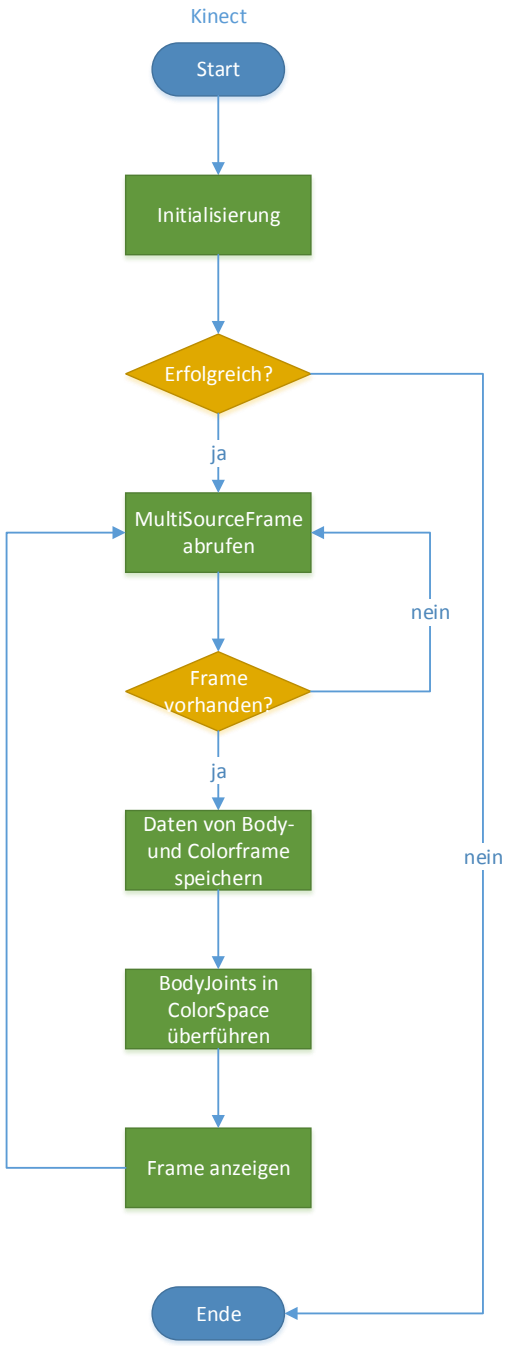


Abbildung 30: Programmablauf Kinect

Während der Initialisierung werden der Sensor geöffnet und die einzelnen Schritte für den Zugriff auf die Datenquellen, wie in Abschnitt 2.2.4 beschrieben, durchgeführt. Als Datenquellen werden die *BodyFrameSource* und die *ColorFrameSource* gewählt. Die Datenquellen werden nicht einzeln, sondern mit dem *MultiSourceFrameReader* verwaltet. Mit den Daten des *BodyFrames* werden die Gelenkpunkte, die sogenannten *Joints*, erfasst. Aus diesen Körperdaten lässt sich die Position und der Zustand, der für die Gestensteuerung benötigten linken Benutzerhand, auswerten und für die Weiterverarbeitung nutzen. Mit der Verarbeitung der *Colorframes* hat die steuernde Person die Möglichkeit, die Bewegungen und die Erfassung der Positionsdaten in einer Bildausgabe zu verfolgen.
Der Abruf eines Frames soll ereignisgesteuert ablaufen. Es wird ein *Event-Handler* für die neuen, abrufbaren Frames eingerichtet. Mit dem *QWinEventNotifier* von *Qt* können Wartefunktionen von Windows genutzt und ein *Handle* mit einem Ereignis verknüpft werden. Steht ein neuer Frame zur Verfügung, wird solch ein Ereignis ausgelöst und der *QWinEventNotifier* sendet eine Benachrichtigung über den Eintritt dieses Ereignisses.
Ist die Initialisierung fehlgeschlagen, weil z.B. der Sensor nicht mit dem USB-Controller verbunden ist oder keine Frames bereitgestellt werden können, muss die Anwendung mit *Exit Application* beendet und neu gestartet werden.
Nach erfolgreicher Initialisierung des Sensors werden die Frames mit dem *Event-Handler* abgerufen. Die Daten des abgerufenen Frames werden zwischengespeichert. Die Farbdaten werden in einem *Mat*-Pufferarray von *OpenCV* und die Körperdaten in einer global erzeugten Personenstruktur gespeichert.

```
// Struct of Body interface information
struct Person
{
        UINT64 trackingID;
        JointOrientation orientation[JointType_Count];
        Joint joint[JointType_Count];
        HandState handStateLeft;
        TrackingConfidence confidenceLeftHand;
        HandState handStateRight;
        TrackingConfidence confidenceRightHand;
};
```

Die Struktur enthält einen Identifikator (ID) der erfassten Person, Informationen über die *Joints* und deren Orientierung sowie die Zustände beider Hände. Da für die gestische Interaktion mit dem Roboterarm nur eine Person notwendig ist, wird auch nur eine Person erfasst. Dabei wird die erste im Sichtfeld der Kinect erkannte Person gewählt. Sind mehrere Personen im Sichtfeld der Kamera und die erfasste Person verlässt den Sichtbereich, wird eine andere verbleibende Person gewählt.
Im nächsten Schritt werden die *Joint*-Positionen aus dem *CameraSpace* der Kinect mit dem *CoordinateMapper* in den *ColorSpace* überführt (s. Abschnitt 2.2.4). Mit *OpenCV* wird ein farbiger Kreis um die Position der linken Hand gezeichnet und ebenfalls

in einem *Mat-Image Container* gespeichert. Wurde der Zustand der Hand nicht erfasst oder ist er nicht identifizierbar, wird der Kreis rot. Eine offene Hand oder mit dem Zustand „Lasso“ wird mit einem gelben Kreis dargestellt. Diese Zustände sollen als neutral gewertet werden. Die Position des Kreises orientiert sich am Körperpunkt *Hand_Left* (s. Abbildung 18). Die für die Gestensteuerung notwendige geschlossene Hand erhält einen grünen Kreis. Der Container mit den Körperdaten wird mit dem Array des Farbframes überlagert und in einem gesonderten Fenster ausgegeben. In Folge dessen wird ein neuer Kinect-Frame abgerufen.

Geometrische Transformationen

Die in Unterabschnitt 3.1.1 und Unterabschnitt 3.1.2 beschriebenen Bezugssysteme und Positionen des UR5-Roboters und der Kinect sind Grundlage für die folgende mathematische Berechnung der Roboterzielposition. Der Übergang vom Kamerakoordinatensystem K in das Roboterbasiskoordinatensystem R wird erreicht durch:

- eine Translation mit dem Vektor v aus dem Ursprung des Kamerasystems in den Ursprung des Roboterbasissystems
- eine Rotation von $+90°$ um die x-Achse im Kamerasystem
- eine weitere Rotation von $-90°$ um die z-Achse im neuen lokalen System

Mit den Grundlagen aus Abschnitt 2.3 gilt für die Transformation eines Zielpunktes aus Kamerasicht ${}^{K}P$ in einen Punkt aus Roboterbasissicht ${}^{R}P$:

$$ {}^{R}P = {}^{R}T_{K}\,{}^{K}P \tag{18} $$

${}^{R}T_{K}$ ist das Matrizenprodukt der aufeinanderfolgenden Transformationen.

$$ {}^{R}T_{K} = Rot_{z}(-90°)Rot_{x}(+90°)Trans(-v_{x}, -v_{y}, -v_{z}) \tag{19} $$

Die negativen Vorzeichen in der Translationsmatrix ergeben sich aus der definierten Richtung des Vektors v.

Die Transformationen wurden mit den von *Qt* bereitgestellten Klassen *QVector3D* und *QMatrix4x4* durchgeführt. Im Folgenden ist die implementierte Funktion der Koordinatentransformation zu sehen.

```
QVector3D CCoordTrans::coordinateTransform(QVector3D robotBase, QVector3D cameraTarget)
{
        QVector3D robotTarget;  // target position from robot view after transformation
        QMatrix4x4 transformMat;        // transformation matrix

        // 3D vector for matrix rotation (around x, around y, around z)
        QVector3D rotAroundX(1.0, 0.0, 0.0);
        QVector3D rotAroundY(0.0, 1.0, 0.0);
        QVector3D rotAroundZ(0.0, 0.0, 1.0);

        // first rotation around x axis
        transformMat.rotate(90, rotAroundX);

        // second rotation around z axis in the resulting coordinate system
        // corresponds to rotation around y axis in camera coordinate system
        transformMat.rotate(-90, rotAroundY);

        // translation by robotBase
        transformMat.translate(-robotBase);

        robotTarget = transformMat * cameraTarget;

        return robotTarget;
}
```

Der übergebene Parameter *robotBase* entspricht dem Vektor v für die Translation und der Parameter *cameraTarget* dem Punkt ${}^{K}P$. Der Vektor *robotTarget* entspricht der transformierten Zielposition, die dem Roboter übergeben wird und ergibt den Punkt ${}^{R}P$. Mit der Matrix *transformMat* werden die aufeinanderfolgenden homogenen Transformationen durchgeführt. Zu beachten ist, dass die Rotation um die z-Achse nicht im lokalen System nach der ersten Rotation stattfindet, sondern im Ursprungssystem, also dem Kamerakoordinatensystem. Die gewünschte Rotation um die z-Achse, entspricht einer Rotation um die y-Achse im Ursprungssystem.

Automatensteuerung

Drückt der Benutzer den Button *Start Robot Control* wird die Robotersteuerung aufgerufen. Die Steurung wird als Automat umgesetzt.

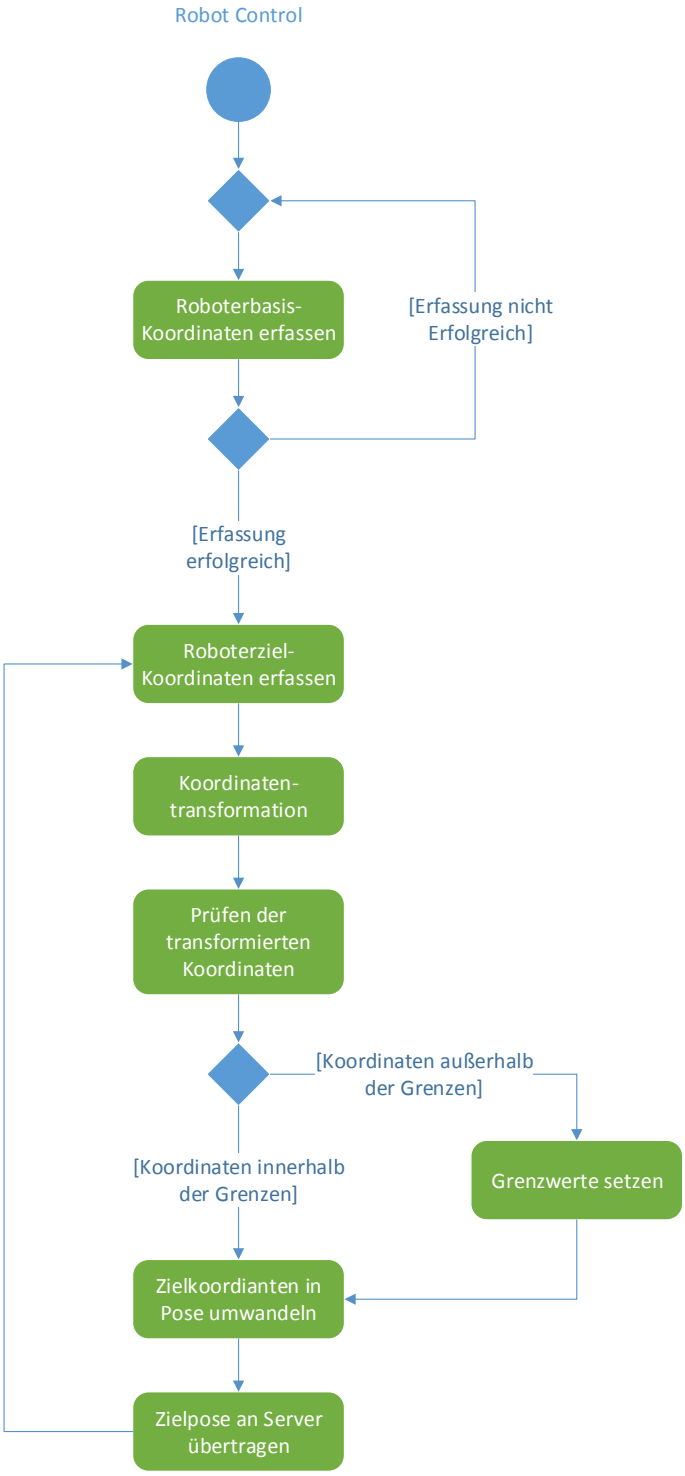


Abbildung 31: Aktivitätsdiagramm der Robotersteuerung

Ein Aktivitätsdiagramm des Automaten ist der Abbildung 31 zu entnehmen. Nach dem Start des Automaten wird der Benutzer aufgefordert die Position der Roboterbasis mit der linken Faust zu markieren. Durch den mechanischen Aufbau des Roboter-Fußflansches kann die Markierung entweder vor oder links neben dem Flansch aus Sicht des Kinect-Sensors erfolgen. Eine Korrektur abgeschätzte Korrektur von 10 cm wird vorgenommen. Mit dem Button *Confirm* wird die Position bestätigt. Wurde die Position nicht erfasst, erfolgt die Aufforderung ein weiteres Mal.
Wurde die Position erkannt, erfolgt die Erfassung der Roboterzielposition. Erneut wird der Benutzer aufgefordert die Position mit der linken Faust zu markieren. Beim ersten Mal, muss die Position mit *Confirm* bestätigt werden. Die Daten der beiden angeforderten Positionen werden aus der globalen Personenstruktur erworben, in der die *BodyFrame*-Daten nach Abruf der Frames gespeichert wurden. Die Übertragung der Daten wird mit einem *QTimer* in einem Intervall von 30 ms synchronisiert. Um den Zugriff auf die Personenstruktur kontrolliert zu gestalten und jeweils nur einem Thread den Zugriff zu erlauben, wird ein *QMutex* eingebunden.
Wurde die Position des Roboterziels übernommen, werden die Koordinaten mit der Funktion aus Abschnitt 3.2.2 transformiert. Nach der Transformation werden die Koordinaten geprüft. Liegen die Koordinaten innerhalb der gesetzten Grenzen aus Unterabschnitt 3.1.3, werden die Koordinaten in eine Pose mit der Form $[x, y, z, \phi x, \phi y, \phi z]$ umgewandelt. Die Koordinaten füllen nur die Positionsparameter x, y, z aus. Den Orientierungsparametern $\phi x, \phi y, \phi z$ werden feste Werte zugeordnet.
Liegen die Koordinaten außerhalb der Grenzen, werden die Werte der Grenzen gesetzt. Anschließend werden auch diese Koordinaten in eine Pose umgewandelt. Um den Bewegungsbefehl des Roboters ausführen zu können, muss die Pose an den Server übertragen werden.
Nach der Übertragung wird die neue Zielposition angefordert. Ab der zweiten Anforderung, wird die Position allein durch die geschlossene Faust der linken Hand des Benutzers erfasst und bestätigt. Für die Zustandsübergänge wird dann, statt eines Button-Ereignisses, ein *Single-Shot-Timer* mit 20 ms verwendet, der den für die Zustandsübergabe verantwortlichen *Slot* ausführt.

TCP-Server

Das letzte Element des Programms ist der TCP-Server. Abbildung 32 zeigt den Programmablauf. Mit der Verwendung von *Qt* ist die Implementierung eines Servers mit wenig Aufwand realisierbar. Der erste Schritt, ist die Verbindung zum Client des Roboters. Ist eine neue Verbindung verfügbar, feuert das *newConnection*-Event des Servers. Die Vorgabe einer IP-Adresse ist nicht zwingend notwendig. Mit *QHostAdress::Any* kann der Server

nach IPv4- und IPv6-Adressen lauschen. Der Port muss vorgegeben werden. Kann keine Verbindung hergestellt werden steigt das Programm mit einer Fehlermeldung aus und muss neu gestartet werden.
Steht eine Verbindung zum Server, wird mit *nextPendingConnection* ein Socket zurückgegeben, mit dem der Datenstrom der Verbindung ausgelesen wird. Ist im ausgelesenen Array vom Typ *Byte* der Anfragestring vorhanden, wird die Zielpose in der Form $[x, y, z, \phi x, \phi y, \phi z]$ als String an den Roboterclient übermittelt.

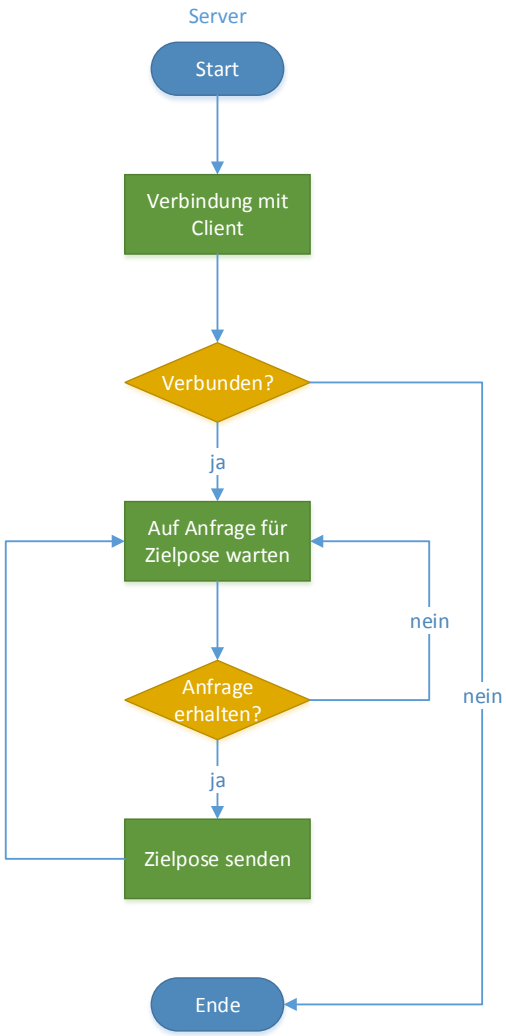


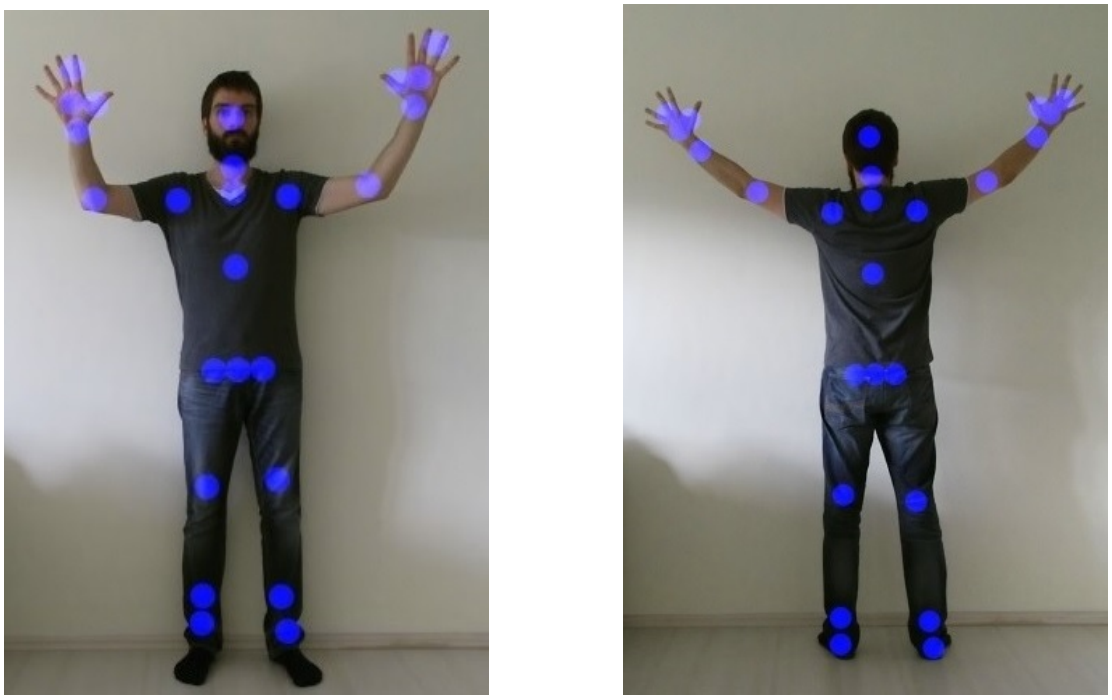
Abbildung 32: Programmablauf des Servers

4 Funktionstest

Für die Tests der implementierten Gestensteuerung des UR5-Roboterarmes wird der Focus auf die Qualität der verwendeten Kinect-Frames und die Gesteninteraktion mit dem Roboterarm in der Testumgebung gelegt.

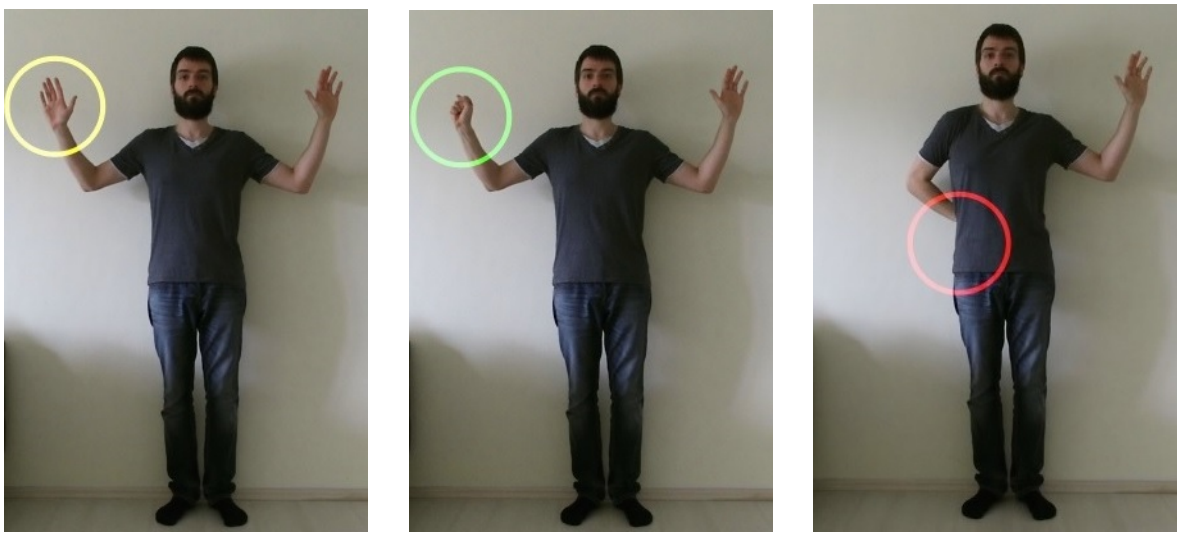
4.1 Abrufen der Kinect-Frames

Für das Abrufen der Frames wird der *MultiSourceFrameReader* und ein ereignisgesteuertes Konzept verwendet (s. Abschnitt 3.2.2). Am Fenster für die Bildausgabe des Programms *KinectRobotControl* lässt sich die Qualität der *Color*- und *Bodyframes* abschätzen.
Unter Verwendung der *ColorFrameSource* ist die Abhängigkeit der Bildfrequenz von den Lichtverhältnissen darstellbar. Bei wenig Umgebungslicht kommt es zu Aussetzern oder sogar zu „eingefrorenen“ Bildern. Mit der Verwendung des *MultiSourceFrameReaders* orientiert sich die gesamte Bildfrequenz an der niedrigsten Frequenz. Somit ist auch die *BodyFrameSource* von Aussetzern betroffen. Für eine gute Bildfrequenz von bis zu 30 fps muss bei der Verwendung der *ColorFrameSource* also für ausreichende Lichtverhältnisse gesorgt werden.
Um die Qualität der Bodydaten besser bewerten zu können, werden die Gelenkpunkte in der Bildausgabe mit *OpenCV* in Form von blauen Punkten sichtbar gemacht. Diese Option wird bei der finalen Version des Programms wieder entfernt, da sie für die Interaktion mit dem UR5 nicht zwingend notwendig ist. Für die korrekte Erfassung der Körperdaten muss der gesamte Körper der Person im Sichtfeld der Kinect vorhanden sein. In Abbildung 33 sind zwei Ausschnitte der Bildausgabe mit den markierten Gelenkpunkten zu sehen. Es ist zu erkennen, dass die Erfassung der Körperpunkte sehr präzise ist. Sogar mit der zum Sensor zugewandten Körperrückseite liegen die Gelenkpunkte an ihrer korrekten Position. Die Übertragung der Koordinaten mit dem *CoordinateMapper* vom *CameraSpace* in den *ColorSpace* erfolgt sauber und genau (s. Abbildung 33). Da die Punkte im Bereich der Hände sehr nah beieinanderliegen, kann es aber durchaus vorkommen, dass die Erfassung instabil ist. Der Handzustand wird zuverlässig erkannt. Abbildung 34 zeigt die Ausgabe der Handzustände im Programm.



(a) von vorn (b) von hinten

Abbildung 33: Erfasste Körperpunkte



(a) offen (gelb) (b) geschlossen (grün) (c) unbekannt (rot)

Abbildung 34: Erkannte Handzustände

Da nur eine Person in diesem Programm erfasst werden soll, muss darauf geachtet werden, welche die aktive Person für die Robotersteuerung ist. Im Kinect-API gibt es keine Referenz für die Auswahl der zu erfassenden Personen. Der Entwickler der Kinect-Anwendung muss je nach Problemstellung selbst für ein geeignetes Verfahren der Personenauswahl sorgen.

4.2 Interaktion mit dem UR5-Roboterarm

Um im Voraus Fehler bei der Berechnung der Roboterzielkoordinaten auszuschließen, wurde ein Testprogramm geschrieben. Das Programm erfordert die manuelle Eingabe der Funktionsparameter. An der Ausgabe lässt sich das Ergebnis abschätzen. Die Funktion ermittelt das korrekte Ergebnis.
Zur Kontrolle der Zielpose, die an das Roboterscript übertragen werden soll, wurde ein Testclient implementiert. Die Zielpose wird in der korrekten Form $[x, y, z, \phi x, \phi y, \phi z]$ übertragen.
Bei den Testfahrten mit dem Roboter wird die Kinect wie in Unterabschnitt 3.1.2 beschrieben positioniert und ausgerichtet. Die Server-Client-Verbindung wird nach dem Start des Roboterscripts und dem Programm *KinectRobotControl* erfolgreich hergestellt. Die Erfassung der Roboterbasis und der Roboterzielposition wird ordnungsgemäß durchgeführt. Zu beachten ist dabei die leichte Ungenauigkeit durch die Korrektur am Fußflansch des Roboters. Nach der Transformation der Zielkoordinaten fährt der Roboter mit einer leichten Abweichung an die Handposition. Die Abweichung bleibt bei weiteren Fahrten sichtlich konstant. Es kann davon ausgegangen werden, dass diese Abweichung aus dem Testaufbau resultiert. Durch die freie Positionierung des Sensors und die vorgenommene Abschätzung des rechten Positionswinkels zum Roboter können Abweichungen in der Berechnung auftreten. Bei der Rotation wird von präzisen Winkelmaßen ausgegangen. Weiterhin müssen bei der Interpretation der Zielpose die gesetzten Grenzen des Bewegungsspielraums des Roboterarmes mit einbezogen werden. So wird die aktuelle Handposition nicht erreicht, wenn sie außerhalb des Bewegungsbereichs liegt. Es ist davon auszugehen, dass der Kinect-Sensor die erfragten Koordinaten ausreichend genau erfasst die Messungen dieser stabil ist.
Da der Roboterarm erst seine Zielposition erreichen muss, bevor eine neue Zielabfrage gemacht wird, sind die Zeiten der implementierten Timer für die Datenübertragung zum Client ausreichend.
Das Verfahren für die Personenverfolgung, wählt das erste Individuum, das vollständig im Sichtfeld des Sensors erkannt wird. Es ist vorgekommen, dass eine andere Person erfasst wurde, wenn die ursprünglich verfolgte Person den Sichtbereich verlassen hat. Dieses Phänomen trat aber nicht während der eigentlichen Fahrsequenz auf, da die bedienende Person in der Regel zu diesem Zeitpunkt den Sichtbereich nicht verlässt. Das Verfahren für die Personenerfassung ist aus Sicht der Anwenderfreundlichkeit dennoch nicht optimal gewählt.

5 Fazit und Ausblick

Der Kinect v2-Sensor hat sich als Kommunikationsgerät für die Interaktion mit dem UR5-Roboterarm bewährt. Mit der Bedienung des Roboters über Handgesten lassen sich alle Dimensionen im Raum nutzen und sie ist eine Alternative zur Steuerung über das GUI *PolyScope*.
Das *Kinect for Windows* SDK 2.0 bietet eine große Auswahl an Beispielen und Tools für den Einstieg bei der Entwicklung von Applikationen unter Windows und mit dem umfangreichen API lässt sich auf eine Vielzahl von Datenquellen des Sensors zugreifen.
Unter Verwendung der *BodyFrameSource* lassen sich durchaus solide Ergebnisse präsentieren. Die für die entwickelte Applikation benötigten Körperpunkte und der Handzustand des Bedieners wurden präzise erfasst und weiterverarbeitet. Lediglich eine leichte Abweichung des Roboterarmes zur Zielposition konnten festgestellt werden. Dieser Abweichung lässt sich gegebenenfalls mit der festen Montage des Sensors an einem definierten Ort entgegenwirken. Die durch den Benutzer notwendige Markierung der Roboterbasis als Bezugssystem für die Zielpose würde damit entfallen. Feste Maße und Zuordnungen können der Berechnungsfunktion in Form von Parametern übergeben werden und damit ist eine eindeutigere geometrische Transformation möglich. Auch eine Kalibrierung ist durchaus sinnvoll und kann zur Verbesserung der Zielgenauigkeit beitragen.
Es konnte festgestellt werden, dass die Bildfrequenz durch den Einsatz der *ColorFrameSource* in einigen Fällen durch geringes Umgebungslicht negativ beeinflusst wird. Aussetzer in der Bildausgabe führten bei der eigentlichen Bewegungssteuerung des Roboters aber zu keinen Problemen. Für eine präzisere Auswertung kann das Programm trotzdem um die Messung der Bildfrequenz und eine anschließende Bildschirmausgabe des Messergebnisses erweitert werden.
Durchaus größere Probleme könnten sich während der Personenverfolgung ergeben. Soll, wie es bei einer Robotersteuerung Sinn macht, nur eine Person erfasst werden, so muss der Entwickler der Anwendung ein auf die Problemstellung abgestimmtes und geeignetes Verfahren finden. Das in dieser Applikation verwendete Verfahren ist einfach gehalten und sollte für solche Fälle überarbeitet werden.
Ein Verfahren mit Spracherkennung könnte zu einer verbesserten Personenverfolgung beitragen. Die Verwendung der *AudioSource* in Verbindung mit anderen Datenquellen der

Kinect kann die Interaktion mit einem Roboter bereichern. So ließen sich durch simple Sprachbefehle z.B. die Zuweisung der bedienenden Person, die Bewegung des Roboters oder der Startvorgang von Programmabläufen umsetzen. Die Abhängigkeit der Bedienbarkeit von einer weiteren Person ließe sich ebenfalls durch Sprachbefehle minimieren. Diese Befehle sollten kurz und präzise sein. Bei der Wahl der Räumlichkeiten sollte unter Nutzung der *AudioSource* auf die Akustik und Umgebungsgeräusche geachtet werden.
Eine weitere empfehlenswerte Datenquelle ist die *DepthFrameSource*. Mit ihr kann eine Objekterkennung über die Auswertung der dreidimensionalen Punktewolke angestrebt werden. Die Position des Objektes kann dann als Zielposition dem Roboter übergeben werden.
Für die Ausführung festgelegter Bewegungen des Roboters kann die Erstellung eines Gestendetektors mit dem *Visual Gesture Builder* interessant sein.

Somit hat sich gezeigt, dass der Kinect v2-Sensor neben dem Einsatz für die Konsole XBox One auch durchaus für Anwendungen in anderen Bereichen der Human Computer Interaction attraktiv ist.

Literaturverzeichnis

[Bil11] Bilton, Nick ; New York Times (Hrsg.): *Microsoft Sells 10 Million Kinects, Breaking Record*. http://bits.blogs.nytimes.com/2011/03/09/microsoft-sells-10-million-kinects-breaking-record/. Version: 2011. – Zuletzt abgerufen am 10.04.2016 11

[Fan15] Fankhauser, M.; Rodriguez D.; Kaestner R.; Hutter M.; Siegwart R. P.; Bloesch B. P.; Bloesch: Kinect v2 for mobile robot navigation. Evaluation and modeling. In: *2015 International Conference on Advanced Robotics (ICAR)*, 2015, S. 388–394 11

[Glo16] Globke, Wolfgang: *Koordinaten, Transformationen und Roboter*. http://www.math.kit.edu/iag2/~globke/media/koordinaten.pdf. Version: 2016. – Zuletzt abgerufen am 17.07.2016 30, 31, 32, 59

[iFi16] iFixit: *Xbox One Kinect Teardown*. https://de.ifixit.com/Teardown/Xbox+One+Kinect+Teardown/19725. Version: 2016. – Zuletzt abgerufen am 20.04.2016 12, 59

[Kau14] Kaushik, R. M.; J. M.; Jain: Gesture Based Interaction NUI. An Overview. In: *International Journal of Engineering Trends and Technology* 9 (2014), Nr. 12, S. 633–636. – ISSN 22315381 7

[Lac15] Lachat, H.; Mittet M.-A.; Landes T.; Grussenmeyer P. E.; Macher M. E.; Macher: First Experiences With Kinect v2 Sensor For Close Range 3D Modelling. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-5/W4 (2015), S. 93–100. – ISSN 2194–9034 13, 18, 59

[Mic13] Microsoft: *Visual Gesture Builder. A Data-Driven Solution to Gesture Detection*. https://onedrive.live.com/view.aspx?resid=1A0C78068E0550B5!77743&app=WordPdf. Version: 2013. – Zuletzt abgerufen am 17.07.2016 29

[Mic14] Microsoft: *Human Interface Guidelines v2.0*. http://download.microsoft.com/download/6/7/6/676611B4-1982-47A4-A42E-4CF84E1095A8/KinectHIG.2.0.pdf. Version: 2014. – Zuletzt abgerufen am 17.07.2016 12, 13, 24, 26, 27, 59

[Mic16a] Microsoft: *Coordinate mapping*. https://msdn.microsoft.com/en-us/library/dn785530.aspx. Version: 2016. – Zuletzt abgerufen am 20.04.2016 28, 59, 61

[Mic16b] Microsoft: *JointType Enumeration*. https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx. Version: 2016. – Zuletzt abgerufen am 08.04.2016 24, 59

[Mic16c] Microsoft: *Kinect for Windows C++ Reference*. https://msdn.microsoft.com/en-us/library/dn791993.aspx. Version: 2016. – Zuletzt abgerufen am 21.07.2016 22, 23, 24, 25, 27, 28

[Mic16d] Microsoft: *Kinect for Windows SDK*. https://msdn.microsoft.com/en-us/library/dn799271.aspx. Version: 2016. – 21.07.2016 20, 29

[Mic16e] Microsoft: *Kinect hardware setup*. https://developer.microsoft.com/en-us/windows/kinect/hardware-setup. Version: 2016. – Zuletzt abgerufen am 22.04.2016 19, 59

[Mic16f] Microsoft: *Kinect Studio*. https://msdn.microsoft.com/en-us/library/dn785306.aspx. Version: 2016. – 23.07.2016

[Ope16] OpenCV dev team: *OpenCV 2.4.13.0 documentation*. http://docs.opencv.org/2.4.13/index.html. Version: 2016. – Zuletzt abgerufen am 21.07.2016 35

[Pag15] Pagliari, L. D.; P. D.; Pinto: Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors. In: *Sensors* 15 (2015), Nr. 11, S. 27569–27589. – ISSN 1424–8220 11, 13, 14

[Pre15] Preim, R. B.; D. B.; Dachselt: *Interaktive Systeme*. 2. Aufl. Berlin : Springer Vieweg, 2015 (eXamen.press). – ISBN 3642452469 7

[Qt 16a] Qt Company Ltd.: *Qt Documentation*. http://doc.qt.io/. Version: 2016. – Zuletzt abgerufen am 25.07.2016 36, 42

[Qt 16b] Qt Company Ltd.: *Signals & Slots | Qt Core 5.7*. http://doc.qt.io/qt-5/signalsandslots.html. Version: 2016. – Zuletzt abgerufen am 25.07.2016 36

[Sar15] Sarbolandi, Damien; Kolb A. Hamed; Lefloch L. Hamed; Lefloch: Kinect range sensing. Structured-light versus Time-of-Flight Kinect. In: *Computer Vision and Image Understanding* 139 (2015), S. 1–20. – ISSN 10773142 12

[Sel14] Sell, P.; Sell John; O'Connor P. J.; O'Connor O. J.; O'Connor: The Xbox One System on a Chip and Kinect Sensor. In: *IEEE Micro* 34 (2014), Nr. 2, S. 44–53. – ISSN 0272–1732 15, 16, 17, 18, 59

[Uni15a] Universal Robots: *Benutzerhandbuch UR5/CB3*. http://www.universal-robots.com/media/8692/ur5_user_manual_de_global.pdf. Version: 2015. – Zuletzt abgerufen am 18.07.2016 9, 10, 59

[Uni15b] Universal Robots: *UR5-Roboter |Automatisierung von Aufgaben bis zu 5 kg*. http://www.universal-robots.com/de/produkte/ur5-roboter/. Version: 2015. – Zuletzt abgerufen am 17.07.2016 9, 59

[Uni15c] Universal Robots: *URScript API Reference*. https://s3-eu-west-1.amazonaws.com/ur-support-site/15731/scriptmanual_en.pdf. Version: 2015. – Zuletzt abgerufen am 20.07.2016 10, 42

[Wüs16] Wüst, Klaus: *Grundlagen der Robotik*. https://homepages.thm.de/~hg6458/Robotik/Robotik.pdf. Version: 2016. – Zuletzt abgerufen am 27.07.2016 31, 33, 34

[Zha16] Zhang, J.: *Angewandte Sensorik*. https://tams.informatik.uni-hamburg.de/lehre/2003ws/vorlesung/angewandte_sensorik/vorlesung_08.pdf. Version: 2016. – Zuletzt abgerufen am 02.08.2016 33

Abbildungsverzeichnis

1 UR5-Roboter von Universal Robots [Uni15b] . . . 9
2 UR5-Gelenke. A: Fußflansch, B: Schulter, C: Ellenbogen, D,E,F: Handgelenk 1, 2, 3 [Uni15a, S. II-3] . . . 10
3 Bedienung UR5 . . . 10
4 Kinect der zweiten Generation [Mic14, S. 5] . . . 12
5 Komponenten der Kinect v2 [iFi16] . . . 12
6 Sichtfeld des Kinect v2 für Infrarot und Tiefemessung [Mic14, S. 7] . . . 13
7 3D Sensorsystem [Sel14, S. 49] . . . 15
8 ToF-Sensor und dessen Signalformen [Sel14, S. 50] . . . 16
9 Unterschiedliche Modulationsfrequenzen [Sel14, S. 51] . . . 17
10 Messgenauigkeit als Funktion der Aufwärmzeit [Lac15, S. 96] . . . 18
11 Kinect-Adapter [Mic16e] . . . 19
12 Auszug des Kinect-Konfigurationsverifizierers . . . 19
13 Zugriff auf Kinect-Sensordaten . . . 20
14 Farbbild in Full HD . . . 22
15 Infrarotbild . . . 23
16 Tiefenbild . . . 23
17 Personenerkennungsbereich [Mic14, S.7] . . . 24
18 Körperpunkte (Joints) [Mic16b] . . . 24
19 Handzustände . . . 25
20 Bodytracking . . . 25
21 Body Index . . . 26
22 Microfon-Array . . . 26
23 CameraSpace-Koordinatensystem der Kinect v2 [Mic16a] . . . 28
24 Datengesteuerter Entwicklungsprozess für einen Gestendetektor mit VGB . . . 29
25 Rotation um die z-Achse [Glo16, S. 6] . . . 31
26 Translation mit $\boldsymbol{v}_0$ [Glo16, S. 19] . . . 32

27 Testaufbau der Robotersteuerung . . . 38
28 Programmablauf Roboterscript . . . 41
29 Oberfläche des *Graphical User Interfaces* . . . 43

30 Programmablauf Kinect 44
31 Aktivitätsdiagramm der Robotersteuerung 48
32 Programmablauf des Servers 50

33 Erfasste Körperpunkte 52
34 Erkannte Handzustände 52

Tabellenverzeichnis

1 Vergleich der beiden Kinect-Versionen 14
2 Vergleich der Kinect-Koordinatensysteme [Mic16a] 28

Anhang

Der Anhang dieser Bachelorthesis befindet sich auf DVD, einzusehen bei den Prüfern Prof. Dr.-Ing. J. Maaß oder Prof. Dr.-Ing. J. Dahlkemper.

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg,	
Ort, Datum	Unterschrift