FH Vorarlberg
University of Applied Sciences

Masterarbeit

Teachen von Industrierobotern mittels Gestenerkennung

Masterarbeit
zur Erlangung des akademischen Grades

Master of Science in Engineering (MSc)

Fachhochschule Vorarlberg
Informatik

Betreut von
Dipl.-Ing. Robert Amann

Vorgelegt von
Helmut Rhomberg
1810249003

Dornbirn, August 2020

Gender-Erklärung

Der Verfasser der vorliegenden Arbeit bekennt sich zu einer geschlechtergerechten Sprachverwendung.

Um die Lesbarkeit zu gewährleisten und zugunsten der Textökonomie werden die verwendeten Personen bzw. Personengruppen fix männlich oder weiblich zugeordnet. Zum Beispiel werden immer die Entwicklerin, der Mitarbeiter, der Anwendende und die Kundin verwendet. Es wurde besonders darauf geachtet, stereotype Rollenbeschreibungen zu vermeiden.

Kurzreferat

Teachen von Industrierobotern mittels Gestenerkennung

Das Teach Pendant wird dazu genutzt um Industrierobotern Zielposen beizubringen, damit diese die Zielposen daraufhin autonom anfahren können. Durch den auf dem Teach Pendant zur Eingabe verbauten Joystick oder die 3D-Maus ist es möglich mit geringem Rechenaufwand Bewegungen und somit Zielposen präzise vorzugeben.

Durch den rasanten Anstieg der Rechenkapazität und den stetigen Erfolgen bei der Erforschung von künstlichen neuronalen Netzwerken sind heutzutage Gestenerkennungssysteme in das Blickfeld der Forschung gelangt. Gestenerkennungssysteme versprechen intuitive und leicht zu erlernende Bedienungskonzepte. Durch den Einsatz von Gesten kann zudem auf ein zusätzliches Gerät in den Händen verzichtet werden, wodurch das Nutzererlebnis gesteigert werden kann. Dies kann bei schweren Eingabgeräten nicht nur die Arme sondern auch die Hände vor Ermüdungserscheinungen schonen.

Das Ziel dieser Arbeit soll es daher darstellen, ein Gestenerkennungssystem mit und ohne ROS-Anbindung zu erstellen, testen und analysieren. Die Gesteninformationen werden von einer Tiefenkamera vom Typ Azure Kinect bereitgestellt. Als Industrieroboter wird hierbei der „WidowX 200“-Lernroboter eingesetzt, da dieser aufgrund seiner kleinen Bauform einfach und effizient zum Testen von neuen Funktionalitäten eingesetzt werden kann. Die Tiefenkamera- und Roboter-Komponente sollen austauschbar bleiben. Die zu entwickelnden Gesten sollen vor allem hohe Ergonomie bereitstellen und vor unbeabsichtigter Durchführung gesichert sein. Die Genauigkeit der Gestenerkennung wird analysiert, um die Zuverlässigkeit der ausgewählten Gesten zu evaluieren. Die Genauigkeit der erreichten Zielposen des „WidowX 200“-Lernroboters wird analysiert und Latenztests werden durchgeführt um das Gestensystem als Gesamtes bewerten zu können.

Abstract

Teaching of industrial robots using gesture recognition

The Teach Pendant is used to teach target poses to industrial robots, so that they can approach the target poses autonomously. The joystick or 3D mouse installed on the teach pendant for input makes it possible to specify movements and thus target poses precisely with little computing effort.

Due to the rapid increase in computing capacity and the constant success in the research of artificial neural networks, gesture recognition systems have become the focus of research today. Gesture recognition systems promise intuitive and easy to learn operating concepts. The use of gestures also makes it possible to do without an additional device in the hands, thus enhancing the user experience. In the case of heavy input devices, this can protect not only the arms but also the hands from signs of fatigue.

The goal of this thesis is therefore to create, test and analyze a gesture recognition system with and without ROS connection. The gesture information is provided by a depth camera of the Azure Kinect type. The „WidowX 200“ learning robot is used as the industrial robot for this purpose, since its small design allows it to be used simply and efficiently for testing new functionalities. The depth camera and robot components are to remain interchangeable. The gestures to be developed should above all provide high ergonomics and be protected against unintentional execution. The accuracy of gesture recognition will be analyzed to evaluate the reliability of the selected gestures. The accuracy of the achieved target poses of the „WidowX 200“ learning robot will be analyzed and latency tests will be performed to evaluate the gesture system as a whole.

Inhaltsverzeichnis

Abkürzungsverzeichnis **7**

1. Einleitung **9**
1.1. Umfeld . . . 9
1.2. Problemstellung . . . 10
1.3. Zielsetzung . . . 10

2. Stand der Technik **12**
2.1. Industrieroboter . . . 12
2.1.1. Bauformen . . . 13
2.1.2. Positions- und Bahngenauigkeit . . . 15
2.2. Teachen von Industrierobotern . . . 17
2.2.1. Offline- und Online-Programmierverfahren . . . 17
2.2.2. Teach Pendant . . . 19
2.3. Arten von Echtzeitverhalten . . . 20
2.3.1. Echtzeitanforderungen . . . 21
2.3.2. Implementationen gängiger Betriebssysteme . . . 22
2.4. Sicherheitsanforderungen im Umgang mit Industrierobotern . . . 23
2.5. ROS . . . 24
2.5.1. Softwarearchitektur . . . 25
2.5.2. Interoperabilität . . . 26
2.6. Tiefenkamera . . . 26
2.6.1. Funktionsweise . . . 26
2.6.2. Azure Kinect . . . 27
2.7. Vorhandene Gestenerkennungssysteme . . . 28

3. Umsetzung **30**
3.1. Allgemeine Anforderungen . . . 31
3.1.1. Funktionale Anforderungen . . . 31
3.1.2. Nichtfunktionale Anforderungen . . . 32
3.2. Einrichtung des Testsystems . . . 33
3.2.1. Linux-Container . . . 33
3.2.2. Simulationsumgebungen . . . 34
3.3. Anbindung an ROS . . . 34
3.4. Gesten . . . 35
3.4.1. Arten von Gesten . . . 36
3.4.2. Beispiele für unethische symbolische Gesten . . . 37

3.5. Auswahl der zu verwendenden Gesten 38
3.5.1. Verwendete Gesten 38
3.5.2. Zuordnung der Gesten zu den Aktionen 41
3.5.3. Präzisionsmodus 45
3.5.4. Koordinatensysteme in ROS 45
3.5.5. Sicherheitskonzept 45
3.6. Aufbau des Roboter-Gesten-Frameworks 46
3.6.1. Gesten-Subsystem 46
3.6.2. Abstandsberechnung für Gestensystem 48
3.6.3. Roboter-Gesten-Anwendung 49
3.6.4. Regelkreis 52
3.6.5. MoveIt-Interaktion mit InterbotiX-Node 53
3.7. Messvoraussetzungen 53

4. Ergebnisse **57**
4.1. Latenz der Gestenerkennung 57
4.2. Latenz der Roboter-Kommunikation mit und ohne ROS-Anbindung . . . 59
4.3. Genauigkeit der Zielpositionen 67
4.4. Genauigkeit der Gestenerkennung 69

5. Zusammenfassung & Ausblick **70**
5.1. Zusammenfassung 70
5.2. Ausblick 71

A. Anhang **73**
A.1. Einrichtung des Testsystems 73
A.1.1. Erstellen des Linux-Containers 73
A.1.2. Installation & Konfiguration der Pakete 76
A.1.3. Starten der Roboter-Gesten-Anwendung 83

Literaturverzeichnis **86**

Eidesstattliche Erklärung **95**

Abkürzungsverzeichnis

AI	**A**rtificial **I**ntelligence
API	**A**pplication **P**rogramming **I**nterface
AR	**A**ugmented **R**eality
CAD	**C**omputer-**A**ided **D**esign
CPU	**C**entral **P**rocessing **U**nit
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
DNN	**D**eep **N**eural **N**etwork
DOF	**D**egrees **O**f **F**reedom
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FPS	**F**rames **P**er **S**econd
GPU	**G**raphics **P**rocessing **U**nit
GUI	**G**raphical **U**ser **I**nterface
HCI	**H**uman-**C**omputer **I**nteraction
KI	**K**ünstliche **I**ntelligenz
KISS	**K**eep **I**t **S**imple, **S**tupid
LTS	**L**ong **T**erm **S**upport
LXC	Linux Containers
LXD	Linux Container Hypervisor
NUI	**N**atural **U**ser **I**nterface
PTP	**P**oint-**t**o-**P**oint
ROS	**R**obot **O**perating **S**ystem
RAM	**R**andom-**A**ccess **M**emory
REST	**Re**presentational **S**tate **T**ransfer
RTOS	**R**eal **T**ime **O**perating **S**ystem

SCARA	**S**elective **C**ompliance **A**ssembly **R**obot **A**rm
SDK	**S**oftware **D**evelopment **K**it
TCP	**T**ool **C**enter **P**oint
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
UX	**U**ser E**x**perience
VR	**V**irtual **R**eality

1. Einleitung

Industrieroboter erfreuen sich heutzutage großer Beliebtheit bei Unternehmen um Produkte schnell und effizient herstellen zu können. Die Einsatzbereiche reichen hierbei von Laserschneiden über Schweißen und Fräsen bis hin zu Montageaufgaben, welche mitunter nur schwer von Personen durchgeführt werden können oder sogar lebensbedrohlich für Menschen sind. Zur Vermeidung von lebensbedrohlichen Situationen und zur Erleichterung von Vorgängen können Industrieroboter den Menschen hierbei effizient unterstützen und schwere Aufgaben erleichtern. Nichtsdestotrotz müssen von den Industrieroboterherstellern die geltenden Maschinenrichtlinien eingehalten werden um die Arbeit mit Menschen so gefahrlos wie nur möglich zu gestalten. Ein Beispiel hierfür ist das Erlernen bzw. Teachen von neuen Posen für einen Industrieroboter, sodass dieser seine Aufgaben im späteren Verlauf autonom durchführen kann. Hierzu muss sich die bedienende Person mitunter sehr nahe beim Industrieroboter befinden um Kollisionen mit Objekten frühzeitig erkennen und vermeiden zu können, aber auch um hohe Genauigkeiten für die zu erlernenden Zielposen zu erreichen. Zur Vermeidung von lebensbedrohlichen Situationen müssen die Industrieroboter beim Teachen deswegen unter anderem die Geschwindigkeit und Beschleunigung der Gelenke des Industrieroboters begrenzen.

1.1. Umfeld

Zum Teachen von Industrierobotern werden heutzutage vermehrt Teach Pendants eingesetzt, da durch deren Einsatz von Joystick und 3D-Maus als Eingabemethode sehr feine Bewegungen realisiert werden können. Zudem bietet das Teach Pendant durch das ergonomische Bedienen des „Enable device“-Schalters eine für Menschen sichere Möglichkeit einen Industrieroboter sicher zu bedienen. Bei zu starker oder zu schwacher Betätigung des „Enable device“-Schalters wird vom Teach Pendant eine Gefahrensituation angenommen und der Industrieroboter zum Stillstand gebracht um die bedienende Person vor lebensbedrohlichen Situationen zu schützen. Im Großen und Ganzen kann gesagt werden, dass ein Teach Pendant ein Multifunktionsgerät zur Programmierung von Industrierobotern darstellt und ohne großen Rechenaufwand und entsprechend komplizierten Algorithmen zum selbstständigen Finden der Wege auskommt. Diese Designentscheidung war unter anderem notwendig, da der Rechenaufwand zum selbständigen Finden von Wegen zu der Zeit der ersten Teach Pendants viel zu hoch gewesen wäre. Zudem ist es nicht immer möglich die Wege selbstständig von einer Software finden zu lassen, da zumeist zuwenige Informationen über die Umgebung bereitstehen. Daher musste eine alternative Möglichkeit entwickelt werden um Industrierobotern Posen beibringen zu können, welche anschließend autonom abgefahren werden können. Die Möglichkeit dem Industrieroboter

Posen mittels Teach Pendant zu erlernen hat sich aufgrund des Rechenaufwands daher bis heute als sehr hilfreich erwiesen. Aus diesem Grund haben sich Teach Pendants zum De-facto-Standard zur Programmierung von Industrierobotern entwickelt. Die Funktionalität der Teach Pendants wurde zudem um Hilfsfunktionen, wie z.B. Bewegungsmodi und Beschleunigungsanpassungen, und Hardwarefunktionalitäten, wie z.B. einen Touchpad, erweitert, um die Bedienung des Teach Pendants und so auch des Industrieroboters für die bedienende Person noch intutitiver zu gestalten.

1.2. Problemstellung

Obwohl Teach Pendants sehr weit verbreitet sind und sich über die Jahre als sehr hilfreich beim Teachen von Industrierobotern erwiesen haben, wird dennoch unter anderem durch den rasanten Anstieg der Rechenkapazität an neuen Eingabemethoden für Industrieroboter geforscht. Die Eingabemethoden erstrecken sich hierbei unter anderem von touchbasierten Eingabemethoden auf einer ebenen Fläche [Wer19], über Freihandgesten im 3D-Raum [Gra, S. 2 ff.] und Datenhandschuhen [Bra, S. 1 f.] bis hin zu künstlichen neuronalen Netzwerken, welche durch Zuhilfenahme von Positionsvorgaben selbstständig eine für den Industrieroboter optimale Bewegungsbahn ermitteln können [NNPPM10]. Ein zukunftsweisender Ansatz liegt in der Verwendung von NUI um der bedienenden Person die Bedienung des Industrieroboters mittels des eigenen Körpers so intutitiv wie nur möglich zu gestalten. Eine Möglichkeit eine natürliche Benutzterschnittstelle zu entwickeln besteht in der Verwendung von Gesten. Gesten sind intutitiv zu erlernen und einfach anzuwenden, wodurch die Verwendung eines zusätzlichen Geräts, wie z.B. eines Teach Pendants, entfällt. Bei schweren Eingabegeräten, welche in der Hand gehalten werden, ist dies besonders spürbar, da ein längeres Bedienen des Geräts zu Ermüdungserscheinungen in den Händen und Armen führen kann. In der Gestenerkennung wird aus diesem Grund eine einfache und intutitive Bedienungsmöglichkeit gesehen, welche eine ergonomischere Bedienung als mit einem herkömmlichen Teach Pendant ermöglichen kann. Die Sicherheitsaspekte dürfen dabei jedoch aber auch nicht vernachlässigt werden, da die Maschinenrichtlinien genauestens spezifizieren wie die zu bedienende Maschine mit Menschen interagieren muss um lebensbedrohliche Verletzungen zu vermeiden. Unter anderem muss hierfür die Geschwindigkeit und die Beschleunigung des Industrieroboters gedrosselt werden um es der bedienenden Person zu ermöglichen unbeabsichtigten Bewegungen des Industrieroboters zeitgerecht ausweichen zu können. In dieser Arbeit wird das Anfahren von Posen in der Betriebsart Handbetrieben betrachtet. Die entwickelte Benutzerschnittstelle ist nicht für den Automatikbetrieb ausgelegt.

1.3. Zielsetzung

In dieser Arbeit stellt das Erstellen, Testen und Analysieren einer Gestenerkennung, welche die Tiefeninformationen von einer Tiefenkamera erhält, im Vordergrund. Hierbei sollen ergonomische Gesten entwickelt werden, welche nur schwer unbeabsichtigt durchge-

führt werden können, die Genauigkeit der Gestenerkennung und der Zielposen analysiert und Latenztests durchgeführt werden. Zur genauen Umsetzung soll eine Azure Kinect als Tiefenkamera eingesetzt und der „WidowX 200“-Lernroboter zur Umsetzung und Überprüfung der Gestenerkkenung verwendet werden. Die zu erstellende Gestenerkennung soll jedoch nicht nur auf die Azure Kinect und den „WidowX 200“-Lernroboter beschränkt sein. Hierfür sollen deshalb Schnittstellen geschaffen werden, welche das einfache Austauschen der Tiefenkamera- und Roboter-Komponente ermöglichen sollen. Aus diesem Grund soll zudem ermittelt werden ob eine Anbindung über ROS hinsichtlich der Latenzen eine die Anforderungen erfüllende Alternative zu einer direkten Kommunikation mit einem Industrieroboter darstellt. Daher soll eine konkrete Kommunikation über ROS und eine direkte Kommunikation mit dem „WidowX 200“-Lernroboter implementiert werden. Zu guter Letzt sollen die beiden Implementierungen auf Grundlage von Latenztests miteinander verglichen werden. Im Anschluss kann ein Fazit gezogen werden ob ROS den geforderten Latenzenansprüchen von Industrierobotern gerecht werden kann. Alle in dieser Arbeit erstellten Dateien können zudem auf der beigefügten DVD oder im Gitlab-Repository unter https://gitlab.com/TeachIndustrialRobots/Teach-Industrial-Robots-Media öffentlich eingesehen werden. Der Einfachheit halber wird die DVD oder das Gitlab-Repository in dieser Arbeit als „Begleitmedium“ bezeichnet.

2. Stand der Technik

Industrieroboter können eine Vielzahl von verschiedenen Aufgabengebieten übernehmen. Die Aufgabengebiete reichen hierbei von Bedienungs- über Bearbeitungs- bis hin zu Montageabläufen [Ind]. Die Schwierigkeit besteht hierbei zunächst die Anforderungen an den erforderlichen Industrieroboter zu definieren und wenn möglich einzugrenzen. Da die Kommunikation mit und ohne ROS getestet werden soll ist es von Vorteil einen Industrieroboter mit einer bereits vorhandenen ROS-Unterstützung zu verwenden. Zudem ist es erforderlich den Industrieroboter über eine direkte Schnittstelle ansprechen zu können um die gewünschten Abläufe, welche geteacht werden sollen, mit einer direkten Kommunikation ohne Netzwerklatenzen testen und einen Vergleich auf Grund des Zeitverhaltens anstellen zu können. Die Interoperabilität mit Simulationsumgebungen ist auch abzuwägen, da die Tests mithilfe einer Simulationsumgebung ohne Sicherheitsbedenken für die bedienende Person durchgeführt und zudem für Regressionstests der Gestenerkennungssoftware praktikabel ohne einen realen Roboter durchgeführt werden können. Im weiteren Verlauf muss zudem die Entscheidung für einen Tiefensensor durchgeführt werden um eine Gestenerkennung, welche zur Steuerung des Inustrieroboters eingesetzt werden soll, realiseren zu können. Bei der Gestenerkennung steht vor allem die Ergonomie und die Sicherheit der bedienenden Person im Vordergrund. Hierbei muss unter anderem beachtet werden, dass zufällige Gesten nicht als Aktion gewertet werden, da dies ansonsten die Sicherheit der bedienenden Person gefährden kann. Bei der Programmiersprache und Programmierumgebung ist dabei darauf zu achten, dass diese zu ROS kompatibel ist, da ansonsten erheblicher Mehraufwand bei der Umsetzung entstehen würde.

2.1. Industrieroboter

Ein Industrieroboter ist ein universell einsetzbarer Bewegungsautomat, welcher im industriellen Einsatzgebiet eingesetzt wird. Zu erwähnen ist, dass Industrieroboter zumeist auf ein bestimmtes Problem zugeschnitten sind. Aus diesem Grund werden diese je nach der Positionsgenauigkeit, Tragfähigkeit, Arbeitsbereich, Arbeitsgeschwindigkeit und maximaler Reichweite unterschieden. Die maximale Reichweite ist bei ortsfesten Industrierobotern durch die Armlänge und die Freiheitsgrade begrenzt. Im Gegensatz zu ortsfesten Industrierobotern können bewegliche Industrieroboter sich zusätzlich in der Umgebung bewegen und werden daher durch die Bewegungsvorrichtung begrenzt [Ind].

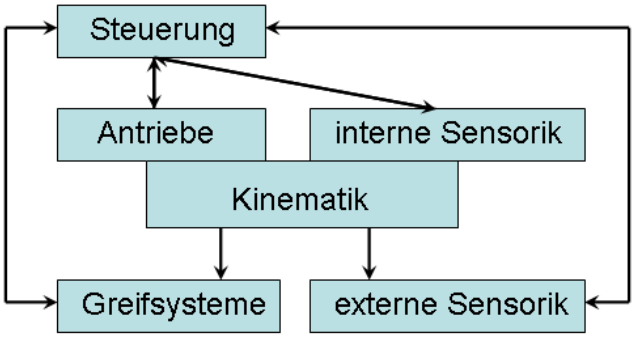

Abbildung 2.1.: Struktur eines Industrieroboters
Quelle: [Ind]

Wie in Abbildung 2.1 zu sehen ist, besteht ein Industrieroboter grundlegend aus einem Roboterarm, welcher auch Manipulator genannt wird, einer Steuerungseinheit, welche für die Überwachungung und Übersetzung der Aktionen auf den spezifischen Industrieroboter zuständig ist und einem Endeffektor (z.B. Greifsystem). Über den Manipulator, welcher Gelenke und Antriebe aufweist, wird die Kinematik realisiert, welche es ermöglicht den Endeffektor im 3D-Raum zu bewegen. Um jeden Punkt im 3D-Raum ansteuern zu können sind mindestens 3 DOF erforderlich, welche über Gelenke realisiert werden. Um jeden Punkt im Raum mit jeder beliebigen Orientierung ansteuern zu können benötigt es jedoch 6 DOF, wobei 3 Gelenke für translatorische und weitere 3 Gelenke für rotatorische Bewegungsabläufe zuständig sind. Der Endeffektor kann auf verschiedene Arten realisiert werden. Zumeist ist es jedoch ein Werkzeug, welches z.B. für Schweiß-, Bohr-, Beschichtungs-, Klebe-, Montage- oder Schneidearbeiten eingesetzt werden kann. Da der Endeffektor austauschbar ist, besteht jedoch aber auch die Möglichkeit einen Greifer oder einen anderen spezifisch für eine Aufgabe konzipierten Endeffektor zu montieren und zu verwenden. Damit die Steuerungseinheit die Gelenkpositionen und Antriebsgeschwindigkeiten ermitteln kann sind zudem Messsysteme erforderlich, welche als interne Sensoren bezeichnet werden. Als optionale Komponente können zudem externe Sensoren beim Industrieroboter verbaut sein, welche unter anderem zur Ermittlung von Objektpositionen im 3D-Raum und deren Größen verwendet werden können. Je nach Arbeitsumfeld kann es auch erforderlich sein, dass ein System zum Wechseln des Endeffektors eingesetzt wird um mehrere Arbeitsschritte mit einem einzigen Industrieroboter durchführen zu können [HS06].

2.1.1. Bauformen

Industrieroboter können entweder zur seriellen oder parallelen Kinematik zugeordnet werden. Der Unterschied besteht darin, dass die Achsen bei der seriellen Kinematik seriell angeordnet sind, wohingegen bei der paralleln Kinematik die Achsen parallel angeordnet werden.

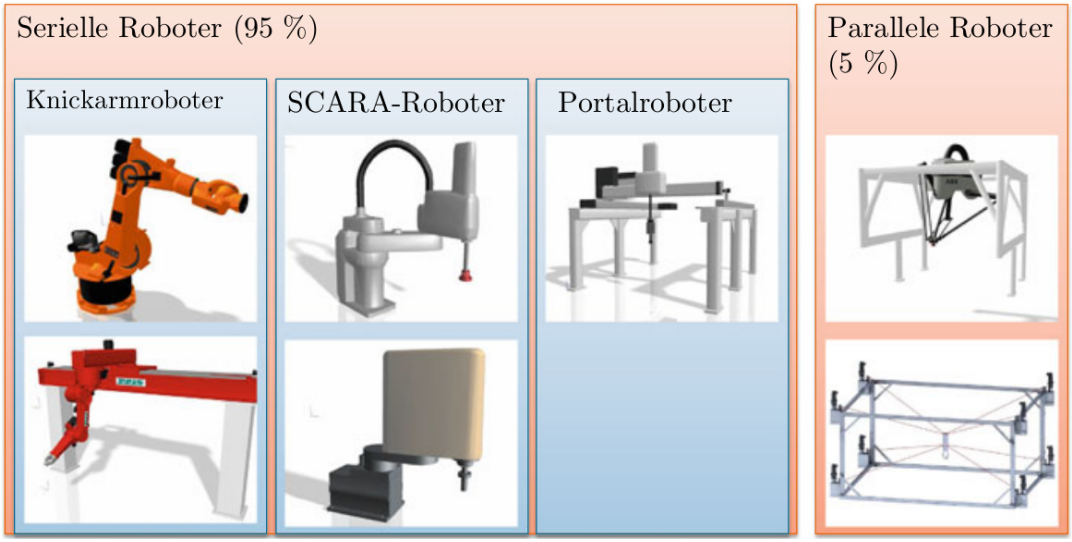

Abbildung 2.2.: Bauformen von Industrierobotern
Quelle: [PD19, S. 18]

Mit 95 % Anteil haben sich Industrieroboter mit serieller Kinematik aufgrund ihrer Bauweise und den bei Knickarmrobotern sehr großen Bewegungsfreiheit durchgesetzt, wie aus der Abbildung 2.2 zu entnehmen ist. Nur ein geringer Teil von 5 % der Industrieroboter besitzen eine parallele Kinematik, zu denen unter anderem der Delta- und Seil-Roboter zählen. In der Abbildung 2.2 ist der Delta-Roboter im Bereich der parallen Roboter oben und der Seilroboter im unteren Bereich zu sehen. Die parallele Kinematik zeichnet sich dabei durch mindestens eine geschlossene kinematische Kette aus. Zu den seriellen Industrierobotern zählen unter anderem Knick-, SCARA- und Portal-Roboter und alle anderen Roboter, welche eine offene kinematische Kette aufweisen. Industieroboter mit serieller Kinematik zählen zu den am weitest verbreitesten Industrierobotern. Im Gegensatz zu den seriellen Industrierobotern bieten parallele Industrieroboter durch die direkte Verbindung zum Greifsystem den Vorteil, dass die nachfolgenden Glieder und deren Massen nicht wie beim seriellen Industrieroboter die Dynamik des Systems beeinflussen können. Dies führt dazu, dass eine sehr hohe Genauigkeit und eine sehr hohe Steifigkeit garantiert werden kann. Zudem sind durch die parallele Kinematik hochpräzise Ansteuerungsmanöver mit sehr hoher Geschwindigkeit durchführbar. Der Nachteil besteht jedoch darin, dass parallele Industrieroboter ortsfest sind. Die parallele Kinematik zählt daher eher zur Ausnahme und wird aus diesem Grund vermehrt für „Pick-and-Place“-Aufgaben oder Sondermaschinen eingesetzt, wo Geschwindigkeit, eine hohe Präzision und hohe Dynamik erforderlich sind [PD19, S. 17 ff.].

Der Knickarmroboter, welcher zu den seriellen Industrierobotern zählt und zumeist 6 DOF aufweist, bietet den Vorteil, dass dieser für weitere Bewegungsfreiheit auf eine zusätzliche Linearachse montiert werden kann. Dadurch ist es unter anderem Möglich verschiedene Arbeitspositionen eines großen Objekts ansteuern zu können. Da der

Knickarmroboter eine armartige Form aufweist, ist dieser zumeist für schwer erreichbare Stellen geeignet. Aufgrund der offenen kinematischen Kette und der Möglichkeit einer sehr großen Nutzlast leidet jedoch die Absolutgenauigkeit darunter. Typischwerweise werden Knickarmroboter für Schweiß-, Handhabungs- und Klebearbeiten eingesetzt. Eine weitere Möglichkeit um serielle Industrieroboter zu realisieren stellen die SCARA-Roboter dar. Diese weisen im Gegensatz zu den Knickarmrobotern weniger Freiheitsgrade auf. Typischwerweise werden zumeist 3 oder 4 DOF verbaut, wodurch die Bewegungsfreiheit eingeschänkt wird. Zudem sind aufgrund der Bauform nur geringere Arbeitslasten und kleinere Arbeitsbereiche nutzbar. Dies zeigt sich auch im Aufgabengebiet, welches großteils aus der Montage und „Pick-and-Place"-Aufgaben besteht. Portale, welche auch zu den seriellen Industrierobotern zählen, können sehr hohe Nutzlasten und große Bewegungen durchführen. Sie sind nach den Knickarmrobotern die zweithäufigste Bauform. Zu den Aufgabengebieten zählen Pick-and-Place, Maschinenbestückungen und Kommisionsarbeiten [PD19, S. 17 ff.].

2.1.2. Positions- und Bahngenauigkeit

Um Genauigkeitsmessungen an einem Industrieroboter durchzuführen ist es erforderlich die spezifischen Genauigkeiten eines Industrieroboters zu kennen und diese in einen Zusammenhang bringen zu können. Bei Industrierobotern ist die Genauigkeit in sehr vielen Aufgabenbereichen, wie z.B. Laserschneiden oder Schweißen, sehr entscheident. Eine hohe Genauigkeit spart Kosten und Zeit ein, da eine teure und zeitintensive Nachbearbeitung entfällt. Aus diesem Grund ist es entscheident, dass der Industieroboter für spezielle Aufgabenbereiche eine hohe Positions- und Bahngenauigkeit aufweist. Um die Positions- und Bahngenauigkeit von Industrierobotern beurteilen zu können verwendet man die Absolut- und Wiederholgenauigkeit. Beide Kennwerte beschreiben die Qualität der Übereinstimmung der Realität und des Modells und sind in der ISO 9283, welche die Robotergenauigkeit beinhaltet und beschreibt, standardisiert [PD19, S. 28 ff.].

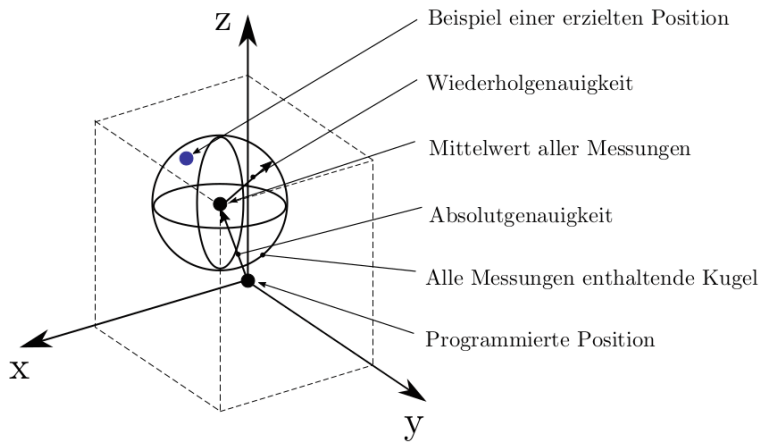

Abbildung 2.3.: Absolut- und Wiederholgenauigkeit
Quelle: [PD19, S. 29]

In der Abbildung 2.3 ist der Zusammenhang zwischen der Absolut- und Wiederholgenauigkeit zu erkennen. Die Wiederholgenauigkeit wird ermittelt indem der Industrieroboter über mehrere Zyklen hinweg immer wieder die gleiche Pose aus einer spezifischen Richtung anfährt. Am Ende werden die dabei entstandenen Abweichungen gemittelt um die Wiederholgenauigkeit zu erhalten. Hierbei beeinflussen systematische Gegebenheiten, wie z.B. termische Ausdehnung durch Umgebungstemperatur, sowie aber auch durch die Motorwärme, Justagefehler der Achsen, Fertigungstoleranzen oder sogar Kollisionen die Genauigkeit des Gesamtsystems um die Zielpose exakt ansteuern zu können. Aus diesen Gründen ist es empfehlenswert den Industrieroboter jährlich auf die Genauigkeit zu prüfen und gegebenenfalls zu justieren. Im Gegensatz zur Wiederholgenauigkeit beschreibt die Absolutgenauigkeit wie genau der Industrieroboter eine theoretisch programmierte Zielpose, welche z.B. mittels CAD-Software erlernt wurde, im Bezug zum Roboterkoordinatensystem im Durchschnitt aus einer beliebigen Richtung ansteuern kann. Standardmäßig beträgt die Wiederholgenauigkeit bei Industrierobotern 0,1 mm wohingegen die Absolutgenauigkeit standardmäßig eine maximale Abweichung von 1 mm aufweist [PD19, S. 28 ff.].

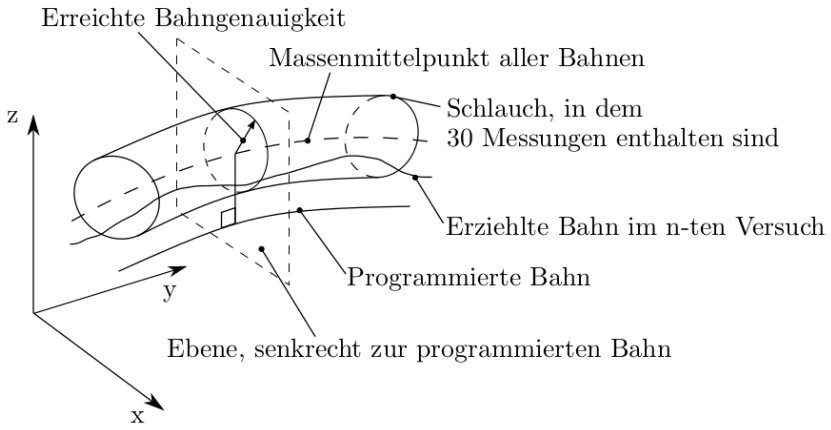

Abbildung 2.4.: Bahngenauigkeit
Quelle: [PD19, S. 30]

Bei einer großen Abweichung der Absolutgenauigkeit führt dies jedoch zu einem schlechteren Erreichen der Zielpose und dadurch auch zu einem schlechteren Bahnverhalten, wie in Abbildung 2.4 durch die erzielte Bahn im n-ten Versuch ersichtlich ist. Die erzielte Bahn weist deutliche Schwingungen auf und weicht daher erkennbar von der zuvor programmierten Bahn ab. Aus diesem Grund bieten viele Hersteller gegen einen Aufpreis absolutvermessene Industrieroboter an um diesen Genauigkeitsfehler zu kompensieren [PD19, S. 29 f.]. Wenn eine sehr hohe Wiederholgenauigkeit gewünscht ist kann auch mit Spezialrobotern eine Wiederholgenauigkeit von bis zu 1 µm erreicht werden [Gen].

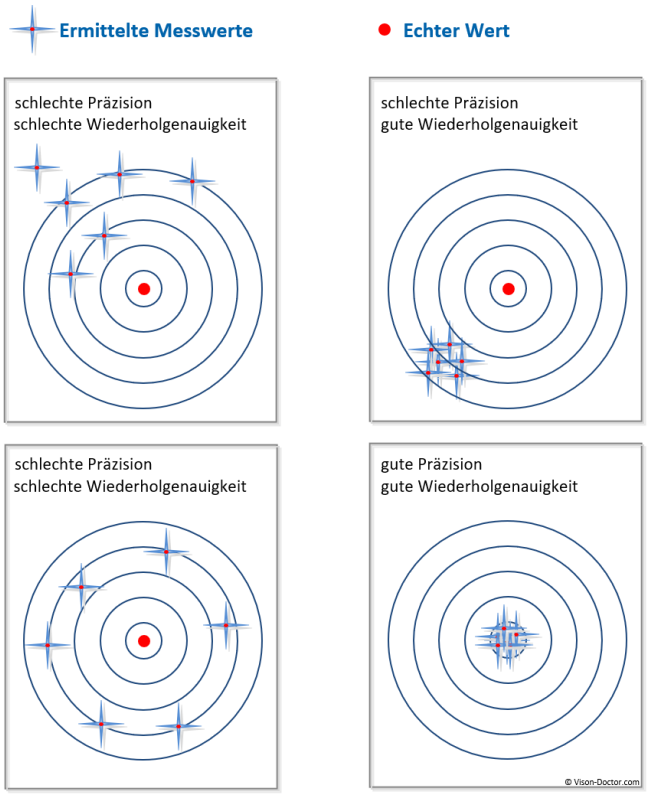

Abbildung 2.5.: Wiederholgenauigkeit
Quelle: [Pra]

Wie in Abbildung 2.5 zu sehen ist, kann daraus geschlossen werden, dass keine hohe Präzision durch eine schlechte Wiederholgenauigkeit möglich ist. Aus diesem Grund kann auch gesagt werden, dass die Wiederholgenauigkeit entscheidenter als die Absolutgenauigkeit zum erreichen der Zielposition ist, da umgekehrt nicht gesagt werden kann, dass eine hohe Präzision durch eine gute Wiederholgenauigkeit erreicht werden kann [Gen].

2.2. Teachen von Industrierobotern

Teachen, welches auch als Teach-In bezeichnet wird, ist eine von mehreren möglichen Varianten einem Industrieroboter gewünschte Posen beizubringen damit dieser seine Arbeitsaufgabe autonom durchführen kann.

2.2.1. Offline- und Online-Programmierverfahren

Unterschieden wird bei den Lernmethoden zwischen Offline- und Online-Programmierverfahren. Die am weitest verbreitesten Offline-Methoden sind die textuelle Programmierung und das Erlernen mit einer CAD-Software. Bei der textuellen Programmierung ist es notwendig über ausreichend Programmierkenntnisse in der Programmiersprache des

Industrieroboter-Herstellers oder einer kompatiblen Programmiersprache zu verfügen, wohingegen bei einer CAD-Software einem virtuellen Industrieroboter in einer virtuellen Umgebung die zu erlernenden Posen beigebracht werden können. Beide Methoden benötigen im Vergleich zum Teachen sehr viel mehr Vorwissen und sind daher nur nach erheblichem Lernaufwand nutzbar. Das Teachen zählt neben der Playback-Methode, welche ein direktes Führen des Roboterarms ermöglicht, zu den Online-Programmierverfahren. Beim Teachen kommt eine Steuerkonsole zum Einsatz mit deren Hilfe es möglich ist dem Industrieroboter Posen beizubringen. Zuerst wird eine Pose mit mehreren aufeinander folgenden Steuerungsbefehlen angefahren. Anschließend, wenn eine gewünschte Pose erreicht wurde, kann diese einzelne Pose per Steuerungsbefehl in der Steuerungseinheit gespeichert werden. Das Anfahren und Speichern der Posen wird solange wiederholt bis die gesamte Arbeitsaufgabe dem Industrieroboter beigebracht wurde. Wenn notwendig, können im Nachhinein die gespeicherten Posen korrigiert werden um eine höhere Genauigkeit zu erzielen oder um Fehler zu korrigieren. Im Gegensatz zum Teachen wird beim Playback-Verfahren genau die Bewegung gespeichert, welche durch das direkte Führen des Roboterarms entsteht, und beim Ausführen der Arbeitsaufgabe auf die gleiche Weise wiederholt [Ind]. Beim Teachen hingegen kann zwischen den Zielpunkten die Geschwindigkeit, Beschleunigung und sogar die Art des Pfades eingestellt werden. Bei der Art des Pfades kann frei zwischen Point-to-Point und Continuous Path gewählt werden. Point-to-Point erzwingt den für den Industrieroboter geometrisch günstigsten Pfad zwischen den Punkten zu wählen, welcher normalerweise keine gerade lineare Bahnbewegung darstellt und daher zu einem unbekannten Pfad führt. Bei Continuous Path wird hingegen von einem Punkt direkt zum nächsten Punkt entlang eines vordefinierten Pfades gefahren, welcher z.B. eine gerade lineare oder eine durch einen Hilfspunkt erzeugte kreisförmige Bahnbewegung sein kann. Durch diese Einstellungsmöglichkeiten ist es beim Teachen möglich noch höhere Genauigkeiten als bei der Playback-Methode zu erzielen. Im Gegenzug steigt durch die höhere Genauigkeit, aber auch der Programmieraufwand, da für eine hohe Genauigkeit viele Zwischenpunkte beigebracht werden müssen [Tea].

Für alle Programmierverfahren, ob Offline- oder Online-Programmierverfahren, welche zur Programmierung von Industrierobotern eingesetzt werden können, kann gesagt werden, dass jedes Verfahren einen gewissen Lernaufwand bei der Einarbeitung benötigt. Der Lernaufwand um ein Programmierverfahren zu erlernen sinkt jedoch erheblich für untrainierte Personen, wenn eine visuelle Lernmethode eingesetzt wird. In Zukunft besteht daher ein sehr großes Potenzial zur Erforschung und Verbesserung von Gesten, Sprach und visuellen Systemen, da diese Systeme eine für den Menschen vertraute Interaktionsmöglichkeit bieten [BM].

2.2.2. Teach Pendant

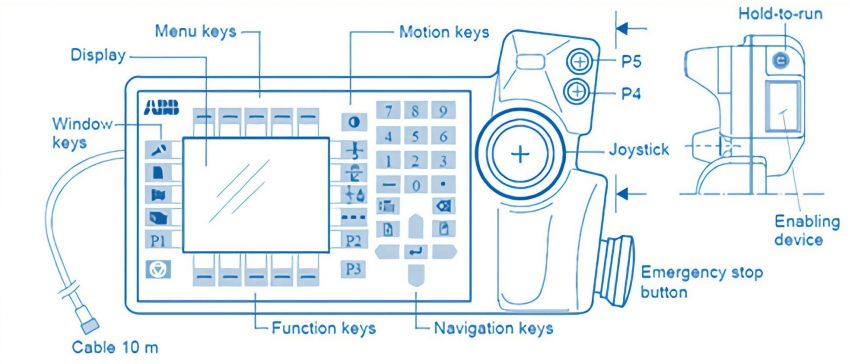

Abbildung 2.6.: Teach Pendant im Querformat von ABB
Quelle: [Pro]

Zur Programmierung mit dem Teach-In-Verfahren werden Teach Pendants, wie in Abbildung 2.6 zu sehen, eingesetzt, welche im Gegensatz zu früheren Modellen je nach Hoch- oder Querformat im oberen oder im linken Bereich des Teach Pendants über ein Display mit oder ohne Touch-Funktionalität verfügen [Nof99]. Unterschieden wird bei den herstellerspezifischen Teach Pendants zwischen dem Hoch- und Querformat. Beim Hochformat wird das Teachpendant mit beiden Händen gehalten und mithilfe der Daumen werden die Knöpfe betätigt. Im Gegensatz zum Hochformat ist in einem Teach Pendant mit Querformat ein größeres Display verbaut und das Teach Pendant kann im Querformat mit beiden Händen oder nur mit der linken oder rechten Hand gehalten werden. Die rechte oder linke Hand kann dazu verwendet werden Eingaben auf dem Teach Pendant zu tätigen oder mittels des verfügbaren 3 DOF Joysticks oder der 6 DOF 3D-Maus zum Steuern des Industrieroboters eingesetzt werden. Beim Betätigen des Joysticks oder der 3D-Maus reagiert der Industrieroboter bereits bei sehr feinen Bewegungen des Eingabegerätes mit einer entsprechenden Bewegung in die jeweilige vorgegebene Richtung. Je weiter der Joystick oder die 3D-Maus in eine Richtung bewegt wird, desto schneller bewegt sich auch der Industrieroboter in die ensprechende Richtung [Pra+04, S. 48 f.]. Jedes Teach Pendant muss einen Notstop-Schalter aufweisen um die Maschine und das Teach Pendant jederzeit außer Betrieb setzen zu können. Dieser Notstop-Schalter ist zumeist rechts oben oder rechts unten, wie in Abbildung 2.6 zu sehen ist, als große rote Taste realisiert. Als weitere Schutzvorkehrung muss das Teach Pendant einen „Enable device“-Schalter aufweisen. Der „Enable device“-Schalter wird Umgangssprachlich auch „Dead Man’s Switch“ genannt und muss während die Maschine bedient wird ununterbrochen gehalten werden. Dabei ist zu beachten, dass wenn der Notstop-Schalter losgelassen oder zu fest gedrückt wird, der Industrieroboter hierdurch sogleich anhält. Wenn der „Enable device“-Schalter zu fest gedrückt oder losgelassen wird, geht das Teach Pendant umgehend davon aus, dass die Person, welche die Kontrolle über das Teach Pendant hat, bewusstlos, tot oder sogar nicht mehr an der Maschine ist. Durch das Auslösen dieses Sicherheitsschalters wird der Industrieroboter zum Anhalten gebracht

und weiterer Schaden und gefährliche Situationen verhindert [Dea].

Weitere Tasten wie z.B. Menü, Funktionen, Bewegung und Navigation, wie in Abbildung 2.6 zu sehen ist, sind herstellerspezifisch und daher optional. Aus diesem Grund kann ein Teach Pendant je nach Hersteller unterschiedliche herstellerspezifischen Funktionstasten aufweisen, welche zumeist Zugriff auf die am häufigsten verwendeten Funktionalitäten des Teach Pendants bieten [Din15]. Neben diesen Sicherheitsfeatures gibt es auch herstellerübergreifende Features um den Industrieroboter zu steuern. Eines dieser Features ist das Wechseln zwischen unterschiedlichen Bewegungsmodi. Zu diesen Bewegungsmodi zählen der Gelenk-, Weltkoordinatensystem- und Tool-Koordinatensystem-Modus. Der Gelenk-Modus erlaubt es die Positionen der Gelenke mithilfe des Teach Pendants jeweils einzeln oder alle Gelenke zusammen zu bewegen. Dieser Modus ermöglicht daher die volle Kontrolle über alle Gelenke des Industrieroboters zu erlangen und diese an die jeweiligen Gegebenheiten der Umgebung anzupassen. Der Gelenk-Modus wird daher häufig zum Erreichen von schwer zugänglichen Stellen eingesetzt. Im Gegensatz zum Gelenk-Modus erlaubt der Weltkoordinatensystem-Modus den Mittelpunkt des Endeffektors (TCP) in der x-, y- oder z-Achse zu bewegen und Rotationen um die x-, y- oder z-Achse im Weltkoordinatensystem durchzuführen. Der Sockel des Industrieroboters stellt den Mittelpunkt des Weltkoordinatensystems dar. Für diesen Modus werden die Positionen der Gelenke über die inverse Kinematik, ermittelt und automatisch vom Teach Pendant eingestellt [Jog]. Wenn bei Knickarmrobotern die Pose des Endeffektors bei der Berechnung der inversen Kinematik berücksichtigt wird, kann es je nach gewählter Pose des Endeffektors bis zu acht mögliche Gelenksstellungen für den Knickarmroboter geben. Hierdurch kann die inverse Kinematik für diesen Fall nicht mehr eindeutig für die gewählte Pose bestimmt werden [PD19, S. 25]. Die letzte Möglichkeit den Industrieroboter einzustellen besteht darin, den TCP des Industrieroboters im Tool-Koordinatensystem-Modus einzustellen. Der Tool-Koordinatensystem-Modus besitzt die gleichen Möglichkeiten, wie der Weltkoordinatensystem-Modus, verwendet jedoch aber als Referenzkoordinatensystem den TCP [Jog].

2.3. Arten von Echtzeitverhalten

Im Umgang mit Industrierobotern ist es notwendig Echtzeitsysteme einzusetzen, welche für den jeweiligen spezifischen Anwendungsfall die Anforderung an die Reaktionszeit des Systems sicherstellen. Die Anforderung an das Echtzeitsystem können vielseitig sein. Diese können bei sehr zeitkritischen und sicherheitsrelevanten Systemem wenige Millisekunden und bei weniger relevanten Aufgaben, wie z.B. das Speichern von Daten auf eine Festplatte, bis zu mehrere Minuten, Stunden, Tage oder sogar länger in Anspruch nehmen. Die entscheidente Herausforderung für das Echtzeitsystem ist es deshalb zu garantieren, dass das Ergebnis in dem für den Anwendungsfall erforderlichen Zeitintervall geliefert wird. Die möglichen Zeitintervalle werden durch die Hardware limitiert, wordurch eine für den Anwendungsfall entsprechend schnelle Hardware erforderlich ist. Anwendungsfälle mit genügend kleinen Zeitinvervallen können daher nur mit entspre-

chend schneller Hardware realisiert werden [Ech].

2.3.1. Echtzeitanforderungen

Bei Echtzeitsystemen wird zwischen harter, fester und weicher Echtzeitanforderung unterschieden. Unter harter Echtzeitanforderung versteht man, dass das Ergebnis innerhalb des vorgegebenen Zeitintervalls erfolgen muss, da ansonsten das System versagt hat. Bei Nichteinhaltung des Zeitintervalls besteht ansonsten die Möglichkeit, dass es zu einem Schaden kommen kann. Als Beispiel kann die Notbremsung eines autonom fahrenden Autos eine harte Echtzeitbedingung darstellen. Im Gegensatz zur harten Echtzeitanforderung stellt die feste Echtzeitanforderung eine etwas gelockertere Echtzeitanforderung dar. Bei der festen Echtzeitanforderung wird davon ausgegangen, dass kein unmittelbarer Schaden zustande kommt, wenn das vorgegebene Zeitintervall überschritten wird. Daher kann das vorgegebene Zeitintervall problemlos bei der festen Echtzeitanforderung überschritten werden. Das Ergebnis wird beim Überschreiten des Zeitintervalls jedoch zumeist ungültig und weitere darauf aufbauende Berechnungen können daher abgebrochen werden. Ein Beispiel für eine feste Echtzeitanforderung kann z.B. sein, dass die derzeitige Position eines Fahrzeugs nach dem Weiterfahren nicht mehr die gleiche Position ist und Berechnungen auf diesen Daten schlicht zu falschen Ergebnissen führen würden. Aus diesem Grund würde es keinen Sinn machen mit den veralteten Daten weiterzuarbeiten. Bei weichen Echtzeitanforderung ist das Zeitintervall für die Bearbeitung der Aufgabe als Richtwert zu sehen und kann bei der Abarbeitung der Aufgabe wenig oder sehr stark überschritten werden. Im Mittel werden die Aufgaben jedoch zeitgerecht abgearbeitet. Bei Multimediasystemen führen z.B. geringe Überschreitungen des Zeitintervalls zu verzögert wiedergegebenen Bildern. In einem gewissen Rahmen sind diese Verzögerung jedoch akzeptabel. Ab dem Zeitpunkt ab dem das Video anfängt zu stottern und auszusetzen und die Verzögerung von Menschen wahrgenommen werden kann ist zumeist die Akzeptanzgrenze bereits weit überschritten [Wö06, S. 321 f.].

Um die harte, feste oder weiche Echtzeitanforderungen auf konventionellen Computern mit mehreren Prozessen umsetzen zu können werden sogenannte Echtzeitbetriebssysteme (RTOS) eingesetzt. Bei einem Echtzeitbetriebssystem stellt der RTOS-Kernel die notwendige Komponente dar, welche die Echtzeitanforderung realisiert[Wö06, S. 343 f.]. Ein Kernel ist die grundlegene Komponente eines Betriebssystems, welche die Hardwareabstraktion, Prozessverwaltung, Speicherverwaltung und andere wiederverwendbare Ressourcen bereitstellt um die Entwicklung von Programmen zu erleichtern [Wö06, S. 356]. Im Gegensatz zu einem Betriebssystem ohne Echtzeitanforderung ist es bei einem Echtzeitbetriebssystem erforderlich, dass das Prozess-Scheduling, welches bestimmt wann und wie viel Zeit ein Prozess erhält, deterministisch abläuft, sodass die Prozesse die Ergebnisse zeitgerecht im benötigten Zeitintervall erhalten. Bei einer Echtzeitanwendung muss nicht nur der Prozess-Scheduler die Echtzeitfähigkeit unterstützen, sondern auch die Anwendung explizit die Echtzeitanforderungen beachten um vollen Gebrauch von der Echtzeitfähigkeit des Systems machen zu können. Neben dem Prozess-Scheduler können auch Ein- und Ausgaben, zu denen unter anderem das Betätigen von Tasten, die

Netzwerkkommunikation und das Schreiben und Lesen von Daten von einem Speichermedium gezählt werden, die Echtzeitfähigkeit eines Systems beeinflussen. Aus diesem Grund sollte der virtuelle Speicher, welcher zum Auslagern auf die Festplatte oder SSD verwendet werden kann, deaktiviert sein um nicht-deterministisches Verhalten zu vermeiden. Für Echtzeitsysteme ist es auch erfordlich, dass der Bedarf an Ressourcen, welches den verfügbaren Arbeitsspeicher miteinschließt, bekannt ist. Zumeist wird dies erreicht indem im Vorhinein genügend Arbeitsspeicher für die Anwendung reserviert wird. Das Reservieren von Arbeitsspeicher oder das Aufrufen von Kernelfunktionen ist ein sehr aufwändiger Prozess, welcher die Anwendung über das Zeitinvervall hinaus blockeren kann. Daher sind diese Funktionalitäten vor Benutzung mit Bedacht abzuwägen. Zudem ist von der Verwendung von Laufzeitumgebungen mit einem Garbage Collector, welcher zum Freigeben von reserviertem aber nicht mehr verwendetem Arbeitsspeicher eingesetzt wird, abzusehen [Ech]. Ein Garbage Collector ist für die automatische Speicherbereinigung einer Anwendung zuständig um so reservierte aber nicht mehr verwendete Speicherbereiche im Arbeitsspeicher freizugeben. Dieser Vorgang kann jedoch nur durchgeführt werden, wenn die Anwendung von der Laufzeitumgebung unterbrochen und somit blockiert wird [Gar].

2.3.2. Implementationen gängiger Betriebssysteme

Die gängigen Kernel für weit verbreitete Betriebssysteme, zu denen XNU für macOS [Xnu], Windows NT für Windows, Linux für linuxbasierte Distributionen und der FreeBSD-Kernel für das FreeBSD-Betriebssystem zählen, verfügen nur über eingeschränkte weiche Echtzeitfähigkeit, welche lediglich nicht-deterministisches Verhalten garantieren können. Dies ist dem Umstand geschuldet, dass Normalanwender kein echtzeitfähiges System benötigen und hingegen ein energiesparsames System fordern, welches als Mehrzweck-Betriebssystem für ein breites Aufgabenspektrum eingesetzt werden kann. Echtzeitfähigkeit und Energiesparsamkeit stehen jedoch im direkten Konflikt zueinander, da das Echtzeitsystem jederzeit Ressourcen für anfallende Aufgaben bereithalten muss [Wha]. Linux bietet im Vergleich zu XNU, Windows NT oder dem FreeBSD-Kernel integrierte Prozess-Scheduler an um weiche und feste Echtzeitfähigkeit für Anwendungen zu garantieren. Die echtzeitfähigen Prozess-Scheduler müssen jedoch zuerst aktiviert werden, da standardmäßig der Completely Fair Scheduler eingesetzt wird. Die auswählbaren echtzeitfähigen Prozess-Scheduler sind SCHED_FIFO, welches auf dem „First In - First out“-Prinzip basiert, SCHED_RR, welches ein echtzeitbasiertes „Round Robin“-Verfahren realisiert und SCHED_DEADLINE, welches auf dem „Earliest Deadline First“-Prinzip basiert und eine Weiterentwicklung von SCHED_FIFO und SCHED_RR darstellt [Sch]. Zudem soll in Zukunft durch Integration der Real-Time-Patches in den Hauptzweig von Linux der Kernel von Haus aus über harte Echtzeifähigkeit verfügen. Hierzu wurden bereits weitreichende Änderungen in Linux 5.3 übernommen, welches den Weg in diese Richtung ebnen soll [Lee]. Abseits von Linux mit seinen Real-Time-Patches verfügt auch der „Windows CE“-Kernel über harte Echtzeitfähigkeit [CHA13]. Der „Windows CE“-Kernel wurde speziell für Embedded Systeme, Thin Clients und eine Vielzahl von mobilen Geräten entwickelt [Hal05]. Anzumerken ist jedoch, dass der „Windows

CE“-Kernel nicht mit dem „Windows NT“-Kernel verwechselt werden sollte und zuletzt im Juni 2013 aktualisiert wurde [Mica]. Der „Windows NT“-Kernel kann aber wie Linux mit einer Echtzeit-Erweiterung, wie z.B. Real Time eXtensions for Windows oder durch iNtime, um harte Echtzeitfähigkeiten erweitert werden [Mar04]. Im Normalfall verfügen abgespeckte Echtzeitbetriebssysteme im Vergleich zu Linux oder Windows, welche mit Echtzeiterweiterungen ausgestattet werden können, über eingeschränkte Funktionalität, wodurch die Entwicklungszeit von Anwendungen drastisch ansteigt. Es ist daher empfehlenswert ein Echtzeitbetriebssystem mit zusätzlichen Funktionalität, wie z.B. Linux oder Windows mit Echtzeiterweiterungen, zu verwenden um die Entwicklungszeit drastisch zu verkürzen [Ech, S. 433].

2.4. Sicherheitsanforderungen im Umgang mit Industrierobotern

Die Sicherheitsanforderungen im Umgang mit Industrierobotern sind den Maschinenrichtlinien zu entnehmen, welche auch Industrieroboter miteinschließen. Hierzu gibt es unter anderem die europäische Richtlinie, welche in 2006/42/EG definiert ist, und eine internationale Norm, welche in der ISO EN 10218 festgehalten wurde [Ind]. Die wichtigsten Aspekte der „ISO EN 10218“-Norm sind das Einhalten von einem ausreichenden Sicherheitsabstand zur Maschine, wenn möglich das räumliche Trennen von Mensch und Maschine und die Verringerung der Beschleunigung und Geschwindigkeit der Maschine auf ein für den Menschen ungefährliches Niveau. Diese Sicherheitsvorkehrung ist einzuhalten, wenn ein Mensch die Maschine im Betriebsmodus Handbetrieb bedient oder sich sogar in unmittelbarer Nähe befindet. Im Betriebsmodus Automatik müssen die Sicherheitsvorkehrungen wirksam sein und das Betreten durch Menschen unterbunden werden. Die räumliche Trennung kann hierbei mittels Schutzgitter sichergestellt werden. Das Schutzgitter darf zudem nur über eine Schutztüre betreten werden können. Wenn die Schutztüre geöffnet oder eine Lichtschranke unterbrochen wird, muss außerdem sichergestellt werden, dass der Industrieroboter sofort anhält. Zudem muss das Handbediengerät eine Notstop-Funktion aufweisen und ein unabsichtliches Betätigen eines Schalters auf dem Handbediengerät, welches den Industrieroboter zu unvorhergesehenen Manövern bewegen würde, augeschlossern werden können. Für den Menschen sollte es zudem ersichtlich sein in welchem Zustand sich die Maschine befindet um gefährliche Situationen vermeiden zu können. Es muss zudem sichergestellt werden, dass die Notstop-Funktion und weitere personenbezogene Sicherheitsfunktionen durch redundante Steuerkreise jederzeit funktionsfähig bleiben [BM08]. Um mit der Maschinenrichtlinien 2006/42/EG und der „ISO EN 10218“-Norm im Einklang zu stehen muss aber auch sichergestellt werden, dass das Handeingabegerät, welches auch kabellos betrieben werden kann, eine deterministische Eingabemöglichkeit bietet. Die Latenz des Kommunikationskanals des Eingabegerätes muss daher weniger als 50 ms betragen und ein deterministisches Verhalten aufweisen, damit die personenbezogenen Sicherheitsfunktionen auch jederzeit funktionsfähig bleiben. Hierzu eignet sich ein Echtzeitbetriebssystem, welches diese ge-

ringe Latenz von unter 50 ms mit einem deterministischen Verhalten und einer festen Echtzeitanforderung sicherstellen kann [Pra+04, S. 55].

2.5. ROS

Das Robot Operating System, welches auch kurz ROS genannt wird, ist kein Betriebssystem im eigentlichen Sinne. Es ist vielmehr ein Framework, welches es ermöglicht die Roboterentwicklung zu vereinfachen und modularer zu gestalten. Vor allem die Modularität ist die größte Stärke von ROS, da hierdurch wiederverwendbare Packages entstanden sind. Packages reichen hierbei von GUI-Tools bis hin zu wiederverwendaren Algorithmen, welche früher für jeden Roboter aufwendig von Grund auf neuprogrammiert werden mussten. Dadurch stieg das Bedürfnis nach wiederverwendaren Softwarebausteinen um die Roboterentwicklung zu vereinfachen [Wö06, S. 517]. ROS verwendet für die Roboterentwicklung hierzu die Stärke des „Open Source"-Ökosystems, welches es ermöglicht ein komplexes System mit vielen über die Welt verteilten und unabhängigen Entwicklern zu gestalten. Die unabhängigen Entwickler können wiederum von ROS profitieren und durch die Arbeiten der ROS-Community hohe Entwicklungskosten einsparen. Die Offenheit des Quellcodes und die lebendendige ROS-Community sind daher entscheidente Gründe warum ROS so erfolgreich wurde [Kou19, S. 411 f.].

Zurzeit ist ROS in Version 2 verfügbar, jedoch aber aufgrund vieler noch nicht auf ROS 2 portierter Packages sehr eingeschränkt nutzbar. Es wird daher von der offiziellen ROS-Seite[1] empfohlen ROS 1 zu verwenden, da es von ROS 2 nur Vorabversionen gibt. Zudem können bei ROS 2 zwischen den Versionen noch inkompatible Änderungen stattfinden [Rosb]. Darüber hinaus ist es empfehlenswert auf die LTS-Version Melodic Morenia zu setzen, welche bis April 2023 offiziell Langzeitunterstützung erhält [Roba]. In Zukunft soll ROS 2 im Gegensatz zu ROS 1 Echtzeitunterstützung und Sicherheitsfeatures, welche es verhindern sollen ROS-Messages zu manipulieren, erhalten. Manipulierte ROS-Messages könnten ansonsten von Angreifern ausgenützt werden um andere Menschen in gefährliche Situationen zu bringen [Kou19, S. 613 ff.]. Derzeit wird aufgrund des sehr Sicherheitslücken behafteten Softwaredesigns von ROS 1 empfohlen ein speziell für einen Roboter erstelltes Netzwerk für ROS 1 zu verwenden um Manipulationen von einem Angreifer ausschließen zu können [Poh14]. Wie schon erwähnt unterstützt ROS 1 keine harte Echtzeitfähigkeit (siehe Kapitel 2.3.1), kann jedoch aber mit echtzeifähigen Komponenten zusammenarbeiten [Roba].

[1] https://www.ros.org/

2.5.1. Softwarearchitektur

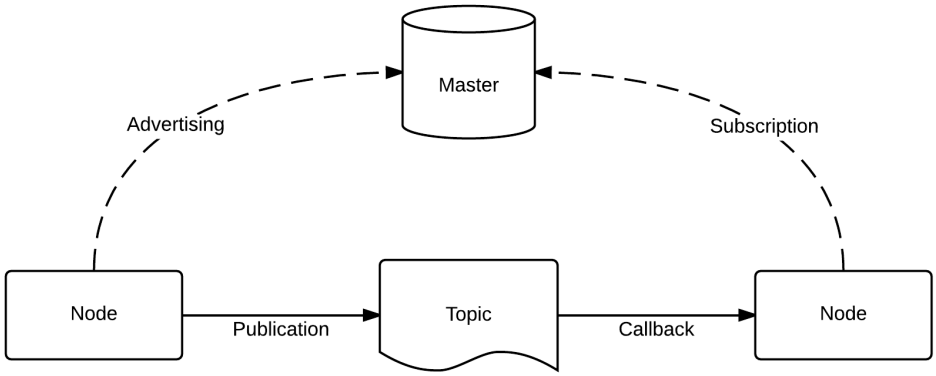

Abbildung 2.7.: Ros Architektur
Quelle: [Rosd]

Wie in Abbildung 2.7 zu sehen ist, besitzt ROS eine serviceorientierte Architektur, welche dazu verwendet werden kann um über ein Netzwerkprotokoll zu kommunizieren. Die Kernkomponente stellt der ROS-Master dar, welcher als zentrale Anlaufstelle für alle ROS-Nodes dient und die Kommunikation der verschiedenen ROS-Nodes untereinander überhaupt erst ermöglicht. Ein ROS-Node ist eine Komponente, welche eigenständig Aufgaben, wie z.B. das Steuern eines Industrieroboters oder Berechnungen, ausführen und diese gegebenenfalls über den ROS-Master an anderen ROS-Nodes weiterleiten kann. Die Kommunikation erfolgt dabei über sogenannte ROS-Messages, welche je nach Nachrichtenart eine andere Datenstruktur aufweisen. ROS stellt standardmäßig bereits vordefinierte Datenstrukturen zur Verfügung, wobei jedoch auch selbstdefinierte ROS-Messages mit eigenen Datenstrukturen nach Belieben für eigene Anwendungsfälle erstellt werden können. Bei der Kommunikation zwischen den ROS-Nodes kann zwischen Topics und Services unterschieden werden, welche je nach Anwendungsfall eingesetzt werden sollten. Topics werden eingesetzt, wenn eine lose Kopplung notwendig ist, bei der das Programm die Anfragen und Antworten asynchron abarbeiten soll. Ebenso wie Topics werden auch Services eingesetzt, wenn eine lose Kopplung gewünscht ist. Bei Services erfolgt die Anfrage und die Antwort jedoch nicht asynchron und es wird daher auf die Antwort blockierend gewartet. Es ist ersichtlich, dass Topics dann eingesetzt werden sollten, wenn keine direkte Antwort auf die geschickte Anfrage erforderlich ist oder während der Bearbeitung der Anfrage andere Aufgaben vom Sender der ROS-Message durchgeführt werden sollen. ROS-Nodes, Topics und Services müssen im Vorhinein bevor sie verwendet werden können beim ROS-Master kundgemacht werden. Dieser Kundmachungsprozess um dem ROS-Master den ROS-Node, Topic oder Service über einen eindeutige Namen verfügbar zu machen nennt man Advertising. Wenn man einen Topic oder Service in ROS nutzen will, muss man sich zuvor auf diesen registrieren. Dieser Registrierungsprozess erfolgt über eine Subscription. ROS-Messages können nach dem Registrieren auf einen Topic

gepublished oder per Service-Aufruf übertragen werden. Wie in Abbildung 2.7 zu sehen ist werden alle Subscriber, welche sich auf einen Topic registriert haben, nach dem Erhalt der ROS-Message mit der gleichen ROS-Message benachrichtigt. Topics erlauben es daher ROS-Messages von vielen Publishern zu vielen Subscribern zu schicken. Services hingegen erlauben nur eine Eins-zu-eins-Relation von Publisher und Subscriber [Rosa].

2.5.2. Interoperabilität

Aufgrund des Softwaredesigns von ROS, welches es ermöglicht die Aufgaben auf unterschiedliche Rechner zu verteilen und diese über ein Netzwerkprotokoll ansprechen zu können, ist es überhaupt erst möglich ROS unabhängig von der Programmiersprache und der Systemarchitektur verwenden zu können [Rose]. Zudem wird ROS auf den gängigen Betriebssystemen, zu denen Windows, Linux und macOS zählen, unterstützt. Die Interoperabilität um von einer „ROS 2“-Komponente auf eine „ROS 1“-Komponente zugreifen zu können ist jedoch noch nicht vollständig gegeben und soll in der weiteren Entwicklung von ROS 2 weiter verbessert werden [Rosc]. Die weitere Stärke von ROS liegt auch in der Austauschbarkeit von Komponenten, da diese gegen eine stabile und standardisierte Schnittstelle programmiert werden können [Robb].

2.6. Tiefenkamera

Zur Messung von Abständen im räumlichen Raum können Tiefensensoren, welche in Tiefenkameras zu finden sind, eingesetzt werden. Unterschieden wird dabei zwischen aktiven und passiven Infrarot-Sensoren [Und].

2.6.1. Funktionsweise

Eine Tiefenkamera, auch RGB-D Kamera genannt, liefert neben Tiefeninformationen, auch RGB-Farbbilder. Die Tiefeninformationen werden hierbei über eine Tiefenkamera und die RGB-Farbbilder über eine RGB-Kamera aufgenommen. Da zwei verschiedene Systeme zum Einsatz kommen, können auch die Auflösungen der Tiefeninformation und des RGB-Farbbildes verschieden sein [Mö12]. Der aktive Tiefensensor verwendet zur Ermittlung der Distanzen einen in regelmäßigen Abständen ausgesendeten Infrarot-Lichtimpuls [Und]. Die Zeit die das Licht benötigt um das Objekt zu erreichen und wieder zurück zu reflektieren ist dabei proportional zur vorhandenen Distanz und liegt z.B. bei einer Distanz von einem 2,5 m entfernten Objekt und zurück, aufgrund der Lichtgeschwindigkeit von 299,710 km/s bei nur 16,7 ns. Bei der Analyse von Latenzen im Umfeld von Tiefensensoren ist daher diese Zeit im Vergleich zu den anderen Komponenten, wie z.B. Berechnungen auf den Tiefeninformationen, vernachlässigbar klein [Tof]. Es sollte jedoch aber die Belichtungszeit mit eingerechnet werden, da diese bereits im Millisekunden-Bereich liegen und so die Echtzeitfähigkeit eines Systems massiv beinträchtigen kann. Zudem gibt es passive Tiefensensoren, welche die Infrarot-Strahlung von Objekten nur absorbieren, um so die Dinstanz zu messen [Und]. Lichtsensoren

können je nach Model eine Messdistanz von mehreren Dezimetern bis hin zu 40 m unterstützen. Je besser die Auflösung des Tiefensensors ist, desto besser ist zudem auch die Auflösung der ermittelten Distanzen. Der Vorteil der derzeit verfügbaren Tiefenkameras und Tiefensensoren liegt darin, dass diese schnelle und in den meisten Fällen auch zuverlässige Distanzmessungen ermöglichen. In Ausnahmezuständen, wenn unter anderem Totalreflexion gegeben ist, mehrere gleichzeitig betriebene Tiefensensoren eingesetzt werden oder bei zu starker Sonneneinstrahlung, können die Ergebnisse der Tiefenkamera stark verfälscht werden. Aus diesem Grund sollte beim Einsatz von Tiefensensoren darauf geachtet werden, dass diese besonderen Umstände nicht auftreten. Bei der Programmierung mit Tiefenkameras sollte zudem beachtet werden, dass der empfohlene Abstand je nach Tiefenkamera eingehalten wird, da ansonsten die Tiefenkamera nicht wie gewünscht funktionieren kann. Außerdem sollte beachtet werden, dass Objekte, welche z.B. von einer Hand verdeckt werden, nicht von der Tiefenkamera erfasst werden können [Tof].

2.6.2. Azure Kinect

Die Azure Kinect ist als Entwicklerkit verfügbar und integriert eine 12-Megapixel-RGB-Kamera, eine 1-Megapixel-Tiefenkamera, ein räumliches Mikrofon, welches aus sieben in Verbund geschaltenen Mikrofonen besteht, welche in einem Kreis hexagonal und in dessem Mittelpunkt angeordnet sind, und einem Orientierungssensor um Aufgaben in unterschiedlichen industriellen Bereichen, wie z.B. Robotik, Verkauf, Logistik und Gesundheitswesen, durchführen zu können. Aus diesem Grund bietet die Azure Kinect auch die Möglichkeit, mehrere Azure Kinects über Synchronisationspins miteinander zu verbinden. Im Gegensatz zu den vorherigen Iterationen der Kinect, welche die Kinect 1 und Kinect 2 miteinschließen, bietet die Azure Kinect einen höher aufgelösten Tiefensensor und eine höher aufgelöste RGB-Kamera. Zudem richtet sich die Azure Kinect im Gegensatz zu den vorherigen Iterationen vielmehr an industrielle Anforderungen anstatt an die Anforderungen der Spielebranche, wie auch die Anbindung an die Azure Cloud darauf schließen lässt. Die Azure Cloud kann optional mit der Azure Kinect verwendet werden und bietet vor allem die Möglichkeit an, AI-Aufgaben in die Azure Cloud auszulagern [Azua]. Die Azure Kinect benötigt zwingend einen Rechner mit Windows 10 oder mindestens Ubuntu 18.04 als Betriebssystem und eine Hardwareausstattung mit mindestens einem „Intel Core i3“-Prozesser der siebten Generation, einen „USB 3.0“-Port und 4 GB Arbeitsspeicher um in der Basiskonfiguration ohne zusätzliche Features, wie z.B. Body Tracking, funktionsfähig zu sein. Ein Betrieb der Azure Kinect ohne einen zusätzlichen Rechner ist nicht vorgesehen [Azub]. Die Azure Kinect kann in einem Temperaturbereich von 10 - 20 °C und im Bereich von 8 bis 90 % der relativen Luftfeuchtigkeit betrieben werden, wobei jedoch keine Kondensation stattfinden darf. Der Tiefensensor kann in einer Distanz von 0,25 bis maximal 5,46 m je nach verwendetem Tiefensensor-Modus eingesetzt werden, wobei jedoch immer die Reflexionsfähigkeit von Objekten beachtet werden muss. Daher können je nach Objekt auch geringere oder größere Distanzen möglich sein. Der Blickwinkel variiert zudem je nach verwendetem Tiefensensor-Modus zwischen 75°x65° bis hin zu 120°x120°. Im aktiven Modus, welcher zur Ermittlung der Distanzen

einen in regelmäßigen Abständen ausgesendeten Infrarot-Lichtimpuls auswertet, benötigt die Tiefenkamera für die Belichtungszeit je nach Tiefensensor-Modus zwischen 12,8 und 20,3 ms. Im passiven Modus, bei dem der Tiefensensor die Infrarot-Strahlung von Objekten ausschließlich absorbiert, beträgt hingegen die Belichtungszeit der Tiefenkamera nur 1,2 ms [tesb].

2.7. Vorhandene Gestenerkennungssysteme

An der FH Vorarlberg wurden bereits zwei wissenschaftliche Arbeiten zur Gestenerkennung erstellt. Eine dieser Arbeiten wird derzeit in einem laufenden Forschungsprojekt an der FH Vorarlberg weiterentwickelt. Das genannte Forschungsprojekt ermöglicht die Steuerung eines Industrieroboters mithilfe einer touchfähigen, 50 Zoll großen und senkrecht vor dem Industrieroboter stehende Glasscheibe. Mithilfe dieses Touchpanels können Bewegungen für den Industrieroboter mit den Fingern vorgegebenen werden, welche wiederum vom Industrieroboter direkt nachgefahren werden. Ein weiteres Forschungsprojekt an der FH Vorarlberg beschäftigt sich mit der Handsteuerung des „Fanuc"-Industrieroboters. Dieses Forschungsprojekt wird mithilfe einer Kamera realisert. Ein Nachteil dieses Ansatzes besteht jedoch darin, dass die Kamera für ein ordnungsgemäßes funktionieren der Gestensteuerung auf die Körpergröße der jeweiligen Person eingestellt werden muss. Dies ist einmalig vor der Benutzen der Gestensteuerung notwendig, solange die Einstellungen für die jeweilige Körpergröße nicht durch andere Personen geändert werden [Wer19].

In der Bachelorarbeit mit dem Titel „Eignungsuntersuchung von 2D- und 3D-Gesten für Kollaboration" wurde analysiert wie gut sich 2D- und 3D-Gesten für kollaborative Aufgaben innerhalb eines Teams eignen [Gra, S. 2 ff.]. Für die 3D-Gesten wird unter anderem vorgeschlagen beide Arme auf die Seite oder die Arme nach oben auszustrecken, mit den Händen Kreisbewegungen vor sich in der Luft auszuführen oder die Hände vor sich in der Luft in verschiedene Richtungen zu bewegen. Alle diese genannten 3D-Gesten führen eine jeweils zuvor zugewiesene Aktion aus [Gra, S. 39 ff.]. Zur Umsetzung der vorgeschlagenen Gesten wird die Kinect 2 verwendet um die 3D-Gesten vor einem Bildschirm erkennen zu können [Gra, S. 33 ff.]. Der Author des Textes kommt abschließend zum Fazit, dass die 3D-Gestenerkennung ihre Tücken hat. Nicht nur ist die 3D-Gestenerkennung mit der Kinect 2 zumeist ungenau, sondern auch die ausgewählten Gesten wirkten auf die Probanden unergonomisch und verursachten bei Verwendung deutliche Ermüdungserscheinungen [Gra, S. 83 f.].

Neben der Tiefenkamera basierten Gestenerkennung gibt es auch die Möglichkeit die Gestenerkennung über einen Datenhandschuh zu realisieren wie es im artec-Paper Nr. 42, welches den Titel „Gestenerkennung mit einem Datenhandschuh" trägt, erläutert wird. Der Vorteil besteht vor allem darin, dass der Datenhandschuh im Gegensatz zur stationären Tiefenkamera mobil einsatzfähig ist und nicht wie eine stationäre Tiefenkamera an einem Ort verbleiben muss. Der Nachteil besteht jedoch darin, dass für die

bedienende Person eine mitunter störende technische Ausstattung an den Händen getragen werden muss [Bra, S. 1 f.]. In dieser Arbeit konnte nachgewiesen werden, dass Gesten mit höherwertigen Datenhandschuhen eingesetzt werden können um Gesten mit mindestens 85%iger Wahrscheinlichkeit als korrekte Gesten zu erkennen [Bra, S. 11 f.].

3. Umsetzung

Zur Umsetzung der Gestensteuerung eines Industrieroboters wird der Lernroboter WidowX 200, welcher zur InterbotiX X-Serie gehört und von Trossen Robotics hergestellt wird, als Vorlage für einen Industrieroboter verwendet. Mit seinen 5 DOF eignet sich der „WidowX 200“-Lernroboter für Testzwecke als sichere Alternative zu einem Knickarmroboter. Bei Tests geht außerdem im Gegensatz zu einem industriellen Roboter eine geringere Gefahr vom „WidowX 200“-Lernroboter aus, da dieser keine so hohen Kräfte, Geschwindigkeiten und Beschleunigungen wie ein Industrieroboter entwickeln kann. Zudem bietet der „WidowX 200“-Lernroboter den Vorteil, dass dieser ein umfangreiches Dynamixel-SDK bereitstellt, welches die Integration in ROS und die direkte Kommunikation mit dem „WidowX 200“-Lernroboter überschaubar gestaltet. Hierdurch kann die Entwicklung des für die Gestensteuerung konzipierten Systems drastisch beschleunigt werden. Für den Gelenk-Modus stellt der „WidowX 200“-Lernroboter auch die Möglichkeit bereit die Gelenke individuell ansprechen und deren Gelenkpositionen verändern zu können. Die Dynamixel-Servo-Motoren können individuell an die Befürfnisse der Aufgabenstellung angepasst werden, da unter anderem die maximale Geschwindigkeiten und Beschleunigungen der einzelnen Servo-Motoren individuell konfigurierbar sind. Die Reichweite des Roboterarms beträgt in eine Richtung 55 cm. Der Arbeitsbereich des „WidowX 200“-Lernroboters ist kreisförmig und liegt bei einem Durchmesser von 110 cm. Im Arbeitsbereich des „WidowX 200“-Lernroboters sollten sich keine ungewollten Objekte befinden und es sollte sichergestellt werden, dass sich der „WidowX 200“-Lernroboter in diesem Bereich frei von menschlichen Einflüssen bewegen kann. Anzumerken ist, dass die Wiederholgenauigkeit des „WidowX 200“-Lernroboters bei 1 mm liegt. Für individuelle Aufgabenstellungen kann der Endeffektor jederzeit durch bereits verfügbare oder selbst z.B. durch einen per 3D-Druck erstellten Endeffektor ersetzt werden. Das maximale Tragegewicht des „WidowX 200“-Lernroboters beträgt 200 g, welches nicht überschritten werden sollte, da ansonsten Fehler bei der Änderung der Gelenkpositionen auftreten können [Wid].

Zur Gestenerkennung wird die Azure Kinect aufgrund der eingebauten Tiefenkamera verwendet. Auf die Azure-Cloud-Integration wird jedoch verzichtet, da das Azure Kinect Body Tracking SDK durch ein DNN eine ausreichend genaue Methode bereitstellt [qm1b] um 32 spezifische Positionen des menschlichen Körpers, zu denen unter anderem die Augen, Schulter, Knie, Daumen und Zeigefinger zählen, zu erkennen [qm1a]. Zudem wird durch den Einsatz des Azure Kinect Body Tracking SDK die Entwicklung des für Gestensteuerung konzipierten Systems drastisch beschleunigt. Einzelne Finger können aufgrund der Limitierung des Azure Kinect Body Tracking SDKs und der verbauten Tiefenkamera der Azure Kinect allerdings nicht erfasst werden. Gesten, welche jedoch die

Hand im Allgemeinen beinhalten, wie z.B. die Stopp-Geste, sind dagegen unter anderem durch die vorhandenen räumlichen Positionsinformationen realisierbar [Micc].

Die Gestensteuerung des „WidowX 200“-Lernroboters, welche auf dem für diese Arbeit erstellten Roboter-Gesten-Framework basiert und als Basis für andere gestengesteuerte Roboter dienen kann, wird näher beschrieben und verschiedene Gesten werden aufgrund der Ergonomie evaluiert. Anschließend werden Gesten, welche mit der Azure Kinect umsetzbar sind, ausgewählt und mit Sicherheitsvorkehrungen kombiniert, welche es verhindern sollen, dass sich der Industrieroboter unbeabsichtigt bewegt.

3.1. Allgemeine Anforderungen

Um die Gestensteuerung des „WidowX 200“-Lernroboters bewerten zu können müssen die allgemeinen Anforderungen an die Software zur Gestensteuerung eines Industrieroboters aufgezeigt werden. Hierzu zählen die funktionalen sowie auch die nichtfunktionalen Anforderungen an die Software zur Gestensteuerung eines Industrieroboters.

3.1.1. Funktionale Anforderungen

Die Anforderungen an die Software zur Gestensteuerung eines Industrieroboter können in die nachfolgenden Punkte gegliedert werden [Kir06] [Bra, S. 2 f.]:

- **Nutzung von verbreiteten Paradigmen für die Entwicklung**: Um mehrere Komponenten, welche auf unterschiedlichen Systemen verteilt sein können, miteinander kommunizieren lassen zu können bedarf es eines gemeinsamen Konzepts. Bei Industrierobotern wird hierbei vermehrt ROS eingesetzt, welches es erlaubt ROS-Messages über ein Netzwerkprotokoll verschicken zu können.
- **Standardisiertes Austauschformat**: Der Austausch der Informationen kann bei ROS über sogenannte ROS-Messages erfolgen und beim Speichern und Laden von Informationen in ein standardisiertes Austauschformat, wie z.B. JSON oder XML, übersetzt werden. Hierdurch kann die Interoperabilität zwischen den Systemen gewährleistet werden.
- **Zuverlässigkeit der Übertragung**: Die ROS-Messages müssen unverändert und vollständig über das Netzwerk an den ROS-Master geschickt werden können. Hierfür wird zumeist TCP/IP als Netzwerkprotokoll eingesetzt, welches die Vollständigkeit und richtige Reihenfolge der zugrundeliegenden Datenpakete gewährleistet.
- **Zuverlässigkeit der Gestenerkennung**: Die Gestensteuerung sollte eine möglichst kleine Falsch-Positiv-Rate aufweisen um unbeabsichtigte Gesten möglichst gering zu halten und so gefährlichen Situationen entgegenwirken zu können.
- **Echtzeitfähigkeit**: Das System muss sicherstellen, dass es die Informationen in einem gewissen Zeitintervall abarbeiten kann um die Sicherheit des Systems und der beteiligten Menschen gewährleisten zu können.

- **Adaptionsmöglichkeit an verschiedene Tiefenkameras**: Es muss die Möglichkeit bestehen im System die Tiefenkamera durch eine andere Tiefenkamera austauschbar zu machen. Hierbei muss sichergestellt werden, dass die Tiefenkameras die gleichen Schnittstellen verwenden um die gleiche Kommunikation zu ermöglichen.
- **Adaptionsmöglichkeit an verschiedene Industrieroboter**: Durch die schiere Anzahl an Industrierobotern muss es zudem auch möglich sein den Industrieroboter durch einen anderen Industrieroboter, unabhängig von seiner bisher vorhandenen Schnittstellen, ersetzen zu können.
- **Ergonomische Gesten**: Die Gesten müssen leicht erlernbar, einfach anwendbar und leicht zu erlernen sein. Zudem sollte ein längeres Verwenden der Gesten höchstens zu geringen Ermüdungserscheinungen führen und das System von Links- sowie auch Rechtshändern bedienbar sein.
- **Genauigkeit der Erreichung der Zielpose**: Die Zielpose muss je nach Genauigkeit des Industrieroboters über die Gestensteuerung erreichbar sein. Ungenauigkeiten durch die Gestensteuerung, welche aber auch bei Teach Pendants vorhanden sind, sind subjektiver Natur.
- **Sicherheitsvorkehrungen**: Es muss sichergestellt werden können, dass der Industrieroboter jederzeit durch einen Notstop unverzüglich gestoppt werden kann um lebensbedrohliche Situationen vorbeugen zu können.

3.1.2. Nichtfunktionale Anforderungen

Da in dieser Arbeit das Roboter-Gesten-Framework mit und ohne ROS lauffähig sein soll, müssen die Anforderungen an die Schnittstelle zwischen dem Industrieroboter und der Tiefenkamera genauer spezifiziert werden. Die Anforderungen an die Schnittstelle zwischen dem Industrieroboter und der Tiefenkamera sind daher die Folgenden, welche aufgrund von ROS den Anforderungen von Server-Client-Anwendungen entsprechen [Ost04]:

- **Performanz**: Bei der Kommunikation zwischen den einzelnen Systemen, zu denen die Tiefenkamera und der Industrieroboter zählen, sollte die Anfrage so schnell wie möglich erfolgen und daher feste Echtzeitfähigkeit gewährleisten können (siehe Kapitel 2.3.1).
- **Zuverlässigkeit**: Die Zuverlässigkeit beschreibt, wie verlässlich ein System seine Aufgaben ausführt. Ausfälle von ROS-Nodes oder sogar des ROS-Masters selbst reduzieren hierbei unter anderem die Zuverlässigkeit des Systems drastisch.
- **Modularität**: Spiegelt den Grad der Aufteilung in Komponenten wider, welche vom eigenen Projekt oder von Dritten wiederverwendet werden können.
- **Erweiterbarkeit**: Bezeichnet die Fähigkeit des Systems durch weitere Komponenten, welche dem System hinzugefügt werden können, angepasst zu werden. Bei ROS können zusätzliche ROS-Packages von Drittanbietern installiert werden, wodurch das System erweitert werden kann.

3.2. Einrichtung des Testsystems

Zur Umsetzung der Gestensteuerung von Industrierobotern wird ROS 1 mit dem Codenamen Melodic Morenia, welches als LTS-Version verfügbar ist und daher Langzeitunterstützung bietet, eingesetzt [Roba]. Als Betriebssystem kommt Ubuntu 18.04 Bionic Beaver zum Einsatz, da dies die neueste unterstützte LTS-Version von Ubuntu darstellt, welche von ROS Melodic Morenia empfohlen wird [Rosb].

3.2.1. Linux-Container

Zur besseren Einsetzbarkeit auf unterschiedlichen Linux-Distributionen wird das genannte Ubuntu 18.04 Bionic Beaver und ROS Melodic Morenia in einem Linux-Container installiert und per Image zur Verfügung gestellt. Der Vorteil durch die Container-Technik stellt hierbei das einfachere und schnellere Deployen, der in dieser Arbeit entwickelten Anwendung zur Gestensteuerung des „WidowX 200“-Lernroboters, dar. Zudem bietet die Container-Technik im Gegensatz zu einem Hypervisor, wie z.B. VMware, VirtualBox oder Hyper-V, den Vorteil, dass kein vollständiges Linux-Betriebssystem virtualisiert werden muss, welches einen impliziten Ressourcen-Overhead, wie z.B. hohe RAM- und CPU-Auslastung, mit sich ziehen würde. Zur Umsetzung der Trennung von Prozessen werden die Kernelnamensräume und für die Ressourcenverwaltung cgroups eingesetzt, welche im Linux-Kernel enthalten sind. Dies führt dazu, dass der Linux-Container auf den gleichen Kernel wie das Host-System zugreifen kann, jedoch aber der Zugriff auf sensible Ressourcen durch den Linux-Kernel beschränkt oder sogar vollständig untersagt wird [Lxca]. Linux-Container werden auch kurz LXC genannt, welches auch die gleichnamige Bezeichnung für das Userspace-Tool ist um die Linux-Container zu verwalten. Die Erweiterung von LXC wird LXD genannt und ermöglicht die Verwaltung von mehreren LXC-Instanzen über einen System-Daemon, welcher per REST-API angesprochen werden kann. Zudem bietet LXD unter anderem die Möglichkeit Snapshots von Linux-Containern zu erstellen, zu erzeugen, zu starten und zu stoppen [Lxcb]. Für diese Arbeit wird als Host-System, auf dem die LXC-Instanz installiert und ausgeführt wird, Ubuntu 20.04 Focal Fossa verwendet, welches eine LTS-Version von Ubuntu ist und den Linux-Kernel 5.4 einsetzt. Andere Linux-Distributionen sind grundsätzlich auch möglich solange die für diese Arbeit benötigten Features im Linux-Kernel verfügbar sind. Die entwickelte Roboter-Gesten-Anwendung wird im Linux-Container mit echtzeitfähiger Priorität ausgeführt, da die maximale Latenz des Kommunikationskanals des Eingabegerätes aufgrund der Maschinenrichtlinien 2006/42/EG und der „ISO EN 10218“-Norm weniger als 50 ms betragen und zudem ein deterministisches Verhalten aufweisen muss [Pra+04, S. 55]. Die benötigten Features des Linux-Kernels und zudem das Einrichten des Linux-Schedulers, welcher feste Echtzeitfähigkeit garantieren muss, können zudem aus der Installationsanleitung, welche sich im Anhang A.1 befindet, entnommen werden.

3.2.2. Simulationsumgebungen

Zum Durchführen von Tests werden oft Simulationsumgebungen verwendet, da diese den Vorteil bieten, dass kein realer Roboter vorhanden sein muss und dadurch gefährliche Situationen für den Menschen erst gar nicht entstehen können. Während der Entwicklung eines Roboters kann dieser sogar in einer virtuellen Umgebung getestet werden um so das Verhalten und technische Limitierungen des Roboters vor der Herstellung zu ermitteln. Dies spart Zeit und Kosten, da das fertige Produkt erst nach ausführlichen Tests hergestellt wird. Mit Simulationsumgebungen können zudem Tests schneller und effizienter durchgeführt werden, da unter anderem Differenzen zu exakten Zielposen einfacher und genauer als mit manuellen Messungen bestimmt werden können [Gaz]. Zwei der bekanntesten Simulationsumgebungen, welche frei auf dem Markt verfügbar sind und unterschiedliche Roboter unterstützen, sind Gazebo[1] und Coppelia Sim[2], welches der Nachfolger von vRep darstellt. Beide Simulationsumgebungen sind unter Windows, Linux und macOS verfügbar und bieten die Möglichkeit die standardmäßige Physik-Engine nach belieben mit einer anderen Physik-Engine auszutauschen um z.B. eine genauere Simulation durchführen zu können. Copellia Sim und Gazebo besitzen zudem beide einen Szenen-Editor und bei Coppelia Sim ist zudem im Gegensatz zu Gazebo ein 3D-Model-Editor integriert. Bei Gazebo wird hierfür eine externe Anwendung benötigt um die 3D-Modelle den jeweiligen Bedürfnissen anpassen zu können. Coppelia Sim ist zudem aufgrund der Stabilität der GUI und der Plugins und der besseren Dokumentation im Gegensatz zu Gazebo zu bevorzugen. Viele Roboter werden aufgrund der ROS-Integration standardmäßig mit Out-of-the-Box-Unterstützung für Gazebo ausgestattet, wodurch bei Verwendung von Coppelia Sim zumeist ein Mehraufwand zum Einrichten des Roboters vonnöten ist. Beide Simulationsumgebungen bieten die Möglichkeit um über ROS-Messages angesprochen zu werden. Hierdurch ist eine Integration in ROS leicht umsetzbar. Copellia Sim und Gazebo bieten jedoch aber auch die Möglichkeit vollständig ohne ROS verwendet und so in beliebigen Softwareumgebungen eingesetzt zu werden [Vre]. In dieser Arbeit wird Gazebo eingesetzt, da der „WidowX 200“-Lernroboter bereits eine vollständige Gazebo-Unterstützung mitliefert. Die benötigten Schritte zum Einrichten der Simulationsumgebung für den „WidowX 200“-Lernroboter können aus dem Anhang A.1 entnommen werden.

3.3. Anbindung an ROS

Da Gazebo standardmäßig eine ROS-Integration bereitstellt müssen nur die ROS-Packages des „WidowX 200“-Lernroboters installiert werden um die Gazebo-Integration einzurichten, wie aus dem Anhang A.1 zu entnehmen ist. Die Simulationsumgebung kann daraufhin gestartet und z.B. für Testzwecke genutzt werden. Die ROS-Packages des „WidowX 200“-Lernroboters werden von Trossen Robotics entwickelt und der Quellcode kann öffentlich eingesehen werden um eigene Anpassungen daran durchführen zu können. Der

[1] http://gazebosim.org/
[2] https://www.coppeliarobotics.com/

„WidowX 200“-Lernroboter kann aufgrund des „InterbotiX“-Nodes von überall innerhalb des Roboter-Netzwerks angesprochen werden. Zudem ist durch die ROS-Unterstützung des „WidowX 200“-Lernroboters überhaupt erst möglich MoveIt zu unterstützen. MoveIt ist ein ROS-Package, welches es vor allem ermöglicht Bewegungen zu Planen, die inverse und direkte Kinematik zu berechnen oder aber auch Kollisionschecks mit einem Tiefensensor zu ermöglichen [Mova]. Zudem bietet MoveIt eine C++-API an um die ROS-Services zu überspringen und eine höhere Performanz als mit einer vollständigen ROS-Umsetzung zu erhalten [Movb].

Für das Azure Kinect Sensor SDK, welches direkten Zugriff auf die Sensoren der Azure Kinect ermöglicht, und das Azure Kinect Body Tracking SDK, welches zur Gestenerkennung eingesetzt werden kann, werden keine ROS-Packages von Microsoft bereitgestellt [tesa]. Daher ist es notwendig diese direkt in einen eigenen ROS-Node oder direkt in den ROS-Node der in dieser Arbeit entwickelten Roboter-Gesten-Anwendung zu integrieren. Aufgrund der Komplexität der Gestenerkennung, der Notwendigkeit zur schnellen und sicheren Reaktion auf die Gesten und der Möglichkeit von sehr schnell aufeinander folgenden Gesten und Bewegungsabläufen der Hände wurde entschieden die Gestenerkennung über eine C++-Schnittstelle bereitzustellen um möglichst jederzeit sicherstellen zu können, dass die Nutzereingaben schnell genug an die Steuereinheit übermittelt werden können. Zudem bietet es bei der Messung der Latenz der Übertragung mit und ohne ROS-Anbindung den Vorteil, dass schnell darauf geschlossen werden kann wo der Flaschenhals im System liegt. Dies wäre mit sehr vielen selbst erstellten und verteilten ROS-Nodes nur sehr schwer bis überhaupt nicht einsehbar, da das System zu komplex werden würde [Why]. Der ROS-Node zur Gestenerkennung muss durch diesen Ansatz nur noch die benötigten Steuerbefehle per MoveIt oder über die ROS-Schnittstelle des Industrieroboters an die Motoren des Industrieroboters senden um Gelenkbewegungen auslösen zu können.

3.4. Gesten

Zu Gesten zählen alle Formen der Bewegung, welche der nonverbalen Kommunikation dienen und mit den Armen, den Händen, dem Kopf, den Füßen oder Beinen ausgeführt werden können [Gesb]. Wissenschaftler unterscheiden bis zu 5000 verschiedene Gesten, welche weltweit bewusst oder unbewusst eingesetzt werden. Handgesten werden vor allem dafür eingesetzt um Aussagen zu untermauern und deren Wirksamkeit zu verstärken. Bei Gesten sind auch die verschiedenen Posen des eingesetzten Körperbereichs entscheident zu beachten, da die Geste je nach Pose unterschiedliche Bedeutung besitzen kann. Z.B. ist das Greifen des Kinns mit dem Zeigefinger und Daumen eine nachdenkliche Geste, welche jedoch durch überdecken des verschlossenen Mundes mit dem Zeigefinger zu einer anderst interpretierbaren Geste wird. Das Überdecken des verschlossenen Mundes mit dem Zeigefinger kann als ein bewusstes Unterdrücken von Informationen angesehen werden und spiegelt daher nicht mehr die nachdenkliche Pose wider [Mat07, S. 137 f.].

3.4.1. Arten von Gesten

Je nach Religion, Situation, Kultur, sozialer Herkunft, unterschiedlicher Ansichten und je nach Gefühlslage können Gesten eine andere Bedeutung besitzen. Aus diesen Gründen ist es sehr wichtig die verschiedenen Arten von Gesten zu kennen um so möglichst ethisch neutrale Gesten für die Kommunikation mit einem Industrieroboter auswählen zu können [Gesc]. Daraus wird ersichtlich, dass nicht nur die Geste selber für das interpretieren notwendig ist, sondern aber auch der Kontext in der die Geste eingesetzt wird. Je nach Kontext sind körpernahe und körperferne Gesten, welche z.B. zum Zeigen auf Objekte eingesetzt werden, daher gleichermaßen für Missinterpretationen anfällig [Mat07, S. 137 f.], da z.B. beim Erschrecken die Hände reflexartig ausgestreckt werden, bei Angstzuständen oder starker Wärme der Schweiß von der Stirn gewischt oder bei Rückenschmerzen an den Rücken gegriffen wird [Mat].

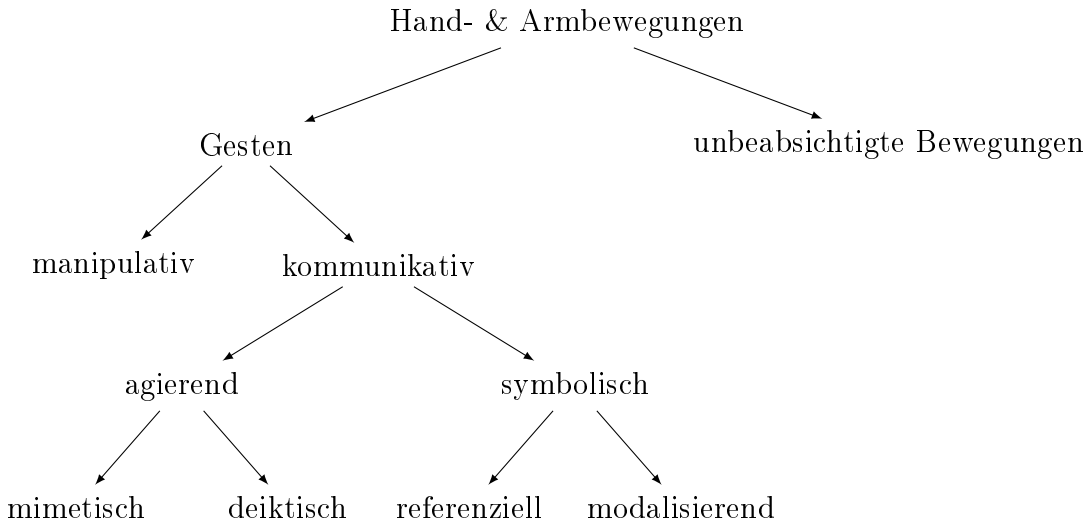

Abbildung 3.1.: Einteilung der Gesten
Quelle: Eigene Darstellung, in Anlehnung an [PSH97, S. 680]

Eine der bekanntesten Gesten-Taxonomien für HCI mit denen alle Hand- und Armbewegungen, welche in Abbildung 3.1 ersichtlich sind, klassifiziert werden können, stammt von F.K. Quek, einem Professor für Informatik und Mathematik am Virginia Polytechnic Institute und der State University. Diese Taxonomie teilt die Hand- und Armbewegungen in Gesten und unbeabsichtigte Bewegungen ein. Nur wenn die Geste mit einer absichtlichen Hand- oder Armbewegung durchgeführt wurde, wird die Hand- oder Armbewegung in diesem Modell als Geste klassifiziert. Alle anderen Hand- oder Armbewegungen sind daher nach diesem Modell keine Gesten, da diese für den Computer keine sinnvollen Informationen bieten. Gesten können nach dieser Taxonomie wiederum in manipulative und kommunikative Gesten unterteilt werden. Manipulative Gesten werden dazu verwendet mittels z.B. Bewegungsgesten oder Drehgesten auf Objekte in der Umgebung

einzuwirken und diese zu verändern, wohingegen kommunikative Gesten einen inhärenten kommunikativen Zweck erfüllen und zumeist durch die Sprache unterstützt werden. Kommunikative Gesten können wiederum in agierende und symbolische Gesten unterteilt werden. Der Unterschied zwischen symbolischen und agierenden Gesten besteht darin, dass symbolische Gesten im Gegensatz zu agierenden Gesten eine bildliche Beschreibung einer Aktion darstellen und mit einer zuvor definierten Aktion verknüpft werden müssen um diese interpretierbar zu machen. Agierende Gesten stehen hingegen im direkten Zusammenhang mit der Aktion und deren Auswirkung ist daher auch direkt erkennbar. Die referenzielle Geste, welche zur symbolischen Gestenart zählt, kann z.B. eine kreisförmige Bewegung mit dem Zeigefinger sein, welche angibt dass die Bewegung auf ein Rad bezogen ist und dieses sich bewegen soll. Neben der referenziellen Geste existiert auch die modalisierende Geste, welche im Gegensatz zur referenziellen Geste eine Aktion durch eine symbolische Geste und zusätzlicher Verwendung der Sprache vorgibt. Bei symbolischen Gesten sind daher der Kreativität keine Grenzen gesetzt und deshalb können diese Gesten nach belieben mit verschiedenen Aktionen verknüpft werden. Zu den agierenden Gesten zählen mimetische und deiktische Gesten, welche im direkten Zusammenhang mit der Interpretation der Geste stehen. Deiktische Gesten werden dabei mit dem Zeigefinger durchgeführt und geben z.B. eine Position vor, wohingegen mimetische Gesten bestimmte Aktionen imitieren. Im HCI-Kontext werden überwiegend symbolische Gesten eingesetzt, da diese zumeist durch verschiedene statische Handhaltungen dargestellt werden können und so zumeist anstrengende und ermüdende Gesten vermeiden [PSH97, S. 680].

3.4.2. Beispiele für unethische symbolische Gesten

Gesten können je nach Religion, Situation, Kultur, sozialer Herkunft, unterschiedlicher Ansichten und je nach Gefühlslage auf eine andere Art und Weise interpretiert werden. Die auszuwählenden Gesten sollten daher für alle Menschen ethisch vertretbar sein um Missverständnissen und peinlichen Momenten vorzubeugen [Gesc]. Aus diesem Grund werden hierfür beispielhaft ein paar Gesten aus aller Welt miteinander verglichen und deren Bedeutungen in den unterschiedlichen Ländern erklärt.

Daumen hoch: Die „Daumen hoch“-Geste wird im deutschsprachigen Raum, USA und Korea dazu verwendet um eine besondere Leistung hervorzuheben. In anderen Ländern, wie z.B. Russland, Frankreich, Israel, Griechenland und Italien, ist es jedoch eine obszöne Aufforderung zum Sex und sollte daher nicht verwendet werden um eine unangenehme Situation und daraus resultierende Probleme zu vermeiden. In China und Japan hingegen bedeutet die „Daumen hoch“-Geste, dass man fünf Stück von einer Sache bestellen will und in Japan kann es zudem auch als Synonym für einen festen Freund missinterpretiert werden. Die „Daumen hoch“-Geste kann in Australien, Afghanistan, Iran, Irak und Nigeria auch als provokante Aufforderung zum Gehen missverstanden werden [Han].

Daumen-Zeigefinger-Kreis: Die „Daumen-Zeigefinger-Kreis“-Geste wird im deutsch-

sprachigen Raum, Nordamerika/USA, England, Mexiko, Skandinavien und Kanada positiv aufgefasst und bedeutet in den meisten Fällen „Okay“ oder, dass zumeist eine Aufgabe sehr gut ausgeführt wurde. In der Türkei, Griechenland und Russland zeigt man hierdurch symbolisch eine Körperöffnung, welche in diesen Ländern als obszön zu betrachten ist. Die „Daumen-Zeigefinger-Kreis“-Geste wird in Brasilien als obszöne Beleidigung aufgefasst und in Belgien und Tunesien wird damit sogar ausgedrückt, dass die andere Person eine Null und dadurch nutzlos ist [Han].

Zeigefinger ausstrecken: Das Richten des Zeigefingers auf andere Personen ist in den meisten Ländern, wie z.B. dem deutschsprachigen Raum, Frankreich, Italien, China und Japan, ein Zeichen der Respektlosigkeit und der Verachtung. Wenn der Zeigefinger auf sich selbst gezeigt wird, stellt es hingegen ein Zeichen der eigenen Selbstverliebtheit dar, welches in den genannten Ländern, wie auch das Zeigen mit dem Zeigefinger auf andere Personen, als negativ zu werten ist [Gesa].

Mit Zeigefinger an die Stirn tippen: Im deutschsprachigen Raum wird diese Geste als sehr abwärtend wahrgenommen und bedeutet im Normalfall, dass die andere Person ein Idiot ist. In Nordamerika und Peru hingegen ist es eine Anerkennung und ein Ausdruck dafür, dass die andere Person sehr schlau bzw. clever ist. In den USA wird diese Geste sogar dazu verwendet um andere Personen im Straßenverkehr freundlicherweise darauf hinzuweisen, dass in der Nähe eine Polizeikontrolle stattfindet [Han].

Im Großen und Ganzen kann resultierend gesagt werden, dass es nur schwer möglich ist Gesten so zu gestalten damit sie für alle religiösen, kulturellen und sozialen Gegebenheiten und in jeglichem Kontext frei von Missinterpretationen sind. Es ist jedoch möglich die Gesten aufgrund der bisher erarbeiteten Grundlage weitgehend ethisch zu gestalten und auszuwählen um möglichen Missinterpretationen entgegenwirken zu können.

3.5. Auswahl der zu verwendenden Gesten

Aufgrundlage der ermittelten Limitierungen der Azure Kinect und der untersuchten ethischen Voraussetzungen können nun entsprechende Gesten entwickelt werden, welche die Anforderungen an die sicherheitskritische Steuerung eines Industrieroboters zur Vermeidung von Unfällen erfüllen. Zudem müssen die Gesten auch aus ergonomischer Sicht einfach und angenehm ausführbar sein und im besten Fall während des Teachens keine großen Ermüdungserscheinungen aufweisen [Sch20, S. 97].

3.5.1. Verwendete Gesten

Um die zuvor genannten Kriterien für die Gestensteuerung zu ermöglichen ist es unter anderem notwendig möglichst leicht zu erlernende Gesten im Gegensatz zu komplizierten Gesten zu wählen, da hierdurch die Erlernbarkeit der vorhandenen Gesten vereinfacht wird [Sch20, S. 97]. Aus Gründen der Ergonomie sind Freihandgesten mit ausgestreckten

Armen möglichst zu vermeiden, wenn diese auf Dauer ausgeführt werden müssen, da diese schnell zu Ermüdungserscheinungen führen [Neu, S. 84]. Eine direkte und natürliche Interaktion, welche intuitiv und einfach gestaltet ist, steigert zudem die Zufriedenheit der Personen, welche die Gesten einsetzen müssen [Sch20, S. 103]. Zudem ist eine ausreichende Rückmeldung des Gestensystems praktisch, da hierdurch die Zufriedenheit der bedienenden Person und dessen Vertrauen zum System gesteigert wird [Sch20, S. 132 f.]. Im Gegensatz zu VR oder AR können beim Verzicht auf diese Technologien keine virtuellen Objekte, wie in AR in der realen Umgebung oder wie bei VR in einer virtuelle Umgebung, platziert werden um der bedienenden Person z.B. eine Hilfestellung zur Verwendung der Gestensteuerung zu geben. In AR und VR wäre es möglich z.B. ein Tutorial bereitzustellen, welches die Bewegungsbahnen und Gesten in der jeweiligen Umgebung einblendet. Die bedienende Person hätte dann die Möglichkeit diese Gesten nachzuahmen und die Bedienung spielerisch zu erlernen [Wei07]. Eine gute Benutzererfahrung wird aber auch damit verbunden, dass das System die Gesten zuverlässig genug erkennt und die Gesten während des Einsatzzweckes für die bedienende Person nicht zu anstrengend werden. Zudem sind Gesten im Oberkörperbereich zu bevorzugen, da mit den Händen und Armen in diesem Bereich ohne die Arme übermäßig strecken zu müssen, verschiedene Bewegungen möglich sind [Now17]. Bei kurzen Aufgabenstellungen von wenigen Sekunden können Freihandgesten gewählt werden, da diese auf kurze Zeitspannen nicht anstrengend sind. Auf längere Sicht wirken die Freihandgesten jedoch ermüdend auf die bedienende Person, da es sehr anstrengend sein kann die Hand vom Körper weg zu halten [Kra18]. Die Hand muss zudem über eine längere Zeitdauer hinweg, genau nach den Anforderungen des Gestenerkennungssystems bewegt werden, wordurch die Durchführung der Geste wiederum erschwerender wird [eri]. Im Allgemeinen kann bei Bedienungsaufgaben und bei Gesten zudem gesagt werden, dass größere Schaltflächen mit einem größeren Betätiungsbereich im Gegensatz zu kleineren Schaltflächen mit einem kleinerem Betätigungsbereich zu bevorzugen sind, da diese von verschiedenen Personen mit unterschiedlichen Körperproportionen besser betätigt werden können [BMM13].

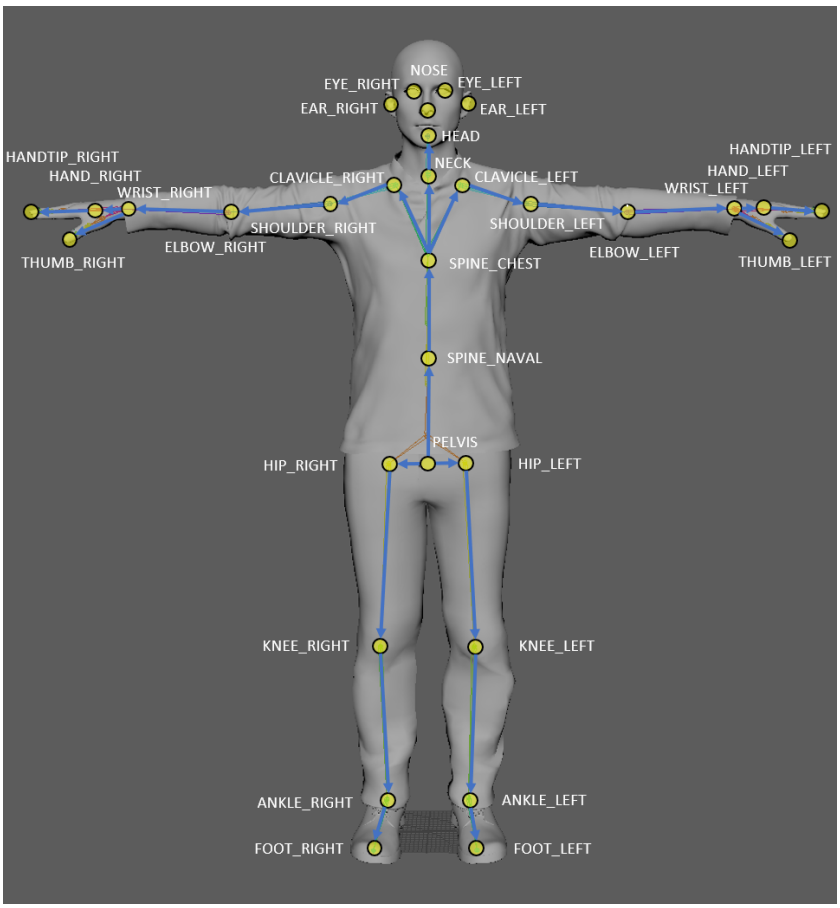

Abbildung 3.2.: Azure Kinect: 32 spezifische Positionen des menschlichen Körpers
Quelle: [qm1a]

Wie in Abbildung 3.2 zu sehen ist, können bei der linken und rechten Hand unter anderem der Zeigefinger, Daumen, ca. die Mitte der Hand und das Gelenk über das Azure Kinect Body Tracking SDK, welches 32 spezifischen Positionen des menschlichen Körpers identifzieren kann, genutzt werden. Die Bestimmung und Nutzung der einzelnen Finger ist aufgrund der Limitierung des Azure Kinect Body Tracking SDKs und der verbauten Tiefenkamera der Azure Kinect nicht möglich. Aus diesem Grund können die einzelnen Finger mit der Azure Kinect nicht zur Gestensteuerung eines Industrieroboters eingesetzt werden. Deswegen wird auf feinmotorische Gesten, welche mit den einzelnen Fingern durchgeführt werden und zu denen z.B. das Skalieren mit zwei Fingern zählt, verzichtet [qm1a]. In der Tabelle 3.1 sind die für die Gestensteuerung des Industrieroboters aufgestellten Gesten ersichtlich. Zum besseren Verständnis der Gesten wurden diese zudem mit den in Abbildung 3.2 ersichtlichen 32 spezifischen Positionen des menschlichen Körpers, welche durch die Azure Kinect bereitgestellt werden, übersichtlich verknüpft. Bei der Entwicklung der Gesten wurde darauf geachtet, dass die Gesten für sowohl Links- aber auch Rechtshänder über die für das Teachen vorgesehene Zeit eine entspannte Möglichkeit bieten den Industrieroboter zu bedienen. Bei der Entwicklung der Gesten war

es zudem sehr wichtig der bedienenden Person bei längeren Bedienungsvorgängen, wie z.B. der Gelenkstellung erhöhen, eine Möglichkeit zum Ablegen der Hände zu bieten um so Ermüdungserscheinungen vorbeugen zu können. Auf der anderen Seite war bei der Entwicklung der Gesten aber auch wichtig, die Gesten in einer gewissen Distanz voneinander anzuordnen um so unbeabsichtigte Betätigungen mehrerer gleichzeitiger Gesten während des Bedienungsvorganges zu minimieren [Pro13]. Unbeabsichtigte Gesten können jedoch aber auch durch reflexartige Bewegungen zum Schutz des eigenen Leibes entstehen. Diese reflexartigen und unbeabsichtigten Gesten, wie z.B. das Schützen des Kopfes mit den Händen, [Mat07, S. 95] wurden bei der Entwicklung der Gesten mitberücksichtigt und werden von der entwickelten Roboter-Gesten-Anwendung größtmöglich herausgefiltert.

3.5.2. Zuordnung der Gesten zu den Aktionen

Die Gesten in der Tabelle 3.1 wurden aufgrund dieser Auswahlkriterien entwickelt, wobei deren Ergonomie noch durch praxisnahe Teachvorgänge getestet werden muss. Bei allen in dieser Ausarbeitung entwickelten Gesten ist es entscheident, dass die bedienende Person Sichtkontakt zur Tiefenkamera und daher auch Sichtkontakt zum Industrieroboter hat um eine Aktion mit den entwickelten Gesten ausführen zu können. Bei Sichtunterbrechung zur Tiefenkamera oder bei Bestimmung mehrerer Personen im Blickfeld der Tiefenkamera wird die Bedienung des Industrieroboters mittels der Gesten zur Sicherheit deaktiviert. Ein Weiterarbeiten ist erst dann möglich, wenn die Sicht zur Tiefenkamera wiederhergestellt ist und sich nur noch eine einzige Person im Blickfeld der Tiefenkamera befindet.

Aktion	Beschreibung mit spezifischer Geste	Anmerkung
Gelenkstellung erhöhen	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz oder länger zwischen das **rechte Schlüsselbein** (*CLAVICLE_RIGHT*) und die **rechte Schulter** (*SHOULDER_RIGHT*) legen	*1 *4 *5
Gelenkstellung verringern	**Rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz oder länger zwischen das **linke Schlüsselbein** (*CLAVICLE_LEFT*) und die **linke Schulter** (*SHOULDER_LEFT*) legen	*1 *4 *5
Position auf Weltachse erhöhen	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz oder länger zwischen das **rechte Schlüsselbein** (*CLAVICLE_RIGHT*) und die **rechte Schulter** (*SHOULDER_RIGHT*) legen	*1 *4 *6

Position auf Weltachse verringern	**Rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz oder länger zwischen das **linke Schlüsselbein** (*CLAVICLE_LEFT*) und die **linke Schulter** (*SHOULDER_LEFT*) legen	*1 *4 *6
Endeffektor auseinander-fahren	**Rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz oder länger in die **Mitte der Brusthöhe** (*SPINE_CHEST*) legen	*1
Endeffektor zu-sammenfahren	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz oder länger in die **Mitte der Brusthöhe** (*SPINE_CHEST*) legen	*1
Endeffektor-Modus umschalten	**Linker Zeigefinger** (*HANDTIP_LEFT*) ca. für 2 s auf das **linke Knie** (*KNEE_LEFT*) legen **ODER** **rechter Zeigefinger** (*HANDTIP_RIGHT*) ca. für 2 s auf das **rechte Knie** (*KNEE_RIGHT*) legen	*2 Verfügbare Modi: • gesperrt • nicht gesperrt Standardmäßig: • nicht gesperrt
Zum vorigen Gelenk wechseln	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz auf das **linke Ohr** (*EAR_LEFT*) oder das **linke Ohrläppchen** legen oder darüber schweben lassen	*2 *3 *5 *7
Zum nächsten Gelenk wechseln	**Rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz auf das **rechte Ohr** (*EAR_RIGHT*) oder das **rechte Ohrläppchen** legen oder darüber schweben lassen	*2 *3 *5 *7
Zur vorigen Weltachse wechseln	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz auf das **linke Ohr** (*EAR_LEFT*) oder das **linke Ohrläppchen** legen oder darüber schweben lassen	*2 *3 *6 Verfügbare Mo-di/Reihenfolge: • X (links/rechts) • Z (aufwärts/-abwärts) • Y (vorwärts /rückwärts) Standardmäßig: • X (links/rechts)

Zur nächsten Weltachse wechseln	**Rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz auf das **rechte Ohr** (*EAR_RIGHT*) oder das **rechte Ohrläppchen** legen oder darüber schweben lassen	*2 *3 *6 Verfügbare Modi/Reihenfolge: • X (links/rechts) • Z (aufwärts/-abwärts) • Y (vorwärts /rückwärts) Standardmäßig: • X (links/rechts)
Präzisions-modus umschalten	**Linker Zeigefinger** (*HANDTIP_LEFT*) ca. für 2 s in den **Bereich des Bauchnabels** (*SPINE_NAVEL*) legen **ODER** **rechter Zeigefinger** (*HANDTIP_RIGHT*) ca. für 2 s in den **Bereich des Bauchnabels** (*SPINE_NAVEL*) legen	*2 Verfügbare Modi: • präzise (Beschleunigung deaktiviert) • Beschleunigung aktiviert Standardmäßig: • präzise (Beschleunigung deaktiviert)
Positionen speichern	**Linker Zeigefinger** (*HANDTIP_LEFT*) ca. für 2 s auf die **linke Hüfthöhe** (*HIP_LEFT*) legen **ODER** **rechter Zeigefinger** (*HANDTIP_RIGHT*) ca. für 2 s auf die **rechte Hüfthöhe** (*HIP_RIGHT*) legen	*2

Koordinaten-system-Modus umschalten	**Linker Zeigefinger** (*HANDTIP_LEFT*) kurz auf die **rechte Armbeuge** (*ELBOW_RIGHT*) legen **ODER** **rechter Zeigefinger** (*HANDTIP_RIGHT*) kurz auf die **linke Armbeuge** (*ELBOW_LEFT*) legen	*2 Verfügbare Modi: • Weltkoordinatensystem-Modus • Gelenk-Modus Standardmäßig: • Weltkoordinatensystem-Modus

Tabelle 3.1.: Implementierte Gesten
Quelle: Eigene Ausarbeitung

*1 Wenn der Präzisionsmodus ausgeschalten ist (Beschleunigung aktiviert), führt ein längeres verharren in dieser Pose zu einer Beschleunigung der jeweiligen Bewegung.

*2 Längeres verharren in dieser Pose ändert die Einstellung nur einmal. Um die Einstellung mehrmals zu ändern muss die Pose verlassen und anschließend neu initiiert werden.

*3 Die Anzahl der Gelenke sind vom jeweiligen Industrieroboter abhängig.

*4 Die Bewegung ist Abhängig von der Ausrichtung des Koordinatensystem des ausgewählten Gelenks/Weltachse.

*5 Die Geste ist nur im Gelenk-Modus verfügbar. Die Aktion ist daher abhängig vom verwendeten Koordinatensystem-Modus.

*6 Die Geste ist nur im Weltkoordinatensystem-Modus verfügbar. Die Aktion ist daher abhängig vom verwendeten Koordinatensystem-Modus.

*7 Das standardmäßig ausgewählte Gelenk und die Reihenfolge beim Wechseln der Gelenke ist abhängig vom jeweiligen Industrieroboter.

In der Tabelle 3.1 sind die Aktionen, welche vom Industrieroboter ausgeführt werden können und mit Gesten verknüpft wurden, aufgelistet. Bei der Entwicklung der Gesten wurden die erarbeiteten Auswahlkriterien für Gesten beachtet und bei den Gesten miteinbezogen. Bei längeren Aktionen, wie z.B. die Gelenkstellung erhöhen oder verringern, wurde darauf geachtet diese Aktionen mit möglichst angenehmen Gesten zu verknüpfen, welche es erlauben die Hände abzulegen. Dies beugt Ermüdungserscheinungen bei der Durchführung der Gesten vor, da der bedienenden Person eine Möglichkeit zum Ausruhen zur Verfügung gestellt wird. Zu jeder Geste wird in der Tabelle 3.1 eine genaue Beschreibung angegeben um diese ordnungsgemäß nach der Anleitung ausführen zu können. In manchen Fällen, wie z.B. beim Umschalten des Präzisionsmodus, kann dies je nach Präferenz durch die linke oder rechte Hand erfolgen. Hierdurch steht

es der Person frei die Geste seiner Wahl für den jeweiligen Anwendungsfall auszuwählen. Zur Erhöhung der Sicherheit wurde des Weiteren eine Möglichkeit implementiert den Endeffektor zu sperren um keine weiteren Aktionen mit ihm ausführen zu können. Z.B. wenn der Endeffektor eine Last transportiert ist es wichtig, dass die Last nicht plötzlich durch ungewolltes Öffnen des Endeffektors fallen gelassen wird. Dies würde ansonsten einen ungewollten Schaden verursachen. Standardmäßig ist der Endeffektor nicht gesperrt, da im Normalfall am Anfang noch keine Last transportiert wird.

3.5.3. Präzisionsmodus

Der Präzisionsmodus ermöglicht das Wechseln zwischen einem präzisen und einem unpräziseren Modus. Mithilfe des präzisen Modus kann sichergestellt werden, dass die Geschwindigkeit, welche für den jeweiligen Industrieroboter voreingestellt wird, gleichbleibend ist. Der unpräzisere Modus ermöglicht hingegen die Beschleunigung der Bewegung bis zu einem für den Industrieroboter vorgegebenen Schwellwert.

3.5.4. Koordinatensysteme in ROS

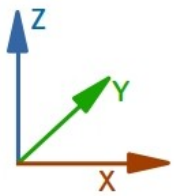

Abbildung 3.3.: Kartesisches Koordinatensystem von ROS
Quelle: Eigene Darstellung, in Anlehnung an [GRB19]

Zur Berechnung von Koordinaten im entsprechenden Koordinatensystem verwendetet ROS hierfür das kartesische Koordinatensystem, welches in Abbildung 3.3 ersichtlich ist [GRB19]. Im implementierten Weltkoordinatensystem-Modus der Roboter-Gesten-Anwendung werden die Bewegungen daher im Bezug zu den Weltachsen durchgeführt. Die in der Tabelle 3.1 erwähnten Weltachsen X, Y und Z entsprechen hierbei den Achsenbezeichnungen, welche in Abbildung 3.3 zu sehen sind. Die in der Tabelle 3.1 aufgezeigten Gesten wurden für ähnliche Aktionen wiederverwendet um das Erlernen der Gesten zu errleichtern und die Bedienung intuitiv zu gestalten. Die Aktionen für die Gelenkstellung („Gelenkstellung erhöhen“) und die Position auf der Weltachse zu verändern („Position auf Weltachse erhöhen“, „Position auf Weltachse verringern“) wurden vereinheitlicht, sowie die Aktionen um zwischen Gelenken zu wechseln („Zum vorigen Gelenk wechseln“, „Zum nächsten Gelenk wechseln“) und zum Wechseln zwischen Weltachsen („Zur vorigen Weltachse wechseln“, „Zur nächsten Weltachse wechseln“) wurden aneinander angeglichen.

3.5.5. Sicherheitskonzept

Obwohl die implementierten Gesten so ausgelegt wurden, dass reflexartige und unbeabsichtigte Gesten weitaus unwahrscheinlicher als korrekte Gesten identifiziert werden, kann auf den

„Dead Man's Switch“ nicht verzichtet werden. Der „Dead Man's Switch“ schützt die bedienende Person vor dem Industrieroboter bei fälschlicherweise als korrekt erkannten Gesten und ist daher unverzichtbar zum Schutz des eigenen Leibes. Eine ergonomische Lösung für die Form und Art des „Dead Man's Switch“ zu finden würde jedoch über den Rahmen dieser Arbeit hinausgehen, da das Designen eines ergonomischen und sicheren „Dead Man's Switch“ bereits eine Herausforderung für sich selbst darstellt. In dieser Arbeit wurde daher auf den „Dead Man's Switch“ und außerdem auf einen Not-Aus verzichtet, da die Ergonomie der Gesten und das Erkennen von Engpässen im Bezug auf die Übertragungslatenzen im Vordergrund stehen. Zudem wird bereits mit der Tiefenkamera überprüft ob die bedienende Person bei Gestenausführung zum Industrieroboter und dadurch auch zur Tiefenkamera schaut. Zudem stellt der „WidowX 200“-Lernroboter aufgrund seiner Größe, Beschleunigung und maximaler Geschwindigkeit keine lebensbedrohliche Gefahr für Menschen dar. Bei Verwendung der entwickelten Roboter-Gesten-Anwendung außerhalb eines Laborbetriebs (durch geschultes Personal) oder bei Verwendung eines Industrieroboters, welche hohe Beschleunigungen und Geschwindigkeiten zulassen, wird die Einhaltung der entsprechenden Normen (siehe Kapitel 2.4) gefordert. Bei Problemen oder beim Schließen der entwickelten Roboter-Gesten-Anwendung kann angegeben werden ob der Industrieroboter aus Sicherheitsgründen in die Ausgangsposition zurückgefahren werden soll. Dies kann je nach Industrieroboter unterschiedlich sinnvoll sein um z.B. beim „WidowX 200“-Lernroboter ein Zusammenstürzen des Lernroboters zu verhindern, da die Servo-Motoren des „WidowX 200“-Lernroboters nur bei angelegter Spannung die Gelenkpositionen halten können. Bei praxisnahen Industrierobotern ist es zumeist nur sinnvoll den Industrieroboter in seine Ausgangsposition zurückzufahren, wenn der Industrieroboter über einen Kamera verfügt um die bedienende Person nicht ungewollt zu verletzten. Diese Möglichkeit sollte daher mit Bedacht hinsichtlich der Sicherheit der bedienenden Person gewählt werden. Standardmäßig ist diese Option zur höheren Sicherheit der bedienenden Person daher deaktiviert.

3.6. Aufbau des Roboter-Gesten-Frameworks

Die in dieser Arbeit entwickelte Roboter-Gesten-Anwendung baut auf dem Roboter-Gesten-Framework, welches im Rahmen dieser Arbeit erstellt wurde, auf. Der Vorteil des Roboter-Gesten-Frameworks besteht darin, dass durch dessen Einsatz die Entwicklung von einer Anwendung zur Gestensteuerung vereinfacht und dadurch auch beschleunigt wird. Das Roboter-Gesten-Framework bietet neben der Möglichkeit vordefinierte Gestenerkennungsalgorithmen, welche unter anderem Vektorberechnungen auf Basis der Tiefenkamerainformationen durchführen, einzusetzen. Zudem bietet es auch die Möglichkeit an um den Industrieroboter über eine einheitliche Schnittstelle ansprechen zu können. Die Gestensteuerung kann jedoch unabhängig von einem Roboter und ROS erfolgen, da die Gesten- und Roboterkomponenten lose gekoppelt wurden und so mit verschiedenen Systemkonstellationen zusammenarbeiten können. Die Roboter-Gesten-Anwendung wurde als Demonstrationsanwendung entworfen um eine mögliche Nutzung des Roboter-Gesten-Frameworks vorzuzeigen.

3.6.1. Gesten-Subsystem

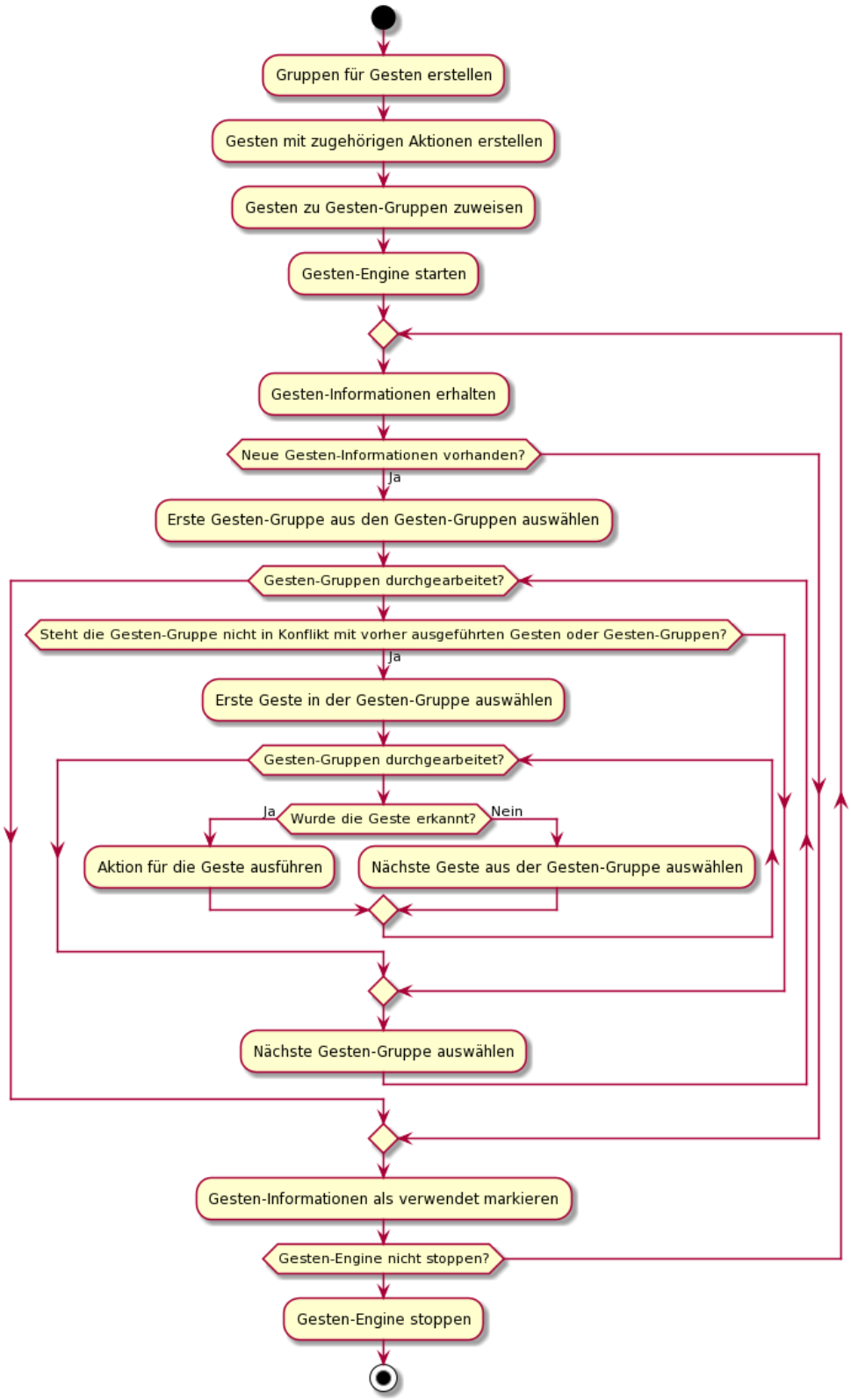

Abbildung 3.4.: Gesten-Subsystem (Aktivitätsdiagramm)
Quelle: Eigene Darstellung

Das Gesten-Subsystem, dessen Ablauf in Abbildung 3.4 ersichtlich ist, verwaltet die zugewiesenen Gesten und deren auszuführende Aktionen. Die Gesten können dabei in sogenannte Gesten-Gruppen unterteilt werden um die Gesten einer kategorischen Gliederung zu unterwerfen. Zudem bieten die Gesten-Gruppen den Vorteil, dass ähnliche Gesten oder gegensätzliche Aktionen auf unkomplizierte Art und Weise voneinander isoliert werden können. Hierzu können Gesten in Gesten-Gruppe eingegliedert und Gesten und Gesten-Gruppen Prioritäten zugewiesen werden. Neben der Zuweisung von Prioritäten können Gesten auch individuelle Ausschlusskriterien zugeordnet werden. Die Gesten-Engine stellt zudem sicher, dass die gleichen Informationen, welche von der Tiefenkamera übertragen werden, nicht mehrmals wiederverwendet werden. Da Aktionen in Gesten-Gruppen zueinander gegensätzlich sein können, kann auch spezifiziert werden ob mehrere Gesten für beide Hände zur gleichen Zeit erkannt werden können.

3.6.2. Abstandsberechnung für Gestensystem

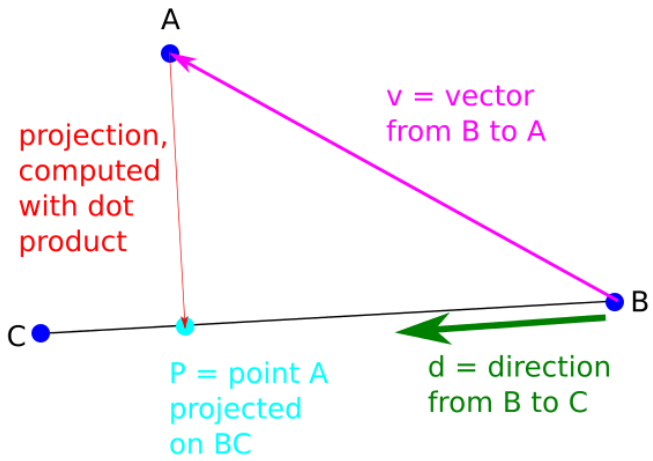

Abbildung 3.5.: Abstandsberechnung für Gestenerkennung
Quelle: [Geo]

In Abbildung 3.5 wird einer der Gestenerkennungsalgorithmen zur Erkennung der Distanz zwischen einem räumlichen Punkt zu einer Linie visualisiert. Dieser Algorithmus kann z.B. dazu genutzt werden um zu erkennen ob sich eine Hand in einem gewissen Bereich befindet. Dies kann sinnvoll sein um unbeabsichtigte Bewegungen außerhalb eines nicht spezifizierten Bereichs zu unterbinden. Zur Visualierung der Punkte A, B und C kann z.B. die Aktion „Gelenkstellung erhöhen“ aus der Tabelle 3.1 herangenommen werden. Die Punkte A und B stellen hierbei das Schlüsselbein und die rechte Schulter dar und der Punkt C wird durch den linken Zeigefinger abgebildet. Hierzu wird zuerst der Punkt A auf die Linie von Punkt B zu Punkt C projiziert. Dieser neue Punkt P hat die geringste Distanz von Punkt A zur Linie BC und zudem steht die Linie AP lotrecht auf die Linie BC. Der neue Punkt P kann nun dazu genutzt werden um die Distanz zwischen P und A zu berechnen. Diese Distanz wird anschließend verwendet um aufgrundlage der maximalen und minimalen Distanz des Bereichs zu entscheiden ob die Geste in einem erlaubten Bereich ausgeführt wurde. In mathematischer Schreibweise kann die

Abbildung 3.5 auch verkürzt als $\left|\frac{|\overrightarrow{BA}\times\overrightarrow{BC}|}{|\overrightarrow{BC}|}\right|$ dargestellt werden.

3.6.3. Roboter-Gesten-Anwendung

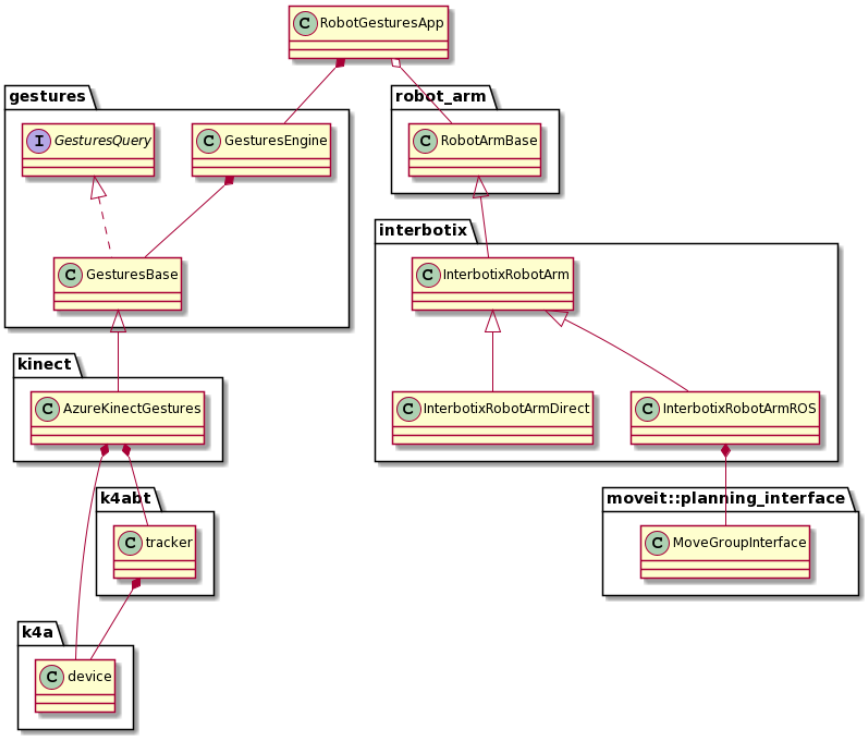

Abbildung 3.6.: Modularität der Roboter-Gesten-Anwendung (Klassendiagramm)
Quelle: Eigene Darstellung

Die Roboter-Gesten-Anwendung wurde mit dem Roboter-Gesten-Framework erstellt und nutzt daher dieses als Grundlage, wie in Abbildung 3.6 mithilfe der „gestures“- und „robot_arm“-Pakete visualisiert wurde. In dieser Anwendung wurden zudem die ausgewählten Gesten mithilfe der Gestenerkennungsalgorithmen des Roboter-Gesten-Frameworks umgesetzt. Hierzu verwendet es die vom Roboter-Gesten-Framework angebotenen Programmierschnittstellen um die Azure Kinect und den „WidowX 200“-Lernroboter, welcher über ROS mittels MoveIt oder über eine direkte Kommunikation angesteuert werden kann, miteinander kommunizieren lassen zu können. Die „AzureKinectGestures“-Klasse verwendet hierbei die von Microsoft bereitgestellte Klasse „tracker“, welche für das Body Tracking zuständig ist, und die Klasse „device“, welche eine Azure Kinect verwaltet. Um vergleichbare Kommunikationsimplementierungen für ROS und die direkte Kommunikation mit dem „WidowX 200“-Lernroboter gewährleisten zu können, musste für den „WidowX 200“-Lernroboter das „InterbotiX“-SDK um eine direkte Kommunikation erweitert werden. Hierfür wurden die vom Roboter-Gesten-Framework bereitgestellten Programmierschnittstellen, wie z.B. die „GesturesBase“-Klasse und das „GesturesQuery“-Interface, verwendet um die einheitliche Kommunikation mit dem „WidowX 200“-Lernroboter zu realisieren. Für den „WidowX 200“-Lernroboter und die Azure Kinect wurden daher die entsprechenden Programmierschnittstellen für die Roboter-Gesten-Anwendung implementiert. Die

genannte Klassenhierarchie und Vererbungsmöglichkeiten in Abbildung 3.6 ermöglicht jedoch die Unterstützung beliebiger Roboter und Tiefenkameras, solange die entsprechenden Programmierschnittstellen implementiert werden.

Listing 3.1: „GesturesQuery“- und „GesturesBase“-Programmierschnittstelle zur Anbindung einer Tiefenkamera

```
class GesturesQuery {
public:
    virtual bool IsGesture(uint32_t startJointIndex, uint32_t
      endJointIndex, uint32_t jointIndex, double minDistance,
      double maxDistance) const = 0;
    virtual bool IsGesture(uint32_t jointIndex1, uint32_t
      jointIndex2, double minDistance, double maxDistance) const
      = 0;
    virtual bool IsJointVisible(uint32_t jointIndex) const = 0;
};

class GesturesBase : public GesturesQuery {
public:
    virtual void NextCycle() = 0;
    virtual bool IsNewDataAvailable() = 0;
};
```

In Listing 3.1 sind die in Abbildung 3.6 ersichtlichen Basisklassen „GesturesBase“ und „GesturesQuery“ ersichtlich. Diese zwei Basisklassen sind notwendig um eine beliebige Tiefenkamera anbinden zu können. Eine Referenzimplementierung wurde bereits für die Azure Kinect implementiert. Die Klasse „GesturesQuery“ ist dafür vorgesehen um die Low-Level-Gestenimplementierung der Tiefenkamera von der Anwendung zu entkoppeln. Der Vorteil besteht darin, dass hierdurch die Geste vom Entwickler nur einmal implementiert werden muss und die eigentliche Low-Level-Implementierung stattdessen von der Tiefenkamera-Implementation abstrahiert wird. Hiedurch wird der Zugriff auf die Funktionalitäten der Tiefenkamera vereinheitlicht und ein einfaches Austauschen der Tiefenkamera ohne aufwändige Anpassungen ermöglicht. Die „GesturesQuery“-Klasse kann mit der Methode „IsGesture“ abfragen ob zwei oder drei Punkte eines erkannten Körperbereichs in einem bestimmten Bereich zwischen minimaler und maximaler Distanz liegt. Z.B. kann hierdurch abgefragt werden ob der Zeigefinger zwischen dem Schlüsselbein und der Schulter liegt. Zudem kann abgefragt werden ob ein Körperbereich sichtbar ist und daher von der Tiefenkamera erkannt wurde. Dies ist unter anderem sinnvoll um zu erkennen ob die bedienende Person während der Durchführung der Gesten zur Tiefenkamera schaut. Mit der „GesturesBase“-Klasse können mit der Methode „NextCycle“ neue Tiefeninformationen verarbeitet und mit der Methode „IsNewDataAvailable“ überprüft werden ob neue bereits verarbeitete Tiefeninformationen bereitstehen.

Listing 3.2: „RobotArmBase“-Programmierschnittstelle zur Anbindung eines Industrieroboters

```
class RobotArmBase {
```

```
public:
    virtual std::unordered_map<JointNameImpl, JointState>
      GetJointStates() = 0;
    virtual bool GetCurrentPose(const JointName&
      endEffectorJointName, geometry_msgs::Pose& pose) = 0;
    virtual void SendJointCommand(const JointName& jointName,
      double value) = 0;
    virtual void SendJointCommands(const std::unordered_map<
      JointNameImpl, double>& jointValues) = 0;
    virtual void SendJointTrajectory(const std::unordered_map<
      JointNameImpl, JointTrajectoryPoint>& jointTrajectoryPoints)
      = 0;
    virtual void SendGripperCommand(double value) = 0;
    virtual void SendGripperTrajectory(const std::unordered_map<
      JointNameImpl, JointTrajectoryPoint>& jointTrajectoryPoints)
      = 0;
    virtual void SendPose(const geometry_msgs::Pose& pose, const
      JointName& endEffectorJointName) = 0;
    virtual void SetOperatingMode(const OperatingMode&
      operatingMode, const AffectedJoints& affectedJoints, const
      JointName& jointName, bool useCustomProfiles, int
      profileVelocity, int profileAcceleration) = 0;
    virtual std::shared_ptr<RobotInfo> GetRobotInfo() = 0;
    virtual double CalculateAccelerationDistance(const JointName&
      jointName, const std::chrono::milliseconds& duration) = 0;
    virtual double CalculateAccelerationDistance(const std::chrono::
      milliseconds& duration) = 0;
};
```

Aufgrund des Sicherheitsaspekts muss zudem die „RobotArmBase“-Klasse, welche als Programmierschnittstelle zum Industrieroboter dient und in Listing 3.2 ersichtlich ist, zur Anbindung eines Industrieroboters an das Roboter-Gesten-Framework für einen spezifischen Industrieroboter separat implementiert werden. Dies ist notwendig, da jeder Industrieroboter unterschiedliche Anforderungen an die Gelenkbeschleunigung stellt, wobei jedoch eine zu hohe Gelenkbeschleunigung zu lebensbedrohlichen Situationen führen kann. Daher ist es sinnvoll die Verantwortung einer geeigneten Gelenkbeschleunigung an die Entwickler des spezifischen Roboters abzugeben. Eine Referenzimplementierung wurde bereits für den „WidowX 200“-Lernroboter implementiert. Die „RobotArmBase“-Klasse bietet verschiedene Methoden an um Gelenkpositionen und Posen zu setzen und Informationen über den Industrieroboter zu erhalten. Mit der Methode „GetJointStates“ können alle derzeitgen Gelenpositionen des Industrieroboters abgefragt und mit der Methode „GetCurrentPose“ die Posen spezifischer Gelenke ermittelt werden. Die Methode „SendJointCommand“ erlaubt es für ein spezifisches Gelenk die Gelenkposition zu setzen und mit der Methode „SendJointCommands“ können die Gelenkpositionen aller vorhandener Gelenke zugleich eingestellt werden. Mit den Methoden „SendJointTrajectory“ und „SendGripperTrajectory“ können Bewegungsbahnen für spezifische Gelenke oder den TCP an den Industrieroboter gesendet werden. Zur Steuerung des TCP kann die

Methode „SendGripperCommand“ verwendet werden. Die Methode „SendPose“ setzt die Pose eines spezifischen Gelenks und die Methode „SetOperatingMode“ ermöglicht unter anderem das Wechseln zwischen Positions-, Spannungs- und Pulsweitenmodulations-Modus sowie weiteren Modi. Mit der Methode „CalculateAccelerationDistance“ kann mit der für den Industrieroboter zulässigen Gelenkbeschleunigung und Zeitdauer die zulässige Distanz der Gelenkposition ermittelt werden. Abschließend ist noch die Methode „GetRobotInfo“ zu erwähnen, welche es ermöglicht detailliertere Informationen, wie z.B. die Anzahl der Gelenke, die Schlafposition, die Startposition oder die Verfügbarkeit eines TCPs des jeweiligen Industrieroboters, zu ermitteln.

3.6.4. Regelkreis

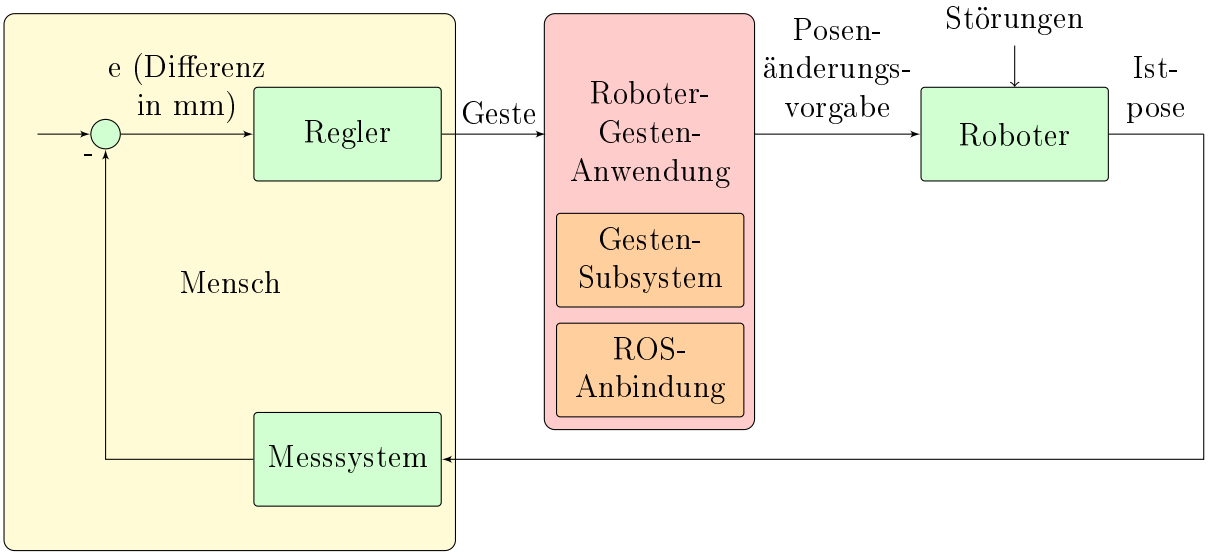

Abbildung 3.7.: Regelkreis der Roboter-Gesten-Anwendung
Quelle: Eigene Darstellung

Der Industrieroboter wird über den Mensch, welcher in dieser Arbeit die bedienende Person ist, mithilfe der Roboter-Gesten-Anwendung bewegt. Dieser Datenfluss kann mit dem Regelkreis in Abbildung 3.7 visualisiert werden. Bei einer durch das Gesten-Subsystem korrekt erkannten Eingabe des Menschen werden die Gelenke des Industrieroboters auf die notwendige Art und Weise eingestellt um die gewünschte Zielpose zu erreichen. Hierbei fungiert der Mensch, wie in Abbildung 3.7 zu sehen ist, als Regler und die Roboter-Gesten-Anwendung als Stellsystem. Die ROS-Anbindung ermöglicht es zudem die Steuerbefehle neben der direkten Kommunikation auch über ROS zu versenden. Störungen, wie z.B. Gelenkungenauigkeiten, können dazu führen, dass der Roboter die gewünschte Pose nicht wie gewünscht erreicht. Der Mensch korrigiert deswegen die Pose des Industrieroboters solange bis die gewünschte Zielpose für den Anwendungsfall präzise genug erreicht wurde. Die Messung der Zielpose kann dabei z.B. über eine Schätzung oder ein genaues Ablesen in einer Simulationsumgebung über den Mensch erfolgen. In der derzeitgen Implementation der Roboter-Gesten-Anwendung ist es nicht möglich die aufgezeichneten Gelenkpositionen mit unterschiedlichen Geschwindigkeiten abzufahren. Der

Industrieroboter wählt zudem beim Abspielen der aufgezeichneten Positionen im PTP-Modus seine eigene Bahn anstatt die Zwischenpunkte zwischen den aufgezeichneten Positionen anhand einer vorgegebenen Bahn, wie z.B. einer Kurve oder einer Gerade, zu interpolieren.

3.6.5. MoveIt-Interaktion mit InterbotiX-Node

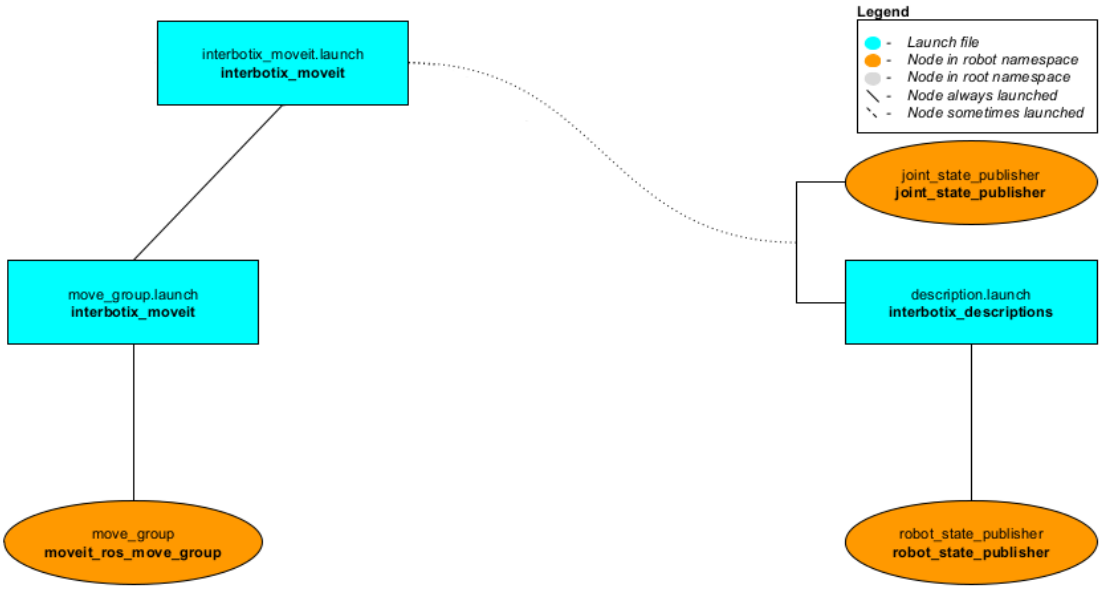

Abbildung 3.8.: Interaktion von MoveIt mit InterbotiX-Node
Quelle: Eigene Darstellung, in Anlehnung an [Int]

In Abbildung 3.8 ist die ROS-Kommunikaton von MoveIt mit dem verwendeten InterbotiX-Node, welcher mit dem „WidowX 200“-Lernroboter kommuniziert, visualisiert. Hierbei ist anzumerken, dass der InterbotiX-Node durch MoveIt gesteuert werden kann und so die Steuerungsbefehle zuerst an MoveIt und anschließend an den InterbotiX-Node gesendet werden. MoveIt kann daher auf die Informationen der Gelenke mithilfe des „robot_state_publisher“-Nodes lesend darauf zugreifen und zudem mithilfe des „moveit_ros_move_group“-Nodes Steuerungsbefehle an den InterbotiX-Node senden um die Gelenkpositionen des „WidowX 200“-Lernroboters vom InterbotiX-Node anpassen zu lassen.

3.7. Messvoraussetzungen

Zur Messung der Übertragunslatenz der Gestensteuerung und zum Messen der Übertragungslatenz der Anbindung mit oder ohne ROS müssen die Messvoraussetzungen für die Replizierbarkeit zuerst definiert werden. Die Messungen und deren Visualierungen basieren auf den Best Practices, welche im wissenschaftlichen Artikel „Ten Simple Rules for Effective Statistical-Practice“ veröffentlicht wurden. Diese Regeln des wissenschaftlichen Artikels lauten wie folgt [Kas+16]:

1. **Statistische Methoden sollen Daten ermöglichen, wissenschaftliche Fragen zu beantworten**: Es soll nicht auf der Struktur der Daten sondern aufgrundlage des wisschenschaftlichen Ziels der Daten gearbeitet werden. Z.B. können Daten aus einer Tabelle nach unterschiedlichen Aspekten klassifiziert werden anstatt diese direkt zu verwenden.
2. **Signale sind rauschbehaftet**: Analoge Signale sind immer rauschbehaftet, da diese kontinuierliche Signale darstellen, und das Messen von physikalischen Größen immer mit Messfehlern behaftet ist.
3. **Vorausplanen**: Es ist sinnvoll sich im Vorhinein Fragen zu stellen, welche den möglichen Ausgang der Messergebnisse betreffen. Zudem ist es auch hilfreich sich die möglichen Störgrößen klarzumachen um mögliche Probleme im Vorhinein zu lösen.
4. **Über die Datenqualität nachdenken**: Es sollte überprüft werden ob die Daten vollständig sind, Anomalien in den Daten auftauchen und die Daten bereits vorbehandelt wurden.
5. **Statistische Analyse ist mehr als eine Reihe von Berechnungen**: Es ist entscheident, dass die analythische Technik in geeigneter Weise mit den zu beantwortenden Fragen verknüpft ist. Daher sollte die Wahl der statischen Methode ausreichend begründet werden.
6. **Simplizität (KISS)**: Eine große Anzahl von Messungen, Nachbearbeitungsmethoden oder ähnlichem kann zu einem starken Anstieg der Komplexität führen. Daher sollten die Daten und Messvorgänge so aufbereitet werden, dass diese nachvollziehbar und reproduzierbar sind.
7. **Bewertung der Variabilität**: Je nach unterschiedlichen Umgebungskriteren bei den Messungen können mitunter Fehler nicht vollständig ausgeschlossen werden. Es ist sinnvoll die Daten hinsichtlich vorhandener Fehler zu hinterfragen.
8. **Überprüfen der Annahmen**: Es sollten die Annahmen überprüft werden und bei Abweichungen sollten diese begründet werden können.
9. **Wenn möglich replizieren**: Die Messungen sollten mehrmals in unterschiedlichen Testsituationen durchgeführt werden um so die Robustheit der Daten sicherstellen zu können.
10. **Analyse reproduzierbar machen**: Die Ergebnisse sollten nachvollziehbar gestaltet werden, sodass diese von Dritten auf die gleiche Art und Weise repliziert werden können.

Die Azure Kinect wird in einer Höhe von 0,85 m in einer zur Oberfläche geraden Ausrichtung positioniert. Der Abstand der testenden Person zur Azure Kinect beträgt mindestens 0,5 bis maximal 1,80 m und die testende Person hat eine Körpergröße von 1,78 m. Der Testraum ist ein mit sonnenlicht belichteter Raum, welcher eine Raumtemperatur von 22,7 °C aufweist und die empfohlene Betriebsumgebung für die Azure Kinect besitzt [tesb]. Der Testrechner auf dem die Roboter-Gesten-Anwendung ausgeführt wird weist eine SSD und 8 GB RAM auf und hat einen AMD Ryzen 5 3600X 6-Kernprozessor, welcher über bis zu 12 logische Kerne verfügt, verbaut. Die Grafikkarte des Testsystems ist eine AMD Radeon RX 5700 XT wodurch das Body Tracking der Azure Kinect nur im CPU-Modus verwendet werden kann, da das Body Tracking der Azure Kinect nur CUDA basierte Nvidia Grafikkarten per Grafikkartenbeschleunigung zulässt [Enc]. Der CPU-Modus des Azure Kinect Body Tracking SDKs stellt daher der kleinste gemeinsame Nenner für alle Systeme dar und spiegelt so die Performanz eines allgemein verfügbaren Systems wider [Azuc].

Neben der Übertragungslatenz sind die genannten Messvoraussetzungen, aber auch für die

Messung der Genauigkeit der erreichten Zielpositionen erforderlich. Zur Messung der Genauigkeit der erreichten Zielpositionen wird ein möglichst praxisnaher Teachvorgang durchgeführt. Es soll eine Kontur so präzise wie möglich nachgefahren werden. Dabei ist zu beachten, dass in einem industriellen Szenario hierbei der Endeffektor z.B. für Laserschneiden ausgelegt sein könnte. Bei der in dieser Arbeit durchgeführten Tests wird hingegen auf den standarmäßig verbauten Endeffektor des „WidowX 200“-Lernroboters zurückgegriffen.

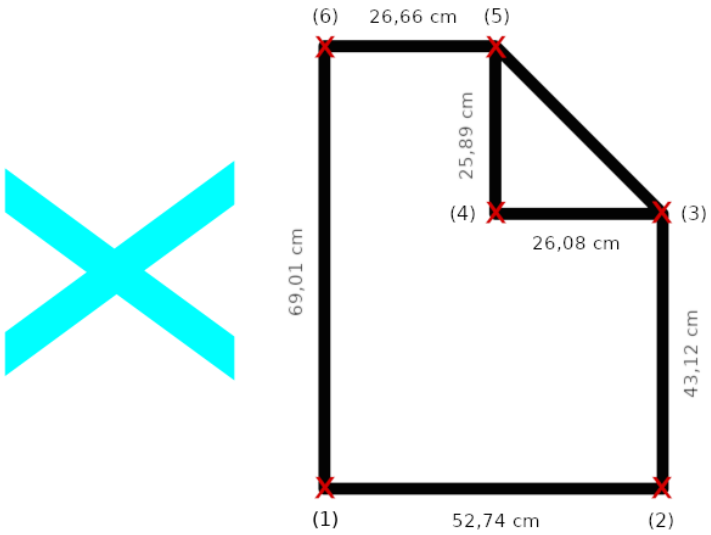

Abbildung 3.9.: Teachbeispiel: Weiße Seite mit gefalteter Ecke
Quelle: Eigene Darstellung

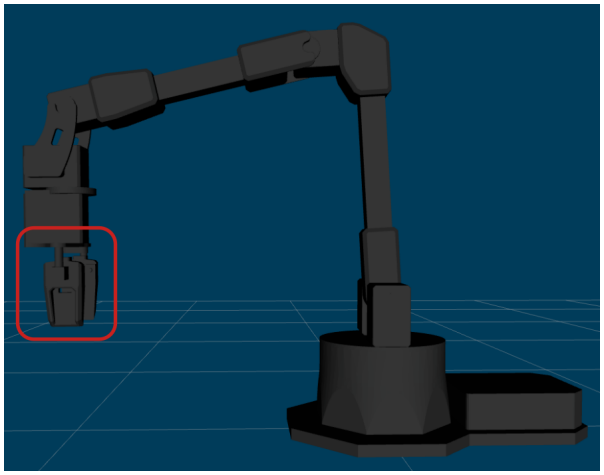

Abbildung 3.10.: Endeffektorpose für Testbeispiel
Quelle: Eigene Darstellung

Zum Messen der Genauigkeit der anzufahrenden Ziele und zur Ermittlung der Genauigkeit der Gestenerkennung sollen die mit einem roten Kreuz markierten Positionen, wie in Abbildung 3.9 zu sehen sind, unabhängig von der Orientierung in der Simulationsumgebung Gazebo

angefahren werden. Aus diesem Grund wird hierbei aus der Vielfalt der Testmöglichkeiten der ISO-Norm „ISO 9283“, zu denen die Genauigkeit der Roboterposition, die Wiederholgenauigkeit und die Bahngenauigkeit zählt [140a], nur die Genauigkeit der Roboterposition gemessen. Da sich die Norm „ISO/TR 13309“ zudem auf die ISO-Norm „ISO 9283“ bezieht wird hierdurch der Test auf die Positionsgenauigkeit, welcher beim „WidowX 200“-Lernroboter wenige Millimeter betragen sollte, beschränkt. Die Norm „ISO/TR 13309“ beinhaltet insgesamt 14 Kriterien zu denen unter anderem die Posengenauigkeit, Wiederholgenauigkeit, Pfadgenauigkeit und die minimale Zeit zum Erreichen der Pose gezählt werden [140b]. Bei den durchzuführenden Messungen startet der „WidowX 200“-Lernroboter immer in seiner Standard-Ausgangsposition. Der Mittelpunkt des Industrieroboter-Sockels steht auf dem Mittelpunkt des türkisen Kreuzes. Um die Kontur nachfahren zu können soll der Endeffektor aus der Anfangs schrägen Pose zuerst so wie in Abbildung 3.10 positioniert werden. Daraufhin sollen die Mittelpunkte der rot markierten Kreuze mit dem Greifer angefahren werden, sodass der Mittelpunkt des Greifers über dem roten Kreuzpunkt liegt. Die Zeit zum Erreichen der Zielpositionen und die Abweichung zu den Zielpositionen werden über einen Testlauf hinweg gesammelt und daraufhin ausgewertet. Die vorraussichtliche Durchlaufzeit für den Testlauf wird auf 15 Minuten geschätzt. Um die Latenz der Gestenerkennung und der Kommunikation mit dem Roboter zu ermitteln und statistisch aussagekräftige Daten zu erhalten werden zudem Latenztests anhand eines realen „WidowX 200“-Lernroboters durchgeführt. Anzumerken ist, dass aufgrund der 5 DOF des „WidowX 200“-Lernroboters nur ein Teilbereich der Posen des 3D-Raums ansteuerbar ist. Hierfür wäre ein Industrieroboter mit mindestens 6 DOF notwendig. Im Weltkoordinatensystem-Modus und daher für „Continuous Path“-Pfade kann es deshalb aufgrund der 5 DOF des „WidowX 200“-Lernroboters passieren, dass dieser seine Gelenkpositionen mit sehr großen Positionsänderungen, welche in sprunghaften Änderungen des Roboters ersichtlich sind, anpassen muss um Positionen im 3D-Raum erreichen zu können.

4. Ergebnisse

Auf Grundlage der zuvor genannten Messvoraussetzungen konnte die Roboter-Gesten-Anwendung verschiedenen Tests unterzogen und deren Ergebnisse ausgewertet werden. Zur Durchführung der Tests wurde die Roboter-Gesten-Anwendung mit zusätzlichen „*_MEASUREMENT“-Flags, zu denen die Flags „GESTURE_MEASUREMENT“, „COMMUNICATION_MEASUREMENT“ und „POSITION_MEASUREMENT“ zählen, kompiliert um ausführliche Statistikdaten für den jeweils durchgeführten Testfall zu erhalten. Die durchgeführten Tests reichen hierbei von Latenztests der Gestenerkennung mittels der Azure Kinect über Latenztests der Kommunikation mittels ROS und der direkten Kommunikation mit dem „WidowX 200“-Lernroboter bis hin zu der Genauigkeit der anzufahrenden Zielpositionen. Die erhobenen statistischen Daten wurden analysiert und werden nachfolgend mittels Diagrammen veranschaulicht und kritisch hinterfragt.

4.1. Latenz der Gestenerkennung

Zum Testen der Latenz der Gestenerkennung wurde das Azure Kinect Body Tracking SDK im CPU-Modus anstatt mit der Grafikkartenbeschleunigung gestartet. Dies war notwendig, da eine Grafikkarte von AMD statt einer Nvidia Grafikkarte im Testsystem eingesetzt wird. Zudem kann hiermit der kleinste gemeinsame Nenner der im Umlauf befindlicher Systeme widergespiegelt werden, welche keine propritären Features von Nvidia Grafikkarten, wie z.B. CUDA, verwenden. Die Latenz der Gestenerkennung beinhaltet die Zeit von der Bildaufnahme bis hin zur Latenz der Kommunikation mittels USB 3.0 und der Verarbeitung der Daten durch das Azure Kinect Body Tracking SDK. Zum Testen der Latenz der Gestenerkennung wurde die Roboter-Gesten-Anwendung und deren Abhängigkeiten mittels der Flags „MEASUREMENT“ und „GESTURE_MEASUREMENT“, welche im Anhang A.1.2 aufgezeigt werden, kompiliert. Bei der Durchführung des Tests wurden von dem Proband die in dieser Arbeit entwickelten Gesten abwechselnd in zufälliger Reihenfolge vor der Azure Kinect ausgeführt. Die Datei „Messung der Gestenerkennung.csv“, welche bei der Durchführung des Tests von der Roboter-Gesten-Anwendung erstellt wurde, ist auf dem Begleitmedium im Verzeichnis „Testdaten/“ zu finden.

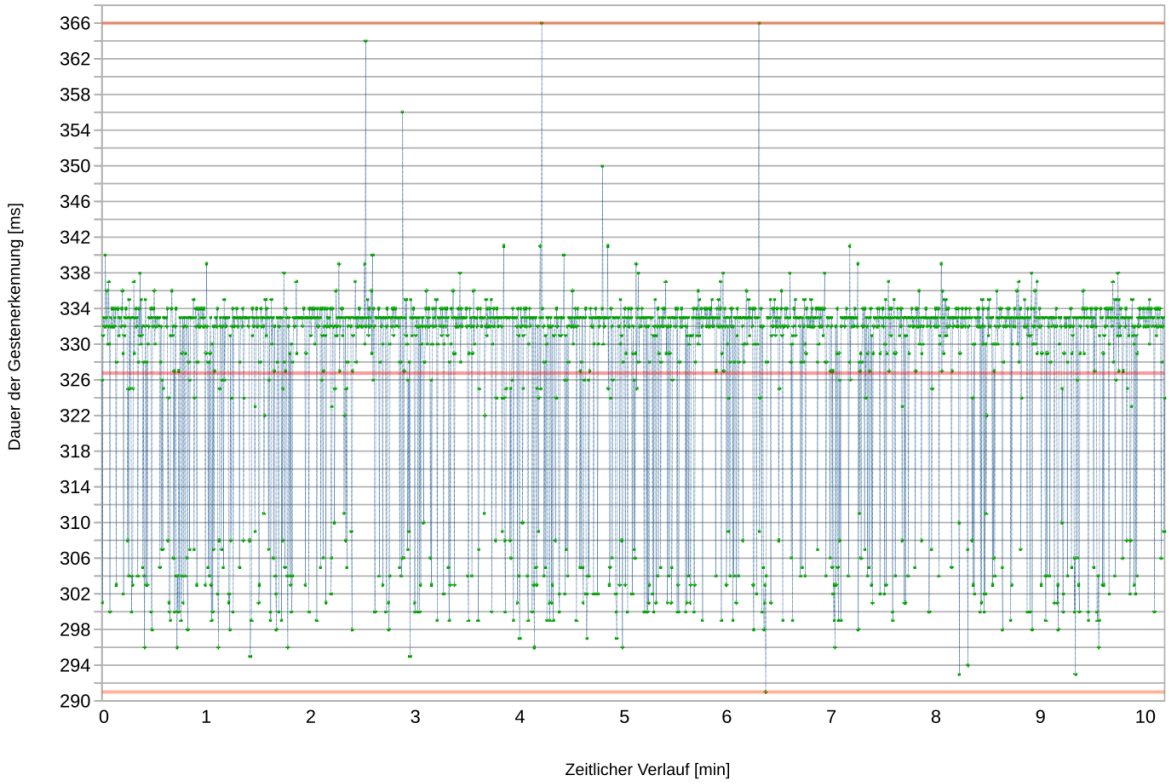

Abbildung 4.1.: Zeitlicher Latenzverlauf der Gestenerkennung der Azure Kinect
Quelle: Eigene Ausarbeitung

In Abbildung 4.1 ist zu sehen, dass der Mittelwert der Gestenerkennungsdauer, welcher durch die hellrote Linie visualisiert wird, in Richtung 326,29 ms tendiert. Die minimale Dauer der Gestenerkennung, welche durch die untere orange Linie visualisiert wird, beträgt 291 ms. Die maximale Dauer der Gestenerkennung, welche durch die obere orange Linie visualisiert wird, beträgt hingegen 366 ms. Die Tendenz der Gestenerkennungsdauer scheint zudem annähernd gleich zu bleiben und wird höchstwahrscheinlich nur durch Schwankungen beeinflusst. Es ist zudem anzumerken, dass der Großteil der Messdaten vermehrt im Bereich zwischen 322 und 340 ms zu liegen scheint. Daher ist anzunehmen, dass die Gestenerkennungsdauer vermehrt in diesem Bereich zu erwarten ist.

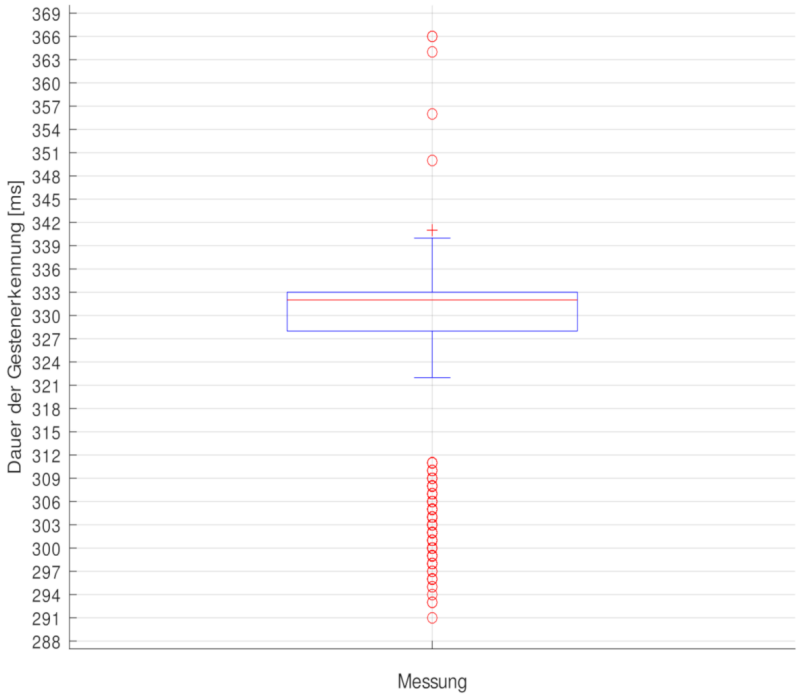

Abbildung 4.2.: Latenzverteilung der Gestenerkennung der Azure Kinect
Quelle: Eigene Ausarbeitung

Wie in Abbildung 4.2 zu sehen ist, liegt der Großteil der Messwerte im Bereich zwischen 320 und 340 ms. Dabei ist zu beachten, dass 50 % der Messwerte jedoch in einem kleineren Bereich zwischen 328 und 333 ms liegen. Der Median, welcher die kleinsten 50 % der Messwerte aufzeigt, liegt zudem bei 332 ms. Da der Median im oberen Bereich der Box liegt, kann darauf geschlossen werden, dass eine linksschiefe Verteilung der Messdaten vorliegt. Dies bedeutet, dass der Median größer als der Mittelwert und die Gestenerkennungsdauer daher vermehrt im oberen Bereich der Verteilung angesiedelt ist. Anzumerken ist zudem, dass sich die Gestenerkennungsdauer im Durchschnitt stabil im Bereich zwischen 322 und 340 ms bewegt. Die annähernd gleich bleibende Gestenerkennungsdauer kann auf das DNN des Azure Kinect Body Tracking SDKs und den eingesetzten Prozess-Scheduler zurückgeführt werden. Eine nichtdeterministische Berechnung würde hierbei vermehrt Spikes und unvorhersagbarere Änderungen bei der Gestenerkennungsdauer bewirken. Nichtsdestotrotz liegen diese Werte weit über den 50 ms, welches für ein Teach Pendant empfohlen wird [Pra+04, S. 55]. Zudem ist eine Verzögerung der Gestenerkennung bei über 100 ms für die bedienende Person bereits deutlich spürbar [Mil68].

4.2. Latenz der Roboter-Kommunikation mit und ohne ROS-Anbindung

Zum Testen der Latenz der Roboter-Kommunikation mit und ohne ROS-Anbindung wurde bei der direkten Kommunikation zum Roboter die Latenz zwischen der aufrufenden Roboter-

Gesten-Anwendung und der InterbotiX-Schnittstelle gemessen. Bei der ROS-Anbindung wird die Latenz der Kommunikation zwischen der Roboter-Gesten-Anwendung über den TCP/IP-Stack und daher über das Netzwerk bis hin zu der InterbotiX-Schnittstelle analysiert und gemessen. Bei der Untersuchung der Netzwerklatenz werden verschiedene praxisnahe Netzwerkkonstellationen simuliert um unvorhergesehene Netzwerklatenzen auffinden und diesen gegebenenfalls entgegenwirken zu können. Die Tests werden in einem Gigabit-Ethernet-Netzwerk durchgeführt, da Gigabit-Ethernet zum Zeitpunkt der Veröffentlichung dieser Arbeit als Standard angesehen wird und bereits weit verbreitet bei Routern und Clientsystemen ist [Gig].

Zur Simulation der Netzwerkumgebungen wird NetEm[1] mit Token Bucket Filter[2] eingesetzt. NetEm stellt ein Netzwerkemulator dar und durch den Token Bucket Filter kann eine Warteschlange simuliert werden. Mithilfe von NetEm und Token Bucket Filter können verschiedenste Netzwerkschnittstellen simuliert werden. Hierzu werden die im Linux-Kernel vorhandenen Möglichkeiten zur Steuerung des Netzwerkverkehrs eingesetzt. Bei ausgehenden Paketen können so Verzögerungen, Paketverlust, Duplizierung, Datenratenlimits und weitere Merkmale emuliert werden um eine Netzwerkschnittstelle an die Gegebenheiten eines vorhandenen Netzwerks anzupassen und z.B. Tests ohne die reale Netzwerkhardware reproduzierbar zu machen [Hem]. Ein Gigabit-Ethernet-Netzwerk, welches mit einem Router verbunden ist, kann wie in Listing 4.1 simuliert werden.

Listing 4.1: Gigabit-Ethernet-Netzwerksimulation inklusive Simulation einer Routeranbindung mit NetEm und Token Bucket Filter

```
sudo tc qdisc add dev lo root handle 1: tbf rate 940Mbit burst 8192 limit 16384
sudo tc qdisc add dev lo parent 1:1 handle 10 netem corrupt 0.5% loss 0.1% delay \
0.2ms 0.05ms distribution normal reorder 3% 50%
```

In Listing 4.1 wird das Kommandozeilenprogramm „tc“ verwendet um auf die Funktionalitäten von NetEm und des Token Bucket Filters zugreifen zu können. Zur Verwendung von „tc“ werden Administratorrechte benötigt, da mit diesem Tool grundlegende Netzwerkeigenschaften, wie z.B. Datenraten, von vorhandenen Netzwerk-Interfaces, wie z.B. lo, eth*, eno* und wlo*, verändert werden können. „qdisc“ ist die Abkürzung für „Queueing Discipline“ und stellt den Netzwerkpakete-Scheduler von Linux dar, welcher die Netzwerkpakete entgegennimmt. Mit „add dev“ kann ein Netzwerkinterface hinzugefügt werden um diesem bestimmte Netzwerkeigenschaften zuweisen zu können. In diesem Fall wird „lo“, welches das Loopback-Interface darstellt und über „localhost“ oder „127.0.0.1“ angesprochen werden kann, ausgewählt. „root“ und „parent“ stellen hierbei Klassen dar, wobei einer „root“-Klasse eine „parent“-Klasse zugewiesen werden kann. Daher wird in Listing 4.1 über „parent 1:1“ das erste verfügbare „root“-Element dem Netzwerk-Interface zugewiesen. Mit der Option „handle“ wird ein Bezeichner zugewiesen über den die jeweilige Klasse angesprochen werden kann. Hierbei ist die Option „handle 1:“ gleichbedeutend mit der Option „handle 1:0“. Aus diesem Grund kann daher über die Option „parent 1:1“ auf die „root“-Klasse zugegriffen werden. Die Optionen „tbf“ und „netem“ spezifizieren den Toket Bucket Filter und NetEm. Mit „rate“ kann zusätzlich die maximale Datenrate des Netzwerkinterface angegeben werden [Tcn]. Bei einem Gigabit-Ethernet-Netzwerk kann hierbei eine Nettorate von 940 Mbit/s erreicht werden [Dat]. Die Option „burst“ gibt an wie viele Bytes

[1] http://manpages.ubuntu.com/manpages/bionic/man8/tc-netem.8.html
[2] http://manpages.ubuntu.com/manpages/bionic/man8/tc-tbf.8.html

an Daten im Schnellzugriffs-Buffer gehalten werden können und die Option „limit“ gibt an wie viele Bytes an Daten insgesamt in der Warteschlange gepuffert werden können bis diese gesendet werden müssen [Tcn]. Bei Gigabit-Ethernet ist ein Burst von 8192 Bytes und ein Limit von 16384 Bytes möglich [Dem07]. Mit der Option „corrupt“ kann die Anzahl fehlerhaften Pakete in Prozent angegeben werden. Die Option „loss“ gibt an wie viele Pakete in Prozent über das Netzwerk verloren gehen sollen. Eine Verzögerung kann mit dem ersten Argument und eine Schwankung mit dem zweiten Argument von der Option „delay“ spezifiziert werden. Eine Verteilungsfunktion kann dazu genutzt werden um eine der Verteilung ähnelnde Verzögerung zu erreichen. Die Pakete können zudem mit der Option „reorder“ neu angeordnet werden um verspätete Netzwerkpakete zu simulieren. Das erste Argument der Option „reorder“ gibt die Anzahl der neu anzuordnenden Pakete und das zweite Argument die Korrelation in Prozent an [Tbf]. Bei einem Gigabit-Ethernet-Netzwerk und einer Routeranbindung dazwischen ist eine Fehlerrate von 0,5 %, eine Verlustrate von 0,1 % und eine Verzögerung von 0,2 ms mit einer Schwankung von 0,05 ms und einer normalverteilten Verzögerung realitätsnah. Eine Neuanordnung der Netzwerkpakete von 3 % mit einer Korrelation von 50 % stellt eine praxisnahe Annäherung an die Gegebenheiten eines Gigabit-Ethernet-Netzerks mit Routeranbindung dar [Adm18].

Bei der Durchführung der Tests wurden von dem Proband die in dieser Arbeit entwickelten Gesten abwechselnd in praxisnaher Reihenfolge vor der Azure Kinect ausgeführt um die Latenz der Kommunikation an einem praxisnahen Testbeispiel testen zu können. Die Tests wurden zuerst an dem Loopback-Interface mithilfe des Gelenk-Modus ohne den in Listing 4.1 angegeben Beschränkungen getestet um eine lokale Interprozesskommunikation mithilfe des TCP/IP-Stacks zu testen. Anschließend wurden die in Listing 4.1 angegebenen Beschränkungen an drei verschiedenen Netzwerkkonstellationen angewandt und vom Proband im Weltkoordinatensystem- und Gelenk-Modus getestet. Die Netzwerkkonstellationen schließt hierbei eine Netzwerksimulation ohne zusätzlichen Netzwerktraffic und eine Netzwerksimulation mit hohem Netzwerktraffic mit ein um die zwei Randfälle testen zu können. Bei der Netzwerksimulation mit hohem Netzwerktraffic werden drei 4K-Bilder-ROS-Nodes simuliert um das Gigabit-Ethernet-Netzwerk mit einer Datenrate von maximal 1176 MB/s vollständig auszulasten. Dies stellt sicher, dass das Testnetzwerk zu jederzeit voll ausgelastet ist. Zum Testen der Latenz der Roboter-Kommunikation mit und ohne ROS-Anbindung wurde die Roboter-Gesten-Anwendung und deren Abhängigkeiten zudem mittels der Flags „MEASUREMENT“ und „COMMUNICATION_MEASUREMENT“, welche im Anhang A.1.2 aufgezeigt werden, kompiliert. Die Dateien „interbotix_robot_arm_communication_measurement.csv“ und „interbotix_sdk_communication_measurement.csv“, welche von der Roboter-Gesten-Anwendung während der jeweiligen Tests erstellt wurden, sind in den jeweiligen Unterordnern „Messung der direkten Kommunikation“, „Messung der ROS-Kommunikation (ohne Netzwerksimulation)“, „Messung der ROS-Kommunikation (mit Netzwerksimulation & niedriger Netzwerkauslastung)“ und „Messung der ROS-Kommunikation (mit Netzwerksimulation & hoher Netzwerkauslastung)“ auf dem Begleitmedium im Verzeichnis „Testdaten/Messung der Kommunikation/“ zu finden.

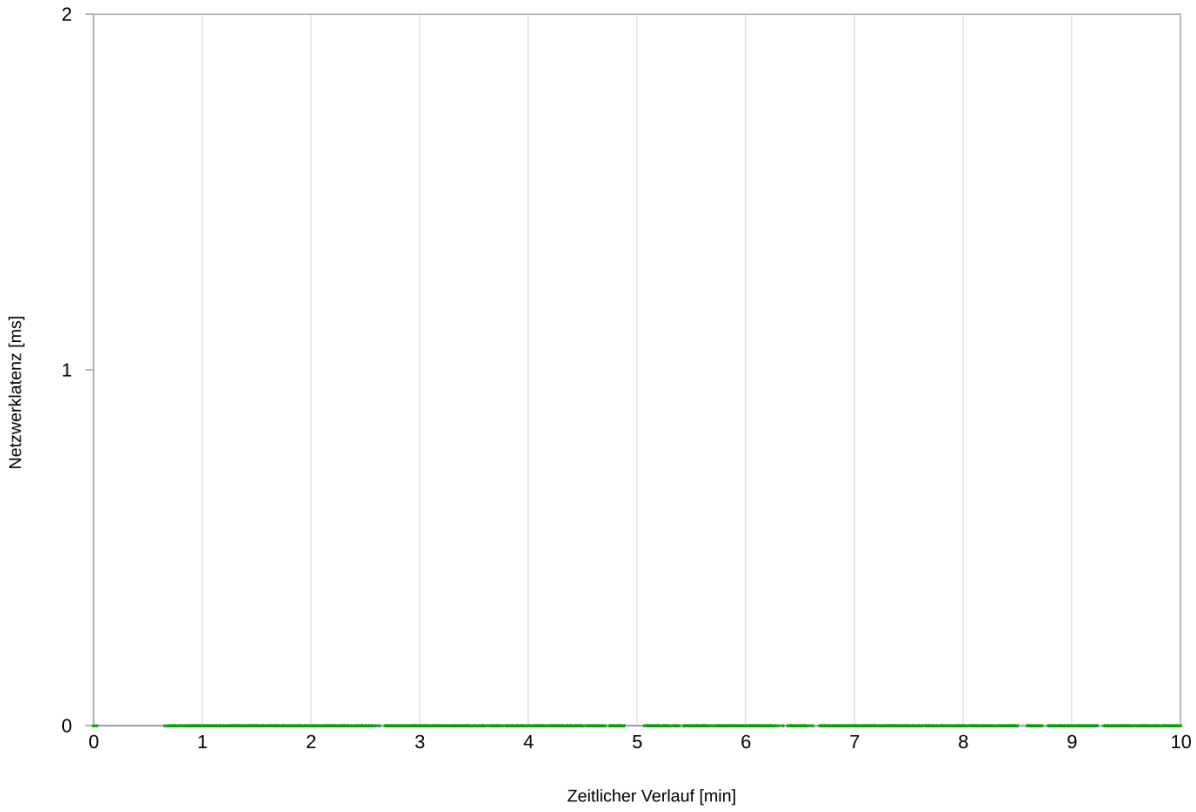

Abbildung 4.3.: Zeitlicher Latenzverlauf der direkten Kommunikation des „WidowX 200“-Lernroboters
Quelle: Eigene Ausarbeitung

Bei der Messung der direkten Kommunikation zum „WidowX 200“-Lernroboter, welche in Abbildung 4.3 ersichtlich ist, konnte eine nicht messbare Verzögerung von deutlich unter 1 ms festgestellt werden. Dies ist damit zu erklären, weil die Kommunikation zwischen der Roboter-Gesten-Anwendung über einen geteilten Speicherbereich erfolgt und die Übertragung der Steuerungsbytes zum „WidowX 200“-Lernroboter über das „USB 2.0“-Kabel aufgrund der geringen Datenmenge mit einer nicht messbaren Verzögerung erfolgen kann. Zudem kann davon ausgegangen werden, dass die Kommunikation der Roboter-Gesten-Anwendung mit der InterbotiX-Schnittstelle sehr effizient erfolgt. Die in Abbildung 4.3 z.T. ersichtlichen schmalen Spalte zwischen den grünen Punkten enstanden durch das Außsetzen oder durch das Nichterkennen der vom Proband durchgeführten Gestenbewegungen. Aufgrund der geringen Kommunikationslatenz zwischen der Roboter-Gesten-Anwendung und der InterbotiX-Schnittstelle ist daher die direkte Kommunikation mit dem „WidowX 200“-Lernroboter wegen der kurzen Übertragungsdauer vernachlässigbar klein. Bei den Tests mit der Netzwerkkommunikation kann aus diesem Grund davon ausgegangen werden, dass die Netzwerkkommunikation höchstwahrscheinlich einen signifikanteren Einfluss auf die Kommunikationslatenz als die Übertragung der Steuerungsbytes zum „WidowX 200“-Lernroboter über das „USB 2.0“-Kabel haben wird.

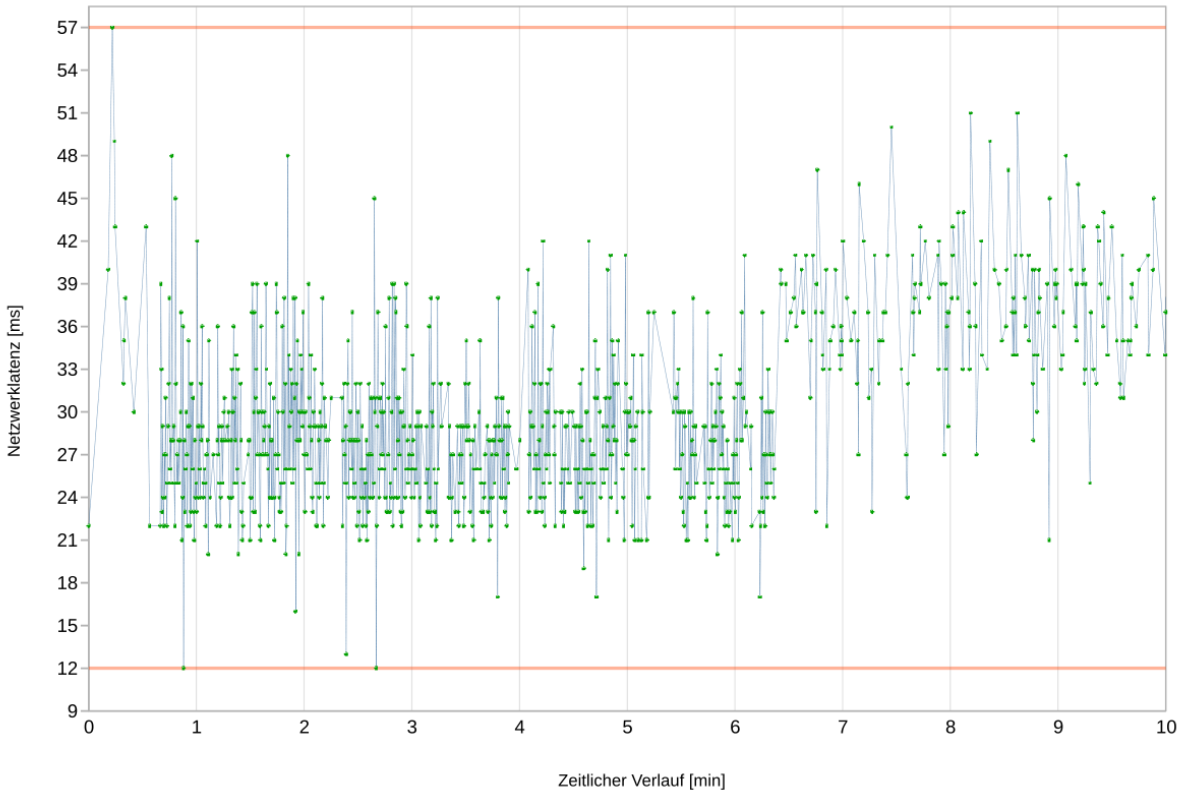

Abbildung 4.4.: Zeitlicher Latenzverlauf der ROS-Kommunikation des „WidowX 200“-Lernroboters (ohne Netzwerksimulation)
Quelle: Eigene Ausarbeitung

Bei der Messung der ROS-Kommunikation über das Loopback-Interface und ohne eine Netzwerksimulation konnte ein Anstieg von Durchschnittlich 29 ms im Gegensatz zur direkten Kommunikation verzeichnet werden. Dies ist damit zu erklären, dass die Kommunikation über den TCP/IP-Stack erfolgt und daher der Netzwerk-Overhead mit zu berücksichtigen ist. Die minimale Latenzzeit der ROS-Kommunikation über das Loopback-Interface, welche durch die untere orange Linie visualisiert wird, beträgt im Extremfall 12 ms. Die maximale Latenzzeit der ROS-Kommunikation über das Loopback-Interface, welche durch die obere orange Linie visualisiert wird, beträgt hingegen im Extremfall 57 ms. Die Latenzzeit liegt im Durchschnitt zwischen 25 und 33 ms. Es ist zudem zu erwähnen, dass der Proband am Anfang des Tests für 30 Sekunden im Weltkoordinatensystem-Modus, danach bis 6 Minuten nach dem Start des Tests im Gelenk-Modus und anschließend wieder im Weltkoordinatensystem-Modus die praxisnahen Gestenbewegungen durchgeführt hatte. Anzumerken ist zudem, dass beim Weltkoordinatensystem-Modus zuerst die inverse Kinematik von MoveIt berechnet werden muss und dieser daraufhin über das Netzwerk und daher über den TCP/IP-Stack zur Roboter-Gesten-Anwendung übertragen werden muss. Daraufhin können die berechneten Gelenkpositionen von der Roboter-Gesten-Anwendung über das Netzwerk und daher über den TCP/IP-Stack mittels MoveIt an den InterbotiX-Node weitergeleitet werden. Aufgrund der extremen Schwankungen ist es womöglich empfehlenswert zeitkritische Steuerungsbefehle mehrmals zu senden um sicherzugehen,

dass diese auch zeitgerecht ankommen. Hierzu müsste jedoch ein Algorithmus implementiert werden um identische Steuerungsbefehle erkennen und ein mehrmaliges Ausführen der Steuerungsbefehle verhindern zu können.

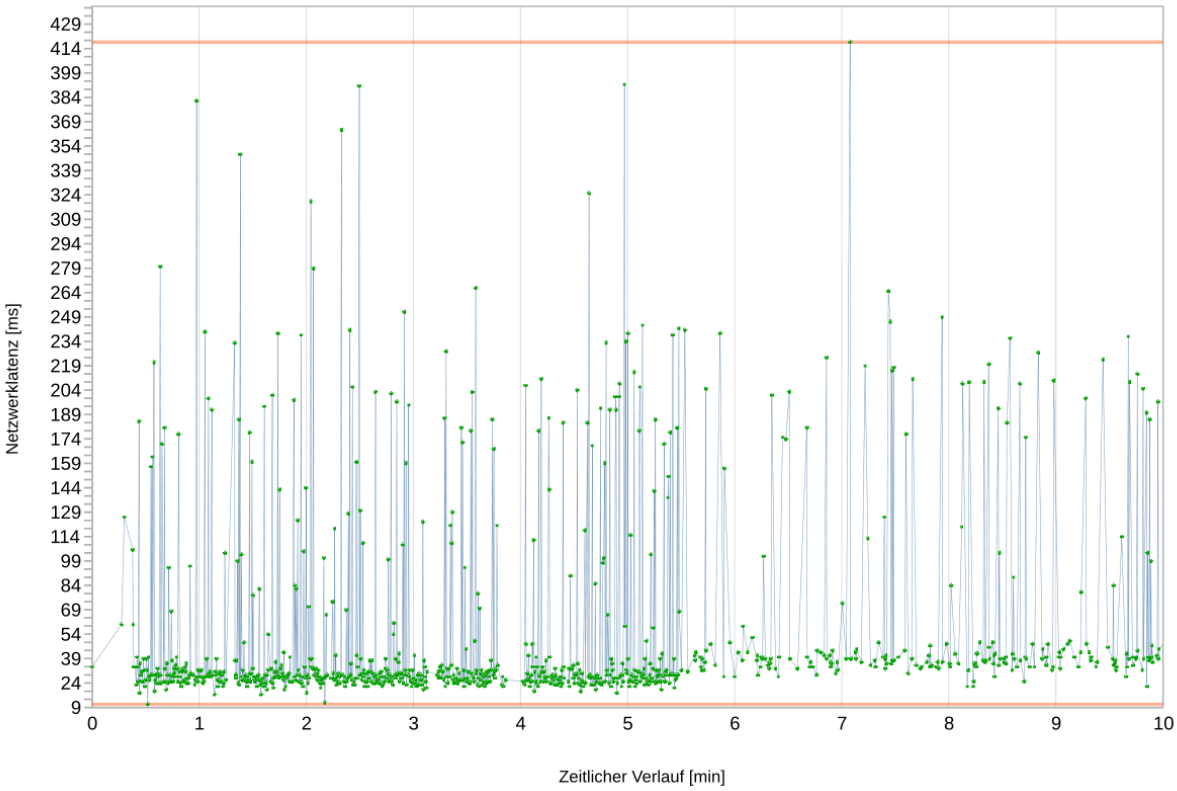

Abbildung 4.5.: Zeitlicher Latenzverlauf der ROS-Kommunikation des „WidowX 200“-Lernroboters (mit Netzwerksimulation & niedriger Netzwerkauslastung)
Quelle: Eigene Ausarbeitung

Bei der Messung der ROS-Kommunikation über das Gigabit-Ethernet-Netzwerk mit einer Netzwerksimulation und einer niedrigen Netzwerkauslastung stellte sich wie erwartet heraus, dass die Latenzeit bei der Kommunikation über das Gigabit-Ethernet-Netzwerk zu mehr Spikes als mit dem Loopback-Interface tendiert. Es ist wiederum anzumerken, dass der Proband am Anfang des Tests für 30 Sekunden im Weltkoordinatensystem-Modus, danach bis 5 Minuten und 30 Sekunden nach dem Start des Tests im Gelenk-Modus und anschließend wieder im Weltkoordinatensystem-Modus die praxisnahen Gestenbewegungen durchgeführt hatte. Bei der Messung der ROS-Kommunikation über das Gigabit-Ethernet-Netzwerk und daher mit einer Netzwerksimulation konnte ein Anstieg von mindestens 20 ms im Gegensatz zur ROS-Kommunikation ohne Netzwerksimulation verzeichnet werden. Die minimale Latenzzeit der ROS-Kommunikation über das Gigabit-Ethernet-Netzwerk, welche durch die untere orange Linie visualisiert wird, beträgt im Extremfall 11 ms. Die maximale Dauer der Gestenerkennung, welche durch die obere orange Linie visualisiert wird, beträgt hingegen im Extremfall 418 ms. Die Latenzzeit liegt im Durchschnitt jedoch im Bereich zwischen 18 und 116 ms. Dies ist damit

zu erklären, dass die Kommunikation über ein realitätsnahes Netzwerk erfolgt und daher der Netzwerk-Overhead und der Overhead des Routers mit zu berücksichtigen ist. Aufgrund der Spikes von Durchschnittlich bis zu 116 ms ist es zudem unmöglich die 50 ms, welche für ein Teach Pendant empfohlen werden [Pra+04, S. 55], einzuhalten.

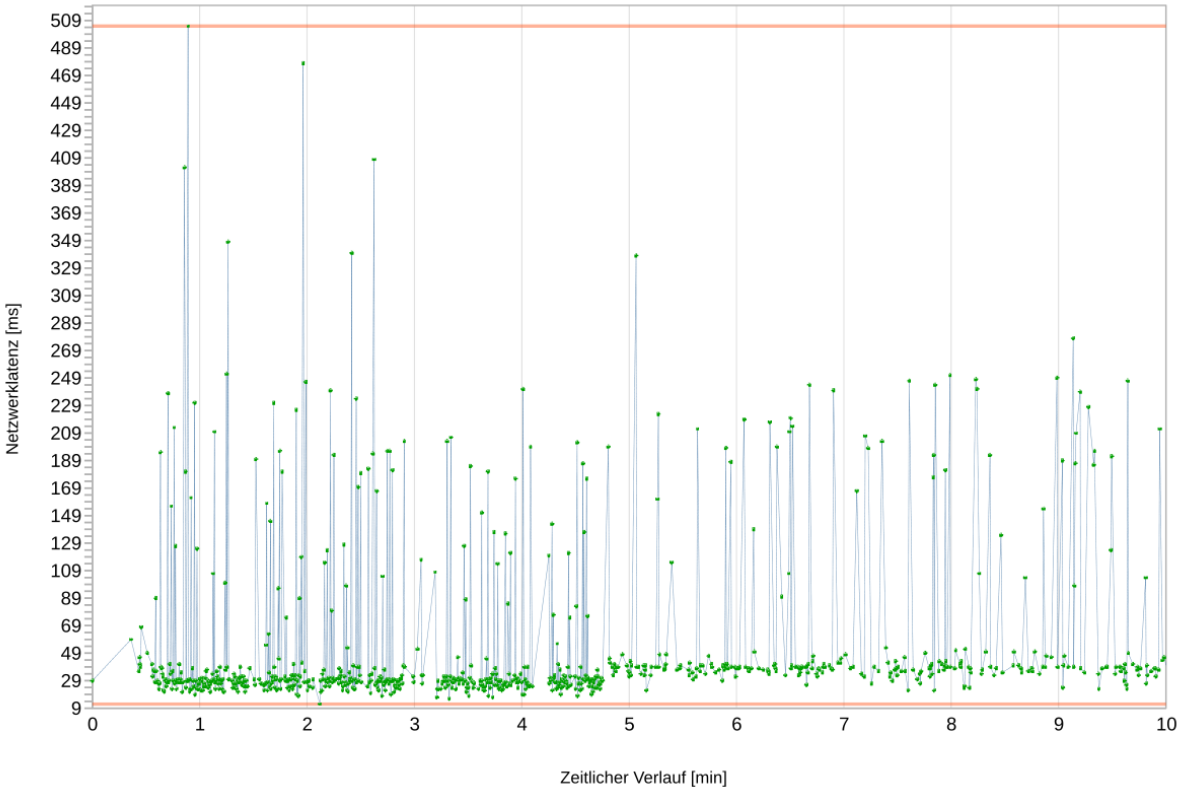

Abbildung 4.6.: Zeitlicher Latenzverlauf der ROS-Kommunikation des „WidowX 200“-Lernroboters (mit Netzwerksimulation & hoher Netzwerkauslastung)
Quelle: Eigene Ausarbeitung

Bei der Messung der ROS-Kommunikation über das Gigabit-Ethernet-Netzwerk mit einer Netzwerksimulation und einer hohen Netzwerkauslastung stellte sich heraus, dass die Latenzeit bei der Kommunikation über das Gigabit-Ethernet-Netzwerk dazu tendiert annähernd gleich zu bleiben. Es konnte zudem kein signifikanter Anstieg im Gegensatz zur ROS-Kommunikation mit Netzwerksimulation und geringer Netzwerkauslastung verzeichnet werden. Nur die maximale Latenzzeit stieg auf 505 ms an, was jedoch aufgrund der geringen Häufigkeit als Ausreißer anzusehen ist. Es ist zudem zu erwähnen, dass der Proband am Anfang des Tests für 30 Sekunden im Weltkoordinatensystem-Modus, danach bis 5 Minuten nach dem Start des Tests im Gelenk-Modus und anschließend wieder im Weltkoordinatensystem-Modus die praxisnahen Gestenbewegungen durchgeführt hatte. Da bei den Tests mit niedriger und hoher Netzwerkauslastung annähernd die gleichen Latenzeiten gemessen wurden, lässt dies darauf schließen, dass die zu übertragende geringe Datenmenge der Planungsdaten und der Steuerungsbefehle die Hauptursache für die beinahe gleich bleibende Latenz darstellt.

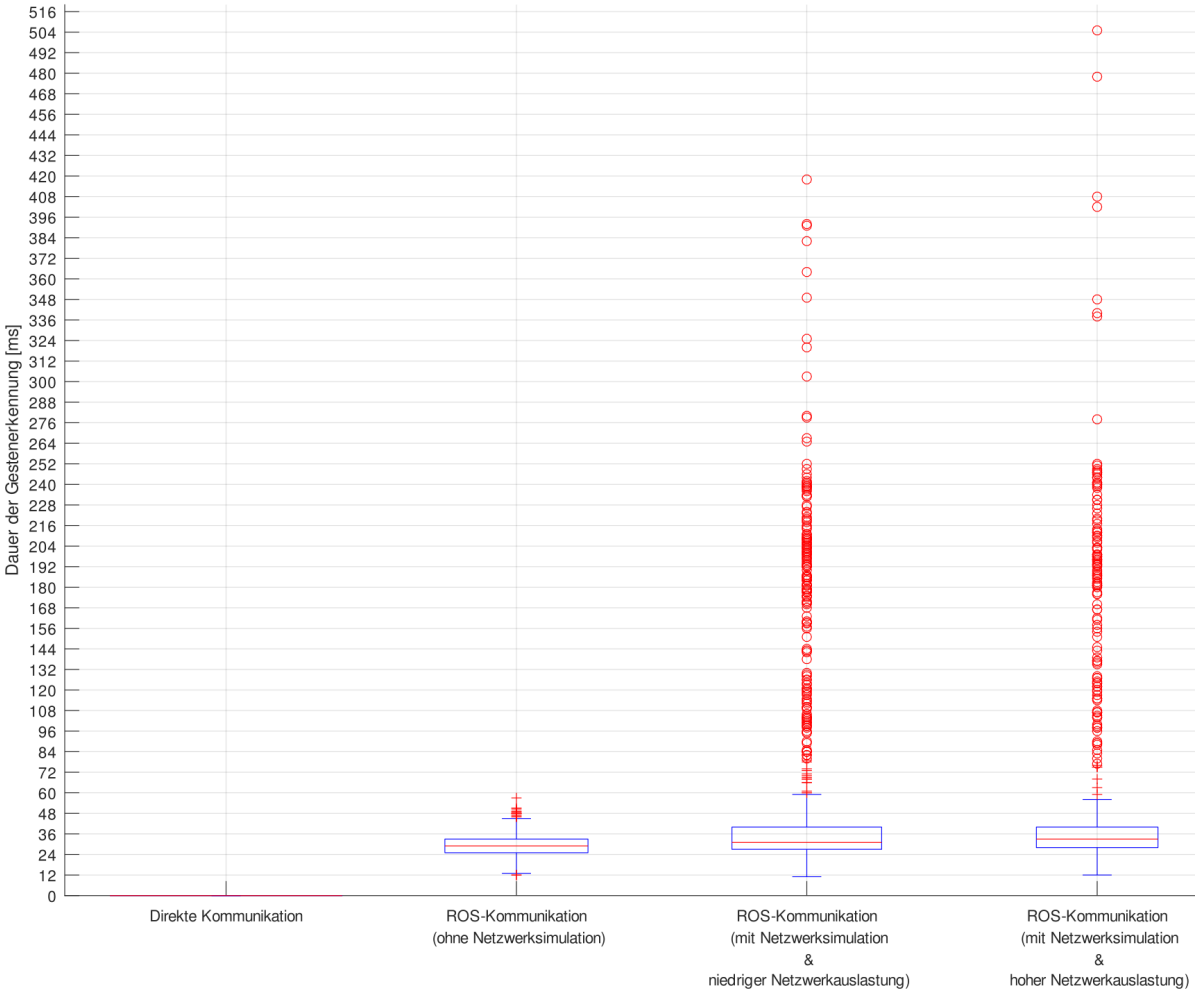

Abbildung 4.7.: Latenzverteilung der direkten Kommunikation und der ROS Kommunikation
Quelle: Eigene Ausarbeitung

Wie in Abbildung 4.7 zu sehen ist, liegen 50 % der Messwerte für die ROS-Kommunikation ohne Netzwerksimulation im Bereich zwischen 25 und 33 ms. Die geringen Ausreißer liegen bei der ROS-Kommunikation ohne Netzwerksimulation wie erwartet vermehrt im oberen Bereich der Verteilung. Bei der ROS-Kommunikation mit Netzwerksimulation und geringer Netzwerkauslastung sind bereits durch den Netzwerk-Overhead starke Ausreißer im oberen Bereich der Verteilung auszumachen. Anzumerken ist, dass es sich hierbei um eine rechtsschiefe Verteilung handelt und der Median daher kleiner als der Mittelwert ist. Die Gestenerkennungsdauer ist daher vermehrt im unteren Bereich der Verteilung angesiedelt. Im Gegensatz zur ROS-Kommunikation ohne Netzwerksimulation befinden sich bei der ROS-Kommunikation mit Netzwerksimulation und geringer Netzwerkauslastung der Großteil der Messwerte im Bereich zwischen 11 und 59 ms. Bei der ROS-Kommunikation ohne Netzwerksimulation befinden sich hingegen der Großteil der Messwerte im Bereich zwischen 13 und 46 ms. Wie auch bei der ROS-Kommunikation mit Netzwerksimulation und geringer Netzwerkauslastung sind auch bei der ROS-Kommunikation

mit Netzwerksimulation und hoher Netzwerkauslastung vermehrt Ausreißer im oberen Bereich der Verteilung festzustellen, wobei jedoch auch einzelne Messwerte über die bisherigen Werte hinausgehen. Der Median ist im Gegensatz zur ROS-Kommunikation mit Netzwerksimulation und geringer Netzwerkauslastung hierbei leicht nach oben verschoben, wodurch eine geringe Tendenz zu höheren Latenzzeiten festzustellen ist. Resümierend kann gesagt werden, dass je höher die Auslastung des Kommunikationskanals ist, auch eine geringfügig höhere Latenzzeit zustande kommt. Bei größeren zu übertragenden Datenmengen wäre das Ergebnis höchstwahrscheinlich unterschiedlicher ausgefallen. Nichtsdestotrotz konnte ermittelt werden wie ROS in kleinen und überschaubaren Anwendungen mit hoher Netzwerkauslastung zurechtkommt.

4.3. Genauigkeit der Zielpositionen

Zum Testen der Genauigkeit der Zielpositionen wurde wiederum das Azure Kinect Body Tracking SDK im CPU-Modus anstatt mit der Grafikkartenbeschleunigung gestartet. Zudem stand es dem Proband, welcher bereits sehr viel Erfahrung mit der Roboter-Gesten-Anwendung sammeln konnte, frei zwischen dem Weltkoordinatensystem- und Gelenk-Modus zu wechseln um die anzufahrenden Zielpositionen, wie in Abbildung 3.9 zu sehen sind, anzusteuern. Das Ziel war es den Mittelpunkt des TCPs so exakt wie nur möglich über dem Mittelpunkt der roten Kreuze zu platzieren. Die Zeitdauer wurde zudem erfasst um auswerten zu können ob die benötigte Zeitdauer einen Einfluss auf die Genauigkeit der anzufahrenden Zielpositionen hatte. Um die Messung der angefahrenen Zielpositionen genauer durchführen und analysieren zu können, wurde der Test im Simulator Gazebo durchgeführt. In der Simulationsumgebung wurde hierfür ein virtueller Raum erschaffen, in welchem die exakten Zielpositionen bereits genau vordefiniert wurden. Die ermittelten Positionen konnten daraufhin mit den exakten Positionen verglichen und die Differenzen ermittelt werden. Für diesen Testfall wurde die Roboter-Gesten-Anwendung und deren Abhängigkeiten mittels der Flags „MEASUREMENT“ und „POSITION_MEASUREMENT“, welche im Anhang A.1.2 aufgezeigt werden, kompiliert. Die Dateien „position_measurement.csv“, „robot_arm_positions.json“, „gazebo_state.log“, „Gazebo.mp4“ und „Durchführung der Gesten.mp4“, welche bei der Durchführung des Tests erstellt wurden, sind auf dem Begleitmedium im Verzeichnis „Testdaten/Messung der Zielpositionen/“ zu finden.

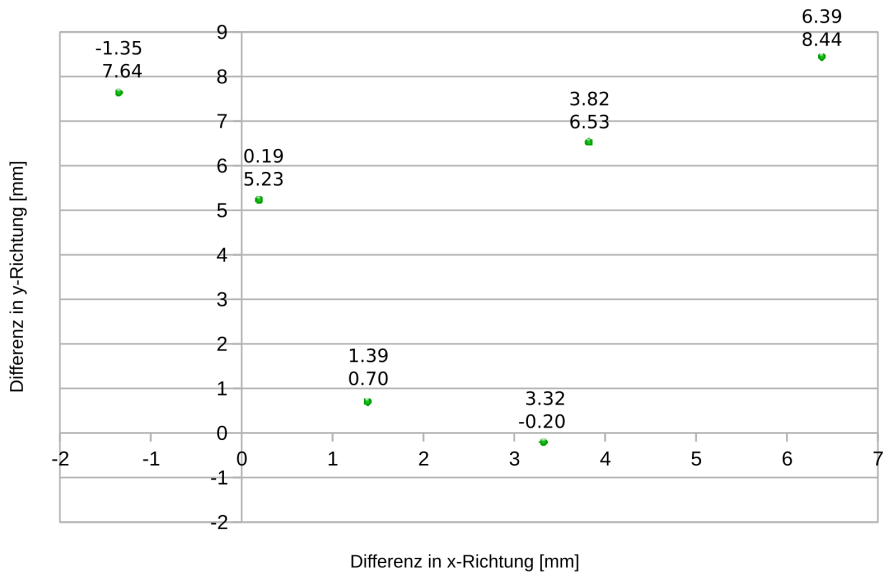

Abbildung 4.8.: Positionsdifferenzen beim Teachen mit der Roboter-Gesten-Anwendung
Quelle: Eigene Ausarbeitung

In Abbildung 4.8 sind die Differenzen in x- und y-Richtung zu den sechs nacheinander angefahrenen Zielpositionen eingetragen. Es ist zu sehen, dass maximal eine Differenz von 8,44 mm und minimal eine Differenz von 0,20 mm in eine Richtung erreicht wurden. Die Differenz in x- und y-Richtung ist ansonsten jedoch annähernd gleich und der Unterschied kann als vernachlässigbar klein angesehen werden. Die erzielten Differenzen liegen im zuvor angenommenen Bereich von wenigen Millimetern und sind daher für den „WidowX 200“-Lernroboter akzeptabel. Für einen Industrieroboter wären hierbei höhere Genauigkeiten notwendig. Da jedoch die Zielpositionen im Nachhinein angepasst werden können ist dieser Mangel jedoch unwesentlich klein.

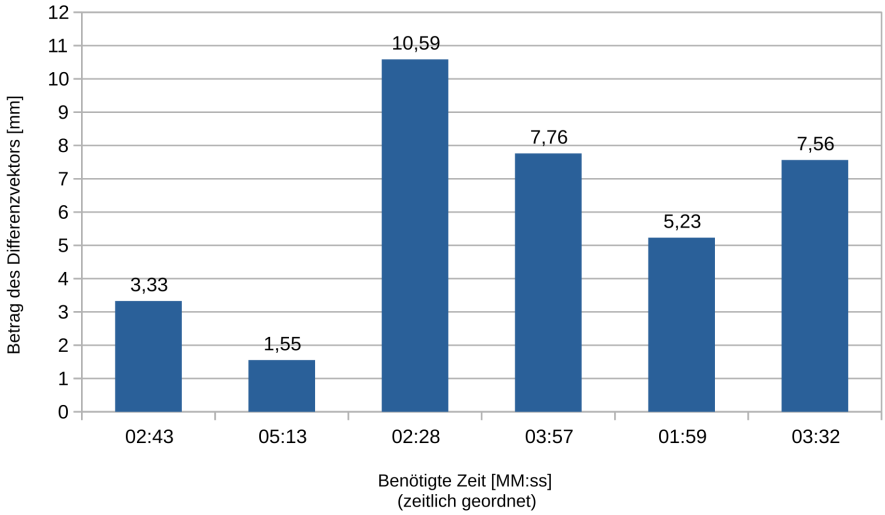

Abbildung 4.9.: Betrag der Differenzvektoren beim Teachen mit der Roboter-Gesten-Anwendung
Quelle: Eigene Ausarbeitung

Bei Betrachtung der benötigten Zeit und der dazugehörigen Genauigkeit für diesen Test, wie in Abbildung 4.9 zu sehen ist, kann gesagt werden, dass bei zunehmender Teach-Dauer des „WidowX 200“-Lernroboters auch die Genauigkeit abgenommen hat. Dabei wurden die ersten zwei Zielpositionen noch mit der geringsten Fehlerrate angefahren. Ab der dritten Zielposition war jedoch bereits die größte Differenz vorhanden. Danach hatte sich die Differenz jedoch wieder normalisiert und auf eine fast gleich bleibenden Fehlerrate eingependelt. In den meisten Fällen in denen die benötigte Zeit zum Anfahren einer Zielposition länger gewählt wurde, verringerte sich auch zumeist die Fehlerrate. Die durchschnittliche Teachdauer zum Erreichen der sechs Zielpositionen lag in diesen Testbeispielen bei ca. 3 Minuten. Diese Zeitdauer zum Teachen einer Zielposition bei der vorhandenen Distanz zwischen den Zielpositionen kann als akzeptabel angesehen werden. Abschließend kann gesagt werden, dass die Fehlerrate bei den sechs anzufahrenden Zielpositionen im tolerierbaren Wertebereich verblieben ist und sich bei längerer Benutzung der Roboter-Gesten-Anwendung annähernd gleichbleibend verhält.

4.4. Genauigkeit der Gestenerkennung

Bei Analyse der Aufnahme „Durchführung der Gesten.mp4“, welches auf dem Begleitmedium im Verzeichnis „Testdaten/Messung der Zielpositionen/“ zu finden ist, stellte sich heraus, dass der Proband mehrmals reflexartige Bewegungen durchgeführt und die Brille zudem am Brillensteg hinaufgeschoben hatte. Reflexartige Bewegungen gehen zumeist unterbewusst vonstatten, wie auch das hinaufschieben der Brille, welches zumeist aus Bequenlichkeitsgründen unterbewusst durchgeführt wird [Mat]. Es kann daher aufgrund der Aufnahme gesagt werden, dass die Roboter-Gesten-Anwendung gegen die genannten reflexartigen und unterbewussten Bewegungen gesichert ist.

5. Zusammenfassung & Ausblick

Nach der Analyse und Auswertung der Ergebnisse können nun die in dieser Arbeit durchgeführten Arbeitsschritte resümiert und daraufhin mit einem Fazit abgerundet werden. Zu guter Letzt soll zudem auch ein Ausblick über weitere Forschungsmöglichkeiten hinsichtlich der Gestensteuerung eines Industrieroboters mittels Tiefenkameras gegeben werden.

5.1. Zusammenfassung

Es konnte aufgezeigt werden, dass eine Gestensteuerung als Alternative zu einem herkömmlichen Teach Pendant auf Grundlage von ROS mit der Zuhilfenahme eines „WidowX 200“-Lernroboters und einer Azure Kinect implementiert werden kann. Die entwickelten Gesten, welche einerseits unbeabsichtigte Bewegungen nicht zulassen und andererseits zudem Ermüdungserscheinungen entgegen wirken sollen, haben sich während der durchgeführten Tests bewährt. Durch die Roboter-Gesten-Anwendung wurde es einem erfahrenen Probanden hierdurch ermöglicht ein praxisnahes Teachbeispiel durchzuführen und so einem „WidowX 200“-Lernroboter in angemessener Zeit Zielpositionen beizubringen. Zudem ist es möglich die Zielpositionen in einer Datei zu speichern und daraufhin vom „WidowX 200“-Lernroboter erneut autonom abspielen zu lassen. Das erstellte Roboter-Gesten-Framework ist modular aufgebaut und kann bei Implementierung der spezifischen Schnittstellen neben dem „WidowX 200“-Lernroboter daher auch mit anderen Industrierobotern und neben der Azure Kinect auch mit anderen Tiefenkameras betrieben werden.

Die Zeit von unter 50 ms, welche für ein Teach Pendant empfohlen wird [Pra+04, S. 55], kann vom Azure Kinect Body Tracking SDK mit den im Durchschnitt zwischen 314,02 und 338,55 ms erzielten Latenzen nicht eingehalten werden. Aus diesem Grund muss die Berechnung des neuronalen Netzwerks der Gestenerkennung noch schneller oder auf speziell darauf ausgelegter Hardware, wie z.B. FPGAs, GPUs oder KI-Beschleuniger, durchgeführt werden [Wel].

Zudem liegt die Latenzzeit der ROS-Kommunikation in einem praxisnahen Gigabit-Ethernet-Netzwerk weit über den maximalen 50 ms, welche für ein Teach Pendant empfohlen werden [Pra+04, S. 55]. Selbst bei Betrachtung der maximalen Durchschnittlichen Latenzzeit von 116 ms liegt diese noch zu weit über den 100 ms um einer bedienenden Person hierbei das Gefühl einer inputlag freien Erfahrung zu vermitteln [Mil68]. Aufgrund der extremen Schwankungen der Latenz der ROS-Kommunikation über ein Gigabit-Ethernet-Netzwerk wäre es außerdem empfehlenswert zeitkritische Steuerungsbefehle mehrmals senden zu können. Dies könnte sicherstellen, dass die Steuerungsbefehle auch zeitgerecht ankommen. Hierzu müsste jedoch ein Algorithmus implementiert werden um identische Steuerungsbefehle erkennen und ein mehrmaliges Ausführen der Steuerungsbefehle verhindern zu können. Bei der lokalen Interprozesskommunikation über das Loopback-Interface konnte sich die ROS-Kommunikation jedoch beweisen, da die Latenzzeit von Durchschnittlich zwischen 23 und 36 ms in den meisten echtzeitnahen

Anwendungsgebieten ausreichend gering sein sollte. Anzumerken ist, dass die lokale Interprozesskommunikation über das Loopback-Interface dennoch immer noch ein Trade-off zwischen Modularität und Performanz darstellt. Wenn jedoch die Modularität im Vordergrund steht, kann die ROS-Kommunikation über das Loopback-Interface eine Möglichkeit darstellen um die verschiedenen Komponenten des Systems austauschbar zu halten. Das Problem besteht jedoch immer noch in der Tatsache, dass sich die Zeiten für die Gestenerkennung und der Latenzzeit zwischen der Kommunikation mit der InterbotiX-Schnittstelle summieren und dadurch weit über den erwähnten Latenzzeiten liegen. ROS ist daher keine Allzwecklösung für alle Problemstellungen, sondern nur eine mögliche Implementierung die sich erst noch über die Jahre im industriellen Einsatz behaupten muss [Why].

5.2. Ausblick

Der CPU-Modus des Azure Kinect Body Tracking SDKs ist mit einer gemessenen Latenz von Durchschnittlich zwischen 314,02 und 338,55 ms mit minimalen Inputlags für den „WidowX 200“-Lernroboter nutzbar, für den industriellen Einsatz jedoch nicht geeignet. Die Eignung muss neu bewertet werden, wenn das Azure Kinect Body Tracking SDK neben der CUDA basierten Grafikkartenbeschleunigung, welche nur auf Nvidia basierten Grafikkarten verfügbar ist [Enc], eine herstellerunabhängige Implementierung erhält. Als plattformunabhängige und alternative Programmierschnittstelle zu CUDA wäre hier z.B. OpenCL oder Vulkan anzumerken [Vul]. Dies würde vor allem die Performanz der Gestenerkennung erhöhen, die freie Wahl der Rechnerkomponenten ermöglichen und zudem den Wettbewerb zwischen den Herstellern fördern.

Im Vergleich zur Kinect 2, welche eine 0,3-Megapixel-Tiefenkamera verbaut hat, ist die Azure Kinect, welche eine 1-Megapixel-Tiefenkamera besitzt, bereits eine deutliche Weiterentwicklung zu ihrem Vorgängermodell. Nichtsdestotrotz ist hier jedoch durch den großen Sprung von einer 0,3- auf eine 1-Megapixel-Tiefenkameraauflösung für die Zukunft noch genügend Verbesserungspotenziall vorhanden um noch höhere Auflösungen erreichen zu können [Bam+18]. Mit einer höheren Auflösung der Tiefenkamera wäre es möglich die Gesten noch genauer zu erkennen und die False-Positive-Rate noch weiter zu verringern. Mittels Eye-Tracking, welches z.B. in der HoloLens 2 verbaut ist [Hol], könnte die Sicherheit der bedienenden Person zusätzlich erhöht und die Gestenerkennung noch zuverlässiger gestaltet werden. Hiermit könnte unter anderem erkannt werden ob die bedienende Person Ermüdungserscheinungen aufweist und daher besser eine Pause einlegen sollte oder sogar erkennen ob die bedienende Person nicht mehr bei Bewusstsein ist, weil die bedienende Person z.B. ungewollt auf dem Boden liegt. Es wäre zudem denkbar, dass die Umgebung des Industrieroboters durch weitere Tiefenkameras abgescannt werden könnte um eine Kollisionserkennung für den Industrieroboter zu realisieren. Hierdurch könnte der Industrieroboter auf Hindernisse vorzeitig reagieren und so Personen sowie auch andere Objekte frühzeitig erkennen. Dadurch könnte die Sicherheit des Systems höchstwahrscheinlich zusätzlich verbessert und die Gefahr von kritischen Situationen nochmals verringert werden. Durch die Verwendung von mehreren im Raum verteilten Tiefenkameras könnten zudem tote Winkel bei der Bedienung der Roboter-Gesten-Anwendung verringert und die Bedienung der Roboter-Gesten-Anwendung aus verschiedenen Blickrichtungen fehlerfreier ermöglicht werden.

Weiteres Forschungspotenzial besteht auch im Bereich der Sicherheit und Ergonomie der in dieser Arbeit entwickelten Roboter-Gesten-Anwendung. Hierbei könnte erforscht werden wie gut sich z.B. smarte Fußeinlagesohlen zur Sicherheits- und Ergonomiesteigerung eignen würden. Hierdurch würden die Hände zur Gestensteuerung frei bleiben und es könnte womöglich eine Art von „Dead Man's Switch“ mit einer smarten Fußeinlagesohle realisiert werden. Hierdurch wäre es vermutlich machbar sehr schnelle und reflexartige Druckverteilungen und Bewegungen zu Erkennen und den Industrieroboter bei dieser Art von Bewegung zu stoppen. Zu wenig oder kein Druck auf den Fußeinlagesohlen könnten mehrere Gründe haben, da z.B. die Sohlen abgezogen wurden oder die bedienende Person ungewollt auf dem Boden liegt. Das Starten der Gestenerkennung könnte womöglich durch eine korrekt ausgeführte Kombination aus Fußbewegungen realisiert werden [Tan+15]. Eine andere Möglichkeit würde darin bestehen einen smarten Handschuh mit einem Beschleunigungssensor und einem Gyroskop zu verwenden, welcher zu schnelle Bewegungen erkennen könnte, und die Bewegung des Industrieroboters bei zu auffälligen Bewegungen anhält [Ghi+19]. Zur Realisierung der Erkennung von auffälligen Bewegungen könnte zudem ein künstliches neuronales Netzwerk auf diese auffälligen Bewegungen trainiert werden.

A. Anhang

A.1. Einrichtung des Testsystems

Zur Einrichtung des Testsystems wird zuallererst ein Linux-Container erstellt, welcher ein einfaches und schnelles Deployen auf unterschiedlichen Linux-Distributionen ermöglicht. Zum Hosten des Linux-Containers wird in dieser Installationsanleitung Ubuntu 20.04 Focal Fossa verwendet, welches eine LTS-Version von Ubuntu darstellt. Nach der Einrichten des Linux-Containers können die benötigten Pakete in dieser neu erstellten Umgebung installiert und daraufhin entsprechend konfiguriert werden. Der vorkonfigurierte Linux-Container ist als Datei „ros-tir.tar.gz“ auch auf dem Begleitmedium im Verzeichnis „Linux-Container/“ vorzufinden. Dabei ist jedoch zu beachten, dass auf dem Host-System dennoch ein paar wenige Konfigurationsschritte für das Host-System durchgeführt werden müssen. Die Schritte für den Linux-Container können hingegen mit dem vorkonfigurierten Linux-Container übersprungen werden.

A.1.1. Erstellen des Linux-Containers

Für die Installation der Linux-Container-Tools werden Administratorrechte auf dem Zielsystem benötigt. Die benötigten Installationsschritte zum Einrichten des Linux-Containers sind wie folgt [Lxd].

1. LXD 4.0 installieren:

   ```
   sudo snap install lxd
   ```

   bzw.

   ```
   sudo apt install lxd
   ```

2. LXD mit den Standardeinstellungen initialisieren:

   ```
   lxd init
   ```

3. Den derzeitigen User zur LXD-Gruppe hinzufügen:

   ```
   sudo adduser $(whoami) lxd
   ```

   Logout & Login durchführen oder mittels des folgenden Befehls den User im Terminal neu anmelden:

   ```
   su - $(whoami)
   ```

4. Erstellen des Linux-Containers:
   Der Linux-Container wird mit dem folgenden Befehl im Default-Linux-Container-Pool erstellt. Der Platzhalter „<container name>“ muss durch einen entsprechenden Namen für den Linux-Container ersetzt werden.

```
CONTAINER_NAME="<container name>"
lxc init ubuntu:18.04 $CONTAINER_NAME
```

Mit der Option „-s“ kann ein anderer Linux-Container-Pool ausgewählt werden.

```
lxc init ubuntu:18.04 $CONTAINER_NAME -s <container pool name>
```

5. Die Eigenschaften des Image des neu erstellten Linux-Containers können mit dem folgenden Befehl eingesehen werden:

```
lxc storage list
```

6. Dem Linux-Container erlauben den X11-Traffic weiterzuleiten:

   6.1. Die Datei „gui-profile.yaml“ mit dem folgenden Inhalt erstellen:

```
config:
  environment.DISPLAY: :0
  raw.idmap: both 1000 1000
  user.user-data: |
    #cloud-config
    package_upgrade: true
    packages:
      - x11-apps
      - mesa-utils
      - pulseaudio
    runcmd:
      - 'sed -i "s/; enable-shm = yes/enable-shm = no/g" /etc/pulse/
   client.conf'
      - 'echo export PULSE_SERVER=unix:/tmp/.pulse-native | tee --
   append /home/ubuntu/.profile'
description: GUI LXD profile
devices:
  PASocket:
    path: /tmp/.pulse-native
    source: /run/user/1000/pulse/native
    type: disk
  X0:
    path: /tmp/.X11-unix/X0
    source: /tmp/.X11-unix/X0
    type: disk
  mygpu:
    type: gpu
name: gui
used_by:
```

   6.2. Bei der Erstellung der Datei „gui-profile.yaml“ ist zu beachten, dass die IDs im Eintrag „raw.idmap: both 1000 1000“ folgende Bedeutung haben:

```
raw.idmap: both <host user id> <user id in container>
```

<host user id>:	Die User-ID, welche zum Starten des Linux-Containers verwendet wird.
<user id in container>:	Die User-ID des Users innerhalb des Linux-Containers. Normalerweise ist dies die User-ID 1000, da der erste User, welcher im Linux-Container eingerichtet wird, standardmäßig die User-ID 1000 erhält.

6.3. Das Profil kann daraufhin über die Datei „gui-profile.yaml“ geladen werden:

```
lxc profile create gui
cat gui-profile.yaml | lxc profile edit gui
```

6.4. Abschließend muss das Profil dem neu erstellten Linux-Container zugewiesen werden:

```
lxc profile add $CONTAINER_NAME gui
```

7. Den Linux-Container starten und eine Shell im Linux-Container öffnen:

```
lxc start $CONTAINER_NAME
lxc exec $CONTAINER_NAME -- sudo --user ubuntu --login
```

8. Um Rendering-Probleme zu vermeiden sollten die neuesten Grafikkartentreiber im Linux-Container installiert werden:

```
sudo add-apt-repository ppa:oibaf/graphics-drivers -yn
sudo apt upgrade -y
```

9. Um die durchgeleitete Grafikkarte zu testen muss der Linux-Container neu gestartet werden! Anschließend müssen die folgenden Befehle eingegeben werden:

```
lspci | grep VGA
```

bzw.

```
sudo apt-get install glmark2
glmark2
```

Wichtig:

Anstatt „llvmpipe“ sollte die jeweilige verbaute Grafikkarte angezeigt werden.

10. Für die Soundausgabe sollte PulseAudio getestet werden. Es sollte beim Ausführen des folgenden Befehls ein Rauschen zu hören sein:

```
pacat < /dev/urandom
```

11. Von Zeit zu Zeit sollte auf dem Host-System ein Backup vom Linux-Container erstellt werden:

```
lxc export $CONTAINER_NAME ${CONTAINER_NAME}.tar.gz
```

bzw. zum Erstellen eines Snapshots

```
lxc snapshot $CONTAINER_NAME snap01
lxc publish ${CONTAINER_NAME}/snap01 --alias <new-alias>
lxc image export <new-alias> ${CONTAINER_NAME}.tar.gz
```

12. Das Backup kann auf dem Host-System mittels des folgenden Befehls zurückgespielt werden:

    Wichtig:

    Beim Import des Linux-Containers muss sichergestellt werden, dass zuvor alle mit dem Linux-Container verknüpften Linux-Container-Profile vorhanden sind, da es ansonsten zu Import-Fehlern kommt.

    ```
    lxc import <new-alias> ${CONTAINER_NAME}.tar.gz
    ```

    bzw. der Snapshot

    ```
    lxc image import ${CONTAINER_NAME}.tar.gz --alias <new-alias>
    ```

A.1.2. Installation & Konfiguration der Pakete

Mit der zuvor installierten „Ubuntu 20.04 Focal Fossa“-Umgebung können nun die für diese Arbeit benötigten Pakete innerhalb dieser Umgebung installiert und konfiguriert werden.

1. Installation von ROS 1:
   Zur Nutzung von ROS 1 muss ROS Melodic Morenia installiert werden. Die Installationsanleitung kann man hierbei der offiziellen ROS-Webseite (http://wiki.ros.org/melodic/Installation/Ubuntu) entnehmen. Bei der Installation des ROS-Pakets muss das „ros-melodic-desktop-full“-Paket mit dem folgenden Befehl installiert werden.

   ```
   sudo apt install ros-melodic-desktop-full
   ```

2. Installation des WidowX 200:

   2.1. Hierzu müssen zuvor die benötigten Abhängigkeiten installiert werden:

   ```
   sudo apt install ros-melodic-robot-state-publisher ros-\
   melodic-rqt-plot ros-melodic-joy
   ```

   2.2. Anschließend müssen die auf der Webseite https://github.com/Interbotix/interbotix_ros_arms#quickstart beschriebenen Schritte 1 bis 6 durchgeführt werden.

Wichtig:

Anstelle des „https://github.com/Interbotix/interbotix_ros_arms.git“-Repositories muss, dass für diese Arbeit modifizierte Repository, welches auf dem Begleitmedium im Verzeichnis „Repositories/interbotix_ros_arms“ beigelegt ist, verwendet werden. Zum Klonen des modifizierten Repositories kann man hierzu den folgenden Befehl verwenden:

```
git clone <Pfad zum Begleitmedium>/Repositories/interbotix_ros_arms/
```

2.3. Danach müssen die folgenden Einstellungen auf dem Host-System durchgeführt werden, damit der unprivilegierte Linux-Container Zugriff auf den Dynamixel U2D2 erhält:

```
wget https://raw.githubusercontent.com/Interbotix/interbotix_ros_arms/\
master/interbotix_sdk/10-interbotix-udev.rules
sudo cp 10-interbotix-udev.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules && sudo udevadm trigger

lxc config device add $CONTAINER_NAME dynamixel_u2d2_ttydxl unix-char path\
=/dev/ttyDXL required=false
```

2.4. Innerhalb des Linux-Containers muss zudem der folgende Befehl ausgeführt werden, um als normaler User auf den Dynamixel U2D2 zugreifen zu können:

```
echo "# Interbotix WidowX 200
sudo chown $(whoami) /dev/ttyDXL" >> ~/.bashrc
source ~/.bashrc
```

3. Testen des WidowX 200:

Wichtig:

- Genügend Abstand einhalten
- Den Roboter festschrauben
- Beim Beenden der Softwaretools den Roboter festhalten, da dieser ansonsten in sich zusammenfällt.

Zum Testen der grundlegenden Funktionalität des WidowX 200 können ein paar spezifische ROS-Befehle an diesen gesendet werden.

```
roslaunch interbotix_descriptions description.launch robot_name:=wx200 jnt_pub\
_gui:=true

roslaunch interbotix_sdk arm_run.launch robot_name:=wx200

rosservice call /wx200/torque_joints_on

roslaunch interbotix_gazebo gazebo.launch robot_name:=wx200
rosservice call /gazebo/unpause_physics
```

4. Azure Kinect installieren:

   4.1. Außerhalb des Linux-Containers müssen die folgenden Einstellungen erfolgen um die Azure Kinect für den derzeitgen User verfügbar zu machen [Micb]:

```
wget https://raw.githubusercontent.com/microsoft/Azure-Kinect-Sensor-SDK/\
develop/scripts/99-k4a.rules
sudo cp 99-k4a.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules && sudo udevadm trigger
```

   4.2. Außerhalb des Linux-Containers müssen die folgenden Einstellungen erfolgen, damit der unprivilegierte Linux-Container Zugriff auf die Azure Kinect erhält:

```
lxc config device add $CONTAINER_NAME microsoft_generic_superspeed_usb\
_hub usb vendorid=045e productid=097a
lxc config device add $CONTAINER_NAME microsoft_generic_usb\
_hub usb vendorid=045e productid=097b
lxc config device add $CONTAINER_NAME azure_kinect_depth\
_camera usb vendorid=045e productid=097c
lxc config device add $CONTAINER_NAME azure_kinect_4k\
_camera usb vendorid=045e productid=097d
lxc config device add $CONTAINER_NAME azure_kinect_microphone_\
array usb vendorid=045e productid=097e
```

   4.3. Den Linux-Container neustarten:

```
lxc restart $CONTAINER_NAME
```

5. Azure Kinect SDKs installieren:

   5.1. Microsoft-Paketquelle hinzufügen [Micb]:

```
sudo apt install curl
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
sudo apt-add-repository https://packages.microsoft.com/ubuntu/18.04/prod
```

   5.2. Installieren des Azure Kinect SDKs:

```
sudo apt install k4a-tools=1.3.*
sudo apt install libk4a1.3-dev
```

> **Wichtig:**
>
> Zur installierten Version des „k4a-tools“-Pakets muss zudem die jeweilige Development-Version des „libk4a“-Pakets installiert werden:
>
> ```
> sudo apt install libk4a<major>.<minor>-dev
> ```
>
> Der Befehl kann z.B. wie folgt aussehen:
>
> ```
> sudo apt install libk4a1.3-dev
> ```

   5.3. Um Paketkollisionen beim Updaten zu verhindern sollte das Paket „k4a-tools“ vom Updateprozess ausgeschlossen werden:

```
sudo apt-mark hold k4a-tools
sudo apt-mark showhold
```

5.4. Nun kann das Paket „libk4abt1.0-dev“ ohne Paketkollisionen installiert werden:

```
sudo apt install libk4abt1.0-dev
```

6. Azure Kinect Firmware aktualisieren:

6.1. Überprüfen ob die Azure Kinect erkannt wird:

```
AzureKinectFirmwareTool -l
```

6.2. Die Ausgabe sollte in etwa wie folgt aussehen:

```
 == Azure Kinect DK Firmware Tool ==
Found 1 connected devices:
1: Device "000846393812"
```

6.3. Die jeweilige Version der Azure Kinect Firmware von der Webseite https://github.com/microsoft/Azure-Kinect-Sensor-SDK/blob/develop/docs/usage.md#msis herunterladen:

```
wget https://download.microsoft.com/download/1/9/8/198048e8-63f2-45c6-\
8f96-1fd541d1b4bc/AzureKinectDK_Fw_1.6.102075014.bin
```

6.4. Die heruntergeladene Azure Kinect Firmware daraufhin flashen:

```
AzureKinectFirmwareTool -u AzureKinectDK_Fw_1.6.102075014.bin
```

7. Azure Kinect testen:

7.1. Device-Stream testen:

```
k4aviewer
```

7.2. Body Tracking testen:
Die OpenGL-Version, der Grafikkartenname und der eingesetzte Grafikkartentreiber kann mit den folgenden Befehlen abgefragt werden:

```
sudo apt install mesa-utils
glxinfo | grep -E "Device:|OpenGL version"
```

Die Ausgabe kann z.B. wie folgt aussehen:

```
   Device: AMD Radeon RX 5700 XT (NAVI10, DRM 3.35.0, 5.4.0-39-generic,\
 LLVM 10.0.0) (0x731f)
OpenGL version string: 4.6 (Compatibility Profile) Mesa 20.2.0-devel (git\
-c0c03f4 2020-06-27 bionic-oibaf-ppa)
```

Wichtig:

Für das Body Tracking wird ein Grafikkartentreiber mit mindestens „OpenGL 4.4“- und „OpenGL Compatibility Context“-Unterstützung benötigt. Bei den Open-Source-Mesa-Treibern tritt laut Stand 29. März 2020 mit dem „k4a-tools“-Paket in der Version 1.3 oder niedriger der Fehler

„Shader Error: 0:2(38): error: ‘gl_FragColor’ undeclared
0:2(38): error: value of type vec4 cannot be assigned to variable of type error“

auf, welcher das Starten der „k4abt_simple_3d_viewer“-Anwendung verhindert.

Hierzu muss die „simple_3d_viewer“-Anwendung mit dem in dieser Arbeit erstellten GLSL-Patch[a], welcher bereits in den Hauptzweig des „Azure-Kinect-Samples“-Repositories eingepflegt wurde, neukompiliert werden. Getestet wurde das Kompilieren mit der Commit-Version „ac696e4015648dd82dd019e59d94ba169f3c81aa“. Wenn ein Fehler beim Kompilieren auftritt, kann nach dem Klonen des Repositories explizit mit dem folgenden Befehl auf den in dieser Arbeit verwendeten Stand des Repositories zugegriffen werden.

```
git checkout ac696e4015648dd82dd019e59d94ba169f3c81aa
```

Zum Kompilieren der „simple_3d_viewer“-Anwendung müssen die folgenden Schritte eingehalten werden.

1. Das „Azure-Kinect-Samples“-Repository klonen:

   ```
   git clone --recursive https://github.com/microsoft/Azure\
   -Kinect-Samples.git
   cd Azure-Kinect-Samples/
   ```

2. Um einen Kompilierfehler[b] zu vermeiden muss in der Datei „CMakeLists.txt“ zwischen „FIND_PACKAGE(k4a REQUIRED)“ und „FIND_PACKAGE(k4abt REQUIRED)“ der Text „FIND_PACKAGE(k4arecord REQUIRED)“ eingefügt werden. Um einen weiteren Kompilierfehler[b] zu vermeiden muss zudem in der Datei „CMakeLists.txt“ im Verzeichnis „body-tracking-samples/simple_3d_viewer/“ zwischen „k4abt“ und „window_controller_3d::window_controller_3d“ der Text „k4arecord“ eingefügt werden.

[a] https://github.com/microsoft/Azure-Kinect-Samples/commit/15c303fb9a3a6b3a2c54a57cb11996823389595a
[b] https://github.com/microsoft/Azure-Kinect-Samples/pull/54

3. Notwendige Pakete zum Kompilieren der „simple_3d_viewer“-Anwendung installieren:

```
sudo apt install libxi-dev cmake ninja-build
sudo apt install libxrandr-dev libxinerama-dev libxcursor\
-dev libgl-dev # for GLFW
sudo apt install mesa-vulkan-drivers # for Vulkan support in
                                     # GLFW (optional)
```

4. Die „simple_3d_viewer“-Anwendung komplieren und ausführbar machen:

```
mkdir build
cd build
cmake .. -GNinja
ninja

chmod +x ./bin/simple_3d_viewer
```

Wenn eine Nvidia-GPU verbaut ist, kann die GPU-Beschleunigung eingesetzt werden. Hierzu muss das CUDA-Paket installiert werden:

```
k4abt_simple_3d_viewer
```

Daraufhin kann die „simple_3d_viewer“-Anwendung mit GPU-Beschleunigung gestartet werden:

```
k4abt_simple_3d_viewer
```

bzw.

```
~/Azure-Kinect-Samples/build/bin/simple_3d_viewer
```

Wenn keine Nvidia-GPU verbaut ist muss ansonsten der CPU-Modus verwendet werden:

```
k4abt_simple_3d_viewer CPU
```

bzw.

```
~/Azure-Kinect-Samples/build/bin/simple_3d_viewer CPU
```

8. Zum Kompilieren der Roboter-Gesten-Anwendung müssen die folgenden Pakete installiert werden:

```
sudo apt install cmake gdb gcc python python-catkin-tools
```

9. Source Code kompilieren:

Wichtig:

Die Roboter-Gesten-Anwendung wird mit der Option „Release“ kompiliert. Dies ist empfehlenswert, wenn die Roboter-Gesten-Anwendung im Produktivbetrieb eingesetzt werden soll. Die Option „Release“ kann durch die Option „Debug“ ersetzt werden um eine Entwicklerversion mit zusätzlicher Ausgabe zu erhalten. Um eine Version zur Durchführung der in dieser Arbeit erläuterten Messungen zu erhalten wird die Option „Release“ und zusätzlich eine der folgenden Optionen für den jeweiligen Testfall verwendet:

```
--cmake-args -DCMAKE_CXX_FLAGS="-DMEASUREMENT -DGESTURE_MEASUREMENT"
--cmake-args -DCMAKE_CXX_FLAGS="-DMEASUREMENT -DCOMMUNICATION_MEASUREMENT"
--cmake-args -DCMAKE_CXX_FLAGS="-DMEASUREMENT -DPOSITION_MEASUREMENT"
```

9.1. InterbotiX SDK kompilieren:

```
cd ~/interbotix_ws
rm -rf build devel/
catkin build -DCMAKE_BUILD_TYPE=Release
source ~/.bashrc
```

9.2. Den Source Code der Roboter-Gesten-Anwendung auf dem Begleitmedium aus dem Verzeichnis „Repositories/Teach-Industrial-Robots“ klonen, Abhängigkeiten installieren und anschließend die Roboter-Gesten-Anwendung kompilieren:

```
mkdir -p ~/tir_ws/src
cd ~/tir_ws/
catkin build -DCMAKE_BUILD_TYPE=Release
echo "# ROS
source ~/tir_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc

cd ~/tir_ws/src
git clone <Pfad zum Begleitmedium>/Repositories/Teach-Industrial-Robots/

sudo apt install libglm-dev libeigen3-dev rapidjson-dev libk4a1.3\
-dev libk4abt1.0-dev

cd ~/tir_ws
catkin build -DCMAKE_BUILD_TYPE=Release
source ~/.bashrc
```

10. Entwicklungsumgebung installieren (optional):

10.1. Temporärer Ordner erstellen:

```
echo "# Temporary user folder
sudo mkdir -p /run/user/$(id --user)
sudo chown $(whoami) /run/user/$(id --user)" >> ~/.bashrc
source ~/.bashrc
```

10.2. VS Code installieren:

```
sudo snap install code --classic
```

10.3. Starten der Entwicklungsumgebung mit dem folgenden Befehl:

```
code
```

10.4. Mittels Strg + P in VS Code das „Quick Open“-Menü öffnen und jeweils eine der folgenden Befehle eingeben um die notwendigen Plugins zu installieren:

```
ext install ms-vscode.cpptools
ext install ms-vscode.cmake-tools
ext install ms-python.python
ext install ms-iot.vscode-ros
```

11. Wenn der Speicherplatz des Linux-Containers zu knapp wird, kann der Speicherplatz des Linux-Containers wie folgt auf dem Host-System vergrößert werden:

11.1. Image des Linux-Containers z.B. um 5 GB vergrößern:

```
sudo truncate -s +5G /var/snap/lxd/common/lxd/disks/default.img
```

11.2. Host-System neustarten

11.3. Anschließend das Image des Linux-Containers im zuvor erstellten Ordnerpfad „/<mount point>“ einhängen und daraufhin die neue Größe des Images eingeben:

```
sudo mount /var/snap/lxd/common/lxd/disks/default.img /<mount point>
sudo btrfs filesystem resize maxG /<mount point>
```

A.1.3. Starten der Roboter-Gesten-Anwendung

Für die Roboter-Gesten-Anwendung wird der „SCHED_DEADLINE“-Scheduler eingesetzt, welcher für diese Arbeit feste Echtzeitfähigkeit im 50 ms Bereich garantieren kann.

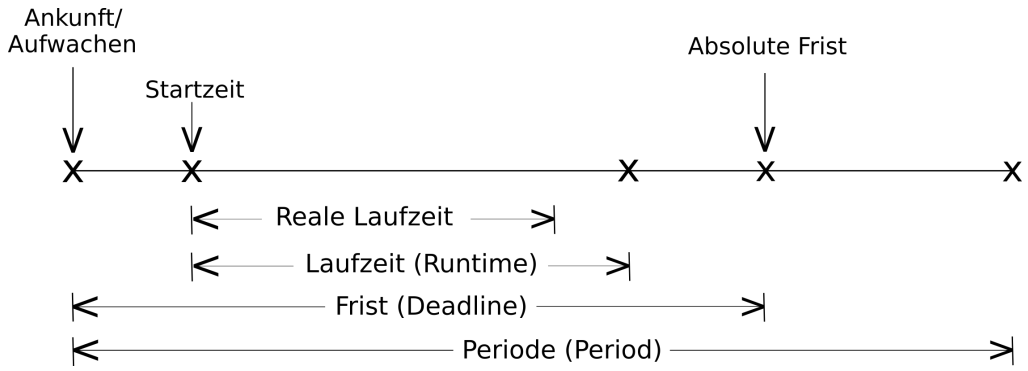

Abbildung A.1.: Ablauf von SCHED_DEADLINE
Quelle: Eigene Darstellung, in Anlehnung an [Man]

Für den „SCHED_DEADLINE“-Scheduler müssen hierfür die Parameter, welche in Abbildung A.1 ersichtlich sind, ordnungsgemäß eingestellt werden. Der „Runtime“-Parameter beträgt zur Einhaltung der 50 ms Frist hierbei 50000000 ns, der „Deadline“-Parameter beträgt 50000000

ns und der „Periode“-Parameter beträgt 50000000 ns. Der „Runtime“-Parameter steht hierbei für die maximal benötigte Ausführungszeit der Aufgabe, der „Deadline“-Parameter für die Zeit die die Aufgabe mit überzogener Ausführungszeit maximal benötigen darf und der „Periode“-Parameter für wann die nächste Periode gestartet werden muss [Man]. Diese Einstellungen wurden im Listing A.1 bei den Bash-Variablen „SCHED_RUNTIME“, „SCHED_DEADLINE“ und „SCHED_PERIOD“ im Start-Script bereits voreingestellt.

Listing A.1: Einstellungen des Start-Scripts der Roboter-Gesten-Anwendung

```
USE_ROS_COMMUNICATION=true
USE_SIMULATION=true # Only usable with "USE_ROS_COMMUNICATION=true"
DO_MEASUREMENT=false # Only usable with "USE_SIMULATION=false" and
                     # "USE_ROS_COMMUNICATION=true"
SIMULATE_ADDITIONAL_ROS_NODES=false # Only usable with "DO_MEASUREMENT=true"
ADDITIONAL_ROS_NODES_COUNT=3 # Only usable with "SIMULATE_OTHER_ROS_NODES=true"
LXC_USER_NAME="ubuntu"
LXC_INSTANCE="<container name>"
ADDITIONAL_APP_PARAMETERS="--move-home-at-exit --move-home-at-error"
TEACH_POSITIONS=true # If "true" positions can be teached. Otherwise the positions
                     # in the file "robot_arm_positions.json" will be executed.
CONFIGURATION_FILE_PATH="" # Empty for the default file path
                           # ("~/robot_arm_positions.json")
SCHED_RUNTIME=50000000
SCHED_DEADLINE=50000000
SCHED_PERIOD=50000000
```

Die Roboter-Gesten-Anwendung kann mit dem Start-Script „start_tir.sh“, welches sich auf dem Begleitmedium im Verzeichnis „Repositories/Teach-Industrial-Robots“ befindet, durch das Host-System im Linux-Container gestartet und dem „SCHED_DEADLINE“-Scheduler zugewiesen werden. Die einstellbaren Optionen des Start-Scripts sind im Listing A.1 ersichtlich. Zum Ausführen des Start-Scripts muss im Listing A.1 der Bash-Variable „LXC_INSTANCE“ der Name des Linux-Containers zugewiesen werden. Die Bash-Variable „USE_ROS_COMMUNICATION“ steuert ob die Roboter-Gesten-Anwendung über ROS kommunizieren oder eine direkt Verbindung mit dem Industrieroboter aufbauen soll. Um der Roboter-Gesten-Anwendung mitteilen zu können ob diese im Lern- oder Abspielmodus gestartet werden soll, wird die Bash-Variable „TEACH_POSITIONS“ eingesetzt. Zur Auswahl der verwendeten Positionsdatei kann die Bash-Variable „CONFIGURATION_FILE_PATH“ mit dem Pfad zur Positionsdatei genutzt werden. Mit der Bash-Variable „USE_SIMULATION“ kann eingestellt werden ob die Anwendung im Simulator gestartet werden soll. Die Bash-Variable „DO_MEASUREMENT“ kann bei Verwendung eines realen Industrieroboters dazu verwendet werden um Messvoraussetzungen, wie z.B. ein simuliertes Netzwerk, bereitzustellen. Zudem kann bei aktivierter „DO_MEASUREMENT“-Bash-Variable die Bash-Variable „SIMULATE_ADDITIONAL_ROS_NODES“ aktiviert werden um zusätzlichen Netzwerktraffic zu generieren. Mit der Bash-Variable „ADDITIONAL_ROS_NODES_COUNT“ kann bei aktivierter „SIMULATE_ADDITIONAL_ROS_NODES“-Bash-Variable auch die Anzahl der zu simulierenden ROS-Nodes angegeben werden. Es werden 4K-Kamera-ROS-Nodes simuliert, welche jeweils 4K-Bilder mit 60 FPS per UDP über die Netzwerkschnittstelle transferieren. Um weitere von der Roboter-

Gesten-Anwendung unterstützte Parameter anwenden zu können, kann die Bash-Variable „ADDITIONAL_APP_PARAMETERS“ mit den nachstehenden Parametern modifiziert werden.

Kurzschreibweise des Parameters	Ausgeschriebener Parameter	Beschreibung
-pfp="<path>"	--positions-file-path="<path>"	Dateipfad zum Speichern oder Laden der aufgezeichneten Positionen (Default-Positionsdateipfad: "~/robot_arm_positions.json")
-opf	--overwrite-positions-file	Ermöglicht das Überschreiben der Positionsdatei. Zuvor sollte ein Backup der Positionsdatei gemacht werde um Datenverlust zu vermeiden.
-rrp	--repeat-recorded-positions	Die aufgezeichneten Positionen aus der Positionsdatei wiederholen lassen
-tp	--teach-positions	Positionen lernen
-rn="<name>"	--robot-name="<name>"	Name des Roboters (Default: "wx200")
-rm="<model>"	--robot-model="<model>"	Robotermodell (In der Regel der Name des Roboters Default: "wx200")
-ur	--use-ros	Mittels ROS oder einer direkten Verbindung mit dem Industrieroboter kommunizieren
-mhaex	--move-home-at-exit	Beim Beenden der Anwendung in die Ausgangsposition zurückfahren
-mhaer	--move-home-at-error	Bei einem Fehler der Anwendung in die Ausgangsposition zurückfahren
	--help	Zeigt die Hilfe

Literatur

[140a] *ISO 9283:1998*. en. Library Catalog: www.iso.org. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/22/22244.html (besucht am 22. 07. 2020).

[140b] *ISO/TR 13309:1995*. en. Library Catalog: www.iso.org. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/16/21679.html (besucht am 22. 07. 2020).

[Adm18] Admin. *Understanding Network Latency in Ethernet Switches*. en-US. Library Catalog: www.fiberopticshare.com Section: switch. Sep. 2018. URL: http://www.fiberopticshare.com/network-latency-in-ethernet-switches.html (besucht am 23. 07. 2020).

[Azua] *Azure Kinect*. en. Page Version ID: 945489630. März 2020. URL: https://en.wikipedia.org/w/index.php?title=Azure_Kinect&oldid=945489630 (besucht am 08. 06. 2020).

[Azub] *Azure Kinect DK – Develop AI Models | Microsoft Azure*. en. Library Catalog: azure.microsoft.com. URL: https://azure.microsoft.com/en-us/services/kinect-dk/ (besucht am 08. 06. 2020).

[Azuc] *Legacy Body Tracking (like Kinect v2!!)* de. Library Catalog: feedback.azure.com. URL: https://feedback.azure.com/forums/920053-azure-kinect-dk/suggestions/39945454-legacy-body-tracking-like-kinect-v2 (besucht am 24. 07. 2020).

[Bam+18] Cyrus S Bamji u. a. „From: C. Bamji et al., "1Mpixel 65nm BSI 320MHz Demodulated TOF Image Sensor with 3.5m Global Shutter Pixels and Analog Binning," ISSCC Deg. Tech. Papers, pp. 94-95, Feb 2018. IEEE Explore Link: https://ieeexplore.ieee.org/document/8310200/". en. In: (2018), S. 1–3.

[BM] Geoffrey Biggs und Bruce MacDonald. „A Survey of Robot Programming Systems". en. In: (), S. 5.

[BM08] Kevin Behnisch und Bjoern Matthias. „ISO 10218 – von der neuen Norm zur Anwendung / ISO 10218 – Putting the New Standard into Practice". In: Juni 2008, S. 2–14.

[BMM13] Susanne Boll, Susanne Maaß und Rainer Malaka. *Mensch & Computer 2013 – Tagungsband: 13. fachübergreifende Konferenz für interaktive und kooperative Medien*. de. Walter de Gruyter, Aug. 2013, S. 338–340. ISBN: 978-3-486-78122-9.

[Bra] Volker Brauer. „Gestenerkennung mit einem Datenhandschuh“. de. In: ().

[CHA13] SANTANU CHATTOPADHYAY. *EMBEDDED SYSTEM DESIGN*. en. PHI Learning Pvt. Ltd., Apr. 2013, S. 106–107. ISBN: 978-81-203-4730-4.

[Dat] *Datendurchsatz*. de. Page Version ID: 202023710. Juli 2020. URL: `https://de.wikipedia.org/w/index.php?title=Datendurchsatz&oldid=202023710` (besucht am 23. 07. 2020).

[Dea] *Dead man's switch*. en. Page Version ID: 960453369. Juni 2020. URL: `https://en.wikipedia.org/w/index.php?title=Dead_man%27s_switch&oldid=960453369` (besucht am 04. 06. 2020).

[Dem07] Klaus Dembowski. *Lokale Netze: Handbuch der kompletten Netzwerktechnik*. de. Pearson Deutschland GmbH, 2007. ISBN: 978-3-8273-2863-2.

[Din15] Keith Dinwiddie. *Basic Robotics*. en. Cengage Learning, Jan. 2015, S. 77. ISBN: 978-1-305-53705-7.

[Ech] *Echtzeitsystem*. de. Page Version ID: 200650574. Juni 2020. URL: `https://de.wikipedia.org/w/index.php?title=Echtzeitsystem&oldid=200650574` (besucht am 05. 06. 2020).

[Enc] Jean-Philippe Encausse. *Body Tracking without cuDNN*. de. Library Catalog: feedback.azure.com. URL: `https://feedback.azure.com/forums/920053-azure-kinect-dk/suggestions/38129473-body-tracking-without-cudnn` (besucht am 21. 06. 2020).

[eri] erickjpaul. *Komfort - Mixed Reality*. de-de. Library Catalog: docs.microsoft.com. URL: `https://docs.microsoft.com/de-de/windows/mixed-reality/comfort` (besucht am 18. 06. 2020).

[Gar] *Garbage Collection*. de. Page Version ID: 200472744. Mai 2020. URL: `https://de.wikipedia.org/w/index.php?title=Garbage_Collection&oldid=200472744` (besucht am 06. 06. 2020).

[Gaz] *Gazebo*. URL: `http://gazebosim.org/` (besucht am 15. 06. 2020).

[Gen] *Genauigkeit, Wiederholgenauigkeit und Auflösung*. URL: `https://www.pfeiffer-vacuum.com/de/know-how/mechanische-komponenten-im-vakuum/manipulatoren-und-mechanische-durchfuehrungen/genauigkeit-wiederholgenauigkeit-und-aufloesung/` (besucht am 02. 06. 2020).

[Geo] *geometry - Find perpendicular distance from point to line in 3D?* Library Catalog: math.stackexchange.com. URL: https://math.stackexchange.com/questions/1905533/find-perpendicular-distance-from-point-to-line-in-3d (besucht am 01. 07. 2020).

[Gesa] *Gesten im Ausland: Achtung Redner und Reisende! - Redenwelt.* URL: https://www.redenwelt.de/rede-tipps/gesten-im-ausland/ (besucht am 16. 06. 2020).

[Gesb] *Gestik.* de. Page Version ID: 197084398. Feb. 2020. URL: https://de.wikipedia.org/w/index.php?title=Gestik&oldid=197084398 (besucht am 15. 06. 2020).

[Gesc] *Liste von Gesten.* de. Page Version ID: 200588792. Juni 2020. URL: https://de.wikipedia.org/w/index.php?title=Liste_von_Gesten&oldid=200588792 (besucht am 15. 06. 2020).

[Ghi+19] Adarsh Ghimire u. a. „Smart Glove“. In: Jan. 2019, S. 1–3.

[Gig] *Gigabit Ethernet.* en. Page Version ID: 968139871. Juli 2020. URL: https://en.wikipedia.org/w/index.php?title=Gigabit_Ethernet&oldid=968139871 (besucht am 23. 07. 2020).

[Gra] Krystian Graczyk. „Eignungsuntersuchung von 2D und 3D Gesten für Kollaboration“. de. In: ().

[GRB19] Huaqun Guo, Hongliang Ren und Aishwarya Bandla. *IRC-SET 2018: Proceedings of the 4th IRC Conference on Science, Engineering and Technology.* en. Springer Nature, Okt. 2019, S. 418. ISBN: 978-981-329-828-6.

[Hal05] Mike Hall. „Windows CE 5.0 for real-time systems“. en. In: (2005).

[Han] *Handzeichen – kennst du ihre Bedeutung wirklich?* de-DE. Library Catalog: www.reden-ohne-worte.com Section: Gestik. Sep. 2018. URL: https://www.reden-ohne-worte.com/handzeichen-und-ihre-bedeutung/ (besucht am 16. 06. 2020).

[Hem] Stephen Hemminger. „Network Emulation with NetEm“. en. In: (), S. 1–9.

[Hol] *HoloLens 2—Overview, Features, and Specs | Microsoft HoloLens.* en-us. Library Catalog: www.microsoft.com. URL: https://www.microsoft.com/en-us/hololens/hardware (besucht am 21. 06. 2020).

[HS06] Martin Hägele und Timo Schäfer. *Aufbau von Industrierobotern.* de. Hrsg. von Hans-Jürgen Gevatter und Ulrich Grünhaupt. VDI-Buch. Berlin, Heidelberg: Springer, 2006, S. 743–744. ISBN: 978-3-540-34823-8. DOI: 10.1007/3-540-34823-9_27.

[Ind] *Industrieroboter*. de. Page Version ID: 199907434. Mai 2020. URL: https://de.wikipedia.org/w/index.php?title=Industrieroboter&oldid=199907434 (besucht am 01. 06. 2020).

[Int] *Interbotix/interbotix_ ros_ arms*. en. Library Catalog: github.com. URL: https://github.com/Interbotix/interbotix_ros_arms (besucht am 24. 07. 2020).

[Jog] *Jogging the Robot*. en-US. Library Catalog: motioncontrolsrobotics.com. Nov. 2017. URL: https://motioncontrolsrobotics.com/jogging-the-robot/ (besucht am 04. 06. 2020).

[Kas+16] Robert E. Kass u. a. „Ten Simple Rules for Effective Statistical Practice“. en. In: *PLOS Computational Biology* 12.6 (Juni 2016). Hrsg. von Fran Lewitter, S. 1–8. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004961. (Besucht am 21. 06. 2020).

[Kir06] Herbert Kircher. *IT: Technologien, Lösungen, Innovationen*. de. Springer Science & Business Media, Nov. 2006, S. 8–15. ISBN: 978-3-540-46164-7.

[Kou19] Anis Koubaa, Hrsg. *Robot Operating System (ROS): The Complete Reference (Volume 3)*. en. Studies in Computational Intelligence. Springer International Publishing, 2019. ISBN: 978-3-319-91589-0. DOI: 10.1007/978-3-319-91590-6. (Besucht am 07. 06. 2020).

[Kra18] Julia Krammling. „User Experience in der Gestensteuerung – Systematische Vergleichsstudie zwischen Arm- und Fingergestik“. Deutsch. In: (2018), S. 28.

[Lee] Thorsten Leemhuis. *Linux 5.3 freigegeben: Prioritäten deckeln und Trouble für Nvidia*. de. Library Catalog: www.heise.de. URL: https://www.heise.de/ct/artikel/Linux-5-3-freigegeben-Prioritaeten-deckeln-und-Trouble-fuer-Nvidia-4470638.html (besucht am 06. 06. 2020).

[Lxca] *LXC*. de. Page Version ID: 199359030. Apr. 2020. URL: https://de.wikipedia.org/w/index.php?title=LXC&oldid=199359030 (besucht am 14. 06. 2020).

[Lxcb] *LXC und LXD: was sind Linux Container? - Cloud & Hosting*. de-DE. Library Catalog: www.webhod.de Section: Cloud & Hosting. Dez. 2017. URL: https://www.webhod.de/lxc-und-lxd-was-sind-linux-container/ (besucht am 14. 06. 2020).

[Lxd] *LXD 2.0: Installing and configuring LXD [2/12]*. en. Library Catalog: ubuntu.com. URL: https://ubuntu.com/blog/lxd-2-0-installing-and-configuring-lxd-212 (besucht am 24. 06. 2020).

[Man] *sched(7) - Linux manual page*. URL: https://www.man7.org/linux/man-pages/man7/sched.7.html (besucht am 22. 06. 2020).

[Mar04] A. Marchesin. „Using Linux for real-time applications“. In: *IEEE Software* 21.5 (Sep. 2004). Conference Name: IEEE Software, S. 20. ISSN: 1937-4194. DOI: 10.1109/MS.2004.1331295.

[Mat] *Glossar Körpersprache und Wirkung - Monika Matschnig*. de-DE. Library Catalog: www.matschnig.com. URL: https://www.matschnig.com/medien-presse/glossar-koerpersprache/ (besucht am 16. 06. 2020).

[Mat07] Monika Matschnig. *Körpersprache*. de. Gräfe Und Unzer, Sep. 2007. ISBN: 978-3-8338-0789-3.

[Mica] *Microsoft Windows CE*. de. Page Version ID: 194448187. Nov. 2019. URL: https://de.wikipedia.org/w/index.php?title=Microsoft_Windows_CE&oldid=194448187 (besucht am 06. 06. 2020).

[Micb] *microsoft/Azure-Kinect-Sensor-SDK*. en. Library Catalog: github.com. URL: https://github.com/microsoft/Azure-Kinect-Sensor-SDK/blob/develop/docs/usage.md (besucht am 25. 06. 2020).

[Micc] *microsoft/Azure-Kinect-Sensor-SDK*. original-date: 2018-12-10T17:50:05Z. Juni 2020. URL: https://github.com/microsoft/Azure-Kinect-Sensor-SDK (besucht am 11. 06. 2020).

[Mil68] Robert B. Miller. „Response time in man-computer conversational transactions“. en. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*. San Francisco, California: ACM Press, 1968, S. 267–277. DOI: 10.1145/1476589.1476628.

[Mova] *MoveIt Motion Planning Framework*. URL: https://moveit.ros.org/ (besucht am 15. 06. 2020).

[Movb] *MoveIt Tutorials — moveit_tutorials Melodic documentation*. URL: https://ros-planning.github.io/moveit_tutorials/ (besucht am 15. 06. 2020).

[Mö12] Holger Mönnich. *Ein Steuersystem für die telemanipulierte und autonome robotergestützte Chirurgie*. de. KIT Scientific Publishing, 2012, S. 78. ISBN: 978-3-86644-777-6.

[Neu] Andreas Neupert. „Natürliche Benutzerschnittstellenkonzepte“. In: (). URL: ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3587/DIP-3587.pdf.

[NNPPM10] Pedro Neto, J. Norberto Pires und A. Paulo Moreira. „High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition“. en. In: *Industrial Robot: An International Journal* 37.2 (März 2010), S. 1. ISSN: 0143-991X. DOI: 10.1108/01439911011018911. URL: https://www.emerald.com/insight/content/doi/10.1108/01439911011018911/full/html (besucht am 04. 08. 2020).

[Nof99] Shimon Y. Nof. *Handbook of Industrial Robotics*. en. John Wiley & Sons, März 1999, S. 341–344. ISBN: 978-0-471-17783-8.

[Now17] Tobias Nowack. *[PDF] Mensch-Technik-Interaktion mittels Freiraumgesten | Semantic Scholar*. en. Library Catalog: www.semanticscholar.org. 2017.

[Ost04] Christian Osterrieder. *Komponentenmodelle für Web-Anwendungen*. de. diplom.de, Juli 2004, S. 80–86. ISBN: 978-3-8324-8119-3.

[PD19] Andreas Pott und Thomas Dietz. *Industrielle Robotersysteme: Entscheiderwissen für die Planung und Umsetzung wirtschaftlicher Roboterlösungen*. de. Springer Vieweg, 2019. ISBN: 978-3-658-25344-8. DOI: 10.1007/978-3-658-25345-5.

[Poh14] Hartmut Pohl. „Robot Operating System (ROS): Safe & Insecure“. In: *Automationspraxis* (Mai 2014), S. 5–7.

[Pra] *Präzision und Wiederholgenauigkeit*. URL: https://www.vision-doctor.com/wiederholgenauigkeit.html (besucht am 02. 06. 2020).

[Pra+04] Erwin Prassler u. a. *Advances in Human-Robot Interaction*. en. Springer Science & Business Media, Okt. 2004. ISBN: 978-3-540-23211-7.

[Pro] *programEN.gif (GIF-Grafik, 500 × 243 Pixel)*. URL: http://www.machinery-export.com/de/roboter/abb/img/programEN.gif (besucht am 04. 06. 2020).

[Pro13] Heike Proff. *Radikale Innovationen in der Mobilität: Technische und betriebswirtschaftliche Aspekte*. de. Springer-Verlag, Dez. 2013. ISBN: 978-3-658-03102-2.

[PSH97] V.I. Pavlovic, R. Sharma und T.S. Huang. „Visual interpretation of hand gestures for human-computer interaction: a review“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.7 (Juli 1997). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. ISSN: 1939-3539. DOI: 10.1109/34.598226.

[qm1a] qm13. *Azure Kinect body tracking joints*. en-us. Library Catalog: docs.microsoft.com. URL: https://docs.microsoft.com/en-us/azure/kinect-dk/body-joints (besucht am 11. 06. 2020).

[qm1b] qm13. *Azure Kinect Body Tracking SDK download*. en-us. Library Catalog: docs.microsoft.com. URL: https://docs.microsoft.com/en-us/azure/kinect-dk/body-sdk-download (besucht am 11. 06. 2020).

[Roba] *Robot Operating System*. de. Page Version ID: 200591087. Juni 2020. URL: https://de.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=200591087 (besucht am 07. 06. 2020).

[Robb] *Roboter schneller programmieren*. de. Library Catalog: www.industr.com. URL: https://www.industr.com/de/schnellere-entwicklung-anwendungen-robot-operating-system-ros-1712541 (besucht am 07. 06. 2020).

[Rosa] *de/ROS/Concepts - ROS Wiki*. URL: http://wiki.ros.org/de/ROS/Concepts (besucht am 07. 06. 2020).

[Rosb] *Distributions - ROS Wiki*. URL: http://wiki.ros.org/Distributions (besucht am 07. 06. 2020).

[Rosc] *Features Status*. URL: https://index.ros.org/doc/ros2/Features/#features (besucht am 07. 06. 2020).

[Rosd] *ROS-master-node-topic.png (PNG-Grafik, 1725 × 769 Pixel)*. URL: https://upload.wikimedia.org/wikipedia/commons/e/e7/ROS-master-node-topic.png (besucht am 07. 06. 2020).

[Rose] *ROS.org | Is ROS For Me?* en-US. Library Catalog: www.ros.org. URL: https://www.ros.org/is-ros-for-me/ (besucht am 07. 06. 2020).

[Sch] *SCHED_DEADLINE*. en. Page Version ID: 951203928. Apr. 2020. URL: https://en.wikipedia.org/w/index.php?title=SCHED_DEADLINE&oldid=951203928 (besucht am 06. 06. 2020).

[Sch20] Tim Schleicher. „Einführung und Überblick“. de.
In: *Kollaborierende Roboter anweisen: Gestaltungsempfehlungen für ergonomische Mensch-Roboter-Schnittstellen*. Hrsg. von Tim Schleicher. Gestaltung hybrider Mensch-Maschine-Systeme/Designing Hybrid Societies. Wiesbaden: Springer Fachmedien, 2020.
ISBN: 978-3-658-29051-1. DOI: 10.1007/978-3-658-29051-1_1.

[Tan+15] Adin Tan u. a. „Design of Low Cost Smart Insole for Real Time Measurement of Plantar Pressure“.
In: *Procedia Technology* 20 (Dez. 2015), S. 117–119.
DOI: 10.1016/j.protcy.2015.07.020.

[Tbf] *Ubuntu Manpage: tbf - Token Bucket Filter*.
URL: http://manpages.ubuntu.com/manpages/bionic/man8/tc-tbf.8.html (besucht am 23. 07. 2020).

[Tcn] *Ubuntu Manpage: NetEm - Network Emulator*.
URL: http://manpages.ubuntu.com/manpages/bionic/man8/tc-netem.8.html (besucht am 23. 07. 2020).

[Tea] *Teach-in (Technik)*. de. Page Version ID: 197654136. März 2020.
URL: https://de.wikipedia.org/w/index.php?title=Teach-in_(Technik)&oldid=197654136 (besucht am 04. 06. 2020).

[tesa] tesych. *About Azure Kinect DK*. en-us.
Library Catalog: docs.microsoft.com.
URL: https://docs.microsoft.com/en-us/azure/kinect-dk/about-azure-kinect-dk (besucht am 15. 06. 2020).

[tesb] tesych. *Azure Kinect DK hardware specifications*. en-us.
Library Catalog: docs.microsoft.com.
URL: https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification (besucht am 08. 06. 2020).

[Tof] *TOF-Kamera*. de. Page Version ID: 189220682. Juni 2019.
URL: https://de.wikipedia.org/w/index.php?title=TOF-Kamera&oldid=189220682 (besucht am 08. 06. 2020).

[Und] *Understanding Active & Passive Infrared Sensors (PIR) and Their Uses*. en. Library Catalog: www.arrow.com.
URL: https://www.arrow.com/en/research-and-events/articles/understanding-active-and-passive-infrared-sensors (besucht am 08. 06. 2020).

[Vre] *V-REP, Gazebo or ARGoS? A robot simulators comparison :: LENKASPACE ::* Library Catalog: lenkaspace.net.
URL: http://lenkaspace.net/tutorials/programming/robotSimulatorsComparison (besucht am 15. 06. 2020).

[Vul] *Vulkan (API)*. en. Page Version ID: 966915347. Juli 2020. URL: https://en.wikipedia.org/w/index.php?title=Vulkan_(API)&oldid=966915347 (besucht am 22. 07. 2020).

[Wei07] Jens-Martin Weidenhausen. „Mobile Mixed Reality Platform“. ger. phd. Darmstadt: Technische Universität, Jan. 2007, S. 127.

[Wel] *Welche Hardware eignet sich besser zur KI-Beschleunigung?* de. Library Catalog: www.elektronikpraxis.vogel.de. URL: https://www.elektronikpraxis.vogel.de/welche-hardware-eignet-sich-besser-zur-ki-beschleunigung-a-720869/ (besucht am 25. 07. 2020).

[Wer19] Wolfgang Werth. *Konferenz der Mechatronik Plattform Österreich 2019: "Machine Learning und Data Science in der Mechatronik": Tagungsband : Villach, 21. November 2019 FH Kärnten, Campus Villach*. de. Fachhochschule Kärnten, 2019. ISBN: 978-3-99076-100-7.

[Wha] *What is an RTOS - Real Time Operating System Information and Training*. en. Library Catalog: www.highintegritysystems.com. URL: https://www.highintegritysystems.com/rtos/what-is-an-rtos/ (besucht am 06. 06. 2020).

[Why] *Why don't we use ROS?* en-US. Library Catalog: www.pulurobotics.fi. URL: https://www.pulurobotics.fi/blog/pulurobotics-blog-1/post/why-don-t-we-use-ros-7 (besucht am 15. 06. 2020).

[Wid] *WidowX 200 Robot Arm - X-Series Robotic Arm*. URL: https://www.trossenrobotics.com/widowx-200-robot-arm.aspx (besucht am 11. 06. 2020).

[Wö06] Heinz Wörn. *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*. de. Springer-Verlag, Mai 2006. ISBN: 978-3-540-27416-2.

[Xnu] *XNU*. de. Page Version ID: 195027335. Dez. 2019. URL: https://de.wikipedia.org/w/index.php?title=XNU&oldid=195027335 (besucht am 06. 06. 2020).

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 2. August 2020 Helmut Rhomberg