TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

DIPLOMARBEIT

Modellierung und Simulation kollaborativer Roboter mit ROS

Modelling and simulation of collaborative robots with ROS

eingereicht von
cand. ing. Shaowen Han
geb. am 07.12.1991 in Shaanxi, China
Matrikel-Nummer: 4649156

Betreuer/in:

- Prof. Dr.-Ing. habil. Karsten Menzel / Prof. Dr.-Ing. Raimar J. Scherer
- Dipl.-Ing. Nicolas Mitsch
- Prof. Dr.-Ing. Frank Will / Dipl.-Ing. Robert Zickler

Dresden, den 27.08.2021

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich reiche sie erstmals als Prüfungsleistung ein. Mir ist bekannt, dass ein Betrugsversuch mit der Note „nicht ausreichend“ (5,0) geahndet wird und im Wiederholungsfall zum Ausschluss von der Erbringung weiterer Prüfungsleistungen führen kann.

Name: Han

Vorname: Shaowen

Matrikelnummer: 4649156

Dresden, den 27.08.2021

_____

Unterschrift cand. ing. Shaowen Han

Abkürzungsverzeichnis

SLAM	Simultane Lokalisierung und Kartierung
6R/7R	Sechs-Achsen-Roboterarm, Sieben-Achsen-Roboterarm
EKF	Extended Kalman Filter
DLR	Deutsches Zentrum für Luft- und Raumfahrt
PSO	Partikel-Schwarm-Optimierung
URDF	Unified Robot Description Format
API	Application Programming Interface
GUI	Graphical User Interface
KDL	Kinematics and Dynamics Library
PTAM	Parallel Tracking and Mapping
BA	Bundle Adjustment
CLSF	Constrained Local Submap Filter
MMBs	Map Merging Bases
ISAM	Indexed Sequential Access Method
UAV	Unmanned aerial vehicle
EM	Expectation-Maximization
TOF	Time of Flight
AMCL	Adaptive Monte Carlo Localization
IAS	Intelligente Assistenzsysteme
MMK	Mensch-Maschine-Kollaboration

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung der Arbeit

Mit der Vertiefung des Industrie 4.0-Prozesses ist Deutschland ein entwickeltes Land mit einer starken industriellen Präsenz. Aufgrund der abnehmenden einheimischen Arbeitskräfte, der steigenden Arbeitskosten und einiger Verhältnisse, die für manuelle Arbeit aufgrund einer anspruchsvollen Umgebung und hohem Risikofaktor nicht geeignet sind, wurden Roboter weithin eingesetzt. Derzeit entwickeln sich Roboter in Richtung Intelligenz und werden perfekt mit intelligenten Geräten, dem industriellen *Internet of Things*, industrieller Steuerung und anderen Bereichen kombiniert, um Produktionslinien mit automatisierten Systemen zu bilden.

Seit ihren Anfängen haben Industrieroboter immer den Eindruck erweckt, sperrig, gefährlich und schwer handhabbar zu sein. Der Betrieb von Industrierobotern erfordert von ihren Bedienern ein gewisses Maß an Fachwissen, das durch Programmierkenntnisse und theoretisches Wissen über die Robotik unterstützt wird. [1] So wurden Industrieroboter in einigen großen Verarbeitungsbetrieben eingesetzt, was für die Verbreitung auf anderen Märkten nicht förderlich ist. Mit dem kontinuierlichen Fortschritt der Technologie werden immer mehr Roboter in Unternehmen eingesetzt. Bis jetzt sind die Anforderungen der Unternehmen an Roboter nicht nur, einige einfache Bahnplanungen wie Schweißen, Palettieren und andere Funktionen zu erledigen, sondern auch die Nachfrage nach intelligenten Funktionen von Robotern wächst, zum Beispiel Bildverarbeitung, Hindernisvermeidung, optimale Bahnplanung und so weiter. Da die traditionellen Industrie-Roboter jedoch nicht über diese Funktionen verfügen, werden die oben genannten intelligenten Funktionen zweifellos die Kosten erhöhen. [1] Um diese Funktionen zu vereinen, vereinheitlichen die Forschungs- und Entwicklungsteams der Robotik Unternehmen weltweit die für den Entwicklungsprozess benötigten Testplattformen und arbeiten an der Implementierung von Roboterfunktionen.

Laut Forschung ist das ROS (Robot Operating System) in der Entwicklung von Robotern weit verbreitet. Da es eine einheitliche Knotensteuerung gibt, können sowohl die Code-Wiederverwendungsrate als auch die Modularität des Systems verbessert werden, um verschiedene komplexe Roboter zu steuern, indem die Module im traditionellen Robotik System durch Knoten in einer einfachen Denkweise ersetzt werden.

Mit der Entwicklung der Robotik ist die Zusammenarbeit von Mensch und Maschine möglich geworden. Die Einführung mehrerer neuer Konzepte kollaborativer Roboter hat nicht nur neue Ideen für die Industrie und sogar für Dienstleistungen geliefert, sondern kollaborative Roboter auch zu einer der heißesten Forschungsrichtungen im Bereich der Robotik gemacht.

In den letzten Jahrzehnten wurden umfangreiche Forschungen zur Entwicklung von kollaborativen autonomen Systemen durchgeführt, mehrere Übersichtsarbeiten haben die neuesten Ergebnisse auf diesem Gebiet zusammengefasst. Der Einsatz mehrerer Roboter bringt eine Reihe neuer Herausforderungen mit sich, wie z. B. die Wahrnehmung durch mehrere Roboter [2] [3], simultane Multi-Roboter Lokalisierung und Kartierung (SLAM) [4] [5], verteiltes System und koordinierte Exploration, um nur einige zu nennen. Unter den oben genannten Forschungsschwerpunkten ist die kollaborative Wahrnehmung, Lokalisierung und Kartierung, bei der es darum geht, die von einzelnen Robotern wahrgenommenen Daten zu kombinieren und zu verteilen, um die relative Position zu schätzen und ein umfassendes Verständnis der Umgebung zu rekonstruieren, eines der kritischsten Themen.

1.2 Aufbau der Arbeit

Das Ziel dieser Arbeit ist es, eine Simulation eines kollaborativen Roboterarms für die Steuerung eines Roboters mit festem Roboterarm zu entwerfen. Im nächsten Kapitel wird dargestellt, was ein kollaborativer Roboter ist, seine Entwicklungsgeschichte dargestellt und eine Einführung in den Mainstream der internationalen kollaborativen Roboter sowie die aktuelle Entwicklung der kollaborativen Roboter in Deutschland und Europa gegeben.

Im darauffolgenden Abschnitt werden die Perspektiven und Forschungsrichtungen von kollaborativen Roboterarmen aufgezeigt, die in folgende Punkte unterteilt sind: Mensch-Computer-Interaktion, Sicherheitsforschung, Dynamikforschung und Analyse in Kombination mit spezifischen Produkten.

Bevor das Programm geschrieben wird, muss ein Konzept namens SLAM verstanden werden, dass sich mit dem Problem der Lokalisierung und Navigation von mobilen Robotern in unbekannten Umgebungen befasst. In diesem Beitrag wird SLAM in der Einzelroboterarchitektur und in der Multi-Roboter-Systemarchitektur vorgestellt und diskutiert.

Der nächste Teil der Simulation wird in der Ubuntu-Umgebung über die ROS-Plattform geschrieben. Um diese Aufgabe zu erfüllen, muss der Roboter in einer Simulationsumgebung simuliert werden, die es ihm ermöglicht, die Aktion des Greifens eines Objekts und des Ablegens in einer definierten Position selbstständig durchzuführen. Dazu gehören die Erkennung von Objekten und das Erkennen und Umgehen von Hindernissen, die während des Greifvorgangs vorhanden sind.

Das Steuerungssystem basiert auf dem Betriebssystem ROS. Bei der Verwendung dieses neuen Systems ist es notwendig, den Umgang mit der Roboterplattform zu erlernen, um die Simulation zu entwerfen und das Steuerungssystem auf der ROS-Plattform zu entwickeln, sowie Informationen auf der ROS-Plattform zu übertragen. Darüber hinaus ist es erforderlich zu lernen, wie die Parameter des

simulierten Objekts in der Simulation eingestellt werden, einschließlich der Geschwindigkeit und der Lauftrajektorie beim Greifen des Ziels und einigen Plugins wie MoveIt mit SLAM-Funktionspaket. Die Ergebnisse der Arbeit werden anhand eines simulierten Gazebo demonstriert.

Um die oben genannten Ziele zu erreichen, wird in dieser Arbeit die technische Grundlage und Forschung durchgeführt.

Nachdem das Simulationsprogramm ausgeführt wurde, wird eine Analyse der kooperativen Steuerung mehrerer Roboter durchgeführt. Dazu gehört die kooperative Lageregelung von zwei und mehr Robotern, die kooperative Regelung von Kräften verschiedener Roboter mit unterschiedlichen Positionierungstechniken wird diskutiert und analysiert.

2 Stand der Technik

2.1 Geschichte der Entwicklung von kollaborativen Robotern

In den letzten Jahren, in denen sich die Marktnachfrage nach Robotern von dem Modell großvolumig und Einzelmodus zu einer geringvolumigen und diversifizierten Richtung ändert, sind die Anforderungen der Unternehmen an Maschinen nicht nur auf sich wiederholende Betriebsaufgaben beschränkt, sondern die Anforderungen an die Kosten- und Ertragseffizienz sind stark gestiegen, und es wird gehofft, dass Roboter schnell in mehreren Arbeitsaufgaben eingesetzt werden können, was eine einfache Programmierung und einfach zu bedienende Roboter erfordert. Infolgedessen gewinnen kollaborative Roboter allmählich die Aufmerksamkeit von Ländern auf der ganzen Welt. Ein kollaborierender Roboter ist nach ISO 10218-2 definiert als ein Roboter, der in der Lage ist, direkt mit einem Menschen in einem bestimmten Kollaborationsbereich zu interagieren. [6] [7]Im Vergleich zu den traditionellen Industrierobotern haben kollaborierende Roboter die Vorteile der hohen Sicherheit, Vielseitigkeit, Sensibilität und Präzision, der einfachen Bedienung und der leichten Zusammenarbeit zwischen Mensch und Maschine. Die oben genannten Vorteile ermöglichen nicht nur den Einsatz kollaborierender Roboter in der Fertigung, sondern haben auch potenzielle Anwendungen in Bereichen wie Heimservice und Rehabilitationsmedizin. Zu den typischen kollaborativen Robotern gehören derzeit *iiwa, Yumi, Sawyer*, und *UR*, und andere. [8] [9]

Kollaborative Roboter haben mehr Variationen und Unterschiede in ihren Konfigurationen. Das Konfigurationsdesign des Roboters ist die Grundlage und der Schlüssel des Roboterdesigns, und hat einen wichtigen Einfluss auf die Leistung des Roboters. Die existierenden kollaborierenden Roboter verwenden unterschiedliche Konfigurationen, die meisten von ihnen verwenden redundante Roboterkonfigurationen mit sechs bis sieben Freiheitsgraden.

Eine der fruchtbareren Untersuchungen zu den *7 Freiheitsgrade (7R)* - Konfigurationen ist eine kinematisch optimale Konstruktionsansicht des Mechanismus, die von Hollerbach [10] 1985 in den USA vorgeschlagen wurde. Durch das Hinzufügen einer Rotierende Teilen zu jeder der Schultern, dem Ellbogen und dem Handgelenk des Mechanismus der optimalen Konfiguration *6R* und das Entfernen der degenerierten und sich wiederholenden äquivalenten Freiheitsformen, wurde schließlich eine Reihe von Roboterarmkonfigurationen erhalten und die optimale Konfiguration des Roboters *7R* empfohlen.

2.2 Einführung in den internationalen Mainstream kollaborativer Roboter

Kollaborative Roboter sind eine aufstrebende und beliebte Forschungsrichtung im Bereich der Industrierobotik, und nach Jahren der Forschung und Entwicklung haben internationale Robotik-Konzerne eine Vielzahl von kollaborativen Roboterprodukten auf den Markt gebracht.

Abbildung 2-1 Universal Robots [66]

Einige der typischen internationalen kollaborativen Roboter sind in der folgenden Abbildung dargestellt. **Abbildung 2-1** zeigt die UR-Serie des weltweit ersten

kollaborativen Roboters, der 2009 von Universal Robots aus Dänemark eingeführt wurde, das erste Industrieroboterprodukt, das mit dem Ziel der Kollaboration entwickelt wurde.

Die Geburt des ersten kollaborativen Roboters löste einen Trend in der Entwicklung der Mensch-Roboter-Interaktion in der Robotik aus, und Robotik Unternehmen auf der ganzen Welt übernahmen diesen Roboter als Designstandard, den es in Bezug auf Mechanik, Hardware-Antriebe und Leistung nachzuahmen galt.

Abbildung 2-2 YUMI Zweiarmiger kollaborativer Roboter [11]

Der YUMI-Doppelarmroboter, wie in **Abbildung 2-2** dargestellt, hat die wichtige Eigenschaft, dass er nach dem menschlichen Doppelarm konstruiert ist, und die Doppelarmstruktur und der Arbeitsbereich sind dem normalen menschlichen Arbeitsbereich nachempfunden. Daher nimmt der Roboter beim Arbeiten nicht zu viel Arbeitsraum ein und eignet sich für einige Präzisionsinstrumentenmontagearbeiten.

Die YUMI-Roboter verfügen über eine redundante 7-Achsen-Konstruktion für jeden einzelnen Arm, eine 14-Achsen-Doppelarmkonstruktion für geringes Gewicht und Flexibilität, einen Arbeitsradius von 559 mm, eine Tragfähigkeit von 0,5 kg, eine rückseitig montierte kompakte IRC5-Steuerung, die das Hinzufügen von Hardware-Geräten und Sensoren unterstützt, sowie eine hohe Positioniergenauigkeit von 0,02 mm. [11]

Das 1898 gegründete deutsche Unternehmen KUKA ist heute einer der weltweit führenden Hersteller von Industrierobotern. 2013 brachten KUKA und das DLR gemeinsam den LBR iiwa auf den Markt, einen 7-achsigen licht- und sensitiven Roboter, wie in **Abbildung 2-3** unten gezeigt.

Abbildung 2-3 KUKA Kollaborativer Roboterarm [12]

Der LBR iiwa wurde ursprünglich in zwei Ausführungen konzipiert: LBR iiwa 7 mit einer Traglast von 7 kg, einem Eigengewicht von ca. 23,9 kg und einer Armspannweite von 800 mm; LBR iiwa 14 mit einer Traglast von 14 kg, einem Eigengewicht von ca. 29,9 kg und einer Armspannweite von 820 mm; beide Roboter sind an eine KUKA smartPAD-Steuerung angeschlossen und unterstützen das Schlepp-Teaching. Der LBR iiwa eignet sich besonders für Branchen, die eine hohe Flexibilität, Biegsamkeit und Präzision erfordern, wie z. B. Elektronik, Pharmazie, Präzisionsinstrumente und andere Industriebereiche. [12]

2.3 Stand der Forschung im Bereich der Roboter-Trajektorien Planung

Die Forschung auf dem Gebiet der Roboterbewegung konzentriert sich auf die Trajektorien Planung und die Kontrolle der Genauigkeit. Die Trajektorien Planung wurde 2005 vom Indian *Institute of Technology (IIT)* untersucht, indem die

Beziehung zwischen dem Zerspanungseffekt eines von einem Roboter bearbeiteten Teils und der Schnitttiefe des Werkzeugs, der optimalen Arbeitsgeschwindigkeit während der Bearbeitung des Roboterteils und der vom Werkzeug während der Bearbeitung des Teils genommenen Trajektorie analysiert wurde. Im Jahr 2007 untersuchte der Japaner *Nagata* eine Bewegungssteuerung der Roboteroberfläche durch die Position des Robotergelenks und die Kraft des Endgelenks und bildete einen geschlossenen Regelkreis durch Kraftrückkopplung und Krafterkennung, um die Bewegungsgenauigkeit der Trajektorie zu verbessern, indem er die Bewegungsbahn des Robotergelenks kontinuierlich entsprechend der aktuellen Gelenkkraft korrigierte. [13] *Professor Guo Xiaotong* schlug eine Forschungsmethode zur Verbesserung der Steuerung der Robotergelenksposition vor, indem er die Robotergelenksposition durch PD-Feedback-Daten kontinuierlich änderte, und führte Experimente an einem vorhandenen Roboter durch. [14] Die optimale Lösung der inversen Kinematik des Roboters wird eingesetzt, um die Größe jedes Gelenkwinkels zu finden, indem die Trajektorienplanungsmethode mit den Gelenkwinkeln im kartesischen Koordinatensystem gelöst wird, und es werden numerische Simulationen durchgeführt, um die Ergebnisse zu überprüfen. Im Jahr 2017 führt *Ross A. Knepper* eine Kollisionserkennung bei der Trajektorienplanung durch eine Zufallsstichprobenmethode durch, die nicht nur in der Lage ist, Hindernisse in engen Räumen zu vermeiden, sondern auch eine gute Echtzeitleistung aufweist. [15] Durch die Analyse der obigen Forschungsergebnisse sind die meisten der aktuellen Robotertrajektorienplanungsmethoden darauf ausgelegt, die Positionssteuerung des Roboters im kartesischen Raum zu erreichen. Durch eine eingehende Untersuchung der inversen Roboterkinematik wird die Lösung der inversen Roboterkinematik durch das Vorhandensein von Problemen wie Singularitäten des Roboters und Robotergelenkgrenzen erschwert. Um die Untersuchung der Roboter Trajektorienplanung zu erleichtern, schlugen die Stanford University und Willow Garage Robotics, Inc. dieses Konzept im Jahr 2007 während der Durchführung einer Projektentwicklung vor. [16] Nach spezifischen Forschungen und Experimenten wurde das ROS-System 2010 offiziell eingeführt,

wobei das MoveIt-Paket eine Lösung für die Bibliothek zur Bewegungsplanung des Roboters bietet, welche viele Raum- und Steuerungsalgorithmusplaner in der OMPL-Bibliothek zur Bewegungsplanung enthält, die auf das mit MoveIt konfigurierte Robotermodell angewendet werden können. [17] OMPL ist in der Lage, verschiedene Berechnungen auf der Grundlage unterschiedlicher Trajektorien Algorithmen unter Verwendung der KDL-Operatorbibliothek durchzuführen. Diese ist in der Lage, Bewegungsberechnungen für komplexe Algorithmen durchzuführen und kann Werte mit hoher Genauigkeit berechnen, wodurch eine erhebliche Menge an Rechenzeit für die Robotertrajektorienplanung eingespart wird. [18]

Die Recherche und Lektüre zahlreicher Robotik-Algorithmus-Literatur fasst folgendes zusammen.

1. Um den Roboter durch den Algorithmus zu steuern, muss zunächst die inverse Kinematik des Roboters gelöst werden, was die Bewegungsgenauigkeit des Roboters beeinträchtigt, wenn die inverse Kinematik nicht korrekt gelöst werden kann.
2. Wenn sich der Roboter an einem bestimmten Punkt im kartesischen Raum befindet, ist die Hindernisvermeidung und die Trajektorien Planung nach einem einzigen Algorithmus rechenintensiv und zeitaufwändig. In diesem Beitrag schlagen wir eine Methode vor, um die beiden Algorithmen zu kombinieren und den Algorithmus in die OMPL-Algorithmus-Bibliothek in der ROS-Umgebung zu laden, um die Fähigkeit der Roboter-Trajektorien Planung zu verbessern, die Planungszeit zu reduzieren und die Bewegungsgenauigkeit zu erhöhen.

3 Entwicklungsmöglichkeiten und Forschungsrichtungen

Im Jahr 2002 prognostizierte Rodney Brooks (derzeitiger Vorsitzender und CTO von *Rethink Robotics*), dass "bis 2020 Roboter unser ganzes Leben beherrschen werden". [19] Diese Ansicht wurde dann vom reichsten Mann der Welt und Gründer von Microsoft, Bill Gates, aufgegriffen, der im Jahr 2007 sagte, dass "in naher Zukunft Roboter in Millionen von Haushalten stehen werden". [20]

Manche bezeichnen 2016 als das Jahr der kollaborativen Robotik. Laut einer Prognose der Barclay Bank wird der weltweite Umsatz mit kollaborativen Robotern bis 2025 auf 11,5 Milliarden US-Dollar steigen, gegenüber 116 Millionen US-Dollar im Jahr 2015. Kollaborative Roboter richten sich in erster Linie an kleine und mittelständische Unternehmen, von denen es weltweit etwa 6 Millionen gibt und die fast 70 % der globalen Produktion ausmachen. Die Wachstumsrichtung von kollaborativen Robotern ist also sehr klar, und wenn kollaborative Roboter in Zukunft versuchen, im Dienstleistungssektor zu expandieren, wird es einen höheren Entwicklungstrend geben.

Angesichts dieser breiten Marktperspektive haben viele Unternehmen und Forschungseinrichtungen ihre Forschungs- und Entwicklungsarbeit an kollaborierenden Robotern verstärkt, hauptsächlich in den folgenden vier Bereichen.

3.1 Mensch-Computer-Interaktion

Die Mensch-Maschine-Interaktion ist der Schlüssel zum Erreichen der Mensch-Maschine-Kollaboration. Die "Interaktion" findet hier nicht nur auf der Ebene der Stimme oder des Körpers statt, sondern auch im weiten Sinne der Steuerung und Rückmeldung.

In Bezug auf die Steuerungsmittel behalten die meisten kollaborativen Roboter

noch den Steuerungsmodus traditioneller Industrieroboter bei, d.h. sie sind mit einem speziellen Bedienfeld ausgestattet. Einige innovative Produkte, wie z.B. Baxter, Sawyer und FRANKAEMIKA, haben sich bis zu einem gewissen Grad von der Abhängigkeit von externen Geräten gelöst und sich von "Tischmaschinen" zu "*All-in-One-Maschinen*" entwickelt. Die neuen Produkte, wie *baxter, sawyer und FRANKA, EMIKA*, haben bis zu einem gewissen Grad die Abhängigkeit von externen Geräten verlassen und den Wandel von einer "Tischmaschine" zu "einer Maschine" realisiert.

In Bezug auf die Steuerungsmethoden besteht ein großer Spielraum für Erweiterungen. Die traditionelle Methode ist die Programmierung von Befehlen zur Steuerung der Roboterbewegung mit dem Controller [21]. Heutzutage, mit dem engen Kontakt zwischen dem Menschen und dem Roboterarm, ist die ziehende Demonstration ("Hands on") weit verbreitet. Einige der Roboter sind auch in das Visionssystem integriert, wodurch der Roboter durch "Beobachten" lernen und sich anpassen kann. [22]

Darüber hinaus haben einige Forscher Inertialmesseinheiten (IMUs) verwendet, um Robotern beizubringen, berührungslose Bewegungen mit Menschen zu synchronisieren, was in Dienstleistungsbranchen, in denen keine Präzision erforderlich ist, sehr nützlich ist. Die interessanteste Erforschung von kollaborativen Robotern im Bereich der künstlichen Intelligenz ist derzeit im Gange. Mit der starken Leistung von AlphaGo ist künstliche Intelligenz, insbesondere Deep Learning, zu einem weltweiten Diskussionsthema geworden, und kollaborative Roboter selbstlernend zu machen, ist ein beliebtes Thema für Unternehmen und Forschungseinrichtungen geworden. [23]

3.2 Untersuchungen zur Sicherheit

Sicherheit ist eine Voraussetzung für die gemeinsame Nutzung der Arbeitsumgebung durch Menschen und Maschinen. Zur Sicherheitsforschung von kollaborierenden Robotern haben *ABB-Forscher* herausragende Beiträge

geleistet. Zunächst haben sie die Kontaktflächen und Verletzungsarten zwischen Menschen und Maschinen klassifiziert und eine Methode zur Risikobewertung vorgeschlagen [24] [25]. Als nächstes wurden Crashexperimente [26], Fehlerdiagnosen und Fehlereingrenzung durchgeführt [27]; kürzlich wurde eine Strategie zur Bewegungssteuerung mit Sicherheit als harte Bedingung vorgeschlagen. [28] [29] Die Sicherheit von YUMI wurde von der Industrie aufgrund der umfassenden theoretischen und experimentellen Unterstützung gut angenommen. Darüber hinaus haben auch viele andere Forscher umfangreiche und tiefgreifende Forschungen durchgeführt, darunter Forschungen zu kollaborierenden Robotern, die während der Arbeit ein hohes Maß an Nachgiebigkeit aufrechterhalten, sowie Forschungen zur Verwendung von Restsignalen, um zu erkennen, ob eine Person in Kontakt mit einer Maschine ist sowie zur Verwendung von Tiefensensoren, um den Ort des Kontakts zu bestimmen. Man geht davon aus, dass mit dem Fortschreiten der Forschung die Koexistenz von Mensch und Maschine keine psychische Belastung mehr sein wird und gleichzeitig Arbeitsaufgaben effizienter und in Zusammenarbeit erledigt werden können. [30] [31]

3.3 Kinetische Untersuchungen

Die Kinematik ist seit langem ein wichtiges Forschungsgebiet in der Robotik. Besonders bei kollaborierenden Robotern wird die genaue Erfassung von Informationen über externe Kräfte zum Schlüssel für deren Bewegungssteuerung und Sicherheitsschutz. Ein intuitiverer Ansatz ist die Installation von Drehmomentsensoren an jedem Gelenk zur Messung, die wiederum die externe Kraft am Ende des Roboters abschätzen. [32] Diese Methode erfordert einen hohen Hardwareaufwand und ist in der praktischen Produktion sehr kostspielig. Deshalb versuchen viele Forscher, Kraftmessungen mit anderen Methoden durchzuführen, um die Sensoren loszuwerden. Zum Beispiel haben einige Forscher Interferenzbeobachter für die Schätzung der externen Kräfte am Ende des Roboters und der Interferenzmomente an jedem Joints verwendet [33]. Auf

der Grundlage des physikalischen Modells eines einzelnen Gelenks des humanoiden Roboters entwarfen die Forscher einen entsprechenden Störungsbeobachter und einen Regler, um den Einfluss des zeitlich veränderlichen nichtlinearen Gravitationsmoments auf die Regelungsleistung der Gelenksteuerung zu lösen. Die Leistung wird mit der eines PID-Reglers verglichen und die Auswirkung des Gelenkmodells auf den Störungsbeobachter wird diskutiert. Die Simulationsergebnisse zeigen die Machbarkeit und Effektivität der Methode bei der Steuerung von Gelenken humanoider Roboter [34]; Andere haben die Informationen über die externe Kraft rekonstruiert, indem Strom- und Drehmomentsignale sowie Gelenkdrehwinkel vom Motor erhalten wurden. [35] Wenn die Entwicklung der Kinematik weiter erforscht wird, werden die Kosten für kollaborative Roboter weiter sinken und flexiblere Materialien eingeführt werden, sodass sich die Sicherheit und Flexibilität der kollaborativen Roboter erheblich verbessern wird.

4 Gleichzeitige Lokalisierung und Kartierung-SLAM

4.1 Der aktuelle Zustand von SLAM für Einzelroboter

Das Problem der Lokalisierung und Navigation von mobilen Robotern in unbekannten Umgebungen wurde bereits 1986 von *H. Durrant-Whyte und T. Bailey* offen diskutiert und untersucht. [36] Ihr Verlangen, eine Bewertungsmethode auf das Problem der Zusammensetzung und Lokalisierung anzuwenden, hat SLAM-Techniken seither in den Fokus der Öffentlichkeit gerückt.

LIDAR war fast der erste Sensor, der für SLAM verwendet wurde. Seine Eignung als Sensor wurde 2007 von *Prof. Cyrill Stachniss* von der Universität Bonn, Deutschland, vorgeschlagen. Er ist eine auf Partikelfilterung basierende LIDAR-SLAM-Lösung und Open-Source-Anwendung für das bisher weit verbreitete GMapping. [37] In diesem Rahmen können mit LIDAR ausgestattete Roboter in Echtzeit eine zweidimensionale Karte der Umgebung erstellen. Obwohl LIDAR in Bezug auf Lokalisierung und Navigation viel weniger rechenintensiv ist als Vision und eine hohe Echtzeitleistung aufweist, hat LIDAR große Nachteile in Bezug auf die Identifizierung von Umgebungsinformationen und die kumulative Fehlerbeseitigung, und die Kosten für LIDAR sind erheblich, so dass die Entwicklung von LIDAR SLAM bis zu einem gewissen Grad eingeschränkt wird und die Akzeptanz für eine zivile Nutzung erschwert wird. Im Gegensatz dazu ist visuelles sensorgestütztes SLAM im Vergleich zu LIDAR zwar rechenintensiver und algorithmisch komplexer, aber in der Lage, die überwiegende Mehrheit der Texturinformationen der Umgebung effektiv zu identifizieren, und bietet damit einen großen Vorteil bei der Re-Lokalisierung und Szenenklassifizierung. In der frühen Phase der Entwicklung von visuellem SLAM konzentrierten sich die Wissenschaftler aufgrund der unzureichenden Rechenleistung der Computerhardware hauptsächlich auf die Theorie und die Rahmenbedingungen.

MonoSLAM [38]ist das erste monokulare Echtzeit-Vision-SLAM-System, das den

Extended Kalman Filter (**EKF**)-Algorithmus [39] verwendet, um die Positionierung zu schätzen und dessen Mittelwert und Kovarianz mit dem aktuellen Zustand der Kamera und allen Wegpunkten als Zustandsgrößen zu aktualisieren, während es spärliche Merkmalspunkte am Frontend verfolgt. Die Nachteile dieser Methode, die Posen durch Filterung zu schätzen, liegen jedoch auch auf der Hand, nämlich: kleinere Anwendungsszenarien, begrenzte Anzahl von verfügbaren Straßenschildern und leichter Verlust von spärlichen Merkmalspunkten. Yujiang Ju vom Harbin Institute of Technology, China, führte die *Partikel-Schwarm-Optimierung (**PSO**)* in den lokalen und globalen Lokalisierungsprozess im Kontext eines Hauskehrroboters ein, was die Lokalisierungsgenauigkeit des monokularen SLAM-Algorithmus und die Genauigkeit der konstruierten Karten stark verbesserte. Der Algorithmus *Parallel Tracking And Mapping (PTAM)* [40]war der erste, der die nichtlineare Optimierung in SLAM einführte und Pionierarbeit bei der Unterscheidung zwischen *Front-End* und *Back-End* leistete. Der PTAM-Algorithmus reduziert den Rechenaufwand durch die Einführung des Keyframe-Konzepts erheblich. PTAM hat jedoch den Nachteil kleinerer Anwendungsszenarien und eines leichten Verlustes der Keyframe-Nachführung. Im Jahr 2015 wurde *ORB-SLAM* entwickelt, um viele Probleme im SLAM-Bereich zu lösen. *ORB-SLAM* verwendet innovativ drei Threads, um SLAM basierend auf den ORB-Funktionen (*Oriented FAST und BRIEF*) durchzuführen. Diese drei Threads sind der Tracking-Thread für die Echtzeitverfolgung von Merkmalspunkten Die drei Threads sind der *Tracking-Thread* für die Echtzeitverfolgung von *Feature-Punkten*, der Optimierungs-Thread für die lokale **BA** [41] (Bundle Adjustment) und der *Loopback-Erkennungs-* und *Optimierungs-Thread* für den globalen Pose-Graphen. Das Highlight des Algorithmus ist die Verwendung von lokalen und globalen Optimierungsstrategien, um die Genauigkeit des SLAM-Algorithmus erheblich zu verbessern. Im Jahr 2017 verwendete Jing Wang von der Tsinghua-Universität eine verbesserte Methode, die auf einer inversen Tiefenparametrisierung für die lokale Kartenoptimierung von SLAM mit monokularer Sicht basiert, um die Genauigkeit der Positionsschätzung zu verbessern. Experimentelle Ergebnisse im Außenbereich zeigen, dass das

Verfahren eine hohe Genauigkeit bei der gleichzeitigen Lokalisierung und Kartenerstellung in großen Außenumgebungen erreichen kann. In den 2018 von Shen Shaojie von der Hong Kong University of Science and Technology vorgeschlagenen monokularen Inertial-Vision-Kilometerzählern VINS-Mono und VINS-Mobile wird ein eng gekoppelter, auf nichtlinearer Optimierung basierender Ansatz verwendet, um hochgenaue monokulare Vision-Inertial-Kilometerzähler durch die Fusion von vorintegrierten IMU-Messungen und Merkmalsbeobachtungen zu erreichen. [42]

4.2 Der aktuelle Zustand von SLAM für Mehrfachroboter

4.2.1 Multi-Roboter-Systemarchitektur

Gegenwärtig umfassen die Forschungsrichtungen von kollaborativem Multi-Roboter-SLAM das Design der Systemarchitektur, kooperative Multi-Roboter-Lokalisierung und Kartenfusion. Entsprechend der Architekturform von Multi-Roboter-Systemen kann das kollaborative Multi-Roboter-SLAM als zentralisiert und verteilt klassifiziert werden. [43] Mit zentralisiert ist gemeint, dass eine Master-Steuerung die Posen der einzelnen Roboter schätzt und gemeinsam eine globale Karte erstellt, indem die Beobachtungen der einzelnen Robotermitglieder verarbeitet werden. Wie in **Abbildung 4-1** dargestellt, ist die zentralisierte obere Schicht die Prozessoreinheit und die untere Schicht ist der Roboter, der die Daten sammelt. Die einzelnen Roboter sind unabhängig voneinander und tauschen nur mit der übergeordneten Steuerung Informationen aus und teilen diese. Die zentralisierte Architektur hat eine geringere Systemkomplexität und eine bessere

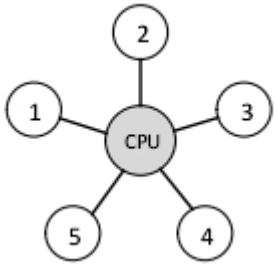


Abbildung 4-1 Zentralisierte Multi-Roboter-Architektur

Gesamtleistungsoptimierung, erfordert jedoch ein leistungsstärkeres CPU-Modul als zentrale Workstation und ist weniger anpassungsfähig an dynamische und komplexe Umgebungen.

Bei einer verteilten Architektur, wie in Abbildung 4-2 dargestellt, verfügt jeder Roboter über eine eigene Datenverarbeitungseinheit und ist in der Lage, seine eigenen Aufgaben, d. h. Positionsschätzung und lokale Kartenerstellung, unabhängig durchzuführen, während er gleichzeitig über die Kommunikation durch die Nutzung des gleichen WiFi zwischen den Robotern Informationen zur globalen Erkennung austauschen kann. Eine verteilte Architektur ist die Zukunft der Multi-Roboter-Kollaboration, aber dies erfordert die Einrichtung sehr ausgeklügelter Mechanismen für die Kommunikationsvernetzung zwischen den Maschinen.

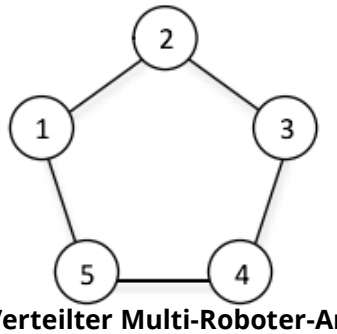


Abbildung 4-2 Verteilter Multi-Roboter-Architekturen

Im Laufe der Jahre wurden viele theoretische und experimentelle Studien durchgeführt, um das Problem des kooperativen Multi-Roboter-SLAM zu lösen. Im Jahr 2002 schlug *J.W. Fenwick* ein kooperatives SLAM-Schema für eine zentralisierte Robotersystemarchitektur vor, die einen zentralen Prozessor und einen Kalman-Filter benötigt, um die Pose jedes Roboters an globalen Koordinaten zu schätzen und eine globale Karte zu erstellen, wenn die anfängliche Position bekannt ist. [44] Dieses Konzept erfordert jedoch, dass die anfänglichen Position jedes Roboters bestimmt werden und die Datenkorrelation vor dem Experiment hergestellt wird, daüber hinaus wird das Experiment in einem sehr idealisierten Zustand durchgeführt, der sich nicht gut auf komplexere reale Umgebungen anwenden lässt. Im Jahr 2004 schlug *RAJ Madhavan* ein kollaboratives SLAM-Verfahren für verteilte Robotersysteme vor, das einen erweiterten Kalman-Filter-Algorithmus für die Positionsschätzung und eine

bildbasierte Fusion der Abstandsinformationen von Umgebungsmerkmalen für das Mapping verwendet. Das Schema hat gute Ergebnisse bei Tests in statischen Umgebungen, ist aber in dynamischen Umgebungen nur schwer anwendbar. [45] Dies liegt hauptsächlich daran, dass die Berechnungszeit exponentiell ansteigt, wenn dem Algorithmus neue Feature-Parameter hinzugefügt werden, was eine große Einschränkung darstellt. Im Jahr 2018 schlug der Wissenschaftler *Quande Yuan* eine vollständigere Multi-Roboter-SLAM-Lösung vor. Basierend auf der Untersuchung der Multi-Roboter-Systemstruktur, der Aufgabenzuweisung, der kooperativen Lokalisierung und der Datenassoziation kombiniert die Lösung den *Mean-Shift-Algorithmus* für die Extraktion von Straßenschildern und schlägt dann einen SLAM-Algorithmus basierend auf natürlichen Straßenschildern vor. [46] Bei dieser kollaborativen Multi-Roboter-Lösung nutzt jede Robotermitglied die Umgebungsbeobachtungen der anderen Mitglieder als Erweiterung seiner eigenen Sensoren, um eine globale Karte zu erstellen. Experimentelle Ergebnisse zeigen, dass diese Lösung einen größeren Vorteil bei großflächigem Umgebungs-SLAM hat.

4.2.2 Informationsverknüpfung zwischen Robotern

Die effektive Verknüpfung der Informationen zwischen den Robotern ist eine der Schlüsselfragen bei Multi-Robot-Vision-SLAM-Systemen. Derzeit gibt es zwei Methoden zur Informationszuordnung, eine basiert auf dem *In-Line-of-Sight*-Beobachtungsschema und die andere ist ein gemeinsames Flächenerkennungsschema. Die Wissenschaftler *Sebastian Thrun und Michael Montemerlo* schlugen 2005 einen Algorithmus zur sofortigen Lokalisierung und Kartenerstellung für mehrere Roboter vor. [47] Der Algorithmus war in der Lage, die kollaborative Kartenerstellung auch dann korrekt durchzuführen, wenn die relativen Startpositionen zwischen den Robotern unbekannt waren oder die Landmarken mehrdeutig waren. Der Algorithmus repräsentiert die Zusammensetzung der Karte und des Roboters in Form eines *Gaußschen Markov-Zufallsfeldes*, verwendet spärliche Informationsfiltertechniken für die kollaborative Kartenerstellung und erreicht die Angleichung der lokalen an die

globalen Mappings durch einen baumbasierten Algorithmus. [48] Bei der lokalen Suche wird die Relevanz der lokalen Suche durch die Ermittlung von ähnlichen lokalen Maps maximiert. Es ist erwähnenswert, dass dieser auf erweiterter Informationsfilterung basierende Ansatz eine Analyse der Informationsmatrix während der Berechnung und sehr große Matrizen zur Umrechnung bei der Schätzung von Karten und Posen erfordert, die die Anwendung des Algorithmus wiederum einschränken. Im selben Jahr schlugen *S.B. Williams; G. Dissanayake* und *H. Durrant-Whyte* an der *University of Sydney* eine Lösung mit geringerer Rechenkomplexität vor, die, statt Beobachtungen direkt in die globale Abbildung der Umgebung einzubeziehen, die Rechenkomplexität des Algorithmus verwaltet, indem die globale Abbildung durch die Fusion von Beobachtungen eingeführt wird und so die Leistung der Effizienz des Algorithmus bei der Datenzuordnung steigert. Jeder Roboter erstellt eine lokale Unterkarte durch *CLSF (Constrained Local Sub map Filter)* und aktualisiert kontinuierlich seine eigenen Karteninformationen und seine eigenen Positionsinformationen und sendet die Karteninformationen und seine eigene Pose an die globale Karte, sodass eine groß angelegte globale Kartenerstellung mit geringen Rechenkosten erreicht wird. [49] Im Jahr 2006 schlugen die drei Wissenschaftler *Andrew Howard, Lynne E. Parker und Gaurav S. Sukhatme* ein Multi-Roboter-SLAM-Schema vor, das auf Partikelfilterung basiert. Dieses Schema erfordert bekannte anfängliche Roboterposen oder die Fähigkeit, sich innerhalb der Beobachtungslinie zu treffen. Durch die Kombination der relativen Transformationsmatrizen zwischen den Robotern mit *Map Merging Bases (MMBs)* wird ein Algorithmus für die Gaußsche probabilistische Kartenfusion vorgeschlagen, der wiederum die Effizienz der Closed-Loop-Erkennung und die Echtzeitleistung von SLAM verbessert. [50] Der Autor überprüfte die Machbarkeit des Algorithmus durch Simulationen und Experimente, um die Genauigkeit der fusionierten Karten zu verbessern, aber die Partikelfilterung wird bis zu einem gewissen Grad von der Anzahl der Partikel beeinflusst. Im Jahr 2014 schlugen *Zhong Jin* von der Technischen Universität Harbin ein visuelles SLAM für die Zusammenarbeit von zwei Robotern vor, bei dem jeder Roboter zunächst unabhängig seine eigene Pfadkarte aus Knoten und

Kanten erstellt. Wenn zwei Roboter in das Sichtfeld des jeweils anderen eintreten, werden die für die Kartenfusion erforderlichen Informationen durch den Austausch von Informationen zwischen den Robotern gewonnen, sodass die Übertragungsmatrix zwischen den Robotern berechnet wird. Im Jahr 2015 entwickelte *Nived Chebrolu* ein Framework für ein kollaboratives Multi-Roboter-Vision-SLAM, das ein zentralisiertes Multi-Roboter-System darstellt. [51] Dabei führt jeder Roboterknoten zunächst sein eigenes Einzelroboter-LSD-SLAM durch und jeder baut seine eigene lokale Karte auf, während der zentrale Prozessor für den Abgleich und die Kartenfusion gemeinsamer Regionen zwischen den Robotern verantwortlich ist. [52] Sobald eine gültige Übereinstimmung erkannt wird, führt er eine Koordinatentransformation, eine Back-End-Graph-Optimierung und eine globale BA und Kartenfusion durch, die eine global konsistente Sparse zu erstellen. Dieser Ansatz liefert global konsistente spärliche Karten, womit dieses Schema zu einem der gängigsten Multi-Roboter-SLAM-Schemata geworden ist, die heute entwickelt werden.

4.2.3 Globale Optimierung

Aufgrund der Überlegenheit und kontinuierlichen Verbesserung der Idee der Graphen Optimierung konzentrieren sich immer mehr Wissenschaftler auf SLAM-Schemata, die auf der Graphen Optimierung basieren, und es werden immer mehr Ergebnisse erzielt. 2010 schlugen *Been Kim; Michael Kaess; Luke Fletcher* et al. einen Multi-Robot-Vision-SLAM-Algorithmus vor, der auf einer Graphen Optimierung basiert. [53]Der Algorithmus nutzt die Beziehung zwischen Multi-Bit-Positionskarten und Optimierung, um die *Indexed Sequential Access Method (ISAM)* auf Multi-Roboter-SLAM anzuwenden. [54]Der Algorithmus löst das Problem der Berechnung relativer Posen zwischen Robotern unter Verwendung einer indirekten Begegnung zwischen Robotern (d. h. ein bestimmter Wegpunkt oder Merkmalspunkt kann jederzeit von mehreren Sub-Robotern beobachtet werden) mit einer Graph-SLAM, um Karten- und Pose-Aktualisierungen durchzuführen. Außerdem wurde bei dem Algorithmus mit der Anwendung verschiedener Robotersysteme (UAV und Bodenwagen) experimentiert, die

ebenfalls die Machbarkeit des Algorithmus bestätigten. Die Wissenschaftler *Vadim Indelman; Erik Nelson; Nathan Michael; Frank Dellaert* et al. haben im Jahr 2014 die Entwicklung eines Multi-Roboter-SLAM-Systems auf Basis eines Graph-basierten Optimierungsansatzes durchgeführt. [55] Dieses System führt auch die Umrechnung von lokalen in globale Koordinaten durch, indem es gemeinsame Bereiche zwischen den Robotern teilt und eine globale Karte zeichnet. Um die Fehler beim Abgleichen zwischen verschiedenen Robotern zu minimieren, verbessern die Autoren außerdem die Genauigkeit der globalen Lokalisierung durch ein EM-basiertes (Expectation-Maximization) Verfahren zur Schätzung der Anfangsposition des Roboters. Im Jahr 2014 schlugen *Sajad Saeedi* und andere Wissenschaftler gemeinsam ein auf Sub-Maps basierendes kollaboratives Multi-Roboter-SLAM-System vor. [56] Dieses System kann durch die Verwendung von Stereosicht und relationalen metrischen Techniken eine intuitiv dichtere 3D-Karte der Umgebung konstruieren. In der Zwischenzeit entwirft das System für globale Multi-Roboter-Karten eine neue Topologie, die globale Multi-Roboter-Karten in Innen-, Außen- und Hybridumgebungen rekonstruieren kann. Im Jahr 2018 entwickelten *Stuart Golodetz* et al. ein kollaboratives 3D-Rekonstruktionssystem mit mehreren Knoten, um durch einen verbesserten, auf Graphen-Optimierung basierenden Back-End-Optimierungsalgorithmus die Inter-Frame-Drift zu eliminieren, sodass die Wahrscheinlichkeit einer Fehlanpassung zwischen Roboterkarten stark reduziert wird. [57] Das verbesserte System war in der Lage, eine effektive Rekonstruktion der globalen Karte in größeren Szenen durchzuführen, und zeigte eine beeindruckende 3D-Rekonstruktion.

Zusammenfassend lässt sich feststellen, dass es reichhaltige Ergebnisse in der SLAM-Forschung gibt, und dass es viele erfolgreiche Anwendungen von bildverarbeitungsbasiertem SLAM in der Praxis gibt. Verglichen mit der Einzelroboter-Vision-SLAM-Technologie müssen die Forschungsergebnisse in Bezug auf das kollaborative Multi-Roboter-SLAM jedoch noch weiter verbessert werden. Derzeit befindet sich die Forschung zum Multi-Roboter-Vision-SLAM noch in der Anfangs- und Erkundungsphase.

Der Forschungsschwerpunkt des kollaborativen SLAM mit mehreren Robotern liegt hauptsächlich auf der Positionsfusion zwischen den Robotern und der Erstellung von Karten. Bei der Fusion von Posen zwischen Robotern gibt es zwei Arten von Posen Funktion, die am häufigsten verwendet werden. Das erste ist das auf Roboterbegegnungen basierende Positionsfusionsschema, das die Posen zwischen den Robotern in einem kleinen Bereich effektiv fusionieren kann, aber den offensichtlichen Nachteil hat, dass es die Tatsache ignoriert, dass sich die Roboter überhaupt nicht begegnen, wenn die Karte groß ist.

Das zweite ist das Schema zur Erkennung eines gemeinsamen Bereichs, das die Begegnungsbedingung vermeidet, aber sehr strenge Kriterien für die Bestimmung des gemeinsamen Bereichs erfordert und bis zu einem gewissen Grad zu Fehlanpassungen führen und damit die Genauigkeit der globalen Karte beeinträchtigen kann. Die meisten aktuellen Schemata zur Kartenerstellung basieren auf Rasterkarten oder wenigen Merkmalspunkten. Diese Methode kann zwar die Umgebung prägnant darstellen, lässt aber zu viele Informationen über die Umgebung außer Betracht, sodass die Anschaulichkeit der Karte stark reduziert wird und die nachfolgenden Aufgaben deutlich eingeschränkt sind.

5 Aufbau des Roboters

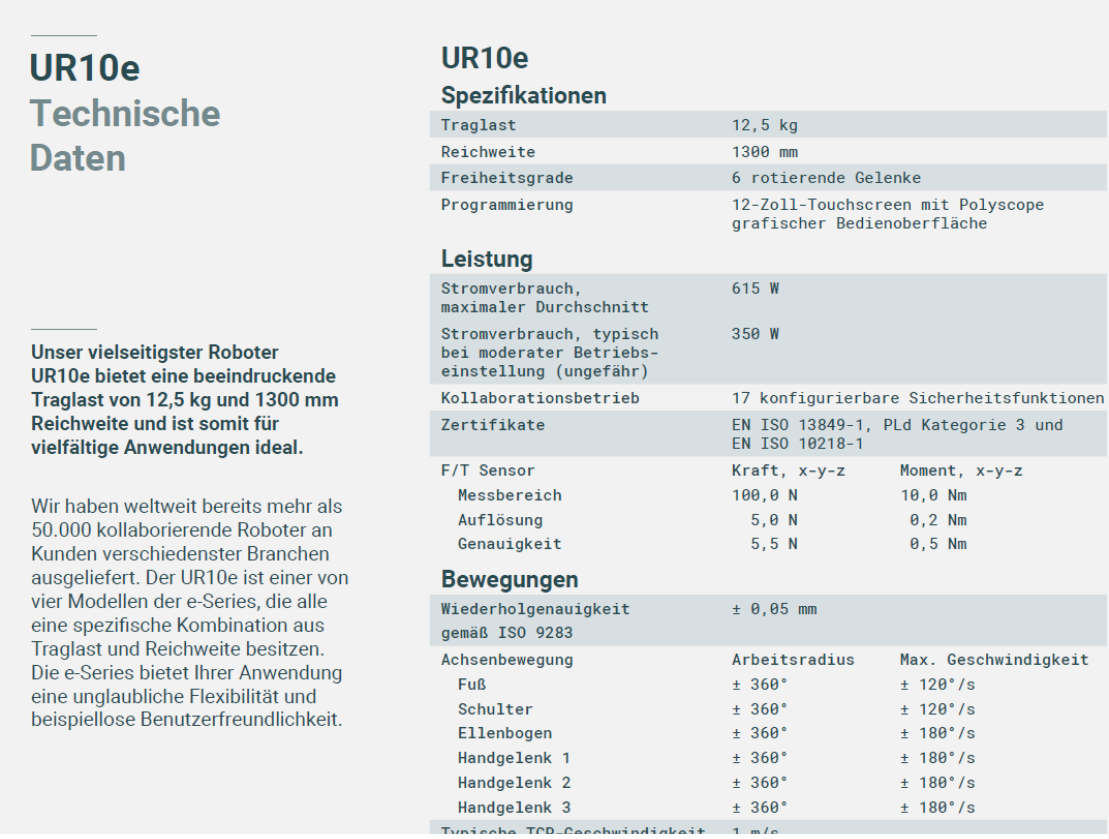
5.1 Funktionen des Roboters

In Anbetracht der Hindernisse auf der Baustelle müssen für das Materialhandling ein oder mehrere Roboter entworfen und simuliert werden, um diese Aufgabe zu realisieren. Für den Hauptroboter wird ein kollaborativer Roboterarm entworfen, der in der Lage sein wird, Objekte von einer festen Position aus zu identifizieren, mit einem mechanischen Greifer zu packen und auf einem anderen Roboterwagen abzulegen, der in der Lage sein wird, eine Wegerkennung und Hindernisvermeidung in der Umgebung durchzuführen.

Deshalb soll in dieser Arbeit der Aufbau der beiden Roboter, der Betrieb zur gleichen Zeit und die Zusammenarbeit der beiden Roboter diskutiert und analysiert werden.

5.2 Auswahl des Primärroboter-UR10e

Hier wird der Subjektroboter *UR10e* ausgewählt. Wie bereits in Abschnitt 2.2 vorgestellt, ist der kollaborative Roboterarm UR10e ein klassischer kollaborativer Roboterarm, der von einer dänischen Firma entwickelt wurde, da sein Modellcode Open Source ist, kann er bei unseren Simulationen verwendet werden. UR10e ist ein Roboterarm mit 6 Freiheitsgraden, der physikalisch 10KG an Fracht tragen kann. Die Gesamtlänge des Roboterarms beträgt 1513 mm, was bedeutet, dass der maximale Arbeitsradius etwa 1530 mm beträgt, da er die Hälfte der Länge der Greifer des Roboters umfasst. Einige technische Daten sind in der untenstehenden Abbildung dargestellt.



UR10e
Technische Daten

Unser vielseitigster Roboter UR10e bietet eine beeindruckende Traglast von 12,5 kg und 1300 mm Reichweite und ist somit für vielfältige Anwendungen ideal.

Wir haben weltweit bereits mehr als 50.000 kollaborierende Roboter an Kunden verschiedenster Branchen ausgeliefert. Der UR10e ist einer von vier Modellen der e-Series, die alle eine spezifische Kombination aus Traglast und Reichweite besitzen. Die e-Series bietet Ihrer Anwendung eine unglaubliche Flexibilität und beispiellose Benutzerfreundlichkeit.

UR10e

Spezifikationen

Traglast	12,5 kg	
Reichweite	1300 mm	
Freiheitsgrade	6 rotierende Gelenke	
Programmierung	12-Zoll-Touchscreen mit Polyscope grafischer Bedienoberfläche	

Leistung

Stromverbrauch, maximaler Durchschnitt	615 W	
Stromverbrauch, typisch bei moderater Betriebs-einstellung (ungefähr)	350 W	
Kollaborationsbetrieb	17 konfigurierbare Sicherheitsfunktionen	
Zertifikate	EN ISO 13849-1, PLd Kategorie 3 und EN ISO 10218-1	
F/T Sensor	Kraft, x-y-z	Moment, x-y-z
Messbereich	100,0 N	10,0 Nm
Auflösung	5,0 N	0,2 Nm
Genauigkeit	5,5 N	0,5 Nm

Bewegungen

Wiederholgenauigkeit gemäß ISO 9283	± 0,05 mm	
Achsenbewegung	Arbeitsradius	Max. Geschwindigkeit
Fuß	± 360°	± 120°/s
Schulter	± 360°	± 120°/s
Ellenbogen	± 360°	± 180°/s
Handgelenk 1	± 360°	± 180°/s
Handgelenk 2	± 360°	± 180°/s
Handgelenk 3	± 360°	± 180°/s
Typische TCP-Geschwindigkeit	1 m/s	

Abbildung 5-1 technische Daten von UR10e

5.3 Auswahl des Sekundärroboter-TurtleBot

Als nächstes muss ein Foo-Roboter konstruiert werden, um mit ihm eine Frachtlieferplattform zu ersetzen. Da ein Transportvorgang simuliert werden soll, wird nach der Auswahl eines Roboterarms, der greifen und platzieren kann, TB-TurtleBot als Lieferroboter ausgewählt. Ein Demonstrationsdiagramm des Subroboters ist in **Abbildung 5-2** dargestellt.

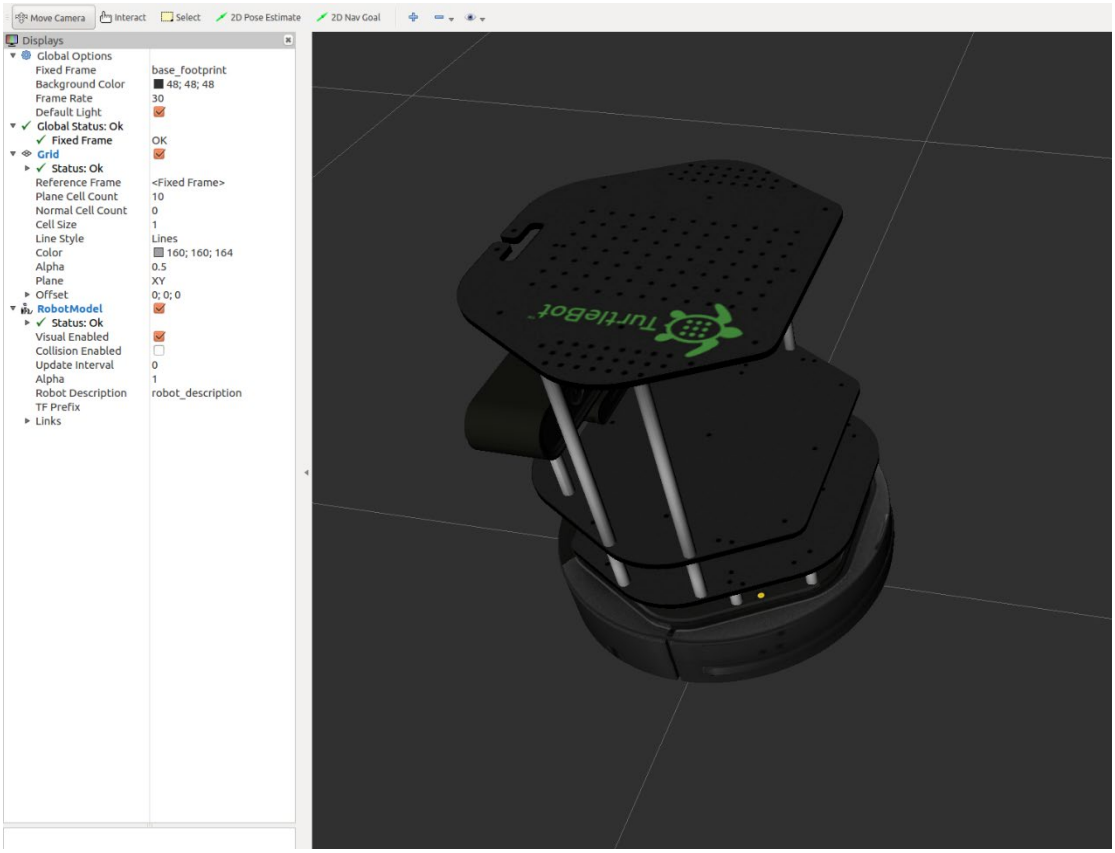


Abbildung 5-2 TurtleBot im RVIZ

Das Modell TurtleBot wurde ausgewählt, weil es ein ausgereiftes Open-Source-Wagenmodell ist, das mittels der ROS-Plattform manipuliert werden kann. Da die Aufgabe des Nebenroboters darin besteht, die von ihm platzierten Objekte vom Primärroboter, dem Roboterarm, aufzunehmen und somit zu transportieren, und da der TB-Roboter eine ebene Fläche auf der Oberseite hat, auf der Objekte platziert werden können, wurde dieses Robotermodell als unser Sekundärroboter ausgewählt.

5.4 Roboterkinematik-Analyse

Die Roboterkinematik ist am wichtigsten, um die Verbindung zwischen der Roboter-Endeffektor-Struktur und jedem seiner Joints und Achsen zu veranschaulichen, und besteht aus zwei Hauptaspekten: der rechnerischen Lösung der Vorwärts- und Rückwärtskinematik.

5.4.1 Positionsanalyse des Roboters

Die Posen und Koordinatentransformationen des Roboters sind die Grundlage für die kinematische Analyse von Industrierobotern. Der Roboter besteht hauptsächlich aus Achsen und Joints. Die Joints verursachen die Positionsänderung des Roboters, dabei wird eine bestimmte Art und Weise gewählt, um den Roboterzustand zu beschreiben, und zwar in Bezug auf die Positionsbeziehung zwischen jedem Gelenkkoordinatensystem des Roboters und dem Basiskoordinatensystem, die im Allgemeinen durch seine sekundäre Transformationsmatrix mit der folgenden Gleichung ausgedrückt wird.

$$T = \begin{pmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In der obigen Formel steht p für den Ort und T für die Transformation zwischen dem Ursprung jeder Gelenkkoordinate und dem Ort der Basiskoordinaten des Roboterarms. n, o und a beziehen sich auf die Richtungsvektoren jedes Joints in x, y und z im Basiskoordinatensystem des Roboterarms. n, o und a bilden die Rotationsmatrix R jedes Joints, wie unten gezeigt.

$$R = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix}$$

R ist die Matrix T zur Darstellung des Zustands jedes Joints, die zusammen mit der Positionsinformation p die Lageänderung jedes Joints im Basiskoordinatensystem des Roboterarms bildet.

5.4.2 D-H-Modellierung des Roboterarms

Die Gelenke des UR10e-Roboterarms und ihre Koordinatensysteme sind in **Abbildung 5-3** dargestellt. Der Arm hat sechs drehbare Gelenke, wobei die Gelenke 1 bis 6 dem Koordinatensystem {1} bis {6} entsprechen und mit der Basis verbunden sind.

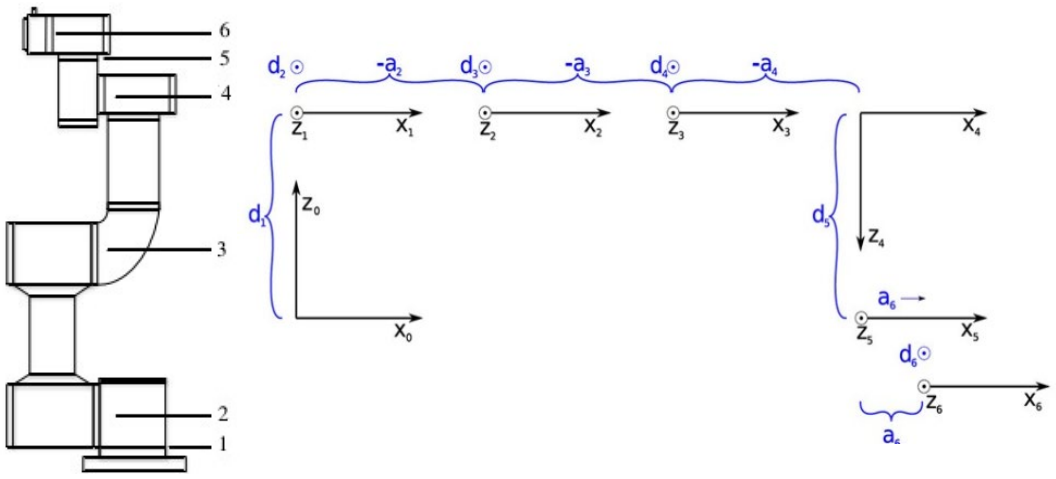


Abbildung 5-3 Skizze der Roboterarmgelenke und jedes

Jedes Gelenk des Roboters kann viele Formen annehmen, wie z. B. gleitend, drehend usw., sodass Abweichungen von Achse zu Achse auftreten können. Gleichzeitig können die Gelenke des Roboters variable Längen haben und sich in einer beliebigen Positionsebene befinden, sodass der Roboterarm modelliert und analysiert werden muss. Die D-H-Methode wird gewählt, um den Roboterarm für die Analyse zu modellieren.

Entsprechend der Position und der Lage der einzelnen Gelenkkoordinaten des UR10e-Roboterarms wird die Positionslage der Nullposition des Roboterarms so eingestellt, dass der gleiche Zustand wie in **Abbildung 5-3** erhalten bleibt, wobei die Parameter des Roboters sich ergeben wie in **Tabelle 5-1** dargestellt. Dabei bezieht sich i auf die Nummer der Gelenk-Seriennummer, a_{i-1} beschreibt die Winkelgröße der Stange i relativ zur Stange $i-1$, a_{i-1} beschreibt die Stangengröße von $i-1$, d_i bezieht sich auf die Versatzlänge, wenn die Gelenk-Seriennummer i ist, θ_i bezieht sich auf den Winkel, wenn die Gelenk-Seriennummer i ist, und die Werte der Nullposition θ_i sind jeweils [0, -π/2, 0, -π/2, 0, 0].

Tabelle 5-1 UR10eD-H-Parameter [58]

i	a_{i-1}(°)	a_{i-1}(mm)	d_i(mm)	θ_i (Reichweite)	θ(°)
1	π/2	0	180,7	±175	0
2	0	-612,7	0	±175	0
3	0	-571,55	0	±175	0
4	π/2	0	174,15	±175	0
5	-π/2	0	119,857	±175	0
6	0	0	116,55	±175	0

5.4.3 Roboter positive Kinematik Lösung

Die Drehwinkel der Gelenke des Roboterarms sind bekannt und die Lage des Handgelenk-Koordinatensystems des Roboterarms in Bezug auf das Basis-Koordinatensystem wird gelöst [43], die chi-Quadrat-Transformation des Stab-Koordinatensystems {i} in Bezug auf das Koordinatensystem {i-1} kann gemäß der D-H-Parameter-Tabelle gefunden werden, die in sechs Teilprobleme unterteilt ist, und sechs chi-Quadrat-Transformationsmatrizen werden entsprechend gelöst. Die sechs Matrizen werden multipliziert, um die Position des räumlichen Roboterarm-Koordinatensystems in Bezug auf das Basiskoordinatensystem zu erhalten. Die bündige Transformation des Gelenks {i} in das Gelenk {i-1} wird mit ${}^{i-1}_{i}T$ bezeichnet, die Formel ist unten dargestellt.

$$ {}^{0}_{6}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T \cdot {}^{4}_{5}T \cdot {}^{5}_{6}T $$

Die Zickzack-Koordinaten von ${}^{i-1}_{i}T$ können wie folgt ausgedrückt werden.

$$^{i-1}_{i}T = Screw_x(a_{i-1}, \alpha_{i-1}) Screw_z(d_i, \theta_i)$$

$$= \begin{bmatrix} cos\,\theta_i & -sin\,\theta_i & 0 & a_{i-1} \\ sin\,\theta_i\,cos\,\alpha_{i-1} & cos\,\theta_i\,cos\,\alpha_{i-1} & -sin\,\alpha_{i-1} & -sin\,\alpha_{i-1}\,d_i \\ sin\,\theta_i\,sin\,\alpha_{i-1} & cos\,\theta_i\,sin\,\alpha_{i-1} & cos\,\alpha_{i-1} & cos\,\alpha_{i-1}\,d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$Screw_x(a_{i-1}, \alpha_{i-1})$ ist eine Rotation α_{i-1} um die X_{i-1}-Achse und eine weitere Translation α_{i-1}entlang der X_{i-1}Achse. $Screw_z(d_i, \theta_i)$ ist um die Z-Achse um θ_i gedreht und dann entlang der Z-Achse um d_i bewegt.

In der **Tabelle 5-1 UR10eD-H-Parameter** bezieht sich θ_i auf jede Gelenkvariable des Roboterarms, die restlichen Werte können aus der Tabelle entnommen werden, wobei die Parameter in den oben genannten separat ersetzt werden, um jede Transformation der Verbindung wie folgt zu erhalten.

$$^{0}_{1}T = \begin{bmatrix} cos\,\theta_1 & -sin\,\theta_1 & 0 & 0 \\ sin\,\theta_1 & cos\,\theta_1 & -1 & 0 \\ 0 & 0 & 0 & 0{,}181 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}_{2}T = \begin{bmatrix} cos\,\theta_2 & -sin\,\theta_2 & 0 & -0{,}6127 \\ 0 & 0 & 0 & 0 \\ -sin\,\theta_2 & -cos\,\theta_2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}_{3}T = \begin{bmatrix} cos\,\theta_3 & -sin\,\theta_3 & 0 & -0{,}5176 \\ -sin\,\theta_3 & -cos\,\theta_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{3}_{4}T = \begin{bmatrix} cos\,\theta_4 & -sin\,\theta_4 & 0 & 0 \\ -sin\,\theta_4 & -cos\,\theta_4 & -1 & 0 \\ 0 & 1 & 0 & 0{,}1742 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
{}^{4}_{5}T = \begin{bmatrix} cos\,\theta_5 & -sin\,\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin\,\theta_5 & -cos\,\theta_5 & 0 & 0{,}1199 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{5}_{6}T = \begin{bmatrix} cos\,\theta_6 & -sin\,\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ sin\,\theta_6 & -cos\,\theta_6 & 1 & 0{,}1199 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Indem ${}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T \cdot {}^{4}_{5}T \cdot {}^{5}_{6}T$ in die vorherige Gleichung eingesetzt wird, wird die kinematische Vorwärtslösung für *UR10e* erhalten. Die Robotics Toolbox der MATLAB-Plattform wird in diesem Projekt als Simulationsplattform für die Vorwärtskinematik des Roboters verwendet, die Ausgangspositionen sind [0, -π/2, 0, -π/2, 0, 0], und das 3D-Schema des Roboterarms wird wie in **Abbildung 5-4** gezeigt gezeichnet.

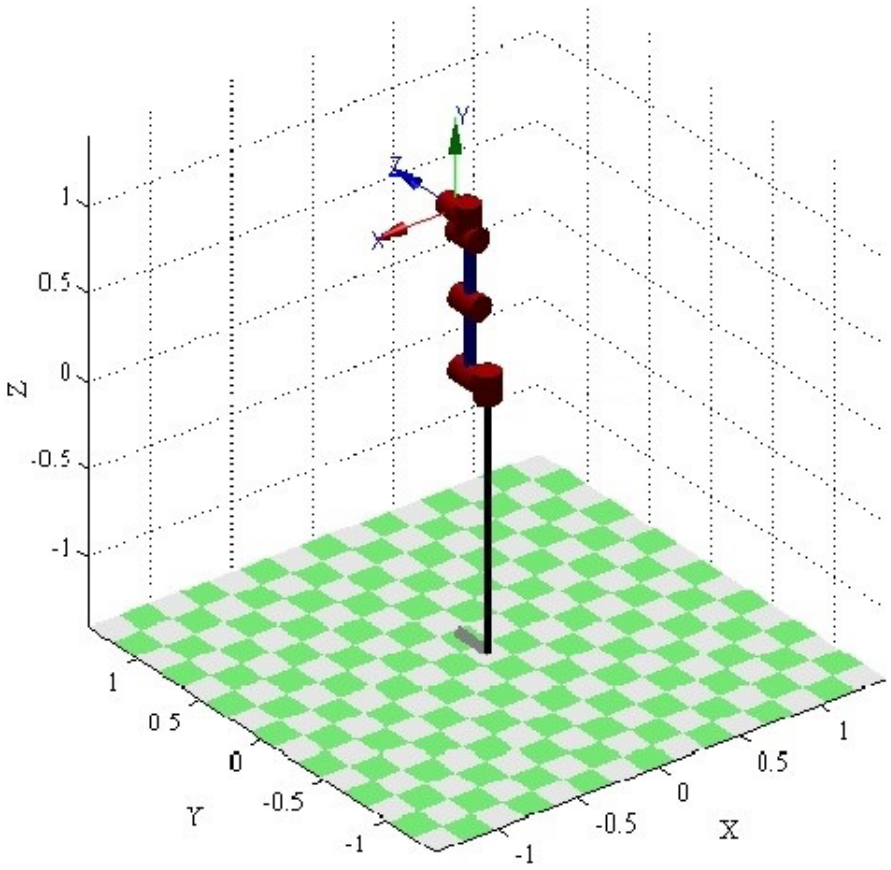


Abbildung 5-4 UR-Roboterarmsimulation Modelldiagramm

5.4.4 Robotik inverse Kinematik Lösung

Die Lösungsmethoden für die inverse Kinematik von Robotern umfassen in der Regel drei Arten: geometrische Formlösungsmethode, algebraisch-analytische Methode und numerische Berechnungsmethode. [59] Die algebraisch-analytische Methode wird gewählt, um die inverse Lösung des Roboters zu berechnen, und die Parameter der gemeinsamen Variablen können durch die Formel ${}_{i}^{i-1}T$ gefunden werden. Da jedoch unterschiedliche Transformationsmethoden zu unterschiedlichen Lösungsschwierigkeiten führen können, wird die Reihenfolge der Lösungen der gemeinsamen Variablen als θ_1 , θ_3 , θ_2 , θ_5 angenommen , θ_4 , θ_6 , und das Lösungsverfahren verwendet die bivariate inverse Tangensfunktion $a\tan 2$, die im Bereich [-180°, 180°] liegt.

a) Berechnung von θ_1

$$ {}_{6}^{0}T \cdot \left[{}_{6}^{5}T \right]^{-1} = {}_{1}^{0}T \cdot {}_{2}^{1}T \cdot {}_{3}^{2}T \cdot {}_{4}^{3}T \cdot {}_{5}^{4}T $$

Nach der Gleichung θ_1 wird ${}_{6}^{0}T \cdot \left[{}_{6}^{5}T \right]^{-1}$ berechnet und durch Vereinfachung wird: $-s_1 p_x + c_1 p_y = 0$ erhalten, wodurch zwei Lösungen für θ_1: gefunden werden.

$$ \begin{cases} \theta_{11} = a\tan 2(p_y, p_x) \\ \theta_{12} = a\tan 2(-p_y, -p_x) \end{cases} $$

b) Berechnung von θ_3

$$ \begin{cases} c_1 p_x - s_1 p_y - a_1 = c_{23} a_3 + s_{23} d_4 + c_2 a_2 \\ p_z - d_1 = s_{23} a_3 + c_{23} d_4 + s_2 a_2 \end{cases} $$

wobei die Parameter $c_1 = \cos\theta_1$, $s_1 = \sin\theta_1$, $c_{23} = \cos(\theta_2 + \theta_3)$, $s_{23} = \sin(\theta_2 + \theta_3)$ angewendet werden, dann wird, durch Vereinfachung, die folgende Gleichung erhalten.

$$ a_3 c_3 - d_4 s_3 = \left(c_1 p_x + s_1 p_y - a_1\right)^2 + (p_z - d_1)^2 - {a_2}^2 - {a_3}^2 - {d_4}^2 $$

Sei $k = \left(c_1 p_x + s_1 p_y - a_1\right)^2 + (p_z - d_1)^2 - {a_2}^2 - {a_3}^2 - {d_4}^2$, dann finde die beiden Lösungen von θ_3

$$\begin{cases} \theta_{31} = a\tan 2(a_3, d_4) - a\tan 2(k, \sqrt{a_3^2 + d_4^2 - k^2} \\ \theta_{32} = a\tan 2(a_3, d_4) - a\tan 2(k, -\sqrt{a_3^2 + d_4^2 - k^2} \end{cases}$$

c) Berechnung von θ_2

$${}_6^0T \cdot [{}_3^0T]^{-1} = {}_4^3T \cdot {}_5^4T \cdot {}_6^5T$$

Nach der Vereinfachung ergibt sich die folgende Gleichung:

$$\begin{cases} s_{23} = \dfrac{-(a_3 + a_2 c_3)(p_z - d_1) + (d_4 - a_2 s_3)(c_1 p_x + s_1 p_y - a_1)}{(c_1 p_x + s_1 p_y - a_1)^2 + (p_z - d_1)^2} \\ c_{23} = \dfrac{(a_3 + a_2 c_3)(c_1 p_x + s_1 p_y - a_1) + (d_4 - a_2 s_3)(p_z - d_1)}{(c_1 p_x + s_1 p_y - a_1)^2 + (p_z - d_1)^2} \end{cases}$$

Die Berechnung ergibt

$$\begin{cases} s_2 = s_3 c_{23} + c_3 s_{23} \\ c_2 = c_3 c_{23} - s_3 s_{23} \end{cases}$$

Deshalb wird θ_2 wie folgt gefunden:

$$\theta_2 = a\tan 2(s_1, c_2)$$

d) Berechnung von θ_5

Viele bisherige Studien haben θ_4 berechnet, dann durch Berechnung des Sinus und Kosinus von θ_4 , und schließlich θ_5 und θ_6. Bei der Berechnung von θ_4 wird der Sinus von θ_5 unter der Annahme, dass s_5 nicht 0 ist, untersucht und die Richtung von 5s wird hier untersucht, da die Richtung von s_5 einen Einfluss auf die Ergebnisse der Lösung haben kann. So kann die folgende Gleichung durch Berechnung erhalten werden:

$$\begin{cases} -c_4 s_5 = c_1 c_{23} a_x + s_1 c_{23} a_y - s_{23} a_z \\ s_4 s_5 = s_1 a_x - c_1 a_y \end{cases}$$

Unter der Annahme von s5 ungleich 0 kann die obige Gleichung wie folgt

umgerechnet werden:

$$\begin{cases} c_4 = \dfrac{-(c_1 c_{23} a_x + s_1 c_{23} a_y - s_{23} a_z)}{s_5} \\ s_4 = \dfrac{s_1 a_x - c_1 a_y}{s_5} \end{cases}$$

Nach Vereinfachung ergibt sich die folgende Gleichung:

$$c_5 = c_1 s_{23} a_x + s_1 s_{23} a_y - c_{23} a_z$$

Mit Hilfe des inversen Kosinus ist es möglich, zwei Lösungen von θ_5 zu erhalten:

$$\begin{cases} \theta_{51} = a\cos(c_5) \\ \theta_{52} = -a\cos(c_5) \end{cases}$$

Schließlich werden die Werte von th4 und th6 durch die Größe von s_5 berechnet. Unter der Annahme, dass s_5 gleich Null ist, können $\theta_4$und θ_6 die Werte von $\theta_4$und θ_6 aus dem vorherigen Interpolationspunkt verwenden; unter der Annahme, dass s_5 nicht gleich Null ist, folgen Sie den Schritten zur Lösung von $\theta_4$und θ_6.

e) Berechnung von θ_4

$$\theta_4 = a\tan 2(s_4, c_4)$$

f) Berechnung von θ_6

$$\begin{cases} -s_6 = (c_1 c_{23} s_4 + s_1 c_4) n_x + (s_1 c_{23} s_4 - c_1 c_4) n_y - s_{23} s_4 n_z \\ c_6 = [(c_1 c_{23} c_4 - s_1 s_4) c_5 + c_1 s_{23} s_5] n_x + [(s_1 c_{23} c_4 + c_1 s_4) c_5 + s_1 s_{23} s_5] n_y \\ \quad + (-s_{23} c_4 c_5 + c_{23} s_5) n_z \end{cases}$$

Die Berechnung ergibt:

$$\theta_6 = a\tan 2(s_6, c_6)$$

Bisher sind die Lösungen für alle Fälle von θ_1, θ_2, θ_3, θ_4, θ_5,und θ_6 abgebildet worden. Da es für alle drei Winkel von θ_1 , θ_3, und θ_5 zwei Lösungen gibt, können schließlich 8 Sätze von machbaren Lösungen kombiniert werden.

5.4.5 Roboterarm-Trajektorienplanung

Der Trajektorienplanungsteil der Bewegungsplanung konzentriert sich auf die Variation von Winkelverschiebung, Winkelgeschwindigkeit und Winkelbeschleunigung des Roboterarms während der Bewegung und die Auswirkung auf die Bewegung des Roboterarms. Das ultimative Ziel ist es, den Roboterarm in die Lage zu versetzen, eine bestimmte Aufgabe stabiler zu erfüllen. Die Trajektorienplanung ist eine praktische Anwendung kinematischer inverser Lösungen, dazu gibt es zwei Arten der Trajektorienplanung, basierend auf kubischem Polynom und basierend auf fünffacher Polynominterpolation. Wenn der Roboterarm von der Ausgangsposition [0, -π/2, 0, -π/2, 0, 0] zur Position [π/4, 0, π/5, π/2, - π/4, π/6] bewegt wird, ist die Endposition des Roboterarms in der **Abbildung 5-5** dargestellt.

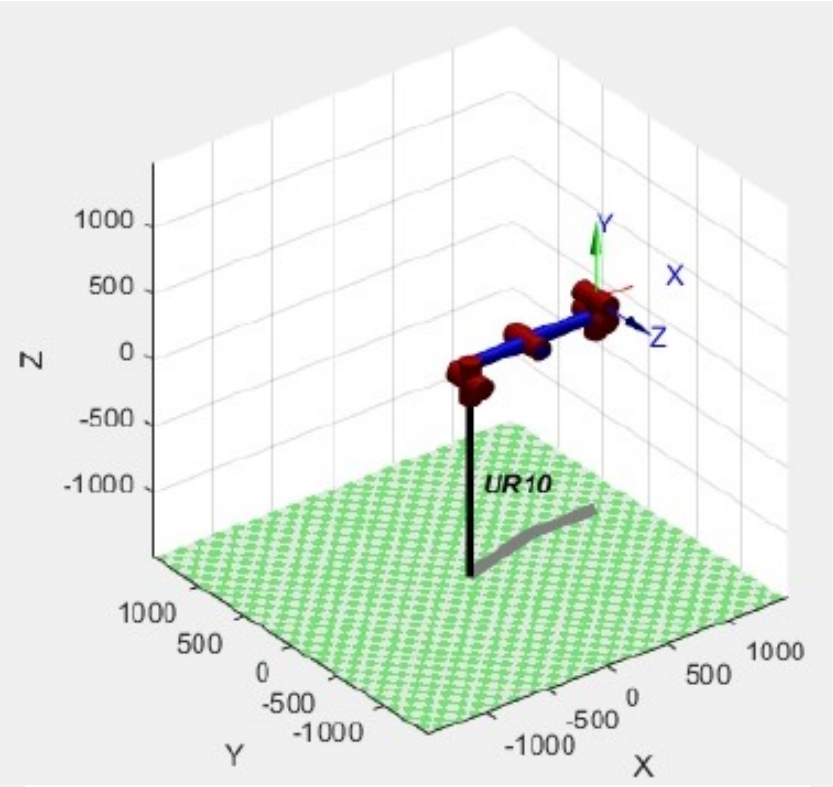


Abbildung 5-5 Variation der Endposition des Roboterarms

An diesem Punkt ändern sich die Winkelverschiebung, die Geschwindigkeit und die Beschleunigung der sechs Gelenke, wie in **Abbildung 5-6** gezeigt, nachdem die Trajektorie des Roboterarms geplant wurde.

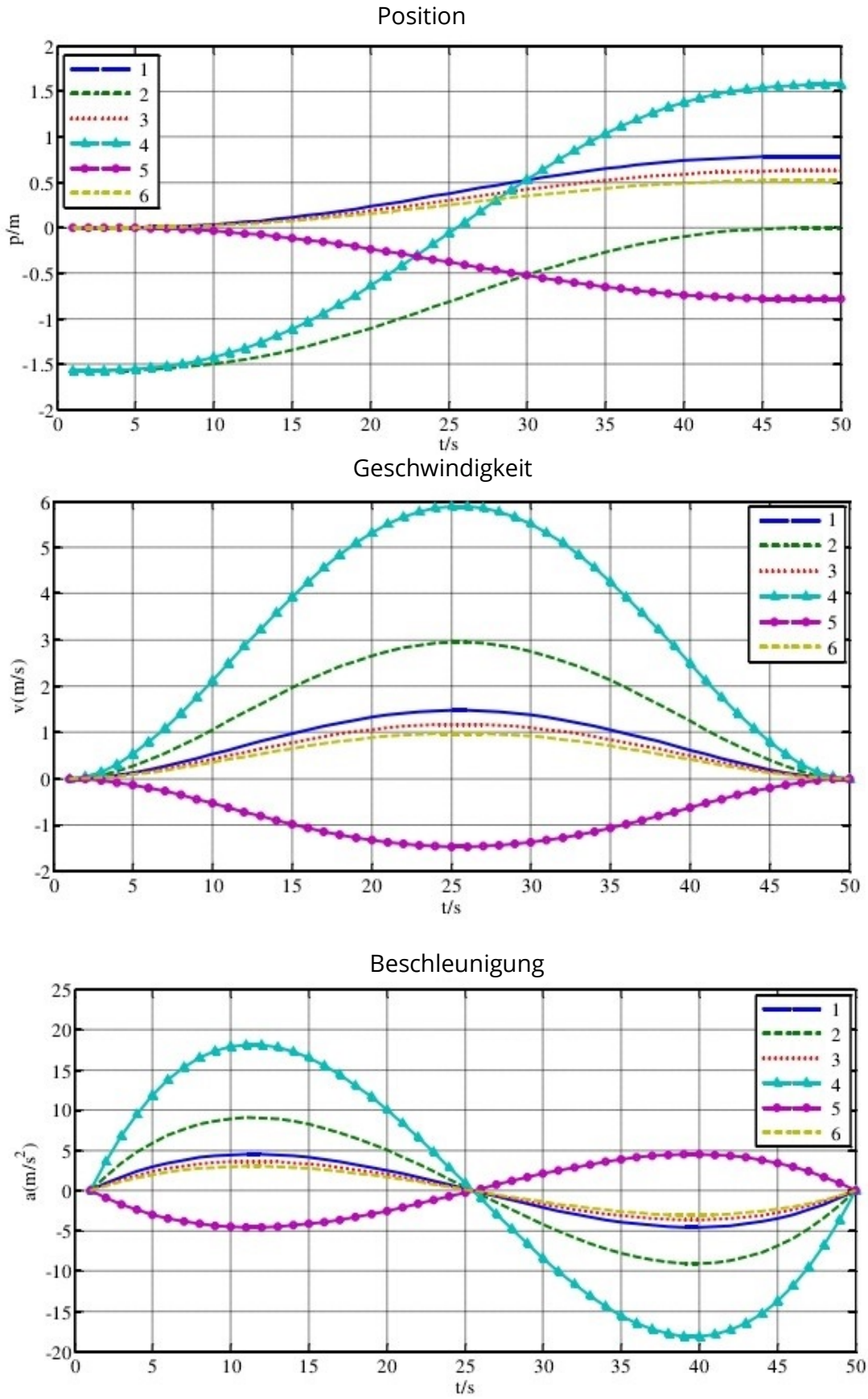


Abbildung 5-6 Positions-Geschwindigkeits-Beschleunigungs-Variationsdiagramm

5.5 Kennenlernen der neuen Roboterbeschreibungsdatei *.xacro

Bevor der Open-Source-Code importiert wird, benötigt man vorab die Kenntnis zweier verschiedener Beschreibungsdateien, **.urdf und *.xacro*. In der vorangegangenen Projektarbeit „Steuerung gängiger Industrieroboter mit ROS“ wurde aufgrund der Einfachheit des entworfenen Wagens nur die einfache URDF-Roboter-Modelldatei zur Konstruktion des simulierten Roboters verwendet. Obwohl die Eigenschaften und Positionsinformationen jedes einfachen Modells direkt in der gleichen Systemsprache XML dargestellt werden, hat es nur eine einzige Methode der Informationsdarstellung, kann nur Schritt für Schritt in Modulen darstellen, die zwischen den einzelnen Körpern durch Joint verbunden sind, kann nicht auf Makros verweisen und kann nicht die Körperparameter des gesamten Moduls definieren. Die Modelldatei *.urdf.xacro kann jedoch als fortgeschrittene Version der URDF-Modelldatei betrachtet werden, die sich durch die Möglichkeit auszeichnet, Modellcodes zu generieren, einschließlich der Möglichkeit, Makrodefinitionen zu erstellen, und Direktverweise auf homologe xacro-Modelldateien herzustellen. Außerdem bietet es programmierbare Schnittstellen wie Konstanten, Variablen und vor allem die Verwendung von mathematischen Berechnungen und bedingten Anweisungen.

Typischerweise besteht die *.urdf.xacro-Robotermodelldatei aus mindestens 2 Dateien, aufgeteilt in eine Body-Datei und eine Referenzdatei. Die Hauptkörperdatei ist in der Regel eine allgemeine Beschreibung jedes Moduls des Roboters, d. h., es handelt sich um eine XML-Sprachbeschreibung jedes Moduls des Roboters mit Hilfe von *xacro*, die sich auf die enthaltenen Funktionen beziehen können. Ein Teil des Codes ist unten dargestellt.

```
1.  <?xml version="1.0"?>
2.  <robot xmlns:xacro="http://wiki.ros.org/xacro" name="ur10" >
3.    <!-- common stuff -->
4.    <xacro:include filename="$(find ur_description)/urdf/common.gazebo.xacro" />
```

```
5.  <!-- ur10 -->
6.  <xacro:include filename="$(find ur_description)/urdf/ur10.urdf.xacro" />
7.  <!-- arm -->
8.  <xacro:ur10_robot prefix="" joint_limited="false"/>
9.  <!-- robotiq 85 -->
10. <xacro:include filename="$(find robotiq_85_description)/urdf/robotiq_85_gripper.urdf.xacro" />
11. <!-- gripper -->
12. <xacro:robotiq_85_gripper prefix="" parent="ee_link" >
13.   <origin xyz="0 0 0" rpy="0 0 0"/>
14. </xacro:robotiq_85_gripper>
15.   <!-- kinect -->
16. <xacro:include filename="$(find ur10_arm_tufts)/urdf/sensors/kinect/kinect.urdf.xacro" />
```

Die Zeilen 6, 8, 10 und 16 des Codes zeigen, dass diese Body-Datei 4 kleine Abschnitte enthält, die den Roboterarm, die Greifer und die Kinect mit dem Abschnitt für die Bilderfassung enthalten. Als nächstes folgt jeder Abschnitt, der von der Hauptdatei aufgerufen wird. Dieser ist relativ unabhängig und Änderungen an seinen eigenen internen Parametern stören die Body-Datei nicht. Am Anfang dieser Datei werden die Konstanten für den zu konstruierenden Roboter definiert, d. h. die grundlegenden Eigenschaften des Roboters, einschließlich der Länge der einzelnen Teile des Roboters, der Größe des Radius, der Farbe und so weiter. Diese Parameter werden an den Anfang der Datei gestellt, da die Werte der Parameter direkt geändert werden können, um das Debuggen und die Überprüfung des Modells zu erleichtern. Mit diesen Parametern kann das Modell während des restlichen Digitalisierungsprozesses direkt referenziert werden, was die Beschreibung des Roboters schneller und einfacher macht als eine normale *.urdf-Datei. Diese Änderung wirkt sich nicht auf jedes verknüpfte Teil des Roboters aus. Ein Teil des Codes wird unten gezeigt.

```
1.  <?xml version="1.0"?>
```

```
2.  <robot xmlns:xacro="http://wiki.ros.org/xacro">
3.   <xacro:include filename="$(find ur_description)/urdf/ur.transmission.x
    acro" />
4.   <xacro:include filename="$(find ur_description)/urdf/ur.gazebo.xacro" /
    >
5.   <xacro:macro name="ur10_robot" params="prefix joint_limited
6.        shoulder_pan_lower_limit:=${-3}    shoulder_pan_upper_limit:=${3}
7.        shoulder_lift_lower_limit:=${-pi}    shoulder_lift_upper_limit:=${pi}
8.        elbow_joint_lower_limit:=${-pi}    elbow_joint_upper_limit:=${pi}
9.         wrist_1_lower_limit:=${-pi}    wrist_1_upper_limit:=${pi}
10.        wrist_2_lower_limit:=${-pi}    wrist_2_upper_limit:=${pi}
11.        wrist_3_lower_limit:=${-pi}    wrist_3_upper_limit:=${pi}">
```

Ein weiterer wichtiger Grund, dass die xacro-Modelldatei in Include-Form vorliegt, ist, dass bei der Programmierung die am Roboterarm montierten Greifer und verschiedene Sensorgeräte, wie z. B. die Kinect, mitgegeben werden müssen. Diese Modelle können in einer separaten xacro-Datei verschachtelt werden, so dass diese direkt mit unserem Roboter kombiniert werden können.

5.6 Kennenlernen des Robot UR10e-Plugin

Nachdem der *UR10e* als kollaborativer Roboterarm für unsere Simulationsumgebung ausgewählt wurde, muss auch ein Robotergreifer bestimmt werden, um das Greifen und Platzieren von Objekten zu implementieren. Hier kann ein Plugin namens robotiq_85 ausgewählt werden.

5.6.1 Plugin von Robotiq_85

Robotiq_85 ist ein Funktionspaket, das die Montage einer Klaue an einem Roboterarm in einer ROS-Umgebung ermöglicht. Es enthält sowohl die Modelldatei für die Klaue als auch den Laufzeittreiber. Es kann im Simulator in der ROS-Umgebung geladen und erkannt werden. Das Robotiq-Plugin ist in zwei und drei Greifer unterteilt, in diesem Abschnitt wird Robotiq_85 verwendet, der eine Greifer mit zwei Greifer Armen ist.

Nachdem das Funktionspaket für dieses Plugin heruntergeladen und kompiliert wurde, lässt sich das Rviz-Mockup durch die Eingabe des folgenden Befehls öffnen und in der Ros-Umgebung in der Simulation ausführen.

```
1. source robotiq/devel/setup.bash
2. roslaunch robotiq_arg2f_model_visualization test_robotiq_arg_2f_85_model.launch
```

Dadurch wird RVIZ geöffnet und das elektrische Klauenmodell angezeigt.

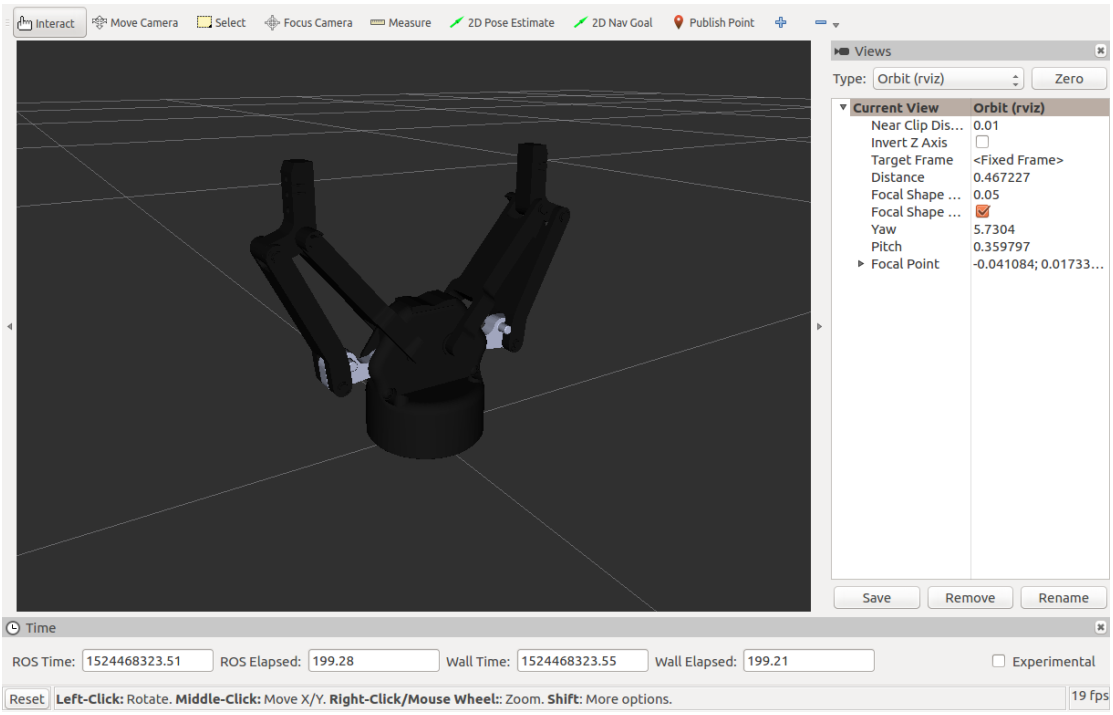


Abbildung 5-7 Robotiq_85 im Rviz

Als Nächstes muss die xacro-Datei für das Robotiq-Plugin durch Verweis auf die xacro-Datei zum Körper des Roboterarms hinzugefügt werden. Der Code des Verweises ist unten dargestellt.

```
1. <!-- robotiq 85 -->
2. <xacro:include filename="$(find robotiq_85_description)/urdf/robotiq_85_gripper.urdf.xacro" />
3. <!-- gripper -->
4. <xacro:robotiq_85_gripper prefix="" parent="ee_link" >
5.   <origin xyz="0 0 0" rpy="0 0 0"/>
6. </xacro:robotiq_85_gripper>
```

5.6.2 Funktionspaket von Gazebo-ros-control

Das Funktionspaket Gazebo-ros-control enthält zwei Bedienelemente, eines für den UR10-Roboterarm und das andere für den Zwei-Finger-Handgriff (*Robotiq_85*). Mit diesem Plugin wird beim Laden des Pavillons jeweils die entsprechende Steuerung gestartet. In der Launch-Datei von ur10 wird mit der folgenden Anweisung der entsprechende Controller gefunden.

```
1.  de file="$(find ur_gazebo)/launch/controller_utils.launch"/>
2.
3.   <rosparam file="$(find ur_gazebo)/controller/arm_controller_ur10.yaml
" command="load"/>
```

Der Controller für den ur10 ist der arm_controller_ur10.yaml aus dem offiziellen ur10-Paket und verwendet den *position_controllers/JointTrajectoryController* zur Steuerung des Gelenktrajektorien-Controllers. Die Steuerung der elektrischen Klaue wird mithilfe der Konfigurationsdatei *gripper_controller_robotiq.yaml* umgesetzt.

```
1.  <rosparam file="$(find robotiq_85_gazebo)/controller/gripper_controller
_robotiq.yaml" command="load"/>
2.
3.   <node name="arm_controller_spawner" pkg="controller_manager" typ
e="controller_manager" args="spawn arm_controller gripper" respawn="
false" output="screen"/>
```

Erst die Deklaration von ros-control ermöglicht hier die Ansteuerung von Arm und Handgriff, ähnlich dem Protokoll des Antriebs.

5.6.3 Grundstruktur von MoveIt!

MoveIt! ist eine einfach zu bedienende integrierte Entwicklungsplattform, ein leistungsstarkes ROS-Paket. MoveIt! bietet APIs für Python und C++, diese Schnittstellen decken fast alle Aspekte der Bewegungssteuerung ab, einschließlich Dynamik, Gelenksteuerung, Planungsszenen und Kollisionserkennung, Bewegungsplanung und 3D-Wahrnehmung. Außerdem bietet es eine benutzerfreundliche GUI-Schnittstelle für industrielle, kommerzielle, F&E- und andere Anwendungen. Seine Systemarchitektur ist in **Abbildung 5-8** dargestellt.

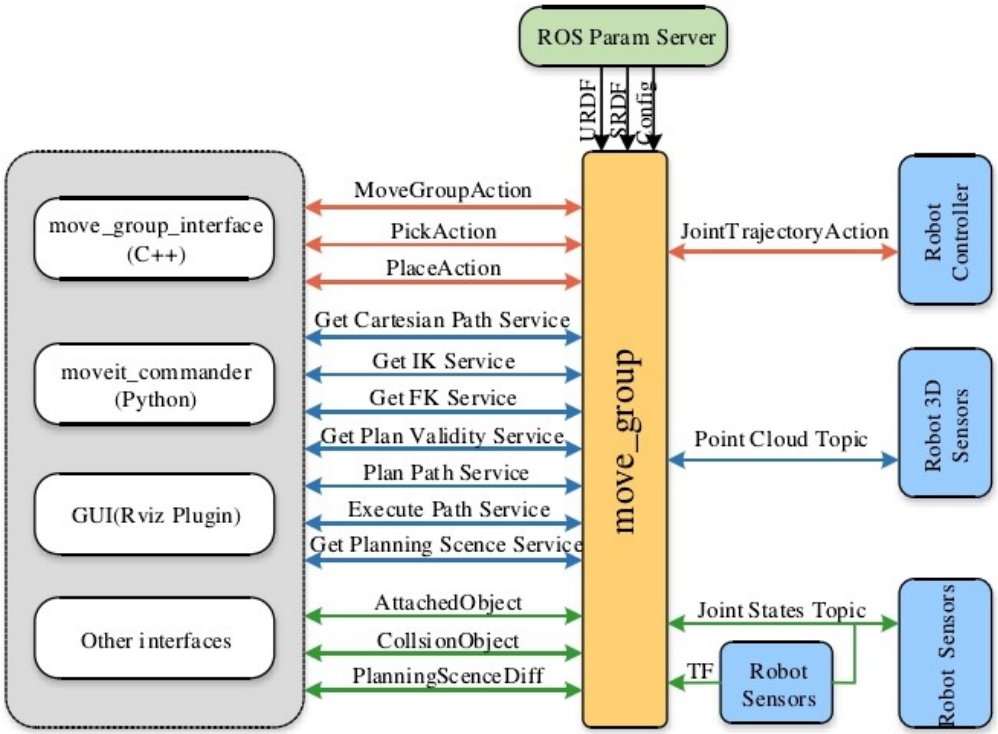


Abbildung 5-8 Systemarchitektur-Diagramm von MoveIt!

Das Kernstück von MoveIt! ist die *move_group*, deren Hauptaufgabe es ist, die 3D-Objektinformationen, die Echtzeit-Zustandsinformationen jedes Gelenks und den vom Roboter gesendeten tf-Baum mittels Nachrichten usw. zu erhalten. Gleichzeitig liefert das Parameter-Service-Paket von ROS die kinematischen Parameter des Roboterarms, bei der Anwendung des Antriebsassistenten werden die gelieferten Parameter aufgebaut, um sein Profil zu erhalten.

Bewegung Schwenken (Motion Panning): Der Bewegungsplanungsalgorithmus wird hauptsächlich durch das Motion-Planer-Plug-in durchgeführt, um die

Trajektorie zu planen. Zum Laden des gewünschten Bewegungsplaners kann die Planiglob-Schnittstelle im Roboterbetriebssystem verwendet werden.

Planungsszene (Planning Scene): Es ist ein abstraktes Modell, das den Zustand der Umgebung darstellt, es kann die Platzierung von Hindernissen und den Zustand des Roboterarms überwachen, es kann eine konkrete Arbeitsumgebung für den Roboterarm erstellen, indem es einige Hindernisse hinzufügt. Somit kann es zur Kollisionserkennung und Zwangsbedingungszuweisung verwendet werden. Alle geplanten Aktionen müssen eine bestehende Bewegungsgruppe angeben und innerhalb der angegebenen Planungsszene ablaufen.

Kinematik (Kinematics): Kinematik-Algorithmen sind das Herzstück der verschiedenen Algorithmen für den Roboterarm, einschließlich der FK-Lösung (Forward Kinematics) und des IK-Algorithmus (Inverse Kinematics). Das Standard-Plugin für die inverse Kinematik ist so konfiguriert, dass es KDL verwendet und typischerweise kann der Benutzer seinen eigenen Kinematik-Löser ausführen, der die vorwärts gerichteten und inversen Lösungen basierend auf dem angegebenen Gelenk oder Endzustand finden kann.

Kollisionsprüfung (Collision Checking): Bei der Verwendung von MoveIt! zur Steuerung des Roboterarms ist es erforderlich, dass der Roboterarm zunächst mit CollisionWorld eine Kollisionsprüfung durchführt, wobei in der Regel das FCL-Paket (Flexible Collision Library) ausgewählt wird. Der Hauptzweck dieser Funktion ist es, Kollisionen zwischen dem Arm und sich selbst oder seiner Umgebung zu vermeiden. Um Aktionen zu reduzieren und Verarbeitungszeit zu sparen, wird bei der Konfiguration von MoveIt! in der Regel eine kollisionsfreie Matrix eingestellt und die Kollisionsprüfung in einigen Gelenken, die immer oder nie kollidieren, ausgeschaltet.

Abschließend wird der Konfigurationsassistent gestartet, um ein neues Bewegungskonfigurationspaket zu erstellen, einschließlich der Konfiguration der kollisionsfreien Matrix; dem Hinzufügen virtueller Knoten, um den Roboterarm mit der Außenwelt zu verbinden; der Planung der Gruppe, der Definition des linken Arms, des rechten Arms, des Greifers usw.; der Pose des Roboters, der

Beschriftung der Endeffektoren; dem Hinzufügen passiver Gelenke, die der Planer nicht planen kann; und schließlich der Erzeugung der Konfigurationsdatei.

5.6.4 Konfiguration von MoveIt!

Das Robotermodell wird mit MoveIt! über die grafische Benutzeroberfläche MoveItSetupAssistant konfiguriert, dies ermöglicht die Steuerung des Robotermodells. Der Konfigurationsprozess von MoveItSetupAssistant ist in **Abbildung 5-9** dargestellt.

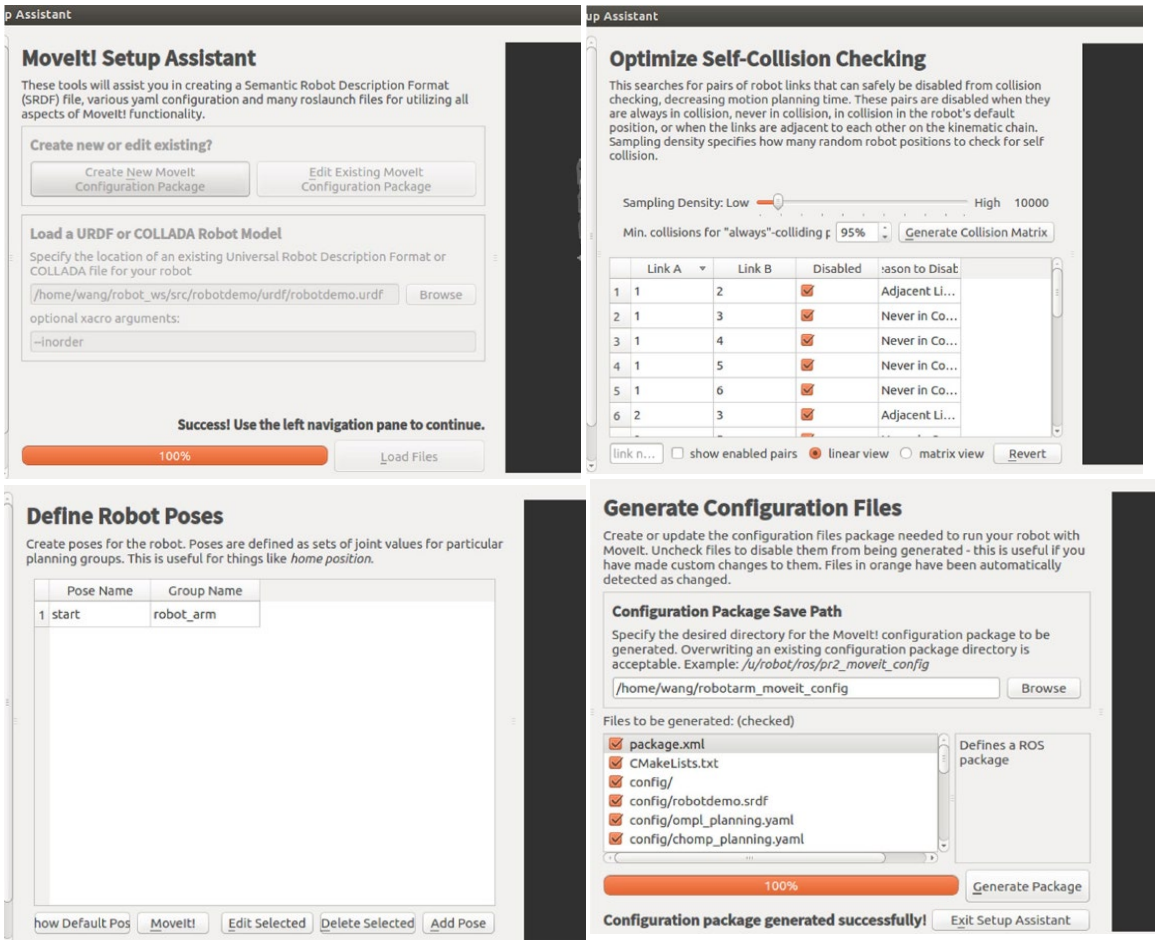


Abbildung 5-9 Setup-Assistent Konfigurationsprozess

Zuerst wird die urdf-Datei des Roboters geladen, dann das Kollisionserkennungsmodul eingerichtet, danach die Bibliothek für die Bewegungsplanungsberechnung, anschließend die Roboterplanungsgruppe, die Anfangsposition des Roboters und schließlich ein Funktionspaket für die

Konfigurationsdatei namens robotarm_moveit_config erstellt.

Wenn das obige Modell erfolgreich generiert wurde, wird der folgende Code ausgeführt, um die Modelldatei zu laden, die über den Simulator eingerichtet wurde.

```
1. roslaunch robotarm_moveit_config demo.launch
```

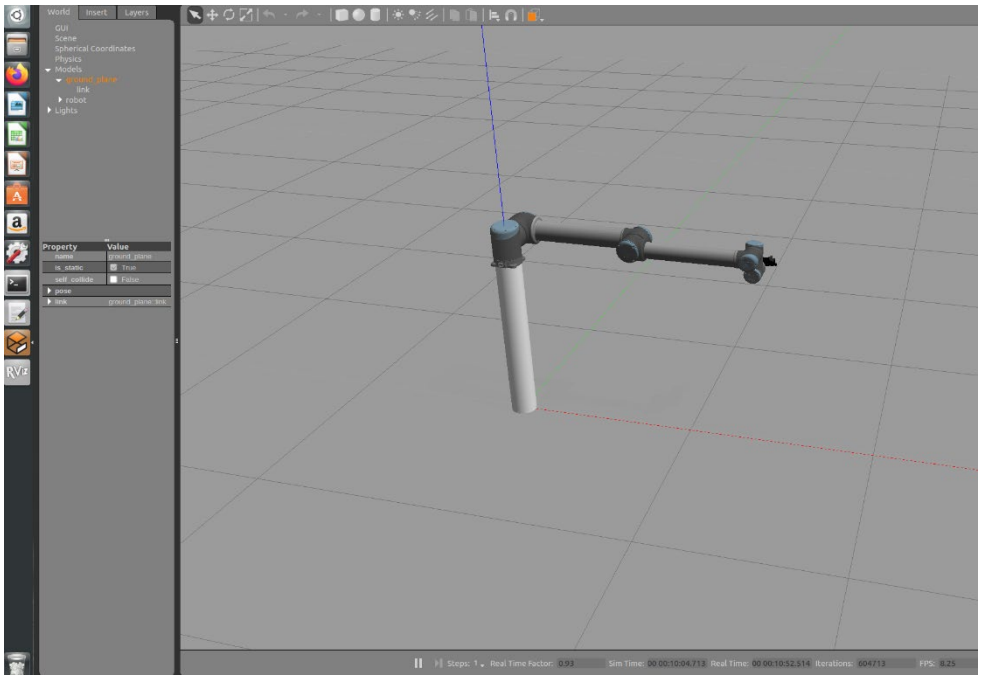


Abbildung 5-10 Roboteranzeige in RViz

5.6.5 Kinect

5.6.5.1 Tiefenkamera-Abbildungsmodell

Unter Tiefenkamerabildung wird der Prozess der Beschreibung von drei Dimensionen in zwei Dimensionen unter Verwendung einer Projektionstransformation verstanden. Der KinectV2-Sensor verwendet das Kleinbildverfahren. Das Abbildungsprinzip ist in **Abbildung 5-11** dargestellt, wobei sich O_C auf den Mittelpunkt der optischen Achse des Tiefensensors bezieht, A die Abbildungsebene bezeichnet, B die virtuelle Ebene, C die Ebene beschreibt, in der sich das Zielobjekt befindet, und das Bild des Zielpunkts PW auf A im Weltkoordinatensystem ein invertiertes reales Bild ist, die Größe des Bildes auf B gleich der auf A ist und die Richtung des ursprünglichen Ziels gleich ist. Das Koordinatensystem O_C — $X_CY_CZ_C$ wird mit der optischen Achse als Zentrum festgelegt, wobei die Z-Achse und die optische Achse des Sensors parallel gehalten werden.

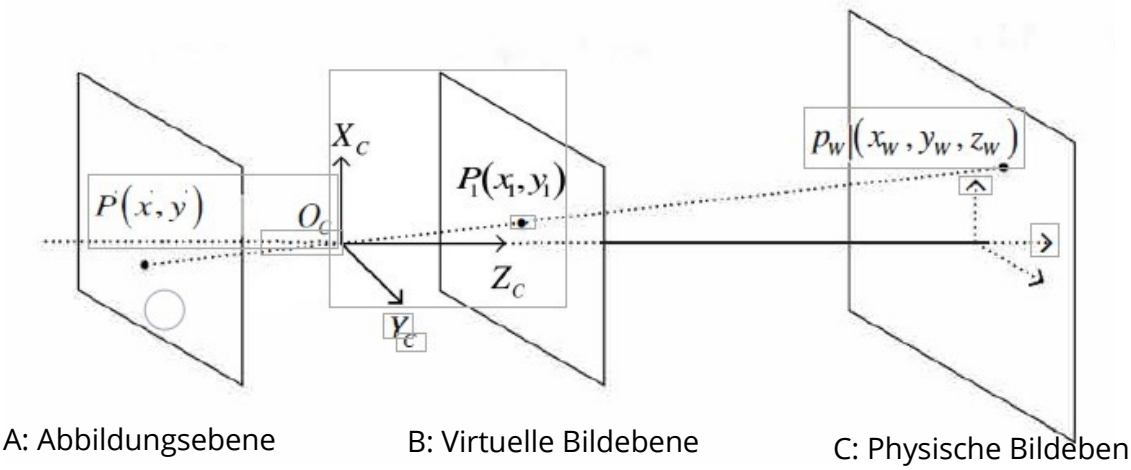


Abbildung 5-11 Schema der Kameraabbildung

5.6.5.2 Messprinzip der Tiefenkamera Vision

Unter einem Tiefenbild versteht man ein Bild, das durch den Abstand der Tiefensensorebene zu den Punkten im dreidimensionalen Raum, in dem sich der Sensor befindet, als Pixelwert gebildet wird. Um ein Tiefenbild im dreidimensionalen Raum zu erfassen, muss daher der Abstand von jedem Punkt in der Szene zur Kameraebene gemessen werden. Die Tiefenkameramessmethode wird mit der binokularen Kamera verglichen, die aktiv die Messung jedes Pixels der Tiefe vervollständigen kann. In diesem Projekt wird der Kinect V2 Tiefensensor mit TOF-Prinzip ausgewählt. Die TOF-Kamera hat einen ähnlichen Aufbau wie die binokulare Kamera, aber ihr Mess- und Abbildungsprinzip unterscheidet sich von dem der binokularen Kamera. Der Kinect V2-Sensor verwendet die *Time of Flight* (TOF)-Methode, um das Tiefenbild in der Szene zu erfassen, das TOF-Prinzip ist in **Abbildung 5-12** dargestellt. Die zentrale Idee des Prinzips ist die Messung der Zeitdifferenz zwischen der Laufzeit des Lichts, also dem Zeitpunkt, zu dem das Licht vom Sender ausgesendet wird, und dem Zeitpunkt, zu dem das Licht vom Empfänger empfangen wird.

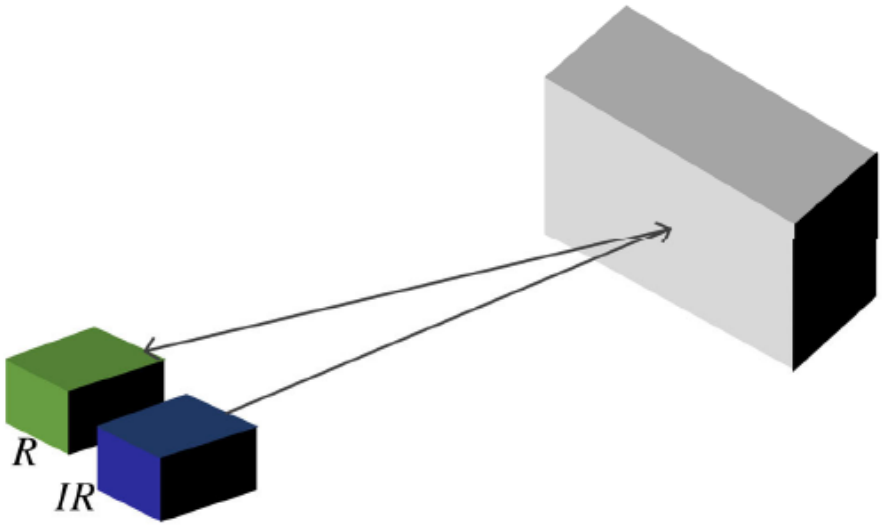


Abbildung 5-12 Kinect V2 Kamera Messschema

Wenn der Kinect V2-Sensor eine Tiefenkarte erzeugt, erhält jeder Pixel ein Messsignal, mit dem der Abstand vom Zielobjektpunkt zum Tiefensensor gemessen werden kann. Wenn der Abstand an diesem Punkt auf d gesetzt wird, kann die Gleichung wie folgt erhalten werden.

$$d = \frac{1}{2}\Delta t \times c = \frac{1}{2}\Delta\varphi \times \frac{1}{f} \times c$$

Δt oben bezieht sich auf die Laufzeit des Lichts, $\Delta\varphi$ bezieht sich auf die Größe der Phasendifferenz zwischen Sender und Empfänger des Lichts, f stellt die Laserfrequenz dar und c beschreibt die Ausbreitungsgeschwindigkeit des Lichts.

5.6.5.3 Umrechnung des Koordinatensystems

Es besteht eine lineare Transformationsverbindung zwischen den 3D-Koordinaten des Objekts und den Bildkoordinaten, die hauptsächlich vier Koordinatensysteme umfasst: Welt, Bild, Kamera und Pixel. Die Beziehung zwischen den vier Koordinaten ist in **Abbildung 5-13** dargestellt. Ein Objektpunkt in der realen Welt hat die Koordinaten P(X,Y,Z) im Weltkoordinatensystem, und nachdem die Kamera das Bild aufgenommen hat, sind die Pixelkoordinaten auf dem Bild m(u,v).

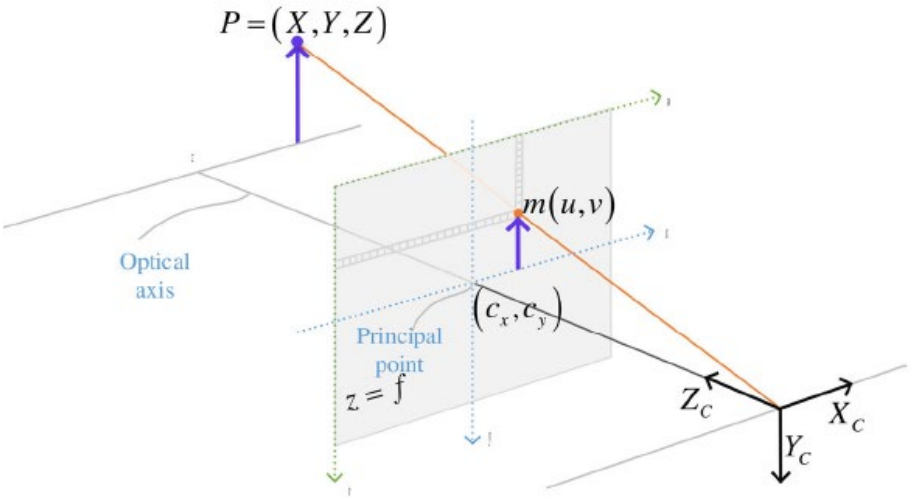


Abbildung 5-13 Koordinatensystem-Beziehungsdiagramm

Das Weltkoordinatensystem wird verwendet, um die Position des in dieser dreidimensionalen Umgebung platzierten Objekts zu beschreiben, und die Koordinaten werden in (X,Y,Z) ausgedrückt, die zur Charakterisierung eines beliebigen Objekts im dreidimensionalen Raum verwendet werden können. Die Position des Tiefensensors relativ zu den Weltkoordinaten ist durch (Xc,Yc,Zc) gekennzeichnet, die Ursprungsposition des Tiefensensor-Koordinatensystems ist das optische Zentrum, und die optische Achse ist die Z-Achse. Die Koordinatenwerte des Bildkoordinatensystems werden durch (x, y) dargestellt

und der Mittelpunkt der Kameraebene (cx, cy) wird als Ursprung genommen Die Koordinatenwerte des Pixelkoordinatensystems werden durch (u, v) dargestellt, es wird der obere linke Punkt des Bildes als Ursprungsposition genommen. Dabei sind die u- und v-Achsen parallel zu den x- und y-Achsen. Die Umrechnung zwischen den Pixelkoordinaten und dem Bildkoordinatensystem ist in **Abbildung 5-14** dargestellt. In der Abbildung bezieht sich (u_0,v_0) auf die Position in Pixelkoordinaten.

Die Transformationsgleichung für Bild- und Pixelkoordinaten kann aus **Abbildung**

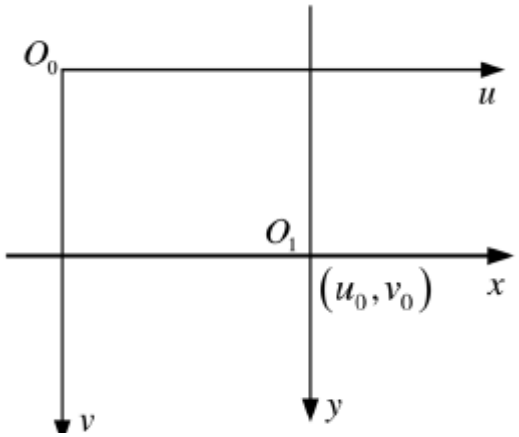


Abbildung 5-14 den Pixelkoordinaten und dem Bildkoordinatensystem

5-14 wie folgt ermittelt werden.

$$\begin{cases} u = \frac{x}{dx} + u_0 \\ v = \frac{y}{dy} + v_0 \end{cases}$$

Dabei werden dx und dy als Maßänderungen in Richtung der x-Achse bzw. der y-Achse dargestellt. Daher wird die Chi-Quadrat-Transformation wie folgt durch Gleichung ausgedrückt.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Die Umrechnungsbeziehung zwischen dem Bildkoordinatensystem und dem Kamerakoordinatensystem ist unten dargestellt.

$$\begin{cases} \frac{x}{f} = \frac{X_c}{Z_c} \\ \frac{y}{f} = \frac{Y_c}{Z_c} \end{cases}$$

Dabei ist f die Brennweite und die Sekundärtransformation ergibt eine Matrixdarstellung wie folgt:

$$Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

Die Gleichung für die Umrechnung der Koordinaten des Kinect V2-Tiefensensors in Weltkoordinaten ist unten dargestellt. Dabei steht R für die 3 × 3 orthogonale Rotationsmatrix und t bezieht sich auf den 3D-Translationsvektor.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Durch Einsetzen der obigen Gleichung ergibt sich die Beziehung zwischen dem Pixelkoordinatensystem und dem Weltkoordinatensystem wie folgt:

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Einschließend:

$$\begin{cases} f_x = \dfrac{f}{dx} \\ f_y = \dfrac{f}{dy} \end{cases}$$

5.6.5.4 Target-Matching-Methode basierend auf Bildmerkmalen

Da es im Allgemeinen Merkmalspunkte des Zielobjekts gibt, die z. B. die zentrale Position des Objekts in Bezug auf die Eckpunkte sind, werden die Merkmale des Zielobjekts aus dem Bild des Zielobjekts extrahiert und dann werden die Merkmale verwendet, um das vorhandene Bild des Zielobjekts abzugleichen, was der auf Bildmerkmalen basierende Algorithmus für den Zielabgleich ist. In dieser Arbeit werden Punktmerkmale für die Untersuchung verwendet, wobei andere Kantenmerkmale sowie Regionsmerkmale nicht weiter diskutiert werden können.

Der Moravec-Algorithmus ist einer der gebräuchlichsten Algorithmen für den Abgleich von Punktmerkmalen in Bildern und einer der frühesten Algorithmen zur Erkennung von Eckpunkten. Er wandelt das Originalbild in ein Graustufenbild um, extrahiert die Merkmalspunkte durch die Eckpunkte in der Graustufenkarte, die auch als polare Punkte bekannt sind, vergleicht dann die Ähnlichkeit der Merkmalspunkte und damit den Grad der Übereinstimmung. Allerdings ist das Bildrauschen der Hauptgrund, der die Genauigkeit des Moravec-Algorithmus beeinträchtigt und es ist relativ schwierig, die Domäne für den Moravec-Algorithmus auszuwählen.

5.6.5.5 Die Funktionsweise von OpenCV in dieser Arbeit

Erstens wird das Bildthema durch Kinect abonniert und die 2d-Position des zu erkennenden Objekts erhalten. Daraufhin wird die Beziehung zwischen der Position des Schreibtisches und der Position des Objekts durch die eingestellten Kinect-Koordinaten berechnet, was eine gute Grundlage für die Koordinatenumrechnung ist, die als nächstes durchgeführt werden muss.

Dann werden das von der Kinect-Kamera erzeugte Foto und die Formatkonvertierung durch das OpenCV-Plugin und die Farbgamut-

Segmentierung vorgenommen, um die Bildpixel in die drei Kanäle R-G-B zu trennen. Um im Anschluss das OpenCV in dem Plug-in-Glättungsfilter zu erhalten RGB drei Kanäle von Daten. Abhängig von der Farbe des Objekts wird eine Farbskala-Schwelle verwendet, um schwarze Tischplatten mit verschiedenfarbigen Objekten zu erkennen und die entsprechenden Positionsinformationen zu erhalten. Schließlich wird eine TF-Koordinatenumrechnung durchgeführt und die Koordinaten werden an den Roboterarm ausgegeben.

Der oben beschriebene Prozess und das Prinzip der Objekterkennung ist die Funktionsweise des Vision-Teils der Greifaktion des Roboterarms.

5.7 Modellierung der Umgebung in Gazebo

Nun muss die Betriebsumgebung des Roboters in der Gazebo-Software eingerichtet werden, die die Startposition des gelieferten Gegenstands, die Zielpunktposition und die Einrichtung von Hindernissen in seinem Weg umfasst.

5.7.1 Einrichtung der Standortumgebung in Gazebo

Gazebo ist eine professionelle Modellsimulations-Demonstrationssoftware, die eine Funktion namens Feldeditor enthält, mit der das Feld, auf dem sich der Simulationsroboter befindet, bearbeitet werden kann.

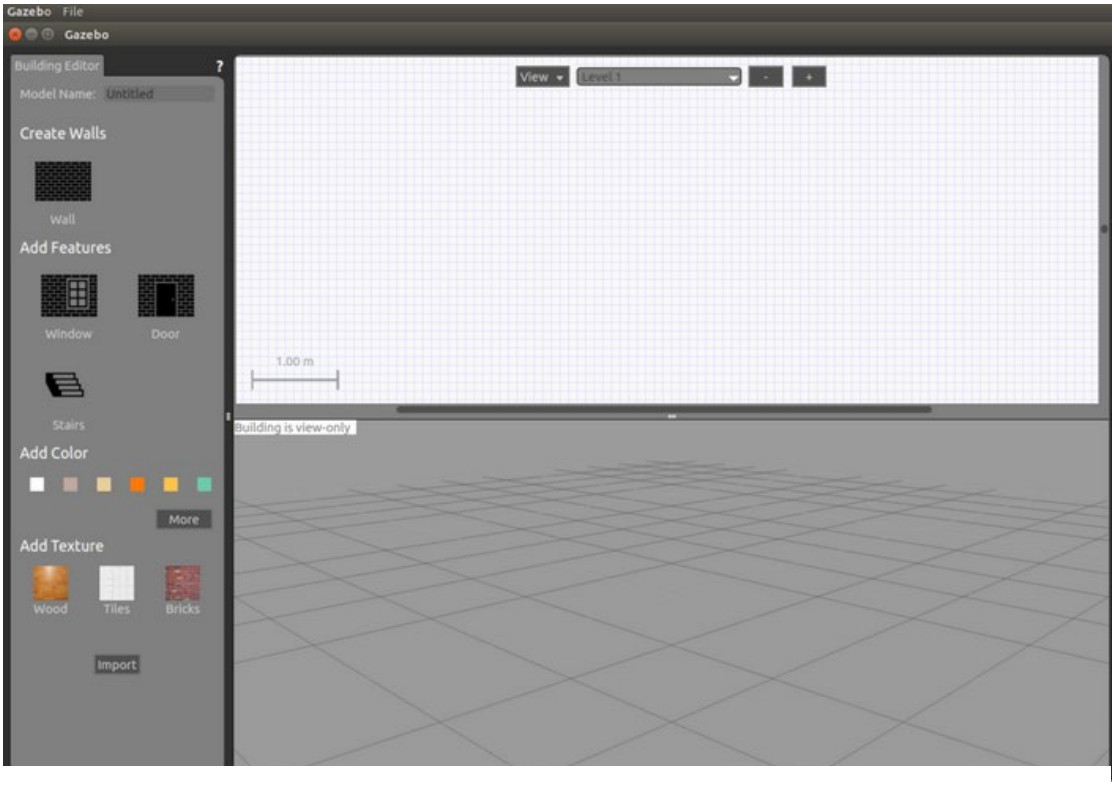


Abbildung 5-15 Szeneneditors im Gazebo

Wie in der **Abbildung 5-15** unten dargestellt, ist dies das Initialisierungsmodell des Szeneneditors in Gazebo.

Der Editor besteht aus den folgenden 3 Bereichen:

a) Palette

In diesem Bereich kann eine Auswahl an Gebäudeeigenschaften und Materialien getroffen werden.

b) 2D-Ansicht

In diesem Fenster kann ein Grundriss importiert oder gezeichnet werden und der Editor fügt automatisch Wände, Fenster, Türen und Treppen anhand des Plans ein.

c) 3D-Ansicht

Auf dieser Seite wird eine Vorschau auf Ihr Gebäude angezeigt. Außerdem kann der Benutzer den verschiedenen Gebäudeteilen darin Farben und Texturen zuweisen.

Da in komplexen Geländesituationen keine Untersuchungen erforderlich sind, reicht es aus, ein einfaches Hausmodell in den Simulator einzufügen, das in der Lage ist, beide Roboter gleichzeitig zu bedienen.

Wie in **Abbildung 5-16** unten gezeigt, handelt es sich um ein vorläufiges Modell, das in der Lage ist, zwei Roboter gleichzeitig zu beladen, wobei der Primärroboter UR10, zur gleichen Zeit wie der Sekundärroboter TB2 in der Umgebung vorhanden ist.

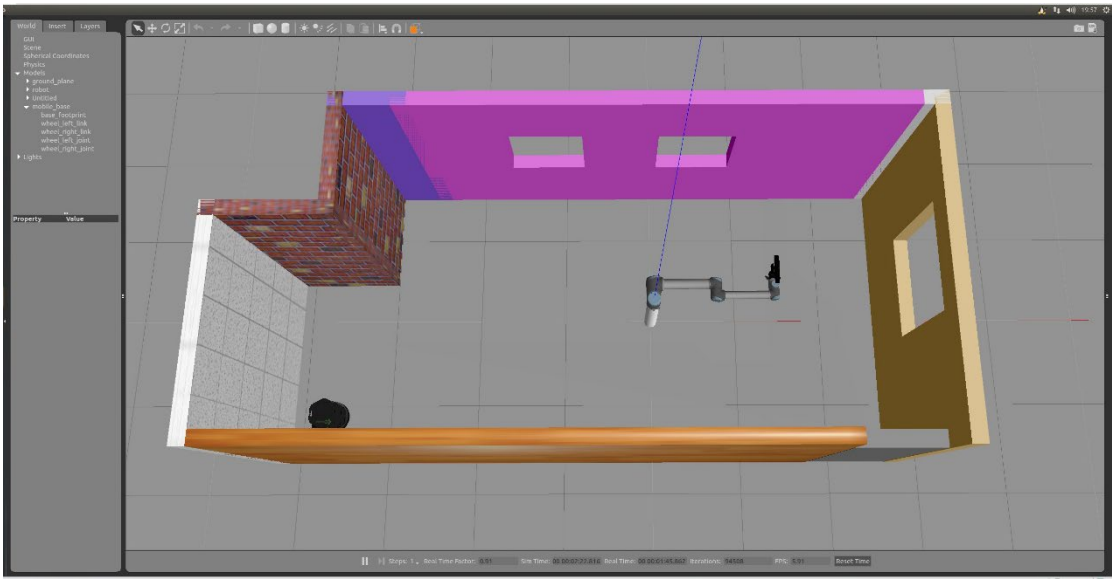


Abbildung 5-16 Robotern im Simulator

5.8 Navigationssimulation des Sekundärroboter TurtleBot

Für diesen Roboter ist es erforderlich, dass er in eine definierte Position fahren kann und in der Lage ist, Objekte, die vom Roboterarm gegriffen werden, anzunehmen und zurück in die Ausgangsposition zu transportieren, wie in Abschnitt 5.3 beschrieben.

Navigation und Lokalisierung sind ein wichtiger Teil der Robotik-Forschung. Im Allgemeinen müssen Roboter in unbekannten Umgebungen Lasersensoren (oder in Laserdaten umgewandelte Tiefensensoren) verwenden. In solchen Fällen wird zunächst die Karte modelliert und dann die Navigation und Lokalisierung auf der Grundlage des erstellten Kartenmodells durchgeführt.

Im Allgemeinen sind die drei häufigsten Funktionspakete, die zur Navigation in der ROS-Plattform verwendet werden, angegeben.

1. move_base: Bahnplanung basierend auf der referenzierten Nachricht, um den mobilen Roboter an die angegebene Position zu bringen.

2. GMapping: Erstellen von Karten aus Laserdaten (oder aus Tiefendaten simulierten Laserdaten).

3. AMCL: Positionierung auf der Grundlage bereits vorhandener Karten.

5.8.1 Übersicht über das GMapping Funktionspaket

Wenn ein mobiler Roboter in einer unbekannten Umgebung navigiert, empfängt er Informationen von Wegstreckenzählern, Sensorströmen und konstruiert ein Kartenmodell der Arbeitsumgebung und navigiert und ortet dann basierend auf der Karte und den Positionsinformationen, das Ziel.

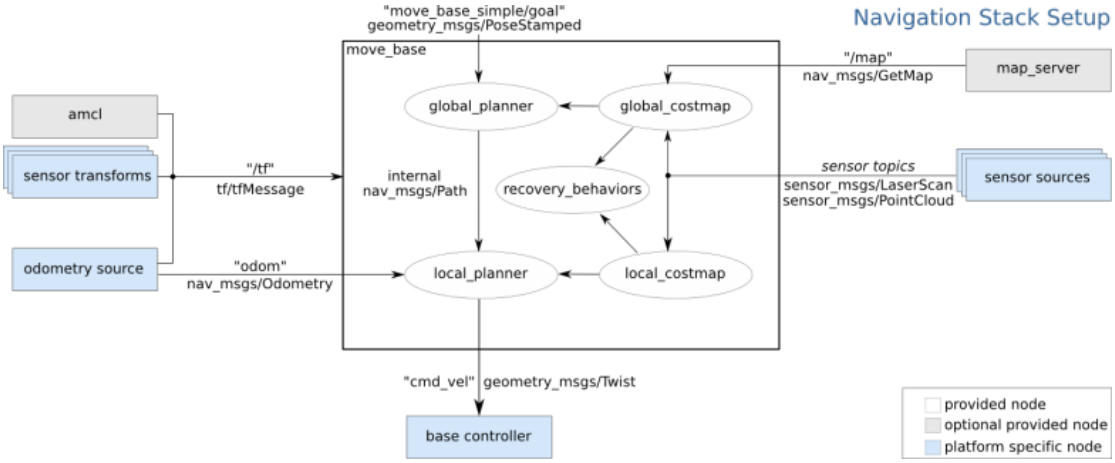


Abbildung 5-17 Gmapping Funktionspaket

Das ROS-Navigationsfunktionspaket ist eines der leistungsfähigsten Funktionen des ROS-Systems und ermöglicht die autonome Navigation und Bewegung des Roboters, kann aber nur für Roboter mit Zweirad-Differentialantrieb oder Vollradantrieb verwendet werden. Das Navigationsfunktionspaket setzt voraus, dass der Roboter für den Betrieb auf eine bestimmte Weise konfiguriert ist. Die folgende **Abbildung 5-17** zeigt eine Übersicht über diese Konfiguration. Die weißen Komponenten sind diejenigen, die für die Navigation notwendig sind und von ROS bereitgestellt werden, die grauen Komponenten sind optionale Komponenten, die von ROS bereitgestellt werden und die blauen Komponenten sind diejenigen, die beim Entwurf jedes Roboters erstellt werden müssen.

Bei der Konstruktion von Robotern müssen die folgenden 4 Punkte beachtet werden:

1. Der Roboter soll Informationen über die Beziehung aller Gelenke und Sensorpositionen (tf) senden.

2. Der Roboter soll Informationen über Linear- und Winkelgeschwindigkeiten senden (odom).

3. Der Roboter soll Informationen vom LIDAR erhalten, um den Kartenaufbau und die Lokalisierung zu vervollständigen (Sensorquellen).

4. Die Basissteuerung des Roboters muss erstellt werden, die für die Veröffentlichung der Liniengeschwindigkeit und der Lenkwinkelinformationen an die Hardwareplattform (Basissteuerung) verantwortlich ist.

Das GMapping-Paket verwendet einen Partikelfilter-Algorithmus, um laserbasiertes SLAM (gleichzeitige Lokalisierung und Kartenerstellung) mit einem Knoten namens *slam_gmapping* zu ermöglichen. Mit *slam_gmapping* kann eine 2D-Rasterkarte aus den vom mobilen Roboter erfassten Laser- und Positionsbeziehungsdaten erstellt werden. Das heißt, die Topics *tf (tf/tfMessage)* und *scan (sensor_msgs/LaserScan)* werden abonniert, und die Topics *map_metadata (nav_msgs/MapMetaData)* und *map (nav_msgs/OccupancyGrid)* werden veröffentlicht.

Nachdem TurtleBot erfolgreich in der Simulationsumgebung platziert wurde, kann der folgende Befehl ausgeführt werden, um die Struktur des TF-Koordinatentransformationsbaums zu zeigen:

```
1. rosrun tf view_frames
```

Unten ist ein Beispielbild(**Abb.5-18**), das einen Teil des TF-Baums zeigt.

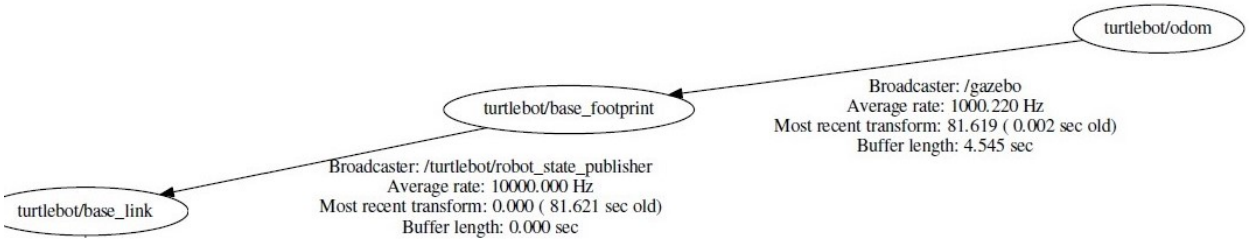


Abbildung 5-18 TF-Baum

5.8.2 Trajektorie Planung (move_base)

Wie in **Abbildung 5-17** zu sehen ist, stellt das Funktionspaket *move_base* die Schnittstelle für die Konfiguration, Bedienung und Interaktion der ROS-Navigation zur Verfügung, die, wie unten gezeigt aus zwei Hauptteilen besteht.

1. Globaler Planer: die Planung der Gesamtstrecke auf der Grundlage eines vorgegebenen Zielorts.

2. Lokaler Planer: Ausweichroutenplanung basierend auf nahegelegenen Hindernissen.

Da der Roboter unter den aktuellen Bedingungen zu einem festen Ort fahren muss, um die vom Roboterarm gegriffenen Güter aufzunehmen, kann davon ausgegangen werden, dass der aktuelle Umgebungsort bekannt ist. Es muss also nur die Lokalisierung des mobilen Roboters in der 2d-Umgebung durchgeführt werden. Daher muss das vom ROS-System bereitgestellte AMCL-Paket verwendet werden, um das Ziel zu erreichen.

5.8.3 Positionierung (AMCL)

AMCL ist ein probabilistisches Lokalisierungssystem für mobile Roboter in einer 2D-Umgebung. Es implementiert eine adaptive (oder KLD-Sampling basierte) Monte-Carlo-Lokalisierungsmethode, die Partikelfilter verwendet, um die Posen des Roboters entlang einer bekannten Karte zu verfolgen. Im aktuellen Fall arbeitet der Knoten nur mit Laserscan-Daten und LIDAR-Karten. Amcl kann durch Modifikation des Codes auf andere Sensordaten erweitert werden. In praktischen Anwendungen wird oft gewünscht, dass Roboter in der Lage sind, autonom und ohne menschliches Zutun zu orten und zu navigieren, um intelligenter nutzbar zu sein. In dieser Arbeit werden das Anfangsziel des Sekundärroboters und die Koordinaten des Zielpunktes in der Karte angegeben, dann navigiert der Roboter automatisch zum Ziel und baut eine übersichtliche Karte mit der AMCL-Funktion auf.

Der AMCL empfängt laserbasierte Karten, Laserscan- und tf-Transform-

Nachrichten und gibt Bit-Pose-Schätzungen aus. Beim Start initialisiert AMCL seinen Partikelfilter gemäß den angegebenen Parametern. Hinweis: Wenn keine Initialisierungsparameter eingestellt sind, ist der anfängliche Filterzustand eine mittelgroße Partikelwolke, die gemäß den Standardwerten bei (0,0,0) zentriert ist. Die Bit-Pose kann in rviz mit der Schaltfläche 2D Pose-Schätzung initialisiert werden.

6 Programmierung und Implementierung

Nachdem die entsprechende Programmierumgebung eingerichtet wurde, d. h. nachdem die angemessene Version der Ros-Plattform auf dem Linux-System installiert wurde, wird zunächst das Robotermodell über die Launch-Datei geladen.

6.1 Launch- Gesamtdokument

6.1.1 Launch-Datei über Modelle und Umgebungen

An diesem Punkt sind die Launch-Dateien in drei Teile aufgeteilt, da die Simulation in Gazebo und Rviz gleichzeitig abläuft und beide Roboter über die Launch-Datei geladen werden müssen. Bevor dies möglich ist, muss eine funktionierende Simulation der Baustelle in der Simulationsumgebung erstellt werden. Dazu muss die Simulationswelt, die bereits erstellt wurde, über die XML-Sprache zur Startdatei hinzugefügt werden.

Der erste Teil der Launch-Datei enthält also folgende Abschnitte: erstens das Starten der Gazebo-Simulationswelt, zweitens das Laden der Roboterarm-Modelldatei und drittens das Laden der Gelenk- und Greifers Teuerung.

Beim Laden dieser Dateien muss auch die Konfigurationsdatei für das Roboterarmmodell, d.h. die *.yaml-Datei, eingelesen werden. Mit diesen Dateien ist es möglich, die Ausgangsposition des Roboterarms, die Ausgangsstellung und die Form der elektrischen Klaue festzulegen. Diese Datei wird bei der Initialisierung des Modells durch MoveIT erzeugt.

```
1. <launch>
2.  <arg name="paused" default="false"/>
3. .....
4.  <arg name="world_file"  default="/home/robot/building_editor_models/
   Untitled/2.world"/>
```

 # Starten der Pavillon-Simulationswelt -->

```
5.  <include file="$(find gazebo_ros)/launch/empty_world.launch">
6.  ...
7.  </include>
8.
```

#Laden der Modelldatei des Roboterarms

```
9.  <param name="robot_description" command="$(find xacro)/xacro --ino
    rder '$(find ur10_arm_tufts)/urdf/ur10_arm.urdf.xacro'"/>
10. ...
11. <rosparam file="$(find robotiq_85_gazebo)/controller/gripper_controlle
    r_robotiq.yaml" command="load"/>
12.
```

#Ladegelenksteuerung Klemmbackensteuerung

```
13. <node name="arm_controller_spawner" pkg="controller_manager" typ
    e="controller_manager" args="spawn arm_controller" respawn="false" o
    utput="screen"/>
14. <node name="gripper_controller_spawner" pkg="controller_manager" t
    ype="spawner" args="gripper --shutdown-timeout 0.5" />
15. <include file="$(find ur10_arm_moveit_config)/launch/ur10_moveit_pla
    nning_execution.launch">
16.   <arg name="debug" default="$(arg debug)" />
17.   <arg name="sim" default="$(arg sim)" />
18. </include>
19.
20. <include file="$(find ur10_arm_moveit_config)/launch/moveit_rviz.launc
    h">
21. ...
22. </include>
23. ...
24. </launch>
```

6.1.2 Launch-Datei zum Aufrufen des Grip-Servers (Grip-Erkennung)

Diese Launch-Datei ist ein Abonnement des Dienstes, der die Greifer steuert, die in dem Funktionspaket *oveit_simple_graspsp* enthalten sind. Hier sind einige

Parameter des gebauten Roboters zu ändern, darunter der Robotername und die yaml-Parameterdatei, die auf den Roboterarm und die Greifer Einheit verweist. Die geänderte Konfigurationsdatei für *grasp_generator_server.launch* ist unten dargestellt.

```
1.  <launch>
2.   <arg name="robot" default="ur10"/>
3.   <arg name="group"       default="arm"/>
4.   <arg name="end_effector" default="gripper"/>
5.   <node pkg="moveit_simple_grasps" type="moveit_simple_grasps_serve
     r" name="moveit_simple_grasps_server">
6.    <param name="group"       value="$(arg group)"/>
7.    <param name="end_effector" value="$(arg end_effector)"/>
8.    <rosparam command="load" file="$(find arm_manipulation)/config/$(
     arg robot)_grasp_data.yaml"/>
9.   </node>
10. </launch>
```

6.2 Virtuelle Identifikation von Roboterarmen

6.2.1 Grundlagen der OpenCV von Objekten–Merkmalsextraktion

Die Merkmalsextraktion ist ein Konzept der Computer Vision und der Bildverarbeitung. Es handelt sich um die Verwendung eines Computers, um Bildinformationen zu extrahieren und zu entscheiden, ob die Punkte eines jeden Bildes zu einem Bildmerkmal gehören. Bei der Merkmalsextraktion werden die Punkte eines Bildes in verschiedene Teilmengen unterteilt, die oft zu isolierten Punkten, kontinuierlichen Kurven oder kontinuierlichen Regionen gehören.

6.2.2 Definition von Merkmalen

Bislang gibt es keine allgemeingültige und präzise Definition eines Merkmals. Die genaue Definition eines Merkmals wird oft durch das Problem oder die Art der Anwendung bestimmt. Merkmale sind der "interessante" Teil eines digitalen

Bildes und bilden den Ausgangspunkt für viele Algorithmen zur Computerbildanalyse. Der Erfolg eines Algorithmus wird daher häufig durch die von ihm verwendeten und definierten Merkmale bestimmt. Eine der wichtigsten Eigenschaften der Merkmalsextraktion ist daher die "Wiederholbarkeit": Die extrahierten Merkmale sollten für verschiedene Bilder der gleichen Szene gleich sein.

Die Merkmalsextraktion ist eine primäre Operation in der Bildverarbeitung, das heißt, sie ist der erste operative Prozess, der an einem Bild durchgeführt wird. Es untersucht jeder Pixel, um festzustellen, ob dieser Pixel ein Merkmal darstellt. Wenn er Teil eines größeren Algorithmus ist, dann untersucht der Algorithmus im Allgemeinen nur die Merkmalsbereiche des Bildes. Als Vorbedingung für die Merkmalsextraktion wird das Eingangsbild im Allgemeinen mit Hilfe eines Gaußschen Unschärfekerns im Skalenraum geglättet. Danach werden ein oder mehrere Merkmale des Bildes mit Hilfe einer partiellen Ableitungsoperation berechnet.

6.2.3 Funktionalität und Implementierungsmethoden

Sobald die Simulationsumgebung eingerichtet ist, müssen im folgenden Schritt die entsprechenden Kontroll- und Serviceknoten ergänzt werden. Bevor die

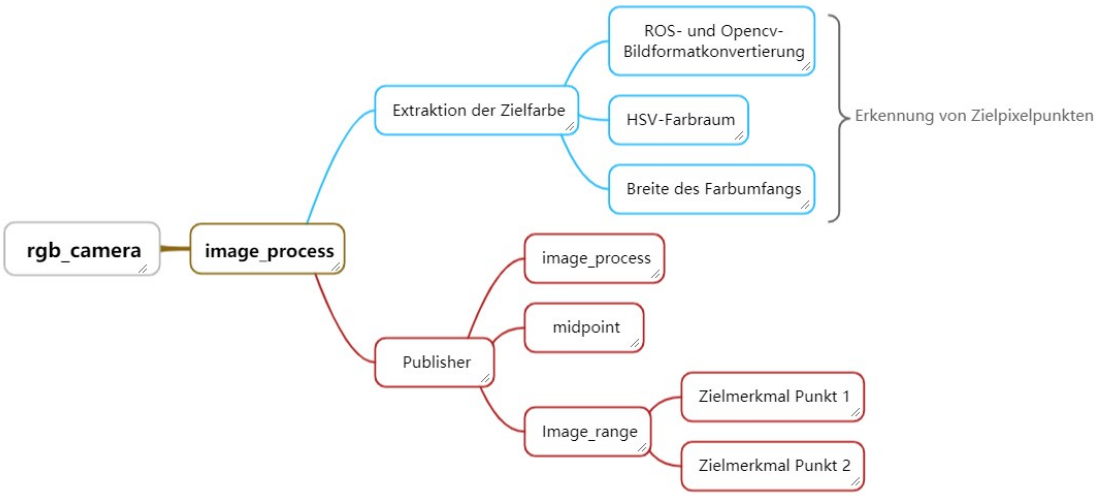


Abbildung 6-1 Flussdiagramm des Bildverarbeitungsknotens

entsprechenden Knoten geschrieben werden, ist es wichtig, die Funktion jedes

Knotens und die logische Beziehung zwischen den einzelnen Knoten zu klären, das Ziel, das in diesem Artikel erreicht werden soll, die Art der Knoten, die zum Erreichen dieses Ziels benötigt werden, und die spezifische Art und Weise, wie jeder Knoten implementiert wird.

Oben **Abbildung 6-1** ist ein Flussdiagramm des Bildverarbeitungsknotens, der die Eingabe *image_raw* von der *rgb_camera* entgegennimmt, dann den Knoten *image_process* durchläuft und schließlich den Zielschwerpunkt und den Zielbereich ausgibt.

6.2.4 Robotarm_visions Codeabschnitt und Analyse

ROS veröffentlicht Bilder in seinem eigenen *sensor_msgs/Image* Nachrichtenformat, aber die Bilder müssen in ein Format konvertiert werden, das OpenCV verwenden kann.

CvBridge ist eine ROS-Bibliothek, die die Schnittstelle zwischen ROS und OpenCV bildet. Hier werden die Funktionen *bridge.imgmsg_to_cv2* und *bridge.cv2_to_imgmsg* verwendet, und das gewählte Bildformat ist "bgr8". Das Format sollte vorher und nachher das gleiche sein.

Ein Teil des Codes ist unten abgebildet.

```
1.  #include "smallrobot_vision/vision.h"
2.  #include <tf/transform_broadcaster.h>
3.  VisionManager::VisionManager(ros::NodeHandle n_, float length, float br
    eadth) : it_(n_)
4.  {
5.      this->table_length = length;
6.      this->table_breadth = breadth;
```

#Abonnieren von Bildthemen

```
7.      image_sub_  = it_.subscribe("/camera_kinect_head_mount/depth/i
    mage_raw", 1, &VisionManager::imageCb, this);
8.      image1_pub_ = it_.advertise("/table_detect", 1);
9.      image2_pub_ = it_.advertise("/object_detect", 1);
10. }
```

```
# 2d-Standort abrufen
void VisionManager::get2DLocation(const sensor_msgs::ImageConstPtr &msg, float &x, float &y)
{
    cv::Rect tablePos;
    #Identifizierung von Tischpositionen
    detectTable(msg, tablePos);
```

 # Identifizierung der Position von Objekten

```
    detect2DObject(msg, x, y, tablePos);
    convertToMM(x, y);
}
void VisionManager::detectTable(const sensor_msgs::ImageConstPtr &msg, cv::Rect &tablePos)
```

Dieser Teil des Codes, der anfänglich das Bildthema über Kinect abonniert, verwendet die Bildinformationen, um die Länge und Breite der Arbeitsfläche und die relative Position zu erhalten, die die Position der Arbeitsfläche und die Position des Objekts relativ zur Arbeitsfläche umfasst.

```
cv::Mat BGR[3];
```

 # opencv-Bildformatkonvertierung

```
    try
  {
    cv_ptr_ = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
  }
  catch (cv_bridge::Exception &e)
  {
    ROS_ERROR("cv_bridge exception: %s", e.what());
    return;
  }   cv::Mat &image = cv_ptr_->image;
```

#Aufteilung der Bildpunkte auf die drei Kanäle R G B

```
    split(image, BGR);
```

```
12.     cv::Mat gray_image_red = BGR[2];
13.     cv::Mat gray_image_green = BGR[1];
14.     cv::Mat gray_image_blue = BGR[0];
15.     cv::Mat denoiseImage;
16.     cv::Mat denoiseImage2;
17.     cv::Mat denoiseImage3;
```

#Glättende Filterung

```
18.     cv::medianBlur(gray_image_red, denoiseImage, 3);
19.     cv::medianBlur(gray_image_green, denoiseImage2, 3);
20.     cv::medianBlur(gray_image_blue, denoiseImage3, 3);
```

 # Abrufen von drei Kanälen mit R G B-Daten

```
21.     cv::Mat binaryImage = denoiseImage;
22.     cv::Mat binaryImage2 = denoiseImage2;
23.     cv::Mat binaryImage3 = denoiseImage3;
```

Der obige Teil des Codes verwendet die Bildkonvertierungsfunktion in OpenCV, um die von der Kamera aufgenommenen Bildinformationen zuerst in das BRG8-Format zu konvertieren, damit diese von OpenCV verwendet werden können. Da ROS Bilder in seinem eigenen *sensor_msgs/Image* Nachrichtenformat veröffentlicht, muss das Bild in ein Format konvertiert werden, das OpenCV verwenden kann. Daher wird *CvBridge* verwendet, ein von ROS bereitgestelltes Funktionspaket, das die Schnittstelle zwischen ROS und OpenCV bildet.

Die nächste verwendete Funktion ist OpenCV, die Glättungsfilterung und die Segmentierung des Farbraums. Das aufgenommene Bild wird mit dem Befehl *split(image, BGR)* in Pixel zerlegt, wodurch die Bildpixel in die Kanäle R, G und B getrennt werden. Das Bild wird dann mit dem Befehl *medianBlur (*)* gefiltert, um die Daten für die Kanäle R, G und B zu erhalten und auszugeben.

Anschließend wird in zwei for-Schleifen die Position der gesamten Schwarztabelle ermittelt, indem die Farbwerte der Bilddaten gelesen werden. Das Prinzip besteht darin, dass zur Bestätigung der Position der Schwarztabelle sowohl im roten als

auch im grünen Kanal der Pixelwert des Punktes an der Tabellenposition auf 255 und der Pixelwert der anderen Position auf 0 gesetzt wird. Der Code ist unten dargestellt.

#Beurteilung der Farbschwelle Bestimmung von Schwarz

```
for (int i = 0; i < binaryImage.rows; i++)
{
    for (int j = 0; j < binaryImage.cols; j++)
 {
    int editValue = binaryImage.at<uchar>(i, j);
    int editValue2 = denoiseImage2.at<uchar>(i, j);
        int editValue3 = denoiseImage3.at<uchar>(i, j);
```

#Überprüfen von Werten, die innerhalb des Bereichs liegen.

```
    if ((editValue < 10) &&  (editValue2 < 10)  && (editValue3 < 10))
    { // check whether value is within range.
        binaryImage.at<uchar>(i, j) = 255;
    }
    else
    {
        binaryImage.at<uchar>(i, j) = 0;
    }
  }
}
dilate(binaryImage, binaryImage, cv::Mat());
```

Der nächste Schritt besteht darin, den Mittelpunkt der Arbeitsfläche zu finden, indem zunächst die Stelle mit einem Pixelwert ungleich Null gefunden wird: *cv::findNonZero(binaryImage, nonZeroPoints)*. Anschließend wird mit der Funktion *boundingRect* das parallel zu den oberen und unteren Grenzen des Bildes verläuft. Die Position des Mittelpunkts der Arbeitsfläche wird dann auf der Grundlage der Position des Kastens berechnet. Der Code ist unten dargestellt.

#Berechnung des Verhältnisses von Pixeln zu physikalischen Größen

```
std::vector<cv::Point> nonZeroPoints;
cv::findNonZero(binaryImage, nonZeroPoints);
cv::Rect bbox = cv::boundingRect(nonZeroPoints);
cv::Point pt;
pt.x = bbox.x + bbox.width / 2;
pt.y = bbox.y + bbox.height / 2;
cv::circle(image, pt, 4, cv::Scalar(0, 0, 255), -1, 8);
pixels_permm_y = bbox.height / table_length;
pixels_permm_x = bbox.width  / table_breadth;
  tablePos = bbox;
```

#Berechnung der Pixel-zu-Physikalischen-Skala und Bestimmung des Mittelpunkts der schwarzen Tabelle

```
std::cout << "Pixels in y" << pixels_permm_y << std::endl;
std::cout << "Pixels in x" << pixels_permm_x << std::endl;
std::vector<std::vector<cv::Point>> contours;
std::vector<cv::Vec4i> hierarchy;
cv::findContours(binaryImage, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0));
for (int i = 0; i < contours.size(); i++)
{
        cv::Scalar color = cv::Scalar(255, 0, 0);
        cv::drawContours(image, contours, i, color, 3, 8, hierarchy, 0, cv::Point());
}
image1_pub_.publish(cv_ptr_->toImageMsg());
}
```

Das liegt daran, dass die Farbe des aufgenommenen Objekts vorgegeben ist. Auf dieselbe Weise wird nur die rote Farbe identifiziert, um die Positionsinformationen des kleinen Objektblocks zu bestimmen. An diesem Punkt werden die erhaltenen Pixelkoordinaten in TF transformiert, um die Kamerakoordinaten zu erhalten, und

schließlich wird die Objektposition im Kamerakoordinatensystem ausgegeben.

Damit ist dieser Teil der Zielerkennung abgeschlossen. Zusammengefasst handelt es sich um die Erkennung der Kamera, die Umwandlung der Zahlen, aus denen der Standort des unbesiegbaren Bruders bestimmt wird, und schließlich die Zusammenführung dieser Standortinformationen, die umgewandelt und veröffentlicht werden.

6.3 Pick and place

6.3.1 Erste individuelle Planung des Roboterarms

Nachdem das Roboterarmmodell zusammen mit dem Bilderkennungssystem konfiguriert wurde, besteht der nächste Schritt darin, den Roboterarm mit den von der Bilderkennung erhaltenen Daten zu steuern. Ursprünglich war vorgesehen, dass der Roboterarm den Objektblock von Tisch 1 greift und ihn mit Hilfe der von der Kinect-Kamera erkannten Informationen auf Tisch 2 ablegt. Da unser Ziel darin besteht, dass der Wagen den Tisch 2 ersetzt und transportiert, wobei der Bewegungsprozess des Roboterarms unverändert bleibt, wird zuerst der Bewegungsprozess des Roboterarms geschrieben.

Die benötigten Informationen müssen zunächst in die von geometry_msgs.msg und moveit_msgs.msg veröffentlichten Nachrichtendateien eingegeben werden. Dazu gehören Informationen über die Robotersteuerung, die Szenenplanung, die Position der mit der vorherigen Kinect-Kamera identifizierten Objekte usw. Als Nächstes werden die erforderlichen Objektparameter über die self-Anweisung definiert, es wird eine Nachricht erstellt, um die Positionsinformationen zu übermitteln, die der Roboter ergreifen muss, und dann werden ein Umgebungshandle und ein Steuerungshandle erstellt. Ein Teil des Codes ist unten abgebildet:

```
1.  class Pick_Place:
2.      def __init__(self):
```

 # Abrufen von Objektparametern

```
3.      self._table_object_name = rospy.get_param('~table_object_name', 'Grasp_Table')
4.      ...
5.      ...
6.      self._approach_retreat_desired_dist = rospy.get_param('~approach_retreat_desired_dist', 0.2)
7.      self._approach_retreat_min_dist = rospy.get_param('~approach_retreat_min_dist', 0.1)
8.
```

 # Erstellen von Orten zur Erfassung von Nachrichtenfreigaben

```
9.      self._grasps_pub = rospy.Publisher('grasps', PoseArray, queue_size=1, latch=True)
10.     self._places_pub = rospy.Publisher('places', PoseArray, queue_size=1, latch=True)
11.
```

 # Erstellen von Umgebungshandles und Steuerhandles

```
12.     self._scene = PlanningSceneInterface()
13.     self._robot = RobotCommander()
14.     rospy.sleep(1.0)
```

Danach muss die Umgebungsszene initialisiert und die Tabelle mit dem zu simulierenden Objektblock in der Szene platziert werden. Nachdem die Tischplatte und das Objekt in der simulierten Umgebung vorhanden sind, muss die Position des Objekts ermittelt werden, d. h. die x,y,z-Koordinaten des Objekts in der simulierten Umgebung. Die Pose des Objekts wird dann an den Roboterarm gesendet, damit dieser die Ausrichtung des zu greifenden Objekts bestimmen kann. Es ist zu beachten, dass sich der base_link 0,9 m über der Ebene der simulierten Umgebung befindet. Daher ist beim Senden der Objektinformationen ein Abzug von 0,9 m vom Wert der z-Koordinate erforderlich.

Ebenso müssen die Greifer beim Aufnehmen des Objekts nicht bis zum Boden des Objekts ausgefahren werden, so dass ein Δ-Wert als Abstand zwischen der Spitze

der Greifer und dem Boden des Objekts definiert werden muss. Für diesen Wert wird Δ = 0,05 m angenommen.

Da der Roboterarm das gegriffene Objekt auf dem Wagen ablegen muss, besteht unsere Entwurfslogik darin, den Wagen in eine feste Position zu fahren, dann beginnt der Roboterarm mit der Greifbewegung und legt das Objekt auf dem Wagen ab, und schließlich kehrt der Wagen in seine Ausgangsposition zurück. Bei der Planung des Bewegungswegs des Arms muss das Objekt also nur auf eine feste Koordinate platziert werden, die auch die Zielkoordinate für die Fahrt des Wagens ist.

Ein Teil des Codes ist unten abgebildet:

 # Erhalten Objekt 1 Pose

```
self._pose_place = Pose()
    self._pose_place.position.x = self._pose_coke_can.position.x
    self._pose_place.position.y = self._pose_coke_can.position.y
    self._pose_place.position.z = self._pose_coke_can.position.z
    self._pose_place.orientation = Quaternion(*quaternion_from_euler(0.0, 0.0, 0.0))
    self._pose_coke_can.position.z = self._pose_coke_can.position.z - 0.9
    self._pose_place.position.z = self._pose_place.position.z + 0.05
```

 # Erhalten Objekt 2 Pose

```
    self._pose2_place = Pose()
    self._pose2_place.position.x = self._pose_coke1_can.position.x
    self._pose2_place.position.y = self._pose_coke1_can.position.y+0.2
    self._pose2_place.position.z = self._pose_coke1_can.position.z
    self._pose2_place.orientation = Quaternion(*quaternion_from_euler(0.0, 0.0, 0.0))
    self._pose_coke1_can.position.z = self._pose_coke1_can.position.z - 0.9 # base_link is 0.9 above
    self._pose2_place.position.z = self._pose2_place.position.z + 0.05 # target pose a little above
```

 # Festlegen der Platzierung

```
15.     self._pose1_place = Pose()
16.     self._pose1_place.position.x = 0.0
17.     self._pose1_place.position.y = -1.0
18.     self._pose1_place.position.z = 0.45
19.     self._pose1_place.orientation = Quaternion(*quaternion_from_euler
        (0.0, 0.0, 0.0))
20.     self._pose3_place = Pose()
21.     self._pose3_place.position.x = 0.0
22.     self._pose3_place.position.y = -1.05
23.     self._pose3_place.position.z = 0.45
24.     self._pose3_place.orientation = Quaternion(*quaternion_from_euler
        (0.0, 0.0, 0.0))
```

Sobald die Koordinaten des Tisches und des Objekts verarbeitet sind, wird der Roboterarm gesteuert. Zuerst werden die Greifer und der Arm gruppiert, indem das Selbst definiert wird, dann wird die Definition des Greifdienstes erstellt, indem der *grasps_ActionClient* festgelegt wird und einige der Parameter gesetzt werden, sowie die Befehle für den gesamten Greifprozess erstellt werden. Anschließend wird der Prozess der Planung der Position des Roboterziels definiert. Unter jeder Definition wird eine *if*-Anweisung verwendet, um die Aktion zu wiederholen, wenn sie nicht erreicht wurde.

Ein Teil des Codes ist unten abgebildet:

#Erhalten der Gruppenbildung von Greifern und Roboterarmen

```
1.  self._arm     = self._robot.get_group(self._arm_group)
2.      self._gripper = self._robot.get_group(self._gripper_group)
3.      self._arm.set_named_target('pickup')
4.      self._arm.go(wait=True)
5.      print("Pickup pose")
```

#Erstellen eines Pickup-Service

```
6.      self._grasps_ac = SimpleActionClient('/moveit_simple_grasps_server/
    generate', GenerateGraspsAction)
7.      if not self._grasps_ac.wait_for_server(rospy.Duration(5.0)):
8.          rospy.logerr('Grasp generator action client not available!')
9.          rospy.signal_shutdown('Grasp generator action client not availabl
    e!')
10.         return
```

 # Erstellen eines Erfassungsaktionsclients

```
11.     self._pickup_ac = SimpleActionClient('/pickup', PickupAction)
12.     if not self._pickup_ac.wait_for_server(rospy.Duration(5.0)):
13.         rospy.logerr('Pick up action client not available!')
14.         rospy.signal_shutdown('Pick up action client not available!')
15.         return
```

 # Erstellen der Aktion-Client von Zielposition des Roboterarms Planung

```
16.     self._place_ac = SimpleActionClient('/place', PlaceAction)
17.     if not self._place_ac.wait_for_server(rospy.Duration(5.0)):
18.         rospy.logerr('Place action client not available!')
19.         rospy.signal_shutdown('Place action client not available!')
20.         return
```

Nach dem obigen Code ist die Vorbereitung des Roboterarms für das Greifen des Objekts abgeschlossen. Anschließend ist es an der Zeit, die anfängliche Definition zu definieren, wie das Objekt gegriffen werden soll. Der Greifprozess des Arms besteht aus 5 Posenwechseln, so dass fünf While-Schleifen erforderlich sind, um den Prozess zu definieren. Die fünf Haltungswechsel sind: von der Ausgangsposition zum Greifen des ersten Objekts, Greifen des ersten Objekts an der Zielposition, Zurückbringen der Zielposition in die Ausgangsposition, Greifen des zweiten Objekts von der Ausgangsposition und Greifen des zweiten Objekts an der Zielposition. Anschließend werden die beiden Tischplatten mit den darauf befindlichen Objektblöcken in die Simulationsumgebung geladen und ihre xyz-Koordinaten kalibriert. Anschließend sind die beiden Tische mit den Objektblöcken in die Simulationsumgebung zu laden, ihre xyz-Koordinaten zu

kalibrieren und der Grip-Generator zu starten, um ein Array von Grip-Positionsdaten für die Visualisierung zu erzeugen und danach das Grip-Ziel an den Grip-Server weiterzuleiten.

Einige der wichtigsten Codes sind unten aufgeführt:

```
1.  def _generate_grasps(self, pose, width):
```

 # Erstellen eines Ziels

```
2.      goal = GenerateGraspsGoal()
3.      goal.pose  = pose
4.      goal.width = width
5.      options = GraspGeneratorOptions()
6.      options.grasp_direction = GraspGeneratorOptions.GRASP_DIRECTIO
        N_UP
7.      options.grasp_rotation  = GraspGeneratorOptions.GRASP_ROTATIO
        N_FULL
```

 # Senden der Bezugsdaten

 # Warten auf die Rückgabe der Bewegungsergebnisse durch den Aktionsserver

```
8.      state = self._grasps_ac.send_goal_and_wait(goal)
9.      if state != GoalStatus.SUCCEEDED:
10.         rospy.logerr('Grasp goal failed!: %s' % self._grasps_ac.get_goal_sta
            tus_text())
11.         return None
12.     grasps = self._grasps_ac.get_result().grasps
13.     self._publish_grasps(grasps)
14.     return grasps
```

Es folgt der Abschnitt über die Erzeugung der Position des Roboterarms zum Ablegen des Objekts. Dies beinhaltet die Erstellung der Ausgangsposition und der Zielposition, dann die Erstellung der Bewegungsrichtung und die Erstellung der Positionsinformationen vor und nach der Platzierung.

Nach der Erstellung der Platzierungspose wird MoveIT aufgerufen, um die Erfassung und Platzierung des Ziels abzuschließen. In diesem Prozess werden erst

die Zielinformationen zum Greifen und Platzieren erstellt und die Zielplanungsoptionen konfiguriert; dann wird ein Ziel mithilfe der Planungsgruppe platziert.

Einige der wichtigsten Codes sind unten aufgeführt:

```
1.    def _pickup(self, group, target, width):
```

 # Erhalten mögliche Griffe vom Grasp-Generator-Server

```
2.        grasps = self._generate_grasps(self._pose_coke_can, width)
```

 # Erstellen und senden Pickup-Ziel

```
3.        goal = self._create_pickup_goal(group, target, grasps)
4.        state = self._pickup_ac.send_goal_and_wait(goal)
5.        if state != GoalStatus.SUCCEEDED:
6.            rospy.logerr('Pick up goal failed!: %s' % self._pickup_ac.get_goal_st
          atus_text())
7.            return None
8.        result = self._pickup_ac.get_result()
```

 # Prüfung auf Fehler

```
9.        err = result.error_code.val
10.       if err != MoveItErrorCodes.SUCCESS:
11.           rospy.logwarn('Group %s cannot pick up target %s!: %s' % (group,
          target, str(moveit_error_dict[err])))
12.           return False
13.       return True
```

 # Erhalten Platzierung

```
14.   def _place(self, group, target, place):
15.       places = self._generate_places(place)
```

#Erstellen und senden Pickup-Ziel

```
16.       goal = self._create_place_goal(group, target, places)
17.       state = self._place_ac.send_goal_and_wait(goal)
18.       if state != GoalStatus.SUCCEEDED:
19.           rospy.logerr('Place goal failed!: %s' % self._place_ac.get_goal_statu
          s_text())
```

```
            return None
        result = self._place_ac.get_result()
        err = result.error_code.val
        if err != MoveItErrorCodes.SUCCESS:
            rospy.logwarn('Group %s cannot place target %s!: %s' % (group, target, str(moveit_error_dict[err])))
            return False
        return True
```

Schließlich wird eine if-Anweisung mit einer for-Schleife verwendet, um die Grab- und Place-Operationsanweisungen abzuschließen. Der Code ist unten abgebildet.

 # Pickup

```
def _publish_grasps(self, grasps):
    if self._grasps_pub.get_num_connections() > 0:
        msg = PoseArray()
        msg.header.frame_id = self._robot.get_planning_frame()
        msg.header.stamp = rospy.Time.now()
        for grasp in grasps:
            p = grasp.grasp_pose.pose
            msg.poses.append(Pose(p.position, p.orientation))
        self._grasps_pub.publish(msg)
```

 # Platzierung

```
def _publish_places(self, places):
    if self._places_pub.get_num_connections() > 0:
        msg = PoseArray()
        msg.header.frame_id = self._robot.get_planning_frame()
        msg.header.stamp = rospy.Time.now()
        for place in places:
            msg.poses.append(place.place_pose.pose)
        self._places_pub.publish(msg)
```

6.3.2 Gemeinsamer Ablauf beim Beladen von Wagen

Nachdem die Planung der Bewegung des Roboterarms abgeschlossen ist, wird ein Wagen entworfen, der anstelle der zweiten Tischplatte bewegt werden kann, um die vom Roboterarm gegriffenen Gegenstände zu transportieren, wie es das Thema erfordert. Die Bewegung des Wagens wird in dieser Python-Datei wie folgt dargestellt.

Der Befehl zur Steuerung der Bewegung des Wagens besteht darin, dass der Wagen die Nachricht *msg = Twist()* abonniert und zwei *for*-Schleifen durchläuft, um sich von der Ausgangsposition zur zweiten Tischplatte zu bewegen und anschließend zur Ausgangsposition zurückzukehren, wenn er das Objekt eingesammelt hat. Ein Teil des Codes ist unten abgebildet:

#Erhöhung der Häufigkeit von Rostopic-Auslösungen

```
1.  rate = rospy.Rate(50)
```

#Befehle zur Bewegungssteuerung

```
2.  msg = Twist()
3.  for row in range(0,580):
4.    msg.linear.x=float(0.3)
5.    msg.angular.z=float(0.01)
6.    car_pub.publish(msg)
7.    rate.sleep()
8.  main()
```

#Rücklaufteil des Wagens

```
9.  for row in range(0,580):
10.   msg.linear.x=float(-0.3)
11.   msg.angular.z=float(-0.01)
12.   car_pub.publish(msg)
13.   rate.sleep()
```

Damit ist diese Datei pick_and_place_and_car.py abgeschlossen.

Im Paket *moveit_simple_grasps* wurde die Datei simple_grasp_server.cpp geschrieben, die einen Grasp-Generierungsserver mit dem Nachrichtentyp

GenerateGraspsAction erstellt.

Die Hauptaufgabe der von uns geschriebenen Datei *pick_and_place_and_car.py* besteht darin, einen *Action-Client* mit dem Namen */moveit_simple_grasps_server /generate* zu erstellen, um die Grasp-Bit-Posen abzufragen, wiederum mit dem Nachrichtentyp *GenerateGraspsAction*. Der */pickup-Client* und der */place-Client* der Bewegungsgruppe werden ebenfalls erstellt und nach erfolgreicher Generierung des *grasps*-Befehls zur Verarbeitung an den entsprechenden Server der Bewegungsgruppe gesendet, so dass der Arm gesteuert werden kann.

Das von *rosgraph* erzeugte strukturierte Baumdiagramm ist im Abb.6-2 abgebildet.

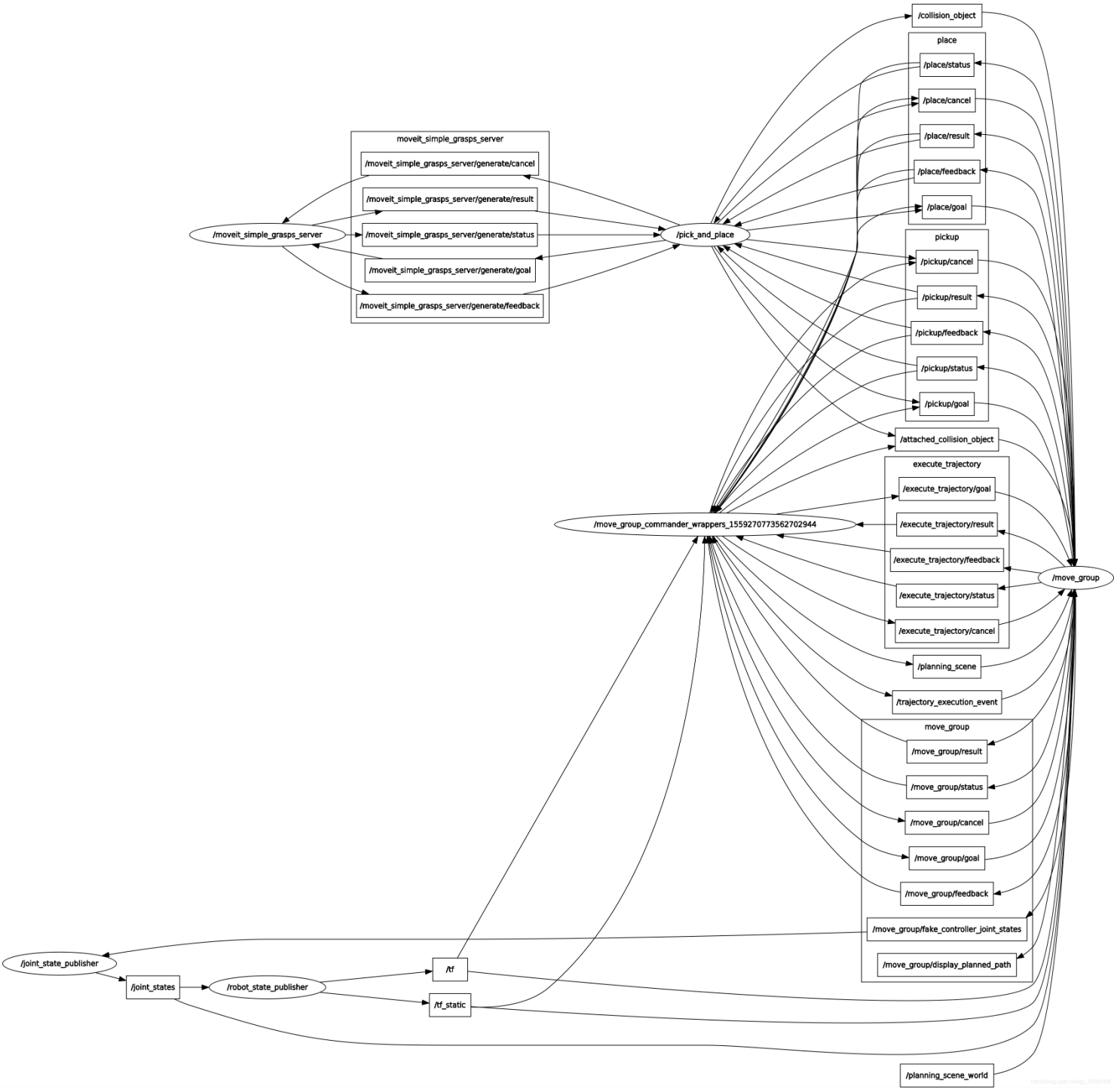


Abbildung 6-2 Rosgraph von Pickup und Place

6.4 Gmapping und AMCL

In Abschnitt 5.9 wurden die Konzepte von gmapping und AMCL vorgestellt, in diesem Abschnitt wird gezeigt, wie der Prozess der Navigation von Sekundärroboter durch Code implementiert wird.

Da es sich bei den Funktionspaketen Gmapping und AMCL um Funktionspakete handelt, die direkt unter der ROS-Plattform aufgerufen werden können, braucht nur die entsprechende Launch-Datei geschrieben zu werden.

6.4.1 Launch-Datei zur Kartenerkennung und Positionierung

Am Anfang wird in dieser Launch-Datei die Map-Datei über arg mit den Sensoren im TurtleBot aufgerufen, darunter der Rgb-Sensor und der Tiefensensor. Gleichzeitig liest der Knoten des *map_server_package* die Karteninformationen und speichert die von den Sensoren erfassten Informationen.

Im Funktionspaket *turtlebot_navigation* finden sich dann Informationen zu den Parametern, die für die Positionierung verwendet werden können, die ebenfalls über den Befehl arg aufgerufen werden. Schließlich werden die Positionsdaten in *move_base* eingegeben, um den Schlitten zu bewegen. Dies ist das Grundprinzip der Positionierung und Navigation, ein Teil des Codes wird unten gezeigt.

```
1.  <launch>
```

 # Map server

```
2.   <arg name="map_file" default="/home/hsw/map/gazebo/my_map.yaml
"/>
3.   <arg name="3d_sensor" default="$(env TURTLEBOT_3D_SENSOR)"/>  <!-
- r200, kinect, asus_xtion_pro -->
4.   <node name="map_server" pkg="map_server" type="map_server" args
="$(arg map_file)" />
```

 # Localization

```
5.   <arg name="initial_pose_x" default="0.0"/>
6.   <arg name="initial_pose_y" default="0.0"/>
7.   <arg name="initial_pose_a" default="0.0"/>
```

```
 <arg name="custom_AMCL_launch_file" default="$(find turtlebot_navigation)/launch/includes/AMCL/kinect_AMCL.launch.xml"/>
 <include file="$(arg custom_AMCL_launch_file)">
   <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
   <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
   <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
 </include>
```

 # Move base

```
 <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml"/>
</launch>
```

6.4.2 Programmausführung mit generierten Kartendateien

Sobald der Arm und der Wagen erfolgreich in den Simulator in der ROS-Umgebung geladen wurden, wird die Launch-Datei aus dem vorherigen Kapitel ausgeführt, um eine Karte mit den 3D-Sensoren im TurtleBot zu erstellen, die entsprechend der Bewegung des Wagens verfeinert wird. Nachdem der Wagen den Transport des Objekts beendet hat, wird eine Draufsicht auf die gesamte Umgebung gemessen. Das Programmablaufdiagramm ist unten in Abbildung 6-3 dargestellt.

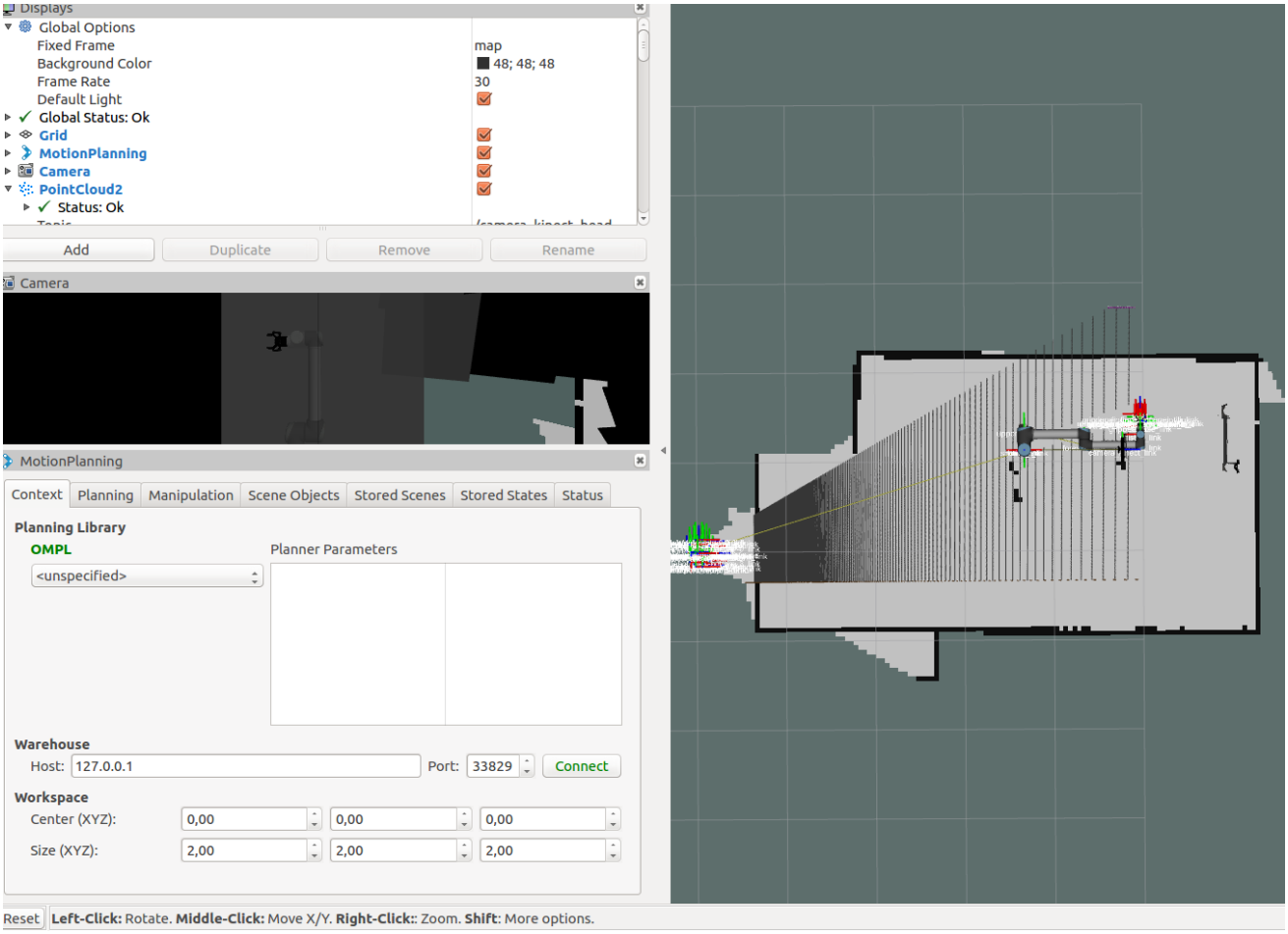


Abbildung 6-3 Programmablaufdiagramm

7 Aufgabenzuweisung für kooperative Multi-Roboter-Systeme

Der Zuweisung von Multi-Roboter-Aufgaben ist die wichtigste Forschungskomponente der Multi-Roboter-Kooperation und das zu lösende Problem ist, welcher Roboter welche Aufgabe ausführt oder wie viele Roboter einer Aufgabe zugewiesen werden. Die Aufgabenzuweisung ist die Grundlage für den Betrieb des gesamten Multi-Roboter-Systems und hat einen entscheidenden Einfluss darauf, wie das System seine Aufgaben erfüllt, wie sich die Roboter untereinander koordinieren und auf andere Aspekte wie Systemziele oder Leistung durchfuhren. Der Mechanismus der Aufgabenzuweisung als Grundlage der Zusammenarbeit des Systems schränkt die Lösungsmöglichkeiten für andere Kooperationsprobleme wie z. B. die Konfliktlösung ein. Daher hat das Problem der Aufgabenzuweisung zunehmend an Aufmerksamkeit gewonnen und ist zu einem Schwerpunkt in der kollaborativen Multi-Robotik-Forschung geworden. [60]

7.1 Aufgabe und Zusammenarbeit

Die Aufgabe ist eng mit dem Design von Multi-Roboter-Systemen verbunden sowie dem Verständnis, wie die Komplexität der Aufgabe und der Umgebung das Design von Multi-Roboter-Systemen beeinflusst, deshalb bleibt dies eine große Herausforderung innerhalb des aktuellen Multi-Roboter-Forschungsfeldes [61]. Obwohl es noch nicht möglich ist, die Wechselwirkungen zwischen Aufgaben und Kontrollstrukturen von Multi-Roboter-Systemen, Kooperationsmodellen usw. in einem formalen oder präzisen mathematischen Ausdruck darzustellen, kann die qualitative Analyse der Aufgabencharakteristika und ihrer Komplexität den Forschern helfen, rationale Kontrollstrukturen, Kooperationsmechanismen und Kommunikationsmodelle zu entwerfen.

7.2 Große "MMK"-Projekte in der EU-Tech-programme

Die Zusammenarbeit zwischen Mensch und Maschine wurde im Rahmen des 7. Förderrahmens der Europäischen Union mit vier großen Wissenschafts- und Technologieprogrammen seit 2006 gefördert:

(1) PISA (2006~2010) Raum-Zeit-Teilung; [62]

(2) ROSETTA (2009~2013) Naturkommunikation; [63]

(3) VALERI (2011~2015) Mobile Assistenz; [64]

(4) Robo-Mate (2013~2016) Tragkraft. [65]

Die Forschungsprogramme zeichnen sich durch ihre lange Dauer, Stärke, Anzahl der Einheiten, breite Bereiche und Ergebnisse aus.

7.2.1 PISA: Flexibles Montagesystem

Das Projekt beinhaltet 18 Partner, 4 Industrieunternehmen (VW, COMAU, etc.), 7 kleine und mittelständische Unternehmen (Senur Elektrik, etc.) und 7 Universitätsinstitute (Fraunhofer IPA, etc.), mit dem übergeordneten Ziel, dass der Mensch immer an der Arbeit beteiligt ist, aber mit leistungsfähigen Werkzeugen ausgestattet werden muss. 9 Teilprojekte waren für das PISA-Projekt geplant. Die drei wichtigsten Technologien, die entwickelt wurden, sind: (1) Intelligente Assistenzsysteme (IAS); (2) Planungswerkzeuge für die Systemintegration; und (3) Rekonfiguration und Wiederverwendbarkeit von Montageausrüstung.

7.2.2 Das ROSETTA-Programm

Das Programm wird von der Firma ABB mit insgesamt 10 Mio. € gefördert. Das Ziel ist eine auf Autonomie, Wissensakkumulation und Selbstlernen basierende Robotersteuerungstechnologie, die auf natürliche Weise mit dem Menschen interagiert, um Aufgaben kompetent zu erfüllen.

Dies basiert auf dem 7. Rahmenprogramm, FP7/2007-2013 (No230902) Herausforderung 2: Kognitive Systeme, Interaktion, Robotik. Die Forschung ist wie

folgt.

(1) Sicherheitstechnische Simulation zur Erfüllung der DACR.

(2) Visuelle Betriebsverfolgung.

(3) Menschenzentrierte Roboterbewegung.

(4) Offline-Programmierung für aufgabenorientierte Montageaufgaben.

(5) Vernetzte, selbstlernende Montageroboter.

(6) Webbasierte semantische Produktions-Wissensbasis.

7.2.3 Industrielle Anwendungsvalidierung von VALERIs

Das Rahmenprogramm der EU für Forschung, technologische Entwicklung und Demonstration unter einer Finanzhilfevereinbarung (NO314774) finanziert. Die Motivation für das Projekt ist die Entwicklung und Validierung eines mobilen Manipulators zur Unterstützung des Menschen bei Produktionsaufgaben in der Luft- und Raumfahrt. Die Partner sind Fraunhofer, KUKA, Airbus und sieben weitere. Visuelle Raumüberwachung entwickelte sich zur Realisierung von Werkzeugabsicherungen und die Zusammenführung von Bildverarbeitung in ein mobiles System mit haptischen Sensoren für Sicherheit und haptische Interaktion am mobilen Roboter.

7.2.4 Robo-Mate tragbares Exoskelett

Es handelt sich um eine Technologie mit größerem Sicherheitsbedarf, an der 12 Unternehmen beteiligt sind und die wie folgt entwickelt wurde:

(1) Die Entwicklung eines intelligenten, kollaborativen, leichten, tragbaren Mensch-Computer-Exoskeletts mit einer benutzerfreundlichen Schnittstelle zur manuellen Bearbeitung verschiedener industrieller Aufgaben.

(2) Das Gerät ist in einem halben Tag einsatzbereit, es ist keine spezielle Programmierung erforderlich und das entwickelte Exoskelett ist hochflexibel und kann direkt in der Handarbeit, Massenproduktion oder Hilfsprozessen eingesetzt

werden.

(3) Verbesserung der Arbeitsbedingungen für Lasten und Erleichterung von sich wiederholenden Hebeaufgaben, wodurch die Häufigkeit von arbeitsplatzbedingten Verletzungen und Krankheiten reduziert wird.

Das Robo-Mate-Exoskelett besteht aus einer Reihe von Modulen, die Arme, Rumpf und Beine unterstützen (Abbildung 6). Das Armmodul stützt aktiv den Arm und das Armmodul des Trägers und spürt beim Heben von schweren Gegenständen nur 1/10 des tatsächlichen Gewichts. Am Armmodul ist das Rumpfmodul angebracht, das den Rücken und die Wirbelsäule stützt und Verdrehungen oder Bandscheibenvorfällen vorbeugt. Das Beinmodul stützt die Innenseite der Oberschenkel wie bei einem Hocksitz, so dass keine zusätzliche Kraft zum Halten der Last erforderlich ist.

7.3 Die Schlüsseltechnologien für die Zusammenarbeit

Um das Problem der Mensch-Roboter-Ko-Kooperation zu lösen, müssen Schlüsseltechnologien in vier Bereichen entwickelt werden: Institutionelles Design, Multidimensionale Wahrnehmung, Ortsüberwachung sowie Planung und Test.

(1) Institutionelles Design: bionischer Elefantenrüssel, Leichtbauroboter, Maschinenhaut, flexible Krallenzange.

(2) Multidimensionale Wahrnehmung: eingebettete taktile Robotersensoren, kapazitive Sensoren zur Näherungserkennung, räumlich genaue druckempfindlicher Teppich.

(3) Ortsüberwachung: Projektions- und Kamerasysteme, Online-Planung von Vollschutzzonen, dynamische Schutzzonenplanung, Sicherheitsschutz-Tools.

(4) Planung und Test: Crashtests, Augmented-Reality-Umgebungen, Planung von Mensch-Roboter-Interaktionszellen.

7.3.1 Institutionelle Auslegung

Die Entwicklung von bionischem Design, Humanoid- und Handdesign, die Entwicklung von elastischer Haut und weichen Strukturen, etc. Der Rüssel eines Elefanten beispielsweise ist bionisch und wird im 3D-Druckverfahren hergestellt, was die Vorteile einer hohen Flexibilität und geringen Dichte (0,94 g/cm^3) hat.

Abbildung 7-1 Roboter mit Elefantenrüssel [67]

7.3.2 Multidimensionale Wahrnehmung

Um eine multidimensionale Wahrnehmung der kollaborativen Mensch-Maschine-Umgebung zu erreichen, werden mehrere Wahrnehmungsmethoden, wie z. B. visuelle Entfernungsortung, sensible Haut- und Entfernungssensoren, eingesetzt.

(1) Visuelle Messmethode.

Ein integriertes visuelles Ortungsgerät, das für mobile Systeme in Werks-, Lager-, Labor- und Heimumgebungen entwickelt wurde. Es nutzt monokulare Selbstlokalisierung und kartografische Navigation (SLAM), wobei natürliche und künstliche Landmarken als Referenzpunkte verwendet werden.

(2) Druckempfindliche Roboterhäute, hybride Roboterhäute.

Konforme haptische Sensoren zur Kollisions- und Näherungserkennung und haptischen Rückmeldung, die eine vibrationsgedämpfte sichere Kollisionserkennung ermöglichen.

(3) Kapazitive Sensoren zur Abstandserkennung.

Dies beschreibt die Möglichkeit statische Objekte im Arbeitsbereich auszublenden, deren Prozess und Arbeitsbereich sich ändern, wenn Sie

Abstandserkennungssensoren für die relative Messung von Objekten verwenden, die in die Sichtlinie eintreten.

Außerdem geleistet wird die Erkennung des Abstandes, der von lebensgefährlichen Körperteilen erreicht wird, während der Roboter mit den zuvor definierten absoluten Werten arbeitet, wobei der Sicherheitsabstand für die Geschwindigkeitsbegrenzung des Roboters kontinuierlich angepasst und berücksichtigt wird.

7.3.3 Platzüberwachung

Zur Überwachung des Arbeitsbereichs werden Projektionen und Kameras eingesetzt, die sichere Arbeitsbereiche erzeugen und überwachen. Diese werden direkt in die Umgebung projiziert und können einen Gefahrenalarm auslösen, wenn der projizierte Strahl gestört wird. Es wird ein duales Versicherungsmodell "harte Sicherheit + weiche Sicherheit" verwendet, bei dem nicht sichtbare Sicherheitszonen mit Nahinfrarotlicht Objekte erkennen, die den Arbeitsbereich betreten und verlassen, und dynamische Änderungen der Form, Position und Größe der Sicherheitszone unterstützen.

8 Schlussbetrachtung

8.1 Analyse der Ergebnisse

Das Hauptziel dieser Diplomarbeit besteht darin, ein Steuerungssystem für einen Roboter zu bauen, das auf einem UR10e sechsachsigen kollaborativen Roboter basiert, und die Roboterplatzierungsumgebung einzurichten, die Bahnplanungsfunktion des Roboterarms auszuführen und in der Lage zu sein, mit dem zweiten Roboterwagen zusammenzuarbeiten, um das Greifen, Platzieren und Liefern des Zielobjekts abzuschließen.

Durch die Modellierung, Echtzeitsteuerung, Simulation und Trajektorienplanung des sechsachsigen kollaborativen Roboters wurde gelernt, wie komplexe Roboter in Ros aufgebaut werden und kollaborativ arbeiten können. Nach der experimentellen Untersuchung des Roboters zeigten die Ergebnisse die Wirksamkeit des Steuerungssystems und die Effektivität der Roboterbahnplanung. Die folgenden Arbeiten wurden in diesem Projekt abgeschlossen.

Basierend auf der Analyse der strukturellen Parameter und D-H-Regeln des sechsachsigen kollaborativen Roboters UR10e wurde ein positives kinematisches Modell erstellt und die inverse Lösung der Kinematik des kollaborativen Roboters mit sechs Freiheitsgraden nach der analytischen Methode gelöst. Die Korrektheit des kinematischen Modells wurde durch die Verwendung der MoveIT-Toolbox in der ROS-Plattform verifiziert und bietet eine theoretische Referenz für zukünftige kinematische Studien von kollaborativen Robotern. Es hilft auch bei der Konfiguration des Roboterarmmodells in der Ros-Plattform und bei der Erstellung und Analyse des kinematischen Modells. Die TF-Softwarebibliothek wurde verwendet, um die Koordinatentransformation zwischen den Parent- und Child-Joints des Roboters in ROS zu implementieren und auf Basis dieser Informationen eine vollständige kinematische Baumstruktur des Roboters zu konstruieren.

Anschließend wurde aufgrund der erforderlichen Erkennung der gegriffenen Objekte das Objekterkennungsgerät, die Tiefenkamera Kinect, eingesetzt. Anhand

der von der Kamera gesammelten Daten wurden mit OpenCV die verschiedenfarbigen Objekte auf dem Tisch mittels der Farbbasierten Segmentierung erkannt. Basierend auf OpenCV wurde dann die Pose der Kalibrierungsplatte relativ zum Kamera-Koordinatensystem und der Hand-Augen-Matrix gelöst; durch visuelle Bildverarbeitung wurde die Position des Zielobjekts im Roboter-Koordinatensystem ermittelt; mit dem RRT_Connect-Algorithmus wurde die Bahnplanung des Bedienroboters von der Ausgangsposition zur Zielposition realisiert und schließlich die Bahnplanung und Bewegungssteuerung im Gazebo-Simulator visualisiert. Die abschließende Visualisierung der Bahnplanung und Bewegungssteuerung erfolgte im Gazebo-Simulator, um den autonomen Greifvorgang des Manipulator Roboters abzuschließen.

Der Sub-Roboter TB2 wurden dann aufgrund seiner Tragbarkeit und Montagefähigkeit ausgewählt und eingesetzt, so dass der Sub-Roboter-Wagen eine Wegerkennung durchführen und Objekte zur Aufnahme und Beförderung ergreifen kann. Dabei wurde das Gmapping-Paket in ROS als Teil seiner Funktionalität verwendet, um den Wagen in die Lage zu versetzen, einen Weg zu erkennen, einen bestimmten Ort zu erreichen und eine Karte seiner Umgebung zu erstellen, wobei ein Zielpunkt und eine Ausgangsposition vorgegeben waren.

Abschließend wurde eine Analyse der Multi-Roboter-Kollaboration vorgestellt und es wurden Sicherheitsfragen und einige Schlüsseltechnologien für die Mensch-Roboter-Kollaboration kurz erläutert.

8.2 Ausblick

Die menschliche Produktion hat sich von der manuellen Arbeit über die halbautomatische bis hin zur vollautomatischen Produktion entwickelt. Die Zukunft wird in die Ära der Mensch-Roboter-Kollaboration eintreten und zu einem normalen Arbeitsmodus werden. Heute dürfen Roboter nur noch in vor- und nachgelagerten Produktionslinien eingesetzt werden, z. B. zum Be- und Entladen,

in der Montage werden Handarbeitsplätze in Verbindung mit Förderbandsystemen eingesetzt, um eine schlanke Produktion von Zellen zu erreichen. Doch konzeptionell haben einige große Bauunternehmen bereits damit begonnen, in bestimmten Bereichen der Bauindustrie und des Hausinneren auf eine Automatisierung umzusteigen, angefangen bei der konzeptionellen Planung der Synergie von menschlicher und Roboterarbeit. Vielleicht werden in der Zukunft in der Bauindustrie Mensch und Roboter miteinander kombiniert, wobei kollaborierende Roboter mit multifunktionalem Greifer arbeiten, die eine geführte und effiziente Programmierung verwenden und die Kostenwettbewerbsfähigkeit des gesamten Montagesystems erhöhen. Kollaborative Roboter mit sechs Freiheitsgraden arbeiten an der Seite von Menschen, und ihre kompakten Bewegungen stören die Arbeit der Arbeiter nicht. In Bezug auf die Arbeitsverteilung zwischen Mensch und Maschine werden einfache, sich wiederholende und arbeitsintensive Aufgaben den Robotern überlassen werden und komplexe geistige Arbeiten dem Menschen selbst.

1) Mensch-Maschine-freundliche Zusammenarbeit

Automatisierte Prozesse erfordern immer noch Menschen, und Menschen als Teil der Lösung können durch die vorhandene Technologie nicht vollständig automatisiert werden. Ein komplettes Mensch-Maschine-Kollaborationssystem mit einfacherer Programmierung bedeutet, dass die Anlage nicht viele Engineering-Ressourcen benötigt.

2) Effektiver und besser

Automatisierung mit minimalen Sicherheitsrisiken und kompaktem Arbeitsraum erleichtert den Fabriken die Nutzung bestehender Standorte für eine effiziente Produktion und Anwendung, und kollaborative Arbeitslinien mit einer Kombination aus Mensch und Maschinen ermöglichen eine schnellere und effizientere Arbeit.

3) Höhere Qualität, weniger Abfall

Menschen und Roboter in der Zusammenarbeit übertreffen die Präzision und

Geschwindigkeit, die der Mensch allein erreichen kann, und resultieren in höherwertigen Produkten und weniger Abfall.

4) Vereinfachung von Programmiertechniken

Die Integration von mehreren Funktionspaketen in die ROS-Plattform ermöglicht klare Designvorstellungen, konkrete Roboterfunktionen und eine einfache manuelle Bedienung. Da die ROS-Plattform iteriert und aktualisiert wird, ist sie in Verbindung mit der internen Steuerung des Roboters in der Lage, in der Programmiertechnologie führend zu sein, so dass es möglich ist, traditionell komplexe Programmierungen durch einen einfachen Code durchzuführen. Jeder kann dies beherrschen, eine spezielle Ausbildung oder besondere Programmierkenntnisse sind nicht erforderlich.

9 Literaturverzeichnis

[1] G. George, Python Robotics Projects, Published by Packt Publishing Ltd., 2018.

[2] E. N. V. I. N. M. F. D. Jing Dong, „Distributed Real-time Cooperative Localization and Mapping using an," *2015 IEEE International Conference on Robotics and Automation,* p. 5807, 2015.

[3] „Day and Night Collaborative Dynamic Mapping in Unstructured," *2020 IEEE International Conference on Robotics and Automation (ICRA),* p. 2981, 31 5 2020.

[4] M. F. A. A. A. A. R. L. F. Abdulmuttalib T. Rashid, „Multi-robot localization and orientation estimation using," *Robotics and Autonomous Systems,,* pp. 108-121, 2015.

[5] C. Z. R. L. C. Y. J. Z. M. W. W. a. D. W. Yufeng Yue∗, „A Hierarchical Framework for Collaborative Probabilistic Semantic Mapping," *2020 IEEE International Conference on Robotics and Automation (ICRA),* pp. 9659-9665, 31 8 2020.

[6] ISO/TS 15066:2016, „Robots and robotic devices-Collaborative robots," International Organization for Standardization, 2016-02-04.

[7] W. H. P. X. e. a. HU Mingwei, „Analysis and simulation on kinematics performance of a," *CAAI transactions on intelligent,* pp. 75-81, 12 2017.

[8] A. Stolt, M. Linderoth, A. Robertsson und R. Johansson, „Robotic assembly of emergency stop buttons," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2013.

[9] E. G. a. E. Ackerman, „How Rethink Robotics built its new Baxter robot worker," IEEE Spectrum, 2012.

[10] J. M. Hollerbach, „Optimum kinematic design for a seven degree of freedom manipulator," *MIT Artificial Intelligence Laboratory.*

[11] Urheberrecht 2021 ABB, „yumi Robot," ABB, 2021. [Online]. Available: https://cobots.robotics.abb.com/en/robots/yumi/.

[12] kuka-de, [Online]. Available: https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/lbr-iiwa.

[13] *. T. H. b. Z. H. b. M. O. b. K. W. Fusaomi Nagata a, „CAD/CAM-based position/force controller for a mold polishing robot," *ScienceDirect,* pp. 207-216, 17 1 2007.

[14] S. Yu, „RESEARCH ON MODELING AND FAULT-TOLERANT CONTROL OF REDUNDANT ROBOT UNDER SINGLE JOINT FAILURE," Harbin Institute of Technology .

[15] V. B. R. A. K. Christoforos I Mavrogiannis, „Socially Competent Navigation Planning by Deep Learning ofMulti-Agent Path Topologies," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 6817-6824, 24 9 2017.

[16] J. C. Lentin Joseph, Mastering ROS for Robotics Programming, Second Edition, Published by Packt Publishing Ltd., 2018.

[17] P. D. Vaish, Python Robotics Projects, Build smart and collaborative robots using Python, Published by Packt Publishing Ltd., 2018.

[18] H. Qi, X. Si-xiang, J. Tian-qi und F. Jian-zhong, „Obstacle avoidance motion planning of manipulator with RGBD camera based on ROS," *Manufacturing Automation,* pp. 56-60, 2019.

[19] R. A. Brooks, Flesh and Machines: How Robots Will Change U, Pantheon Books, 2002.

[20] B. Gates, „A Robot in Every Home," Scientific American, 2008.

[21] C. Schou, J. Damgaard, S. Bøgh und O. Madsen, „Human-robot interface for instructing industrial tasks using kinesthetic teaching," *IEEE ISR 2013,* 24 10 2013.

[22] M. V. Sonia Chernova, „Teaching Collaborative Multi-Robot Tasksthrough Demonstration". *Computer Science Department,.*

[23] Y. Gu, A. Thobbi und W. Sheng, „Human-robot collaborative manipulation through imitation and reinforcement learning," *IEEE International Conference on Information and Automation,* 6-8 6 2011.

[24] B. Matthias, S. Oberer-Treitz, H. Staab, E. Schuller und S. Peldschus, „Injury Risk Quantification for Industrial Robots in Collaborative Operation with Humans," *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics),* 7-9 6 2010.

[25] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg und R. Mellander, „Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept," *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM),* 5 2011.

[26] B. Matthias, S. Oberer-Treitz und H. Ding, „Experimental Characterization of Collaborative Robot Collisions," *ISR/Robotik 2014; 41st International Symposium on Robotics,* 2-3 6 2014.

[27] A. Wahrburg, E. Morara, G. Cesari, B. Matthias und H. Ding, „Cartesian Contact Force Estimation for Robotic Manipulators using Kalman Filters and the Generalized Momentum," *2015 IEEE International Conference on Automation Science and Engineering (CASE),* 24 8 2015.

[28] A. Wahrburg, E. Morara, G. Cesari, B. Matthias und H. Ding, „Cartesian contact force estimation for robotic manipulators using Kalman filters and the generalized momentum," *2015 IEEE International Conference on Automation Science and Engineering (CASE),* 24 8 2015.

[29] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding und B. Matthias, „Safety in Human-Robot Collaborative Manufacturing Environments: Metrics and Control," *IEEE Transactions on Automation Science and Engineering ,* pp. 882-893, 2 4 2016.

[30] L. Rozo, S. Calinon, D. G. Caldwell, P. Jiménez und C. Torras, „Learning Physical Collaborative Robot Behaviors From Human Demonstrations," *IEEE Transactions on Robotics ,* 11 4 2016.

[31] E. Magrini, F. Flacco und A. D. Luca, „Control of generalized contact motion and force in physical human-robot interaction," *2015 IEEE International Conference on Robotics and Automation (ICRA),* pp. 1050-4729 , 26 3 2015.

[32] D. P. Le, J. Choi und S. Kang, „External force estimation using joint torque sensors and its application to impedance control of a robot manipulator," *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013),* pp. 2093-7121, 23 10 2013.

[33] H. Cho, M. Kim, H. Lim und D. Kim, „Cartesian sensor-less force control for industrial robots," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 14 9 2014.

[34] T. Umeno und Y. Hori, „Robust speed control of DC servomotors using modern two degrees-of-freedom controller design," *IEEE Transactions on Industrial Electronics,* pp. 363-368, 10 1991.

[35] A. Wahrburg, S. Zeiss, B. Matthias und H. Ding, „Contact force estimation for robotic assembly using motor torques," *2014 IEEE International Conference on Automation Science and Engineering (CASE),* 18 8 2014.

[36] H. Durrant-Whyte und T. Bailey, „Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine ,* 5 6 2006.

[37] G. Grisetti, C. Stachniss und W. Burgard, „Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Transactions on*

Robotics, 5 2 2007.

[38] A. J. Davison, I. D. Reid, N. D. Molton und O. Stasse, „MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 1052-1067, 23 4 2007.

[39] S. J. K. Uhlmann, „A NewExtensionof the KalmanFilterto Nonlinear System". *TheRoboticsResearch Group,Department of EngineeringScience,.*

[40] G. Klein und D. Murray, „Parallel Tracking and Mapping for Small AR Workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality,* 13-16 11 2007.

[41] B. T. P. F. M. R. I. H. A. W. Fitzgibbon, Bundle Adjustment — A Modern Synthesis, 12: Dept of Engineering Science, University of Oxford, 2002.

[42] T. Qin, P. Li und S. Shen, „VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics ,* pp. 1004-1020, 27 7 2018.

[43] L. Vig und J. Adams, „Multi-robot coalition formation," *IEEE Transactions on Robotics ,* pp. 637-649, 7 8 2006.

[44] J. Fenwick, P. Newman und J. Leonard, „Cooperative concurrent mapping and localization," *Proceedings 2002 IEEE International Conference on Robotics and Automation,* 11 5 2002.

[45] R. M. F. E. PARKER, „Distributed Cooperative Outdoor Multirobot Localization and Mapping," *Autonomous Robots 17, 23–39, 2004,* pp. 24-38, 2004.

[46] R. Collins, „Mean-shift blob tracking through scale space," *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.,* 18 6 2003.

[47] S. Thrun und M. Montemerlo, „The Graph SLAM Algorithm with Applications

to Large-Scale Mapping of Urban Structures," *Stanford AI Lab,* 1 5 2006.

[48] H. Rue und S. Martino, „Approximate Bayesian inference for hierarchical Gaussian Markov random field models," *Journal of Statistical Planning and Inference,* pp. 3177-3192, 1 10 2007.

[49] S. Ding, G. Liu, Y. Li, J. Zhang, J. Yuan und F. Sun, „SLAM and moving target tracking based on constrained local submap filter," *2015 IEEE International Conference on Information and Automation,* 8-10 8 2015.

[50] L. E. P. G. S. S. Andrew Howard, „Experiments with a Large Heterogeneous Mobile Robot Team: Exploration, Mapping, Deployment and Detection," *The International Journal of Robotics Research, 2006,* 1 5 2006.

[51] D. M.-G. P. M. Nived Chebrolu, „Collaborative Visual SLAM Framework for a Multi-Robot System," *Sciences de l'ingénieur [physics] /Automatique / Robotique,* 29 1 2020.

[52] J. Engel, T. Schöps und D. Cremers, „Large-Scale Direct Monocular SLAM," in *European Conference on Computer Vision*, TU-München, 2014, pp. 934-849.

[53] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy und S. Teller, „Multiple relative pose graphs for robust cooperative mapping," *2010 IEEE International Conference on Robotics and Automation,* pp. 1050-4729 , 3-7 5 2010.

[54] R. B. a. D. R. Frank, „An Experiment: A File Management System that Simulates ISAM (Indexed Sequential Access Method)," *A1.1Proceedings of Student/Faculty Research Day, CSIS, Pace University, May 5th, 2006 ,* 5 5 2006.

[55] V. Indelman, E. Nelson, N. Michael und F. Dellaert, „Multi-robot pose graph localization and data association from unknown initial relative poses via expectation maximization," *2014 IEEE International Conference on Robotics and Automation (ICRA),* 7 6 2014.

[56] L. P. M. T. M. S. H. L. Sajad Saeedi, „Map merging for multiple robots using Hough peak matching," *Robotics and Autonomous Systems,* pp. 1408-1424, 10 2014.

[57] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray und P. H. S. Torr, „Collaborative Large-Scale Dense 3D Reconstruction with Online Inter-Agent Pose Optimisation," *IEEE Transactions on Visualization and Computer Graphics,* pp. 2895 - 2905, 15 10 2018.

[58] D. H. P. -. D. Parameters, „DH Parameters for calculations of kinematics and dynamics," 23 6 2021. [Online]. Available: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/.

[59] D. E. KAMKE, DIFFERENTIALGLEICHUNGEN LÖSUNGSMETHODEN UND LÖSUNGEN, SPRINGER FACHMEDIEN WIESBADEN GMBH.

[60] M. Z. D. B. M. K. a. D. D. Klaus Bengler, „Interaction Principles for Cooperative Human-Machine Systems," *From the journal ,* 4 2012.

[61] L. E. Parker, „Cooperative Robotics for Multi-Target Observation," *Full Terms & Conditions of access and use can be found athttps://www.tandfonline.com/action/journalInformation?journalCode=tasj20Intelligent Automation & Soft Computing,* pp. 5-19, 24 10 2013.

[62] R. B. D. S. V. K. K. Schröer, „FLEXIBLE ASSEMBLY SYSTEMS THROUGH WORKPLACE-SHARING AND TIME-SHARING HUMAN-MACHINE COOPERATION," *ISR 2010,* 1 7 2011.

[63] ROSETTA, „ROSETTA final summary report," 11 11 2016. [Online]. Available: http://fp7rosetta.org/?q=node/11.

[64] VALERI, „The final VALERI flyer," 11 2015. [Online]. Available: http://www.valeri-project.eu/files/2015/11/valer i_2015.pdf..

[65] C. C. D. P.-C. Sebastian-IoanStana, „Exoskeleton-centered Process

Optimization in Advanced Factory Environments," *Procedia CIRP,* pp. 740-755, 2016.

[66] universal-robots, „ur10-roboter," [Online]. Available: https://www.universal-robots.com/de/produkte/ur10-roboter/. [Zugriff am 25 5 2021].

[67] „google, picture," GOOGlE. [Online].

10 Anhang

Thesen:

1. Der kombinierte Einsatz von mehreren Robotern und Menschen kann die Effizienz der Technik im Bauwesen steigern.

2. Die kinetischen Anforderungen der Mensch-Roboter-Interaktion und die Untersuchungen zur Sicherheit.

3. Die Anforderungen an die Mensch-Roboter-Kooperation bei Robotern, einschließlich der Wegerkennung mit SLAM, der Hindernisvermeidung und der globalisierten Steuerung.

4. Die bei der Entwicklung von zwei simulierten Robotern zu realisierenden Funktionen, Designideen und Hauptschritte.

5. Entwurf und Implementierung von Verfahrens- und Befehlsprozessen, die es Robotern ermöglichen, ihre Arbeit zu verrichten.

6. Die Übertragung von Farberkennungs- und Positionierungsinformationen ermöglicht es dem Roboterarm, kleine Objekte zu ergreifen.

7. Die Anweisungen move_base und AMCL kontrollieren den Wagen für die Fahrt und den Kartenaufbau der angegebenen Punkte.

8. Aufgabenzuweisung für kooperative Multi-Roboter-Systeme und groß angelegte "Mensch-Roboter-Kooperations"-Projekte im Rahmen des EU-Technologieprogramms.

9. Eine Vision für die Zukunft der Beziehung zwischen Mensch und Maschine, freundlicher Zusammenarbeit, Effektiver und Besser, Höhere Qualität, weniger Abfall, Vereinfachung von Programmiertechniken.

11 Abbildungsverzeichnis