
ROBOTER-ROBOTER INTERAKTION ZUR AUTONOMEN OBJEKTÜBERGABE

Master-Abschlussarbeit,

eingereicht am Fachbereich Campus Minden
der FH Bielefeld

von Robin Rasch,
geboren am 05.02.1991 in Minden

3. Februar 2016

Mit meiner Unterschrift bestätige ich, dass ich die Arbeit selbstständig und nur mit den zugelassenen Hilfsmitteln erstellt habe.

Minden, den

Zusammenfassung

Diese Arbeit befasst sich mit der Entwicklung und Implementierung eines autonomen Roboter Systems. Dieses System bildet das Szenario einer Objektübergabe zwischen zwei selbstständigen Manipulatoren ab. Bei diesen beiden Manipulatoren handelt es sich um den Typ-gleichen Roboterarm des YouBot der Firma Kuka. Einer der Arme ist dabei stationär auf einem Tisch fixiert, der andere auf der mobilen YouBot Plattform.

Nach einer kurzen Einführung über das Nutzen von Robotern im Haushalt, sowie im intelligenten Gebäude, werden zunächst die Grundlagen und folgend ein aktueller Stand der Technik zusammengefasst. Anschließend wird sich diese Arbeit mit den zentralen Aspekten der Entwicklung und Implementierung befassen. Bei der Entwicklung wurden verschiedene Thematiken und Probleme der Robotik und Computer Vision berücksichtigt und bearbeitet. So wird in den folgenden Kapiteln auf das Setup der Roboter eingegangen, sowie den Themen der inversen Kinematik, der Segmentierung und Objekterkennung. Da das System in einem intelligenten Gebäude zum Einsatz kommen soll, ist die Thematik der Vernetzung einzelner Subsysteme und deren Zusammenspiel ebenfalls ein Bestandteil dieser Arbeit. Die Implementierung befasst sich mit der Algorithmik der einzelnen Probleme. Dabei wird Code exemplarisch vorgestellt, die vollständige Implementierung befindet sich im Anhang. Den Abschluss der Arbeit bilden eine Beurteilung und ein Fazit der umgesetzten Lösung, auch wird ein Ausblick über mögliche zukünftige Weiterentwicklungen gegeben.

Abstract

This paper considers the development and implementation of an autonomous robotic system. This system describes a scenario from a handover between two independent manipulators. These two manipulators are equally robot arms from the Kuka YouBot. One is stationary fixed at top of a table, the other one is on a mobile YouBot base.

After a short introduction about the harness of service robots at smart houses, this work outlines the principles and the state of the art. Following this paper attends to the key aspects of the development and the implementation. The development observes different topics and set of problems for robotic and computer vision. There will be a variety of subjects like robover setup, inverse kinematic, segmentation and object recognition in the following chapters. Based on the operation at smart houses, networking and distributed systems are a topic in this paper, too. The implementation sink in algorithmic and solutions for single problems. There will be snippets of programm code, the complete code is at the appendix. A valuation and a conclusion of the implemented solutions, and also a prospect of possible future projects, form the ending of this work.

Inhaltsverzeichnis

1 Einleitung	1
2 Grundlagen	5
2.1 Nomenklatur	5
2.2 ROS: Robotic Operating System	6
2.2.1 Package und Nodes	6
2.2.2 Master und Parameter Server	7
2.2.3 Topics und Messages	8
2.2.4 Services	9
2.2.5 Action-Lib	9
2.3 Kinematik	10
2.3.1 Mechanismus	10
2.3.2 Kinematik	11
2.3.3 Forward Kinematics	11
2.3.4 Inverse Kinematik	12
3 Stand der Technik	14
3.1 Multi-robot system - Konzepte zur Verwaltung mehrerer Roboter	14
3.1.1 Klassifizierung der Koordinierung	14
3.1.2 Probleme der Koordinierung	15
3.1.3 Zusammenfassung	18
3.2 Physically Embedded Intelligent Systems - PEIS	19
3.2.1 Konzept	19
3.2.2 Implementation	19
3.2.3 Zusammenfassung	20
3.3 Grasping und Handover - Arbeiten zu Roboterarmen	20
3.3.1 Grasping - Greifen mit einem Roboterarm	21
3.3.2 Handover - Übergabe zwischen zwei Systemen	22
4 Laboraufbau	23
4.1 Aufbau Teststand	23
4.2 YouBot	24
4.2.1 Manipulator	24
4.2.2 Mobile Plattform	26
4.2.3 Rechner	28
4.3 Sensoren	29
4.3.1 Asus XTion Pro	29
4.3.2 Hokuyo URG-04LX-UG01	29
4.3.3 Argos 3D - P100	30
4.3.4 Netzwerk- und Sensoranbindung	31

5 Entwicklung	33
5.1 Nicht-Funktionale Anforderungen	33
5.1.1 Sicherheit - Safety	33
5.1.2 Zuverlässigkeit	34
5.1.3 Korrektheit	34
5.1.4 Leistung	34
5.1.5 Änderbarkeit	35
5.2 Funktionalitäten	35
5.2.1 Funktion Aufheben	37
5.2.2 Funktion Ablegen	38
5.2.3 Funktion Objekt identifizieren und lokalisieren	39
5.2.4 Funktion Nahfeld Erkennung	40
5.2.5 Funktion Agentenlokalisierung	41
5.2.6 Funktion Übergabe	42
5.3 Agenten	46
5.3.1 Raumüberwachung	46
5.3.2 Nahfelderkennung	47
5.3.3 Lokalisierung	49
5.4 Architektur RATS	50
5.4.1 Konzept	50
5.4.2 RATSACTION und RATSTASK	52
5.4.3 RATSMEMBER	53
5.4.4 RATSCORE	54
5.5 Inverse Kinematik	55
5.5.1 YouBot Kinematiken	55
5.5.2 Vereinfachte inverse Kinematik	57
5.5.3 Weiterentwicklung: Inverse Kinematik	60
5.5.4 Optimierte Pfadplanung	64
5.5.5 Nachtrag Literatur	65
6 Implementierung	66
6.1 Robotersteuerung - RATSYoubot & RATSDummy & RATSRose	66
6.1.1 Ansteuerung der Aktoren	66
6.1.2 Lineare Bahnbewegung	67
6.1.3 Aktionen der Roboter	67
6.2 Raumüberwachung - RATSHawk	69
6.2.1 PCL & ROS	69
6.2.2 Detektion	70
6.2.3 Lokalisierung	71
6.3 Nahfelderkennung - RATSEye	71
6.4 Koordinierung	72
6.4.1 Koordinierung beim RATSMEMBER	72
6.4.2 Koordinierung beim RATSCORE	73

6.5	Bewegung & Lokalisierung der mobilen Plattform	73
6.6	Handoverpoint	74
7	Test und Bewertung	76
7.1	Genauigkeit Inverse Kinematik	78
7.2	Test Übergabeposition	79
7.3	Genauigkeit Navigation	79
7.4	Zusammenfassung der Tests	79
7.5	Robustheit	80
8	Zusammenfassung und Ausblick	81
9	Danksagung	82
10	Literaturverzeichnis	83
A	Datenblätter	89
B	Testdaten Inverse Kinematik	89

Abbildungsverzeichnis

1	Roboter in Seniorenheimen	2
2	Beispiel Szenario 1	3
3	Erweiterung Szenario 1	3
4	Master Service	7
5	Netzwerk Entlastung durch Peer-to-Peer Verbindungen	8
6	Topic Beispiel	8
7	Aufbau Teststand: Draufsicht	23
8	Aufbau Teststand: Seitenansicht	24
9	YouBot Arm Kinematik	25
10	Arbeitsraum YouBot. Bilderquelle:Florek-Jasinska (2015)	26
11	YouBot Base	27
12	Mobile YouBot Plattform mit Sensorplatte. Bilderquelle:Kuka (2015)	28
13	Asus Xtion Pro Live	29
14	Hokuyo URG-04LX-UG01	30
15	Argos 3D - P100	31
16	Schematische Darstellung der Verbindungen	31
17	Use-Case Robotersystem	35
18	Greifer bei der Übergabe	43
19	Sequenzdiagramme für Übergabevarianten	44
20	Raumüberwachung Aufnahme	47
21	Argos3D Punktwolke	48
22	XTion als Naherkennung	49
23	Klassenübersicht RATS	51
24	RATS globaler Speicher	52
25	JSON Action Message	54
26	Graph zur Pfadplanung	65
27	Trajektorien des Greifers	68

Tabellenverzeichnis

1	Nomenklatur	5
2	YouBot Arm Joints	25
3	Gemittelte Messwerte der MoveIt!-Planer-Algorithmen	56
4	Messwerte der Testphase	78

1 Einleitung

Der Demographische Wandel in Deutschland stellt die Gesellschaft und die Politik vor ein großes Problem. Nicht nur der fehlende Nachwuchs, der ein Arbeitnehmerloch hinterlässt, sondern auch eine immer älter werdende Gesellschaft sorgen für viele Fragezeichen. Neben kommunalen Problemen, wie teure Infrastruktur, ist eins der größten Anliegen die Altersarmut. Ein sinkendes Rentenniveau und steigende Kosten werden es in Zukunft unmöglich machen für eine gute Pflegeversorgung zu bezahlen. (Zandonella, 2013) Neben den hohen Kosten in der Pflege sind auch andere Faktoren die zu einer nicht akzeptablen Situation führen. So ist der Pflegeberuf zu unattraktiv für viele junge Menschen, wodurch auch hier ein großes Loch an Arbeitnehmern wahrzunehmen ist. Körperlich anstrengende Arbeit, die in kurzer Zeit ausgeführt werden muss, führen zu vielen physischen und psychischen Erkrankungen der Pfleger. (Badura, 2005)

Einen alternativen Weg bringt die Technik. Einfache zu bedienende Systeme können den Alltag vereinfachen und so auch älteren Menschen ein selbstständiges Leben ermöglichen. Alltägliche Aufgaben müssten nicht mehr von Pflegern übernommen werden, sondern könnten durch Maschinen erledigt werden. Schon heute können einzelne Kleinsysteme Haustätigkeiten wie Staubsaugen und Rasenmähen übernehmen. Ein Vorreiter auf diesem Gebiet ist das Roboterland Japan. Dort überlegte Kobayashi Hisato, Professor für Maschinenbau und Robotik an der Hosei-Universität in Tokio, sich schon 1999 Konzepte für Senioren-Service Roboter. (Wagner, 2009) Nach seinen Vorstellungen sollten dabei Roboter nicht autonom arbeiten, sondern von Familienangehörigen ferngesteuert und überwacht. (Kobayashi, 1999) Neben diesen Konzepten kann man aber auch schon angewandte Robotik in Japans Seniorenpolitik finden. Zwei Musterbeispiele zeigen dabei die unterschiedlichen Anwendungsszenarien für Roboter im privaten Umfeld. *Sinére Kōrien* der Firma Matsushita ist ein digitales Seniorenheim. Neben einer Smarthouse Anbindung gehört auch der Roboterteddy Kō-chan zur Ausstattung der einzelnen Zimmer. Dieser dient als Unterhaltungsroboter und Kommunikationsgerät mit den Pflegern. So kann mit den Kamerasaugen im Teddy eine Aufnahme vom Raum gemacht werden und im Notfall den Pflegern eine Alarmmeldung geschickt werden. Weitere Sensoren, zu Beispiel Gewichtssensoren unter den Betten, geben Informationen über die Abwesenheiten von Patienten. (Wagner, 2009) Ein weiterer Anwendungsbereich für Roboter ist die *robotto serapi* (Robotertherapie). Dabei beschäftigen sich die Senioren mit tierähnlichen Robotern, wie Hunden, Seeroben oder Katzen. Im Zentrum der Therapie steht die Interaktion zwischen Patient und Roboter. Die Robotertherapie soll die Patienten aktivieren und deren Tagesabläufe abwechslungsreich gestalten. Außerdem steigert es die Kommunikation zwischen zwei Patienten, die am selben Roboter arbeiten. (Wagner, 2009)

Nicht nur in Japan, sondern auch in Deutschland wird sich mit dem Thema *Care(rob)bots* befasst. So finden sich unter dem Stichpunkt *Mensch-Maschine-Entgrenzungen* Studien zu der Thematik. Andere Untersuchungen befassen sich mit der Gegenseite, der Akzeptanz der Senioren für Roboter. So ergab eine Befragung der VDE-Studie "Mein Freund der Roboter", dass eine Mehrheit (56%) der Senioren Robotern im Haushalt offen gegenüber stehen und diese einem Pflege-/Altersheim vorziehen würden. Neben den bekannten Staubsauger- und Rasenmärobotern, sind es auch zukünftige Anwendung, wie ein roboterisierter Rollstuhl, die hohe



(a) Roboterteddy Kō-chan Quelle: (Panasonic, 2005)



(b) Roboterkatze zur Therapie Quelle: (Wagner, 2009)

Abbildung 1: Roboter in Seniorenheimen

Akzeptanzwerte erreichen. Die Studie zeigte aber auch, dass Senioren zunächst Robotern skeptisch gegenüberstehen. Spontan lehnten 40 Prozent der Senioren Roboter in ihrem privaten Umfeld ab, 60 Prozent empfanden Robotik sogar als unheimlich. jedoch zeigte sich, dass der Wunsch nach einer selbstständigen Lebensführung ein starker Faktor für die Akzeptanz ist. Dadurch ergibt sich eine Beliebtheit für Serviceroboter. So sind Roboter, die abgrenzbare Tätigkeiten im Haushalt selbstständig erledigen, sehr beliebt. Wichtige Kriterien für die Akzeptanz waren zudem die intuitive Bedienbarkeit, die Robustheit und die Flexibilität gegenüber unterschiedlicher Handicaps. Auch menschliche Faktoren wie Geduld, Verständnis, Höflichkeit und Achtung der Intimsphäre waren den Anwendern wichtig.(Meyer, 2011)

Neben den sozialen Forschungen gibt es in Deutschland auch ingenieurwissenschaftliche Ergebnisse auf diesem Gebiet. So beschäftigt sich das Cluster of Excellence Cognitive Interaction Technology (CITEC) in Bielefeld, zusammen mit der Stiftung Bethel, in der Forschung auf dem Gebiet der Unterstützung für Demenzkranke. Die dort entwickelten Assistenzsysteme helfen den Patienten unter anderem beim Zähneputzen und geben ihnen so ein Stück Selbstständigkeit zurück.

Viele Forschungen und Entwicklungen beschäftigen sich momentan mit der Mensch-Maschine/Roboter Beziehung. Dabei geraten die Roboter-Roboter-Interaktionen in den Hintergrund. Da aber heutige Roboter keine Allround-Lösungen bieten, sondern meist hochgradig spezialisiert sind und nur eine Tätigkeit ausführen können(zum Beispiel Saugroboter, Fensterwischroboter und weitere), ist das Zusammenspiel und die Komposition der Roboter wichtig. Das Szenario in Abbildung 2 zeigt, wie solch eine Komposition aussehen könnte.

Dabei führt jeder Roboter selbstständig und unabhängig von den anderen seine Arbeit aus. Die Kommunikation geht immer von der Zentralen Steuereinheit, dem *Master*, aus. Zwischen den einzelnen Robotern findet kein großer Informationsaustausch statt. Die Änderung an dem Szenario in Abbildung 3 ändert jedoch die Art der Zusammenarbeit und der Kommunikation. Nun ist es nötig, dass die Roboter notwendige Informationen, wie die Übergabepose oder eine Synchronisierung der Gripper, austauschen.



Abbildung 2: Mensch A stößt einen Blumentopf um (1). Ein Kamerasystem dediziert das der Topf umgefallen ist und Erde auf dem Boden liegt. Ein Roboter mit Arm wird gerufen, der zunächst die Vase aufhebt und an ihren Platz zurückstellt (2). Anschließend wird ein Saugroboter aktiviert, der die Erde weg saugt (3). Bei schwereren Verschmutzungen wird noch ein Wischroboter bestellt.

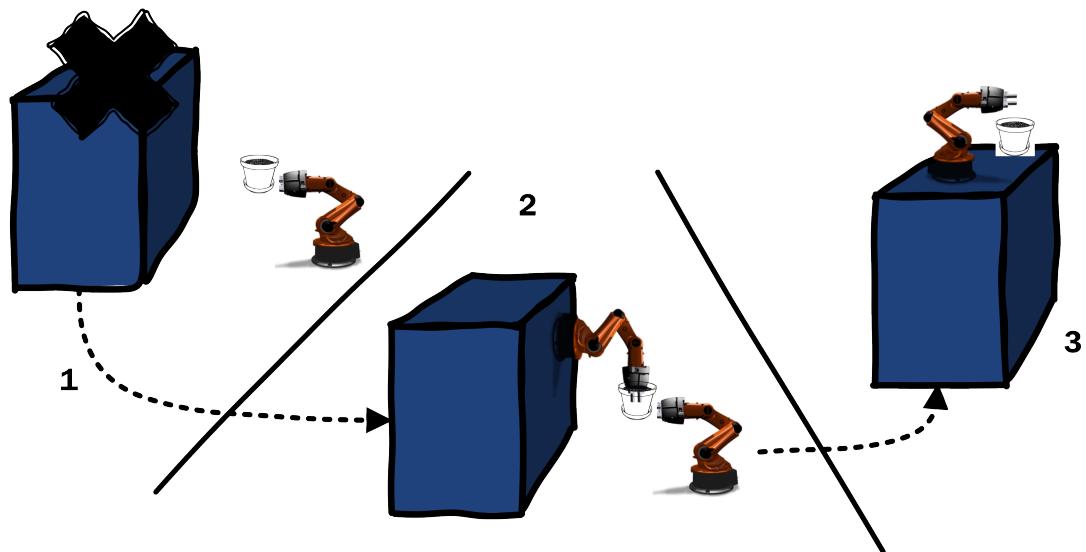


Abbildung 3: Beim Zurückstellen der Vase stellt der Roboter fest, dass er die gewünschte Position nicht erreichen kann(1), weil sein Arm zu kurz ist. Da auf der Anrichte noch ein weiterer Arm steht nimmt er die Kommunikation mit diesem Manipulator auf. Sie kommunizieren einen Übergabepunkt und führen ein Übergabemanöver durch. Arm 2 stellt nun die Vase an die gewünschte Position.

Der zentrale Aspekt dieser Arbeit befasst sich mit der Abstraktion von Szenario 2. Roboter 1 übergibt ein Objekt an Roboter 2. Dabei werden verschiedene Thematiken aufgegriffen, untersucht und erläutert. Das Folgende Kapitel befasst sich mit den Grundlagen dieser Arbeit, es werden einzelne Begriffe aus der Robotik und der Middleware **R**obot **O**perating **S**ystem erläutert, sowie der genutzte Laboraufbau dargestellt. Dabei wird die verwendete Hardware aufgelistet und im Detail betrachtet. Kapitel 3 zeigt schon bestehende Arbeiten im Bereich Robotik, die mit dieser Arbeit in Zusammenhang bestehen. So wird unter anderem der Begriff PEIS (Physically Embedded Intelligent Systems) erläutert und die Verbindung zur Thesis gezeigt. Des Weiteren werden Arbeiten für Multirobot Systeme und Handover-Methoden erklärt. Im darauf folgenden Kapitel TODO

2 Grundlagen

Dieses Kapitel befasst sich mit den Grundlagen für die folgenden Kapitel. Es soll dem geneigten Leser Informationen zum Verständnis der Arbeit geben. Die anschließenden Unterkapitel beschreiben zunächst die verwendeten Mathematischen Ausdrücke und Bezeichner um die Lesbarkeit der späteren Formeln zu erleichtern. Nachfolgend wird die Middleware ROS genauer erläutert und unterschiedliche Funktionen sowie Einschränkungen aufgezeigt, die Auswirkungen auf die entwickelte Softwarearchitektur hatten. Darauf folgt eine Definition des Begriffs der Kinematik, welcher im späteren Verlauf der Arbeit für die Entwicklung der inversen Kinematik benötigt wird. Das abschließende Unterkapitel beschreibt die verwendete Hardware und den Laboraufbau im Detail. Weitere Informationen bezüglich der Robotik und grundlegender Mathematischer Begriffe sind im Buch "Robotics, Vision and Control" von Peter Corke zu entnehmen.

2.1 Nomenklatur

Die folgende Tabelle erläutert die genutzten mathematischen Symbole und orientiert sich an dem Buch "Robotics, Vision and Control" Corke (2011). Von links nach rechts sind das Symbol, die Einheit und die Beschreibung gegeben. Einige Symbole werden mehrfach verwendet und haben je nach Kontext eine andere Bedeutung.

Symbol	Einheit	Beschreibung
T	s	Messintervall
T		homogene Transformation, $T \in SE(2)$ oder $SE(3)$
${}^A T_B$		homogene Transformation zur Darstellung von Koordinatensystem B betrachtet von Koordinatensystem A . Koordinatensystem 0 bezeichnet das Weltkoordinatensystem und muss nicht explizit angegeben werden ${}^0 T_B = T_B()$. Die inverse kann auf zwei Arten betrachtet werden: $({}^A T_B)^{-1} = {}^B T_A$
θ	rad	Winkel im Bogenmaß. Wenn nichts weiteres gegeben ist sind Winkel immer im Bogenmaß zu sehen.
$\boldsymbol{\theta}$	rad	Vektor von Winkel im Bogenmaß.
$\theta_r \theta_p \theta_y$	rad	Roll-Pitch-Gear Winkel, beziehungsweise Rollen
α	°	Winkel in Grad. Wenn nichts weiteres gegeben ist sind Winkel immer im Bogenmaß zu sehen.
v	$m s^{-1}$	Geschwindigkeit.
\boldsymbol{v}	$m s^{-1}$	Geschwindigkeitsvektor. Meist $\in \mathbb{R}^3$.
ω	$rad s^{-1}$	Drehwinkel-Geschwindigkeit.
$\boldsymbol{\omega}$	$rad s^{-1}$	Drehwinkel-Geschwindigkeitsvektor. Meist $\in \mathbb{R}^3$.
\boldsymbol{v}		Geschwindigkeit-Drehung (engl. Twist, Screw). $\boldsymbol{v} \in \mathbb{R}^6$, $\boldsymbol{v} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$
X,Y,Z		Kartesische Koordinaten.
ξ		Abstrakte Darstellung einer 3-dimensionalen Kartesischen Pose $\xi \in \mathbb{R}^6$
${}^A \xi_B$		Abstrakte Darstellung einer relativen 3-dimensionalen Kartesischen Pose, Pose B betrachtet von Pose A
\oplus		Hintereinanderausführung (Addition) zweier Posen
\ominus		Unärer Operator zur Invertierung von Posen

Tabelle 1: Nomenklatur

2.2 ROS: Robotic Operating System

Das Robotic Operating System, kurz ROS, ist ein Middleware-Framework zur Steuerung und Verwaltung von Personal-Robotern. Es verwendet eine Paket und Node Struktur um einzelne Aktoren und Sensoren einzelner Robotersystem zu betreiben. Das ROS dient dabei als Abstraktionsebene für die Entwickler und versucht die Anwendungsebene von der Hardwareebene zu trennen. Das Framework wird am Robotikinstitut Willow Garage in Zusammenarbeit mit der Open Source Robotics Foundation entwickelt, stammt aber ursprünglich vom Stanford Artificial Intelligence Laboratory. Dort wurde 2007 im Rahmen des Stanford-AI-Robot-Projektes die Entwicklung aufgenommen. ROS ermöglicht während der Entwicklung und des Betriebs ein Cross-Plattform System, da es kompatible Versionen für Windows (experimentell), Linux und Mac OS X gibt. Außerdem steht ein Multi(programmier)sprachen System zur Verfügung. So ist es möglich eigene Nodes mit C++, Python oder Java zu entwickeln. Quigley et al. (2009) Neben den Standardkonzepten und -paketen gibt es eine Community, die eigene Pakete auf der ROS-Homepage zur Verfügung stellt. Die folgenden Abschnitte erläutern die ROS-Kernkonzepte, welche für die Entwicklung eingesetzt wurden.

2.2.1 Package und Nodes

Packages dienen der strukturellen Verwaltung aller Dateien im ROS-System. Nodes, Launch-Files, Services, Actions und Messages sind immer an ihr Paket gebunden und können nur unter deren Namen erreicht werden. Für die Verlinkung und die Erreichbarkeit gebauter Pakete ist das ROS-interne Buildsystem *Catkin* zuständig. Es benötigt ebenfalls den **eindeutigen** Package-Namen für den Build und Verwaltungsprozess. Neben den ROS bezogenen Daten können in einem Paket aber auch ROS unabhängige APIs angelegt und verwaltet werden. So werden unter anderem einzelne Pakete für Treiber genutzt. Ein ROS Paket folgt dem "Goldilocks"-Prinzip: Genug Funktionalität um nützlich zu sein, aber nicht zu viel, damit die Verwendbarkeit nicht zu komplex wird. Der Aufbau eines ROS Pakets findet sich in Anhang ??

Neben der strukturellen Verteilung gibt es auch die Funktionale Trennung. Dies wird durch einzelne *Nodes* erreicht. Jeder Node hat einen bestimmten Aufgabenbereich. Erst das Zusammenspiel mehrerer Nodes ergibt ein Robotersystem. So ist ein Node zum Beispiel für die Berechnung der Inversen Kinematik zuständig, ein anderer für die Lokalisierung des Roboters im Raum und ein dritter für das User-Interface. Diese Verteilung der Funktionalitäten hat mehrere Vorteile. Da jeder Node in einem eigenen Prozess gestartet wird ist das komplette System robuster. Jeder Node muss so entwickelt werden, dass er eigenständig weiterlaufen kann, auch wenn ein benötigter Node durch einen Fehler abgestürzt ist. Jeder Node wird mit einem System weit einmaligen Namen aufgerufen unter dem er vom System erreicht und angezeigt werden kann. Nodes kommunizieren untereinander mit Hilfe von Topics (siehe 2.2.3). Soll ein ROS Node gestartet werden kann dieser mit Hilfe des ROS-Befehls aufgerufen werden:

```
rosrun <paket-name> <nodetype-name>
```

Der Paketname und der Nodetype sind erforderliche Felder. Darüber hinaus können Parameter

angegeben werden die für den Node notwendig sind. Eine weitere Möglichkeit einen und mehrere Nodes zu starten ist ein Launchfile (siehe Anhang ??). In diesem können Nodes, Parameter und Argumente aufgelistet werden. Zu beachten ist jedoch, dass die Nodes in einer zufälligen Reihenfolge gestartet werden. Ein Launch-File kann mit folgendem Befehl gestartet werden:

```
roslaunch <paket-name> <launchfile-name>
```

Damit ein Node gestartet werden kann muss vorher ein ROS-Master aktiv sein, mit dem sich der Node verbinden kann. Ist der Node verbunden kann er auf die Topics, Services, Actions und den Parameter Server des Master zugreifen.

2.2.2 Master und Parameter Server

Der ROS Master ist ein zentraler Dienst für ganze System. Er beinhaltet einen Namen- und Registrierungs-Service an dem sich andere Nodes, auch Services, anmelden können. Der Master wird benötigt, damit die Nodes sich gegenseitig im System verbinden können. Der Nachrichten Transport geht nicht direkt über den Master er verbindet nur die beiden Nodes, welche anschließend via einer Peer-to-Peer Verbindung kommunizieren.

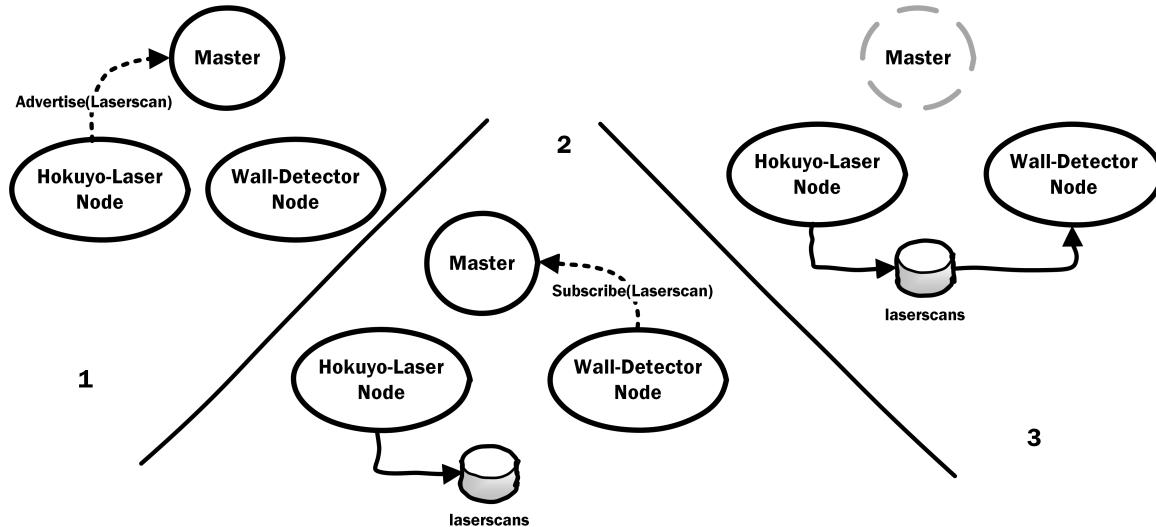


Abbildung 4: Master Service: (1) Der Hokuyo Laser Node meldet sich beim Master an, das er Laserscans zur Verfügung stellt. (2) Der Laser Node stellt seine Daten zur Verfügung und der Wall Detector meldet sich beim Master, dass er Laserscans benötigt. Der Master stellt die Verbindungsinformationen bereit und zieht sich dann zurück (graue Markierung). (3) Die Verbindung zwischen Laser Node und Detector Node ist hergestellt.

Dieses Verfahren reduziert die Netzwerklast stark im Vergleich zu einem Broadcasting-Verfahren. Tests zeigten, dass ein Kamera-Node (openni) ein Netzwerk mit seinen 3D-Punktwolken vollständig auslastet. Dabei wurde die Punktwolke auf der physikalischen Maschine erst erzeugt und auf eine zweite Maschine via WLAN übertragen. Die Delay-Zeit der Wolken war dabei sehr hoch > 1 Sekunde und wurde, je länger die Übertragung dauerte höher. Werden die Daten jedoch auf der Maschine vorverarbeitet und nur noch die wichtigen Daten übertragen, sowie die Übertragungsrate reduziert, ist die Übertragung der Daten unkritisch. Nun gehen nur

noch die Verbindungsdaten zwischen Nodes und Master über das Netzwerk und die reduzierte Datenmenge.



Abbildung 5: Netzwerk Entlastung durch Peer-to-Peer Verbindungen. Links: Ohne Datenvorverarbeitung, Rechts: Mit Datenvorverarbeitung

Neben dem Namen-Service verwaltet der ROS Master auch den Parameter Server. Dieser besteht aus einem Dictionary, einer großen Key-Value-Map, welches global erreicht werden können. Nodes haben so die Möglichkeit vordefinierte Daten ab zugreifen oder selber welche zu schreiben. So können Sensoren oder Aktoren kalibriert und konfiguriert werden. Der Zugriff auf diese Daten hat keine hohe Performance. Er ist für einen statischen Zugriff gedacht und kann mit einer Baum-Struktur geordnet werden, so lassen sich zusammenhängende Daten mit einer Abfrage am Server abrufen.

2.2.3 Topics und Messages

ROS bietet für die Kommunikation zwischen Nodes ein *Topic*-System. Jeder Topic entspricht dabei einem benannten Bus, auf welchen anonyme *Publisher* und *Subscriber* zugreifen können. Die Daten können dabei als TCP-Pakete (TCPROS) oder als UDP-Pakete(UDPROS) gesendet werden. Ein Topic besitzt einen eindeutigen Namen im System und einen eindeutigen Typen, der anhand des übersendeten Message-Typen definiert wird. Jede Topic kann nur einen Message-Typen versenden. Dieser Type ist nicht im Master bekannt, jedoch können sich Subscriber nur mit dem entsprechenden Typen anbinden. Die Anzahl der Subscriber und Publisher für einen Topic ist nur durch die Systemleistung beschränkt.



Abbildung 6: Beispiel für eine Topic mit dem Namen `/topic/` und dem Type `String`. Ein Publisher schreibt Daten in den Bus, ein Subscriber liest diese aus. Die Verbindung eines zweiten Subscribers schlägt auf Grund des falschen Types fehl.

Publisher veröffentlichen ihre Daten in den Bus und haben nur einen schreibenden Zugriff. Sie

bekommen kein Feedback an wen die Daten gesendet worden sind oder ob die Daten überhaupt empfangen worden sind. Der erste Publisher, der sich auf einem Topic verbindet gibt den Typen vor.

Subscriber lesen die Daten vom Bus und geben sie für die Weiterverarbeitung weiter. Sie bekommen normalerweise keine Informationen über den Absender der Daten. Subscriber können sich nur bei einem Topic registrieren, wenn der Message-Type stimmt.

Subscriber und Publisher haben jeweils eigene Cache-Systeme die Nachrichten vor- oder zurückhalten können. Ein Topic ist ein einfachergerichteter Streaming-Dienst ist. Da in einem Roboter-System auch Remote Procedure Calls benötigt werden, um zum Beispiel Antworten auf Anfragen zu erhalten oder synchronisierte Aufgaben zu erledigen, wurde das Service-Paket als weiteres Kernkonzept angelegt.

2.2.4 Services

Services ermöglichen es einen Remote Procedure Call Node übergreifend durchzuführen. Dazu muss Node A seinen Service am Master registriert haben und einen Service Server gestartet haben. Node B kann nun mit einem Service Client auf den registrierten Service zugreifen und eine Anfragen starten. Während die Anfrage verarbeitet wird, kann Node B nun je nach Anforderung blockieren oder weiter arbeiten. Node A verarbeitet nun die Anfrage und antwortet Node B mit dem Ergebnis. Der Service selbst, sowie die Parameter und die Rückgabewerte, sind für jeden Service eindeutig und werden vorher via zwei Messages, eine für die Request und eine für die Response, festgelegt. Analog zu den Topics werden Services am Master mit einem eindeutigen Namen angemeldet unter dem sie erreichbar sind. Dies ermöglicht auch einen einfachen Austausch von verschiedenen Implementierungen.

2.2.5 Action-Lib

Ein aktiver Service schickt erst eine Rückmeldung an den Aufrufer, wenn er seine Aufgabe erledigt hat. Um auch einen Zwischenstand schicken zu können wurde die Actionlib eingeführt. Dieses Paket ist eigentlich keine Entwicklung der ROS-Entwickler sondern ein Community-Package. Inzwischen wurde es aber von der OSR-Foundation übernommen und als eins der Kernkonzepte eingestuft.

Analog zu den Services besitzt auch die Action-Lib einen Server- und einen Client Teil. Der Client fordert vom Server eine Aktion an, dabei übergibt er ein *Goal*. Dieses beinhaltet die benötigten Parameter für den Server. Der Server arbeitet die Anfrage ab, dabei kann er auch *Feedback*, Zwischenstände, zurückschicken. Ist die Anfrage abgearbeitet schickt der Server ein *Result* an den Client zurück. Wie auch beim Service ist die Art der Aktion eindeutig. Die Anzahl der Parameter bei Goal, Feedback und Result sind fest definiert und können zur Laufzeit nicht geändert werden.

2.3 Kinematik

Dieses Unterkapitel befasst sich mit dem Begriff der Kinematik und den dazu gehörigen Begriffen des Mechanismus und der inversen Kinematik. Außerdem werden einige mathematische Lösungswege für verschiedene Probleme innerhalb der genannten Themen genannt.

2.3.1 Mechanismus

Ein Mechanismus ist der Grundbegriff von beweglichen Körpern. Dabei wird zwischen *Gliedern* (engl. Link, $L = \text{Set of Links}$) und *Gelenken* (engl. Joint, $J = \text{Set of Joints}$) unterschieden. Glieder sind die starren Körper eines Mechanismus und sind durch Gelenke miteinander verbunden. Ein Glied ist nicht auf ein Gelenk beschränkt, sondern kann mehrere haben ($N_{joint} \in \mathbb{N}_0$). Auch die Verbindung zwischen zwei Gliedern kann durch mehrere Gelenke realisiert sein. Ein Gelenk wiederum ist auf genau zwei Glieder beschränkt ($N_{link} = 0$) und ermöglicht so eine eingeschränkte relative Bewegung zueinander. Gelenke werden nicht als eigenständige physikalische Körper gesehen. Gelenkhälften, zum Beispiel die Schenkel eines Kippscharniers, werden als Bestandteil des damit verbundenen Glied betrachtet.

Glieder können mit zwei Parametern definiert werden, der Länge a und der Verdrehung α :

$$l_i \in L$$

$$a_i = \text{Länge von } l_i \quad (1)$$

$$\alpha_i = \text{Verdrehung von } l_i \quad (2)$$

Gelenke werden ebenfalls mit 2 Parametern definiert. Der Gelenk-Offset beschreibt den Abstand zwischen den zwei Gliedern eines Gelenks auf der Gelenkkachse. Der Gelenkwinkel beschreibt die Rotation zwischen den beiden verbundenen Glieder im Bezug zur Gelenkkachse. Corke (2011)

$$j_i \in J$$

$$d_i = \text{Gelenk-Offset von } j_i \quad (3)$$

$$\theta_i = \text{Gelenkwinkel von } j_i \quad (4)$$

Im Anschluss werden nur noch die englischen Begriffe Link und Joint benutzt.

2.3.2 Kinematik

Die *Kinematik* eines Mechanismus beschreibt die Bewegungsmöglichkeiten der Links relativ zueinander. Es wird dabei abstrahiert, ob ein Joint motorisch angetrieben oder passiv bewegt wird. Die Kinematik betrachtet bei der Bewegung auftretende *Geschwindigkeiten* und *Beschleunigungen*. Auftretende Kräfte werden nicht in der Kinematik, sondern in der *Dynamik* untersucht. Zentrale Aspekte dabei sind die Trägheits- und Schwerkraft.

Die *Kinematische Kette* beschreibt den Aufbau eines speziellen Mechanismus. Dabei ist die Anzahl der Joints pro Link auf maximal 2 beschränkt ($N_{joint} \leq 2$). Besitzt jeder Link genau zwei Joints gilt die Kette als geschlossen, ansonsten als offen.

$$\text{Kinematische Kette} = \begin{cases} \text{geschlossen} & \forall l \in L | l_{joints} = 2 \\ \text{offen} & \text{für Rest} \end{cases} \quad (5)$$

Für diese Arbeit ist nur die offene Kette interessant, da die Manipulatoren der meisten Roboter an beiden Enden (Base und End-Effektor) frei sind. Dadurch ergibt sich für $N_{link} = N_{joint} + 1$. Typischerweise wird der erste Joint mit dem Index 1 versehen und der erste Link mit dem Index 0. Link 0 ist die *Base* des Manipulators und Link N beinhaltet den *End-Effektor*. Durch die Bezeichnung ergibt sich, dass Joint j die Links j-1 und j mit einander verbindet und den Link j bewegt. Corke (2011)

Die Transformation eines Link-Koordinatensystems j-1 zu Link j wird durch elementare Rotation und Translation beschrieben:

$${}^{j-1}A_j(\theta_j, d_j, a_j, \alpha_j) = T_{Rz}(\theta_j)T_z(d_j)T_x(a_j)T_{Rx}(\alpha_j) \quad (6)$$

Durch die Vereinigung der einzelnen Matrizen ergibt sich:

$${}^{j-1}A_j(\theta_j, d_j, a_j, \alpha_j) = \begin{pmatrix} \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ \sin\theta_j & \cos\theta_j \cos\alpha_j & -\cos\theta_j \sin\alpha_j & a_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (7)$$

2.3.3 Forward Kinematics

Die Vorwärts-Kinematik (engl. Forward Kinematic) beschreibt die "vorwärts" Rechnung der Kinematik. Aus den gegebenen Joint-Informationen wird die Pose des End-Effectors berechnet. Hier reichen die Daten für den Gelenkwinkel θ_j bei Drehgelenken und über den Winkel-Offset d_j bei Schiebe-/ Schubgelenken, da die restlichen Werte für den Mechanismus als konstant angesehen werden können. Die Forward Kinematic wird häufig als Funktion K angegeben Corke (2011)

$$\xi_E = K(\mathbf{q}) \quad (8)$$

q entspricht dabei der aktuellen Joint-Konfiguration und ξ_E der Pose des End-Effectors. Durch die Kombination der Transformation aus Gleichung 6 ergibt sich für einen Manipulator mit N-JointsCorke (2011)

$$\xi_E \sim {}^0T_E = {}^0A_1 {}^1A_2 \dots {}^{N-1}A_N \quad (9)$$

2.3.4 Inverse Kinematik

Die Inverse Kinematik berechnet die Inverse der Forward-Kinematic. Sie bestimmt die möglichen Joint-Konfigurationen \mathbf{q} für eine Zielpose. Im Gegensatz zur Forward-Kinematic ist die Inverse Kinematik nicht eindeutig. Eine einfache Veranschaulichung kann man am menschlichen Körper sehen. Fixiert man das Handgelenk an einer Position kann man den Ellbogen in verschiedene Positionen bewegen ohne die Position der Schulter zu ändern. Für die Inverse Kinematik ergibt sich folgende FunktionCorke (2011)

$$\mathbf{q} = K^{-1}(\xi) \quad (10)$$

Aus Gleichung 8 und 10 ergibt sich auf Grund der Eindeutigkeit:

$$\xi = K(K^{-1}(\xi))$$

Aber **nicht**

$$\mathbf{q} = K^{-1}(K(\mathbf{q}))$$

Um eine Inverse Kinematik zu lösen gibt es drei Methoden: geometrisch (analytisch), numerisch und heuristisch.Steidl (2011)

Die geometrische Lösung nutzt die einfache geometrische Abstraktion des Mechanismus und versucht diesen mit trigonometrischen Funktionen abzubilden. Damit auch hier mehrere Lösungen bestimmt werden können, werden verschiedene Konfigurationen für bestimmte Abstraktionsmodelle geschaffen und alle Konfigurationen berechnet. Am Beispiel des Menschen wären mögliche Konfigurationen: Ellbogen oben oder Ellbogen unten. Bei der Verwendung der trigonometrischen Funktionen muss beachtet werden, dass die Umkehrfunktionen in bestimmten Bereichen nicht eindeutig, ungenau oder gar nicht definiert sind. Empfohlen wird dafür die Verwendung von atan2, der in den meisten Programmiersprachen vorhanden ist. Diese Methodik funktioniert nur für einfache Mechanismen bei denen möglichst alle Joints die selben Achsen haben.

Die numerische Lösung versucht durch Iterationen eine Näherung an die gewünschte End-Pose zu erreichen. Das ganze fällt in die Thematik der Optimierung. Für die numerische Lösung gibt es unterschiedliche Algorithmen. Dazu gehören Jacobian Transpose, Pseudo Inverse und Damped Least Square. Die numerischen Lösungen sind im Vergleich zur analytischen langsamer und ungenauer, jedoch lassen sich mit ihr größere und komplexere Manipulatoren berechnen.Steidl (2011)

Die heuristischen Lösungen nutzen abstrakte Bezüge und Vorhersagen für Veränderungen der End-Pose im Bezug zur Joint-Konfiguration um mögliche Konfigurationen zu bestimmen. Eine anschließende Güteberechnung (zum Beispiel euklidische Distanz) gibt an, ob eine Konfiguration verworfen oder angenommen wird. Wie für die numerische Lösung gibt es auch bei der heuristischen Lösung mehrere Algorithmen. Diese sind unter anderem Cyclic Coordinate Descent und Lagrange-Multiplier. Steidl (2011)

3 Stand der Technik

In diesem Kapitel werden Themen bezogene Forschungen vorgestellt und untersucht. Da es zu der genauen Thematik kaum Forschungen gibt, werden vor allem Teilespekte der Arbeit aufgezeigt. Dazu gehört die Definition und Entwicklung von Multi-Robot Systemen. In dieser werden die Kategorisierung von Multi-Robot Systemen und Frameworks, sowie Architekturen, für die selbigen vorgestellt. Danach liegt der Blick unter anderem auf der Arbeit von Mathias Broxvall, in dieser wird die Einbindung eines Roboters in eine intelligente Umgebung untersucht. Darauf folgt ein Einblick in die Thematik des Greifens, sowie der Übergabe zwischen verschiedenen Teilnehmern.

3.1 Multi-robot system - Konzepte zur Verwaltung mehrerer Roboter

Die Verwendung mehrerer individueller Roboter ist Forschungsthema seit 1986 und ist ein weit gestecktes Feld. Frühere Themen sind zelluläre Roboter, Motion Planning, Schwarmrobotik und Architekturen. Nach dem neuen Robotik Paradigma der behavior-based Kontrolle orientieren sich viele Forschungen an der Biologie und versuchen Abbildungen der sozialen Strukturen von Insekten und anderen Schwarmtieren zu nutzen. Parker (2003a) Dabei wurden zum Beispiel die Schwarmbildung (siehe Hayes and Dormiani-Tabatabaei (2002)), die Futtersuche (Balch (1999)) und die Spurverfolgung (Vaughan et al. (2000)) betrachtet. Viele Probleme von Multi-Robot Systemen (MRS) haben oft einen Bezug zu Problemen aus anderen Forschungsgebieten und können durch die Anzahl an Robotern meist robuster und schneller gelöst werden. Weitere große Forschungsbereiche sind die verteilte künstliche Intelligenz und der Multi-Agent Ansatz. Beide befassen sich mit der Kooperation zwischen Systemen. Erste Ergebnisse dieser Bereiche wurden mit einer Prioritätsordnung der Subtasks erreicht Durfee et al. (1987). Spätere Arbeiten betrachteten das Gruppieren von Systemen, die Aufgaben miteinander bearbeiteten Shehory and Kraus (1998) und Lau and Zhang (2003). Diese Gruppen, auch Koalitionen genannt, lösen Aufgaben, die nicht an einen einzelnen Roboter gestellt werden können. Die Arbeit Vig and Adams (2005) überführte dann die bekannten Lösungen der Koalitionen in die Probleme der MRS. Weitere Themen aus dem Bereich Multi-Agent, Team-Work (Pynadath and Tambe (2003)), Potenzial-Management (Timm and Woelk (2003)) und Normen (Boella (2002)), wurden für die Koordinierung von MRS angewandt (Lundh et al., 2006).

3.1.1 Klassifizierung der Koordinierung

Die Koordinierung von Robotern beschreibt die Organisation von Bewegungen, welche die Roboter zusammen ausführen sollen. Potenzielle Probleme sind dabei die Fragestellungen wie die Koordination organisiert werden soll, wer organisieren soll oder wie viel Koordination nötig ist um eine Aufgabe zu erledigen.

Bei den Organisationsmöglichkeiten unterscheidet man zwischen zentralisiert und verteilt. In einem stark verteilten System sind die einzelnen Roboter unabhängig und selbstständig. Entscheidungen werden meistens zwischen den Systemen ausgehandelt. Diese Variante ist gegenüber

einem zentralen System robuster, da kein Single-Point-of-Failure auftritt. Dafür ist es meist komplexer eine optimale Lösung für ein Problem zu finden. Außerdem ist der Kommunikationsaufwand wesentlich höher. In einem stark zentralisiertem System übernimmt ein System die Führung und alle Entscheidungen. Das ganze kann dann als ein Robotersystem mit ferngesteuerten Sensoren und Aktoren abgebildet werden. Nachteilig ist hier der beschriebene Single-Point-of-Failure: Fällt das zentrale System aus, kann das ganze System nicht mehr arbeiten. Außerdem sind solche Systeme meist langsamer, da die ganzen Informationen erst auf einer Recheneinheit zusammengefasst werden müssen und anschließend dort ausgewertet werden. Dadurch können aber auch einfacher optimierte Lösungen gefunden werden (Lundh et al., 2006). Weitere Informationen finden sich in den Arbeiten Farinelli et al. (2004) und Dias and Stentz (2003).

Ein weiterer Aspekt der Koordinierung von MRS ist der benötigte Grad an Koordinierung. Dabei wird in der Literatur zwischen den Begriffen der engen (*tight*), beziehungsweise eng-gekoppelt oder starken, Koordinierung und der losen (*loosely*), lose-gekoppelt oder schwache, Koordinierung unterschieden. In der Arbeit Farinelli et al. (2004) ist die starke Koordinierung als eine Koordinierung beschrieben, die auf einem Koordinationsprotokoll aufbaut. Ein Koordinationsprotokoll beschreibt eine Menge an Regeln die spezifizieren wie Roboter miteinander interagieren. Eine schwache Koordinierung baut nicht auf einem solchen Protokoll auf. Die Begriffe eng (eng-gekoppelt) und lose (lose-gekoppelt) definieren die Quantität der Koordinierung. Ein eng-gekoppeltes MRS benötigt viel Koordinierung, ein loses MRS eher wenig. Da die Begriffe viel und wenig ungenau und schwer zu definieren sind, existieren in der Literatur verschiedene Definitionen. Die Arbeit Kalra et al. (2004) unterteilt die Koordinierung in *loosely*, *moderately* und *tightly* anhand der Zerlegung der Aufgaben (Tasks) in Unteraufgaben (Subtasks). Bei einer losen Koppelung können alle Tasks in Subtasks zerlegt werden, die von einem Roboter ausgeführt werden können. Mäßige Kopplungen begrenzen die Koordination auf zeitlich abhängige Subtasks. Dabei beschränkt sich die Koordination nur auf den Startzeitpunkt (t_0) eines Subtasks. Die Ausführung des Tasks wird nicht koordiniert. Eine enge Kopplung erfordert eine permanente Koordination und Tasks können nicht in Subtasks zerlegt werden, die von einem Roboter abgearbeitet werden (Lundh et al., 2006).

Eine weitere Klassifizierung der Literatur bezieht sich auf die Tasks. In der Definition von Gerkey and Matarić (2004) wird zwischen *single-robot tasks* (SR) und *multir-robot tasks* (MR) unterschieden. SR benötigen genau einen Roboter für die Ausführung, zum Beispiel eine Bewegung an eine gewünschte Position. MR können mehrere Roboter benötigen. Außerdem werden Roboter eingegliedert. *Multi-Task* (MT) Roboter können gleichzeitig mehrere Tasks ausführen, *Single-Task* Roboter führen immer nur Tasks nacheinander aus (Lundh et al., 2006).

3.1.2 Probleme der Koordinierung

Bei lose-gekoppelten MRS ist das Hauptproblem die Zuweisung der Tasks, da jeder Task ein SR ist. Dieses Problem wird als *Task allocation* bezeichnet. Darauf baut das Problem der Rollenzuweisung auf, wenn im MRS mit einem Rollensystem gearbeitet wird. Task Allocation beschreibt wie eine Anzahl von Tasks auf eine Anzahl von Systemen verteilt wird. Die einfachste Aufteilung wurde schon 1960 in Gale (1989) beschrieben und seitdem in zahlreichen Arbeiten untersucht:

Ein Task kann nur einem Agenten zugewiesen werden und jedem Agenten kann nur eine Task zur Zeit abarbeiten.

Ein bekannter Lösungsansatz ist das ALLIANCE-System von Lynne E. Parker aus dem Jahr 1998 Parker (1998). Dieser Algorithmus nutzt vier einfache Schritte zur Vergabe der Tasks. In einer Pärchen-Liste von Task-Agent sind alle möglichen Kombinationen im MRS gespeichert und werden nach der Nützlichkeit bewertet. (1) Finde das am besten bewertete Pärchen (2) weise die Task dem Agenten zu (3) entferne das Pärchen aus der Liste (4) ist die Liste nicht leer beginne mit (1), ansonsten terminiere. Der Vorteil von ALLIANCE liegt nicht in der Zuweisung, sondern in der Neuzuweisung. Dazu werden zwei Aspekte der Verhaltenssteuerung genutzt: Ungeduld (*impatience*) und Fügung (*acquiescence*). Ungeduld ermöglicht es, dass ein Agent eine Task eines anderen Agenten übernimmt. Wenn Roboter A die Eindruck bekommt, dass Roboter B die Task nicht lösen kann, wird er ungeduldig und übernimmt die Aufgabe. Die Fügung funktioniert ähnlich, nur das Roboter A feststellt, dass er selbst die Task nicht oder nicht gut genug lösen kann und diese an Roboter B abgibt. Weitere Forschungen gibt es unter anderem bei Werger and Matarić (2000).

Ein weiterer bekannter Ansatz für das Problem ist das *Contract Net Protocol* (CNP) von Davis und Smith aus dem Jahr 1983. Davis and Smith (2003). Das CNP beruht auf einem Auktionshausprinzip. Tasks werden von einem Dealer als Broadcast angeboten. Agenten im MRS haben die Möglichkeit privat auf die Tasks zu bieten. Dabei werden meist Ergebnisse des auszuführenden Task geboten. Diese Ergebnisse können unter anderem Energie- und Zeitkosten beinhalten. Der Dealer entscheidet wer den Zuschlag bekommt. Er kann auf Grund des Gebotes den bestmöglichen Agenten für die Task bestimmen. Die Auswahl kann dynamisch, je nach Priorität (zum Beispiel Zeit vor Energiekosten), ausgewählt werden. Varianten die auf CNP basieren sind GOFER Caloud et al. (1990), M+ Botelho and Alami (1999), TraderBots Dias and Stentz (2000) und Sold! Gerkey and Matarić (2002).

Bei Rollenbasierten Ansätzen werden einzelnen Agenten Rollen zugewiesen und dadurch in ihrer Funktionalität beschränkt. Agenten mit den gleichen Rollen haben die gleichen Funktionalitäten. Ansätze dafür finden sich in den Arbeiten Frias-Martinez et al. (2005), Vail and Veloso (2003) und Stone and Veloso (1999).

Bei eng-gekoppelten MRS ist die primitive Zerlegung wie bei losen Systemen nicht möglich. Dadurch fällt die Koordinierung nicht nur in der Startphase an, sondern auch während des Prozesses. Zentrale Probleme sind nicht nur der Ausführer, sondern die Frage nach der Ausführung, also wie eine Task erledigt wird. Ein möglicher Anwendungsfall ist die Arbeit von Saffiotti et al. (2000). Bei diesem Ansatz arbeiten Roboter in einer Formation. Jeder Roboter besitzt dabei zwei Vorsätze, das Team-Ziel und das Individualziel. Das Team-Ziel ist das übergeordnete Gemeinschaftsziel, zum Beispiel das Tragen eines Objektes. Das Individualziel sind Roboter-eigene Interessen, zum Beispiel die Kollisionsvermeidung. Typische Lösungsansätze für solche Anwendungen ist das Führer-Folger Prinzip, bei dem ein Führer bestimmt wird, dieser übernimmt Entscheidungen, die Folger folgen dem Führer. Dies ist bei komplexeren Anwendungen jedoch nicht ausreichend, sodass eine Planung notwendig wird. Dabei treten die beiden zentralen Fragen auf: **Wer macht was?** und **Wie wird es gemacht?**.

Das Robotik Institute der Carnegie Mellon University hat eine Forschungsgruppe, die sich mit der Frage nach dem *wer* beschäftigt. Dabei werden drei Ansätze verfolgt. Die erste beschäftigt sich mit der Aufgabenverteilung bei komplexen Tasks, die zweite mit der Aufgabenverwaltung bei eng-gekoppelten MRS und die dritte mit Architekturen für enge MRS. Als Grundlagen dient den dreien die schon erwähnte Arbeit von Dias und Stentz, den TraderBotsDias and Stentz (2000) die auf dem CNP aufbauen. Die Arbeit Zlot and Stentz (2005) befasst sich mit der Aufgabenverteilung bei komplexen Tasks. Dabei werden komplexe Tasks als Aufgaben mit mehreren Lösungswegen definiert. Die Erstellung eines Plans für eine komplexe Task liegt meist in *NP-hard*. Im Gegensatz dazu stehen die einfachen (simple) Tasks, die einfach vorwärts gerichtet sind. Ähnlich dem Auktionsverfahren wurde so ein Auktionsansatz für komplexe Tasks entwickelt. Anstatt eine primitive Task anzubieten werden Taskbäume angeboten. Dabei entspricht die Wurzel dem abstrakten komplexen Task, die Knoten den komplexen Subtasks, die Blätter den primitiven Subtasks und die Kanten dem Koordinationsaufwand. Die einzelnen Agenten können nun auf die Blätter bieten. Die zweite Forschungsrichtung befasst sich mit einem Marktansatz für eng-gekoppelte Systeme und wird in den Arbeiten Kalra et al. (2004) und Kalra et al. (2005) behandelt. Das Konzept trägt den Namen Hoplites und lässt sich aus dem antiken Griechenland herleiten. Dort waren die Hopliten eine der besten Infanterieeinheiten und auf komplexe Manöver spezialisiert. Bei diesem Framework werden Agenten rückwirkend bewertet. Die Bewertung erfolgt nach Beendigung eines Task und gibt an wie sehr dieser Agent dem Gemeinschaftsziel gedient hat. Des weiteren nutzt das Framework aktive und passive Koordinierung ein. Passive Koordinierung wird für simple Tasks eingesetzt und solange genutzt, bis die aktive Koordinierung bessere Vorhersagen bietet. Das Framework selbst hat keinen Planer für komplexe Aufgaben, sondern entscheidet nur, wann welcher Planer eingesetzt werden soll. Der dritte Aspekt behandelt Architekturen für eng-gekoppelte MRS und heterogenen Roboter. Die Arbeiten Simmons et al. (2001) und Simmons et al. (2002) zeigen eine drei Schichten Architektur. Diese Schichten (Planungs-, Ausführungs- und Verhaltensschicht) können innerhalb eines Roboters miteinander kommunizieren. Außerdem können die einzelnen Schichten auch Roboter übergreifend kommunizieren. Also Planungsschicht von Roboter A mit Planungsschicht von Roboter B, aber nicht mit Ausführungsschicht von Roboter B (Lundh et al., 2006).

Die schon zitierte Lynne E. Parker leitet die Forschungsgruppe des *Distributed Intelligence Laboratory* an der University of Tennessee. Die zentralen Forschungen der Gruppe befassen sich mit der automatischen Koordination von Roboterteams. 2003 und 2004 präsentierte Parker zwei Arbeiten Parker (2003b) und Parker et al. (2004) in denen Netzwerke von über 70 Robotern eng koordiniert wurden. Dabei wurde zwischen sensorarmen und -reichen Robotern unterschieden. Für die enge Kopplung wurde dabei ein Führer-Folger Prinzip gewählt. Als Führer wurden die sensorreichen Roboter ausgewählt. In der Arbeit wurden zwei Tasks bearbeitet. Die erste wurde als Long-Distance-Navigation beschrieben, dabei übernimmt ein Führer die Kontrolle und berechnet einen Weg mit seinen Sensoren. Die sensorarmen Roboter folgen dem Führer. Am Ziel angekommen bauen alle Roboter ein gemeinsames Sensornetzwerk auf und überwachen den Zielraum. Die Roboter innerhalb des Sensornetzwerkes versorgen sich dabei automatisch mit den benötigten Informationen. Die Konfiguration für dieses MRS ist jedoch statisch durch die Entwickler vorgegeben. In der anschließenden Veröffentlichung Parker et al. (2005) stellte

die Forschungsgruppe *ASyMTRe* (Automated Synthesis of Multi-robot Task solutions through software Reconfiguration) vor. Dieses Konzept, beruhend auf der Schemen Theorie aus Arkin (1987), ermöglicht einem MRS eng-gekoppelte Tasks durch Informationsaustausch zu lösen. Dazu werden verschiedene Schemen so miteinander verbunden, dass die Informationen von den Umweltsensoren zu den Motor-Schemen gelangen. ASyMTRe nutzt dazu einen Algorithmus, der alle Informationen für den am wenigsten fähigen Roboter sammelt und diese anschließend an alle verteilt. Anschließend wird der Roboter als Führer ermittelt, der den größten Informationsgehalt bei kleinster Sensorik hat (Lundh et al., 2006) .

3.1.3 Zusammenfassung

Dieses Kapitel zeigte Arbeiten und Konzepte rund um das Thema Multi-Robot Systeme auf. In dieser Zusammenfassung sollen die benötigten Informationen für diese Arbeit extrahiert und Entscheidungen für die eigene Entwicklung getroffen werden. In diesem Kapitel wurden zunächst die Begriffe Koordination und Konfiguration durch verschiedene Arbeiten definiert. Außerdem wurden die Begriffe verteilte und zentrale, starke und schwache, sowie enge und lose Koordination erläutert. In den Arbeiten stellte sich heraus, dass zentrale Steuerungen durch einen Single-Point-of-Failure durch Störungen stärker betroffen sind als verteilte Koordination. Diese wiederum benötigt einen größeren Aufwand der Algorithmen und der Konfiguration. Daraus folgt für diese Arbeit, dass ein erster Entwurf eine zentrale Koordinierung haben wird, da der Aufwand beschränkt ist. Jedoch soll eine Schnittstelle für eine verteilte Koordinierung vorgesehen werden. Aus der Literatur ergab sich, dass starke Koordination auf einem Koordinationsprotokoll aufbauen und schwache auf dynamischen Algorithmen, die beteiligte Robotersysteme auswählt. Da in dieser Arbeit nur zwei Roboter verwendet werden und diese zwangsweise miteinander arbeiten müssen ist die Auslegung eines dynamischen Zuweisungsmodell überflüssig, soll aber auch in Form von Schnittstellen für weitere Aufgaben vorgesehen werden. Die Begriffe der eng- und lose-gekoppelten MRS wird durch verschiedene Definitionen unterschiedlich ausgelegt. Für diese Arbeit wird auf die Definition von Kalra et al. (2004) zurückgegriffen. Da in dieser die Begriffe anhand der Zerlegung von Tasks definiert werden, kann die Kategorisierung für dieses System erst nach der Anforderungsanalyse geschehen. Auch die Eingliederung der Tasks in Single-Robot und Multi-Robot Tasks kann erst nach der Anforderungsanalyse erfolgen. Dies betrifft ebenfalls die Entscheidung für Single-Task oder Multi-Task Roboter. Auf ein Rollensystem soll zunächst verzichtet werden, da auch dies für dieses kleine System unnötiger Aufwand ist. Eine Erweiterung die für dieses System vorgesehen ist, beschreibt die Arbeit Davis and Smith (2003) mit dem CNP. Die Vergabe der Aufgaben mit Hilfe eines Auktionshauses kann in zukünftigen Entwicklungsschritten auch für komplexere, sogar eng-gekoppelte, MRS genutzt werden. Im folgenden Kapitel wird die Architektur PEIS vorgestellt, die neben Robotersystemen auch weitere Sensoren und Aktoren einschließt. Diese Architektur soll als Ausgangsarchitektur dienen.

3.2 Physically Embedded Intelligent Systems - PEIS

Ähnlich der Einleitung zu dieser Arbeit beschreiben auch Broxvall und Kollegen ihre Zukunftsvision von Robotern im sozialen und alltäglichen Umfeld. Im Fokus ihres Szenarios stehen dabei die unterschiedlichen Sensoren und Aktoren in einem intelligenten Gebäude, im Zusammenspiel mit Robotern. Dieses Zusammenspiel ist das zentrale Konzept der Arbeit von Broxvall und wird als PEIS-Ökologie bezeichnet. PEIS steht für Physically Embedded Intelligent Systems und umfasst die Kernkonzepte von künstlicher Intelligenz, allgegenwärtigem Rechnen und Robotik. Die Grafik ?? zeigt die Überschneidung dieser Themenbereiche, sowie die Position an denen Broxvall und Kollegen ihre Arbeit sehen. Broxvall and Saffiotti (2005)

3.2.1 Konzept

Das Konzept der PEIS-Ökologie beruht auf drei Grundlagen. Zunächst wird die gleichmäßige Notation von PEIS (*uniform notion of PEIS*) vorgestellt. Jedes technische Gerät ist eine mögliche PEIS-Komponente, wenn es über eine Recheneinheit und eine Kommunikationsschnittstelle verfügt, sowie mit der Umwelt interagieren kann. Dies betrifft sowohl Aktoren als auch Sensoren und reicht vom einfachen Toaster über den intelligenten Feuermelder bis zum komplexen Roboter. Das zweite zentrale Konzept ist das einheitliche Kommunikationsmodell (*uniform communication model*). Dieses Modell realisiert die Dynamik eines PEIS. Zu jedem Zeitpunkt können sich neue PEIS-Komponenten anmelden und bestehende Komponenten das Modell verlassen. Das Modell „ermöglicht eine einheitliche Kommunikation und versteckt die Heterogenität zweier PEIS-Komponenten unabhängig vom physikalischen Kommunikationsmedium.“ Broxvall and Saffiotti (2005) Die dritte Kernkompetenz ist das einheitliche Kooperationsmodell (*uniform cooperation model*). Dabei beschreiben Broxvall und Kollegen die Verbindung funktionaler Komponenten. Diese Verbindung ermöglicht es jeder PEIS-Komponente Funktionen anderer PEIS-Teilnehmer anzufordern. Diese drei Konzepte repräsentieren die Stärke eines PEIS: Die Zusammenarbeit mehrerer unabhängiger Systeme. Broxvall and Saffiotti (2005)

3.2.2 Implementation

Neben dem Konzept der PEIS-Ökologie stellen Broxvall und Kollegen in ihrer Arbeit die Implementation einer PEIS-Ökologie vor. Dabei wird zunächst nur die Abstraktion der Robotersysteme, sowie der Umwelt angestrebt. Dazu werden funktionelle Aufgaben und physikalische Systeme getrennt und zu einander zugewiesen. So werden zum Beispiel Kameras und Lasersensoren für die visuelle Raumwahrnehmung, Saugroboter für die Hindernisbeseitigung und die Reinigung eingeteilt. Neben der Funktionalität müssen die PEIS-Komponenten Schnittstellen anbieten, damit andere PEIS-Komponenten die Funktionen anfordern können. Der nächste Schritt ist die Entwicklung des Kommunikationsmodells. Broxvall und Kollegen empfehlen die Verwendung eines Hybridmodells zwischen einem Event-basierten und einem Tuple-Space Modell. Diese Empfehlung lässt sich auf die Abbildung ?? zurückführen. Arbeiten in den Bereichen Umgebungsintelligenz (siehe dazu Arregui et al. (2003) und Siegemund (2004)), Sensornetzwerke

(siehe Adjie-Winoto et al. (1999) und Heidemann et al. (2001)) sowie Robotik (Caceres et al. (2003)) setzen auf diese Kommunikationsmodelle, sowie auf die Hybridform. Für PEIS wird ein Eventbus-System ähnlich den ROS-Topics gefordert. Dieses ermöglicht das autonome An- und Abmelden von Komponenten. Das Tupel-Space Modell kann für die PEIS-Ökologie simple gehalten werden und wird von Broxvall und Kollegen mit der Form beschrieben:

```
<peisID, compID, key, val0, ..., valN>
```

peisID und compID identifiziert dabei die PEIS-Komponente, sowie die innere logische Einheit. key bezeichnet das Tupel selbst. Die dritte Komponente der Implementierung ist die Konfiguration der Ökologie. Diese beinhaltet alle Informationen über die Funktionalitäten des System, sowie die Verbindungen der einzelnen Komponenten untereinander. Dazu werden die Informationen ebenso in Tupeln gespeichert wie beim Kommunikationsmodell. Durch diesen Ansatz können Methoden aus der künstlichen Intelligenz genutzt werden um Aufgabenbereiche für einzelne PEIS-Komponenten zu arrangieren. Lundh et al. (2005) Der vierte und letzte Bestandteil der Implementierung von Broxvall und Kollegen ist der PEIS-Kern. Dieser Kern ermöglicht beinhaltet das Kommunikationsmodell, sowie die Konfiguration. Der Kern verbindet sich selbst mit weiteren Instanzen von PEIS-Kernen, die er im selben Netzwerk erkennt. Dies funktioniert unabhängig vom Typen des Netzwerkes. PEIS-Kerne tauschen ihre Konfigurations-Tupel untereinander aus. Dadurch kann eine PEIS-Komponente eine Funktionalität bei einer anderen Komponente finden und anfordern. Broxvall and Saffiotti (2005)

3.2.3 Zusammenfassung

Das Konzept der PEIS-Ökologie funktioniert nach dem Prinzip des teile und herrsche. Unterschiedliche Funktionalitäten werden nicht in einem allmächtigen Roboter System zusammengefasst, sondern in spezialisierte Roboter-, Aktoren- und Sensoren-Systeme verteilt. PEIS schafft eine Möglichkeit diese unterschiedlichen Systeme zusammen arbeiten zu lassen. Der Gedanke mit dem Tupel und dem Event-System ergibt sich logischerweise aus der benötigten Dynamik des Modells. Bedenklich ist dabei die lose Struktur innerhalb einer Tupels, da Daten nicht ordentlich gewartet werden können. Auch der Schritt der Konfiguration ist umständlich, da so nur starre Verbindungen zwischen einzelnen Komponenten hergestellt werden können. Durch eine Lockereitung der Konfiguration und einem CNP-System kann die Aufgabenverteilung optimiert werden. Ansonsten ist das Konzept praktikabel und zweckgemäß für das Zusammenspiel von Robotern und einer intelligenten Umgebung und soll im Rahmen dieser Arbeit verwendet werden.

3.3 Grasping und Handover - Arbeiten zu Roboterarmen

Dieses Kapitel befasst sich mit Arbeiten zu dem Thema Roboterarmen. Zunächst wird das Greifen eines Gegenstandes, *Grasping*, aufgezeigt. Dabei werden Arbeiten zum allgemeinen Greifen, zur Erkennung geeigneter Greifpositionen an Gegenstände und Kamera-Unterstütztem Grasping aufgeführt. Im Anschluss wird die Thematik der Übergabe, dem zentralen Thema dieser Arbeit, aufgegriffen und Literatur zu Roboter-Mensch Übergaben betrachtet.

3.3.1 Grasping - Greifen mit einem Roboterarm

Als Ausgangsarbeit für dieses Thema bietet sich die Arbeit Bicchi and Kumar (2000) an. In diesem werden die Grundlagen des Greifens erläutert. Dabei wird zunächst die menschliche Hand und ihre Verwendungszwecke betrachtet. Der Mensch nutzt seine Hand zur Erkundung, zum Festhalten und zur Manipulation von Objekten. Das Erkunden von Objekten ist der eigene große Forschungsbereich *Haptik* und kann bisher, auf Grund von fehlender Sensorik, in der Robotik nicht vollständig eingesetzt werden. Zwischen dem Festhalten, Fixieren und der Manipulation wird in der Arbeit unterschieden, sowie zwischen dem Manipulieren mit den Fingern und der Manipulation mit dem gesamten Arm. Dies liegt vor allem an den Anwendungs- und Forschungsbereichen. Während das Fixieren eines Objektes, beziehungsweise die Manipulation mit dem gesamten Arm, in der Industrie oft eingesetzt wird, ist die filigrane Arbeit mit den Fingern noch ein Thema der Forschung. Die ersten Arbeiten zu diesem Thema der Robotik gehen auf Asada (1979) und Mason and Salisbury Jr (1985) zurück. Neben dem Greifen mit den Fingern der Hand gibt es auch Griffe mit dem Kompletten Arm, bezeichnet mit *whole arm grasps* (Townsend (1988), Bicchi (1994) und *power grasps* (Mirza and Orin (1990)) (Bicchi and Kumar, 2000).

Das Greifen unterliegt physischen Grenzen, dabei wird der Griff, beziehungsweise das Halten, unter anderem durch den Kraftvektor und dem Haftkoeffizienten beeinflusst. Ein Griff wird durch N Kontakte beschrieben. Dabei wird angenommen, dass alle Kontakte punktuell sind. Auch ein Kontakt auf einer Linie oder Oberfläche wird durch zwei oder mehr Punktkontakte abgebildet. Die Literatur unterscheidet diese Kontakte in reibungslose, reibungsbedingte oder weiche Punktkontakte (Salisbury and Roth, 1983). Ein reibungsloser Kontakt kann nur eine Kraft entlang der gemeinsamen Normalen erzeugen. Ein reibungsbedingter kann neben der normalen auch eine tangentiale Kraft und ein weicher Kontakt zusätzlich einen Drehmoment erzeugen. Die Kontaktart ist abhängig von den Oberflächeneigenschaften des Grippers und des Objektes. Bei einer gummiähnlichen Oberfläche des Grippers wird das Kontakt als weich modelliert. Haben Gripper und Objekt beide harte und rauhe Oberflächen wird ein reibungsbedingter Kontakt angenommen. Sind die Kontaktstellen auf Gripper und Objekt glatt und ist ein kleiner Reibungskoeffizient gegeben, gilt der Kontakt als reibungslos (Bicchi and Kumar, 2000).

Neben dem Greifen selbst existiert auch die Problematik des Greifpunktes. Also der Stelle am Objekt an welcher der Gripper ansetzt. Dazu existieren verschiedene Arbeiten. Die ersten zu diesem Thema waren die Arbeiten Kamon et al. (1996), Coelho et al. (2001) und Bowers and Lumia (2003). In diesen wird mit Hilfe von Sensoren planare 2D-Objekte gegriffen. Ergebnisse mit 3D-Objekten erreichte die Arbeit Saxena et al. (2008). Dabei werden zunächst mit maschinellem Lernen und gelabelten Testdatensätzen gute Griffpunkte für 3D-Objekte angelernt. Anschließend kann der Algorithmus unbekannte 3D-Objekte bewerten und die besten Griffpunkte finden. Diese Grundlagenforschung wird in der Arbeit Maitin-Shepard et al. (2010) genutzt um an unbekannten 3D-Objekten aus Stoff Ecken zum Greifen und Zusammenlegen zu finden. In dieser werden vor allem Ansätze nach dem RANSAC-Verfahren gewählt, um bestimmte Strukturen gezielt zu finden. Für weitere Informationen lassen sich in der Literatur noch viele verschiedene Anwendungsfälle und Ansätze zum Greifen finden.

3.3.2 Handover - Übergabe zwischen zwei Systemen

Die Übergabe zwischen zwei Systemen ist eine häufige Interaktion im Alltag. Dieses betrifft oft die Manipulation eines Objektes. Die Arbeit Huber et al. (2008) beschäftigt sich mit der Übergabe zwischen Mensch und Roboter und beinhaltet zunächst eine Analyse einer Übergabe zwischen zwei Menschen. Dabei stellt sich folgende Aktionsreihenfolge für eine Übergabe heraus (Huber et al., 2008):

1. Der Geber nimmt das Zielobjekt
2. Der Geber bewegt die Hand Richtung Übergabeposition
3. Der Nehmer bewegt die Hand Richtung Übergabeposition
4. Transaktion
5. Beide nehmen ihre Hände zurück

Eigene Beobachtungen vom Mensch-Mensch Übergaben ergaben, dass diese Form der Übergabe nur einem Interaktionsart ist und als *Geben* bezeichnet werden kann, da die Interaktion vom Geber ausgeht. Eine Alternative dazu stellt das *Nehmen* dar, bei dem die Interaktion vom Nehmer ausgeht. Eine optimierte dritte Interaktionsart wäre eine synchrone Bewegung beider Akteure, bei der eine Verzögerung durch die Reaktionszeit des Interaktionspartners entfällt.

In der Robotik existieren mehrere Arten der Übergabe. Diese unterscheiden sich anhand des Interaktionspartners und des Anwendungsfeldes. In der Industrie, zum Beispiel dem Autobau, werden die Werkstücke nicht direkt zwischen den einzelnen Robotern übergeben, sondern befinden sich auf einem Förderband. Dieses fährt das Werksstück auf eine definierte Position und die einzelnen Roboter führen ihren Arbeitsschritt aus. Anschließend wird das Werksstück zur nächsten Station gebracht. Eine weitere Art der Übergabe ist die Mensch-Roboter-Interaktion. Dieses Thema ist ein weitverbreitetes Thema mit vielen Aspekten. So beschäftigen sich die Arbeiten Huber et al. (2008) und Shibata et al. (1995) mit dem Timing-Verhalten, Mainprice et al. (2010) und Kulić and Croft (2005) befassen sich mit der sicheren Planung von Übergabe-Interaktionen, weitere Arbeiten (unter anderem Prada et al. (2014) und Basili, Huber, Kourakos, Hirche, Brandt, and Glasauer (Basili et al.)) beschäftigen sich mit den Reaktionen auf menschliche Aktionen, wie Gesten (zum Beispiel: Handfläche nach oben geöffnet als Geste fürs Nehmen) und Veränderungen während der Übergabe.

In dieser Arbeit steht jedoch die Roboter-Roboter Übergabe im Fokus. Dieses Thema ist in der Literatur nicht vorhanden oder wird nur als Randaspekt erwähnt.

4 Laboraufbau

Dieses Kapitel befasst sich mit dem Aufbau der Versuchsanlage und der verwendeten Hardware. Dabei werden die Aspekte des räumlichen Aufbau genauso betrachtet wie die Netzwerkinfrastruktur und die genutzten Roboter und Sensoren.

4.1 Aufbau Teststand

Der Aufbau vom Teststand ist in den beiden Abbildungen 7 in der Draufsicht und 8 in der Seitenansicht dargestellt. Rote Flächen stellen die Wände dar. Tische sind in blau gezeichnet. Die grüne Fläche ist die Arbeitsplatte auf welcher der Roboter *Dummy* (orange) fixiert ist. Türkise Objekte stellen Geräte wie Kameras und Monitore dar.

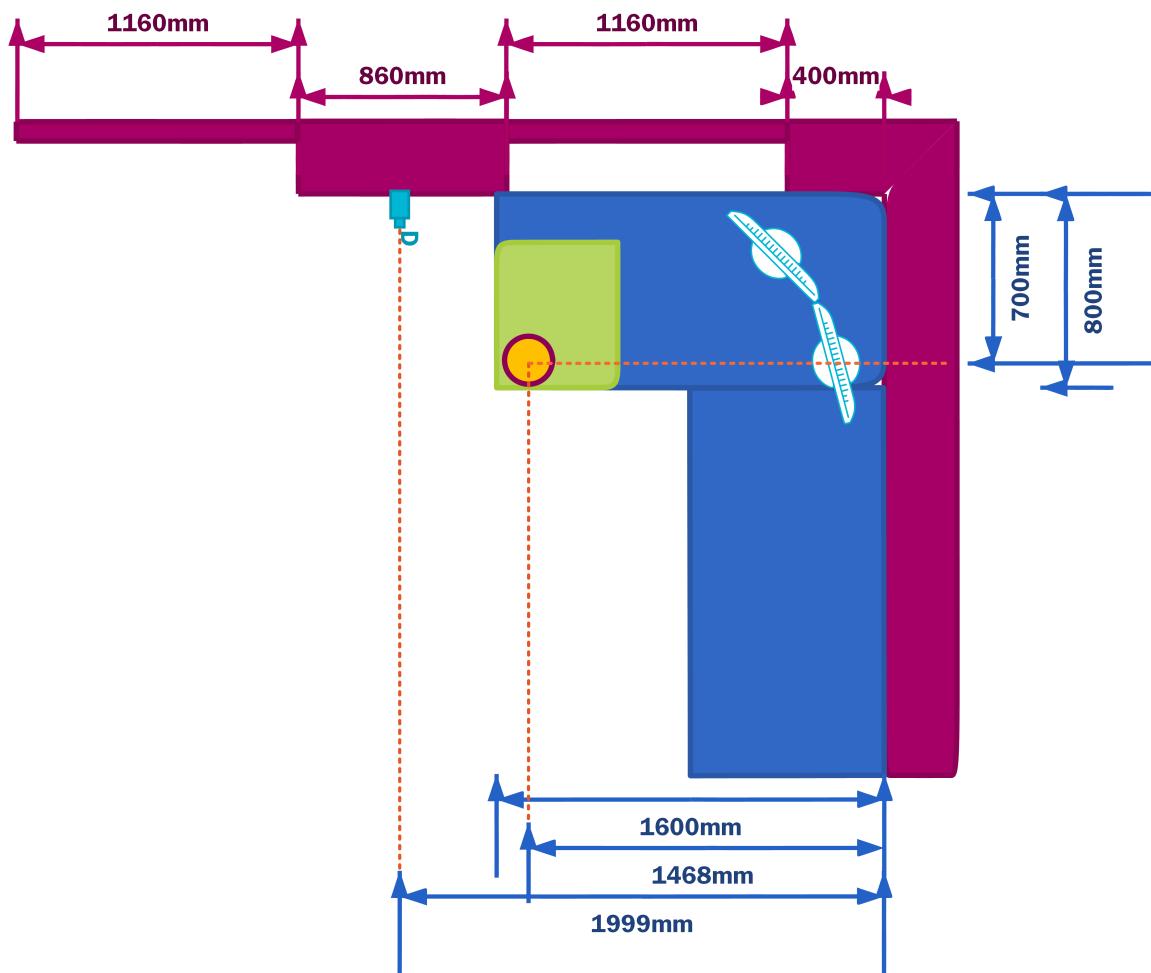


Abbildung 7: Der Aufbau aus der Draufsicht vom Teststand. Rot: Wände, Blau: Tische, Grün: Arbeitsplatte Roboter, Orange: Roboter, Türkis: Geräte

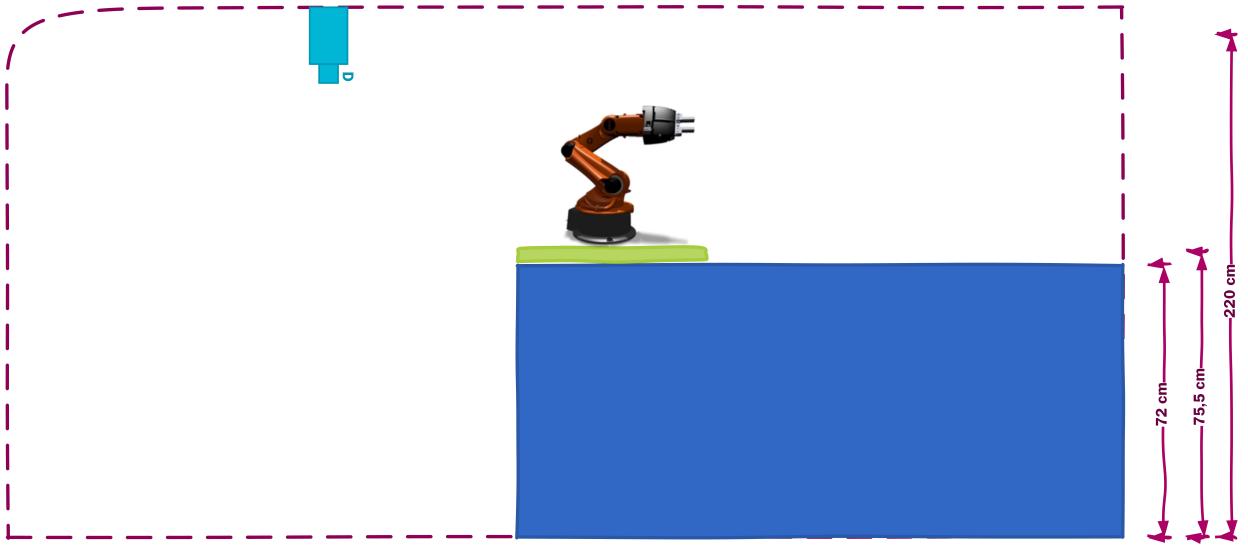


Abbildung 8: Der Aufbau aus der Draufsicht vom Teststand. Rot gestrichelt: Wände, Blau: Tische, Grün: Arbeitsplatte Roboter, Orange: Roboter, Türkis: Kamera

4.2 YouBot

In dieser Arbeit werden zwei Roboter genutzt. Bei beiden handelt es sich um YouBots der Firma Kuka. Diese wurden 2010 erstmals auf der Automatica in München vorgestellt. Kuka ist ein deutscher Hersteller von Industrierobotern mit Sitz in Augsburg. Kuka hat aber auch Erfahrung im Bau von experimentellen Robotern wie etwa die Arme für Justin, einem Service-Roboter.

Die beiden YouBots unterscheiden sich beim Aufbau. YouBot 1 besteht aus einem Manipulator mit Gripper, YouBot 2 besteht aus Manipulator mit Gripper und mobiler Plattform mit omnidirektionalen Rollen. Zur besseren Unterscheidung und zur Lesbarkeit dieser Arbeit haben beide Roboter einen Namen bekommen. YouBot 1 entspricht dabei **Dummy**, YouBot 2 ist **Rose**. Im Weiteren Verlauf der Arbeit werden nur noch diese Namen genutzt um eine Verwechslung zu vermeiden.

4.2.1 Manipulator

Die Roboterarme von Dummy und Rose sind eine offene kinematische Kette mit fünf Joints, die von der Basis aufsteigend nummeriert sind. An Joint 5 sind die Gripper montiert, diese bestehen aus zwei individuellen Fingern, die linear auf einer Achse bewegt werden können.

Joint 1 und Joint 5 sind zwei Drehgelenke um die Z-Achse, die bei ausgestreckter Haltung (Candle-Pose) parallel liegen. Joint 2, 3 und 4 sind Kippgelenke um die X-Achse, die immer parallel liegen. Durch diese Konstellation ergeben sich für den kompletten Arm fünf Freiheitsgrade (5-DOF). Durch die Einschränkung der Achsen auf die X- und Z-Achsen der einzelnen Joints ist ein seitlicher Griff bei $x_{base} = 0$ oder $y_{base} = 0$ nicht möglich. Die Höhe in Candle-Pose beträgt inklusiv Gripper 655mm. Die einzelnen Abstände lassen sich der Grafik 9 entnehmen.

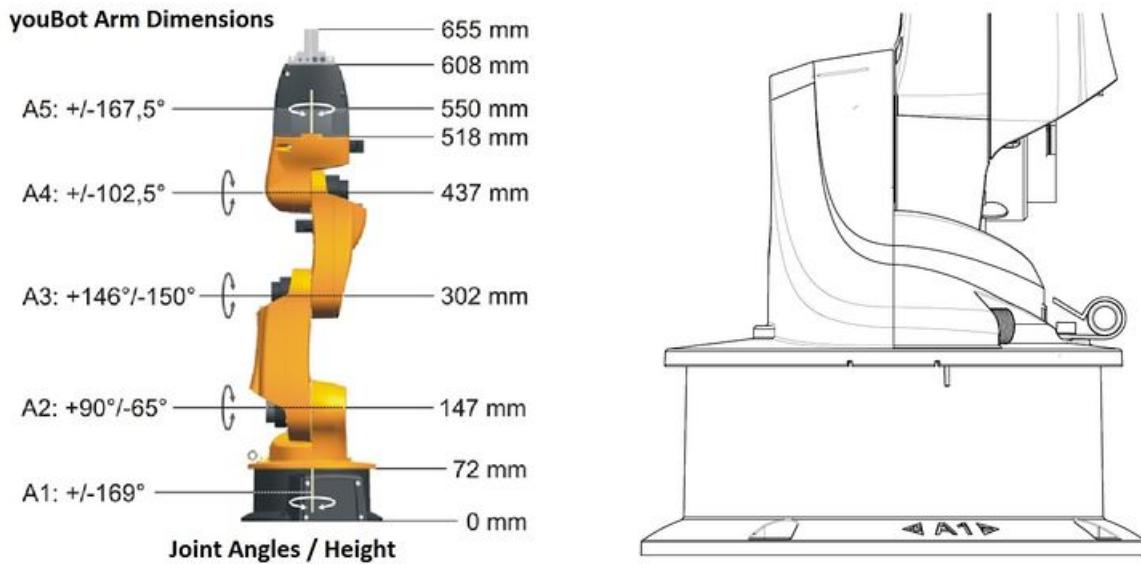


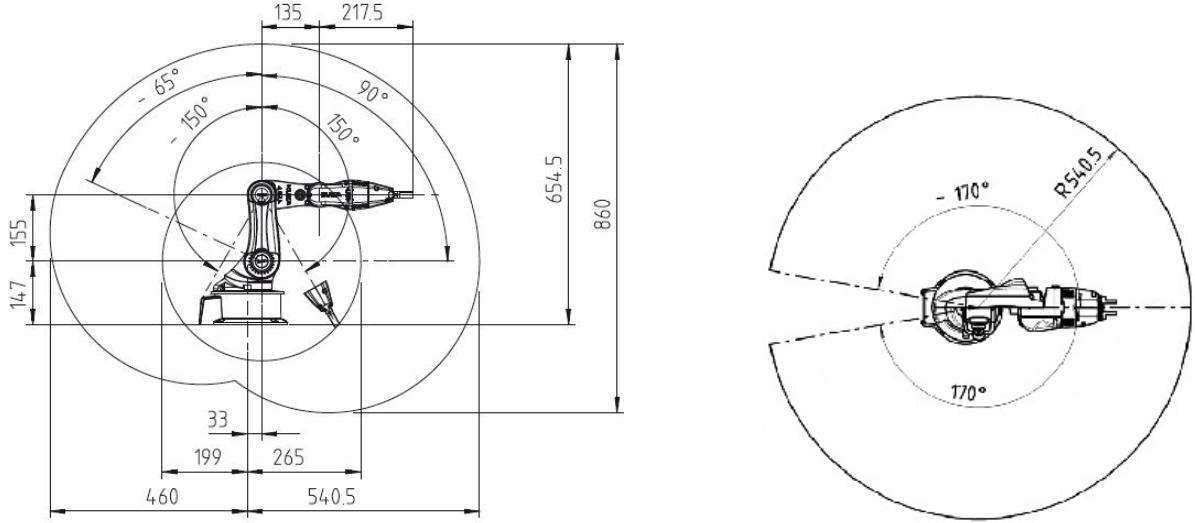
Abbildung 9: YouBot Arm Kinematik. Links: Detaillierter Mechanismus mit Joint und Link Angaben. Rechts: Vergrößerte Darstellung der Basis. Bildquelle: Florek-Jasinska (2015)

Tabelle 2 stellt noch einmal den Drehbereich aller Joints dar. Dabei sind neben Bezeichner auch die minimalen und maximalen Winkel, sowie die Winkelgeschwindigkeiten. Eine Besonderheit stellt Joint 3 dar, durch den Aufbau bedingt wurde die Achse innerhalb der Treiber gespiegelt, sodass die Winkel um den 0° Punkt gespiegelt wurden. In der Tabelle und den folgenden Zeichnungen ist diese Spiegelung nicht beachtet.

Bezeichnung	Max./Min. Winkel	Winkelgeschwindigkeit (rad / s)
Joint 1	± 2.94	$\frac{\pi}{2}$
Joint 2	$+ \frac{\pi}{2} / -1.13$	$\frac{\pi}{2}$
Joint 3	$+2.54 / -2.63$	$\frac{\pi}{2}$
Joint 4	± 1.78	$\frac{\pi}{2}$
Joint 5	± 2.91	$\frac{\pi}{2}$

Tabelle 2: YouBot Arm Joints. Quelle: Florek-Jasinska (2015)

Der Arbeitsraum des YouBot Arms beschränkt sich durch die Joints. Die Grafiken 10(a) und 10(b) stellen den Arbeitsbereich eingeschränkt dar. Die Darstellung 10(a) zeigt mit Hilfe von Konturen die einzelnen Bahnen unter Beschränkung der Joints, sowie der End-Effektor Ausrichtung. Die äußerste Kontur gibt die maximale Reichweite für den End-Effektor an. Die Z-Achse steht dabei orthogonal zur Tangente an der Konturposition. Abbildung 10(b) gibt eine Draufsicht auf den Arbeitsraum. Durch die Beschränkung von Joint 1 befindet sich ein toter Winkel im "Rücken" der Arms. Dieser kann durch einen Überschlag des ganzen Arm, insbesondere Joint 2 und 3, und einer 180 °Drehung von Joint 1 dennoch erreicht werden. Dies wird durch den linken Bereich in Abbildung 10(a) klar.



(a) YouBot Arm Arbeitsraum. Reichweite der einzelnen Gelenk und einzelne Abmessung der Links und Link-Offsets. Einheitslose Angaben sind in mm.

(b) Draufsicht YouBot Arm Arbeitsraum.

Abbildung 10: Arbeitsraum YouBot. Bilderquelle:Florek-Jasinska (2015)

4.2.2 Mobile Plattform

Rose besitzt neben dem Arm noch eine mobile Plattform. Diese ist 590 mm lang, 380 mm breit und 140 mm hoch. Die Plattform hat vier Räder mit einem Durchmesser von 47.5 mm. Jedes Rad lässt sich getrennt ansteuern. Bei den Rädern handelt es sich um Mecanum-Räder, die eine Steuerung in alle Richtungen ermöglicht. Dazu sitzen auf jeder Felge sechs drehbare tonnenförmige Rollen die im Winkel von 45° zur Achse des Rades angebracht sind. Damit Omni-Direktionale Bewegungen möglich sind werden die Räder abwechselnd zum Nachbar auf +45° oder -45° angeordnet. Die minimale Geschwindigkeit der Plattform beträgt $0.01 \frac{m}{s}$, die maximale $0.8 \frac{m}{s}$. Abbildung 11 zeigt den Aufbau und Abmessungen der Plattform.

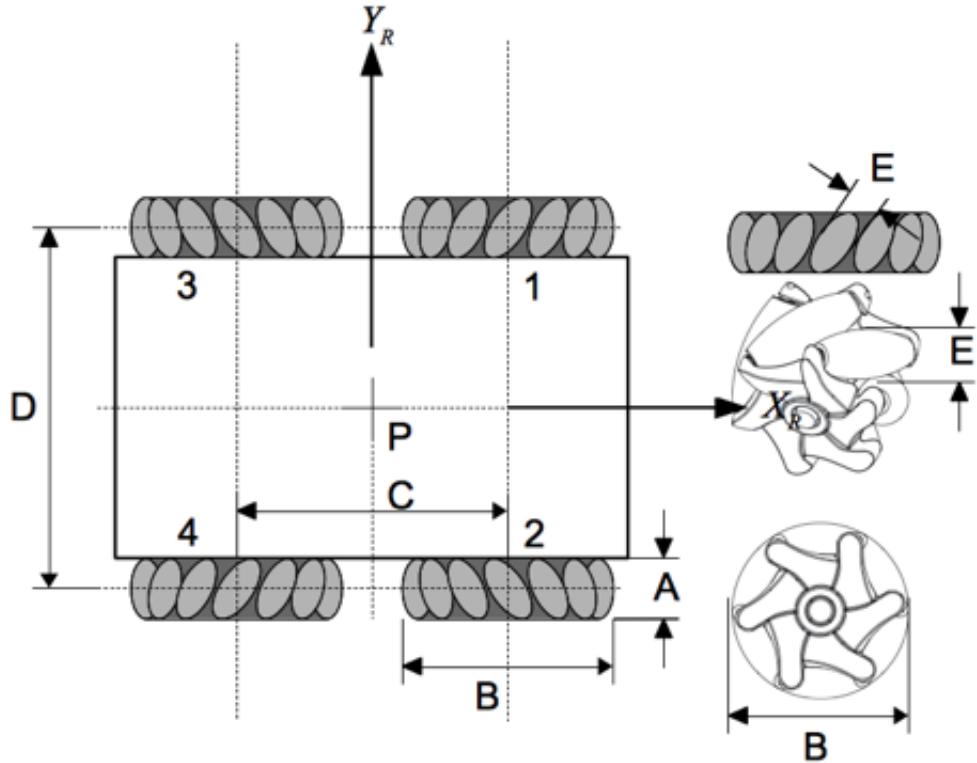
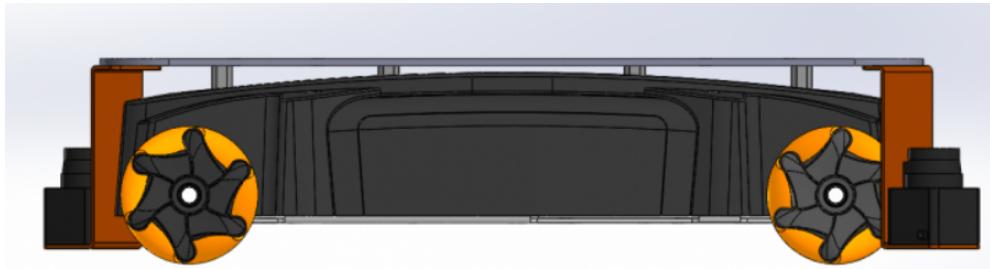
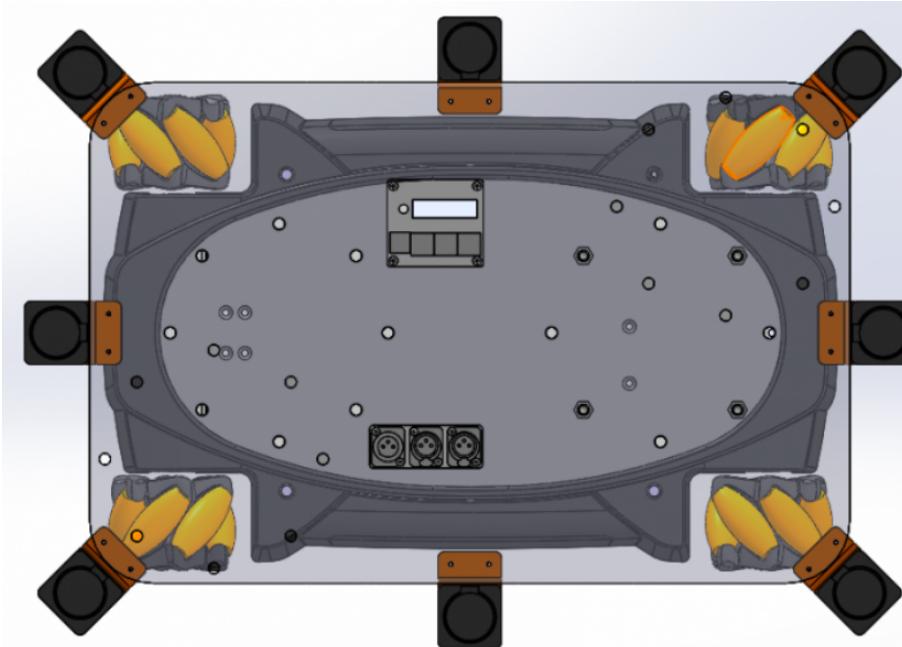


Abbildung 11: YouBot Base. $A = 74.87 \text{ mm}$, $B = 100 \text{ mm}$, $C = 471 \text{ mm}$, $D = 300.46 \text{ mm}$ $E = 28 \text{ mm}$. Die Bodenfreiheit der Plattform beträgt 20 mm. Bildquelle: Florek-Jasinska (2015)

Die Plattform von Rose ist modifiziert und besitzt eine Aufsatzplatte. Diese dient zur Montage zusätzlicher Sensoren und zur Schutz der Räder bei Kollisionen. Diese ist 600 mm lang und 396 mm breit. Durch die Abstandsbolzen und die Dicke der Platte (5 mm) erhöht sich die Plattform auf 150 mm (siehe Abbildung 12(a)).



(a) Seitenansicht mobile Plattform mit Sensorplatte



(b) Draufsicht mobile Plattform mit Sensorplatte

Abbildung 12: Mobile YouBot Plattform mit Sensorplatte. Bilderquelle:Kuka (2015)

4.2.3 Rechner

Für die Rechenleistung der beiden Roboter sorgen neben den verbauten, eingebetteten Boards zwei Computer mit Linux Betriebssystemen. Der für Rose zuständige PC ist ein Mini-ITX und in der mobilen Plattform eingebaut. Dieser läuft mit einem Intel Atom D510 Dual Core mit 1.66 GHz als CPU und 2 GB DDR2 RAM. Als Festplattensystem ist eine 32 GB SSD verbaut. Der Rechner stellt einige IO-Schnittstellen zur Verfügung: sechs USB 2.0, ein VGA und zwei LAN Anschlüsse sind nutzbar. Dabei ist ein LAN Anschluss für den Ethercat-Anschluss der Plattform belegt. Diese wiederum bringt nochmal zwei LAN Anschlüsse mit. Der Rechner für Dummy ist ein TODO

Weitere Details zu den Armen, der Plattform oder den Rechnern befindet sich im Anhang A.

4.3 Sensoren

Für die Erkennung von Objekten und der Positionierung im Raum sind mehrere Sensoren nötig. In dieser Arbeit wird dabei auf das Kamerasystem Asus XTion Pro Live und auf einen Laserscanner Hokuyo URG-04LX-UG01 zurückgegriffen.

4.3.1 Asus XTion Pro

Die XTion Pro ist ein Tiefenkamerasystem von Asus. Es besteht aus einer RGB-Kamera, einer Tiefen-Kamera und zwei Mikrofonen. Die Kamera wurde von Prime Sense entwickelt und ist eine um gelabelte Microsoft Kinect. Die XTion ist kleiner und leichter als die Kinect und damit besser für die Robotik geeignet. Angeschlossen wird die XTion mit einem USB Kabel und betrieben mit dem ROS Paket OpenNI2. Der Node published eine PointCloud mit RGB Daten, sowie eine PointCloud für die Tiefenkamera, als auch Bilder der RGB Kamera.

Das Sichtfeld der Kamera beträgt Horizontal 58°, 45°Vertikal und 70°Diagonal. Die Auflösung der Tiefenkamera beträgt bei 30 Frames pro Sekunde 640x480 und bei 60 FPS 320x240. Die Auflösung des RGB-Bildes 1280x1024. Die nutzbare Distanz der Tiefenkamera beträgt zwischen 800 mm und 3500 mm. Asus (2015) Diese Distanz schränkt die Nutzbarkeit der Kamera ein, so ist sie nicht für eine visuelle Unterstützung der Gripper geeignet, wenn sie an diesen montiert ist, da die Distanz zwischen Kamera und Gripper kleiner 800 mm ist.



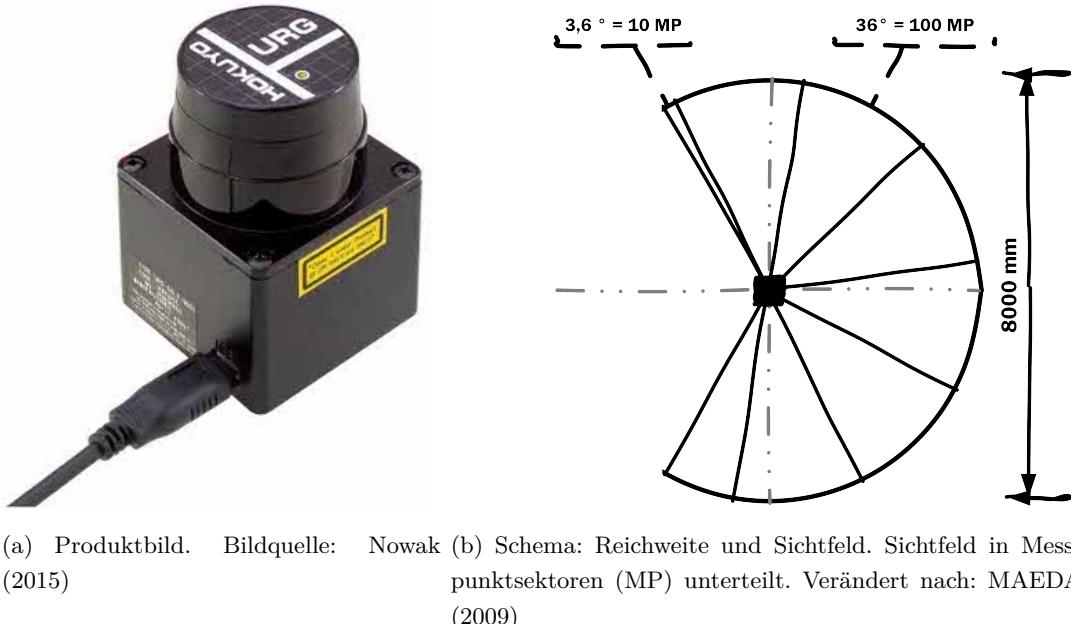
Abbildung 13: Asus XTion Pro Live. Bildquelle: Asus (2015)

4.3.2 Hokuyo URG-04LX-UG01

Der URG-04LX-UG01 ist ein optischer Abstandsmesser. Dieser kann Entfernung in einer zirkulären Ebene messen. Entwickelt wurde er von der Firma Hokuyo, welche neben dem URG noch weitere, ähnliche Sensoren anbietet, die andere Reichweiten und Sichtfelder abdecken. Die Sensoren nutzen zur Messung der Entfernung eine Infrarot-Diode mit einer Wellenlänge von 785 nm. Die Distanz wird mit Hilfe der Phasenverschiebung berechnet. Diese Technik reduziert den Einfluss durch das Oberflächenmaterial des gemessenen Objektes. Versorgt wird der Sensor mit 5V DC, welche über den USB-Anschluss angelegt werden. Da die Startleistung des Sensors die

maximale Abgabeleistung eines einzigen USB-Anschluss überschreitet, müssen ein Y-USB-Kabel und zwei USB-Anschlüsse genutzt werden. Angesteuert wird der Sensor mit dem ROS-Paket Hokuyo-Node, welche eine PointCloud published.

Der URG-Sensor hat eine Reichweite von maximal 4000 mm und benötigt für eine Messung 100 ms, was einer Messrate von 10 FPS entspricht. Das Sichtfeld beträgt 240°. Die minimale Graduelle Auflösung beträgt 0.36°. Dadurch ergeben sich die ungefähren Disparitäten von 20 mm bei 2000 mm und maximal 40 mm bei 4000 mm Radius. Die Ungenauigkeit beträgt maximal 3 Prozent des gemessenen Wertes, was bei der maximalen Reichweite von 4000 mm einer Abweichung von 120 mm entspricht. MAEDA (2009)



(a) Produktbild. Bildquelle: Nowak (2015) (b) Schema: Reichweite und Sichtfeld. Sichtfeld in Messpunktsektoren (MP) unterteilt. Verändert nach: MAEDA (2009)

Abbildung 14: Hokuyo URG-04LX-UG01

4.3.3 Argos 3D - P100

Ein weiterer Tiefensensor ist die Argos 3D – P100 von Bluetechnix. Diese arbeitet mit dem Time of Flight (ToF) Prinzip. Der Sensor wird mit einem USB-Kabel angesteuert, benötigt aber noch eine weitere Stromzufuhr. Der Sensor kann aus ROS mit einem eigenen Node angesprochen werden. Der Sensor hat eine Reichweite zwischen 100 mm und 3000mm. Die Framerate beträgt bis zu 160 FPS bei einer Auflösung von 160 x120. Das Sichtfeld beträgt 90 °.(Bluetechnix, 2015)



Abbildung 15: Argos 3D - P100. Bildquelle: Bluetechnix (2015)

4.3.4 Netzwerk- und Sensoranbindung

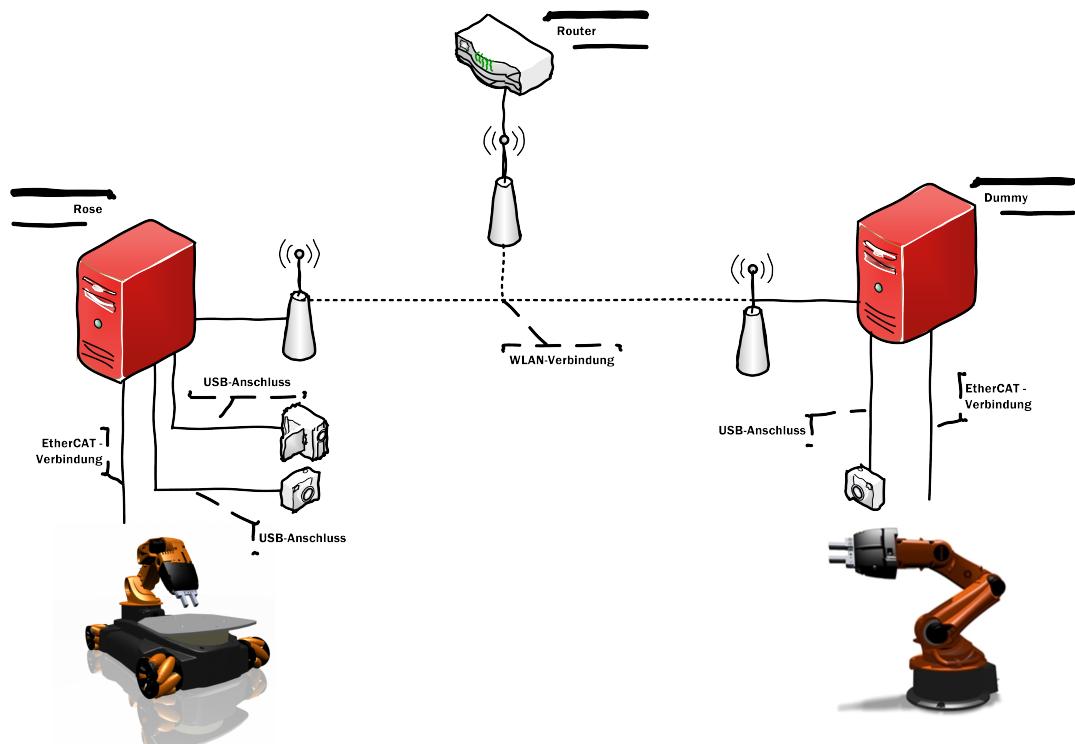


Abbildung 16: Schematische Darstellung der Verbindungen

Abbildung 16 stellt den genutzten Verbindungsplan dar. Die zentrale Komponente ist der Router, welcher mit einem externen Access-Point (AP) eine WLAN-Verbindung ermöglicht. Mit diesem WLAN verbinden sich die beiden Steuerungsrechner *Rose* und *Dummy*, welche beide unter diesen Hostnames im Netz erreichbar sind. Die Verbindungen auf beiden Seiten sind gleich gehalten. Die Verbindung zwischen Rechner und Roboter sind mit der EtherCAT-Schnittstelle (Ethernet for Control Automation Technology) realisiert. Diese Ethernet-Variante wurde für Echtzeit-Anforderungen entwickelt, indem Wert auf kurze Zykluszeiten ($\leq 100 \mu\text{s}$) und niedrigem Jitter für exakte Synchronisierung ($\leq 1 \mu\text{s}$) gelegt wurde. (und Holger Büttner (2003)) Die Verbindung zwischen den Rechnern und den Sensoren ist mit USB-Verbindungen umgesetzt. Dabei werden

alle USB-Anschlüsse auf der mobilen Plattform von Rose vollständig durch die Sensoren, sowie dem WLAN-USB-Dongle und dem Tastatur/Maus-Dongle belegt. Weitere USB-Geräte sind nur durch einen zusätzlichen HUB möglich.

Eine Alternative zu diesem Plan ist ein vermaschtes Netz, wie bei PEIS gefordet. Dadurch könnte der Router wegfallen, da alle Endgeräte direkt miteinander verbunden sind. Des Weiteren könnten einige Sensoren auf eigene Rechner ausgegliedert werden. Dies würde die einzelnen Rechner, besonders den leistungsschwachen Rechner von Rose, entlasten, aber auch die Netzwerkkomplexität und die Kosten steigern.

5 Entwicklung

Dieses Kapitel befasst sich mit einzelnen Aspekten der Entwicklung. Nach der Auflistung der Nicht-funktionalen Anforderungen, sowie der Funktionen, werden grundlegende Konzepte und Architekturen vorgestellt. Dabei werden erste Sensoren, Aktoren und bestehende Software Bibliotheken getestet und für eine Anschlussverwendung bewertet. Die weiteren benötigten Softwarepaket werden im nachfolgenden Kapitel Implementierung TODO umgesetzt.

5.1 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben Qualitätsmerkmale des MRS. Diese sind in verschiedene Kategorien gruppiert und haben Auswirkungen auf die Funktionalität und die Architektur. Im Folgenden werden nur die für diese Arbeit relevanten Kategorien erläutert und in Bezug gebracht.

5.1.1 Sicherheit - Safety

Im englischen wird der Begriff Sicherheit in die Terme *Security* und *Safety* unterteilt. Ersteres beschäftigt sich unter anderem mit der Verschlüsselung von Daten. Der Begriff *Safety* bezieht sich auf die Anwendersicherheit, zum Beispiel ob dieser durch das System Verletzungen erleiden kann oder ob sich das System selbst oder die Umwelt beschädigt. Da die Roboter in dieser Arbeit nicht, wie Industrieroboter, durch Gitterzäune gesichert sind wird eine Lichtwarnanlage installiert. Diese arbeitet ähnlich einer Ampel. Befindet sich das System in einer sicheren Stellung, in der es sich nicht bewegt, leuchtet die Lichtanlage in grün. Wird eine Bewegung geplant oder ist das System vor einer Bewegung blinkt die Lichtanlage fünf Sekunden gelb. Innerhalb dieses Intervalls bewegt sich das System nicht. Auf das Blinken folgt ein rotes Leuchten der Lichtanlage. In dieser Zeit führen die Roboter ihre Aktionen aus. Während dieser Aktionen nutzt das Robotersystem die Sensoren und berechnet kollisionsfreie Pfade um nicht mit sich selbst oder der Umwelt zu kollidieren. Objekte, die sich erst zur Laufzeit auf dem Pfad befinden, werden in dieser Arbeit noch nicht beachtet und können so zur Kollision führen. Um bei Bewegungen der mobilen Plattform oder zwischen zwei Tasks der Arme die Sicherheit dieser zu sicheren existieren die beiden Posen *Candle* und *Fold*. Die erste Pose streckt den Arm senkrecht in die Höhe und ermöglicht so kollisionsfreie Pfade in die meisten anderen Posen. In der zweiten Pose befinden sich alle Gelenke in ihrer minimalen Konfiguration. Sollte dem Roboter der Strom fehlen, bedingt durch einen Stromausfall oder leere Batterien, fällt der Roboterarm in sich zusammen, da die Gelenkmotoren die Position nicht mehr halten können. In der *Fold*-Pose kann der Roboter nicht weiter in sich zusammenfallen, da alle Glieder aufeinander liegen. Deshalb sollten die Roboter diese Pose in längeren Inaktivitätsphasen einnehmen. Diese Aspekte werden während der Entwicklung der Funktionen berücksichtigt.

5.1.2 Zuverlässigkeit

Die Zuverlässigkeit des Systems beschreibt unter anderem das Systemverhalten bei einem Fehlerfall oder Teilsystemausfall. Das Robotersystem darf dabei nicht in einen unsicheren Zustand geraten. Da das Robotersystem auf ROS aufbaut werden einzelne Aktoren und Sensoren als ROS-Node betrieben. Jeder Node stellt dabei ein einzelnes Subsystem dar, welcher ausfallen kann. ROS ist so konzipiert, dass das ganze System nicht abstürzt, wenn ein einzelner Node ausfällt. Eine Schwachstelle bildet hierbei der ROS-Core. Dieser stellt den Mittelpunkt eines ROS-Systems dar. Fällt dieser aus bricht die Kommunikation zwischen den einzelnen Nodes zusammen. In diesem Fall müssen die Nodes in einen sicheren Fehlerzustand übergehen. Dies betrifft vor allem die Aktoren, besonders die Steuerungen der Roboter.

Ein weiterer Aspekt der Zuverlässigkeit ist die Wiederherstellbarkeit. Dies betrifft ob und wie komplex ein bestimmter fehlerfreier Systemstatus wiederhergestellt werden kann. Da die Nodes einzelne Subsysteme sind, können diese nach einem Ausfall einfach wieder gestartet werden und der Systemstatus ist wiederhergestellt. Dies betrifft auch den ROS-Core. Wird dieser auf der selben Adresse (IP und Port) neu gestartet, nehmen alle Nodes die Kommunikation wieder auf. Diese beiden Aspekte werden vor allem bei der Entwicklung der Architektur für das MRS berücksichtigt. Dabei ist das zentrale Koordinierungs- und Konfigurations-System eine Komponente, die zu einem potenziellen Single-Point-of-Failure führen kann.

5.1.3 Korrektheit

Die Korrektheit lässt sich in dieser Entwicklung durch die Genauigkeit der Funktionen definieren. So dürfen die Positionen und Orientierungen der Arme und der mobilen Plattform nur um bestimmte Grenzen abweichen. Dabei gelten für die End-Effektor-Pose eine Distanzabweichung von maximal 5mm für jede Dimension und für die Orientierung eine Abweichung von maximal 1° pro Achse. Dies muss bei der Entwicklung der inversen Kinematik berücksichtigt werden. Für die mobile Plattform gelten andere maximale Abweichungen. Für die Distanzabweichungen sind das 5cm in X- und Y-Richtung. Für die Z-Dimension darf sie maximal 5mm sein. Für die Rotation gilt: Z-Achse 5°, X- und Y- Achse 1°.

5.1.4 Leistung

Die Leistung bezieht sich auf die Effizienz des MRS. Besonders der Zeit- und Energieaufwand sollen möglichst minimiert werden. Der Zeitaufwand bezieht sich dabei nicht auf die Performance-Optimierung der Algorithmen, sondern der Bewegungen und deren Abläufe. Dieser Aspekt muss bei der Entwicklung der Funktionen und der Implementierung berücksichtigt werden. Der Energieaufwand ist stark abhängig von den benutzten Aktoren und wird bei der Entwicklung der inversen Kinematik für die Arme und die Pfadbestimmung für die mobile Plattform in Betracht gezogen.

5.1.5 Änderbarkeit

Die Änderbarkeit besteht aus Sicht der Entwicklung aus mehreren Aspekten. Für diese Arbeit ist dabei die Erweiterbarkeit ein wichtiger Aspekt. Zum einen geht es um die Erweiterbarkeit des MRS um weitere Sensoren und Aktoren. Diese sollen möglichst einfach in das bestehende MRS eingebunden werden können ohne bestehende Subsysteme zu verändern. Zum anderen soll in der Weiterentwicklung dieses MRS weitere Konzepte eingebunden werden. So müssen für eine automatische Konfiguration und verteilte Koordinierung Schnittstellen in der Architektur entworfen werden. Diese Schnittstellen sollen unter anderem eine Markt-basierte Konfiguration ermöglichen. Diese beiden Aspekte der Erweiterbarkeit müssen in der Entwicklung der Architektur und der Funktionen berücksichtigt werden. So sind grobe Kostenangaben für die Funktionen ein möglicher Ansatz für die Gebote am Markt.

5.2 Funktionalitäten

In diesem Kapitel werden die Funktionen des Robotersystem aufgelistet. Das folgende Use-Case Diagramm 17 stellt die groben Funktionsanforderungen an das Robotersystem dar.

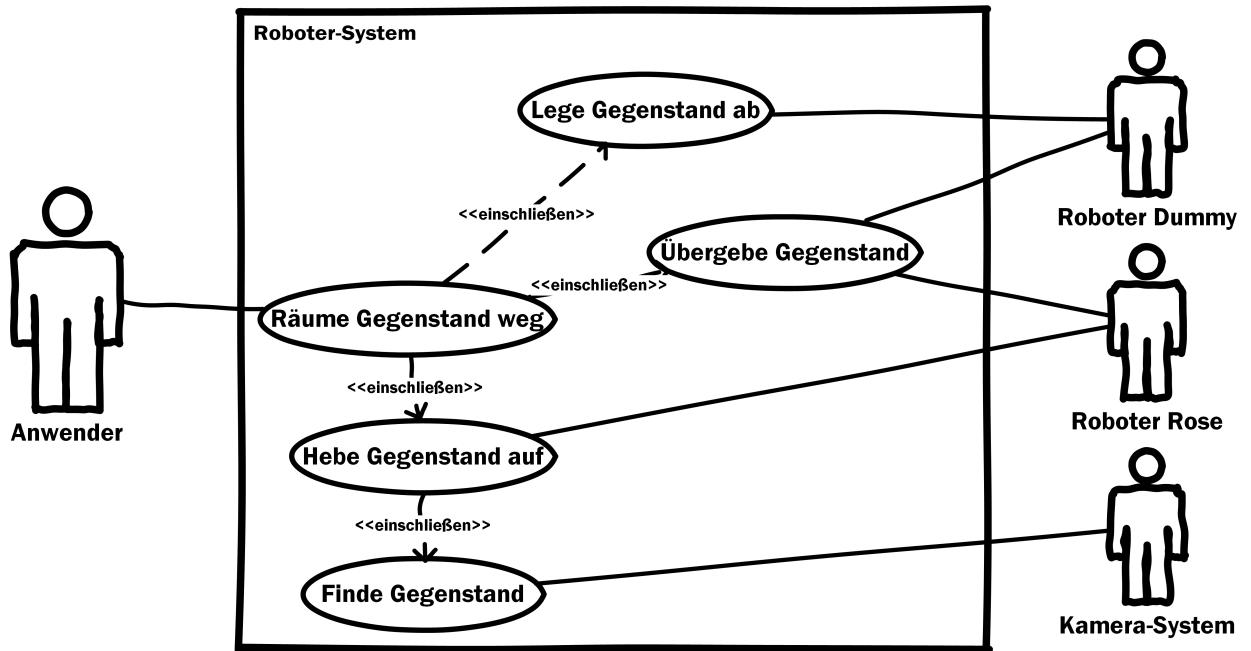


Abbildung 17: Vereinfachtes Use-Case Diagramm zur groben Funktionsübersicht des Roboter-systems

Der Anwender gibt dem System die Anweisung einen Gegenstand wegzuräumen. Das Robotersystem gliedert die Anweisung in einzelne Tasks auf. Diese Tasks werden von den unterschiedlichen Akteuren mit ihren Funktionalitäten sequentiell oder parallel ausgeführt. In diesem konkreten Fall soll zunächst der Roboter Rose den entsprechenden Gegenstand aufheben. Dies erfordert zunächst eine Lokalisierung des Objektes die von einem Kamerasytem angefordert

wird. Anschließend übergeben sich die Roboter Rose und Dummy den Gegenstand, bevor Dummy ihn anschließend an einer gewünschten Position ablegt. Diese sehr grobe Darstellung des Vorgangs beinhaltet die vier zentralen Funktionen, die das System umsetzen muss: Gegenstand aufheben (*pick-up*), Gegenstand ablegen (*place*), Gegenstand lokalisieren und Gegenstand übergeben(*handover*).

Die folgenden Anforderungen dienen zur Strukturierung der Arbeit und zerlegen das Robotersystem in Teilsysteme. Dies wird benötigt um einen Aufwand für die Koordinierung und Konfiguration zu bestimmen. Deshalb sind die Anforderungen grobe Darstellungen und aus dem Aspekt der Software-Entwicklung nicht detailliert genug. Dafür müssten die Funktionen weiter runter gebrochen werden. Dies würde jedoch den Umfang dieser wissenschaftlichen Arbeit überschreiten.

Alle Funktionen werden nach dem Schema aus Lundh et al. (2006) angegeben:

$$f = \langle Id, r, I, O, \Phi, Pr, Po, Freq, Cost \rangle$$

Jede Funktion f entspricht dabei einem Tupel und ist einem oder mehreren Agenten im System zugewiesen. Diese Zuweisung wird im Feld r in Kapitel TODO zugewiesen. Die weiteren Felder des Tupels stehen für:

Id Eindeutiger Bezeichner für die Funktion.

$I = \{i_1, i_2, \dots, i_n\}$ Eine Menge aller Eingangsparameter.

$O = \{o_1, o_2, \dots, o_n\}$ Eine Menge aller Ausgangsparameter.

Φ Definiert den Übergang zwischen Eingabe und Ausgabe.

Pr Zustand $s \in S$ aus dem diese Funktion gestartet werden kann.

$Po : S \times I \rightarrow S$ Eine Funktion die den Folgezustand s' anhand des Inputs und des Startzustandes s angibt.

$Freq$ Frequenz in der die Funktionalität ausgeführt werden soll.

$Cost$ Kosten der Ausführung, zum Beispiel Zeit oder Energie. Können konstant aber auch funktionell angegeben werden.

Je nach Typ des Agenten sind die Eingabemenge bei Sensoren $I = \emptyset$ und die Ausgabemenge bei Aktoren $O = \emptyset$. Außerdem wird jede Funktion beschrieben und für die Verwendung in ein MRS eingeordnet. Aus dem Anwendungsfall für das Robotersystem ergeben sich folgende funktionale Anforderungen.

5.2.1 Funktion Aufheben

f_0 (Aufheben):

Id	pickup
I	Objektposition im globalen oder lokalen Koordinatensystem.
O	Erfolgsmeldung.
Φ	Wenn die Objektposition durch den Agenten erreichbar und der Agent voll funktionsfähig ist erfolgt eine positive Rückmeldung. Ansonsten eine negative Erfolgsmeldung.
Pr	Zustand s_1 , Objekt im Nahfeld gefunden und Arm in Candle-Stellung.
Po	Zustand s_2 , Arm in Candle-Stellung und Objekt gegriffen, oder Fehlerzustand s_e bei Fehlschlag.
$Freq$	Einmalige Funktion.
$Cost$	Dynamisch zu ermitteln. Abhängig von Weg und Erreichbarkeit des Objektes.

Beschreibung Die Funktion *pickup* wird in den beiden Robotern Rose und Dummy implementiert und wird für das Aufheben von Objekten genutzt. Beschränkt wird diese Funktionalität durch die Erreichbarkeit der Objektposition. Gerade für den stationären Dummy ist der Arbeitsraum stark eingeschränkt. Rose hat die Möglichkeit während dieser Funktionalität die eigene Position zu verändern um das Objekt zu erreichen. Das Aufheben bedingt, dass das Objekt auf einer parallelen Ebenen zur X-Y-Ebene liegt, da die Greifer Z-Achse parallel zur globalen Z-Achse greift. Die Rotation um die Z-Achse des Objektes ist dabei frei wählbar, da diese durch eine Rotation des Greifers ausgeglichen werden kann. Damit der Roboter nicht auf dem Weg zum Objekt dasselbe berührt wird eine Position über dem Objekt angefahren, bevor der Greifer an der globalen Z-Achse hin abfährt. Diese Beschränkung reduziert die Arbeitsräume der Roboterarme, da ein senkrechter Griff die Zahl der möglichen Gelenkpositionen stark einschränkt und jede Position auf dem senkrechten Pfad erreichbar sein muss. Eine nicht erreichbare Position führt zum Abbruch der Funktion. Eine anschließende sichere Konfiguration des Roboterarms ist nicht gewährleistet. Für Rose gilt des Weiteren auch ein undefinierter Standort nach Abbruch. Wird die Funktion erfolgreich beendet befindet sich der Arm in einer sicheren Konfiguration (*Candle-* oder *Fold-Pose*).

Einordnung Die Funktion ist als Single-Robot Task für einen Single-Task-fähigen Agenten einzuordnen, da nur ein Agent zur Erledigung nötig ist. Die Funktion kann von einem Multi-Task fähigen Agenten intern auch parallel ausgeführt werden, da einzelne Subtasks unabhängig von einander erledigt werden können. Ein Beispiel ist das Öffnen des Greifers während der Bewegungsphase des restlichen Armes oder der mobilen Plattform bei Rose. Diese Multi-Task Anwendung fordert aber einen höheren Koordinierungsaufwand, da Funktionen zeitlich voneinander abhängen (siehe TODO). Für eine Single-Task Anwendung ist der Koordinierungsaufwand

gering, da alle Subtasks seriell ausgeführt werden. Der Konfigurationsaufwand ist mittel, da zwei Agenten für die Funktionalität bereitstehen, welche sich über den Aktionsradius für die Ausführung qualifizieren.

5.2.2 Funktion Ablegen

f_1 (Ablegen):

Id	place
I	Zielposition im globalen oder lokalen Koordinatensystem.
O	Erfolgsmeldung.
Φ	Wenn die Zielposition durch den Agenten erreichbar und der Agent voll funktionsfähig ist erfolgt eine positive Rückmeldung. Ansonsten eine negative Erfolgsmeldung.
Pr	Zustand s_2 , Arm in Candle-Stellung und Objekt gegriffen.
Po	Ruhezustand s_r , Arm in Candle-Stellung oder Fehlerzustand s_e bei Fehlschlag.
$Freq$	Einmalige Funktion.
$Cost$	Dynamisch zu ermitteln. Abhängig von Weg und Erreichbarkeit der Zielpositionen.

Beschreibung Ähnlich Funktion f_0 wird diese Funktionalität den beiden Robotern Dummy und Rose implementiert. Dabei legt der Arm ein Objekt an einer gewünschten Position ab. Beschränkt wird die Funktion auch durch den Arbeitsraum der einzelnen Roboter. Das Objekt kann ebenfalls nur auf einer Ebene abgelegt werden, die parallel zur X-Y-Ebene ist. Die Rotation um die Z-Achse kann beim ablegen vorgegeben werden. im Gegensatz zu f_0 nähert sich der Greifer nicht dem Objekt an, sondern entfernt sich nach dem Ablegen positiv auf der globalen Z-Achse vom Objekt um dieses in einer anschließenden Bewegung nicht durch eine Berührung zu manipulieren. Bei einer erfolgreichen Ausführung beendet die Funktionalität befindet sich der Arm in einer sicheren Konfiguration (*Candle-* oder *Fold-Pose*). Bei einem Fehlschlag ist diese Konfiguration nicht garantiert.

Einordnung Die Funktion gilt als Single-Robot Task. Die Subtasks der Funktion können seriell als auch parallel ausgeführt werden. Dadurch eignen sie sich für Multi-Task, sowie Single-Task Agenten. Je nach Typ ist der Koordinierungsaufwand unterschiedlich hoch. Der Single-Task Agent benötigt wenig Koordinierung da nur ein Startsignal notwendig ist. Bei einem Multi-Task Agenten ist die Koordinierung aufwendiger, da Subtasks zeitlich voneinander abhängen. Zum Beispiel darf der Greifer erst öffnen, wenn das Objekt in der Position ist. Der Konfigurationsaufwand ist gering, da nur der Agent die Aktion ausführen kann, der den Gegenstand hält.

5.2.3 Funktion Objekt identifizieren und lokalisieren

f_2 (Objekt im Raum finden):

Id	findObj
I	Gewünschte Objektdaten.
O	Position des Zielobjektes im globalen Koordinatensystem.
Φ	Findet der Agent anhand der Objektdaten das Objekt wird die Position zurückgegeben. Ansonsten folgt eine Fehlermeldung.
Pr	Ruhezustand s_r , Keine Vorbedingung nötig.
Po	Zustand s_0 , Objekt gefunden. Oder Fehlerzustand s_e
$Freq$	Einmalige Funktion.
$Cost$	Konstant. Abhängig von Prozessor der Recheneinheit, Algorithmus und Kamerasystem.

Beschreibung Diese Funktion identifiziert und lokalsiert ein bestimmtes Objekt im Großraum (Bodenfläche im Arbeitsraum größer als ein Quadratmeter). Je nach Typ der Identifizierung werden unterschiedliche Objektdaten benötigt. Eine Identifizierung basierend auf der Farbe des Objektes benötigt die Objektfarbe. Eine Identifizierung basierend auf der Form benötigt ein 3D-Modell oder eine Kostenfunktion für das Objekt. Die Identifizierung ist abhängig vom verwendeten Algorithmus. Dieser wird in TODO genauer erklärt. Findet die Funktion das gewünschte Objekt nicht oder tritt ein Fehler bei der Erkennung auf bricht die Funktion ab und gibt eine Fehlermeldung zurück. Bei einer erfolgreichen Identifizierung wird anhand von 3D-Daten die Position des Objektes berechnet. Bei dieser Lokalisierung soll eine Genauigkeit von fünf cm erreicht werden und damit nur eine grobe Position. Genauere Ergebnisse werden mit der Naherkennung f_3 erreicht.

Einordnung Ein simpler Single-Robot Task, der von einem Agenten ausgeführt wird. Der Task wird in zwei Subtasks (Identifizieren und Lokalisieren) zerlegt, die serielle ausgeführt werden. Damit bringen Multi-Task Agenten keinen Vorteil. Der Koordinierungsaufwand ist sehr gering, da die Funktion nur angestoßen werden muss. Sollte die Funktion jedoch mit mehreren Kamerasystemen genutzt werden steigt der Koordinierungsaufwand, da der Task entweder parallel oder seriell von allen Systemen ausgeführt werden kann. Je nach Typ steigt der Konfigurationsaufwand, bei einer parallelen Ausführung ist kein Aufwand nötig, da immer alle Kamerasysteme genutzt werden. Eine serielle Ausführung würde die Komplexität steigen, da die Kamerasysteme in einer, möglicherweise priorisierten, Reihenfolge angestoßen werden.

5.2.4 Funktion Nahfeld Erkennung

f_3 (Nahfeld Erkennung):

Id	findObjNear
I	Gewünschte Objektdaten.
O	Position des Zielobjektes im lokalen Koordinatensystem.
Φ	Findet der Agent anhand der Objektdaten das Objekt wird die Position zurückgegeben. Ansonsten folgt eine Fehlermeldung.
Pr	Zustand s_0 , Objekt im Raum gefunden und im Arbeitsraum der Kamera.
Po	Zustand s_1 , Objekt im Nahfeld gefunden. Oder Fehlerzustand s_e
$Freq$	Einmalige Funktion.
$Cost$	Konstant. Abhängig von Prozessor der Recheneinheit, Algorithmus und Kamerasystem.

Beschreibung Diese Funktion identifiziert und lokализiert ein bestimmtes Objekt im Nahfeld (Bodenfläche im Arbeitsraum kleiner als ein Quadratmeter). Es gelten die gleichen Merkmale wie bei f_2 . Der verwendete Algorithmus wird in TODO genauer erklärt. Findet die Funktion das gewünschte Objekt nicht oder tritt ein Fehler bei der Erkennung auf bricht die Funktion ab und gibt eine Fehlermeldung zurück. Zusätzlich zur Position wird auch die Orientierung des Objektes berechnet. Die bestimmte Position entspricht dem Flächenschwerpunkt der Oberfläche des Objektes und soll eine Genauigkeit von fünf mm entsprechen.

Einordnung Wie f_2 einzuordnen. Kann jedoch nicht mit mehreren Kamerasystemen ausgeführt werden, da jedes Kamerasystem auf einem Roboter montiert und mit diesem eng-gekoppelt ist. Dadurch ist der Konfigurationsaufwand konstant, da jedem Roboter ein System zugewiesen wird.

5.2.5 Funktion Agentenlokalisierung

f_4 (Lokalisierung Agent):

Id	loc
I	\emptyset
O	Position des Agenten im globalen Koordinatensystem.
Φ	Kann der Agent sich selbst im Raum lokalisieren, wird die Position zurückgegeben. Ansonsten folgt eine Fehlermeldung.
Pr	In jedem Zustand s_s möglich
Po	Wie der Zustand beim Start s_s .
$Freq$	Abhängig vom Sensor.
$Cost$	Konstant. Abhängig von Prozessor der Recheneinheit, Algorithmus und Kamerasystem.

Beschreibung Diese Funktionalität lokalisiert den entsprechenden Agenten im Raum. Mögliche Algorithmen sind das Nutzen der Odometrie-Daten, bildbasierte Odometrie, bildbasierte Lokalisierung beruhend auf externen Sensoren und Funktionen (f_2) oder bildbasierte Lokalisierung basierend auf Merkmals-Erkennung. Es existieren noch weitere Methoden der Lokalisierung, dafür fehlen aber in dieser Arbeit die entsprechenden Sensoren und Aktoren.

Einordnung Eine an den Agenten enge-gekoppelte Funktionalität die an einen mobilen Roboter oder externen Sensor vergeben werden kann. Ein mobiler Roboter muss multi-tasking fähig sein, da diese Funktion in einer bestimmten Frequenz wiederholt wird und von anderen Funktionen benötigt wird. Ein Beispiel ist f_0 , bei dieser Funktion benötigt Rose die eigene Position zur Fahrwegbestimmung. Dies erhöht den Grad der Koordinierung. Die Konfiguration ist je nach Algorithmus einfach bis komplex. Ein festmontierter Sensor auf einer mobilen Plattform benötigt eine einmalige Konfiguration. Externe Sensoren, zum Beispiel Kamerasysteme, benötigen eine größeren Konfigurationsaufwand. Da zunächst das Kamerasystem ausgewählt werden muss in dessen Arbeitsraum sich der gesuchte Agent befindet. Dadurch steigt auch der Koordinierungsaufwand.

5.2.6 Funktion Übergabe

f_5 (Übergabe Objekt):

Id	handover
I	\emptyset
O	\emptyset
Φ	Bei Fehlschlag erfolgt Fehlermeldung.
Pr	Aus Zustand s_2 möglich. Ein Objekt wurde gegriffen.
Po	Endet in Zustand s_2 . Im Gegensatz zu Pr befindet sich das Objekt im Greifer des anderen Roboters. Bei Fehlschlag endet die Funktion in s_e .
$Freq$	Einmalig.
$Cost$	Dynamisch. Abhängig von der Distanz zwischen den beiden Akteuren und der benötigten Bewegungen der Arme.

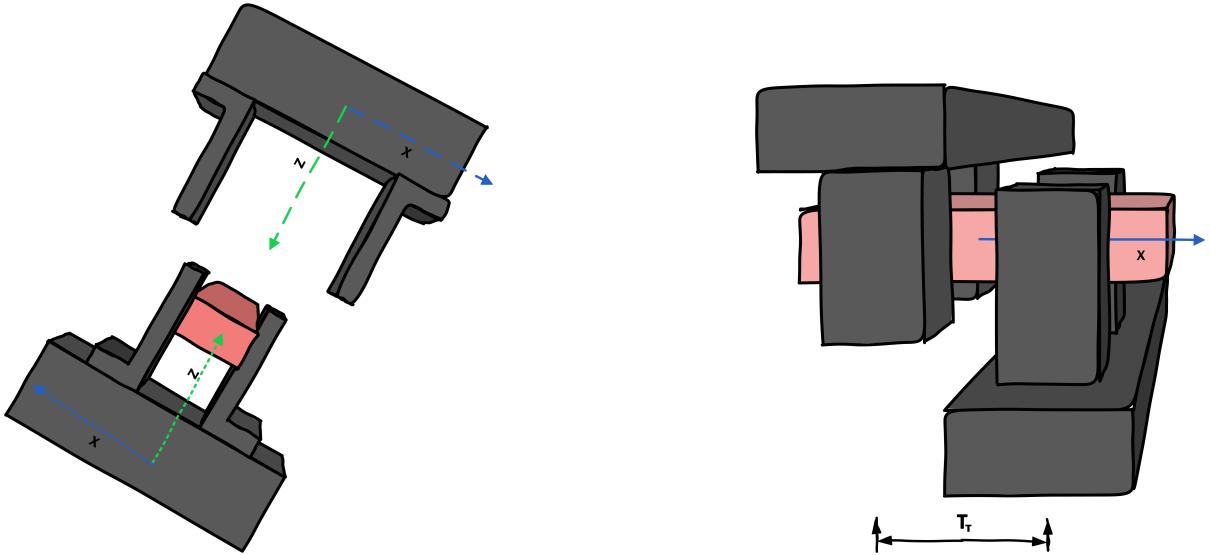
Beschreibung Diese Funktionalität bildet die zentrale Aufgabe dieser Arbeit ab und ist deshalb etwas . Ein Objekt soll zwischen zwei Robotern übergeben werden. Roboter A hat das Objekt gegriffen und soll nun dieses nur an Roboter B übergeben. Dazu müssen zunächst die beiden Arbeitsräume der Roboter eine Schnittmenge aufweisen in der das Objekt übergeben werden kann. Danach muss eine Übergabeposition bestimmt werden. Diese sollte möglichst Energieeffizient sein. Neben der Position spielt die Orientierung im Raum einen wichtigen Faktor, da je nach Objekt und Greifer die Griffpositionen an dem Objekt variiert. Da in dieser Arbeit mit einem einfachen symmetrischen Objekt, einem langgezogenen Quader, und einem Gripper mit parallelen Fingern gearbeitet wird, ist ein entgegengesetzter Griff möglich (siehe Abbildung 18(a)). Dabei entspricht die Pose von Greifer B der invertierten Pose von Greifer A:

$$\xi_{GreiferB} = \ominus \xi_{GreiferA} \quad (11)$$

Neben der Greifposition ist auch die Pfadplanung zur Greifposition wichtig, da der sich nährende Greifer weder mit dem Objekt noch mit dem anderen Greifer kollidieren darf. Ebenfalls kann die Symmetrie der Übergabe genutzt werden. Unter der Bedingung, dass die Finger von Greifer A nicht die selbe Position haben dürfen wie die Finger von Greifer B, muss die Gleichung 11 um eine Translation T_τ entlang der X-Achse des Objekts erweitert werden. Dies ist in Abbildung 18(b) dargestellt. Die Länge der Translation $|T\tau|$ entspricht dabei mindestens der Hälfte der aufsummierten Breiten der Finger der Greifer.

$$\xi_{GreiferB} = \ominus \xi_{GreiferA} \oplus T_\tau \quad (12)$$

Da diese Funktionalität ein MR-Task ist, werden Nachrichten zwischen den Agenten ausgetauscht, um den Ablauf zu koordinieren. Dieser Nachrichtenaustausch wird in den folgenden Sequenz-Diagrammen dargestellt und analysiert. Dabei werden die drei Arten der Übergabe Geben (Abbildung 19(a)), Nehmen (Abbildung 19(b)) und Rendezvous (Abbildung 19(c)) auf



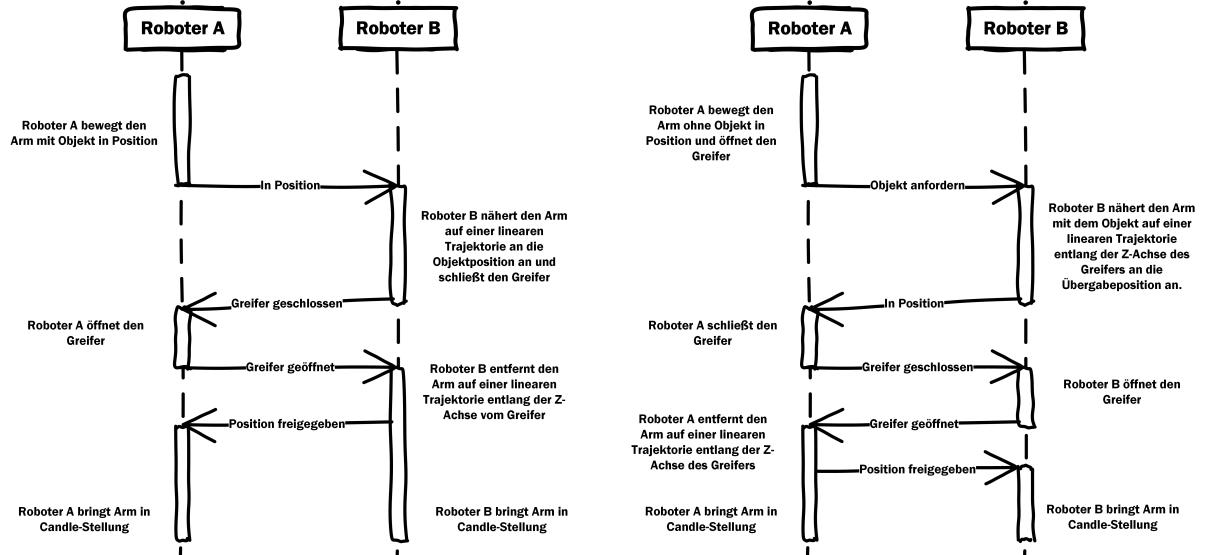
(a) Entgegengesetzter Griff: Die Ausrichtung der Greifer (grau) ist zueinander invertiert. Die Achsen liegen alle parallel, sind jedoch in gegenseitige ausgerichtet. Die Symmetrie des Objekts (rosa) und die parallelen Finger der Greifer ermöglichen eine lineare Projektion entlang der Z-Achsen (grün) ohne Kollisionen mit dem anderen Greifer oder dem Objekt.

(b) Translation(T_τ) an der X-Achse (blau) des Objektes (rose) zur Kollisionsvermeidung der Finger (grau). Die minimale Länge der Translation beträgt die Hälfte der aufsummierten Breite der Finger.

Abbildung 18: Greifer bei der Übergabe

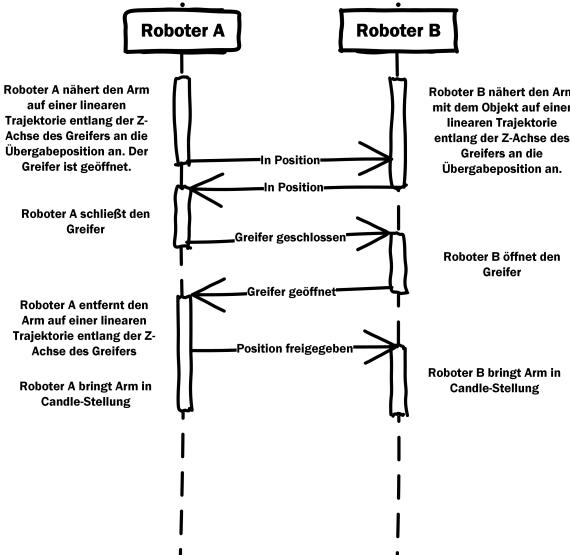
die Anzahl der Nachrichten k , sowie deren Komplexität, untersucht. Ein weiterer Aspekt ist das zeitliche Verhalten t und der Unterschied im Ablauf der drei Varianten.

Die in Abbildung 19(a) dargestellte Variante des *Gebens* wird von dem Roboter initialisiert, der das Objekt trägt (Roboter A). Nach Bestimmung der Übergabe Position und Orientierung bewegt sich zunächst Roboter A in die Pose. Dabei ist eine lineare Annäherung nicht nötig, da noch kein Kollisionspotential besteht. Anschließend wird eine einfache Bestätigung an Roboter B geschickt, der sich anschließend in Bewegung setzt. Diese besteht zunächst aus einer Positionierung nahe des Objektes. Der Greifer ist dabei schon geöffnet und dem anderen Greifer entgegengesetzt orientiert wie in Abbildung 18(a). Folgend kommt die Annäherung auf der linearen Trajektorie entlang der Z-Achse an das Objekt. Sobald der Greifer in Position ist wird dieser geschlossen und eine einfache Bestätigung an den anderen Roboter gesendet. Dieser öffnet seinen Greifer und bestätigt ebenfalls. Durch diese Nachricht löst sich nun Roboter B, ebenfalls auf der Z-Achse, um das Objekt aus dem Greifer von Roboter A zu entfernen. Darauf folgt eine Freigabe an Roboter A, welcher sich nun auch von der Position entfernt, und der Übergang in die sichere Candle-Stellung. Alle Nachrichten, die in dieser Variante ausgetauscht werden, sind einfache Bestätigungen oder Freigaben, die nur vom Zeitverhalten her dynamisch sind und keine weiteren Informationen beinhalten. Benötigt werden, ohne die Positionsbestimmung, vier Nachrichten zur vollständigen Koordinierung ($k_{Geben} = 4$). Für das Zeitverhalten gilt: $t_{Geben} = move_A + move_B + near_B + close_B + open_A + near_B + max(move_A, move_B)$ unter der Bedingung, dass $move_B \gg open_B$, da der Greifer von B während der Bewegung geöffnet wird. $move$ entspricht dabei der Zeit die für eine Bewegung von einer Pose in eine Zielpose



(a) Dieses Sequenzdiagramm zeigt den Ablauf des Gebens. Die Aktion wird dabei vom objekttragenden Roboter (A) initialisiert.

(b) In diesem Sequenzdiagramm wird der Ablauf des Nehmens abgebildet. Die Aktion wird dabei vom objektlosen Roboter initialisiert.



(c) Dieses Sequenzdiagramm zeigt die Rendezvous Variante der Übergabe. Die Aktion kann von beiden Agenten initialisiert.

Abbildung 19: Sequenzdiagramme für Übergabevarianten

benötigt wird, *near* ist die Zeit für die positive oder negative Annäherung, *open* und *close* sind die benötigten Zeiten für das Öffnen und Schließen der Greifer.

In Abbildung 19(b) ist das *Nehmen* dargestellt. Im Gegensatz zum *Geben* wird die Aktion vom dem Roboter initialisiert, an den das Objekt übergeben wird. Nach der Positionsbestimmung bewegt Roboter A den Greifer ohne Objekt an die Übergabeposition und öffnet diesen während der Bewegung. Anschließend fordert er das Objekt an. Darauf nähert sich Roboter B mit dem Objekt auf der linearen Trajektorie entlang der Z-Achse dem Greifer A an und führt den Gegenstand zwischen den Fingern von Greifer A ein. Nach einer Positionsbestätigung schließt Roboter A den Greifer. In diesem Zustand wird das Objekt von beiden Robotern fixiert. Nach der Schließung des Greifers schickt Roboter A eine Nachricht an Roboter B, welcher nun seinen Greifer öffnet und dies mit einer Nachricht bestätigt. Roboter A entfernt nun das Objekt aus dem Greifer von Roboter B und gibt die Position frei nachdem er den Kollisionsraum verlassen hat. Anschließend bewegen beide Roboter die Arme in die sichere Candle-Stellung.

Der Nachrichtenaufwand dieser Variante beträgt $k_{Nehmen} = 5$. Wie beim *Geben*, ist die Komplexität der Nachrichten eher gering, da es sich nur um zeitkritische Freigaben handelt. Für das Zeitverhalten gilt, unter den gleichen Bedingungen und Angaben wie beim *Geben*: $t_{Nehmen} = move_A + move_B + near_B + close_A + open_B + near_A + \max(move_A, move_B)$.

Die Rendezvous-Übergabe in Abbildung 19(c) wird die Initialisierungsbewegung nicht explizit von einem Roboter ausgeführt, sondern von beiden gleichzeitig. Dabei bewegen sich zunächst beide Arme auf die Übergabeposition, wobei sich beide Greifer das letzte Wegstück auf der linearen Trajektorie annähern. Beide Roboter bestätigen ihre Position, sobald sie diese erreicht haben. Anschließend schließt der Roboter A, ohne Objekt, den Greifer und bestätigt. Roboter B öffnet den Greifer und übergibt so das Objekt. Wie bei den anderen Varianten bewegt sich Roboter A nach einer Nachricht nun mit Objekt auf einer linearen Bahn von Greifer B weg. Nach der anschließenden Positionsfreigabe bewegen sich beide Roboter in die sichere Candle-Stellung.

Die Nachrichtenkomplexität ist bei dieser Variante etwas höher, als bei den anderen. Den Unterschied macht die erste Koordinierung, nachdem beide Arme in Position gegangen sind, da nicht definiert ist, welcher Roboter zuerst die Stellung erreicht und somit die Reihenfolge der Nachrichten nicht definiert ist. Die Komplexität des Inhaltes bleibt bei dieser Variante genauso gering, wie bei den anderen Varianten, da es sich nur um Bestätigungen handelt. Die Anzahl der Nachrichten für diese Variante ist $k_{Rendezvous} = 5$. Für das Zeitverhalten gilt, unter den bekannten Bedingungen und Angaben: $t_{Rendezvous} = \max(move_A + near_A, move_B + near_B) + close_A + open_B + near_A + \max(move_A, move_B)$.

Ein Vergleich der Kennzahlen der Varianten zeigt, dass der Koordinierungsaufwand beim *Geben* am geringsten ist. Dies liegt vor allem an der Anzahl der Nachrichten: $k_{Geben} < k_{Nehmen} = k_{Rendezvous}$. Außerdem ist die Komplexität der Nachrichten untereinander identisch, mit der Ausnahme am Anfang der *Rendezvous*-Variante. Für das Zeitverhalten gilt $t_{Rendezvous} < t_{Geben} = t_{Nehmen}$. Dies liegt vor allem an der Parallelisierung der Bewegung am Anfang der Aktion.

Einordnung Diese Funktion ist eng-gekoppelt und benötigt einen hohen Aufwand an Koordinierung. Neben der Übergabeposition und -orientierung müssen auch die Zeitvorgaben für das Öffnen und Schließen der Greifer, sowie die Bewegungen koordiniert werden. Dies macht diese Funktionalität zur komplexesten im ganzen System. Dies liegt vor allem daran, dass diese Funktion als MR-Task eingeordnet wird, da sie von zwei Agenten zusammen ausgeführt wird. Diese Agenten sollten Multi-Task fähig sein, da einige Subtasks parallel ausgeführt werden sollen, zum Beispiel das Öffnen des Greifers, während der Arm in eine Position fährt. Ebenfalls komplex ist die Positionsbestimmung der Übergabe. Zunächst müssen dabei die beiden Arbeitsräume einen Schnittraum aufweisen. Eine notwendige Positionskorrektur kann dabei nur vom mobilen Roboter ausgeführt werden. Dies muss unter anderem in der Konfiguration berücksichtigt werden. Dadurch steigt auch der Konfigurationsaufwand für die Aufgabe, da sichergestellt werden muss, dass die Arbeitsräume der beiden Roboter sich überhaupt schneiden können. Weitere Aspekte der Konfiguration sind die Kompatibilität der Greifer zum Objekt und die Erreichbarkeit der nachfolgenden Ziele.

5.3 Agenten

Aus den Nicht-Funktionalen Anforderungen und den Funktionen lassen sich die folgenden Agenten im MRS extrahieren. Diese Agenten bestehen aus den Sensoren aus Kapitel 4.3 und den Robotern Rose und Dummy. Für die Roboter steht die Zuweisung der Funktionen und Rollen im System fest, die anderen Aktoren und Sensoren werden den Funktionen zugewiesen. Für Funktionen bei denen unterschiedliche Sensoren oder Aktoren zur Verfügung stehen wird evaluiert welcher Hardware die Funktion am besten löst.

5.3.1 Raumüberwachung

Die Raumüberwachung ist zur Identifizierung und Lokalisierung von Objekten zuständig (siehe f25.2.3). Dabei soll das Kamerasytem eine große Fläche mit einer hohen Auflösung observieren. Als Hardware Lösungen standen dabei das XTion Kamera System oder der Argos3D Sensor zur Auswahl. Da der Tiefensor jedoch eine kleinere Auflösung als die XTion hat und kein Farbbild liefert, wird für die Umsetzung des MRS die XTion ausgewählt. Diese wurde plangemäß nach Abbildung 7 und 8 an der Wand montiert. Zur Befestigung wurde eine zusätzliche Halterung mit dem 3D-Drucker gedruckt. Diese ermöglicht einen größeren Winkel in der Ausrichtung. Der eingestellte Winkel beträgt -62 ° um die globale X-Achse. Durch die Höhe und den eingestellten Winkel ergibt sich eine Observationsfläche auf der globalen X-Y-Ebene von 5,075 m². In Abbildung 20 sieht man eine aufgenommene Punktfolge der angebrachten Kamera. Auf dieser ist die Trapezform der Punktfolge sehr gut sichtbar. Diese ergibt sich aus der Position und Orientierung der Kamera. Bedingt dadurch weisen nähere Objekte eine größere Punktdichte auf als entfernte Objekte. Auf der Abbildung sind zwei Bereiche ohne Punkte markiert. Die blaue Markierung zeigt auf eine Verschattung durch den Roboterarm. Der grüne Bereich weist viele schlecht gematchte Punkte zwischen Farb- und IR-Bild auf. Dies liegt vor allem an den schlecht reflektierten IR-Punkten auf dem Heizkörperrgitter.

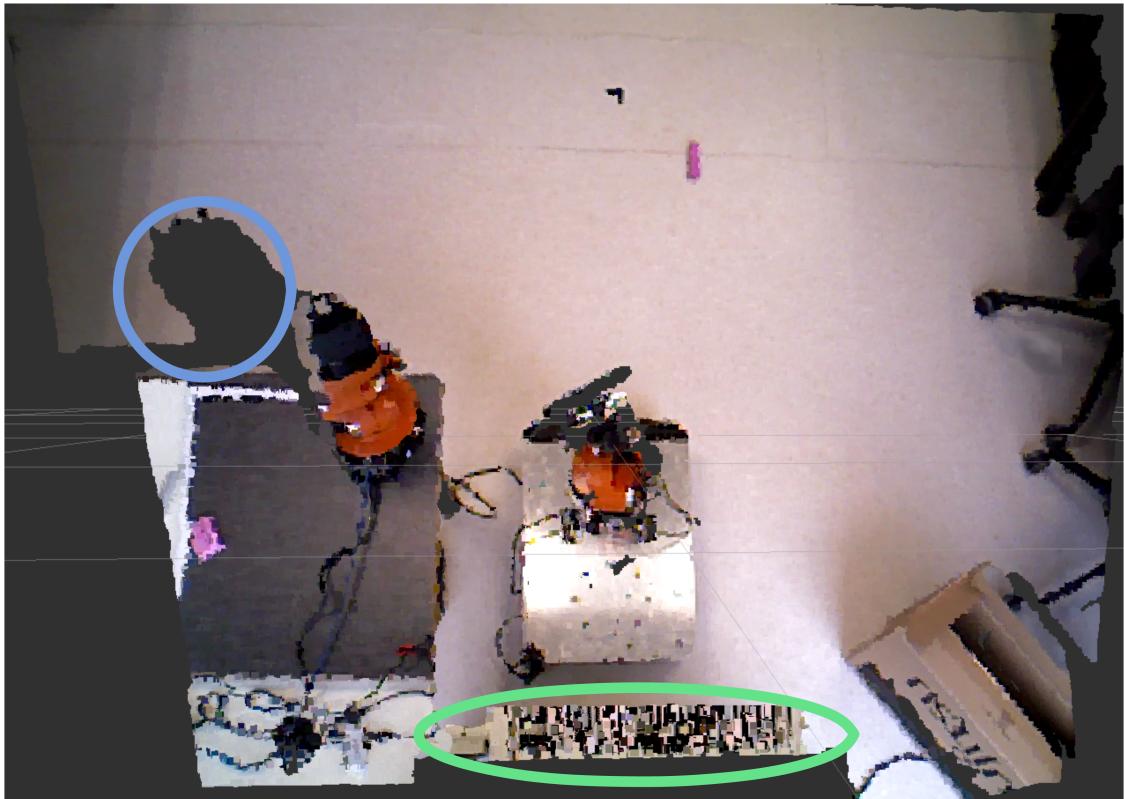
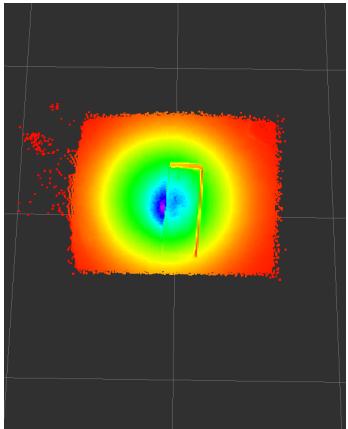


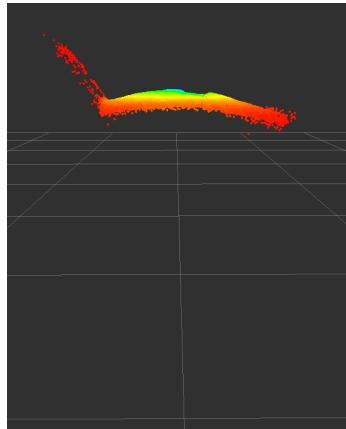
Abbildung 20: Aufnahme der XTion als Raumüberwachung. Blaue Markierung: Toter Winkel der IR Aufnahme. Grüne Markierung: Heizkörper mit diffusen IR-Reflektionen.

5.3.2 Nahfelderkennung

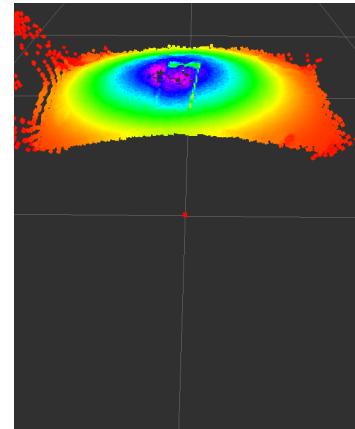
Die Nahfelderkennung setzt die Funktionalität f_3 (siehe 5.2.4) um. Dabei wird der Sensor auf dem mobilen Roboter befestigt, um ein Objekt auch im Nahfeld identifizieren und lokalisieren zu können. Die Lokalisierung soll dabei wesentlich genau sein, als bei der Raumüberwachung. Zur Auswahl standen dabei wieder die XTion und der Argos3D Sensor. Der Vorteil des Sensors ist die geringe minimale Reichweite. Diese funktioniert schon ab 100 mm, während die XTion einen mindest Abstand von 800 mm braucht. So ist eine Montage des Argos Sensors am Greifer direkt möglich, während die XTion mit Abstand montiert werden muss. Nachteil ist die zusätzliche Stromversorgung, die der Sensor benötigt, und der fehlende Farbkanal. Dennoch wurde der erste Prototyp mit dem Argos3D Sensor entwickelt, da er direkt am Greifer montiert werden kann. Ziel des Prototypen war es die Punktwolke eines Objektes von dem planaren Hintergrund zu separieren. Der Prototyp nutzt dafür den RANSAC-Algorithmus mit einem Flächenmodell. Jedoch ergab sich schnell, dass dieser Ansatz nicht mit dem Sensor funktioniert. Die Abbildung 21 zeigt drei Aufnahmen einer Punktwolke die mit dem Sensor aufgenommen wurden. Die Aufnahmen entstanden von einem Objekt, dass auf einem Tisch liegt. In der Abbildung 21(a) ist die Draufsicht auf die Punktwolke dargestellt. Dabei stellen die unterschiedlichen Farben die Intensität der gemessenen IR-Werte. Das Objekt ist auf dieser Ansicht nur durch die Gradiennten der Intensität an den Konturen zu erkennen. Die Abbildung 21(b) ist eine Seitenansicht der selben Punktwolke. Das Objekt ist an den fehlenden Punkten in der Mitte der Punktwolke erkennbar. Diese Punkte sind nicht sichtbar, da die Punktwolke konvex gewölbt ist. Punkte



(a) Topansicht der Punktwolke, die Farben repräsentieren die Intensität der zurückgeworfenen IR-Strahlen. In der Mitte sind die Umrisse des Objektes erkennbar. Am Rand sind einige abweichende Punkte zu sehen.



(b) Seitenansicht der Punktwolke. Sichtbar sind in der Mitte das Objekt und die kubische Grundform der Punktwolke, die eigentlich der planaren Ebene entsprechen sollte.

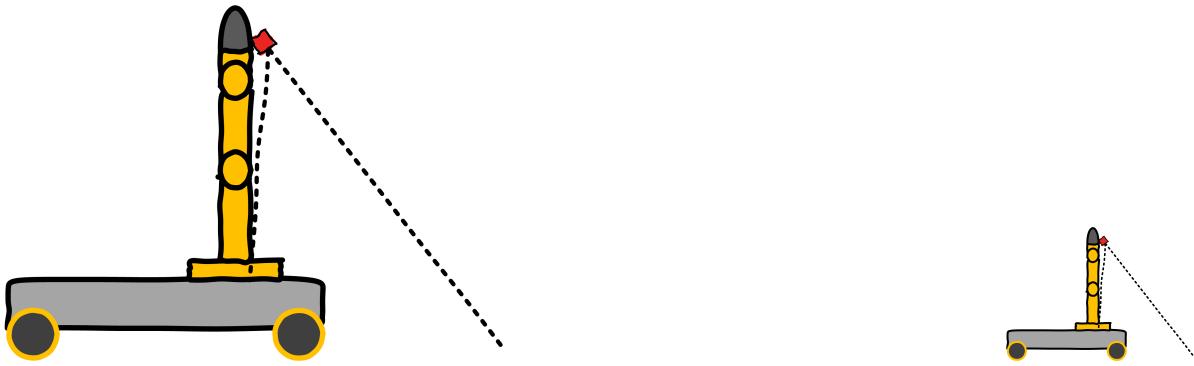


(c) Seitenansicht der Punktwolke mit zusätzlicher Stromversorgung am Sensor. Sichtbar ist wieder das Objekt in der Mitte. Die Krümmung der Oberfläche ist schwächer als bei Abbildung 21(b)

Abbildung 21: Aufgenommene Punktwolke mit dem Argos3D Sensor. Abgebildet ist ein quaderförmiges Objekt auf einem Tisch.

die näher an der Kamera sind verschwinden somit aus dieser Perspektive hinter der restlichen Punktwolke. Die konvexe Wölbung der Punktwolke ist das Problem der Separierung mit einem planaren Modell. Damit das planare Modell auf die Punktwolke angewandt werden kann muss die Fehlerschranke des Algorithmus sehr hoch gesetzt werden. Dadurch werden aber auch die Punkte des Objektes in die Fläche eingeschlossen. Eine Alternative zum planaren Modell wäre die Anwendung eines sphärischen Modells. Dabei muss die Sphäre aber geschlossen sein und nicht nur einen Teilabschnitt der Oberfläche sein. Die Abbildung 21(c) zeigt nochmal die Seitenansicht. Diesmal wurde jedoch der Sensor mit einer zusätzlichen externen Stromquelle versehen. Dieses ergab immer noch eine gekrümmte Oberfläche, welche jedoch schwächer ausgeprägt war als vorher. Ein Versuch die Krümmung aus der Punktwolke herauszurechnen, ähnlich wie bei einer Fischaugenlinse, endete in ziemlichen Abweichungen der Punkte, da die Entfernung zur Oberfläche und die Linsenkrümmung unbekannt ist.

Nach den negativen Tests mit dem Sensor folgten Überlegungen, ob man die XTion Kamera für die Naherkennung einsetzen kann. Problem war die große minimal Distanz für die Erkennung. Damit die benötigten 800 mm zum Boden erreicht werden, ist die Kamera am Greifer montiert, siehe Abbildung 22(a). Befindet sich der Arm in der Candle-Pose ist die Kamera 800 mm vom Untergrund entfernt und erfasst das Feld direkt vor der mobilen Plattform. TODO Infos. Durch eine Drehung des Armes, um das erste oder fünfte Gelenk, kann das Sichtfeld der Kamera erweitert werden. Jedoch gab es auch bei den ersten Tests mit der Kamera einige Probleme. Durch den eingebauten Weißabgleich wurde die Helligkeit des Bildes bei einer zu starken Einstrahlung reduziert. Das führte im Labor unter anderem zu einem schwarzen Bild und resultierte aus der Lichtreflexion der Aluminiumplatte die auf der mobilen Plattform montiert ist. Der Fehler konnte durch Abkleben der reflektierenden Oberfläche beseitigt werden.



(a) Die XTion Kamera (rot) am Greifer montiert. Das Sichtfeld liegt vor der mobilen Plattform. Durch eine Drehung des Armes kann das Sichtfeld um den Roboter erweitert werden.

(b) TODO

Abbildung 22: Die Asus XTion als Hardware Lösung für die Naherkennung

5.3.3 Lokalisierung

Für die Lokalisierung, f_4 in Kapitel ??, sind alle Sensoren potenziell möglich. Die einfachste Variante wäre die Auswertung der Odometriedaten. Dazu wird die Geschwindigkeit der Plattform mit der Laufzeit multipliziert um den zurückgelegten Weg zu berechnen. Um die Genauigkeit zu bestimmen wurden im Rahmen dieser Arbeit Versuche durchgeführt. Dazu wurden zunächst linearen Bewegungen entlang einer Achse ausgeführt und gemessen. Anschließend wurden kombinierte lineare Bewegungen entlang der X- und Y-Achse getestet. Die dabei verwendeten Geschwindigkeiten und Laufzeiten wurden variiert. Gemessen wurde anschließend die zurückgelegte Distanz bei zehn Versuchen pro Messgruppe. Für die anschließende Auswertungen wurden die arithmetischen Mittelwerte für die absolute und relative Abweichung über den Versuchsgruppen berechnet. Die anschließende Bewertung wurde für die einzelnen Versuchsgruppen im Bezug aufeinander und der Geschwindigkeit, sowie Laufzeit vorgenommen. Dabei ergab sich, dass es eine starke positive Kovarianz zwischen Geschwindigkeit und Abweichung gibt. Diese liegt vor allem an den nicht sehr genauen Motoren-Controllern und dem Anhalten, da die Plattform nicht langsam negativ beschleunigt, sondern die Geschwindigkeit direkt auf null setzt. Auch die Kovarianz zwischen Laufzeit und Abweichung ist positiv ausgeprägt, erreicht aber nicht den Wert der Geschwindigkeit. Die größten relativen Abweichungen zeigen die kombinierten Bewegungen bei hohen Geschwindigkeiten. Auch die seitlichen Bewegungen auf der Y-Achse der Plattform zeigen starke Abweichungen. Diese sind auf die omnidirektionalen Räder zurückzuführen. Aufällig sind auch die hohen relativen Abweichungen bei kurzen Laufzeiten. Nach Beobachtung der mobilen Plattform lässt sich dies mit dem Schlupf der Räder und dem glatten staubigen Untergrund erklären. So scheinen die Räder zunächst durchzudrehen. Ein weiteres Problem scheint die Ansteuerung zu sein. Das Testprogramm zur Versuchssteuerung schickt zunächst asynchron das Signal mit den Geschwindigkeiten an die Motorensteuerung. Anschließend pausiert die Steuerung für die eingestellte Laufzeit, bevor das Stopp-Signal asynchron an die Motoren geschickt wird. Dabei entsteht eine unbekannte Verzögerung durch das asynchrone Steuersignal. Zusammengefasst ist diese Methodik alleine zur Lokalisierung des Roboters ungeeignet, da die Abweichungen zu groß sind. Eine mögliche Fehlerkorrektur ist auf Grund des hohen Standardfehlers und den

zufälligen Umweltfaktoren (Schlupf und Signalverzögerung) nur schwer realisierbar.

Eine weitere Option zur Lokalisierung ist die visuelle Odometrie. Dabei wird der optische Fluss einer Bildsequenz genutzt um die eigene Geschwindigkeit und Positionsveränderung zu bestimmen. Dazu müssen zunächst in zeitlich aufeinanderfolgenden Bildern Features detektiert und untereinander zugewiesen werden. Durch die Veränderung der Position auf dem Bild aller Features entsteht ein Vektorfeld. Dieses wird zur Bewegungsschätzung der Kamera genutzt. Diese Methodik benötigt jedoch ein Kamerasystem mit einer geringen Verzögerungszeit. Das XTion Kamerasystem hat an dem Rechner von Dummy eine Verzögerung von einer Sekunde. An dem eingebauten Rechner in der mobilen Plattform beträgt die Verzögerung über drei Sekunden und die Bildrate unter ein Bild pro Sekunde, was für eine zuverlässige Ortung nicht reicht.

Auf der Entwicklungsseite des YouBots wird der Lasersensor Hokuyo URG-04LX-UG01 für die Navigation angeboten. Auch Peter Corke nutzt in (Corke, 2011, Kapitel 6) einen Hokuyo Lasersensor zur Navigation. Dabei stellt TODO SLAM er ein Kartenverfahren vor und nutzt den TODO. Ein Problem beim Mapping ist der gekidnappte Roboter. Diesem fehlen Informationen zu vorhergegangenen Position. Dies kann auf fehlerhaften Berechnungen oder fehlenden Sensordaten beruhen. Da vom Labor keine Karte existiert und das Erstellen dieser den zeitlichen Aufwand der Arbeit überschreitet, wird in dieser Arbeit mit einem Hybridverfahren zwischen Karten- und Odometriebasierter Lokalisierung gearbeitet. Dazu wird eine grobe Position mit Hilfe der Odometrie errechnet. Diese wird in regelmäßigen Abständen mit den Daten des Lasersensors korrigiert, um einen möglichen Schleppfehler zu verhindern.

5.4 Architektur RATS

In diesem Kapitel wird die Software-Architektur *RATS* vorgestellt. RATS steht für *Roboter Action and Task System* und setzt die Anforderungen für dieses MRS um. Dabei liegt ein besonderes Augenmerk auf der einfachen Erweiterbarkeit und Konfiguration des ganzen Systems. So können neue Aktoren und Sensoren ohne großen Aufwand und Veränderung des bestehenden Systems eingefügt werden. Im Folgenden werden das Konzept und die einzelnen Bestandteile der Architektur erklärt. Da das System auf ROS aufbaut wurde schon in der Architektur dem Package und Node Konzept gefolgt.

5.4.1 Konzept

Das Klassendiagramm in Abbildung 23 bildet die Architektur von RATS ab. Das Motiv hinter der Architektur ist die Zerlegung der Aufgaben in Actions und Tasks für Agenten in MRS. Jede Action entspricht dabei einer Funktion die ein Agent anbieten kann. Jede Task ist eine Abfolge von Actions, welche parallel oder seriell vom Agenten ausgeführt werden, und ist ebenfalls eine Action. Dadurch können Tasks eigene Subtasks ausführen. Jede Action und Task besitzt eine ID, durch diese ID kann einem Agenten mitgeteilt werden, welche Action oder Task er ausführen soll. Dieser Befehl kann von dem zentralen RATSCore oder jedem anderen Agenten im System ausgerufen werden. Die Agenten werden in dieser Architektur durch die RATSMember

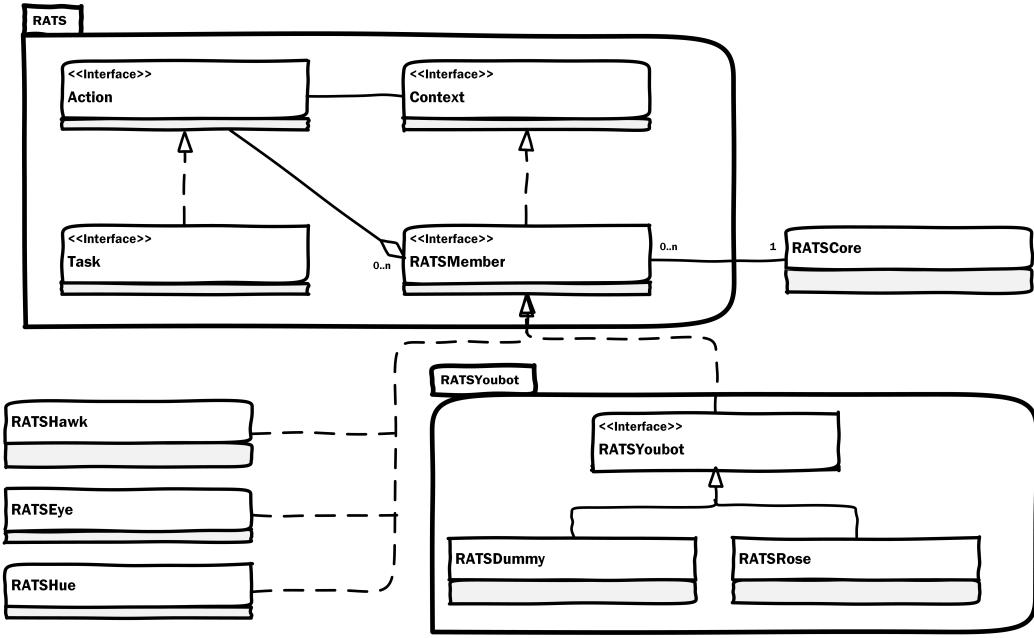


Abbildung 23: Klassendiagramm RATS in der Übersicht. Die zentrale Komponente ist das RATS Paket. Dieses definiert die Schnittstellen Action, Task und RATSMember. Der MRS Controller RATSCore gehört nicht zum RATS Paket, da er je nach Konfigurations- und Koordinierungsaufwand unterschiedlich implementiert werden kann. Das Paket RATSYoubot implementiert die RATSMember von Rosie und Dummy. Alle anderen Sensoren und Aktoren implementieren die RATSMember Schnittstelle und sind jeweils in eigenen Paketen organisiert.

abgebildet. Jeder RATSMember besitzt eine Menge an Actions und Tasks, die von dem Agenten ausgeführt werden. Diese Menge an Actions und Tasks werden bei der Initialisierung an den RATSCore übertragen. Dieser übernimmt die Koordinierung und Konfigurierung im System.

Als Grundlage für die Architektur liegt das Strategy Entwicklungsmuster. Dabei stellt die Action, beziehungsweise der Task, die Strategy und die RATSMember den Kontext dar. Soll eine Action ausgeführt werden wird, lädt der RATSMember die Action. Dazu wird zunächst der aktuelle Kontext an die Action übergeben. Dieser Kontext beinhaltet mögliche Parameter die benötigt werden. Dabei wird zwischen lokalen und globalen Kontext unterschieden. Der lokale ist nur während der Ausführung der Task oder Action verfügbar und wird anschließend gelöscht. Der globale Kontext ist Action und Agent übergreifend. Er kann dazu genutzt werden Umweltinformationen abzubilden und Allgemein verfügbar zu machen. Dieses Konzept ist dem PEIS-Konzept mit seinem Tupelförmigen Speicher angeglichen (siehe Kapitel 3.2.2). Nach der Zuweisung des Kontext wird die execute()-Methode der Action aufgerufen. Diese führt die gewünschte Funktionalität aus. Dabei kann der Kontext geändert werden. Dies ist vor allem bei Sensoren nötig, da diese ihre Daten nicht über einen Rückgabewert an den Aufrufer zurückgeben, sondern die Daten in den globalen Kontext schreiben. Ein Eingabeparameter dient dabei als Pfad für den Speicher. Ein Beispiel ist in Abbildung 24 dargestellt. Dieses zeigt ein Sequenzdiagramm zur Anwendung des globalen Speichers. Leere Antworten wurden zur Übersichtlichkeit ausgelassen. Der Anwender setzt einen Befehl an den Core. Dieser Befehl beinhaltet einen Farbwert-Parameter, der der Oberflächenfarbe eines Objektes entspricht. Dieser Parameter wird unter einem Objekt mit der ID *objid* abgelegt. Anschließend wird ein Sensor aktiviert, der das Objekt finden soll. Bei der

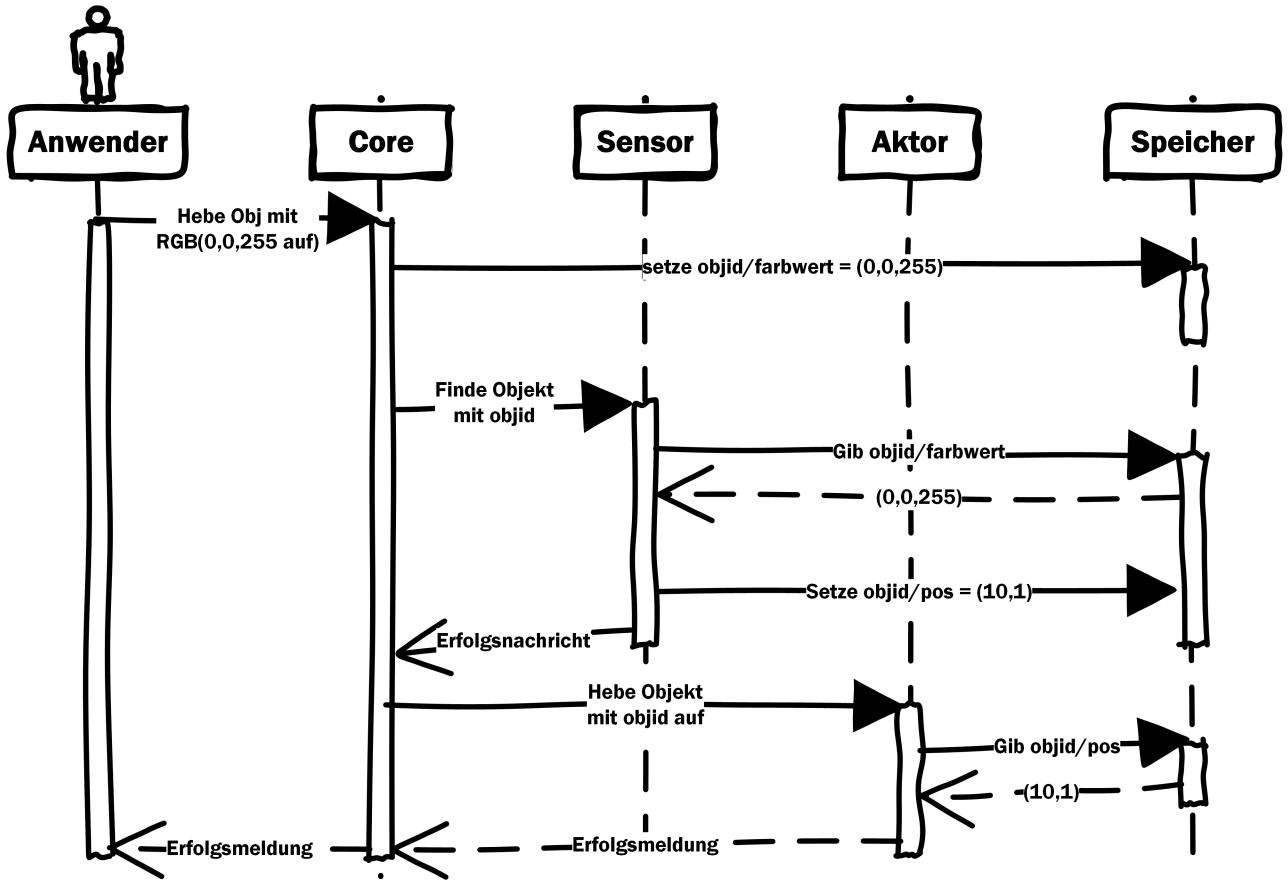


Abbildung 24: Beispielhaftes Sequenzdiagramm zur Erklärung vom globalen Speichersystem. Zur Vereinfachung ist eine Erfolgsmeldung dargestellt und leere Antworten entfernt.

Aktivierung wird der Objektpfad für den Speicher übergeben. Der Sensor holt sich die Daten aus dem globalen Speicher, lokaliert das Objekt und schreibt die Position in den Speicher unter den selben Objektpfad zurück. Im Speicher liegen nun die Daten zu Farbe und Position des Objektes. Nach der Lokalisierung wird an einen Aktor der Befehl zur Aufnahme gesendet. Als Parameter wird nur die Objektid übergeben. Der Aktor lädt die zuvor gesammelten Daten aus dem Speicher und hebt das Objekt an der übermittelten Position auf.

5.4.2 RATSAction und RATSTask

RATSAction und RATSTask bilden die Funktionalitäten innerhalb der Architektur ab. Wie im Konzept beschrieben werden sie von außen angestoßen und durch den Agenten eingesetzt. Damit die Aktion oder der Task erreichbar ist, kann sie über eine ID erreicht werden. Die ID steht dabei für eine Funktionalität des MRS. Eine Funktionalität kann von verschiedenen Agenten unterschiedlich implementiert werden. Sie wird dann als Action oder Task mit der dazugehörigen ID publiziert. Dieses Konzept folgt der Definition aus Kapitel 5.2. Außerdem kann jede Action, einschließlich Tasks, den aktuellen Zustand des Systems ändern. Dies geschieht über den globalen Speicher.

Die zentrale Methode jeder Action ist `execute()`-Methode. Diese besitzt die Logik der Action

und ist immer gleich aufgebaut. Zuerst wird der lokale Kontext ausgelesen. In diesem befinden sich mögliche Zeiger auf den globalen Speicher. Dieser wird darauf ausgelesen. Darauf folgt eine Überprüfung, ob alle benötigten Eingabeparameter vorhanden und gültig sind. Dies betrifft auch den aktuellen Systemzustand. Ist diese Überprüfung ungültig bricht die Funktion mit einer Fehlermeldung ab. Anschließend folgt die Ausführung der Funktion. Auch hier wird bei einer Fehlfunktion eine Rückmeldung gegeben und die weitere Ausführung abgebrochen. Zum Abschluss schreibt die Action zunächst Ergebnisse in den globalen und dann in den lokalen Kontext. Neben der Ausführung besitzt jede Action eine Methode die Kosten für eine Funktion vor zu berechnen. Dabei ist die Kostenfunktion von der Funktion und den Umweltparametern abhängig. Diese Methode wurde für eine automatische Konfiguration vorgesehen. So werden die Actions mit ihrer Funktion-ID und den Umweltparametern, im globalen Speicher, angeboten. Alle Agenten, die eine Action mit dieser ID haben, können nun die Kosten ermitteln und mit diesen bieten. Dabei gewinnt der Agent mit den geringsten Kosten. Die Kostenfunktion greift nur lesend auf den Speicher zu und führt die Aktion auch nicht aus.

RATSTasks bieten die Möglichkeit Actions miteinander zu verbinden. Dabei sind die Tasks selber Actions und können so ineinander verkettet werden. Der einfachste Task ist eine serielle Ausführung der hinterlegten Actions. Bricht eine Action ab, bricht der ganze Task ab. Komplexe Tasks steuern parallele oder bedingte Ausführungen von Actions. Dabei können Fehler abgefangen und behandelt werden. So ermöglicht ein Task, dass ein Roboterarm nach einem Fehlschlag der Action in einen sicheren Zustand (Candle- oder Fold-Pose) wechselt.

5.4.3 RATSMember

Die RATSMember repräsentieren die Agenten im MRS. Jeder RATSMember registriert sich bei der zentralen Komponente, dem RATSCore, mit einer eindeutigen ID. Ist diese ID im System schon vergeben, beendet sich der Agent und es muss eine neue ID vorgegeben werden. Unter dieser ID kann der Agent im MRS erreicht werden. Neben der ID besitzt jeder RATSMember eine Menge an Tasks und Actions die von diesem ausgeführt werden können. Diese Menge übersendet der RATSMember bei der Registrierung an den RATSCore.

Da ROS als Grundlage für diese Arbeit dient, tritt jeder RATSMember als selbstständiger ROS-Node auf. Dies dient vor allem zur Stabilisierung und Wiederherstellbarkeit des Systems. Die ROS Node-ID von einem RATSMember-Node wird auch als ID des RATSMember genutzt. Dadurch kann innerhalb eines MRS die Eindeutigkeit der ID gewährleistet werden. Damit ein RATSMember eine Action ausführt wird, muss der Member eine Nachricht mit der Action-ID und den Parametern erhalten. Zunächst war für diese Kommunikation ein ROS-Topic angedacht, wie im PEIS Konzept vorgesehen (siehe Kapitel 3.2.2). Nach der Entwicklung eines ersten Prototypen zeigte sich jedoch, dass diese Methodik Nachteile bringt. So kann ein Publisher an einem ROS-Topic nicht sicherstellen, ob ein Subscriber die Nachricht erhalten hat. Außerdem ermöglichen Topics keine Rückgabewerte. Deshalb wurde für die weitere Entwicklung die Action-Lib von ROS (siehe 2.2.5) genutzt. Diese ermöglicht einen RPC ähnlichen Aufruf. Gegenüber den Services von ROS können außerdem Zwischenergebnisse zurückgegeben werden. Dies ermöglicht eine Schnittstelle für komplexere MR-Tasks mit hohem Koordinierungsaufwand. Ein Problem

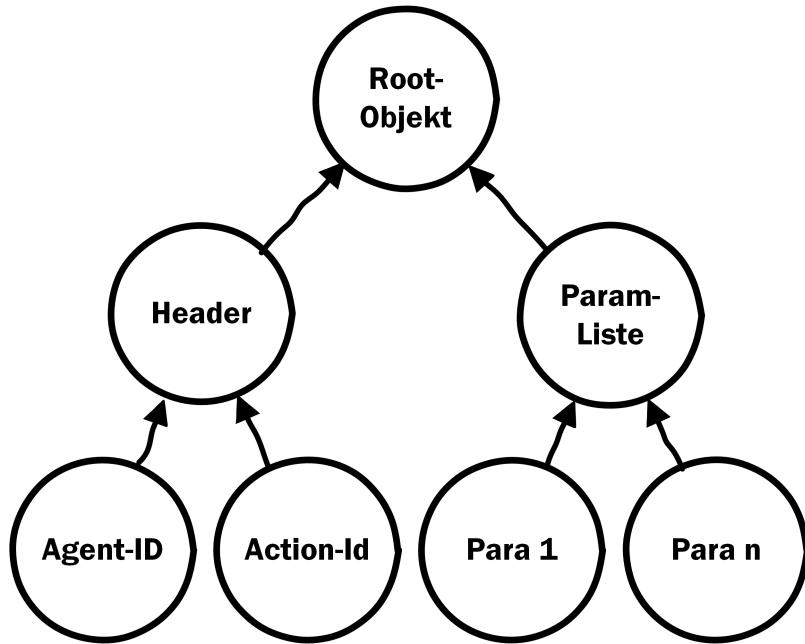


Abbildung 25: Aufbau einer JSON Action Message. Unter dem Root-Element hängen der Header und die Parameter-Liste. Der Header beinhaltet Informationen über die Agent-ID und die Action-ID. Die Parameter-Liste enthält den dynamischen lokalen Kontext.

stellen die dynamischen Parameterlisten dar. Diese sind für das System nötig, da je nach Koordinierungsaufwand unterschiedlich viele Daten mit variablem Datentypen übergeben werden muss. Die Action-Lib, wie auch Services und Topics, lassen keine dynamischen Parameterlisten innerhalb ihrer Messages zu. Darum werden diese Nachrichtenparameter als Strings definiert. Der Inhalt dieser Strings sind JSON-Objekte. Der Aufbau der JSON Objekte ist in Abbildung 25 dargestellt. Das Root-Element besteht aus dem Objekt Header und dem Array der dynamischen Parameterliste. Der Header wird nur in der Kommunikationsschicht ausgewertet und beinhaltet die Agent- und die Action-ID. Die Parameterliste wird vom RATSMember geparsst und dem lokalen Kontext hinzugefügt. Nach Ausführung der RATSAction wird aus dem lokalen Kontext ein JSON-Objekt erstellt, welches anschließend als Rückgabewert in der Message gesendet wird. Sollte eine RATSAction fehlschlagen wird in den Header ein Fehlercode eingetragen, der vom Aufrufer ausgewertet wird. Als JSON-API kommt die RapidJSON Bibliothek zum Einsatz.

5.4.4 RATSCore

Der RATSCore ist der zentrale Koordinierungs- und Konfigurationscontroller. An diesem melden sich alle RATSMember an und ab. Des Weiteren bietet der RATSCore Schnittstellen für Steuerungsoberflächen. Für diese Arbeit wird eine einfache Konsole umgesetzt, aus der die gewünschten Actions und Tasks gestartet werden können. Es sind aber auch eine andere Ansteuerung, wie Sprach- oder Gestensteuerung realisierbar. Die Konfigurations- und Koordinierungsaufgaben werden in dieser Arbeit mit Hilfe von Textdateien vorgegeben. Dabei wird die Agenten-ID, die Action-ID und die nötigen Parameter, zum Beispiel Zeiger auf den globalen Speicher, angegeben. Diese Textdateien werden von dem Core zur Laufzeit eingelesen. Der Core übernimmt dann die

Ansteuerung und Verwaltung der RATSMember.

Der RATSCore ist ein selbstständiger ROS-Node, der mit dem ROS Core zusammen gestartet wird. Da er eine zentrale Komponente darstellt ist er ein potenzieller Single-Point-of-Failure. Fällt er aus bricht das System zusammen. Je nach Systemzustand kann dies zu weiteren Problemen führen. Damit dies abgesichert ist, sind Sicherheitsmechanismen in die unterschiedlichen Kommunikationsschichten eingebaut. So brechen RATSMember ihre Aktionen im Ausfall nicht ab, sondern führen diese bis zum Ende aus. Können sie dann keine Rückmeldung an den RATSCore senden gehen die RATSMember in einen sicheren Zustand über. Dies betrifft besonders mechanische Aktoren, wie die Roboterarme. Bekommt der Core keine Rückmeldung mehr von einem RATSMember nach einer bestimmten Zeit, so bricht er die gesamte Aufgabe ab. Meldet sich ein RATSMember im Ruhezustand ab, werden seine Actions und Tasks aus dem Speicher genommen, so wird er bei zukünftigen Aufgaben nicht mit eingeplant. Dadurch sichert die RATSCore einen sicheren Systemzustand. Ist ein RATSCore einmal abgestürzt müssen auch alle Member einmal neu gestartet werden. Dadurch registrieren sie sich neu an dem RATSCore.

5.5 Inverse Kinematik

In diesem Kapitel wird die Entwicklung der inversen Kinematik vorgestellt. Dabei wird zunächst auf die schon existierenden inversen Kinematiken für den YouBot Arm eingegangen und eine Bewertung dieser vorgenommen. Darauf folgt die Entwicklung eines stark vereinfachten Prototypens, dessen Weiterentwicklung die endgültige inverse Kinematik darstellt. Den Abschluss bildet die Optimierung der Lösungen der inversen Kinematik mit Bezug auf die Pfadfindung.

5.5.1 YouBot Kinematiken

Mit dem YouBot wird bereits das Software-Paket *MoveIt!* zur Bewegungs- und Pfadbestimmung mitgeliefert. Es ist ROS-basiert und nutzt zur Visualisierung und Konfiguration ein RViz-Plugin. Die implementierte Pfadbestimmung verwendet numerische Ansätze für das Lösen der Bewegungsbahnen. Dabei stehen folgende Planungsbibliotheken zur Auswahl: Die *Open Motion Planning Library* (OMPL), *Covariant Hamiltonian Optimization and Motion Planning* (CHOMP) und *Search-based planning* (SBPL). Die OMPL beinhaltet verschiedene Algorithmen zur Bewegungsplanung die alle auf abstrakten Zuständen des Systems beruhen. Dabei wird zwischen probabilistischen (PRM) und Baum-basierten Algorithmen (RRT, EST, KPiece, SBL) unterschieden. Alle Algorithmen lassen sich konfigurieren, dies betrifft vor allem die Anzahl der Iterationen und die Zeitschranke für die Dauer der Berechnung. Für diese Arbeit wurden drei verschiedene Algorithmen (PRM, EST, CHOMP) für eine Bewertung ausgewählt und evaluiert. Bei dieser Evaluierung wurden fünf Bewegungsziele abgesteckt, welche vom Roboter angefahren werden sollten. Die Anzahl der Iterationen wurden in drei Messgruppen (zehn, 100 und 1000 Iterationen) aufgeteilt und mit jedem Algorithmus getestet. Die Zeitschranke wurde auf 60 Sekunden eingestellt, um möglichst optimale Ergebnisse zu erreichen. Pro Bewegungsziel wurde die Dauer der Planung t in Sekunden und die Genauigkeit der Endpose in Zentimetern und Grad gemessen. Dadurch setzt sich ein Bewertungsvektor $x \in \mathbb{R}^3$ zusammen:

	EST	PRM	CHOMP
10	$\begin{pmatrix} 1.84 \\ 6.253 \\ 11.6 \end{pmatrix}$	$\begin{pmatrix} 0.32 \\ 10.486 \\ 17.4 \end{pmatrix}$	$\begin{pmatrix} 0.46 \\ 11.731 \\ 13.2 \end{pmatrix}$
100	$\begin{pmatrix} 15.84 \\ 2.214 \\ 2.1 \end{pmatrix}$	$\begin{pmatrix} 3.84 \\ 3.164 \\ 4.6 \end{pmatrix}$	$\begin{pmatrix} 5.21 \\ 4.756 \\ 2.9 \end{pmatrix}$
1000	OOT	$\begin{pmatrix} 36.15 \\ 1.54 \\ 1.2 \end{pmatrix}$	$\begin{pmatrix} 40.84 \\ 1.78 \\ 0.8 \end{pmatrix}$

Tabelle 3: Die gemittelten Messwerte der MoveIt!-Planer. In den Zeilen sind die unterschiedlichen Iterationen pro Planung gegeben. Die Spalten geben den verwendeten Planer-Algorithmus an. Die Messwerte x setzen sich aus der benötigten Planungszeit x_1 in Sekunden und der euklidischen Distanz der Zielpose zur Planpose zusammen. Dabei wird zwischen linearer x_2 (Zentimeter) und rotierender x_3 (Grad) Distanz unterschieden. OOT steht für eine Überschreitung der Zeitschranke bei allen Bewegungszielen.

$$x = \begin{pmatrix} t \\ \chi \\ \phi \end{pmatrix}$$

Dabei wurden die euklidische Distanz für den linearen und den rotierenden Anteil der Pose ermittelt. Wobei die λ -Funktion den linearen und die θ -Funktion den rotierenden Anteil extrahiert:

$$\chi = \|\lambda(\xi_{Ziel}) - \lambda(\xi_{Plan})\|_2 = \sqrt{\sum_{i=1}^3 (\xi_{Ziel,i} - \xi_{Plan,i})^2}$$

$$\phi = \|\theta(\xi_{Ziel}) - \theta(\xi_{Plan})\|_2 = \sqrt{\sum_{i=4}^6 (\xi_{Ziel,i} - \xi_{Plan,i})^2}$$

Anschließend wurden die Bewertungsvektoren der fünf Bewegungsziele elementweise quadriert aufsummiert.

$$x_{all} = \begin{pmatrix} \sqrt{\sum_{i=1}^5 t_i^2 / 5} \\ \sqrt{\sum_{i=1}^5 \chi_i^2 / 5} \\ \sqrt{\sum_{i=1}^5 \phi_i^2 / 5} \end{pmatrix}$$

Die Ergebnisse pro Iterationen-Messgruppe und Algorithmus sind in Tabelle 3 aufgelistet. In der Auswertung fallen bestimmte Charakteristiken der einzelnen Algorithmen auf. So benötigt EST schon bei wenigen Iterationen mehr Zeit als die anderen Algorithmen. Dafür sind die Abweichungen geringer. PRM und CHOMP sind zeitlich betrachtet sehr ähnlich. Differenzen zeigen sich erst bei der Betrachtung der linearen und rotierenden Abweichungen. So sind die linearen Abweichungen bei PRM immer kleiner als bei den Planungen mit CHOMP. Für die Rotation gilt dies genau umgekehrt. Dort ist CHOMP genauer als PRM. Zusammengefasst sind die Ergebnisse für das MRS dieser Arbeit jedoch ungenügend. Eine Genauigkeit von 1,5 cm linear oder 0,8° rotierend ist für die Anwendung nicht ausreichend. Auch die benötigte Zeit der Planungen ist zu hoch. Ein Grund für diese ist die schlechte Hardware des eingebauten Rechners. Vergleichstests auf Dummy zeigten fünf mal schnellere Ergebnisse.

Eine weitere inverse Kinematik, beziehungsweise Bewegungsplaner, wurde 2013 von der Robotics

& Perception Group der Universität Zürich veröffentlicht (siehe Müggler et al. (2013)). Dieser Planer unterscheidet sich von dem Kuka Ansatz in der Ansteuerung der Hardware. Dabei wird nicht nur die Konfiguration der Gelenke, sondern auch die Drehkraft der Gelenke angesteuert. Dadurch können Bewegungen entlang einer Trajektorie ausgeführt werden. Der Planer basiert ebenso auf der MoveIt!-Basis und nutzt die implementierten Planungsalgorithmen. Für diese Arbeit konnte dieser Bewegungsplaner nicht eingesetzt werden, da die Bibliothek zu dem Zeitpunkt der Entwicklung nicht die installierte Firmware der YouBots unterstützte. Inzwischen sind die benötigten Updates installiert und könnten genutzt werden. Diese Änderung würde jedoch den Aufwand der Arbeit überschreiten. Die inverse Kinematik die dabei genutzt wird ist ein Nachfolger der ersten Entwicklung von Kuka. Diese beruht auf den geometrischen Eigenschaften des Arms und ist in der Arbeit Sharma et al. (2012) beschrieben.

5.5.2 Vereinfachte inverse Kinematik

Dieser erste eigene Ansatz für eine inverse Kinematik war prototypisch für die Funktionen f_0 und f_1 gedacht. Dabei wurde das Modell stark vereinfacht. Dazu wurde die Annahme getroffen, dass die Z-Achse der Pose des Greifers $l_{5,z}$ orthogonal auf der XY-Ebene des Roboter-Koordinatensystems (im Folgenden als *global* bezeichnet) steht:

$$l_{5,z} \parallel -Z_{Global} \quad (13)$$

Diese Annahme lässt sich auf eine konstante Ausrichtung des Greifers beim Aufheben oder Ablegen zurückführen. Dadurch ist nur noch die Rotation um die Z-Achse, sowie die Position im Kartesischen Raum, als Angaben für die Pose nötig. Dadurch wird die Pose ξ aus dem \mathbb{R}^6 -Raum in den \mathbb{R}^4 -Raum reduziert.

Die Überlegung hinter dieser inversen Kinematik sieht vor, die Dimensionen der Pose immer weiter zu reduzieren bis das Problem mit einfachen geometrischen Sätzen zu lösen ist. Dazu wird zunächst der Mechanismus des YouBots untersucht. Dieser beinhaltet fünf Gelenke. j_1 und j_5 sind Drehgelenke um die Z-Achsen ihrer Glieder. j_2 , j_3 und j_4 sind Drehgelenke um die X-Achsen ihrer Glieder. Das globale Koordinatensystem ist im Mittelpunkt der runden Basis des Roboters. Die Z-Achse zeigt dabei nach oben. Der Null-Punkt der Z-Achse liegt auf der Kontaktfläche zum Untergrund. Es gilt:

$$l_{0,z} \parallel +Z_{Global} \quad (14)$$

Aus den Gleichungen 13 und 14 folgt:

$$l_{0,z} \parallel l_{5,z} \quad (15)$$

Dies kann durch Betrachtung noch weiter spezifiziert werden. Da die Z-Achse vom Greifer nach

$-z_{Global}$ und die Z-Achse von l_0 nach $+z_{Global}$ zeigt (Gleichung 13) folgt:

$$l_{0,z} \uparrow\downarrow l_{5,z} \quad (16)$$

Da die vierte Komponente der Pose α die Drehung des Greifers um die Z-Achse ist und die Drehgelenke von j_1 und j_5 in der Z-Achse parallel zu den Z-Achsen ihrer Glieder stehen ergibt sich aus den Gleichungen 13, 14 und 16:

$$j_1 - j_5 = \xi_\alpha \quad (17)$$

j_5 geht negativ ein, weil die Z-Achse des Gelenks entgegengesetzt zur globalen Z-Achse liegt. Umgestellt nach j_5 ergibt sich:

$$j_5 = j_1 - \xi_\alpha \quad (18)$$

Damit ist die erste Gelenkkonfiguration definiert und die Pose kann um eine Dimension, die Rotation des Greifers, reduziert werden. Da beim YouBot alle Glieder nicht in sich verdreht sind, liegen die Dreh-Achsen der Gelenke j_2 , j_3 und j_4 parallel:

$$\forall l \in L : l_\alpha = 0 \implies r(j_2) \parallel r(j_3) \parallel r(j_4) \quad (19)$$

Die $r(j)$ -Funktion extrahiert die Drehachsenkomponente des Gelenks j . Dadurch liegen alle Gelenke und Glieder in einer Ebene. Diese steht senkrecht auf der globalen XY-Ebene. Der Winkel der Schnittgeraden zur X-Achse ist dabei abhängig vom Drehwinkel von j_1 . Überträgt man die X und Y Komponente der Pose aus dem kartesischen Koordinatensystem in Polarkoordinaten ergibt sich:

$$r = \sqrt{\xi_x^2 + \xi_y^2} \quad (20)$$

$$\phi = \cos\left(\frac{\xi_x}{r}\right) \quad (21)$$

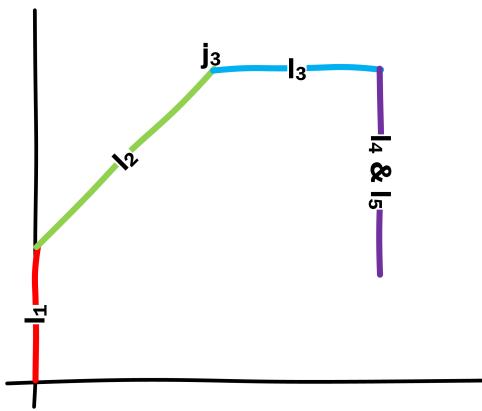
Da die Kosinus-Funktion aber nicht vollständig definiert ist wird auf die atan2 -Funktion mit zwei Parametern zurückgegriffen. Diese ist in allen gebräuchlichen Programmiersprachen vorhanden. Dadurch ergibt sich:

$$\phi = \text{atan2}(\xi_y, \xi_x) \quad (22)$$

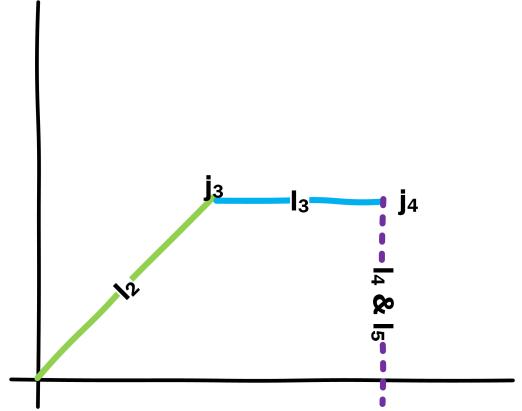
ϕ entspricht dabei dem Drehwinkel von j_1 . r entspricht der Kombination aus x und y . Dadurch kann man die Anzahl der Dimensionen der Pose wieder reduzieren. Diese setzt sich jetzt nur noch aus dem z und dem neuen r Wert zusammen. Die Abbildung 5.5.2 zeigt die nun reduzierte

Umgebung. Dabei wurde der Nullpunkt des Koordinatensystems in die Position von j_2 verlegt. Diese Translation ist durch die Glieder l_0 und l_1 gegeben, die einen Offset entlang der globalen Z-Achse und der X-Achse von l_1 erzeugen. Dieser wird von dem neuen Wert r abgezogen. Da j_5 nur um Z-Achse rotiert und es keinen Offset zwischen l_4 und l_5 gibt können diese zu dem neuen Glied l_{45} zusammengefasst werden. Die Länge des Gliedes ergibt sich durch die Summe der Länge der Glieder.

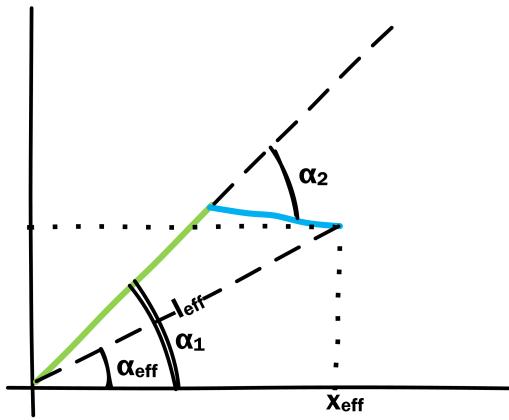
$$a_{45} = a_4 + a_5 \quad (23)$$



(a) Mechanismus des Arms im globalen Koordinatensystem.



(b) Neues Koordinatensystem mit Ursprung in j_2



(c) Kennzeichnung der zu ermittelnden Winkel.
Die Position $\{x_{eff}, y_{eff}\}$ von j_4 wird durch die Offsets von l_1, l_4 und l_5 berechnet.

Da l_{45} senkrecht auf der X-Achse steht kann die Position $\{x_{eff}, y_{eff}\}$ von j_4 aus der Zielpose und der Länge von l_{45} , sowie dem Offset von l_1 berechnet werden:

$$x_{eff} = \xi_x - a_{1,x} \quad (24)$$

$$y_{eff} = \xi_y + a_{45} - a_{1,y} \quad (25)$$

Mit Hilfe des Kosinus-Satzes lassen sich nun die Winkel α_1 und α_2 berechnen:

$$l_3^2 = l_2^2 + l_{eff}^2 + 2 * l_2 * l_{eff} * \cos(\alpha_1 - \alpha_{eff}) \text{ mit } l_{eff} = \sqrt{x_{eff}^2 + y_{eff}^2} \text{ und } \cos(\alpha_{eff}) = \frac{x_{eff}}{l_{eff}} \quad (26)$$

$$l_{eff}^2 = l_2^2 + l_3^2 + 2 * l_2 * l_3 * \cos(\pi - \alpha_2) \quad (27)$$

Umgestellt nach α_1 und α_2 ergibt sich:

$$\alpha_1 = \alpha_{eff} + \arccos\left(\frac{l_3^2 - l_2^2 - l_{eff}^2}{2 * l_2 * l_{eff}}\right) \quad (28)$$

$$\alpha_2 = \pi - \arccos\left(\frac{l_{eff}^2 - l_2^2 - l_3^2}{2 * l_2 * l_3}\right) \quad (29)$$

Damit sind alle Winkel bekannt, die in der Kinematik benötigt werden. Diese werden nun mit den entsprechenden Winkeloffsets addiert. Diese Offsets berechnen sich aus Grundeinstellung der Gelenke.

Diese inverse Kinematik ist, wie beschrieben sehr vereinfacht. Sie ermöglicht nur eine sehr eingeschränkte Menge an möglichen Posen. Dies ist für diese Arbeit nicht ausreichend, da für die Übergabe auch schräge Haltungen des Greifers möglich sein sollen. Auch ein Griff nach oben ist mit der bisherigen inversen Kinematik nicht möglich, da $\alpha_2 > 0$ ist. Deshalb wurde diese inverse Kinematik weiter entwickelt. Diese Weiterentwicklung beruht auf Beobachtungen der Mechanik, dabei lassen sich Überschläge bei zwei Gelenken j_1 und j_3 entdecken. Ein Überschlag bezeichnet den Wechsel der Einstellung eines Winkel α von $\alpha \leq \pi$ zu $\alpha > \pi$. Dies kann bei j_3 als Ellbogen hoch oder runter beschrieben werden. Bei j_1 beschreibt die Vorstellung eines Griffes nach vorne oder nach hinten die Konfigurationen am besten. So zeigt der Arm nach vorne, wenn $j_1 = 0$ und $j_2 > 0$ oder $j_1 = \pi$ und $j_3 < 0$. Dies resultiert in vier möglichen Konfigurationen die der Roboterarm einnehmen kann um eine Position zu erreichen. Im besten Fall sind alle vier Konfigurationen möglich, im schlechtesten erfüllt keine Konfiguration die Pose. Die folgende inverse Kinematik versucht für die vier Konfigurationen die möglichen Gelenkkonfigurationen zu berechnen.

5.5.3 Weiterentwicklung: Inverse Kinematik

Die Weiterentwicklung der inversen Kinematik soll die vollständige Pose und alle möglichen Konfigurationen berechnen. Dazu reichen nicht mehr einfache Extraktionen der benötigten Komponenten, da sich diese nun gegenseitig beeinflussen.

Zunächst wird das Koordinatensystem in den Mittelpunkt von j_1 gebracht. Dazu wird die Ziel-

pose ξ um den T_0 verschoben, diese setzt sich aus den Offsets von l_0 zusammen:

$$\xi_0 = \xi * T_0 \quad (30)$$

Anschließend wird, wie in Gleichung 22, j_1 aus der X und Y Komponenten Pose bestimmt.

$$j_1 = \text{atan2}(\xi_{0_Y}, \xi_{0_X}) \quad (31)$$

j_1 stellt eine der Möglichkeiten für das erste Gelenk dar und bildet den Griff nach *vorne* ab. Der Griff nach *hinten* wird durch eine Rotation um π erreicht. Da für die Einstellung von j_1 gilt:

$$-\pi \leq j_1 \leq +\pi \quad (32)$$

Ergibt sich für die zweite Konfiguration:

$$j_1 = \begin{cases} j_1 + \pi & \text{für } j_1 < 0 \\ j_1 - \pi & \text{für } j_1 \geq 0 \end{cases} \quad (33)$$

Anschließend wird die Pose wieder in ein anderes Koordinatensystem übertragen. Dabei besteht die homogene Transformation T_1 aus einer Translation v_1 und einer Rotation θ_1 . Zur besseren Lesbarkeit werden diese getrennt und die Rotation als Roll-Pitch-Yaw angegeben:

$$\mathbf{v}_1 = \begin{pmatrix} -a_{1_x} \\ 0 \\ -a_{1_z} \end{pmatrix}, \boldsymbol{\theta}_1 = \begin{pmatrix} 0 \\ 0 \\ -j_1 \end{pmatrix}, \xi_1 = \xi_0 * T_1(\theta_1, v_1) \quad (34)$$

Als nächstes wird die Gelenkkonfiguration für j_5 berechnet. Dazu wird die Pose ξ_1 als homogene Koordinate A betrachtet. Indizes weisen auf die Stelle der Matrix:

$$j_5 = \text{atan2}(A_{21} * \cos(j_1) - A_{11} * \sin(j_1), A_{22} * \cos(j_1) - A_{12} * \sin(j_1)) \quad (35)$$

Wie in der einfachen inversen Kinematik folgt jetzt die Transformation aus dem 3D-Raum in die 2D-Ebene des Arms. Da hier aber noch die Rotation des Greifers benötigt wird, muss zunächst diese Rotation übertragen werden. Dazu werden die Rotationskomponente der Y- und Z-Achse der Pose ξ_1 extrahiert:

$$\hat{\boldsymbol{\theta}}_y = \xi_{1_{RotY}}, \hat{\boldsymbol{\theta}}_z = \xi_{1_{RotZ}} \quad (36)$$

Für die weiteren Schritte wird die Normale der Armebene $\hat{\mathbf{m}}$ benötigt:

$$\hat{\mathbf{m}} = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} \quad (37)$$

Aus dieser und der Z-Rotation lässt sich nun der Rotationsvektor $\hat{\mathbf{k}}$ bestimmen um den die Endpose für die Rotation gedreht werden muss:

$$\hat{\mathbf{k}} = \hat{\mathbf{m}} \times \hat{\theta}_z \quad (38)$$

Dadurch ergibt sich für die Z-Achsen Rotation der neuen Pose ξ'_1 :

$$\hat{\theta}'_z = \hat{\mathbf{k}} \times \hat{\mathbf{m}} \quad (39)$$

Als nächstes wird die Rotation um die Y-Achse der Pose $\hat{\theta}'_y$ berechnet. Dazu wird die Rodrigues-Formel angewandt. Dafür müssen zunächst die $\cos\theta$ und $\sin\theta$ Werte berechnet werden. Diese lassen sich mit Hilfe des Kreuz- und Skalarproduktes bestimmen:

$$\cos\theta = \hat{\theta}_z \bullet \hat{\theta}'_z, \sin\theta = (\hat{\theta}_z \times \bullet \hat{\theta}'_z) \bullet \hat{\mathbf{k}} \quad (40)$$

Rodrigues angewandt:

$$\hat{\theta}'_y = \cos\theta * \hat{\theta}_y + \sin\theta * (\hat{\mathbf{k}} \times \hat{\theta}_y) + (1 - \cos\theta) * (\hat{\mathbf{k}} \bullet \hat{\theta}_y) * \hat{\mathbf{k}} \quad (41)$$

Da $\hat{\theta}'_y$ und $\hat{\theta}'_z$ nun bekannt sind lässt sich aus dem Kreuzprodukt die X-Rotation $\hat{\theta}'_x$ bestimmen:

$$\hat{\theta}'_x = \hat{\theta}'_y \times \hat{\theta}'_z \quad (42)$$

Diese drei Komponenten beinhalten die projizierte Rotation der Pose ξ'_1 . Die Position der Pose bleibt erhalten. Als nächstes wird das Nicken des Greifers berechnet. Dieser ergibt sich aus der Summe der Winkel von j_2 , j_3 und j_4 . Dazu wird, wie bei der Berechnung zu j_5 , der Pose als homogene Koordinate A' betrachtet und zunächst x_{eff} und y_{eff} aus der projizierten Pose extrahiert und der Gesamtwinkel α_{eff} berechnet:

$$x_{eff} = A'_{13}, y_{eff} = A'_{13}, \alpha_{eff} = \text{atan2}(y_{eff}, x_{eff}) \quad (43)$$

Daraus lässt sich nun die Position von j_4 berechnen. Dazu wird die Summe der Länge von l_4

und l_5 zusammengefasst $d_{45} = l_4 + l_5$. Die neue Position berechnet sich nun aus:

$$j_{4pos} = \begin{pmatrix} \xi'_{1x} - d_{45} * \sin(\alpha_{eff}) \\ \xi'_{1y} - d_{45} * \cos(\alpha_{eff}) \end{pmatrix} \quad (44)$$

$$|j_{4pos}| = \sqrt{j_{4pos_x}^2 + j_{4pos_y}^2} \quad (45)$$

Nun lässt sich schon bestimmen, ob die Zielpose nicht im Arbeitsraum des Roboters liegt. Die Zielpose liegt nicht im Arbeitsraum, wenn:

$$a_2 + a_3 < |j_{4pos}| \quad (46)$$

Nachdem nun die beiden Endgelenke j_1 und j_5 berechnet wurden, wird mit dem mittleren Gelenk j_3 fortgefahrene. Dies begründet sich durch den zweiten Konfigurationsschritt Ellbogen *hoch* oder *runter*. Diese Konfiguration ist Richtungsvorgabe, in welche sich das Gelenk rotieren soll (positiv oder negativ). Dazu wird zunächst mit dem Kosinussatz der Kosinus von j_3 ermittelt:

$$\cos(j_3) = \frac{s_E - a_2^2 - a_3^2}{2 * a_2 * a_3} \text{ mit } s_E = j_{4pos_x}^2 + j_{4pos_y}^2 \quad (47)$$

Da die *acos*-Funktion nur innerhalb vom Wertebereich $[-1; 1]$ definiert ist und generell die *atan2*-Funktion genutzt werden soll ergibt sich folgende Fallunterscheidung für j_3 :

$$j_3 = \begin{cases} 0 & \text{für } \cos(j_3) > 0.9999 \\ \pi & \text{für } \cos(j_3) < -0.9999 \\ \text{atan2}(\sqrt{1 - \cos(j_3)^2}, \cos(j_3)) & \text{für Rest} \end{cases} \quad (48)$$

Nun kann je nach Konfiguration der Winkel positiv $j_3 = j_3$ oder negativ $j_3 = -j_3$ betrachtet werden. Er wird für die Berechnung von j_2 und j_4 weitergenutzt. Dazu wird vom Vollkreis (2π) der der Winkel zwischen den Gelenken l_2 und l_3 abgezogen:

$$j_2 = 2\pi - \text{atan2}(j_{4pos_y}, j_{4pos_x}) - \text{atan2}(a_3 * \sin(j_3), a_2 + a_3 * \cos(j_3)) \quad (49)$$

Das letzte fehlende Gelenk j_2 ergibt sich aus der dem Nicken des Greifers α_{eff} , abzüglich der Gelenke j_3 und j_2 :

$$j_4 = \alpha_{eff} - j_2 - j_3 \quad (50)$$

Auf alle Winkel muss noch ein Offset addiert werden, da die Grundeinstellung der inversen Kinematik anders ist als bei der Grundeinstellung der Ansteuerung. Bei der Ansteuerung befinden sich alle Gelenke bei null Grad, wenn der Arm in der *Fold*-Pose ist, die inverse Kinematik geht

von der *Candle*-Pose als Grundstellung aus. Die vollständige inverse Kinematik wird im folgenden als $Ik(\xi, c)$ -Funktion genutzt. Dabei ist ξ die Pose und c die Konfigurationsvariante aus der Menge der Konfigurationen C . Für diese inverse Kinematik besteht die Menge aus vier Elementen die abstrakt beschrieben werden können mit: *Vorne-Ellbogen-hoch*, *Vorne-Ellbogen-runter*, *Hinten-Ellbogen-hoch* und *Hinten-Ellbogen-runter*.

5.5.4 Optimierte Pfadplanung

Neben der inversen Kinematik ist auch die Pfadplanung wichtig. Da die inverse Kinematik maximal vier mögliche Konfigurationen zurückgibt, muss die Verkettung der Posenübergänge optimiert werden, um möglichst geringe Kosten (Bewegungsenergie und Zeit) zu erreichen. Durch Betrachtung ergibt sich, dass Posenübergänge besonders teuer sind, wenn die Konfiguration gewechselt muss. Dabei ist die Basisrotation, j_1 betreffend, teurer, als die Ellbogen-Konfiguration, j_3 . Dies liegt an dem Einfluss den eine Konfiguration hat. So beeinflusst eine Konfigurationsänderung von j_1 alle anderen Winkel. Eine Änderung an der Konfiguration von j_3 beeinflusst dagegen nur die Gelenk j_2 und j_4 . Die Kosten K für eine Bewegung ermittelt sich aus den Einstellungen der Winkel bei der Start- und der Zielpose:

$$K(J_{start}, J_{ziel}) = \sum_{i=1}^n (J_{start_i} - J_{ziel_i})^2 \quad (51)$$

Für eine Bewegung Γ , die aus n Posen besteht $\Gamma : \xi_0 \oplus \xi_1 \oplus \dots \oplus \xi_n$, gilt auf Grund der Beobachtung folgende Lösung als optimal (ξ_0 entspricht der aktuellen Pose):

$$K_{min} = \min_{c \in C} \left(\sum_{i=0}^{n-1} K(Ik(\xi_i, c), Ik(\xi_{i+1}, c)) \right) \quad (52)$$

Dies funktioniert nur solange alle Posen innerhalb einer Konfiguration erreicht werden können. Ist dies nicht möglich muss die günstigste Alternative gefunden werden. Für dieses Problem eignet sich die Graphentheorie. Die Knoten entsprechen dabei allen Gelenkeinstellungen aus der Kombination der Posen und der Konfigurationen der inversen Kinematik. Die Kanten repräsentieren die Kosten für den Übergang zwischen zwei Gelenkeinstellungen. Jeder Knoten besitzt gerichtete Kanten zu den Knoten die auf der folgenden Pose basieren. Am Übersichtlichsten ist der Graph, wenn Knoten basierend auf der gleichen Pose nebeneinander und Knoten mit der gleichen Konfiguration untereinander angeordnet werden. Exemplarisch ist ein solcher Graph in Abbildung 26 dargestellt. Dabei sind die Kantengewichtungen aus Gründen der Lesbarkeit ausgelassen worden. Nur zwei Kanten in blau stellen die Beziehungen exemplarisch dar. J_0 beschreibt die Gelenkkonfiguration vor Bewegungsbeginn. Die Pose und die Konfiguration sind dabei unbekannt. Existiert für eine Kombination aus Pose und Konfiguration keine Gelenklösung entfällt der dazugehörige Knoten und Kanten. Gesucht wird nun der günstigste Weg von J_0 zur untersten Ebene.

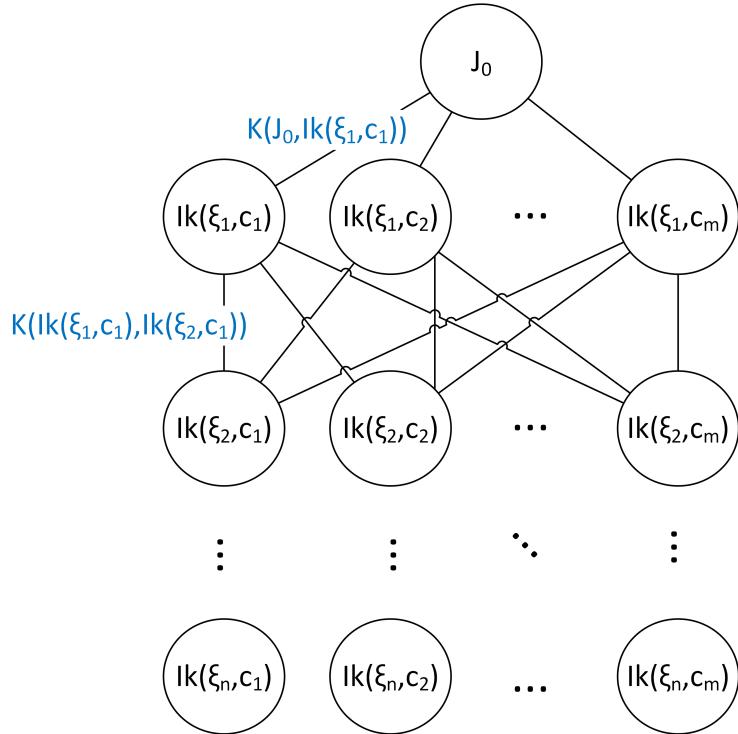


Abbildung 26: Der dargestellte Graph bildet die Planung des Bewegungspfad ab. Knoten stellen eine Gelenkeinstellung zu einer Pose und einer Konfiguration dar. Dabei werden Knoten mit der gleichen Konfiguration senkrecht und mit der gleichen Pose waagerecht angeordnet. Aufeinander folgende Posen sind untereinander sortiert. Die Kanten werden mit den Übergangskosten gewichtet. Zur Übersichtlichkeit wurden nur zwei Kanten exemplarisch in blau gekennzeichnet.

5.5.5 Nachtrag Literatur

Nach der Entwicklung und der Niederschrift über die inverse Kinematik wurde im Rahmen einer erweiterten Literaturrecherche das Paper Sharma et al. (2012) gesichtet, das eine ähnliche inverse Kinematik beschreibt. Diese geht auch von einer geometrischen Lösung aus. Der Unterschied bei der Bestimmung ist vor allem die Einbindung der mobilen Plattform in die inverse Kinematik. Dies ist für den stationären Arm unnötig und kann zwar auch aus der Gleichung entfernt werden. Dennoch wurde entschieden, dass die eigene inverse Kinematik genutzt werden soll. Dies ist vor allem durch die Wartbarkeit und Änderbarkeit, aber auch durch Interesse an der eigenen Arbeit begründet.

6 Implementierung

In diesem Kapitel werden einige Aspekte der Entwicklung aufgegriffen und im Detail ausgeführt. Der Name Implementierung bezieht sich nicht auf die Programmierung, sondern die Umsetzung von Schnittstellen und den Algorithmen hinter speziellen Problemen. So wird zum Beispiel in dem Unterkapitel 6.1 auf die Bewegung auf einer linearen Trajektorie eingegangen. Außerdem werden Probleme beschrieben und gelöst, die während der Umsetzung der Konzepte auftraten. Des Weiteren werden die genutzten Bibliotheken erwähnt und die Anbindung von ROS an die Konzepte. Den zentralen Aspekt in diesem Kapitel stellt jedoch Unterkapitel 6.6 dar. In diesem wird die Bestimmung für die Übergabeposition vorgestellt.

6.1 Robotersteuerung - RATSYoubot & RATSDummy & RATSRose

In diesem Kapitel wird die Ansteuerung für die Roboter und deren Arme vorgestellt. Diese beruht vor allem auf der Kuka API für den Youbot. Außerdem wird die Bewegung auf einer linearen Trajektorie, sowie die einzelnen Aktionen die ein Roboter ausführen kann kurz aufgelistet und beschrieben.

6.1.1 Ansteuerung der Aktoren

Die Aktoren innerhalb der Roboter werden mit Hilfe von ROS Topics angesteuert. Diese werden von dem Controllernode *youbot_driver* ausgelesen und in die Steuersignale für die Motoren umgewandelt. Für die Ansteuerung des Arms existieren zwei Topics. Der *velocity_command*-Topic erwartet für jedes Gelenk eine Geschwindigkeitsangabe. Für diese Arbeit wird jedoch mit Positionsangaben gearbeitet. Der dazu gehörige Topic *arm_controller/position_command* benötigt ein Array mit Gelenkkonfigurationen. Dafür wird die ROS-Bibliothek *BRICS* genutzt. Diese beinhaltet Nachrichten für inverse Kinematiken, wie zum Beispiel: Posen, Vektoren und Rotationen im Kartesischen Raum oder Aktoren Nachrichten, wie Gelenkeinstellung, Gelenkgeschwindigkeiten und Gelenkbeschleunigungen. Neben der Armansteuerung gibt es für die Gripper eine eigene Topic *gripper_controller/position_command*. Dieser transportiert ebenfalls Gelenkeinstellungen. Dabei kann die Position der einzelnen Finger des Grippers angegeben werden. Die Werte entsprechen dabei der Distanz der Finger zu ihrer Basisposition. Ein Nachteil der Ansteuerung über die Topics ist das fehlende Feedback. Der Absender erfährt nicht, ob das Signal angekommen ist und wann die Bewegung beendet ist. Sollen Bewegungen seriell ausgeführt werden muss der ausführende Thread blockiert werden, bis die Gelenke die Zielkonfiguration erreicht haben. Ein paralleler Thread liest die aktuelle Gelenkkonfiguration aus und berechnet eine Differenz für jedes einzelne Gelenk und die Quadratsumme über alle Gelenke. Solange die Differenz für mindestens ein Gelenk oder die Summe der Quadrate ihre Schwellwerte überschreiten blockiert ein Mutex den ausführenden Thread. Ein weiteres Problem während der Implementierung stellte die Ungenauigkeit der Ansteuerung dar. Dabei kam es großen Abweichungen an einem Gelenk, wenn es eine große Winkeländerung hatte. Dieses Überschlagen lässt sich auf die Massenträgheit des Armes und dem abrupten Abbremsen der Motoren zurückführen. Dies konnte

jedoch auch durch den parallelen Thread abgefangen werden. Dieser überprüft nun auch neben der Distanz zu der Zieleinstellung, die Distanz zur letzten gemessenen Gelenkeinstellung. Ist diese nahe null, also der Arm nicht mehr in Bewegung, und die Distanz zum Ziel zu groß wurde nochmal die selbe Gelenkeinstellung an den Motorencontroller geschickt.

Die Ansteuerung der mobilen Plattform wird mit einer einzelnen Topic *cmd_vel* umgesetzt. Dabei wird eine einzelne Nachricht übermittelt. Diese beinhaltet eine Twist Nachricht aus dem *geometry_msg* Paket. In dieser Nachricht kann eine lineare und eine Winkelgeschwindigkeit angegeben werden. Diese können für die einzelnen Achsen zerlegt werden. Für die mobile Plattform werden jedoch nur die X- und Y- Achsen der linearen, sowie die Z-Achse der Rotationsgeschwindigkeit ausgelesen und als absoluten Wert gesetzt. Der Controller rechnet anschließend die benötigten Steuersignale für die einzelnen Räder und Motoren aus.

6.1.2 Lineare Bahnbewegung

Die lineare Bahnbewegung wird beim Aufheben, Ablegen und während der Übergabe benötigt. Dies liegt vor allem an der Kollisionsvermeidung des Objektes mit der Umwelt. Da die inverse Kinematik die Gelenkeinstellungen und nicht die Gelenkgeschwindigkeiten berechnet ist es nicht so einfach auf einer gezielten Trajektorie entlang zufahren. Der Übergang zwischen zwei Gelenkeinstellung wird nicht genau definiert. Wie in Abbildung 27 dargestellt kommt es dabei durch unterschiedliche Gelenkgeschwindigkeiten zu einem Ausschlagen des Greifers. Je größer die Distanz zwischen zwei Posen, desto größer der Ausschlag. Deshalb kann dies verhindert beziehungsweise minimiert werden, indem Zwischenposen auf der Trajektorie ermittelt werden, die nacheinander angefahren werden. Dies erfordert zwar eine häufiges Aufrufen der inversen Kinematik, dafür kann eine Trajektorie mit einem geringen Ausschlag entlang gefahren werden. Tests mit dem YouBot Armen und der inversen Kinematik ergaben eine Distanz von 5 Millimetern zwischen den Zwischenposen als optimal. Bei einer kleineren Distanz reduzierte sich die Amplitude des Ausschlages weniger als vorher und steigert die Kosten durch einen häufigeren Aufruf der inversen Kinematik. Bei der Implementierung nutzt dafür vor allem die Orocus KDL Api, die auch bei der Implementierung der inversen Kinematik genutzt wird. Diese ermöglicht das Rechnen mit geometrischen Primitiven, so wurde der KDL::Frame für die Abbildung von Koordinatensystemen und Posen genutzt. Dieser ermöglicht unter anderem die Bestimmung der Zwischenposen.

6.1.3 Aktionen der Roboter

In diesem Unterkapitel wird sehr kurz auf die Aktionen eingegangen, die für die Roboter implementiert wurden. Aus Gründen des Umfangs wird dabei auf Grafiken, wie zum Beispiel UML-Diagramme, verzichtet. Aktionen die nur für einen Roboter implementiert wurden, sind speziell gekennzeichnet.

Greifer öffnen und schließen Diese Aktion sind eigentlich zwei. Für beide Roboter wurde eine Aktion für das Öffnen und eine für das Schließen implementiert. Beim Öffnen fahren die

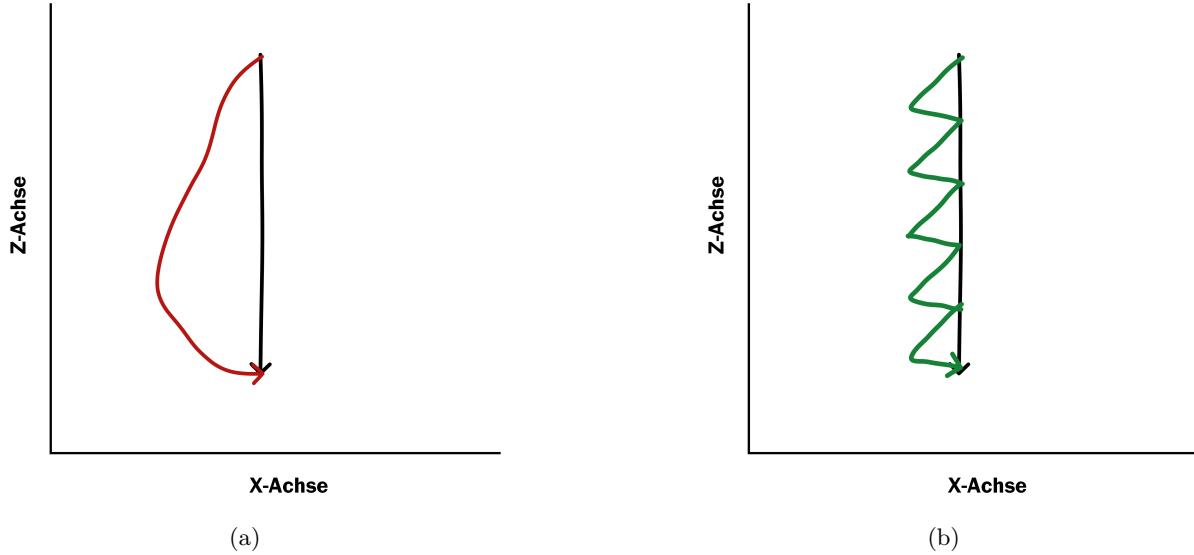


Abbildung 27: Die dargestellten Graphen stellen die gewünschte Trajektorie (schwarz) und die realen Trajektorien des Greifers dar. Dabei handelt es sich um eine lineare Absenkung auf der Z-Achse. Links: Trajektorie (rot) bei Angabe der Zielpose. Rechts: Trajektorie (grün) bei Angabe von Zwischenposen auf der gewünschten Trajektorie. Bildquelle: Müggler et al. (2013)

beiden Finger des Greifers auf ihre maximal Position. Beim Schließen fahren die beiden Finger auf die Nullstellung. Befindet sich ein Objekt im Greifer versuchen die Finger den Schluss. Da die Kraft des Greifers jedoch nicht reicht und die Finger nur auf Spindeln sitzen, befinden sich die Finger am Ende der Aktion nicht in der Nullstellung.

Greifer in Pose bringen Bei dieser Aktion wird als Parameter die sechsdimensionale Pose benötigt. Diese kann sich im globalen, als auch im lokalen Koordinatensystem des Roboters befinden und muss als solche mit einem Flag gekennzeichnet sein. Bei der Aktion wird zunächst die Gelenkeinstellung durch die inverse Kinematik ermittelt und anschließend mit der Methodik aus Kapitel 6.1.1 angesteuert. Zwei Sonderfälle dieser Aktion wurden für die Bewegung in die *Candle*- oder *Fold*-Pose implementiert.

Greifer an eine Pose annähern Mit der Annäherung ist in diesem Fall eine lineare Trajektorie entlang der Z-Achse des Greifers in der Endpose gemeint. Auch hier wird als Parameter die Zielpose im lokalen oder globalen Koordinatensystem benötigt. Aus dieser Pose wird ein Vektor bestimmt. Dieser läuft entlang der Z-Achse der Greifers und hat eine Länge von vier Zentimetern. Entsprechend der Entwicklung aus Kapitel 6.1.2 werden acht Posen im fünf Millimeter Abstand auf diesem Vektor bestimmt und nacheinander angefahren.

Objekt aufheben Diese Aktion ist eine Task, bei der seriell Aktionen ausgeführt werden. Als Parameter wird die Position des Zielobjektes benötigt, diese kann als lokaler Kontext angegeben werden oder mit einem Zeiger auf das Objekt im globalen Kontext. Neben der Position des Objektes wird die Rotation um die Z-Achse des Objektes berücksichtigt, da die X-Achse des

Greifers parallel zur X-Achse des Objektes liegen muss. Bei der Durchführung wird zunächst der Greifer in einem vier Zentimeter Abstand in Position gebracht. Anschließend wird Greifer geöffnet und an die Position des Objektes angenähert. Folgend wird der Greifer geschlossen und der Arm in die *Candle*-Pose gebracht. Für Rose gibt es eine erweiterte Form dieser Aktion, dabei wird Kontakt zur Naherkennung aufgenommen, um das Objekt visuell zu lokalisieren. Diese Koordinierung wird genutzt, um den Arbeitsraum des Roboters an die Position des Objektes anzupassen.

Objekt ablegen Ähnlich dem Aufheben ist auch diese Aktion eine Task. Im Gegensatz dazu werden die Aktionen in einer anderen Reihenfolge ausgeführt. So fährt der Greifer zunächst in eine Pose etwas über dem Ziel. Öffnet anschließend die Finger um sich abschließend von dem Objekt zu entfernen. Dies geschieht auf einer linearen Trajektorie um das Objekt nicht zu verschieben. Abschließend bewegt der Arm sich in die *Candle*-Pose.

Position einnehmen Diese Aktion betrifft nur Rose. Dabei fährt die mobile Plattform an eine neue Position. Dafür wird als Parameter eine X- und Y-Koordinate im globalen Koordinatensystem benötigt. Es werden keine rotierenden Bewegungen ausgeführt. Damit bleibt die Ausrichtung der Plattform gleich. Zur Lokalisierung wird dabei eine Mischung aus SLAM und Odometriedaten genutzt (siehe Kapitel 6.5).

Übergabe Die Übergabe ist die zentrale Aktion dieser Arbeit. Der Ablauf dieser ist bereits in Kapitel 5.2.6 und in den Abbildungen 19(a), 19(b) und 19(c) erläutert. Die Koordinierung dieser Aufgabe wird durch den RATSCore abgebildet. Die Positionsbestimmung kann durch eine direkte Koordinierung zwischen den Robotern stattfinden.

6.2 Raumüberwachung - RATSHawk

Die Raumüberwachung übernimmt die Detektion und Lokalisierung von Objekten im Raum. Dazu dient eine XTion-Kamera die an der Wand angebracht ist. Für das MRS wurde dafür ein eigener RATSMember, der RATSHawk, geschaffen. Momentan besitzt dieser Agent nur eine einzige Aktion: ein Objekt anhand seiner Farbe im Raum zu detektieren und die Position im globalen Kontext zu bestimmen. Als Parameter wird dafür ein Zeiger auf das gesuchte Objekt im globalen Kontext benötigt. Die Informationen über die Farbwerte des Objektes im HSV-Farbraum müssen im globalen Kontext unter dem Objekt abgelegt sein. Nach Ausführung der Aktion wird im Erfolgsfall die Position des Objektes in den globalen Kontext eingetragen. Die folgenden Kapitel befassen sich mit der verwendeten API *Point Cloud Library* und den Subsequenzen zur Detektion und Lokalisierung des Objektes.

6.2.1 PCL & ROS

Für die Anbindung der Kamerasysteme an ROS existieren die Pakete OpenNI2 und OpenNI2Launch. Diese generieren aus den Bildern der XTions Punktwolken, die mit Hilfe von Topics

empfangen werden können. Für die Weiterverarbeitung der Punktwolken wird in dieser Arbeit die Point Cloud Library (PCL) genutzt. Diese bietet unter anderem verschiedene Filter, Feature und Keypoint Detektion, sowie Segmentierungen und Erkennungen. Einziges Problem ist Konvertierung der einzelnen Punktwolken-Formate. Jedoch stellt die PCL-API Methoden zur Umwandlung bereit. Dabei muss jedoch die Art der Punktwolke, zum Beispiel chlorierte Punktwolke (XYZRGB) oder Intensität-Punktwolke (XYZI), berücksichtigt werden. Für die XTion Kamerae systeme können entweder XYZ-Punktwolken ohne Farbwerte oder XYZRGB-Punktwolken mit Farbwerten im RGB-Farbraum genutzt werden.

6.2.2 Detektion

Die Detektion in dieser Arbeit basiert auf den Farb- und Geometriemerkmale des Objektes. Dazu sind mehrere Schritte nötig. Nach der Konvertierung der Punktwolke P wird zunächst die Anzahl der Punkte p_i reduziert, indem alle Punkte die zum Boden P_B gehören aus der Punktwolke entfernt werden. Die dazu gehörigen Punkte werden mit Hilfe des RANSAC-Algorithmus ausgewählt. Dafür wird ein planares Modell angenommen. Damit nicht auch Punkte des auf dem Boden liegenden Objektes entfernt werden, wird der Schwellwert für die Distanz zwischen Modell und Punkt auf 5 Millimeter gesetzt. Die Anzahl der benötigten Iterationen ist durch die Anzahl der Punkte, die für das Modell benötigt werden, und dem Anteil der Punkte des Bodens gegeben. Diese Funktion wird von der PCL-API zur Verfügung gestellt. Anschließend wird die restliche Punktwolke nach den übergebenen Farbwerten im HSV-Farbraum gefiltert. Dabei werden alle Punkte die außerhalb der Schranken liegen aus der Punktwolke entfernt. Dadurch ergibt sich für die neue Punktwolke P_{Farb} :

$$\forall p_i \in (P - P_B) : ((h_{min} \leq p_i^h \leq h_{max}) \wedge (s_{min} \leq p_i^s \leq s_{max}) \wedge (v_{min} \leq p_i^v \leq v_{max})) \implies p_i \in P_{Farb} \quad (53)$$

Anschließend wird P_{Farb} in einzelne Punktwolken $P_{Obj}^i \in S_{Seq}$ segmentiert. Dies ist nötig, damit verschiedene Objekte mit der gleichen Farbe, unterschieden werden können. Dafür wird folgender Algorithmus verwendet. Zunächst wird eine leere Menge S_{Seq} definiert. Diese enthält alle Punktwolken-Segmente die durch den Algorithmus gefunden werden. Außerdem wird eine Liste L mit Punkten initialisiert. Für alle Punkte $p_i \in P_{Farb}$ werden folgende Schritte durchgeführt: p_i wird L hinzugefügt. Dabei werden alle Punkte p_j^i die in der Nachbarschaft P^i von p_i liegen gesucht. Die Nachbarschaftsbeziehung kann durch einen k-d-Baum angegeben werden. Wenn die Distanz zwischen p_j^i und p_i kleiner dem gegebenen Schwellwert d ist, wird der Punkt p_j^i der Liste L hinzugefügt und aus P_{Farb} entfernt. Anschließend wird der Schritt für alle neuen Punkte in L wiederholt. Ist in der letzten Wiederholung kein neuer Punkt hinzugekommen und ist $|L| > N$ wird L als neue Punktwolke S_{Seq} hinzugefügt und anschließend geleert. Dieser Vorgang wird wiederholt bis $|P_{Farb}| = 0$. Anschließend hat man eine Menge S_{Seq} an Punktwolken, deren

Punkte nah beieinander liegen und eine entsprechende Größe N haben:

$$\forall P_i \in S_{Seq} : |P_i| > N \quad (54)$$

Um nun ein Segment dem Objekt zuzuweisen wird die Anzahl der Punkte eines Segments mit der hinterlegten Anzahl an Punkten für das Objekt verglichen. Da die Anzahl der Punkte von der Entfernung zur Kamera und Lage des Objektes abhängig ist, gilt dieses nur als grobes Kriterium. Bei der Berechnung zur Lokalisierung werden deshalb noch weitere geometrischen Merkmale überprüft.

6.2.3 Lokalisierung

Nachdem die Punktwolke für das Objekt extrahiert wurde, wird nun die Position und Rotation des Objektes bestimmt. Dafür wird zunächst eine Oriented Bounding Box (OBB) für das Objekt bestimmt. Eine Oriented Bounding Box entspricht einer Axis Aligned Bounding Box (AABB) die an den Eigenvektoren der Punktwolken orientiert ist. Eine Bounding Box entspricht einem Quader, in dem alle Punkte der Punktwolke enthalten sind. Die AABB ist an den Achsen des Koordinatensystems ausgerichtet. Für die Bestimmung der einzelnen Merkmale existiert in der PCL-API bereits ein vorgefertigter Algorithmus. Mit Hilfe diesem lässt sich neben dem minimalen und maximalen Punkt der Punktwolke auch der Mittelpunkt und die Rotationsmatrix für die OOB bestimmen. An Hand der Daten der OOB wird nun nochmal ein Abgleich der Geometrischen Eigenschaften des Objektes gemacht, dabei wird das Seitenverhältnis mit den hinterlegten Daten verglichen. Stimmen diese überein, wird aus der Kamerapose im globalen Koordinatensystem und der Objektpose im Kamera-Koordinatensystem die Objektpose im globalen Koordinatensystem berechnet. Jedoch zeigte ein Vergleich zwischen der realen und der berechneten Position Differenzen. Eigene Tests zeigten, dass die Differenzen Positionen bezogen sind. Die Ursache davon liegt vermutlich bei einer Verzerrung der Kameralinse oder einem fehlerhaften Parameter beim Registrierung der RGB-Bilder auf das IR-Bild. Um diese Verzerrung zu korrigieren wird nun ein Ausgleich-Modell verwendet, dass auf den gemessenen Testdaten beruht. Dazu wird zunächst auf dem Mittelpunkt des Kamerabildes der Normalenvektor der Kameraebene gesetzt und der Schnittpunkt mit der planaren Ebene des Bodens bestimmt. Anschließend wird die Distanz in der X- und Y- Richtung der Bildebene zum Objekt bestimmt und je nach Richtung mit einem Korrekturfaktor multipliziert. Diese Korrektur erbringt eine Genauigkeitssteigerung, sodass die Positionsbestimmung eine Genauigkeit von fünf Zentimetern erreicht.

6.3 Nahfelderkennung - RATSEye

Im Gegensatz zur Raumüberwachung sitzt die Kamera nicht an der Wand, sondern am Roboter direkt. Dies betrifft in diesem Fall Rose. Die Nahfelderkennung ist als selbstständiger RATS-Member, dem RATSEye, umgesetzt. Genau wie der RATSHawk besitzt das RATSEye nur eine Aktion zur farbbasierten Detektion und Lokalisierung. Aufgrund des geringen Abstandes zum

Objekt hat dieser Agent gegenüber dem RATSHawk Vor- aber auch Nachteile. Auf der einen Hand ist die Genauigkeit der Ergebnisse auf dieser Distanz viel höher. Auch notwendige Korrekturen, wie beim RATSHawk, sind nicht nötig. Auf der anderen Hand benötigt der Agent die Vorarbeit eines anderen Agenten um arbeiten zu können. Da die Kamera am Arm von Rose angebracht ist, muss dieser erst in die *Candle*-Pose gebracht werden. dies erhöht den Koordinierungsaufwand. Ein weitere Nachteil ist die schlechte Hardware, die dem Agenten zur Verfügung steht. Dadurch, dass die Kamera an den eingebauten Rechner in der mobilen Plattform gebunden ist, dauern Auswertungen der Punktwolken länger. Außerdem ist durch die Belastung des Systems eine Verzögerung der Punktwolken gegeben. Da die Berechnung für das Hardwaresystem belastend sind wird von einer wiederholenden Ausführung abgesehen und die Aktion nur bei Bedarf ausgeführt. Dies und die Verzögerung kann dazu führen, dass die Daten nicht dem aktuellen Ausschnitt entspricht, an dem sich der Agent befindet. Um dies zu umgehen wurde ein paralleler Thread aufgebaut. Dieser empfängt die Punktwolken und legt die aktuellste in einem Speicher ab. Wird nun die Hauptfunktion ausgeführt blockiert diese ihren eigenen Thread mit einem Mutex. Der Datenthread hebt diese Sperre auf, sobald eine neues Punktwolke im Speicher liegt. Daraufhin blockiert der Hauptthread den Datenthread mit einem weiteren Mutex. Dies ist nötig, da während der Analyse neue Punktwolken eintreffen könnten und den Speicher überschreiben würden. Nach der Detektion und Lokalisierung wird diese Sperre wieder aufgehoben. Die Detektion und Lokalisierung laufen genauso ab, wie bei der Raumüberwachung. Nur die Korrektur der Lokalisierung ist nicht nötig.

Während der Testphase kam es bei der Nahfelderkennung zu zunächst unerklärlichen temporären Problemen. Dabei konnte das Objekt nicht detektiert werden. Dies lies sich auf zu kleine Punktwolken zurückführen. Eine Inspektion der aufgenommenen Bilder ergab, dass einzelne Bilder sehr dunkel waren. Bei einer genaueren Untersuchen stellte sich heraus, dass Reflexionen der Beleuchtung durch die spiegelnde Oberfläche der mobilen Plattform direkt in die Kamera blendeten. Diese führte einen automatischen Helligkeitsausgleich durch und verdunkelte das komplette Bild. Diese Problem konnte mechanisch umgangen werden, indem die spiegelnde Fläche mit schwarzem Klebeband verdeckt wurde.

6.4 Koordinierung

Die Koordinierung innerhalb dieses MRS ist über zwei Methoden möglich: die zentrale Koordinierung über den RATSCore oder die direkte Kommunikation zwischen zwei Agenten. Beide Varianten nutzen die gleiche Schnittstelle beim Empfänger, dadurch bleibt das System flexibel und erweiterbar. Diese Schnittstelle ist mit der ROS Action-Lib realisiert, wie in TODO beschrieben. Im Folgenden soll an Beispielen erklärt werden, wie die Koordinierung beim RATSCore und beim RATSMember implementiert wurden.

6.4.1 Koordinierung beim RATSMember

Jeder RATSMember instantiiert beim Start einen Action-Server der ROS Action-Lib. Diese Schnittstelle kann mit Hilfe eines Action-Clients angesprochen werden. Jeder RATSMember hat

deshalb auch einen Action-Client instantiiert, um Verbindungen zu anderen RATSMembern aufzunehmen. Ein Beispiel dafür ist die Naherkennung, die eine Koordination zwischen RATSEye und RATSRose benötigt. Folgende Aufgabe beschreibt das Aufheben eines Objektes im Raum: RATSRose bekommt die Aufgabe. Dazu wird zunächst die Position des Objektes im globalen Koordinatensystem aus dem Kontext gelesen. Anschließend fährt die mobile Plattform zum Objekt. Da sowohl die RATSHawk, als auch die mobile Plattform ungenau sind, ist eine Positions korrektur nötig. Dafür fordert der RATSRose bei RATSEye eine lokale Position des Objektes an. RATSEye wiederum fordert dafür zunächst von RATSRose an, den Arm in die Candle-Pose zu bringen, bevor eine Aufnahme gemacht werden kann. Mit dem Ergebnis der Lokalisierung kann RATSRose abschätzen, ob das Objekt innerhalb des Aktionsradius liegt. Ist dies nicht der Fall macht RATSRose eine Ausgleichsbewegung, die das Objekt in Reichweite des Greifers bringt. Anschließend wird wieder eine Lokalisierung von RATSEye angefordert. Dieser Vorgang geschieht sooft, bis das Objekt in Reichweite liegt. Diese Koordinierung ist eng-gekoppelt. Der Aktionspartner ist in diesem Fall statisch eingebunden.

6.4.2 Koordinierung beim RATSCore

Die Koordinierung mit dem RATSCore dient auch als Schnittstelle mit Personen, die das MRS bedienen möchten. Dabei können Actions und Tasks verschiedener Agenten miteinander verbunden werden. Diese müssen jedoch über die passenden Schnittstellen verfügen. Ein Beispiel dafür ist das Szenario aus der Einführung in Abbildung 2. Abgesehen vom Staubsaugerroboter ist dieses Szenario im Rahmen dieser Arbeit implementiert worden. Zur Koordinierung der einzelnen Agenten wurde eine Sequenz von Aktionen angelegt die mit der beschriebenen Schnittstelle aufgerufen wird. Als erstes werden die Informationen für das Objekt in dem globalen Kontext hinterlegt. Dazu gehören die Farb- und Geometrischen Eigenschaften des Objektes. Danach wird RATSHawk mit der Raumüberwachung beauftragt und das Ergebnis dem globalen Kontext hinzugefügt. Anschließend soll RATSRose das Objekt aufheben. Die Koordination dafür mit RATSEye übernimmt RATSRose, wie beschrieben, selbst. Nach Beendigung sollen RATS-Dummy und RATSRose das Übergabe durchführen. Auch hier wird die Koordinierung durch den RATSCore übernommen. Nur die Positionsfindung führen die beiden Roboter untereinander durch. Die zeitliche Koordinierung wird vom RATSCore durchgeführt. Diese Koordinierung kann auch zwischen den Robotern durchgeführt werden, wurde jedoch für diese Arbeit exemplarisch über den zentralen Knoten umgesetzt.

6.5 Bewegung & Lokalisierung der mobilen Plattform

Die Lokalisierung für RATSRose setzt sich zusammen aus einem vereinfachten visuellen Verfahren und der Auswertung der Odometriedaten. Die grobe Lokalisierung findet dabei mit der Odometrie statt. Dazu ist in RATSRose die eigene Position im globalen Koordinatensystem hinterlegt. Bewegt sich RATSRose wird dafür der Differenzvektor r zwischen der aktuellen Position und der Zielposition ermittelt. Anschließend wird der Geschwindigkeitsvektor v ermittelt. Dieser ergibt sich aus einer Fallunterscheidung anhand der Distanz pro Dimension. Dies ist nötig, da

die Geschwindigkeit auf Grund der Genauigkeit gering gehalten werden soll, aber auch nicht zu gering, da die minimale Geschwindigkeit der Plattform TODO Meter pro Sekunde beträgt. Anschließend wird aus r und v die Zeit t ermittelt. Daraufhin werden die Steuersignale an die mobile Plattform gesendet. Nach Ablauf der Zeit t wird die Plattform gestoppt. Danach erfolgt eine Aktualisierung der aktuellen Position. Zunächst wird die Zielposition als neue Position angenommen. Diese wird anschließend durch die Sensordaten aus der Lasermessung aktualisiert. Aus den Messdaten wird eine zweidimensionale Punktwolke erzeugt. Der Ansatz, der in dieser Arbeit verwendet wird, basiert aus dem Vorwissen über die grobe Position, Rotation und der Umwelt. Das Ziel ist es aus dieser Punktwolke die zwei Wände des Labores zu identifizieren, da der Schnittpunkt dieser den Ursprung des globalen Koordinatensystems bildet. Dafür wird, wie auch bei der Objektlokalisierung, der RANSAC-Algorithmus angewendet. Diesmal jedoch nicht mit einem planaren Modell, sondern einem Geraden-Modell. Die PCL-API bietet dafür eine Schnittstelle. Diese liefert die Koeffizienten der Gerade in einer Zwei-Punktform zurück. Für die folgende Analyse wird jedoch die hessesche Normalenform benötigt, diese gibt den Normalenvektor der Geraden und den Abstand der Geraden zum Ursprung an. Der Normalenvektor wird dazu genutzt die Gerade der entsprechenden Wand zuzuweisen. Dafür wird die bekannte Z-Rotation der Plattform genutzt. Diese Unterscheidung ist wichtig, damit der Abstand der Geraden der richtigen Positions-Komponente zugewiesen werden kann. Dabei entspricht der Abstand zur Ost-Wand der X-, und der Abstand zur Nord-Wand der Y-Komponente.

Ein Problem bei dieser Methodik ist die Nord-Wand mit den Fenstereinlässen. Bei diesen kann es passieren, dass die gefunden Geraden nicht die Wand, sondern die Fenstertiefe detektieren. Um dies zu vermeiden wird ein Abgleich mit der groben Position, die durch die Odometrie erfasst wird, abzuleiten. Dabei kann es immer noch zu starken Abweichungen führen, da die Position aus der Odometrie um fünf Zentimeter abweicht. Dadurch kann der Abgleich auch nicht zwischen Wand und Fensterreihe unterscheiden.

6.6 Handoverpoint

Der Übergabepunkt zwischen den beiden Robotern ist eine der zentralen Komponenten dieser Arbeit. Er ist jedoch nicht komplex und kurz zu erklären. Ziel dieser Komponente ist es einen Pose zu finden, die beide Roboter einnehmen können, während die Greifer gespiegelt orientiert sind. Da die Arme der Roboter auf fünf Freiheitsgrade beschränkt sind, kann kein seitlicher Griff die Zielbedingung erfüllen. Dadurch kann die Übergabe nur in einer Ebene stattfinden. Diese Ebene wird durch die globale Z-Achse und einem Vektor, der zwischen den beiden Basen der Arme liegt, definiert. In der Parameterform notiert mit \vec{p}_1 und \vec{p}_2 für die Stützvektoren der Basispunkte der Arme:

$$\vec{e} = \vec{p}_1 + s(\vec{p}_2 - \vec{p}_1) + t * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ mits, } t \in \mathbb{R} \quad (55)$$

Der optimale Punkt für beide Roboter ist jeweils von ihrer vorherigen Gelenkkonfiguration

abhängig. Da jedoch, aus Sicherheitsgründen, beide Arme in der Candle-Pose stehen, liegt die erste Annahme nah, dass der optimale Punkt auf der Geraden zwischen den beiden Greifern liegt. Jedoch ist die Candle-Pose die obere Schranke für die Z-Achse von Rose. Da Dummy einen höheren Basispunkt als Rose hat, muss diese immer nach oben greifen. Auch eine gerade zwischen dem Greifer von Rose und der Basis von Dummy kann zu Punkten führen die immer außerhalb des Arbeitsraumes von Rose liegen. Dadurch ergibt sich, dass die Gerade g auf der die Übergabeposition p_h liegt, durch die beiden Basen von Rose und Dummy definiert ist:

$$g = \vec{p_1} + s(\vec{p_2} - \vec{p_1}) \quad (56)$$

Die Orientierung der Greifer ist dabei durch die Gerade vorgegeben. So muss die Z-Achse der Greifer deckungsgleich mit g sein. Die Rotation um die Z-Achse ist dabei beliebig, da beide Greifer alle Rotationen erreichen können. Wo der Punkt auf der Geraden liegen muss, verhandeln die beiden Roboter untereinander. Dabei geht der Vorschlag von dem fixierten Roboter (Dummy) aus und kann von dem mobilen angenommen werden. Dummy wählt dabei einen für ihn möglichst weit entfernten Punkt aus, da die Armbewegungen genauer sind, als Bewegungen mit der Plattform. Kann Rose diesen Punkt nicht mit einer Armbewegung erreichen, hat sie die Möglichkeit die Punktbestimmung zu verwerfen und sich Dummy zu nähern. Dieser Korrekturvorgang wird sooft wiederholt, bis ein Übergabepunkt ermittelt wurde. Die dafür notwendige Bewegung wird entlang der Geraden g ausgeführt, wobei nur die XY-Komponenten genutzt werden. Da Rose aber auch nicht zu nah an Dummy stehen darf, wird die Distanz zwischen den Robotern halbiert. Unterschreitet die Distanz in der XY-Ebene einen Schwellwert entfernt sich Rose wieder um zehn Prozent der Distanz. Da die Bewegungen von Rose, auf Grund der Genauigkeit, eher entlang ihrer X-Achse sind, verschiebt sich ihr Basispunkt zufällig. In einer ersten Implementierung wurde dieser Zufallsfaktor durch eine Lokalisierung solange ausgeglichen, bis Rose an der gewünschten Position stand. Jedoch ergab dies eine hohe Zahl an Korrekturen. Außerdem konnte es passieren, dass die Korrektur zwischen Punkten außerhalb von Dummys Arbeitsraum pendelte. Deshalb wird bei der finalen Entwicklung diese Abweichungen genutzt, um eine mögliche Übergabeposition zu finden. Dabei wird die neue zufällige Position als gewollt Zielposition angenommen.

7 Test und Bewertung

Dieses Kapitel umfasst Testreihen zur Bewertung des MRS. Diese Testreihen folgen dem *Top-Down* Prinzip und sind nicht als vollwertige Softwaretests zu sehen, da unter anderem auch das Zusammenspiel der Konzepte und der Hardware getestet wird. Zunächst wird das Gesamtsystem getestet, da die einzelnen Algorithmen und Konzepte zum Teil schon in ihrer Entwicklungsphase getestet wurden. Kommt es zu Fehlschlägen des Gesamtsystems, wird der Fehlschlag analysiert und die betreffenden Subsysteme getestet. Diese Tests sind vor allem als funktionsorientierter Test zusehen. Es werden also keine Daten- oder Kontrollflüsse getestet, sondern der Fokus auf Positiv- und Negativtests gelegt. Auch die Ursache-Wirkung-Analyse ist für die Subsysteme angewendet.

Die Tests für das Gesamtsystem werden mit Hilfe von funktionalen Äquivalenzklassen durchgeführt. Dazu wurde folgendes Szenario 24 mal ausgeführt: Das Zielobjekt liegt an einer Position im Raum. Rose befindet sich in einer definierten Startposition. Das System bekommt die Aufgabe das Objekt aufzuheben und an einer definierten Stelle auf dem Tisch abzulegen. Die Position des Objektes im Raum variiert bei den Testfällen und ist in Äquivalenzklassen unterteilt. Diese sind an den notwendigen Bewegungen der Plattform orientiert. Die Äquivalenzklassen setzen sich aus der Achse und der Strecke der Bewegung zusammen: *X-Kurz*, *X-Lang*, *X-Kurz*, *Y-Kurz*, *X-Lang*, *Y-Kurz*, *X-Kurz*, *Y-Lang*, *X-Lang*, *Y-Lang*. Wird eine Achse nicht erwähnt findet keine Bewegung entlang dieser statt. Kurz steht für eine Bewegungsdistanz kleiner 50 Zentimeter entlang der angegeben Achse. Die Tabelle 4 bildet alle Testfälle mit allen Messergebnissen, die während dieses Szenarios erzielt wurden, ab. Dazu zählen die Genauigkeit der Raumüberwachung γ und der Differenz zwischen Odometriedaten und Laserdaten η in Zentimetern, die Anzahl der Korrekturversuche k der mobilen Plattform, die Anzahl der Verhandlungsversuche j für den Übergabepunkt und die Genauigkeit bei der Übergabe ι in Zentimetern. Die letzte Spalte bildet den Erfolg des Testfalls oder den auftretenden Fehler ab. Dabei traten drei Fehlerwirkungen auf: *Fehler 1* steht für einen Abbruch bei der Aufnahme des Objektes, zum Beispiel durch eine Überschreitung der Versuchsschranke, zehn Versuche, oder einem Fehlgriff. *Fehler 2* bildet den Abbruch während der Bestimmung der Übergabeposition ab. Dies kann nur durch ein überschreiten der Verhandlungsschranke, maximal zehn Verhandlungen, ausgelöst werden. *Fehler 3* tritt bei einer falschen Übergabe auf. Die Ursache der Fehler wird im folgenden analysiert.

Testlauf	Äquivalenzklasse	γ	η	k	j	ι	Erfolg oder Fehler
1	X-Kurz	$\begin{pmatrix} 2, 2 \\ 1, 6 \end{pmatrix}$	$\begin{pmatrix} 3, 2 \\ -0, 4 \end{pmatrix}$	0	2	<1cm	✓
2	X-Kurz	$\begin{pmatrix} -2, 1 \\ 1, 3 \end{pmatrix}$	$\begin{pmatrix} 4, 1 \\ -0, 3 \end{pmatrix}$	0	4	<1cm	✓
3	X-Kurz	$\begin{pmatrix} 1, 4 \\ 2, 2 \end{pmatrix}$	$\begin{pmatrix} 3, 5 \\ -0, 5 \end{pmatrix}$	0	2	<1cm	✓
4	X-Kurz	$\begin{pmatrix} 2, 3 \\ -2, 1 \end{pmatrix}$	$\begin{pmatrix} 3, 7 \\ -0, 3 \end{pmatrix}$	0	2	<1cm	✓

5	X-Lang	$\begin{pmatrix} -1, 6 \\ 1, 8 \end{pmatrix}$	$\begin{pmatrix} 5, 2 \\ -0, 7 \end{pmatrix}$	1	2	<1cm	✓
6	X-Lang	$\begin{pmatrix} 1, 4 \\ 1, 4 \end{pmatrix}$	$\begin{pmatrix} 6, 1 \\ -0, 8 \end{pmatrix}$	1	3	<1cm	✓
7	X-Lang	$\begin{pmatrix} 1, 3 \\ -2, 2 \end{pmatrix}$	$\begin{pmatrix} 7, 2 \\ -0, 8 \end{pmatrix}$	0	2	<1cm	✓
8	X-Lang	$\begin{pmatrix} 5, 4 \\ 1, 3 \end{pmatrix}$	$\begin{pmatrix} 6, 8 \\ -0, 5 \end{pmatrix}$	1	2	<1cm	✓
9	X-Kurz, Y-Kurz	$\begin{pmatrix} 2, 1 \\ -1, 5 \end{pmatrix}$	$\begin{pmatrix} 10,4 \\ -4,8 \end{pmatrix}$	>10	-	-	Fehler 1
10	X-Kurz, Y-Kurz	$\begin{pmatrix} 1, 2 \\ 3, 8 \end{pmatrix}$	$\begin{pmatrix} 4, 2 \\ -7, 4 \end{pmatrix}$	1	3	<1cm	✓
11	X-Kurz, Y-Kurz	$\begin{pmatrix} -2, 4 \\ 4, 2 \end{pmatrix}$	$\begin{pmatrix} 3, 7 \\ -6, 3 \end{pmatrix}$	2	4	<1cm	✓
12	X-Kurz, Y-Kurz	$\begin{pmatrix} -3, 2 \\ -2, 1 \end{pmatrix}$	$\begin{pmatrix} 6, 1 \\ -4, 2 \end{pmatrix}$	1	2	3,2	Fehler 3
13	X-Lang, Y-Kurz	$\begin{pmatrix} -4, 1 \\ -2, 8 \end{pmatrix}$	$\begin{pmatrix} 3, 8 \\ -7, 2 \end{pmatrix}$	1	3	4,2	Fehler 3
14	X-Lang, Y-Kurz	$\begin{pmatrix} 7, 2 \\ 1, 2 \end{pmatrix}$	$\begin{pmatrix} 9, 4 \\ -4, 8 \end{pmatrix}$	2	4	4,5	Fehler 3
15	X-Lang, Y-Kurz	$\begin{pmatrix} -1, 4 \\ 8, 1 \end{pmatrix}$	$\begin{pmatrix} 6, 2 \\ -5, 5 \end{pmatrix}$	2	3	<1cm	✓
16	X-Lang, Y-Kurz	$\begin{pmatrix} 2, 4 \\ 1, 3 \end{pmatrix}$	$\begin{pmatrix} -1, 2 \\ -4, 2 \end{pmatrix}$	2	3	<1cm	✓
17	X-Kurz, Y-Lang	$\begin{pmatrix} 1, 2 \\ -3, 4 \end{pmatrix}$	$\begin{pmatrix} 5, 4 \\ -22, 8 \end{pmatrix}$	>10	-	-	Fehler 1
18	X-Kurz, Y-Lang	$\begin{pmatrix} 3, 8 \\ 2, 1 \end{pmatrix}$	$\begin{pmatrix} 3, 2 \\ -17, 2 \end{pmatrix}$	2	>10	-	Fehler 2
19	X-Kurz, Y-Lang	$\begin{pmatrix} 2, 5 \\ 2, 4 \end{pmatrix}$	$\begin{pmatrix} 1, 6 \\ -20, 1 \end{pmatrix}$	1	7	<1cm	✓
20	X-Kurz, Y-Lang	$\begin{pmatrix} 1, 6 \\ 2, 7 \end{pmatrix}$	$\begin{pmatrix} 4, 2 \\ -19, 8 \end{pmatrix}$	2	8	4,2cm	Fehler 3
21	X-Lang, Y-Lang	$\begin{pmatrix} 3, 2 \\ -1, 7 \end{pmatrix}$	$\begin{pmatrix} 3, 8 \\ -21, 2 \end{pmatrix}$	1	6	5,6cm	Fehler 3
22	X-Lang, Y-Lang	$\begin{pmatrix} 2, 4 \\ 2, 1 \end{pmatrix}$	$\begin{pmatrix} 2, 8 \\ -18, 4 \end{pmatrix}$	2	5	<1cm	✓
23	X-Lang, Y-Lang	$\begin{pmatrix} -3, 3 \\ 1, 6 \end{pmatrix}$	$\begin{pmatrix} 2, 8 \\ -31, 4 \end{pmatrix}$	>10	-	-	Fehler 1
24	X-Lang, Y-Lang	$\begin{pmatrix} 2, 7 \\ 1, 8 \end{pmatrix}$	$\begin{pmatrix} 3, 1 \\ -16, 1 \end{pmatrix}$	2	8	6,2	Fehler 3

Tabelle 4: Messwerte der Testphase

Zusammengefasst ergibt sich, dass das Robotersystem eine Erfolgsquote von ca. 60 Prozent hat. Dabei ist auffällig, dass die Fehler sich bei einer großen Distanz auf der Y-Achse häufen. Um die Fehlerquelle genauer identifizieren zu können werden zunächst die einzelnen Aktionen aufgelistet. Anschließend wird eine Vorauswahl getroffen, welche Aktionen für den Fehler verantwortlich sein können. Anschließend werden diese Aktionen genauer untersucht. Fehler 1 gehen fünf Aktionen voraus: Objekt identifizieren und lokalisieren, Rose an die Zielposition bewegen, Naherkennung des Objektes, Korrekturbewegung und Objekt aufheben. Diese können alle potenzielle Fehlerquellen sein. Jedoch lassen sich einzelne Aktionen durch Subsystemtests schon ausschließen. So ist die Raumüberwachung mit einer Genauigkeit von fünf Zentimetern entwickelt worden. Die Werte aus der Tabelle ergeben, dass diese Genauigkeit auch eingehalten wurden. Ausnahmen bilden die Testläufe 14 und 15. Zwischen Fehler 1 und 2 liegt nur die Bestimmung der Übergabeposition, sowie die Korrektur der Position von Rose. Dies betrifft also die Lokalisierung und Positionierung der mobilen Plattform, sowie das Konzept der Bestimmung der Korrektur. Zu Fehler 3 führen die Positionierung und die inverse Kinematik. Im Folgenden werden diese einzelnen Aktionen getestet und mögliche Folgefehler analysiert.

7.1 Genauigkeit Inverse Kinematik

Für die Tests der inversen Kinematik wurden 25 Posen im lokalen Koordinatensystem von Dummy definiert. Diese wurden nacheinander vom Greifer von Dummy angefahren. Zwischen den einzelnen Posen wurde die *Candle*-Pose eingenommen. Bei allen Posen wurde die Distanz zwischen Greiferpose und Zielpose aufgenommen. Die Greiferposen wurden mit der RViz-Anwendung von ROS gemessen. Diese gibt Position und Rotation des Greifers aus. Anschließend wurden die Differenzen der einzelnen Posen berechnet und der arithmetische Mittel-, der Min- und der Max-Wert festgestellt und in eine CSV-Datei abgelegt. Die Ergebnisse sind Anhang B zu entnehmen. Entscheidend sind die maximalen Abweichungen, diese betragen linear 3,78 Millimeter und 0,66 ° für den rotierenden Anteil. Diese Ergebnisse der inversen Kinematik sind für diese Arbeit sehr gut und erfüllen die selbst gesetzten Anforderungen. Folglich ist die inverse Kinematik nicht die Ursache für die obigen Fehler.

Ein weiterer Bestandteil ist der Bewegungsplaner für die linearen Trajektorien. Bei diesem werden Zwischenpose berechnet. Da diese jedoch durch Matrixmultiplikationen berechnet werden, ist die Wahrscheinlichkeit für einen Fehler so gering, dass die genutzte KDL-Api nicht zusätzlich getestet wurde.

7.2 Test Übergabeposition

Die Übergabeposition hat zwei mögliche Fehlerquellen: das Konzept der Verhandlung oder der Korrekturbewegung. Auffällig ist Testlauf 18, der als einziger in der Übergabephase abgebrochen hat. Bei diesem konnte man beobachten, dass die mobile Plattform die Korrekturbewegungen nur auf ihrer lokalen X-Achse durchführte. Dabei war die Distanz zwischen den Robotern so gering, dass die Posen der Greifer nicht zu einander passten. Dies führte zu wiederholenden Verhandlungsabbrüchen. Die anschließende Korrektur führte dabei auf die Position vor der letzten Korrektur zurück. Dieses Pendeln zwischen den beiden Position wurde schon in Kapitel 6.6 erwähnt und sollte eigentlich durch die Abweichung der Bewegung beseitigt sein. Die Abweichungen auf der lokalen X-Achse der mobilen Plattform ist jedoch zu gering. Dies führte bei Testlauf 18 zum Fehler. Ein Lösungsansatz dafür ist eine bessere Korrekturbewegung, die dynamisch die Korrekturweite bestimmt.

Für die Testläufe mit Fehler 3 ist eine Ungenauigkeit der Position festzustellen. Diese ist auf Schleppfehler durch die Korrekturen zurückzuführen und von der Navigation der mobilen Plattform abhängig.

7.3 Genauigkeit Navigation

Die Genauigkeit der Odometrie wurde schon während der Entwicklung getestet und in Kapitel 5.3.3 analysiert. Dabei wurden starke Abweichungen und Varianzen bei Bewegungen entlang der Y-Achse der mobilen Plattform festgestellt. Deshalb wurde die Kopplung an die Sensordaten entwickelt. Wie Tabelle 4 zeigt sind die Abweichungen zwischen Odometrie o und Laserdaten l gerade bei den genannten Bewegungen sehr hoch. Wobei die o entlang der Y-Achse immer kleiner und entlang der X-Achse größer als l sind. Neben l und o wurde bei den fehlgeschlagenen Testläufen auch die Differenz zur realen Position p im globalen Koordinatensystem genommen. Auffällig dabei ist Testlauf 9. Dabei ist $\|p - o\| < \|p - l\|$. Dies lässt sich mit dem Einwand aus Kapitel 6.5 erklären, dass die Auswertung der Laserdaten die Fensterreihe und nicht die Wand detektiert hat. Ansonsten gilt $\|p - o\| > \|p - l\|$. Aber auch die Lokalisierung mit Hilfe der Laserdaten hat Abweichungen von mehreren Zentimetern. Diese addiert mit der geringen Ungenauigkeit der Kamera führt dazu, dass Objekte komplett verfehlt werden. Auch die nötigen Korrekturen erzeugen auf Grund der schlechten Odometriedaten Schleppfehler.

7.4 Zusammenfassung der Tests

Durch die Ungenauigkeit der Navigation der mobilen ergeben sich Schleppfehler, die zu großen Ungenauigkeiten führen. Diese sind vor allem bei Bewegungen der lokalen Y-Achse der Plattform merkbar. Auch die eingesetzten Lasersensoren bringen bei diesem Konzept der Steuerung keinen Vorteil. Eine Alternative dazu wird in Kapitel 8 vorgestellt. Die andere Fehlerquelle ist das Korrekturkonzept bei der Übergabe. Bei einer Überarbeitung der Navigation kann die Steigerung der Genauigkeit für ein einfacheres Korrekturkonzept genutzt werden. Dieses beruht auf einer inversen Kinematik, welche die Freiheitsgerade der mobilen Plattform berücksichtigt.

7.5 Robustheit

Dieser Test fällt im Vergleich zu den anderen Tests aus der Reihe. Hierbei geht es um das Verhalten im Fehlerfall. Dies soll das RATS-Konzept auf die Nicht-Funktionalen Anforderung der Zuverlässigkeit prüfen. Dazu werden Fehler im System provoziert. Dabei werden folgenden Testszenarien durchgeführt:

1. beliebigen RATSMember beenden und neu starten
2. RATSMember während der Koordinierung beenden
3. RATSCore zurücksetzen

Das gewünschte Ergebnis für alle Testfälle ist ein Abbruch der aktuellen Aktion und der Übergang in einen sicheren Zustand, dies betrifft vor allem die Roboter. Diese stellen ansonsten ein Sicherheitsrisiko dar. Jedes Szenario wird zehnmal getestet, wobei beim ersten und zweiten die RATSMember variiert werden. Jeder dieser Negativ-Tests endet Positiv. Alle Subsysteme erfüllen die Anforderungen. Dabei bewegen die Roboter ihre Arme nach Inaktivität in die *Fold*-Pose und sind damit in einem mechanisch sicheren Zustand. Beim dritten Szenario beendet der aktuell arbeitende RATSMember zuerst seine Aktion, da keine Kommunikation zwischen RATSMember und RATSCore stattfindet. Im Anschluss folgt der Übergang in den sicheren Zustand.

Im Verlauf dieses Tests kann auch die Wiederherstellbarkeit des Systems getestet werden. Dazu werden nach der Deaktivierung einzelner Knoten diese neu gestartet und die Aktion neu angestoßen. Dabei wird der Aufwand des Neustarts gemessen. Als Kennzahl dient dafür die Anzahl der Eingriffe durch den Anwender. Dabei stellt sich heraus, das ein Neustart des RATSCores am Aufwendigsten ist, da alle RATSMember neu gestartet werden müssen, damit sie sich neu registrieren können. Die Anzahl der Eingriffe bei den anderen Szenarien ist konstant bei zwei. Zum einen muss der RATSMember neu gestartet werden, zum anderen muss die Aktion neu angestoßen werden.

8 Zusammenfassung und Ausblick

Hier eine Zusammenfassung und der Ausblick

9 Danksagung

Ich bin allen Leuten furchtbar dankbar, vor allem den Jungs bei Google.

10 Literaturverzeichnis

Literatur

- Adjie-Winoto, W., E. Schwartz, H. Balakrishnan, and J. Lilley (1999). The design and implementation of an intentional naming system. In *ACM SIGOPS Operating Systems Review*, Volume 33, pp. 186–201. ACM.
- Arkin, R. C. (1987). Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, Volume 4, pp. 264–271. IEEE.
- Arregui, D., C. Fernstrom, F. Pacull, and J. Gilbert (2003). Stitch: Middleware for ubiquitous applications. In *Proc. of the Smart Object Conf.*
- Asada, H. (1979). Studies on prehension and handling by robot hands with elastic fingers. *PhD dissertation Kyoto University*.
- Asus (2015, dec). Xtion pro live.
- Badura, B. (2005). *Fehlzeiten-Report 2004 Gesundheitsmanagement in Krankenhäusern und Pflegeeinrichtungen*. Badura, Prof. Dr. Bernhard Schellschmidt, Dr. Henner Vetter, Christian. ISBN 978-3-540-27051-5.
- Balch, T. (1999). The impact of diversity on performance in multi-robot foraging. In *Proceedings of the third annual conference on Autonomous Agents*, pp. 92–99. ACM.
- Basili, P., M. Huber, O. Kourakos, S. Hirche, T. Brandt, and S. Glasauer. Approach and handover for human-robot interaction.
- Bicchi, A. (1994). On the problem of decomposing grasp and manipulation forces in multiple whole-limb manipulation. *Robotics and Autonomous Systems 13*(2), 127–147.
- Bicchi, A. and V. Kumar (2000). Robotic grasping and contact: A review. In *ICRA*, pp. 348–353. Citeseer.
- Bicchi, A., J. Salisbury, and D. Brock (1993). Experimental evaluation of friction data with an articulated hand and intrinsic contact sensors. *Experimental robotics: III (eds Chatila R., Hirzinger G.)*, Berlin/Heidelberg, Germany: Springer.
- Bluetchnix (2015, dec). Depth sensing - bluetchnix.
- Boella, G. (2002). Norms and cooperation: Two sides of social rationality.
- Botelho, S. C. and R. Alami (1999). M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, Volume 2, pp. 1234–1239. IEEE.
- Bowers, D. L. and R. Lumia (2003). Manipulation of unmodeled objects using intelligent grasping schemes. *Fuzzy Systems, IEEE Transactions on 11*(3), 320–330.

- Broxvall, M. and A. Saffiotti (2005). Peis ecologies: Ambient intelligence meets autonomous robotics. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, sOc-EUSAI '05, New York, NY, USA, pp. 277–281. ACM.
- Caceres, D. A., H. Martinez, M. Zamora, and L. Tomás (2003). A real-time framework for robotics software. In *Int Conf on Computer Integrated Manufacturing*.
- Caloud, P., W. Choi, J.-C. Latombe, C. Le Pape, and M. Yim (1990). Indoor automation with many mobile robots. In *Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on*, pp. 67–72. IEEE.
- Coelho, J., J. Piater, and R. Grupen (2001). Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot. *Robotics and Autonomous Systems* 37(2), 195–218.
- Corke, P. (2011). *Robotics, Vision and Control - Fundamental Algorithms in MATLAB* (1st ed. 2011 ed.). Berlin Heidelberg: Springer Science & Business Media.
- Davis, R. and R. G. Smith (2003). Negotiation as a metaphor for distributed problem solving. In *Communication in Multiagent Systems*, pp. 51–97. Springer.
- Dias, M. B. and A. Stentz (2000). A market approach to multirobot coordination. Technical report, DTIC Document.
- Dias, M. B. and A. Stentz (2003). A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Volume 3, pp. 2279–2284. IEEE.
- Durfee, E. H., V. R. Lesser, and D. D. Corkill (1987). Coherent cooperation among communicating problem solvers. *Computers, IEEE Transactions on* 100(11), 1275–1291.
- Farinelli, A., L. Iocchi, and D. Nardi (2004). Multirobot systems: a classification focused on coordination. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 34(5), 2015–2028.
- Florek-Jasinska, M. (2015, dec). Youbot detailed specifications.
- Frias-Martinez, V., E. Sklar, and S. Parsons (2005). Exploring auction mechanisms for role assignment in teams of autonomous robots. In *RoboCup 2004: Robot Soccer World Cup VIII*, pp. 532–539. Springer.
- Gale, D. (1989). *The theory of linear economic models*. University of Chicago press.
- Gerkey, B. P. and M. J. Matari (2002). Sold!: Auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on* 18(5), 758–768.
- Gerkey, B. P. and M. J. Matarić (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9), 939–954.

- Hayes, A. T. and P. Dormiani-Tabatabaei (2002). Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, Volume 4, pp. 3900–3905. IEEE.
- Heidemann, J., F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan (2001). Building efficient wireless sensor networks with low-level naming. In *ACM SIGOPS Operating Systems Review*, Volume 35, pp. 146–159. ACM.
- Huber, M., M. Rickert, A. Knoll, T. Brandt, and S. Glasauer (2008). Human-robot interaction in handing-over tasks. In *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, pp. 107–112. IEEE.
- Kalra, N., D. Ferguson, and A. Stentz (2005). Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1170–1177. IEEE.
- Kalra, N., T. Stentz, and D. Ferguson (2004). Hoplites: A market framework for complex tight coordination in multi-agent teams. Technical report, DTIC Document.
- Kamon, I., T. Flash, and S. Edelman (1996). Learning to grasp using visual information. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, Volume 3, pp. 2470–2476. IEEE.
- Kobayashi, H. (1999). *Technolife Series. Robotto wa tomodachi da! [Technolife-Serie. Der Roboter ist ein Freund!]*. Tokyo: Ohmsha.
- Kuka (2015, dec). Mounting and sensor plate.
- Kulić, D. and E. A. Croft (2005). Safe planning for human-robot interaction. *Journal of Robotic Systems* 22(7), 383–396.
- Lau, H. C. and L. Zhang (2003). Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pp. 346–350. IEEE.
- Lundh, R., L. Karlsson, and A. Saffiotti (2005). Can emil help pippi. In *Proc. of the ICRA-05 Workshop on Cooperative Robotics*.
- Lundh, R., L. Karlsson, and A. Saffiotti (2006). Plan-based configuration of a group of robots.
- MAEDA (2009, aug). *Scanning Laser Range Finder URG-04LX-UG01 (Simple-URG) Specifications*. Hokuyo.
- Mainprice, J., E. A. Sisbot, T. Siméon, and R. Alami (2010). Planning safe and legible hand-over motions for human-robot interaction. In *IARP workshop on technical challenges for dependable robots in human environments*, Volume 2, pp. 7.
- Maitin-Shepard, J., M. Cusumano-Towner, J. Lei, and P. Abbeel (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In

Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 2308–2315. IEEE.

Mason, M. T. and J. K. Salisbury Jr (1985). Robot hands and the mechanics of manipulation.

Meyer, D. S. (2011, oct). *Mein Freund der Roboter*. BMBF/VDE Innovationspartnerschaft AAL. ISBN-10: 3800733420.

Mirza, K. and D. Orin (1990). Force distribution for power grasp in the digits system. In *Proc. 8th CISM-IFTOMM Symp. Theory Practice Robots Manipulators*.

Müggler, E., M. Fässler, D. Scaramuzza, S. Huck, and J. Lygeros (2013). Torque control of a kuka youbot arm.

Nguyen, V.-D. (1988). Constructing force-closure grasps. *The International Journal of Robotics Research* 7(3), 3–16.

Nowak (2015, dec). Hokuyo urg-04lx-ug01.

Panasonic (2005, may).

Parker, L. E. (1998). Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on* 14(2), 220–240.

Parker, L. E. (2003a). Current research in multirobot systems. *Artificial Life and Robotics* 7(1-2), 1–5.

Parker, L. E. (2003b). The effect of heterogeneity in teams of 100+ mobile robots. *Multi-Robot Systems* 2, 205–215.

Parker, L. E., M. Chandra, and F. Tang (2005). Enabling autonomous sensor-sharing for tightly-coupled cooperative tasks. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pp. 119–130. Springer.

Parker, L. E., B. Kannan, F. Tang, and M. Bailey (2004). Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, Volume 1, pp. 1016–1022. IEEE.

Prada, M., A. Remazeilles, A. Koene, and S. Endo (2014). Implementation and experimental validation of dynamic movement primitives for object handover. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2146–2153. IEEE.

Pynadath, D. V. and M. Tambe (2003). An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems* 7(1-2), 71–100.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, Volume 3, pp. 5.

- Saffiotti, A., N. B. Zumel, and E. H. Ruspini (2000). Multi-robot team coordination using desirabilities. In *Proc of the 6th Intl Conf on Intelligent Autonomous Systems*, pp. 107–114.
- Salisbury, J. K. and B. Roth (1983). Kinematic and force analysis of articulated mechanical hands. *Journal of Mechanical Design* 105(1), 35–41.
- Saxena, A., J. Driemeyer, and A. Y. Ng (2008). Robotic grasping of novel objects using vision. *The International Journal of Robotics Research* 27(2), 157–173.
- Sharma, S., G. K. Kraetzschmar, C. Scheurer, and R. Bischoff (2012, May). Unified closed form inverse kinematics for the kuka youbot. In *Proceedings of 7th German Conference on Robotics, ROBOTIK 2012*; pp. 1–6.
- Shehory, O. and S. Kraus (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence* 101(1), 165–200.
- Shibata, S., K. Tanaka, and A. Shimizu (1995). Experimental analysis of handing over. In *Robot and Human Communication, 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE International Workshop on*, pp. 53–58. IEEE.
- Siegemund, F. (2004). A context-aware communication platform for smart objects. In *Pervasive Computing*, pp. 69–86. Springer.
- Simmons, R., S. Singh, D. Hershberger, J. Ramos, and T. Smith (2001). First results in the coordination of heterogeneous robots for large-scale assembly. In *Experimental Robotics VII*, pp. 323–332. Springer.
- Simmons, R., T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. Zlot (2002). A layered architecture for coordination of mobile robots. In *Multi-robot systems: from swarms to intelligent automata*, pp. 103–112. Springer.
- Steidl, D. (2011).
- Stone, P. and M. Veloso (1999). Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* 110(2), 241–273.
- Timm, I. J. and P.-O. Woelk (2003). Ontology-based capability management for distributed problem solving in the manufacturing domain. In *Multiagent System Technologies*, pp. 168–179. Springer.
- Townsend, W. T. (1988). The effect of transmission design on force-controlled manipulator performance.
- und Holger Büttner, D.-I. D. J. (2003). Ethercat – der ethernet-feldbus. Technical Report 23, Elektronik.
- Vail, D. and M. Veloso (2003). Multi-robot dynamic role assignment and coordination through shared potential fields. *Multi-robot systems*, 87–98.

- Vaughan, R. T., K. Støy, G. S. Sukhatme, and M. J. Matarić (2000). Whistling in the dark: cooperative trail following in uncertain localization space. In *Proceedings of the fourth international conference on Autonomous agents*, pp. 187–194. ACM.
- Vig, L. and J. A. Adams (2005). Issues in multi-robot coalition formation. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pp. 15–26. Springer.
- Wagner, C. (2009). „Tele-Altenpflege“ und „Robotertherapie“: Leben mit Robotern als Vision und Realität für die alternde Gesellschaft Japans.
- Werger, B. B. and M. J. Matarić (2000). Broadcast of local eligibility for multi-target observation. In *Distributed autonomous robotic systems 4*, pp. 347–356. Springer.
- Zandonella, B. (2013, oct). Bevölkerungsentwicklung und Renten.
- Zlot, R. and A. Stentz (2005). Complex task allocation for multiple robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1515–1522. IEEE.

A Datenblätter

B Testdaten Inverse Kinematik

Pose	Diff Lin in cm	Diff Rot in °
1	0,0431	0,4581
2	0,2226	0,1110
3	0,3716	0,0594
4	0,0341	0,3559
5	0,1523	0,6661
6	0,0380	0,1763
7	0,0970	0,6651
8	0,1287	0,6553
9	0,3358	0,1190
10	0,0310	0,6070
11	0,0327	0,3736
12	0,0847	0,3640
13	0,3369	0,3008
14	0,1834	0,4781
15	0,0294	0,1417
16	0,3784	0,2441
17	0,2717	0,1017
18	0,1019	0,4557
19	0,1778	0,1373
20	0,2016	0,4597
21	0,2314	0,1427
22	0,0504	0,1256
23	0,1733	0,5850
24	0,0187	0,2859
25	0,1995	0,5118
Mittelwerte	0,1570	0,3432
Max	0,3784	0,6661
Min	0,0187	0,0594