Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Aaron Gornott

Kollaboration humanoider Roboter im Kontext einer Transportaufgabe

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Aaron Gornott

Kollaboration humanoider Roboter im Kontext einer Transportaufgabe

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke

Eingereicht am: 17. November 2014

Aaron Gornott

Thema der Arbeit
Kollaboration humanoider Roboter im Kontext einer Transportaufgabe

Stichworte
Kollaboration, Robotik, Nao, Computer Vision, Bewegungsplanung

Kurzzusammenfassung
Im Rahmen dieser Bachelorarbeit wurde ein Softwaresystem zur Kollaboration zweier humanoider Roboter entwickelt. Ziel war der gemeinsame Transport eines langen Objektes entlang eines rechtwinklig abknickenden Ganges.
Hierzu wurden notwendige Informationen zur Planung und Steuerung des Transports von den Bildern einer Kamera, welche den Versuchsaufbau überblickt, abgeleitet.

Aaron Gornott

Title of the paper
Collaboration Of Humanoid Robots In The Context Of A Transport Task

Keywords
Collaboration, Robotics, Nao, Computer Vision, Motion Planning

Abstract
In the context of this bachelor thesis a software system for the collaboration of two humanoid robots was developed. A joint transport of a long object along a right angled hallway was intended.
The information for the planning and control of the transport was deduced by a camera with a test set-up overview.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Roboter werden im Alltag immer allgegenwärtiger. Das sich dieser Trend auch in Zukunft fortsetzt wird u.a. von der Wirtschafts-Wachstumsstrategie des Japanischen Regierungsausschusses bestätigt. Ministerpräsident Abe (2014) kündigte unter dem Schlagbegriff Roboter-Revolution, zum Jahr 2020 eine Verdopplung des japanischen Industrierobotermarktes, sowie eine 20-fache Erhöhung im Bereich der Serviceroboter, auf insgesamt 2,4 Billionen Yen, an.
In Zukunft werden Roboter somit in Lebensbereiche von Menschen vordringen und dabei in Umgebungen operieren, welche für die menschliche Anatomie gestaltet worden sind. Um mit diesen Umgebung interagieren zu können, wird häufig eine humanoide Konstruktion angestrebt. Fortschritte in der Entwicklung humanoiden Roboter werden durch Wettbewerbe, wie der Robotics Challenge von der amerikanischen Defense Advanced Research Projects Agency (2014), gefördert.

Service Roboter werden körperlich anstrengende oder sich wiederholende Tätigkeiten übernehmen. Der Transport von schweren oder sperrigen Objekten, durch schmale Korridore, ist ein Anwendungsfall, der in diese Kategorien fällt. Roboter können in einer solch limitierenden Umgebung nicht beliebig in ihrer Dimension oder Kraft aufwärts skaliert werden. Durch die Unfallgefahr mit Menschen ist dies auch nicht erwünscht. Stattdessen ist eine Kooperation mehrerer Roboter denkbar, wie sie auch in der Zusammenarbeit zwischen Menschen angewandt wird.

Die übergeordnete Vision dieser Bachelorarbeit ist ein Umzugsservice bei dem der Möbeltransport von autonomen Roboter-Umzugshelfern abgewickelt wird.

1.2 Aufgabenstellung

Das Ziel der Bachelorarbeit besteht darin, eine Transportaufgabe mit humanoiden Robotern durchzuführen. Dafür werden zwei Nao Roboter der Firma Aldebaran Robotics verwendet. Die Roboter sollen gemeinsam eine Transportlast einen rechtwinklig abknickenden Gang entlang tragen. Erschwerende Bedingung ist, dass das zu transportierende Objekt mindestens doppelt so lang wie der Gang breit ist. Seitliche Wände verhindern, das Teile der Roboters oder der Transportlast den zulässigen Bereich verlassen.
Die Roboter werden in gleicher Blickrichtung aufgestellt. Einer am vorderen Ende der Traglast, der andere am hinteren. Zu Beginn des Transportes steht der vordere Nao am Anfang der Abknickung im Gang. Der Transport gilt als erfolgreich, wenn der hintere Nao das Ende der Abknickung erreicht hat.

Eine Kamera filmt den Versuchsaufbau aus der Vogelperspektive. Aus den erhaltenen Kamerabildern werden notwendige Informationen zur Planung und Steuerung des Transports abgeleitet. Das daraus gewonnene Modell der Umgebung dient als Handlungsgrundlage der beiden Roboter. Es findet keine separate Erfassung der Umgebung, durch Kameras oder Sensoren der Roboter statt, welche in das gemeinsame Modell integriert werden. Zudem verwaltet kein Roboter ein eigenes isoliertes Modell, welches über den Funktionsumfang der Betriebssoftware hinausgeht.

1.3 Lesehinweise

- Alle als Listing bezeichneten Codebeispiele in dieser Arbeit sind in Python 2.7 der Python Software Foundation (2014) geschrieben.
- Werden der Bachelorarbeit beiliegende Dokumentationsdateien referenziert, die keinen Quellcode enthalten, so wird der Dateiname (ohne Suffix) in einfachen Anführungszeichen genannt.

2 Analyse

Im Analysekapitel wird untersucht welche Bestandteile zum Lösen der Aufgabenstellung Abschnitt 1.2 nötig sind. Dazu wird ein Überblick von möglichen Verfahren und Lösungsansätzen erstellt.

2.1 Computer Vision

Ein Roboter, der eine komplexe Aufgabe über eine Vielzahl an Bewegungen zu erfüllen hat, kann diese häufig nicht aufgrund Vorprogrammierter Bewegungsabläufe bewältigen. Denn vereinfachte kinematische Modelle berücksichtigen keine physikalischen Einflüsse. Zudem können minimale Messfehler der Servomotoren nicht identifiziert werden, was zu einem Kaskadeneffekt führt. Die interne Repräsentation entspricht dann nicht mehr der tatsächlichen Position. Aus diesem Grund benötigt ein Roboter, zur Bewältigung einer komplexer Aufgabenstellung, Kenntnis seiner Umwelt, welche er mit seiner internen Repräsentation abgleichen und diese gegebenenfalls anpassen kann. Es gibt eine Vielzahl an Möglichkeiten die Umwelt maschinell zu erfassen. In dieser Thesis wird dazu lediglich das automatische auswerten von Kameradaten verwendet.

2.1.1 Bildverständnis

Das Ziel der Computer Vision ist es aus den Bild-Rohdaten Informationen abzuleiten. Dabei kann das Bildverständnis, nach Crevier und Lepage (1997), als ein wissensbasierter Prozess betrachtet werden. Hierarchisch liegen die Bildverständnis-Schichten übereinander, wobei höhere Schichten einen höheren Auswertungsaufwand benötigen.

1. Segmentierung:
   Bilder können auf Kanten, Texturen oder Merkmale von Regionen untersucht werden.

2. Bildverstehen auf niedriger Ebene:
   Räume, 3D-Flächen und 3D-Umrisse werden zugeordnet.

3. Objekt Erkennung:
   Objekte können identifiziert werden und Bewegungsabläufe werden nachvollzogen.

4. Bildverstehen auf hoher Ebene:
   Ereignisse, Abläufe und Objekt-Konfigurationen sind erfassbar.

Das SimpleCV Framework von Sight Machine Inc. (2014a) kann für die genannten Aufgabenbereiche niedriger Ebene eingesetzt werden. Weitere Details dazu sind dem Abschnitt 3.3 Computer Vision des Design und Realisierung Kapitels zu entnehmen.
Der Abschnitt 3.4 Interne Repräsentation im Kapitel Design und Realisierung befasst sich mit dem Bildverständnis auf höherer Ebene.

2.2 Bewegungsplanung

Aufgabe der Bewegungsplanung ist es, ein komplexes Bewegungsvorhaben in eine Abfolge von konkret ausführbaren Einzelbewegungen zu zerlegen. Als Informationsgrundlage muss der Bewegungsplanung eine Angabe von Start- und Ziel-Zustand vorliegen.

Die maximale Länge einer Stange, die durch einen rechtwinklig abknickenden Gang transportiert werden kann, beträgt $2\sqrt{2}$, bezogen zur einheitlichen Gangbreite. Auf dem Idealpfad dreht sich die Stange mittig um die innere Ecke und erreicht die Begrenzung in ihrer maximalen Abmessung bei einem 45° Winkel zum Gang. Dies ist nach Finch (2003) als Moving Ladder Problem bekannt (Abbildung 2.2 Moving Ladder Problem). Das Moving Ladder Problem ist

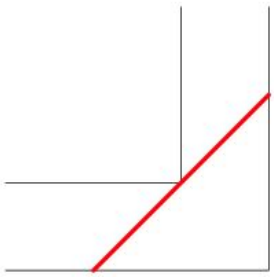

Abbildung 2.1: Moving Ladder Problem, von Eric W Weisstein (2014)

jedoch kein ausreichendes Modell für die Transportaufgabe dieser Bachelorthesis. Denn die tragenden Roboter werden darin nicht berücksichtigt. Eine geeignete Planungsgrundlage kann mithilfe der Open Motion Planning Library (OMPL) von der Rice University - Physical and Biological Computing Group (2014) aufgebaut werden, die eine Vielzahl an Zustandsräumen

und Planungsalgorithmen bietet. In den folgenden Unterabschnitten wird die Funktionsweise von zwei geeigneten Planungsalgorithmen erläutert.

2.2.1 Rapidly Exploring Random Tree

Der Rapidly Exploring Random Tree (RRT) Algorithmus von Lavalle (1998) benutzt zur Planung einen sich vom Startzustand räumlich ausbreitenden Baum. Erste Kindknoten versuchen dabei eine möglichst große Distanz zurückzulegen, wobei bevorzugt in große leere Regionen expandiert wird. Leere Regionen lassen sich durch die Zerlegung des Raumes in ein Voronoi-Diagramm nach Aurenhammer (1991) messbar darstellen. Jeder Knoten wird dabei das Zentrum einer Region, deren Grenze sich dadurch definiert, dass kein anderer Knoten der Region näher ist. Anschaulich wird dies durch die Abbildung 2.2.1 Voronoi-Diagramm.

Der RRT reduziert mit der Tiefe des Baumes die Expansionsdistanz, abhängig davon wie

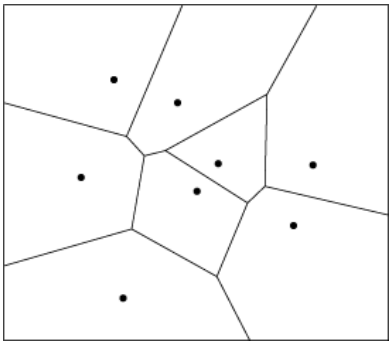

Abbildung 2.2: Voronoi-Diagramm, von National Instruments (2011)

nah der dichteste Nachbarknoten ist. Durch diese Methode wird zunehmend mehr Raum ausgefüllt, ohne dass sich Äste des Baumes kreuzen (Abbildung 2.2.1 Rapidly Exploring Random Tree). Trifft ein expandierender Knoten zufällig den Zielzustand, terminiert der Algorithmus. Lösungspfad der Bewegungsplanung ist der Pfad von der Wurzel zu dem Knoten auf dem Zielzustand, der Pfad kann (ohne Baumsuche) über die Elternbeziehung extrahiert werden.

Die Expansionsstrategie lässt sich beliebig erweitern, z.B indem der Baum sich bevorzugt in Richtung Zielzustand ausbreitet. Um Hindernisse zu umgehen, wird ein neuer Ast beim Hinzufügen zum Baum darauf untersucht, dass er sich im gültigen Zustandsraum befindet.

RRT ist für das Lösen von Planungsaufgaben mit nicht konvexen geometrischen Figuren konzipiert, dies entspricht der Problemklasse des abknickenden Gangs in dieser Thesis. Die geometrische Form der Problemstellung ist symmetrisch aufgebaut: im Zentrum befindet sich der kritische Punkt, dass Durchqueren der Kurve, während sich das Ein- und Ausschwenken

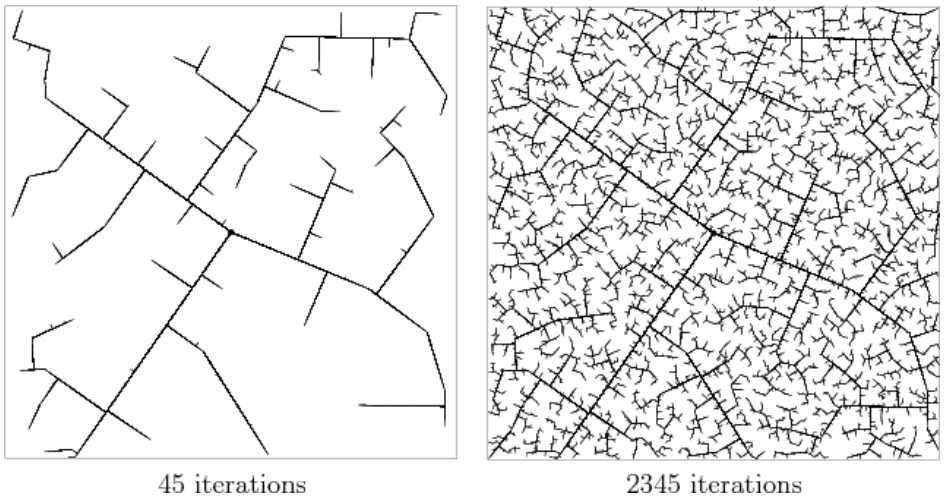

Abbildung 2.3: Rapidly Exploring Random Tree, nach Steven M LaValle (2006)

in die Kurve spiegeln lässt. Für diese Besonderheit eignet sich die Variante RRT-Connect von Kuffner und LaValle (2000). RRT-Connect startet die Expansion eines Baumes sowohl am Start- als auch am Ziel-Zustand. Ein gültiger Lösungspfad ist gefunden, sobald die Bäume sich berühren.

2.2.2 Probabilistic Roadmap

Ein weiterer Ansatz zur Bewegungsplanung ist das Erstellen einer Probabilistic Roadmap (PRM) nach Kavraki u. a. (1996). Hier wird der Zustandsraum mit zufällig bestimmten Zuständen gefüllt. Im ersten Schritt werden die Zustände zu einem kantengewichteten Graphen verbunden. Dazu wird definiert welche Zustände sich verbinden sollen. Dies kann in Abhängigkeit ihrer Distanz zueinander geschehen. Berücksichtigt wird zudem, wo sich Hindernisse im Zustandsraum befinden, dort sind keine Verbindungen zulässig. Der Lösungspfad, vom Start- zum Ziel-Zustand, wird anschließend mit dem Algorthmus des kürzesten Pfad nach Dijkstra (1959) bestimmt.
Selbst wenn es einen gültigen Pfad vom Start- zum Ziel-Zustand gibt, kann dieser nur gefundenen werden, wenn die Probabilistic Roadmap genügend Verbindungen zwischen den zufällig bestimmten Zuständen aufgebaut hat.

2.3 NAOqi

Für die Interaktion mit den Naos stellt Aldebaran Robotics (2014c) das NAOqi Framework zur Verfügung. NAOqi selbst basiert auf einem Linux Kernel, bietet als Cross-Platform-Framework aber die Möglichkeit auf Windows, Linux oder Mac Systemen zu entwickeln.

Der NAOqi Broker ermöglicht Methodenaufrufe über eine Netzwerkverbindung und über-

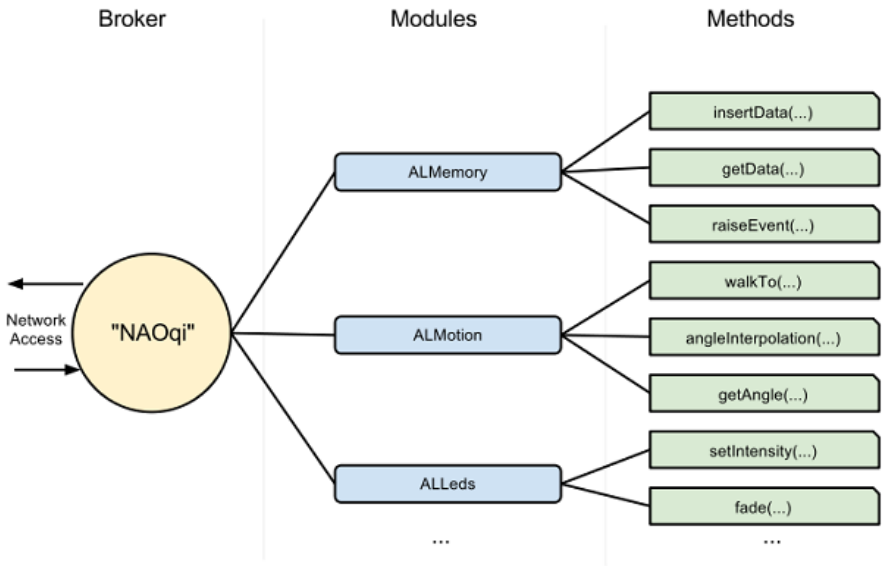

Abbildung 2.4: NAOqi Struktur, nach Aldebaran Robotics (2014d)

nimmt die Aufgabe von einem Verzeichnisdienst zur Lokalisierung von Modulen (Abbildung 2.3 NAOqi Struktur). Dabei unterscheidet der Broker nicht, ob sich ein Modul im lokalen Speicher des Naos befindet oder an einer externen Stelle im Netzwerk.
NAOqi Module können in C++ und Python geschrieben werden. Darüber hinaus wird eine API[1] in C#, Visual Basic, F#, Java, Matlab und Urbi angeboten, die über einen Wrapper C++ oder Python Code aufruft.

[1] Application Programming Interface

Aufgrund der externen Kamera liegt der Fokus in dieser Bachelorarbeit weniger auf den sensorischen Funktionen der Naos und mehr auf der Aktorik. Zentrales Modul hierfür ist ALMotion, mit dem sich sämtliche Freiheitsgrade direkt beeinflussen lassen. Eine Übersicht über die ansteuerbaren Freiheitsgrade bietet die Abbildung 2.3 Nao Freiheitsgrade. Zudem bietet ALMotion Bewegungsmethoden auf hohem Abstraktionsniveau, um den Roboter mit koordinatenbasierten Parametern zu einem Zielort zu bewegen. Dabei muss nicht jeder Schritt einzeln vom Anwender geplant werden; auch Störeinflüsse, wie ein schräger Untergrund, werden autonom ausgeglichen.

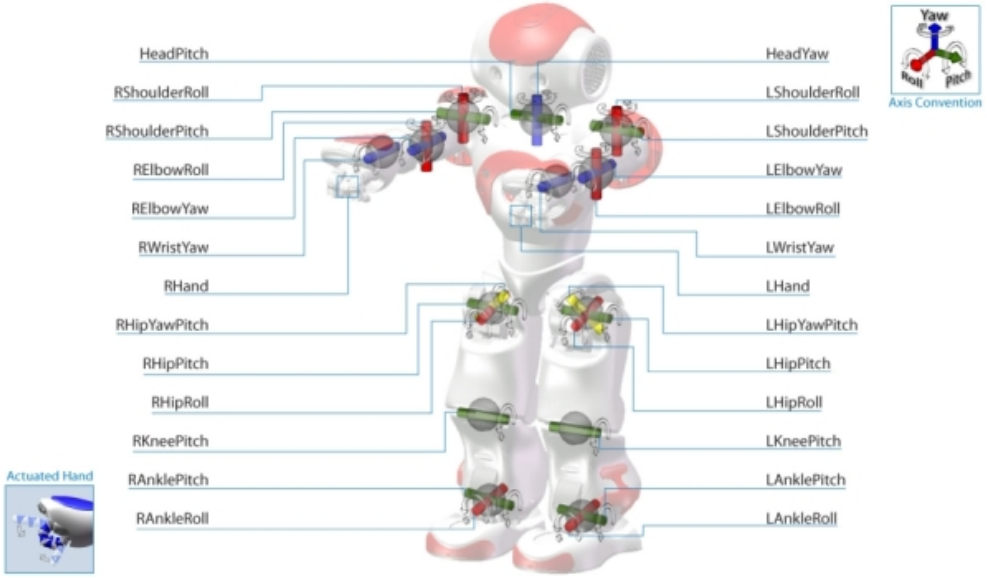

Abbildung 2.5: Nao Freiheitsgrade, nach Aldebaran Robotics (2014b)

3 Design und Realisierung

3.1 Laboraufbau

Im Abschnitt Laboraufbau wird die verwendete Hardware im Einzelnen erklärt. Die meisten Elemente, welche in den folgenden Unterabschnitten behandelt werden, sind auf der Abbildung 3.1 Versuchsaufbau zu identifizieren.

3.1.1 Roboter und Transportlast

Es werden zwei Nao H25 V3.3 Roboter von Aldebaran Robotics (2014a) verwendet. Die Roboter haben eine Standhöhe von 574mm und sind 275mm Breit. Naos der Version 3.3 besitzen gegenüber der Vorgängerversion 3.2 um 21mm verlängerte Arme, auf insgesamt 311mm. Dies soll bei Stürzen zu einem robusteren eingreifen des Fallmanagers führen und insbesondere einen größeren Bewegungsfreiraum für Transportaufgaben gewähren.
Transportiert wird eine 115cm lange Holzstange mit einem Durchmesser von 24mm. Die Finger der Naos bestehen aus glattem Kunststoff und können keine starken Kräfte beim Greifen entwickeln. Aus diesem Grund werden Finger und Stange bei der Transportaufgabe mit Klebeband zusammengehalten.
Für die Kameraortung werden die Schultern mit je einer runden Farbmarkierung beklebt. Die rechte Schulter, des in Blickrichtung vorderen Naos, bekommt einen grünen Kreis. Die verbleibenden Markierungen sind blau.

3.1.2 Hindernisse

Der Transport findet in einem Gang von 350cm Länge statt, welcher auf der Hälfte der Strecke im rechten Winkel abknickt. Die Seitenwände stehen einheitlich im Abstand von 50cm zueinander und beschränken den Bewegungsfreiraum der Roboter und der Stange. Nach oben ist der Gang einsehbar. Der Gang ist modular aufgebaut, so dass er für unterschiedliche Testläufe, an die jeweiligen Anforderungen angepasst werden kann.

3.1.3 Peripherie

Mit einer Logitech QuickCam Pro 9000 USB Kamera wird der Laboraufbau, über ein Stativ, aus der Vogelperspektive aufgenommen. Ein Notebook verarbeitet den Kamerainput und ist der zentrale Punkt in dem Versuchsaufbau. Über einen Router verbindet sich das Notebook mit den Robotern, welche per NAOqi-Proxy angesteuert werden.

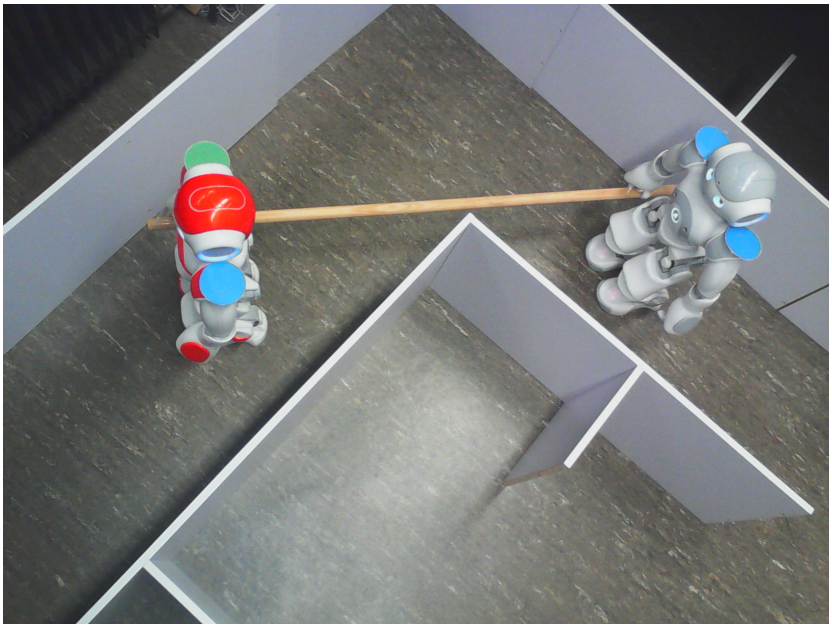

Abbildung 3.1: Versuchsaufbau

3.2 Softwarearchitektur

Im folgenden Abschnitt wird auf die verwendete Softwarearchitektur eingegangen. Das Komponentendiagramm (Abbildung 3.2) bietet einen Überblick über die Verteilung der einzelnen Komponenten.

Das Artefakt Main initialisiert alle verwendeten Komponenten. Der Controller verwaltet

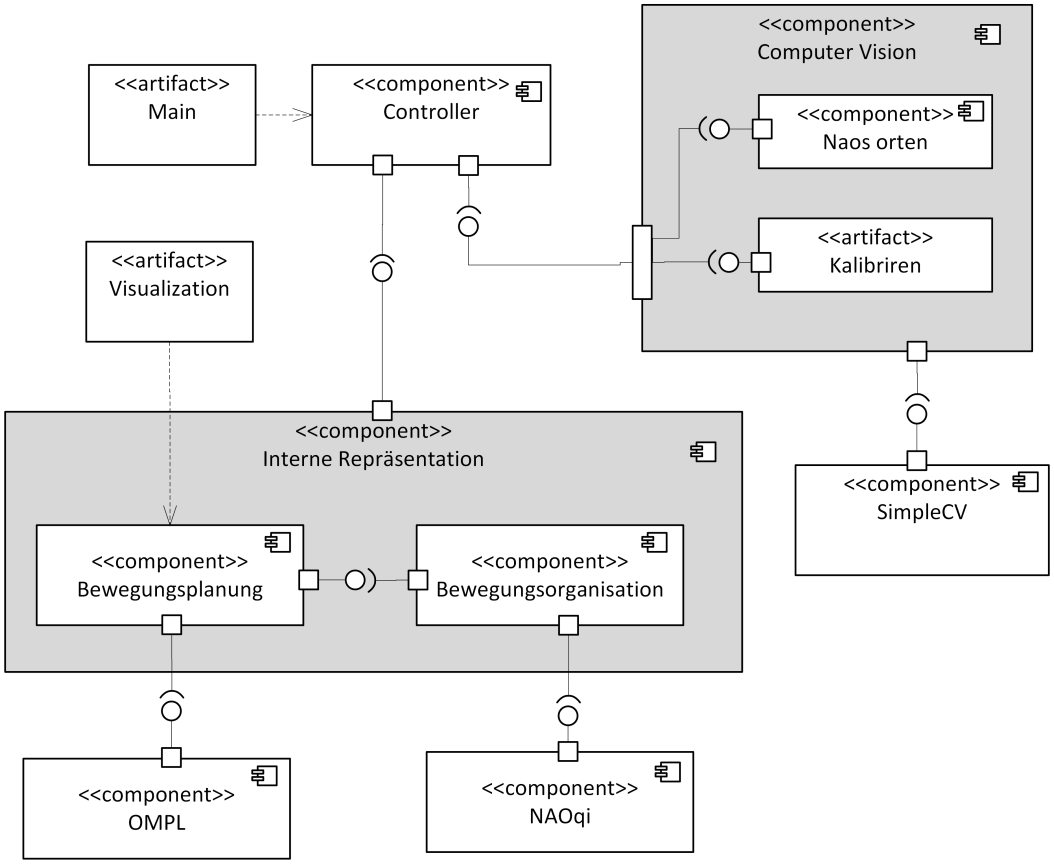

Abbildung 3.2: Komponentendiagramm

die Komponenten der internen Repräsentation, sowie der Computer Vision in der Hauptschleife mainLoop (Listing 3.1), durch Aufruf der executeIteration Methode. Des Weiteren organisiert der Controller den Datenfluss von der Computer Vision Komponente zur internen Repräsentation und ist für den Programmfluss verantwortlich.
Die Computer Vision Komponente arbeitet auf Basis des SimpleCV Frameworks.
Die Subkomponente Bewegungsorganisation greift auf das NAOqi Framework, sowie die

Bewegungsplanung zu; während die Subkomponente Bewegungsplanung die Open Motion Planning Library benutzt.
Visualization als Stand-Alone Artefakt ermöglicht eine grafische Aufbereitung der Bewegungsplanung.

```
def mainLoop(self):
  while self.environment.getDisp().isNotDone():
    self.environment.executeIteration(self._programState)
    self.worldModel.setEnvironment(
          self.environment.getCalibrationFaktor(), ##
          self.environment.getBorderPoints(),      ##
          self.environment.getNaoMarker().getLastValidGreenPoint(),
          self.environment.getNaoMarker().getLastValidBluePoints())
    self.worldModel.executeIteration(self._programState)
    self.programmFlow()
    self.frameRate()
  self.environment.quitCV()
```

Listing 3.1: Controller.mainLoop

Die Bestandteile des von Main initialisierten Systems sind nach dem Konzept des objektorientierten Programmierparadigma nach Kay (1996) entworfen worden. Sämtliche Strukturierung und Kommunikation wird über Objekte abgewickelt.

3.2.1 Verwendete Architekturmuster

Folgende Architekturmuster definieren gemeinsam die Grundstruktur der Software:

- Komponentenbasierte Entwicklung, McIlroy (1968)
  Die Komponenten (Abbildung 3.2: Komponentendiagramm) besitzen an den aufgezeigten Schnittstellen nur eine lose Kopplung, um einfach ausgetauscht werden zu können.

- Separation of Concerns, Dijkstra (1982)
  Beim Design wurde darauf geachtet, dass die Zuständigkeitsbereiche klar definiert und voneinander getrennt sind. Die Verwendung von objektorientierter Programmierung und komponentenbasierter Entwicklung unterstützt die Einhaltung dieser Anforderung.

- Design by Contract (Teilweise angewendet), Meyer (1992)
  Im Zusammenspiel der internen Repräsentation und der Bewegungsorganisation müssen eine Reihe von Invarianten eingehalten werden. Dazu werden unterschiedliche Pre- und Post-Conditions aufgestellt, im Abschnitt 3.4 Interne Repräsentation und 3.6 Bewegungsorganisation wird dies detailliert erläutert.

- Schichtenarchitektur (Teilweise angewendet), Eckerson (1995)
  Viele Komponenten kommunizieren ausschließlich unidirektional mit Komponenten tieferer Schichten. Dennoch wird dieses Architekturmuster nicht durchgehend eingesetzt. Insbesondere der Informationsfluss auf gleicher Ebene in der internen Repräsentation steht hierzu im Widerspruch.

- Proxy (Durch Framework angewendet), Gamma u. a. (1995)
  Proxy ist nicht den Architekturmustern zuzuordnen, sondern den Entwurfsmustern. Aufgrund der zentralen Bedeutung wird es hier dennoch erwähnt. Alle an die Naos gesendeten Befehle werden von der Stellvertreterschnittstelle, dem NAOqi Proxy, entgegengenommen. Das Interface entspricht einem lokalen Aufruf, wodurch absolute Zugriffstransparenz[1] implementiert wird.

3.3 Computer Vision

Durch die Computer Vision Komponente wird die Umgebung visuell erfasst und ausgewertet. Identifiziert werden sollen die Position und Ausrichtung der Roboter. Hierfür werden die Roboter mit Farbmarkierungen ausgestattet. Über das Computer Vision System wird auch die Hindernisanordnung erfasst, welche überwunden werden soll.
Vereinfacht wird davon ausgegangen, dass eine Kamera aus der Vogelperspektive den aufgabenrelevanten Bereich komplett beobachten kann. So entfällt die Notwendigkeit aus den Einzelperspektiven aller beteiligten Akteure ein kohärentes Gesamtbild zu konstruieren.
Verwendet wird hierfür das Framework SimpleCV. Dieses liefert eine Python Schnittstelle zur verbreiteten OpenCV Bibliothek von Intel Corporation, Willow Garage, Itseez (2014). OpenCV ist in C geschrieben und wird als Standardbibliothek für ComputerVision Aufgaben eingesetzt. Die Verwendung von SimpleCV bietet den Vorteil, alle Komponenten einheitlich in Python entwickeln zu können. Eine Python Kapselung von OpenCV reduziert erwartungsgemäß die Verarbeitungsgeschwindigkeit. Mit dem minimalen Aufgabenspektrum, welches der Computer

[1] Einheitliches Interface für lokale- und entfernte-Ressourcen

Vision im Rahmen der Implementierung zukommt, stellt die Performance allerdings keine kritische Komponente dar.

3.3.1 Kalibrierung

Um die Kameradaten auswerten zu können, ist eine Kalibrierung nötig. Kalibriert wird dafür zunächst, die Umrechnung von Bildpixeln zu einer metrische Längeneinheit. Hierzu wird in der Versuchsumgebung ein Muster mit bekannten Abmessungen ausgelegt. Aus der fixierten Position des Kamerabildes werden nun die Umrechnungsfaktoren berechnet.
Zum Erkennen von Farben, in der jeweiligen Belichtungssituation, werden Grenzwerte im RGB-Farbraum[2] ermittelt. Ziel ist es, diese Grenzwerte so stabil zu wählen, dass die Farbmarkierungen der Roboter stets zuverlässig erkannt werden können, ohne jedoch Umgebungsdetails fälschlicherweise mit den Markierungen zu verwechseln. Mehr Informationen sind im der Evaluation, Abschnitt 4.3 Kamera Kalibrierung, zu finden.

3.3.2 Umgebungshindernisse

Die Begrenzungen, in denen sich die Naos bewegen sollen sind statisch. Es besteht somit kein Bedarf deren Position zur Laufzeit zu erfassen. Um die Komplexität und Fehleranfälligkeit der Computer Vision Komponente so gering wie möglich zu halten, werden die Umgebungshindernisse nicht automatisch erkannt. Stattdessen wird der Hindernisverlauf des Versuchsaufbaus über das Kamerabild manuell, per Mausklick, in das System eingeben.

3.3.3 Mobile Objekte

Hauptaufgabe der Computer Vision Komponente ist es Auskunft über die Position der Roboter zu liefern, indem vier farbige Punkte erfasst werden. Eine Validierung der Bedeutung und Richtigkeit der Daten findet nicht innerhalb der Computer Vision Komponente statt, sondern wird im Abschnitt 3.4 der internen Repräsentation betrachtet.

[2] Rot, Grün, Blau

Abbildung 3.3: Unbearbeitetes Bild, von Sight Machine Inc. (2014d)

Abbildung 3.4: Farbabstand zu Gelb, von Sight Machine Inc. (2014b)

Abbildung 3.5: Reduziertes Bild für den Zielfarbraum, von Sight Machine Inc. (2014c)

Die Farbmarkierungen werden von der Klasse NaoMarker identifiziert. Für einen gültigen Suchlauf müssen genau eine grüne und drei blaue Markierungen gefunden werden[3]. Diese liefert die SimpleCV Methode findBlobs (Listing 3.2), indem das Bild dafür auf die gesuchte Farbe reduziert wird. Dies ist keine isolierte Betrachtung eines einzelnen Farbraums, die 3-Tupel (Rot,Grün,Blau) Bildpunkte werden dabei zusammenhängend betrachtet. Von allen 3-Tupel Bildpunkten wird zunächst der Abstand zum Zielfarbraum ermittelt (Abbildung 3.4 Farbabstand zu Gelb), dieser Abstand wird vom Eingabebild subtrahiert. Im neu errechneten Bild leuchtet dadurch die gesuchte Farbe hell auf, während andere Farben schwarz ausgeblendet werden (Abbildung 3.5 Reduziertes Bild für den Zielfarbraum).

```
1 def blueSegmentation(self):
2   return (self._img) -(self._img.colorDistance(Color.BLUE))
3
4 def findBlueMarker(self):
5   # DEFAULT: findBlobs(threshval=-1, minsize=10, maxsize=0,
6   # threshblocksize=0, threshconstant=5, appx_level=3)
7   self._blueBlobs = self.blueSegmentation().findBlobs(
8                     self._blueThreshVal, 8, 0, 0, 5, 3)
```

Listing 3.2: NaoMarker findBlobs

Auf eine zusätzliche Binärisierung[4] des segmentierten Bildes wird bewusst verzichtet, um bei den Parametereinstellungen der findBlobs Methode mehr Flexibilität zu behalten.

[3] Nur grüne und blaue Farbspektren konnten im Versuchsaufbau zweifelsfrei identifiziert werden.
[4] Definierter Grenzwert reduziert die Bildpunkte auf Schwarz oder Weiß.

Nach erfolgreicher Identifikation zeichnet die drawMarker Methode die gefundenen Positionen in das Kamerabild ein. Dieses bearbeitete Bild wird über den Display ausgegeben und liefert dem Betrachter die Möglichkeit, die interne Verarbeitung zur Laufzeit nachvollziehen zu können.

3.4 Interne Repräsentation

Nachdem sich Kapitel 3.3 Computer Vision, mit dem Erfassen der Umgebung beschäftigt hat, ist der nächste logische Schritt der Aufbau einer internen Repräsentation der Welt. Dazu wird ein abstraktes Modell des Versuchsaufbaus generiert, dass möglichst einfach und minimal ist. Je einfacher ein Modell ist, desto eher lassen sich Fehlerquellen erkennen und Modifikationen vornehmen. Andererseits muss die interne Repräsentation robust genug sein, so dass die Roboter möglichst alle eventuellen Störeinflüsse bei ihrer Transportaufgabe autonom lösen können. Die zu treffende Designentscheidungen sind eine Abwägung zwischen diesen Anforderungen.

Über den Controller werden die Daten der Computer Vision Komponente an die interne Repräsentation weitergeleitet. Zentrales Element ist hierbei die Klasse WorldModel, welche die Daten über ihre setEnvironment Schnittstelle entgegennimmt. Alle entgegengenommenen Parameter werden von Pixel in eine metrische Längeneinheit umgerechnet.
Damit ist die Modellierung der als Polygon übergebenen Hindernisumgebung abgeschlossen. Die evaluateEnvironment Methode übernimmt das Auswerten für die Nao Instanzvariablen, in denen Informationen über aktuelle Position und Orientierung gespeichert werden.

3.4.1 Auswertung

Ziel der Auswertung ist es, die Farbmarkierungen, aufgrund ihrer Position, den Nao-Schultern zuordnen zu können. Es wird dabei vereinfachend angenommen, dass die Naos während des gesamten Transportes im gleichen Winkel zueinander stehen und ihren Griffpunkt an der Transportstange nicht variieren. Diese Annahme des Starrkörper Paradigmas widerspricht allerdings den tatsächlichen Roboterbewegungen in der Praxis. Im folgenden Absatz wird erklärt, unter welchen Annahmen dennoch damit gearbeitet werden kann.
Die Zugehörigkeit der grünen Markierung ist klar definiert, als rechte Schulter des führenden Roboters. Die diesem Punkt nächste blaue Markierung wird als linke Schulter des führenden Roboters zugeordnet. Dies ist aufgrund der Abmessungsunterschiede der Roboterbreite zur Transportlast eine relativ solide Annahme. Die verbleibenden zwei blauen Markierungen lassen sich, aufgrund ihrer Position, hingegen nicht eindeutig zuordnen sobald der folgende Roboter Orientierungsabweichungen unterliegt. Diese Zuordnung ist folglich extrem fehleranfällig.

Daher wird die Zusatzbehauptung aufgestellt, dass die Winkelabweichung der Roboter zueinander 0,5 π Radiant nicht übersteigt. Wird dennoch eine größere Abweichung erkannt, nimmt die interne Repräsentation an, einen Fehler gemacht zu haben und tauscht die falsch zugeordneten Schulterpunkte.
Übersteigt die Orientierungsabweichung in der Praxis tatsächlich einen Winkel von 0,5 π Radiant, führt dies zu einen von der Software nicht erkennbaren Fehler. Beim Eintreten einer solch erheblichen Abweichung zum Sollwert muss bereits eine Störung vorangegangen sein, die vermutlich ohnehin zu einem Abbruch führen sollte.

Nachdem die Positionen der Naos eindeutig bestimmt wurden, finden weitere Validitätsprüfungen statt, um von der Computer Vision gemachte Fehler identifizieren zu können. Verifiziert wird, ob der Abstand der Schultern mit der Spezifikation übereinstimmt.
Es wird überprüft, ob die Naos sich bewegt haben. Falls nicht, kann dies auch bedeuten, dass das Kamerabild eingefroren ist. In beiden Fällen wäre ein Überschreiben der vorherigen Bewegungsbefehle nicht erwünscht. Wird eine dieser Assertions nicht eingehalten, werden alle zuvor extrahierten Informationen verworfen und nicht weiter berücksichtigt. Die Liste der einzuhaltenden Invarianten ließe sich noch beliebig verlängern. Zum Beispiel indem geprüft wird, ob ein Roboter, sich in einer Zeitspanne, nicht weiter als eine physikalisch mögliche Distanz bewegt hat. Die nötige Robustheit, für die Laborversuche der Bachelorarbeit, wird aber mit dem Beschriebenen erreicht.
Als mathematische Grundlage für die Berechnungsschritte wurde die Utility-Klasse Calculation implementiert. Calculation bietet eine Vielzahl grundlegender Klassenmethoden für Winkel-, Längen- und Koordinaten-Operationen.

3.5 Bewegungsplanung

Für die Bewegungsplanung wird das Open Motion Planning Library (OMPL) Framework verwendet. OMPL bietet eine Vielzahl an Planungsalgorithmen. In der Laborumgebung wurde das PRM Verfahren, sowie RRTConnect getestet, siehe Analysekapitel Abschnitt 2.2.1 Rapidly Exploring Random Tree und 2.2.2 Probabilistic Roadmap. Durch den Versuch vom Startpunkt aus möglichst große Distanzen zu überwinden, wirken die von RRTConnect erzeugten Pfade weniger geradlinig und weisen chaotischere Muster gegenüber den PRM Lösungen auf. Daher wurde PRM letztendlich bei Laborversuchen vorgezogen.
Des Weiteren bietet OMPL eine Vielzahl an Zustandsräumen. Passend auf die interne Repräsentation wurde die Implementation SE2StateSpace gewählt. SE2StateSpace ist für die Rotation

und Translation von Starrkörpern im zweidimensionalen Raum ausgelegt. Die Boolesche-Methode isStateValid (Listing 3.3) definiert den gültigen Zustandsraum. Ein mathematisches Modell, der internen Repräsentation, muss für jeden möglichen Zustand (Position und Rotation) im Zustandsraum einen booleschen Wert annehmen, der definiert, ob er gültig ist oder nicht. Der Methodenparameter state, im isStateValid Code, entspricht einem von OMPL definierten Zustand aus dem möglichen Zustandsraum. Polarkoordinaten sind im Format [Abstand, Winkel] aufgebaut. Die Variable robotDimensionsPolarCoordinate enthält die vier Starrkörpers-Nao-Schulterpunkte, relativ vom Starrkörperzentrum.

```
def isStateValid(state):
  robotPoints = []
  for p in robotDimensionsPolarCoordinate:
    angle = Calculation.normalizeAngleToPi(state.getYaw() + p[1])
    relativeX = math.cos(angle) * p[0]
    relativeY = math.sin(angle) * p[0]
    robotPoints.append( [state.getX() + relativeX,
                         state.getY() + relativeY] )
    robotPolygon = path.Path(robotPoints)
  return( borderPolygon.contains_path(robotPolygon) and not
          borderPolygon.intersects_path(robotPolygon, False) )
```

Listing 3.3: OMPL planningKernel.isStateValid

Dem Planer muss zusätzlich eine Problemdefinition mitgeliefert werden. Minimal besteht diese aus der Angabe von Start- und Ziel-Zustand. Als optionaler Parameter kann die Auflösung der Zustandsvalidierung angeben werden, was erheblichen Einfluss auf die Laufzeit der Pfadberechnung hat. Die Auflösung wird dabei als Prozentsatz über die maximale Ausdehnung des Zustandsraums definiert.

3.5.1 Visualisierung

Die möglichen Pfade werden als Matrix ausgegeben. Jede Zeile repräsentiert einen Wegpunkt mit dem Spalteninhalt der Reihenfolge X-Position, Y-Position und Gierwinkel. Diese Werte gelten für das Starrkörperzentrum und sind wenig anschaulich. Zum Debuggen, wie zur Dokumentation, wurde deshalb mit der von Hunter u. a. (2014) entwickelten matplotlib Bibliothek eine Visualisierung erstellt. Das mathematische Grundmodell entspricht dem Aufbau vom Listing 3.3. In der Methode animate (Listing 3.4) wird der gleitende Verlauf der robotDimensionsPolarCoordinate zwischen den einzelnen Wegpunkten berechnet. Die Struktur data ist ein zweidimensionales Array dessen Inhalt der zuvor beschriebenen Matrix entspricht. Die

Variable frames definiert die Anzahl an Bildern pro Wegpunkt. Wobei der Methodenparameter i die aktuellen Animationsschritte bis zu einem Maximalwert von `len(data)*frames` hochzählt. Rückgabewert der animate Funktion ist das zu verschiebende Polygon movingBody. Der Dokumentationsordner '5 - Basic Motion Planning and Scalability' enthält zwei Visualisierungsvideos, sowie Textdateien mit der zugrundeliegende Datenbasis.

```
def animate(i):
  dataIndex = i / frames
  if dataIndex >= len(data) - 1:
    dataIndex = len(data) - 2
    i = dataIndex * frames + frames

  driftX = (data[dataIndex + 1][0] - data[dataIndex][0]) / frames *
           (i - dataIndex * frames) + data[dataIndex][0]
  driftY = (data[dataIndex + 1][1] - data[dataIndex][1]) / frames *
           (i - dataIndex * frames) + data[dataIndex][1]

  angleDiff = Calculation.getOrientationDifference(data[dataIndex][2],
              data[dataIndex + 1][2])
  driftYaw = angleDiff / frames * (i - dataIndex * frames) +
             data[dataIndex][2]
  robotPosX = []
  robotPosY = []
  for p in robotDimensionsPolarCoordinate:
    angle = driftYaw + p[1]
    relativeX = math.cos(angle) * p[0]
    relativeY = math.sin(angle) * p[0]

    robotPosX.append(relativeX + driftX)
    robotPosY.append(relativeY + driftY)

  movingBody.set_data(robotPosX, robotPosY)
  return movingBody
```

Listing 3.4: Visualization.animate

3.5.2 Positionsdesignation

Die Klasse Planning ist die Schnittstelle zwischen der Bewegungsplanung und der Bewegungsorganisation (Abschnitt 3.6). Planning liefert konkrete Weltkoordinaten und Ausrichtungen zugeschnitten auf die Positionsrolle der Roboter. Die Positionsrolle definiert welcher Roboter am vorderen Tragpunkt, und welcher am hinteren Tragpunkt, der Transportstange zu stehen hat.
Die Rollenzuweisung geschieht in der getTargetWorldCoordinate Methode (Listing 3.5). Abhängig vom Nao, der per Methodenparameter übergeben wird, wird die relative Zielposition an den vorderen oder hinteren Tragpunkt projiziert. Die absolute Zielposition, für den jeweiligen Nao, ergibt sich durch Addition mit dem transportCenter, welches das Starrkörperzentrum repräsentiert.

```
1  def getTargetWorldCoordinate(self, nao):
2    vectorDirection = -1.0
3    if nao is self.orangeNao:
4      vectorDirection = 1.0
5    # Annahme: Transportlast in rechter Hand !
6    transportDimension = Calculation.getDistance(
7                         self.orangeNao.rightPoint,
8                         self.greyNao.rightPoint)
9    x = math.cos(self.getTargetOrientation() ) *
10       ( transportDimension / 2.0 * vectorDirection )
11   y = math.sin(self.getTargetOrientation() ) *
12       ( transportDimension / 2.0 * vectorDirection )
13   transportCenter = self._wayPoints[self._waypointIndex]
14   return (transportCenter[0] + x, transportCenter[1] + y)
```

Listing 3.5: Planning.getTargetWorldCoordinate

Des Weiteren übernimmt die Planning Klasse die Verwaltung der Wegpunkte. Die evaluatePosition Methode (Listing 3.6) ist verantwortlich für die Prüfung, ob ein Wegpunkt erreicht wurde. Ist dies der Fall, so wird der nächste Wegpunkt gesetzt. Wurden alle Wegpunkte erreicht, wird dies dem System mitgeteilt.

```
def evaluatePosition(self, position):
  if Calculation.getDistance(self._waypoints[self._waypointIndex],
                             position) < self.waypointToleranceInMeter:
    print "Waypoint erreicht: ", self._waypointIndex
    self._waypointIndex += 1
    if self._waypointIndex > len(self._waypoints) - 1:
      self._waypointIndex -= 1
      print "Zielposition erreicht!"
      return True
  return False
```

Listing 3.6: Planning.evaluatePosition

3.6 Bewegungsorganisation

Roboterbewegungen werden mit drei aufeinanderfolgenden Methoden vorbereitet. Zuerst wird beiden Robotern ein zu erreichendes Ziel mitgeteilt. In der zweiten Methode wird das geplante Vorgehen von beiden Robotern miteinander synchronisiert. Die abschließende, dritte Methode gibt den Befehl zum Umsetzen der Bewegung. In der Klasse Maneuver haben die Methoden entsprechend die deskriptiven Namen: 1. assignManeuver, 2. synchronizeManeuver und 3. executeManeuver.

1. Die Zuweisung eines neuen Ziels, zu dem sich die Roboter bewegen sollen, findet in Welt-Koordinaten statt. Damit diese vom Roboter ausgeführt werden können ist zunächst eine Umrechnung auf dessen lokales Koordinatensystem notwendig. Für die Umrechnung mittels trigonometrischer Funktionen wird die Calculation Klassenmethode translateManeuver (Listing 3.7) verwendet.
   Oberste Priorität hat, dass alle Bewegungen innerhalb des lokalen Rotation-Toleranzbereichs ablaufen. Wird diese Invariante verletzt überschreibt ein Korrekturmanöver zum Ausrichten der Orientierung alle vorherigen Befehle.

```
# Uebersetzt die Welt-Koordinaten zu Roboter-Koordinaten
@classmethod
def translateManeuver(self, currCenterPoint, currOrientation,
                      targetWorldCoordinate):
  hypotenuse = self.getDistance(currCenterPoint,
                                targetWorldCoordinate)
  targetOrientation = self.getOrientation(currCenterPoint,
                                          targetWorldCoordinate)
  angleDiff = self.getOrientationDifference(currOrientation,
                                            targetOrientation)
  x = math.cos(angleDiff) * hypotenuse
  y = math.sin(angleDiff) * hypotenuse
  robotCoordinate = (x, y)
  return robotCoordinate
```

Listing 3.7: Calculation.translateManeuver

2. Befindet sich einer der Naos in einem Korrekturmanöver, hat der andere nur die Optionen ebenfalls ein Korrekturmanöver durchzuführen oder zu warten. Die Synchronisierung verhindert alle anderen Befehle. Das die lokale Orientierung des Roboters von zentraler Bedeutung ist führt zu folgenden positiven Effekten:

   a) Bei einer Translationsbewegung[5] der Roboter ist eine Synchronisation weder gewünscht noch zu erreichen, denn der Idealpfad durch den abknickenden Gang, beinhaltet eine kontinuierliche Rotationsbewegung des Gesamtsystem. Der hier fehlende Bewegungsabgleich führt dazu, dass die Roboter gegenseitig unerwünschte Kräfte aufeinander einwirken lassen. Der praktische Effekt, einer identischen Ausrichtung der Roboter ist dabei, dass die Standfestigkeit beim Laufen signifikant erhöht wird. Es treten hauptsächlich Kräfte in, bzw. genau entgegen, der Blickrichtung auf. In diese Richtung liefern die Füße, aufgrund ihrer Abmessungen, hohe Standfestigkeit. Während bei einer seitlichen Belastung ein Nao schnell instabil wird. Zumal sich bei permanenter Bewegung oft nur ein Fuß am Boden befindet, was die seitliche Stabilität weiter verschlechtert.

   b) Eine pseudo starre Ausrichtung des Gesamtsystems ermöglicht erst, dass im gesamten System mit relativ simpler Mathematik für Starrkörper gearbeitet werden kann. Dies Betrifft insbesondere die interne Repräsentation (Abschnitt 3.4) und Bewegungsplanung (Abschnitt 3.5), indirekt aber auch die Computer Vision (Abschnitt

[5] Parallelverschiebung

3.3, Nur durch die Starrkörperannahme wird die Erkennung von vier Entitäten mit nur zwei Farben möglich). Pseudo-starr bedeutet hier, dass das Gesamtsystem der Orientierung der Roboter zueinander, sowie zur Transportlast, versucht einen nicht flexiblen Körper zu emulieren.

c) Zusätzlich entsteht durch die Starrkörperannahme nicht die Gefahr, dass die Roboter in einem so ungünstigen Winkel zueinander stehen, dass ein Roboter seine Bewegung mit der Transportlast blockiert. Es wird somit unmöglich, dass ein Roboter über die zu transportierende Stange stolpert.

Negativer Effekt der Orientierungssynchronisation ist, dass Flexibilität bei der Bewegungsplanung eingebüßt wird. Zudem erhöht sich durch Korrekturmanöver, welche nicht in Translationsbewegungen intrigiert sind, die Zeit um eine bestimmte Distanz zurückzulegen.

3. Bewegungsbefehle werden vom ALMotionProxy an die Naos übermittelt. ALMotionProxy ist Bestandteil vom NAOqi Framework. Neue Befehle überschreiben die Alten, ohne dabei abrupt den Bewegungsfluss zu unterbrechen. Sodass ein in der Ausführung begonnener Schritt immer sauber beendet wird.

3.6.1 Konfiguration

Um die Roboter für den eigentlichen Transport vorzubereiten, bzw. abschließend ihrer Aufgabe zu entbinden, wurden eine Reihe von Konfigurationsscripte geschrieben. Diese Tools erlauben einen schnellen Zugriff auf einfache vordefinierte Bewegungsabläufe. Routineaufgaben, die auf diese Weise zusammengefasst werden, beinhalten:

1. Aufrichten und Hinsetzen des Roboters
2. Starrheitskonfiguration einzelner Gelenke
3. Öffnen und Schließen der Finger zum Greifen der Transportstange

Des Weiteren können einzelne Aspekte von Bewegungsabläufen, mit diesen Tools genauer untersucht werden. Konkret wurden Effekte einer dynamisch veränderbaren Bewegungsgeschwindigkeit untersucht. Sowie die empirische Ermittlung eines Faktors, welcher dass Rutschen in einer Rotationsbewegung ausgleicht.

4 Evaluation

Die Evaluation setzt sich mit den praktischen Erfahrungen auseinander, die bei den Laborversuchen gemacht worden sind.

4.1 Framework Interaktion

Im Folgenden werden die Frameworks, die im Rahmen dieser Thesis verwendet worden sind, auf ihren Auswahlhintergrund und ihrer Kompatibilität zueinander beleuchtet.

OMPL und SimpleCV konnten nicht auf einem gemeinsamen Betriebssystem betrieben werden. OMPL ließ sich problemlos auf Ubuntu 12.04 Systemen einrichten. Funktionierte jedoch nicht auf Windows 7 und Windows Vista Plattformen.
SimpleCV verhielt sich auf Linux und Windows schwerfällig. Auf Linux gab es nicht gelöste Probleme mit dem Kameratreiber. Die Windows Version war ohne eine Veränderung im Framework-Quellcode nicht lauffähig (Im Projektordner findet sich dazu die Datei 'SimpleCV fix after install' mit weiteren Erläuterungen).
Die Betriebssystemseparierung dieser zwei Kernkomponenten führte dazu, dass die Bewegungsplanung (Abschnitt 3.5) nicht wie geplant zur Laufzeit mit der internen Repräsentation (Abschnitt 3.4) zusammenarbeiten kann. Als Lösung wird stattdessen zuerst mit dem Windows7 System, als Kernkomponente für SimpleCV, die Umgebung, sowie Start- und Ziel-Position, der Roboter erfasst. Für den Datenaustausch wurde das planningKernelInput.py Skript geschrieben. Nach einem Systemwechsel zu Ubuntu, berechnet OMPL im planningKernel dazu einen möglichen (Starrkörper-)Pfad. Aufgrund der Notwenigkeit einer Computer Vision (Abschnitt 3.3) während der Transportaufgabe, wird hierfür wieder zum Windows System gewechselt. Dort wird der Pfad abschließend manuell als Parameter des Startskripts main.py eingetragen. Eine automatisierte Parameterübergabe zwischen den Betriebssystemen mittels Virtueller Maschine wurde nicht implementiert.

Als weiteres Bewegungsplanung-Framework wurde OpenRAVE von Rosen Diankov (2013) getestet. Der von OpenRAVE zu Verfügung gestellte Funktionsumfang ist mit OMPL vergleichbar, aber auch OpenRAVE ließ sich nur in der Linuxumgebung einsetzen.
Die OpenCV Grundstruktur lief zwar grundsätzlich auf Ubuntu und Windows 7, allerdings war das Problem der Kameratreiber auf beiden Systemen nicht zu überwinden.

Zur Visualisierung der Planungskomponente wurden Funktionen von matplotlib verwendet. Zuvor war geplant, für die gleiche Aufgabe, dass von Gillies u. a. (2014) entwickelte Shapely einzusetzen. Beide Bibliotheken bieten einen ähnlichen Funktionsumfang, aber Shapely wurde anfangs favorisiert da es:

1. Eine größere Auswahl an vordefinierten visuellen Effekten anbietet.
2. Der Code beim Arbeiten mit Polygonen besser lesbar ist (subjektiv). In matplotlib wird das Konstrukt Path dafür verwendet.

Shapely wurde nicht benutzt, aufgrund systemübergreifender Problemen einen Output-Stream zu generieren.

Ohne Probleme bei der Einrichtung funktionierten Python 2.7, dass NumPy Package für wissenschaftliches Rechnen von Hugunin (2013), sowie das NAOqi Framework in der Version pynaoqi-1.14.5.

4.2 Hardware Beschränkungen

An dieser Stelle werden Schwierigkeiten der im Labor verwendeten Hardware thematisiert und mögliche Lösungen dazu vorgestellt.

Die Fußsohlen der Naos bestehen aus glattem Kunststoff. Die ohnehin schlechte Haftung wird durch einen zusätzlich glatten Untergrund so weit reduziert, dass der Nao nach Vollendung eines Schrittes, durch die auftretende Beschleunigung Distanzen von mehr als 1cm gleitet. Daraus resultiert, dass die Genauigkeit der Ansteuerung erheblich sinkt. Ein weit größeres Problem ergibt sich, wenn zwei Nao versuchen gemeinsam ein Objekt zu transportieren und ihre Bewegungen nicht synchronisiert sind. Anschaulich wird dies in der Versuchsdokumentation, Video '3 - Nao Robot Crash' gezeigt. In einem ersten Versuch wurden stark haftende Gummimatten unter die Sohle geklebt, wie in Abbildung 4.2 Fußsohle zu sehen ist. Dabei musste beachtet werden, dass die 4 Drucksensoren pro Fußsohle, so abgedeckt werden, dass

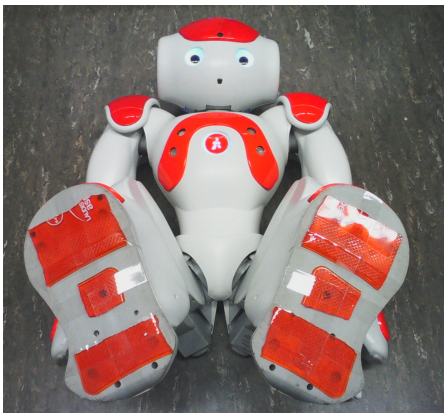

Abbildung 4.1: Fußsohle

sie weiterhin valide Signale liefern können. Es stellte sich aber bei den Testläufen heraus, dass ein gewisses Gleiten bei Bodenkontakt eine notwendige Bedingung für einen stabilen Gang ist. Nachdem die Haftfläche der Gummimatten durch überkleben mit Klebestreifen auf ein Minimum reduziert worden war, zeigte sich ein stabiler Gang. Trotzdem wurde die Gummimatten-Lösung verworfen, da sich schnell Staub an die Matte heftete und sich nach wenigen Schritten jegliche Haftungsverbesserung auflöste. Die wiederholt langsame Abnahme der Haftung, machte den Roboter in weiteren Versuchen mit der Gummisohle zu unberechenbar und ein staubfreier Reinraum stand nicht zur Verfügung.
In Folgeversuchen wurde eine Schnur an den Roboter angebracht. Diese wurde leicht auf Spannung gehalten, was ein Rutschen verhinderte. Dabei wurde der Roboter lediglich stabilisiert und nicht gezogen, was bei dem Nao Gewicht von 5kg eine wesentlich größere Kraft erfordert hätte. Zur Überprüfung dieser Behauptung können die zielgerichteten Beinbewegungen im Video '4 - Cooperative Pole Transport by Two Nao Robots' betrachtet werden.

Für längere Laborsitzungen mit dem Nao ist eine Akkuladung nicht ausgelegt. In den Phasen in denen der Nao nicht aktiv ist wird folglich das Ladekabel angeschlossen. Diese gehäufte Beanspruchung der Steckverbindung ist eine schwer zugängliche Schwachstelle. Der Kontaktpin, in Form einer empfindlichen Vergabelung, wird so zusammengedrückt, dass die elektrische Leitfähigkeit unterbrochen wird. Die Reparatur musste dafür am ausgebauten Akku stattfinden, Abbildung 4.2 Akku.

Konnektivitätsprobleme traten ebenfalls beim Kamerasystem auf, sowohl die interne Nao Kamera, als auch die QuickCam waren von kurzen Bildverlusten betroffen. Dies Verschlech-

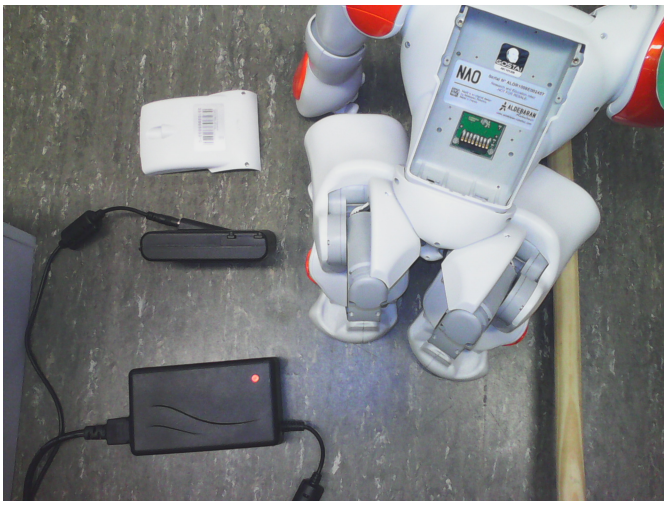

Abbildung 4.2: Akku

terte die Qualität der Versuchsdokumentation. Bei längerem Bildausfall mussten laufende Transporte abgebrochen werden, um Unfälle aufgrund fehlender Kontrollmechanismen zu verhindern. Verantwortlich für die QuickCam-Ausfälle waren Verschleißerscheinungen an der USB Verbindung. Welche Gründe zu Unterbrechungen, in von den Nao-Kameras aufgezeichneten Videos, führten konnte nicht geklärt werden.

4.3 Kamera Kalibrierung

Um nützliche Daten aus dem Kamerainput ableiten zu können, ist eine vorherige Kalibrierung notwendig.
Das Definieren von Pixel in Längeneinheit funktioniert sehr robust. Es wird lediglich manuell ein Abstand zwischen zwei Punkten ermittelt, dessen Distanz in Metern bekannt ist. Bei dieser Methode wird die Krümmung durch die Kameralinse nicht berücksichtigt. Dies führt zu Ungenauigkeiten in den Randbereichen des Bildes. Da der kritische Bereich, die Biegung im Gang, zentriert wird, sind die negativen Effekte allerdings vernachlässigbar. Ein ausbreiten eines Kalibrierung-Schachbrettmusters, mit evtl. partiellen Abbau der Versuchswände, wird dadurch obsolet. Die extrem kurze Kalibrierungszeit, von ca. 10 Sekunden, in dem verwendeten Verfahren ermöglicht einen flexibleren Umgang mit der Kameraposition.

Eine robuste Farberkennung stellte im Sonnen gefluteten Labor eine Herausforderung dar. Auch mit vorgezogenen Gardinen änderte sich der Beleuchtungswert durch Lichtschwankungen von außen so stark, dass die Kalibrierung nach wenigen Minuten wiederholt werden musste. Eine Verbesserung der Situation brachte das Abkleben der Fenster mit Lichtundurchlässiger Folie, siehe Abbildung 4.3 Labor.

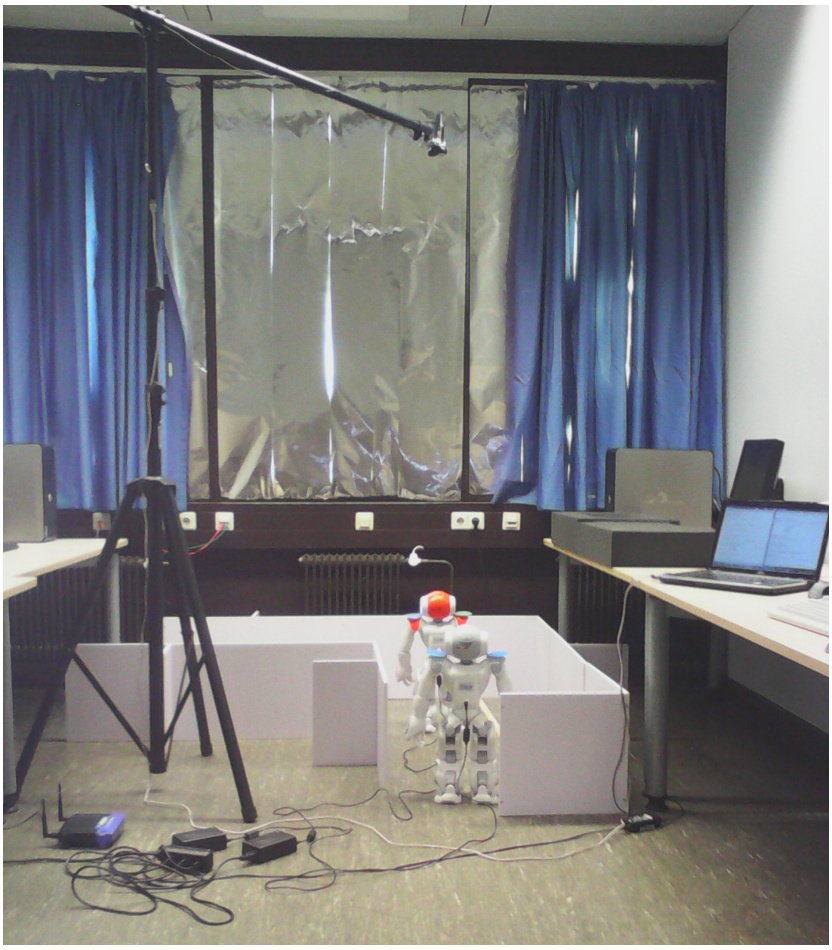

Abbildung 4.3: Labor

Die farbige LED Beleuchtung am Kopf der Naos trug ebenfalls zur fehlerhaften Farberkennung bei. Darum wurde ein Skript in den Transport-Initialisierungsprozess integriert, welches einzelne LEDs deaktiviert.
Ein weiteres Problem bereitete die unterschiedliche Helligkeit, wenn die Naos sich den Lichtkegeln der Deckenbeleuchtung näherten oder sich davon entfernten. Da ein Austausch der Lichtquelle die Mittel dieser Bachelorarbeit überstiegen hätte, wurde stattdessen versucht eine Farbe zu finden, die mit einem statischen Grenzwert zuverlässig in den unterschiedlichen Beleuchtungssituationen zu identifizieren war. Es wurden 30 verschiedenfarbige Markierungen gedruckt, die auf ihre Identifizierbarkeit hin untersucht wurden.
Als Resultat dieser Maßnahmen wurde die Farberkennung sehr zuverlässig. In dem Video '4 - Cooperative Pole Transport by Two Nao Robots' zur Versuchsdokumentation ist zu sehen, dass die Farbmarkierungen nur sehr selten, für einen Sekundenbruchteil, aus dem Fokus verloren wurden (Wenn die gelben Markierungskreuze verschwinden).

4.4 Fazit

Während der Laborversuche wurde deutlich, dass es einen erheblichen Mehraufwand bedeutet, bei Entwicklung sowohl Software- als auch Hardware-Aspekte zu berücksichtigen.

Wie dem Abschnitt 4.1 Framework Interaktion zu entnehmen ist, gestaltete sich die Integration und das Zusammenspiel der verschiedenen Frameworks als komplexe Herausforderung. Die Verwendung von Frameworks ist aber notwendig, wenn auf einem hohen Softwareabstraktionslevel gearbeitet werden soll. Um den Konfigurationsaufwand zu begrenzen und Kompatibilitätsprobleme zu vermeiden, empfiehlt sich die Verwendung von Systemlösungen. Eine Möglichkeit wäre die Bibliothekssammlung des Robot Operating System (ROS) von Willow Garage (2014) zu nutzen.
Was allerdings dazu führen kann, dass Bibliotheken, welche nicht Teil der Systemlösung sind, erhebliche Integrationsprobleme anhaften.

5 Schluss

5.1 Zusammenfassung

Im Rahmen dieser Bachelorthesis wurde kollaborativ, von humanoiden Robotern, der Transport einer Stange durchgeführt. Die Transportsportaufgabe fand in einem rechtwinklig abknickenden, schmalen Gang statt. Es wurden die einzelnen Aspekte, die zum Lösen der komplexen Aufgabe notwendig sind, untersucht. Ausgehend von einer einfachen Versuchskonfiguration, wurde diese sukzessiv, für immer komplexere Teilaufgaben, erweitert. In den Videos der Versuchsdokumentation kann diese Evolution nachvollzogen werden.
Anfangs wurde eine grundlegende Ansteuerung der Nao Aktorik implementiert, zu sehen in '1 - Nao Robot Initial Test Walk'. Durch Integration der Computer Vision wurden komplexere Bewegungsmuster möglich und Manöver mit einer Transportlast konnten durchgeführt werden, zu sehen in '2 - Pole Transport by Single Nao Robot'. Der nächste Schritt, dass Aufbauen einer internen Repräsentation, erlaubte die Integration eines zweiten Nao Roboters für den Transport. Beiden Robotern konnten individuelle Rollen zum Transport zugeordnet werden, wobei erst die Bewegungsorganisation eine erfolgreiche Kollaboration der Roboter ermöglichte. Durch ein gelungenes Zusammenspiel von Bewegungsinvarianten konnte eine Synchronisation hergestellt werden, welche die Roboter befähigte gemeinschaftlich alle zum Transport nötigen Manöver auszuführen, anschaulich in '4 - Cooperative Pole Transport by Two Nao Robots' nachzuvollziehen. Als letzter Aspekt wurde die Bewegungsplanung hinzugefügt. Ein Zielpfad musste nicht länger manuell definiert werden, sondern wurde vom System berechnet. Die Beispiele aus '5 - Basic Motion Planning and Scalability' zeigen die Initialisierungsschritte und gewähren Einblick in den daraus erzeugten Datensatz eines Lösungspfades 'path_simple'. Diese Daten werden durch die implementierte Visualisierung in '3 - Simple' veranschaulicht, und darüber hinaus wird durch das Beispiel '4 - Complex' auf die Skalierungsmöglichkeiten hingewiesen.

Die komponentenbasierte Softwareentwicklung sorgte für Flexibilität in der Entwicklung. Dies ermöglichte, die Versuchsumgebung evolutionär zu entwickeln. Im Entwicklungsprozess der Software konnte somit die Erweiterung, für komplexer werdende Teilaufgaben und Versuche, auf den bereits bestehenden Grundlagen fortgesetzt werden.
Die Robustheit des Systems wurde durch jedes Hinzufügen einer Komponente erhöht. Dennoch kann wenn veränderte Anforderungen bestehen jede Komponente ausgetauscht werden, um die Skalierbarkeit und ein offenes Softwaredesign zu gewährleisten.

Als abschließendes Resultat lässt sich feststellen, dass nicht nur alle Anforderungen der Aufgabenstellung erfüllt worden sind, sondern auch ein solides Grundgerüst für eine Vertiefung der Aufgabenstellung bereitgestellt wird.

5.2 Ausblick

Die entwickelte Software besitzt wesentliche Eigenschaften, um eine hohe Skalierbarkeit zu gewährleisten. Signifikante Erweiterungen, für zukünftige Arbeiten, zu implementieren ist damit unter geringem Aufwand möglich. Im Folgenden werden dazu verschiedene Vorschläge erläutert.

- Die OMPL-basierte Bewegungsplanungskomponente sollte soweit in die Software des Transportsystems integriert werden, dass eine Pfad-Planung zur Laufzeit ermöglicht wird.

- Der Computer Vision kann die Fähigkeit zur Detektion der Umgebungshindernisse hinzugefügt werden. Es wäre damit möglich die Laborumgebung zur Laufzeit zu verändern und die Roboter würden in Echtzeit auf diese Änderung reagieren können. Zusätzlich könnte auch die Erfassung von weiteren mobilen Objekten, wie Robotern, implementiert werden. Wenn diese als Hindernis definiert werden, ist es möglich so anderen Objekten zur Laufzeit auszuweichen.
  Zur Realisierung müsste nur die Computer Vision erweitert werden. Das restliche System, sofern die Bewegungsplanung zur Laufzeit integriert wird, ist bereits für diese Anforderungen ausgelegt. In Listing 3.1 ist zu sehen wie der Controller die Hindernisdaten weiterleitet. Diese Daten wurden nicht zur Laufzeit genutzt, gewährleisten aber eine hohe Erweiterungsflexibilität.

- Die Komplexität des Versuchsaufbaus könnte erhöht werden. Anstelle einer einfachen Abknickung könnte ein komplettes Labyrinth durchlaufen werden. Hierfür wären keinerlei Softwareänderungen erforderlich, da das System bereits den nötigen Anforderungen entspricht. Es müsste lediglich eine größere Laborumgebung aufgebaut werden und eine Kamera mit erhöhtem Überblick installiert werden.
- Es sind erweiterte Laborexperimente mit ineinander gekapselten Hindernissen denkbar. Für einen solchen Aufbau ist die polygonbasierte Modellierung des Starrkörpers, sowie der Umgebungsdetails, nicht ausreichend. Die Modellierung müsste erweitert bzw. durch einen allgemeingültigen Ansatz ausgetauscht werden.
- Ein Versuch mit mehr als zwei Naos könnte entwickelt werden, in dem z.B. vier Naos eine Last gemeinsam transportieren. Für die Skalierbarkeit der Roboteranzahl ist das System grundsätzlich nicht konzipiert worden und es müssten fast alle Komponenten restrukturiert werden. Dennoch wäre der Aufwand überschaubar, da jeweils nur marginale Änderungen von Nöten sind.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[Abe 2014] Abe, Shinzo: *Robot Revolution Growth Strategy. In: Jiji.* September 2014

[Aldebaran Robotics 2014a] Aldebaran Robotics: *Nao - Construction.* November 2014. – URL http://doc.aldebaran.com/2-1/family/robots/dimensions_robot.html. – Abruf: 2014-11-03

[Aldebaran Robotics 2014b] Aldebaran Robotics: *Nao Freiheitsgrade.* 2014. – URL http://doc.aldebaran.com/2-1/_images/hardware_jointname.jpg. – Abruf: 2014-10-21

[Aldebaran Robotics 2014c] Aldebaran Robotics: *NAOqi Framework.* Oktober 2014. – URL http://doc.aldebaran.com/1-14/dev/naoqi/index.html. – Abruf: 2014-10-18

[Aldebaran Robotics 2014d] Aldebaran Robotics: *NAOqi Struktur.* 2014. – URL http://doc.aldebaran.com/1-14/_images/broker-modules-methods.png. – Abruf: 2014-10-21

[Aurenhammer 1991] Aurenhammer, Franz: Voronoi Diagrams&Mdash;a Survey of a Fundamental Geometric Data Structure. In: *ACM Comput. Surv.* 23 (1991), September, Nr. 3, S. 345–405. – URL http://doi.acm.org/10.1145/116873.116880. – ISSN 0360-0300

[Crevier und Lepage 1997] Crevier, Daniel ; Lepage, Richard: Knowledge-Based Image Understanding Systems: A Survey. In: *Computer Vision and Image Understanding* 67 (1997), Nr. 2, S. 161 – 185. – URL http://www.sciencedirect.com/science/article/pii/S1077314296905202. – ISSN 1077-3142

[Cuauhtemoc Carbajal 2012] Cuauhtemoc Carbajal: *Computer Vision Using SimpleCV and The Raspberry PI - ITESM CEM.* 2012. – URL http://homepage.cem.itesm.mx/carbajal/EmbeddedSystems/SLIDES/Computer%

20Vision/Computer%20Vision%20using%20SimpleCV%20and%20the%20Raspberry%20Pi.pdf. – Abruf: 2014-10-14

[Daniela Rus 2011] Daniela Rus: *Robotics systems and science - Configuration Space and Motion Planning*. 2011. – URL http://courses.csail.mit.edu/6.141/spring2011/pub/lectures/Lec09-MotionPlanning-II.pdf. – Abruf: 2014-10-20

[Defense Advanced Research Projects Agency 2014] Defense Advanced Research Projects Agency: *DARPA Robotics Challenge*. 2014. – URL www.theroboticschallenge.org/. – Abruf: 2014-10-28

[Dijkstra 1959] Dijkstra, E. W.: A Note on Two Problems in Connexion with Graphs. In: *NUMERISCHE MATHEMATIK* 1 (1959), Nr. 1, S. 269–271

[Dijkstra 1982] Dijkstra, Edsger W.: On the role of scientific thought (EWD447). In: *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982, S. 60–66

[Eckerson 1995] Eckerson, Wayne W.: Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. In: *Open Information Systems* 10 (1995), Nr. 1

[Eric W Weisstein 2014] Eric W Weisstein: *Moving Ladder Problem*. 2014. – URL http://mathworld.wolfram.com/images/eps-gif/MovingLadderProblem_800.gif. – Abruf: 2014-10-08

[Erion Plaku 2014] Erion Plaku: *Introduction to Robotics - Sampling-Based Motion Planning*. November 2014. – URL http://faculty.cua.edu/plaku/teaching/LecRoboMPSampling.pdf. – Abruf: 2014-10-05

[Finch 2003] Finch, S. R.: Moving Sofa Constant. In: *Mathematical Constants*. Cambridge University Press, 2003, S. 519–523

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0-201-63361-2

[Gillies u. a. 2014] Gillies, Sean ; Bierbaum, Aron ; Lautaportti, Kai ; Tonnhofer, Oliver: *Shapely*. Oktober 2014. – URL https://pypi.python.org/pypi/Shapely. – Abruf: 2014-10-28

[Hugunin 2013] Hugunin, Jim: *NumPy*. 2013. – URL http://www.numpy.org/. – Abruf: 2014-08-11

[Hunter u. a. 2014] Hunter, John ; Dale, Darren ; Firing, Eric ; Droettboom, Michael: *matplotlib*. Oktober 2014. – URL http://matplotlib.org/. – Abruf: 2014-10-09

[Intel Corporation, Willow Garage, Itseez 2014] Intel Corporation, Willow Garage, Itseez: *OpenCV*. August 2014. – URL http://opencv.org/. – Abruf: 2014-09-20

[Jake Vanderplas 2012] Jake Vanderplas: *Matplotlib Animation Tutorial*. August 2012. – URL https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/. – Abruf: 2014-08-05

[Kavraki u. a. 1996] Kavraki, L.E. ; Svestka, P. ; Latombe, J.-C. ; Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: *Robotics and Automation, IEEE Transactions on* 12 (1996), Aug, Nr. 4, S. 566–580. – ISSN 1042-296X

[Kay 1996] Kay, Alan C.: History of Programming languages—II. New York, NY, USA : ACM, 1996, Kap. The Early History of Smalltalk, S. 511–598. – URL http://doi.acm.org/10.1145/234286.1057828. – ISBN 0-201-89502-1

[Kuffner und LaValle 2000] Kuffner, J.J. ; LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on* Bd. 2, 2000, S. 995–1001 vol.2. – ISSN 1050-4729

[Lavalle 1998] Lavalle, Steven M.: Rapidly-Exploring Random Trees: A New Tool for Path Planning / Iowa State University Computer Science Deptartment. Oct 1998. – Forschungsbericht

[Leven und Sharir 1985] Leven, Daniel ; Sharir, Micha: An Efficient and Simple Motion Planning Algorithm for a Ladder Moving in Two-dimensional Space Amidst Polygonal Barriers (Extended Abstract). In: *Proceedings of the First Annual Symposium on Computational Geometry*. New York, NY, USA : ACM, 1985 (SCG '85), S. 221–227. – URL http://doi.acm.org/10.1145/323233.323262. – ISBN 0-89791-163-6

[McIlroy 1968] McIlroy, M. D.: Mass-produced software components. In: *Proc. NATO Conf. on Software Engineering, Garmisch, Germany* (1968)

[Meyer 1992] Meyer, Bertrand: Applying "Design by Contract". In: *Computer* 25 (1992), Oktober, Nr. 10, S. 40–51. – URL http://dx.doi.org/10.1109/2.161279. – ISSN 0018-9162

[National Instruments 2011] National Instruments: *Voronoi Diagramm*. Juni 2011. – URL http://zone.ni.com/images/reference/de-XX/help/371361L-0113/noloc_eps_voronoi_diagram.gif. – Abruf: 2014-10-23

[Pimentel u. a. 2002] Pimentel, Bruno S. ; Pereira, Guilherme A. S. ; Campos, Mario M. F. M.: On the Development of Cooperative Behavior-based Mobile Manipulators. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. New York, NY, USA : ACM, 2002 (AAMAS '02), S. 234–239. – URL http://doi.acm.org/10.1145/544741.544799. – ISBN 1-58113-480-0

[Python Software Foundation 2014] Python Software Foundation: *Python*. November 2014. – URL https://www.python.org/. – Abruf: 2014-11-04

[Rice University - Physical and Biological Computing Group 2014] Rice University - Physical and Biological Computing Group: *The Open Motion Planning Library*. Oktober 2014. – URL http://ompl.kavrakilab.org/l. – Abruf: 2014-10-27

[Rosen Diankov 2013] Rosen Diankov: *Open Robotics Automation Virtual Environment*. März 2013. – URL http://openrave.org/docs/latest_stable/. – Abruf: 2014-08-02

[Sight Machine Inc. 2014a] Sight Machine Inc.: *SimpleCV*. November 2014. – URL http://simplecv.org/. – Abruf: 2014-11-04

[Sight Machine Inc. 2014b] Sight Machine Inc.: *SimpleCV - Farbabstand zu Gelb*. 2014. – URL http://tutorial.simplecv.org/en/latest/_images/parking-car-colordistance.png. – Abruf: 2014-11-03

[Sight Machine Inc. 2014c] Sight Machine Inc.: *SimpleCV - Reduziertes Bild fuer den Zielfarbraum*. 2014. – URL http://tutorial.simplecv.org/en/latest/_images/parking-car-yellow.png. – Abruf: 2014-11-03

[Sight Machine Inc. 2014d] Sight Machine Inc.: *SimpleCV - Unbearbeitetes Bild*. 2014. – URL http://tutorial.simplecv.org/en/latest/_images/parking-car-only.png. – Abruf: 2014-11-03

[Steven M LaValle 2006] Steven M LaValle: *Rapidly Exploring Random Tree*. 2006. – URL http://msl.cs.uiuc.edu/planning/img2043.gif. – Abruf: 2014-10-23

[Willow Garage 2014] Willow Garage: *Robot Operating System*. November 2014. – URL http://www.ros.org/. – Abruf: 2014-11-06

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 17. November 2014 Aaron Gornott