Diplomarbeit

Planung und Ausführung von alltäglichen zweihändigen Manipulationsaufgaben

Diplomarbeit

Daniel Leidner

DLR

Institut für Robotik und Mechatronik	BJ.: 2010
	IB.Nr.: 515-10-01

DIPLOMARBEIT

PLANUNG UND AUSFÜHRUNG VON ALLTÄGLICHN ZWEIHÄNDIGEN MANIPULATIONSAUFGABEN

Freigabe:	Der Bearbeiter:	Unterschriften
	Daniel Leidner	_____
	Betreuer:	
	Franziska Zacharias	_____
	Der Institutsdirektor	
	Prof. Dr. G. Hirzinger	_____

Dieser Bericht enthält 101 Seiten, 35 Abbildungen und 14 Tabellen

Ort: Oberpfaffenhofen	Datum:	Bearbeiter:	Zeichen:

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
DLR

hochschule mannheim

Diplomarbeit

Planung und Ausführung von alltäglichen zweihändigen Manipulationsaufgaben

Daniel Leidner

25. Februar 2010

Eingereicht am 25. Februar 2010
von Daniel Leidner

Studiengang technische Informatik
Hochschule Mannheim

Prüfer: Prof. Dr. Thomas Ihme,
Dipl.-Inform. Franziska Zacharias

Betreuer: Dipl.-Inform. Franziska Zacharias,
Florian Schmidt, MSc

Institut für Robotik und Mechatronik
Deutsches Zentrum für Luft- und Raumfahrt e.V.

Zusammenfassung

Eine der wichtigsten Fähigkeiten für zukünftige, humanoide Serviceroboter ist die autonome Manipulation von Objekten in verschiedensten Umgebungen. Besonders das Zusammenspiel zweier Arme hat sich in der Natur des Menschen als äußerst hilfreich erwiesen. Diese Arbeit befasst sich daher mit der Planung und Ausführung von alltäglichen, zweiarmigen Manipulationsaufgaben, speziell mit der Bewegungsplanung. Dabei werden zunächst Bewegungsabläufe von einarmigen Aktionen untersucht. Das Hauptaugenmerk liegt allerdings auf kombinierten Abläufen beider Arme. Die Arme müssen dabei ihre jeweilige Zielkonfiguration möglichst zeitnah und ohne Kollisionen erreichen. Es wird gezeigt, wie die Unterteilung der Arbeitsraume in verschiedene, disjunkte Bereiche die Planungszeit positiv beeinflusst. Die Komplexität des Ablaufs kann dadurch reduziert werden.

Abstract

One of the most important skills for future humanoid service robots is the autonomous manipulation of objects in different environments. Especially the cooperation of two arms turned out as extremely helpful in the nature of man. This work is therefore concerned with the planning and execution of everyday two-armed manipulation tasks, more specifically with the aspect of path planning. The motion planning is first studied in one-armed actions. The focus, however, is on the combined operations of both arms. The arms have to reach their respective target configuration as soon as possible and without collisions. It is shown that the division of the workspaces into different, disjunct areas positively affects the planning time. The complexity of the task can be reduced in this way.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit

„Planung und Ausführung von alltäglichen zweihändigen Manipulationsaufgaben“

selbständig und ohne Benutzug anderer als die angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht worden. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Oberpfaffenhofen, den 25. Februar 2010

Daniel Leidner

Danksagung

An dieser Stelle möchte ich mich bei Prof. Dr. Thomas Ihme für die Betreuung von Seiten der Hochschule bedanken. Von Seiten des Instituts für Robotik des Deutschen Zentrums für Luft- und Raumfahrt möchte ich besonders zwei Personen danken: Franziska Zacharias für die engagierte administrative und fachliche Unterstützung und Florian Schmidt für die unzähligen technischen Hilfestellungen. Des Weiteren möchte ich mich bei meinen Eltern, für die finanzielle Unterstützung während des Studiums und meiner Freundin, die mich während meiner Zeit in München begleitet hat, bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Roboter der nicht allzu fernen Zukunft werden ein Teil unseres täglichen Lebens werden [33]. Als unsere Bediensteten und Assistenten werden sie uns als helfende Hand zur Seite stehen. In Bereichen der Forst- und Agrarwirtschaft können sie uns unterstützen, bei Arbeiten in gefährlichen oder lebensbedrohlichen Umgebungen möglicherweise sogar ganz ersetzen. Beim Einsatz in Katastrophengebieten oder Untertage erreichen sie unter Umständen für Menschen nur schwer zugängliche Orte. Nicht zu vergessen ist natürlich der Haushaltsroboter für Jedermann. So unterschiedlich und futuristisch diese Szenarien sein mögen, werden die meisten der dort eingesetzten Roboter eines gemeinsam haben: *Die Fähigkeit der autonomen Manipulation.* Mit „Rollin' Justin" wird bereits seit einigen Jahren ein humanoider Roboter entwickelt, der einen Schritt in diese Richtung geht. Mit seinen beiden antropomorphen Armen hat er die Möglichkeit, Aufgaben parallel zu erledigen, oder beide Manipulatoren zum Erreichen eines höheren Gesamtziels zu verwenden.

Abbildung 1.1: Zukunftsvision: Der humanoide Robter Justin unter Menschen

1.1 Motivation

Wie viele andere Forschungsbereiche der Robotik ist auch die autonome Manipulation noch ein weitgehend offenes Kapitel. Leistungsstarke, ausgeklügelte Roboterarme und -hände wurden in den letzten Jahrzehnten entwickelt. Der enorme Vorteil, Bewegungsabläufe mehrerer Arme gleichzeitig zu koordinieren, blieb jedoch bisher ein Randthema. Bereits durch den Einsatz von zwei Manipulatoren kann der Arbeitsraum eines Roboters nahezu verdoppelt werden. Neben erhöhter Geschwindigkeit und Tragkraft ist ein Vorteil besonders hervorzuheben: Gelingt es, die Arme derart geschickt aufeinander abzustimmen, dass diese sich nicht beeinträchtigen sondern fördern, können wesentlich komplexere Aufgaben gelöst werden. Die Entwicklung eines lernfähigen, kognitiven humanoiden Robotersystems ist einerseits ein erstrebenswerter Anreiz, andererseits jedoch mit einer Vielzahl aufwendiger Problemstellungen verbunden. Die autonome zweiarmige Manipulation ist eine große Herausforderung, die eine detaillierte Kenntnis über die Umwelt des Roboters erfordert. Die Kombination mehrerer Strategien und die Einhaltung verschiedener Restriktionen erfordern neue Konzepte für die Interaktion mit Robotern. Für die Arbeit in den unstrukturierten Umgebungen der echten Welt benötigen Serviceroboter ein hohes Maß an Selbstständigkeit.

1.2 Zielsetzung

Im Rahmen dieser Diplomarbeit sollen alltägliche zweihändige Manipulationsaufgaben geplant und ausgeführt werden. Als Basis dafür dient die Analyse verschiedener Pfadplanungsstrategien. Potential Field Guided Path Planning, Probabilistc Roadmaps sowie Rapidly Exploring Random Trees stehen als mögliche Kandidaten zur Auswahl. Verschiedene Ansätze der zweiarmigen Manipulation müssen für den Einsatz auf „Rollin' Justin“ untersucht werden. Ziel ist die Adaption bereits bestehender Planungsmodule auf verschiedenen Ebenen an das vorhandene System. Ferner ist dabei die Planungs- und Simulationsumgebung OpenRAVE als zukünftige Entwicklungsumgebung für Aufgaben- und Pfadplanung zu evaluieren. Dazu werden zunächst einarmige Szenarien zur Bewertung von einfachen Trajektorien eingesetzt. Späterhin sollen Planungen von Bewegungsabläufen für zwei Arme in dieser Umgebung stattfinden. Der kooperierende Ablauf beider Bahnen ist dabei von großer Bedeutung. Beide Manipulatoren müssen unabhängig voneinander Unteraufgaben für das Erreichen eines höheren Gesamtziels lösen. Es wird erwartet, dass dadurch deutlich schnellere Planungs- und Ausführungszeiten entstehen, als dies üblicherweise bei komplexen Zweiarmsystemen der Fall ist. Es werden zunächst mehrere Testszenarien in der Simulation implementiert und anschließend auf dem realen Roboter überprüft. Zu den Anforderungen zählt in erster Linie das Erreichen einer möglichst kurzen Planungs- und Ausführungszeit. Allgemein können auch mit Planern, die dem heutigen Stand der Technik entsprechen, nicht immer instantan Pfade generiert werden. Allerdings kann durch geschicktes Kaskadieren der Unteraufgaben der Ablauf noch weiter beschleunigt werden.

2 Grundlagen und der aktuelle Stand der Forschung

Zu Beginn der Arbeit werden einige Grundlagen erläutert, die für die zweiarmige Manipulation unabdingbar sind. In Hand damit gehen verschiedene Algorithmen zur Pfadplanung und bisher untersuchte Ansätze der kooperativen Manipulation.

2.1 Grundlagen der Robotik

Bevor auf den aktuellen Stand der Technik näher eingegangen werden kann, müssen einige grundliegende Begriffe der Robotik erläutert werden. Darunter fallen in erster Linie die Begriffe *Manipulation* sowie *Pfadplanung.*

2.1.1 Koordinatensysteme und Transformationen

Die Position eines Roboters lässt sich in mehreren Koordinantensystemen bestimmen. Das Bekannteste dabei ist das kartesische Koordinatensystem. Mittels x, y und z-Koordinaten lässt sich ein Punkt im Raum definieren. Die Orientierung eines Objektes kann durch Rotationen um dessen Achsen dargestellt werden. Eine so beschriebene Pose reicht aus, um die genaue Lage des *Tool Center Points (TCP)*, zum Beispiel des Mittelpunkts des Endeffektors, im Raum zu bestimmen. Allerdings können damit nur begrenzt Schlüsse auf die Gelenke des Manipulators gezogen werden. Der *Konfigurationsraum (C_{space})* wird definiert durch die Menge aller realisierbaren Gelenkwinkelstellungen eines Roboters [28]. Der Konfigurationsraum besteht bei einem Roboter mit n Freiheitsgraden *(degrees of freedom, DOF)* aus n Dimensionen. Die Pose des Endeffektors kann, ausgehend von den bekannten Gelenkwinkeln, mit der Vorwärtskinematik des Roboters berechnet werden [32]. Die Inverskinematik bildet das Gegenstück zur Vorwärtskinematik. Mit ihr lassen sich Posen des Endeffektors in Gelenkwinkelstellungen des Roboters überführen. Dies wird sehr häufig gebraucht, da meist bekannt ist *wo* ein Objekt gegriffen werden soll, aber nicht mit welcher Stellung seiner Gelenke der Roboter dorthin gelangt.

2.1.2 Manipulationsplanung und Pfadplanung

Manipulationsplanung beschäftigt sich mit der Frage, wie Objekte verschiedenster Art von einem Roboter im Raum bewegt werden können. Dabei bezieht sich

das Wort *Manipulation* auf die bewusste Veränderung der Umgebung des Roboters durch Bewegung der umliegenden Objekte. Eine Manipulation ist dabei eine Aneinanderreihung von mehreren Bewegungsabläufen zum Lösen einer Manipulationsaufgabe. Bewegungen oder auch Pfade, die zu einem Objekt hinführen, werden als Transitpfade bezeichnet. Pfade, die ein Objekt bewegen, werden Transferpfade genannt. Die Planung des Ablaufes heißt im Forschungsbereich der künstlichen Intelligenz Aufgabenplanung und die Bewegungsplanung Pfadplanung [35]. Die bei der Pfadplanung gewonnenen Pfade werden als Trajektorien bezeichnet und bestehen aus mehreren einzelnen Konfigurationen des Roboters. Ziel der Pfadplanung ist es, einen realisierbaren, nicht kollidierenden Weg von einer Startkonfiguration q_{init} des Roboters zu einer Zielkonfiguration q_{goal} zu finden. Dabei ist es wünschenswert, dass der Pfad zur Laufzeit, also zeitnah, erstellt wird. So kann in einem gewissen Rahmen auf wechselnde Bedingungen reagiert werden. Bei der Pfadplanung wird zwischen lokalen und globalen Bahnplanern unterschieden. Ein lokaler Planer überprüft nur Verbindungen von Konfigurationen im unmittelbaren Umfeld eines gegebenen Zwischenergebnisses. Dabei werden lediglich einfache Wege gesucht. Im besten Fall ist dies eine kollisionsfreie Gerade im Konfigurationsraum. Sollte doch eine Kollision existieren, so wird versucht diese lokal zu umgehen. Da dies nicht immer möglich ist, verharren lokale Planer oft in lokalen Minima. Globale Planer hingegen betrachten das Gesamtproblem. Hierbei wird versucht, über alle möglichen Konfigurationen des erreichbaren Raumes einen realisierbaren Pfad zur Zielkonfiguration zu finden. Um Hindernisse und Engstellen zu umgehen, werden im Gegensatz zu lokalen Planern alternative Konfigurationen im gesamten C_{space} gesucht.

2.2 Stand der Forschung

Eine grundlegende Aufgabe der Robotik ist es, die Bedienung von robotischen Maschinen so einfach wie möglich zu gestalten [28]. Dazu soll der Roboter nur mittels eines *high-level control interfaces*, wie zum Beispiel einer Spracheingabe, gesteuert werden. Die Befehle beschränken sich dabei auf einfache, als Aufforderung gestellte Aufgaben. Der Befehl „serve tea“ wird beispielsweise eindeutig durch den Roboter interpretiert und verarbeitet. Das Lösen dieser Aufgabe sollte dann vollkommen autonom geschehen. Hierzu ist es wichtig, dass der Roboter sowohl den gesamten Ablauf, als auch die in Unteraufgaben benötigten Bewegungen der einzelnen Vorgänge selbstständig plant. Zumindest in der Pfadplanung werden bereits seit einigen Jahren Ansätze der Autonomie verfolgt.

2.2.1 Algorithmen zur Pfadplanung

Der folgende Abschnitt beschäftigt sich mit einigen Algortihmen zur Pfadplanung, die in der Vergangenheit erfolgreich eingesetzt werden konnten. Außerdem wird der Frage nachgegangen wie die Pfadplanung bei Robotern mit mehreren Armen angegangen werden kann.

2.2.1.1 Potential Field Guided Path Planning

Einer der ersten Ansätze zur Pfadplanung basiert auf der Idee von Potentialfeldern [26]. Bei *Potential Field-Guided Path Planning* wird jedes Objekt als Hindernis mit einem künstlichen Kraftfeld gesehen. Der Roboter ist dabei eine Masse, die sich unter dem Einfluss der Potentialfelder durch den Konfigurationsraum bewegt. Die Startkonfiguration bildet eine Potentialquelle und die Zielkonfiguration wird als Potentialsenke definiert. In jeder Konfiguration q wirkt die künstliche Kraft $F(q)$ eine Beschleunigung auf die Masse des Roboters aus. Mittels dieser Begebenheit kann zu jedem Zeitpunkt die Kraft und das Drehmoment der einzelnen Motoren im Roboter bestimmt werden. Die erhaltenen Parameter dienen dann als Kommandos für dessen Servomotoren. Das Verhalten ähnelt dann einer Verschiebung durch die Potentialfelder. Dies wird in in Abbildung 2.1 noch einmal verdeutlicht.

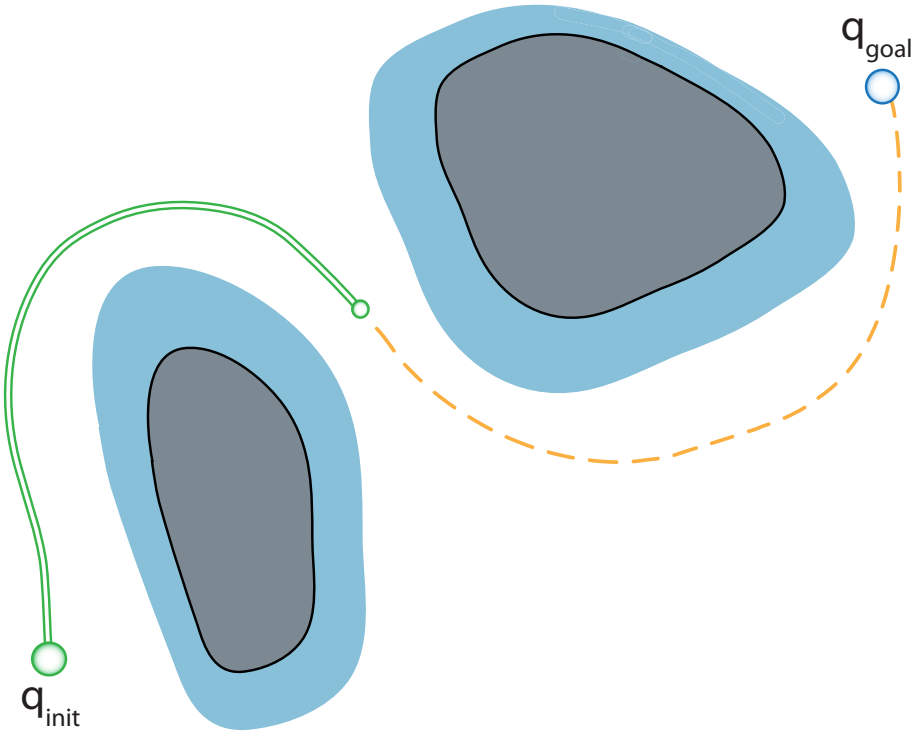


Abbildung 2.1: Roboter unter dem Einfluss von Potentialfeldern

Diese Art der Planung kann zur online Pfadplanung verwendet werden. Bei günstigen Konstellationen der Umgebung mit wenigen bis gar keinen Hindernissen können so durchaus schnell Lösungen gefunden werden. Speziell für mobile Roboter ist dies ein erfolgsversprechender Ansatz. Allerdings ist die Chance groß, dass der Planer dabei lokale Minima der Potentialfunktion nicht überwinden kann. Im Allgemeinen ist die Potential-Feld-Methode zudem ungeeignet für Roboter mit einer hohen Anzahl von Freiheitsgraden. Die Abbildung der Potentiale im kartesischen Raum auf den n-dimensionalen Konfigurationsraum ist sehr kompliziert. In engen Passagen

tendiert der Roboter zudem zunehmend zum Oszilieren, da er von allen Seiten abgestoßen wird [23]. Um dies zu umgehen, muss großer Aufwand beim Erstellen der Potentialfelder betrieben werden.

2.2.1.2 Sampling-basierte Verfahren

Eine eigene Unterklasse der Pfadplanungsalgorithmen bilden sampling-basierte Verfahren. Dabei werden zufällig Konfiguratoinen aus dem kompletten C_{space} gezogen. Die Exploration des kompletten Konfigurationsraums liegt bei diesen globalen Verfahren im Vordergrund. Samplingstrategien sind darauf ausgelegt, den Konfigurationsraum so schnell wie möglich nach bestimmten Kriterien abzudecken. Ein optimales Verfahren würde in möglichst kurzer Zeit den gesamten C_{space} erkunden. Da dies bei einem Konfigurationsraum mit einer unendlichen Größe nicht realisierbar ist, muss versucht werden, an jeden Wert mit einer gewissen Abweichung heranzukommen. Erhällt der Planer unbegrenzt Zeit wird immer eine Verbindung zur Zielkonfiguration gefunden. Planer mit dieser Eigenschaft werden als *Probabilisticly Complete* bezeichnet. Durch Heusristiken können Samplingverfahren beschleunigt werden. Obwohl zufällige Algorithmen nicht deterministisch sind, sind sie dennoch sehr gut geeignet für die Bahnplanung. Nach relativ kurzer Zeit kann eine nahezu optimale Lösung gefunden werden[1]. Im Folgenden werden zwei dieser Verfahren näher betrachtet.

2.2.1.3 Probabilistic Roadmaps

Eine Variante der zufallsbasierten Bahnplanung ist das Verfahren der *Probabilistic Roadmaps* (PRM) [6]. PRMs liegt die Idee von Samplingstrategien zu Grunde. Bei diesen wird der Pfad im C_{space} in zwei Schritten bestimmt. Der erste Schritt besteht darin, die sogenannte *Roadmap*, also eine Straßenkarte, zu erstellen. Diese Phase wird als Lernphase bezeichnet. Dabei werden zufällig kollisionsfreie Konfigurationen erstellt und anschließend durch einen lokalen Planer miteinander verbunden. Hierbei werden jeweils die nächsten Nachbarn untereinander verknüpft, sodass ein dichtes Netz entsteht. Das komplette Netz bildet eine kollisionsfreie Karte in welcher der Konfigurationsraum auf verschiedenen Wegen durchlaufen werden kann. Die zweite Phase ist die Erkundungsphase. In dieser wird versucht, die Startkonfiguration q_{init} mit der Zielkonfiguration q_{goal} mittels einer Graphensuche miteinander zu verbinden. Ist dies erfolgreich, werden die einzelnen Knoten zu einem Pfad P verbunden und danach in einem Glättungsschritt in einen realisierbaren Pfad umgewandelt (Siehe Abbildung 2.2). Einmal erstellte Roadmaps können immer wieder zur Pfadplanung verwendet werden, solange sich die Szene nicht ändert. Sollte wäh-

[1] Die reine Planung für eine einfache Bahn kann durchaus innerhalb weniger Sekunden stattfinden. Jedoch benötigt eine solche Trajektorie unbedingt einen Glättungsschritt um den Verlauf der Bahn natürlich zu gestalten. Die Optimierungszeit beträgt je nach Verfahren ein vielfaches der Planungszeit.

rend der Erkundungsphase mehrmals kein Ergebnis gefunden werden, ist es möglich, die Roadmap zu erweitern. Dabei werden üblicherweise *Bias-Algorithmen*[2] verwendet, die voreingenommen gegenüber unerkundeten Bereichen agieren. Gibt es zum Beispiel einen großen, noch unerkundeten Bereich, so wird die Suche zunächst dort fortgesetzt. Verändert sich die Umgebung in der sich der Roboter bewegt, so kann der Algorithmus um eine erneute Kollisionsabfrage zur Laufzeit erweitert werden. So kann in gewissem Maße auf Veränderungen der Szene eingegangen werden [2].

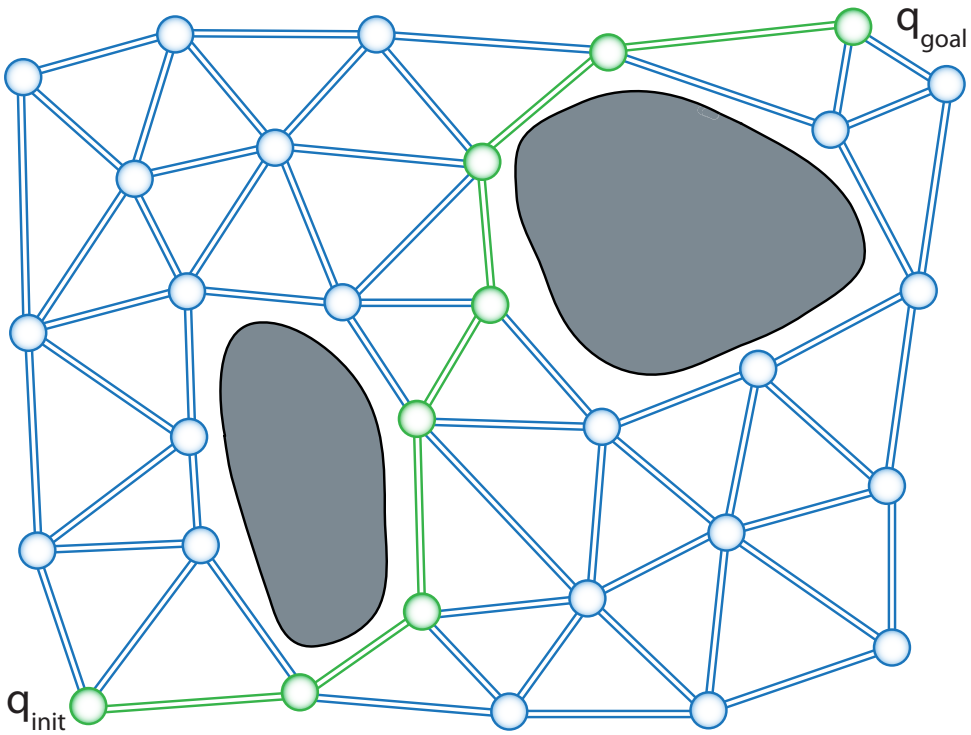


Abbildung 2.2: Erstellen einer Probabilistic Roadmap

Probabilistic Roadmaps eignen sich hervorragend für Industrieszenarien bei denen die gleiche Aufgabe immer wieder durchgeführt werden muss. Dafür ist vorallem die Roadmap verantwortlich. Der detaillierte Aufbau der Karte nimmt sehr viel Zeit in Anspruch. Dafür können im Nachhinein umso schneller realisierbare Bahnen gefunden werden. Umgebungen, bei denen sich Elemente oft bewegen, sind hingegen nicht mit diesem Verfahren zu bewältigen. Das nichtdeterministische Verhalten des Algorithmus erschwert zudem die Reproduzierbarkeit der Pfade und ist speziell mit Hinsicht auf Demonstrationen im Labor ein wichtiger Kritikpunkt.

[2]Bias ist englisch und bedeutet Tendenz, Vorurteil oder Neigung. Ein Bias-Algorithmus handelt daher immer in Richtung einer vorgegebenen Neigung.

2.2.1.4 Rapidly Exploring Random Trees

Die Klasse der *Rapidly-Exploring Random Trees* (RRT) ist eine weitere Algorithmenfamilie zur zufallsbasierten Erkundung des C_{space} [27]. Auch hier wird der Konfigurationsraum mittels Samplingalgorithmen durchsucht. Anders als bei PRMs wird bei RRTs direkt eine Verbindung zwischen den einzelnen Knoten erstellt. Alle Konfigurationen im RRT haben, ausgenommen der Wurzel, genau einen Vorgänger. Somit entsteht die typische Struktur eines Suchbaumes. Der entstehende Baum ist im Gegensatz zu den Netzen der PRMs direkt zielführend. Erstellte Routen finden daher schnell die Zielkonfiguration q_{goal}. Der folgende Pseudocode erläutert das Vorgehen näher:

Algorithm 1 BuildRRT(q_{init})

```
T.Init(q_init);
for k = 1 to K do
  q_rand ← RandomConfig()
  Extend(T , q_rand)
end for
return T
```

Algorithm 2 Extend(T, q)

```
q_near ← NearestNeighbor(q, T)
if NewConfig(q, q_near, q_new) then
  T.AddVertex(q_new)
  T.AddEdge(q_near, q_new)
  if q_new = q then
    return reached
  else
    return advanced
  end if
end if
return trapped
```

Abbildung 2.3: Pseudocode des RRT Algorithmus [12]

In einem Iterationsschritt wird je Schritt eine neue Konfigurtation q_{rand} gezogen. Die Extend-Funktion wählt dann den nächstliegenden Nachbarknoten (q_{near}), im bestehenden Baum aus und versucht diesen mit q_{rand} mit einer einfachen Gerade im Konfigurationsraum zu verbinden. In der NewConfig Funktion überprüft der lokale Planer dabei eventuelle Kollisionen. Es können drei Situationen eintreten: *Reached*, die signalisiert, dass q_{goal} mit diesem Knoten erreicht wurde. *Advanced*, mit der ein kollisionsfreier, neuer Knoten q_{new} an den Baum angefügt wird. Und drittens

Trapped, in dem der neue Punkt in einem kollidierenden Bereich liegt. Um eine noch schnellere Ausbreitung zu erhalten, setzen viele RRT-Varianten dabei auf Bias-Algorithmen. Auch bei Rapidly-Exploring Random Trees wird ein Glättungsschritt benötigt, um eine äquidistante, kürzestmögliche Trajektorie zu gewährleisten. Die Effizienz des RRT-Algorithmus kann gesteigert werden, indem, wie in Abbildung 2.4 gezeigt, jeweils von Start- und Zielkonfiguration ein Suchbaum gestartet wird. Dabei wechseln sich die Bäume beim Erkunden stets ab, sobald der trapped Status einsetzt. Können die beiden Bäume eine Verbindung herstellen, so ist der Weg gefunden [24].

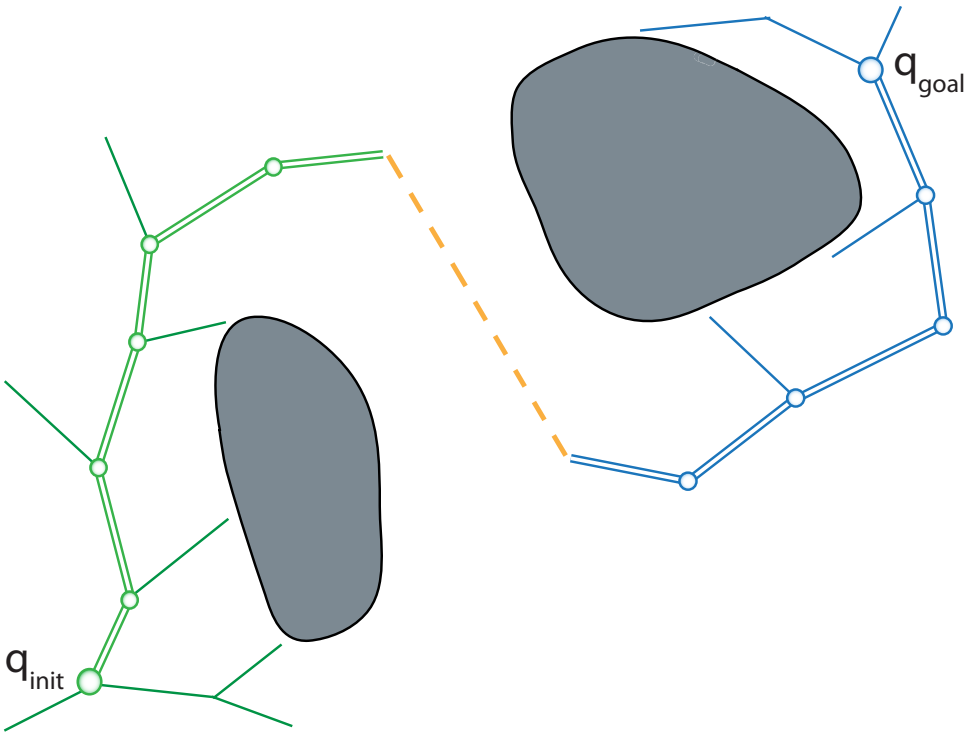


Abbildung 2.4: Erstellung eines BiRRT Suchbaumes

Für einmalige Anfragen sind RRTs gut geeignet, da sie direkt zielführend sind. Ein unerkundeter Raum kann schnell exploriert werden, wodurch zeitnah ein realisierbarer Pfad gefunden wird. Anders als Probabilistic Roadmaps hinterlässt das RRT Verfahren kein ungültiges Netz nach einer Manipulation von Objekten. Bei Zielkonfigurationen in schwer erreichbaren Gebieten findet zudem der zweite Suchbaum schnell eine Lösung aus dieser. Nichtsdestotrotz ist auch dieses Verfahren nicht für dynamische Szenarien einsetzbar, da die Planungszeit immer noch mehrere Sekunden dauern kann.

2.2.2 Pfadplanung mit zwei Armen

Die allgemeine Lösung des Pfadplanungsproblems scheint in noch weitere Ferne zu rücken, wenn der Roboter um einen oder gar mehrere Arme ergänzt wird. Durch Einbinden eines zweiten Armes wird die Dimensionalität des Konfigurationsraumes verdoppelt. Andererseits kann diese Erkenntnis auch als Motivation gesehen werden, denn mit der Erweiterung um einen Arm steigt gleichzeitig die Effektivität des Roboters. Ein weiterer Arm bedeutet eine Geschwindigkeitssteigerung, wenn Aktionen parallel durchgeführt werden können. Außerdem können mit zwei Armen eine Reihe von Szenarien bearbeitet werden, die mit einem Arm nicht zu lösen sind. Besonders schwere Objekte können nur mit kombinierter Kraft angehoben werden. Sogar komplexe Aufgaben werden lösbar: Das Öffnen einer Flasche beispielsweise ist nur mit zwei Armen möglich. Aus diesen Gründen widmet sich diese Arbeit der Planung und Ausführung alltäglicher zweihändiger Manipulationsaufgaben.

In den Arbeiten von Koga [22], Guirard [16], Zöllner [37] und Li [29] wurde Manipulation mit mehreren Armen bereits betrachtet. Nach diesen kann die Art der Manipulation in mehrere Kategorien aufgeteilt werden:

1. **Einarmige Manipulation** – Hierbei wird nur ein Manipulator zum Lösen der Aufgabe verwendet.

2. **Unkoordinierte, zweiarmige Manipulation** – Beide Arme agieren völlig unabhängig voneinander, sodass mehrere Aufgaben oder Teil-Aufgaben simultan gelöst werden können.

3. **Asymmetrisch koordinierte, zweiarmige Manipulation** – Die Bewegungen der Arme sind zeitlich oder logisch voneinander abhängig. Die Aufgabe kann nur gelöst werden, wenn beide Endeffektoren zur richtigen Zeit am richtigen Ort sind (Tee einschänken).

4. **Symmetrisch koordinierte, zweiarmige Manipulation** – Die Aufgabe kann nur gelöst werden, wenn beide Arme das Objekt gleichzeitig halten, dabei entsteht eine geschlossene kinematische Kette. Die Notwendigkeit kann auf das Gewicht oder die Form des Objektes zurück geführt werden.

5. **„Hand over“ Manipulation** – Diese Kategorie wird zusätzlich von Li eingeführt. Dabei interagieren die Arme derart, dass Objekte von einer Hand zur anderen übergeben werden.

Die Manipulationsplanung bei mehreren Armen wirft einige Fragen auf. So ist es unter anderem zu klären, wann welcher Arm ein Objekt greifen soll und wann die Aufgabe einen anderen Arm betrifft. Ein zweiter Aspekt ist die Frage nach der Kooperation der Manipulatoren. In typischen Industrieszenarien werden diese Probleme meist durch zeitabhängige Koordinationsregeln, oder aufwendige Konstruk-

tionen umgangen [29]. Die Aktionen der Roboter laufen nach einem strikten Plan ab. Erst wenn ein Roboter zum Beispiel seine Schweißaktion ausgeführt hat, kann ein anderer Roboter daneben seinen Klebstoff auftragen. Laut Li wurden mehrere Ansätze zur Lösung entwickelt. Bei der zentralisierten Methode wird die Vereinigung der einzelnen Roboterarme als ein einziger Roboter gesehen. Das kartesische Produkt der Konfigurationsräume ergibt einen kombinierten C_{space}. Nur in einem gewissen Bereich, indem sich die Konfigurationsräume überschneiden, sind Kollisionen überhaupt möglich, sodass es ein Lösungsansatz wäre, den Raum exklusiv einem Manipulator zuzuweisen. Ist der Raum allerdings zu groß, so wird die Erkundung aufwendig. Ein weiterer Ansatz ist die entkoppelte Methode, bei der die Bewegungen jeweils einzeln geplant werden. Bewegt sich ein Roboter, so sehen ihn die übrigen als bewegliches Hindernis während ihrer Planungsphase an. Dadurch verstreicht allerdings viel Zeit beim Warten der inaktiven Manipulatoren. Eine andere Variante dieses Algorithmus plant beide Bewegungen gleichzeitig und führt diese simultan aus, verringert jedoch die Geschwindigkeit der Roboter so weit, dass eine Kollision unmöglich ist. Als Beispiel für die simultane Manipulation eines Objektes mit zwei Armen sind die Arbeiten am LAAS-CNRS in Toulouse zu nennen. Diese Problemklasse birgt viele Schwierigkeiten. Die Größte stellt die virtuelle Verbindung der beiden Arme über das gegriffene Objekt dar. Die kinematische Abhängigkeit begrenzt den Interaktionsbereich auf einen kleineren Subraum. Als Idee werden hier PRMs mit mehreren Schichten verwendet, damit Konfigurationen mit verschiedenen Armstellungen unabhängig abbildbar sind [14]. Um Kollisionen zu vermeiden während sich beide Manipulatoren kreuzen, werden die einzelnen Roadmaps auf einander abgebildet und in einem Supergraph abgebildet. Die Planung findet danach nur noch auf dem Supergraph statt, Kollisionen sind somit ausgeschlossen [15].

Die dargestellten Lösungsversuche verdeutlichen die tatsächliche Komplexität der kooperierenden Manipulation, denn der Sachverhalt kann nicht in einem einzelnen Algorithmus beschrieben werden. Um den Arbeitsraum möglichst effizient einzuteilen und damit den maximalen Zeitgewinn zu erreichen, müssen verschiedene Lösungsansätze kombiniert werden. So benötigt jede Situation eine spezifische Strategie. Aus diesem Grund befasst sich diese Diplomarbeit mit der simultanen Manipulation mehrerer Objekte. Zum Erreichen eines höheren Gesamtziels sollen mehrere Aktionen parallel ausgeführt werden. Die Betrachtung kinematischer Abhängigkeiten und simultane Manipulation eines Objekts sind dagegen nicht Teil dieser Arbeit.

2.2.3 Justin – Der mobile, humanoide Roboter des DLR

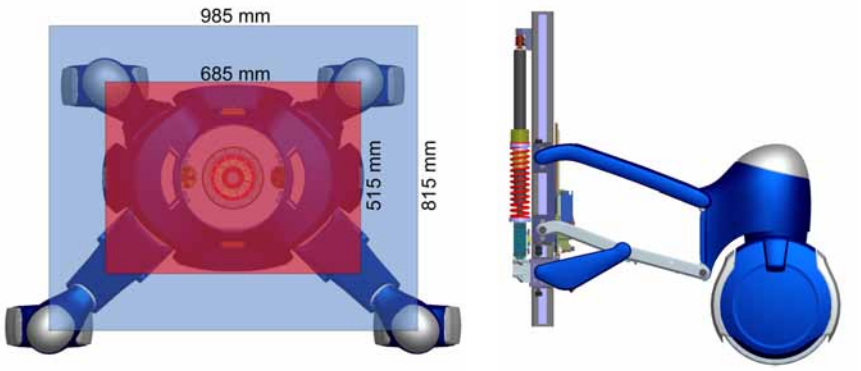
In diesem Abschnitt wird der mobile, humanoide Roboter Rollin' Justin vorgestellt [31]. Er wird seit einigen Jahren am Institut für Robotik und Mechatronik des Deutschen Zentrums für Luft- und Raumfahrt in Oberpfaffenhofen entwickelt. Die menschenähnliche Gestalt seiner Arme sowie die Erweiterung des Systems um eine mobile Plattform, machen ihn zu einem optimalen Forschungswerkzeug für alltägliche, zweiarmige Manipulationsaufgaben.

2.2.3.1 Aufbau von Justin

Basierend auf vorangegangen Forschungsergebnissen des DLR ist der Roboter völlig modular aufgebaut. So besteht Justin in erster Linie aus zwei DLR Leichtbaurobotern (*light weight robot, lwr*) [19], die als Arme dienen und jeweils einer DLR Hand als Endeffektor [5]. Verbunden sind die Arme über einen Torso, welcher ebenfalls aus der Bauweise der Leichtbauroboter heraus entwickelt wurde. Als Kopf des Systems wird der DLR 3D-Modelierer verwendet, um die Umgebung wahrzunehmen. Die neueste Version von Justin, der so genannte *Rollin' Justin*, verfügt zudem über eine mobile Plattform und erhällt somit einen stark erweiterten Arbeitsraum. Insgesamt hat der Oberkörper 44 kontrollierbare Freiheitsgrade.

2.2.3.2 Mobile Plattform

Für komplexe Aufgaben ist die Fähigkeit, einen Manipulator unabhängig zu platzieren, von großem Vorteil. Dies kann durch eine mobile Plattform geleistet werden. So können Suchalgorithmen für die Pfadplanung des Gesamtsystems weit über den eigentlichen Konfigurationsraum der Arme hinaus planen [12]. Außerdem kann der Roboter derart platziert werden, dass der Konfigurationsraum der Arme optimal ausgenutzt wird [36]. Desweiteren kann eine mobile Plattform weitere Vorteile beim Ausführen von Manipulationsaufgaben bringen. Beim Einsatz der Plattform während der Manipulation können zum Beispiel zusätzliche Kräfte aufgebracht werden. Ebenfalls möglich ist die genauere Positionierung durch die Plattform [20]. Bei der



(a) Variable Standfläche (b) Dämpfersystem

Abbildung 2.5: Variable Standfläche, und Dämpfersystem der mobilen Plattform [13].

Konstruktion der mobilen Plattform für Justin wurde auf Räder gesetzt. Im Gegensatz zu einem humanoiden Laufroboter mit zeri Beinen ist ein sicherer Stand garantiert. Außerdem bringen Beine eine Vielzahl von weiteren Problemen mit, die besser gesondert betrachtet werden. Um trotzdem mehr Flexibilität zu erlangen, sind die Räder an adaptiven Beinelementen angebracht. Wird ein sicherer Stand benötigt,

können die Beine ausgefahren werden. Beim Durchqueren von Türen hingegen ist der Platz beschränkt, sodass die Beine besser eingefahren werden (siehe Abbildung 2.5). Kleine Hindernisse, wie Teppiche und Schwellen, können durch zuschaltbare Dämpfer überwunden werden, ohne das die Stabilität des Roboters gefährdet wird.

2.2.3.3 Oberkörper und Kopf

Der Torso von Justin basiert auf der gleichen Leichtbautechnik wie die Arme [3]. Durch den Oberkörper ist Justin in der Lage, Objekte vom Boden gleichermaßen zu erreichen, wie Objekte auf einem Regal (siehe Abbildung 2.6). Die einzelnen Gelenke sind derart voneinander abhängig, dass das Brustsegment keinen hohen Lasten ausgesetzt ist. Eine Seilkonstruktion verteilt die Belastung des Oberkörpers. Das Gewicht von Justin selbst wird vom zweiten und dritten Torsogelenk getragen. Die hohen Drehmomente, die auftauchen wenn Justin seine Arme ausstreckt, werden über die Seile direkt in die mobile Plattform abgeleitet. Oberhalb des Torsos sitzt ein 3D-Modellierer, dessen Kameras als Justins Augen dienen.

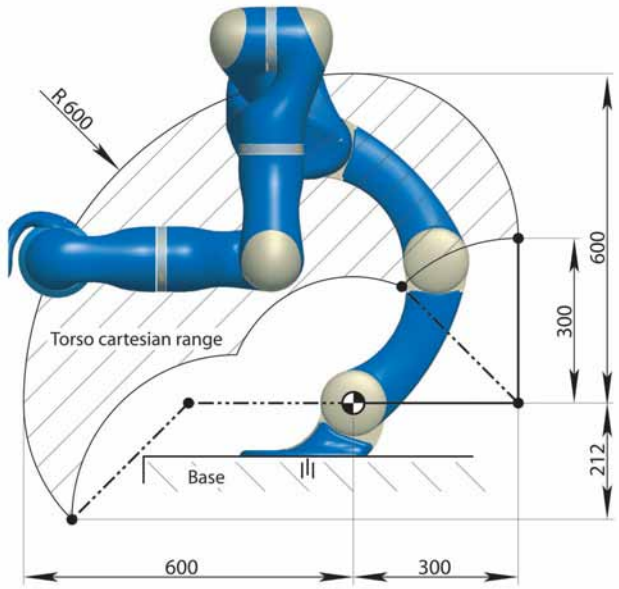


Abbildung 2.6: Aktionsbereich des Torsos [31]

2.2.3.4 DLR Leichtbauarme

Der Leichtbauroboter des DLR befindet sich mittlerweile in der dritten Generation [19]. Mit einem Gewicht von gerade mal 13.5 kg ist er in der Lage eine Last entsprechend seinem Eigengewicht zu stämmen. Anders als bei herkömmlichen Industrierobotern befindet sich beim LWR III die gesamte Steuerungslogik im Gehäuse des Armes. Ebenso wie der menschliche Arm hat der DLR Arm sieben Freiheitsgrade,

wodurch eine dem Menschen ähnliche Flexibilität erreicht wird. Durch die Leichtbautechnik erhält der Arm zudem eine gewisse Nachgiebigkeit, was den Umgang mit dem Roboter im Ganzen sicherer macht. Erste Pilotprojekte in industriellen Fertigungsbetrieben sind bereits umgesetzt.

2.2.3.5 DLR Hand

Die Hand ist das wichtigste Werkzeug des Menschen und das gilt auch für die Serviceroboter der Zukunft. Mit der Hand ist es dem Menschen möglich jedes erdenkliche Objekt zu greifen und aller Art Werkzeuge zu benutzen. Diese erstaunliche Vielseitigkeit wird versucht mit der DLR Hand zu reproduzieren [5]. Die aktuelle Generation der Hand, die an Justin montiert ist, besteht aus vier gleichwertigen Fingern mit je drei Freiheitsgraden. Sie ist weitgehend der menschlichen Hand nachempfunden, muss allerdings mangels Miniaturisierung einen Finger einbüßen. Nichtsdestotrotz ist die DLR Hand sehr gut geeignet für eine Vielzahl von Manipulationsaufgaben. Die Manipulationsfertigkeiten werden umso größer, wenn beide Hände parallel eingesetzt werden.

Abbildung 2.7: DLR Hände und Arme in Aktion [34]

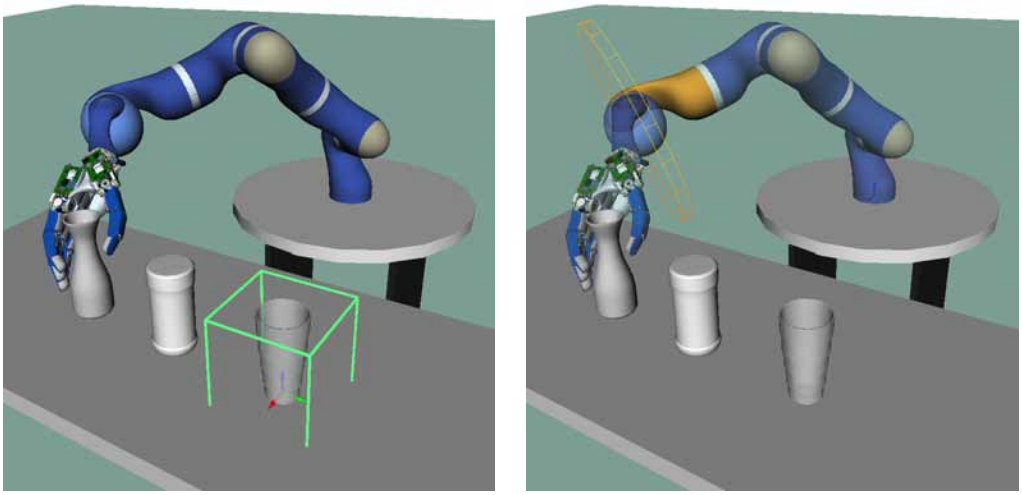
3 Die Planungs- und Simulationsumgebung OpenRAVE

Eine große Herausforderung bei der Entwicklung realer Roboter ist das Testen von komplexen Modulen für verschiedenste Anwendungen. Darunter solche, die für die Bewegungsplanung und Aufgabenplanung zuständig sind. Für diesen Zweck wird eine flexible, anpassbare Entwicklungs- und Simulationsumgebung benötigt. Eine derartige Umgebung bietet OpenRAVE, ***Open** **R**obotics and **A**nimation **V**irtual **E**nvironment*, welches derzeit am Carnegie Mellon University Robotics Institute entwickelt wird. OpenRAVE ist eine Cross-Plattform Softwarearchitektur, die es ermöglicht, Aufgaben- und Pfadplanung für jeden erdenklichen Roboter zu berechnen und zu visualisieren. Eine Plugin-Architektur erlaubt dem Nutzer einfach und schnell neue Komponenten wie Planer, Inverskinematiken, Physikmodule oder Sensoren zu entwerfen. Das Hauptaugenmerk des Benutzers kann sich jedoch auch komplett auf das Entwickeln von skriptgesteuerten Abläufen beschränken, ohne dass die dahinter liegenden APIs zur Kollisionserkennung und Planung genau bekannt sein müssen. Durch diese einfache Struktur ist es möglich, einen schnellen Austausch und Vergleich von Algorithmen zu erreichen [11].

3.1 Einführung in OpenRAVE

OpenRAVE, als solches ist in C++ geschrieben, verwendet mehrere Opensource Bibliotheken zur Visualisierung sowie zur Simulation. Dazu gehören Coin3d, OpenGL, Qt, und ODE. Die eigentliche Benutzeroberfläche beschränkt sich auf den Coin3D Viewer. Diese Oberfläche erlaubt die Betrachtung der Simulation ebenso wie deren Manipulation. Standardmäßig verändert eine *drag n' drop* Aktion den Blickwinkel der Kamera. Mit Druck auf *s* wandelt sich der Cursor in ein Fadenkreuz und es kann ein beliebiger Punkt auf einem Objekt als neuer Fokuspunkt bestimmt werden. Um in den Manipulationsmodus zu wechseln, muss *esc* betätigt werden. Ab sofort können ausgewählte Objekte beliebig im Raum verschoben werden. Ausgewählte Objekte erscheinen transparent. Am Ursprungspunkt des Objektes erscheint ein kleines Koordinatensystem (siehe Abbildung 3.1a).

Während Objekte angewählt sind, werden im oberen, linken Fensterrand Informationen zu Typ, Name und Transformation des Objektes angezeigt. Dies ist besonders nützlich, wenn für eine Aufgaben die Umwelt neu parametrisiert werden sollen. Zum Verschieben der Gegenstände erscheint zusätzlich ein farbiger Rahmen. Ist dieser orange, so steht das Objekt in Kollision. Im kollisionsfreien Fall, erscheint der



(a) Verschieben von Objekten (b) Editieren von Gelenkwinkeln

Abbildung 3.1: Bedienung der OpenRAVE GUI. Die linke Abbildung zeigt die Interaktion mit Objekten und das rechte Bild verdeutlicht die Umgangsweise mit Roboterelementen

Rahmen grün. Ebenso wie einzelne Hindernisse und Gegenstände, lassen sich auch einzelne Elemente des Roboters bewegen. Dazu muss die *strg* Taste gehalten werden, während der Roboter ausgewählt ist. Wie in Abbildung 3.1b gezeigt, erscheint ein Ring um das dazugehörige Gelenk, an dem die Konfiguration des Roboters verändert werden kann. Zudem enthält die API um OpenRAVE noch einige Besonderheiten, auf die später in Kapitel 3.4 eingegangen wird.

3.2 Architektur von OpenRAVE

Der Aufbau von OpenRAVE ist größtenteils modular. Die Kernanwendung, also die eigentliche API, ist in C++ geschrieben und umfasst eine umfangreiche Bibliothek zur Handhabung der Robotermodelle in der Simulationsumgebung. Jedoch kann bereits die graphische Oberfläche, die in der Standardversion auf Coin3D basiert, durch eine beliebige andere graphische Visualisierung ausgetauscht werden. Der eigentliche Fokus der Modularisierung liegt allerdings auf der Möglichkeit, verschiedene Planungsalgorithmen, Sensoren und andere Komponenten besonders leicht und schnell in die Umgebung einzubetten.

3.2.1 Modularer Aufbau durch Plugins

Die Plugins in OpenRAVE sind vielseitig einsetzbar. Sie werden in verschiedene Gruppen unterteilt.

1. **Kollisions-Checker**

   Für die Kollisionsabfrage sind bereits drei Plugins integriert. Darunter ODE (*Open Dynamics Engine*), Bullet und PQP (*Proximity Query Package*). Der Kollisions-Checker ist ausschlaggebend für die Planungsdauer. Je länger ein Kollisionstest dauert, desto länger ist die Planungszeit. Dies wird besonders kritisch, wenn sich viele Objekte im Einflussbereich des Roboters befinden.

2. **Controller**

   Eine weitere Klasse bilden die Controller. Sie sind für die Steuerung virtueller sowie realer Roboter zuständig. Mittels eines Controllers lassen sich Trajektorien abfahren oder zurückgeben. Der aktuelle Roboterstatus kann jederzeit mit einem Controller abgefragt werden.

3. **Berechnung der Inverskinematik**

   Nicht immer ist bekannt, welche Gelenkwinkelstellung eines Roboters zum gewünschten Ziel führt. In einem solchen Fall wenn die Zielposition im kartesischen Raum bekannt ist, kann mittels der Inverskinematik des Roboters eine passende Zielkonfiguration gefunden werden. Die dazu benötigte Inverskinematik wird in einer eigenständigen Plugindefinition beschrieben. Jeder Roboter und auch jeder Manipulator eines Roboters, braucht seine eigene Inverskinematik. Da die Erstellung der Inverskinematik sehr aufwendig ist, kann das *IKFast* Tool benutzt werden, um einen solchen *IK solver* zu generieren. Bei einem sieben DOF Arm wird die Inverskinematik für sechs Freiheitsgrade berechnet und ein Parameter als freier Parameter gesetzt. Dieser wird einfach durchiteriert, bis eine gültige Lösung gefunden wird.

4. **Physikmodule**

   Neben den Kollisionsabfragen kann die Simulation noch um eine *physics engine* erweitert werden. Mithilfe eines Physikmoduls können äußere Kräfte, wie Stöße und Rückstöße oder Schwerkraft, auf das Robotermodell einwirken. Interessant ist dies vor allem für die Dynamik der DLR-Leichtbauroboter. Dadurch, dass die Leichtbauroboter nicht steif sondern nachgiebig sind, stimmen Simulation und Realität in bisherigen Versuchen meist nicht überein. Der reale Roboter hängt bei voll ausgestreckten Armen mehrere Millimeter durch. Wird die Schwerkraft auch im Modell berücksichtigt, können derartige Fehler minimiert werden.

5. **Planer**

   Planer sind der Hauptteil von OpenRAVE. Standardmäßig sind mehrere RRT Varianten implementiert. Denkbar wären allerdings auch verschiedene PRM-Methoden oder Potentialfelder. Auch die Entwicklung und die Validierung

völlig neuer Algorithmen ist in einer Simulationsumgebung wesentlich leichter als in der Realität.

6. **Probleminstanz**

   Um einen Roboter mit einem Planer zu verbinden, wird eine Probleminstanz benötigt. Die Probleminstanz kann spezielle Funktionen für Manipulation oder Navigation beinhalten.

7. **Sensoren**

   Die letzte Pluginklasse definiert verschiedene Sensoren. Sensoren sammeln Informationen über die Umwelt und liefern diese in einem standardisierten Format. Sensoren können an beliebigen Teilen eines Roboters angefügt werden.

3.2.2 Kinematische Definitionen mit dem XML Collada Fileformat

Die Kinematik eines Roboters wird in OpenRAVE im XML Collada Fileformat beschrieben. Die ***COLLA****borative* ***D****esign* ***A****ctivity* beschreibt 3D Objekte jeglicher Art in einer leicht zu verstehenden Form. Einige der bekanntesten CAD Programme unterstützen dieses Format, darunter kommerzielle Programme wie Maya, aber auch Opensource Varianten wie Blender. In OpenRAVE wird eine etwas abgewandelte Art von Collada verwendet. Es können einzelne Objekte, Roboter und ganze Umgebungen beschrieben werden. Listing 3.1 beschreibt eine kleine Beispielumgebung, die sogenannte *Environment*, mit einigen integrierten Objekten.

```
<!-- simple environment -->
<Environment>
  <Robot name="justin" file="justin.robot.xml" />

  <KinBody name="regal" file="shelf.kinbody.xml">
    <rotationaxis> 0 0 1 180 </rotationaxis>
    <translation> 0.65 0.50 0 </translation>
  </KinBody>

  <KinBody name="boden">
    <Body type="static">
      <Geom type="box">
    <extents> 10 10 0.005 </extents>
      </Geom>
    </Body>
  </KinBody>
</Environment>
```

Listing 3.1: Einfache Definition einer Environment Datei

In der Environment werden alle Objekte einer Szene definiert. Roboter, Körper aber auch umgebungsspezifische Variablen, wie die Kameraposition, können hier beschrieben werden. Die meisten Gegenstände, wie zum Beispiel Justin und das Regal, werden aus eigenständigen Dateien importiert und nur noch via Rotation und Translation an ihren Bestimmungsort verschoben. Es ist aber auch möglich, komplett neue Objekte zu erstellen. Der Boden besteht im Wesentlichen aus einem sehr flachen Quader.

Zum Erstellen einer KinBody-Datei, also eines beweglichen Körpers wie einer Tasse oder einem Glas, gibt es zwei Möglichkeiten. Entweder können komplexe Dreiecksnetze (*triangle meshs*) erstellt werden, wobei diese dann das gewünschte Objektmodell direkt bilden. Oder bereits vorhandene Open Inventor Modelle können, wie in Listing 3.2, eingebunden werden. Dies ist besonders praktisch, da im laufe der Jahre eine Vielzahl von Inventor Modellen erstellt wurden. Das komplette Szenario rund um Justin wurde bereits in OpenRAVE importiert. Eine KinBody-Beschreibung besteht immer aus dem KinBody-Hauptteil und einem oder mehreren Bodys. Wenn, wie im Beispiel, eine CAD Datei verwendet wird, kann ein *Rendermodell* und ein *Datamodell* hinterlegt werden. Das Rendermodell ist für die Visualisierung des Körpers zuständig und das Datamodell bildet die Grundlage für die Kollisionsabfragen.

```
<KinBody name="table">
  <modelsdir>./inventor</modelsdir>
  <Body type="dynamic">
    <Geom type="trimesh">
      <render>table.iv</render>
      <data>table.iv</data>
    </Geom>
  </Body>
</KinBody>
```

Listing 3.2: Beschreibung eines KinBodys mit Inventor Dateien

KinBodys sind elementare Objekte in OpenRAVE. Jeder Roboter besteht aus mindestens einem KinBody. Auf die Roboterdefinition wird später in Kapitel 3.5 noch genauer eingegangen.

3.2.3 Programmieren mit den Skriptsprachen Matlab und Python

Ein wichtiger Aspekt für die Autonomie von Robotern ist die Fähigkeit Aufgaben mit so wenig wie möglich Informationen zu bearbeiten. Nichtsdestotrotz muss Justin, mangels künstlicher Intelligenz, einem bestimmten Ablauf folgen, um Probleme zu lösen. Diese Abläufe werden in dieser Arbeit vom Menschen erstellt und vorgegeben. Die Abläufe werden in kleinen Skripten beschrieben. Skripte sind Programme, die ohne Kompilierung ausgeführt werden können. Dies ist wichtig, da nicht jede Auf-

gabe nach dem gleichen Schema abgearbeitet werden kann. Vielmehr braucht ein heutiger Roboter eine Vielzahl von bekannten Abläufen, um im Ansatz autonom zu wirken. Neue Problemlösungsstrategien können zur Laufzeit in ein Robotersystem eingespielt werden. Diese Eigenschafft macht es zudem möglich, Skripte späterhin auch autonom, von einem Aufgabenplaner, der als höhere Instanz der Steuerungslogik agiert, erstellen zu lassen. Bei OpenRAVE können Skripte auf zwei verschiedene Wege erstellt werden.

3.2.3.1 Matlab Skripte

Ein Matlab Skript setzt eine lauffähige Matlab- oder Octave-Umgebung voraus. OpenRAVE ist derart mit Matlab verknüpft, dass OpenRAVE auf Netzwerkeingaben auf einem bestimmten Port wartet. Matlab- oder Octave-Befehle werden automatisch an diese Ports gesendet. Die Matlab-Bibliothek enthält, im Gegensatz zur Python API, nicht alle Schnittstellen der OpenRAVE C-Library. Beim Skripten mit Matlab müssen einige Formatierungsregeln beachtet werden [9].

- Alle OpenRAVE Funktionen starten mit *or*
- Alle KinBody spezifischen Funktionen starten mit *orBody*
- Alle Roboter spezifischen Funktionen starten mit *orRobot*, jeder Roboter kann alle orBody Funktionen nutzen
- Alle Environment Funktionen starten mit *orEnv*
- Alle probleminstanz-spezifischen Funktionen starten mit *orProblem*

Um ein Skript ausführen zu können, muss zuerst OpenRAVE im Hintergrund geöffnet und anschließend das Skript gestartet werden. Beispielcode 3.3 zeigt einen kleinen Codeausschnitt eines Matlabprogramms.

```
orEnvLoadScene('data/lab1.env.xml', 1)

%get manipulator
manipid = orEnvCreateProblem('BaseManipulation', 'BarrettWAM')

%start planning
orProblemSendCommand('MoveManipulator armvals -0.75 1.24 -0.064↩
    2.33 -1.16 -1.548 1.19',manipid)

%wait for robot movements
WaitForController('BarrettWAM')
```

Listing 3.3: Skript zur Planung einer Manipulator Bewegung in Matlab

Bei diesem kleinen Beispiel wird eine Szene in die laufende OpenRAVE Instanz eingeladen und anschließend eine Probleminstanz erzeugt. Diese Probleminstanz kann danach genutzt werden, um eine Planung auszuführen. Mit dem Befehl *MoveManipulator* wird eine Planung im Konfigurationsraum gestartet. Im vergleich zur Python Schnittstelle, ist der Code nicht sonderlich umfangreich. Durch die eingeschränkten Möglichkeiten in Matlab ist er auch nicht sehr flexibel.

3.2.3.2 Python Skripte

Die zweite Möglichkeit, um OpenRAVE via Skript zu steuern, ist Python. Anders als bei Matlab sind alle Funktionen des C++ Interface in Python implementiert. Dadurch kann die Anwendung zwar umso komplexer gestaltet werden, allerdings wird sie auch schneller undurchsichtig. Auch bei Python wird eine Laufzeitumgebung gebraucht, aber anders als bei Matlab muss keine große Entwicklungsumgebung geladen werden. Es hat sich außerdem gezeigt, dass die gleichen OpenRAVE Module unter Python beinahe doppelt so schnell ablaufen wie unter Matlab. Mit dem *import* Befehl werden lediglich die Programmstrukturen geladen, die auch wirklich benötigt werden. So vereint Python die Flexibilität einer Programmiersprache, mit den Vorzügen von Skriptsprachen, welche keine Kompilierung benötigen. Listing 3.4 zeigt den Umgang mit Python und OpenRAVE.

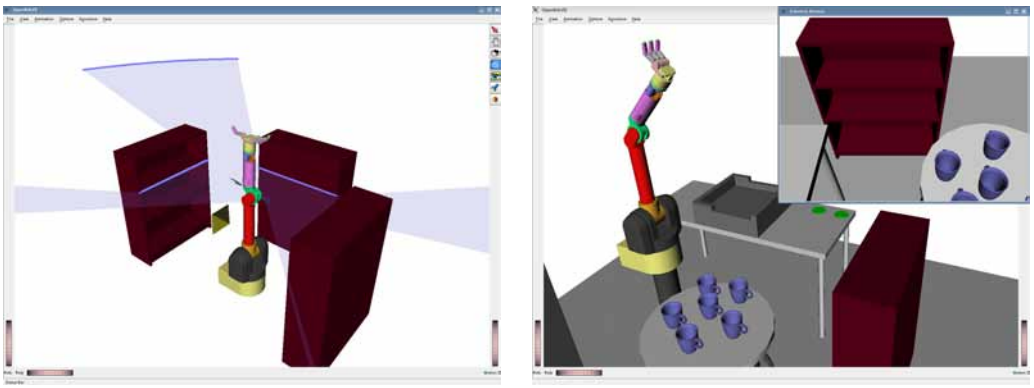
```
#inmport openrave modules for python
from openravepy import *

#initialisation
gEnv = Environment()
gEnv.SetViewer('qtcoin')
gEnv.Load('test.env.xml')

#create problem instance
gRobot = gEnv.GetRobots()[0]
gProb = gEnv.CreateProblem('basemanipulation')
gEnv.LoadProblem(gProb,gRobot.GetName())

gRobot.GetEnv().LockPhysics(True)  #lock mainthread

#start planning
success = gProb.SendCommand('MoveToHandPosition ' +↩
    SerializeTransformToCommand(tcpTrafo))

if success:
  WaitForController(gRobot)  #wait for robot motions

gRobot.GetEnv().LockPhysics(False)  #unlock mainthread
```

Listing 3.4: Skript zur Planung einer Manipulation mit Aufruf des IK Solver in Python

Zunächst muss immer die *openravepy* Bibliothek eingebunden werden. Sie enthält alle Funktionen für den Umgang mit OpenRAVE. Die ersten zwei Befehle erzeugen dann eine Umgebung und den passenden Viewer dazu. Mit dem Aufruf des Viewers startet automatisch eine Instanz der OpenRAVE GUI. *gEnv.Load('test.env.xml')* lädt die gewünschte Umgebung. Anders als bei Matlab kann ein Pointer direkt auf Roboter, sowie deren Manipulatoren oder KinBodys gerichtet werden. Mit *gRobot = gEnv.GetRobots()[0]* wird der erste Roboter des Environments ausgewählt. Die nächsten beiden Zeilen erstellen eine Probleminstanz und verbinden diese mit dem Roboter. Ab jetzt können Manipulationsaufgaben geplant werden. In Python ist es nötig, den Hauptthread für andere Prozesse zu blockieren, wenn eine Planung ausgeführt wird. Dies geschieht mit *gRobot.GetEnv().LockPhysics(True)*. Während dieses kritischen Abschnittes können keine Objekte vom User oder anderen Prozessen verschoben werden. *MoveToHandPosition* ruft den Inverskinematik-Löser auf und startet die Planung, falls eine Möglichkeit besteht, die ausgewählte Endeffektor-Transformation zu erreichen. Nachdem der Roboter die Aktion ausgeführt hat (*WaitForController(gRobot)*), muss der Hauptthread wieder freigegeben werden.

3.3 Weitere Einsatzmöglichkeiten für OpenRAVE

OpenRAVE ist nicht auf den Einsatz der Pfadplanung für Roboter mit Armen beschränkt. Durch die Pluginarchitekur ist es denkbar, Planer für jede Situation zu schreiben. So sind Bahnplanungsstrategien für komplexe Roboter ebenso möglich, wie Algorithmen für mobile Roboter zu Land sowie zur Luft. Auch Laufalgorithmen lassen sich mit Pfadplanung entwickeln und optimieren. Durch die Integration von OpenRAVE beim Institut für Robotik und Mechatronik könnten also noch mehrere Forschungsbereiche profitieren, als nur die Projektgruppe rund um Justin.



(a) Laserscan (b) Kamerabild

Abbildung 3.2: Visualisierung eines Laserscanners und eines Kamerabildes

Ebenso wie die Pfadplanung ist auch die Sensorik ein Teil von OpenRAVE. Mit virtuellen Sensoren und Kameras können Sichtbereiche der realen Geräte schon vor

der Montage optimiert werden. In der virtuellen Umgebung können Algorithmen zur Bildverarbeitung effektiver getestet werden. Sensordaten können in der Simulation direkt visualisiert und Kamerabilder eingeblendet werden. Abbildung 3.2 zeigt die graphische Darstellung einer Kamera und eines Laserscans. Die einzelne Betrachtung der jeweiligen Sensordaten ist ebenso realisierbar, wie eine Sensorfusion.

3.4 Besonderheiten bei OpenRAVE

Dieses Kapitel befasst sich mit einigen Besonderheiten und Ausnahmen von OpenRAVE. Die Inhalte der meisten aufgelisteten Absätze haben sich während der Arbeit mit OpenRAVE heraus kristallisiert. Die Sammlung der verschiedenen Themengebiete bildet also ein Nachschlagewerk mit nützlichen Hilfestellungen.

3.4.1 Matrix Notationen

Alle in OpenRAVE intern verwendeten Matrizen sind im *row-major Format*. Das bedeutet die Darstellung von Matritzen erfolgt auf herkömmliche, mathematische Weise. Die Matrix

$$\begin{pmatrix} r_{01} & r_{02} & r_{03} & t_0 \\ r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

wird demnach der Reihe nach im Speicher aufgelistet.

$$\begin{pmatrix} r_{01} & r_{02} & r_{03} & t_0 & r_{11} & r_{12} & r_{13} & t_1 & r_{21} & r_{22} & r_{23} & t_2 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

Alle serialisierten Matrizen, werden hingegen im *column-major Format* abgespeichert, also den Spalten nach:

$$\begin{pmatrix} r_{01} & r_{11} & r_{21} & 0 & r_{02} & r_{12} & r_{22} & 0 & r_{03} & r_{13} & r_{23} & 0 & t_0 & t_1 & t_2 & 1 \end{pmatrix} \tag{3.3}$$

Diese Konvention soll es vereinfachen, Matrizen in Matlab oder Octave zu verwenden. Diese arbeiten standardmäßig im column-major Format.

3.4.2 Debug-Level

Das Debug-Level spielt bei OpenRAVE eine beondere Rolle. Je nach Einstellung erhält der Nutzer bei der Ausführung eines Programmes verschiedene Informationen. Die Ausgaben sind in sechs verschiedene Stufen unterteilt: Fatal, Error, Warning,

Info, Debug und Verbose. Das Level kann wahlweise über die Menüleiste unter dem Punkt *View –> Debug Levels* oder per Skript aufgerufen werden. Dabei lauten die auswählbaren Level folgend:

- DebugLevel.Fatal
- DebugLevel.Error
- DebugLevel.Warning
- DebugLevel.Info
- DebugLevel.Debug
- DebugLevel.Verbose

Um das Debug-Level beispielsweise auf Verbose zu setzen, muss der Code wie in Listing 3.5 verwendet werden. Es ist sinnvoll das Debug Level so früh wie möglich zu setzen, um im Fehlerfall etwaige Probleme bereits beim Laden der Plugins zu erfahren.

```
from openravepy import *

#start of initialisation
gEnv = Environment()

#set debug level to verbose
debugLevel = DebugLevel.Verbose
gEnv.SetDebugLevel(debugLevel)

#rest of initialisation
gEnv.SetViewer('qtcoin')
gEnv.Load('test.env.xml')

#create problem instance
gRobot = gEnv.GetRobots()[0]
gProb = gEnv.CreateProblem('basemanipulation')
gEnv.LoadProblem(gProb,gRobot.GetName())
```

Listing 3.5: Setzen des Debug Levels mit Python

3.4.3 Physikmodule und Gravitation

Ein wichtiger Punkt für die Zukunft könnte die Simulation der physikalischen Welt um Justin sein. Diese umfasst Gravitation, Impulssätze und verschiedene Kräfte.

Bei der Definition von KinBodys wird zwischen statischen (*static*) oder dynamischen Objekten (*dynamic*) unterschieden. Statische Gegenstände zeigen sich unbeeindruckt von äußeren Einflüssen. Hindernisse wie der Untergrund oder Tische, die nicht verschoben werden sollen, müssen dieses Attribut tragen. Objekte, die zur Manipulation freigegeben werden sollen, müssen als dynamisch deklariert werden. Kollisionen führen dann zu Bewegungen und die Gravitation sorgt für realisitische Flugbahnen. Auch der Roboter und seine Manipulatoren werden dadurch beinflusst. Fehlt die Gegenkraft der Motoren in den Gelenken, so sackt die gesamte Konstruktion einfach in sich zusammen. Mit den richtigen Parametern kann das Modell hingegen ein ganzes Stück näher an die Realität gebracht werden.

3.4.4 Der Hauptthread als kritischer Bereich in OpenRAVE

Eine der größten Schwierigkeiten in OpenRAVE ist die Handhabung des Hauptthreads. Während der Planungszeit muss dieser immer mit dem Befehl *environment.LockPyhsics(True)* geblockt werden, um Konflikte zu vermeiden. Die Planungsalgorithmen greifen auf die OpenRAVE Kernbibliothek zu, die nur exklusiv verwendet werden darf. Greifen mehrere Threads bzw. Prozesse oder Benutzer auf die gleiche Umgebung zu, so können *race conditions* auftreten und das Skript bleibt in einem *deadlock* stehen. Insbesondere wenn mehrere Prozesse unabhängig voneinader arbeiten, treten derartige Fehler auf.

Es ist daher davon abzuraten den *clone*-Befehl zu verwenden, um die Umgebung in OpenRAVE zu multiplizieren. Beim Klonvorgang werden zwar alle Roboter und Objekte eines Environments kopiert, allerdings bleibt die verwendete OpenRAVE Instanz dieselbe. Sogar wenn der Prozess mittels *fork* aufgeteilt wird, verweisen die Filedeskriptoren der einzelnen Prozesse noch auf dieselbe Instanz. Um dieses Problem zu umgehen, muss der Prozess schon vor dem Erstellen der OpenRAVE-Umgebung geteilt werden. Jeder Prozess kann so eine eigene, völlig unabhängige OpenRAVE Instanz starten. So oder so müssen beide Umgebungen aufeinander abgestimmt werden, um Kollisionen der beiden Arme, oder von ihnen verschobenen Objekten, zu vermeiden. Dieser Vorgang wird näher in Kapitel 5.1.3 erläutert.

3.5 Justin als Evaluierungsplattform in OpenRAVE

Justin, als anthropomorpher Roboter, ist die ideale Versuchsplattform für zweiarmige Manipulation. Zusätzlich machen ihn diese Eigenschaften auch zu einer prädestinierten Evaluierungsumgebung für OpenRAVE. Die komplexe Struktur und ein detailliertes Modell des Roboters können die Stärken und Schwächen der Simulationsumgebung ans Tageslicht bringen.

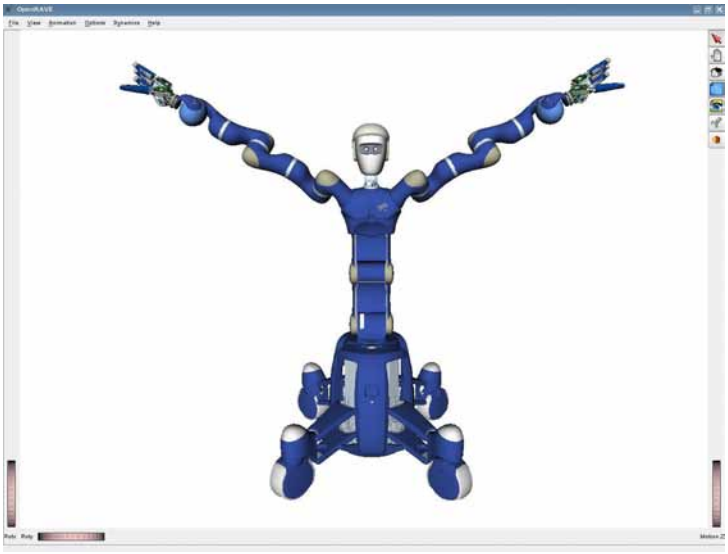
In den meisten 3D-Simulationsumgebungen werden Roboter mittels Denavit-Hartenberg Parametern definiert [8], [7]. Nach dieser Konvention wird der Aufbau eines Roboters mittels Überführung von Ortskoordinatensystemen der einzelnen Elemente dargestellt. Die vier Parameter zur Beschreibung dieser Translationen und Rotationen sind die Segmentlänge, die Segmentverdrehung, der Segmentoffset und der Gelenkwinkel. Auch Justin wurde bisher in DH-Paramtern nach Craig definiert. Da dies nicht sehr intuitiv und auf den ersten Blick nicht sehr durchschaubar ist, werden in OpenRAVE die Robotermodelle anders aufgebaut. In OpenRAVE wird Justin als XML File justin.robot.xml in Collada-Codierung abgespeichert. Diese hat den grundsätzlichen Aufbau der folgenden OpenRAVE Standard Roboterdefinition.

```
<robot>
  <kinbody file="arm.kinbody.xml"/>
  <kinbody file="hand.kinbody.xml"/>
  <attachedsensor>
  ...
  </attachedsensor>
  <manipulator>
  ...
  </manipulator>
</robot>
```

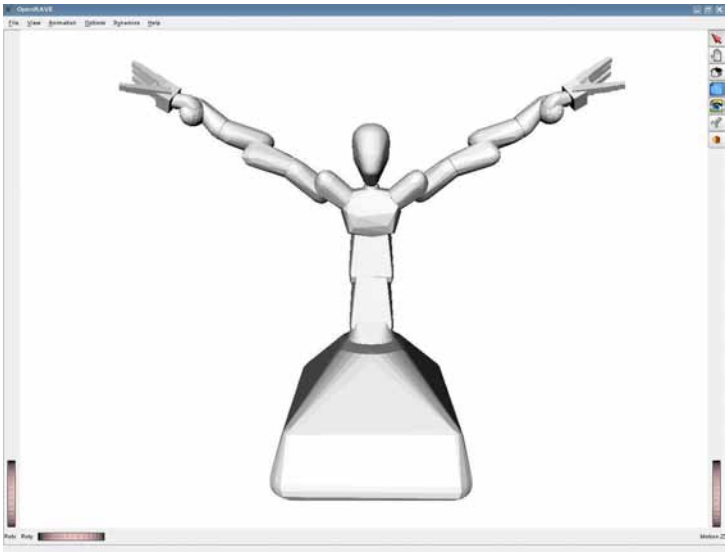
Listing 3.6: Standard Roboter Definition

Die einzelnen Elemente, auch *Links* genannt, werden als eigenständiges Inventor-File erstellt und dann in der Hauptdatei zusammengesetzt. Die Transformation eines Elements zum Nächsten wird dabei, ausgehend von einer Offset-Position, lediglich mit einer Rotation und einer Translation beschrieben. An der Stelle, an der sich zwei Links treffen, wird ein Gelenk oder auch *Joint* definiert. Die Beschreibung der Joints ist gleich der realen kinematischen Beschreibung von Justin. Alle Gelenkwinkelgrenzen und die Abhängigkeiten der gekoppelten Fingerelemente sind eingepflegt. Nur die Torsoabhängigkeit bildet eine Ausnahme, da derart komplexe Konstrukte in OpenRAVE nicht vorgesehen sind.

Wie gewöhnliche KinBodys besteht jeder Link aus einem Renderingfile und einem Datafile. Das Renderingfile ist zuständig für die Darstellung des Roboters während das Datafile das Kollisionsmodell hinterlegt. Es ist nicht immer notwendig, dass das Kollisionsmodell genau so detailreich ist wie das Renderingmodell. Umso detailierter das Kollisionsmodell ist, desto aufwendiger wird die Kollisionsberechnung. Das aktuelle Kollisionsmodell von Justin besteht für den Körper und die Arme aus *konvexen Hüllen*. Um einen zusätzlichen Sicherheitsabstand zu erhalten, sind die Hüllen der Arme 10% größer skaliert. Die Hände werden als *Boundingboxen* dargestellt, um den Abstand zu den Hindernissen zu erweitern. Abbildung 3.3 zeigt Justins Rendering- und Kollsionsmodell im Vergleich.



(a) Renderingmodell



(b) Kollisionsmodell

Abbildung 3.3: Justins Rendering- und Kollsionsmodell im Vergleich

Im Anschluss an die Vorwärtskinematik von Justin werden angrenzende, sich berührende Gelenke mit dem *adjacent* Tag aus der Kollisionsüberprüfung ausgeschlossen. Die Manipulatordefinition umfasst drei Manipulatoren. Zum ersten den Kopf und weiterhin die beiden Arme. Bei den Armen wird neben der Greifkinematik auch noch eine Inverskinematik ausgewählt. Der derzeit verwendete *IKFast Solver* wurde direkt mit einem OpenRAVE Plugin aus der Vorwärtskinematik erzeugt. Teile der Kinematikbeschreibung von Justin können im Anhang A.1 eingesehen werden.

4 Evaluierung von OpenRAVE anhand einarmiger Manipulationsaufgaben

Heutzutage werden Roboter noch hauptsächlich in der Industrie eingesetzt. Dabei handelt es sich meistens um einen einzelnen Roboterarm, der in einem speziellen abgeschlossenen Bereich agiert (siehe Abbildung 4.1). Er muss keinen Hindernissen ausweichen und hat immer die gleichen Aufgaben zu erledigen. Dabei sind die abzufahrenden Pfade fest einprogrammiert. Eine Kollision ist unwahrscheinlich, da sich die Umgebung während sich der Roboter bewegt, nicht verändert. Das Werkstück wird in die Kabine gefahren und der Roboter beginnt die Arbeit. Dies, aber vor allem die Tatsache, dass der Roboter sich den Arbeitsraum nicht teilen muss, vereinfacht die Aufgabenplanung ungemein. Bevor die Planung über zwei Arme an-

Abbildung 4.1: Industrie Roboter im Einsatz [25]

gegangen wird, muss zunächst die Tauglichkeit von OpenRAVE für die einarmige Planung überprüft werden. Dazu ist es nötig, einige Szenarien zu erstellen, welche die Software auf Handhabung und Ausführungszeit hin prüfen. Um Wechselwirkungen der beiden Arme auszuschließen, werden diese Szenarien speziell für die einarmige Planung konzipiert. Allem voran ist es wichtig, dass die Trajektoriengüte und deren Planungszeit in einem verträglichen Rahmen bleiben. Dauert die Planung in den Szenarien zu lange, so sind die Algorithmen nicht einsetzbar.

4.1 Bedingungen für die Versuchsreihen der einarmigen Manipulation

Die Planung wird über sieben Freiheitsgrade durchgeführt, also nur über den rechten Arm. Der linke Arm wird in eine Position gebracht, in der er den Arbeitsraum des ersten Armes nicht berührt. Die Finger bleiben während der Planung steif. In den Versuchsszenarien ist jeder Zeit genug Platz, um die Finger in der Startkonfiguration zu belassen. Der gewünschte Manipulator wird mit dem Befehl *robot.SetActiveManipulator(manipulatorID)* ausgewählt. Dieser ist gleichwertig mit dem Befehl *SetActiveDOFs(dofList)*, bei dem eine Liste der zur Planung verwendeten Gelenke übergeben wird. Die Versuche zur einarmigen Manipulation bestehen alle aus sich wiederholenden Greifoperationen. Der Arm wird dabei in eine kollisionsfreie Ausgangsposition für einen Griff gebracht. Das eigentliche Greifen des Objekts ist nicht Bestandteil der Versuche. Zwischen den Operationen wird weder Justins Körper bewegt, noch dessen Position im Raum verändert. Nach einer ausgeführten Trajektorie wird der Manipulator wieder an die Ausgangsposition zurückgesetzt. Die Interpolation zwischen zwei Pfadpunkten wird mit einer Auflösung von 1° durchgeführt. In jedem Interpolationsschritt findet eine Kollisionsabfrage statt.

Für alle Szenarien wird die gleiche Testumgebung verwendet. Diese besteht aus einem Intel Xeon W3520 Prozessor mit 2.67 GHz Taktfrequenz und 8192 KB Cache. Der Hauptspeicher umfasst 3 GB Speichervolumen. Als Betriebssystem wird openSUSE 11.x eingesetzt. Alle Versuche werden mit SVN Revision 1000 von OpenRAVE durchgeführt.

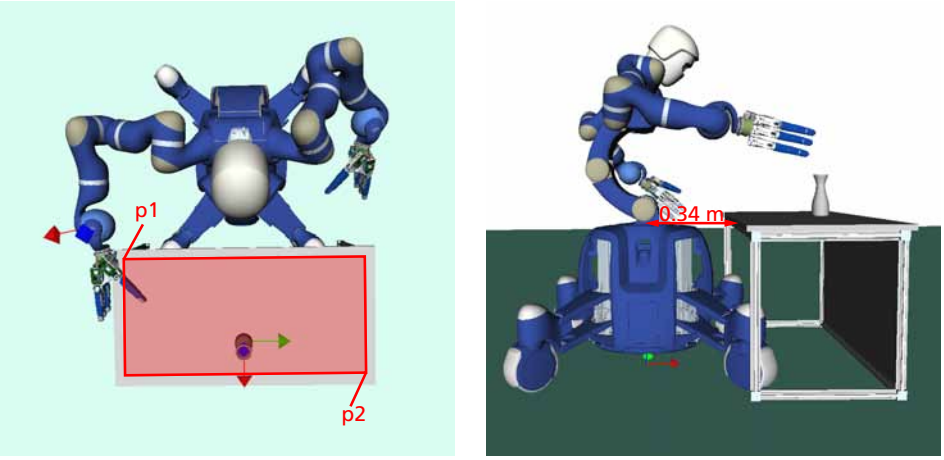
4.2 Szenario I – Pfadplanung bei einer hindernislosen Arbeitsfläche

Das erste Szenario soll verdeutlichen, wie stark die Planungszeit von der Griffrichtung und der Platzierung des Ziels abhängt. Dabei stehen zunächst keine Hindernisse im Weg, die die Planungszeit eventuell verlängern würden. Untersucht werden die Trajektoriengüte und deren Planungszeit. Außerdem soll die IKFast Inverskinematik getestet werden.

4.2.1 Aufbau

Das erste Szenario zur Validierung von OpenRAVE ist sehr einfach aufgebaut. Es besteht nur aus einer Umgebung mit einem Tisch. Auf dem Tisch befindet sich eine Karaffe, die später als Zielobjekt für die Griffoperationen fungiert. Die Karaffe sowie alle anderen verwendeten Gegenstände sind direkte Ebenbilder der Originalobjekte, mit denen Justin zur Zeit hantiert. Die Karaffe ist 21 cm hoch und hat einen maximalen Durchmesser von 9 cm. Justin steht zentriert vor dem Tisch, mit einem

Abstand von 0.34 m zwischen Justins Basiskoordinaten und der Kante des Tisches. Das Weltkoordinatensystem und Justins Basiskoordinatensystem sind äquivalent. Mit den Tischmaßen von 0.766 x 1.28 x 0.68 m (H x B x T), befindet sich die effektive Arbeitsfläche zwischen den kartesischen Punkten $p1 = (0.48, -0.60, 0.766)$ und $p2 = (1.08, 0.60, 0.766)$. Die beschriebenen Gegebenheiten sind in Abbildung 4.2 illustriert. Justins Ausgangsstellung ist bei jeder Planungsphase gleich. Somit



(a) Draufsicht (b) Seitenansicht

Abbildung 4.2: Draufsicht und Seitenansicht von Szenario I

sind auch die Bedingungen immer gleich. Die Gelenkwinkelparameter in Grad sind in Tabelle 4.1 aufgelistet. Die eingesetzte Inverskinematik ist die IKFast Variante von OpenRAVE. Das erste Szenario soll zeigen, ob der Aktionsraum von Justin, der durch die IKFast erreicht wird, mit dem der bisher verwendeten DLR-Inverskinematik übereinstimmt.

Körperteil	Gelenkwinkelparameter in Grad
Torso	0.00 -48.85 77.17 -28.32
Rechter Arm	-12.42 -62.37 -3.07 93.29 56.90 -3.89 12.62
Rechte Hand	0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Linker Arm	-24.37 -89.88 5.01 90.00 35.00 -9.96 39.98
Linke Hand	0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Kopf	0.00 13.00

Tabelle 4.1: Startkonfiguration für das erste einarmige Szenario

4.2.2 Ablauf

Zu Beginn des Szenarios wird Justin in die zuvor gelistete Ausgangsposition gefahren. Anschließend wird eine Zufallsposition für die Karaffe generiert. Ausgehend von Justins Startposition wird nun versucht den rechten Manipulator mittels der Inverskinematik in Griffweite der Karaffe zu positionieren. Der Endeffektor soll zunächst so ausgerichtet werden, dass dessen Y-Achse parallel zur X-Achse der Karaffe steht. Der daraus resultierende Greifframe findet sich in in Anhang A.2. Ist dies möglich, wird ein Zähler auf eins gesetzt. Im nächsten Schritt wird die Hand um ein Grad gegen den Uhrzeigersinn um die Z-Achse der Karaffe gedreht und die Inverskinematik wird erneut berechnet (siehe Abbildung 4.3). Der Zähler wird bei erfolgreicher Berechnung der Inverskinematik inkrementiert.

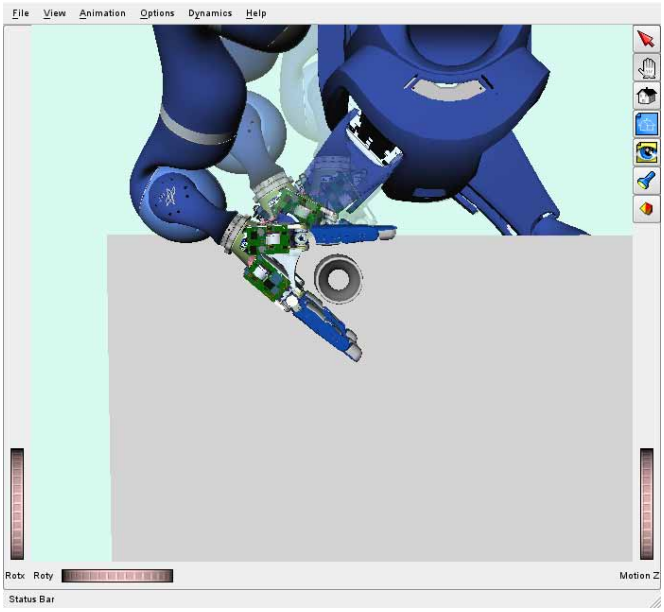


Abbildung 4.3: Schematischer Ablauf des Rotationsschritts um die Z-Achse des Zielobjekts

Zusätzlich wird alle 30° eine Trajektorie zur Zielkonfiguration geplant und falls diese realisierbar ist auch abgespeichert. Sobald alle Posen rund um das Ziel auf Realisierbarkeit überprüft wurden, wird ein farbkodierter Marker auf den Tisch gesetzt. Je nach Farbwert symbolisiert dieser die Erreichbarkeit des Objekts an der jeweiligen Stelle des Tisches. Im Anschluss wird wieder eine neue Position für das Ziel generiert und der Zyklus beginnt von vorne. So entsteht nach und nach eine Erreichbarkeitskarte auf der Ebene des Tisches (siehe Abbildung 4.4). Im Vergleich mit früheren Tests mit der bisher verwendeten Inverskinematik kann dadurch festgestellt werden, ob die verwendete Inverskinematik mit der des realen Roboters übereinstimmt.

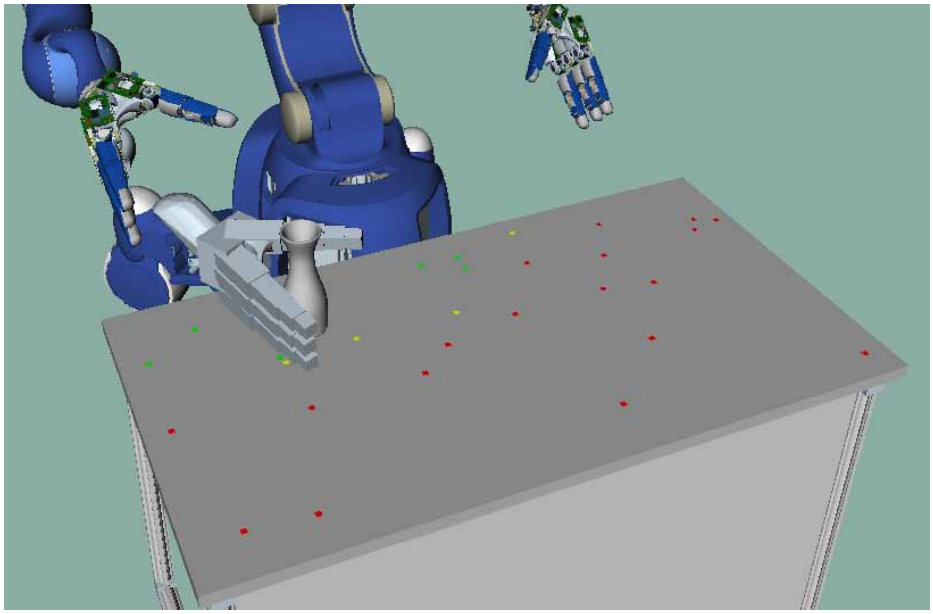


Abbildung 4.4: Entstehung der Erreichbarkeitskarte

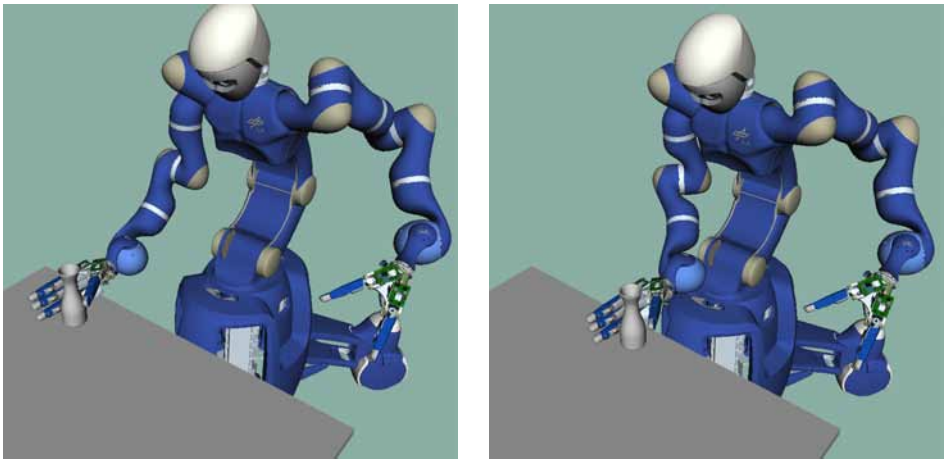
4.2.3 Ergebnis

Die Ergebnisse des ersten Testszenarios soll in erster Linie der Evaluierung von OpenRAVE als Planungsumgebung dienen. Daher ist es wichtig, dass die Trajektorien objektiv bewertbar sind. Aus diesem Grund wird zu jeder geplanten Trajektorie die Planungszeit und die Anzahl der benötigten Konfigurationen aufgezeichnet. Die Aneinanderreihung der einzelnen Konfigurationen ergeben die realisierbare Trajektorie. Der dabei zurückgelegte Weg gibt Aufschluss über die Bewegung des Arms. Für einfache Szenarien wie hier vorliegend, kann angenommen werden, dass ein Pfad mit wenigen Punkten wenig Raum überschritten hat. Werden hingegen viele Konfigurationen benötigt, so hat der Manipulator viel Raum überschritten, der für das Lösen der Aufgabe nicht relevant ist. Solche Bewegungen zeichnen sich oft durch Umkonfigurationen des Arms aus, die unnatürlich wirken. Folgende Werte haben sich für das erste Szenario in 100 Durchläufen ergeben.

	durchschnittliche	Zeit in Sekunden		
	Trajektorienlänge	Min.	Max.	Mittelwert
Szenario I	89	5.55	25.02	11.97

Tabelle 4.2: Numerische Ergebnisse für Szenario I

Alle Zeiten verstehen sich inklusive Optimierung der Trajektorien und Ausführung in der Simulation bei einer maximalen Gelenkwinkelgeschwindigkeit von *90°/s*. Durchschnittlich dauert die Planung ca. 12 Sekunden. Die großen Unterschiede in der Minimal- und Maximalzeit entstehen durch die verschiedenen Bereiche, in denen sich das Zielobjekt aufhalten kann. Steht die Karaffe nah am Körper und der Griffwinkel ist derart ungünstig, dass die Hand wenig Spielraum hat, so entsteht ein *„narrow passage" Problem* an der Zielkonfiguration. Der Oberkörper des Roboters schränkt die Bewegungsfreiheit des Armes ein. Steht die Karaffe weiter entfernt, kann diese von allen erreichbaren Seiten nahezu gleich schnell gegriffen werden (siehe Abbildung 4.5). Eine Einschätzung darüber ob und wie ein Objekt auf dem Tisch gegriffen werden kann, liefert die Erreichbarkeitskarte in Abbildung 4.6. Diese zeigt auf der linken Seite den Tisch in der Draufsicht und rechts dazu die Legende der Farbwerte.



(a) ohne narrow passage (b) mit narrow passge

Abbildung 4.5: Narrow passage Problem durch Zielkonfigurationen nahe am Torso.

Objekte im roten Bereich liegen außerhalb des Manipulator-Arbeitsbereichs. Nur wenn der Oberkörper oder die mobile Basis eingesetzt wird, kann dieser Raum erreicht werden. Die darauf folgende orangefarbene Linie zeigt den schmalen Bereich, in dem Objekte am Rande des Arbeitsraumes erreichbar sind. Befindet sich dort ein Zielobjekt, kann es nur in einem kleinen Winkelausschnitt und mit ausgestrecktem Arm gefasst werden. In den inneren Bereichen kann Aufgabenplanung wesentlich effektiver eingesetzt werden, da dort ein möglicher Zielgegenstand in einem größeren Winkel erreichbar ist. Der visualisierte Bereich der Erreichbarkeit deckt sich mit früheren Beobachtungen [36]. Die Erreichbarkeit der IKFast Inverskinematik ist also kompatibel mit dem bisher verwendeten Modul.

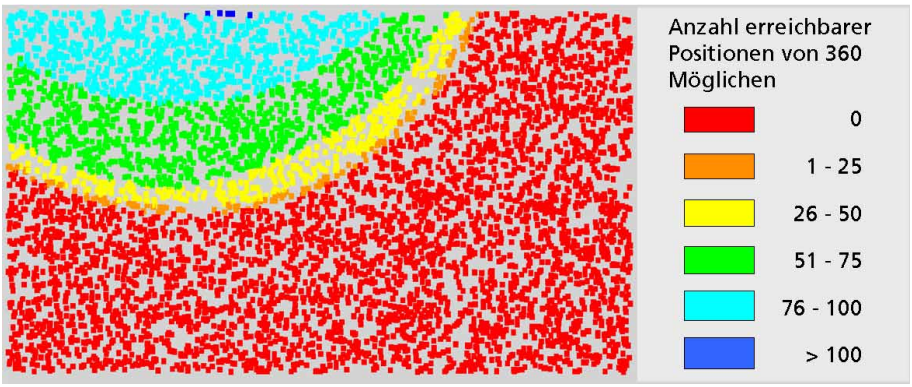


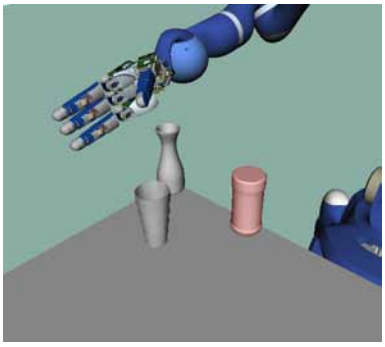
Abbildung 4.6: Erreichbarkeitskarte des rechten Arms.

4.3 Szenario II – Pfadplanung bei einer mit Hindernis versehenen Arbeitsfläche

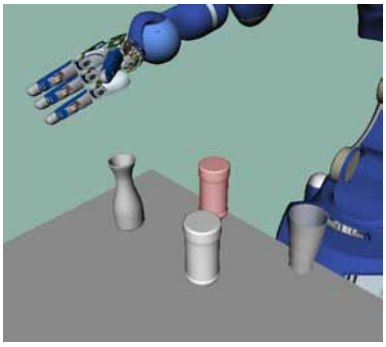
Im zweiten Szenario werden die Eigenschaften der Pfadplanung und der daraus resultierenden Trajektorien untersucht, wenn die Umgebung des Roboters mit Hindernissen zugestellt ist. Um festzustellen, wie sehr die Planungszeit unter den zusätzlichen Gegenständen leidet, wird deren Anzahl Schritt für Schritt erhöht. Auch die Positionierung kann ausschlaggebend für die Planungszeit sein.

4.3.1 Aufbau

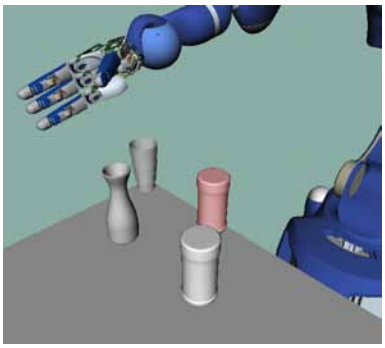
Der grundsätzliche Aufbau des Versuchs ähnelt sehr dem des ersten Szenarios. Die Positionierung von Justin und dem Tisch sind identisch. Auch die Ausgangsposition ist gleich der in Tabelle 4.1. Lediglich das Ziel und die darum befindlichen Hindernisse unterscheiden sich hier. Zu der in Szenario I verwendeten Karaffe kommen nun noch ein Glas und eine Teebox. Das Glas hat eine Höhe von 16 cm und einen maximalen Durchmesser von 8.7 cm. Die Teebox ist 18 cm hoch und weist einen Durchmesser von 9 cm auf. Die Teebox dient in den folgenden Versuchen immer als Zielobjekt für die Pfadplanungen. Hierbei werden vier Fälle unterschieden. Der erste Fall beinhaltet drei Objekte, eine Karaffe, ein Glas, und die Teebox (Abbildung 4.7 Fall 1). Die Fälle zwei und drei beinhalten zusätzlich eine weitere Teebox als Hindernis, zunächst links des Ziels (Fall 2) und anschließend rechts des Ziels (Fall 3). Der vierte Fall besteht dann aus fünf Objekten. Dabei steht der Zielgegenstand mitten unter den Hindernissen (siehe Abbildung 4.7 Fall 4). Die Ausgangslage der vier verschiedenen Hindernismuster wird in Tabelle 4.3 beschrieben.



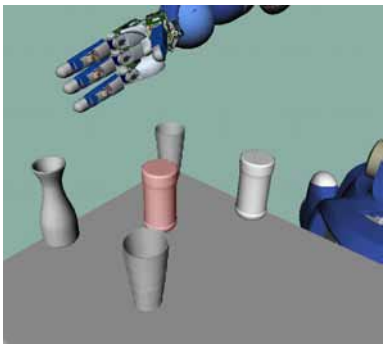
(a) Fall 1



(b) Fall 2



(c) Fall 3



(d) Fall 4

Abbildung 4.7: Verschiedene Objektpositionen für Szenario II

	(X, Y) - Positionierung in Metern				
	Hindernis 1	Hindernis 2	Hindernis 3	Hindernis 4	Ziel
	Glas	Karaffe	Teebox	Glas	Teebox
Fall 1	(0.58, -0.45)	(0.38, -0.00)	-	-	(0.38, -0.30)
Fall 2	(0.58, -0.45)	(0.58, -0.45)	(0.58, -0.15)	-	(0.38, -0.30)
Fall 3	(0.38, -0.60)	(0.58, -0.15)	(0.58, -0.30)	-	(0.38, -0.30)
Fall 4	(0.38, -0.60)	(0.78, -0.60)	(0.38, -0.30)	(0.78, -0.30)	(0.58, -0.45)

Tabelle 4.3: Ausgangsposition der Objekte im zweiten Testszenario. Die Translation in Z-Richtung beträgt jeweils 0.77m

4.3.2 Ablauf

Wie auch im ersten Szenario wird versucht, eine geeignete Griffposition für das Ziel zu finden. Der Roboter ist dabei im Aktionsbereich eingeschränkt, da die umste-

henden Gegenstände den direkten Zugriff eventuell nicht erlauben. Dabei können durchaus Inverskinematik-Lösungen existieren, wegen der umstehenden Hindernisse können diese aber kollisionsbehaftet sein. Es wird abermals versucht, das Zielobjekt aus jeder Richtung zu erreichen. In 5°-Schritten wird eine Planungsphase eingeleitet und die erhaltene Trajektorie abgespeichert. Anders als bei Szenario I werden hier die Objekte nicht zufällig positioniert. Die Hindernisse bleiben in relativem Abstand zum Zielobjekt stehen und das gesamte Setup wird auf dem Tisch zunächst in Y-Richtung und dann in X-Richtung versetzt (siehe Abbildung 4.8). Der Vorgang wird so lange wiederholt, bis das Setup bestehend aus Ziel und Hindernissen den Tisch komplett abgefahren hat.

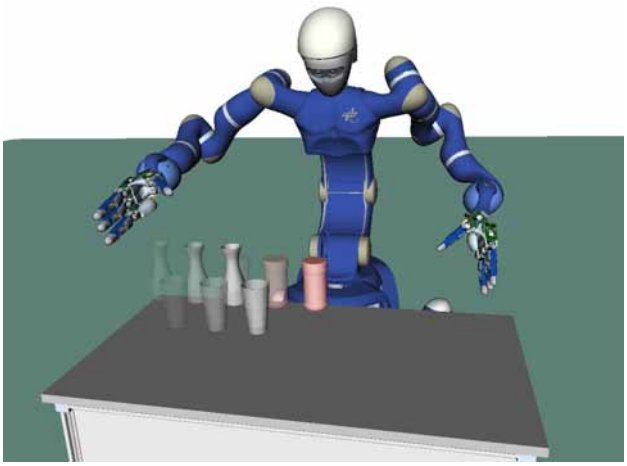


Abbildung 4.8: Ablauf des zweiten Szenarios

4.3.3 Ergebnis

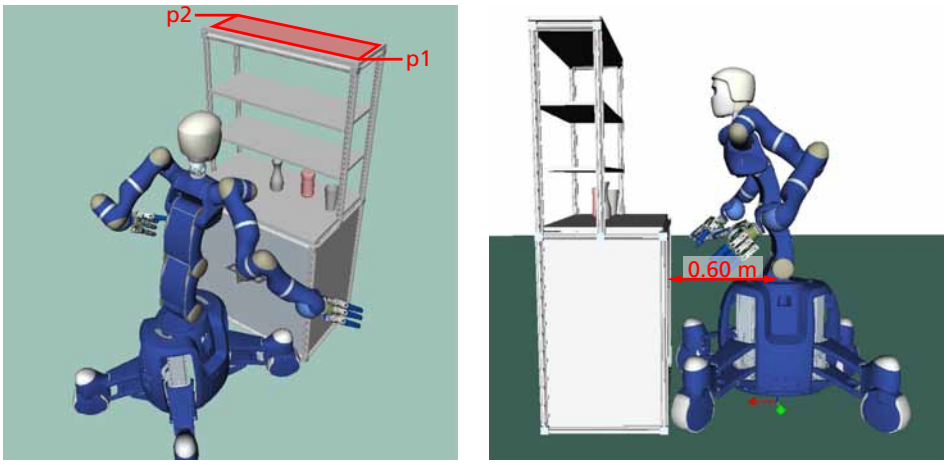
Die Trajektorien werden nach dem Verfahren aus Szenario I bewertet. Der Unterschied der Zeiten in den verschiedenen Versuchsreihen zeigt, wie stark die Planungszeit von den umliegenden Objekten abhängig ist. Der Mittelwert in Fall 1 ist mit 16.78 Sekunden schon deutlich über dem aus Szenario I (11.97 Sekunden). Aber noch interessanter ist der Vergleich mit dem zweiten Fall. Bei diesem ist der Raum rechts des Ziels leer. Anders als bei den anderen Fällen kann der rechte Manipulator, von der rechten Seite ungestört an das Zielobjekt gelangen. So kommt die Pfadplanung relativ schnell zum Ziel, da weniger Kollisionen die Planung behindern. Dies fällt vor allem bei Fall vier ins Gewicht. Hierbei ist das Ziel von allen Seiten blockiert. Die dadurch entstehende *„narrow passage“* ist maßgeblich für die lange Planungszeiten von durchschnittlich 22.06 Sekunden. Auffällig ist, dass beim letzten Fall nur sehr kurze Trajektorien entstehen. Dies ist dadurch zu erklären, dass es durch die vier Hindernisse nur wenige Wege gibt, die zum Erreichen der Teebox führen.

	durchschnittliche	Zeit in Sekunden		
	Trajektorienlänge	Min.	Max.	Mittelwert
Fall 1	125	5.95	49.60	16.78
Fall 2	100	6.07	38.36	14.26
Fall 3	119	6.34	44.64	16.95
Fall 4	75	10.09	45.64	22.06

Tabelle 4.4: Numerische Ergebnisse für Szenario II

4.4 Szenario III – Pfadplanung bei einer Regalwand

Das letzte Szenario für die einarmige Manipulation unterscheidet sich sehr von den bisher untersuchten Umfeldern. Der bisher verwendete Tisch wird durch ein Regal ersetzt. Durch die unterschiedlichen Höhen, die durch die Regalböden festgelegt werden, wird der Arbeitsraum des Manipulators wesentlich besser ausgenutzt. Anhand dieses Beispiels soll gezeigt werden, wie groß die Erreichbarkeit des Armes tatsächlich ist. Zwei Hindernisse links und rechts des Ziels erschweren jedoch die Pfadplanung. Der Roboter hat es an jeder Zielposition mit einem narrow passage Problem zu tun. Wie weit die Planungszeit dadurch beeinträchtigt wird und ob die Trajektorien von einer natürlichen Bahn abweichen, soll dieser Abschnitt zeigen.



(a) Übersicht (b) Seitenansicht

Abbildung 4.9: Szenario III in der Übersicht und von der Seite

4.4.1 Aufbau

Betrachtet man zunächst nur die Anordnung der Objekte in Abbildung 4.9 erinnert auch dieses Szenario an die zuvor betrachteten Probleme. Der größte und wichtigste Unterschied ist aber der Austausch des Tisches durch ein Regal. Das Regal umfasst vier Ebenen. Der Vergleichbarkeit halber wird die unterste Ebene nicht komplett ausgenutzt. Nur der Bereich unterhalb der Regalböden wird als Ablage verwendet. Die Regalböden haben von unten nach oben die Höhen 1.074 m, 1.364 m, 1.674 m und 2.054 m. Bei einer Dicke der Regalböden von 0.019 m, ist der nutzbare Raum zwischen den Böden also 0.271 m, 0.291 m und 0.361 m hoch. Die oberste Ablagefläche wird allein durch die maximale Ausdehnung des Arbeitsraumes begrenzt. Allen gemeinsam ist die effektive Ablagefläche von 0.27m^2 die sich über den beiden kartesischen Punkten $p1 = (-0.450.95)$ und $p2 = (0.451.25)$ aufspannt[3]. Die Ausgangsposition der Objekte findet sich in Tabelle 4.5.

	(X, Y) - Positionierung in Metern		
	Hindernis 1	Hindernis 2	Ziel
	Glas	Karaffe	Teebox
3 Objekte	(1.00, -0.40)	(1.00, 0.00)	(1.05, -0.20)

Tabelle 4.5: Ausgangsposition der Objekte im dritten Testszenario. Die Translation in Z-Richtung beträgt 1.075 m

Körperteil	Gelenkwinkelparameter in Grad
Torso	0.00 14.32 0.00 -14,32
Rechter Arm	-33.67 -73.67 -0.296 58.11 13.02 -23.49 12.62
Rechte Hand	0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Linker Arm	-24.37 -89.88 5.01 90.00 35.00 -9.96 39.98
Linke Hand	0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Kopf	0.00 0.00

Tabelle 4.6: Startkonfiguration für das dritte, einarmige Szenario

Wie schon in den vorherigen Szenarien steht Justin direkt auf dem Nullpunkt. Das Weltkoordinatensystem und Justins Basiskoordinatensystem sind also äquivalent.

[3]Die Höhe variiert und ist daher nicht aufgeführt. Die Höhen können den vorherigen Angaben entnommen werden.

Der Abstand zum Regal ist allerdings etwas größer, als der zu dem bisher verwendeten Tisch. Zwischen der vordersten Kante des Regals bis zu Justins Zentrum liegen 0.65 m. Für eine bessere Gesamterreichbarkeit ist der Torso mit einer aufrechten Ausgangskonfiguration initialisiert. Außerdem ist der planende rechte Arm in einer anderen Startstellung um nicht in Kollision mit dem Regal zu geraten. Die neue Ausgangsstellung ist noch einmal komplett in Tabelle 4.6 einzusehen.

4.4.2 Ablauf

Wie beim zweiten Planungsszenario wird auch hier wieder das gesamte Setup, bestehend aus den zwei Hindernissen und dem Zielobjekt, verschoben. Dabei wandern die Objekte von der untersten Ebene zur obersten. Die Gegenstände werden auch in den hinteren Bereichen platziert. So kann später eingesehen werden, wie weit der Arm kollisionsfrei in das Regal vordringt. Die Inverskinematik wird wie auch zuvor für jede Position um das Ziel herum berechnet. Da zu erwarten ist, dass relativ wenige Konfigurationen zum Ziel gelangen, plant das Skript zu jeder zweiten Inverskinematik eine Bahn und speichert die daraus resultierende Trajektorie ab. Erfolgreich berechnete Inverskinematiklösungen werden als blauer Pfeilmarker auf dem Regal gekennzeichnet, unerreichbare Konfigurationen erhalten einen roten Pfeilmarker. Zu jeder Position des Setups im Regal entsteht dadurch ein zweifarbiges Kuchendiagramm. Das Ergebnis der Erreichbarkeit ist in Abbildung 4.10 zu sehen.

4.4.3 Ergebnis

Wie zu erwarten, variiert die Erreichbarkeit auf den verschiedenen Ebenen des Regals. Während auf der untersten Fläche einige Ziele unerreichbar sind, kann das Ziel in den mittleren beiden Regalböden überall erreicht werden. Auf der obersten Ablage ist Justin nur noch an zwei Positionen in der Lage die Teebox zu greifen. Dennoch zeigt dieses Ergebnis, die enorme Reichweite von Justin. Obwohl der Oberkörper nicht bewegt wird, kann die Erreichbarkeit der Gegenstände im und auf dem Regal beinahe zu 75% abgedeckt werden (siehe Abbildung 4.10). Durch eine bessere Benutzung und Positionierung des Oberkörpers wäre es möglich, Justin auf allen Ebenen des Regals Objekte greifen zu lassen.

	durchschnittliche	Zeit in Sekunden		
	Trajektorienlänge	Min.	Max.	Mittelwert
3 Objekte	186	13.19	99.10	42.00

Tabelle 4.7: Numerische Ergebnisse für Szenario III

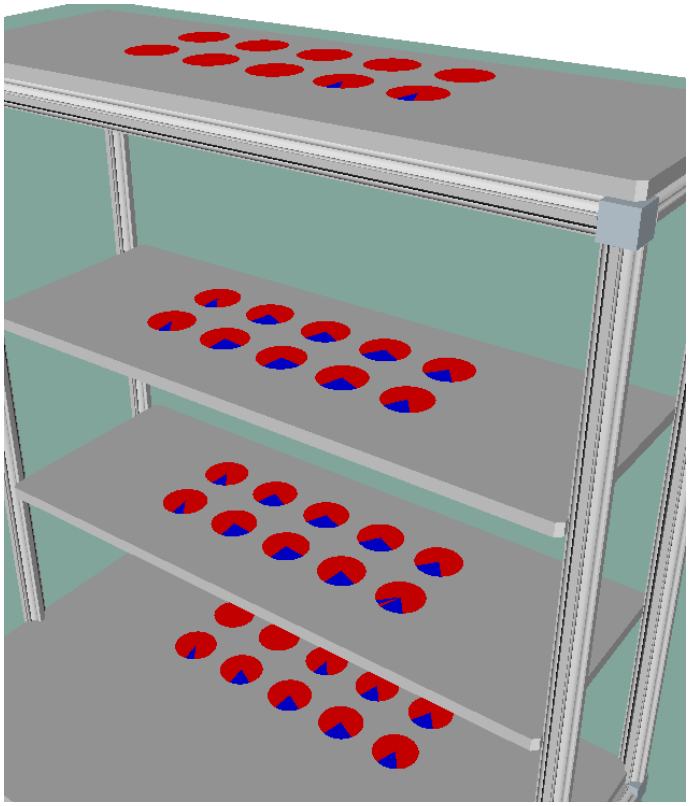


Abbildung 4.10: Erreichbarkeit des Zielobjekts innerhalb und auf dem Regal

Die Hindernisse links und rechts des Ziels behindern zwar die Planung, dennoch ist ein Griff prinzipiell möglich. Leider wirken sich die zusätzlichen Objekte aber negativ auf die Planungszeit aus. Der Mittelwert liegt mit 42 Sekunden weit über den Werten der vorherigen Szenarien. Offensichtlich ist es schwer für den Planer einen direkten Weg zur Zielkonfiguration zu finden. Die durchschnittliche Trajektorienlänge von 186 Konfiguration weißt auf die gemachten Beobachtungen zur Trajektoriengüte hin. Die Pfade führen oft zunächst am Ziel vorbei um dann umständlich doch zum Objekt zu finden. Dabei wird viel Raum überstrichen der zum Lösen der Aufgabe nicht relevant ist.

4.5 Auswertung der einarmigen Szenarien

Die Ergebnisse für die einarmige Manipulation sind vielversprechend. Durch einige Optimierungen können diese durchaus noch an Geschwindigkeit zulegen. Dies ist besonders relevant für die zweiarmige Manipulation. Dadurch, dass sich mit jedem Freiheitsgrad der Suchraum exponentiell vergrößert, liegt die Vermutung nahe, dass die Planungszeit nichtlinear ansteigt, wenn mit 14 Freiheitsgraden geplant wird. Für die einfachen Fälle aus Szenario I und II reicht die beobachtete Trajektoriengüte aus.

Die Bahnen waren durchweg flüssig und es sind so gut wie keine ruckartigen Richtungsänderungen aufgetreten. Es traten also keine Umkonfigurationen auf. Nicht so im letzten Szenario. Die Bahnen waren oft unnötig lang und haben für die Aufgabe irrelevante Bereiche betreten. So wurde das Ziel oft von der falschen Seite angefahren. Die schnellen Richtungsänderungen in der Nähe der Hindernisse würden in der Realität zu unfreiwilligen Überschwingern führen. Die Gegenstände könnten eventuell beschädigt werden. Der Schwerpunkt dieser Arbeit liegt auf der Strukturierung und Parametrisierung des Bahnplaners für einfache, alltägliche Aufgabenszenarios. Engpässe stellen hierbei eher einen Spezialfall dar. Im weiteren Verlauf dieser Arbeit werden derartige Engpässe daher vermieden. Um diese Art von Problemen sicherer und schneller zu lösen, müssen andere Ansätze verfolgt werden, die nicht Teil dieser Arbeit sind.

OpenRAVE hat sich als Planungsumgebung gut bewährt. Das übersichtliche Framework und die leicht zugängliche Pluginstruktur zeigen viel Potenzial für Verbesserungen und eigene Ideen. Wie schon angedeutet ist zum Beispiel der Glättungsschritt nicht optimal implementiert. Es werden zwei zufällige Punkte des RRT-Baumes ausgewählt und versucht mit einer Gerade im Konfigurationsraum zu verbinden. Dieser Vorgang benötigt 75% der gesamten Rechenzeit. Ein systematischer Ansatz wäre vermutlich eine bessere Lösung, die schnellere und bessere Ergebnisse erzielen würde. Bei einem einheitlichen Einsatz im Robotik-Institut könnten Standardbibliotheken leicht aufgebaut und ausgetauscht werden.

5 Parallelisierung von zweiarmigen Manipulationsaufgaben

Ein zweiarmiges System, wie es Justin darstellt, hat enormes Potential eines Tages als täglicher Begleiter des Menschen Fuß zu fassen. Durch seine beiden Arme ist es ihm möglich, ähnliche Aufgaben wie ein Mensch selber zu bewältigen. Kooperatives Verhalten seiner beiden Arme kann ihm helfen, komplexe Aufgaben zu lösen, schwere Gegenstände zu tragen oder einfach nur zwei Dinge gleichzeitig zu erledigen. Doch dafür müssen einige Hürden überwunden werden. Versucht man über 14 Freiheitsgrade zweier Roboterarme parallel zu planen, behindern diese sich unter Umständen mehr, als das sie sich unterstützen. Der Suchraum wird so groß, dass die Planungszeit enorm ansteigt. Es müssen daher Algorithmen entworfen werden, um die Planung beider Arme zu parallelisieren. Dafür ist es nötig bewerten zu können, ob Aufgabenteile abhängig oder unabhängig sind. Nur wenn beide Arme völlig unabhängig voneinander agieren können, ist die Parallelisierung möglich. Besonders wenn die Planung auf mehrere Prozesse aufgeteilt werden kann, verspricht dies eine enorme Zeitersparnis.

Abbildung 5.1: Justin bei einer kooperativen Manipulation beider Arme

5.1 Schwierigkeiten bei der zweiarmigen Pfadplanung

Justin wurde speziell für die Forschung im Bereich zweiarmiger Manipulationsaufgaben gebaut. Als antropomorpher Roboter verfügt er so über einen konkurrierenden Arbeitsraum seiner beiden Arme. Die Schnittmenge, in der sich beide Manipulatoren den Arbeitsraum teilen, zeichnet sich durch eine erhöhte Kollisionsgefahr aus. Wenn sich beide Arme dort aufhalten, sind Zusammenstöße nicht auszuschließen. Dennoch muss versucht werden auch diesen Bereich für beide Arme gleichzeitig nutzbar zu machen. Der Raum kann nicht einfach als in der Informatik üblicher kritischer Abschnitt betrachtet werden, bei dem immer nur ein Prozess gleichzeitig auf eine bestimmte Ressource zugreifen darf. Die Serialisierung der Bewegungsabläufe würde zu viel Zeit beanspruchen. Für Manipulationsaufgaben ist es durchaus gewünscht, dass sich beide Arme kooperativ in einem Bereich aufhalten (siehe Abbildung 5.1). Für diesen Fall müssen sich beide Arme synchronisiert bewegen. Die einfachste Lösung dafür ist die Planung über 14 Freiheitsgrade. Dabei wächst die Dimensionalität des Suchraums auf das Doppelte. Die Größe steigt dabei exponentiell. So werden unter Umständen viele Subräume des Konfigurationsraumes abgesucht, die für die Lösung der Aufgabe nicht relevant sind. Dabei werden jedesmal Kollision mit möglichen Hindernissen überprüft. Dies ist sehr rechenintensiv. Die dabei benötigte Zeit beeinträchtigt möglicherweise die komplette Planung. Ist es jedoch möglich, die Arme unabhängig zu bewegen, können diese Kosten auf mehrere Prozesse und damit auf mehrere Prozessoren verteilt werden. Die Schwierigkeit liegt darin sicherzustellen, dass weder die Endeffektoren noch die übrigen Elemente der Arme sich zu nahe kommen. Denn obwohl es eventuell nicht nötig ist, könnte ein RRT-Bahnplaner den Pfad eines Manipulators durch den Arbeitsbereich des zweiten Arms führen. Um einem möglichen Zeitverlust durch Abhängigkeiten der beiden Arme vorzubeugen, ohne das dabei Kollisionen entstehen, stellen sich nun also folgende Fragen:

- Wie kann der Arbeitsraum möglichst effizient ausgenutzt werden?
- Welche Eigenschaften müssen Unteraufgaben haben, damit sie parallel und unabhängig geplant und ausgeführt werden können?
- Durch welche Veränderungen in der Planungsumgebung kann man Parallelisierbarkeit erreichen?

Diesen Fragen wird in den Kapiteln 5.1.1 bis 5.1.2 nachgegangen. Dabei werden exisitierende Bahnplaner verwendet und parametrisieret. Es müssen keine Speziallösungen entwickelt werden. Allerdings entstehen durch derartige Parametrisierungen des Planers einige neue Probleme. Die Pfadplanung muss teilweise auf zwei Planungsinstanzen aufgeteilt werden, wobei zwei virtuelle Umgebungsrepräsentationen entstehen. Dabei ist es den Instanzen nicht nur unbekannt, wo sich der andere Manipulator aufhält, sondern auch welche Gegenstände dieser eventuell versetzt hat. Um diese Probleme zu beseitigen, müssen die Umgebungen synchronisiert werden. Wie in OpenRAVE zwei Umgebungen verwaltet werden können, wird in Kapitel 5.1.3 gezeigt.

5.1.1 Arbeitsraumanalyse und -aufteilung

Dieser Abschnitt befasst sich mit dem Arbeitsraum von Justin und dessen Aufteilung in verschiedene Bereiche. Dazu müssen einige Einschränkungen gemacht werden. Für die zweiarmigen Szenarien, die im Fokus dieser Arbeit stehen, wird weder der Oberkörper noch die mobile Plattform während der Planungsphase mit einbezogen. Lediglich die beiden Arme mit jeweils sieben Freiheitsgraden werden zur Manipulation verwendet. In diesem speziellen Fall besteht der Gesamtarbeitsraum annähernd aus zwei Sphären. Legt man diese beiden Arbeitsbereiche übereinander verdeutlicht sich der Eindruck des Schnittbereichs aus Abschnitt 5.1. Abbildung 5.2 skizziert den kombinierten Arbeitsraum des linken und des rechten Arms. Im mittleren Bereich ist zu sehen, wie breit der Überlappungbereich tatsächlich ist. Die Darstellung deutet die Aufteilung des Arbeitsraumes in drei Teile an. Diese Teile besitzen verschiedene Eigenschaften. Die äußeren Teile sind nur mit einem Arm erreichbar. Hier können sich die Arme nicht behindern. Der mittlere Bereich kann von beiden Armen betreten werden. Kollisionen sind daher nicht ausgeschlossen. Hier kann nach zwei Möglichkeiten verfahren werden.

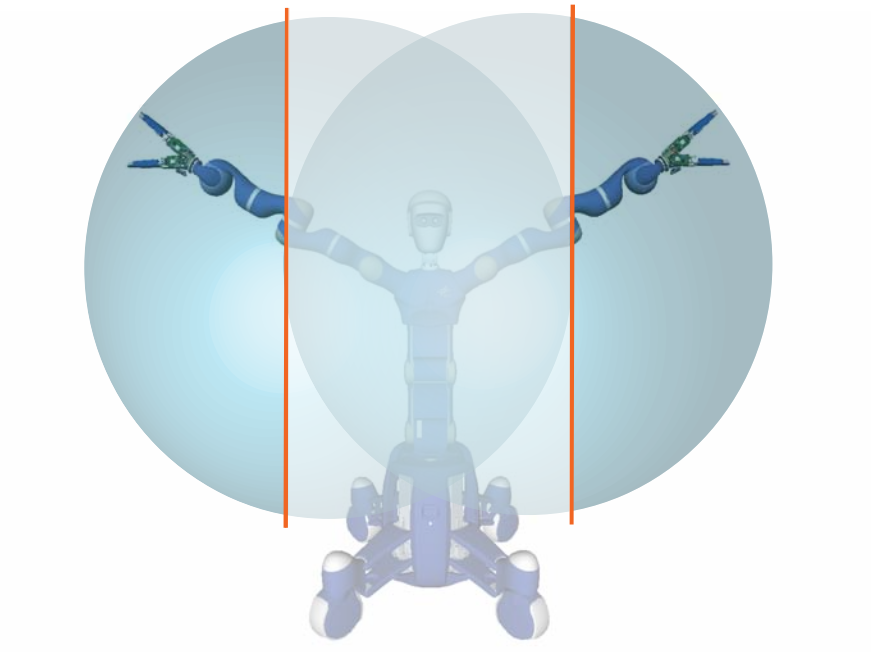


Abbildung 5.2: Aufteilung des Arbeitsbereiches in drei Teile. Die Schnittmenge bildet einen kritischen Abschnitt.

Die erste Möglichkeit besteht darin, den Abschnitt nur exklusiv zu betreten. Arbeiten in diesem Bereich können so nur sequentiell erledigt werden. Kollisionen sind so zwar ausgeschlossen, allerdings steigt dadurch die Ausführungszeit auf das doppelte an. Selbst wenn einer der Manipulatoren nur einen Bruchteil des Raumes für das Lösen seiner Aufgabe tatsächlich benötigt, wird der andere Manipulator ausgeschlossen und vom Arbeiten abgehalten.

Möglichkeit zwei beinhaltet die Strategie, im mittleren Arbeitsbereich immer über 14 Freiheitsgrade zu planen. Auch wenn man sich dafür entscheidet, steigt dabei die Gesamtzeit für Planung und Ausführung beider Arme. Die Suche über 14 DOF in diesem großen Subraum reduziert die Effektivität der Samplingstrategie ungemein, da sich die Arme immerzu gegenseitig behindern und viele Bereiche abgesucht werden, die für das Lösen der Aufgabe irrelevant sind. Um eine effektivere Ausnutzung des Arbeitsraumes zu erhalten, muss eine bessere Aufteilungsstrategie gewählt werden.

In einem natürlichen Umfeld befinden sich Werkzeuge, Gegenstände oder Hindernisse fast immer direkt vor dem Menschen oder dem Roboter. Auch wenn in diesem Bereich Aufgaben zeitgleich zu erledigen sind, müssen die Arme nicht unbedingt ständig koordiniert werden. Erst wenn sich die Arme kreuzen oder kooperativ arbeiten sollen, ist eine Synchronisation nötig. Ist dies nicht der Fall, kann folgende Strategie verfolgt werden: Der Arbeitsraum wird in zwei statt drei Bereiche geteilt. Der mittlere Bereich aus Abbildung 5.2 ist so nicht mehr nur exklusiv nutzbar. Jeder Manipulator kann ungestört bis zur Mitte des Arbeitsraumrs arbeiten. Darüber hinaus muss der Arbeitsraum tatsächlich geteilt werden. Um sicherstellen zu können, dass auch garantiert keine Kollisionen in dem eigentlich überlappenden Arbeitsbereich auftreten können, wird eine virtuelle Wand in der Mitte erzeugt (siehe Abbildung 5.3).

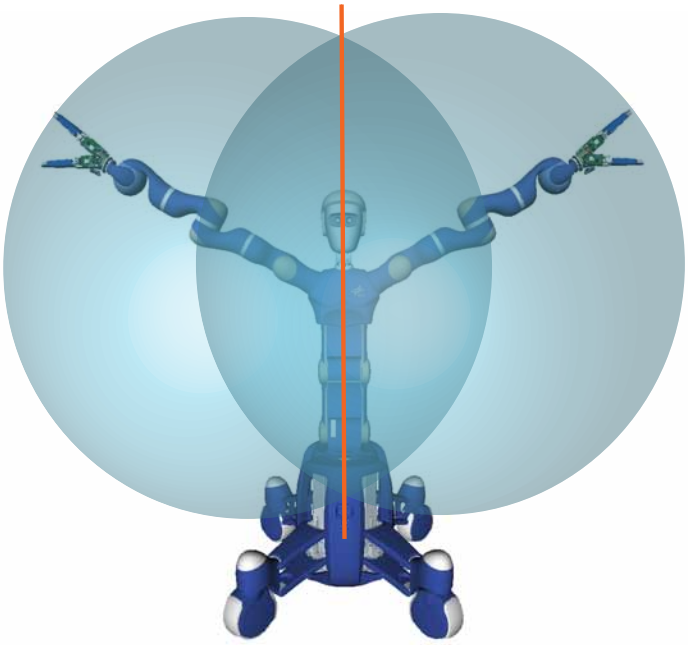


Abbildung 5.3: Aufteilung des Arbeitsbereiches in zwei Teile. Die virtuelle Wand in der Simulation verhindert ungewollte Berührungen der Arme.

Die Wand wird als dünner Quader in der Simulationsumgebung modelliert. Kollisionen mit der Wand während der Planung hindern die Manipulatoren daran, die andere Seite zu betreten. So kann es ermöglicht werden, dass beide Arme parallel in der Mitte des Tisches arbeiten ohne zu kollidieren. Erst wenn ein Manipulator tatsächlich den ihm zugeteilten Bereich verlassen muss, wird diese Grenze aufgelöst und es muss wiederum über alle Gelenke der Arme geplant werden.

5.1.2 Eigenschaften von parallelisierbaren Unteraufgaben

Zum Entwickeln eines geeigneten Algorithmus für die parallele Pfadplanung zweier Manipulatoren ist es unbedingt nötig zu wissen, wann eine Unteraufgabe parallelisiert geplant werden kann und wann nicht. Die Parallelisierbarkeit bezieht sich hierbei immer auf die zeitgleiche Planung und Bewegung der beiden Arme, ohne Kenntnis über den Verbleib des jeweilig anderen Manipulators[4]. Also wann kann beispielsweise der linke Arm völlig gefahrlos ein Glas greifen, während der rechte Arm ein Hindernis verschiebt, ohne dass diese voneinander wissen müssen? Bedenkt man diese Aussage stellt sich also die Frage: *„Wo dürfen sich Manipulatoren zeitgleich aufhalten und wo nicht?“* Im Bezug auf die in Kapitel 5.1.1 betrachtete Arbeitsraumaufteilung in zwei Bereiche resultieren folgende Antworten. Die folgenden Bedingungen müssen erfüllt werden, damit Unteraufgaben parallelisierbar sind:

- Es befinden sich weder der Endeffektor noch etwaige Verbindungselemente des Manipulators in Start- und Zielkonfiguration in dem für diesen gesperrrten Bereich.
- Dies impliziert, dass sich auch kein Element des Armes während der Ausführung der Trajektorie im gesperrten Bereich bewegt oder diesen betritt.
- Ferner dürfen durch eine Manipulation auch keine transportierten Gegenstände in den gesperrten Bereich gelangen.

Es muss also bekannt sein, wo sich der Endeffektor in der Zielkonfiguration befindet. Dabei ist die Lage des TCP, also der Mittelpunkt des Endeffektors, nicht allein ausschlaggebend. Die Ausmaße der Hand und eventuell getragene Objekte müssen mit berücksichtigt werden. Schon bevor der TCP die mittlere Grenze übertritt, können Elemente die Integrität des kollisionsfreien Raumes beeinträchtigen. Aus diesem Grund müssen zwei zusätzliche Schwellwerte angelegt werden. Einer für jeden Arm. Sobald ein Arm den Schwellwert überschreitet, kann zumindest die Ausführung nicht mehr parallelisiert erfolgen ohne dabei über 14 Freiheitsgrade zu planen. Die beiden Schwellwerte und die daraus resultierenden gesperrten Bereiche sind in Abbildung 5.4 skizziert. Erst wenn der linke Manipulator den Schwellwert Th_l überschreitet, oder der rechte Arm den Schwellwert Th_r, sind Kollisionen

[4] Bei dieser Art der parallelen Planung wird je eine Trajektorie des Manipulators zu je sieben DOF in einem Prozess geplant.

möglich. Die Wand, die als orangene Linie dargestellt ist, muss in diesem Fall aufgelöst werden. Die kritischen Bereiche für die beiden Arme sind letztendlich die jeweiligen schwarz umrandeten Kreissegmente.

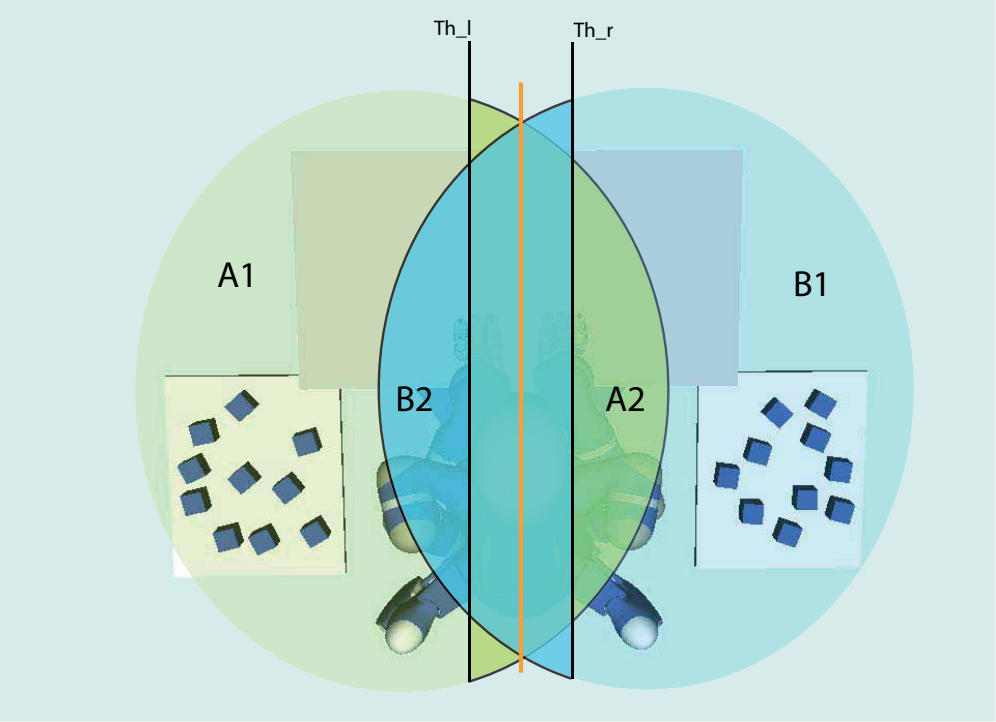


Abbildung 5.4: Grenzwerte und gesperrte Bereiche für parallelisierbare Manipulationsaufgaben.

Zusammenfassend stellen sich also die folgenden Verhalten ein: Wenn sich die Aufgabenziele des linken Arms in $A1$ und die Aufgabenziele des rechten Arms in $B1$ befinden kann die Planung und Ausführung unabhängig auf zwei Prozesse aufgeteilt werden, ohne das Kollisionen befürchtet werden müssen. Sobald der linke Arm aber $A2$ betritt, oder der rechte Arm $B2$, besteht Kollisionsgefahr. In diesem Fall muss die virtuelle Wand aufgelöst werden. Tritt dieser Fall ein können die zwei möglichen Strategien aus Abschnitt 5.1.1 verfolgt werden. Entweder kann die Planung über 14 Gelenkwinkel erfolgen, wodurch auch die parallele Ausführung gewährleistet ist, oder die Planung erfolgt unabhängig und parallel (2 x 7 DOF), wodurch die Ausführung sequenziell zu erfolgen hat. Zurzeit sind die Grenzen noch für jedes Szenario individuell anzupassen. Vorallem wenn zu erwarten ist das die Manipulatoren große Gegenstände mit sich führen, müssen die Schwellwerte erweitert werden, um Kollisonen zu vermeiden. Zum Garantieren einer allgemein gültigen Planungsstrategie, müssen diese für zukünftige Aufgaben adaptiv angepasst werden.

5.1.3 Synchronisation zweier OpenRAVE Umgebungen

Damit die Pfadplanung der beiden Arme tatsächlich unabhängig ausgeführt werden kann, müssen die Planungsräume voneinander entkoppelt werden. Anstatt also über 14 Freiheitsgrade zu planen, muss die Planung zwei mal über sieben DOF stattfinden. Um dies in OpenRAVE zu erreichen, muss das Planungsmodul zweimal aufgerufen werden und damit auch zwei OpenRAVE Environments geladen werden. Daraus resultieren zwei unabhängige Umgebungsrepräsentationen oder Weltmodelle. Die simultane Planung in zwei verschiedenen Repräsentationen birgt einige Risiken. Um Kollisionen tatsächlich vermeiden zu können, ist es unbedingt nötig, die Weltmodelle nach jedem Planungsdurchlauf zu synchronisieren. Während der Ausführung sind die Bewegungen trotz unabhängiger Planung von einander abhängig. Beispielsweise könnte in der Repräsentation für den linken Arm der Bewgungsablauf für das Greifen einer Flasche geplant werden. In der rechten Umgebung wird zeitgleich der Ablauf für das Greifen eines Glases geplant, welches der Flasche sehr nahe steht. Bei der simulatanen Ausführung ist es durchaus möglich, dass sich die Pfade kreuzen. Die Kollision kann nicht vorhergesagt werden. Da in OpenRAVE keine Mechanismen zu synchronisation derartiger Planungsstrategien implementiert sind, gilt es eine geeignete Architektur zu entwerfen. Bevor damit begonnen werden kann, muss festgelegt werden, was eigentlich alles synchronisiert werden muss. Um Bewegungen jederzeit sicher ausführen zu können, brauchen beide Umgebungs-Repräsentationen immer die aktuellen Informationen über

- Gelenkwinkelstellungen aller Teilroboter,
- Position und Orientierung aller Objekte im Raum,
- und die gegriffenen Gegenstände, die an den Manipulatoren angefügt sind.

Während der Planung können die Umgebungen nicht synchronisiert bleiben. In der Planungsphase werden zu viele Konfigurationen ausprobiert, die in der endgültigen Lösung nicht mehr enthalten sind. Vor der Planungsphase ist die Synchronisation allerdings unabdingbar. Aus diesem Grund besteht das Programm aus drei Prozessen: Einem Master, einem Slave und dem Manager, der die Kommunikation zwischen den beiden Weltmodellen ermöglicht. Die Master-Umgebung visualisiert zusätzlich die Bewegungen, wohingegen die Slave-Umgebung nur die jeweilige Trajektorie plant. Der sichtbare Pfad wird nur auf dem Master ausgegeben, da es zu rechenaufwendig ist beide Weltmodelle jederzeit zu Visualisieren. Dies reicht völlig aus, da die Änderungen des Masters und des Slaves im Masterprozess visualisiert werden. Außerdem entstünde dadurch nur eine redundante Betrachtung der Simulation ohne zusätzlichen Nutzen. Der Slave wird nach jeder Bewegung wieder mit dem tatsächlichen Stand abgeglichen. Beide Repräsentationen werden vom Manager erzeugt, der für die eigentliche Synchronisation zuständig ist. Wenn eine Planung in beiden Weltmodellen abgeschlossen ist, wird der Manager informiert. Dieser wartet so lange, bis beide Instanzen fertig sind und fügt anschließend die Trajektorien zusammen. Kann eine Planung über 2 x 7 DOF nicht erfolgen, wird die 14 DOF Bewegung

ausschließlich auf der Masterumgebung geplant und ausgeführt. Auch dann muss die Slaveumgebung wieder auf den neusten Stand gebracht werden. Der Ablaufplan in Abbildung 5.5 erläutert das Vorgehen anhand eines Beispiels.

Unter der Annahme das der Roboter schon in einer für einen Griff realisierbaren Roboter-Gesamtkonfiguration positioniert ist, müssen die Arme nicht zu den Objekten geführt werden. Nur der Masterprozess erhält eine Visualisierung. Ausgehend von dieser Position können bereits beide Objekte gegriffen werden. Dies geschieht simultan auf beiden Repräsentationen. Der Roboter hat also in beiden Simulationen nun zwei Gegenstände an seine Manipulatoren angefügt. Bei der Pfadplanung wird nun darauf geachtet, dass weder der Manipulator noch das angefügte Objekt mit der Umwelt kollidiert. Während der Planung werden die bereits erläuterten Maßnahmen für parallelisierbare Unteraufgaben angewandt. Die Planung kann also völlig unabhängig und zeitgleich mit beiden Armen stattfinden, solange die Unteraufgaben die herausgearbeiteten Kriterien erfüllen. Sobald der Manager die Rückmeldung beider Planungsinstanzen erhalten hat, wird die Trajektorie auf dem Master abgefahren. Der nun erreichte rote Bereich verletzt die Integrität der Synchronisation. Dadurch das die Trajektorie nur auf der Master-Umgebung ausgeführt wird, befindet sich das Weltmodell des Slaves noch auf dem Stand vor der Planung. Der Synchronisationsschritt muss daher direkt im Anschluss erfolgen. Dabei werden die Gelenkwinkelwerte und die Objekttransformationen des Masters auf die des Slaves übertragen.

Ebenso wie *grab_left(left_obj)* oder *grab_right(right_obj)* könnte natürlich auch der *play_trajectory(merged)* Befehl auf beiden Umgebungsrepräsentationen ausgeführt werden. Dadurch wären beide Weltmodelle jederzeit synchron. Die Trajektorie auszuführen ist jedoch um einiges rechenintensiver, als einfach nur den Status der Masterumgebung zu kopieren.

Nichtsdestotrotz ist dieser Vorgang immer noch für Kollisionen anfällig, wenn man davon ausgeht, dass die Planung nicht parallelisiert werden kann, da beide Manipulatoren über Kreuz arbeiten sollen. Unter der Annahme, dass es für die gewünschten Zielkonfigurationen beider Arme keine kollisionsfreien Inverskinematik-Lösungen gibt, kann die Planung dann zwar trotzdem unabhängig geschehen, die Ausführung muss jedoch sequentiell erfolgen. Die Endbedingung beider Planungsphasen muss jedoch im vorraus bekannt sein. Grund dafür ist, dass ein abgesetzter Gegenstand des zuerst ausführenden Manipulators die Bahn des folgenden Armes behindern könnte. Da dies zu Beginn der Pfadplanung noch nicht der Fall war, würde es nicht bemerkt werden und eine Kollision wäre die Folge. In diesem speziellen Fall ist es daher zusätzlich erforderlich, dass die Objekte des konkurrierenden Manipulators bereits im Voraus an ihre Zielkonfiguration übermittelt werden. In den Simulationen herrscht also für eine gewisse Zeit der für eine kollisionsfreie Bahn erforderliche Zustand des nächsten Schritts. Im späteren Verlauf der Arbeit wird dieser Mechanismus benutzt. Eine Illustration zur besseren Verdeutlichung findet sich in Abbildung 5.8 auf Seite 59.

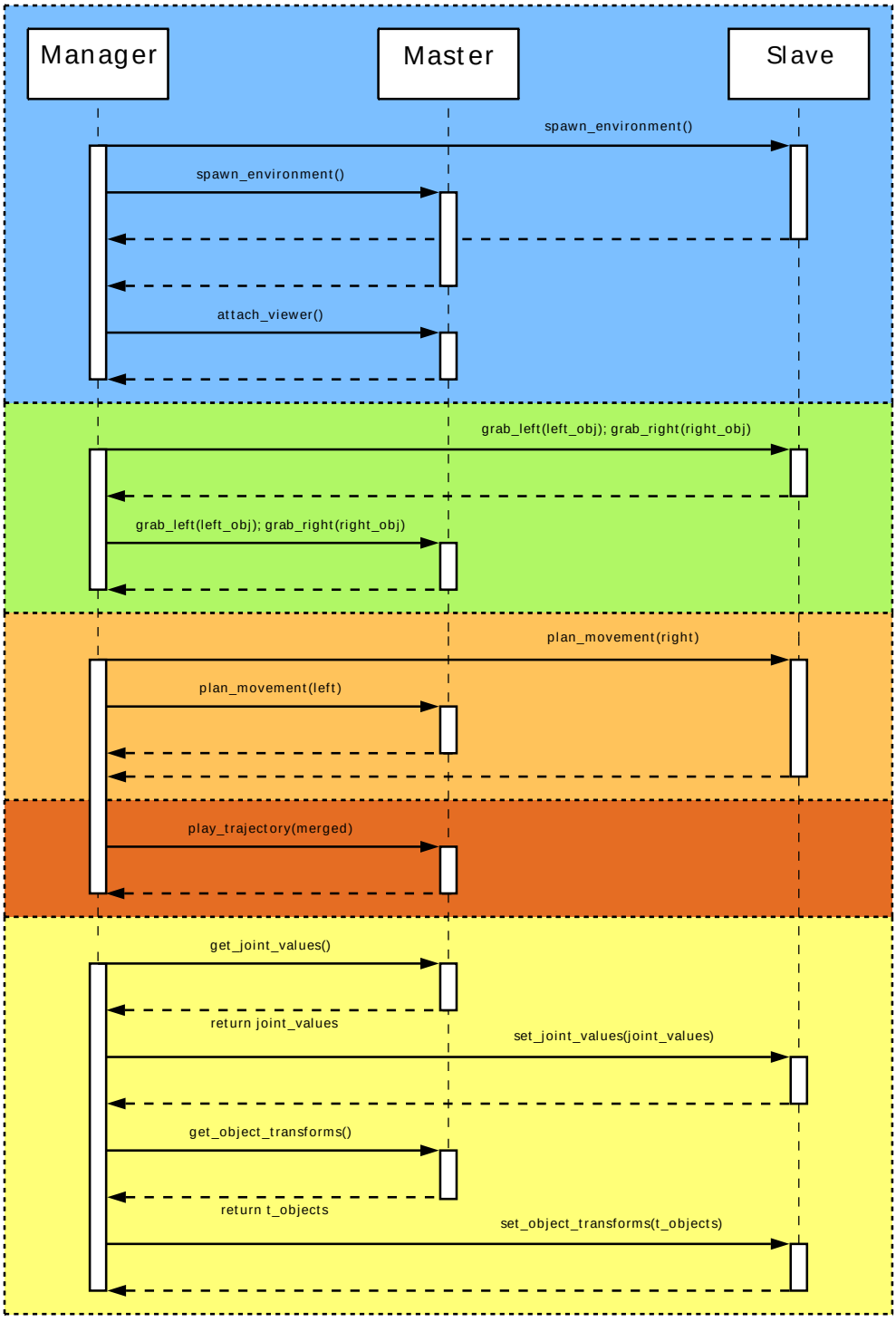


Abbildung 5.5: Synchronisation zweier OpenRAVE Environments. Blaue Abläufe sind Initialisierungen. Grüne Vorgänge spielen sich synchronisiert auf beiden Umgebungen ab. Orangene Bereiche sind Planungsphasen. Rote Bereiche verletzen die Synchronität. Der gelbe Block dient hier als Synchronisationsblock.

5.2 Szenarien für zweiarmige Pfadplanung

Für die Validierung der aufgestellten Thesen zur Parallelisierung von Unteraufgaben im aufgeteilten Arbeitsraum werden verschiedene Testszenarien herangezogen. Zunächst stellt sich die Frage, ob die unabhängige Planung in den beiden verschiedenen Räumen tatsächlich einen Zeitgewinn erzielt. Dazu werden die gleichen Bewegungen einmal über 14 Freiheitsgrade und anschließend 2 x 7 Freiheitsgrade geplant. Im ersten Szenario ist das Design bewusst so gewählt, dass nur unabhängige Aufgaben vorkommen, deren Planung parallelisiert werden kann. Das zweite Szenario beschäftigt sich mit dem kritischen Bereich. Wiederum wird ein kombinierter Pfad aus linker und rechter Bewegung geplant. Einmal über 14 Freiheitsgrade und einmal über 2 x 7 Freiheitsgrade auf zwei Prozesse verteilt. Die Ausführung muss im zweiten Fall aufgrund der unbekannten Bewegungen des konkurrierenden Armes sequentiell erfolgen.

5.2.1 Szenario I - Parallelisierte Pfadplanung im disjunkten Bereich des Arbeitsraumes

Der erste Testfall soll beweisen, dass die Aufteilung der Planung in zwei unabhängige Prozesse tatsächlich Vorteile bringt. Da die Bereiche durch die in Kapitel 5.1.1 eingeführte virtuelle Wand voneinander getrennt sind, können Planung sowie die Bewegung beider Arme jederzeit simultan ausgeführt werden. Dabei teilt sich die Planung auf zwei Prozesse und damit auch auf zwei physikalische Prozessoren auf.

5.2.1.1 Aufbau

Für die zweiarmigen Testszenarien wurde die Umgebung aus den einarmigen Tests verwendet und auf die Bedürfnisse des erweiterten Arbeitsraumes angepasst. Der bereits verwendete Tisch mit den Ausmaßen aus Kapitel 4.2.1 wurde um zwei kleinere Elemente erweitert. Zwischen den kartesischen Punkten $p1_l = (0.40, 0.00, 0.766)$, $p2_l = (0.95, 0.60, 0.766)$ und $p1_r = (0.40, 0.00, 0.766)$, $p2_r$ = (-0.95, -0.60, 0.766) entstehen dadurch zwei weitere Ablageflächen. Innerhalb dieser Bereiche werden je elf Würfel mit einer Kantenlänge von 0.06 m platziert. Zwischen den Würfeln ist genügend Platz für eine kollisionsfreie Griffstellung. Die Posen der Würfel wurden für diesen Versuch einmal zufällig ausgewählt und anschließend abgespeichert. Durch die zufallsbasierte Planung müssen mehrere Durchläufe abgearbeitet werden, um im Anschluss die Zeiten mitteln zu können. Bei jedem Durchlauf befinden sich die Würfel am gleichen Ort. Die Posen der Würfel sind in Tabelle 5.1 festgehalten.

Würfel auf dem linken Tisch				Würfel auf dem rechten Tisch		
ID	(X, Y) Position in Metern	Rotation in Grad		ID	(X, Y) Position in Metern	Rotation in Grad
1	(0.156, 0.675)	37.09		12	(0.249, -0.696)	-15.73
2	(0.340, 0.797)	41.55		13	(0.377, -0.613)	-9.23
3	(0.292, 0.617)	16.44		14	(0.289, -0.866)	-22.74
4	(0.324, 0.908)	11.43		15	(0.068, -0.682)	-2.83
5	(0.010, 0.854)	13.89		16	(0.108, -0.840)	-0.38
6	(0.787, 0.800)	38.81		17	(0.365, -0.750)	-41.69
7	(0.003, 0.751)	24.89		18	(0.069, -0.946)	-14.28
8	(0.119, 0.942)	13.03		19	(0.192, -0.938)	-8.35
9	(0.394, 0.614)	36.98		20	(0.002, -0.790)	-18.98
10	(0.023, 0.602)	30.54		21	(0.390, -0.878)	-30.01
11	(0.217, 0.942)	25.51		22	(0.178, -0.606)	-8.08

Tabelle 5.1: Ausgangsposition der Würfel im ersten zweiarmigen Szenario. Die Translation in Z-Richtung beträgt jeweils 0.77 m

Justins Ausgangsposition ist diesmal exakt an das Szenario angepasst. Der Oberkörper ist so ausgerichtet, dass die Manipulatoren die Tischflächen maximal abdecken. Mittels einer Erreichbarkeitskarte nach [36], kann genau festgestellt werden, welche Bereiche des Raumes mit der aktuellen Torsoposition abgedeckt werden und welche nicht. Auch diese Darstellung zeigt den breiten Überlappungsbereich der beiden Arme. In Abbildung 5.6 wird außerdem gezeigt, dass es nicht nötig ist, den Torso zu bewegen, um die Würfel auf die Hauptfläche des Tisches zu transportieren. Die rötlichen Kugeln zeigen die Grenzen der Arbeitsräume. Die Handkonfiguration ist ebenfalls auf das Greifen der Würfel angepasst. Nur drei Finger werden verwendet um die Würfel zu greifen. Dadurch können kleinere Engstellen durchfahren werden, ohne dass dabei Kollisionen entstehen. Die Gelenkwinkelstellungen sind in Tabelle 5.2 eingetragen. Mittels eines *Greifframes* wird bestimmt, in welcher Pose sich der Endeffektor relativ zum Ziel befinden muss, um es zu greifen. Es beschreibt die Transformation vom Ziel zum TCP des Manipulators. Das genaue Frame findet sich im Anhang. Für die Greifframes der Hände gilt allgemein:

$$^{Welt}T_{TCP} = ^{Welt}T_{Base} . ^{Base}T_{Dock} . ^{Dock}T_1 . ^{1}T_2 . ^{2}T_3 . ^{3}T_4 . ^{4}T_5 . ^{5}T_6 . ^{6}T_7 . ^{7}T_{TCP} \quad (5.1)$$

$$^{Wuerfel}T_{TCP} = ^{Welt}T^{-1}_{Wuerfel} . ^{Welt}T_{TCP} \quad (5.2)$$

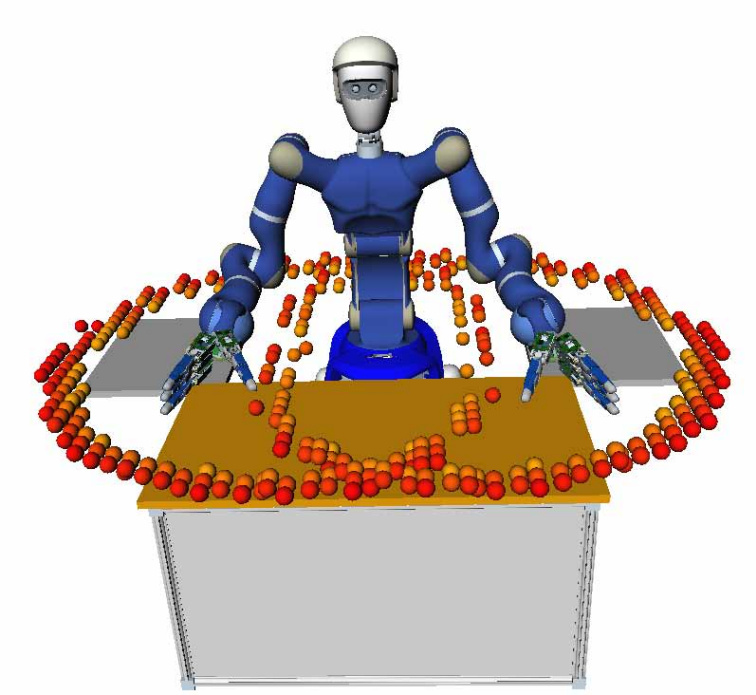


Abbildung 5.6: Flächenschnitt der Erreichbarkeitskarte

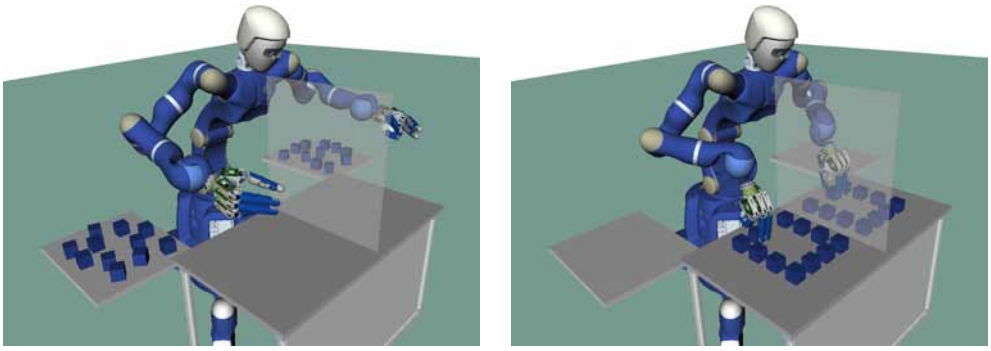
Körperteil	Gelenkwinkelparameter in Grad
Torso	0.00 1.19 44.70 -45.89
Rechter Arm	-12.42 -62.37 -3.07 93.28 56.90 -3.89 12.62
Rechte Hand	-12.03 10.37 0.00 0.00 30.86 0.00 0.00 34.68 -0.00 0.00 71.96 81.02
Linker Arm	-12.42 -62.37 -3.07 93.28 56.90 -3.89 12.62
Linke Hand	12.03 12.43 0.00 0.00 36.15 0.00 0.00 36.15 0.00 0.00 71.96 81.02
Kopf	0.00 33.97

Tabelle 5.2: Startkonfiguration für die zweiarmige Manipulation

In der Mitte des Tisches ist die virtuelle Wand angebracht. Sie soll eine Kollision der beiden Arme verhindern, auch wenn diese vollig unabhängig agieren. Die Wand ist als dünner Quader mit den Ausmaßen (0.65, 0.004, 0.7 m) (H x B x T) dargestellt. Die Größe reicht aus, um kollisionsfreie Pfade im vorderen Bereich des Roboters zu generieren. Auch wenn ein RRT-Planer prinziepell den gesammten C_{space} exploriert, entsteht nach der Optimierung ein unkritischer Pfad.

5.2.1.2 Ablauf

Bei den zweiarmigen Szenarien liegt der Fokus auf der parallelen Planung und wenn möglich auch auf der parallelen Ausführung. In diesem Fall sollen alle Manipulationen parallel ausgeführt werden. Das Design des Szenarios lässt dies jederzeit zu. Der Versuch ist in zwei Teile aufgeteilt: Einen Teil, bei dem mit 14 Freiheitsgraden geplant wird und ein Teil, bei dem 2 x 7 Freiheitsgrade getrennt zum Einsatz kommen. In beiden Teilen verhindert die virtuelle Wand in der Mitte die Kollision der beiden Arme. Auch wenn dies im ersten Durchlauf nicht nötig wäre. Zu Beginn werden die Würfel wie in Tabelle 5.1 platziert und Justin wird in die Ausgangsposition aus Tabelle 5.2 bewegt (siehe Abbildung 5.7a). Eine Heuristik berechnet, welche Würfel als nächstes zu Greifen sind[5]. Ausgehend von dieser Information wird für beide Arme eine Zielkonfiguration mittels der Inverskinematik berechnet.



(a) Initalkonfiguration (b) Zielkonfiguration

Abbildung 5.7: Ablauf der unabhängigen Manipulationsplanung

Im ersten Teil werden alle Trajektorien direkt für 14 DOF geplant. Es ist keine Synchronisation nötig. Die Trajektorien können direkt ausgeführt werden. Sobald also beide Zielkonfigurationen zum Erreichen der Zielobjekte feststehen, beginnt die Planungsphase. Findet sich ein kollisionsfreier Pfad, wird dieser ausgeführt. Beide Arme bewegen sich dann parallel und aufeinander abgestimmt. Am Bestimmungsort angelangt können beide Hände jetzt je einen Würfel greifen. Anhand eines Lageplans aus einer externen Datei erhält der Roboter die neue Zielpositionen für die Würfel. Dieser Punkt wird als Zieltransformation für die nächste Bahn gesetzt. Sind die Würfel nach einer erneuten Planungsphase am Ziel angelangt können diese abgelegt werden. Anhand der zuvor beschriebenen Heuristik erhalten die Manipulatoren ihre neuen Zielobjekte und der Vorgang wiederholt sich so lange, bis alle Objekte ihre Endposition erreicht haben (siehe Abbildung 5.7b). Anschließend wird die Szenerie wieder in den Ursprungszustand versetzt und das komplette Szenario startet erneut. Insgesamt wird das Szenario 100 Mal durchlaufen.

[5]Die Würfel werden nach dem relativen Abstand zur Tischmitte sortiert. Zur Berechnung der Distanz wird der Satz des Pythagoras verwendet.

Der zweite Teil findet aufgeteilt in zwei Prozesse statt. Dazu werden, wie in Kapitel 5.1.3 beschrieben, zwei Umgebungsrepräsentationen benötigt. Die Koordination übernimmt ein Managerprozess. Für jede Repräsentation wird ein Zielframe angegeben. Die erste Repräsentation R_l führt die Planung des linken Armes durch und die zweite Repräsentation R_r kümmert sich um den rechten Arm. Beide Repräsentationen sind völlig unabhängig voneinander. Sobald beide Prozesse die Planung erfolgreich abgeschlossen haben, speichern diese die erhaltene Trajektorie ab. Der Managerprozess führt beide Abläufe in einer Trajektorie zusammen. Die zusammengeführte Bewegung wird nur auf dem Masterprozess, der Repräsentation R_l, ausgeführt. Bevor die Würfel gegriffen werden können, muss also noch einmal synchronisiert werden. R_r übernimmt alle Werte von R_l. Mit beiden Umgebungen auf dem gleichen Stand sind die Würfel griffbereit. Für die Umgebungsrepräsentationen wird danach wieder je eine Zielkonfiguration für den Bestimmungsort der Würfel berechnet. Erneut planen beide Umgebungen die Bewegung unabhängig voneinander bevor der Managerprozess diese zusammenführt. Auch im zweiten Teil müssen alle Würfel an ihre Zielposition versetzt werden. Das Ganze wird 100 Mal wiederholt.

5.2.1.3 Ergebnis

Bei der Auswertung des Szenarios stehen sich die Ergebnisse der 14 DOF Planung und der 2 x 7 DOF Planung direkt gegenüber. Die gemittelten Werte der Planungszeiten und der Ausführungszeiten sind jeweils in Tabelle 5.3 und 5.4 aufgelistet. Es wird zusätzlich noch einmal unterschieden zwischen Transitpfaden, welche zu einem Objekt hinführen und Transferpfaden, die ein Objekt bewegen.

	Deadline	Transitpfade		Transferpfade	
		Planung	Bewegung	Planung	Bewegung
14 DOF ohne Wand	120	27.54	1.62	25.39	1.93
14 DOF ohne Wand	60	23.57	1.16	26.21	1.91
14 DOF mit Wand	120	27.21	1.86	24.48	1.88
14 DOF mit Wand	60	24.08	1.26	25.36	1.74
2x7 DOF mit Wand	60	17.35	0.75	16.73	0.74

Tabelle 5.3: Numerische Ergebnisse der parallelisierten Pfadplanung im geschützten Bereich – Planungszeiten und Ausführungszeiten gemittelt über 100 Durchläufe.

	Deadline	Gesamt-dauer	Erfolgsrate (Transit, Transfer)	Erfolgsrate pro Durchlauf
14 DOF ohne Wand	120	627.47	(91.2%, 100.0%)	52.00%
14 DOF ohne Wand	60	571.64	(79.0%, 97.0%)	13.00%
14 DOF mit Wand	120	615.46	(93.6%, 100.0%)	63.00%
14 DOF mit Wand	60	595.64	(82.1%, 97.5%)	19.00%
2x7 DOF mit Wand	60	423.40	(100.0%, 100,0%)	100.00%

Tabelle 5.4: Numerische Ergebnisse der parallelisierten Pfadplanung im geschützten Bereich – Gesamtdauer und Erfolgsraten gemittelt über 100 Durchläufe.

Die Zeitunterschiede zwischen 2 x 7 DOF und 14 DOF sind nicht ganz so groß wie erwartet. Allerdings sind die Erfolgsraten deutlich besser bei der unabhängigen Planung. Ergebnisse, die außerhalb der *Deadline*[6] liegen, fließen nicht mehr in die Durchschnittszeit ein, sondern schmälern die Erfolgsrate. Dies ist auch der Grund warum die Ausführungszeit der Transferpfade im 14 DOF so niedrig erscheinen. Tatsächlich lagen diese zuweilen weit über 60 Sekunden und wurden deshalb ausgefiltert. Die Erfolgsrate der Transitpfade spiegelt diesen Effekt wieder. Die Planung über alle Gelenkwinkel der Arme erfordert es, zwei *„narrow passage“* Probleme gleichzeitig zu lösen. Die Würfel haben nämlich nur einen geringen Abstand von durchschnittlich etwa 5 cm zueinander. Der Manipulator muss zudem 3 cm in die Ebene der Würfel eingreifen um bei einem Würfel mit 6 cm Kantenlänge einen sicheren Griff zu gewährleisten. Ein Blick auf die Erfolgsrate verrät, dass von 100 Versuchen nur 13 ohne Abbruch eines Pfades durchgeführt werden konnten. Nimmt man die Wand dazu, erreicht die 14 DOF Planung ebenfalls nur 19% Erfolgsrate. Die Erfolgsrate pro Pfad errechnet sich folgendermaßen:

$$\sum_{i=1}^{n} Pfade = (1_{Transitpfad} + 1_{Transferpfad}) \cdot 22_{Wuerfel} \cdot n_{Durchlaufe}) \tag{5.3}$$

$$Erfolgsrate = \frac{\sum Pfade - Abbrueche}{\sum Pfade} \cdot 100 \tag{5.4}$$

Daraus folgt, dass bei $\sum Pfade = 4400$, 569 Pfade die Deadline der Planungszeit von 60 Sekunden überschritten haben. Umso niedriger die Deadline gesetzt wird, umso mehr relativiert sich das Ergebnis. Aber im direkten Vergleich und einer Deadline von 60 Sekunden schneidet die getrennte Planung deutlich besser ab. Mit

[6] Eine Deadline ist umgangssprachlich ein Termin der das Ende einer Frist darstellt. In der Informatik ist die Deadline ein Begriff, der als Qualitätsmerkmal von Prozessen herangezogen wird. Kann ein Prozess seine Aufgabe vor dem Ablauf der Deadline nicht erfüllen, so sind dessen Ergebnisse nicht relevant.

100% Erfolgsrate und 17.34 Sekunden Planungszeit kommen die Ergebnisse nah an die Zeit der einarmigen Szenarien heran. Für die Berechnung der durchschnittlichen Planungszeit im 2 x 7 DOF Fall wird immer das Maximum der jeweiligen Planungsdauer der beiden Arme verwendet. Betrachtet man die virtuelle Wand, liegt die Idee nahe, die nicht erwünschten Konfigurationen die über die Wand hinaus reichen schon während des Samplingschrittes auszufiltern. Versuche die diesen Ansatz verfolgt haben, haben gezeigt, dass dadurch zwar der gleiche Effekt erzielt werden kann, die Planungszeit jedoch nicht so stark reduziert wird wie mit der virtuellen Wand. Dies liegt daran, dass zu jeder Konfiguration die Vorwärtskinematik berechnet werden muss, um die Position des Endeffektors zu bestimmen. Die ständige Berechnung der Vorwärtskinematik ist jedoch aufwendig und verbraucht daher viel mehr Zeit als die Kollisionschecks mit der virtuellen Wand.

5.2.2 Szenario II - Parallelisierte Pfadplanung im überlappenden Bereich des Arbeitsraumes

In diesem Szenario soll herausgefunden werden, ob eine Manipulation innerhalb des kritischen Bereichs schneller über 14 Freiheitsgrade oder über 2 x 7 Freiheitsgrade erfolgt. Die Planung über 14 Freiheitsgrade kann nicht immer gewährleistet werden. Nur wenn für beide Arme eine Zielkonfiguration existiert bei der die Manipulatoren nicht kollidieren, ist die Planung realisierbar. Aber selbst wenn diese Gegebenheit vorhanden ist, kann es sein, dass es nicht möglich ist einen kollisionsfreien Pfad zu finden. Aus diesem Grund ist es nötig, dass beide Bedingungen eintreffen um vergleichbare Werte zu erzielen. Die Planung über 2 x 7 DOF findet wie gewohnt auf zwei Prozessen statt. Nur die Ausführung erfolgt sequentiell, deshalb entfällt auch das Problem der kollidierenden Zielkonfigurationen.

5.2.2.1 Aufbau

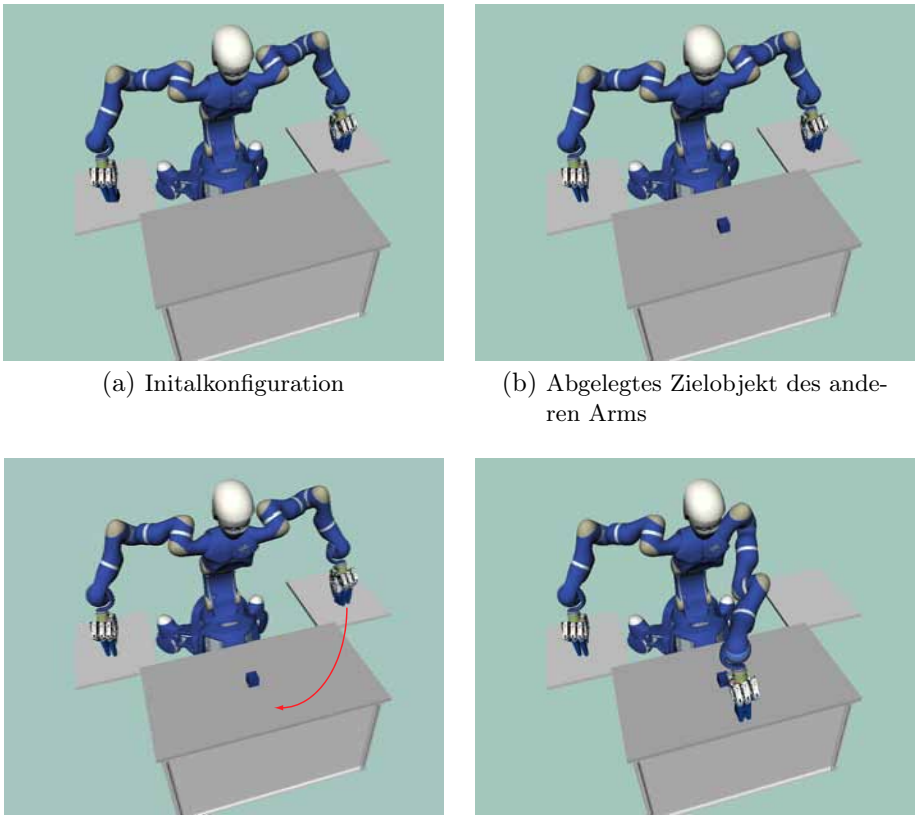
Der grundsätzliche Aufbau des zweiten Szenarios ist identisch zum Aufbau aus Szenario I. Der erweiterte Tisch und Justin stehen an der gleichen Stelle. Die virtuelle Wand wurde für dieses Szenario komplett entfernt. Sie ist für die parallele Planung im kritischen Bereich nicht von Bedeutung. Es befindet sich jetzt nur noch ein Würfel auf jeder Tischseite.

Körperteil	Gelenkwinkelparameter in Grad
Rechter Arm	-53.76 -54.77 -3.07 73.18 -35.04 -31.65 8.83
Linker Arm	-49.75 -47.15 -22.31 71.22 -29.72 -24.01 -7.80

Tabelle 5.5: Startkonfiguration der Arme für das zweite zweiarmige Szenario. Die restlichen Gelenkwinkel sind Tabelle 5.2 zu entnehmen.

Der Würfel auf der linken Seite liegt mit einer Rotation von 15.74° um dessen Z-Achse auf dem kartesischen Punkt p1 = [0.26 0.78 0.77]. Der rechte Würfel ist eine Spiegelung des linken Würfels und liegt damit bei p2 = [0.26 -0.78 0.77]. Die Rotation beträgt -15.74°. Justins Ausgangskonfiguration des Torsos ist identisch. Die Arme befinden sich bereits in einer kollisionsfreien Stellung zum Greifen der Würfel. Das passende Greifframe findet sich in Anhang A.2 Die Konfiguration der Arme findet sich in Tabelle 5.5.

5.2.2.2 Ablauf



(a) Initalkonfiguration

(b) Abgelegtes Zielobjekt des anderen Arms

(c) Gewählte Trajektorie

(d) Zielkonfiguration

Abbildung 5.8: Ablauf der abhängigen Manipulationsplanung. Ohne den Zwischenschritt in Abbildung 5.8b ist eine kollisionsfreie Trajektorie nicht sicher zu gewährleisten

Der Ablauf besteht erneut aus zwei Teilen. Ausgehend von der Startkonfiguration werden beide Würfel mit der entsprechenen Hand gegriffen. Anschließend werden zwei zufällige Zielpositionen für die Würfel gesucht. Die Positionen befinden sich innerhalb der beiden Schwellwerte aus Kapitel 5.1.3 und können auch über Kreuz liegen. Dabei müssen die Gelenkwinkelstellungen in den Zielkonfigurationen für bei-

de Arme kollisionsfrei sein. Ist dies der Fall, wird zunächst eine Trajektorie für beide Arme parallel mit dem 14 DOF Planer geplant. Hierbei haben die Manipulatoren Kenntniss voneinander, wodurch auch die Ausführung simultan erfolgen kann. Der zweite Teil besteht aus der unabhängigen Planung beider Arme. Die Planung kann parallel erfolgen, das heißt jeder Arm plant in seiner eigenen Umgebung für sich. Die Ausführung muss mangels fehlender Synchronisationsmöglichkeiten sequentiell durchgeführt werden. Um Kollisionen mit den vom anderen Arm bereits abgelegten Würfeln zu vermeiden, werden diese vor Planungsbeginn virtuell an die Zielposition transferiert. Der schematische Ablauf für den sequenziellen Vorgang wird zur besseren Verdeutlichung in Abbildung 5.8 dargestellt.

Um den Rückweg nicht planen zu müssen, wird einfach die berechnete Trajektorie noch einmal rückwärts durchlaufen. Der Planungsaufwand ist nämlich dann besonders groß wenn beide Arme bereits über Kreuz stehen und versuchen kollisionsfrei voneinander weg zu fahren. Auch im sequentiellen Durchgang werden die einzelnen Bahnen einfach zurück abgespielt. Der Rückweg muss daher nicht geplant werden.

5.2.2.3 Ergebnis

Die Ergebnisse dieses Szenarios sollen darüber Aufschluss geben, welche Planungsstrategie besser geeignet ist um in einem konkurrierenden Arbeitsbereich zu agieren. Es soll sich zeigen, ob es effektiver ist, über 14 Freiheitsgrade eine asymmetrisch koordinierte Bahn zu planen, oder ob die sequentielle Ausführung beider Trajektorien keinen großen Zeitverlust bedeutet. Es wird unterschieden, ob sich bei der Zielfindung die Manipulatoren überkreuzen mussten oder nicht. Die Ergebnisse stehen in der folgenden Tabelle. Die Deadline von 60 Sekunden wurde nie überschritten, somit liegt die Erfolgsrate immer bei 100%.

	Über Kreuz		Nicht über Kreuz		Alle Pfade	
	Planung	Bewegung	Planung	Bewegung	Planung	Bewegung
14 DOF	20.86	4.31	18.83	3.7	19.48	3.89
2x7 DOF	15.47	7.47	14.38	7.43	14.73	7.45

Tabelle 5.6: Numerische Ergebnisse der parallelisierten Pfadplanung im ungeschützten Bereich – Planungszeiten und Ausführungszeiten gemittelt über 100 Durchläufe.

Wie erwartet dauert die sequentielle Ausführung etwa doppelt so lange wie die parallele Ausführung beider Trajektorien. Und auch die Planungszeit ist wie schon im ersten Szenario im getrennten Planungsfall immer kürzer. Kombiniert man Planungs- und Ausführungszeiten, zeigt sich, dass auch in der Gesamtzeit die getrennte Pla-

nung minimal vorne liegt. Jedoch handelt es sich dabei im Mittel nur um eine Sekunde. Es ist also prinzipiell egal, welche der beiden vorgeschlagen Strategien zum Einsatz kommt, da durch die zufallsbasierte Planung die Zeiten immer um einige Sekunden variieren werden.

5.3 Auswertung der zweiarmigen Manipulationsszenarien

Bei der zweiarmigen Manipulation hat sich gezeigt, dass es sich durchaus lohnen kann, die Pfadplanung auf zwei Planungsinstanzen aufzuteilen. Wenn die Aufgabe nach den Kriterien aus Kapitel 5.1.2 aufgeteilt werden kann, bringt die getrennte Planung enorme Vorteile. Arbeiten die Manipulatoren im geteilten Bereich, sind zwar immer noch kleine Geschwindigkeitsvorteile zu erkennen, aber diese sind bei einer zufallsbasierten Routenfindung zu vernachlässigen. Um möglichst effiziente Ergebnisse zu erzielen, reicht es also nicht, alle Manipulationsaufgaben mit einer Strategie zu bearbeiten. Für jede Situation muss der passende Ablauf gefunden werden. Mit diesen Ergebnissen ist es möglich, ein erstes Konzept für ein zweiarmiges Planungswerkzeug zu entwerfen. Aus den Beobachtungen lässt sich ein Ablauf wie in Abbildung 5.9 extrahieren.

Nach Abbildung 5.9 gibt es drei Möglichkeiten eine Aufgabe mit zwei Armen zu lösen. Wenn sich die Ziele der Arme in disjunkten Bereichen befinden, kann die Planung und die Ausführung auf zwei Prozesse verteilt werden. Ist dies nicht der Fall muss untersucht werden, ob beide Zielkonfigurationen zusammen eine kollisionsfreie Pose ergeben. Tritt dieser Fall ein, kann versucht werden über 14 DOF zu Planen um eine elegante parallelisierte Lösung zu erhalten. Kollidieren die Zielkonfigurationen ist dies nicht möglich. Die Planung kann mit je 2 x 7 DOF durchgeführt werden, unter dem Vorbehalt, dass eventuell zu verschiebende Objekte während der Planungsphase berücksichtigt werden. Die Ausführung muss dann allerdings sequentiell erfolgen. Dazu muss sich der zuerst bewegende Manipulator natürlich nach seiner Interaktion wieder aus dem kritischen Bereich zurückziehen. Dieser Algorithmus soll im nächsten Kapitel als Ausgangspunkt für die Integration von OpenRAVE in die echte Robotersteuerung dienen.

Ausschlaggebend für die kürzeren Planungszeiten bei der 2 x 7 DOF Variante ist die Aufteilung der Rechenzeit auf zwei Prozessoren. Alle zufallsbasierten Planungsmethoden sind prinzipiell gut geeignet für die Mehrkernberechnung. In der Arbeit von Herms wurde dies bereits gezeigt [18]. Gelingt es also die Planung im 14 DOF Raum ebenfalls zu teilen, erhält diese eventuell einen ähnlichen Geschwindigkeitszuschuss wie die unabhängige Planung. Ob dieser genau so groß ausfällt wie die Trennung der Planung auf einer logischen Ebene, damit die Unteraufgaben komplett unabhängig sind, müsste experimentell bestimmt werden.

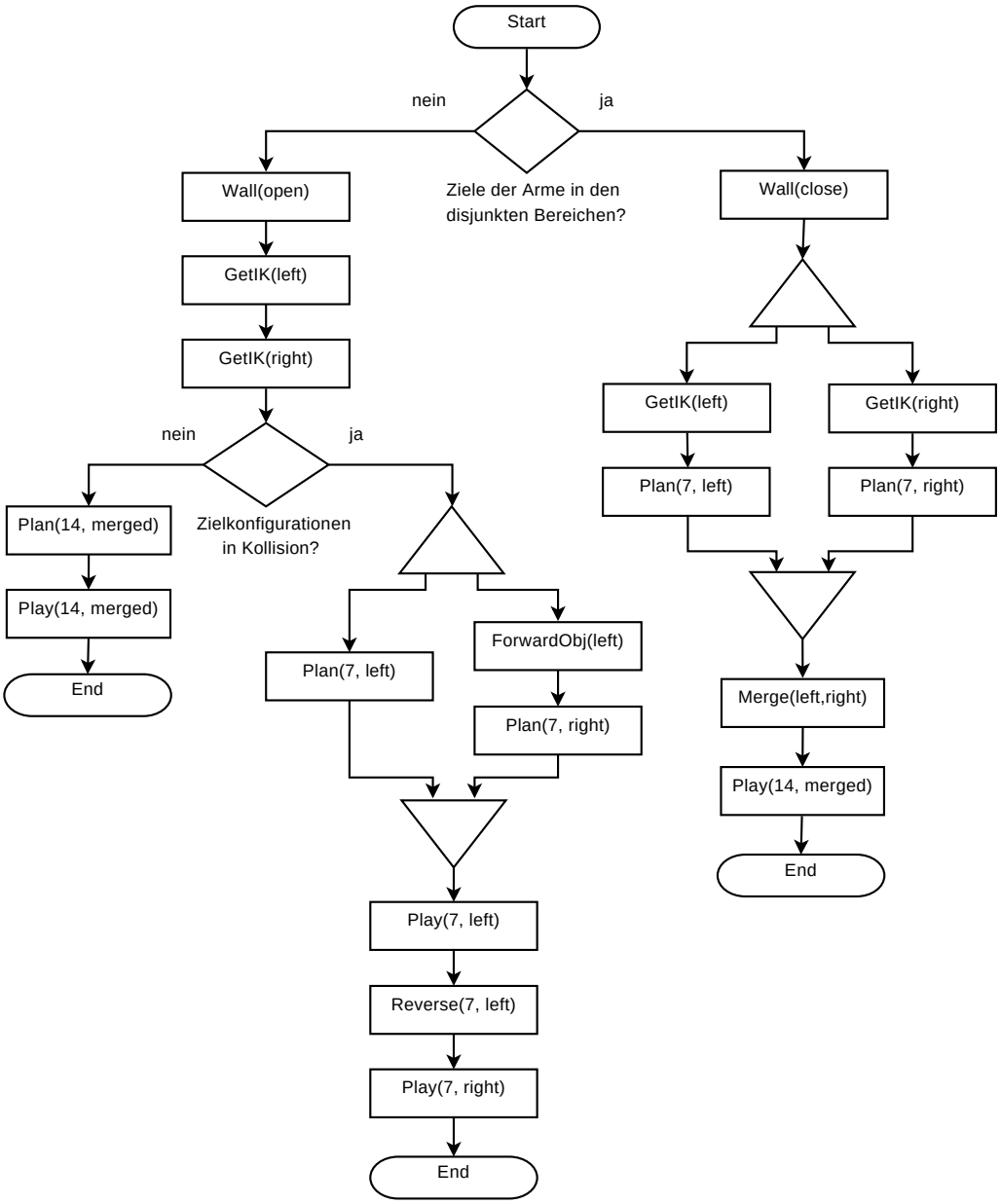


Abbildung 5.9: Allgemeiner Programmablauf für zweiarmige Manipulationsaufgaben

6 Integration der Strategien für zweiarmige Manipulation auf Rollin' Justin

Bei der Integration von Simulationen auf echte Systeme treten oftmals Schwierigkeiten auf, die im voraus nicht berücksichtigt wurden. Deshalb ist es notwendig die zuvor erzielten Ergebnisse in der Realität zu validieren. Äußere Einflüsse wie Gravitation, Geschwindigkeiten oder andere physikalische Größen können speziell bei mechatronischen Systemen ungewollte Nebenwirkungen hervorrufen. Dies ist nicht zu unterschätzen, allein über den Schritt von der Simulation zur Realität sind diverse Arbeiten entstanden [1], [30]. Um also sicherstellen zu können, dass die entwickelten Algorithmen auf dem echten Roboter genau so gut funktionieren wie in der Simulation, ist die Integration ein wichtiger Schritt.

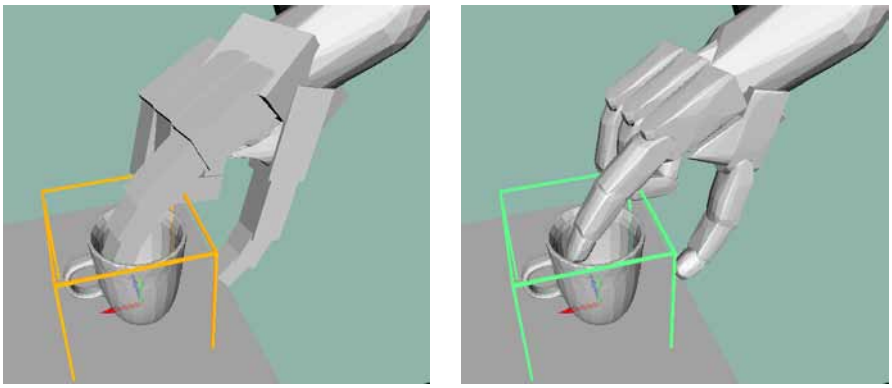
6.1 Differenzen zwischen Simulation und Realität

Wie bereits angedeutet können in der Realität diverse Differenzen zur Simulation auftreten. Versuche an einem Modell können nur bedingt Tests an einem realen Objekt ersetzen. Anhand eines Modells lassen sich einfache Grundlagen gut erklären und bestimmen. Um die aufgestellten Thesen jedoch eindeutig zu bestätigen, sind die Unterschiede jedoch meist zu groß. Dieses Kapitel soll die wichtigsten Unterschiede von Justins Modell in der Simulation zum realen Roboter identifizieren.

6.1.1 Unterschiede zwischen dem Modell und dem echten Roboter

Bei den bisherigen Verfahren zur Steuerung von Justin waren die Ungenauigkeiten im Modell nicht von allzu großer Bedeutung. Bei den verwendeten Verfahren zur lokalen Pfadplanung und vor allem bei ein gelernten Abläufen ist es nicht wichtig die Fehler des Modells genau zu kennen. Beim *teach in* Verfahren wird dem realen Roboter eine genaue Bahnbewegung vorgegeben, die späterhin einfach abgefahren werden kann. Dabei ist es egal ob sich die Simulation anders verhält als die Realität, weil die Realität die bestimmende Größe ist. Nach einer erneuten Kalibration muss der teach in Vorgang eventuell wiederholt werden. Bei der Pfadplanung ist genau das Gegenteil der Fall. Die Simulation geht davon aus, dass das Robotermodell der Wirklichkeit entspricht. Dies ist natürlich nicht unbedingt der Fall und der echte Roboter verhält sich dadurch bei der Ausführung eventuell anders als in der Simulation. Besonders bei sampling-basierten Verfahren ist es jedoch wichtig die Fehler des Modells genau zu kennen, da jede errechnete Bahn anders aussieht und sich der Roboter dadurch anders verhält.

Beginnend mit den offensichtlichsten Unterschieden des Modells muss zunächst noch einmal das Kollisionsmodell von Justin betrachtet werden. Die Reduzierung auf möglichst wenig Polygone ist förderlich für die Geschwindigkeit der Planung. Umso weniger Polygone das Modell hat, desto weniger Kollisionsabfragen müssen durchgeführt werden. Aber genau dieser Punkt ist wiederum hinderlich in der echten Welt. Ist die Hülle des Roboters zu sehr abstrahiert, können feine Bewegungen nicht ausgeführt werden. Die Simulation detektiert zu schnell Kollisionen, wo in Wirklichkeit keine sind. Dies gilt vor allem für die Hände. Bisher ist jedes Segment der Finger als einfacher Quader dargestellt. So genannte *boundingboxes* sind gut geeignet um kurze Planungszeiten zu erzielen. Allerdings sind diese für reale Verhältnisse zu grob. Wie bereits beim Oberkörper müssen daher konvexe Hüllen verwendet werden. Die Grundstruktur ähnelt dabei noch dem Originalkörper, die Polygonzahl ist jedoch stark reduziert. Abbildung 6.1 zeigt den unterschied zwischen der alten und der neuen Hand im Kollisionsmodell. Der orange farbene Kasten symbolisiert dabei die entstehende Kollision mit der Tasse, die durch das ungenaue Modell der Hand auftreten. Die Pose der Hand ist identisch.



(a) boundingboxes (b) konvexe Hüllen

Abbildung 6.1: Kollisionsmodelle der Hand

Zudem hat die Hand noch neue Fingerspitzen erhalten, die einen besseren Halt garantieren sollen. Das neue Element ist lediglich etwas länger und hat eine zusätzliche Krümmung nach innen. Die Kinematikbeschreibung ist identisch. Der restliche Aufbau des Kollisionsmodells, sowie des Renderingmodells bleibt gleich. Die Skalierung der Arme aus Kapitel 3.5 bleibt bestehen. Durch die zusätzliche Ausdehnung werden Ungenauigkeiten zwischen der echten Roboterstellung und der Simulation kompensiert.

Ein weiter Kritikpunkt findet sich in der Planung mit gegriffenen Gegenständen. Bisher wurden die Hände immer nur in die Nähe des Objektes gebracht und nicht gegriffen, sondern nur als gegriffen deklariert und als Teil des Roboters betrachtet. Berührt der Roboter Objekte in der Simulation steht dieser nämlich in Kollision und eine Planung ist nicht zulässig. In der echten Welt ist dies hinderlich, da der

echte Roboter immer in Kollision mit zu manipulierenden Objekten steht. Um die Simulation also an die Realität anzupassen, müssen Gegenstände die dem Roboter angefügt sind aus der Selbstkollision ausgeschlossen werden. Dies erhöht zum einen die Genauigkeit des Modells und liefert eine zulässige Ausgangskonfiguration für die Planung.

6.1.2 Gravitationsbedingte Abweichungen der Torso- und Armkinematik

Anders als beim echten Roboter wirken sich in der Simulation keine Nachgiebigkeiten der Gelenke auf Bewegungen aus. Die Simulation betrachtet also jedes Gelenk als hundertprozentig steif. Die Leichtbauweise von Justins Torso und der Arme kann dieses Verhalten jedoch nicht garantieren. Die *RoboDrive-Antriebe*[7] in den Gelenken, die Seilzüge im Torso und bedingt auch die Kohlefaserstruktur der Arme leiden dabei unter Gravitationseinflüssen. Umso länger der Hebel der Arme desto weiter neigen sich die Gelenke. Bei voll ausgestreckten Armen ergeben sich so mehrere Zentimeter Differenz zwischen Simulation und Modell. In Abbildung 6.2 ist dieser Effekt deutlich zu sehen.

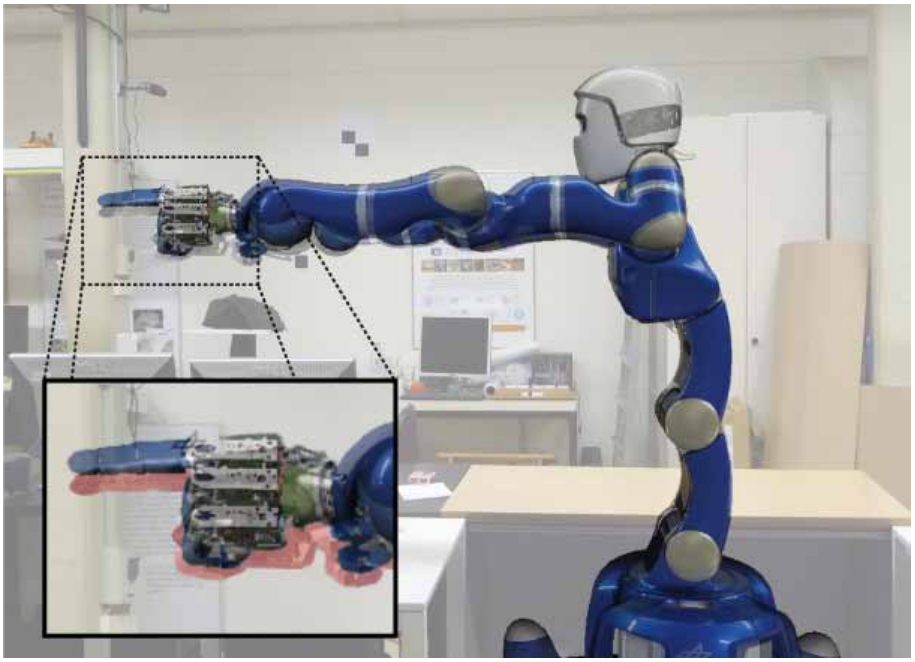


Abbildung 6.2: Gravitationsbedingte Ungenauigkeiten im idealen Robotermodell. Justin hängt in der Wirklichkeit bis zu 3 cm am Endeffektor durch. Der echte Roboter ist im Hintergrund transparent zu erkennen. Der rot eingefärbte Bereich zeigt die echte Hand.

[7] Die RoboDrive-Antriebe die in den Gelenken von Justin eingesetzt werden benutzen *Harmonic Drive* Getriebe. Diese zeichnen sich durch eine hohe Übersetzung und Steifigkeit aus [17].

Dieser Umstand muss für die Planung der echten Roboterbewegung unbedingt berücksichtigt werden. Während eine Trajektorie in der Simulation völlig kollisionsfrei verläuft, könnte die gleiche Bahn in der echten Welt zu ungewollten Berührungen mit der Tischebene oder anderen Hindernissen führen. Dabei reicht es nicht aus die Gelenkwinkelstellung in der Zielkonfiguration einfach nachzubessern. Der Offset in den Gelenkwinkeln muss während der gesamten Planungsphase bekannt sein. Die beiden folgenden Abschnitte zeigen wie der Fehler im Modell abgeschwächt werden kann.

Der entscheidende Punkt hierbei ist das Drehmoment auf den Gelenken. Umso größer das Drehmoment desto größer die Abweichung. Da während der Planung allerdings keine Momentenwerte der echten Gelenke vorhanden sind, muss auf verschiedene physikalische Modelle zurückgegriffen werden. Zum einen ein Steifigkeitsmodell im Torso und zum anderen ein Gravitationsmodell der Arme. Anders als bei den Armen wurde die Steifigkeit in den Torsogelenken bisher noch nicht bestimmt. Insbesondere das vierte Torsogelenk an der Brust gibt stark nach. Die Steifigkeit k wird in $\frac{[Nm]}{[rad]}$ angegeben und ist also ein Maß für die Verdrehung des Gelenks bei einem anliegenden Moment. Zum Bestimmen der Steifigkeit kann ein iterativer Prozess gewählt werden. Justins beide Arme werden dazu Schritt für Schritt voll ausgestreckt und waagerecht in einem Bogen von Hinten nach Vorne Gefahren. Mit den gegebenen Massen der Arme kann ein theoretisches Moment für den Torso bestimmt werden und somit die Torsoverkippung. Bei jedem Schritt wird das anliegende Drehmoment an der vierten Torsoachse aus dem Modell berechnet und die Torsoverkippung in der vierten Achse mittels eines Referenzschalters gemessen. Dadurch entsteht das in Abbildung 6.3 dargestellte Diagramm.

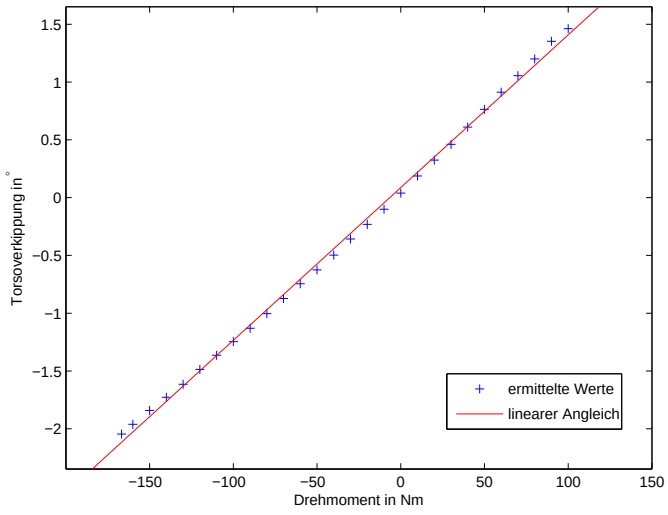


Abbildung 6.3: Steifigkeit der vierten Torsoachse

Die erhaltene Funktion kann nahezu als linear betrachtet werden. Die Abweichung beträgt dabei $< \pm 0.07°$. Während jedes Planungsschritt muss also überprüft werden, wie stark sich die Konfiguration der Arme verändert. Anhand der neuen Armstellung muss die Torsoposition nach gebessert werden. So entsteht ein realitätsnahes Verhalten des Oberkörpers während der simulierten Trajektorie.

Die Belastung in den Armgelenken wird abhängig des bereits erwähnten Massenmodells bestimmt. Dieses beinhaltet die jeweiligen Massen und den Massenschwerpunkt (engl: *Center of gravity, COG*) der Armsegmente. Je nach Stellung der Arme variiert das Drehmoment in den einzelnen Gelenken. Mit diesem Drehmoment und den gegebenen Steifigkeiten der Arme kann die Verdrehung der Gelenke bestimmt werden. Bei Vergleichen der berechneten Drehmomenten mit den gemessenen Drehmomenten waren unter der Verwendung des Steifigkeitsmodelles nur geringe Unterschiede im Millimeterbereich zu erkennen. Der Fehler in den Armgelenken muss auch beim Berechnen der Inverskinematik berücksichtigt werden.

Die Neigung des letzten Torsoelements macht sich natürlich in der Stereobildverarbeitung bemerkbar. Steht der Kopf schief, ist die Lage der aufgenommenen Bilder auch schief. Allerdings können diese Neigungen durch den in Justin integrierten Inertialsensor (engl: *Inertial Measurement Unit, IMU*)[8] erkannt werden. Die Messwerte der IMU fließen in die Lokalisierung der Bildverarbeitung mit ein. In der Simulation gibt es keine IMU, lediglich das zuvor beschriebene Modell des Torsos. Um Gegenstände also relativ zur virtuellen Kamera exakt in der Simulation ausrichten zu können, muss ebenfalls auf die Torsoverkippung zurückgegriffen werden.

6.2 Integration der Pfadplanungsstrategien in die Steuerungssoftware

Die bisher entwickelten Skripte können nicht direkt auf den Roboter übertragen werden. Mittels des *TOP Script Servers* und der dazugehörenden *TOP Command GUI* können Operationen, wie zum Beispiel eine bestimmte Ausgangsstellung anzufahren, ausgeführt werden. Dabei handelt es sich um eine Reimplementierung der in [4] beschriebenen Software. Jede Operation besteht wiederum aus Elementaroperationen, die einzelne Gelenkwinkel steuern. Zwischen diesen Gelenkwinkelstellungen agiert ein Interpolator, der aus den gegebenen Punkten eine Bahn produziert. Besteht die Operation also aus nur einer Zielkonfiguration, sind Kollisionen nicht ausgeschlossen, da kein Planer gestartet wurde. Um einen geplanten Pfad auszuführen, müssen mehrere Elementaroperationen hintereinander gehängt werden. Dabei muss der Interpolator auf verschiedene Kriterien achten. Einzelne Gelenke dürfen die Maximalgeschwindigkeit nicht überschreiten, jedoch soll deren Zielstellung so schnell wie möglich erreicht werden, ohne das dabei zu große Beschleunigungen auf-

[8] Ein Inertialsensor ist ein gekoppelter Sensor bestehend aus drei orthogonal ausgerichteten Beschleunigungssensoren und drei ebenfalls orthogonal ausgerichteten Drehratensensoren.

treten. Diese resultieren in unerwünschten Sprüngen auf dem Roboter. Bewegt sich der Roboter zudem in mehreren Gelenken gleichzeitig, ist es wünschenswert die einzelnen Gelenke miteinander zu synchronisieren. Alle Gelenke sollen gleich lange brauchen um ihre Zielposition zu erreichen. Ihnen werden also unterschiedliche Geschwindigkeiten zugewiesen. Die Skripte müssen also derart angepasst werden, dass jede Unteraufgabe als Operation bereitgestellt wird.

6.3 Testszenario zur Validierung der Planungsstrategien

Das Testszenario am echten Roboter soll der Validierung der bisher entwickelten Strategien dienen. Dazu werden die in Kapitel 5 erzielten Resultate verwertet und auf den echten Roboter überführt. Damit soll überprüft werden, wie gut die Simulation mit dem echten Roboter übereinstimmt und welche Probleme eventuell auftreten können.

6.3.1 Aufbau

Als Versuchsumgebung wird der in Kapitel 5 verwendete Aufbau nachgebildet. Justin wird wie in der Simulation vor dem gewohnten Tisch platziert. Die zwei zusätzlichen Ablageflächen werden durch zwei kleine Beistelltische repräsentiert. Auf diesen werden die bereits vorgestellten Würfel zufällig abgelegt. Zur Unterstützung der Bildverarbeitung sind die Würfel eingefärbt, da die Holzmaserung die Stereoaufnahmen irritiert. An dieser Stelle soll dies nicht weiter stören, da sich diese Arbeit nicht mit Bildverarbeitung beschäftigt. Justins Ausgangsstellung ist der aus den vorherigen zweiarmigen Szenarien weitest möglich nachempfunden. Abbildung 6.4 zeigt den Aufbau in der realen Testumgebung.

Abbildung 6.4: Aufbau des realen Testszenarios

6.3.2 Ablauf

Der Ablauf entspricht in den Grundzügen einer Kombination aus den Eigenschaften von Szenario 1 und 2 aus Kapitel 5. Die Startposition der Würfel ist allerdings nicht bekannt. Vor der Manipulation muss sich Justin also zu den Würfeln wenden und diese lokalisieren. Erst danach kann er seine Ausgangsstellung einnehmen und die Planung beginnen. Von nun an sollen die Würfel auf den Haupttisch versetzt werden. Dabei wird die in Abbildung 5.9 gezeigte Strategie verfolgt. Liegt die Zielposition des jeweiligen Würfels also im unabhängigen Bereich, wird über 2 x 7 DOF geplant. Schneiden sich die Einflussbereiche der Arme wird zunächst versucht eine Lösung mit den 14 Gelenkwinkeln der Arme zu generieren. Ist dies nicht möglich, findet die Planung parallel und die Ausführung sequenziell statt. Am Ende des Szenarios sollen die Würfel den Schriftzug *DLR* auf der Tischoberfläche bilden (siehe Abbildung 6.5).

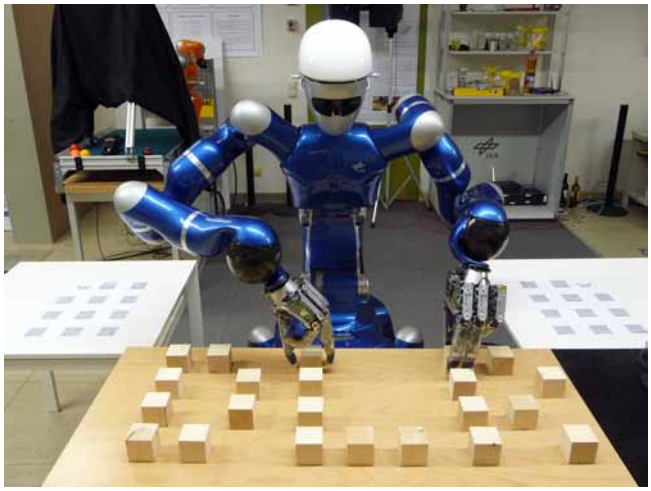


Abbildung 6.5: DLR Schriftzug auf der Tischoberfläche

6.3.3 Ergebnis

Bei diesem Testszenario hat sich bewiesen, dass der verfolgte Ansatz prinzipiell gut geeignet ist für die zweiarmige Manipulation. Durch die eingepflegten Verbesserungen des Modells hat sich die Simulation weitestgehend so verhallten wie der reale Roboter. Leider hat sich gezeigt, das der Oberkörper, und damit auch die Arme, in Realität etwas weiter durchhängen wie angenommen. Dies ist darauf zurückzuführen, dass während der Planung über 2 x 7 Gelenkwinkel nicht berücksichtigt werden kann, wie weit der Oberkörper durchhängt wenn sich beide Arme simultan bewegen. Das Ergebnis ist zwar nicht exakt wie erwartet, dennoch ist diese Erkenntnis wichtig für spätere Arbeiten.

	Transitpfade	Transferpfade			Gesamtdauer
	2 x 7 DOF parallel	2 x 7 DOF parallel	2 x 7 DOF sequenziell	14 DOF	
DLR Logo	23.86	18.45	19.02	26.34	483.50

Tabelle 6.1: Numerische Ergebnisse der kombinierten Pfadplanungstrategien auf Rollin' Justin.

Die numerischen Ergebnisse in Tabelle 6.1 sind alle ohne Ausführungszeiten aufgeführt, da diese auf dem echten Roboter von zu vielen Bedingungungen abhängt. Darunter die maximale Gelenkwinkelgeschwindigkeit und die Übertragungszeiten der kommandierten Trajektorien. Bei der Planungszeit über 14 DOF fällt auf, das diese im Verlgeich relativ hoch liegt. Dies liegt daran, dass die über 14 DOF gelegten Würfel am Rande des Arbeitsbereiches des linken Manipulators lagen. Der rechte Arm würde an diese Stelle um einiges besser gelangen. Unter dem Aspekt der Aufgabenplanung würde diese Situation bereits im Vorraus ausgefiltert werden.

6.4 Auswertung der Integration

Die Integration von OpenRAVE in die Steuerungsumgebung von Justin hat sich als nicht allzu schwer erwiesen. Durch die übersichtliche Pluginstruktur von OpenRAVE und nicht zuletzt durch die Fähigkeit der TOP Software, einfache skriptbasierte Abläufe zu interpretieren, war es möglich die Planungsumgebung schnell und effektiv zu implementieren. Mit OpenRAVE ist es nun auch möglich, die bisher bereits erstellten Bewegungsabläufe in Zukunft auf geplanten, kollisionsfreien Pfaden abzufahren.

Es wurde gezeigt, dass sich die in der Simulation getesteten Ergebnisse auch auf dem realen Roboter erzielen lassen. Während der Roboter die geplante Trajektorie ausführt, ist es möglich bereits den nächsten Schritt zu planen. Dadurch kann ein flüssiger Ablauf erzielt werden. Weiterhin wurde gezeigt, dass es nicht immer reicht erstellte Algorithmen nur auf der Simulation zu überprüfen. Dadurch das die Arme nicht ideal unabhängig sind, ist die unabhängige Planung nur bedingt möglich. Die Nachgiebigkeit des Torsos als verbindendes Element, wirkt sich negativ auf die unabhängige Planung aus. Dies wird insbesondere zum Problem wenn der Roboter zu seinem Eigengewicht noch schwere Gegenstände tragen muss. Für einfache *pick and place* Aktionen, bei denen Genauigkeit keine große Rolle spielt, ist der entwickelte Algorithmus jedoch gut einsetzbar.

7 Weiterführende Arbeiten und Ausblick

Im letzten Kapitel wird noch einmal gezeigt welche Probleme noch zu Lösen sind, und wo schon erste Lösungsversuche angesetzt wurden. Dazu unterteilt sich dieses Kapitel in zwei Abschnitte, zum einen die weiterführenden Arbeiten und danach den Ausblick.

7.1 Weiterführende Arbeiten

Ein großer Kritikpunkt an OpenRAVE und den damit verwendeten Planern ist die Planungszeit. Die Planungszeit steigt mit der Komplexität der Aufgabe an und bildet daher immer ein Flaschenhals. Um die Planungszeit zu reduzieren wurden bereits einige Ansätze verfolgt. Darunter zunächst die inkrementelle Änderung der Gelenkwinkelauflösung. Die Gelenkwinkelauflösung ist maßgeblich für die Schrittweite der Gelenkwinkeländerung während der Planung. Beträgt diese also 1° werden die Gelenke in jedem Iterationsschritt der Planung um 1° verdreht. Zwischen zwei Konfigurationen wird der Weg ohne Kollisionsabfrage interpoliert. Umso kleiner die Gelenkwinkelauflösung desto mehr Konfigurationen beinhaltet eine Trajektorie. Bei einer Auflösung von 1° benötigt die Optimierung der Trajektorie durchschnittlich 76% der Gesamtplanunsgzeit. Wird die Auflösung auf 2° gestellt beträgt dieser Prozentsatz nur noch 61% da die Optimierungszeit um mehr als 5 Sekunden sinkt. Es gilt nun also herauszufinden wie weit die Auflösung verändert werden kann, ohne dabei in Kollision mit möglichen Hindernissen zu geraten. Erste Versuche auf dem echten System haben gezeigt,dass bei einer Auflösung von 2° durchaus noch kollisionsfreie Bahnen erzeugt werden können. Allerdings muss dies noch in mehreren Umgebungen verifiziert werden.

Eine weitere Möglichkeit die gefühlte Wartezeit zu minimieren, ist eine Kaskadierung der Bewegungsabläufe und der nachfolgenden Planung. Sobald die Planung abgeschlossen ist und die Trajektorie auf dem echten Roboter abgefahren wird, kann bereits mit der Planung für den nächsten Schritt begonnen werden. Die Simulation befindet sich zu diesem Zeitpunkt schon auf dem Ausgangsstandpunkt für die nächste Planung. Dieses Verhalten kann ausgenutzt werden um die Planung früher zu beginnen. Die Ersparnis für die Planungszeit ist dabei gleich der Ausführungszeit auf dem echten Roboter. Teilt man die Planung in kürzere Segmente ein, könnten so zusätzlich Sensorinformationen in die Pfadplanung mit einfließen. Denkbar wäre vor allem das Einbinden von Kamerabildern um eine Hand-Augen-Koordination zu erreichen. Diese Art des Pipelinings wird als *Visual Servoing* bezeichnet und wurde beireits in einigen Arbeiten praktiziert [21], [10].

7.2 Ausblick

Im Kontext des Ausblicks müssen vor allem die Auswirkungen der nicht ideal unabhängigen Arme im Bezug auf die parallele unabhängige Pfadplanung, betrachtet werden. Es wurde gezeigt, dass es nötig ist, die Planungsstrategie individuell an die Aufgabe anzupassen, um einen Geschwindigkeitsvorteil zu erreichen. Dabei kann die Parallelisierung von Unteraufgaben, und damit die Verteilung der Rechenzeit auf mehrere Prozesse, einen Teil dazu beitragen. Jedoch hat sich gezeigt, dass die Arme sich nicht so einfach entkoppeln lassen. Die Posen der Arme am echten Roboter sind schwerkraftsbedingt voneinander abhängig. Es muss daher versucht werden, einen Weg zu finden um diese Eigenschaft entweder zu beseitigen, oder zu kompensieren.

Des weiteren kann der entworfene Algorithmus noch verfeinert werden. Bisher stellt die virtuelle Wand eine unverschiebbare Grenze dar. Dabei wäre es um einiges effizienter die Wand in gewissen Fällen nach links oder rechts zu schieben. Versucht zum Beispiel der linke Manipulator in den gemeinsam nutzbaren Bereich einzudringen, während der rechte Arm im disjunkten Bereich agieren will, muss die Planung nicht unbedingt über 14 Freiheitsgrade erfolgen, geschweige denn die Ausführung sequentiell. Verschiebt man nun die Wand in Richtung des rechten Arms, wird dieser nicht behindert und der linke Arm kann ebenfalls unabhängig agieren. Eine Planung mit 2 x 7 DOF könnte so weiterhin gewährleistet werden. Zudem müssen die Schwellwerte dynamisch gestaltet werden, falls die Manipulatoren Gegenstände transportieren.

Langfristig können weitere Möglichkeiten von OpenRAVE ausgeschöpft werden. Zum Beispiel besteht die Möglichkeit Greifplaner in die Umgebung mit aufzunehmen. Auch das virtuelle Kamerabild der Simulation kann zum besseren Vergleich mit der Realität visualisiert werden. Der verwendete Interpolator des echten Roboters kann zudem als Controller des simulierten Roboters dienen, um auch während der Bewegung Fehlerquellen auszuschließen. Das noch junge Framework verspricht zudem noch eine Vielzahl neuer Funktionen in der Zukunft.

Literaturverzeichnis

[1] Balakirsky, S. ; Carpin, S. ; Dimitoglou, G. ; Balaguer, B.: From Simulation to Real Robots with Predictable Results: Methods and Examples. In: *Performance Evaluation and Benchmarking of Intelligent Systems* (2009)

[2] Bohlin, R. ; Kavraki, L.E.: Path planning using lazy PRM. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2000, S. 521–528

[3] Borst, C. ; Ott, C. ; Wimbock, T. ; Brunner, B. ; Zacharias, F. ; Bauml, B. ; Hillenbrand, U. ; Haddadin, S. ; Albu-Schaffer, A. ; Hirzinger, G.: A humanoid upper body system for two-handed manipulation. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2007

[4] Brunner, B. ; Vogel, J. ; Hirzinger, G.: Aufgabenorientierte Fernprogrammierung von Robotern (Task Oriented Teleprogramming of Robots). In: *at-Automatisierungstechnik* 49 (2001), S. 312

[5] Butterfass, J. ; Grebenstein, M. ; Liu, H. ; Hirzinger, G.: DLR-Hand II: Next generation of a dextrous robot hand. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2001

[6] Choset, H.M. ; Hutchinson, S. ; Lynch, K.M. ; Kantor, G. ; Burgard, W. ; Kavraki, L.E. ; Thrun, S.: *Principles of robot motion: theory, algorithms, and implementation*. The MIT Press, 2005

[7] Craig, J.J.: *Introduction to robotics: mechanics and control*. Prentice Hall, 2004

[8] Denavit, J ; Hartenberg, R S.: A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. In: *ASME Journal of Applied Mechanics*, 1955, S. 215–221

[9] Diankov, R.: *Introduction to OpenRAVE Scripting*. http. `http://openrave.programmingvision.com/index.php?title=Started:Scripting`. Version: Februar 2010

[10] Diankov, R. ; Kanade, T. ; Kuffner, J.: Integrating Grasp Planning and Visual Feedback for Reliable Manipulation. In: *IEEE/RAS International Conference on Humanoid Robots*, 2006

[11] Diankov, R. ; Kuffner, J.: OpenRAVE: A Planning Architecture for Autonomous Robotics / Robotics Institute. Version: 2008. `http://openrave.programmingvision.com`. 2008. – Forschungsbericht

[12] Diankov, R. ; Ratliff, N. ; Ferguson, D. ; Srinivasa, S. ; Kuffner, J.: Bispace planning: Concurrent multi-space exploration. In: *Proceedings of Robotics: Science and Systems IV* (2008)

[13] Fuchs, M. ; Borst, C. ; Giordano, P.R. ; Baumann, A. ; Kraemer, E. ; Langwald, J. ; Gruber, R. ; Seitz, N. ; Plank, G. ; Kunze, K. u. a.: Rollin'Justin–Design considerations and realization of a mobile platform for a humanoid upper body. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2009

[14] Gharbi, M. ; Cortés, J. ; Siméon, T.: A sampling-based path planner for dual-arm manipulation. In: *IEEE/ASME International Conference Advanced Intelligent Mechatronics*, 2008

[15] Gharbi, M. ; Cortés, J. ; Siméon, T.: Roadmap Composition for Multi-Arm Systems Path Planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009

[16] Guiard, Y. ; Ferrand, T.: *Asymmetry in bimanual skills.* CRC, 1995

[17] HarmonicDrive: The Basics of Harmonic Drive Gearing. Version: 2006. `http://www.powertransmission.com/issues/0706/harmonic.htm`. 2006. – Forschungsbericht

[18] Herms, Andre: *Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren*, University of Magdeburg, Diploma Thesis, 2004

[19] Hirzinger, G. ; Sporer, N. ; Albu-Schaffer, A. ; Hahnle, M. ; Krenn, R. ; Pascucci, A. ; Schedl, M.: DLR's torque-controlled light weight robot III-are we reaching the technological limits now? In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2002

[20] Ihme, T.: *Steuerung von sechsbeinigen Laufrobotern unter dem Aspekt technischer Anwendungen*, Universität Magdeburg, Diss., 2002

[21] Ihme, T. ; Ruffler, U.: Motion Planning based on Realistic Sensor Data for Six-Legged Robots. In: *Autonome Mobile Systeme 2007 (AMS 2007)*, Springer, 2007, S. 247–253

[22] Koga, Y. ; Latombe, J.C.: On multi-arm manipulation planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 1994, S. 945–945

[23] Koren, Y. ; Borenstein, J.: Potential field methods and their inherent limitations for mobile robot navigation. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 1991, S. 1398–1404

[24] Kuffner, J.J. ; LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*, 2000, S. 995–1001

[25] KUKA: *Präzisionsware „Made in Germany“*. http. `http://www.kuka-robotics.com/germany/de/solutions/`. Version: Februar 2010

[26] Latombe, J.C.: *Robot motion planning*. Springer, 1991

[27] LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. (1998)

[28] LaValle, S.M.: *Planning algorithms*. Cambridge Univ Pr, 2006

[29] Li, T.Y. ; Latombe, J.C.: On-line manipulation planning for two robot arms in a dynamic environment. In: *The International Journal of Robotics Research* 16 (1997), S. 144

[30] Martin, F.: Real robots don't drive straight. In: *AAAI Spring Symposium, Robots and Robot Venues: Resources for AI Education*, 2007

[31] Ott, C. ; Eiberger, O. ; Friedl, W. ; Bäuml, B. ; Hillenbrand, U. ; Borst, C. ; Albu-Schäffer, A. ; Brunner, B. ; Hirschmüller, H. ; Kielhöfer, S. u. a.: A humanoid two-arm system for dexterous manipulation. In: *IEEE/RAS International Conference on Humanoid Robots*, 2006, S. 43

[32] Paul, R.P.: *Robot manipulators*. MIT Pr., 1989

[33] Schraft, R.D. ; Hägele, M. ; Wegener, K.: *Service-roboter-visionen*. Hanser, 2004

[34] SpiegelOnline: *Roboter: Auf der Cebit gibt es neben miniaturisierter Elektronik auch handfestes zu sehen*. http. `http://www.spiegel.de/netzwelt/gadgets/bild-679421-61714.html`. Version: Februar 2010

[35] Zacharias, F. ; Borst, C. ; Hirzinger, G.: Bridging the gap between task planning and path planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, S. 4490–4495

[36] Zacharias, F. ; Borst, C. ; Hirzinger, G.: Capturing robot workspace structure: representing robot capabilities. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007

[37] Zöllner, R. ; Asfour, T. ; Dillmann, R.: Programming by demonstration: Dual-arm manipulation tasks for humanoid robots. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004

A Anhang

A.1 Kinematische Definition von Justin in XML

```
<?xml version="1.0" encoding="utf-8"?>

<Robot name="Justin">
  <KinBody>
    <modelsdir>/inventor/</modelsdir>
    <!-- mobile plattform -->

    <!-- torso -->

    <!-- right arm -->
    <Body name="rightarm1" type="dynamic">  <!-- dyn. robot link -->
      <offsetfrom>torso4</offsetfrom>       <!-- start offset -->
      <RotationAxis>0 1 0 150</RotationAxis>    <!-- rotation -->
      <Translation>0.190 0.088 -0.256</Translation> <!-- transl. -->
      <Geom type="trimesh"> <!-- collision and rendermodel -->
    <data>convhull/right-lwr-3-link12-34-convhull.iv 1.2 1.2 1.2</↩
        data>
         <Render>right-lwr-3-link12-34.iv</Render>
      </Geom>
    </Body>

    <Joint type="hinge" name="RA1">     <!-- joint between -->
      <Body>torso4</Body>           <!-- link1 and -->
      <Body>rightarm1</Body>            <!-- link2 -->
      <offsetfrom>rightarm1</offsetfrom>    <!-- start offset -->
      <weight>1</weight>            <!-- weight -->
      <lostop>-170</lostop>         <!-- upper joint limit -->
      <histop>170</histop>          <!-- lower joint limit -->
      <axis>0 0 -1</axis>           <!-- rotationaxis -->
      <maxvel>1.5708</maxvel>           <!-- max velocity -->
      <resolution>1</resolution>        <!-- joint resolution -->
    </Joint>

    <Body name="rightarm2" type="dynamic">
      <offsetfrom>rightarm1</offsetf-rom>
      <RotationMat>1 0 0 0 0 -1 0 1 0</RotationMat>
      <Geom type="trimesh">
    <data>convhull/right-lwr-3-link23-45-convhull.iv 1.2 1.2 1.2</↩
        data>
         <Render>right-lwr-3-link23-45.iv</Render>
      </Geom>
    </Body>
```

```
<Joint type="hinge" name="RA2">
  <Body>rightarm1</Body>
  <Body>rightarm2</Body>
  <offsetfrom>rightarm2</offsetfrom>
  <weight>1</weight>
  <lostop>-120</lostop>
  <histop>120</histop>
  <axis>0 0 -1</axis>
  <maxvel>1.5708</maxvel>
  <resolution>1</resolution>
</Joint>

<Body name="rightarm3" type="dynamic">
  <offsetfrom>rightarm1</offsetfrom>
  <RotationMat>0 1 0 -1 0 0 0 0 1</RotationMat>
  <Translation>0 0 0.4</Translation>
  <Geom type="trimesh">
<data>convhull/right-lwr-3-link12-34-convhullV2.iv 1.2 1.2 1.2</↩
    data>
     <Render>right-lwr-3-link12-34.iv</Render>
  </Geom>
</Body>

<Joint type="hinge" name="RA3">
  <Body>rightarm2</Body>
  <Body>rightarm3</Body>
  <offsetfrom>rightarm3</offsetfrom>
  <weight>1</weight>
  <lostop>-170</lostop>
  <histop>170</histop>
  <axis>0 0 -1</axis>
  <maxvel>1.5708</maxvel>
  <resolution>1</resolution>
</Joint>

<Body name="rightarm4" type="dynamic">
  <offsetfrom>rightarm1</offsetfrom>
  <RotationMat>0 0 -1 -1 0 0 0 1 0</RotationMat>
  <Translation>0 0 0.4</Translation>
  <Geom type="trimesh">
<data>convhull/right-lwr-3-link23-45-convhull.iv 1.2 1.2 1.2</↩
    data>
     <Render>right-lwr-3-link23-45.iv</Render>
  </Geom>
</Body>

<Joint type="hinge" name="RA4">
  <Body>rightarm3</Body>
  <Body>rightarm4</Body>
  <offsetfrom>rightarm4</offsetfrom>
  <weight>1</weight>
  <lostop>-120</lostop>
  <histop>120</histop>
  <axis>0 0 -1</axis>
  <maxvel>1.5708</maxvel>
```

```
  <resolution>1</resolution>
</Joint>

<Body name="rightarm5" type="dynamic">
  <offsetfrom>rightarm1</offsetfrom>
  <RotationMat>0 -1 0 1 0 0 0 0 1</RotationMat>
  <Translation>0 0 0.79</Translation>
  <Geom type="trimesh">
<data>convhull/right-lwr-3-link56-convhull.iv 1.2 1.2 1.2</data>
    <Render>right-lwr-3-link56.iv</Render>
  </Geom>
</Body>

<Joint type="hinge" name="RA5">
  <Body>rightarm4</Body>
  <Body>rightarm5</Body>
  <offsetfrom>rightarm5</offsetfrom>
  <weight>1</weight>
  <lostop>-170</lostop>
  <histop>170</histop>
  <axis>0 0 -1</axis>
  <maxvel>1.5708</maxvel>
  <resolution>1</resolution>
</Joint>

<Body name="rightarm6" type="dynamic">
  <offsetfrom>rightarm1</offsetfrom>
  <RotationMat>0 0 1 0 -1 0 1 0 0</RotationMat>
  <Translation>0 0 0.79</Translation>
  <Geom type="trimesh">
<data>convhull/right-lwr-3-link67-convhull.iv 1.2 1.2 1.2</data>
    <Render>right-lwr-3-link67.iv</Render>
  </Geom>
</Body>

<Joint type="hinge" name="RA6">
  <Body>rightarm5</Body>
  <Body>rightarm6</Body>
  <offsetfrom>rightarm6</offsetfrom>
  <weight>1</weight>
  <lostop>-45</lostop>
  <histop>80</histop>
  <axis>0 0 -1</axis>
  <maxvel>1.5708</maxvel>
  <resolution>1</resolution>
</Joint>

<!-- right hand base -->
<Body name="rightarm7" type="dynamic">
  <offsetfrom>rightarm1</offsetfrom>
  <RotationMat>-1 0 0 0 0 1 0 1 0</RotationMat>
  <Translation>0 0 0.79</Translation>
  <Geom type="trimesh">
... <!-- geometry hand base -->
  </Geom>
```

```
    </Body>

    <Joint type="hinge" name="RA7">
      <Body>rightarm6</Body>
      <Body>rightarm7</Body>
      <offsetfrom>rightarm7</offsetfrom>
      <weight>1</weight>
      <lostop>-45</lostop>
      <histop>135</histop>
      <axis>0 0 -1</axis>
      <maxvel>1.5708</maxvel>
      <resolution>1</resolution>
    </Joint>

    <!-- thumb (RF1)-->

    <!-- forefinger (RF2)-->

    <!-- middlefinger (RF3)-->

    <!-- ringfinger (RF4)-->

    <!-- left arm -->

    <!-- left hand -->

    <!-- head -->

    <!-- adjacency information to avoid selfcollisions -->

    <!-- adjacency information torso -->

    <!-- adjacency information right arm -->
    <adjacent>rightarmbase rightarm1</adjacent>
    <adjacent>torso4 rightarmbase</adjacent>
    <adjacent>torso3 rightarm1</adjacent>
    <adjacent>torso4 rightarm1</adjacent>
    <adjacent>rightarm1 rightarm2</adjacent>
    <adjacent>rightarm2 rightarm3</adjacent>
    <adjacent>rightarm3 rightarm4</adjacent>
    <adjacent>rightarm4 rightarm5</adjacent>
    <adjacent>rightarm5 rightarm6</adjacent>
    <adjacent>rightarm5 rightarm7</adjacent>
    <adjacent>rightarm6 rightarm7</adjacent>

    <!-- adjacency information right hand -->

    <!-- adjacency information left arm -->

    <!-- adjacency information left hand -->

    <!-- adjacency information head -->

  </KinBody>
```

```
  <!-- used controller -->
  <Controller>IdealController</Controller>

  <!-- right hand manipulator -->
  <Manipulator>
    <base>torso4</base>
    <effector>rightarm7</effector>
    <IkSolver>JustinRightikfast</IkSolver>
    <armjoints>RA1 RA2 RA3 RA4 RA5 RA6 RA7</armjoints>
    <joints>JRF1_0 JRF1_1 JRF1_2
            JRF2_0 JRF2_1 JRF2_2
            JRF3_0 JRF3_1 JRF3_2
            JRF4_0 JRF4_1 JRF4_2</joints>
    <opened>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</opened>
    <closed>0 0 0 0 0 0 0 0 0 0 0 0</closed>
    <Translation>0 0 0</Translation>
  </Manipulator>

  <!-- left hand manipulator -->

  <!-- head manipulator -->

</Robot>
```

A.2 Verwendete Greifframes

Transformationen für das Greifen des Zielobjekts in den einarmigen Szenarien:

$$ {}^{Target}T_{RightTCP} = \begin{pmatrix} 0.00 & 0.100 & 0.000 & -0.300 \\ -0.100 & -0.000 & 0.000 & 0.020 \\ 0.000 & -0.000 & 0.100 & 0.010 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{pmatrix} \tag{A.1} $$

Transformationen für das Greifen der Würfel in den zweiarmigen Szenarien:

$$ {}^{Cube}T_{LeftTCP} = \begin{pmatrix} 0.901 & 0.406 & 0.152 & -0.093 \\ -0.155 & -0.026 & 0.988 & 0.012 \\ 0.406 & -0.914 & 0.039 & 0.390 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{pmatrix} \tag{A.2} $$

$$ {}^{Cube}T_{RightTCP} = \begin{pmatrix} 0.925 & 0.350 & -0.146 & -0.070 \\ 0.137 & 0.054 & 0.989 & -0.012 \\ 0.355 & -0.935 & 0.002 & 0.390 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{pmatrix} \tag{A.3} $$