Technik
Informatik & Medien
Hochschule Ulm
University of
Applied Sciences

Hochschule Ulm

Fakultät Informatik

Studiengang
Technische Informatik

Bachelorarbeit

Erkennen und Greifen von Alltagsgegenständen mittels Katana Manipulatorarm: Bahnplanung und -ausführung

Vorgelegt von

Jonas Brich

25. August 2009

1. Gutachter: Prof. Dr. Christian Schlegel
2. Gutachter: Prof. Dr. Karin Lunde

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht und mit genauer Quellenangabe dargelegt habe.

Ort, Datum	Unterschrift

Zusammenfassung

Diese Bachelorarbeit stellt eine einfache und robuste Methode vor, wie einfache Alltagsgegenstände mit einem Katana Roboterarm manipuliert werden können. Die Manipulation von Gegenständen geschieht unter der Berücksichtigung von Hindernissen. Dabei wird die kollisionsfreie Berechnung des Pfades für den Katana-Arm von einem bereits bestehenden Framework übernommen. Die Erfassung der Umwelt wird mittels eines Lasers bewerkstelligt. Die Funktionsweise und die Ergebnisse der Umgebungserfassung werden in einer parallel entwickelten Bachelorarbeit beschrieben. Diese Bachelorarbeit beschäftigt sich speziell mit der Bahnplanung und –ausführung. Zusammengeführt wird ein System vorgestellt, welches in der Lage ist, eine dynamische Umgebung mit deren Objekten zu erfassen und diese ohne Kollision zu manipulieren.

Danksagung

Bedanken möchte ich mich bei Herrn Prof. Dr. C. Schlegel für das interessante und spannende Thema und für die hilfreichen Kommentare und Diskussionen. Zugleich möchte ich mich auch bei Herrn M. Sc. S. Hochdorfer bedanken, er stand mir bei meiner Arbeit mit Rat und Tat zur Seite.

Außerdem möchte ich mich bei meinem Kommilitonen Manuel Wopfner für drei Jahre gute Zusammenarbeit im Studium wie auch in der Bachelorarbeit und für die vielen anregenden Diskussionen bedanken.

Zusätzlich möchte ich noch Rosen Diankov dem Entwickler von OpenRAVE danken. Er hat mir bei vielen Problemen geholfen und ohne ihn hätte ich nicht alles in der vorgegebenen Zeit geschafft.

Meinen Eltern danke ich ebenfalls ganz besonders, sie haben mich in allen Belangen rund um das Studium stets unterstützt. Zu guter Letzt möchte ich meiner Freundin Julia danken, nicht nur für das Korrektur lesen, sondern auch für die seelische und moralische Unterstützung über die ganze Bachelorarbeit hinweg.

Inhaltsverzeichnis

Kapitel 1 Einleitung

1.1. Problemstellung

Der Bereich der Servicerobotik gewinnt in der heutigen Zeit immer mehr an Bedeutung. Deswegen werden in diesem Bereich viele Projekte durchgeführt. So betreibt *Willow Garage* [WIL09] ein sehr fortgeschrittenes Projekt mit einer autonomen Roboterplattform, welche Türen öffnen kann und in der Lage ist sich selber bei niedrigem Batteriestand in eine Steckdose einzustecken. Obwohl der schon beeindruckenden Leistung des Roboters, ist der Bereich der Servicerobotik noch ein großes Forschungsgebiet. Vor allem im Bereich der *Mobilen Manipulation*, welche sich mit der Manipulation von Objekten beschäftigt, ist noch sehr viel Forschungsarbeit zu leisten.

Einer der größten Unterschiede zu herkömmlichen Industrierobotern ist der, dass sich die Umwelt bei Servicerobotern ständig ändert. Dieses Problem macht das Agieren in der Umwelt für diese Roboter erheblich schwieriger. Desweitern befinden sich im Arbeitsbereich des Roboters Menschen, die nicht behindert oder gar verletzt werden dürfen.

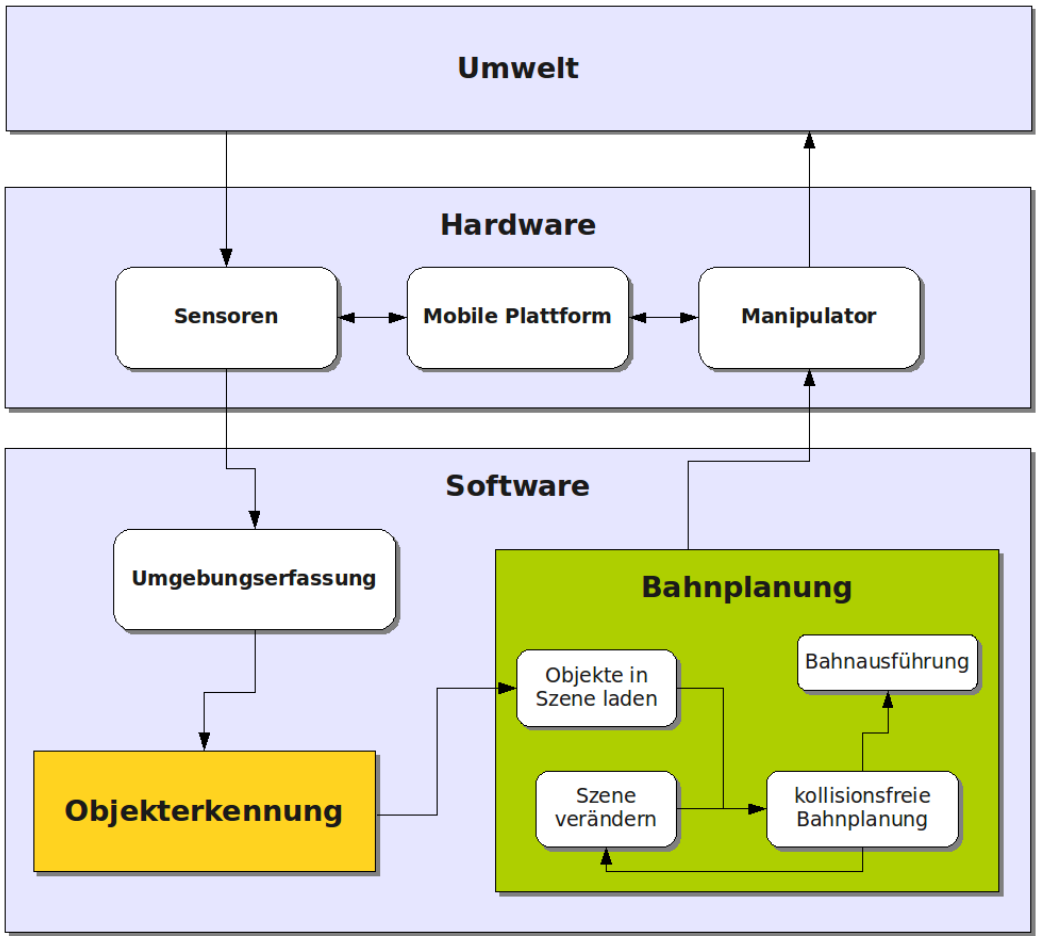

Abbildung 1.1.1: Komponenten einer Mobilen Manipulation (In Zusammenarbeit mit [WOP09])

Die *Mobile Manipulation* setzt sich aus mehreren Teilbereichen zusammen, welche in Abbildung 1.1.1 zu sehen sind. Die drei großen Bereiche der *Mobilen Manipulation* sind *Umwelt*, *Hardware* und *Software*. Die *Umwelt* repräsentiert das, was ein Mensch sieht und wo er agiert. Die *Hardware* setzt sich aus dem fahrbaren Untersatz des Roboters, dem Greifarm (Manipulator) und Kamera oder Laserscanner zusammen, wobei die beiden letzteren auch zusammen verwendet werden können. Durch diese Komponenten wird die Umwelt entweder verändert (Manipulator) oder erfasst (Sensoren). Zum Verändern der Umwelt muss die *Software* auf dem Roboter entsprechendes erfüllen: Die Umwelt muss durch die Sensoren erkannt werden, danach müssen die Objekte klassifiziert werden. Mit den erkannten Objekten wird durch gegebene Pfadplanungsalgorithmen eine kollisionsfreie Bahn zu den zu manipulierenden Objekten errechnet. Diese Bahn wird dann durch den Manipulator ausgeführt.

Das Problem der sich ändernden Umwelt kann beliebig schwer werden. Eine von den leichtesten Situationen ist diese, in der die Umwelt nach dem Erfassen gleichbleibt. Das Gegenstück ist die Situation, in der sich die Umwelt kontinuierlich verändert und der Roboter die Umgebung immer wieder erfassen und dementsprechend schnell agieren muss.

1.2. Motivation

Ziel dieser Bachelorarbeit ist es, ein System zur *Bahnplanung und –ausführung* zu entwickeln. Dieses soll in der Lage sein mit einfachen Alltagsgegenständen wie Bechern, Flaschen und Schachteln umzugehen. Diese Objekte werden von einer parallel entwickelten Objekterkennungssoftware [WOP09] erkannt und klassifiziert. Die erkannten Objekte werden als einfachste Körper angenähert, etwa als Quadern und Zylindern. Mit diesen Objekten und vorhanden Pfadplanungsalgorithmen soll eine einfache und dennoch robuste Methode entwickelt werden, mit der Alltagsgegenstände kollisionsfrei manipuliert werden können.

Kapitel 2 Stand der Technik

2.1. Frameworks

Ein Framework ist eine Software, welche einen Rahmen bereitstellt, in dem sich der Anwender bewegen kann. Das Framework selber ist also nur ein Hilfsgerüst um dem Anwender verschiedene Probleme abzunehmen. Frameworks stellen nach außen hin eine API (Application Programming Interface) bereit. Diese API stellt abstrakte Funktionen bereit, die einfach zu verwenden sind. Die komplizierte Implementierung dahinter wird in den meisten Frameworks verborgen. Außerdem können die Programme, die in den Frameworks ablaufen, sehr abstrakt gehalten werden. Dieses führt zu einer leichteren Portierbarkeit und lässt auch weniger Programmierfehler zu.

Frameworks haben von Anfang an ein standardmäßiges Verhalten, dieses kann durch den Benutzer angepasst werden. Dabei liegt der Kontrollfluss immer beim Framework. Das Framework legt den Ablauf des Programmes fest, der Benutzer kann nur das Programm, das im Framework läuft, ändern. Frameworks sind sehr starr, Änderungen am Framework selber können normalerweise nicht vorgenommen werden. Es ist nur möglich Frameworks an vordefinierten Stellen zu erweitern. Dennoch haben Frameworks auch positive Aspekte. Programme, die von Grund auf gleich sind, können mit Frameworks sehr schnell umgesetzt werden. Auch bieten Frameworks oft eine Art von Visualisierung, welche sehr hilfreich sein kann.

Oft ist es notwendig beziehungsweise gern gesehen, die Funktionen, die ein Framework bietet, fest in die eigene Software zu integrieren. Somit würde die entwickelte Software nicht auf das Framework aufsetzen und dieses benutzen, sondern es zu einem festen Bestandteil machen. Dies bringt aber einige Probleme mit sich. Frameworks haben sehr viele Abhängigkeiten in sich selber. Somit ist es sehr schwer nur einen kleinen Teil des Ganzen zu verwenden. Außerdem wird die vorhin angesprochene Visualisierung in einer eigenen Software meist nicht benötigt. Teils ist es möglich diese abzuschalten, aber vorhanden ist sie dennoch.

2.1.1. OpenRAVE

OpenRAVE [OPR08,OPR09] (Open Robotics and Animation Virtual Environment) ist ein Open-Source Framework, dessen Ziel auf selbstständig agierenden Robotern liegt, welche sich in der realen Welt bewegen. OpenRAVE ist eine modular aufgebaute Anwendung, welche sich durch viele schon vorintegrierte Module auszeichnet. Die Plug-Ins reichen von einer 3D-Grafikumgebung zur Visualisierung über mehrere Pfadplanungsalgorithmen bis hin zu

Skript- und Steuerungsmodulen. OpenRAVE kann nicht nur fahrbare Roboter handhaben, sondern kann auch mit Roboterarmen und sogar mit humanoiden Robotern umgehen.

Für Roboterarme ist es in OpenRAVE möglich, die Inverse Kinematik (siehe hierzu Kapitel 3 Absatz 3.1.3) zu berechnen. Ebenfalls kann eine Pfadplanung mit Kollisionserkennung ausgeführt werden. Zusätzlich dazu ist es auch möglich, das Greifen von Gegenständen zu berechnen. Diese ganzen Aufgaben können von außen über verschiedene Skriptsprachen wie Octave, Matlab oder sogar Python gesteuert werden. Es ist aber auch möglich, OpenRAVE über eine C++ API anzusteuern.

Derzeit gibt es für OpenRAVE kein offizielles Release. Die Software kann nur direkt über das Repository ausgecheckt werden. OpenRAVE steht unter der LGPL[1], ist somit frei verfügbar und kann auch mit proprietärer Software eingesetzt werden. OpenRAVE ist auf Linux und Windows lauffähig.

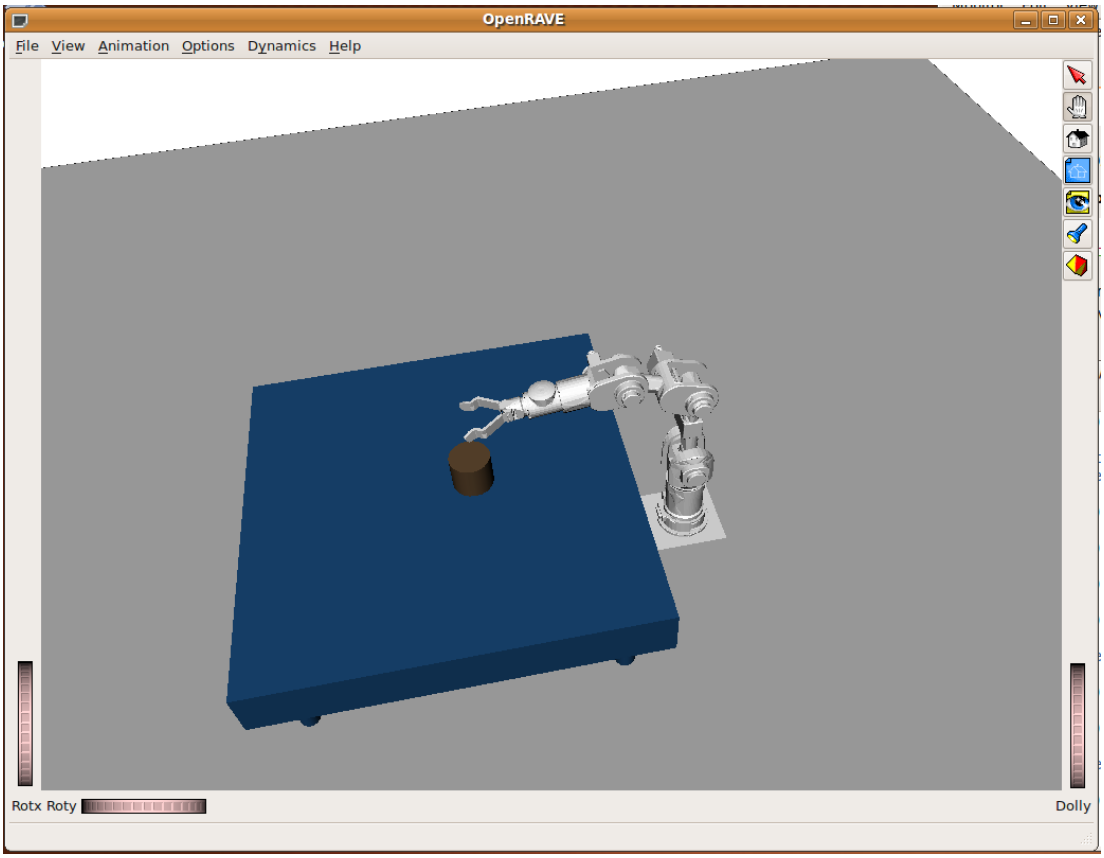

Abbildung 2.1.1: OpenRAVE 3D Visualisierung mit Katana Roboterarm

[1] Lesser General Public License

OpenRAVE ist ein sehr vielversprechender Kandidat für diese Bachelorarbeit, da es bereits mit Pfadplanungsalgorithmen und Kollisionserkennung ausgestattet ist. Außerdem beherrscht es den Umgang mit Roboterarmen. Zudem ist es ein sehr neues, aktives Projekt, welches zur Zeit sehr an Beliebtheit gewinnt. Dennoch muss die Schwierigkeit von einer Einbindung in eine bestehende Software untersucht werden. Zudem muss die Komplexität und Verfügbarkeit der ganzen Komponenten untersucht werden.

2.1.2. Webots

Webots [WEB09] ist ein 3D-Simulator für mobile Roboter. Es können fahrende, laufende und fliegende Roboter simuliert werden. Diese können sich in einer komplexen Welt, die selber erstellt werden kann, bewegen. Webots bietet eine Programmierschnittstelle, die es möglich macht, neue Roboter zu programmieren. Unterstützte Sprachen sind C, C++ oder Java. Es bringt aber auch schon einige Roboter mit sich, zum Beispiel Nao[1], Aibo ERS-7[2] und auch den Katana-Arm. Die simulierten Bewegungs- und Sensordaten können aus Webots ausgelesen und in die realen Roboter eingespeist werden. Zum Erzeugen der Umgebung wird die Beschreibungssprache VRML[3] benutzt.

Webots ist eine kommerzielle Software, die es in zwei Versionen gibt: Webots PRO, die für Entwickler gedacht ist, und eine Webots EDU, die für Lehrzwecke genutzt werden kann. Beide Versionen laufen auf Windows, MacOS und Linux.

Die PRO Version beinhaltet gegenüber der EDU Version folgende Inhalte mehr: Es können eigene physikalische Gesetze definiert werden, mit diesen lässt sich das Verhalten Unterwasser oder in fast Schwerelosigkeit simulieren. Desweitern kann die Grafische Oberfläche abgeschaltet werden, dies lässt ein schnelleres Simulieren zu. Die letzte Erweiterung gegenüber der EDU Version ist der sogenannte Supervisor-Mode. In diesem können übergeordnete Funktionen definiert werden, die zum Beispiel bestimmte Daten automatisch aufzeichnen oder in regelmäßigen Abständen neue Gegenstände hinzufügen. Diese zusätzlichen Funktionen können sehr hilfreich bei der Entwicklung von Programmen oder Algorithmen für Roboter sein. Diese beiden Versionen sind derzeit in der Version 6 erhältlich.

[1] Nao ist ein humanoider Spielzeug Roboter. Nähere Informationen unter http://www.aldebaran-robotics.com/
[2] Aibo ERS-7 ist ein hundeähnlicher Roboter. Nähere Informationen unter http://support.sony-europe.com/aibo/
[3] VRML ist eine Beschreibungssprache für 3D-Szenen. Nähere Information unter http://de.wikipedia.org/wiki/Virtual_Reality_Modeling_Language

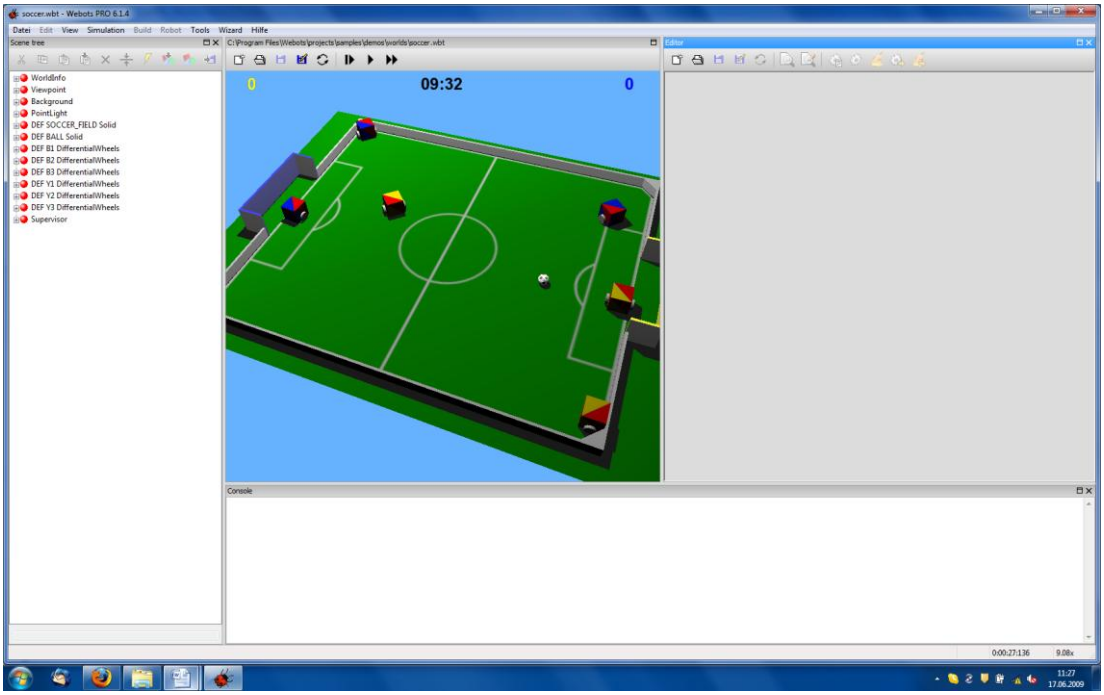

Abbildung 2.1.2: Webots Oberfläche

Webots scheint ein sehr guter Kandidat zu sein, da es schon einen Katana-Arm unterstützt und eine Programmierschnittstelle bietet. Zudem beherrscht es auch Kollisionserkennung. Fraglich ist dagegen ob Webots kollisionsfreie Pfadberechnung durchführen kann. Außerdem ist auch eine Anbindung an eine bestehende Software zu klären.

2.1.3. Microsoft Robotics Studio

Das Microsoft Robotics Studio [MSR09] ist ein auf Windows basierendes Framework, mit welchem es möglich ist, Roboter zu kontrollieren und zu simulieren. Es können alle gängigen .NET-Sprachen zur Roboterprogrammierung verwendet werden. Es ist auch möglich, die Microsoft Visual Programming Language [VPL09] zu verwenden, um vorhandene Sensoren oder Aktuatoren der Roboter anzusteuern beziehungsweise auszulesen, wie in Abbildung 2.1.3 zu sehen ist. Im Vordergrund des Frameworks steht der einfache Zugriff auf die Sensoren und Aktuatoren des Roboters.

Microsoft Robotics Studio ist in mehreren Versionen erhältlich: Standard Edition, Academic Edition und Express Edition. Die derzeitige Version ist 2008 R2.

KUKA [KUK09], ein Spezialist bei der Herstellung und Entwicklung von Industrierobotern, bietet Übungen mit Anleitung für den Umgang mit einem ihrer Roboterarme an. Diese Übun-

gen laufen mit der Vorgängerversion vom Microsoft Robotics Studio. Anhand dieses Beispiels ist zu sehen, dass ein grundsätzlicher Umgang mit Roboterarmen möglich ist.

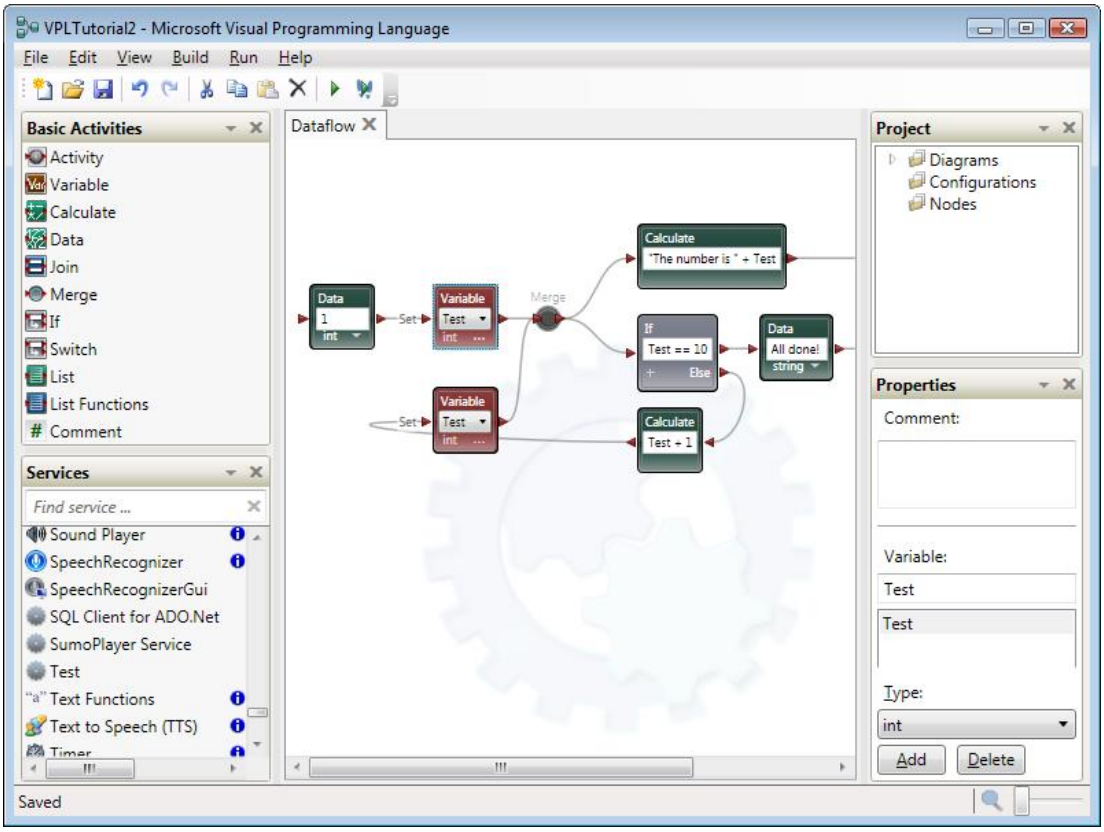

Abbildung 2.1.3: Programmierung mit Microsoft Visual Programming Language im Microsoft Robotics Studio (Quelle [VPL09])

Das Microsoft Robotics Studio ist ein vielversprechender Kandidat, da das Framework sehr viele Möglichkeiten für den Umgang mit Robotern bietet. Durch KUKA wird gezeigt, dass der Umgang mit Roboterarmen möglich ist. Auch die Kollisionserkennung ist vorhanden. Dennoch muss beachtet werden, dass es nicht möglich ist, Pfadplanung im Microsoft Robotics Studio auszuführen; dies muss über ein anderes Tool realisiert werden. Auch ist die Realisierung einer Ansteuerung über eine externe Software noch zu betrachten.

2.1.4. Carmen

Carmen [CAR09] (Carnegie Mellon Robot Navigation Toolkit) ist eine Sammlung von Open-Source Anwendungen, die für den Einsatz im Robotik-Umfeld gedacht sind. Verschiedene Anwendungen sind zum Beispiel: Kollisionsfreies Fahren, Lokalisierung, Pfadplanung und Kartierung. Für eine reibungslose und einfache Benutzung wird eine Reihe von verschiedenen

Robotern unterstützt, wie zum Beispiel der ActivMedia Pioneer[1] und der iRobot ATRV[2]. Es werden außerdem auch verschiedene Laser und Sensoren unterstützt. Die Visualisierung der Roboter und ihrer Sensoren geschieht in einer sehr einfachen 2D-Darstellung.

Carmen ist seit Oktober 2008 in der Version 0.7.4 Beta erhältlich und läuft nur unter Linux. Weiterhin steht Carmen unter der GPL[3] und ist somit frei verfügbar.

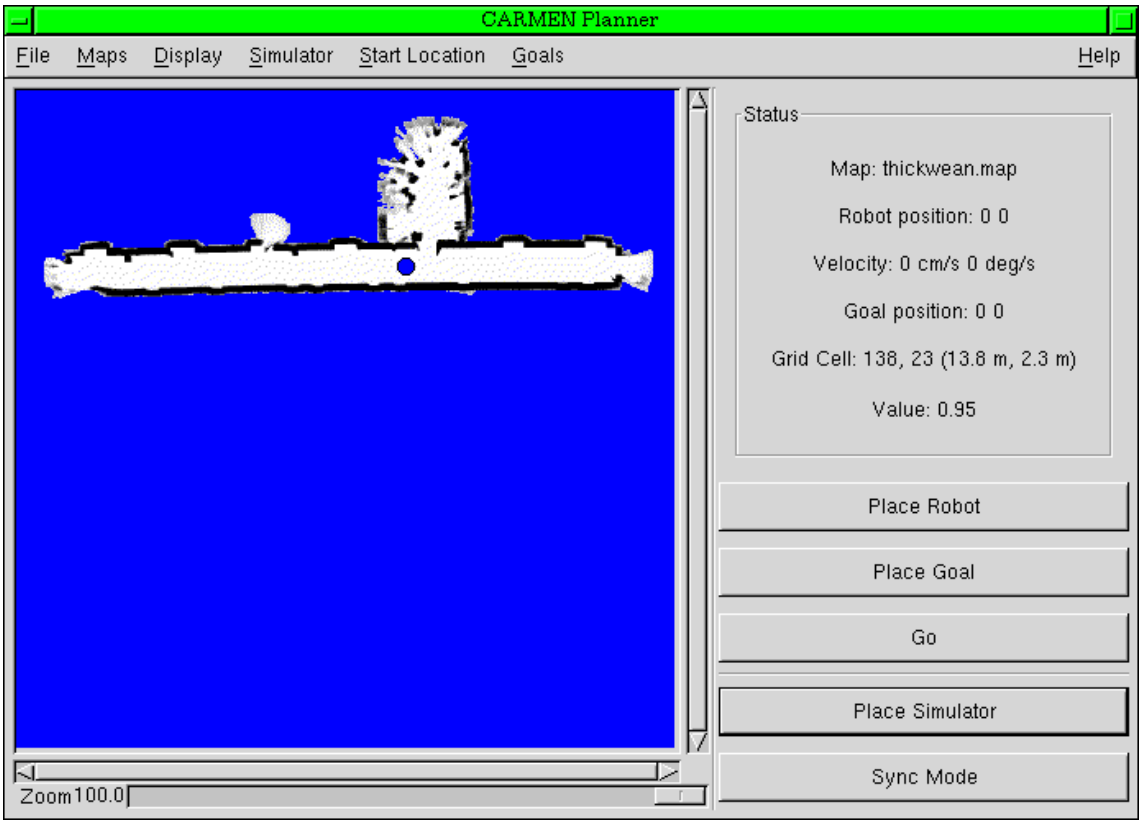

Abbildung 2.1.4: Carmen 2D Oberfläche (Quelle [CAR09])

Der Einsatz von Carmen im Rahmen dieser Bachelorarbeit stellt sich als unwahrscheinlich dar. Das Einsatzgebiet von Carmen ist auf Roboter beschränkt, die sich in einer Ebene bewegen. Da der Schwerpunkt dieser Arbeit auf Manipulation von Gegenständen mit einem Roboterarm in 3D-Umgebung liegt, gestaltet sich die Benutzung von Carmen als äußerst schwierig.

2.1.5. KineoWorks

KineoWorks [KIN09] ist eine kommerzielle Anwendung, die ihren Schwerpunkt bei der Pfadplanung von Objekten hat. Die Visualisierung erfolgt in 3D. KineoWorks beherrscht Kol-

[1] Roboter für das Forschungsumfeld. Näheres unter http://www.activrobots.com/
[2] Haushaltsroboter und Taktikroboter. Näheres unter http://www.irobot.com/
[3] General Public License

lisionsvermeidung und benutzt die Kinematik des zu simulierenden Objekts, um den optimalen Weg aus vielen vorhandenen Pfadplanungsalgorithmen auszuwählen. KineoWorks ist ein komplettes Framework, welches seine Schwerpunkte auf die Integration in einen bestehenden Firmenprozess legt. KineoWorks ist verfügbar für Linux, Windows und MacOS.

KineoWorks beherrscht den Umgang mit Roboterarmen und kann kollisionsfreie Wege berechnen. Dies macht KineoWorks zu einem guten Kandidaten. Dennoch ist zu beachten, wie die Anbindung an eine externe Software realisiert werden kann. Auch scheint KineoWorks mehr in Richtung Systemprozess zu gehen und für Mobile Roboter etwas zu umfangreich zu sein.

2.1.6. CLARAty

CLARAty [CLA08] (Coupled Layer Architecture for Robotic Autonomy) ist ein vom NASA Ames Research Center, dem Jet Propulsion Laboratory, dem Carnegie Mellon und der University of Minnesota entwickeltes Framework, welches eine flexible und wiederverwendbare Anwendung für die Benutzung von Robotern darstellt. CLARAty besteht aus vielen verschiedenen Schichten wie Navigation, Pfadplanung, Kommunikation, Manipulation und vieles mehr, die eigenständig funktionieren und wiederverwendet werden können.

CLARAty ist normalweise nicht für die Öffentlichkeit zugänglich, aber es gibt seit längerem ein öffentliches Release. Dieses hat im Gegensatz zu dem nicht-öffentlichen sehr wenig Funktionalität. Die öffentliche Version enthält Unterstützung für verschiedene Laser und Motoren, dagegen sind nur sehr wenige Algorithmen zur Pfadplanung enthalten. Die öffentliche Version ist seit Juni 2007 in der Version 0.1 Beta verfügbar und läuft unter Linux. Diese Version steht unter der CLARAty TSPA License[1].

Unter der Berücksichtigung, dass es nur möglich ist, mit dem öffentlichen Release zu arbeiten, ist der Einsatz von CLARAty eher fragwürdig. Da die öffentliche Variante nur sehr wenig Funktionalität enthält und die Anbindung an eine bestehende Software ebenfalls fragwürdig ist, ist vom Einsatz dieses Frameworks eher abzuraten.

[1] Nähere Information unter http://claraty.jpl.nasa.gov/man/software/license/public_src/index.php

2.2. Funktionsbibliotheken

Eine Funktionsbibliothek ist eine Sammlung von Funktionen und Klassen, welche über eine API angesprochen werden können. Ein Vorteil von Bibliotheken ist der, dass interne Änderungen sich nicht auf den Aufruf der Funktionen niederschlagen. Bei kleinen Änderungen bleibt die Schnittstelle nach außen hin bestehen. Nur bei sehr großen Änderungen kommen eventuell neue Funktionen hinzu beziehungsweise nicht mehr benötigte werden fallen gelassen. Zusätzlich sind Bibliotheken von Grund auf dafür vorgesehen von anderen Programmen genutzt zu werden. Deswegen ist ein Einbinden der Bibliothek sehr einfach. Vor allem können mehrere kleine Bibliotheken genutzt werden, um verschiedene Teilbereiche eines Problems zu lösen.

Dennoch haben Bibliotheken auch Nachteile. Die Freiheit der API mit ihren willkürlich benutzbaren Funktionen und Klassen überlässt den Ablauf der verschiedenen Aufrufe, im Gegensatz zum Framework, ganz dem Programmierer. Bei großen Programmen mit wiederkehrenden Aufgaben und mehreren Tasks kann schnell der Überblick verloren gehen. Auch bieten Bibliotheken keine Art der Visualisierung. Es ist höchstens die Ausgabe von diversen Ergebnissen auf die Konsole möglich.

2.2.1. OOPSMP

OOPSMP [OOP09] (An Object Oriented Programming System for Motion Planning) ist eine Bibliothek für Pfadplanung, welche einfach zu erweitern, robust und effizient ist. Diese Bibliothek dient als Referenzbibliothek für viele Anwendungen. OOPSMP beinhaltet viele effiziente und sehr weit entwickelte Algorithmen für die Pfadplanung. Beispiele für diese Algorithmen sind: Probabilistic Road Map (PRM), Rapidly-Exploring Random Tree (RRT) oder Bi-Directional Tree Planner. Einige dieser Algorithmen werden im Kapitel Grundlagen ausführlicher erklärt.

OOPSMP besteht schon seit 1997 und wird vom Kavraki Lab weiterentwickelt. Seit kurzem gibt es auch ein Plug-In, mit dem über Google SketchUp [SKE09] die Szene erstellt werden kann.

Personen, die nicht an der Rice University studieren beziehungsweise arbeiten, müssen über eine E-Mail den Quellcode erfragen. Derzeitiger Release ist die Version 1.1.1 vom Februar 2009. Die Bibliothek kann unter Linux, Windows und MacOS verwendet werden.

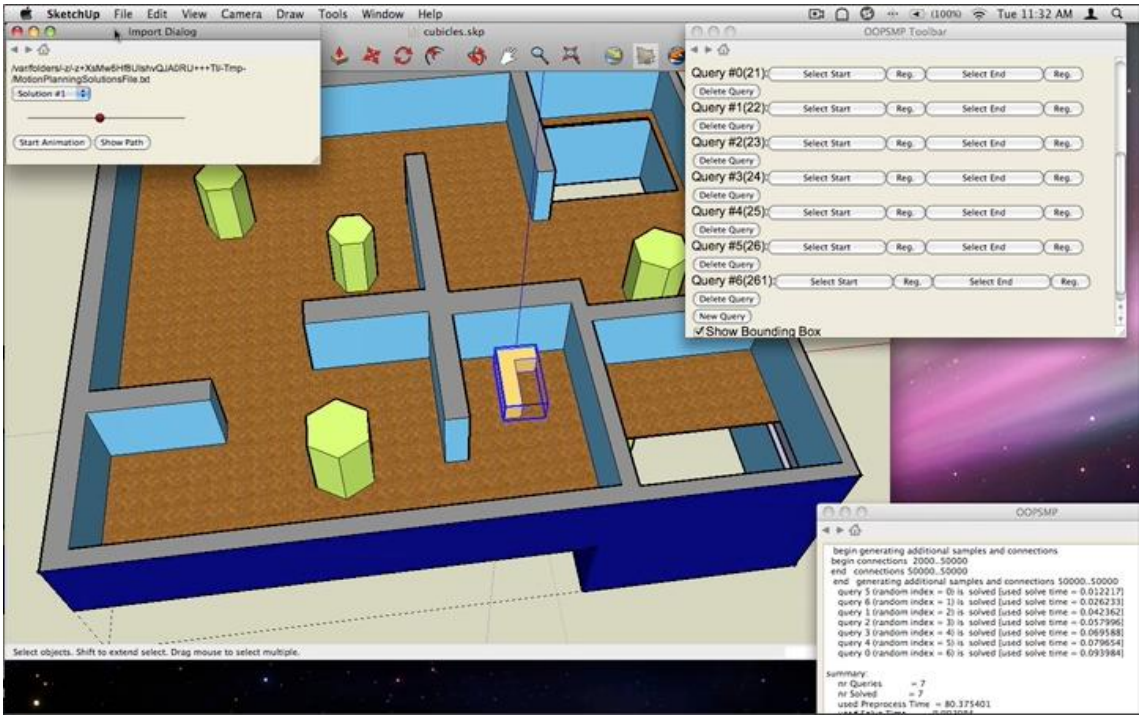

Abbildung 2.2.1: Kavraki mit Google SketchUp (Quelle [OOP09])

Die OOPSMP Bibliothek ist ein vielversprechender Kandidat, weil sie sehr gute Pfadplanungsalgorithmen enthält, mit denen eine kollisionsfreie Bahn berechnet werden kann. Zudem ist ein Einbinden der Bibliothek in eine bestehende Software sehr leicht zu bewerkstelligen. Dennoch ist zu beachten, dass der Umgang mit Roboterarmen nicht gegeben ist. Dieses muss über ein anderes Tool realisiert werden, oder aufbauend auf der Bibliothek selbst bewerkstelligt werden.

2.2.2. MPK

MPK [MPK06] (Motion Planning Kit) ist eine C++ Library für Bewegungs- und Pfadplanung von einzelnen oder mehreren Robotern. Die Bibliothek beinhaltet eine sehr effiziente und exakte Kollisionserkennung. MPK beherrscht den Umgang mit Roboterarmen, vor allem im Bereich Industrierobotik. Eines der neuesten Features in MPK ist das effiziente Planen von Mehr-Ziel-Pfadplanung. Dieses Feature entspricht dem klassischen Traveling-Salesman-Problem[1].

MPK ist für Linux verfügbar und kann auf Anfrage heruntergeladen werden. Die Bibliothek ist in Version 1.0 von Juni 2004 verfügbar.

[1] Travling Salesman Problem ist ein kombinatorisches Optimierungsproblem der theoretischen Informatik. Ausführlich auf http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden nachzulesen.

MPK beherrscht den Umgang mit Roboterarmen und kann kollisionsfreie Pfadplanung für diese durchführen. Dieses macht MPK sehr vielversprechend. Dennoch muss beachtet werden, dass dieses Projekt seit mehreren Jahren nicht aktualisiert wurde, was zu erheblichen Problemen führen könnte.

Kapitel 3 Grundlagen

3.1. Manipulator

3.1.1. Freiheitsgrad

Ein Freiheitsgrad in der Robotik beschreibt die Anzahl und Art der möglichen Bewegungen, die ein Gelenk ausführen kann. Ein Körper im 3D-Raum wird eindeutig durch sechs Freiheitsgrade bestimmt: Drei Koordinatenpunkte und drei dazugehörige Winkel relativ zu den Koordinatenachsen. Manipulatoren mit weniger als sechs Freiheitsgraden sind immer eingeschränkt und können nicht jede Position anfahren. Manipulatoren mit sechs Freiheitsgraden sind in der Lage, jede Position im 3D-Raum anzufahren. Zusätzliche Freiheitsgrade machen aber durchaus Sinn. Die Redundanz wirkt sich hilfreich bei der Kollisionsvermeidung aus. Durch diese zusätzlichen Freiheitsgrade kann eine Position mit mehreren verschiedenen Gelenkwinkeln angefahren werden. Falls somit eine Lösung durch das Hindernis blockiert wird, gibt es weitere Lösungen, die eventuell zum Erfolg führen.

Die Anzahl der Freiheitsgrade in einem zusammengesetzten System berechnet sich aus der Summe der Freiheitsgrade der einzelnen Gelenke.

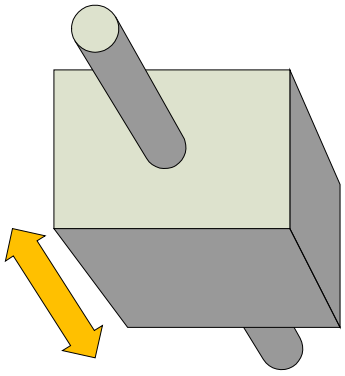

Abbildung 3.1.1: Ein Freiheitsgrad

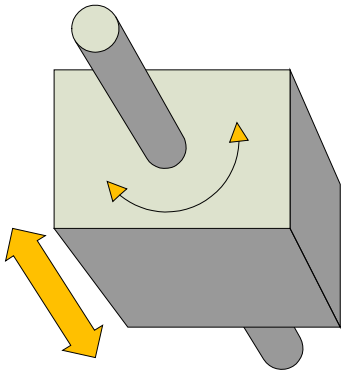

Abbildung 3.1.2: Zwei Freiheitsgrade

Die Abbildung 3.1.1 zeigt einen Freiheitsgrad. Der Körper kann entlang des Zylinders nach oben und unten gleiten. Die Abbildung 3.1.2 zeigt zwei Freiheitsgrade. Hier kann der Körper sich nicht nur nach oben und unten bewegen, sondern kann sich auch um den Zylinder drehen. Dies entspricht dem zweiten Freiheitsgrad.

3.1.2. Vorwärtskinematik

Die Vorwärtskinematik, oder auch manchmal direkte Kinematik, wird hauptsächlich in der Robotik angewendet. Mit dieser ist es möglich, aus den Gelenkwinkeln des Roboters die Position des Endeffektors (z.B. des Greifer) zu bestimmen. Die direkte Kinematik ist das Gegen-

stück zur inversen Kinematik, wie auch in der Abbildung 3.1.3 zu sehen ist. Die direkte Kinematik kann durch einfache Matrizenmultiplikationen berechnet werden. Somit ist es sehr einfach möglich, eine Position aus gegebenen Gelenkwinkeln zu bestimmen. In der Realität ist es aber sehr oft nötig, aus einer gegebenen Position die Gelenkwinkel des Roboters zu bestimmen, also genau der umgekehrte Fall. Dieses kann mit der inversen Kinematik bewerkstelligt werden.

3.1.3. Inverse Kinematik

Die inverse Kinematik ist das Gegenstück zur direkten Kinematik. Mit ihr ist es möglich, aus einer gegebenen Position die Gelenkwinkel des Roboters zu berechnen. Bei der Berechnung wird der Endeffektor in die gewünschte Position gebracht. Die übrigen Glieder des Armes müssen dann entsprechend ihren Freiheitsgraden die passende Lage einnehmen.

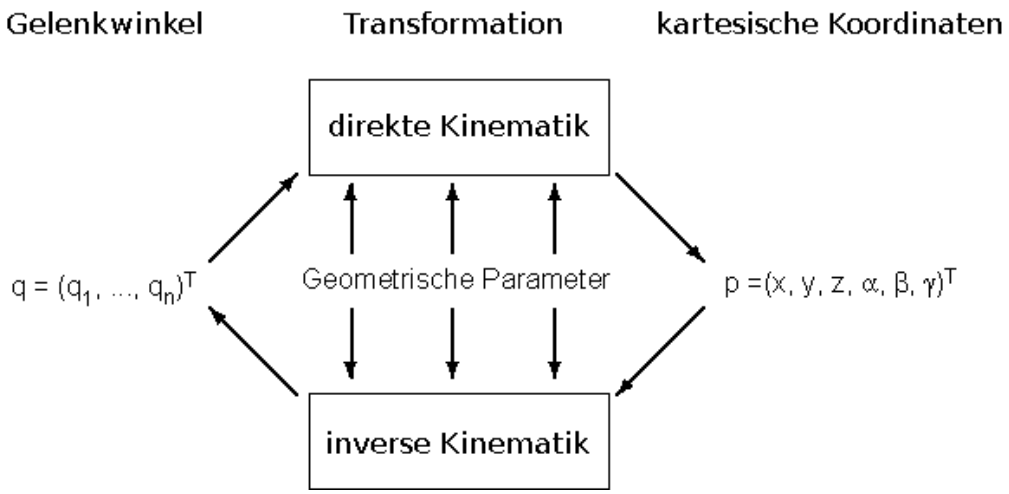

Abbildung 3.1.3: Transformation von der direkten Kinematik in die inverse Kinematik und umgekehrt [WIK05]

Die Schwierigkeit bei der Berechnung der inversen Kinematik ist die, dass sie keine eindeutigen Ergebnisse liefert. Dieses Problem tritt vor allem bei Robotern mit mehr als sechs Freiheitsgraden auf. Diese Roboter besitzen zu einer eindeutigen Position mehrere Kombinationen von Gelenkwinkeln. Dies erschwert zwar die Lösung der inversen Kinematik, wird aber dennoch benötigt, sobald Hindernisse eine Rolle spielen. Die Redundanz ermöglicht es diesen Robotern, Objekte zu erreichen obwohl Hindernisse den direkten Weg blockieren. Roboter mit weniger Freiheitsgraden müssten sich entweder selber komplett an eine andere Position bewegen oder sie können das Objekt schlichtweg nicht erreichen. Zusätzlich dazu kann es vorkommen, dass unzulässige Konfigurationen entstehen. Diese sind mathematisch korrekt,

können aber mit dem Roboter nicht angefahren werden, da die Gelenkwinkel des Roboters eingeschränkt sind.

Diese Probleme machen eine allgemeine Lösung der inversen Kinematik sehr schwer. Es gibt verschiedene Verfahren, die auf bestimmte Fälle angewandt werden können. Die algebraische Methode arbeitet mit einer homogenen Matrix, die die Position und Orientierung des Endeffektors beschreibt. Durch sukzessives Invertieren dieser Matrix wird versucht, die einzelnen Gelenkwinkel zu berechnen. Die geometrische Methode versucht mit Hilfe von Sinus- und Kosinussatz und der vorhandenen Geometrie des Roboters, die Gelengwinkel zu berechnen. Die Numerische Methode versucht dagegen, iterativ eine Lösung der Gelenkwinkel zu finden. Problematisch sind bei dieser Methode jedoch lokale Minima und die Bestimmung eines geeigneten Startwertes.

3.1.4. Koordinatensysteme und Gelenke des Katana Manipulators

Der Katana-Arm besitzt zwei Koordinatensysteme. Das erste Koordinatensystem liegt im Mittelpunkt des zweiten Gelenkes. Die Z-Achse zeigt dabei vom Boden weg. Die X-Achse zeigt in Richtung des Arbeitsbereiches. Die Y-Achse zeigt dabei in die Ebene der Abbildung hinein. Dieses wird in der Abbildung 3.1.4 verdeutlicht dargestellt. Dieses Koordinatensystem ist das Basiskoordinatensystem.

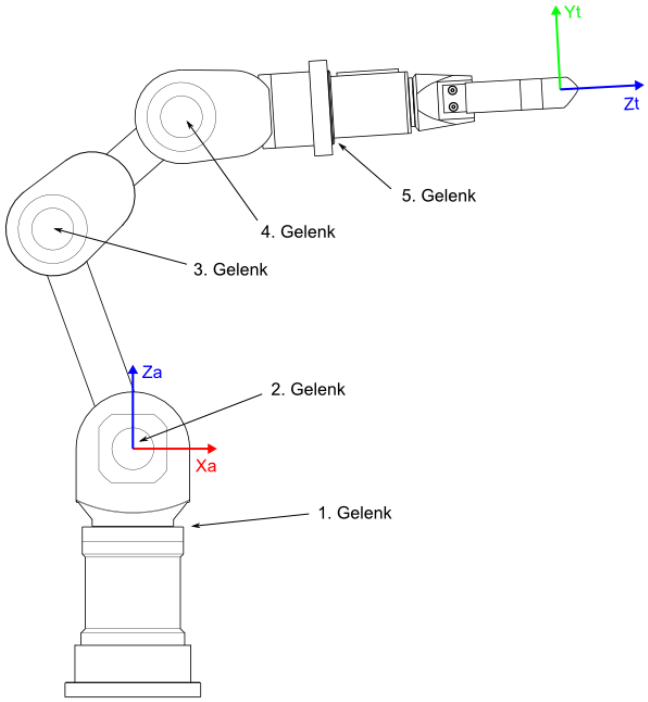

Abbildung 3.1.4: Katana mit seinen Koordinatensystemen und Gelenken (Quelle [WOP09])

Das zweite Koordinatensystem liegt im Endeffektor des Katana. In diesem System zeigt die Z-Achse vom Endeffektor weg. Die Y-Achse zeigt vom Arbeitsbereich weg und die X-Achse zeigt in die Ebene der Abbildung hinein. Dieses Koordinatensystem ist das Endeffektorkoordinatensystem. Dieses Koordinatensystem ist im Endeffekt das gleiche Koordinatensystem wie das Basiskoordinatensystem. Der einzige Unterschied besteht darin, dass das Basiskoordinatensystem immer gleich bleibt und das Endeffektorkoordinatensystem sich mit der Bewegung des Endeffektors verändert. Genauere Beschreibung zu den einzelnen Koordinatensystemen und ihre Umrechnungen können in [WOP09] nachgelesen werden.

3.2. Pfadplanungsalgorithmen

3.2.1. BasicRRT

Ein RRT (Rapidly-Exploring Random Tree) bezeichnet eine Kombination aus Datenstruktur und Algorithmus, die zur effizienten Pfadplanung eingesetzt wird. Dem Algorithmus liegt eine Baumstruktur zugrunde. Diese wird mit jeder Iteration zufällig erweitert. Der Baum wird von der Startposition aus zufällig in jede Richtung expandiert. Es wird stetig versucht, den Baum zu erweitern, solange der Weg frei ist. Trifft eine Erweiterung auf ein Hindernis, so wird in eine andere Richtung erweitert. Somit wächst der RRT um Hindernisse herum und erschließt die ganze Umgebung gleichmäßig.

Dieses gleichmäßige Erschließen ist jedoch auch ein großer Nachteil des Algorithmus. Wenn zum Beispiel das Ziel sehr weit entfernt liegt, so braucht der Algorithmus sehr viele Iterationen bis er in die Nähe des Ziels kommt.

Meist wird dieser Algorithmus nicht alleine verwendet. Stattdessen bauen viele weiterentwickelte Algorithmen auf dem RRT auf oder benutzten diesen intern.

3.2.2. rBiRRT

Der rBiRRT-Algorithmus ist eine verbesserte Version des normalen BiRRT [RRT00] (Bi-Directional Rapidly-Exploring Random Tree). Dieser funktioniert ähnlich wie der BasicRRT. Der größte Unterschied der Algorithmen ist, dass nicht nur ein Baum vom Start zum Ziel erzeugt wird, sondern auch ein Baum vom Ziel zum Start. Zusätzlich wird versucht, die Bäume nicht zur zufällig in jede Richtung zu vergrößern; es wird ebenfalls versucht, die Bäume in die

entgegengesetzte Richtung zu vergrößern. Somit wachsen die beiden Bäume zielgerichtet zueinander. Diese beiden Maßnahmen beschleunigen das Finden einer Lösung enorm.

Der rBiRRT-Algorithmus ist für OpenRAVE optimiert und angepasst. Im Grunde genommen handelt es sich um den normalen BiRRT-Algorithmus; aber mit Verbesserungen, Fehlerbeseitigungen und Geschwindigkeitssteigerungen. Diese Art von Algorithmus gibt es nur in OpenRAVE.

3.2.3. RA*

Der RA* [RAS07] (Randomized A*) Algorithmus baut auf dem normalen A*-Algorithmus auf. Der A* benutzt eine Heuristik- und eine Kostenfunktion um den kürzesten Pfad zu bestimmen. RA* benutzt ebenfalls die Heuristik- und Kostenfunktion. Diese werden aber entweder durch vorgegebene Pfade oder durch das wiederholte Lösen eines gleichbleibenden Problems erstellt. Dieses selbstständige Lernen ermöglicht es, in kurzer Zeit optimale Lösungen zu erstellen. Zudem hat der RA*-Algorithmus gegen über der RRT-Familie den Vorteil, dass er keine Inverse Kinematik benötigt. Die Endgelenk-Konfiguration muss nicht bekannt sein um das Problem zu lösen. Desweitern versucht der RA* vielversprechende Regionen zuerst zu untersuchen, welches zu einer schnelleren Lösung führt.

3.2.4. BiSpace

Der BiSpace-Algorithmus [BIS08] (Bi-Directional Space) ist dem BiRRT-Algorithmus sehr ähnlich. Er baut ebenfalls zwei Bäume auf, einen vom Start zum Ziel und einen vom Ziel zum Start. Der große Unterschied besteht allerdings darin, dass diese zwei Bäume nicht in den gleichen Räumen liegen. BiSpace arbeitet gleichzeitig in zwei verschiedenen Räumen: dem Konfigurationsraum, in welchem die Gelenkwinkel liegen, und dem Arbeitsraum, in dem die Koordinaten liegen.

Der Startbaum wächst im Konfigurationsraum in Richtung des Zieles und legt dabei einen kollisionsfreien Weg an. Der Zielbaum wächst im Arbeitsraum in Richtung der Startposition. Dieser ist aber nur eine Art von Heuristik. Es kann nicht davon ausgegangen werden, dass ein kollisionsfreier Weg produziert wird.

Bei einem Kontakt der beiden Bäume kann der BiRRT-Algorithmus sofort eine Lösung ausgegeben, beide Bäume befinden sich hier im Konfigurationsraum. Beim BiSpace muss allerdings der Weg noch validiert werden. Dieses geschieht vom Konfigurationsraum aus. Der

Startbaum folgt dem Zielbaum und versucht dabei einen kollisionsfreien Weg anzulegen. Erst wenn der Startbaum vollständig bis zum Ziel validiert ist, ist ein Pfad gefunden.

Kapitel 4 Methode

4.1. Analyse der Anforderungen

Für die kollisionsfreie Manipulation von einfachen Alltagsgegenständen muss zuerst die Umgebung erfasst werden. Dies geschieht durch einen auf den Roboterarm angebrachten Laserscanner. Die Umgebung wird, sobald ein Laserscan gestartet wurde, als statisch angenommen. Veränderungen nach dem Laserscan werden nicht berücksichtigt. Die aus einer Punktwolke bestehende Umgebung wird in der parallel entwickelten Bachelorarbeit [WOP09] verarbeitet. Diese Verarbeitung gibt die Umwelt in einfachsten Körpern wieder, welche sogenannten Bounding Boxen entsprechen. Diese einfachen Körper müssen darauffolgend in eine Form gebracht werden, mit der bestehende Pfadplanungsalgorithmen umgehen können.

Für das Manipulieren von Gegenständen ist außerdem zu beachten, dass ein Objekt, sobald es vom Roboterarm gegriffen wird, Teil des Roboters wird und dann in die Pfadplanungsberechnung mit einbezogen werden muss. Desweitern muss eine Selbstkollision des Roboters bei der Berechnung des Pfades ausgeschlossen werden.

4.2. Potentielle Ansätze und Probleme

4.2.1. Betrachtung der Frameworks und Funktionsbibliotheken

4.2.1.1. OpenRAVE

Das OpenRAVE Framework bietet eine große Vielfalt an Features. Es wurde für den Umgang mit Roboterarmen und humanoiden Robotern konzipiert und kann zudem Pfadplanung, Selbstkollision des Roboterames und Kollision mit anderen Objekten berechnen. Die Inverse Kinematik des Roboterarms muss nicht vorhanden sein, da OpenRAVE in der Lage ist, aus den vorhandenen Gelenkwinkeln die Inverse Kinematik zu berechnen. Alleine diese Features machen OpenRAVE zu einem mächtigen Framework.

Es bietet aber auch noch weitere Features, die sich bei der Lösung des vorhandenen Problems als sehr hilfreich erweisen könnten. Der Roboter und die Umwelt werden getrennt behandelt. Dies führt dazu, dass die Umwelt dynamisch aktualisiert werden kann, ohne dabei den Roboter immer wieder mit einzubeziehen. Desweitern ist es in OpenRAVE möglich, dass Objekte, die zuvor Teil der Umgebung waren, zum Teil des Roboters werden. Dies ist sehr hilfreich

bei der Pfadplanung. Auch bietet OpenRAVE eine Visualisierung, die gerade in der Anfangsphase als sehr hilfreich erachtet werden kann.

Dennoch muss beachtet werden, dass es sich bei OpenRAVE um eine sehr junge Software handelt, von der noch kein offizielles Release existiert. Es muss mit eventuell vorhandenen Fehlern und nicht funktionierenden Funktionen gerechnet werden.

4.2.1.2. Webots

Webots bietet einen bereits integrierten Katana-Arm, der voll funktionstüchtig ist. Ein paar mitgelieferte Beispiele zeigen, wie dieser Arm benutzt werden kann. Zudem kann die Umwelt sehr leicht über eine VRML-Datei spezifiziert werden. Der Aufbau des Formats und der Schwierigkeitsgrad für die Darstellung einfachster Körper müssen noch recherchiert werden. Desweitern bietet Webots auch eine Physikengine und Kollisionserkennung.

Dennoch ist zu beachten, dass Webots keine Pfadplanung durchführen kann. Somit muss ein weiteres Tool benutzt werden oder ein Pfadplanungsalgorithmus ist selber im Supervisor-Mode zu erstellen. Dieser Algorithmus steuert den Arm beziehungsweise erteilt Befehle über den Supervisor-Mode. Um dieses Feature zu nutzen muss die PRO-Version verwendet werden, was im Gegensatz zu der EDU-Version Mehrkosten verursacht.

Die Verwendung mehrerer Tools würde das Zusammenführen in eine einheitliche Software erschweren. Im anderen Fall muss nach einer guten Implementierung eines Pfadplanungsalgorithmus gesucht werden, der verwendet werden kann.

4.2.1.3. Microsoft Robotics Studio

Das Microsoft Robotics Studio bietet von Grund auf keine Roboterarme an. Wie aber die Beispiele von KUKA zeigen, ist es durchaus möglich mit Roboterarmen im Microsoft Robotics Studio umzugehen. Ein großes Problem besteht dennoch bei der Pfadplanung. Die Inverse Kinematik muss selber berechnet werden beziehungsweiße muss vorhanden sein, da das Studio keine Möglichkeit anbietet, diese zu erstellen. Zudem muss die Pfadplanung mit einem extra Tool realisiert werden, da das Studio dieses ebenfalls nicht anbietet. Auch ist die Kollisionsvermeidung mit Objekten fraglich.

Dagegen kann die Szene im Microsoft Robotics Studio gut dargestellt werden. Einfachste Körper kann das Studio handhaben, sogar mit Masse und richtiger Physikberechnung. Dagegen scheint es ein größeres Problem zu sein, den Roboter darzustellen.

Somit ist der Einsatz des Microsoft Robotics Studio möglich, aber bei dem Einbringen des Roboters mit einem großen Aufwand verbunden. Zudem muss auch hier ein separates Tool dazu verwendet werden, die Inverse Kinematik und die Pfadplanung zu berechnen. Demnach ist der Umgang mit zwei Tools zu handhaben und dadurch ist die Schwierigkeit gegeben, diese Tools miteinander und mit der Objekterkennungssoftware in Einklang zu bringen.

4.2.1.4. Carmen

Der Einsatz des Tools Carmen ist von Anfang an fraglich. Carmen kann nur im 2D-Bereich agieren, was für einen Roboterarm absolut nicht ausreichend ist. Zudem kann es nicht mit Roboterarmen umgehen und das Einbringen der vorhandenen Umgebung in Carmen ist nicht bekannt.

Carmen kann allerdings kollisionsfreie Bahnplanung für Roboter im 2D-Bereich berechnen. Es könnte versucht werden, auf dieser Grundlage aufzubauen. Dennoch ist zu beachten, dass 3D-Algorithmen wesentlich komplexer und schwieriger sind als 2D-Algorithmen. Zudem sollte man nicht neu erfinden, was in anderen Tools schon im Überfluss verfügbar ist.

4.2.1.5. KineoWorks

KineoWorks ist eine professionelle und kommerzielle Software, die auf den Umgang mit Roboterarmen und kollisionsfreier Pfadplanung spezialisiert ist. Somit sollte es für diese Software durchaus möglich sein, mit der geforderten Problemstellung umzugehen.

Dennoch muss beachtet werden, dass KineoWorks hauptsächlich in großen Firmen eingesetzt wird. Zudem ist es auch für die Integration in einen bestehenden Geschäftsprozess gedacht. Diese Software stellt sehr hohe Ansprüche und es ist sehr fraglich, ob ein Forschungsprojekt wie dieses diese komplexe Software benötigt. Es gibt eventuell bessere Alternativen, die bei geringeren Anforderungen das gleiche.

4.2.1.6. CLARAty

CLARAty kann für dieses Projekt nur in der öffentlichen Version verwendet werden, da die nicht-öffentliche Version nur Projekten der amerikanischen Regierung zur Verfügung steht.

Somit sind die verfügbaren Module sehr eingeschränkt. Es gibt keine kollisionsfreie Pfadplanung, auch ist der Umgang mit Roboterarmen nur begrenzt möglich. Zudem ist das Einbringen der Umgebung in CLARAty nicht bekannt.

Gleichzeitig ist zu beachten, dass die Weiterentwicklung des öffentlichen Releases ungewiss ist. Aus diesen Gründen sollte auf einen Einsatz des Tools verzichtet werden. Es gibt Alternativen, die wesentlich mehr beherrschen und aktiver gepflegt werden.

4.2.1.7. OOPSMP

Die OOPSMP Bibliothek besticht mit ihren sehr weit entwickelten Pfadplanungsalgorithmen. Die Bibliothek kann mit Kollisionen umgehen und bietet zudem eine grafische Oberfläche, mit der die ganze Szene visualisiert werden kann. Es können einfachste Körper dargestellt werden. Dieses ist für die Szene ausreichend, leider gilt diese Darstellung auch für den Roboter. Somit können nur Roboter, die mit Quadern beziehungsweise Zylindern annäherbar sind, simuliert werden.

Der Umgang mit Roboterarmen muss also durch ein anderes Tool oder durch Eigeninitiative gelöst werden. Auch muss das Problem der Inversen Kinematik gelöst werden. Da diese Probleme nicht trivial sind und diese in anderen Tools bereits gelöst sind, sollten diese Alternativen bevorzugt werden.

4.2.1.8. MPK

Die Motion-Planning-Kit-Funktionsbibliothek beherrscht den Umgang mit Roboterarmen, sogar mit mehreren gleichzeitig. Auch kann sie kollisionsfreie Pfadplanung durchführen. Dennoch ist zu beachten, dass nicht klar ist, wie neue Roboter hinzugefügt werden können. Auch ist nicht bekannt, ob die Inverse Kinematik dazu bekannt sein muss oder ob MPK diese selber berechnet.

Vor allem ist auf die scheinbar eingestellte Weiterentwicklung der Bibliothek hinzuweisen. Dies kann zu erheblichen Schwierigkeiten bei der Benutzung dieser Bibliothek führen.

4.2.2. Zusammenfassung

Zusammenfassend bleibt zu sagen, dass es von Vorteil ist, ein Framework auszuwählen anstatt einer Funktionsbibliothek. Gerade in den Punkten Visualisierung und Funktionsumfang sind Frameworks besser ausgerüstet. Die Visualisierung wird gerade am Anfang benötigt, um heraus zu finden wie sich bestimmte Algorithmen verhalten beziehungsweise wie sich der Roboterarm verhält.

Bei den Frameworks gibt es einige gute Kandidaten wie OpenRAVE, Webots und Microsoft Robotics Studio. Dennoch haben Webots und Microsoft Robotics Studio im Gegensatz zu OpenRAVE diverse Nachteile. Weder Webots noch das Robotics Studio bieten eine richtige Pfadplanung. Zum anderen ist Webots eine kostenpflichtige Software und das Studio nur auf Windows beschränkt. Dagegen bietet OpenRAVE alles, was für dieses Szenario gebraucht wird. Es kann kollisionsfreie Pfadplanung übernehmen. Die Umgebung kann dynamisch aktualisiert werden. Es benutzt die Inverse Kinematik des selber spezifizierten Roboters und beherrscht das Umhängen eines Objektes beim Greifvorgang.

Somit bleibt abschließend zu sagen, dass OpenRAVE ein ausgezeichneter Kandidat für die kollisionsfreie Manipulation von einfachen Alltagsgegenständen ist.

4.3. Ausgewählter Ansatz und Detaillösung

Die Software für die kollisionsfreie Manipulation von einfachen Alltagsgegenständen besteht aus fünf verschiedenen Vorgängen. Die einzelnen Schritte sind in Abbildung 4.3.1 zu sehen.

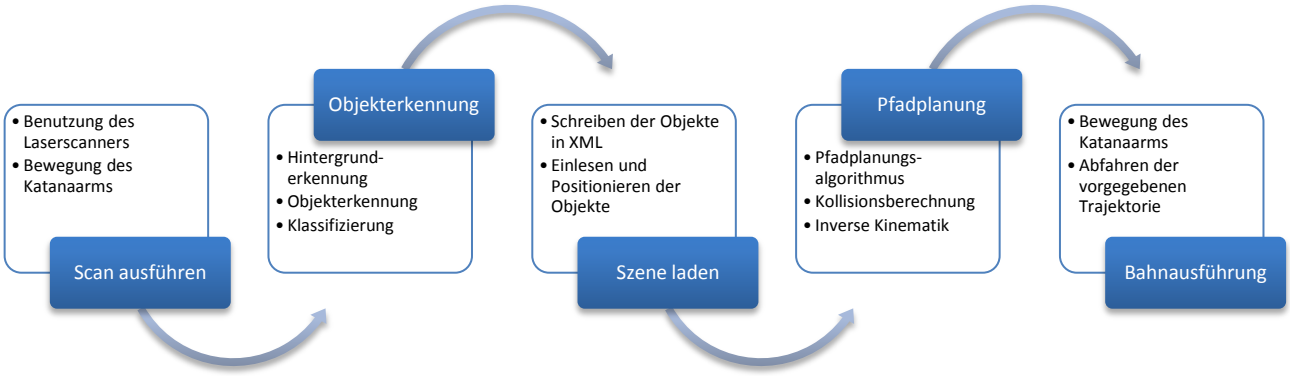

Abbildung 4.3.1: Einzelne Vorgänge bei der Manipulation von einfachen Alltagsgegenständen

Zuerst wird ein Laserscan ausgeführt, dabei werden der Laserscanner und der Katana-Arm verwendet. Der Laserscan gibt eine 3D-Punktwolke als Ergebnis aus. Diese wird in der Objekterkennung [WOP09] verwendet und verarbeitet. Aus dieser 3D-Punktwolke wird der Hintergrund extrahiert und die einzelnen Objekte werden erkannt und klassifiziert. Diese klassifizierten Objekte werden in das XML-Format von OpenRAVE gebracht und in die Szene geladen. Diese Szene repräsentiert dann die Wirklichkeit. Darauf folgend können kollisionsfreie Pfade zu den einzelne Objekten geplant werden. Dazu wird ein Pfadplanungsalgorithmus verwendet, der sich die Inverse Kinematik des Katana-Arms zu Hilfe nimmt. Die Pfadplanung generiert eine Trajektorie[1]. Jeder geplante Pfad wird zuerst in OpenRAVE selber ausgeführt und anschließend ebenfalls auf dem Katana-Arm.

In den weiteren Punkten wird im Detail darauf eingegangen wie diese fünf Schritte miteinander verbunden sind und welche Einzelheiten diese Schritte beinhalten.

[1] Trajektorie ist in diesem Fall ein Objekt, das die einzelnen Wegpunkte für den kollisionsfreien Pfad enthält.

4.3.1. Erfassen der Umwelt mittels Laser

Die Umwelt wird mittels eines auf den Katana-Arm montierten Lasers erfasst. Die genau Befestigung und die Umrechnung des Koordinatensystems vom Endeffektor des Katanas, an welchem der Laser befestigt ist, in das Grundkoordinatensystems des Katanas, wird in [WOP09] beschrieben.

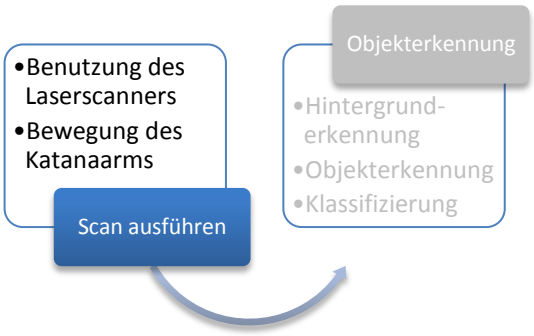

Abbildung 4.3.2: Der Scanvorgang im Detail

In Abbildung 4.3.2 ist zu erkennen, dass ein Scanvorgang im Wesentlichen aus zwei Teilschritten besteht. Diese Schritte sind: Die Benutzung des Laserscanners zur Erfassung der Umgebung und die Benutzung des Katana-Arms zur Bewegung des Laserscanners.

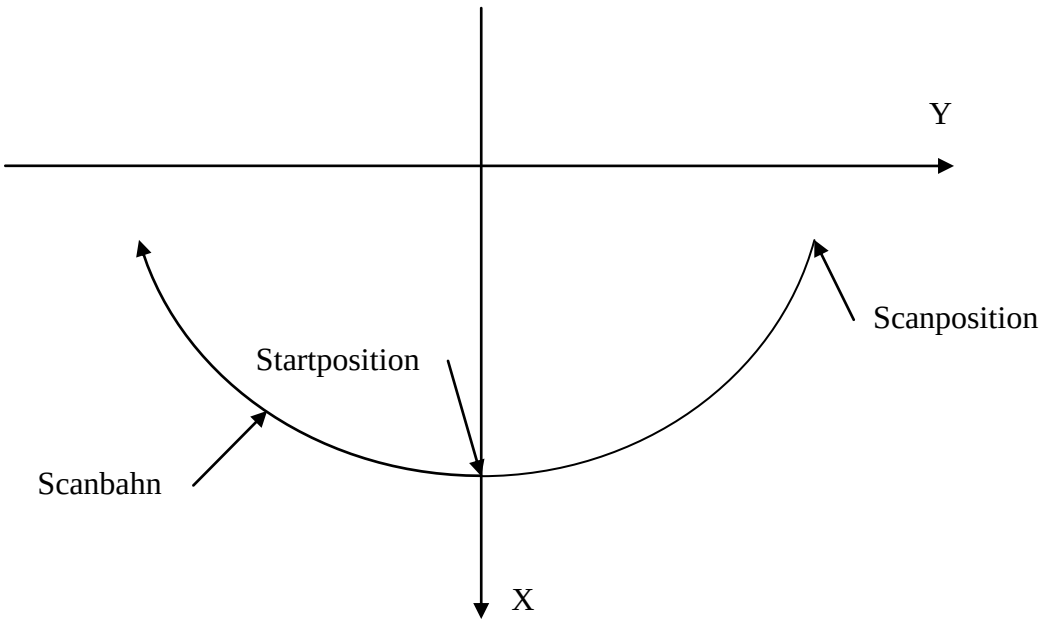

Abbildung 4.3.3: Scanbahn im Grundkoordinatensystems des Katanas

Ein Scanvorgang ist im Einzelnen wie folgt aufgebaut, dieses ist auch in Abbildung 4.3.3 verdeutlicht. Zuerst wird der Katana-Arm in die Startposition gebracht, diese kann in Abbildung 4.3.4 gesehen werden. In dieser Position zeigt der Endeffektor in die positive X-Richtung des Basiskoordinatensystems und liegt auf der Nullkoordinate der Y-Achse. Darauf folgend wird das erste Gelenk um 80 Grad in die positive Y-Achse gedreht, somit befindet sich der Katana in der Scanposition. Ab dieser Position kann der eigentliche Scanvorgang beginnen.

Der eigentliche Scanvorgang besteht aus vielen kleinen, sich immer wiederholenden Teilschritten. Zuerst wird ein Laserscan mit dem Laserscanner gemacht. Sobald dieser abgeschlossen ist, bewegt sich der Katana um einen bestimmten Gradanteil mit dem ersten Gelenk. Danach folgt wieder der erste Schritt, in dem ein Laserscan gemacht wird. Diese zwei Vorgänge werden so lange wiederholt bis der gesamte Scanbereich abgefahren ist. In der derzeitigen Ausarbeitung sind dies 220 Wiederholungen.

Abbildung 4.3.4: Katana-Arm in Startposition

Die Bahn des Scanvorgangs wird nicht mit OpenRAVE berechnet. Diese Bahn ist fest kodiert und berücksichtigt keine Hindernisse. Es wäre auch nicht möglich, Hindernisse zu berücksichtigen, da noch keine Informationen über die Umgebung vorliegen. Diese sind erst nach dem Laserscan verfügbar.

4.3.2. Objekterkennung

Die Objekterkennung besteht aus mehreren Teilbereichen. Die detaillierte Beschreibung und Ausarbeitung kann in [WOP09] nachgelesen werden.

In Abbildung 4.3.5 sind die einzelnen Teilschritte der Objekterkennung zu erkennen. Zuerst werden Hintergrundobjekte erkannt. Unter diese Objekte fällt zunächst der Tisch, auf dem später die greifbaren Objekte stehen. Danach werden die übriggebliebenen Punktwolken zu Objekten zusammengefasst. Als letztes werden die erkannten Objekte gegen eine bestehende Datenbank verglichen. Falls es zu Treffern kommt, werden diese Objekte als wiedererkannte Objekte klassifiziert. Wiedererkannte Objekte sind vorher eingelernte Objekte wie Becher, Flaschen oder Schachteln. Falls es zu keinem Treffer kommt, werden diese, wie der Tisch, als Hindernisse klassifiziert.

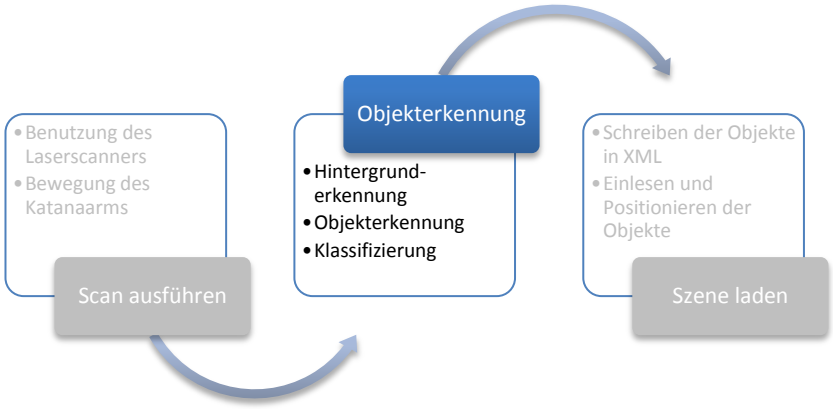

Abbildung 4.3.5: Die Objekterkennung im Detail

Die Objekterkennung liefert Bounding Boxen als Abmaß für die einzelnen Objekte. Desweitern wird auch der Typ des Objektes mitgeliefert.

Es wurden Bounding Boxen als Objekt-Repräsentation gewählt, da Punktwolken von OpenRAVE nicht dargestellt werden können. OpenRAVE benutzt solide Körper, wie sie in der realen Welt vorhanden sind. Aus diesen Körpern können nützlichen Informationen wie Masse und Abmaße ausgelesen werden.

4.3.3. Schnittstelle Objekterkennung und Pfadplanung

Die Schnittstelle zwischen der Objekterkennung und der Pfadplanung ist der Austausch von einfachsten Körpern. Die Bounding Boxen, die von der Objekterkennung generiert werden, werden auf einfache Art in OpenRAVE gebracht. Dazu werden alle Objekte, wiedererkannte

Objekte und Hindernisse, in separate XML-Dateien geschrieben. Zusätzlich zu jeder XML-Datei wird ein Pose-Objekt gehalten. Dieses Objekt beinhaltet die Position und die Orientierung des Objektes in der Szene. Mit diesen Informationen kann die geschriebene XML-Datei von OpenRAVE eingelesen und das Objekt in der Szene positioniert werden.

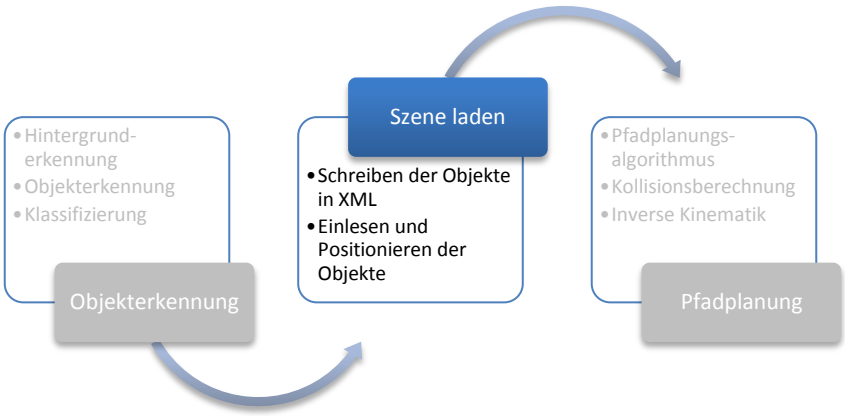

Abbildung 4.3.6: Teilschritte Szene laden

4.3.3.1. Objektvergrößerung und -verkleinerung

Bei der jetzigen Implementierung werden die Abmaße der Objekte, bevor sie in eine XML-Datei geschrieben werden, angepasst. Dieses Vorgehen ist von Nöten, da der Pfadplanungsalgorithmus sehr eng um die Objekte herum plant.

Deswegen wurden die Abmaße der Hindernisse erhöht. Dieses gibt eine zusätzliche Sicherheit bei der Planung. Zugleich wurden die Abmaße der wiedererkannten Objekte verringert. Diese Maßnahme ist wegen des Greifvorgangs nötig. Der Katana-Arm weist nur einen sehr einfachen Greifer vor. Zudem ist dieser eher für runde Sachen ausgelegt. Deswegen fällt es schwer einen Becher, der mit einer Bounding Box beschrieben wird, mittig zu greifen. Ohne die Verringerung der Abmaße kollidieren die beiden Greifzangen an der Bounding Box und es wird kein Pfad berechnet. Durch die Verringerung der Abmaße ist ein sauberer und mittiger Greifvorgang möglich.

4.3.3.2. Schreiben der XML-Datei

Für das Schreiben der XML-Datei wurde ein XML-Writer (siehe hierzu Punkt 4.4.1) entwickelt. Dieser schreibt den Typ (in diesem Fall nur, ob es sich um eine Bounding Box handelt) und die Abmaße des Körpers in einen Filestream. Diese Informationen werden dann zusammen mit speziellen Tags in die XML-Datei geschrieben. Diese besondere Form ist von Nöten, damit OpenRAVE das XML beim Einlesen als richtiges Format erkennt.

4.3.3.3. Pose-Objekt

Zusätzlich zu dem Schreiben der XML-Datei wird ein Pose-Objekt gehalten. Dieses Objekt enthält die Koordinaten des Mittelpunktes und die Orientierung des Körpers. Für die Orientierung des Körpers werden drei Winkel benutzt: der Yaw, Pitch und der Roll. Diese Winkel sind in Abbildung 4.3.7 zu sehen.

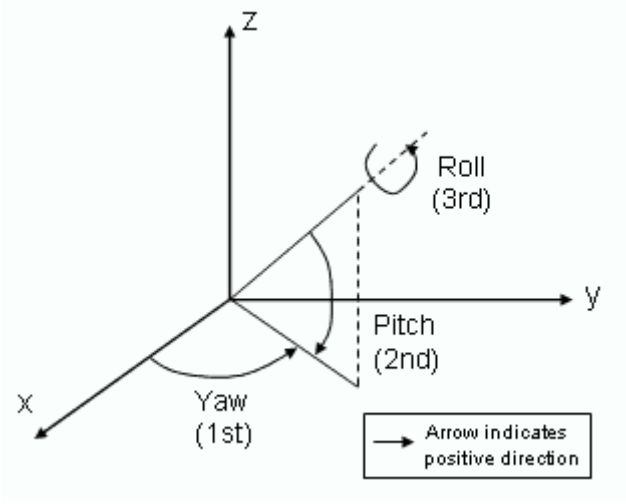

Abbildung 4.3.7: Yaw, Pitch und Roll des Pose Objektes (Quelle [MRP09])

4.3.3.4. Einlesen des Objektes

Beim Einlesen in OpenRAVE werden OpenRAVE die XML-Datei und das Pose-Objekt übergeben. OpenRAVE liest dabei zuerst die Datei ein und transformiert anschließend mit dem Pose-Objekt das eingelesene Objekt an die richtige Position. Alle Objekte werden so behandelt: wiedererkannte Objekte, Hindernisse und der Tisch. Zusätzlich wird intern eine Liste gehalten, in der alle greifbaren Objekte hinzugefügt werden. Diese Liste wird in der späteren Pfadplanung benötigt, um die zu greifenden Objekte zu identifizieren.

Somit ist die Schnittstelle zwischen der Objekterkennung und der Pfadplanung sehr einfach gehalten. Die XML-Datei kann jederzeit erweitert werden. Auch ist die Weitergabe des Pose-Objekts sehr einfach, da es nur eine Klasse beziehungsweise eine Matrix ist. Diese Flexibilität und Einfachheit sind von Vorteil, da die Objekterkennung und die Pfadplanung eigenständige Probleme sind und auch als solche behandelt werden.

4.3.4. OpenRAVE Struktur

Die Umgebung wird in OpenRAVE durch das Spezifizieren von XML-Dateien veranschaulicht. Für jedes Objekt wird eine eigene Datei angelegt. Somit besteht immer die komplette Kontrolle über alle Änderungen. Ändert sich nur ein Objekt, so muss nur diese eine Datei geändert werden; die übrigen Dateien bleiben unverändert. Dieses spart Zeit und auch Arbeitsaufwand.

Zusätzlich, um mit sehr vielen einzelnen Objekten umgehen zu können, ist es auch möglich in einer Haupt-XML-Datei auf andere Objekte zu verweisen. Somit wird nur eine Datei geladen und die gesamte Umgebung mit allen Objekten wird in der Szene dargestellt.

Die XML-Dateien beinhalten nicht nur die Abmaße und Lage der Objekte im Raum, sondern können auch Masse, Farbe und Reflexionsrate des Lichtes beinhalten. Diese Struktur gibt also nicht nur Aussehen und Position des Objektes an, sondern wirkt sich auch auf die Physik aus.

Die XML-Struktur beinhaltet viele verschiedene Attribute. Mit diesen ist es möglich, verschiedene Formen und Roboter zu spezifizieren. Es gibt zum Beispiel die Möglichkeit, Gelenke mit ihren Öffnungswinkel zu beschreiben oder Lichtquellen zu erzeugen. Zudem ist es nicht nur möglich, einfachste Körper aus Boxen und Zylindern zu beschreiben; es ist auch möglich, meshartige Objekte (siehe hierzu Kapitel 6 Punkt 6.2.1) einzubinden. Dabei unterstützt OpenRAVE die beiden Formate IV[1] und VRML. Objekte können in diesen Formaten modelliert und dann in die XML-Datei eingebunden werden.

Diese XML-Struktur macht das Ändern und Aktualisieren der Umgebung sehr einfach. Auch Erweiterungen können einfach realisiert werden.

4.3.4.1. Roboterarm

Der Roboterarm wird normalerweise in zwei Dateien spezifiziert. Die erste Datei ist das Grundgerüst des Roboterarms. Die zweite Datei referenziert auf das Grundgerüst und beinhaltet zusätzliche Attribute über den Roboterarm.

[1] IV ist ein Standardformat für den Austausch von 3D-Objekten. Nähere Information unter http://oss.sgi.com/projects/inventor/

In der ersten Datei werden die einzelnen Armsegmente angegeben. Anschließend ist es möglich, diese Segmente über Gelenke zu verbinden. Dabei kann Drehrichtung, Drehachse und Öffnungswinkel angegeben werden. In der zweiten Datei werden Manipulator und Inverse Kinematik Lösungsinstanz spezifiziert. Mit diesen Angaben wird der Arm detailliert beschrieben.

Das Modell für den Katana-Arm wurde von der Neuronics [NEU09] Webseite heruntergeladen. Das vorhandene Modell war in einer STP[1]-Datei speichert. Die STP-Datei wurde mit Hilfe von Rhinoceros[2] in einzelne VRML-Dateien zerlegt. Diese Dateien konnten dann in der XML-Datei benutzt werden, um die einzelnen Armsegmente darzustellen.

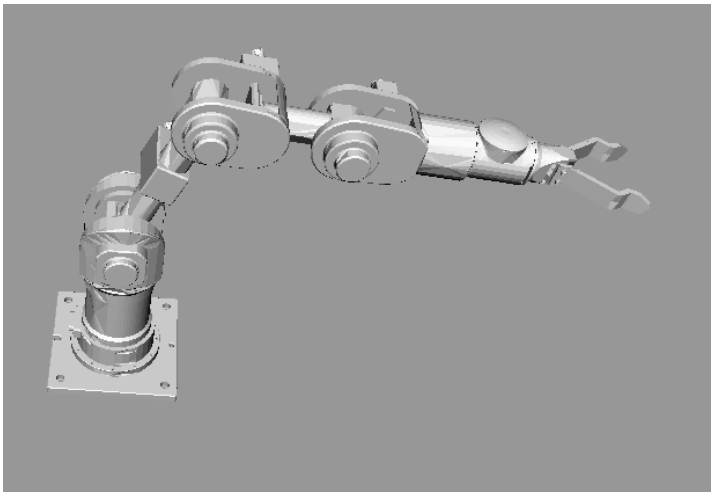

Abbildung 4.3.8: Katana Modell in OpenRAVE

Mit dieser Methode kann ein sehr präziser Katana-Arm in OpenRAVE, wie in Abbildung 4.3.8 zu sehen ist, modelliert werden. Ein großer Vorteil dieser genauen Darstellung ist die sehr exakte Kollisionsberechnung.

Abbildung 4.3.9: realer Katana

Anfangs wurde der Katana-Arm manuell von außerhalb geladen. In einer späteren Version von OpenRAVE wurde der Katana-Arm von Diankov in OpenRAVE direkt integriert. Dieses hat diverse Vorteile gegenüber dem manuellen Laden. Die Inverse Kinematik des Arms wird

[1] STP oder STEP ist ein standardisiertes Dateiformat für den Austausch von CAD Dateien. Nähere Informationen unter http://www.prostep.org/de/standards-amp-infos/was-ist-step.html
[2] Rhinoceros ist eine Modellierungssoftware. Nähere Informationen unter http://www.de.rhino3d.com/

direkt beim Kompilieren von OpenRAVE angelegt und muss nicht selber erstellt werden. Auch das Laden des internen Katana-Arms ist einfacher.

4.3.4.2. Statische Objekte

Statische Objekte sind in OpenRAVE Objekte, die nicht in die Kollisionsberechnung mit einberechnet werden. Auch wird deren Masse nicht beachtet. Ein statisches Objekt in OpenRAVE ist der Boden. Dieser dient nur dazu, dass Objekte besser gesehen werden können und damit der Betrachter weiß, wo oben und unten ist.

4.3.4.3. Dynamische Objekte

In OpenRAVE sind alle Objekte, die irgendetwas mit der Umgebung zu tun haben, als dynamisch deklariert. Somit sind sowohl der Roboterarm als auch der Tisch dynamische Objekte. Alle dynamischen Objekte werden in die Kollisionsberechnung mit einbezogen.

OpenRAVE kann also nicht unterscheiden, ob ein Objekt greifbar oder ein Hindernis ist. Diese Trennung muss in einer übergeordneten Instanz vorgenommen werden.

4.3.5. Pfadplanungsberechnung durch OpenRAVE

Die Berechnung des kollisionsfreien Pfades wird durch OpenRAVE ausgeführt. Die Pfadplanungsberechnung baut auf der eingelesenen Umgebung auf. Das Umgebungsmodell ist zugleich auch das Kollisionsmodell, welches zur Planung benötigt wird.

Die Inverse Kinematik des Katana-Arms wird benutzt, um die Endgelenkwinkelkonfiguration zu bestimmen. Diese Konfiguration und die Anfangsgelenkwinkelkonfiguration werden für den Pfadplanungsalgorithmus benötigt. Mit diesen Parametern kann der Algorithmus in der vorhandenen Szene planen.

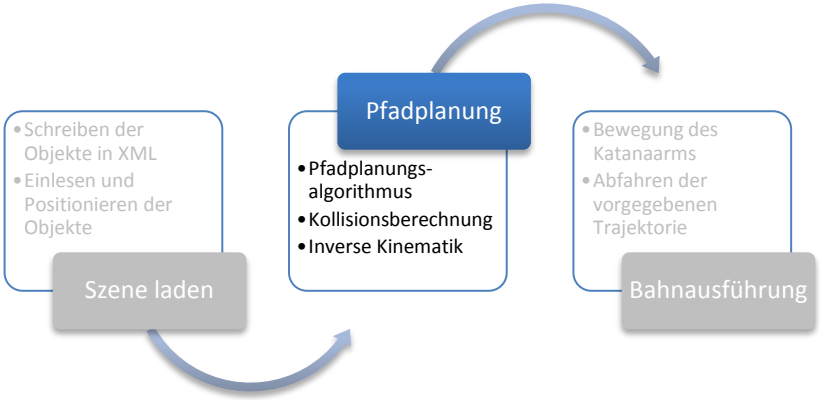

Abbildung 4.3.10: Pfadplanung im Detail

Es werden alle Objekte, egal ob Hindernisse, greifbare Objekte oder der Roboterarm selber, mit in die Kollisionsberechnung einbezogen. Es werden also Kollisionen mit der Umgebung und mit dem Roboterarm selber berechnet.

Durch die Bedingung, dass das zu greifende Objekt auch ein Hindernis ist, muss der Greifer vor der Planung geöffnet sein, sonst würde er mit dem zu greifenden Objekt kollidieren. Dadurch könnte bei der Pfadplanung kein Weg gefunden werden.

4.3.5.1. Inverse Kinematik

Die Inverse Kinematik des Katana-Arms wird von OpenRAVE generiert. Dazu wird die XML-Datei des Katanas verwendet. In dieser stehen alle benötigten Daten, wie die Gelenkwinkelkonfiguration, die für die Berechnung nötig sind.

Die Inverse Kinematik ist Teil des Roboters und kann direkt verwendet werden. Dazu wird in der Roboterklasse eine Funktion aufgerufen. Diese benötigt eine Matrix, welche die Zielposi-

tion des Endeffektors beschreibt. Das Ergebnis der Inversen Kinematik sind die zur Zielposition gehörenden Gelenkwinkel.

Die Inverse Kinematik berücksichtigt dabei Selbstkollisionen mit dem Arm und Kollisionen mit der Umwelt. Sobald aufgrund dessen eine Gelenkstellung nicht möglich ist, wird keine Lösung generiert.

4.3.5.2. Pfadplanungsalgorithmus

Der Pfadplanungsalgorithmus benötigt die Anfangsgelenkwinkelkonfiguration und die Endgelenkwinkelkonfiguration. Die Anfangsgelenkwinkelkonfiguration kann von dem realen Katana ausgelesen werden. Hierzu stellt die Katana-API Methoden zur Verfügung. Dabei muss beachtet werden, dass der Gelenkwinkelbereich umgerechnet werden muss. Hierzu kann die Tabelle 4.3.1 verwendet werden.

Umrechnung realer Katana zu OpenRAVE
OpenRAVE Gelenkwinkel 1 = PI – realer Gelenkwinkel 1
OpenRAVE Gelenkwinkel 2 = PI/2 – realer Gelenkwinkel 2
OpenRAVE Gelenkwinkel 3 = PI – realer Gelenkwinkel 3
OpenRAVE Gelenkwinkel 4 = PI – realer Gelenkwinkel 4
OpenRAVE Gelenkwinkel 5 = PI – realer Gelenkwinkel 5

Tabelle 4.3.1: Umrechnung der Gelenkwinkel vom Katana-Arm zu OpenRAVE

Die umgerechneten Gelenkwinkel ergeben die Anfangsgelenkwinkelkonfiguration. Die Endgelenkwinkelkonfiguration wird aus der Inversen Kinematik des Katana-Arms berechnet wie in Punkt 4.3.5.1 erläutert.

Sobald die Gelenkwinkelkonfigurationen vorhanden sind, kann der Pfadplanungsalgorithmus mit diesen Parametern initialisiert werden. Zusätzlich zu den Winkeln benötigt der Algorithmus auch den Roboter. Nach der Initialisierung kann die eigentliche Pfadplanung gestartet werden. Diese wird komplett von OpenRAVE übernommen und liefert als Ergebnis ein Trajektorie-Objekt, welches Punkte des kollisionsfreien Pfades beinhaltet.

Verschiedene Algorithmen

Für die Pfadplanung stellt OpenRAVE mehrere verschiedene Algorithmen bereit. Diese vier Algorithmen wurden in Kapitel 3 erläutert. Für diese Ausarbeitung wurde der rBiRRT-Algorithmus verwendet. Dieser Algorithmus wurde ausgewählt, da er am schnellsten und am zielsichersten funktioniert. Dieses wird im Kapitel 5 mit Versuchen belegt.

4.3.5.3. Selbstkollision und Kollision mit der Umwelt

Für die Kollisionsberechnung wird die Umgebung verwendet, welche mit XML-Dateien spezifizierte wurde. Somit ist die visuelle Darstellung auch gleich dem Kollisionsmodell.

Die Kollisionsberechnung wird intern von OpenRAVE verwaltet. Bei der Berechnung der Endgelenkwinkelkonfiguration durch die Inverse Kinematik kann angegeben werden, ob Kollision mit Objekten berücksichtigt werden soll oder nicht. Das gleiche kann bei der Pfadplanungsberechnung angegeben werden.

4.3.5.4. Umhängen von Welt- in Roboterkoordinatensystem

Bei einem Greifvorgang wird zuerst der Pfad zu dem zu greifenden Objekt berechnet. Dabei ist das zu greifende Objekt ein Objekt der Szene und wird als Hindernis angesehen. Somit werden Kollisionen mit diesem Objekt berechnet.

Beim Greifen des Objektes wird das Objekt, welches vorher zur Szene gehört hat, zum Objekt des Roboterarms. Dies hat den Vorteil, dass bei einer weiteren Pfadplanung das gegriffene Objekt mitberücksichtigt wird. Somit wird im neuen Pfad nicht nur die Kollision des Armes gegen die Umwelt getestet, sondern es wird der erweiterte Arm, der nun auch das gegriffene Objekt beinhaltet, gegen die Umwelt getestet. Durch dieses Vorgehen ist es in OpenRAVE möglich, sehr große Objekte zu greifen und dennoch einen kollisionsfreien Pfad zu berechnen. Wäre dies nicht möglich, so müsste es manuell in Form eines Offsets zum errechneten Pfad berücksichtigt werden.

Für das Umhängen des Objekts müssen sowohl Koordinaten wie auch Orientierung verändert werden. Die vorherigen Koordinaten des Objekts sind nicht mehr korrekt, da es sich nicht mehr im Weltkoordinatensystem befindet, sondern im Koordinatensystem des Roboters. Hierzu müssen die neuen Koordinaten berechnet werden. Diese können aus dem Roboter ausgelesen werden; der Roboter bietet eine Funktion an, welche die Endeffektor-Transformation liefert. Diese beinhaltet die Koordinaten und Orientierung für den Greifer und somit auch für

das gegriffene Objekt. Die Umrechnung für die Orientierung wurde in dieser Ausarbeitung nicht gelöst.

Somit kann es vorkommen, dass Objekte nicht richtig gedreht und dadurch auch nicht richtig gegriffen werden. Dies ist aber bei den einfachen Körpern unerheblich, da Becher und Schachteln fast ebenso hoch wie breit sind und es somit nicht zu großen Komplikationen kommt. Für andere Objekte wie Flaschen müsste dieses jedoch richtig berechnet werden, da es sonst vorkommen kann, dass es zu Kollisionen kommt, weil die Umgebung von OpenRAVE nicht mit der Wirklichkeit übereinstimmt.

4.3.6. Bahnausführung

Die Ausführung der Bahn besteht aus zwei separaten Bereichen. Einmal wird die berechnete Bahn in OpenRAVE ausgeführt, damit das Planen der nächsten Bahn ermöglicht wird. Das andere Mal wird die Bahn auch an den Katana-Arm weitergeleitet und dort ausgeführt.

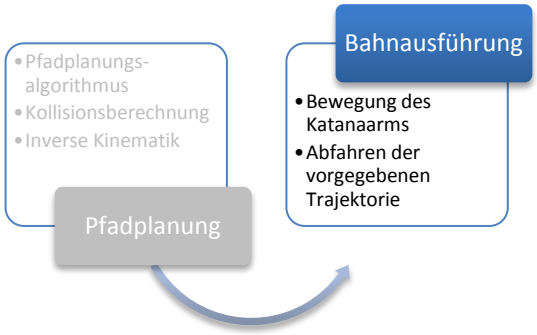

Abbildung 4.3.11: Die Bahnausführung im Detail

Beide Bereiche führen jeweils die von der Pfadplanung erzeugte Trajektorie aus, welche im Codeauszug 4.3.1 zu sehen ist. Das Trajektorie-Objekt beinhaltet je nach Komplexität der Umgebung zwischen 50 und 200 Punkten. Jeder dieser Punkte besteht aus mehreren Gelenkwinkeln des Arms. Die Gelenkwinkel sind absolute Gelenkwinkel und geben somit eine eindeutige Position an. Dieses hat den Vorteil, dass bei längerer Bewegung keine Abweichungen zwischen OpenRAVE und dem Katana-Arm auftreten können.

4.3.6.1. Ausführung in OpenRAVE

Die Bahnausführung in OpenRAVE wird intern abgewickelt. Die von der Pfadplanung erstellte Trajektorie kann OpenRAVE direkt über einen Funktionsaufruf übergeben werden. Das Abfahren der Bahn wird somit intern von OpenRAVE gehandhabt.

```
166 5 0
0.00788265   -0.531624    2.14462      1.91059      2.87052
0.0040648    -0.528107    2.14309      1.90119      2.85301
0.00024695   -0.52459     2.14156      1.89179      2.83551
-0.0035709   -0.521072    2.14003      1.88239      2.818
-0.00738875  -0.517555    2.13849      1.87299      2.8005
```

Codeauszug 4.3.1: Auszug aus dem Trajektorie-Objekt

4.3.6.2. Reale Roboter Bewegung

Für die Bewegung des Katana-Arms müssen die einzelnen Punkte in der Trajektorie abgefahren werden. Diese müssen aber zuerst umgerechnet werden, da die Modellierung der Gelenke in OpenRAVE anders ist als die des realen Arms. Für die Umrechnung kann die Tabelle 4.3.2 verwendet werden.

Umrechnung OpenRAVE zu realem Katana
realer Gelenkwinkel 1 = PI – Gelenkwinkel 1 von OpenRAVE
realer Gelenkwinkel 2 = PI/2 – Gelenkwinkel 2 von OpenRAVE
realer Gelenkwinkel 3 = PI – Gelenkwinkel 3 von OpenRAVE
realer Gelenkwinkel 4 = PI – Gelenkwinkel 4 von OpenRAVE
realer Gelenkwinkel 5 = PI – Gelenkwinkel 5 von OpenRAVE

Tabelle 4.3.2: Umrechnung OpenRAVE zu realem Katana

Mit dieser Umrechnung werden die Gelenkwinkel in den richtigen Winkelbereich gebracht. Danach müssen die Gelenkwinkel dem Katana-Wrapper (siehe hierzu Punkt 4.4.3) übergeben werden. Die Methode des Wrappers nimmt die Gelenkwinkel eines Punktes auf. Diese werden dann an die Katana-API weitergeleitet. Die Gelenke werden einzeln und ohne aufeinander zu warten angesteuert. Dies führt dazu, dass sich die Gelenke fast gleichzeitig bewegen. Außerdem ist die Änderung von einem Punkt zum nächsten so gering, dass die Bewegung der einzelnen Gelenke minimal ist und sehr schnell von statten geht. Somit führt dieses zu einem sehr flüssigen Ablauf.

Nach vollständiger Abarbeitung der Trajektorie hat der Katana-Arm die vorberechnete Bahn ohne Kollision mit Hindernissen abgefahren.

4.4. Software-Design

Die Software für die kollisionsfreie Manipulation einfacher Alltagsgegenstände besteht aus drei Hauptkomponenten. Diese Komponenten sind in Abbildung 4.4.1 zu sehen. Die *Objekterkennung* wurde in der parallel entwickelten Bachelorarbeit erstellt. Die *Pfadplanung* wurde in dieser Arbeit entwickelt. Der *Hardware*-Bereich wurde zum Teil in dieser Arbeit entwickelt (Katana-Arm) und zum Teil von Herr Wopfner (Laserscanner).

Alle orange markierten Bereiche sind in dieser Arbeit entwickelt worden. Die grünen Bereiche sind in beiden Arbeiten entwickelt worden. Die weißen Bereiche sind nur von Herr Wopfner entwickelt worden.

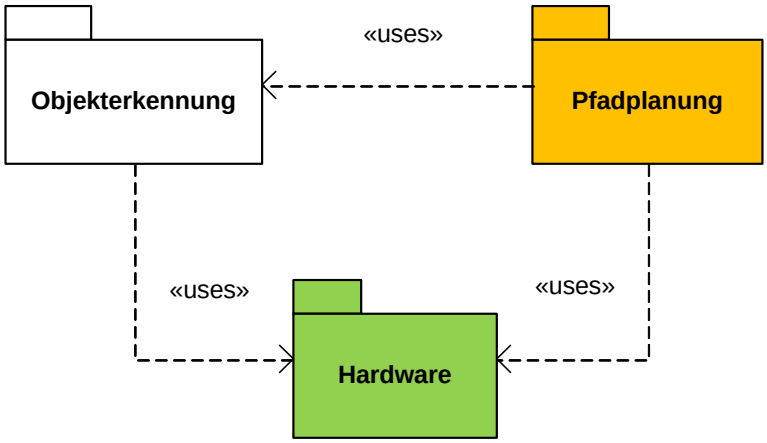

Abbildung 4.4.1: Übersicht der einzelnen Komponenten

Die *Hardware* wird von beiden anderen Komponenten benutzt. So benötigt die *Objekterkennung* den Laserscanner und den Katana-Arm. Die *Pfadplanung* benötigt zusätzlich zu dem Katana-Arm auch noch die Objekterkennung.

In Abbildung 4.4.2 wird die Pfadplanungs- und Hardware-Komponente detaillierter dargestellt. Die Pfadplanung besteht aus zwei Klassen: *OpenRave* und *ObjektXMLWriter*. Die *Katana*-Klasse gehört zu der Hardware-Komponente. Die *HardwareController*-Klasse koordiniert und initialisiert den Laserscanner und den Katana-Arm. Der *MainController* fügt die drei Komponenten zusammen. Diese Klasse initialisiert alle wichtigen Klassen und steuert den Programmablauf.

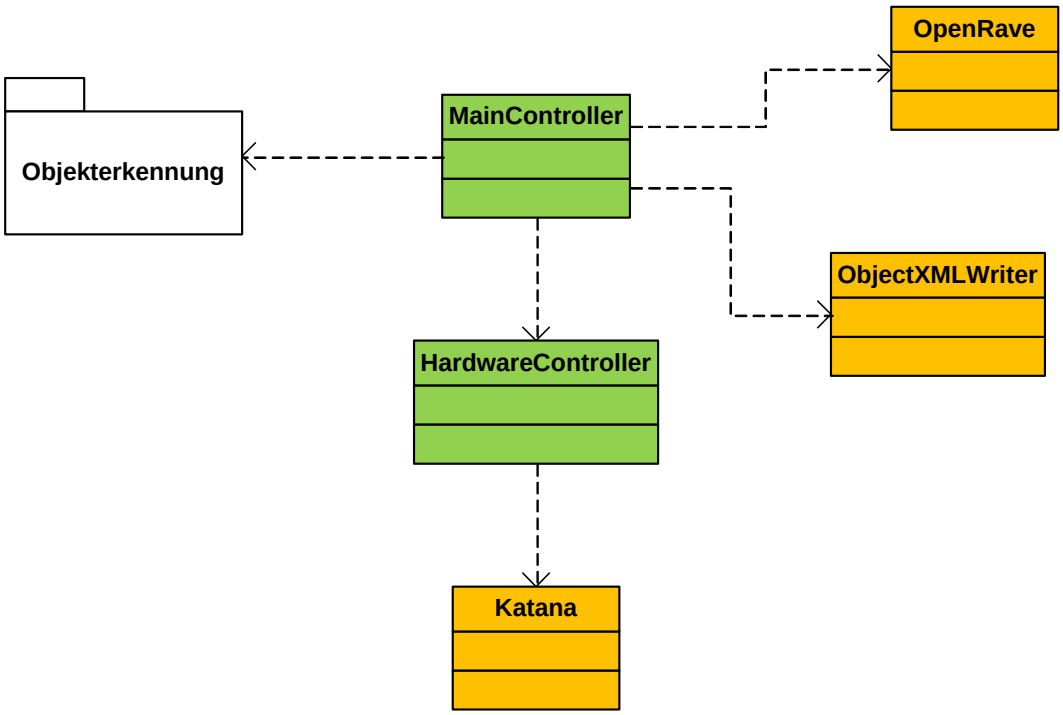

Abbildung 4.4.2: Klassendiagramm der Pfadplanung

Bei einem Szenario holt sich die *MainController*-Klasse alle wichtigen Daten von der Objekterkennung und steuert durch den *HardwareController* den Laserscanner und den Katana-Arm an. Danach werden die Daten der Objekterkennung in OpenRAVE eingespeist. Der berechnete Pfad wird wiederum an den *HardwareController* weitergeleitet und somit auf dem Katana-Arm ausgeführt.

4.4.1. ObjectXMLWriter

Die ObjectXMLWriter-Klasse ist eine einfache Klasse, die nur einen *Filestream* als privates Element beinhaltet. Desweitern sind einige wichtige Schlüsselwörter als konstante *Strings* deklariert um den Zugriff zu erleichtern. Diese Schlüsselwörter sind für die XML Struktur von OpenRAVE von Bedeutung.

Die einzige Methode der Klasse ist die *writeKinBodyBox*()-Methode. Diese Methode nimmt den Dateinamen und die Abmaße des zu schreibenden Objektes entgegen. Die Methode erzeugt mit den Abmaßen und den Schlüsselwörtern eine XML-Datei, die von OpenRAVE gelesen werden kann.

4.4.2. OpenRave-Wrapper

Die OpenRave-Wrapperklasse ist die Schnittstelle zwischen der hier entwickelten Software und OpenRAVE. Der Wrapper kapselt komplizierte Ausführungen und bietet einfache Methoden an. Zudem enthält diese Klasse alle wichtigen Informationen, die nötig sind um OpenRAVE zu initialisieren, und macht somit eine einfache Interaktion mit OpenRAVE möglich.

```
1  this->robot->GetActiveDOFValues(params.vinitialconfig);
2  this->calculateIKSolution(target, params.vgoalconfig,
3                                                iteration, offset);
4
5  Trajectory* traj = this->environment->CreateTrajectory(
6                                  this->robot->GetActiveDOF());
7
8  this->planner->InitPlan(this->robot, &params);
9  bool success = this->planner->PlanPath(traj);
10
11 if (success)
12 {
13        return traj;
14 }
```

Codeauszug 4.4.1: Auszug aus der planPath-Methode der OpenRave Klasse

Codeauszug 4.4.1 zeigt einen Auszug aus der *planPath*-Methode. Diese Methode ist eine der wichtigsten der Klasse. Diese Methode plant den kollisionsfreien Pfad zu einem Objekt. Zuerst wird die Anfangsgelenkwinkelkonfiguration des Roboterarms in OpenRAVE ausgelesen (1). Dieser ist mit dem realen Katana-Arm synchronisiert. Danach wird mithilfe der Inversen Kinematik die Endgelenkwinkelkonfiguration berechnet (2). Um mit dem Planner einen Pfad generieren zu können, muss zuerst ein Trajektorie-Objekt (5) angelegt werden. Danach kann der Planner mit dem Roboter und mit den Gelenkwinkelkonfigurationen initialisiert werden (8). Nach der erfolgreichen Initialisierung kann der kollisionsfreie Pfad erzeugt werden (9). Am Ende wird geprüft ob ein Pfad erstellt wurde.

Diese Methode zeigt die Komplexität, die von der Wrapperklasse gekapselt wird. Bei der Verwendung dieser Methode muss nur das Zielobjekt übergeben werden. Als Ergebnis wird das Trajektorie-Objekt geliefert, in welchem der kollisionsfreie Pfad gehalten wird.

Somit ist dieser Wrapper eine gute Trennung zwischen der Programmausführung und OpenRAVE. Sollten Änderungen an OpenRAVE auftreten, so können diese in dem Wrapper behandelt werden. Die Programmausführung bleibt davon unberührt. Dieses erspart Arbeit und Zeit.

4.4.3. Katana-Wrapper

Der Katana-Wrapper ist eine Klasse, welche die sehr umfangreiche Katana-API von Neuronics abkapselt. Die API enthält sehr viele Methoden die teils mit vielen verschiedenen Parametern initialisiert werden müssen.

Sinn dieser Klasse ist es, die Komplexität der Katana-API zu verbergen und einfache Methoden anzubieten.

```
1  void Katana::moveMotorsToAngles_Rad(const vector<double>& angles)
2  {
3         for (unsigned int i = 0; i < angles.size(); ++i)
4         {
5                m_katana->moveMotorTo(i, angles[i], false);
6         }
7  }
```

Codeauszug 4.4.2: Methode aus der Katana-Klasse

In dem Codeauszug 4.4.2 ist eine Methode der Katana-Klasse zu sehen. Diese Methode nimmt einen Vektor mit Gelenkwinkeln entgegen (1). Für jeden Gelenkwinkel in diesem Vektor wird eine *moveMotorTo*-Methode (5) der Katana-API aufgerufen. Dort werden zuerst das benutze Gelenk und danach der Winkel spezifiziert. Der letzte Parameter gibt an, ob auf das erfolgreiche Ausführen gewartet werden soll (*true*) oder ob der Befehl ohne zu warten an den Katana-Arm abgesetzt werden soll (*false*).

Durch diese Vereinfachung kann in der weiteren Benutzung sehr einfach auf den Katana-Arm zugegriffen werden. Es müssen keine unnötigen Parameter angegeben werden und die Funktionen sind genau auf die gewollte Funktionalität zugeschnitten. Dies erleichtert das Programmieren und verhindert Leichtsinnsfehler.

Kapitel 5 Ergebnisse

5.1. Validierung des Gesamtkonzeptes

Die in dieser Bachelorarbeit entwickelte Lösung für das Problem der kollisionsfreien Manipulation von einfachen Alltagsgegenständen erweist sich als sehr effizient und robust. Dieses wird durch mehrere Tests belegt, welche unter Punkt 5.2 und 5.3 näher beschrieben werden.

Für das einwandfreie Funktionieren des Scans muss der Roboterarm mit seiner Y-Achse parallel zum Tisch stehen. Ist dieses nicht gegeben, wird nicht der ganze Tisch gescannt, sondern nur ein Teilbereich davon. Die genaue Position mit dem Arbeitsbereich des Katanas kann in Abbildung 5.1.1 gesehen werden.

Der Arbeitsbereich des Katanas liegt in der positiven X-Achse und reicht abhängig von der Höhe des Tisches von ungefähr 55 bis 65 cm. Dabei muss beachtet werden, dass der Katana-Arm noch etwas entfernt vom Tisch stehen muss, damit er sich überhaupt bewegen kann. Dies beschränkt den effektiven Greifbereich auf dem Tisch auf ungefähr 45 bis 55 cm. Auch dürfen Objekte nicht zu nah am Arm stehen, da diese sonst nicht gegriffen werden können. Die kürzeste Reichweite beträgt 25 cm.

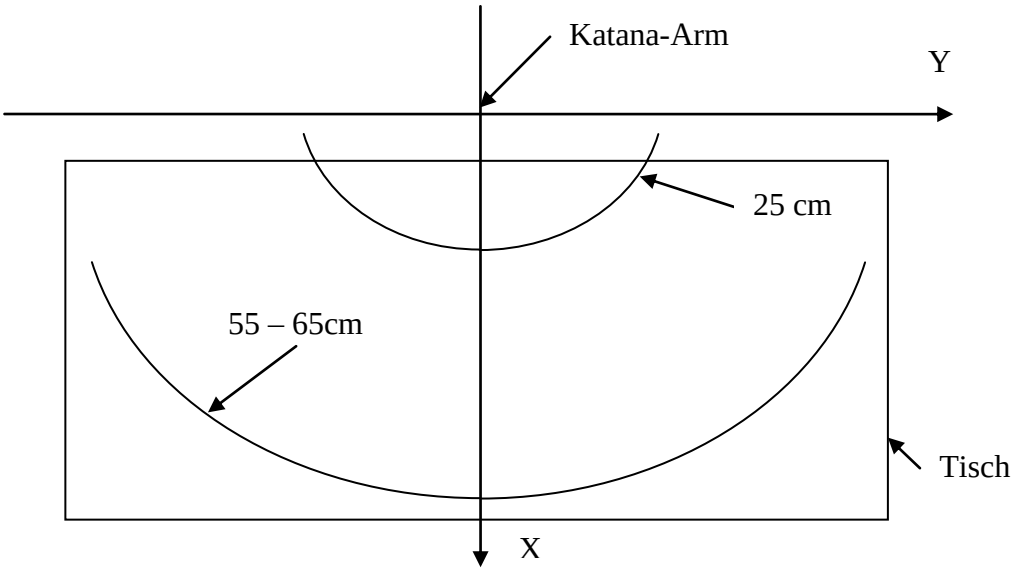

Abbildung 5.1.1: Arbeitsbereich des Katana-Arms

Es können einfache Objekte wie Becher und Dosen gegriffen werden. Diese dürfen nicht mehr als 500 g wiegen. Objekte wie Schüsseln oder Teller können aufgrund der Bauweise des Greifers nicht richtig oder nur sehr schwer gegriffen werden. Auch dürfen keine zu hohen Objekte in der direkten Sicht auf das zu greifende Objekt sein, da der Arm sonst keine Möglich-

keit hat, das Objekt zu greifen. Diese Bedingung ergibt sich aus den Freiheitsgraden des Katana-Arms. Durch diese ist es dem Arm nur möglich, Objekte geradlinig zu greifen. Somit kann nicht um das Hindernis herum gegriffen werden. Stattdessen muss versucht werden, über das Hindernis zu greifen, was dazu führt, dass das Hindernis nicht zu hoch sein darf. Zudem sollten die Objekte nicht zu nah aneinander stehen, da der Greifer einen gewissen Platz für seine Aktionen braucht. Auch muss beachtet werden, dass die Objekte in OpenRAVE aufgrund ihrer Bounding Boxen etwas größer beziehungsweise anders geformt sind. Deswegen sollte ein Mindestabstand von ungefähr 5 cm eingehalten werden, damit ein sauberer Greifvorgang garantiert werden kann.

Die kollisionsfreie Pfadplanung funktioniert zufriedenstellend, auch unter der Berücksichtigung, dass gegriffene Objekte mit in die Planung einbezogen werden. Es werden fast alle Bewegungen des Katana-Arms kollisionsfrei berechnet. Die einzige Bahn, die nicht kollisionsfrei garantiert werden kann, ist die Scanbahn. Diese Bahn kann aber auch nicht kollisionsfrei geplant werden, da die Umgebung erst nach dem Scan bekannt ist.

5.2. Beschreibung der Testfälle

5.2.1. Großer Tisch

Für diesen Test wurde ein großer Tisch gewählt. Dieser Tisch ist 73 cm hoch und hat eine Tischfläche von 75 x 75 cm. Dieser Tisch entspricht einem normalen Esszimmertisch und bietet daher optimale Versuchskonditionen. Die Wahl fiel auf diesen Tisch, da ein Serviceroboter in der Lage sein sollte einen normalen Esszimmertisch abzuräumen.

Abbildung 5.2.1: Objekte auf großem Tisch

Es wird versucht, mehrere Becher von einem mit zusätzlichen Hindernissen gedeckten Tisch abzuräumen. Dabei soll keine Kollision mit den Hindernissen auftreten. Das genaue Szenario ist in Abbildung 5.2.1 zu sehen.

5.2.2. Kleiner Tisch

In diesem Test wurde ein kleiner Tisch verwendet. Dieser Tisch ist 45 cm hoch und hat eine Tischfläche von 55 x 55 cm. Diese Tischhöhe entspricht ungefähr der Höhe eines Wohnzimmertisches. Deswegen ist auch dieser Tisch ein gutes Szenario, da ein Serviceroboter ebenfalls in der Lage sein sollte, einen Wohnzimmertisch abzuräumen.

Dieses Szenario beinhaltet mehrere Becher und Hindernisse auf dem Tisch. Es wird versucht, diese abzuräumen. Auch hier sollen Kollisionen mit den Hindernissen vermieden werden. Eine genaue Anordnung der Objekte ist in Abbildung 5.2.2 zu sehen.

Abbildung 5.2.2: Objekte auf kleinem Tisch

5.2.3. Komplexes Szenario

Dieses Szenario soll zeigen, dass Pfadplanung und Objekterkennung in der Lage sind, auch in einer sehr komplizierten und mit vielen Objekten versehenen Umgebung zu Recht zu kommen.

Abbildung 5.2.3: Komplexes Szenario auf großem Tisch

Alle Becher auf dem Tisch sollen, soweit es technisch möglich ist, abgeräumt werden. Für die genaue Betrachtung des Szenarios kann Abbildung 5.2.3 herangezogen werden.

5.2.4. Abweichung der Position von Laserscanner und Katana-Arm

Dieser Test soll die Genauigkeit der Übereinstimmung zwischen dem Laserscan und dem Katana-Arm zeigen. Hierzu wurde zuerst die maximale Arbeitsfläche des Katanas ausgemessen. Diese ist in Abbildung 5.2.4 mit Klebeband markiert zu sehen. Für diesen Test wird nur ein Becher verwendet. Dieser Becher wird zuerst von einem Laserscan erfasst und anschließend gegriffen. Danach wird dieser an eine zufällige Position innerhalb des Greifbereiches gesetzt. Dieser Vorgang wird mehrere Male wiederholt.

Abbildung 5.2.4: Greifreichweite des Katana-Arm

5.2.5. Pfadplanungsalgorithmen

Dieser Test misst die Berechnungszeiten der verschiedenen Pfadplanungsalgorithmen, die OpenRAVE mitliefert. Dabei wurde ein Becher auf den großen Tisch gestellt und der Pfad zu diesem Becher und zurück in die Ausgangsstellung berechnet. Es wurden nur die Zeiten gemessen, die der Algorithmus zum Berechnen des Pfades benötigt. Zudem wurde auch festgehalten, ob überhaupt ein Pfad berechnet werden konnte. Die verwendeten Algorithmen waren BasicRRT, rBiRRT und RA*. Der BiSpace-Algorithmus konnte nicht verwendet werden, da er bei der Pfadplanung regelmäßig mit diversen Fehlermeldungen abstürzte.

5.3. Bewertung der erzielten Ergebnisse

5.3.1. Großer Tisch

Auf dem großen Tisch wurden drei Becher und ein Hindernis platziert. Es wurden alle drei Becher, ohne mit dem Hindernis zu kollidieren, abgeräumt. Somit ist das Agieren des Roboterarmes auf einem Esszimmertisch möglich.

5.3.2. Kleiner Tisch

Der kleine Tisch wurde mit drei Bechern und einem Hindernis gedeckt. Es konnten alle drei Becher abgeräumt werden. Dabei gab es keine Kollision mit dem Hindernis. Dieses zeigt, dass der Roboterarm in der Lage ist, auf einem Wohnzimmertisch zu agieren.

5.3.3. Komplexes Szenario

Für das komplexe Szenario wurden drei Hindernisse und drei Becher verwendet. Es konnten zwei von drei Bechern abgeräumt werden. Der dritte Becher, wie auch in Abbildung 5.3.1 zu sehen ist, konnte nicht abgeräumt werden. Dies ist zum einen auf das Objekt direkt im Weg zum Becher zurückzuführen und zum anderen auf die Entfernung zum Objekt. Durch das davor stehende Objekt kann der Becher nicht waagerecht gegriffen werden; deshalb wird versucht, den Becher in einem Winkel zu greifen. Dies führt aber ebenfalls nicht zu einer Lösung, da der Becher für dieses Vorgehen zu weit entfernt ist.

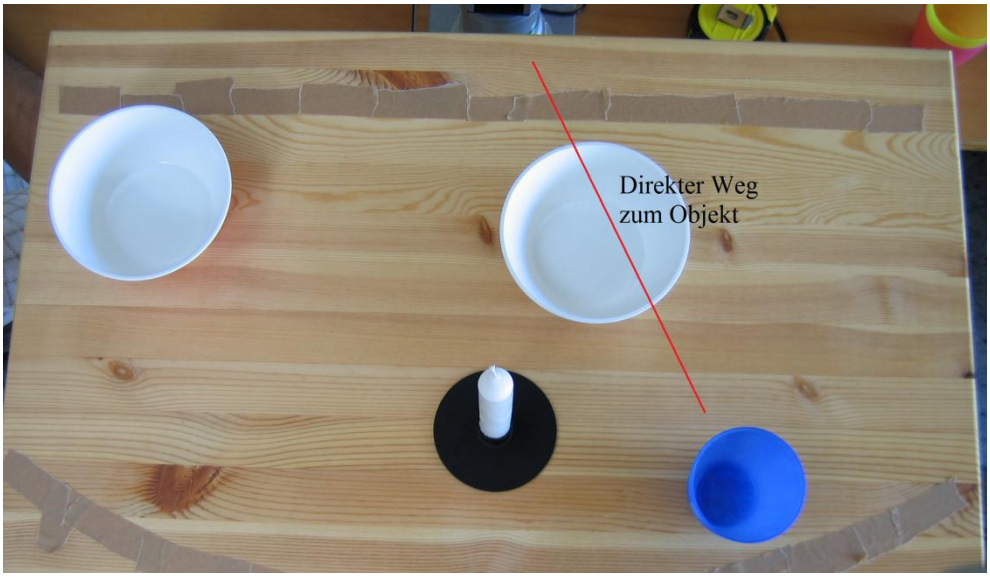

Abbildung 5.3.1: Komplexes Szenario nach dem Abräumen

5.3.4. Abweichung der Position von Laserscanner und Katana-Arm

In Abbildung 5.3.2 kann die Abweichung von der gemessenen Laserposition zu der berechneten zufälligen Position, jeweils für die X-Achse und Y-Achse, gesehen werden. Anhand dieses Diagramms ist zu sehen, dass die Abweichung auf der Y-Achse im Durchschnitt geringer ausfällt als die auf der X-Achse. Dies lässt sich auf den nicht perfekten Greifvorgang zurückführen. Beim Greifvorgang kann es nämlich vorkommen, dass das Objekt nicht immer genau gleich gegriffen wird. Dieses macht sich beim Abstellen des Objektes bemerkbar. Dieser Fehler schlägt sich mehr in der X-Achse nieder als in der Y-Achse, da der Abstand von der Y-Achse wesentlich geringer ist als der von der X-Achse. Der Abstand von der X-Achse beträgt bis zu 50 cm, der Abstand der Y-Achse dagegen nur bis zu maximal 30 cm.

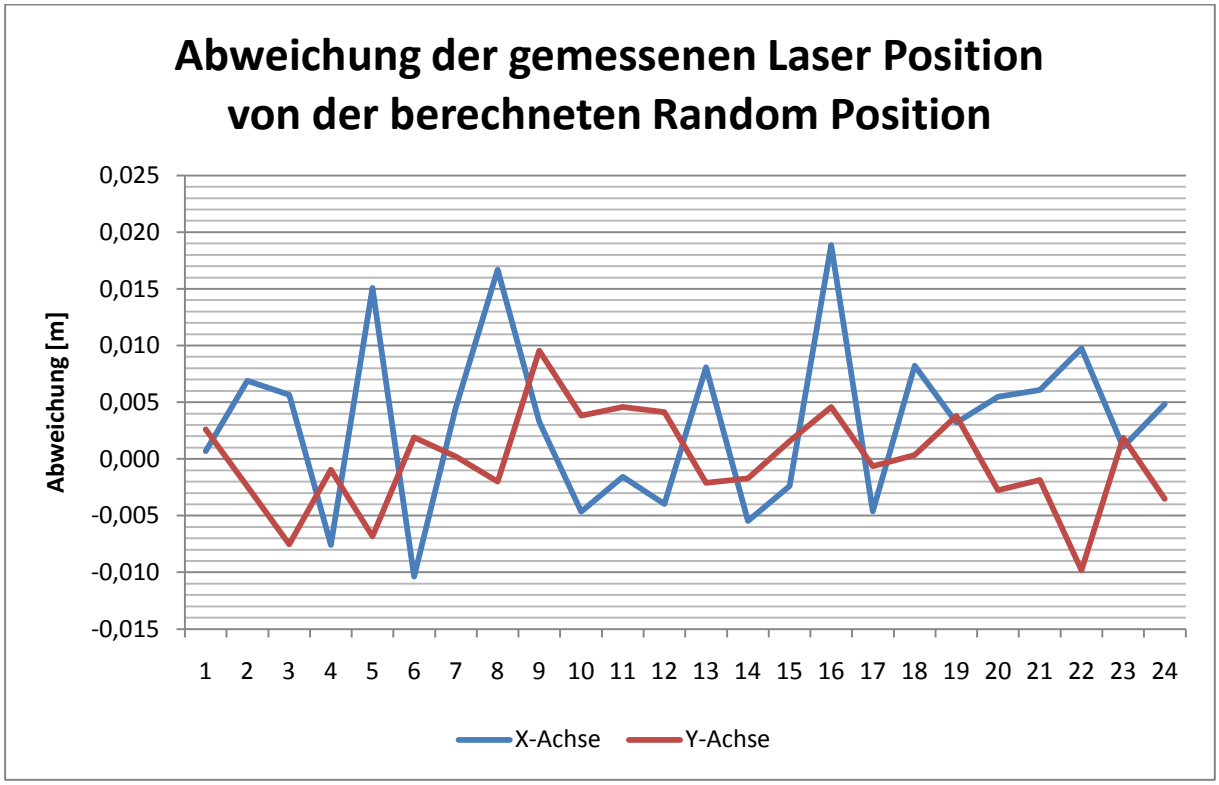

Abbildung 5.3.2: Abweichung der gemessenen Laser-Position von der berechneten zufälligen Position

Die Standardabweichung für die X-Achse beträgt 7,5 mm und für die Y-Achse 4,3mm. Diese Abweichung wurde zusammen mit den zwei verschiedenen Punktepaaren in die Abbildung 5.3.3 eingetragen. In diesem Diagramm ist die genaue Platzierung des Bechers (Laserposition) und die errechnete Position (zufällige Position) zu sehen.

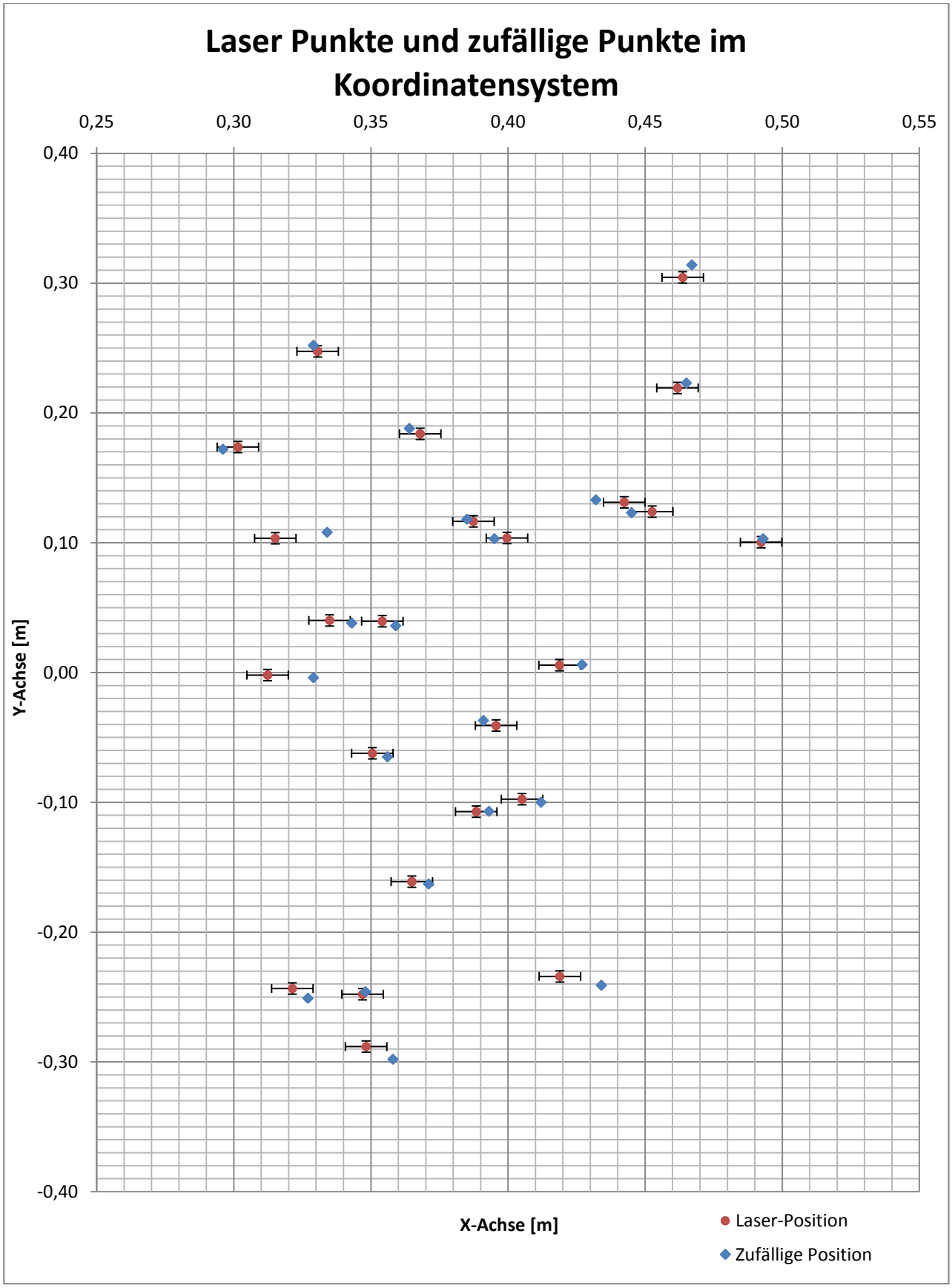

Abbildung 5.3.3: Laserpunkte und zufällige Punkte im Koordinatensystem

5.3.5. Pfadplanungsalgorithmen

In Abbildung 5.3.4 sind die Zeiten des BasicRRT-Algorithmus zu sehen. Zuerst muss festgehalten werden, dass der BasicRRT nur drei erfolgreiche Pfadplanungen durchführen konnte. Diese waren beim zweiten, fünften und zehnten Durchlauf. Alle anderen Durchläufe wurden nach 1000 Iterationen, was ungefähr 130 bis 140 Sekunden entspricht, abgebrochen. Es wäre durchaus möglich gewesen, dass der BasicRRT eine Lösung gefunden hätte; diese hätte einen sinnvollen Zeitbereich aber weit überschritten.

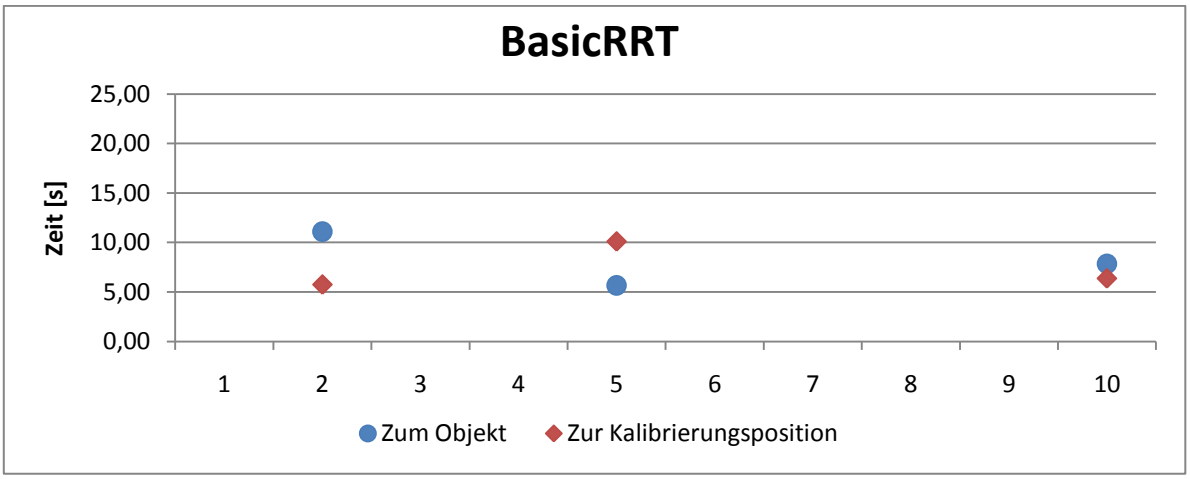

Abbildung 5.3.4: Laufzeiten des BasicRRT-Algorithmus

Der rBiRRT-Algorithmus, dessen Laufzeiten in Abbildung 5.3.5 zu sehen sind, konnte bei allen zehn Durchläufen eine Lösung finden. Es ist zu erkennen, dass die Planung zum Objekt etwas länger in Anspruch nimmt, als zurück zu der Kalibrierungsposition. Dies ergibt sich aus der Positionierung des Greifers um das zu greifende Objekt herum.

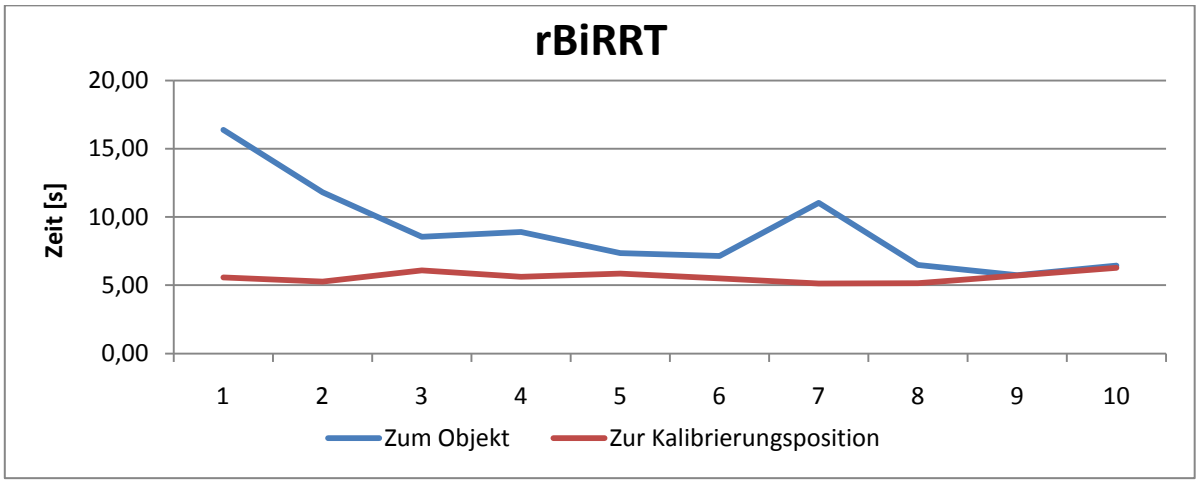

Abbildung 5.3.5: Laufzeiten des rBiRRT-Algorithmus

In Abbildung 5.3.6 sind die Zeiten des RA*-Algorithmus zu sehen. Dabei ist zu beachten, dass der RA* keine erfolgreiche Pfadplanung durchführen konnte. Der Algorithmus terminierte selber, nach den unten aufgeführten Zeiten.

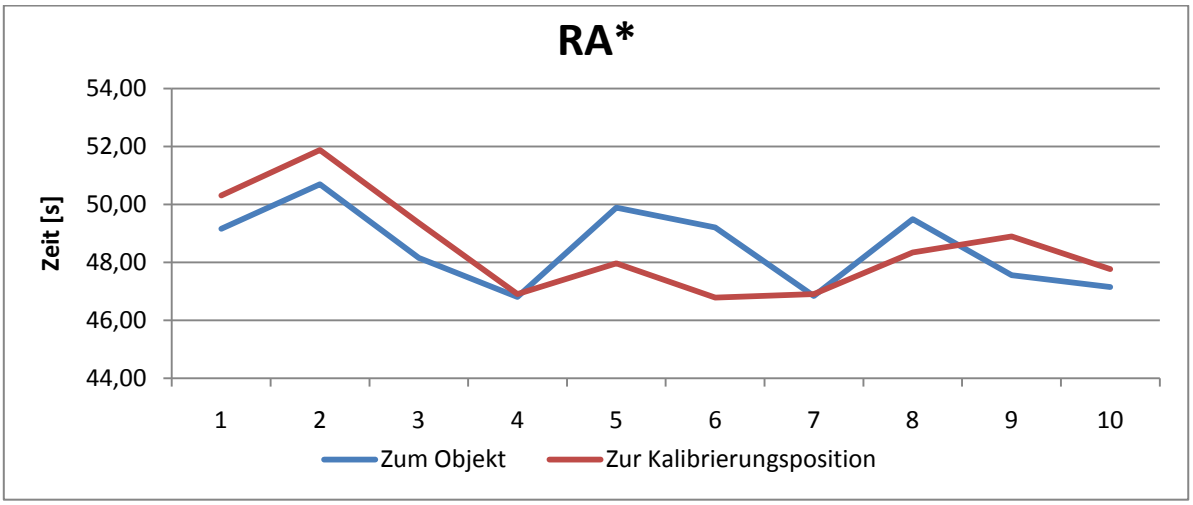

Abbildung 5.3.6: Laufzeiten des RA*-Algorithmus

Aus diesen drei Diagrammen kann erkannt werden, dass der rBiRRT-Algorithmus der schnellste und zielsicherste ist. Der BasicRRT ist für den Alltagsgebrauch nicht einzusetzen, er braucht zum Finden einer Lösung viel zu lange und es kann vorkommen, dass er gar keine Lösung findet. Der RA* findet gar keine Lösung und terminiert selber, dieses lässt einen Einsatz des Algorithmus nicht zu. Dennoch ist zu beachten, dass der RA*-Algorithmus verschiedene Parameter benötigt, wie in Punkt 3.2.3 beschrieben wurde. Diese Parameter wurden bei der Durchführung mit den Standardeinstellungen initialisiert. Bei eventuell anderer Einstellung der Parameter könnte es durchaus passieren, das der RA* ein sehr gutes Ergebnis liefert.

5.4. Zusammenfassung der Testergebnisse

Die Tabelle 5.4.1 zeigt nochmal zusammengefasst, welche Versuche erfolgreich waren beziehungsweise fehlgeschlagen sind.

Versuchsszenario	Ergebnis
Großer Tisch	✓
Kleiner Tisch	✓
Komplexes Szenario	✓ (eingeschränkt)
Abweichung der Position von Laserscanner und Katana-Arm	✓
Pfadplanungsalgorithmen	✓ (eingeschränkt)

Tabelle 5.4.1: Zusammenfassung der Testergebnisse

Anhand dieser Versuche ist festzuhalten, dass der Roboterarm auf unterschiedlichen Tischhöhen agieren kann. Dieses wurde mit den Szenarien *Großer Tisch* und *Kleiner Tisch* gezeigt. Auch wurde mit dem *Komplexen Szenario* gezeigt, dass es durchaus möglich ist in einer Umgebung zu agieren, die mit sehr vielen Hindernissen versehen ist. Auch hat dieses Szenario gezeigt, dass der Aktionsbereich des Katana-Arms eingeschränkt ist. Objekte, bei denen ein Hindernis im direkten Weg zum Katana steht, können nur eingeschränkt gegriffen werden. Die Hindernisse dürfen nicht zu hoch sein, da über das Hindernis gegriffen werden muss, und das zu greifende Objekt darf nicht zu weit entfernt sein, damit es im Winkel gegriffen werden kann. Nur dann ist ein erfolgreicher Greifvorgang möglich.

Durch den *Abweichungstest* konnte festgestellt werden, dass das Zusammenspiel zwischen Laserscanner und Katana-Arm sehr gut ist. Es ist möglich, einen Becher auch nach mehrmaligem Umstellen in einem akzeptablen Bereich wieder zu platzieren. Diese Ungenauigkeit von nicht mal 10 mm reicht für die Manipulation von Alltagsgegenständen aus. Auch wird durch diesen Test gezeigt, dass die Pfadplanung von OpenRAVE und die Ausführung dieses Pfades auf dem Katana-Arm synchronisiert sind und nicht mit der Zeit voneinander abweichen.

Der *Algorithmen*-Test zeigte, dass der rBiRRT der schnellste und zielsicherste Algorithmus ist. Dennoch muss beachtet werden, dass der RA*-Algorithmus eventuell auch sehr gut eingesetzt werden kann. Es ist durchaus möglich, dass durch die richtige Einstellung der vorhandenen Parameter die Leistung des RA* optimiert werden kann.

Kapitel 6 Zusammenfassung und Ausblick

6.1. Zusammenfassung

In dieser Arbeit wurde ein System entwickelt, welches gleichzeitig aus einfachen Komponenten besteht, aber dennoch sehr zuverlässig und effizient funktioniert. Dieses System ist in der Lage, einfache Alltagsgegenstände wie Becher und Dosen zu manipulieren ohne dabei mit Hindernissen zu kollidieren. Um dieses zu erreichen, wurde auf ein bestehendes Framework (OpenRAVE) gesetzt. Mit OpenRAVE kann die von der Objekterkennungssoftware [WOP09] verarbeitete Umgebung dargestellt werden. Auch ist es möglich, auf Basis dieser Darstellung die kollisionsfreie Pfadplanung mit OpenRAVE durchzuführen. Somit konnte mit der vorhandenen API für den Katana-Arm, der Objekterkennungssoftware und OpenRAVE eine Software erstellt werden, die in der Lage ist, einfache Alltagsgegenstände in einer angemessenen Zeit ohne Kollision zu manipulieren.

6.2. Ausblick

6.2.1. Benutzung von Meshes für die Objektdarstellung

In der jetzigen Ausarbeitung werden die Alltagsgegenstände als einfache Boxen angenommen. Diese Darstellung ist sehr einfach, hat aber viele Nachteile. In einer eingescannten Umgebung kann nicht mehr erkannt werden, was diese einfachen Boxen in Wirklichkeit repräsentieren. Für Hindernisse ist es irrelevant ob eine Kerze eine Kerze oder ein großes Rechteck ist. In diesem Fall ist es sogar eher von Vorteil, da die Vergrößerung mehr Sicherheit in der Pfadplanung bringt. Aber gerade für ein Objekt von Interesse, also ein greifbares Objekt, macht es Sinn zu wissen, wie dieses Objekt im Detail aussieht.

Die Verwendung von Meshes[1] würde die Darstellung von Objekten detailgetreuer machen. Dies würde zum einen für den Betrachter von Vorteil sein, aber vor allem würde der Greifvorgang davon profitieren. Es wäre dann nämlich möglich, das in OpenRAVE eingebaute Grasping zu verwenden. Auch die Pfadplanung würde dann noch exakter funktionieren. Die genaue Verwendung und die Vorteile von Grasping werden in Punkt 6.2.2 beschrieben.

Dennoch hat die Verwendung von Meshes auch Nachteile, so sind Boxen sehr einfach in der Handhabung und können sehr leicht in einer Kollisionsberechnung errechnet werden. Meshes

[1] Meshes oder Meshing ist die Annäherung der wirklichen Körper durch Dreiecke oder Vierecke.

dagegen sind sehr komplex und detailliert. Dieses würde die Kollisionsberechnung und somit die Pfadplanung verlangsamen.

Es wäre abzuwägen, ob ein besseres Greifen oder eine schnelle Pfadplanung eingesetzt werden soll. Zudem müsste auch der genaue Geschwindigkeitsverlust der Pfadplanung gemessen werden.

6.2.2. Verwendung von Grasping in OpenRAVE

OpenRAVE bietet ein erweitertes Grasping an. Dieses schließt nicht nur den Greifer und öffnet ihn, sondern berechnet mögliche Greifpositionen. Dazu muss zuerst das Objekt auf mögliche Greifpositionen analysiert werden. Dieses ist in Abbildung 6.2.1 sehr gut zu sehen. Die roten Striche auf dem Becher geben mögliche Positionen an, an denen der Greifer ansetzen kann.

Danach wird eine Grasp-Tabelle erzeugt. Diese besteht aus mehreren Paaren, wobei ein Paar aus der Endeffektor-Position und dem Zielobjekt besteht. Eine solche Position ist in Abbildung 6.2.2 zu sehen. Dort ist eine der möglichen Greifpositionen dargestellt. Die Erzeugung der Grasp-Tabelle kann zum Teil sehr viel Zeit in Anspruch nehmen. Die Zeit kann durch die Anzahl möglicher Greifpositionen eingeschränkt werden. Für 3000 mögliche Greifpositionen dauert das Erzeugen der Tabelle ungefähr 60 Minuten.

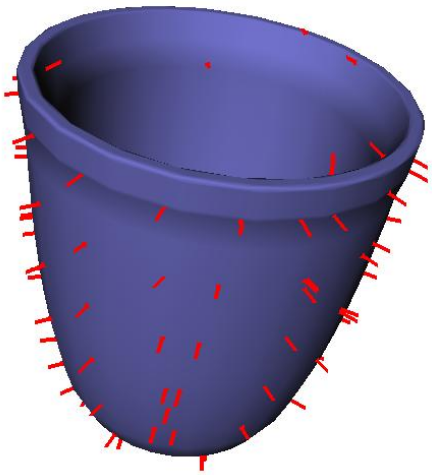

Abbildung 6.2.1: Becher mit seinen möglichen Greifpositionen (Quelle [OPR09])

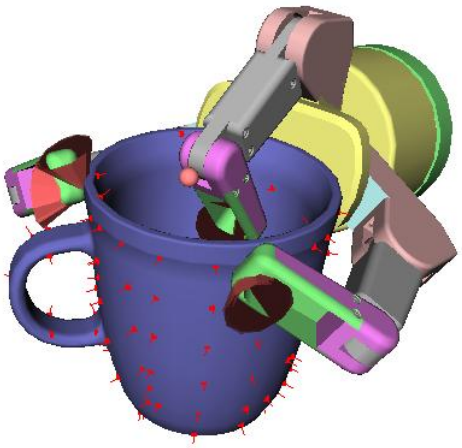

Abbildung 6.2.2: Becher wie er von einer Roboterhand gegriffen werden kann (Quelle [OPR09])

Eine Greifposition wird wie folgend erzeugt. Der Endeffektor wird in eine Ausgangsposition gebracht, die nahe an dem zu greifenden Objekt liegt. Danach wird langsam an das Objekt herangetastet. Sobald der Endeffektor kollidieren würde, werden die Greifer geschlossen. Es

wird ebenfalls die Kollision mit den Greifern und dem Objekt berechnet. Zusätzlich wird auch die Druckstärke mit einbezogen. Somit werden nur Greifpositionen in der Tabelle abgespeichert, die zu einem festen und stabilen Greifen führen.

Der Einsatz von Grasping setzt Meshes voraus. Grasping macht nur Sinn, wenn das durch Meshing erzeugte Objekt eine saubere Kontur besitzt und es der Wirklichkeit entspricht. Zudem muss evaluiert werden, inwiefern es für den Katanagreifer Sinn ergibt, dieses zu benutzen. Auch muss das Erzeugen der Grasp-Tabellen untersucht werden: wie viele mögliche Greifkombinationen sind sinnvoll und wie viel Zeit nimmt die Erzeugung in Anspruch? Zudem sollte auch der Zeitpunkt der Erzeugung bedacht werden. Ein guter Zeitpunkt wäre zum Beispiel beim Einlernen eines neuen Objektes.

Die Benutzung von Grasping würde die Genauigkeit und Robustheit der bisherigen Methode um einiges verbessern. Aber es würde bei der Ausführung auch mehr Zeit in Anspruch genommen werden. Es muss abgewägt werden, ob es Sinn macht Grasping mit dem Katana-Arm einzusetzen und wie lange die Erstellung der Grasp-Tabellen benötigt.

6.2.3. Multithreading

In der jetzigen Ausarbeitung ist der Programmfluss sequenziell, dieses erleichterte das Programmieren und bringt keine komplizierten Synchronisationsprobleme mit sich. Damit ergibt sich natürlich das Problem, dass während der Bewegung des Roboterarmes keine anderen Aufgaben ausgeführt werden konnten. Während der Bewegung könnte schon die nächste Pfadplanung stattfinden. Dies würde die Wartezeit zwischen den Bewegungen des Roboterarms enorm verkürzen.

Derzeit besteht die Software aus zwei Threads. Der erste Thread, der GUI-Thread, beinhaltet die grafische Oberfläche von OpenRAVE. Der zweite Thread, der Main-Thread, verbindet alle bestehenden Komponenten und steuert den Programmfluss. Es könnte nun zusätzlich ein neuer Thread hinzugefügt werden, der den Katana-Arm beinhaltet. Somit wäre es möglich, dass vom Main-Thread Funktionen der Katana-API aufgerufen werden ohne das dieser blockiert wird. Der Main-Thread könnte dann gleich den nächsten Befehl an OpenRAVE schicken, in dem die Pfadplanung für das nächste Segment berechnet würde.

Diese Vorgehensweise würde einen enormen Geschwindigkeitsschub mit sich bringen. Der Ablauf des ganzen Szenarios würde dann flüssiger und alltagtauglicher werden. Dennoch ist

der Aufwand zu beachten, den ein neuer Thread mit sich bringt. Zusätzlich müsste an die Synchronisation gedacht werden. Auch müsste die exakte Geschwindigkeitssteigerung gemessen werden. Desweitern muss auch untersucht werden, ob wirklich jeder Aufruf des Katanas parallel benutzt werden kann beziehungsweise ob es Funktionen gibt, auf deren Terminierung gewartet werden muss.

Abbildungsverzeichnis

Tabellenverzeichnis

Codeauszugsverzeichnis

Literaturverzeichnis

[BIS08] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "BiSpace Planning: Concurrent Multi-Space Exploration," 2008.

[CAR09] CARMEN-Team. (2009, Jun.) *CARMEN.* Homepage http://carmen.sourceforge.net/home.html

[CLA08] Jet Propulsion Laboratory, NASA Ames Research Center, Carnegie Mellon, University of Minnesota. (2008, Sep.) *CLARAty.* Homepage http://claraty.jpl.nasa.gov/man/overview/index.php

[KIN09] Kineo CAM Corporation. (2009) *Automatic motion and path planning software development kit.* Homepage http://www.kineocam.com/kineoworks-library.php

[KUK09] KUKA Robotics Corp. (2009) *KUKA Industrial Robots.* Homepage http://www.kuka-robotics.com/usa/en/

[MPK06] Stanford University. (2006,) *MPK - Motion Planning Kit.* Homepage http://robotics.stanford.edu/~mitul/mpk/

[MRP09] J. L. B. Claraco, "Development of Scientific Applications with the Mobile Robot Programming Toolkit," University of Malaga, 2009.

[MSR09] Microsoft Corporation. (2009) *Microsoft Robotics.* Homepage http://msdn.microsoft.com/en-us/robotics/default.aspx

[NEU09] Neuronics AG. (2009) *Neuronics AG - Downloads.* Homepage http://www.neuronics.ch/cms_de/web/index.php?id=387&s=downloads

[OOP09] Rice University. (2009) *OOPSMP: An Object-Oriented Programming System for Motion Planning, Kavraki Lab.* Homepage http://www.kavrakilab.org/OOPSMP/index.html

[OPR08] R. Diankov and J. Kuffner, "OpenRAVE: A Planning Architecture for Autonomous Robotics," Robotics Institute Carnegie Mellon University, 2008.

[OPR09] R. Diankov. (2009,) *Main Page - OpenRAVE.* Homepage http://openrave.programmingvision.com

[RAS07] R. Diankov and J. Kuffner, "Randomized Statistical Path Planning," 2007.

[RRT00] S. M. LaValle and J. J. Kuffner, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," 2000.

[SKE09] Google Inc. (2009) *Google SketchUp.* Homepage http://sketchup.google.com/

[VPL09] Microsoft Corporation. (2009) *VPL Introduction.* Homepage http://msdn.microsoft.com/en-us/library/bb483088.aspx

[WEB09] Cyberbotics Ltd. (2009, Nov.) *Cyberbotics Webots.* Homepage http://www.cyberbotics.com/products/webots/

[WIK05] Wikipedia. (2005,) *Datei:RoboterKinematik.png.* Homepage http://de.wikipedia.org/wiki/Datei:Roboterkinematik.png

[WIL09] Willow Garage, Inc. (2009, Jul.) *Willow Garage.* Homepage http://www.willowgarage.com/

[WOP09] M. Wopfner, "Erkennen und Greifen von Alltagsgegenständen mittels Katana Manipulatorarm: Umgebungserkennung mittels 3D Entfernungsdaten," Hochschule Ulm Bachelorarbeit, 2009.