Bildgestütztes Teach-In eines mobilen Manipulators in einer virtuellen Umgebung

Von der Fakultät für Elektrotechnik und Informationstechnik
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Diplomatouchos Electrical Engineer

Alexandros Matsikis

aus Thessaloniki, Griechenland

Berichter: Universitätsprofessor Dr.-Ing. Karl-Friedrich Kraiss
Universitätsprofessor Dr.-Ing. Uwe D. Hanebeck

Tag der mündlichen Prüfung: 21. Januar 2005

Kurzfassung

Der Anwendungsbereich der Robotik verschiebt sich von Industrieapplikationen auf die Unterstützung von Menschen in natürlichen Umgebungen. Zur Erfüllung der neuen Aufgaben wird oft ein Roboterarm auf einer mobilen Plattform montiert, der Objekte bildgestützt greifen, betätigen und transportieren soll. Diese so genannte mobilen Manipulatoren sind noch auf den Laborbereich beschränkt, da die Realisierung von Verfahren, die aus Bildinformation entsprechende Manipulatorbewegungen ableiten, keine triviale Aufgabe darstellt. Existierende Systeme sind in der Lage einfache Objekte zu greifen und an einen anderen Ort zu transportieren, berücksichtigen jedoch keine Hindernisse, die den direkten Greifvorgang behindern.

Gegenstand der Arbeit ist die Entwicklung einer Architektur für mobile Manipulation, die das bildgestützte Greifen von Objekten in einer teilweise bekannten Umgebung unter Berücksichtigung von Hindernissen ermöglicht. Zugleich wird ein Ansatz vorgestellt, um das gewünschte, bildgestützte Greifen anhand von Aufnahmen virtueller Kameras aus einer virtuellen Umgebung zu erlernen. Zur Evaluierung dient ein Pick-and-Place-Szenario. Der mobile Manipulator muss ein Objekt, das eine bestimmte Greifrichtung erfordert, von Hindernissen umgeben und arbiträr auf einem Tisch positioniert ist, greifen und an einen anderen Ort transportieren.

Die Schwierigkeit der Aufgabenstellung liegt in der zunächst unbekannten Position von Zielobjekt und Hindernissen, die durch Bildverarbeitung detektiert und lokalisiert werden müssen. Die folgende, bildgestützte Ausführung des Greifvorgangs erfordert, dass sich der Manipulator dem Objekt aus einer optimalen Richtung annähert, wobei auf neue Bildinformation direkt zu reagieren ist, um Kollisionen zu vermeiden. Das System soll echtzeitfähig und in seiner Planung flexibel sein, um bei misslungenen Aktionen alternative Handlungsweisen zu wählen.

Zur Lösung ist eine erweiterte drei-Ebenen Architektur realisiert. Die unterste, reaktive Ebene beinhaltet spezialisierte Regelkreise, die für eine Reaktion des Roboterarms auf Umgebungsänderungen sorgen. Diese Regelkreise sind explizit programmiert oder mit Neuronalen Netzen (Radial Basis Function Networks und Neuro-Fuzzy) erlernt. Sie setzen die Information der Roboterkameras ein, um den Manipulator zum Ziel zu führen, das Objekt im Sichtfeld der Manipulatorkamera zu halten oder den Greifer bzw. die Manipulator-Segmente vor Kollisionen zu schützen. Soll der mobile Manipulator beim Greifen des Zielobjektes Hindernisse berücksichtigen, müssen Zielführung, Zielverfolgung und Kollisionsvermeidung aktiviert und koordiniert werden. Bayesian Belief Netzwerke realisieren in der mittleren Ebene diesen Koordinationsmechanismus. Sie bestimmen die Anwendbarkeit der Komponenten der reaktiven Ebene und ermitteln die nächste Manipulatorbewegung anhand der aktuellen Bildinformation und des auszuführenden Planungsschrittes. Die hierarchisch höchste, deliberative Ebene arbeitet strategische Pläne aus und zerlegt eine Aufgabe in eine Sequenz von Planungsschritten für den Manipulator und die mobile Plattform. Hier werden auch alternative Handlungsweisen und Greifpositionen bestimmt, falls ein Planungsschritt misslingt. Die Komponenten der reaktiven Ebene

und der Koordinationsmechanismus werden in einer virtuellen Umgebung mit Modellen der Objekte, des Roboters und Aufnahmen von virtuellen Kameras trainiert. Zwei bildgestützte Alternativen zum positionsbasierten Teach-In sind für das Training in der virtuellen Umgebung realisiert: das algorithmische Teach-In für die Zielführung und das stochastische Teach-In für die Kollisionsvermeidung und den Koordinationsmechanismus.

Mit dieser Arbeit ist ein System zur mobilen Manipulation in teilweise bekanntenen Umgebung unter Berücksichtigung von Hindernissen entstanden. Die realisierte 3T-Architektur ermöglicht den parallelen Einsatz reaktiver zielführender und kollisionsvermeidender Verfahren; bisher wurde nur die sequentielle Ausführung der Komponenten der reaktiven Ebene implementiert. Weiterhin wurde der Einsatz virtueller Umgebungen zum Teach-In der reaktiven, bildgestützten Komponenten sowie des Koordinationsmechanismus untersucht. Die trainierten Verfahren kommen erfolgreich auf dem realen mobilen Manipulator zum Einsatz.

Danksagung

Die vorliegende Arbeit ist im Rahmen meiner Tätigkeit als DAAD-Stipendiat und anschließend als wissenschaftlicher Angestellter am Lehrstuhl für Technische Informatik der Rheinisch-Westfälischen Technischen Hochschule Aachen entstanden.

Mein besonderer Dank gilt Herrn Univ.-Prof. Dr.-Ing. Karl-Friedrich Kraiss für die Betreuung und Förderung der Arbeit, für seine Anregungen und fachlichen Diskussionen sowie für die Unterstützung und Hilfe während meiner Tätigkeit am Lehrstuhl. Herrn Univ.-Prof. Dr.-Ing. Uwe Hanebeck danke ich für das Interesse an meiner Arbeit und die freundliche Übernahme des Koreferats.

Meine Tätigkeit am Lehrstuhl wurde im besonderen Maße durch das gute Arbeitsklima mit meinen Kollegen geprägt, die das Arbeiten am Lehrstuhl für Technische Informatik zu Spaß gemacht haben. Danksagen möchte insbesondere den Kollegen der Robotik-Gruppe Frank Broicher, Claudia Gönner, Michael Schmitt und Jost Wunderlich. Einen Besonderen Dank gilt meinen Kollegen Pablo Alvarado für die Betreuung und Entwicklung der Bildverarbeitungsbibliothek des Lehrstuhls für Technische Informatik, sowie an allen Kollegen, die dabei mitgewirkt haben. Weiterhin möchte ich mich auch bei Sascha Ferrein für die viele Ratschläge im Bereich der Planung bedanken.

Herzlich danken möchte ich auch meinen Diplomanden Michalis Lazaridis, Fabian Schulte, Theodoros Zoumpoulidis, Florian Zschocke und Nils Springob, da ohne sie diese Arbeit nicht möglich wäre. Dank gilt auch den wissenschaftlichen Hilfskräften Peter Mathes, Yusuf Kangoez, Stefan Kauke, Stephan Bichmann und Ralf Jennen.

Für die akribische Durchsicht des Manuskriptes sowie für die beim abendlichen Bier erfahrene Aufmunterung in stressigen Zeiten bin ich Gregor Darlinski und Kalex Tröger und Cathrin Alfter sehr verbunden. Ein besonderer Dank gilt meinen guten Freunden und Kollegen Martin Rous und Florian Bley für ihre zahlreichen Anregungen und die konstruktive Kritik innerhalb und auch außerhalb des Lehrstuhls.

Ganz herzlich bedanken möchte ich mich bei meiner Familie, bei meinen Freunden und bei Päivi Harju, die mir durch ihre verständnisvolle und geduldige Unterstützung die Stärke gegeben haben, diese Arbeit auch erfolgreich abzuschließen.

Alexandros Matsikis

Aachen, im Januar 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In der industriellen Automation sind Roboter, insbesondere Manipulatoren, weit verbreitet. Sie dienen der Arbeitserleichterung für den Menschen und führen wiederholt definierte Reihenfolgen von festgelegten Bewegungen in einer bekannten, stationären Umgebung aus. Dabei hat die Forschung in der Robotik technische Fragen, insbesondere des Entwurfs der Mechanik und der Regelung von kinematischen Ketten, grundlegend untersucht und beantwortet [Dil98a]. Inzwischen sind stationäre Manipulatoren fester Bestandteil in Produktionsanlagen in der Industrie.

Allerdings verschiebt sich der Anwendungsbereich der Robotik von der reinen industriellen Automation hin zur Unterstützung von Menschen in natürlichen Umgebungen. In naher Zukunft sollen Roboter als Assistenten für Dienstleistungsanwendungen (*Service Tasks*) zum Einsatz kommen und in direkter Nähe von Menschen arbeiten [Dil98b]. Mögliche Einsatzszenarien umfassen unter anderem Assistenzsysteme für ältere Personen, die Unterstützung von Behinderten, die Inspektion in für den Menschen schädlichen Umgebungen sowie den Einsatz als Unterhaltungs-, Reinigungs- und Überwachungroboter.

Diese neuen Anwendungsbereiche haben auch die Forschungsschwerpunkte der Robotik verändert. Damit die Roboter in der Gesellschaft Akzeptanz finden und für Unterstützungsaufgaben in alltäglichen, teilweise bekannten Umgebungen eingesetzt werden, müssen sie frei navigieren und mit ihrer Umgebung interagieren ohne den Einsatz spezieller oder aufwendiger Infrastruktur. Ihre Funktion als Assistenzsystem erfordert, dass sie die Einsatzumgebung durch ihre Sensorik selbst erfassen, die Sensordaten interpretieren und entsprechend ihre Aktionen planen. Weiterhin muss ein Roboter auf unvorhergesehene Ereignisse reagieren und seine Pläne anhand der neu entstandenen Situation in der Umgebung anpassen. Je anspruchsvoller die durchzuführende Aufgabe und je dynamischer die Umgebung, desto wichtiger wird eine detaillierte Wahrnehmung und ein situationsabhängiges, intelligentes Verhalten des Roboters.

1.1 Mobile Manipulation

In den letzten Jahren wurde der Forschungsbereich der autonomen Navigation von Robotern intensiv untersucht. Die ersten mobilen Systeme sind als kommerziell erhältliche Produkte erschienen, wie beispielsweise die Staubsaugerroboter der Firmen iRobot [DF02], Electrolux [Ele01] und Kärcher [Kär03]. Eine mobile Plattform kann jedoch nicht mit Objekten interagieren. Sollen mobile Roboter

für Unterstützungsaufgaben in Umgebungen ohne speziell konzipierte Infrastruktur eingesetzt werden, so müssen sie zielstrebig Gegenstände greifen und transportieren, Türen und Schränke öffnen und schließen bzw. Schalter betätigen.

Um Objekte manipulieren zu können, wird ein Roboterarm auf die autonome mobile Plattform montiert. Die ersten so genannten mobilen Manipulatoren [BFG+97], [Bro90], [Has96] setzten abstandsmessende Sensorik zur Ansteuerung der Aktuatorik ein. In aktuellen Forschungsarbeiten kommen jedoch hauptsächlich Kameras aufgrund ihrer Leistungsfähigkeit als Sensoren zum Einsatz, wie in [WN01], [MA99] oder [Kra01]. Kameras bieten den Vorteil, Information bezüglich der räumlichen Struktur der Umgebung zu ermitteln, und können dadurch zur Lokalisierung und zum Greifen von Objekten im Raum verwendet werden. Sie registrieren schnell Änderungen im Arbeitsbereich des Roboters und ermöglichen eine direkte Reaktion der Aktuatoren. Zusätzlich sind Kameras vielfältig anwendbar; sie können sowohl zur Identifikation und Positionsbestimmung von Objekten als auch zur Steuerung bzw. Regelung der Bewegungen eingesetzt werden. Nachteilig wirkt sich dagegen die Anfälligkeit der Bildqualität für Beleuchtungsunterschiede in einer alltäglichen Büroumgebung aus.

Kragic [Kra01] teilt die Vorgangsweise von mobilen Manipulatoren während der Ausführung einer Aufgabe in bestimmten Phasen auf. Dabei muss der Roboter kollisionsfrei zu einem Zielort navigieren, die Objekte am Einsatzort robust detektieren und lokalisieren und eine geeignete Anfangsposition für den Greifvorgang bestimmen. Anschließend soll er das Zielobjekt, auch wenn von Hindernissen umgeben, mit dem Manipulator erreichen und entsprechend seiner Aufgabenstellung greifen oder betätigen. Dementsprechend können die wichtigsten Forschungsthemen der mobilen Manipulation in den folgenden Punkten zusammengefasst werden:

- Aufteilung der Aufgabe in Planungsschritte für mobile Plattform und Manipulator (Aktionsplanung),
- Koordination von mobiler Plattform und Roboterarm,
- Pfadplanung und Navigation der Plattform,
- Objektdetektion und Objektlokalisierung,
- Berechnung eines robusten Griffes für das Objekt,
- Bildgestützte Führung des Roboterarmes an die Greifposition und
- schnelle Neuplanung im Fall eines Misserfolgs oder einer ungeeigneten Greifposition.

Besonders hohe Ansprüche stellt dabei die bildgestützte Führung des Roboterarmes zur Greifposition an ein Zielobjekt. Einerseits müssen während des Greifvorgangs Kollisionen mit Hindernissen in der näheren Umgebung des Manipulators vermieden werden. Dazu sollte der Roboterarm auf neue Sensorinformationen, die die Position von detektierten Hindernissen angeben, reagieren und seine Bewegungen entsprechend anpassen. Andererseits erfordert die Implementierung von bildgestützten Verfahren die Realisierung einer nicht-linearen Abbildung der Bilddaten zu Roboterbewegungen sowie eine genaue Kenntnis der Kinematik und Dynamik des Manipulators. Die Verfahren müssen den Roboterarm zu einer objektspezifischen Endposition und Orientierung am Zielobjekt führen, um anschließend das Objekt robust greifen und effektiv manipulieren zu können. Ist in diesen Verfahren eine bildgestützte Hindernisvermeidung einbezogen, so erhöht sich die Komplexität des Gesamtsystems. Aus diesem Grund beschränken sich viele implementierte Ansätze auf die Entwicklung von Verfahren, die den Manipulator durch Rückkopplung von visuellen Daten zu einem gewünschten Ziel bzw. Objekt führen (*Visual Servoing*) [HHC96], ohne Hindernisse zu berücksichtigen.

1.2 Mobile Manipulation mit Hindernisvermeidung

Gegenstand dieser Arbeit ist die Entwicklung einer Architektur für mobile Manipulation, die das Greifen von Objekten in einer teilweise unbekannten Umgebung unter Berücksichtigung von Hindernissen ermöglicht. Zugleich wird ein Verfahren vorgestellt, um das gewünschte bildgestützte Verhalten des Manipulators anhand von Aufnahmen virtueller Kameras aus einer virtuellen Umgebung zu erlernen (Abbildung 1.1). Dabei werden Modelle des realen Roboters und seiner Kameras in virtuellen Umgebungen eingesetzt, um die entsprechenden bildgestützten Steuerungsalgorithmen zu trainieren und auszuwerten. Nach der Trainings- und Evaluierungsphase findet ein Transfer der erlernten Parameter der Algorithmen auf den realen Roboter statt und das System wird in der realen Umgebung eingesetzt. Dieser lernbasierte Ansatz bietet den Vorteil, dass er die Integration von Wissen, hier die bildgestützte Steuerung des Roboterarms, vereinfacht sowie das Gesamtsystem an wechselnde Umweltbedingungen und Aufgaben leichter anpassen kann.

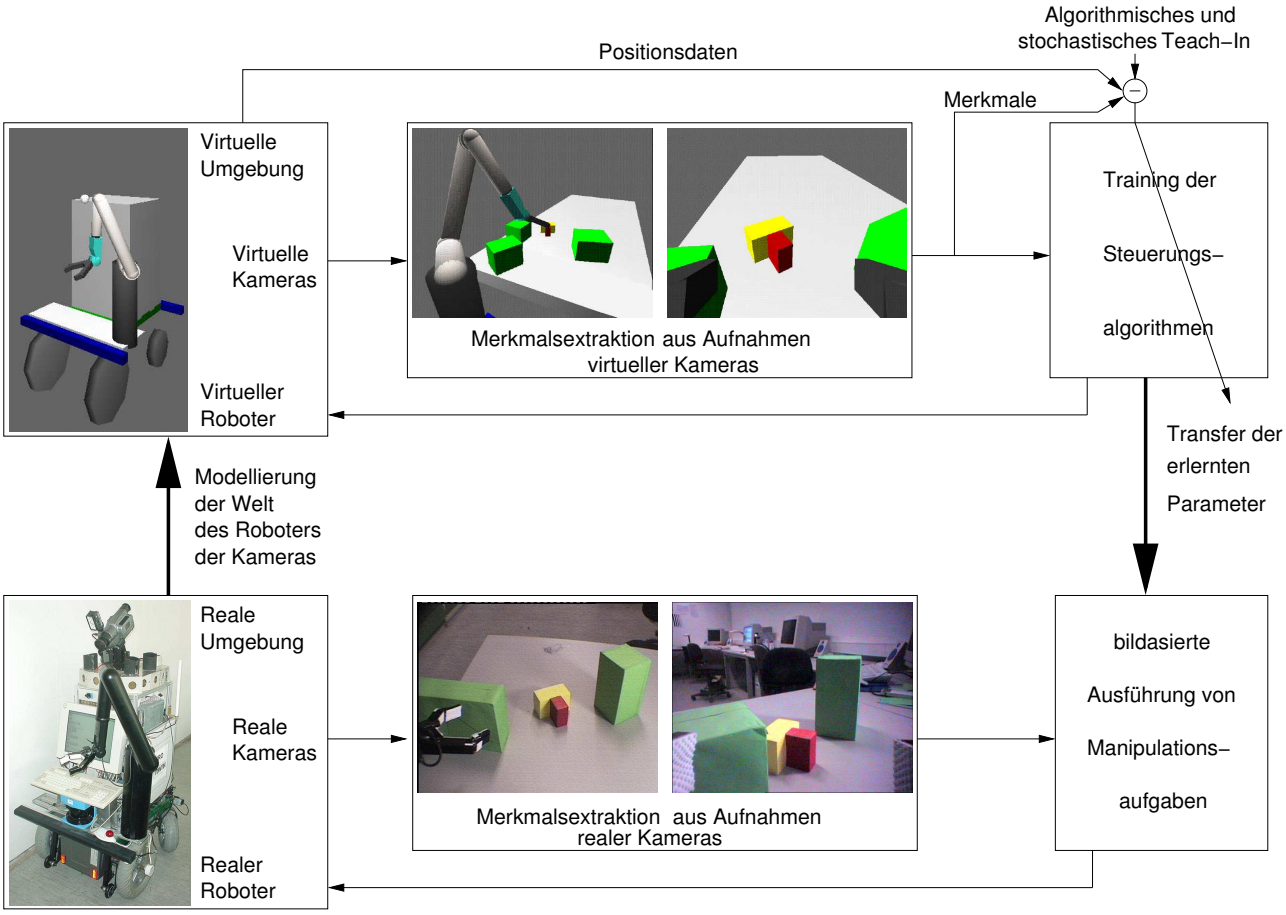


Abbildung 1.1: Anwendung Virtueller Realität als eine Trainings- und Simulationsumgebung.

Zur Evaluierung des implementierten Ansatzes wird eine *Pick-And-Place* Aufgabe als Testszenario verwendet, da ähnliche Tätigkeiten einen Großteil der Aufgaben eines mobilen Manipulators ausmachen. Das Szenario ist folgendermaßen definiert:

> Der mobile Manipulator muss ein Zielobjekt, das eine bestimmte Greifrichtung erfordert und arbiträr auf einem Tisch positioniert ist, greifen, zu einem anderen Ort transportieren,

und dort ablegen. Die Position des Tisches im Raum ist nur zum Teil bekannt und die Umgebung beinhaltet Hindernisse sowohl für den Manipulator als auch für die mobile Plattform.

Der eingesetzte mobile Manipulator TAURO[3] (Abbildung 1.2) besteht aus einer Rollstuhlplattform der Firma Invacare [Inv] und den Manipulator Manus der Firma Exact Dynamics [Dyn]. Die Plattform verfügt über zwei unabhängige Antriebe an den Vorderrädern, während die Hinterräder nicht angetrieben sind. Zur Navigation werden die Positionsdaten aus zwei Drehgeber an den Vorderrädern und die Abstandsmessungen eines Laserscanners verwendet. Der Manipulator verfügt über sechs Gelenke und einen Greifer. Zwei Kameras werden zur Ansteuerung des Roboterarmes eingesetzt. Die eine befindet sich auf dem Greifer in einer so genannten *eye-in-hand* Konfiguration. Eine zweite Kamera ist auf einem Schwenk-Neige-Kamerakopf auf der mobilen Plattform montiert. Da ihre Position von der Manipulatorbewegung nicht beeinflusst wird, handelt es sich um eine *eye-to-hand* Konfiguration.

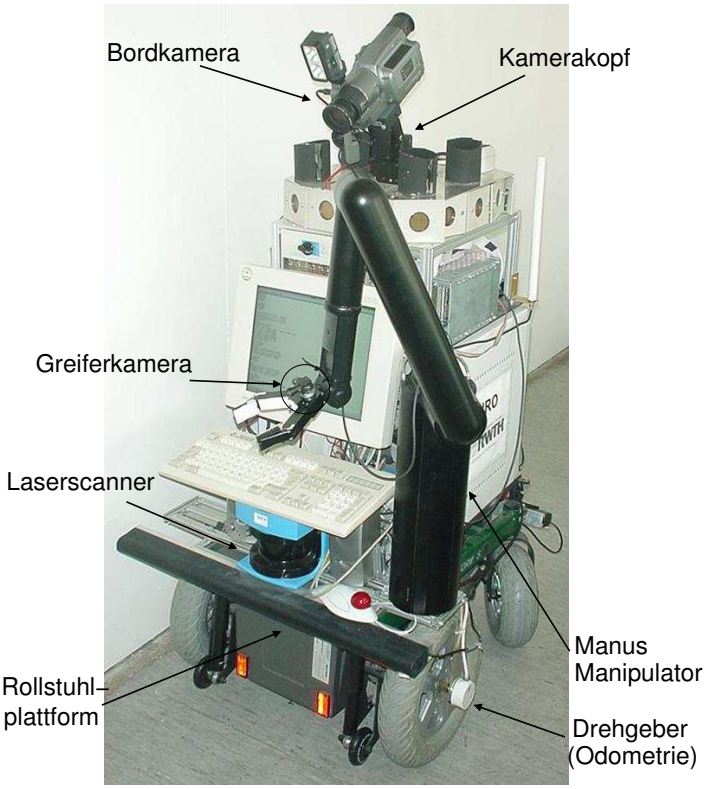


Abbildung 1.2: Der mobile Manipulator TAURO[3].

Die Testobjekte bestehen aus einem Zielobjekt und mehreren Hindernissen unterschiedlicher Größe. Um die Probleme der Bildverarbeitung beherrschbar zu machen, ist das Aussehen der Zielobjekte und Hindernisse so gewählt, dass eine einfache Segmentierung und Erkennung möglich ist. Die Hindernisse sind grüne Quader (Abbildung 1.3) und das Zielobjekt besteht aus einem roten Griff und einer gelben Basis. Die Form des Zielobjektes erfordert eine bestimmte Greifrichtung (Abbildung 1.4, in Richtung des Pfeiles), um einen robusten Griff zu gewährleisten. Diese Form wurde ausgewählt, weil in der Regel der Greifvorgang objektspezifisch sein muss. Eine Türklinke muss etwa von oben, ein Glas oder eine Flasche dagegen von der Seite gegriffen werden. Dabei ist auch der Pfad zur Zielposition von der Form und der Funktionalität des Objektes abhängig.

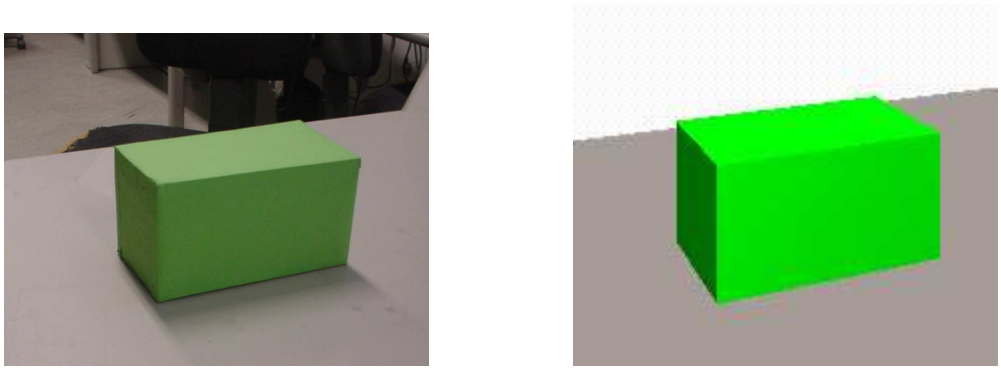


Abbildung 1.3: Hindernisse in der realen (links) und in der virtuellen (rechts) Umgebung.

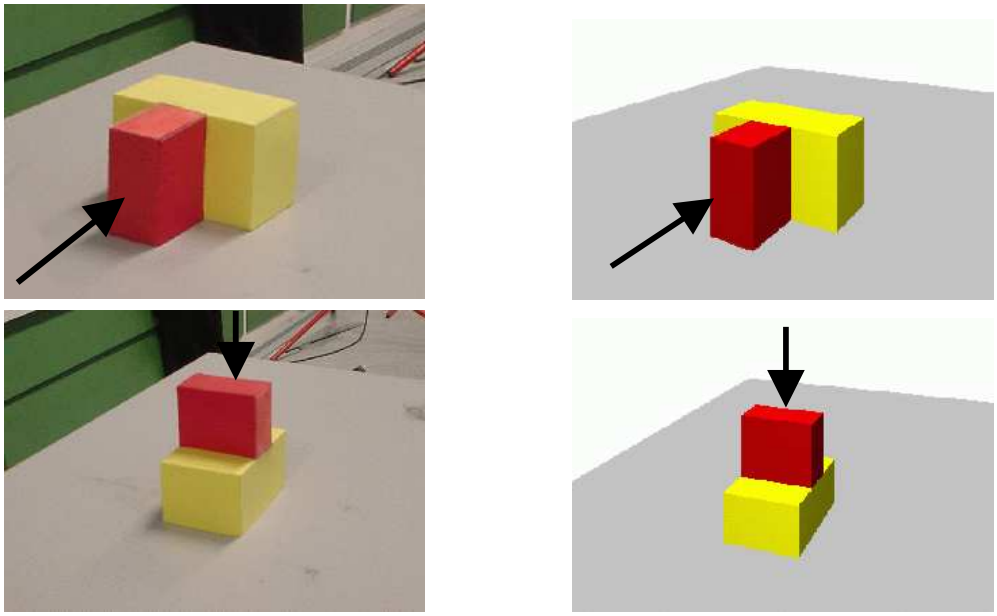


Abbildung 1.4: Zielobjekt in der realen (links) und der virtuellen (rechts) Umgebung und vorgeschriebene Greifrichtung.

Die Schwierigkeit der Aufgabenstellung liegt in der zunächst unbekannten Position des Zielobjektes und der Hindernisse, die den direkten Greifvorgang stören. Da sich die mobile Plattform in einer unbekannten, veränderlichen Umgebung befindet, müssen die Objekte zuerst detektiert und lokalisiert werden. Die folgende, bildgestützte Ausführung des Greifvorgangs erfordert, dass sich der Manipulator dem Objekt aus der erforderlichen Greifrichtung annähert, wobei auf neue Bildinformation direkt zu reagieren ist, um Kollisionen des Greifers oder der Manipulator-Segmente mit Hindernissen zu verhindern. Verdeckungen durch Hindernisse und eine fehlerbehaftete Merkmalsextraktion aus den Bilddaten erschweren die Gesamtaufgabe.

Um diesen Anforderungen entgegenzukommen ist eine komplexere Architektur der zugrunde liegenden Steuerung als bei einem stationären Manipulator in einer statischen Umgebung notwendig. Ein reaktives Verhalten des Manipulators sowie eine voraussehende Planung der Aktionen von mobiler Plattform und Manipulator, um die gestellte Aufgabe anhand der Umgebungsituation zu lösen, sind erforderlich. Zugleich muss das System flexibel sein, um alternative Lösungswege zu finden und auszuführen, wenn eine Aktion misslingt.

Aus diesem Grund wurde in dieser Arbeit eine Architektur entworfen und implementiert, die aus drei hierarchischen Ebenen besteht (Abbildung 1.5). Die reaktive Ebene beinhaltet einfache, spezialisierte Module, die die Sensoren bzw. Kameras mit den entsprechenden Aktuatoren verbinden. Dadurch kann der Roboter sofort auf Änderungen der Umgebung reagieren. Diese Module realisieren die elementaren Verhalten (*Behaviours*) [Bro91] des mobilen Manipulators und sind entweder explizit programmiert oder mit neuronalen Netzen erlernt. Sie setzen die Information der Kameras ein, um den Manipulator zum Ziel zu führen, das Objekt im Sichtfeld der Greiferkamera zu halten oder den Greifer bzw. die Manipulator-Segmente vor Kollisionen zu bewahren.

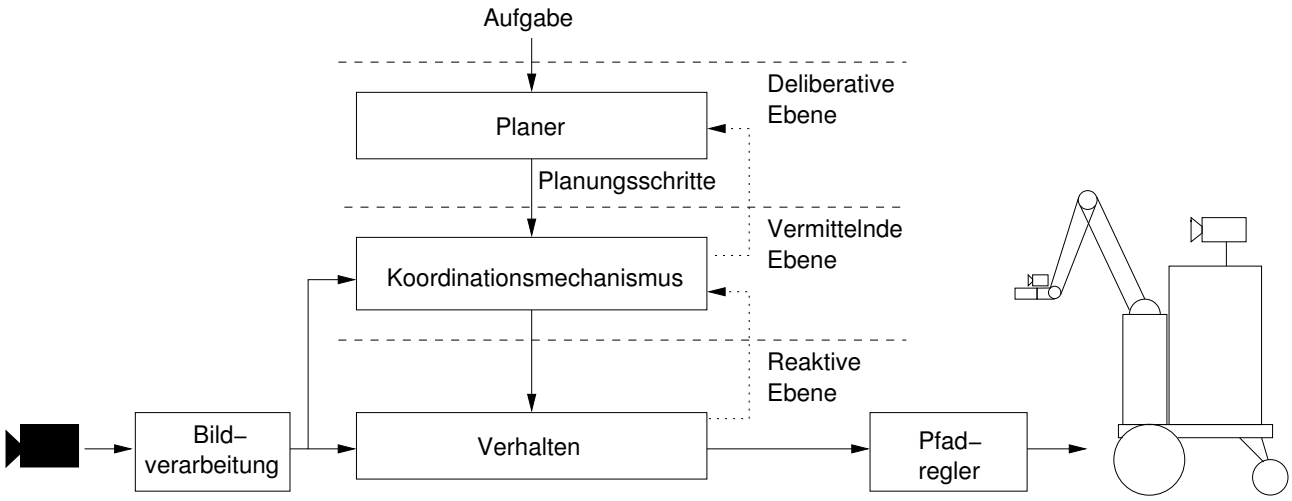


Abbildung 1.5: Realisierte drei-Ebenen Architektur.

Die hierarchisch höchste, deliberative Ebene zerlegt eine Aufgabe in eine Reihenfolge von Planungsschritten für den Manipulator und die mobile Plattform. Hier arbeitet ein symbolischer Planer strategische Pläne aus und setzt Zwischenziele, indem er die Pläne auf ihre Wirkung in die Zukunft projiziert. Die Zwischenziele bestehen aus Sequenzen von Planungsschritten, deren Ausführungsreihenfolge von den Ergebnissen der ausgeführten Aktionen abhängt. Diese Aufteilung vereinfacht die Komplexität der Planung und ermöglicht einerseits eine zielstrebige, globale Planung, die auch für komplexe Aufgaben Lösungen findet, und andererseits eine schnelle Anpassung existierender Pläne an unvorhersehbare Ereignisse auf lokaler Ebene.

Als Verbindung dieser zwei Ebenen dient eine dritte, vermittelnde Ebene, die anhand des auszuführenden Planungsschrittes die entsprechenden Verhalten auswählt und aktiviert. Auf dieser Ebene ist auch ein Koordinationsmechanismus implementiert, der die Ausgaben der aktiven bildgestützten Verhalten auf Basis der vorliegenden Aufnahmen gewichtet und fusioniert. Dadurch kann der Roboterarm zum Ziel geführt werden, während zugleich der Greifer und die Manipulatorsegmente Hindernisse vermeiden und die Roboterkamera das Ziel in ihrem Sichtfeld behält. In ähnlicher Weise werden auch die reaktiven Verhalten der mobilen Plattform koordiniert[1]. Die Koordination von mobiler Plattform und Manipulator miteinander findet über die Ausführungsreihenfolge der Planungsschritte statt, die in der deliberativen Ebene festgelegt wird.

Sowohl die bildgestützten Verhalten wie auch der Koordinationsmechanismus werden in einer virtuellen Umgebung mit Modellen der Objekte und des eingesetzten Serviceroboters trainiert und evaluiert.

[1] Auf der mobilen Plattform kommt für die reaktive und die vermittelnde Ebene der Ansatz von Pauly zum Einsatz. Der interessierte Leser kann eine genaue Beschreibung in [Pau98] finden.

Grund dafür ist, dass der Trainingsvorgang in der realen Welt aufwändig ist. Er erfordert einen präzisen Aufbau zur Akquisition der Trainingsdaten und ständige Überwachung. Besonders zu Beginn des Trainings sind Kollisionen mit Objekten im Arbeitsbereich nicht auszuschließen und können die Hardware beschädigen. Durch die Verwendung der virtuellen Umgebung ist dagegen die Automatisierung des Trainingvorgangs von bildgestützten Verhalten möglich, ohne die ständige Überwachung des Systems und ohne die Hardware zu gefährden. Als Trainingsdaten dienen Aufnahmen aus virtuellen Kameras und Positionsdaten des Manipulators. Für das Lernen von zielführenden Verhalten werden objektspezifische Pfade anhand einer Vorgabe automatisch generiert und entlang der Pfade die Trainingsdaten gesammelt (*algorithmisches Teach-In*). Beim *stochastischen Teach-In* dagegen erforscht der Roboterarm seinen Einsatzraum ohne eine Vorgabe. Dieses Verfahren basiert auf *Reinforcement Learning*-Techniken [SB98] und kommt beim Training der kollisionsvermeidenden Verhalten sowie des Koordinationsmechanismus zum Einsatz. Die erlernten Algorithmen werden anschließend auf den realen Serviceroboter transferiert und eingesetzt (siehe auch Abbildung 1.1).

1.3 Gliederung der Arbeit

In Kapitel 2 wird der aktuelle Stand der Technik präsentiert und ähnliche Ansätze in der Literatur vorgestellt. Anschließend findet die Präsentation des Gesamtkonzeptes und der realisierten Architektur sowie eine Abgrenzung von bestehenden Systemen statt.

Das dritte Kapitel behandelt die virtuelle Umgebung, die als Trainings- und Simulationsumgebung des mobilen Manipulators dient. Hier werden insbesondere die benötigten Bildverarbeitungsschritte zum Abgleich der Aufnahmen aus realen Kameras mit Bildern virtueller Kameras diskutiert. Zum Anschluss stellt das Kapitel das algorithmische und das stochastische Teach-In in der virtuellen Umgebung vor.

Die reaktive Ebene und die realisierten, bildgestützten Verhalten erläutert Kapitel 4. Dabei wird die Struktur der Fertigkeiten und das Training in der virtuellen Umgebung vorgestellt und Ergebnisse des Einsatzes in der virtuellen und in der realen Umgebung präsentiert. In diesem Kapitel wird auch ein so genanntes planendes Verhalten beschrieben, dessen Ausgabe als grobe Vorgabe zu Beginn einer Greifbewegung dient.

Kapitel 5 befasst sich mit der vermittelnden Ebene. Es stellt die implementierte Verhaltensauswahl und das Konzept der Verhaltenskoordinierung vor. Insbesondere wird auf die Topologie und Eigenschaften des Koordinationsmechanismus eingegangen und die Möglichkeiten zum Training in der virtuellen Umgebung werden untersucht. Ergebnisse des Trainings und des späteren Einsatzes des erlernten Koordinationsmechanismus auf dem realen Roboter werden am Ende des Kapitels vorgestellt.

Gegenstand vom sechsten Kapitel ist die deliberative Ebene. Die Aufteilung der Aufgabenplanung und deren Durchführung von einem reaktiven und einem symbolischen Planer wird erläutert und untersucht. Zusätzlich wird der Ansatz zur Pfadplanung für die mobile Plattform vorgestellt. In Kapitel 7 folgt die Zusammenfassung der Arbeit und ein Ausblick auf mögliche Erweiterungen.

Kapitel 2

Einführung in die mobile Manipulation

Im Folgenden wird der Stand der Technik bei der mobilen Manipulation wiedergegeben. Das Kapitel befasst sich zuerst mit existierenden Systemarchitekturen für Roboter und mit den Besonderheiten der mobilen Manipulation. Anschließend wird auf die einzelnen Komponenten einer Systemarchitektur eingegangen und insbesondere auf Verfahren zur reaktiven, bildgestützten Manipulation, zur Koordination von Verhalten, zur Planung von Aufgaben und zum Einsatz von virtuellen Umgebungen für das Lernen von Roboteralgorithmen. Als Nächstes folgt die Beschreibung von ausgewählten Systemen zur mobilen Manipulation sowie von weiteren Systemen, die mit dem in dieser Arbeit entwickelten mobilen Manipulator verwandt sind. Zum Abschluss wird das hier entwickelte Systemkonzept präsentiert und von ähnlichen Arbeiten abgegrenzt.

2.1 Allgemeine Systemarchitekturen für Roboter

Die Systemarchitektur bezieht sich auf den konzeptionellen Aufbau des Roboters. Sie beschreibt auf welche Weise ein System aus verschiedenen Komponenten zusammengesetzt ist und wie diese zusammenarbeiten, um Aktionen aus Sensordaten und Aufgabenstellung zu generieren. Nach Matarič [Mat01] ist zwischen vier Kategorien von Roboterarchitekturen zu unterscheiden:

- deliberative (*deliberative*),
- reaktive (*reactive*),
- verhaltensbasierte (*behaviour-based*) und
- hybride (*hybrid*) Architekturen.

Deliberative Architekturen beruhen auf dem *Sense-Plan-Act* (SPA) Ansatz [Gat98]. Sie bestehen hauptsächlich aus drei funktionalen Elementen: einem Sensoriksystem, einem Planungsagenten und einer ausführenden Komponente [Nil80] (siehe Abbildung 2.1a). Die Sensorik sammelt Informationen aus der Umgebung, um ein Weltmodell zu erstellen oder zu aktualisieren. Der Planungsagent operiert auf dem Weltmodell und sucht nach einer Sequenz von Zustands- und Aktionsschritten, die die gestellte Aufgabe optimal löst [Mat01]. Die ausführende Komponente ist für die Umsetzung des Planes in Roboterbewegungen verantwortlich. Beispiele für deliberative Architekturen sind NASREM zur Teleoperation [AML89] oder die TEAM Architektur [SSML90]. Da die Planerstellung zeitaufwendig ist, kann der SPA Ansatz nur dann erfolgreich eingesetzt werden, wenn keine zeitkritischen

Aktionen auszuführen sind und das Weltmodell dem realen Zustand der Welt entspricht. Dies ist jedoch bei mobilen Manipulatoren nicht der Fall, die in nur teilweise bekannten, veränderlichen Umgebungen ihre Aufgaben durchführen müssen.

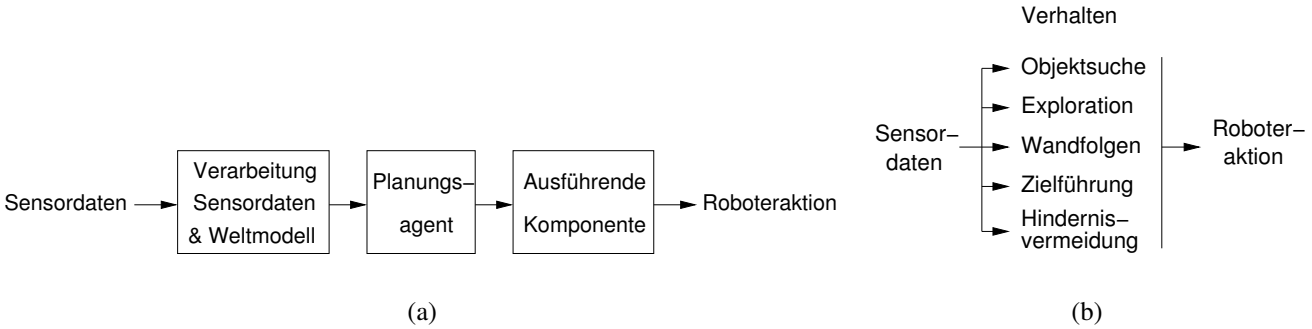


Abbildung 2.1: Struktur der deliberativen (links) und der reaktiven (rechts) Architekturen.

Reaktive Architekturen wurden von Brooks Mitte der 80er Jahre vorgestellt [Bro86] (Abbildung 2.1b). Anders als bei den deliberativen Architekturen speichern sie minimale Information über sich oder die Welt. Sie stellen dagegen eine direkte Verbindung zwischen Sensoreingängen und Aktuatorbewegungen her, so dass der Roboter schnell auf Änderungen in dynamisch veränderlichen Umgebungen reagiert. Dafür besitzen sie eine Menge von Bedingungs-Aktions Paaren, so genannte Verhalten (*Behaviours*), die durch eine *Lookup*-Funktion selektiert werden. Bekannte reaktive Architekturen umfassen unter anderem die *Subsumption* Architektur [Bro86], die *Situated Automata* [Ros85] oder *PENGI* [AC87]. Systeme mit reaktiven Architekturen sind dadurch beschränkt, dass sie über kein Gedächtnis und keine interne Repräsentation der Umgebung verfügen und daher nicht lernen können [Bro91]. Außerdem sind sie nicht *taskable*, d.h. es ist nicht möglich ihre Aufgabe neu zu definieren ohne ihre Kontrollstruktur zu ändern [Gat98]; dadurch sind sie bedingt erweiterbar. Weiterhin sind rein reaktive Architekturen besonders bei komplexen Aufgaben, die eine analytische Planung erfordern, nicht einsetzbar.

Verhaltensbasierte Architekturen sind nicht auf eine *Lookup*-Funktion beschränkt, sondern benutzen intern komplexe Berechnungsmethoden und Modelle [Mat01]. Ihre Realisierung findet mit einem *bottom-up* Ansatz statt. Zuerst wird eine Menge von Verhalten erstellt, die Basisfunktionalitäten realisieren, wie Hindernisvermeidung oder Zielführung. Anschließend werden weitere Verhalten implementiert, die komplexere Fähigkeiten zur Verfügung stellen, wie Wandfolgen, Exploration oder Zielverfolgung, bis der Roboter die erwünschte Menge an Gesamtfähigkeiten erreicht hat. Die Verhalten sind in Netzwerken organisiert (siehe Abbildung 2.2a), die in ihrer Struktur Repräsentationen der Umgebung, des Roboters selbst und weiterer Roboter abspeichern können. Diese Information wird zur Erstellung von Plänen und zur Kooperation mit anderen Robotern eingesetzt. Die verhaltensbasierte Architekturen sind dadurch für kooperierende Multiroboter-Systeme geeignet, besonders falls eine vorausschauende Planung der Aktionen weiterer Roboter in der Umgebung erforderlich ist [Mat01]. Die bekanntesten verhaltensbasierten Architekturen sind die *Hierarchical Abstract Behavior Architecture* [NM02] und die Aktivierungsnetzwerke von Maes [Mae90]. In eine ähnlichen Richtung gehen auch die Arbeiten von Sun [SS00c], [SS00b] und von Digney [Dig98] für das Lernen von Aktionsfolgen.

Demgegenüber sind hybride Architekturen eher für einzelne Roboter geeignet [Mat01]. Sie versuchen die besten Elemente der reaktiven und der deliberativen Architekturen zu kombinieren. Als Resultat

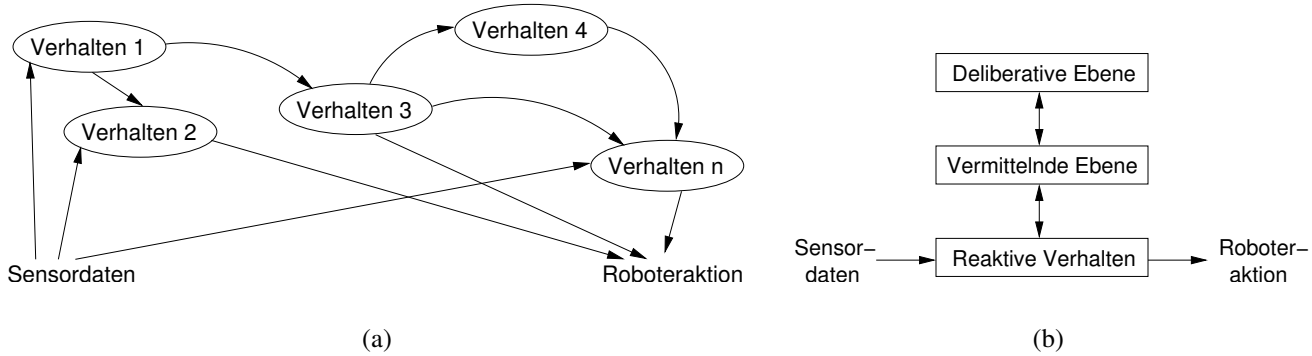


Abbildung 2.2: Struktur einer verhaltensbasierten (links) und einer hybriden Architektur mit drei Ebenen (rechts).

verfügt das System über zwei Komponenten, eine deliberative und eine reaktive, die miteinander interagieren, um eine schlüssige Ausgabe zu produzieren. Die reaktive Komponente reagiert auf Änderungen der Umgebung und steuert die Motorik des Roboters anhand der aktuellen Sensordaten. Die deliberative Komponente setzt dagegen eine interne, symbolische Darstellung der Welt ein und arbeitet auf der höchsten Abstraktionsebene, um strategische Pläne zu produzieren. Reaktive und deliberative Komponente müssen miteinander kommunizieren. Die deliberative Komponente setzt die Zwischenziele für die reaktive Ebene, um den Roboter zu effizienten Strategien zu führen und ihn vor lokalen Minima, wie z.B. Sackgassen, zu bewahren. Die reaktive Ebene muss dann die Zwischenziele des Lösungsplanes anhand der aktuellen Sensordaten erreichen und die interne Weltdarstellung der Planungsebene aktualisieren, falls sich die Welt ändert. Diese Interaktion zwischen reaktiver und deliberativer Ebene erfordert eine vermittelnde Ebene [Mat01]. Deswegen sind die hybriden Architekturen auch als drei-Ebenen (*three layer*) Systeme bekannt, da sie meistens aus einer reaktiven, einer vermittelnden und einer deliberativen Ebene bestehen (Abbildung 2.2b). Ihr bekanntester Repräsentant ist die 3T-Architektur [BFG$^+$97]. In dieser Architektur beinhaltet die reaktive Ebene die reaktiven Fertigkeiten bzw. Verhalten des Roboters (*Skills* oder *Behaviours*) und einen *Skill-Manager*, der die Kommunikation mit der vermittelnden Ebene übernimmt und die Fertigkeiten verwaltet. Die vermittelnde Ebeneverfügt über vordefinierte Folgen von Verhalten, die sie entsprechend der Sensordaten und dem aktuellen Zwischenziel aktiviert. Sie kann mehrere Fertigkeiten zugleich aktivieren, aber lediglich eine einzelne erhält die Kontrolle über die Motorik; die restlichen dienen zur Verarbeitung der Sensordaten für das Verhalten, das den Roboter ansteuert. Die deliberative Ebene schlägt einen Plan vor, der die gestellte Aufgabe löst, und setzt die Zwischenziele für die vermittelnde Ebene.

Weitere hybride Architekturen mit drei Ebenen, die auf mobilen Plattformen Anwendung finden und eine ähnliche Funktionsweise zur 3T-Architektur aufweisen, umfassen Atlantis [Gat92], AuRA [Ark98], IPDI [Sar95], ROGUE [HV97], die Arbeiten von Noreils und Chatila [NC95], RA (*Remote Agent*) [MNPW98], die MACTA Architektur [ACBGH97], die motivationsbasierte Architektur von Stoytchev und Arkin [SA01] und die Blackboard-Architektur von Pauly [Pau98]. Alternativ gibt es auch zwei-Ebenen Architekturen, wobei hier oft die vermittelnde Ebene fehlt. Das führt automatisch zur Integration reaktiver Elemente in der Planungsebene. Zu dieser Gruppe gehören unter anderem DAMN von Rosenblatt [Ros95], ISAC [BCN$^+$95], die *Multi-Valued Logic* [Saf97], Saphira [KM00] und CLARAty [VNE$^+$01]. Weitere hybride Architekturen umfassen die Architektur des Museumroboters Rhino [BAB$^+$00], die mit den SPA-Ansätzen verwandt ist, die konnektionistische DRAMA Architektur [BH99] und die Architektur des Roboters JANUS [RSB97], die zu den verhaltensbasier-

ten Systemen tendiert.

Auf mobile Manipulatoren kommen hauptsächlich hybride Ansätze zum Einsatz, wie beispielsweise in [BK95], [Bro90], [FMB+97], [NY97], [WN01], [MA99], [Pet02], [HB01], [IS01] und [Ste94][1]. Durch die Kombination von mobiler Plattform und Roboterarm stellen sich jedoch bei den mobilen Manipulatoren komplexere Fragestellungen und höhere Anforderungen als in den Bereichen der stationären Manipulation und der autonomen Navigation. Deswegen können Architekturen, die für stationäre Roboterarme oder mobile Plattformen entwickelt sind, nicht direkt auf mobile Manipulatoren übertragen werden. In Tabelle 2.1 sind die wichtigsten Gemeinsamkeiten und Unterschiede der drei Forschungsbereiche aufgelistet.

		stationäre Manipulation	**autonome Navigation**	**mobile Manipulation**
Sensorik	Sensorplatzierung in der Umgebung	√		√ √
	Sensorplatzierung auf dem Roboter	√	√ √	√ √
	Positionssensoren	√ √		√
	Odometrie		√ √	√
	Tastsensoren		√	√
	Kraftsensoren	√		√
	abstandsmessende Sensoren		√ √	√
	Kameras	√	√	√ √
Umgebung	veränderliche Umgebung		√ √	√ √
	variable Umgebungsbedingungen (z.B. Beleuchtungsvarianz)		√	√ √
	Sensorrauschen	√	√ √	√ √
	Hindernisse		√ √	√ √
Objekte	objektabhängige Greifrichtung	√		√ √
	Präzision des Greifens	√ √		√
Aufgaben	Inspektion		√	√
	Transport		√	√
	Pick-and-place	√		√ √
	Montage	√ √		√
	Interaktion mit Menschen		√	√ √

Tabelle 2.1: Gemeinsamkeiten und Unterschiede bei der stationären Manipulation, der autonomen Navigation und der mobilen Manipulation (√: oft vorhanden, √ √: sehr wichtig).

Da mobile Manipulatoren in alltäglichen Umgebungen Einsatz finden, ist eine detaillierte, dreidimensionale Wahrnehmung ihres Arbeitsumfeldes sehr wichtig, die jedoch ein abstandsmessendes Sensorsystem nicht liefern kann. Zur Zeit erscheint die Anwendung von Kameras als die beste Möglichkeit, um genügend Information zu sammeln und Zielobjekt und Hindernisse in einer veränderlichen Umgebung auch mit verrauschten Daten robust zu detektieren und zu lokalisieren. Während eines Greifvorganges muss der Manipulator Hindernissen ausweichen und sich zugleich dem Objekt annähern, ohne dass er in *Deadlock*-Situationen gerät. Diese Fusion der Ausgaben von Hindernisvermeidung und Zielführung ermöglicht ein effizientes Greifen in schwierigen Situationen, stellt aber

[1] Weitere mobile Manipulatoren, deren Architektur und Eigenschaften jedoch in der Literatur nicht ausführlich beschrieben sind, umfassen unter anderem Hermes [BG98], Roman [EFHS98] und Robutler [HOB+04].

auch den höchsten Anspruch bei der Realisierung des Systems [Mat01]. Der Greifvorgang selbst sollte die Form und Funktionalität des Objektes berücksichtigen. Diese erzwingen eine objektabhängige Greifrichtung für den Manipulator, der an das Zielobjekt über einen objektspezifischen Pfad gelangen sollte, um ein robustes Greifen zu gewährleisten. Werden neue Aufgaben oder neue Fertigkeiten für den mobilen Manipulator definiert, um dessen Einsatzmöglichkeiten zu erweitern, dann sollte man sie im System einfügen können, ohne dass eine langwierige Anpassung der Parameter notwendig ist. In den nächsten Unterkapiteln wird auf Verfahren eingegangen, die als Komponenten einer hybriden Architektur eingesetzt werden können und den Anforderungen der mobilen Manipulation entgegenkommen. Anschließend werden einige ausgewählte Systeme zur mobilen Manipulation vorgestellt, die den Stand der Technik in diesem Bereich wiederspiegeln.

2.2 Reaktive Verhalten für Manipulatoren

In einer hybriden Architektur besteht die unterste, reaktive Ebene aus Verhalten (*Behaviours*), oft auch als Fähigkeiten oder Fertigkeiten (*Skills*) bekannt. Es handelt sich um Module, die für eine Aufgabe spezialisiert sind und die Sensoren mit den entsprechenden Aktuatoren verbinden[2]. Dadurch kann der Roboter sofort auf Änderungen der Umgebung reagieren. Verhalten setzen die Information der Sensoren ein, um den Roboter zum Ziel zu führen, ein Objekt zu verfolgen oder Aktuatoren vor Kollisionen zu schützen. Da bei der mobilen Manipulation Kameras als primäre Sensoren eingesetzt werden, beschränken sich die nächsten Abschnitte auf bildgestützte Verhalten und insbesondere auf die bildgestützte Zielführung und Hindernisvermeidung für Manipulatoren[3].

2.2.1 Bildgestützte Zielführung

Bei der bildgestützten Führung eines Roboters wird zunächst ein Satz von Merkmalen aus den Kameraaufnahmen extrahiert und daraus die nächste Bewegung des Roboters bestimmt. Corke [Cor96] unterteilt die verschiedenen Ansätze des bildgestützten Greifens in zwei Kategorien:

Open-loop Systeme verwenden keine visuelle Rückkopplung. Hier wird anhand einer einzelnen Aufnahme die Lage des Zielobjektes bestimmt und der Manipulator dorthin geführt. Diese Systeme sind dann erfolgreich, wenn die Bildverarbeitung die exakte Position des zu greifenden Objekten ermitteln kann. Diese Voraussetzung kann jedoch für einen mobilen Manipulator in einer teilweise unbekannten Umgebungen nicht immer eingehalten werden.

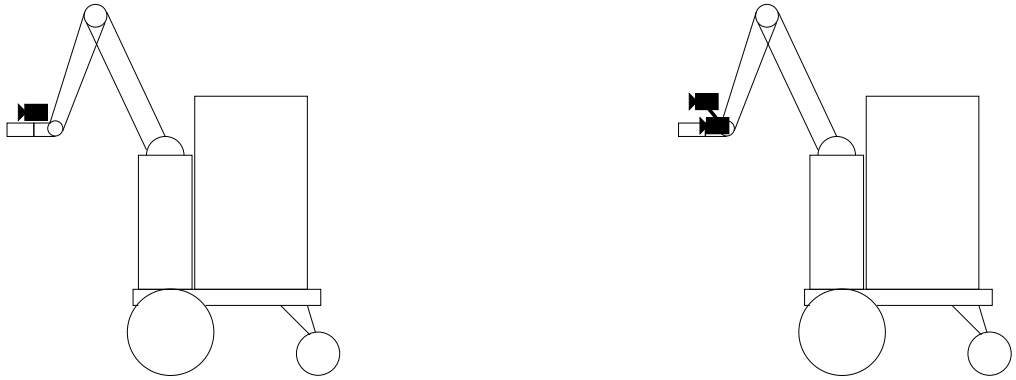
Closed-loop Systeme werten kontinuierlich Merkmale aus Kamerabildern aus, um den Fehler zwischen Ist- und Soll-Position des Greifers am Objekt zu minimieren. Diese Systeme sind als *Visual Servoing* Systeme bekannt. In [Kra01] ist ein guter Überblick über den Stand der Technik in diesem Bereich zu finden.

Weiterhin sind die Systeme anhand der Anzahl und der Position der Kameras relativ zum Roboterarm zu unterscheiden [Kra01]:

[2] Somit entsprechen sie den Verhalten der reaktiven Architekturen.

[3] Die reaktiven Verhalten zur Navigation der mobilen Plattform des hier präsentierten Systems stammen aus der Arbeit von Pauly [Pau98]; deswegen wird hier auf dieses Thema nicht näher eingegangen. Der interessierte Leser kann eine genaue Beschreibung der reaktiven Ebene der mobilen Plattform in [Pau98] finden.

Eye-in-hand Konfigurationen haben eine oder mehrere Kameras auf dem Greifer des Manipulators montiert, wie in Abbildung 2.3 zu sehen ist. Vorteil dieser Konfiguration ist die steigende Genauigkeit im Bild, je näher der Greifer dem Zielobjekt kommt. Außerdem können die Manipulatorsegmente das Ziel im Bild nicht verdecken. Jedoch liefert die Konfiguration keine vollständige Sicht der Szene und bestimmte Bewegungen des Roboterarms können das Zielobjekt außerhalb des Kamerasichtfeldes bringen. Verfahren, die eine Kamera in einer *eye-in-hand* Konfiguration einsetzen (Abbildung 2.3a), sind unter anderem in [SBP97], [MC01], [BRS99], [DC00] beschrieben. Sie haben den Nachteil, dass eine einzelne Aufnahme keine Tiefeninformation liefern kann und deswegen benötigen sie Modellwissen über das Zielobjekt, um es im Raum zu lokalisieren. Dieses Problem löst die Konfiguration mit zwei Kameras (Abbildung 2.3b). Sie wird jedoch selten eingesetzt, da der kleine Abstand[4] zwischen den Kameras keine genaue Rekonstruktion der 3D-Struktur des Arbeitsbereiches erlaubt [Kra01]. In dieser Kategorie gehören die Ansätze in [ALH+98] und [MCB00].



(a) (b)

Abbildung 2.3: Mögliche *eye-in-hand* Konfigurationen mit einer (a) und zwei (b) Kameras.

Eye-to-hand Konfigurationen sind Systeme mit Kameras, deren Position nicht von der Roboterbewegung beeinflusst wird (Abbildung 2.4). Sie liefern einen besseren Szenenüberblick als ein *eye-in-hand* System, die Manipulatorsegmente können jedoch wichtige Teile der Aufnahmen verdecken. Die Konfiguration mit einer Kamera, in Abbildung 2.4a dargestellt, wurde früher oft eingesetzt, wie in [YA94], [FLM92] und [Kra01]. Sie benötigt aber Modellinformation, um die Tiefe der Szene zu schätzen. Systeme mit Stereokameras (Abbildung 2.4b) liefern sowohl Tiefeninformation als auch eine gute Szenenübersicht. Solche Systeme sind in [RH99], [GMOS96], [NK95], [Kra01], [SC00], [SS98] und [Hag95] beschrieben.

Multikamera Systeme setzen eine Kombination von Kameras auf dem Greifer und auf der Plattform bzw. im Raum ein [SD98], wie in Abbildung 2.5 zu sehen ist.

Unabhängig von der Anzahl und Position der Kameras müssen alle Systeme aus den Kameraaufnahmen die aktuelle Ist-Position des Greifers mit der Soll-Position am Zielobjekt vergleichen und

[4] Der Abstand zwischen den Kameras bei der *eye-in-hand* Konfiguration ist meistens durch die Dimension des Greifers festgelegt bzw. begrenzt.

(a) (b)

Abbildung 2.4: Mögliche *eye-to-hand* Konfigurationen mit einer (a) und zwei (b) Kameras.

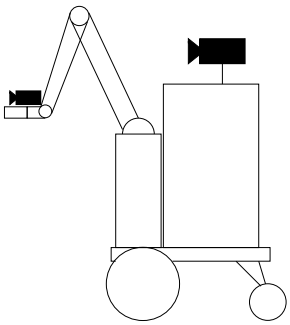


Abbildung 2.5: Multikamera Konfiguration

eine Roboterbewegung ermitteln, die zur Soll-Position führt. Sei $\vec{s}$ der Vektor der aus den Aufnahmen extrahierten Merkmale, $\vec{q}$ der Positionsvektor des Greifers und die Beziehung von $\vec{s}$ zu $\vec{q}$ mit n Funktionen f_i angegeben:

$$\begin{aligned} s_1 &= f_1(q_1, ..., q_m) \\ &\vdots \\ s_n &= f_n(q_1, ..., q_m) \end{aligned} \tag{2.1}$$

Zusätzlich sei $\Delta\vec{s}$ die Änderung des Merkmalsvektors von der Ist- zur Soll-Position. Dann ist die Roboterbewegung $\Delta\vec{q}$ gesucht, die die Änderung $\Delta\vec{s}$ verursachen kann. Da es beim *Visual Servoing* um einen Regelungsprozess handelt, kann man $\Delta\vec{s}$ und $\Delta\vec{q}$ in einem kleinen zeitlichen Abschnitt Δt betrachten und sie somit durch die Ableitungen von $\vec{s}$ und $\vec{q}$ ersetzen, also $\dot{\vec{s}}$ und $\dot{\vec{q}}$. Die Beziehung von $\dot{\vec{s}}$ zu $\dot{\vec{q}}$ wird mit Hilfe der Jacobi Matrix dargestellt, die die partielle Ableitungen der Funktionen f_i beinhaltet:

$$\dot{\vec{s}} = \begin{bmatrix} \frac{\vartheta f_1}{\vartheta q_1} & \cdots & \frac{\vartheta f_1}{\vartheta q_m} \\ \vdots & \ddots & \vdots \\ \frac{\vartheta f_n}{\vartheta q_1} & \cdots & \frac{\vartheta f_n}{\vartheta q_m} \end{bmatrix} \dot{\vec{q}} = \mathbf{J}(\vec{q})\dot{\vec{q}} \tag{2.2}$$

und dadurch ist die gesuchte Roboterbewegung $\dot{\vec{q}}$:

$$\dot{\vec{q}} = \mathbf{J}^{+}(\vec{q})\dot{\vec{s}} \tag{2.3}$$

Dabei ist $\mathbf{J}$ die Jacobi Matrix und $\mathbf{J}^+$ ihre Pseudoinverse. Gleichung 2.3 drückt das Bild-Motor Modell (*Visual-Motor Model*) aus, das die Beziehung der erwünschten Änderungen der Bildmerkmale $\dot{\vec{s}}$ zu den dazu benötigten Roboterbewegungen $\dot{\vec{q}}$ beschreibt (Abbildung 2.6). Ist $\vec{q}$ in kartesischen Koordinaten angegeben, dann handelt es sich in Gleichung 2.3 um die Bild-Jacobi Matrix (*Image Jacobian* oder *Interaction Matrix*). Falls $\vec{q}$ Gelenkpositionen beinhaltet, also einer Gelenkstellung entspricht[5], dann ist $\mathbf{J}$ die Bild-Gelenk Jacobi Matrix. Entsprechend wird die Beziehung zwischen Gelenkraum und kartesischen Raum mit der Roboter-Jacobi Matrix (*Robot Jacobian*) ausgedruckt (Abbildungen 2.7 und 2.8). Falls die Kinematik des Roboterarms bekannt ist, dann ist die Roboter-Jacobi Matrix leicht zu berechnen [SS00a].

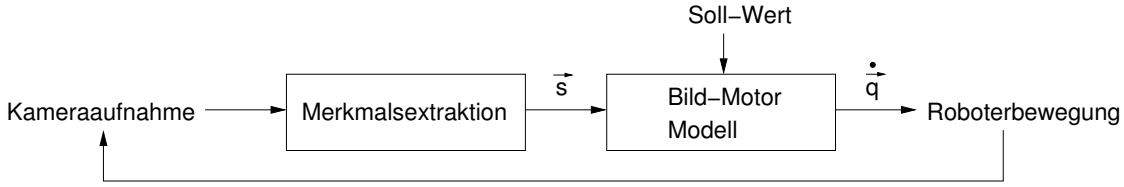


Abbildung 2.6: Ablauf beim *Visual Servoing*.

In $\vec{s}$ kommen unter anderem folgende Merkmale zum Einsatz:

- Eckpunkte, wie in [WWH97], [KC02], [DC00], [MC01], [MCB98], [HDE98], [RH99].
- Linien, Kreise [WH97], [ALH+98], [RTHN97].
- Geometrische Momente [MZBK02], [Cha02], [YKD97].
- Snakes [SP96], [DC99].
- Eigenwerte [ZKS00], [KC02].
- Optischer Fluss [SBP97], [vdS95].
- Korrelationstechniken mit Fensterregionen (*Regions of Interest*) [KC99], [BSP94], [HT98].
- Kombinationen davon [WVT96], [WT98].

Kragić [Kra01] unterscheidet die Verfahren zum *Visual Servoing* anhand des Bild-Motor Modells. Das Modell kann entweder a priori bekannt sein, oder geschätzt werden. Beim ersten Fall kann man wiederum die Ansätze in positionsbasierte, bildbasierte und 2 1/2D unterscheiden. Ausschlaggebend ist dabei, ob der Regelungsvorgang des *Visual Servoings* im kartesischen Raum oder im Bildbereich stattfindet. Bei der Schätzung kann es sich entweder um eine analytische Berechnung handeln oder das Bild-Motor Modell wird erlernt.

A priori bekanntes Bild-Motor Modell

Positionsbasierte Ansätze beschreiben das Fehlersignal E für den Regelungsvorgang des *Visual Servoings* in den kartesischen Koordinaten des Greifers. Deswegen muss das System zunächst aus der Bildinformation das Zielobjekt und eventuell den Manipulator im kartesischen Raum lokalisieren und den Unterschied zwischen aktueller und erwünschter Lage des Roboterarms ermitteln. Dies

[5] Die Grundlagen über Manipulatoren, Manipulatorkinematiken und Koordinatentransformationen werden in Anhang C.1 vorgestellt.

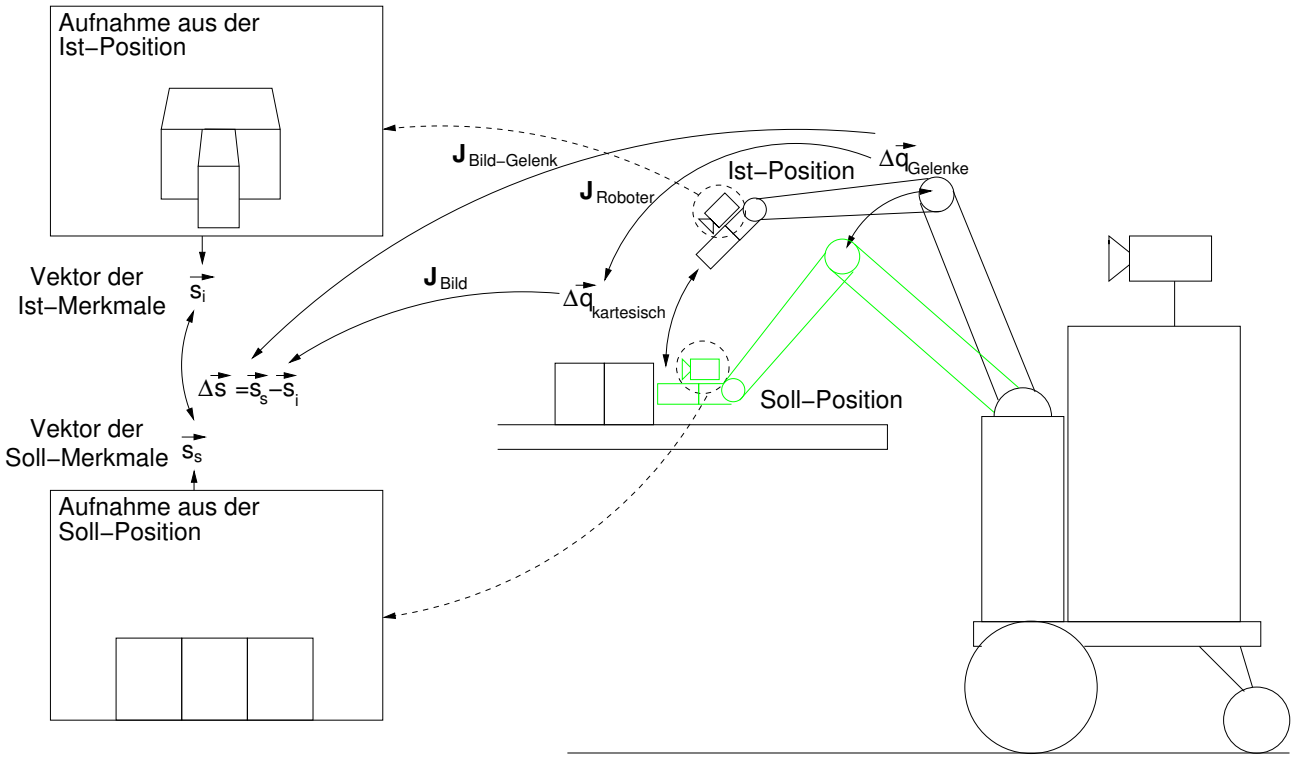


Abbildung 2.7: Jacobi Matrizen bei einem mobilen Manipulator.

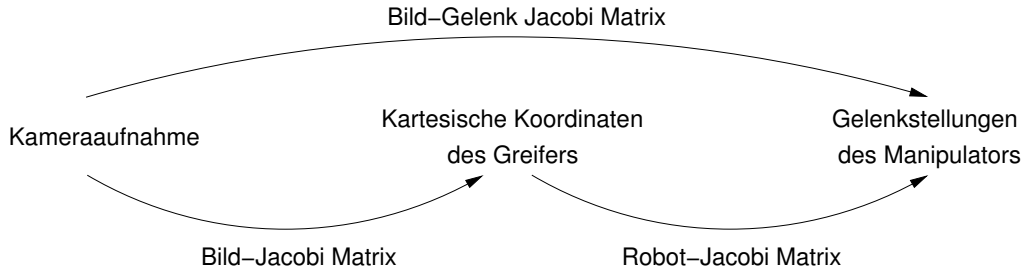


Abbildung 2.8: Beziehungen zwischen den Jacobi Matrizen.

erfordert die genaue Kenntnis der internen Kameraparameter, die die Abbildungseigenschaften der Kamera beschreiben (intrinsische Parameter), sowie der Position der Kamera zum Roboter (extrinsische Parameter) [HHC96]. Die Ermittlung der intrinsischen und der extrinsischen Parameter wird als Kalibrierung der Kameras bezeichnet [SHB99].

In Abbildung 2.9 ist das Prinzip des positionsbasierten *Visual Servoings* dargestellt. Seien ${}^{RB}\vec{p}_{Gs}$ und ${}^{RB}\vec{o}_{Gs}$ die Vektoren der kartesischen Soll-Position und Soll-Orientierung des Greifers bezüglich der Roboterbasis[6] und ${}^{RB}\vec{p}_{Ga}$ und ${}^{RB}\vec{o}_{Ga}$ die aktuellen Werte, die aus der Kinematik des Roboterarms bekannt sind. Für ${}^{RB}\vec{p}_{Gs}$ und ${}^{RB}\vec{o}_{Gs}$ muss das Objekt aus der Bildverarbeitung bezüglich der Kamera(s) lokalisiert und anhand der extrinsischen Kalibrierungsparameter bezüglich des Greifers transformiert werden. Das Fehlersignal E ist:

$$E = \begin{pmatrix} \Delta^{RB}\vec{p}_{Gsa} \\ \Delta^{RB}\vec{o}_{Gsa} \end{pmatrix} = \begin{pmatrix} {}^{RB}\vec{p}_{Gs} - {}^{RB}\vec{p}_{Ga} \\ {}^{RB}\vec{o}_{Gs} - {}^{RB}\vec{o}_{Ga} \end{pmatrix} \tag{2.4}$$

[6]Nach der Schreibweise von Craig [Cra89] entspricht ${}^{A}\vec{a}$ einem Vektor, der relativ zum Bezugssystem $\{A\}$ beschrieben ist. Der führende hochgestellte Buchstabe entspricht dem Bezugskoordinatensystem, hier $\{A\}$.

und die nächste Bewegung des Roboters im Fall eines P-Reglers ist:

$$\dot{\vec{q}}_{Gr} = \mathbf{K}_P E \tag{2.5}$$

mit $\mathbf{K}_P$ eine konstante Verstärkungsmatrix (*Gain Matrix*). Die Roboter-Jacobi Matrix $\mathbf{J}_{Roboter}$ kommt dann zum Einsatz, um die Gelenkbewegung $\dot{\vec{q}}_{Gl}$ zu ermitteln:

$$\dot{\vec{q}}_{Gl} = \mathbf{J}^{+}_{Roboter}(\vec{q}_{Gl})\dot{\vec{q}}_{Gr} \tag{2.6}$$

Das positionsbasierte *Visual Servoing* hat den Vorteil, dass die Roboterbewegung im kartesischen Raum bestimmt wird, was die Planung einer Trajektorie zur gleichzeitigen Zielführung und Hindernisvermeidung erleichtert. Zusätzlich ist es auch bei großräumigen Bewegungen des Manipulators anwendbar. Anderseits haben Kalibrierungsfehler einen großen Einfluss auf das Verfahren und können den Manipulator weit vom Ziel wegführen. Der Ansatz kann außerdem bei einer *eye-in-hand* Konfiguration nicht garantieren, dass das Objekt während der Roboterbewegung im Sichtbereich der Kamera erhalten bleibt [HHC96].

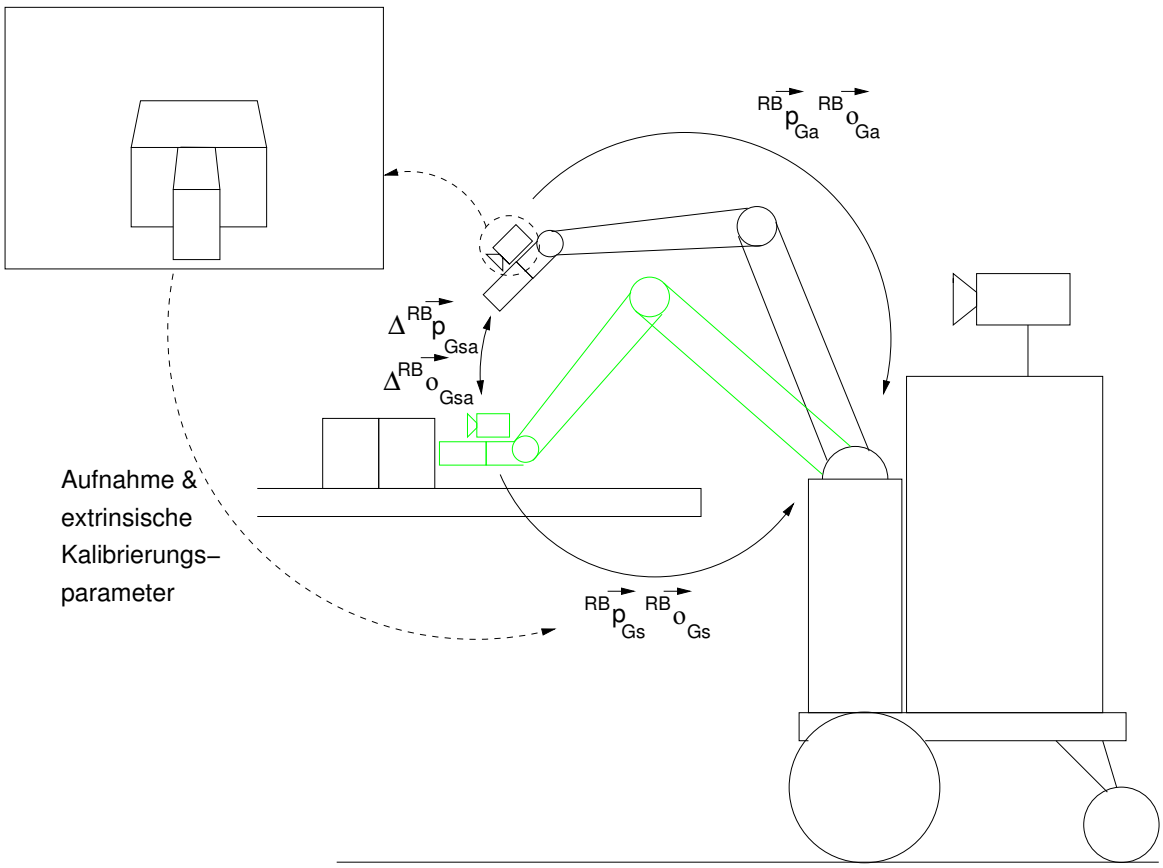


Abbildung 2.9: Positionsbasiertes *Visual Servoing*. Hier wird zuerst der Fehler zwischen aktueller und erwünschter Lage des Roboterarms im kartesischen Raum ermittelt. Anschließend versucht man mit einem P-Regler diesen Fehler zu minimieren.

Bildbasierte Ansätze drücken das Fehlersignal aus als die Differenz der Ist-Merkmale $\vec{s}_i$ im aktuellen Kamerabild und der Soll-Merkmale $\vec{s}_s$[7] in der erwünschten Endposition (Abbildung 2.10):

[7] Um $\vec{s}_s$ zu generieren, wird oft ein so genanntes *Teaching by Showing* angewendet [BAH94]. Dabei wird der Roboter an die gewünschte Position gestellt und der Merkmalsvektor aufgezeichnet.

$$E(\vec{s}) = \vec{s}_i - \vec{s}_s \tag{2.7}$$

Ziel des Ansatzes ist, den Roboter so zu bewegen, dass die extrahierten Bildmerkmale einen erwünschten Wert bzw. eine erwünschte Position im Bild erreichen [HHC96] und dadurch die Fehlerfunktion E minimiert wird. Dafür wird die Bild-Jacobi Matrix $\mathbf{J}_B(\vec{q})$, die die Beziehung zwischen Änderungen in der Greiferposition und Änderungen im Bild repräsentiert, verwendet:

$$\dot{\vec{s}} = \mathbf{J}_B(\vec{q})\dot{\vec{q}} \tag{2.8}$$

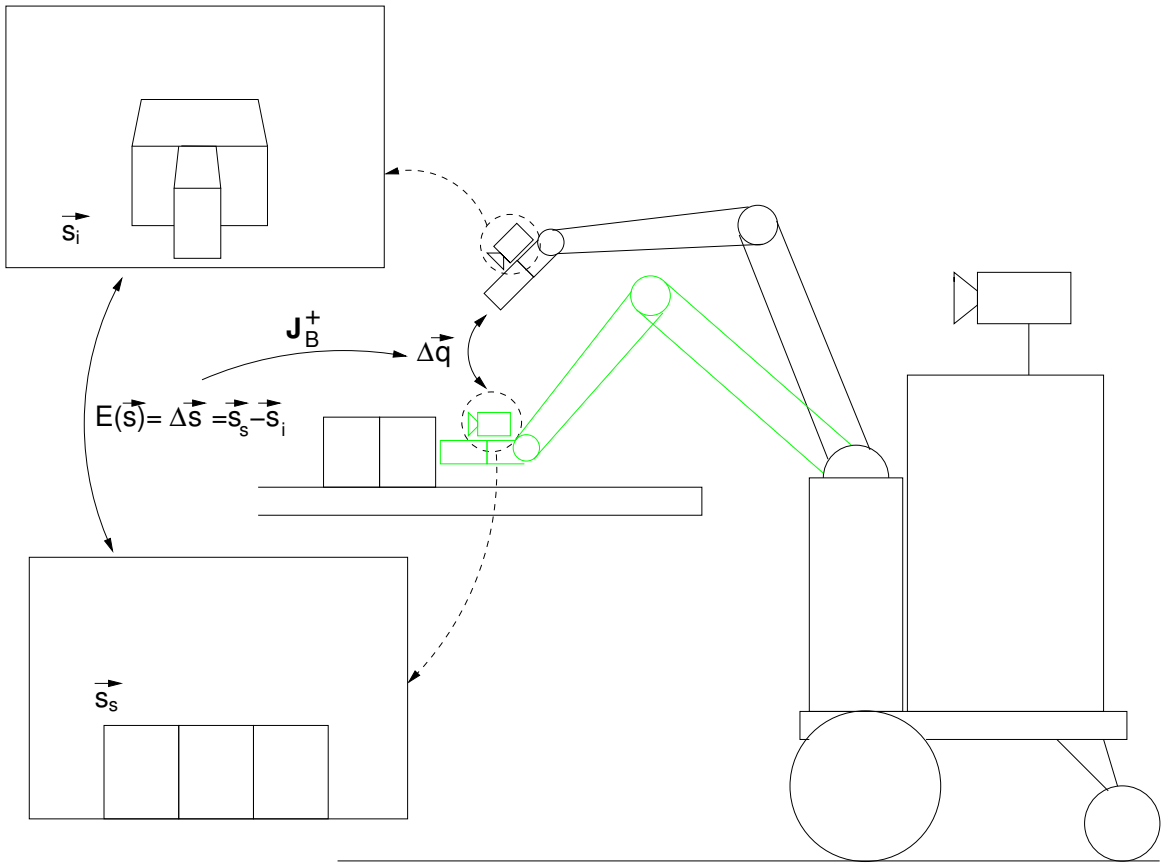


Abbildung 2.10: Bildbasiertes *Visual Servoing*. Hier wird der Regelfehler als Differenz der Ist-Merkmale $\vec{s}_i$ und der Soll-Merkmale $\vec{s}_s$ berechnet und über die Pseudoinverse der Bild-Jacobi Matrix $\mathbf{J}_B(\vec{q})$ zu einer Änderung der kartesischen Greiferposition transformiert.

Aus Gleichungen 2.8 und 2.7 ergibt sich für die nächste Roboterbewegung [HHC96]:

$$\dot{\vec{q}} = \mathbf{K}_B\mathbf{J}_B^+(\vec{q})E(\vec{s}) \tag{2.9}$$

wobei $\mathbf{K}_B$ eine konstante Verstärkungsmatrix ist und $\vec{q}$ die kartesische Position und Orientierung des Greifers.

Beim bildbasierten *Visual Servo* mit zwei Kameras kann die Fehlerfunktion für beide Bilder simultan minimiert werden [Kra01]. Diese Eigenschaft erlaubt das einfache Erstellen der Bild-Jacobi des gesamten Systems, indem man die Bild-Jacobi Matrizen für linke und rechte Kamera:

$$\begin{aligned} \dot{\vec{s}}_l &= \mathbf{J}_{Bl}(\vec{q})\dot{\vec{q}} \\ \dot{\vec{s}}_r &= \mathbf{J}_{Br}(\vec{q})\dot{\vec{q}} \end{aligned} \tag{2.10}$$

in eine Matrix $\mathbf{J}_{Blr}(\vec{q})$ zusammenfasst (*stacking*):

$$\dot{\vec{s}}_{lr} = \begin{pmatrix} \dot{\vec{s}}_l \\ \dot{\vec{s}}_r \end{pmatrix} = \mathbf{J}_{Blr}(\vec{q})\dot{\vec{q}} = \begin{pmatrix} \mathbf{J}_{Bl}(\vec{q}) \\ \mathbf{J}_{Br}(\vec{q}) \end{pmatrix} \dot{\vec{q}} \tag{2.11}$$

Dabei beinhalten die Vektoren $\vec{s}_l$ und $\vec{s}_r$ die Merkmale, die aus den Aufnahmen der linken und der rechten Kamera extrahiert wurden.

In Gleichungen 2.9 und 2.11 ist keine Information über die Position der Kamera bezüglich des Roboterarms vorhanden. Daher haben die extrinsischen Kalibrierungsparameter keinen Einfluss auf das bildbasierte *Visual Servoing* und der Ansatz ist robuster gegenüber fehlerbehafteter Kalibrierung. Nachteilig wirkt sich jedoch die Abhängigkeit der Pseudoinversen $\mathbf{J}_B^+$ von $\vec{q}$ aus. Insbesondere im Fall eines monokularen Systems ist die Berechnung von $\mathbf{J}_B^+$ schwierig. Deswegen gehen die meisten monokularen Systeme von einer konstanten Jacobi-Matrix aus. Diese ist jedoch nur in einem begrenzten Bereich um die erwünschte Zielposition gültig und garantiert außerhalb dieser Region keine Konvergenz des Verfahrens [Cha98]. Ein aktuelles Forschungsthema ist deswegen die Erweiterung der Verfahren, so dass sie den gesamten Arbeitsbereich des Roboterarms überdecken [MC01], [CV04]. Eine weitere Beschränkung des bildbasierten *Visual Servoings* ist der Einsatz von charakteristischen Bildpunkten[8] als Merkmale, da sie für zwei Kameras eine einfache Berechnung der Bild-Jacobi Matrix ermöglichen [HHC96]. Punkte können aber in einer realen Umgebung mit Beleuchtungsunterschieden nicht immer robust extrahiert werden, es sei denn, sie sind speziell markiert. Alternativ bieten sich Bildmomente aus segmentierten Regionen der Aufnahmen. Chaumette stellt ein erstes Verfahren für die analytische Berechnung der Bild-Jacobi für Bildmomente vor [Cha02], [Cha04]. Die Forschung in diesem Bereich ist jedoch erst in den Anfängen.

2 1/2D Ansätze sind eine Fusion von positionsbasierten und bildbasierten Verfahren für eine *eye-in-hand* Konfiguration. Sie versuchen die Vorteile von beiden Verfahren zu kombinieren: die Robustheit des bildbasierten *Visual Servoings* mit den großräumigen Bewegungsmöglichkeiten der positionsbasierten Ansätze. Das Prinzip ist in [MCB98] präsentiert und basiert auf der unabhängigen Schätzung der Kameratranslation und -rotation zwischen aktueller und erwünschter Sicht des Objekts. Zu diesem Ziel kommen Verfahren aus der 3D-Rekonstruktion zum Einsatz, die aus zwei Bildaufnahmen die Rotation und die skalierte Translation[9] der Kameralagen bei den Aufnahmepositionen berechnen. Zusätzlich werden die homogenen Bildkoordinaten der Merkmale[10] ermittelt [Fau93]. Die Regelung verläuft für die Rotation im kartesischen Raum anhand der Ergebnisse der 3D-Rekonstruktion und für die Translation im Bildbereich anhand der berechneten homogenen Bildkoordinaten (siehe auch Abbildung 2.11). Die Fehlerfunktion ist dann:

$$E(\vec{s}^*) = [\vec{s}_i^* - \vec{s}_s^* \vec{e}^T \theta]^T \tag{2.12}$$

Hier repräsentieren $\vec{e}$ den Vektor der Rotationsachse und θ den Rotationswinkel. Die Vektoren $\vec{s}_i^*$ und $\vec{s}_s^*$ beinhalten die homogenen Bildkoordinaten der Merkmale in der aktuellen und der Zielposition. Die Roboterbewegung wird dann ähnlich zu Gleichung 2.9 bestimmt.

[8] Als charakteristische Bildpunkte werden meistens die Eckpunkte des Zielobjektes im Bild gewählt.

[9] Der Skalierungsfaktor ist dabei unbekannt.

[10] Ähnlich wie bei dem bildbasierten *Visual Servoing* verwendet man auch hier als Merkmale Eckpunkte des Zielobjektes im Bild.

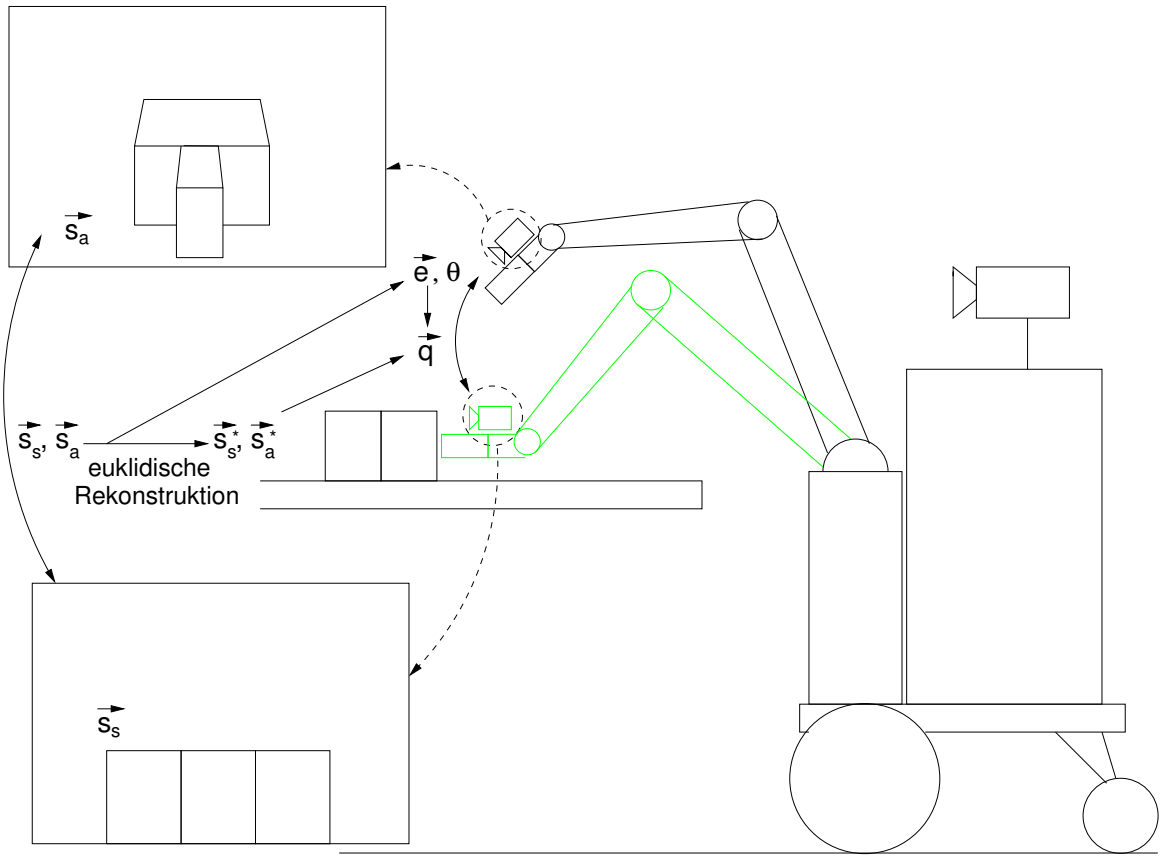
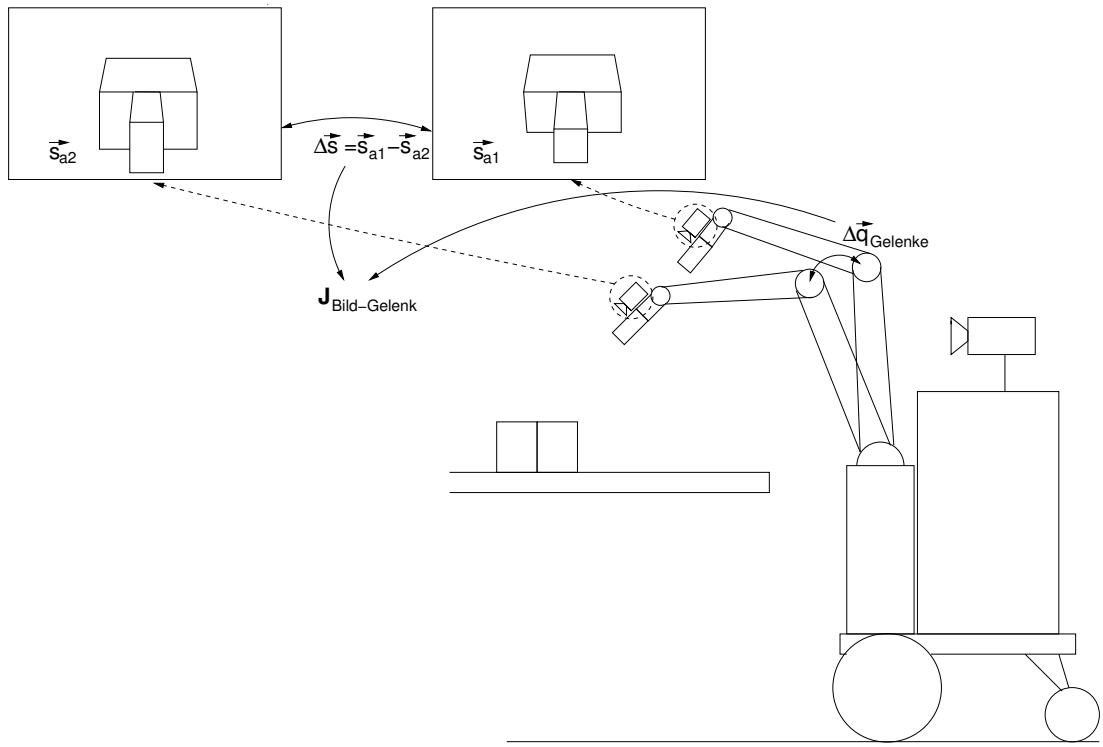


Abbildung 2.11: 2 1/2D *Visual Servoing*. Hier wird aus der aktuellen und der erwünschten Sicht die Orientierung des Manipulators zwischen Ist- und Soll-Position im kartesischen Raum ermittelt. Anschließend wird die Rotation im kartesischen Raum und die Translation im Bildbereich anhand der extrahierten Merkmale geregelt.

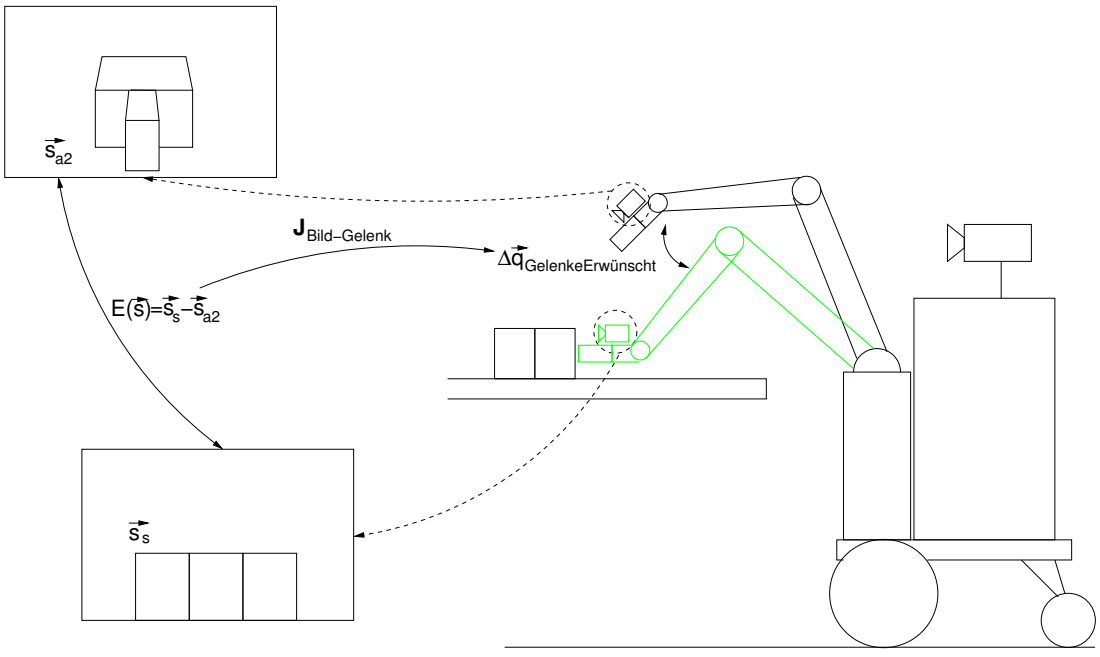
Visual Servoing mit Schätzung

Ansätze zur analytischen Schätzung umfassen die Bestimmung der Bild-Jacobi Matrix und anschließend die Regelung des Roboterarms nach Gleichung 2.9. Jägersand [JN96], [Jäg95] und Hosoda et al. [HA94] drücken das *Visual Servoing* als ein nicht lineares Problem aus, das mit einer Broyden Jacobian Berechnung zu lösen ist. Dadurch wird die Bild-Gelenk Jacobi Matrix $\mathbf{J}_{Bild-Gelenk}$ bestimmt und ständig aktualisiert, während der Roboterarm sich zu einem statischen Ziel bewegt (siehe Abbildung 2.12). Sowohl Ziel als auch Roboter sind von einer stationären *eye-to-hand* Kamera aus sichtbar, das Verfahren ist jedoch auch bei einer *eye-in-hand* Konfiguration anwendbar. Der Ansatz ist für Roboterarme mit drei, sechs und zwölf Gelenken getestet worden. Für den Fall von Verdeckungen werden vordefinierte Trajektorien positionsbasiert durchgeführt. Das Verfahren kann jedoch nicht die Berechnung der wahren Werten der Jacobi Matrix garantieren. Peipmaier [PML99] hat diese Verfahren für ein sich bewegendes Ziel erweitert und an einem Manipulator mit zwei Gelenken evaluiert.

Diese Verfahren berechnen die Jacobi Matrix in Richtung der Zielbewegung. Dadurch ist die Jacobi Matrix nur für einen Teil des Arbeitsbereiches des Roboters gültig. Außerdem kann sie mit der Zeit singulär werden, wenn wenige Gelenke bei der Bewegung teilnehmen. In [SSV98] wird ein Verfahren mit einer *eye-in-hand* Konfiguration vorgestellt, das gezielte Explorationsbewegungen durchführt, um



(a)



(b)

Abbildung 2.12: Schätzung der Bild-Gelenk Jacobi Matrix $\mathbf{J}_{Bild-Gelenk}$. Das Verfahren teilt sich in zwei Phasen auf: nach einer Bewegung wird aus der Änderung der Merkmale und der Gelenkpositionen die Bild-Gelenk Jacobi Matrix $\mathbf{J}_{Bild-Gelenk}$ neu berechnet (a) und zur Bestimmung der nächsten erwünschten Bewegung der Gelenke $\Delta\vec{q}_{GelenkeErwünscht}$ verwendet (b).

Singularitäten zu vermeiden. Dasselbe Ziel hat auch das Verfahren von Buessler et al. [BU97], der die Ansätze zur Schätzung der Jacobi Matrix mit einem Gedächtnis in Form eines hierarchischen, zweischichtigen neuronalen Netzes erweitert.

Ansätze zum Lernen des Bild-Motor Modells verwenden neuronale Netze. In diesem Rahmen hat der Einsatz von *Back Propagation* (BP) Netzen[11] ergeben, dass eine grobe Abbildung schnell zu erreichen ist, eine präzise Abbildung jedoch wegen des globalen Charakters der Netze schwierig ist [Tor00], [BN94]. Deswegen werden in den letzten Jahren lokale Approximatoren verwendet, wie *Cereberal Model Articulation Controller* (CMAC) [Alb75], *Self Organising Maps* (SOM) und *Radial Basis Function* (RBF) Netze [Hay94]. Zum Training wird oft die so genannte *Piaget's Circular Reaction*[12] benutzt.

Miller [MHGK90] hat als einer der ersten einen Ansatz vorgestellt, der keine BP-Netze verwendet. Es handelt sich um ein adaptives System mit zwei CMAC Netzen in einer *eye-in-hand* Konfiguration und vier Freiheitsgraden, das ein Objekt auf einem sich bewegenden Band verfolgt. Ziel ist, den Roboterarm so zu bewegen, dass das Objekt im Bild eine bestimmte Trajektorie verfolgt. Der Roboterarm greift jedoch das Objekt nicht. In [CD98] wird ein dem CMAC ähnliches, selbstorganisierendes Netz, der *Feature* CMAC, präsentiert, um aus einer Aufnahme eines Objektes dessen 2D Lage und Orientierung zu bestimmen. Allerdings geht das Verfahren von einem bekannten, konstanten Abstand zum Objekt am Anfang aus, was es für mobile Manipulatoren ungeeignet macht.

Wegen ihrer Topologie-erhaltenden Eigenschaft sind Kohonen Netze, auch als SOM bekannt, geeignet für das *Visual Servoing* von Manipulatoren. SOMs haben jedoch den Nachteil, dass sie eine lange Trainingsphase beanspruchen [BN94]. Ritter et al. verwenden in [RMS89] 3D-Kohonen Netze für die Steuerung eines simulierten Manipulators mit drei Gelenken und zwei *eye-to-hand* Kameras. Das SOM Netz erlernt die Transformation der Bildkoordinaten des Zielpunktes auf Gelenkstellungen; da nur drei Gelenke zum Einsatz kommen, wird nur die Position und nicht die Orientierung des Greifers angesteuert. Bei der Lernphase werden die Gewichtungen der aktivierten Neuronen angepasst und die Gelenkposition des Roboterarms sowie eine modifizierte Jacobi Matrix abgespeichert (Abbildung 2.13). Die Topologie-erhaltende Eigenschaft der Kohonen Netze wird während der Ausführung eines Greifvorgangs ausgenutzt, um nicht erlernte Zielpositionen aus den aktivierten Neuronen, die den nächsten erlernten Positionen entsprechen, zu interpolieren. In ähnliche Weise arbeiten auch die Ansätze in [WS93] und [JV94]. Erweiterungen, um die Orientierung des Greifers zu erlernen, basieren auf hierarchischen SOMs, wie in [RMS92], [dAT97] und [HSvdSS94]. Zu dieser Gruppe können auch die Verfahren in [Wal96] und [BG97] zugeordnet werden. All diese Ansätze sind jedoch nur auf *eye-to-hand* Konfigurationen mit zwei Kameras anwendbar und setzen keine objektspezifische Trajektorien zum Greifen ein. Ausnahme ist das in [SD98] beschriebene System mit zwei stationären und zwei auf dem Greifer montierten Kameras. Die Bewegung wird in eine grobe und eine feine Phase

[11] Verfahren, die BP-Netze einsetzen, sind unter anderem in [KvdKG90], [SvdLKG95], [vdS95], [WH99], [ZKS00], [WVT96], [YKD97] und [WS97] vorgestellt. All diese Verfahren, außer in [YKD97] und [WS97], haben einen kleinen Aktionsradius und können nicht für große Regionen um das Objekt eingesetzt werden. Weiterhin kann keiner der erwähnten Ansätze eine objektspezifische Trajektorie zum Ziel ausführen.

[12] *Piaget's Circular Reaction* ist eine Methode, um Trainingsdaten für den Roboter zu generieren. Dabei wird eine willkürliche Bewegung erzeugt, vom Roboterarm durchgeführt und die entsprechenden Sensordaten werden gemessen. Das System verwendet die Positions- und der Merkmalswerte zu Beginn und am Ende der Bewegung, um ein neuronales Netz mit einem überwachten Lernverfahren anzupassen. Somit exploriert der Roboter seine Arbeitsumgebung selbstständig und lernt, wie er bei einem gegeben Merkmalsvektor eine erwünschten Position erreichen kann. Das Verfahren ist jedoch nicht zum Erlernen der Hindernisvermeidung verwendbar, da es Kollisionen des Roboters mit Objekten in der Umgebung nicht ausschließen kann.

unterteilt. Ein hierarchisches Kohonen Netz steuert die grobe Bewegung an, bis der Roboterarm in der Nähe des Objektes ist. Danach führen für die Region um das Objekt trainierte BP Netze anhand der Aufnahmen der *eye-in-hand* Kameras die feine Manipulationsbewegung aus. Jedoch kann auch dieses Verfahren keine objektspezifische Trajektorien zum Greifen ausführen.

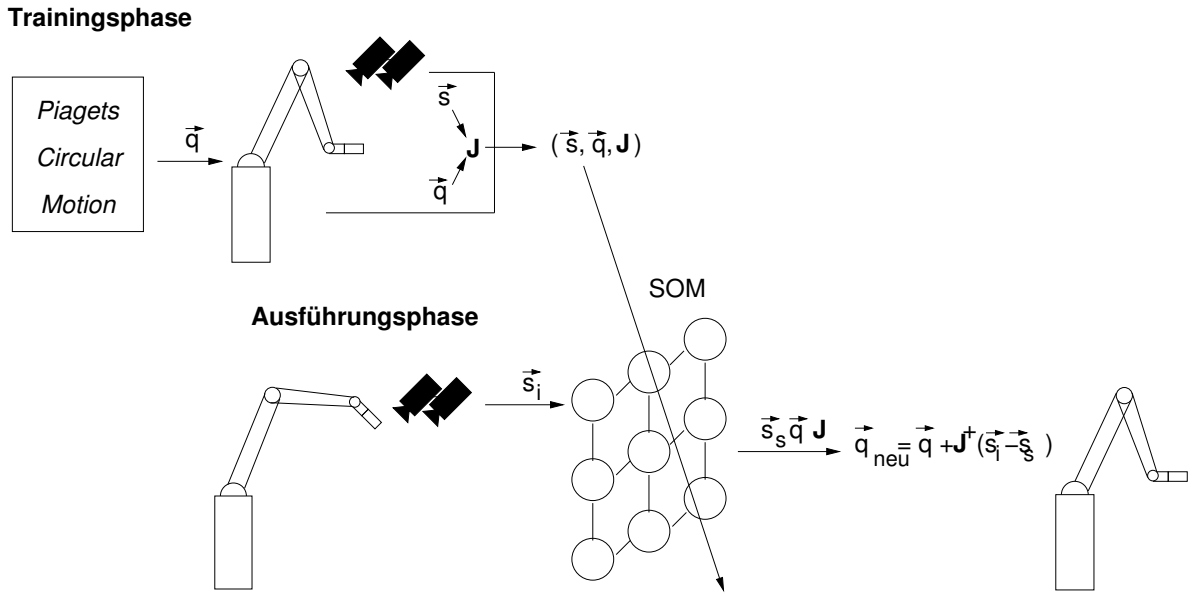


Abbildung 2.13: *Visual Servoing* mit Hilfe von SOMs.

Einen interessanten Ansatz zur Positionsbestimmung eines Objektes stellen Wunsch et al. [WWH97] vor. Sie verwenden ein modifiziertes Kohonen Netz mit einer vordefinierten Topologie, so dass die Neuronen Positionen um das Objekt entsprechen und dadurch die aktivierten Neuronen die Orientierung des Objektes relativ zur Kamera angeben (siehe Abbildung 2.14). Während eines Greifvorgangs können bei jedem Schritt die sichtbaren Merkmalen aus der Menge aller trainierten Merkmale als Eingang für das SOM gewählt werden, um dadurch das Problem der Verdeckungen zu lösen und den Greifer im kartesischen Raum zu führen. Im kartesischen Raum operiert auch das System GripSee [BKM+98], [BKM+99], das zwei *eye-to-hand* Kameras auf einem Kamerakopf verwendet. Hier kommen zwei SOM-ähnliche Netze [WS93] zum Einsatz. Das eine steuert das Fokussieren der Kameras zu einem Punkt im Raum an und das zweite erlernt anhand der Position des Kamerakopfes die entsprechende kartesische Greiferposition, die der Roboterarm annehmen muss. In beiden Verfahren sind für das Objekt entsprechende Griffpositionen definiert, es werden jedoch keine objektspezifische Pfade zur Zielposition am Objekt ermittelt.

Teach-In von Robotern Bis vor kurzem war das klassische *Teach-In* die häufigste angewandte Programmiermethode für Manipulatoren in der industriellen Praxis. Herkömmliche Teach-In Verfahren basieren auf der Aufzeichnung von zeitlichen Zustandsfolgen während eines Vorführungsvorgangs. Damit lassen sich einfache lineare Programmsequenzen generieren [Dil94]. In seiner einfachsten Form werden die bei Vorführungen entstehenden Roboterbewegungen einfach abgespeichert, bei Bedarf aufgerufen und möglichst genau wiederholt.

Programmieren durch Vormachen (*Programming by Demonstration*, PbD) stellt eine Erweiterung des Teach-Ins dar. Ein Benutzer demonstriert die Lösung einer Aufgabe und der Roboter beobachtet,

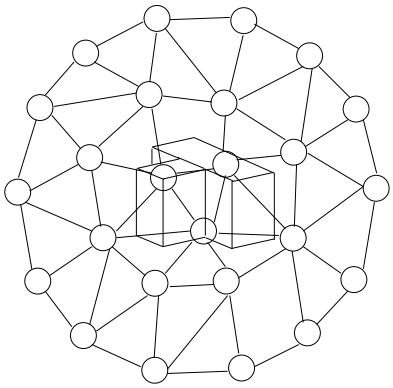


Abbildung 2.14: SOM mit sphärigartiger Topologie zur Schätzung der Kameraorientierung relativ zum Objekt.

analysiert und interpretiert sie, indem er Verallgemeinerungsoperatoren anwendet [DRE+99]. Danach muss der Roboter in der Lage sein, in ähnliche Situationen ähnliche Handlungsweisen aufzurufen [ERZD01].

Die PbD-Ansätze können nach ihrer Abstraktionsebene klassifiziert werden [DRE+99]. Das Ziel kann dabei sein, entweder elementare Fertigkeiten bzw. Verhalten zu lernen oder eher abstraktes, aufgabenspezifisches Know-How zu erlangen[13]. In der ersten Kategorie wird ein Abbildungsmechanismus benötigt, um die Beziehungen zwischen Aktuatoraktionen und Sensordaten zu erlernen. Diese Aufgabe übernehmen in den meisten Fällen neuronale Netze. Diese PbD-Verfahren werden hauptsächlich eingesetzt, um die Abbildung von Kraftvektoren und Momente auf Roboterbewegungen zu erlernen [KD96] [KH95]. Weitere Anwendungen im Bereich der elementaren Fertigkeiten umfassen die Ultraschall-basierte Navigation von mobilen Plattformen [CS00], [YYW03]. Der Einsatz von PbD Verfahren ist jedoch nicht geeignet, um *Visual Servoing* zu erlernen. Der Roboter lernt zwar bildgestützte Imitationsbewegungen durchzuführen, wie in [AYH00], [ZRDZ02] und [JU02], kann aber die Bewegung nicht generalisieren. Das ist darauf zurückzuführen, dass das Erlernen von Kamera-Roboterarm-Bewegungen eine große Anzahl von Beispielen erfordert, wenn es über eine Nachahmung hinausgehen soll. Diese Beispiele können nicht auf praktikable Weise von einem Benutzer generiert werden. Auch die Erfassung der Daten ist problematisch, denn sie erfordert eine große Anzahl von Sensoren.

2.2.2 Reaktive Hindernisvermeidung für Manipulatoren

Die reaktive Hindernisvermeidung hat als Aufgabe, sowohl für den Greifer als auch für die Segmente des Manipulators Ausweichbewegungen zu berechnen, um sie vor Kollisionen zu schützen. Der Forschungsbereich ist mit der Pfadplanung eng verwandt (siehe Abschnitt 2.4.2). Er unterscheidet sich jedoch bezüglich der Reaktivität. Die meisten Pfadplanungsalgorithmen gehen davon aus, dass die Umgebung entweder bekannt oder mit Sensorik genau erfasst ist und sich während der Manipulationsphase nicht ändert. Wenige Planungsansätze berücksichtigen auch eine nur teilweise bekannte Arbeitsumgebung, die aber bei mobilen Manipulatoren oft vorkommt. In diesem Fall muss der Roboter reaktiv neu erfasste Hindernisse vermeiden; eine ständige Neuplanung ist zu aufwendig.

[13] Da man bei der Akquisition von aufgabenspezifischem Know-How von einer Menge vorhandener elementarer Fertigkeiten ausgeht, die nicht erlernt werden [ERZD01], wird diese Gruppe von PbD-Ansätzen hier nicht weiter behandelt.

Die verschiedenen Ansätze zur reaktiven Hindernisvermeidung für Manipulatoren können grob in folgende Kategorien unterteilt werden:

- Methoden, die auf Vektorfeldern basieren,
- Methoden, die SOM Netze anwenden und
- heuristische Methoden.

Vektorfelder Damit ein Roboterarm sich kollisionsfrei bewegen kann, muss die Information über die Abstände der Manipulatorsegmente zu Hindernissen in entsprechende Gelenkbewegungen transformiert werden. Die Abstandsinformation kann leicht im kartesischen Raum bestimmt werden; ihre direkte Transformation im Gelenkraum (*Joint Space*), dessen Achsen durch die Gelenke des Roboterarms definiert sind (siehe Anhang C.1), ist jedoch mit steigender Anzahl der Gelenke zu rechenintensiv. Eine Lösung liefern virtuelle Vektorfelder und die Roboter-Jacobi Matrix. Wenn für ein Hindernis ein virtuelles abstoßendes Kraftfeld definiert ist, übt es auf jedes Segment des Roboterarms einen abstoßenden Kraftvektor aus, der mit Annährung des Segmentes zum Hindernis zunimmt. Für das Ziel wird ein anziehendes Feld definiert (Abbildung 2.15a). Jeder Kraftvektor $\vec{f_i}$ verursacht auf die Manipulatorgelenke Momente, die im Vektor $\vec{\tau_i}$ zusammengefasst und mit der transponierten Roboter-Jacobi Matrix für Segment i, $\mathbf{J}_i^T$, berechnet werden [Cra89] (Abbildung 2.15b):

$$\vec{\tau_i} = \mathbf{J}_i^T \vec{f_i} \tag{2.13}$$

Wenn alle Segmente berücksichtigt werden, dann ist die hindernisvermeidende Gelenkbewegung, die die Kraftvektoren $\vec{f_i}$ verursachen, proportional zur Summe der Moment-Vektoren $\vec{\tau_i}$ (Abbildung 2.15c):

$$\dot{\vec{q}} = \mathbf{K}_V \sum_{i=1..n} \vec{\tau_i} \tag{2.14}$$

Dabei ist $\mathbf{K}_V$ eine Verstärkungsmatrix, $\dot{\vec{q}}$ entspricht dem Vektor der Gelenkgeschwindigkeiten und n gibt die Anzahl der Segmente an. Durch Gleichungen 2.13 und 2.14 kann man die drei-dimensionalen Kraftvektoren auf den hoch-dimensionalen Gelenkraum des Manipulators abbilden. Freiheitsgrade, die nicht für die Hindernisvermeidung notwendig sind, können für zusätzliche Aufgaben benutzt werden (*Null Space Control*). Als Vektorfelder werden oft Potentialfelder [Kha98] eingesetzt, da sie einfach zu berechnen sind. Ansätze, die auf Vektorfeldern basieren, sind unter anderem in [BK98], [BK99], [TK99] und [CK00] beschrieben.

Vektorfeldverfahren sind in der Hindernisvermeidung für Manipulatoren sehr verbreitet. Jedoch bringen sie einige Nachteile mit sich: Verfahren wie *Visual Servoing* oder reaktive Verhalten, die mit anderen Techniken realisiert sind, sind mit diesem Ansatz nicht zu kombinieren. Objektspezifische Greiftrajektorien sind über Freiheitsgrade, die nicht für die Hindernisvermeidung notwendig sind, realisierbar, sie müssen jedoch vom Programmierer aufgabenspezifisch einprogrammiert werden. Die Implementierung erfordert eine ausführliche Kenntnis der Roboterkinematik und aufwendiges Experimentieren für die Bestimmung der Parameter. Außerdem erzwingt das Verfahren die Ermittlung der genauen Objektposition im kartesischen Raum, was bei einer fehlerbehafteten Kalibrierung der Sensorik ein Nachteil ist. Vektorfeldverfahren, die bei mobilen Manipulatoren auch die Bewegung der Plattform berechnen, sind nicht anwendbar auf Plattformen, die nicht zugleich alle ihre Freiheitsgrade ansteuern können (nicht-holonome Roboter). Grund dafür ist, dass die Plattform nicht alle Bewegungen, die die Vektorfelder produzieren, ausführen kann. Erste Ansätze dieses Problem zu lösen, wie in [TLK03] und [PPP02], befassen sich hauptsächlich mit dem kooperativen Transport und nicht mit dem Greifen von Objekten.

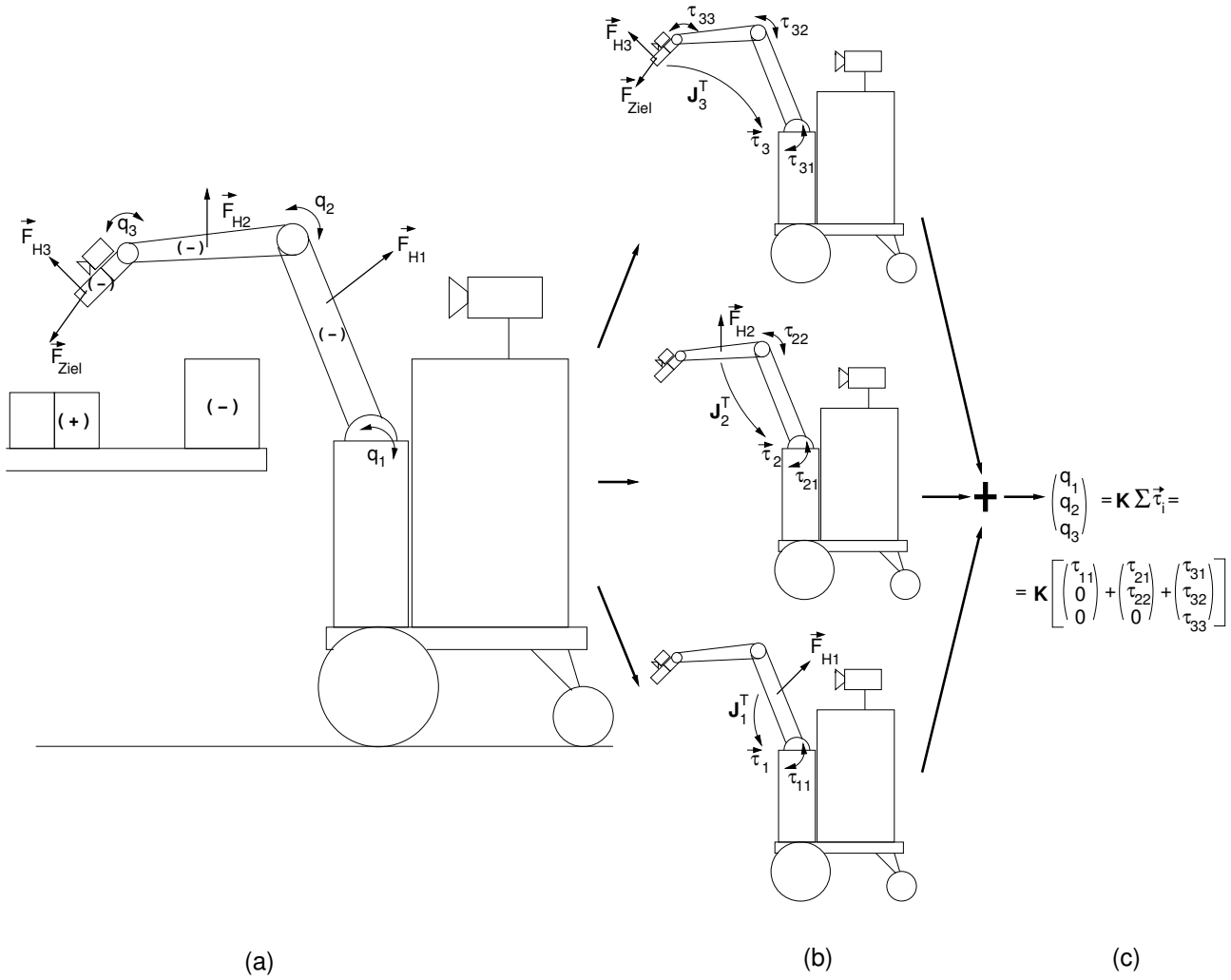


Abbildung 2.15: Vektorfeldverfahren zur Hindernisvermeidung des Manipulators. Hindernisse verursachen virtuelle abstoßende Kräfte während das Ziel eine anziehende Kraft produziert (a). Die einzelnen Kraftvektoren verursachen Momente auf die Gelenke (b), die zusammenaddiert die ausweichende Bewegung produzieren (c).

SOM Netze Insbesondere die SOMs in Form von dynamischen, topologisch organisierten Karten haben Anwendung bei der Hindernisvermeidung gefunden. Wie bei normalen SOMs wird jedes Neuron nur mit seinen Nachbarn verbunden und seine Position auf der Neuronenschicht entspricht einer diskreten Gelenkstellung des Manipulators. Die Werte der Neuronen werden über Differenzialgleichungen bestimmt. Hindernisse, die die entsprechende Gelenkstellung besetzen, bewirken einen negativen Eingang für das Neuron; das Ziel dagegen bewirkt einen positiven Wert. Die Werte von benachbarten Neuronen fließen auch in der Differenzialgleichung ein, wobei nur positive Werte propagiert werden. Dadurch hat ein Ziel globalen und die Hindernisse lokalen Einfluss. Die Trajektorie zum Ziel ergibt sich über ein Gradientenabstiegsverfahren, angewandt auf die Ausgabeschicht des Netzes (Abbildung 2.16). Dadurch wird ein Weg von dem Anfangs- zum Zielneuron gefunden, der über freie Neuronen bzw. Gelenkstellungen verläuft. Der Ansatz kann über die Differenzialgleichungen auch sich bewegende Hindernisse behandeln. Er benötigt keine Trainingsphase und keine explizite Suche über den gesamten Arbeitraum. Das Verfahren kann auf beliebig viele Freiheitsgrade erweitert werden, jedoch nimmt die Berechnungszeit für mehr als drei Gelenke schnell zu. Systeme, die

zur Hindernisvermeidung SOMs einsetzen, sind unter anderem in [YM01], [GKG96] und [SL98] beschrieben. Die Autoren erwähnen jedoch nicht, wie sie aus der Bildinformation die Objektpositionen im kartesischen Raum bzw. im Konfigurationsraum ermitteln. Weiterhin kann man diese Verfahren nur mit einer *eye-to-hand* Konfiguration verwenden.

Verwandt zu den SOM-Verfahren ist auch der Ansatz über neuronale Felder (*Neural Fields*) beim mobilen Manipulator Cora für die Hindernisvermeidung des Greifers [IS01], [IS04]. Die neuronale Felder sind SOM-ähnliche neuronale Netze, die als dynamische, topologisch organisierte Karten eingesetzt werden. Die Neuronenebene stellt dabei eine kartesische Ebene parallel zu einer Tischoberfläche ($x-y$ Position des Greifers) dar; die z-Koordinate des Greifers, die die Höhe zur Tischoberfläche angibt, wird nicht von den neuronalen Feldern beeinflusst sondern nimmt linear zur Zielkoordinate ab. Da keine Orientierung für den Greifer berücksichtigt wird, ist ein objektspezifisches Greifen mit diesem Ansatz nicht möglich.

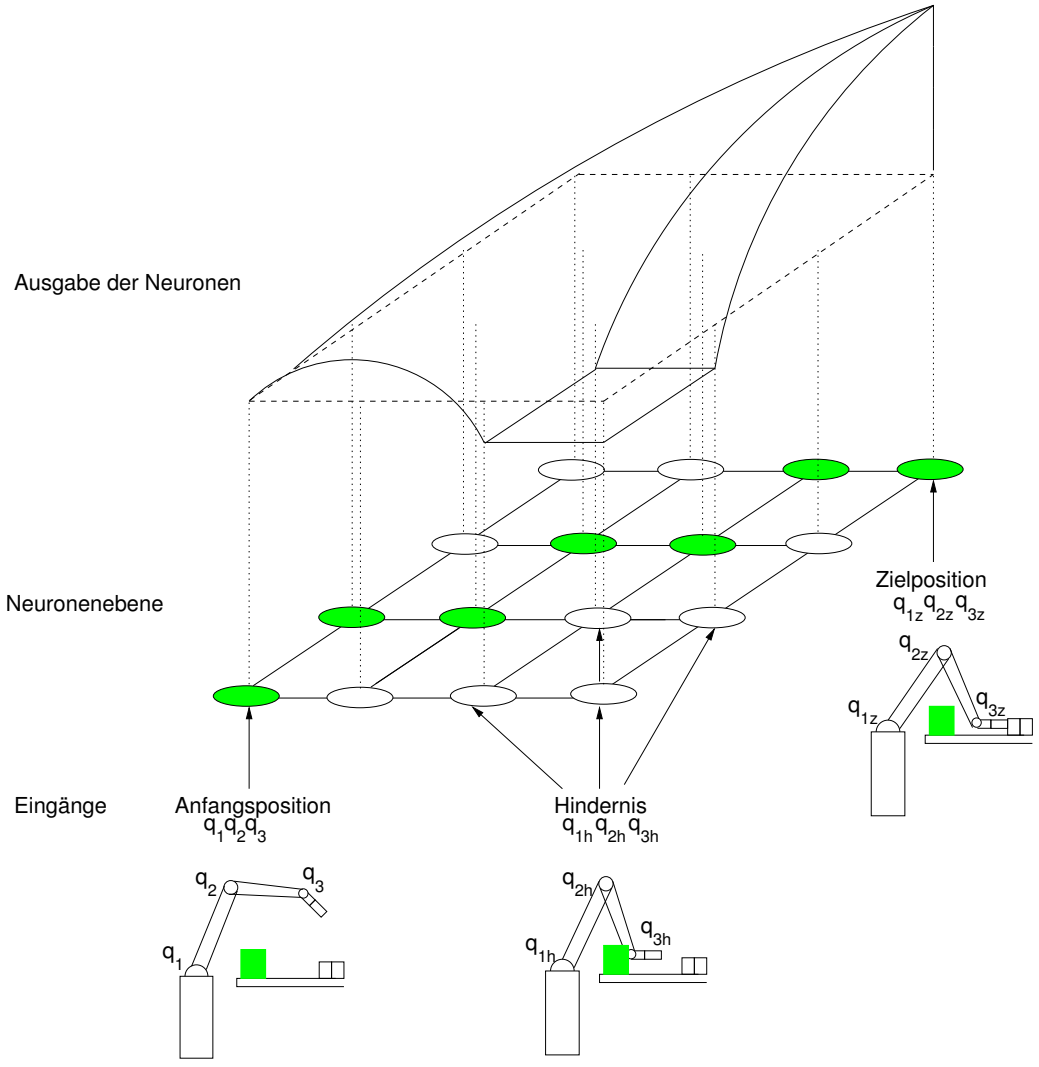


Abbildung 2.16: Hindernisvermeidung mit Kohonen Netze. Die grau schattierte Netzknoten markieren den gefundenen Weg, der von der Anfangsgelenkstellung zur Zielstellung führt.

Heuristische Verfahren Heuristische Verfahren setzen allgemeingültige Regeln ein, die die Hindernisvermeidung garantieren können; jedoch können sie keine optimale Ausweichbewegung berechnen. In dieser Kategorie gehört das Verfahren in [PRG+03] und [HSA95], das Epipolarlinien [Fau93] nutzt, um einen kollisionsfreien, kartesischen Pfad für den Greifer zu finden[14]. Der Ansatz berücksichtigt jedoch die Manipulatorsegmente nicht und kann nur mit zwei *eye-to-hand* Kameras verwendet werden. Er ist nur dann einsetzbar, wenn der Greifer und das Hindernis sichtbar sind und der Manipulator keine Verdeckungen im Bild verursacht.

2.3 Verfahren zur Koordination reaktiver Verhalten

Die Verhaltenskoordinierung oder Verhaltensselektion ist in der Literatur als *Behaviour Coordination Problem* oder *Action Selection Problem* bekannt und behandelt das Problem, wie verschiedene Verhalten miteinander koordiniert bzw. aktiviert werden müssen um eine gestellte Aufgabe zu lösen. Seien n Verhalten $\beta_i(\vec{s_i})$ während der Durchführung einer Aufgabe aktiv. Zu jedem Verhalten ist ein Vektor $\vec{s_i}$ zugeordnet, der die aus den Sensormessungen extrahierten Merkmalen zusammenfasst. Seien die aktiven Verhalten mit der Funktion $B_j = (\beta_{j1}, \beta_{j2}, \ldots, \beta_{jn_j})$ zusammengefasst und $\mathbf{S} = (\vec{s_1}, \vec{s_2}, \ldots, \vec{s_n})^T$ die Matrix der jeweils relevanten Stimuli $\vec{s_i}$. $\vec{R} = (\vec{r_1}, \vec{r_2}, \ldots, \vec{r_n})^T$ fasst die Ausgabevektoren der einzelnen Verhalten zusammen. Dann ist:

$$\vec{R} = B_j(\mathbf{S}) \tag{2.15}$$

Nach Arkin [Ark98] ergibt sich dann die auszuführende Aktion $\vec{\rho}$ des Roboters zu:

$$\vec{\rho} = C(\mathbf{G} \cdot \vec{R}) \tag{2.16}$$

Hierbei kann $\mathbf{G}$ als eine Diagonalmatrix von Gewichtungsfaktoren aufgefasst werden, welche die jeweils aktuelle Priorität oder Anwendbarkeit (*Applicability*) der einzelnen Verhalten beschreibt.

$$\mathbf{G} = \begin{pmatrix} g_1 & 0 & \ldots & 0 \\ 0 & g_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & g_n \end{pmatrix}$$

C ist die Koordinierungsfunktion. Ihre Aufgabe ist die Bestimmung einer resultierenden Ausgabe aus der Menge aller n gewichteten Verhaltensausgaben. Die genauen Werte der Funktion C und der Matrix $\mathbf{G}$ sind im Allgemeinen von der jeweilig aktuellen Situation abhängig und müssen dementsprechend anhand der aktuellen Sensordaten neu berechnet werden.

Die Verhaltenskoordinierung umfasst den Entwurf der Diagonalmatrix $\mathbf{G}$ und der Koordinierungsfunktion C. Dafür werden verschiedene Ansätze vorgeschlagen, die sich grob in zwei Klassen, *Arbitration*- und *Command Fusion*-Mechanismen, einteilen lassen (Abbildung 2.17). Pirjanian gibt in [Pir99] einen guten Überblick über die wichtigsten Verfahren der Verhaltenskoordinierung.

[14] Das Verfahren benötigt zwei *eye-to-hand* Kameras und basiert auf folgendem Prinzip: Ein Pfad hat zwei Projektionen auf den Bildflächen der Kameras, eine für die linke Kamera und eine für die rechte. Eine ausreichende Bedingung für einen kollisionsfreien Pfad ist, dass mindestens eine seiner zwei Bildprojektionen nicht durch Hindernisse verläuft [HA94]. Wenn dies jedoch der Fall ist, dann genügt es, die Stützstellen einer der beiden Pfadprojektionen entlang den Epipolarlinien außerhalb der Hindernisse im Bild zu verschieben. Dadurch entsteht ein kollisionsfreier Pfad im Raum, der durch bildbasiertes *Visual Servoing* ausgeführt werden kann.

Die meisten dieser Ansätze wurden jedoch ausschließlich auf mobile Plattform zur Navigation getestet. Nur wenige, wie [Con89], [MA99], [PAKC00], [SvS00] und [Ste94], wurden auch auf mobilen Manipulatoren angewendet; sie implementieren dennoch nur die sequentielle Anwendung von Verhalten auf dem Manipulator.

2.3.1 Das *Arbitration*-Prinzip

Auf dem *Arbitration*-Prinzip aufbauende Mechanismen wählen ein einzelnes Verhalten aus und geben diesem die komplette Kontrolle über die Aktuatoren (Abbildung 2.18). Im nächsten Zeitschritt findet dann eine erneute Auswahl statt. Sei β_j das ausgewählte Verhalten, dann gilt für Matrix $\mathbf{G}$ und Koordinierungsfunktion C:

$$\mathbf{G}_{arb} = \begin{pmatrix} 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & g_j & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix}$$

$$\vec{\rho} = C_{arb}(\mathbf{G}_{arb} \cdot \vec{R}) = \mathbf{G}_{arb} \cdot \vec{R}$$

Bezüglich des Kriteriums zur Verhaltensauswahl, also der Aufstellung der Matrix $\mathbf{G}$ wird zwischen *Priority-based*, *State-based* und *Winner-takes-all* basierten Mechanismen unterschieden. Beim *Priority-based* Verfahren erfolgt die Auswahl anhand eines fest einprogrammierten Prioritätsschemas [Bro86], [Con89], [Ste94], [GFZ00], [ANH95]. Systeme, die nach dem *State-based* Prinzip arbeiten, besitzen einen endlichen Zustandsautomaten, der das jeweils aktive Verhalten in seinen Zuständen kodiert [KB94], [FZ02], [Tan03], [KAHW00], [SIJ+00], [PAKC00], [KD95]. Beim *Winner-takes-all* Mechanismus enthält der Verhaltensselektor ein so genanntes Aktivierungsnetzwerk, das anhand von Vorbedingungen ein Verhalten auswählt [Mae90], [FMB+97], [AC96], [AUH99], [FPCR99].

Die Systeme, die auf dem *Arbitration*-Prinzip basieren, zeigen ein robustes Verhalten, das die gestellten Aufgaben mit einer hohen Erfolgsrate lösen kann. Dennoch kann der Ansatz das System oft in lokale Minima führen, wenn die ausgeführte Aktion eines Verhaltens vom nächsten Verhalten aufgehoben wird und im nächsten Zeitschritt erneut aktiviert wird (*Deadlock*-Situation). Weiterhin können diese Verfahren zwei Fertigkeiten, die unterschiedliche Freiheitsgrade ansteuern, wie z.B. ein Hindernisvermeidungsverhalten für den Greifer und eins für die Segmente, nicht gleichzeitig anwenden, auch wenn beide in einer kritischen Situation notwendig sind. Sie haben zusätzlich den Nachteil, dass eine schnelle Ausführung der Fertigkeiten und ein schnelles Umschalten zwischen den Verhalten erforderlich sind, damit der Koordinationsmechanismus erfolgreich sein kann. Existieren im System rechenintensive Verhalten, wie z.B. die Hindernisvermeidung für einen Roboterarm, dann ist die Erfolgswahrscheinlichkeit bei *Arbitration*-Mechanismen gering.

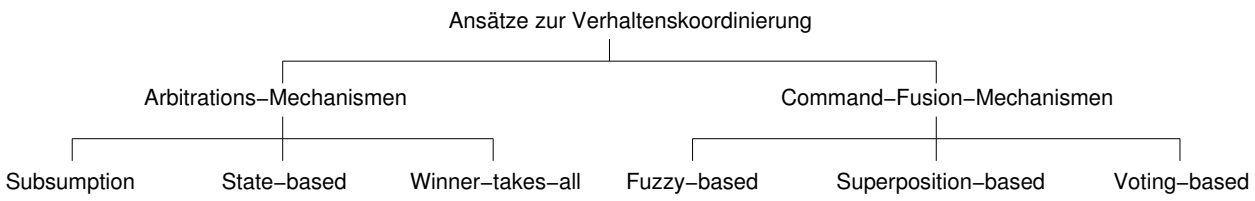


Abbildung 2.17: Gruppierung der Ansätze zur Verhaltenskoordinierung

2.3.2 Das *Command Fusion*-Prinzip

Command Fusion-Algorithmen interpolieren die Ausgaben aller aktiven Verhalten, um die resultierende Aktion zu bestimmen (Abbildung 2.19). Dabei machen sie von den Sensordaten und bekannten oder erlernten Kenntnissen über die Wirksamkeit des Verhalten in bestimmten Situationen Gebrauch. Die Gewichtungsmatrix $\mathbf{G}$ enthält hier mehrere von Null verschiedene Elemente. Pirjanian [Pir99] unterscheidet zwischen *Fuzzy-based*, *Voting-based* und *Superposition-based* Systemen, die in den folgenden Unterabschnitten dargestellt sind.

Command Fusion-Mechanismen haben gegenüber dem *Arbitration*-Prinzip den Vorteil, dass sie Verhalten, die kritisch für die Sicherheit des Roboters sind, zugleich anwenden können. So ist beispielsweise ein schnelles Erreichen des Zielobjektes mit gleichzeitiger Hindernisvermeidung des Greifers und der Segmente möglich. *Command Fusion*-Mechanismen sind weiterhin weniger anfällig auf *Deadlock*-Situationen als die auf *Arbitration* basierende Ansätze. Sie ermöglichen auch das einfachere Implementieren von Roboterverhalten, da komplexe Fertigkeiten in mehreren Komponenten zerlegt werden können, deren Ausgaben über den Koordinationsmechanismus fusioniert werden. Insgesamt produzieren sie glattere Robotertrajektorien und dadurch weniger Belastung für die Mechanik, da kein Umschalten zwischen sich oft wiedersprechenden Verhalten sondern eine gewichtete Fusion der Ausgaben stattfindet. Der Nachteil der *Command Fusion*-Prinzips liegt hauptsächlich bei der Realisierung des Fusions-Mechanismus, der ein langwieriges Experimentieren erfordert, um die geeigneten Gewichtungen für Matrix $\mathbf{G}$ je nach Situation zu bestimmen. Wird zusätzlich ein neues Verhalten im System eingeführt, dann muss dieser Vorgang meistens erneut ausgeführt werden.

Fuzzy Logik

In der mobilen Robotik wurde der Einsatz von Fuzzy Logik eingehend untersucht. Oft sind Verhalten einer mobilen Plattform mit Hilfe von Fuzzy Logik realisiert, wie in [Pau98] oder [ACH97]. Bei Aycard [ACH97] wird für jedes Verhalten des mobilen Roboters eine Regelbasis aufgestellt. Nach der parallelen Defuzzifizierung aller Verhalten ergeben sich die von jeder Fertigkeit vorgeschlagenen Aktionen. In einer zweiten Phase wählt das System eine auszuführende globale Aktion als das arithmetische Mittel aller Aktionen aus. Pirjanian [PM99] zeigt, dass es Situationen gibt, in denen dieser einfache Mechanismus zu unerwünschten Bewegungen führt, weil es zu einem Konflikt zwischen Verhalten mit unterschiedlichen Zielsetzungen kommt. Daher schlägt Saffiotti [SKR95], [Saf97] in der *Multi-Valued* Architektur für die Ansteuerung einer mobilen Plattform ein so genanntes *Context-dependent Blending* vor. Hierbei werden die Fuzzy Ausgaben der einzelnen Verhalten mit Fuzzy Metaregeln gewichtet und anschließend defuzzifiziert. Dadurch wird jedes Verhalten mit seinem eigenen

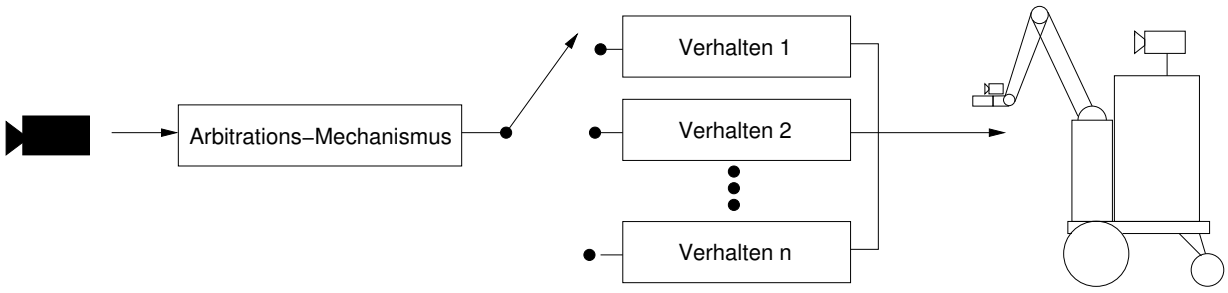


Abbildung 2.18: Ansatz der *Arbitration*-Mechanismen

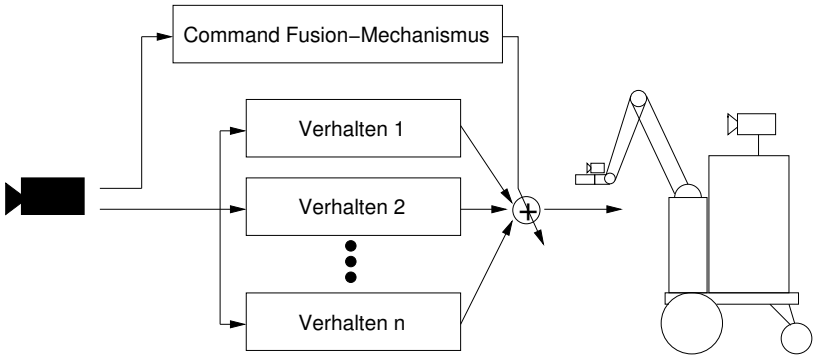


Abbildung 2.19: Ansatz der *Command Fusion*-Mechanismen

Context of Applicability assoziiert, das den Grad der Anwendbarkeit in einer bestimmten Situation vorgibt. Dieses Vorgehen entspricht einer gewichteten Koordinierung und damit der Bestimmung der Matrix $\mathbf{G}$. Alternativ schlägt Bonarini [BILM03] vor, die Voraussetzung für die Aktivierungen von Verhalten und die Ziele von jedem Verhalten in zwei Regelsätzen darzustellen. Aus der Regelbasis der Voraussetzung werden diejenigen Verhalten aktiviert, deren Zugehörigkeitswert über einem Schwellenwert liegt. Ihre Ausgaben werden entsprechend der Regelbasis der Ziele gewichtet und interpoliert. Ähnliche Ansätze sind in [Has96], [PM99] und [ZK99] präsentiert.

Neben der beschriebenen statischen Festlegung von Regeln sollte ein intelligenter Agent in der Lage sein, die Verhaltensselektion und -fusion zu erlernen. Eine Möglichkeit dies zu tun ist der Übergang zu so genannten Neurofuzzy Systemen [vA93], [NODP01]. Durch Rückführung des Fehlers zwischen Ist- und Sollausgabe während der Trainigsphase wird die Anpassung der Gewichte der Neuronen und dadurch die Anpassung der Parameter des Fuzzy Logik Systems ermöglicht. Eine Alternative zu Neurofuzzy bieten genetische Algorithmen, wie in [TLJ97].

Die Verhaltenskoordinierung mit Hilfe von Fuzzy-Regeln erlaubt es, mehrere Zielsetzungen zugleich zu verfolgen. Für die gewünschte Bewegung brauchen die Verhalten keinen exakten Wert anzugeben, sondern können einen unscharfen Bereich vorschlagen und so dem Verhaltensselektor mehr Freiheit bei der Koordinierung geben. Jedoch können bei der Defuzzifizierung unerwünschte Effekte auftreten. Da das Training auf Basis von neuronalen Netzen erfolgt, ist es nicht möglich, ein trainiertes Netz nachträglich zu erweitern, ohne es neu zu trainieren. Deshalb hat bisher kein *Fuzzy-based* Ansatz Anwendung auf einem mobilen Manipulator gefunden.

Voting-based Ansätze

Bei *Voting-based* Ansätzen gibt jedes Verhalten, abhängig vom Zustand der Umgebung und dem Ziel des Gesamtsystems, eine Stimme zugunsten einer Aktion aus der Gesamtmenge der möglichen Aktionen ab. Die verteilten Stimmen werden als Gewichte zur Interpolation benutzt, um die auszuführende Aktion zu berechnen. In diesem Rahmen setzen Peterson und Kragic [PAKC00], [KC00], [KPC02] einen *Voting*-Mechanismus zum Verfolgen von Bildmerkmalen ein und Zalama et al. [ZGPP02] präsentiert eine Architektur mit mehreren Kohonen Netzen zum Erlernen der Verhaltenskoordination.

Ein charakteristisches Beispiel eines *Voting-based* Mechanismus ist die *Distributed Architecture for Mobile Navigation* (DAMN) [Ros95]. Hier weist der so genannte *Mode Manager* jedem Verhalten eine Priorität zu, die von der Gesamtsituation und dem übergeordneten Ziel abhängt. Der *Voter* sammelt

die Stimmen der Verhalten ein und legt so die günstigste Aktion mit den meisten Stimmen fest. Weiterhin kann ein Verhalten in der DAMN Architektur sowohl ein konventionelles reaktives Verhalten als auch ein deliberatives Planungsmodul sein. Dieses diktiert nicht die Entscheidung, sondern gibt, wie alle anderen Verhalten, lediglich eine Stimme ab. Damit besitzt es einen lenkenden Charakter, der aber von anderen reaktiven Verhalten modifizierbar oder auch überschreibbar ist.

Superposition-based Ansätze

Ansatz über Differentialgleichungen Diese Architektur, bei Steinhage [SB97b] auch *Dynamic Approach to Behavioral Organization* genannt, beruht auf der ständigen Auswertung von miteinander gekoppelten Differentialgleichungen. Hierbei wird abhängig von den Sensorinformationen, dem internen Status und logischen Regeln, die das Zusammenspiel der Verhalten betreffen, eine Menge von aktiven Verhalten berechnet. Jede Differenzialgleichung, je eine pro Verhalten i, hat zwei mögliche stabile Zustände: $n_i \approx 1$, der den aktiven Status beschreibt, oder $n_i \approx 0$, der dem passiven Status entspricht. Die resultierende Aktion ergibt sich als Summe aller vorgeschlagenen Aktionen der ausgewählten Verhalten, d.h. $n_i \approx 1$. Zur Bildung der Summe wird keine Gewichtung der aktiven Verhalten untereinander durchgeführt. Der Ansatz wird auf dem mobilen Manipulator Cora angewendet, um die Koordination der Verhalten der mobilen Plattform, des Manipulators und des Kamerakopfes zu realisieren [SvS00]. Dabei verfügen die verschiedenen Verhalten über unterschiedliche Zeitkonstanten; die Plattform reagiert langsam, der Manipulator mittelschnell, um die Bewegungen der Plattform nachzukompensieren, und der Kamerakopf am schnellsten. Zu jedem Zeitpunkt hat jedoch ein einzelnes Verhalten volle Kontrolle über einen Aktuator.

Hashimotos Verfahren für schnelles, bildgestütztes Greifen mit einem statischen Manipulator stützt ebenfalls die Verhaltensauswahl und -fusion auf Differenzialgleichungen [HNI01]. Dabei verfügt das System über vier Verhalten zum bildgestützten Verfolgen eines Objektes, zum Erreichen einer Zielposition, zur Manipulation eines Objektes und zur Hindernisvermeidung des Manipulators. Diese werden mit Hilfe von Differenzialgleichungen anhand der aktuellen Sensorinformation gewichtet und interpoliert. Ein großer Nachteil des Verfahrens ist jedoch, dass es nicht leicht erweiterbar ist. Weiterhin können die Parameter der Differenzialgleichungen nur durch aufwendiges Experimentieren gesetzt werden.

Motor Schemas Dieser Ansatz läßt sich ebenfalls in die Klasse der *Superposition-based* Verfahren einordnen. Er basiert auf künstlichen Potentialfeldern, wobei ein Ziel eine anziehende Kraft und Hindernisse abstoßende Kräfte auf den Roboter auswirken. Durch geeignete Interpolation der Potentialfelder über Vektoraddition entstehen so genannte *Motor Schemas*, die für einfache Aufgaben spezialisiert sind, wie Zielführung, Hindernisvermeidung oder Wandfolgen. Somit entsprechen sie den reaktiven Verhalten der 3T-Architektur.

Die Aktivierungsreihenfolge der *Motor Schemas* bestimmt ein *State-based* Ansatz über endliche Zustandsautomaten (*Finite State Automata*) [CACE99]. Dabei können mehrere *Motor Schemas* gleichzeitig aktiv sein und aus den Sensordaten eine Bewegung für der Roboter vorschlagen. Zur Bestimmung der auszuführenden Roboterbewegung werden die *Motor Schema*-Ausgaben mit sensorabhängigen Gewichten multipliziert und durch eine Vektorsummation fusioniert. Die Art und Weise der Gewichtungsberechnung bleibt der jeweiligen Implementierung überlassen. Manche Ansätze verwenden hierfür lernfähige Verfahren, um die geeigneten Gewichtungen für verschiedene Situationen zu

erlernen, wie das Lernen mit Bewerter [Bal97] oder genetische Algorithmen [RABP94]. Die Verhaltensfusion wurde auf mehreren Plattformen erfolgreich getestet, es liegen jedoch keinen Angaben über einen Einsatz auf Manipulatoren vor. MacKenzie [MA99] setzt z.B. auf seinen mobilen Manipulator *Motor Schemas* ein, die Verhaltensfusion ist jedoch nur für die mobile Plattform realisiert.

Weitere Superposition-based Verfahren Einen weiteren Ansatz schlägt Hager vor, der einem *Visual Servoing* System primitive Fertigkeiten hinzufügt [Hag97]. Für jede Fertigkeit werden eine Fehlerfunktion definiert, die erforderlichen Merkmale bestimmt und die entsprechende Jacobi Matrix berechnet. Ein komplexes Verhalten lässt sich über ein Übereinandersetzen der Jacobi Matrizen (siehe Gleichung 2.11) definieren. Zusätzliche Aufgaben können auch durchgeführt werden, solange sie Freiheitsgrade beeinflussen, die bei der Hauptaufgabe nicht gebraucht werden (*Null Space Control*). Der Ansatz ist jedoch dadurch beschränkt, dass er nur Verhalten, die mit bildbasiertem *Visual Servoing* implementiert sind, fusionieren kann.

2.3.3 Manipulator-Plattform Koordination

Die Koordination des Manipulators mit der mobilen Plattform ist ein wichtiges Thema bei der mobilen Manipulation. Während bei manchen mobilen Manipulatoren die kinematischen und dynamischen Eigenschaften der mobilen Plattform es erlauben, Plattform und Manipulator als Einheit zu berücksichtigen und für beide zusammen zu planen, wie in [BK98], erzwingen einige Eigenschaften, wie eine nicht-holonome Plattform, unterschiedliche Reaktionsgeschwindigkeiten von Plattform und Roboterarm oder die ungenaue Lokalisierung der Plattform, die Behandlung des Roboters als zwei unterschiedliche Systeme [Yam94]. Diese werden nur grob miteinander koordiniert, falls eine parallele Bewegung von Plattform und Manipulator benötigt wird, wobei die Plattform eine grobe Bewegung durchführt und der Manipulator feine Anpassungen macht. Bei den meisten Aufgaben eines mobilen Manipulators ist jedoch keine parallele Bewegung von Plattform und Manipulator notwendig[15]. Die Plattform transportiert den Roboterarm zum Einsatzbereich und der Roboterarm führt anschließend die Manipulationsaufgabe aus[16].

2.4 Planung

Ein Planer ist eine funktionale Einheit, die für die Erstellung einer Sequenz von Aktionen verantwortlich ist. Diese kann entweder geometrisch in der Form von Pfaden sein (Pfadplanung) oder in der Form von abstrakten Aktionen wie `fahr zur Tür`, `öffne Tür` und `fahre durch Tür` (Aktionsplanung[17]). Im zweiten Fall gibt es zwei Möglichkeiten. Bei wissensbasierten Planern findet die Planung unter Berücksichtigung der Roboterfertigkeiten und der vorhandenen Information über

[15] Koordinierte Bewegung von mobiler Plattform und Manipulator werden hauptsächlich bei Transportaufgaben benötigt, die jedoch in dieser Arbeit nicht berücksichtigt werden.

[16] Die mobile Manipulatoren, die in Abschnitt 2.6 beschrieben sind, gehören alle außer Cora zu der zweiten Kategorie und betrachten Plattform und Manipulator als zwei unterschiedliche Agenten, die separat angesteuert werden. Die Koordination erfolgt dann meistens über den *Arbitrations*-Mechanismus der vermittelnden bzw. der deliberativen Ebene, indem der Plattform und dem Manipulator die entsprechenden Aufgaben zugewiesen werden.

[17] Bei einem mobilen Manipulator ermittelt die Aktionsplanung für eine gestellte Aufgabe eine Reihenfolge von Aktionen, die mobile Plattform und Manipulator ausführen müssen, damit der Roboter seine Aufgabe erfolgreich löst. Somit ist sie auch für die Koordination der Aktionen von mobiler Plattform und Manipulator verantwortlich.

die Welt, die aus einer symbolischen Darstellung der Umgebung stammt, statt. Bei reaktiven Planern wird jedoch die nächste Aktion anhand der aktuellen Sensorinformation, dem Ergebnis der letzten Aktion und vorhandener Teilpläne generiert. In Abbildung 2.20 ist die Unterteilung der Planer in verschiedenen Gruppen zu sehen. Das Gebiet der Planung ist sehr umfangreich und umfasst einen Großteil der KI-Forschung, weshalb hier nur ein Überblick gegeben wird.

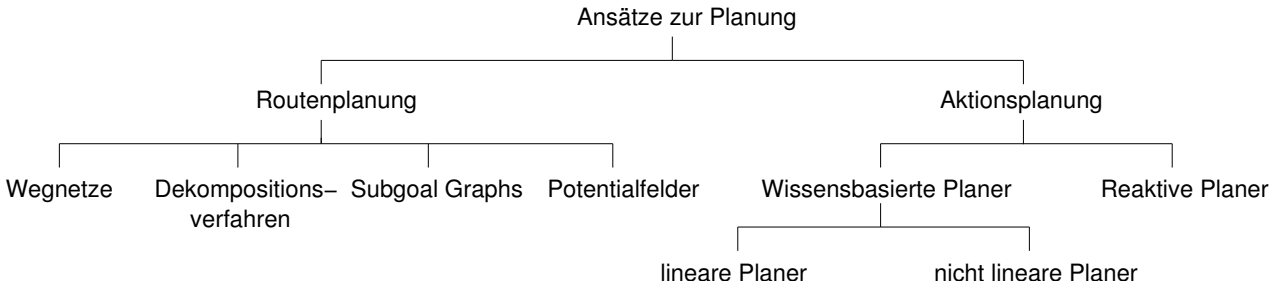


Abbildung 2.20: Gruppierung der Planungsansätze

2.4.1 Aktionsplanung

Wissensbasierte Planung

Wissensbasierte Planer operieren durch Symbolmanipulation auf abstrahierten Weltmodellen und bestimmen einen effizienten Lösungsweg für eine Aufgabe. Die Pläne werden auf Basis von Information über die Umwelt erstellt, die vor der Planerstellung vollständig vorliegen muss. Die Sensorik des Roboters ist dann dafür verantwortlich, in konsistenter Weise physikalische Objekte der Umgebung auf Symbole im Weltmodell abzubilden. Bei fehlerbehafteter Ausführung eines Plans oder bei unerwarteten Änderungen in der Umgebung, sind eine Aktualisierung des Weltmodells und eine Neuplanung erforderlich. Diese Neuberechnung des vollständigen Lösungsplanes ist derart zeitaufwendig, dass sie nicht in Echtzeit zu bewältigen ist.

Planungsverfahren können anhand des Suchraumes, in dem der Planer die Lösung eines Problems sucht, unterteilt werden. Eine Möglichkeit bildet der Zustandsraum[18]. Der Zustandsraum wird durch einen Graphen repräsentiert, wobei jeder Knoten des Graphen für einen Weltzustand steht, jede Kante für eine Aktion bzw. Fertigkeiten des Roboters, die die Umgebung von einem Zustand in den nächsten überführt. Viele Planungssysteme, wie LOPE [GMB97], BURIDAN [KHW93] und Prodigy [HV98], bauen den Zustandsgraph über *backward-chaining* vom Zielzustand aus, indem sie Aktionen rückwärts anwenden, um nicht erfüllten Voraussetzungen für schon eingefügte Aktionen zu erreichen (Abbildung 2.21). Falls im Graphen mindestens ein Knoten mit den aktuellen Zustand präsent ist, dann existiert mindestens ein Plan, der zum Ziel führt. Deswegen erfolgt anschließend eine Suche vom Anfangs- zum Zielzustand im Graphen. Um die Suche zu beschleunigen, verwenden einige Planer Suchregeln (*Search Control Rules*), die den Suchraum beschränken, eine Suchrichtung stark befürworten und ein Ziel oder Aktion in einer bestimmten Situation auswählen oder ausschließen [HV98]. Im Gegensatz dazu expandiert Graphplan, ein weiteres Planungssystem, zuerst einen Suchgraphen von der aktuellen Situation aus, indem es nacheinander Aktionen und dadurch entstehende Zustände als Knoten einfügt, bis die Vorbedingungen des Zielzustandes erfüllt sind (*forward*

[18] Der Zustandsraum umfasst alle mögliche Zustände, die die Welt bzw. die Einsatzumgebung annehmen kann.

chaining) [Wel99]. In der nächsten Phase wird eine *backward-chaining* Suche im Graphen durchgeführt und nach einem Plan gesucht, der das Problem löst. Ansonsten wird der Graph um zwei Ebenen erweitert und der Prozess wiederholt sich. Bekannte Planer, die auf dem Graphplan Algorithmus basieren, sind der IPP [KNHD97], der SGP [WAS98], der PGraphplan [BL99] und der DT-Graphplan [PC03].

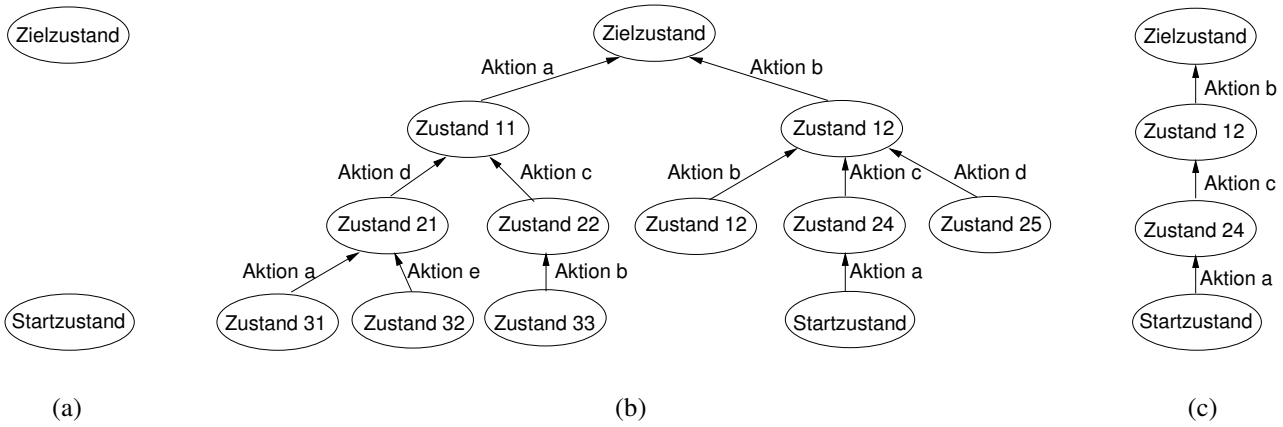


Abbildung 2.21: Aufbau des Plangraphes über *backward-chaining*.

Eine Alternative zum Zustandsraum ist die Suche im Raum der Teilpläne. Der Planraum ist hier auch durch einen Graphen repräsentiert, jedoch steht jeder Knoten für einen Teilplan und jede Kante für eine Aktion, die den Plan verfeinert und näher an die Erfüllung seines Zieles bringt. Zur Laufzeit werden die Pläne weiterentwickelt und falls ein besserer Plan entsteht, ersetzt der neue Plan den alten. Somit entspricht das Planen einer Suche im Planraum. Beispiele von Planern die im Raum der Teilpläne operieren, sind der XFRM [BM94], [BAB$^+$01], der *Partial Ordered Planner* (POP) und der UCPOP [PW92].

SAT-Planer basieren auf *Propositional Satisfiability* Methoden [Wel99]. Sie erhalten ein Planungsproblem als Eingabe und generieren eine logische Formel (*propositional Formula*), die, wenn erfüllt, die Existenz eines Planes beweist. Aus dem Beweis der Formel kann man einen Plan ermitteln, der das Planungsproblem löst. Einige Implementierungen vom SAT-Planer berücksichtigen auch Wahrscheinlichkeiten, um die Fehlerwahrscheinlichkeit einer Aktion in der Planung einzubeziehen [WN96]. Bekannte SAT-Planer sind Blackbox [KS98], GSAT [SLM92], WALKSAT [SKC96] und der Planer von Nasa's Deep Space One [WN96], [MNPW98],

Heuristische Planungsverfahren definieren Kosten für jeden Zustand, die den Abstand des Zustandes vom Zielzustand beschreiben, um dadurch die Graphsuche in die Richtung, die den meisten Erfolg verspricht, zu führen. Danach berechnen Heuristiken die Kosten für jeden Zustand und setzen einen *hill-climbing*, *best-first* oder A* Algorithmus ein, um die Graphsuche zu beschleunigen. Heuristische Planungsverfahren, wie die HSP, HSPr und HSP2 Planer [BG01] oder GRT [RV03], können jedoch in lokale Minima führen oder finden nicht immer eine Lösung, auch wenn es eine gibt. Einen ähnlichen Ansatz verfolgen die *Markov Decision Processes* (MDP), die aber eine Suche im Planraum ausführen [DKKN93], [DB94]. Weiterhin können sie Erfolgswahrscheinlichkeiten für Aktionen bei der Planerstellung berücksichtigen.

Die meisten der beschriebenen Verfahren aus dem Bereich der wissensbasierten Planung gehen davon aus, dass eine vollständige und konsistente Beschreibung der Umgebung vorliegt. Stimmt der eigent-

liche Zustand nicht mit dem Weltmodell überein oder tritt eine unerwartete Situation ein, dann sind eine konsistente Aktualisierung des Weltmodells und eine zeitaufwändige Neuplanung erforderlich. Je komplexer der Anwendungsbereich ist und je mehr mögliche Aktionen bzw. Teilpläne existieren, die der Roboter ausführen kann, desto schwieriger und rechenaufwändiger wird die Suche nach einem Lösungsplan. Deshalb schlagen Levesque und Reiter [LR99] vor, anstatt eines Planers ein so genanntes *high-level* Programm zu verwenden. *High-level* Programme können, wie wissensbasierte Planer, eine ausführliche Suche zur Lösungsfindung durchführen. Sie können aber auch Entscheidungspunkte beinhalten, um während der Planausführung anhand der aktuellen Sensordaten eine entsprechende Aktion aus mehreren möglichen auszuwählen. Im Gegensatz zur klassischen wissensbasierten Planung können *high-level* Programme Information über die Reihenfolge der auszuführenden Aktionen beinhalten. Dadurch beschränkt sich der Suchraum erheblich und eine schnelle Erstellung von Roboterplänen ist möglich. GOLOG (*alGOL in LOGic*) ist eine Programmiersprache, die auf dem Situationskalkül[19] basiert [LR99] und das Erstellen von *high-level* Programmen ermöglicht. Sie ist verwandt mit den SAT-Planungsverfahren [Wel99]. Statt jedoch nach einer Folge von Aktionen zu suchen, die den Agenten von einem Ausgangszustand in einen Zielzustand überführt, wird hier eine Folge von Aktionen gefunden, die eine gültige Ausführung eines nichtdeterministischen *high-level* Programms darstellt. Nachfolger[20] von GOLOG haben die Sprache erweitert und sie handhabbar gemacht. IndiGolog [GL98] erweitert z.B. GOLOG um die Möglichkeit der Wahrnehmung. Es kann mittels Aktionen selbst Informationen über den aktuellen Zustand der Umwelt gewinnen und den Plan entsprechend anpassen.

Reaktive Planung

Reaktive Planer operieren nicht auf einem vollständigen Weltmodell, sondern richten sich bei der Auswahl des nächsten durchzuführenden Schrittes nach der durch die Sensoren erfassten Situation. Sie bestehen meist aus einer Sammlung von Teilplänen, die in entsprechenden Situationen und bei erfüllten Vorbedingungen auszuführen sind. Dadurch benötigen reaktive Planer kein Weltmodell und kein großes Vorwissen zu Beginn einer Mission. Es ist keine erneute vollständige Planung bei geänderten Umgebungsbedingungen notwendig, es wird nur der entsprechende Teilplan ausgewählt und ausgeführt. Dadurch ist das System an die neue Situation schnell anpassbar und die Echtzeitfähigkeit der Planerstellung sichergestellt. Reaktive Planer umfassen unter anderem Schoppers *Universal Plans* (UP) [Sch87], die *Reactive Action Packages* (RAP) [FKPS95] und den Planer von Bagchi [Bag94], der im Rahmen des ISAC-Projektes entstanden ist [BCN$^+$95]. Zur reaktiven Planung können auch die *State-based* Ansätze zur Verhaltensaktivierung zugeordnet werden (Abschnitt 2.3.1).

Reaktive Planer können sowohl nicht vorausplanen als auch nicht garantieren, dass sie eine Lösung finden, obwohl eine solche existiert. Durch ihre Reaktivität haben sie aber den Vorteil, dass Unsicherheiten über Objektposition bzw. Zustände der Umgebung nicht zum Misserfolg des gesamten Plans führen. Außerdem brauchen sie, im Gegensatz zu wissenbasierten Planern, keine große Informationsmenge über die Umgebung zu Beginn der Planung.

[19] Das Situationskalkül ist eine Logik zweiter Ordnung und eignet sich zur Repräsentation von Situationen [McC68]. Die Grundelemente sind Aktionen und Situationen. Das Situationskalkül beschreibt, wie sich die Welt infolge von Aktionen verändert.

[20] Nachfolger von GOLOG umfassen unter anderem cc-Golog [GL00], pGolog [GL00] und ConGolog [GLL97].

2.4.2 Routenplanung für Manipulatoren

Die Routenplanung, auch als Pfadplanung (*Pathplanning*) bzw. Bewegungsplanung (*Motion Planning*) bekannt, berechnet einen geometrischen, kollisionsfreien Weg von einem Start- zu einem Zielpunkt. Sie operiert auf einer geometrischen Darstellung des Arbeitsbereiches des Roboters. Diese kann den kartesischen Raum oder den Konfigurationsraum[21] (*Configuration Space*) darstellen, eine alternative, Roboter-spezifische Darstellung des Arbeitsbereiches, die den Roboterarm auf einen Punkt reduziert [LP83].

Pfadplaner können vollständig (*complete*) oder heuristisch (*heuristic*) sein. Vollständigen Planungsverfahren, die lange Berechnungszeiten in Anspruch nehmen, finden eine Lösung oder beweisen, dass es keine gibt. Heuristische Planer sind schnell, finden aber nicht immer einen Weg zum Ziel [CHS95]. Die bis heute implementierten vollständigen Planer sind abhängig von der Auflösung der Umgebungsdarstellung und deswegen spricht man von *resolution-complete* Algorithmen [Lat99]. Wegen der langen Berechnungszeiten, die einem mobilen Manipulator nicht zur Verfügung stehen, sind sie als on-line Planer nicht einsetzbar.

Eine weitere charakteristische Eigenschaft der Routenplaner betrifft die Region, die zur Planung berücksichtigt wird. Globale Methoden berücksichtigen den gesamten Arbeitsbereich. Dabei führen die meisten die zeitaufwendige Berechnung des Konfigurationsraumes vor dem eigentlichen Planungsprozess aus. Opportunistische (*opportunistic*) Planungsmethoden [Bag98] berechnen dagegen Regionen des Konfigurationsraumes sobald ein Pfad vorgeschlagen wird, der durch diese Regionen verläuft. Lokale Planer berücksichtigen bei der Planung eine begrenzte Region um eine Gelenkstellung. Dadurch sind sie relativ schnell, haben jedoch den Nachteil, dass sie zu lokalen Minima, wie Sackgassen, führen können.

Nach Latombe [Lat99] können die verschiedenen Pfadplanungsansätze für einen einzelnen Roboter zu den folgenden vier Kategorien zugeordnet werden:

- Wegnetzmethoden (*Skeleton* bzw. *Roadmap*),
- Dekompositionsverfahren (*Cell Decomposition*),
- Zwischenzielgraph-Verfahren (*Subgoal Graph*) und
- Potentialfeld-Verfahren (*Potential Fields*).

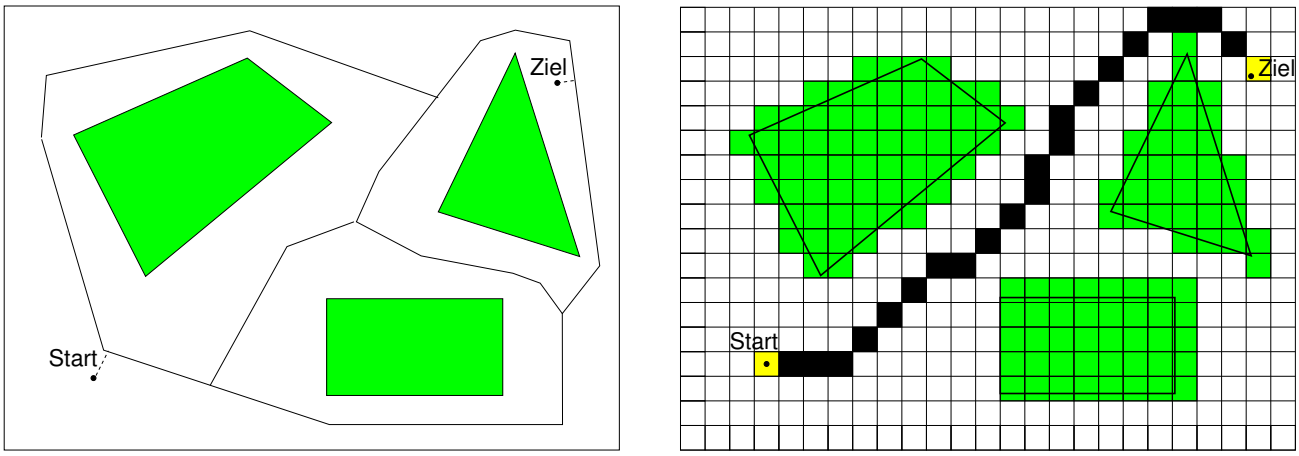
Die drei ersten Verfahren berechnen den gesuchten Pfad mit Hilfe eines Graphen, der den Freiraum des Roboters darstellt. Dadurch wird die Pfadplanung zu einem Graphensucheproblem transformiert.

Wegnetzmethoden repräsentieren die Umgebung durch eine Menge von befahrbaren, hindernisfreien Wegsegmenten, die miteinander verbunden sind (Abbildung 2.22a). Die Wegsegmente werden in einem Graphen abgespeichert und bilden das Routennetz des Roboters. Bei der eigentlichen Planung wird eine Verbindung des Start- und Zielpunktes zum Wegnetz hergestellt. Danach schließt sich

[21] Hier bilden die Freiheitsgrade des Roboterarms die Achsen des Raumes und somit entspricht jede Position im Konfigurationsraum einer eindeutigen Gelenkstellung. Dadurch ist ein im Konfigurationsraum definierter Pfad vom Manipulator immer durchführbar. Zur Planung von hindernisfreien Pfaden im Konfigurationsraum müssen zuerst die Gelenkstellungen, die eine Kollision des Manipulators mit einem Hindernis in der realen Welt verursachen, als besetzt markiert werden. Dabei kann ein Hindernis in mehreren Regionen des Konfigurationsraumes abgebildet sein, da der Greifer eines Roboterarms die gleiche kartesische Position mit unterschiedlichen Gelenkstellungen annehmen kann (siehe Anhang C.1).

eine Suche im Graphen an. Beispiele für Wegnetzmethoden sind die *Visibility Graphs* [Nil69], das *Voronoi Diagram* [OY82], sowie der *Hierarchical Generalized Voronoi Graph* (HGVG) [CKR97], eine on-line Alternative zu den Voronoi Diagrammen für mehrere Freiheitsgrade. Wegnetzmethoden haben den Nachteil, dass sie für Roboterarme mit mehr als vier Freiheitsgraden zu zeitaufwendig sind.

Dekompositionsverfahren erzeugen nicht nur einzelne, befahrbare Strecken. Sie unterteilen die gesamte Umgebung ist in befahrbaren und nicht befahrbaren Regionen, so genannte Zellen (Abbildung 2.22b). Die Zellen bilden die Knoten des Graphes und benachbarte Zellen werden über Kanten im Graph miteinander verbunden. Dekompositionsverfahren sind in exakte und nicht exakte Verfahren unterteilt. Exakte Verfahren stellen den freien Raum mit Hilfe von Polygonen dar, deren Nachbarschaftsbeziehungen in dem Graphen archiviert sind. Zu dieser Kategorie gehört z.B. die *Trapezoidal Cell Decomposition* [Cha87]. Nicht exakte Dekompositionsverfahren verwenden eine Rasterdarstellung, die die Umgebung auf ein meist vorgegebenes Raster abbildet. Der Inhalt einer Rasterzelle gibt Auskunft darüber, ob sich an dieser Position ein Objekt befindet oder nicht. Beispiele für nicht exakte Dekompositionsverfahren umfassen die *Approximate Cell Decomposition* [LP81], *Quadtrees* [Pau98], [KTP98] und die *Wavefront Expansion* [RH96]. Verfahren, die SOMs zur Hindernisvermeidung einsetzen, können auch den nicht exakten Dekompositionsverfahren zugeordnet werden (siehe Abschnitt 2.2.2). Wie die Wegnetzmethoden haben auch Dekompositionsverfahren den Nachteil, dass sie für mehr als vier Freiheitsgrade zu zeitaufwendig sind. Bessere Ergebnisse liefern opportunistische Verfahren, wie in [WHW98] und [War93]. Sie testen eine Rasterzelle des Konfigurationsraumes auf Kollisionen mit Hindernissen, nur wenn ein vorgeschlagener Pfad durch sie führt. Ist die Rasterzelle belegt, dann werden die benachbarten Zellen getestet. Dies ermöglicht bei einfachen Umgebungen eine schnelle Pfadberechnung.



(a) (b)

Abbildung 2.22: Wegnetzmethoden (links) und Dekompositionsverfahren (rechts).

Subgoal Graphs sind hybride Ansätze, die Einsatz bei vieldimensionalen Konfigurationsräumen finden. Dabei produziert ein globaler Planungsalgorithmus wichtige Konfigurationen, die in einem Graphen verwaltet werden. Solche Konfigurationen befinden sich z.B. in der Nähe von Hindernissen.

Lokale Algorithmen übernehmen die Pfadplanung zwischen den Konfigurationen; meistens handelt es sich um einfache Linienverbindungen (Abbildung 2.23a). In diese Kategorie gehören die Pfadplaner von Canny und Lin [CL93], die ein Wegnetz zur globalen und Potentialfelder zur lokalen Planung einsetzen, und der Ansatz in [FW91], der ein grobes globales und mehrere feine lokale Gittermodelle hierarchisch einsetzt.

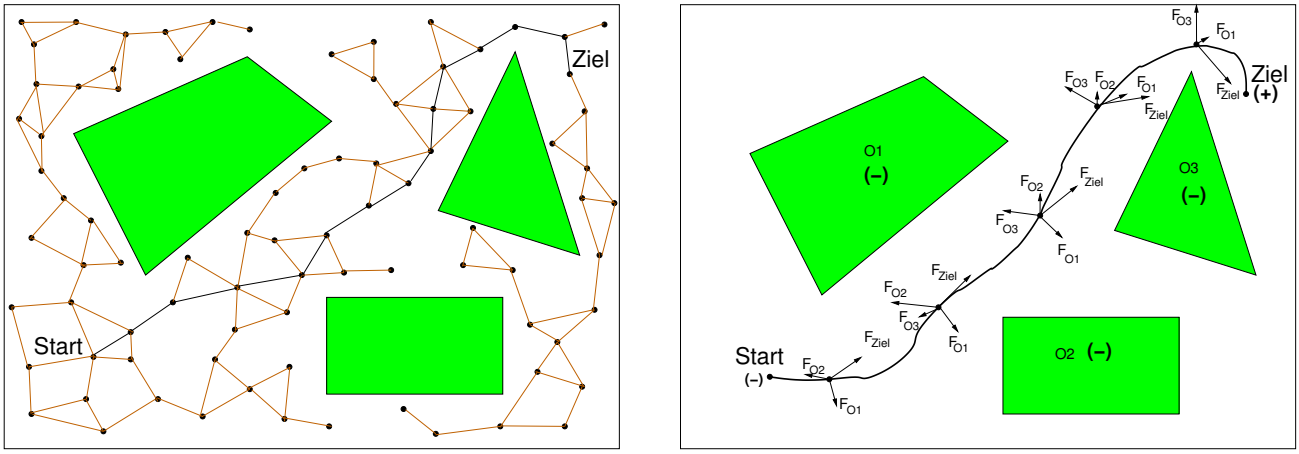
Bei den *Probabilistic Roadmap Methods* (RPM) werden zufällig Punkte im Konfigurationsraum generiert, wobei in der Nähe von Hindernissen gezielt mehr Punkte verteilt und auf Kollisionsfreiheit getestet werden. Nah beieinanderliegende Punkte können über einen lokalen Planer miteinander verbunden werden, falls ein Pfad zwischen diesen Punkten existiert. Die Punkte bilden die Knoten eines Graphen, die mit dem lokalen Planer gefundene Verbindungen stellen die Kanten des Graphen dar. Ist kein Weg vom Start- zum Zielpunkt vorhanden, dann werden zusätzliche Punkte in der Nähe von Hindernissen zufallsverteilt eingefügt [KL94a], [KL94b], [HST94], [ABD+98a], [AW96], [ABD+98b]. Die verschiedenen Arbeiten in diesem Bereich unterscheiden sich bei der Generierungsstrategie der Punkte und bei der Auswahl des lokalen Planers.

Subgoal Graphs mit opportunistischen Pfadplanungsverfahren nehmen eine hierarchische Dekomposition des Konfigurationsraumes vor, wie SANDROS [CH98], das ZZ-Planungsverfahren [Gla91], [Bag96], [Kug94], der BB-Ansatz [Bag98], die Arbeiten von Hourtash et al. [HT01] und von Hsu et al. [HLS99], [HKLR00]. Die Planung beginnt mit einer groben Auflösung des Konfigurationsraumes. Falls der lokale Planer keinen Weg von Anfangs- zur Zielposition findet, dann generiert der globale Planer Gelenkstellungen in einer feineren Auflösung des Konfigurationsraumes und überprüft sie auf Kollisionen. Der lokale Planer versucht die Gelenkstellungen miteinander zu verbinden und das Schema wiederholt sich bis ein Weg zum Ziel gefunden ist.

Eine Kombination von Dekompositionsverfahren und *Subgoal Graphs* ist das *Decomposition-based Planning* [BK00] für mobile Manipulatoren. Die Planung wird in Unterprobleme unterteilt, die einzeln schnell lösbar sind. Das Verfahren bestimmt im kartesischen Raum zuerst den Freiraum, der Start- und Zielposition verbindet und über ein Mindestvolumen verfügt, und approximiert ihn über Sphären. Dabei handelt es sich um ein Problem mit drei Freiheitsgraden, das schnell mit einem Dekompositionsverfahren, wie der *Wavefront Expansion*, zu lösen ist. Danach berechnen lokale Methoden, wie z.B. Potentialfelder, im entsprechenden Konfigurationsraum die genaue Bewegung des Roboters.

Potentialfelder belegen alle Objekte in der Umgebung mit einer abstoßenden, das zu erreichende Ziel mit einer anziehenden virtuellen Ladung. Dadurch wird eine virtuelle Kraft auf den Roboter ausgeübt [Ark98], [Kha98] (siehe Abbildung 2.23b). Bei einer mobilen Plattform entspricht der resultierende Kraftvektor der Bewegungsrichtung des Roboters. Bei einem Manipulator müssen jedoch die Momente, die der Kraftvektor verursacht, über die Gelenke des Roboterarms propagiert werden, um die Bewegung der Segmente zu bestimmen (siehe auch Abschnitt 2.2.2).

Die Potentialfeldverfahren haben den Vorteil, dass sie schnell zu berechnen sind. Deswegen werden sie auch oft zur Navigation bzw. reaktiven Hindernisvermeidung des Roboter eingesetzt. Jedoch können sie leicht in ein lokales Minimum führen und daher das Erreichen des Zielpunktes unmöglich machen. Einige Gruppen lösen das Problem, indem sie gezielt Rauschen einführen, um aus lokalen Minimas zu entkommen. Die meisten Ansätze benutzen jedoch die Potentialfelder als lokales Verfahren, um die Planung zwischen den Stützstellen von berechneten Pfaden zu verfeinern oder Pfade dynamisch anzupassen. Beispiele für solche Ansätze sind die *Virtual Springs* [MC96], die *Elastic Strips* [BK98], [BK99] und der *Randomized Path Planner* (RPP) [BLL92].



(a) (b)

Abbildung 2.23: *Subgoal Graphs* (*Probabilistic Roadmap*) (links) und Potentialfelder (rechts).

2.5 Virtuelle Realität und Robotik

Der Einsatz von virtuellen Umgebungen ist in der Robotik immer mehr verbreitet. Sie dienen hauptsächlich zur Visualisierung und Teleoperation, wie in [BAH01] [HH00], [HHF+95], [FD94], für die automatische Kinematik- und Aktionsplanung bzw. Simulation des Roboters [KBC+02], [FR00], [GL02], [BBD+00], [BAHK95] oder als Leitstand für das Aufgabenmanagement eines Roboters [Sch01], [HHF+95], [STB99]. In diesen Fällen wird die virtuelle Umgebung benutzt, um die Roboterbewegungen und die Einsatzumgebung für einen Operateur zu visualisieren. Ansätze, die Aufnahmen von virtuellen Sensoren für die Robotersteuerung verwenden, sind eher Ausnahmefälle. Gracanin et al. [GVTM99] und Yeh et al. [YYW03] setzen z.B. simulierte abstandsmessende Sensoren ein, um das Navigationssystem von mobilen Robotern in einer virtuellen Umgebungen zu evaluieren bzw. zu erlernen.

Virtuelle Umgebungen haben gegenüber 2D Simulationsumgebungen den großen Vorteil, dass sie die Simulation von virtuellen Kameras ermöglichen. Dies machen sich die optischen Lokalisierungsverfahren von Schmitt [Sch01] und Gracias [GSV01] zunutze. Sie vergleichen Aufnahmen virtueller Kameras aus einem Modell der Umgebung mit realen Aufnahmen, um die mit der Odometrie berechneten Roboterposition zu korrigieren.

Einen Schritt weiter geht der Ansatz in [NRT95], der visuelle Information zur autonomen Steuerung von Avataren in einer virtuellen Umgebung einsetzt. Das Verfahren wird jedoch nicht in eine reale Umgebung transferiert. In eine ähnliche Richtung geht auch der Ansatz von Marchand [MC00], der *Visual Servoing* einsetzt, um die Bewegung einer freien Kamera in einer virtuellen Umgebung bezüglich eines im Bild sichtbaren Zieles zu steuern bzw. zu regeln. In diesem Ansatz kann die Kamera Hindernissen und Verdeckungen ausweichen. Da sie aber nicht auf einem Roboter montiert ist, werden keine Bild-Manipulator Transformationen berücksichtigt. Der Ansatz soll in Kinofilmen, Animationen und Spielen Einsatz finden und ist nur für Nutzung in virtuellen Umgebungen implementiert.

Einsatz in einer realen Umgebung findet das Teleoperationsystem in [Nel01], das visuelles Feedback aus einer virtuellen Umgebung verwendet, um die Sollwerte für das *Visual Servoing* bei der Zielpositi-

on zu definieren. Ein Operateur muss vorher den Roboter in der virtuellen Umgebung zur Zielposition geführt haben, damit der reale Roboter eine ähnliche Operation durchführt (*Teaching by Showing*). Allerdings findet hier kein Erlernen von bildgestützten Verhalten statt. Ähnlich arbeitet auch das Konzept der aufgabenorientierten Roboter-Programmierung (*Task Oriented Teleprogramming of Robots*) [BAH01], auch als *TeleSensor Programming* bekannt [BAH94], [BAHK95], [HLBS96]. Das System VirtualWorkcell von Paulin [Pau04], das erst vor kurzem vorgestellt wurde, setzt eine virtuelle Umgebung zur Planung von *Visual Servoing* Strategien und Trajektorien, die von einem realen Manipulator ausgeführt werden. Das System setzt virtuelle Kameras ein und sammelt Bilddaten, um eine effiziente *Visual Servoing* Strategie für einen Roboterarm zu entwickeln. Allerdings findet auch bei diesem System kein Erlernen von bildgestützten Manipulatorverhalten oder vom Koordinationsmechanismus statt.

2.6 Spezielle Systemarchitekturen für mobile Manipulation und verwandte Ansätze

In diesem Unterkapitel wird der Stand der Technik bei der mobilen Manipulation präsentiert und auf die wichtigsten Eigenschaften bzw. Defiziten existierender Systeme eingegangen. Zugleich werden weitere Systeme aus dem Bereich der Robotik vorgestellt, die ähnliche Charakteristika zu dem in dieser Arbeit entstandenen mobilen Manipulator aufweisen. Tabellen 2.2 und 2.3 beinhalten die wichtigsten Eigenschaften dieser Systeme.

MacKenzie und Arkin [MA99] haben MissionLab, eine Modifizierung von AuRA [Ark98] mit zwei Ebenen, eingesetzt, um Proben aus Gefäßen bildgestützt ohne die Berücksichtigung von Hindernissen zu sammeln. Sowohl AuRA als auch MissionLab basieren in der reaktiven Ebene auf *Motor Schemas*, wobei das Ziel eine anziehende Kraft und Hindernisse abstoßende Kräfte auswirken. Die *Motor Schemas* werden über Vektoraddition interpoliert, um ein Vektorfeld als Verhalten zu produzieren, das den Roboter ansteuert. Die Architektur unterstützt zwar die gewichtete Fusion[22] der *Motor Schema*-Ausgaben, jedoch findet beim Roboterarm eine solche Fusion nicht statt.

Petersson hat mit einem Szenario zum Türöffnen die *Mobile Manipulation Control Architecture* (MMCA) und mit einer *pick-and-place* Aufgabe die *Distributed Control Architecture* (DCA), eine Generalisierung von MMCA, getestet [Pet02]. Beide Architekturen verfügen über zwei Ebenen. Zum Greifen von Objekten wird ein objektspezifischer Griff berechnet und der Manipulator bildgestützt dorthin geführt [Kra01]. In beiden Architekturen existiert keine Hindernisvermeidung für den Manipulator und keine Verhaltenskoordination.

Im Rahmen des Projektes Morpha [Lay00] wurden mehrere Architekturen mobiler Manipulatoren entwickelt. Eine ist die Architektur von Care-O-bot II von Hans et al. [HB01][23], die eine 3T-ähnliche Struktur aufweist, jedoch keine bildgestützten Fertigkeiten beinhaltet und keine Verhaltenskoordination unterstützt. In [HB01] wird auch nicht erwähnt, ob der Manipulator bei einem Greifvorgang Hindernisse vermeiden kann.

[22] Drei alternative Fusionsmethoden sind möglich: sequentiell (*temporal sequencing*), gegenseitig ausschließend (*competitive*) und kooperativ (*cooperative*). Im letzten Fall kann eine gewichtete Vektorsummation der *Motor Schemas* stattfinden. Allerdings kommt diese Fusion nur bei der mobilen Plattform zum Einsatz.

[23] Eine modifizierte Version ohne Planungsebene soll auf dem Assistenzroboter rob@work für Industrieanwendungen Einsatz finden [HNS01].

		MacKenzie [MA99]	**Petersson** [Pet02]	**Hans** [HB01]	**Iossifidis** [IS04]	**Knoop** [KVZ+04]
Mobiler Manipulator		√	√	√	√	√
Jahr		1999	2001-02	2001	2001-04	2004
Architektur	3 Ebenen	-	-	√	-	-
	2 Ebenen	√	√	-	-	-
	hybrid	-	-	-	√	√
Sensorik	Kameras	√	√	?	√	√
	Laserscanner	√	-	?	-	-
	Künstliche Haut	-	-	?	√	-
Hindernis-vermeidung	Greifer	-	-	?	√	?
	Segmente	-	-	-	×	?
Verhaltens-fusion	ungewichtet	◯	-	-	◯	?
	gewichtet	◯	-	?	◯	?
Lern-fähigkeit	Verhalten	◯	-	?	-	-
	Verhaltensfusion	◯	-	-	-	?
Objektspezifisches Greifen		-	√	?	-	√
Virtuelle Umgebung		-	-	-	-	-

Tabelle 2.2: Eigenschaften und Vergleich von Systemen zur mobilen Manipulation (√: eingesetzt, ◯: möglich, aber nicht benutzt, -: nicht vorhanden, ?: keine Angaben, ×: Manipulator-spezifisch).

Der mobile Manipulator Cora von Iossifidis et al. [IBT+02], [IS04] ist ebenfalls im Morpha-Projekt entstanden und besteht aus einer mobilen Plattform, einem Roboterarm mit sieben Freiheitsgraden und einem Greifer, einem Stereokamerakopf und einer künstlichen Haut (*Artificial Skin*) auf dem Manipulator. Die eingesetzte Architektur ist die *Dynamic Approach to Behavioral Organization* [SB97b]. Hierbei wird abhängig von den Sensorinformationen, dem internen Status und logischen Regeln, die das Zusammenspiel der bildgestützten Verhalten betreffen, eine Menge von aktiven Verhalten berechnet und parallel angewandt. Der Ansatz erlaubt auch die Fusion der Ausgaben von parallel ausgeführten Verhalten sowie das Erlernen der Verhaltensaktivierung in einer Explorationsphase. Dies findet jedoch nur für die mobile Plattform statt, Experimente für den Roboterarm werden nicht präsentiert. Der Greifer kann während eines Greifvorganges Hindernisse über den Ansatz der neuronalen Felder (*Neural Fields*) [IS01] vermeiden. Dabei handelt es sich um SOM-ähnliche neuronale Netze, die als dynamische, topologisch organisierte Karten eingesetzt werden. Die Neuronenebene stellt dabei eine kartesische Ebene parallel zu einer Tischoberfläche ($x - y$ Position des Greifers) dar; die z-Koordinate wird nicht über das Netz angesteuert, sondern nimmt linear zur Zielkoordinate ab. Der Manipulator kann auch Hindernisse für die Segmente ausweichen, indem er den siebten Freiheitsgrad des Manipulators einsetzt, um über die Hindernisse zu kommen. Der realisierte Ansatz ist jedoch nicht auf Manipulatoren mit weniger als sieben Freiheitsgraden anwendbar. Da keine Orientierung für den Greifer berücksichtigt wird, ist ein objektspezifisches Greifen mit diesem Ansatz nicht möglich; der Greifer erreicht die Objekte immer von oben.

Der Serviceroboter Albert2 von Knoop et al. [KVZ+04] basiert auf eine hybride Architektur, die eher mit den verhaltensbasierten Systemen verwandt ist. Das System setzt den PbD-Ansatz ein, um aufgabenspezifisches Know-How für den Roboter zu erlangen. Dabei geht man von einer bekannten Menge elementarer Fertigkeiten aus, dessen Aktivierungsreihenfolge durch den PbD-Ansatz erlernt

		Wichert [WN01]	**Blase** [BP98]	**Fu** [FSZ00]	**Brunner** [BAH94]	**Hashi.** [HNI01]	**TAURO**^{3}
Mobiler Manipulator		√	-	-	-	-	√
Jahr		2001-03	1998	2000	1994	2001	2003-04
Architektur	3 Ebenen	√	-	-	-	√	√
	2 Ebenen	-	-	-	-	-	-
	hybrid	-	-	-	√	-	-
Sensorik	Kameras	√	√	√	√	√	√
	Laserscanner	√	-	-	√	-	-
	Künst. Haut	√	-	-	-	-	-
Hindernis-vermeidung	Greifer	-	√	-	-	√	√
	Segmente	√	-	√	-	√	√
Verhaltens-fusion	ungewichtet	√	√	-	-	-	√
	gewichtet	-	-	-	-	√	√
Lern-fähigkeit	Verhalten	-	√	√	√	-	√
	Verhaltensfusion	-	-	-	-	-	√
Objektspezifisches Greifen		√	-	-	-	?	√
Virtuelle Umgebung		-	-	√	√	-	√

Tabelle 2.3: Eigenschaften und Vergleich von Systemen zur mobilen Manipulation sowie von weiteren verwandten Systemen (√: eingesetzt, ◯: möglich, aber nicht benutzt, -: nicht vorhanden, ?: keine Angaben). **TAURO**^{3} ist der in dieser Arbeit realisierte mobile Manipulator.

wird; für die Fertigkeiten selbst findet kein Training statt. In [KVZ+04] gibt es keine Angaben über eine Hindernisvermeidungskomponente für den Manipulator. Weiterhin wird nicht erwähnt, ob das System die Fusion der Ausgaben von unterschiedlichen Fertigkeiten unterstützt.

Der mobile Manipulator von Wichert et al. hat einen ähnlichen Aufbau wie Cora, verfügt jedoch zusätzlich über einen 2D Laserscanner [vWKN+02]. Der Manipulator setzt die Stereokamera und den Laserscanner ein, um die Position der Hindernisse im Raum zu bestimmen [WN01], [WNvWK02]. Eine Planungskomponente berechnet einen kollisionsfreien Anfangspfad im Gelenkraum, der von dem *Reactive Plan Executor* als Vorgabe für die reaktive Steuerung des Manipulators benutzt wird. Dieser fusioniert nach dem *Superposition-based Command Fusion*-Prinzip die Pfade von der Planungskomponente und von zwei vektorfeldbasierten Hindernisvermeidungsmodulen, ohne sie jedoch zu gewichten. Der Roboter kann acht unterschiedliche Aufgaben ausführen, wie das Greifen oder das Ablegen eines Objektes, und zehn unterschiedliche Objekte erkennen. Für jedes Objekt ist ein Griff definiert, der Roboterarm führt jedoch keine objektspezifische Trajektorie aus, um zum Ziel zu gelangen. Außerdem werden die Hindernisvermeidungsverhalten nur auf Gelenke angewendet, die für das Erreichen des Zieles nicht verwendet werden; dadurch beschränken sich die Bewegungsmöglichkeiten des Manipulators bei der Hindernisvermeidung. Das System verfügt auch nicht über die Möglichkeit zu lernen. Somit werden die Parameter der verwendeten Algorithmen durch aufwendiges Experimentieren gesetzt. Weiterhin ist keine deliberative Planungsebene im System vorhanden: diese ist durch eine Mensch-Maschine Schnittstelle ersetzt. Daher kann der Roboter keine komplexe Aufgaben selbstständig lösen.

Für die bildgestützte Zielführung eines stationären Manipulators in einer *eye-in-hand* Konfiguration trainieren Blase et al. [BP98] RBF-Netze, um Objekte anhand des optischen Flusses zu lokalisie-

ren. Das Zielobjekt erzeugt dann ein anziehendes und Hindernisse abstoßende virtuelle Vektorfelder, die zu einem Feld interpoliert werden. Der Greifer wird durch den resultierenden Kraftvektor im kartesischen Raum zum Ziel geführt. Zusätzlich werden die Segmente des Manipulators in der Endposition auf Kollisionen mit Hindernissen überprüft. Eine Hindernisvermeidung für die Gelenke über die gesamte Bewegung findet jedoch nicht statt. Das Verfahren berücksichtigt außerdem nicht die Orientierung des Zielobjektes.

Eine interessante Arbeit über Hindernisvermeidung für stationäre Manipulatoren präsentieren Fu et al. [FSZ00]. Sie setzen ein SOM-ähnliches Netz ein, um den *Perception Control Manifold* (PCM) eines stationären Manipulators mit drei Freiheitsgraden zu erlernen. Der PCM entsteht aus der Erweiterung des Gelenkraumes um den Merkmalsraum[24]. Das Netz lernt die Topologie des PCMs und dessen Ausgangsneuronen werden mit Greiferpositionen assoziiert, die aus der bekannten Geometrie des Manipulators leicht zu berechnen sind. Während der Ausführung eines Greifvorgangs werden Neuronen, deren zugeordnete Gelenkstellungen von Hindernissen besetzt sind, entfernt. Danach wird mit einem Gradientenabstiegsverfahren auf der Neuronen-Ebene nach einem Pfad zum Zielpunkt gesucht (siehe auch Abschnitt 2.2.2). Interessant ist zusätzlich, dass das Training in einer virtuellen Umgebung mit Daten aus virtuellen Kameras stattfindet. Zu den Nachteilen des Verfahrens zählt dass es nicht mit einer *eye-in-hand* Konfiguration anwendbar ist und dass die Erweiterung auf mehr als drei Freiheitsgrade schwierig ist. Beim Greifvorgang ist die Orientierung des Zielobjektes nicht berücksichtigt, wodurch man das Objekt nicht aus einer vordefinierten Richtung greifen kann.

Das Konzept der aufgabenorientierten Roboter-Programmierung (*Task Oriented Teleprogramming of Robots*) [BAH01], auch als *TeleSensor Programming* bekannt [BAH94],[BAHK95],[HLBS96], ist im Rahmen des Projektes Rotex für Weltraumroboter entstanden. Um Totzeiten bei der Verbindung zwischen Basisstation und Roboter in den Orbit zu kompensieren wird eine prädiktive Grafiksimulation eingesetzt, die dem Operateur einen Überblick über die Ausführung der kommandierten Roboteroperationen verschafft, sowie die Konsistenz des zurückgemeldeten Systemstatus mit der vorausberechneten überprüft [BAH01]. Die Grafiksimulation kommt auch bei der bildbasierten Robotersteuerung zum Einsatz, um die Sensormuster an der Soll-Position des Roboterarms nach dem Prinzip des *Teaching by Showing* [BAH94], [HLBS96] zu generieren. Als Sensoren für die Ansteuerung des Manipulators werden Laser, Kraft-Moment Sensoren und eine Stereokamera verwendet und in der Grafiksimulation entsprechend simuliert. Damit das Konzept funktioniert, ist eine genaue Kalibrierung des Systems erforderlich [BAH94]. Das vorgestellte Verfahren geht davon aus, dass Modell und reale Umgebung exakt übereinstimmen. Das ist bei mobilen Manipulatoren, die in nur teilweise bekannten Umgebungen operieren, nicht der Fall. Das System ist dadurch nicht in der Lage, mit Hindernissen oder neuen Objektkonstellationen ohne die Unterstützung des Operateurs zurecht zu kommen. Es werden keine Verhalten parallel angewendet oder ihre Koordination erlernt. Das System hat auch keine Möglichkeit auf neue Information zu reagieren und einen existierenden Plan entsprechend anzupassen. Diese muss der Operateur registrieren und eine neue Planung veranlassen.

Hashimoto et al. präsentieren in [HNI01] eine interessante Architektur mit drei Ebenen für das schnelle Greifen mit einem stationären Manipulator. In der niedrigsten Ebene, dem *Control Layer*, sind Geschwindigkeitsregler für den Roboterarm realisiert. Der *Planning Layer* entspricht der Verhaltensebene der 3T-Architektur und generiert Pfade zum Verfolgen, Greifen und Behandeln von Objekten sowie zur Hindernisvermeidung des Manipulators. Der *Adaption Layer* hat die Aufgabe, die verschiedenen Pfade des *Planning Layers* anhand der aktuellen Sensordaten zu gewichten, zu fusionieren und dementsprechend die Sollwerte des *Control Layers* einzustellen. Die Gewichtungen werden mit Hil-

[24] Als Merkmal gilt die Pixelposition des Greifers im Bild.

fe von Differenzialgleichungen[25] ermittelt. Das System kann dadurch die Ausgaben von mehreren Verhalten fusionieren. Allerdings ist es mit neuen Verhalten nur bedingt erweiterbar und die Gewichtungsparameter müssen durch aufwendiges Experimentieren bestimmt werden. Die Architektur verfügt nicht über eine deliberative Planungsebene, die bei einem mobilen Manipulator notwendig ist, um komplexe Aufgaben auszuführen. In den vorgestellten Experimenten ist die Hindernisvermeidung erst nach dem Greifen aktiviert.

2.7 Ein Konzept zur mobilen Manipulation

Das in dieser Arbeit vorgestellte Konzept zur mobilen Manipulation versucht die Defizite von existierenden Systemen im Bereich der Hindernisvermeidung, der Verhaltenskoordination und dem Erlernen von Bildgestützten zu vermeiden, um dadurch einen robusten Einsatz in einer teilweise bekannten Umgebung zu ermöglichen. Zwei Ansätze werden hier verfolgt: einerseits ist eine erweiterte drei-Ebenen Architektur[26] implementiert, um ein schnelles, situationsabhängiges Verhalten des mobilen Manipulators zu erlangen (Abbildung 2.24). Anderseits werden die bildgestützten Komponenten der reaktiven und der vermittelnden Ebene in einer virtuellen Umgebung erlernt und anschließend auf den mobilen Manipulator übertragen.

Beim Roboterarm beinhaltet die unterste, reaktive Ebene die elementaren Verhalten, die die Kameras mit dem Manipulator verbinden. Dadurch kann der Roboter sofort auf Änderungen der Umgebung reagieren und seine Bewegung an die neue Situation anpassen. Die Verhalten setzen die Information der Roboterkameras ein, um den Manipulator bildgestützt zum Ziel zu führen, das Objekt im Sichtfeld der Greiferkamera zu halten und den Greifer und die Manipulator-Segmente vor Kollisionen zu schützen. Sie sind entweder explizit programmiert oder mit neuronalen Netzen erlernt. Im letzteren Fall werden lokale Approximatoren verwendet[27], um die Abbildung der gewünschten Änderung der Bildmerkmale auf entsprechende Roboterbewegungen zu realisieren. In dieser Ebene wird auch ein Routenplaner als Verhalten eingesetzt, um besonders zu Beginn einer Greifbewegung eine grobe Bewegungsrichtung vorzugeben und dadurch den Manipulator vor *Deadlocks* zu bewahren und Konflikte aufzulösen. Der Routenplaner beschränkt seine Planung auf eine begrenzte Region des Arbeitsraums um die aktuelle Gelenkstellung des Roboterarms und die Position des Zielobjektes.

Die vermittelnde Ebene implementiert den Verhaltensaktivierungs- und Verhaltenskoordinationsmechanismus. Die Verhaltensaktivierung wählt nach dem *Arbitration*-Prinzip mit einem endlichen Zustandsakzeptor (*Finite State Acceptor*, FSA) die Gruppen von parallel auszuführenden Fertigkeiten aus und aktiviert sie. Der Koordinationsmechanismus basiert auf dem *Superposition Command Fusion*-Prinzip und ist mit *Bayesian Belief Netzwerken* (BBN) [Cha91], [Hec95] realisiert, die die Anwendbarkeit der Verhalten bestimmen und die Ausgaben der aktiven Verhalten zum Roboter miteinander interpolieren. Der Einsatz der BBN erlaubt das Lernen der Anwendbarkeit der Verhalten in verschiedenen Situationen, sowie die Modellierung der Fehlerwahrscheinlichkeit bei der Merkmalsextraktion aus den Bilddaten und erhöht dadurch die Robustheit des Koordinationsmechanismus. Auf der mobilen Plattform kommt für die reaktive und die vermittelnde Ebene der Ansatz von Pauly [Pau98] zum Einsatz.

[25] Der genaue Vorgang bei diesem Ansatz wurde schon in Abschnitt 2.3.2 diskutiert.

[26] Dabei existieren zwei reaktive und zwei vermittelnde Ebenen - je eine für den Roboterarm und für die mobile Plattform.

[27] Wie z.B. *Radial Basis Function Networks* (RBF)[Hay94] für die bildgestützte Zielführung und *Neurofuzzy* [vA93] für die Hindernisvermeidung des Greifers.

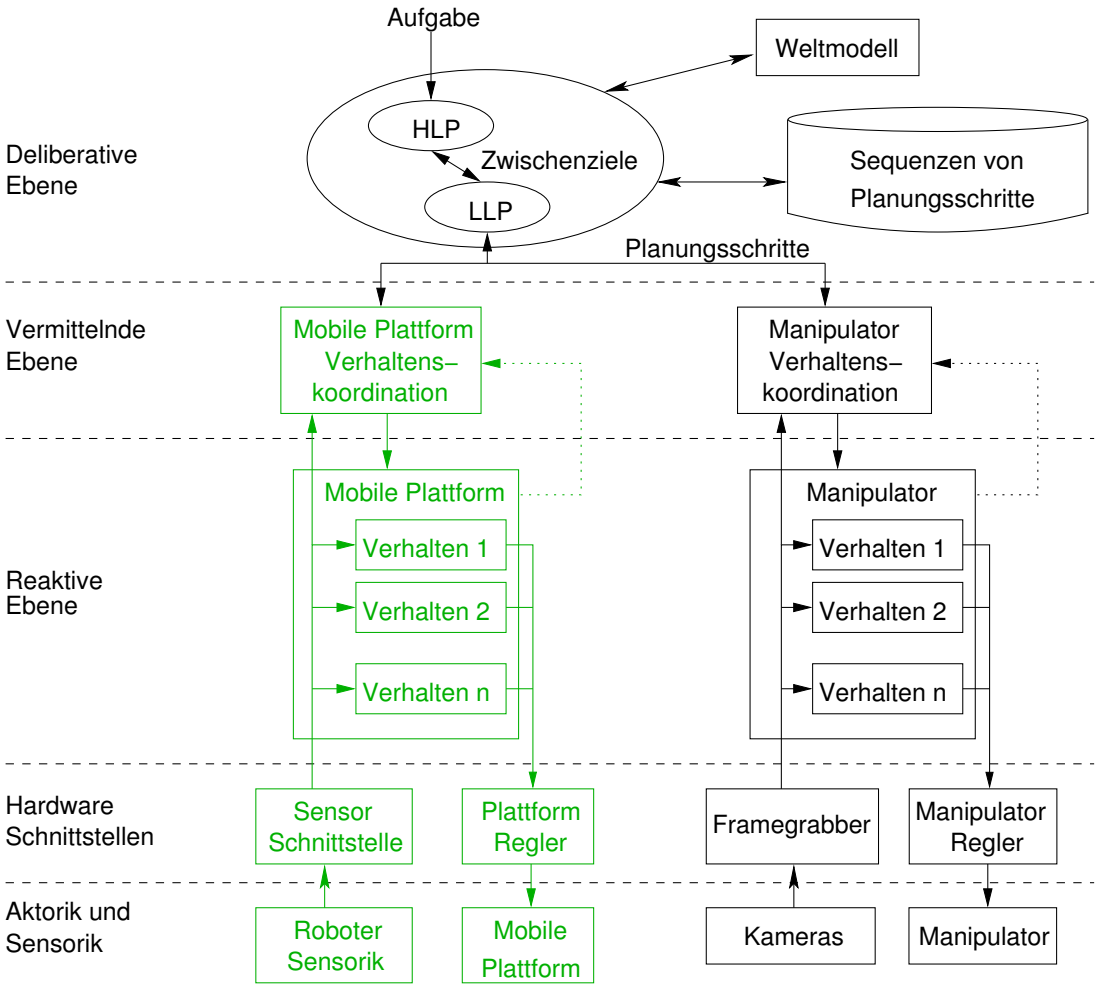


Abbildung 2.24: Die erweiterte drei-Ebenen Architektur des mobilen Manipulators. Grau eingezeichnet sind die Komponenten, die aus der Arbeit von Pauly [Pau98] übernommen wurden.

Mobile Plattform und Manipulator werden über die deliberative Ebene koordiniert. Diese zerlegt eine Aufgabe in eine Reihenfolge von Planungsschritten für den Manipulator und die mobile Plattform. Hier arbeitet eine auf dem *Situationskalkül* [McC68] basierte Planungskomponente, der *High-Level Planner* (HLP), strategische Pläne aus und setzt Zwischenziele, indem sie die Pläne auf ihre Wirkung in die Zukunft projiziert. Ein schnelles reaktives Planungssystem, der *Low-Level Planner* (LLP), zerlegt die Zwischenziele in vordefinierte Sequenzen von Planungsschritten anhand der Ergebnisse des zuletzt ausgeführten Planungsschrittes und der mit den Sensoren erfassten Situation der Umgebung. Die deliberative Ebene umfasst auch einen probabilistischen Routenplaner, der mögliche Pfade für die mobile Plattform zum Lösen einer Aufgabe berechnet. Diese Aufteilung der deliberativen Ebene[28] in HLP und LLP vereinfacht die Planung und ermöglicht einerseits eine zielstrebige, globale Planung, die für komplexe Aufgaben Lösungen finden kann, und andererseits eine schnelle lokale Anpassung der Pläne auf unvorhersehbare Ereignisse.

Ein weiterer Kernpunkt dieser Arbeit ist das Training und die Evaluierung der bildgestützten Verhalten und des Koordinationsmechanismus in einer virtuellen Umgebung mit Modellen des Zielobjektes,

[28] In dieser Arbeit wird die deliberative Ebene als gegeben angenommen und nur kurz im Kapitel 6 auf deren Eigenschaften eingegangen.

der Hindernisse und des eingesetzten Serviceroboters. Als Trainingsdaten dienen Aufnahmen aus virtuellen Kameras und Positionsdaten des Manipulators. Für das Lernen von zielführenden Verhalten werden objektspezifische Pfade anhand einer Vorgabe automatisch generiert und entlang der Pfade die Trainingsdaten gesammelt (*algorithmisches Teach-In*). Kollisionsvermeidende Verhalten sowie der Koordinationsmechanismus setzen zum Training *Reinforcement Learning*-Techniken [SB98] ein (*stochastisches Teach-In*). Die erlernten Algorithmen werden dann auf den realen Serviceroboter transferiert und eingesetzt. Durch den Einsatz der virtuellen Umgebung ist die Automatisierung des Trainingvorgangs von bildgestützten Verhalten möglich, ohne die ständige Überwachung des Systems und ohne Gefährdung der realen Hardware. Besonders das *Reinforcement Learning*, das unter anderem in frühen Trainingsphasen Kollisionen mit Hindernissen nicht abfängt, kann problemlos in der virtuellen Umgebung ablaufen. Zugleich können dieselben Situationen zur Wiederholung des Trainings oder zur Evaluierung der Ergebnisse reproduziert werden.

2.8 Abgrenzung von anderen Arbeiten

Das vorgestellte Konzept unterscheidet sich in einer Reihe von Aspekten von den in der Literatur beschriebenen Verfahren. Die bildgestützte Zielführung basiert, ähnlich wie in [BP98], auf einer Struktur von RBF-Netzen. Sie hat jedoch nicht nur die Aufgabe, die Position des Objektes sondern auch dessen Orientierung zu bestimmen. Weiterhin berechnet sie einen objektspezifischen Pfad zur Greifposition am Objekt, was sie von anderen Arbeiten in dem *Visual Servoing* Bereich unterscheidet. Dadurch erlaubt sie die Implementierung von unterschiedlichen, objektspezifischen Greifstrategien und schließlich ein effizientes Greifen und Manipulieren von Objekten.

Unterschiedlich zu den Verfahren in [IBT+02], in [FSZ00] und in [PRG+03] berücksichtigt die hier implementierte Hindernisvermeidung sowohl den Greifer als auch die Segmente des Manipulators. Dabei werden zwei unterschiedliche Kamerakonfigurationen verwendet: die *eye-in-hand* Kamera liefert die Daten für die Hindernisvermeidung des Greifers und die *eye-to-hand* Kamera der mobilen Plattform akquiriert die Daten für Hindernisvermeidung der Manipulatorsegmente. Systeme, die in der Literatur referenziert sind, benutzen nur eine Kamerakonfiguration und zwar die *eye-to-hand* Kameras der Plattform. Verhalten, die die Hindernisvermeidung über die Greiferkamera implementieren, sind bei anderen Systemen nicht vorhanden. Da bei der Greiferkamera keine Verdeckung der Szene im Bild durch die Manipulatorsegmente vorkommt, ist eine robuste Hindernisvermeidung für den kompletten Roboterarm möglich.

Bis auf die Ansätze von Wösch et al. [WN01] und Hashimoto [HNI01], wenden hybride Architekturen, wie z.B. MissionLab [MA99], keine Verhaltenskoordinierung bei einem Manipulator an; bestenfalls erlauben sie dem Programmierer den Koordinationsmechanismus für den Roboterarm selbst zu realisieren. In dieser Arbeit wird ein einheitliches Konzept zur Koordination der bildgestützten Verhalten des Roboterarms präsentiert, erlernt und mit einer *pick-and-place* Aufgabe evaluiert. Dabei handelt es sich nicht um eine einfache Interpolation der Verhaltensausgaben, wie in [WN01], sondern um eine gewichtete Summierung anhand des gegenwärtigen Sensorkontexts, die besonders in kritischen Bereichen den Roboterarm effektiv vor Kollisionen schützt. Weiterhin ist in den BBNs die Unsicherheit der oft verrauschten Merkmale modelliert und im Gewichtungsprozess berücksichtigt. Der Koordinationsmechanismus ist, unterschiedlich zu dem Verfahren in [HNI01], leicht um neue Verhalten erweiterbar und erfordert kein aufwendiges Experimentieren zum Setzen der Parameter, die in einer virtuellen Umgebung erlernt werden. Außerdem wird hier die Hindernisvermeidung

direkt während des Greifvorgangs angewendet. Der implementierte Verhaltenskoordinationsmechanismus ermöglicht die situationsabhängige Fusion der Verhaltensausgaben und somit eine robuste Hindernisvermeidung und zugleich ein effizientes objektspezifisches Greifen.

Im Gegensatz zu anderen Ansätzen zur Verhaltenskoordination wird hier keine rein reaktive oder rein vorausplanende Strategie verfolgt. Stattdessen wird eine Kombination benutzt, so dass die Vorteile aus beiden Strategien ausgenutzt werden: ein Pfadplanungsverfahren ist als Verhalten eingeführt und gleicht damit einer Komponente des DAMN Ansatzes [Ros95]. Die vermittelnde Ebene kann dann den vorab berechneten Pfad des Planungsverhaltens situationsabhängig in die resultierende Aktion einbeziehen bzw. reaktiv anpassen. Zugleich bewahrt die Ausgabe des Planungsverhaltens den Manipulator vor *Deadlocks* und trägt beim Auflösen von Konflikten zwischen den reaktiven Verhalten bei.

Die virtuelle Umgebung zum Training der bildgestützten Verhalten muss nicht mit der Einsatzumgebung genau übereinstimmen, wie bei [HLBS96]. Sie ermöglicht das Erlernen der bildgestützten objektspezifischen Zielführung über das algorithmische Teach-In und der Hindernisvermeidung des Greifers über das stochastische Teach-In. Das Training des Koordinationsmechanismus der Fertigkeiten in der virtuellen Umgebung ist auch nicht in anderen Systemen, wie [HLBS96] oder [FSZ00] vorgesehen. Das algorithmische und das stochastische Teach-In erlauben die Automatisierung des Trainings von bildgestützten Steuerungsalgorithmen, ohne dabei den Roboter zu gefährden. Weiterhin ermöglicht der Trainingsvorgang in der virtuellen Umgebung mit mehreren Objektkonstellationen die situationsabhängige Optimierung der Parameter der eingesetzten Algorithmen und macht ein langwieriges Experimentieren unnötig.

Kapitel 3

Eine virtuelle Umgebung zum Teach-In von Robotern

Bei der Implementierung von bildgestützten Steuerungsalgorithmen für Roboter, insbesondere von Verfahren zur Hindernisvermeidung, birgt die Test- und Evaluierungsphase erhebliche Gefahren für die Hardware mit sich. Fehlerhafte Parametrisierung der Algorithmen kann leicht zu Kollisionen und Beschädigung des Roboters führen. Diese Gefahr ist bei lernfähigen Verfahren umso größer, da in den frühen Trainingsphasen die Algorithmen nicht über die notwendige Information verfügen, um Kollisionen mit Objekten im Arbeitsbereich zu verhindern. Zusätzlich ist der Trainingsvorgang in der realen Welt aufwendig, denn er erfordert einen präzisen Aufbau zur Akquisition der Trainingsdaten und ständige Überwachung.

Diese Nachteile können durch die Verwendung einer virtuellen Simulationsumgebung aufgehoben werden. Nach Kraiss [Kra03a] ist eine virtuelle Umgebung eine im Computer erzeugte, nur scheinbar vorhandene Welt, die ein Betrachter mit seinen natürlichen Sinnen (sehen, hören, fühlen) als real erleben und mit der er interagieren kann. Dabei handelt es sich um eine in Echtzeit handhab- und veränderbare Raumumgebung mit physikalisch erfahrbaren Eigenschaften. Importiert man Modelle des Roboters und seiner Sensorik in eine Simulationsumgebung, dann kann man die Daten der Sensoren zur Detektion der Änderungen der virtuellen Umgebung verwenden. Erhalten die Steuerungsalgorithmen diese Daten, so können sie den virtuellen Roboter entsprechend sensorbasiert führen. Da die Steuerungskomponente sowohl in der realen als auch in der virtuellen Umgebung dieselbe ist, müssen realer und virtueller Roboter ein ähnliches Verhalten aufweisen. Diese Eigenschaft kann man ausnutzen, um die Steuerungsalgorithmen des mobilen Manipulators zuerst in der virtuellen Umgebung zu entwickeln, zu evaluieren und eventuell anzupassen, bevor sie auf dem realen Roboter Anwendung finden.

Derselbe Gedanke liegt auch hinter der Verwendung von 2D-Simulationsumgebungen, die oft für das Training und die Evaluierung von Navigationsalgorithmen für mobile Plattformen zum Einsatz kommen. Eine virtuelle Umgebung verfügt jedoch über zwei entscheidende Vorteile gegenüber 2D-Simulationsumgebungen. Einerseits bietet sie die Möglichkeit, virtuelle Aufnahmen der Szenen zu akquirieren und sie zum Training von bildgestützten Verfahren einzusetzen. Andererseits erlaubt sie die Ausführung von dreidimensionalen Bewegungen und demnach Manipulationsbewegungen mit einem Roboterarm. Somit ist durch die Verwendung der virtuellen Umgebung die Automatisierung des Trainingvorgangs von bildgestützten Verhalten möglich, ohne dabei das System ständig überwachen zu müssen und ohne die Gefahr, die Hardware während des Trainings zu beschädigen. Zugleich

können die selben Situationen zur Wiederholung des Trainings oder zur Evaluierung der Ergebnisse reproduziert werden.

In dieser Arbeit kommt eine virtuelle Umgebung zum Einsatz, um bildgestützte, lernfähige Steuerungsalgorithmen zu trainieren [BHK02]. Die implementierte und für das Training der Algorithmen eingesetzte Simulationsumgebung basiert auf dem WorldToolKit (WTK) der Firma Sense8 [Sen01]. Sowohl die bildgestützten Verhalten als auch ihr Koordinationsmechanismus werden in dieser Umgebung mit Modellen der Objekte und des eingesetzten mobilen Manipulators trainiert und evaluiert.

Für das Lernen von zielführenden Verhalten werden in der virtuellen Umgebung objektspezifische Pfade anhand einer Vorgabe automatisch generiert und entlang der Pfade die Trainingsdaten gesammelt (*algorithmisches Teach-In*). Beim *stochastischen Teach-In* erforscht dagegen der Roboterarm seinen Einsatzraum ohne eine Vorgabe. Dieses Verfahren basiert auf *Reinforcement Learning*-Techniken [SB98] und kommt beim Training der kollisionsvermeidenden Verhalten sowie des Koordinationsmechanismus zum Einsatz. Beide Verfahren werden in diesem Kapitel beschrieben, nachdem zuerst die Umsetzung vorgabegetreuer Roboterkinematiken und der Abgleich der Kameradaten präsentiert wurde.

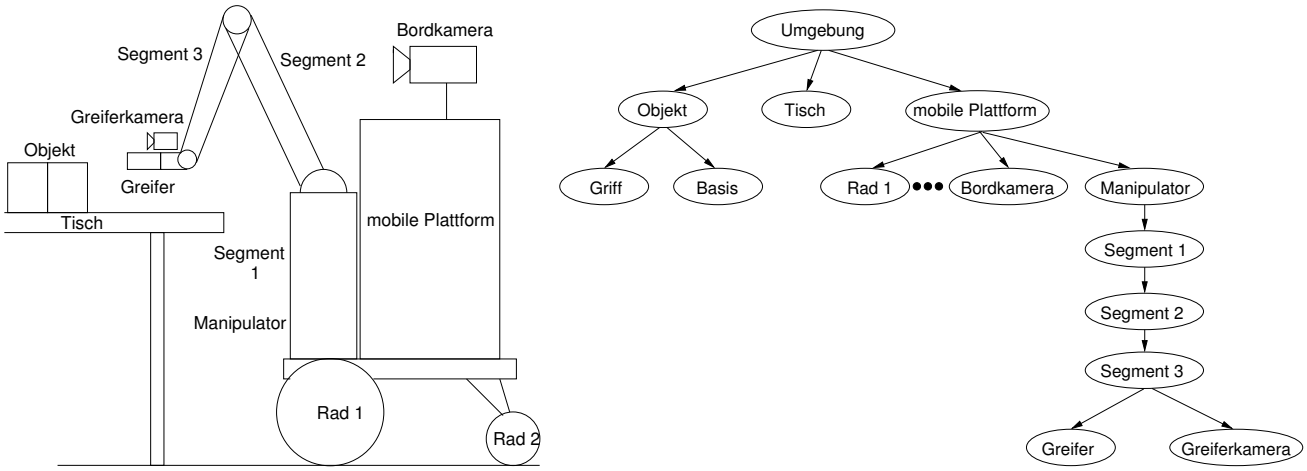
3.1 Umsetzung vorgabegetreuer Roboterkinematiken

Das Modell des eingesetzten mobilen Manipulators wird aus einfachen geometrischen Körpern bzw. Primitiven, wie Kugeln, Quader oder Kegel zusammengestellt. Die Lage dieser Primitiven ist über ein körperspezifisches Koordinatensystem relativ zu einem globalen Bezugskoordinatensystem definiert. Durch Verschiebung und Rotation dieser Systeme kann man die Körper in der virtuellen Umgebung bewegen. Um ein komplexes Objekt zu erzeugen, können mehrere Primitive gruppiert werden. Dabei lassen sich auch Hierarchien definieren, so dass die Lage eines oder einer Gruppe von Körpern O_i bezüglich eines Koordinatensystems eines hierarchisch höheren Körpers O_0 bestimmt werden kann. Bewegt sich O_0, so bewegen sich auch die mit ihm verknüpften Körper O_i, da die Lage ihrer Koordinatensysteme relativ zu O_0 konstant bleibt, sich relativ der Umgebung jedoch wegen der Bewegung von O_0 verändert. Diese Hierarchien in der virtuellen Umgebung lassen sich in einem gerichteten azyklischen Graphen abbilden, dem so genannten Szenengraphen (*Scene Graph*), der eine Szene beschreibt (Abbildung 3.1). Im Szenengraphen kann man Vater-Kinder Beziehungen abbilden, die vertikal dargestellt sind, während Bruder-Beziehungen ebenenweise horizontal aufgetragen sind.

Mit dem Szenengraphen lassen sich einfach kinematische Ketten (siehe Anhang C.1) und dementsprechend Roboterarme in der virtuellen Umgebung definieren und bewegen. Benutzt man die DH-Parameter des Manipulators, um die Koordinatensysteme der Segmente und Gelenke des Manipulators im Szenengraphen festzulegen, so entsteht ein virtueller Manipulator mit denselben kinematischen Eigenschaften wie der reale [Nev00]. Dadurch können der reale und der virtuelle mobile Manipulator bei demselben Bewegungskommando identische Bewegungen ausführen.

In dieser Arbeit wurde das Modell der eingesetzten nicht-holonomen[1], mobilen Plattform TAURO[3] und des Manipulators Manus im Szenengraphen von WTK definiert [Nev00]. Unterliegende C++-Klassen implementieren die Schnittstellen der Kinematik zu den Steuerungsalgorithmen. Der virtuelle Manipulator kann, wie auch der reale Roboter, über Gelenkgeschwindigkeiten angesteuert werden und liefert die aktuellen Gelenkpositionen zurück. Allerdings ist wegen einer langsamen Verbindung

[1] Eine nicht-holonomer mobiler Roboter ist ein Roboter, der sich nicht in jeder Richtung direkt bewegen kann.



(a) (b)

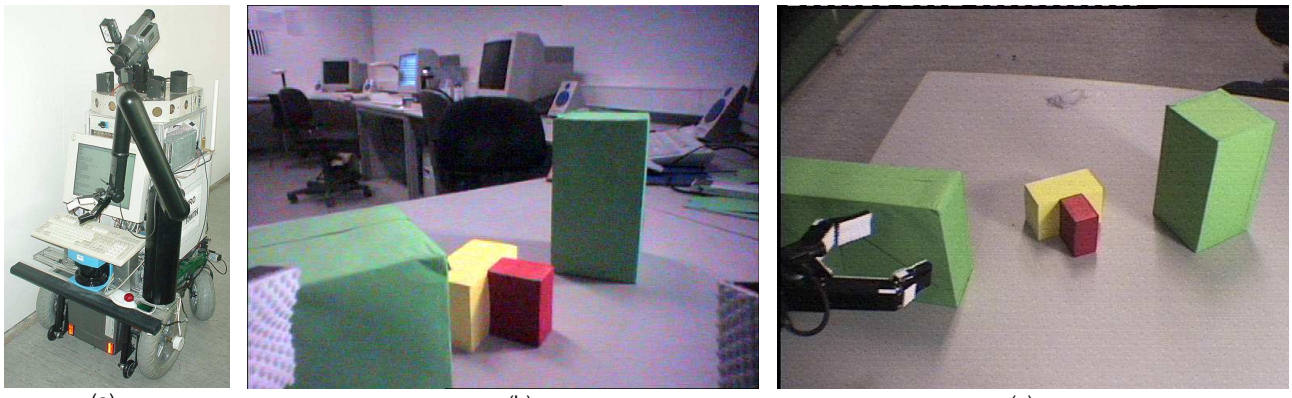
Abbildung 3.1: Beispiel einer Szene und entsprechender Szenengraph.

zwischen Simulations- und Ansteuerungsrechner die Manipulatorregelung während der gleichzeitigen Übertragung von virtuellen Aufnahmen nicht möglich. Deswegen wurde eine zusätzliche positionsbasierte Steuerung für den Roboterarm implementiert. In diesem Fall wird die auszuführende Trajektorie extrapoliert und die Stellung des Roboterarmes zu bestimmten Zeitpunkten berechnet. Der Manipulator kann diese Gelenkstellungen direkt annehmen, ohne die Zwischenpositionen der Trajektorie anfahren zu müssen. Da während der Aufnahme eines Bildes der Manipulator keine Positiondaten erhält, muss er solange an seiner Aufnahmeposition warten, bis das Bild übertragen ist; erst dann führt er seine Bewegung weiter bzw. nimmt die nächste berechnete Stellung der Trajektorie an. Indem man jedoch die Abstände der Zeitpunkte verändert, kann man die Frequenz, mit der ein Verhalten den Manipulator regelt, unabhängig von der zeitlichen Ausführung der Manipulatorbewegung beeinflussen.

3.2 Abgleich der Daten virtueller und realer Kameras

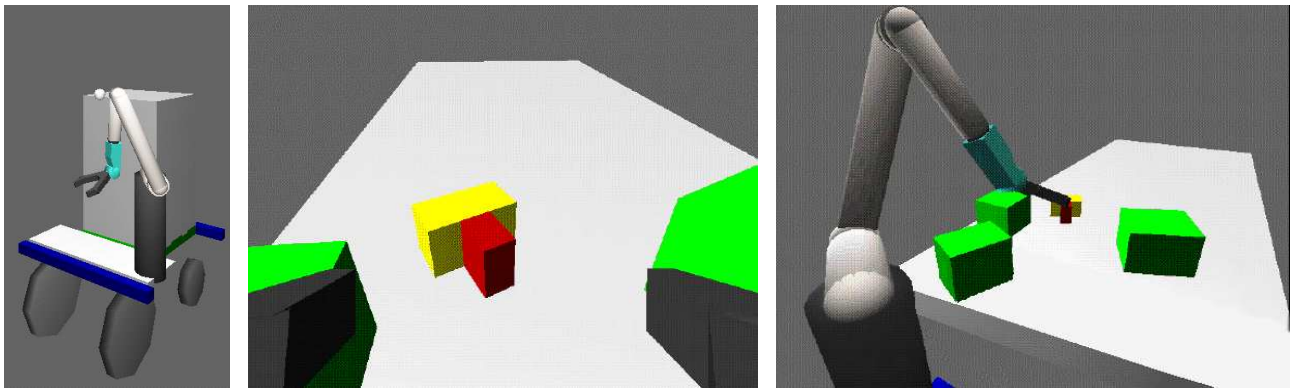
Eine virtuelle Kamera kann als Fenster in die virtuelle Welt verstanden werden, dessen Position und Ausrichtung frei wählbar sind [San01]. Aufnahmen der virtuellen Kameras entsprechen Bildschirmausdrucken der WTK-Fenster, die mit diesen Kameras verknüpft sind. Ihr Öffnungswinkel entspricht in WTK bei einem Fenster der Größe 640x480, einem Winkel von ungefähr $64,0^0$ in Bildbreite und von $48,0^0$ in Bildhöhe.

Der mobile Manipulator ist mit zwei Kameras ausgestattet, einer Bordkamera auf der mobilen Plattform (*eye-to-hand* Konfiguration) und einer Greiferkamera in einer *eye-in-hand* Konfiguration (Abbildung 3.2). Für die virtuelle Umgebung wird ein Modell mit entsprechender Kameraausstattung eingesetzt, wie auch in Abbildung 3.3 zu sehen ist. Die Positionen der virtuellen Kameras sind so gewählt, dass sie den Positionen der realen Kameras auf dem Roboter entsprechen. Indem sie über die Szenengraphen mit Segmenten des Roboters verknüpft werden, bewegen sie sich nur, wenn die entsprechenden Segmente des Roboters sich bewegen.



(a) (b) (c)

Abbildung 3.2: Realer Roboter (a) und Ansicht von der Greifer- (b) und der Bordkamera (c).



(a) (b) (c)

Abbildung 3.3: Modell des virtuellen Roboters (a) und Ansicht von der Greifer- (b) und der Bordkamera (c).

Trotz der Ähnlichkeit zwischen realen und virtuellen Szenen verfügen Aufnahmen aus unterschiedlichen Kameras über entscheidende Unterschiede bezüglich ihres Öffnungswinkels, ihrer Linseneigenschaften und der Aufnahmequalität. Diese Unterschiede erlauben keinen direkten Vergleich der virtuellen und realen Aufnahmen; die Daten müssen zuerst abgeglichen werden. Der Verlauf des Abgleiches ist in Abbildung 3.4 zu sehen.

Der Öffnungswinkel und die Linseneigenschaften beeinflussen die Projektion der Szene auf der Bildoberfläche und somit die akquirierten Aufnahmen. Weisen virtuelle und reale Kameras unterschiedliche Projektionseigenschaften auf, dann müssen die Aufnahmen der realen Kameras so transformiert werden, dass sie den Eigenschaften der virtuellen Kameras entsprechen. Dafür sind entsprechende Projektionsparameter notwendig, die bei der so genannten Kalibrierung der Kameras identifiziert werden. Zugleich liefert die Kalibrierung die genaue Lage der realen Kameras relativ zum Roboter, die somit im Szenengraphen des virtuellen mobilen Manipulators genau justiert werden kann.

Weitere Faktoren, die den Vergleich der realen mit den virtuellen Aufnahmen beeinflussen, sind Beleuchtungsunterschiede und die Existenz von Rauschen in den Bildern der realen Kameras. Die Beleuchtungsunterschiede beeinflussen die Erscheinung der Farben in den Aufnahmen und erschweren somit die Segmentierung der Objekte im Bild anhand der Farbinformation. Diese Unterschiede können nicht eliminiert werden, jedoch kann man ihren Einfluss über den Einsatz eines Filters zur

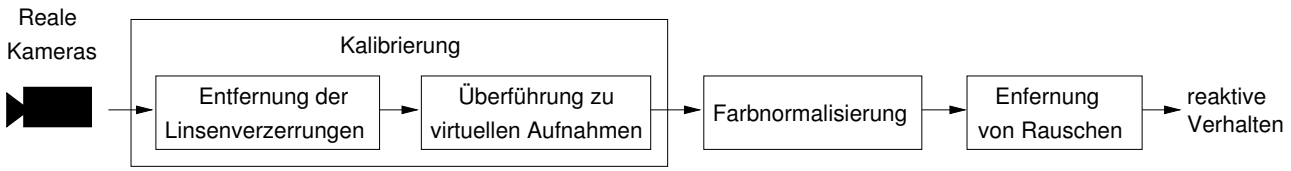


Abbildung 3.4: Ablauf des Abgleiches der realen mit den virtuellen Aufnahmen.

Farbnormalisierung vermindern. Rauschen in den realen Aufnahmen ist auf die Kameraqualität, die Beleuchtungsvariation und die Störungen bei der Übertragung des analogen Videosignals von Kamera zum Rechner und der nachfolgenden Digitalisierung zurückzuführen. Da Rauschen eine fehlerbehaftete Segmentierung im Bild verursachen kann, wird hier ein Filter zum Entfernen von Rauschen aus den realen Aufnahmen angewendet.

3.2.1 Kamerakalibrierung

Wegen unterschiedlicher Kameraeigenschaften können zwei Aufnahmen aus zwei unterschiedlichen Kameras nicht deckungsgleich sein. Deswegen müssen die Eigenschaften bestimmt und die Aufnahmen der realen Kameras entsprechend den Parametern der virtuellen Kameras transformiert werden, damit ein Vergleich realer und virtueller Bilder möglich ist.

Zur Bestimmung der Kameraparameter ist jedoch ein Modell notwendig, das die Projektionseigenschaften der Abbildung von Raumpunkten auf die Bildebene beschreibt. Das Lochkameramodell stellt ein solches Modell dar. Es besteht aus einer Bildebene und einem Loch mit unendlich kleinen Dimensionen (Brennpunkt, *Focal Point* oder *Principal Point*), durch das Lichtstrahlen auf die Bildebene fallen. Dadurch lassen sich Raumpunkte auf die Bildebene projizieren. Der Abstand zwischen Brennpunkt und Bildebene ist als Brennweite oder Fokallänge f (*Focal Lenght*), die orthogonale Projektion des Brennpunktes auf der Bildebene als Bildhauptpunkt bekannt. Die Gerade, die Bildhauptpunkt und Brennpunkt verbindet, wird als Sichtachse (*Optical Axis*) der Kamera bezeichnet. Die Position der Bildpunkte relativ zum Bildhauptpunkt werden im Koordinatensystem der Bildebene $\{I\}$ beschrieben. Das Kamerakoordinatensystem $\{K\}$ gibt dagegen die Position des Brennpunktes und die Orientierung der Kamera relativ zu einem Weltkoordinatensystem $\{W\}$ an, bezüglich welchem die Koordinaten der Raumpunkte beschrieben sind (Abbildung 3.5).

Sei M ein Raumpunkt mit Koordinaten $({}^{K}x,\ {}^{K}y,\ {}^{K}z)^T$ bezüglich des Kamerakoordinatensystems, dann ist das Verhältnis von M zu seiner Projektion auf der Bildebene $({}^{I}x,\ {}^{I}y)$ durch Gleichung 3.1 angegeben.

$$ {}^{I}x = \frac{f\ {}^{K}x}{{}^{K}z}, \qquad {}^{I}y = \frac{f\ {}^{K}y}{{}^{K}z} \tag{3.1} $$

Gleichung 3.1 gibt jedoch nicht die Pixelkoordinaten im Bild wider. Dafür müssen ${}^{I}x$ und ${}^{I}y$ zusätzlich mit zwei Parametern a_u und a_v skaliert werden, um die Einheiten in Pixel umzuwandeln. Der Parameter a_u repräsentiert die Skalierung in der u-Achse des Kamerabildes und in ähnlicher Weise bestimmt a_v die Skalierung in der v-Achse. Diese Beziehungen werden in der Kalibrierungsmatrix $\mathbf{K}$ zusammengefasst:

$$ \mathbf{K} = \begin{bmatrix} -fa_u & -fa_{shear} & -u_0 \\ 0 & -fa_v & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

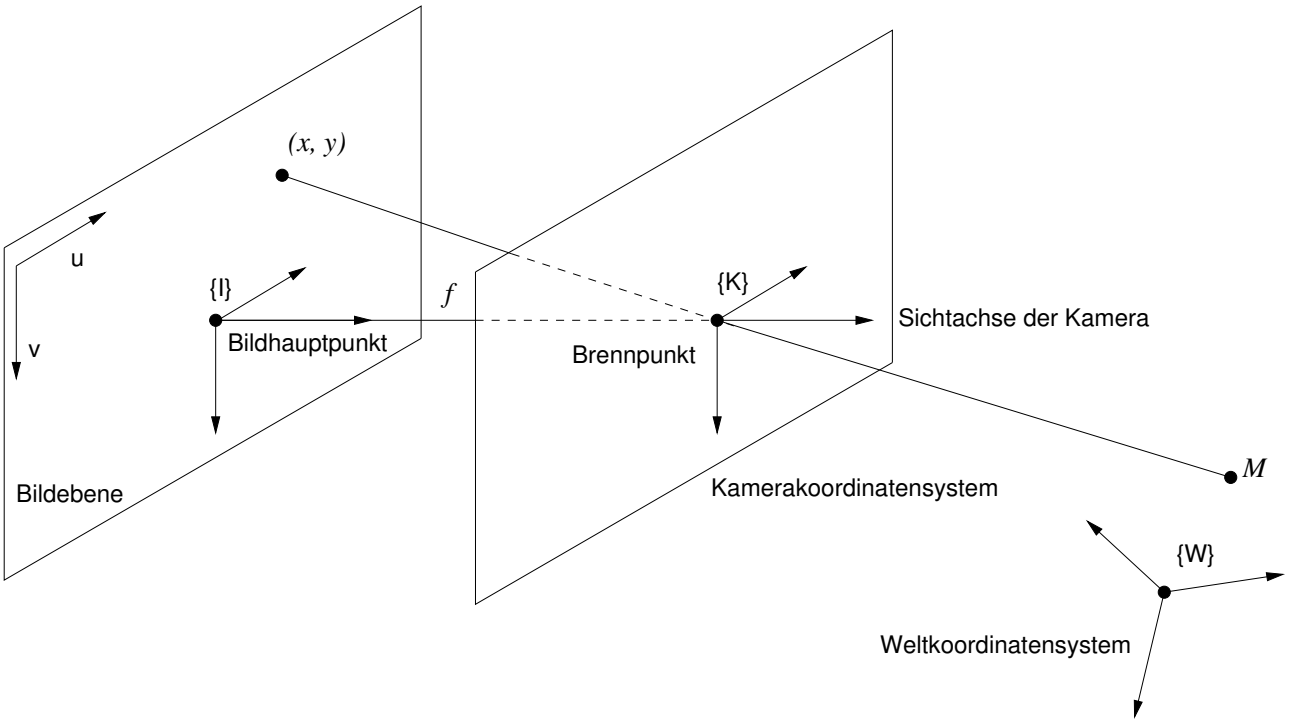


Abbildung 3.5: Lochkameramodell und entsprechende Koordinatensysteme.

Dabei stellen u_0 und v_0 die Koordinaten des Bildhauptpunktes in Pixel dar und a_{shear} misst die Abweichung in Pixel der x-Achse des Kamerakoordinatensystems von der u-Achse des Kamerabildes[2].

Die Parameter, die in Gleichung 3.2 vorkommen, also $(fa_u, fa_v, fa_{shear}, u_0, v_0)$, sind auch als intrinsische Parameter bekannt; sie beschreiben die Eigenschaften der Kamera, wie z.B. die Abbildungseigenschaften des Objektives und die Anordnung der Bildebene innerhalb der Kamera. Die so genannten extrinsischen Parameter beschreiben dagegen die Lage des Kamerakoordinatensystems zum Weltkoordinatensystem. Dafür sind drei Parameter für die Translation und drei für die Rotation notwendig, die in der homogenen Transformationsmatrix[3] ${}^K_W\mathbf{T}$ der Welt bezüglich der Kamera zusammengefasst sind. Seien ${}^W\vec{P}$ die homogenen Koordinaten eine Raumpunktes im Weltkoordinatensystem und $(U, V, W)^T$ die homogenen Pixelkoordinaten im Bild, die gegenüber den Pixelkoordinaten $(u, v)^T$ um W skaliert sind. Dann gilt:

$$(U, V, W)^T = \mathbf{K}\,\mathbf{M}\,{}^K_W\mathbf{T}\,{}^W\vec{P} \tag{3.3}$$

und demnach:

$$(u, v, 1)^T = \frac{1}{W}\,\mathbf{K}\,\mathbf{M}\,{}^K_W\mathbf{T}\,{}^W\vec{P} \tag{3.4}$$

wobei $\mathbf{M}$ eine konstante 3×4 Matrix ist, die die Multiplikation der 3×3 Matrix $\mathbf{K}$ mit der 4×4 Matrix ${}^K_W\mathbf{T}$ ermöglicht:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

[2]Die Ausrichtung der v-Achse des Bildes stimmt immer mit der y-Achse des Kamerasystems überein, die Orientierung von der u-Achse kann jedoch von der x-Achse der Kamerakoordinatensystems abweichen [SHB99].

[3]Diese Arbeit verwendet die Notation nach Craig [Cra89] (siehe auch Symbolverzeichnis). Eine Einführung in Koordinatentransformationen ist in Anhang C.1 zu finden.

Unter dem Begriff der Kamerakalibrierung ist der Prozess der Schätzung der intrinsischen und extrinsischen Parameter einer Kamera, also von $\mathbf{K}$ und ${}^{K}_{W}\mathbf{T}$, zu verstehen. Für die Bestimmung der fünf intrinsischen und der sechs extrinsischen Parameter sind mindestens elf linear unabhängige Gleichungen notwendig; aus Gleichung 3.4 erhält man für einen bekannten Raumpunkt und dessen Projektion im Bild zwei Gleichungen. Daher sind sechs Raumpunkte zur Bestimmung von $\mathbf{K}$ und ${}^{K}_{W}\mathbf{T}$ erforderlich. Jedoch ist es ratsam mehr Punkte, die auch im Bild verteilt sind, einzusetzen, um eine genaue Kalibrierung zu erhalten.

Die Kalibrierungsansätze lassen sich in lineare und nichtlineare Verfahren unterteilen. Lineare Verfahren können $\mathbf{K}$ und ${}^{K}_{W}\mathbf{T}$ berechnen, indem sie die Methode des kleinsten Fehlerquadrates (*Least Squares*) auf die Gleichungen anwenden [Aya91]. Von den benötigten sechs Raumpunkten dürfen je vier nicht auf derselben Ebene liegen, damit die resultierenden zwölf Gleichungen linear unabhängig sind. Die linearen Methoden berücksichtigen keine Bildverzerrungen durch das Linsensystem und sind deswegen ungenau. Mit diesem Ansatz sind die virtuellen Kameras, die keine Linsenverzerrungen aufweisen, kalibriert[4] worden. Dabei wurde ein drei-dimensionales Punktmuster mit 20 Raumpunkten verwendet.

Alternativ berücksichtigen Tsai [Tsa87] und Heikkilä [Hei00] ein Linsenverzerrungsmodell und wenden nichtlineare, iterative Methoden an. Dadurch können sie den Kalibrierungsfehler weitgehend minimieren und genauere Resultate als die linearen Methoden liefern. Sie berechnen folgende intrinsische Parameter:

- die radiale Linsenverzerrung erster und zweiter Ordnung (κ_1 und κ_2),
- das Zentrum der radialen Linsenverzerrung auf der Bildebene (u_{0u}, v_{0u}),
- die Brennweite f und
- den Skalierungsparameter s_k, der den Quotienten der Pixellänge zur Pixelhöhe angibt und gleich $\frac{a_u}{a_v}$ ist.

Um aus diesen Parametern die Kalibrierungsmatrix zu bestimmen, muss a_u aus der Länge des CCD-Chips der Kamera und der Pixellänge der Aufnahmen berechnet werden. Die Verfahren erlauben die Verwendung von zwei Berechnungsvarianten: eine mit koplanaren Kalibrierungspunkten und eine mit einem nicht-planaren Kalibrierungsmuster. Die zweite Variante wurde hier für die realen Kameras verwendet; die berechneten Kameraparameter sind in Anhang B präsentiert.

Entfernen von Linsenverzerrungen

Da die virtuellen Kameras keine Linsenverzerrungen enthalten, die realen Aufnahmen jedoch von den Linseneigenschaften beeinflusst werden, müssen bei Bildern der realen Kameras die Verzerrungen entfernt werden. Insbesondere bei der Bordkamera von TAURO[3] ist dies notwendig, da hier der Einsatz eines Weitwinkelobjektivs große Abweichungen bei der Lokalisierung der Objekte bewirkt[5]. Seien (u_v, v_v) die verzerrt abgebildeten Bildkoordinaten, (u_u, v_u) die unverzerrten Koordinaten und δu und δv die Korrekturterme. Dann ist:

$$\begin{aligned} u_u &= u_v + \delta u \\ v_u &= v_v + \delta v \end{aligned} \tag{3.5}$$

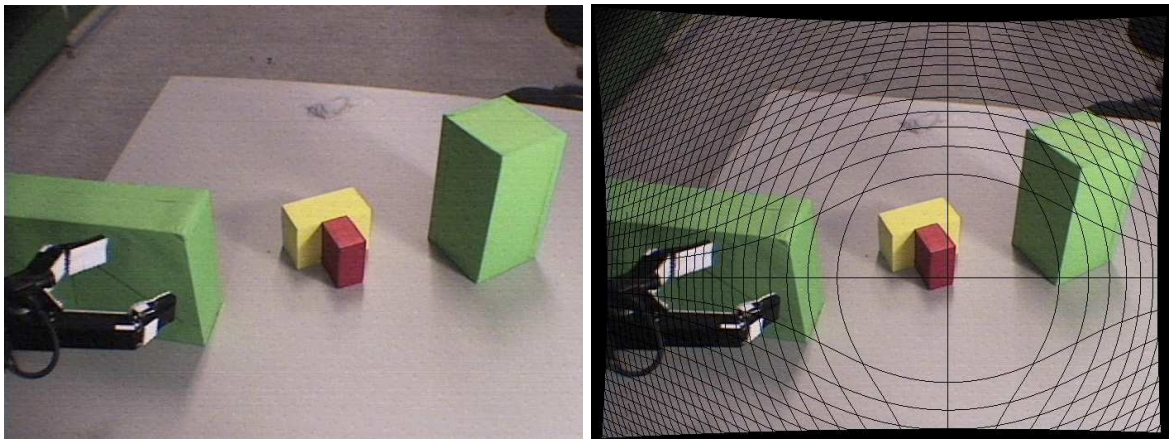
[4]Die Resultate der intrinsischen Kalibrierung sind in Anhang B zu finden.

[5]Bei der Sony DCR-VX700E mit Weitwinkelobjektiv, die hier als Bordkamera eingesetzt ist, beträgt der Fehler zwischen verzerrten und unverzerrten Pixelkoordinaten in der Nähe der Ränder etwa 32 Pixel [Sch01].

Die Linsenverzerrungen lassen sich mathematisch als Polynome mit geradzahligen Exponenten modellieren und korrigieren [SHB99]. Nach der Fachliteratur erhält man bereits mit einem Polynom vierten Grades eine gute Näherung:

$$\begin{aligned} \delta u &= (u_v - u_{0u})(\kappa_1 r^2 + \kappa_2 r^4) \\ \delta v &= (v_v - v_{0u})(\kappa_1 r^2 + \kappa_2 r^4) \\ r^2 &= (u_v - u_{0u})^2 + (v_v - v_{0u})^2 \end{aligned} \tag{3.6}$$

Im Rahmen dieser Arbeit hat sich herausgestellt, dass auch ein Polynom zweiten Grades ($\kappa_2 = 0$) zum Entfernen der Linsenverzerrungen der realen Kamerabildern genügt. Das Resultat ist in Abbildung 3.6 vorgestellt.



(a) (b)

Abbildung 3.6: Aufnahme vor und nach der Entfernung der Linsenverzerrungen. Die Berechnung der entzerrten Pixelpositionen aus den Gleichungen 3.5 und 3.6 sind reelwertig, wobei gültige Bildpixel ganze Zahlen als Positionswerte annehmen. Aus diesem Grund entstehen schwarze Linien im entzerrten Bild (b).

Intrinsische Parameter

Da die Kameras unterschiedliche Projektionseigenschaften aufweisen, unterscheidet sich auch der Öffnungswinkel jeder Kamera und demnach der Winkel, der einem Pixel zugeordnet ist. Dementsprechend wird derselbe Raumpunkt bei zwei Kameras mit denselben extrinsischen aber unterschiedlichen intrinsischen Parametern auf unterschiedliche Pixel projiziert. Deswegen ist hier eine Transformationsvorschrift gesucht, die die Pixelkoordinaten der realen Kamera in den entsprechenden Koordinaten der virtuellen Kamera abbildet.

Im Fall der realen Kameras erhält man aus Gleichung 3.3 für einen Raumpunkt ${}^{W}\vec{P}$, der auf den homogenen Pixelkoordinaten $(U_{rl}, V_{rl}, W_{rl})^T$ abgebildet ist:

$${}^{W}\vec{P} = {}^{K}_{W}\mathbf{T}^{-1}_{rl}\ \mathbf{M}^T\ \mathbf{K}^{-1}_{rl}\ (U_{rl}, V_{rl}, W_{rl})^T \tag{3.7}$$

wobei der Index rl für die reale Kamera steht.

Betrachtet man diesen Raumpunkt aus derselben Position mit der virtuellen Kamera, dann ist dieser Raumpunkt auf den homogenen Pixelkoordinaten $(U_{vr}, V_{vr}, W_{vr})^T$ abgebildet:

$$(U_{vr}, V_{vr}, W_{vr})^T = \mathbf{K}_{vr}\, \mathbf{M}\, {}^K_W\mathbf{T}_{vr}\, {}^W\vec{P} \tag{3.8}$$

wobei mit vr die virtuelle Kamera gekennzeichnet ist.

Aus Gleichungen 3.8 und 3.7 folgt die Gleichung zur Transformation des Bildes der realen Kamera in eine entsprechende Aufnahme der virtuellen Kamera:

$$(U_{vr}, V_{vr}, W_{vr})^T = \mathbf{K}_{vr}\, \mathbf{M}\, {}^K_W\mathbf{T}_{vr}\, {}^K_W\mathbf{T}^{-1}_{rl}\, \mathbf{M}^T\, \mathbf{K}^{-1}_{rl}\, (U_{rl}, V_{rl}, W_{rl})^T \tag{3.9}$$

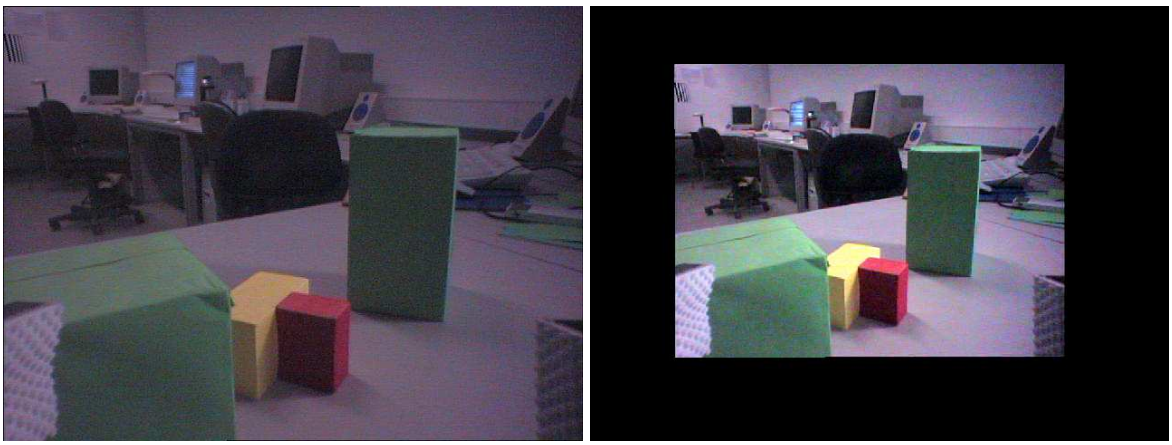
Stimmt die Position der beiden Kameras überein, dann ist ${}^K_W\mathbf{T}_{vr} = {}^K_W\mathbf{T}_{rl}$:

$$(U_{vr}, V_{vr}, W_{vr})^T = \mathbf{K}_{vr}\, \mathbf{K}^{-1}_{rl}\, (U_{rl}, V_{rl}, W_{rl})^T \tag{3.10}$$

Die entsprechenden Pixelkoordinaten der transformierten Aufnahme sind dann:

$$(u_{vr}, v_{vr}, 1)^T = (\frac{U_{vr}}{W_{vr}}, \frac{V_{vr}}{W_{vr}}, 1)^T \tag{3.11}$$

Das Resultat der Transformation ist in Abbildung 3.7 zu sehen.



(a) (b)

Abbildung 3.7: Vor (a) und nach (b) der Transformation des Bildes der realen Kamera in eine entsprechende Aufnahme der virtuellen Kamera.

Extrinsische Parameter

Damit die Aufnahmen der virtuellen und realen Kameras vergleichbar sind, müssen die Kameras dieselbe Position bezüglich des Roboterarms annehmen. Deswegen muss die Position der realen Kameras genau gemessen werden, um die virtuellen Kameras im Modell entsprechend zu positionieren. Aus der Kalibrierung erhält man die relative Lage des Kamerasystems bezüglich des Koordinatensystems, in dem die Kalibrierungspunkte angegeben sind. Im Fall der virtuellen Umgebung handelt

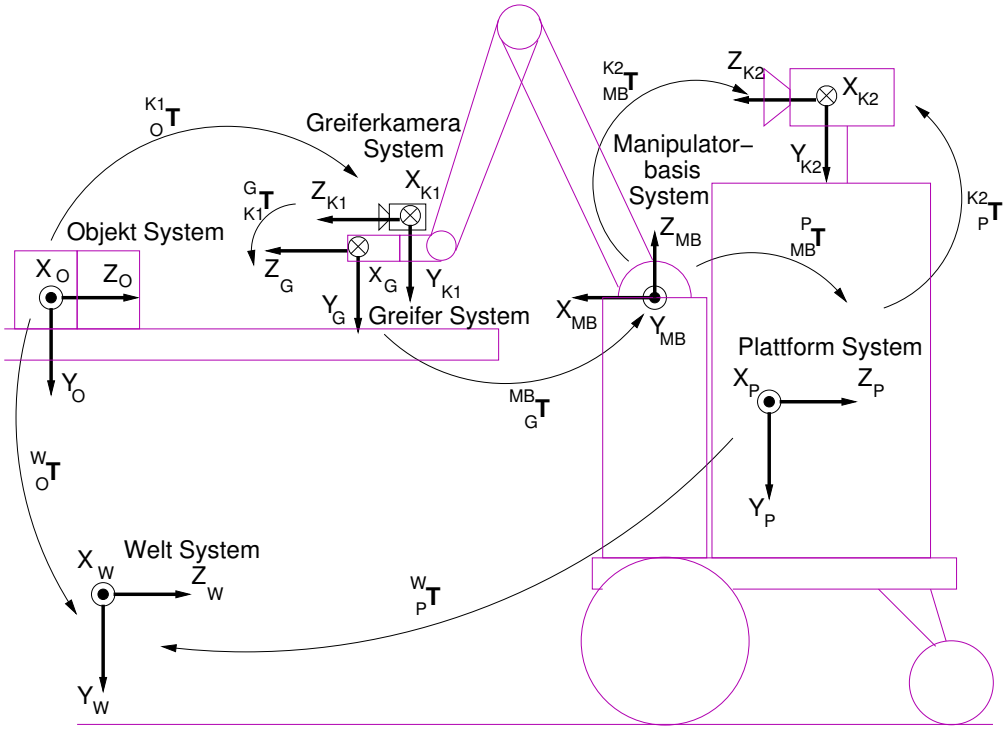


Abbildung 3.8: Koordinatensysteme beim mobilen Manipulator in der virtuellen Umgebung.

es sich um das Bezugssystem der virtuellen Umgebung selbst, also ${}^{W}_{K}\mathbf{T}_{vr}$. Beim realen mobilen Manipulator werden jedoch die Kameras bezüglich des Greiferkoordinatensystems kalibriert, indem das Kalibrierungsmuster auf dem Greifer positioniert wird und somit ist ${}^{G}_{Ki}\mathbf{T}_{rl}$, $i = 1, 2$, bekannt. Da die Position der virtuellen Greiferkamera im Szenengraphen relativ zum Greifer definiert wird, kann man sie anhand von ${}^{G}_{K1}\mathbf{T}_{rl}$ genau im Modell positionieren.

Die Bordkamera wird dagegen im Szenengraphen relativ zur mobilen Plattform positioniert. Dementsprechend ist hier ${}^{P}_{K2}\mathbf{T}_{vr}$ gesucht, also das Koordinatensystem der Kamera relativ zum System der Plattform in der virtuellen Umgebung. In Abbildung 3.8 sind die verschiedenen Koordinatensysteme beim mobilen Manipulator dargestellt. Es gilt:

$$ {}^{P}_{K2}\mathbf{T}_{vr} = {}^{P}_{MB}\mathbf{T}_{rl}\, {}^{MB}_{G}\mathbf{T}_{rl}\, {}^{G}_{K2}\mathbf{T}_{rl} \tag{3.12} $$

wobei ${}^{P}_{MB}\mathbf{T}_{rl}$ das System der Manipulatorbasis zur Plattform und ${}^{MB}_{G}\mathbf{T}_{rl}$ der Greifer zur Manipulatorbasis sind. ${}^{MB}_{G}\mathbf{T}_{vr}$ ist aus der Kinematik des Roboterarmes und ${}^{P}_{MB}\mathbf{T}_{vr}$ aus der Konstruktion des mobilen Manipulators bekannt.

3.2.2 Farbnormalisierung

Das Licht, das die Linse einer Kamera erreicht, ist eine Funktion der Oberflächenreflektion, der Lichtquellenfarbe und der Anordnung der Lichtquellen [FSC98]. Ein Mensch besitzt die Fähigkeit nur die Oberflächenreflektion wahrzunehmen; der Einfluss der Farbe und der Anordnung der Lichtquellen wird weitgehend ausgefiltert. Somit erscheint ein weißes Blatt weiterhin weiß, unabhängig davon, ob es mit Sonnenlicht oder mit künstlichem Licht beleuchtet ist. Beim maschinellen Sehen verändern jedoch die Variationen in der Anzahl, Farbe oder Position der Lichtquellen die Objektfarben in den Aufnahmen und stellen somit ein Problem für die Farbsegmentierung dar.

Um den Einfluss der Beleuchtungsvariationen zu mindern, werden Farbnormalisierungsfilter auf den Aufnahmen angewendet. Angenommen die Kamera habe eine lineare Intensitätsfunktion, dann werden die Intensitätswerte der Pixel um einen Faktor a skaliert, sobald die Intensität des Lichtes um a skaliert wird. Dementsprechend werden die RGB-Werten (r, g, b) von jedem Pixel zu (ar, ag, ab). Um den Einfluss von a zu eliminieren ist folgender Normalisierungsfilter einsetzbar:

$$(\frac{r}{r+g+b}, \frac{g}{r+g+b}, \frac{b}{r+g+b}) \tag{3.13}$$

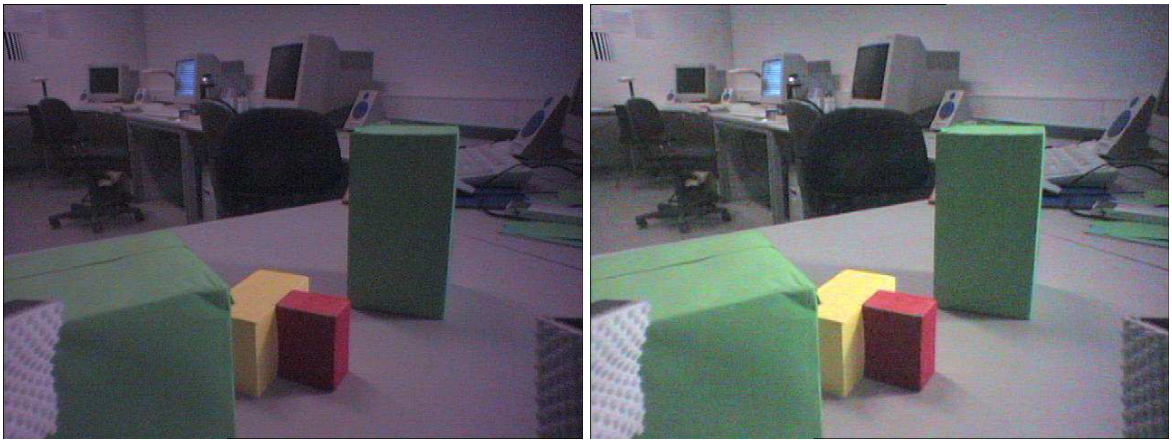
Diese Normalisierung wird oft in der Bildverarbeitung angewendet [SW95], [MMK95], [CB97]. Sie ergibt sehr gute Resultate unter unterschiedlichen Bedingungen und macht die Aufnahme unabhängig von der Intensität der Lichtquelle. Nach Finlayson et al. [FSC98] kann Gleichung 3.13 auch den Einfluss unterschiedlicher Lichtquellenanordnungen mindern.

Der Einfluss der Farbe der Lichtquelle ist ebenfalls einfach zu modellieren. Seien (r_1, g_1, b_1) und (r_2, g_2, b_2) die RGB-Werte der Abbildung von zwei Raumpunkten im Bild, und seien $(\alpha r_1, \beta g_1, \gamma b_1)$ und $(\alpha r_2, \beta g_2, \gamma b_2)$ die RGB-Werte derselben Punkte mit einer anderen Lichtquelle. Dann kann nach [WB86] der Einfluss von α, β und γ mit Gleichung 3.14 eliminiert werden:

$$(\frac{2r_1}{r_1+r_2}, \frac{2g_1}{g_1+g_2}, \frac{2b_1}{b_1+b_2}), (\frac{2r_2}{r_1+r_2}, \frac{2g_2}{g_1+g_2}, \frac{2b_2}{b_1+b_2}) \tag{3.14}$$

Dieser Fall für zwei Pixel kann für n Pixel erweitert werden. Dann ist der Nenner die Summe aller Pixel und der Zähler mit n multipliziert.

Jedoch kann keine der Gleichungen 3.13 und 3.14 gut funktionieren, wenn Leuchtdichte und Farbe gleichtzeitig variieren. Deswegen schlägt Finlayson et al. [FSC98] eine iterative, abwechselnde Anwendung der beiden Filter vor und beweist, dass die iterative Anwendung die Effekte der Farbe und Geometrie der Lichtquellen stark minimiert. Dieses Verfahren wird auch hier angewendet, um den Einfluss der Lichtvariationen zu reduzieren (Abbildung 3.9).



(a) (b)

Abbildung 3.9: Aufnahme vor und nach der Farb- und Leuchtdichtenormalisierung.

3.2.3 Entfernen von Rauschen

Das Reduzieren oder sogar Entfernen des Rauschens in einem Bild, ohne dabei die Aufnahme selbst zu verfälschen, ist ein Thema, das die Bildverarbeitung seit langem beschäftigt. Lineare Methoden umfassen unter anderem den *Sliding Mean* Filter [McD81] und den *Gaussian* Filter [Can86]. Der *Sliding Mean* Filter ersetzt jedes Pixel mit dem Mittelwert seiner Nachbarn. Der Gaussian operiert ähnlich, gewichtet aber die Nachbarn bei der Bildung des Mittelwertes anhand ihres Abstandes vom zu ersetzenden Pixel nach einer gaußschen Abstandsfunktion. Kompliziertere, nichtlineare Methoden umfassen unter anderem den *Weighted Median Filter* [Bro84], den *K-nearest Neighbour Operator* [DR78], den SMCM Filter [SMCM91] oder den *Gradient Inverse Weighted Operator* [WVL81]. Eine ausführliche Beschreibung der Filter ist in [SB97a] zu finden.

Nach der Fachliteratur erzielt der *SUSAN Noise Filtering* Algorithmus sowohl beim Entfernen des Rauschens als auch bei der Erhaltung der Struktur des Bildes [SB97a] gute Resultate. Der Algorithmus basiert auf dem *SUSAN*[6] Prinzip. Dabei wird eine kreisförmige Maske verwendet, deren Zentralpixel als Nukleus (*Nucleus*) bekannt ist. Wenn man die Pixel der Maske mit dem Nukleus vergleicht, dann kann eine Region der Maske definiert werden, die dieselbe oder ähnliche Intensitätswerte wie der Nukleus aufweist. Diese Region wird als *Univalue Segment Assimilating Nucleus* oder einfach USAN bezeichnet. Die Ähnlichkeit der Pixel wird mit folgender Funktion bewertet:

$$f(\vec{u}, \vec{u}_0) = e^{-(\frac{I(\vec{u})-I(\vec{u}-0)}{I_t})^{\alpha}} \tag{3.15}$$

wobei $\vec{u}_0 = (u, v)^T$ die Position des Nukleus ist, $\vec{u} = (u + i, v + j)^T$ die Position eines Pixels der Maske im Bild, $I(\vec{u})$ die Intensität des Pixels mit Position $\vec{u}$ und I_t ein Schwellwert für die Ähnlichkeit der Intensität. Der Parameter α ist eine Konstante, die normalerweise den Wert 2 annimmt.

Der *SUSAN Noise Filtering* Algorithmus erhält die Struktur des Bildes, indem er nur die Pixel berücksichtigt, die zur USAN-Region gehören. Bei einer Aufnahme mit $n \times m$ Pixel ist die Gleichung des Filters:

$$I_{neu}(v, u) = \frac{\sum\limits_{(i,j)\neq(0,0)} I(u + i, v + j) f(\vec{u}, \vec{u}_0) e^{-\frac{d^2}{2\sigma^2}}}{\sum\limits_{(i,j)\neq(0,0)} f(\vec{u}, \vec{u}_0) e^{-\frac{d^2}{2\sigma^2}}} \tag{3.16}$$

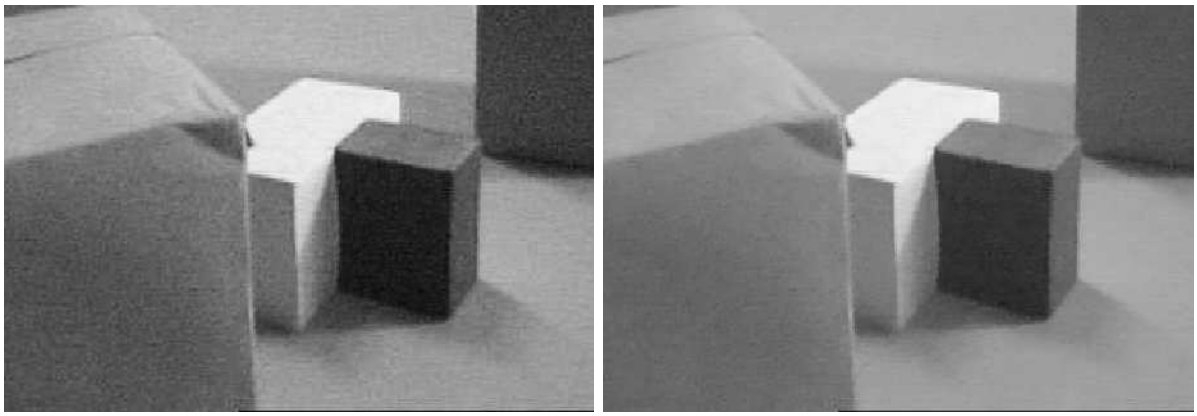
mit $u = 1, \ldots, n$, $v = 1, \ldots, m$, $\alpha = 2$, d der Abstand des Nachbarn vom Nukleus, also $d = \sqrt{i^2 + j^2}$, und σ ein Parameter, der die Unschärfe kontrolliert. $I_{neu}(v, u)$ ist der Intensitätswert des Pixels mit den Koordinaten (u, v) im neuen, gefilterten Bild. Falls der Nenner in Gleichung 3.16 Null ist, dann wird der Mittelwert der acht nächsten Pixel für $I(v, u)$ eingesetzt.

Der *SUSAN Noise Filtering* Algorithmus wird auf allen realen Aufnahmen angewendet; da virtuelle Aufnahmen kein Rauschen vorweisen, ist der Algorithmus dort überflüssig. In Abbildung 3.10 werden zwei Aufnahmen mit der realen *eye-in-hand* Kamera vor und nach Anwendung des *SUSAN Noise Filtering* Algorithmus präsentiert.

3.3 Teach-In in virtuellen Umgebungen

Im Fall des manuellen Teach-Ins werden die erwünschten Pfade, die zur Zielposition am Objekt führen, definiert, indem ein Operator den Manipulator manuell von variablen Anfangspositionen zur

[6] *Smallest Univalue Segment Assimilating Nucleus*



(a) (b)

Abbildung 3.10: Aufnahme vor (a) und nach (b) Anwendung des *SUSAN Noise Filtering* Algorithmus (Ausschnitt).

Zielposition in der Nähe des Objektes führt. Diese Vorgehensweise ist nicht praktikabel anwendbar, sobald eine große Anzahl von Pfaden vorgezeigt werden muss. Eine bessere Alternative ist, eine Sequenz von ähnlichen Pfaden, die den Manöverraum des Manipulator decken, anhand einer erwünschten Form zu generieren (algorithmisches Teach-In). Anderseits kann man auch dem Manipulator selbst die Exploration seiner Arbeitsumgebung überlassen (stochastisches Teach-In).

Beide Formen des Teach-Ins sind nur in der virtuellen Umgebung möglich, da Explorationsbewegungen des Manipulators in der realen Welt leicht zu Beschädigungen der Hardware führen können. Weiterhin lässt sich in der virtuellen Umgebung der Lernvorgang automatisieren, so dass wesentlich mehr Durchgänge als bei einem realen Training möglich sind. Bei Bedarf besteht sogar die Möglichkeit, die exakt gleiche Situation, Objekt- und Roboterpositionen zum mehrmaligen Training und zur Überprüfung des Greifvorgangs herzustellen.

Bei beiden Formen des Teach-Ins handelt es sich um Methoden zur Generierung und Akquisition der Trainingsdaten für die Steuerungsalgorithmen des Manipulators und nicht um Lernverfahren. Das algorithmische Teach-In generiert und sammelt die Daten für überwachte Lernmethoden, während das stochastische Teach-In beim Lernen mit Bewerter zum Einsatz kommt. Deswegen werden in den nächsten Abschnitten diese beide Methoden zum Lernen kurz erläutert. Anschließend werden die zwei Formen des Teach-Ins ausführlich vorgestellt.

3.3.1 Überwachtes Lernen

Beim überwachten Lernen (*Supervised Learning*), auch als Lernen mit Lehrer bekannt, erzeugt ein Lehrer die Trainingsdaten aus einer Menge von Eingangs-Ausgangs Beispielen der richtigen Handlungsweise bei einer gegebenen Situation [Kra03b]. Das zu lernende System, auch als Agent bezeichnet, hat die Aufgabe, aus der Menge der Beispiele eine Abbildung des Eingangsraumes auf den Ausgangsraum zu realisieren, die konsistent zur Trainingsmenge ist. Bei dieser Trainingsphase erfolgt für jeden präsentierten Eingangsvektor der Vergleich der Ausgabe des Agenten mit der erwünschten Soll-Ausgabe. Der Agent wird dann mit Hilfe des Eingangsvektors und des Fehlersignals angepasst; als Fehlersignal ist hier die Differenz zwischen Ist- und Soll-Ausgabe gemeint (siehe Abbildung 3.11).

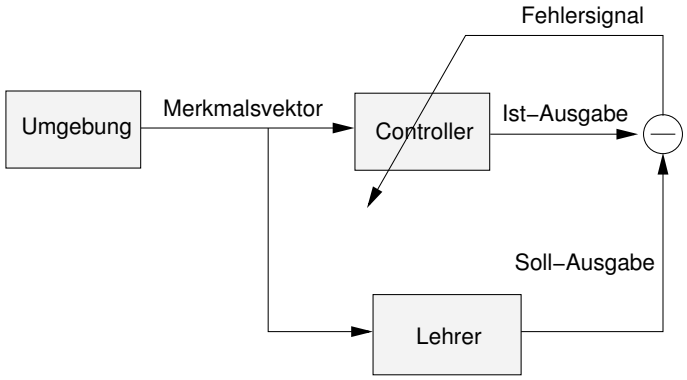


Abbildung 3.11: Verlauf beim Lernen mit Lehrer.

Ein Nachteil des überwachten Lernens ist, dass für die Trainingsdaten immer die Soll-Ausgaben des Systems[7] bekannt sein müssen. Entstehen außerdem Situationen, die vom Lehrer bzw. den Trainingsdaten nicht behandelt wurden[8], dann kann der Agent keine geeignete Ausgabe generieren bzw. er generiert eine falsche Antwort. Dies erfordert die Existenz eines Lehrers, der zu jeder Situation die entsprechende richtige Ausgabe produziert. So ein Lehrer ist jedoch in vielen Fällen nicht bekannt bzw. schwer zu definieren. Beispielsweise kann man bei der Verhaltenskoordination ohne aufwendiges Experimentieren nicht genau sagen, mit welchem Anteil jedes Verhalten bei der Bewegung des Manipulators beisteuern sollte. Man kann jedoch die neue Situation nach einer Roboteraktion qualitativ bewerten und Aussagen treffen, ob die Aktion den Roboter näher zu seinem Zielzustand gebracht hat oder nicht. Für solche Fälle ist das Lernen mit Bewerter geeignet [Mat94], das in dieser Arbeit für die Hindernisvermeidung des Greifers und den Koordinationsmechanismus der Verhalten zum Einsatz kommt.

3.3.2 Reinforcement Learning

Das Lernen mit Bewerter (*Reinforcement Learning*, RL) gehört zu der Klasse des nicht überwachten Lernens (*Unsupervised Learning*). Der lernende Agent probiert in einer Situation eine Aktion aus und erhält dafür von seiner Umgebung eine positive oder negative Reaktion. Anhand dieser Reaktion passt er seinen Auswahlmechanismus an. Dabei bekommt er für jede gute Aktion eine positive Bewertung, für jede schlechte eine negative. Ziel ist, dass der Agent sich so anpasst, dass die anschließend erhaltene Belohnung mit der Zeit maximiert wird.

Im Allgemeinen besteht das Lernen mit Bewerter aus vier Komponenten [SB98]:

- einer Handlungsstrategie π (*Policy*),
- einer Belohnungsfunktion $R()$ (*Reward Function*),
- einer Wertfunktion $V()$ (*Value Function*) und
- einem Weltmodell (optional).

[7] In dieser Arbeit entspricht die Sollausgabe des Systems dem objektspezifischen Pfad zur Zielposition, wenn keine Hindernisse vorhanden sind.

[8] Beispielsweise wenn Hindernisse vorhanden sind.

Die Handlungsstrategie entspricht der steuernden Komponente des Agenten. Sie bestimmt, welche Aktion der Agent in welcher Situation wählen soll und realisiert eine Abbildung von dem Zustandsraum[9] $\mathcal{Z}$ der Umgebung in den Aktionsraum[10] $\mathcal{A}$ zu: $\mathcal{Z} \mapsto \mathcal{A}$.

Die Belohnungsfunktion bewertet die Reaktion der Umgebung zu den Aktionen des Agenten und demnach die ausgeführte Aktion der Handlungsstrategie. Mataric [Mat94] teilt die Belohnungsfunktionen in ereignisgesteuerte (*event-driven*) und fortschrittschätzende (*progress-estimator*) Funktionen ein. Immer wenn ein für die Lösung der Aufgabe wichtiges Ereignis eintritt, gibt die ereignisgesteuerte Belohnungsfunktion als Antwort eine entsprechende, festgelegte Belohnung zurück. Tritt kein vorher definiertes Ereignis ein, wird der Wert Null zugewiesen:

$$R_{\text{event-driven}}(Z) = \begin{cases} r_{E_1} & \text{wenn Ereignis } E_1 \text{ eintritt,} \\ r_{E_2} & \text{wenn Ereignis } E_2 \text{ eintritt,} \\ \vdots & \vdots \\ 0 & \text{sonst.} \end{cases} \tag{3.17}$$

Im Gegensatz dazu messen fortschrittschätzende Belohnungsfunktionen zu jedem Zeitpunkt den Grad des Fortschrittes des Agenten bezüglich seines festgelegten Ziels. Diese Funktion ist nicht von bestimmten Ereignissen abhängig, sondern gibt jederzeit eine bestimmte Belohnung aus einer kontinuierlichen Wertemenge[11]. Sie kann positiv oder negativ sein, je nachdem, ob sich die Situation verbessert oder verschlechtert hat.

$$R_{\text{progress-estimator}}(Z) = \begin{cases} i & \text{wenn ein Fortschritt vom Grad } i \text{ gemacht wurde,} \\ -j & \text{wenn ein Rückschritt vom Grad } j \text{ gemacht wurde,} \\ 0 & \text{sonst.} \end{cases} \tag{3.18}$$

Der Grad i bzw. j ist ein Maß des Fortschrittes bzw. Rückschrittes und seine genaue Definition und Wertebereich hängen von dem spezifischen Problem ab.

Im Gegensatz zur Belohnungsfunktion schätzt die Wertfunktion, wie vorteilhaft es für den Agenten ist, in einem gegebenen Zustand zu sein (*State Value Function*) bzw. wie vorteilhaft eine Aktion in einem gegebenen Zustand ist (*Action Value Function*). Diese Schätzung berücksichtigt nicht nur die aktuelle sondern auch die erwartete zukünftige Belohnung, die der Agent aus dem gegebenen Zustand mit der Handlungsstrategie bis zum Ziel erhalten kann. Die beste Handlungsstrategie, die beim RL gesucht bzw. zu erlernen ist, ist diejenige, die die erhaltene Gesamtbelohnung über die Zeit maximiert; dadurch entspricht der besten Handlungsstrategie, auch als optimale *Policy* bezeichnet, die höchst mögliche Wertfunktion (*Optimal Value Function*). Ist diese bekannt, dann ist es einfach die optimale Handlungsstrategie zu bestimmen: Die Aktion, die von einem Zustand Z_i zu einem anderen Zustand Z_{i+1} mit dem höchsten Wert V_{max} der Wertfunktion führt, ist die Aktion, die eine optimale Handlungsstrategie ergibt.

Das Weltmodell dient der Vorhersage einer Situation, die sich aus der Anwendung einer oder mehrerer Aktionen in der Zukunft ergeben wird (*Prediction*). Demnach ermöglicht das Weltmodell die Berechnung der Abbildung $\mathcal{Z}_t \times \mathcal{A} \mapsto \mathcal{Z}_{t+n}$ und dadurch die Berechnung der Werte der Wertfunktion. Dennoch kann auch ohne Weltmodell ein optimales Verhalten erlernt werden, indem die optimale

[9] Der Zustandsraum umfasst alle möglichen Zustände, die die Umgebung annehmen kann.

[10] Der Aktionsraum beinhaltet alle Aktionen, die der Roboter ausführen kann, um den Zustand der Umgebung zu ändern.

[11] Dabei ist für die Berechnung der Belohnung die geeignete Soll-Ausgabe des Systems nicht bekannt; dies unterscheidet sie vom überwachten Lernen.

Wertfunktion anhand der gegebenen Belohnung geschätzt wird. Verfahren, die dies ermöglichen, sind beispielsweise der Monte Carlo Algorithmus und der TD-Ansatz, der im Anhang C beschrieben ist.

Sei π die Funktion der Handlungsstrategie und W das Weltmodell. Dann ist die generelle Form des RL Algorithmus:

1. Initialisiere den internen Zustand des lernenden Agenten zu I_0.
2. Erfasse den Zustand der Umgebung Z_i.
3. Wähle Aktion $\vec{\rho} = \pi_I(Z_i)$.
4. Führe Aktion $\vec{\rho}$ durch.
5. Berechne das *Reinforcement Signal* $r = R(Z_{i+1}, \vec{\rho})$
6. Aktualisiere den internen Zustand und die Handlungsstrategie π_I mit $I_{i+1} = U(W, I_i, Z_i, r, \vec{\rho})$.
7. Gehe zu Schritt 2.

Die Funktion U (*Update Function*) passt den internen Zustand des Agenten und daher die entsprechende Handlungsstrategie an die erhaltene Belohnung an. Ihre genaue Form ist vom eingesetzten Agenten abhängig.

Zur Bewertung der erzielten Lernergebnisse kann die erzielte Belohnung über die Anzahl der Lerndurchgänge benutzt werden [SB98]. Hierbei wird in den meisten Fällen eine über alle k Lerndurchgänge akkumulierte gemittelte Belohnung benutzt.

$$R_{akk}(k) = \frac{1}{k}\sum_{i=1}^{k} R_i = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{n_i}\sum_{j=1}^{n_i} r_j \tag{3.19}$$

Dabei stellt n_i die Anzahl der Lernschritte pro Durchgang dar. Diese Kurve konvergiert idealerweise gegen die maximal mögliche Belohnung, je näher die erlernte *Policy* sich der optimalen Handlungsstrategie nähert. In der Praxis gibt man sich jedoch mit einem gewissen prozentuellen Anteil hiervon zufrieden[12].

Nach Arkin [Ark98] kann das Lernen mit Bewerter auf intelligenten Robotern mit der Struktur aus Abbildung 3.12 implementiert werden. Der *Controller* stellt die eigentliche Robotersteuerung dar und enthält die zu erlernende Handlungsstrategie. Der Roboter führt die gewählte Aktion aus, deren Ergebnis wiederum eine Reaktion der Welt erzeugt. Diese Reaktion evaluiert der Bewerter (*Critic*) und gibt eine Belohnung an die Robotersteuerung zurück. Diese Struktur ist jedoch nicht ohne weiteres anwendbar, wenn die Robotersteuerung die Ausgaben von mehreren untergeordneten Komponenten kombinieren muss, um die ausführende Aktion zu bilden. Dies ist beispielsweise der Fall, wenn mehrere Verhalten gleichzeitig an der Bildung der Gesamtreaktion des Systems beteiligt sind. Dann ist es nicht einfach, einer individuellen Komponente eine Belohnung zu zuweisen, da der Bewerter nur eine Gesamtbelohnung für die Gesamtaktion verteilen kann, aber nicht weiß, wie diese berechnet wurde. Dieses Problem ist als *Credit Assignment Problem* bekannt und muss beim Entwurf eines RL-Systems beachtet werden [Ark98]. In dieser Arbeit wird dieses Problem bei der Verhaltenskoordinierung dadurch gelöst, dass für jedes Verhalten eine entsprechende Belohnungsfunktion definiert und bei jedem Lernszenario nur der Beitrag einer einzelne Fertigkeit angepasst wird[13].

[12] Alternativ kann man im Fall einer bekannten optimalen Handlungsstrategie die erlernte hiermit vergleichen. Dafür wird der prozentuelle Anteil der optimal getroffenen Entscheidungen aus allen Entscheidungen der erlernten *Policy* über die Anzahl der Lerndurchgänge aufgetragen.

[13] Andere Verhalten können gleichzeitig aktiv sein, sie dürfen jedoch nicht eine ähnliche Zielsetzung wie das zu erlernende Verhalten haben.

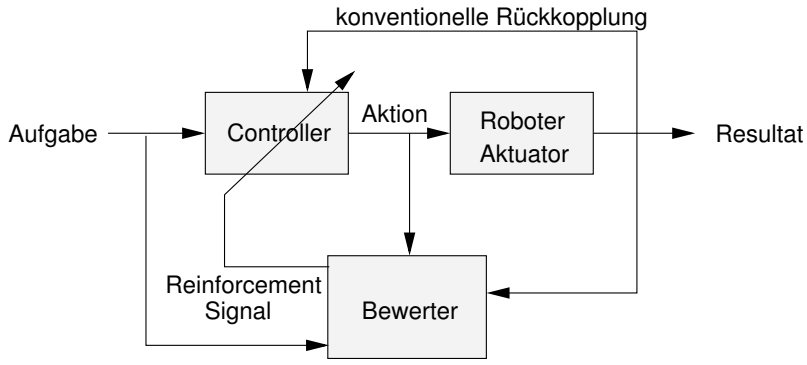


Abbildung 3.12: *Reinforcement Learning* nach Arkin [Ark98].

3.3.3 Algorithmisches Teach-In

Beim algorithmischen Teach-In wird eine Sequenz von ähnlichen Pfaden, die den Manöverraum des Manipulators abdecken, anhand einer erwünschten Form generiert. Der hier ausgewählte Vorgang, um die erwünschte Form des Pfades zu definieren, ist, die Pfade aus dreidimensionalen Flächen, wie einem elliptischen Paraboloid oder einem einschaligen kreisförmigen Hyperboloid, zu generieren [MZBK02]. Solche Flächen haben gemeinsam, dass bei Änderung der Parameter ihre Gleichungen eine neue Fläche mit ähnlichen Charakteristika produzieren. Ein Pfad mit der erwünschten Form wird dann über den Schnitt der Fläche mit einer Ebene definiert.

Im speziellen Fall des Objektes des Testszenarios eignet sich ein modifiziertes einschaliges kreisförmiges Hyperboloid (Abbildung 3.13, Gleichung 3.20), da es eine trichterähnliche Form aufweist, die den Manipulator zur Zielposition am Griff eines Objektes führt. Diese Fläche ergibt sich, indem man in der Gleichung des einschaligen Hyperboloiden den Exponenten von z von 2 auf 4 setzt.

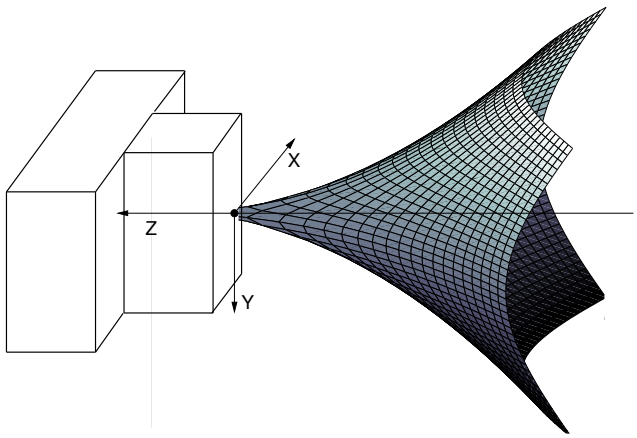


Abbildung 3.13: Modifiziertes einschaliges kreisförmiges Hyperboloid.

$$\frac{x^2}{\alpha^2} + \frac{y^2}{\alpha^2} - \frac{z^4}{\gamma^2} = 1, \qquad z \leq 0 \tag{3.20}$$

Der Parameter α definiert den Radius des Schnittes des Hyperboloiden mit der $x-y$-Ebene und somit die Genauigkeit der Endposition am Zielobjekt. Durch Variieren des Parameters γ wird dagegen eine Folge von ähnlichen Hyperboloiden definiert, die den Arbeitsbereich um das Objekt abdecken und durch dieselbe Endposition am Objekt (Schnitt mit der $x-y$-Ebene) verlaufen.

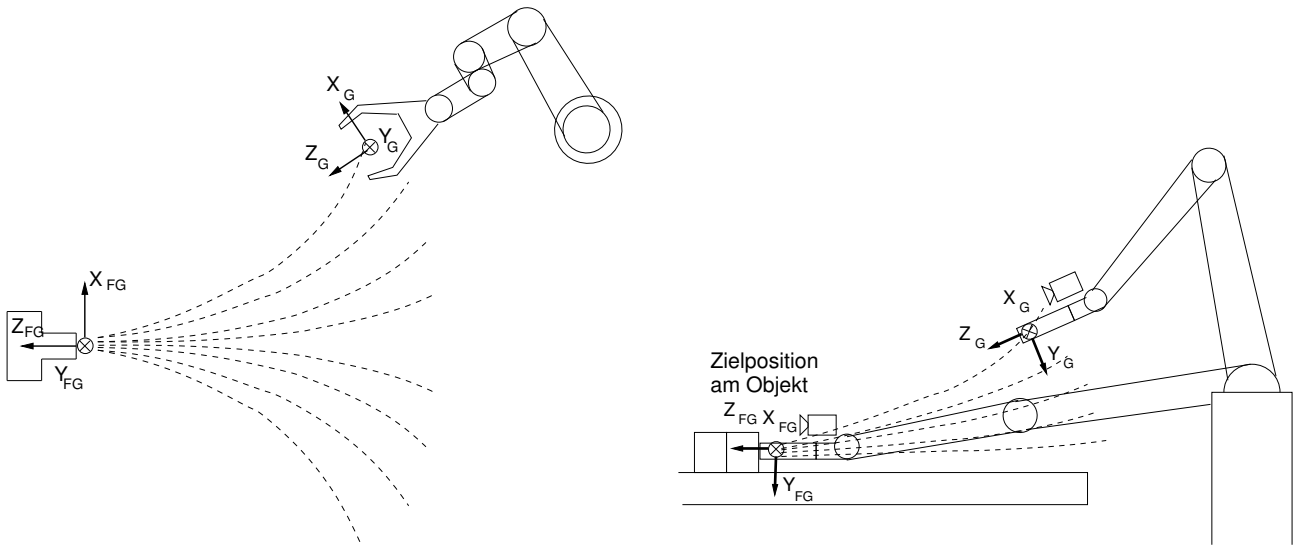
Gleichung 3.21 definiert eine Ebene, die entlang der z-Achse verläuft, den Ursprung des Koordinatensystems beinhaltet und einen Winkel θ mit der y-Achse bildet. Indem man den Wert von θ variiert, wird die Ebene um die z-Achse rotiert.

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = 0 \tag{3.21}$$

Auflösen nach y und Ersetzen in Gleichung 3.20 ergibt:

$$\frac{x^2}{\frac{\alpha^2}{1+\cot^2(\theta)}} - \frac{z^4}{\gamma^2} = 1 \tag{3.22}$$

Im Fall $\theta = 0^0$ entspricht die Ebene der $y-z$-Ebene des Koordinatensystem und die entsprechenden Pfade für verschiedene Werte von γ sind in Abbildung 3.14(a) dargestellt. Für $\theta = 90^0$ stimmt die Ebene mit der $x-z$-Ebene überein und die Pfade sind in Abbildung 3.14(b) zu sehen. Der Fall mit $\theta = 90^0$ und einer Objektposition, die ein Greifen von oben erfordert, ist in Abbildung 3.15 präsentiert.



(a) (b)

Abbildung 3.14: Generierte Pfade aus dem Schnitt des einschalige kreisförmigen Hyperboloiden mit der Ebene aus Gleichung 3.21 für $\theta = 0^0$ (links) und $\theta = 90^0$ (rechts).

Um die Pfade zu definieren, wird zuerst die erwünschte Zielposition und Orientierung am Objekt bestimmt; daher ist die Transformationsmatrix ${}^{O}_{FG}\mathbf{T}$ des Greiferkoordinatensystems in der Endposition relativ zum Objekt bekannt (siehe Abbildung 3.16). Diese Position definiert auch den Ursprung und die Ausrichtung des Hyperboloiden. Dann wird der Parameter α anhand der erwünschten Genauigkeit an der Zielposition gesetzt. Indem man θ und γ variiert, wird eine Familie von ähnlichen Pfaden im

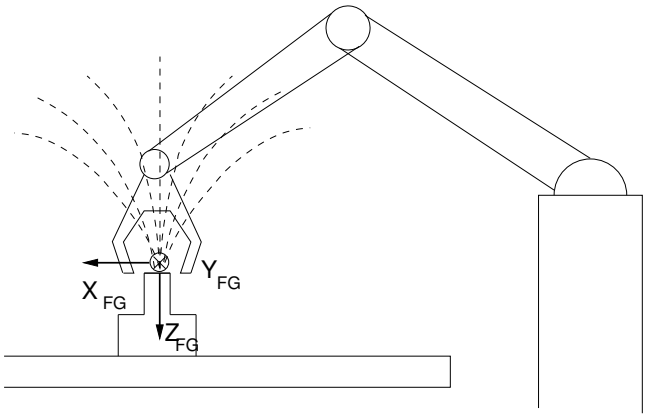


Abbildung 3.15: Pfade, wenn der Manipulator das Objekt von oben greift.

Raum erzeugt. Diese Pfade sind relativ zur Endposition des Greifers am Objekt definiert. Die Anzahl der produzierten Pfade ist das Produkt der Anzahl der verschiedenen Parameter γ und θ.

Anhand von ${}^{O}_{FG}\mathbf{T}$ können die Pfade von dem Greiferkoordinatensystem in der Zielposition zu dem Objektkoordinatensystem transformiert werden, indem jeder Punkt der Pfade ${}^{FG}\vec{P}_i$ relativ zum Objektsystem, also ${}^{O}\vec{P}_i$, transformiert wird:

$$ {}^{O}\vec{P}_i = {}^{O}_{FG}\mathbf{T}\, {}^{FG}\vec{P}_i \tag{3.23}$$

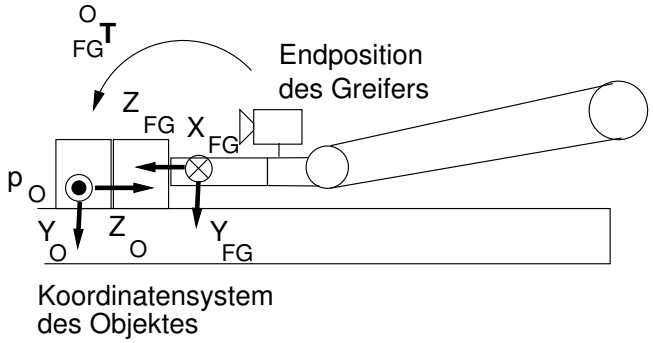


Abbildung 3.16: Koordinatensysteme bei der erwünschten Zielposition des Greifers am Objekt.

Die Pfade definieren Positionen, jedoch nicht die Orientierungen des Greifers im kartesischen Raum. Für die Orientierung kann man den tangentialen Vektor des Pfades zu jeder Pfadposition berechnen und seine Ausrichtung als Orientierung verwenden; diese Lösung hat jedoch den Nachteil, dass während der Bewegung die Manipulatorkamera das Objekt aus ihrem Sichtfeld verlieren kann. Deswegen wird die Orientierung des Greifers auf dem Pfad so berechnet, dass die Manipulatorkamera das Zielobjekt im Zentrum ihres Sichtfeldes behält. Dafür muss zuerst die Position des Objektes relativ zur Kamera berechnet werden. Sei ${}^{W}_{O}\mathbf{T}$ die homogene Matrix des Objektkoordinatensystems bezüglich des Ursprungs der virtuellen Umgebung, ${}^{W}_{P}\mathbf{T}$ die Lage der Plattform in der virtuellen Umgebung, ${}^{P}_{MB}\mathbf{T}$ die Lage der Manipulator Basis relativ zur mobilen Plattform, ${}^{MB}_{G}\mathbf{T}$ das Koordinatensystem des Greifers bezüglich der Manipulator Basis und ${}^{G}_{K}\mathbf{T}$ die Transformationsmatrix des Kamerakoordinatensystems zum Greifer (Abbildung 3.8). Dann ist die Objektposition relativ zur Kamera:

$$ {}^{K}_{O}\mathbf{T} = ({}^{W}_{O}\mathbf{T})^{-1}\, {}^{W}_{P}\mathbf{T}\, {}^{P}_{MB}\mathbf{T}\, {}^{MB}_{G}\mathbf{T}\, {}^{G}_{K}\mathbf{T} \tag{3.24}$$

${}^{W}_{O}\mathbf{T}$ und ${}^{W}_{P}\mathbf{T}$ sind aus der virtuellen Umgebung bekannt, ${}^{P}_{MB}\mathbf{T}$ und ${}^{G}_{K}\mathbf{T}$ sind abhängig von der Plattform - Manipulator Konfiguration und müssen aus der Geometrie des Roboters bzw. aus der Kamerakalibrierung ermittelt werden. ${}^{MB}_{G}\mathbf{T}$ kann aus der Kinematik des Roboterarmes berechnet werden (Anhang C.1).

Um den Greifer und dadurch auch die Kamera zum Objekt auszurichten, werden die *yaw*- und *pitch*-Winkel des Greifers angesteuert; der *roll*-Winkel wird von der Ausrichtung an der Zielposition definiert und auf konstantem Wert gehalten[14]:

$$\begin{aligned} pitch &= atan2(t_{14},\ t_{34}) \\ yaw &= -atan2(t_{24}\cos(pitch),\ t_{34}) \end{aligned} \tag{3.25}$$

Dabei sind $t_{i4},\ i = 1, 2, 3$, die Elemente der vierten Spalte der ${}^{K}_{O}\mathbf{T}$ Matrix und repräsentieren die Position des Objektes im Koordinatensystem der Kamera.

Um die Trainingsdaten zu sammeln, werden die generierten Pfade durchlaufen und die Aufnahmen der virtuellen Kameras mit den entsprechenden Positionsdaten des Manipulators gespeichert. Pfade, die durch die Auflageflächen in der virtuellen Umgebung laufen bzw. Kollisionen des Roboterarms verursachen, können über Kollisionsmeldungen der Simulationsumgebung erkannt und vom weiteren Training ausgeschlossen werden.

Sei $\vec{\mathcal{P}}$ ein 6×1 Vektor, der die erwünschte Position und Orientierung des Greifers auf dem Pfad relativ zum Objekt repräsentiert. Um einen generierten Pfad $\mathcal{P}_j$, $j = 1, .., n$, mit n die Anzahl der generierten Pfade, zu speichern, wird die Position und Orientierung des Greifers $\vec{\mathcal{P}}_{ji}$ auf dem Pfad in regulären Abständen in einer Liste gespeichert. $\vec{\mathcal{P}}_{ji}$ sind auch als Stützstellen des Pfades bekannt. Dadurch kann ein Pfad als eine Reihenfolge von Stützstellen dargestellt werden:

$$\mathcal{P}_j = \{\vec{\mathcal{P}}_{j1}, .., \vec{\mathcal{P}}_{ji}, .., \vec{\mathcal{P}}_{jm}\} \tag{3.26}$$

mit m die Anzahl der Stützstellen. Falls auf alle Pfade äquidistante Intervalle verwendet werden, dann verfügen die Pfade $\mathbf{P}_j$ über dieselbe Anzahl von Stützstellen und dadurch dieselbe Länge. Alle Pfadlisten werden abgespeichert und ihr Index j wird benutzt, um sie zu referenzieren.

3.3.4 Stochastisches Teach-In

Beim algorithmischen Teach-In ist die gewünschte Bewegung des Manipulators bekannt. Es gibt jedoch Fälle, wo die gewünschte Aktion des Manipulators nicht a priori bekannt oder nur schwer zu definieren ist. Dies ist beispielsweise der Fall bei der Verhaltenskoordination. Hier ist die Anwendbarkeit der Verhalten in einer Situation nicht genau bestimmbar, man kann jedoch eine qualitative Bewertung der Verhaltensanwendung anhand der sich ergebenden Situation machen[15]. Ähnlich ist auch bei der Hindernisvermeidung die optimale Bewegung des Manipulators in einer gegebenen Situation nicht genau bekannt. Nach Matarić [Mat94] ist für solche Fälle das Lernen mit Bewerter geeignet.

Das stochastische Teach-In ist ein Verfahren zur Generierung und Akquisition der Trainingsdaten, um das Lernen mit Bewerter in einer virtuellen Umgebung zu unterstützen. Es generiert mit einem

[14] $atan2(x, y)$ entspricht der $\tan^{-1}(\frac{x}{y})$ Funktion, berücksichtigt jedoch die Vorzeichen von x, y, um den Winkel im Bereich $(-180, 180]$ zu berechnen.

[15] Diese Bewertungen können beispielsweise folgende Form haben: "In der gegebenen Situation sollte dieses Verhalten eine hohe Anwendbarkeit haben, das andere jedoch nicht".

Zufallszahlengenerator eine beliebige Konfiguration in der virtuellen Umgebung von n Hindernissen mit variabler Größe und dem Zielobjekt auf einer Auflagefläche. Nachdem der Manipulator in eine Ausgangsposition gebracht wird, wird ein Greifvorgang initiiert. Bei jedem Schleifendurchlauf wird der jeweils aktuelle Merkmalsvektor $\vec{s}_t$ der aktuellen Aufnahmen, der Positionsvektor des Manipulators $\vec{\mathcal{P}}_t$ und die Aktion $\vec{\rho}_t$, die der Roboterarm ausführen wird, gespeichert. Weiterhin wird jedes Mal der Bewerter[16] aufgerufen, der aufgrund der gegebenen Situation eine Belohnung r_t erzeugt.

$$r_t = R(\vec{\mathcal{P}}_t, \vec{s}_t, \vec{\rho}_t) \tag{3.27}$$

Unterschreitet der minimale Abstand des Greifers bzw der Manipulatorsegmente von einem der Objekte einen Schwellwert d_{Kol}, wird dies als Kollision interpretiert und der Schritt wird abgebrochen. Erst dann werden die Parameter der Steuerungsalgorithmen, die im internen Zustand I_t zusammengefasst sind, anhand der vergebenen Belohnungen für jeden Schritt angepasst, so dass keine Kollisionen mit Hindernissen stattfinden. Dabei können auch in anderen Schritten vergebene Belohnungen in Betracht gezogen werden, um das Lernen effizienter zu gestalten:

$$I_t = U(I_t, \vec{\mathcal{P}}_t, \vec{s}_t, r_t, ..., r_{t-n}, \vec{\rho}_t) \tag{3.28}$$

Danach wird der Vorgang aus derselben Position und derselben Konfiguration wiederholt, bis die Ausgangssituation bewältigt ist (Abbildung 3.17).

Durch das stochastische Teach-In kann der Manipulator mit beliebigen Konstellationen von Hindernissen und Zielobjekt konfrontiert werden und seine Steuerung entsprechend anpassen, ohne dass eine direkte Vorgabe der richtigen Aktion notwendig ist[17]. Da besonders zu Beginn des Trainings viele Kollisionen auftreten können, hat das stochastische Teach-In den Vorteil, gefahrlos abzulaufen. Der Lernvorgang lässt sich automatisieren, so dass wesentlich mehr Durchgänge als bei einem realen Training möglich sind. Außerdem besteht die Möglichkeit, bei Bedarf die exakt gleiche Situation, Objekt- und Roboterpositionen zum mehrmaligen Training des Greifvorgangs herzustellen.

[16] Der Bewerter ist anwendungsspezifisch und die genaue Implementierung hängt von den zu lernenden Algorithmen ab.

[17] Der Ansatz besitzt eine Ähnlichkeit zu *Piagets' Circular Motion* [YKD97], [WT98], [BG97]. Beide Verfahren produzieren eine Bewegung des Roboterarmes, die für das Erlernen der bildgestützten Steuerung des Roboterarmes eingesetzt wird. Jedoch gibt es einen grundlegenden Unterschied. *Piagets' Circular Motion* ist ein Verfahren zur Unterstützung von überwachten Lernverfahren, da der Roboter nach seiner Bewegung die neue Ist-Position mit dem Merkmalsvektor als Trainingspaar verwendet. Beim stochastischen Teach-In dagegen findet eine Bewertung der Aktion des Manipulators statt und anhand dieser wird die Robotersteuerung angepasst.

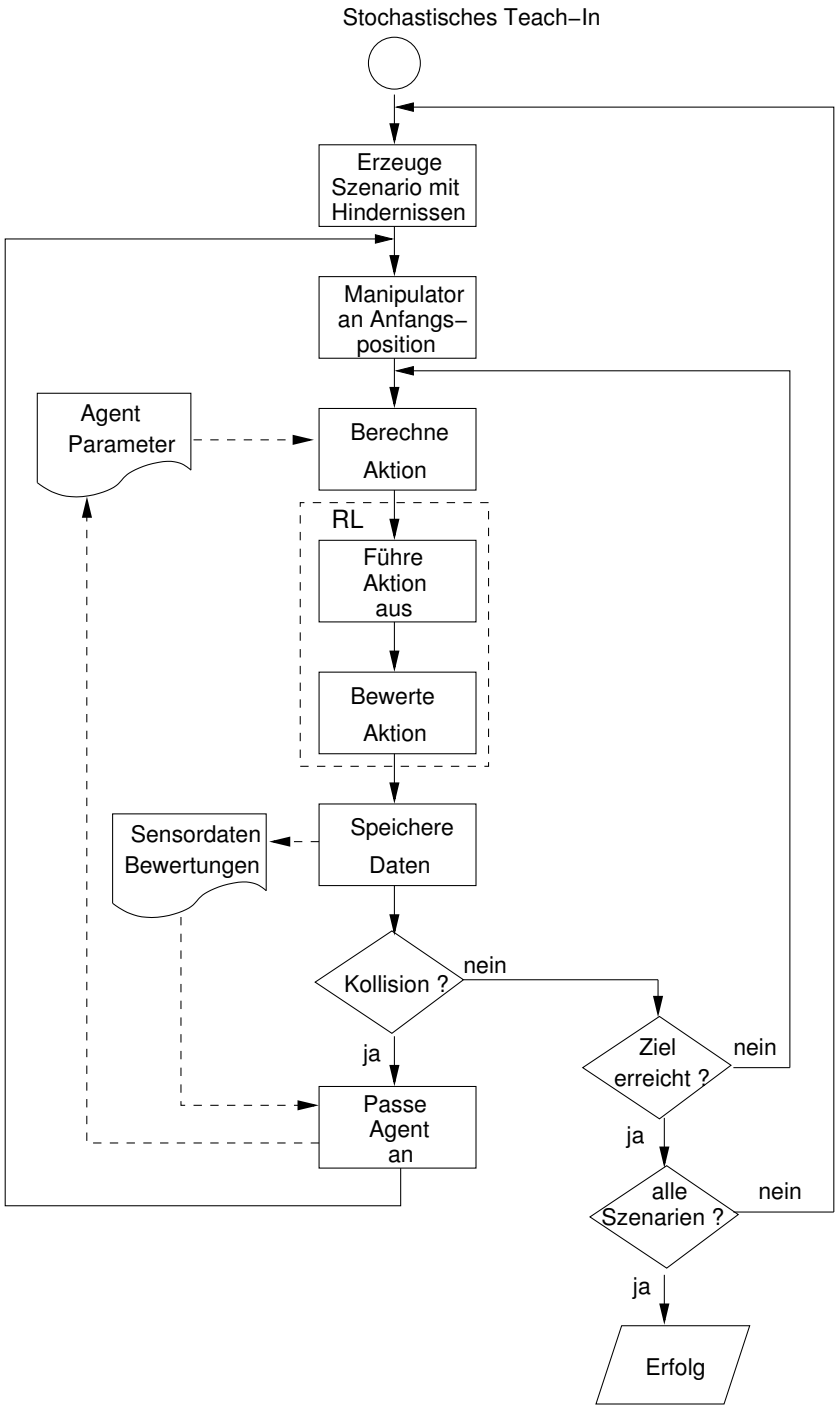


Abbildung 3.17: Flussdiagramm des stochastischen Teach-Ins.

Kapitel 4

Bildgestützte reaktive Verhalten zur Manipulation

Die reaktive Ebene des mobilen Manipulators hat die Aufgabe, direkt auf Änderungen der Umgebung zu reagieren und die Bewegung des Roboter entsprechend anzupassen. In den meisten hybriden Architekturen besteht sie aus mehreren Fertigkeiten, die der mobile Manipulator einzeln oder auch parallel einsetzen kann, um eine gestellte Aufgabe zu bewältigen. Dabei handelt es sich meistens um Module mit einem festgesetzten Ziel, die über kein Modell der Umgebung verfügen [Bro86] und die einfach realisiert sind, um eine schnelle Reaktion und Anpassung der Roboterbewegung auf neue Sensordaten zu ermöglichen.

In Rahmen dieser Arbeit wurde die reaktive Ebene des Manipulators implementiert und die benötigten Verhalten realisiert. Die implementierten Verhalten verwenden Kameraaufnahmen, um den Manipulator zum Ziel zu führen, das Objekt im Sichtfeld der Manipulatorkamera zu halten oder den Greifer bzw. die Manipulatorsegmente vor Kollisionen zu bewahren. Weiterhin ist ein Pfadplanungsverhalten realisiert worden, um die reaktiven Verhalten zu unterstützen und Situationen mit lokalen Minima auflösen zu helfen. Zum Training und zur Evaluierung der Verhalten kommen die virtuelle Umgebung und die beiden Teach-In Verfahren, die in Kapitel 3 schon beschrieben wurden, zum Einsatz.

Der allgemeine Verlauf einer in dieser Arbeit implementierten Fertigkeit ist in Abbildung 4.1 zu sehen. Die Bildvorverarbeitung umfasst den Abgleich der Aufnahmen der realen mit den entsprechenden virtuellen Kameras. In der Merkmalsextraktion werden zuerst die verschiedenen Objekte anhand ihrer Farbe im Bild detektiert und als Zielobjekt oder Hindernisse klassifiziert. Danach werden Merkmale extrahiert, die die Position und Form der Objekte im Bild bzw. im Raum beschreiben, und als Eingaben für die Steuerungskomponente eingesetzt. Diese berechnet die nächste Bewegung des Roboterarmes in Form eines Pfades[1].

Ein Pfad (*Path*) besteht aus einer Sequenz von kartesischen Greiferpositionen bzw. von Gelenkstellungen, auch als Stützstellen bekannt, die der Roboter nacheinander abfahren muss, um zum Ziel zu kommen. Der Pfad ist somit eine rein geometrische Beschreibung einer Bewegung. Eine Trajektorie

[1] Alternativ kann man die Bewegung des Manipulators mit einem Geschwindigkeitsvektor beschreiben. Dies erfordert jedoch eine sehr schnelle Ausführungsfrequenz für jedes Verhalten, die, wenn man alle Software-Komponenten des mobilen Manipulators berücksichtigt, nicht garantiert werden kann. Deswegen empfiehlt sich, dass Verhalten Pfade mit der erwünschten Bewegungsform ausgeben. Diese kann der Roboterarm ausführen, bis neue Daten von der reaktiven Ebene vorliegen.

(*Trajectory*) beinhaltet dagegen zusätzliche Information bezüglich der zeitlichen Ausführung der Bewegung; hier ist die erwünschte Position und Geschwindigkeit des Roboters zu jedem Zeitpunkt eindeutig definiert [Cra89]. Damit der Roboterarm einen Pfad, den ein Verhalten ermittelt hat, abfahren kann, muss er ihn in eine Trajektorie transformieren, die er dann mit Hilfe eines Trajektoriereglers[2] ausführt. Der resultierende Gesamtpfad wird anschließend mit dem Trajektorieregler, der in Anhang C.1 zusammengefasst ist, ausgeführt.

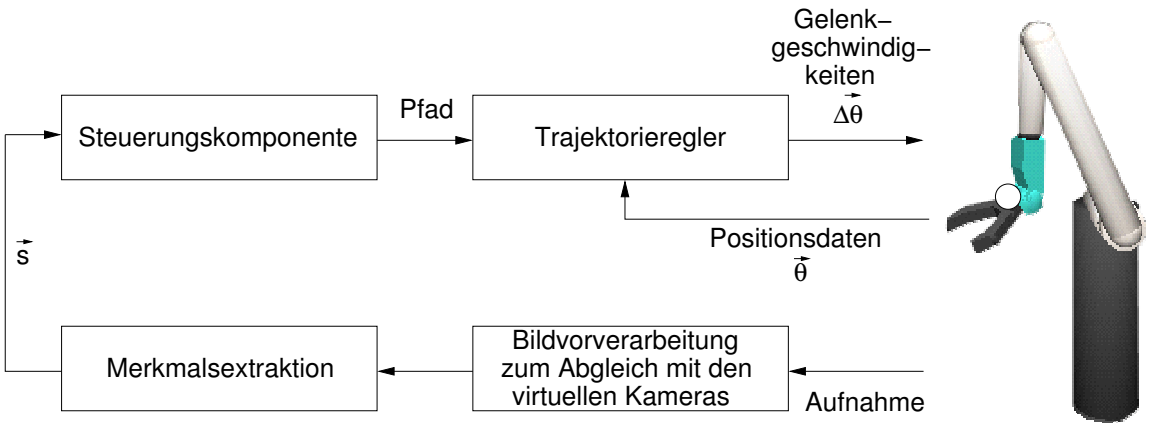


Abbildung 4.1: Komponenten eines Verhaltens.

Die Steuerungskomponenten der Verhalten sind entweder explizit programmiert oder mit neuronalen Netzen erlernt. Das Lernen hat dabei die Vorteile, dass es die Anpassung an interne und externe Änderungen ermöglicht und die Integration von Wissen, insbesondere für den Fall eines schwer zu modellierenden Systems, vereinfacht. In der vorliegenden Arbeit kommen beide Ansätze zum Einsatz: zwei bildgestützte Verhalten, die Zielführung und die Hindernisvermeidung des Greifers, wurden mit künstlichen neuronalen Netzen erlernt während zwei weitere, die Hindernisvermeidung der Gelenke und das Pfadplanungsverhalten, explizit programmiert wurden. Diese Verhalten werden auch in den nächsten Unterkapiteln detailliert vorgestellt[3].

4.1 Bildgestützte Zielführung

Ein Großteil der Aufgaben eines mobilen Manipulators erfordert das Erreichen mit dem Greifer einer objektspezifischen Zielposition an einem Objekt, um dieses anschließend robust zu greifen und zu manipulieren. Der Roboterarm muss eine Türklinke von oben greifen, um beim Herunterdrücken sein eigenes Gewicht auszunutzen, während das Zielobjekt im Testszenario an dem roten Griff senkrecht zur gelben Basis gegriffen werden sollte, um einen rutschfreien Griff zu gewährleisten. Der Annäherungsvorgang sollte auch objektspezifisch sein und den Greifer entlang eines stetigen Pfades zur Zielposition führen, da ein solcher Pfad robust auszuführen und zu regeln ist. Weiterhin macht ein glatter, objektspezifischer Pfad das Verhalten des mobilen Manipulators für die mit ihm interagierenden Menschen voraussehbar und erleichtert dadurch den Umgang mit dem Roboter.

[2] Alternativ werden bei der Koordination von mehreren Verhalten die berechneten Pfade der Fertigkeiten an die vermittelnde Ebene weitergeleitet, um dort miteinander interpoliert und anschließend ausgeführt zu werden.

[3] Die theoretischen Grundlagen, die den implementierten Fertigkeiten zugrunde liegen, sind im Anhang C zusammengefasst.

Im Abschnitt 3.3.3 wurde das algorithmische Teach-In vorgestellt, das Pfade einer erwünschten Form in der virtuellen Umgebung erzeugt. Diese Pfade kann man als Vorgabe zum Erlernen des zielführende Verhaltens verwenden. Nach dem Training erkennt das Verhalten anhand der aktuellen Aufnahme der Greiferkamera, welche der im algorithmischen Teach-In generierten Pfade am nächsten zur aktuellen Position des Greifers verlaufen. Indem man diese Pfade nach ihrem Abstand zur aktuellen Greiferposition gewichtet interpoliert, entsteht ein neuer Pfad, der mit genügend Genauigkeit zur Zielposition führt.

Abbildung 4.2 stellt den Ablauf des hier implementierten Verhaltens dar. Die Aufnahmen der Kamera werden nach der Bildvorverarbeitung segmentiert und die Merkmalsvektoren $\vec{s}_A$, $\vec{s}_P$ für die Steuerungskomponente extrahiert. Die Steuerungskomponente selbst besteht aus zwei Ebenen, wobei die erste den relativen Abstand des Greifers zum Objekt ermittelt. Die zweite Ebene berechnet die Gewichtungen für die im algorithmischen Teach-In generierten Pfade entsprechend ihres Abstandes zur aktuellen Greiferposition. Anhand der Information aus beiden Ebenen werden in einem nachfolgenden Schritt die Pfade gewichtet interpoliert. Anschließend übergibt die Steuerungskomponente dem Trajektorieregler den resultierenden Gesamtpfad. Das Verhalten wartet dann auf eine neue Aufnahme aus der Greiferkamera, um erneut ausgeführt zu werden, bis der Greifer die Endposition am Zielobjekt erreicht hat.

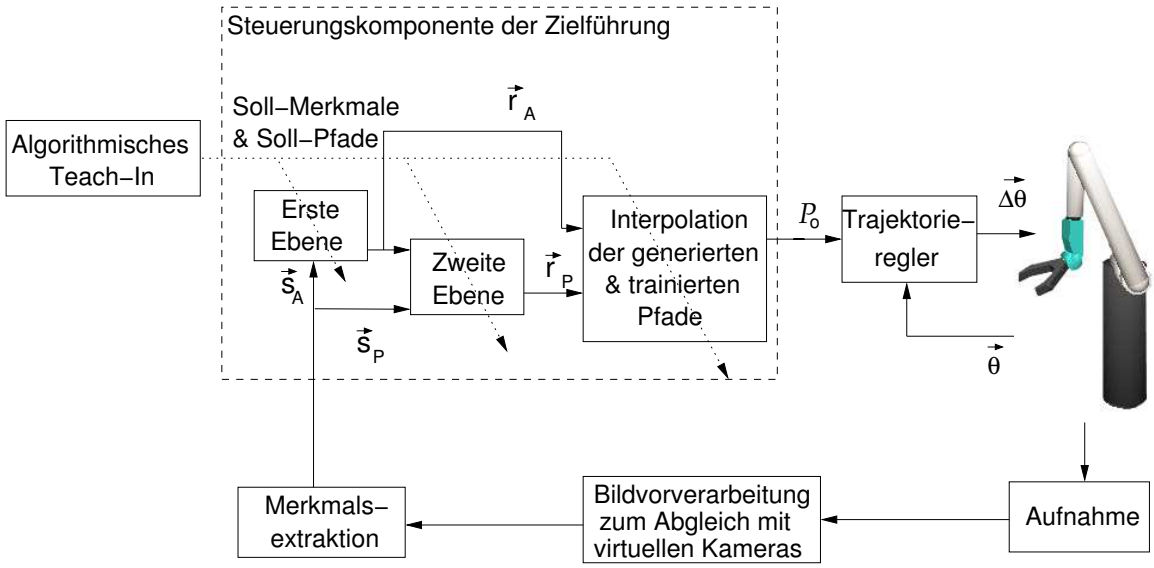


Abbildung 4.2: Ablauf der bildgestützten Zielführung.

Der realisierte Ansatz ist den positionsbasierten Verfahren zuzuordnen. Dadurch kann er, im Gegensatz zu bildbasierten Ansätzen, in einer großen Region um das Zielobjekt zum Einsatz kommen. Alternative Verfahren, wie das 2 1/2D *Visual Servoing* [MCB98] oder die Schätzung der Jacobi-Matrix [JN96], [HA94], können hier nicht zum Einsatz kommen, da die Ungenauigkeit des Manipulators bei der Positionsangabe zu groß ist.

In den folgenden Unterabschnitten werden zunächst die einzelnen Komponenten des Verhaltens vorgestellt. Anschließend wird auf das Training in der virtuellen Umgebung eingegangen und eine Evaluierung der Fertigkeit in der virtuellen und der realen Umgebung vorgenommen.

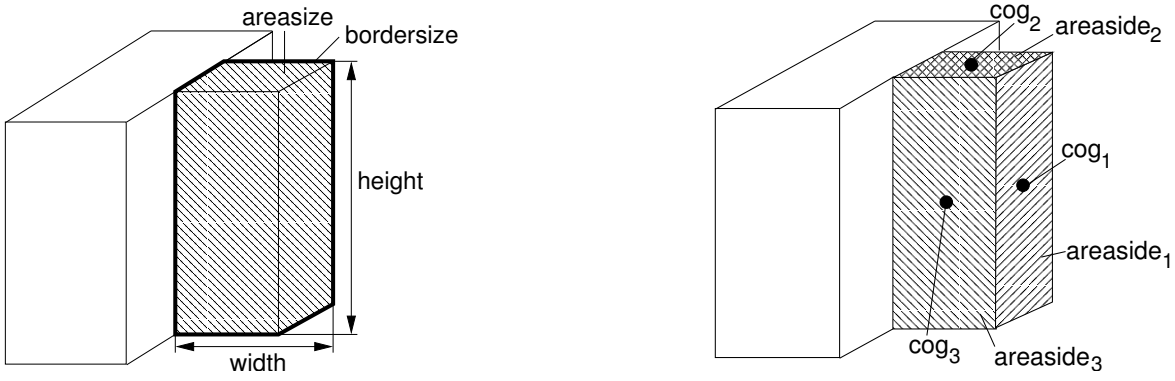
4.1.1 Merkmalsextraktion

Aus den Aufnahmen der Greiferkamera müssen Merkmale extrahiert werden, die die Lage des Zielobjektes im Bild beschreiben und dadurch die Aufnahmeposition eindeutig charakterisieren. Als besonders robust haben sich einfache Flächenmerkmale erwiesen, wie der Flächeninhalt, der Schwerpunkt, der umgebende Quader oder die Kompaktheit, da sie auf die Berechnung von Flächeninhalten basieren. Somit haben Segmentierungsfehler über die Summierung der Pixel einen kleineren Einfuß als bei der Extraktion einzelner Punkte[4].

Der Merkmalsvektor für die erste Ebene der Steuerungskomponente muss genügend Information über die relative Distanz zum Objekt beinhalten. Hier kommen Merkmale zum Einsatz, die die Fläche des roten, in Abbildung 4.3a schraffiert dargestellten Griffes[5] in der Aufnahme beschreiben, denn je näher die Greiferkamera dem Objekt kommt, desto größer erscheint dieses im Kamerabild. Zusätzlich wird der Schwerpunkt der Fläche im Bild bestimmt. Der Merkmalsvektor ist

$$\vec{s}_A = (\, areasize,\, bordersize,\, height,\, width\,), \quad \vec{u}_S = (u_S, v_S) \tag{4.1}$$

wobei $areasize$ der Flächeninhalt des extrahierten Griffes in Pixel ist, $bordersize$ die Peripherie der extrahierten Fläche, $height$ und $width$ die Höhe und Breite des umgebenden Quaders angeben und $\vec{u}_S$ dem Schwerpunkt der extrahierten Fläche im Bild entspricht.



(a) (b)

Abbildung 4.3: Bestandteile der beiden Merkmalsvektoren (a) $\vec{s}_A$ und (b) $\vec{s}_P$.

In der zweiten Ebene werden die Pfade entsprechend ihres Abstandes zur aktuellen Aufnahmeposition der *eye-in-hand* Kamera gewichtet. Wie in Abbildung 4.4 zu sehen ist, kann man für einen bestimmten Abstand vom Objekt die nächsten Pfade über die Orientierung der aktuellen Kameraposition zur Zielposition ermitteln. Die Information über die Orientierung ist wiederum in den Verhältnissen der Seiten des Griffes zueinander enthalten. Somit kann man folgenden Merkmalsvektor als Eingabe der zweiten Ebene einsetzen (siehe Abbildung 4.3b):

[4] Insbesondere bei stark verrauschten Aufnahmen, wie bei den Bildern der hier eingesetzten Manipulatorkamera, hat sich die Detektion und Lokalisierung von Pixelpunkten im Bild als besonders schwierig erwiesen. Deshalb sind hier Punkte als Merkmale ungeeignet.

[5] In einer Vorverarbeitungsstufe wird zuerst der rote Griff des Zielobjektes anhand seiner Farbe extrahiert. Danach werden die einzelnen Seiten erneut farbbasiert extrahiert, korrespondiert und ihre Flächeninhalte berechnet.

$$\vec{s}_P = \begin{pmatrix} \frac{areaside_2}{areaside_1}, \frac{areaside_3}{areaside_1}, \frac{areaside_3}{areaside_2}, \frac{areaside_4}{areaside_1}, \\ \|cog_1 - cog_2\|, \|cog_1 - cog_3\|, \|cog_2 - cog_3\|, \\ \|cog_1 - cog_4\|, \|cog_2 - cog_4\|, \|cog_3 - cog_4\| \end{pmatrix} \tag{4.2}$$

wobei der Index 4 der in Abbildung 4.3b verdeckten Seite des Griffes entspricht.

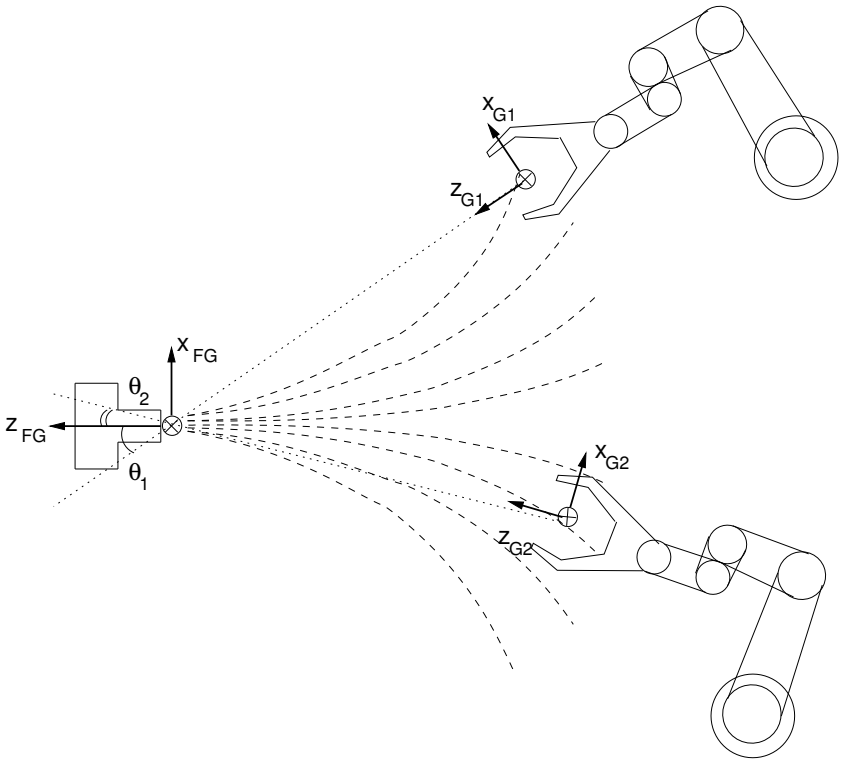


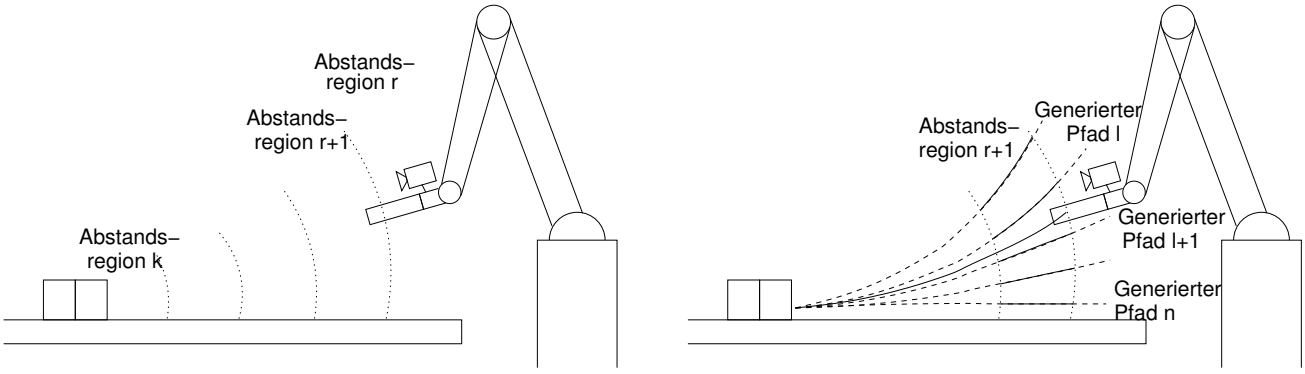
Abbildung 4.4: Die Auswahl des nächsten Pfades (fett gestrichelte Linie) kann über den relativen Winkel der aktuellen Aufnahmeposition zur Zielposition eindeutig bestimmt werden, da zu jeder Aufnahmeposition ein einziger Winkel θ entspricht.

4.1.2 Die Steuerungskomponente der Zielführung

Beide Ebenen sind mit *Radial Basis Function* (RBF) Netzen realisiert, um das Verhalten erlernen zu können. RBF-Netze wurden ausgewählt, weil sie schnell zu trainieren sind und ein Sicherheitsmaß ihrer Klassifikation des Eingangsmerkmalvektors bezüglich der trainierten Merkmalsvektorgruppen erzeugen; je näher die aktuelle Aufnahmeposition zu einem Pfad ist, desto größer wird der Sicherheitsmaß der Klassifikation der Aufnahme zu der Klasse des entsprechenden Pfades. Diese Sicherheitsmaße, die vom RBF in einem Ausgabevektor zusammengefasst werden, sind auch ein Maß für den relativen Abstand der aktuellen Position zu den Pfaden und können als die Gewichtungen für die Interpolation der nächsten Pfade verwendet werden. Weiterhin sind RBF-Netze lokale Approximatoren und erzeugen somit keine Klassifizierung für Aufnahmen, die außerhalb der trainierten Region akquiriert wurden[6].

[6]Eine alternative Möglichkeit wäre der Einsatz von SOM Netzwerken, die auch lokale Approximatoren sind und eine Gewichtung anhand der aktivierten Neuronen der Ausgabeschicht liefern können. Jedoch haben SOMs gegenüber RBF-Netzen den Nachteil, dass sie weit längere Trainingszeiten erfordern.

Die erste Ebene der Steuerungskomponente besteht aus einem RBF-Netzwerk, während die zweite mehrere RBF-Netzwerke beinhält. Für die erste Ebene unterteilt man die mit dem algorithmischen Teach-In generierten Pfade in mehrere Abstandsregionen vom Zielobjekt. Das RBF, das als Eingabe den Vektor $\vec{s}_A$ erhält, schätzt die Abstandsregion anhand der aktuellen Aufnahme der Greiferkamera (Abbildung 4.5a). Aus dem geschätzten Abstand zum Objekt ergibt sich, welches RBF der zweiten Ebene zu aktivieren ist. Jedes RBF der zweiten Ebene wird innerhalb einer einzelnen Abstandsregion trainiert, anhand des Merkmalsvektors $\vec{s}_P$ die Pfade zu erkennen und zu gewichten, die am nächsten zur aktuellen Position des Greifers liegen (Abbildung 4.5b).



(a) (b)

Abbildung 4.5: Trainierte Abstandsregionen und Pfade.

4.1.3 Interpolation der Pfade zum Ziel

Zur Berechnung des Pfades, der den Greifer zum Ziel führt, werden die Ausgabevektoren $\vec{r}_A$ und $\vec{r}_P$ der RBFs verwendet, die den Abstand der aktuellen Position zum Objekt und zu den nächstliegenden Pfaden zum Ausdruck bringen. Zuerst werden zwei Schwellenwerte k_A und k_P auf die Ausgabevektoren angewandt, so dass kleine Werte, die wegen Rauschen vorhanden sind, die Berechnung des Gesamtpfades nicht beeinflussen:

$$\widehat{r}_{Ai} = \begin{cases} r_{Ai} & : \quad \text{falls} \quad r_{Ai} \geq k_A, \\ 0 & : \quad \text{sonst.} \end{cases} \tag{4.3}$$

$$\widehat{r}_{Pi} = \begin{cases} r_{Pi} & : \quad \text{falls} \quad r_{Pi} \geq k_P, \\ 0 & : \quad \text{sonst.} \end{cases} \tag{4.4}$$

Die Stützstellen des Pfades kann man als die gewichtete Summe der entsprechenden Stützstellen der im algorithmischen Teach-In generierten Pfade berechnen[7]:

$$\vec{\mathcal{P}}_{oi} = \frac{\sum_{j=1}^{n} (\widehat{r}_{Pj} \vec{\mathcal{P}}_{ji})}{\sum_{j=1}^{n} \widehat{r}_{Pj}} \tag{4.5}$$

[7]Wegen den Schwellwerten in Gleichungen 4.3 und 4.4 beeinflussen jedoch nur die benachbarten Pfade die Berechnung des Gesamtpfades.

In Gleichung 4.5 entspricht der Parameter n der Anzahl der generierten Pfaden und i nimmt die Werte $1, ..., m$ an, wobei m die Anzahl der Stützstellen der Pfade ist[8]

Die Anfangsposition auf dem Pfad ist auf ähnliche Weise zu berechnen:

$$\vec{\mathcal{P}}_{os} = \frac{\sum\limits_{i=1}^{k}(\widehat{r}_{Ai}\frac{\sum\limits_{j=1,n}(\widehat{r}_{Pj}\vec{\mathcal{P}}_{ji})}{\sum\limits_{j=1,n}\widehat{r}_{Pj}})}{\sum\limits_{i=1}^{k}\widehat{r}_{Ai}} \tag{4.6}$$

In Gleichungen 4.5 und 4.6 sind die Stützstellen relativ zum Zielobjekt angegeben. Damit der Manipulatorarm die Pfade ausführen kann, muss man die Koordinaten der Stützstellen relativ zur Manipulator-Basis transformieren. Seien die Position und die Orientierung des Greifers relativ zum Zielobjekt an der i-ten Stützstelle in der Transformationsmatrix ${}^{O}_{G}\mathbf{T}_{oi}$ zusammengefasst. Da für die Startposition die Transformationsmatrix ${}^{MB}_{Ga}\mathbf{T}$ des Greifers zur Manipulator-Basis bekannt ist[9] und aus Gleichung 4.6 die Matrix des Objektsystems zur aktuellen Greiferposition ${}^{Ga}_{O}\mathbf{T}$ leicht zu berechnen ist, kann man die Stützstellen des Pfades relativ zur Manipulator-Basis wie folgt transformieren:

$${}^{MB}_{G}\mathbf{T}_{oi} = {}^{MB}_{Ga}\mathbf{T}\,{}^{Ga}_{O}\mathbf{T}\,{}^{O}_{G}\mathbf{T}_{oi} \tag{4.7}$$

Dieser Ansatz funktioniert, solange das Zielobjekt sich in der Nähe der Sichtachse der Greiferkamera befindet. Ist dies nicht der Fall, dann sind die Stützstellen des berechneten Pfades in Gleichungen 4.5 und 4.6 fehlerbehaftet, da die im algorithmischen Teach-In aufgenommenen Positionsdaten $\vec{\mathcal{P}}_{ji}$ von einer Ausrichtung der Greiferkamera zum Objekt ausgehen. Um diesen Fehler zu beheben, ist die Orientierung der Kamera an den Stützstellen des Pfades so zu korrigieren, dass während der Bewegung das Zielobjekt in die Nähe der Sichtachse rückt. Dazu benötigt man die internen Kalibrierungsparameter der Kamera. Sind diese aus der Kalibrierung bekannt, dann ist die notwendige Drehung der Kamera, damit das Objekt ins Bildzentrum (u_{0u}, v_{0u}) rückt:

$$\theta_K = (u_S - u_{0u})atan2(1, fa_u) \tag{4.8}$$

$$\phi_K = (v_S - v_{0u})atan2(1, fa_v) \tag{4.9}$$

wobei die Winkel θ_K, ϕ_K die Drehung um die x- bzw. y-Achse der Kamera angeben. Die angepasste Transformationsmatrix der Stützstellen wird anschließend aus folgender Gleichung berechnet:

$${}^{MB}_{Gh}\mathbf{T}_{oi} = ({}^{G}_{K}\mathbf{T}\,{}^{Kh}_{Ka}\mathbf{T}\,{}^{K}_{G}\mathbf{T}\,{}^{G}_{MB}\mathbf{T}_{oi})^{T} \tag{4.10}$$

wobei ${}^{Ka}_{Kh}\mathbf{T} = {}^{Kh}_{Ka}\mathbf{T}^{T}$ das Koordinatensystem der Kamera mit dem Objekt im Hauptpunkt relativ zur aktuellen Kameralage angibt:

$${}^{Ka}_{Kh}\mathbf{T} = \begin{pmatrix} \cos(\phi_K) & \sin(\phi_K)\sin(\theta_K) & \sin(\phi_K)\cos(\theta_K) & 0 \\ 0 & \cos(\theta_K) & -\sin(\theta_k) & 0 \\ -\sin(\phi_K) & \cos(\phi_K)\sin(\theta_K) & \cos(\phi_K)\cos(\theta_K) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

[8] Der Parameter m kann gleich der Anzahl der Abstandsregionen k gewählt werden.

[9] Die Startposition entspricht der aktuellen Greiferposition, die aus der Manipulatorkinematik bekannt ist.

4.1.4 Erlernen der generierten Pfade

Für das Erlernen des Verhaltens werden alle beim algorithmischen Teach-In generierten Pfade mit einer immer gleichen Anzahl von Abstandsregionen aufgeteilt (Abbildung 4.5). In einer Abstandsregion wird der virtuelle Manipulator auf jedem Pfad platziert und die Greiferkamera akquiriert eine Aufnahme des Zielobjektes. Die Bildverarbeitungskomponente extrahiert anschließend aus jeder Aufnahme den Merkmalsvektor $\vec{s}_{Pj}$[10]. Alle Eingangs-Ausgangspaare $(\vec{s}_{Pj}, j)$ aus derselben Abstandsregion werden für das Training des entsprechenden RBF-Netzwerkes verwendet, das für diese Abstandsregion spezialisiert ist. Diese Prozedur wiederholt sich für jede Abstandsregion und somit für jedes RBF der zweiten Ebene, bis die Region um das Objekt, bei der ein Greifvorgang noch möglich ist, abgedeckt wird.

Für das RBF der ersten Ebene kommen alle akquirierten Bilder, gruppiert nach ihrem Abstand vom Objekt, zum Einsatz. Sei mit i die Distanzregion relativ zum Objekt notiert, aus der die Aufnahmen gemacht wurden, dann bestehen die Trainingsdaten aus $(\vec{s}_{A1i}, ..., \vec{s}_{Aji}, ..., \vec{s}_{Ani}, i)$, wobei der Index j den Pfad angibt, auf dem das Kamerabild aufgenommen wurde.

4.1.5 Resultate des Trainings und Evaluierung der Zielführung

Das Training des Verhaltens findet in der virtuellen Umgebung ohne Hindernisse statt (Abbildung 4.6). Für die eingesetzte Zielführung generiert das algorithmische Teach-In 80 Pfade, indem der Parameter γ der Gleichung 3.20 des einschaligen kreisförmigen Hyperboloiden 8 unterschiedliche Werte annimmt und für die Ebene (Gleichung 3.21) 10 Rotationen um die z-Achse definiert werden. Um einen großen Arbeitsbereich des Manipulators zu berücksichtigen, werden 25 Abstandsregionen bis zu einer Distanz von 65 cm vom Zielobjekt definiert. Jede Distanzregion hat ein Breite von 2.5 cm.

(a) (b)

Abbildung 4.6: Sicht während des Trainings und der Evaluierung der bildgestützten Zielführung in der virtuellen Umgebung.

Die Evaluierung des Verhaltens findet anschließend in der virtuellen Umgebung ohne Rückkopplung statt; in diesem Fall akquiriert die Greiferkamera eine einzelne Aufnahme des Zielobjektes und anhand dieser versucht die Steuerungskomponente den Roboterarm zur Zielposition zu führen. Das Objekt ist innerhalb des trainierten Arbeitbereiches des Manipulators an sechs unterschiedliche, nicht

[10]Der Index j gibt den Pfad an, auf dem das Kamerabild aufgenommen wurde.

trainierte Positionen platziert. Außerdem werden drei Positionen außerhalb des trainierten Bereiches getestet; in diesem Fall ist ein Greifvorgang unerwünscht. Bei jeder Position nimmt das Objekt drei unterschiedliche Orientierungen um seine y-Achse an. Die Ergebnisse der Zielführung für Objektpositionen innerhalb des trainierten Arbeitsbereiches sind in Tabelle 4.1 zusammengefasst.

		Trainierter Bereich
Objektpositionen		18
Erste Ebene	Richtige Abstandsregion	17
	Benachbarte Abstandsregion	1
	Falsche Abstandsregion	0
Zweite Ebene	Nächster Pfad	16
	Zweitnächster Pfad	2
	Falsche Pfadauswahl	0
Gesamtpfad	Zielposition erreicht	12
	Zielposition $\leq$ 3cm	6
	Zielposition $\geq$ 3cm	0

Tabelle 4.1: Ergebnisse der Zielführung in der virtuellen Umgebung für verschiedene, nicht trainierte Positionen des Objektes innerhalb des trainierten Arbeitbereiches des Manipulators. Sechs Objektpositionen mit drei unterschiedliche Orientierungen pro Position wurden getestet.

Während der Evaluierung ermittelt die Steuerungskomponente in 17 Fällen die richtige Abstandsregion relativ zum Zielobjekt und 1 mal die nächste. Die zweite Ebene ergibt in 16 Fällen die nächste Trajektorie und in 2 Fällen die zweitnächste. Der interpolierte Pfad bringt den Greifer in 12 der Fälle zur richtigen Zielposition und in 6 der Fälle ist eine zusätzliche, feine Bewegung von 2 cm notwendig, um das Greifen zu vervollständigen. Alle Positionen außerhalb des trainierten Bereiches werden richtig erkannt; für diese Positionen initiiert der Roboterarm keinen Greifvorgang.

Die Fehler bei der Ausführung der Pfade in der virtuellen Umgebung sind auf Fälle zurückzuführen, wo das Zielobjekt nicht in der Nähe des Bildhauptpunktes war. Je weiter die Projektion des Objektes vom Bildhauptpunkt entfernt ist, desto unterschiedlicher erscheint das Objekt in der Aufnahme verglichen mit dem trainierten virtuellen Prototyp. Dieser Effekt ist umso größer, je weiter entfernt sich das Objekt von der Kamera befindet; dann verursacht ein kleiner Abstand vom Bildhauptpunkt eine verhältnismäßig große Veränderung des Flächeninhaltes der Seiten des roten Griffes in der Aufnahme. So entstehen bei den RBFs der ersten und zweiten Ebene falsche Gewichtungen bei der Klassifizierung der nächsten Pfade und somit ein ungenauer Gesamtpfad zur Zielposition. Deswegen ist die Zuverlässigkeit des Verhaltens niedrig, wenn das Ziel in der Nähe des Bildrandes zu sehen ist. Dieses Problem wird jedoch durch die Anwendung des Verhaltens in einem geschlossenen Regelkreis und die Korrektur der Kameraorientierung nach Abschnitt 4.1.3 aufgehoben[11].

Dieselbe Evaluierung wird mit dem realen Roboterarm ausgeführt (Tabelle 4.2). Während der Tests wählt die erste Ebene in 16 Fällen die richtige und 2 mal die zweitnächste Abstandsregion aus. Die RBFs der zweiten Ebene ermitteln für 11 der Ansichten den nächsten Pfad und in 4 Fällen den zweitnächsten. Drei falsche Klassifizierungen werden durch die Bildreflektionen und das Rauschen in den Aufnahmen verursacht, die eine fehlerhafte Segmentierung und Merkmalsextraktion zur Folge haben.

[11] Alternativ könnte man die Trainingsmenge mit Aufnahmen des Objekt am Rande des Kamerabildes ergänzen. Dies würde jedoch die Trainingsdaten um ein Vierfaches vergrößern.

		Trainierter Bereich
Objektpositionen		18
Erste Ebene	Richtige Abstandsregion	16
	Benachbarte Abstandsregion	2
	Falsche Abstandsregion	0
Zweite Ebene	Nächster Pfad	11
	Zweitnächster Pfad	4
	Falsche Pfadauswahl	3
Gesamtpfad	Zielposition erreicht	10
	Zielposition $\leq$ 3cm	5
	Zielposition $\geq$ 3cm	3

Tabelle 4.2: Ergebnisse der Zielführung mit dem realen mobilen Manipulator für nicht trainierte Positionen des Objektes innerhalb des trainierten Arbeitbereiches des Roboterarmes. Die evaluierten Objektpositionen stimmen mit den entsprechenden Positionen in der virtuellen Umgebung überein (siehe auch Tabelle 4.1).

4.2 Hindernisvermeidung

Die Zielführung kann zwar den Manipulator entlang eines objektspezifischen Pfades zur Greifposition am Zielobjekt führen, sie kann ihn jedoch nicht vor Kollisionen schützen. Diese Aufgabe übernehmen zwei Verhalten zur reaktiven Hindernisvermeidung. Das erste bestimmt eine hindernisvermeidende Bewegung für den Greifer des Manipulators während das zweite die restlichen Segmente des Roboterarms vor Kollisionen mit Hindernissen schützt. Diese Aufteilung in zwei Komponenten hat als Ziel, die Hindernisvermeidung zu vereinfachen und die Rechenzeit niedrig zu halten, da jedes Verhalten nur eine beschränkte Anzahl der Gelenke des Manipulators berücksichtigen muss. Außerdem können die Daten der Greifer- und der Bordkamera gezielt und effizient für jedes Verhalten verwendet werden; die Greiferkamera überwacht die Bewegung des Greifers, während die Bordkamera, die den kompletten Roboterarm erfasst, die restlichen Segmente vor Kollisionen schützt. Alternativ wäre eine rechenaufwendige Fusion der extrahierten Merkmale aus beiden Kameras notwendig.

Beide hindernisvermeidende Verhalten verfügen über eine ähnliche Merkmalsextraktionsphase (Abbildung 4.7). In einem ersten Schritt versuchen sie die Hindernisse und das Zielobjekt im Raum relativ zum Manipulator zu lokalisieren. Dieser Vorgang, vorgestellt im nächsten Abschnitt, ist für beide Fertigkeiten derselbe; nur die eingesetzte Kamera unterscheidet sich bei jedem Verhalten. Diese geometrische Information wird dann von der Steuerungskomponente der Fertigkeiten, vorgestellt in Abschnitten 4.2.2 und 4.2.3, verwendet, um einen kollisionsfreien Pfad für den Greifer bzw. für die Segmente des Manipulators zu bestimmen.

4.2.1 Lokalisierung der Hindernisse im Raum

Bei der Hindernisvermeidung hat die Bildverarbeitungskomponente der Verhalten die Aufgabe, aus den Kameraaufnahmen die Hindernisse im Raum zu erkennen und relativ zum Manipulator zu lokalisieren. Aus dem Abstand und der Orientierung der Hindernisse zum Roboterarm kann die Steuerungskomponente der Verhalten ein Ausweichmanöver berechnen. Da jedoch ein einzelnes Bild keine Tie-

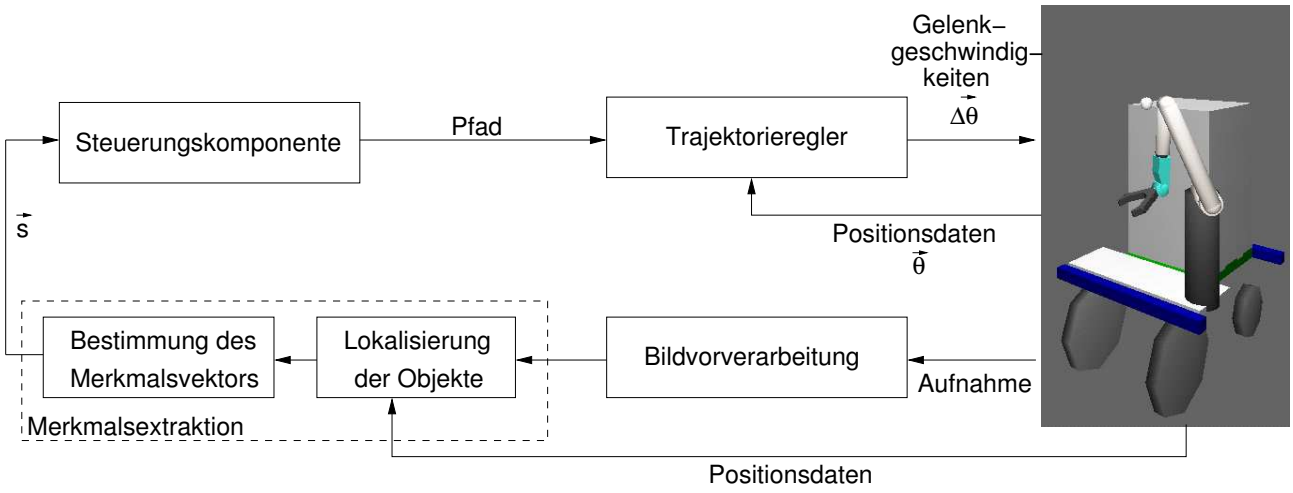


Abbildung 4.7: Allgemeiner Ablauf der hindernisvermeidenden Verhalten.

feninformation liefert, ist eine Bewegung der Kamera notwendig, um die Objekte zu lokalisieren[12]. So werden zwei Aufnahmen der Szene aus unterschiedlichen, bekannten Kamerapositionen akquiriert. Aus korrespondierenden Punkten in den Aufnahmen und der bekannten Aufnahmegeometrie kann man die 3D-Struktur der Szene ermitteln [Fau93].

Nach der Bildakquisition werden die Objekte in einer Vorverarbeitungsphase in beiden Kamerabilder identifiziert und anhand ihrer Farbe aus dem Hintergrund segmentiert (Abbildung 4.8). Danach detektiert der *Susan Corners* Algorithmus [SB97a] Punkte an den Kanten der Objekte. Dabei liefert er eine große Anzahl von Kantenpunkten zurück, die bei der nachfolgenden Lokalisierung zu erhöhten Berechnungszeit führen kann. Daher werden mit dem *Recursive Line Fitting* Algorithmus, beschrieben in [Cro85], die Eckpunkte der Objekte detektiert und die restlichen Punkte ausgefiltert (Abbildung 4.9).

Sei M ein charakteristischer Punkt eines Objektes im Raum und M_{K_1}, M_{K_2} seine Projektionen auf den Bildern aus der ersten und zweiten Aufnahmeposition. Nach der Abbildungvorschrift der Kameras (Gleichung 3.4) gilt für die homogenen Pixelkoordinaten ${}^{K_1}\vec{U}$ und ${}^{K_2}\vec{U}$ von M_{K_1}, M_{K_2}:

$$\begin{aligned} {}^{K_1}\vec{U} &= \mathbf{K}\,\mathbf{M}\,{}^{K_1}_{W}\mathbf{T}\,{}^{W}\vec{P} \\ {}^{K_2}\vec{U} &= \mathbf{K}\,\mathbf{M}\,{}^{K_2}_{W}\mathbf{T}\,{}^{W}\vec{P} \end{aligned} \tag{4.12}$$

wobei ${}^{W}\vec{P}$ der Vektor der homogenen kartesischen Koordinaten von M im Weltkoordinatensystem ist, $\mathbf{K}$ die Matrix der intrinsischen Kameraparameter und ${}^{K_1}_{W}\mathbf{T}$, ${}^{K_2}_{W}\mathbf{T}$ die Transformationsmatrizen des Weltkoordinatensystem bezüglich der ersten bzw. der zweiten Aufnahmeposition. Hat man das System kalibriert und ist zugleich die Lageänderung der Kamera bekannt, dann kann man die Terme $\mathbf{K}\,\mathbf{M}\,{}^{K_1}_{W}\mathbf{T}$ und $\mathbf{K}\,\mathbf{M}\,{}^{K_2}_{W}\mathbf{T}$ berechnen. Um dann ${}^{W}\vec{P}$ aus den Aufnahmen zu bestimmen und dadurch M im Raum zu lokalisieren, muss man aus der Menge der extrahierten Eckpunkte aus beiden Aufnahmen die eigentlichen Projektionen von M finden, also M_{K_1} und M_{K_2} (Korrespondenzsuche). Anhand der homogenen Pixelkoordinaten ${}^{K_1}\vec{U}$ und ${}^{K_2}\vec{U}$ von M_{K_1} und M_{K_2} kann man anschließend Gleichungssystem 4.12 nach ${}^{W}\vec{P}$ auflösen.

[12] Alternativ kann man mit einem Modell eines Hindernisses seine Lage im Raum bestimmen. Dies würde jedoch die Hindernisvermeidung nur auf Hindernisse begrenzen, für die ein Modell dem mobilen Manipulator vorliegt. Der Manipulator könnte dann keinen Hindernissen mit unbekannten Dimensionen ausweichen.

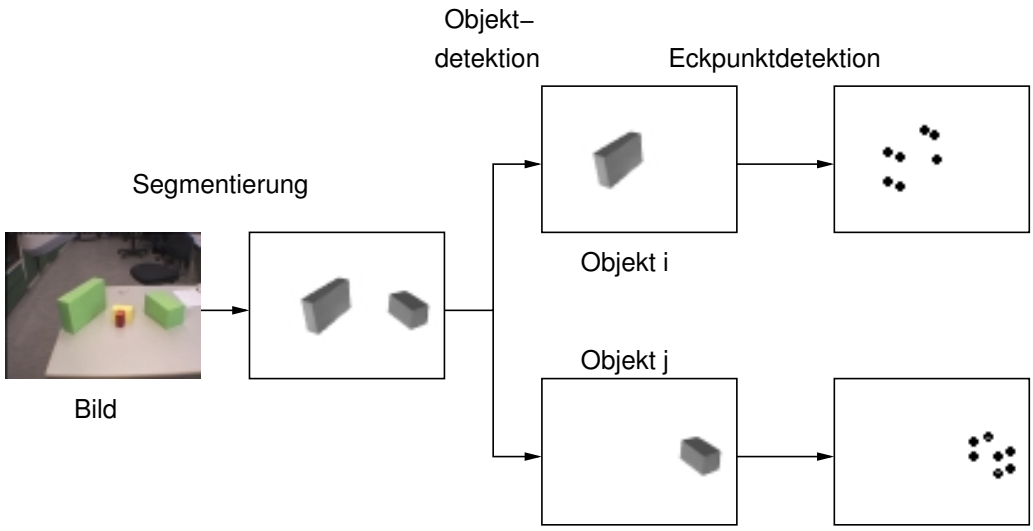


Abbildung 4.8: Ablauf der Bildverarbeitung bei der Hindernisvermeidung.

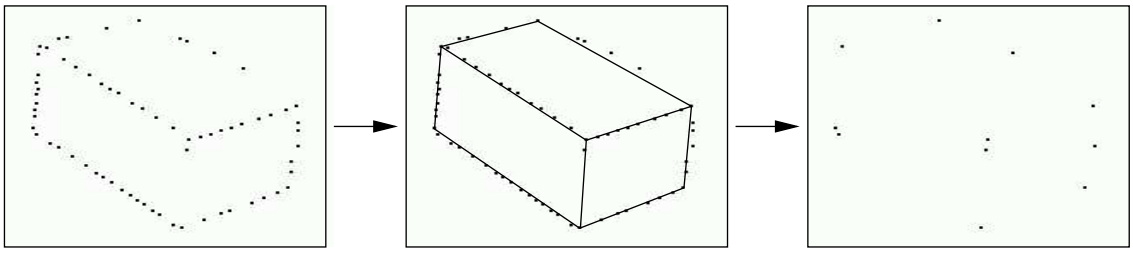


Abbildung 4.9: *Recursive Line Fitting* Algorithmus zum Filtern der Eckpunkte.

Bei der Korrespondenzsuche liefert die epipolare Geometrie eine wirkungsvolle Einschränkung in Bezug auf die mögliche Lage der Abbildung eines Raumpunktes M auf den Bildebenen: M_{K_2} muss auf der zu M_{K_1} zugehörigen epipolaren Linie $\epsilon_{K_1\vec{U}}$ in der zweiten Aufnahme liegen [Fau93] (Abbildung 4.10).

Die epipolare Linie im zweiten Kamerabild, die dem Pixel M_{K_1} mit homogenen Pixelkoordinaten ${}^{K_1}\vec{U}$ in der ersten Aufnahme entspricht, ist folgendermaßen definiert:

$$\epsilon_{K_1\vec{U}} : au_{K_2} + bv_{K_2} + c = 0 \tag{4.13}$$

mit

$$(a, b, c)^T = {}^{K_1}\vec{U}^T\mathbf{F} \tag{4.14}$$

Dabei ist $\mathbf{F}$ die so genannte Fundamentalmatrix; sie ist ausschließlich von den internen und externen Kameraparametern abhängig[13].

Der Abstand eines Bildpunktes ${}^{K_2}\vec{U} = (u_{K_2}, v_{K_2}, 1)^T$ aus der zweiten Aufnahme zur epipolaren Linie beträgt

$$d(\epsilon_{K_1\vec{U}},\ {}^{K_2}\vec{U}) = |\frac{a\,u_{K_2} + bv_{K_2} + c}{\sqrt{a^2 + b^2}}| \tag{4.15}$$

[13]Eine kurze Einführung in die epipolare Geometrie und die Berechnung der Fundamentalmatrix kann man in Anhang C.2 finden.

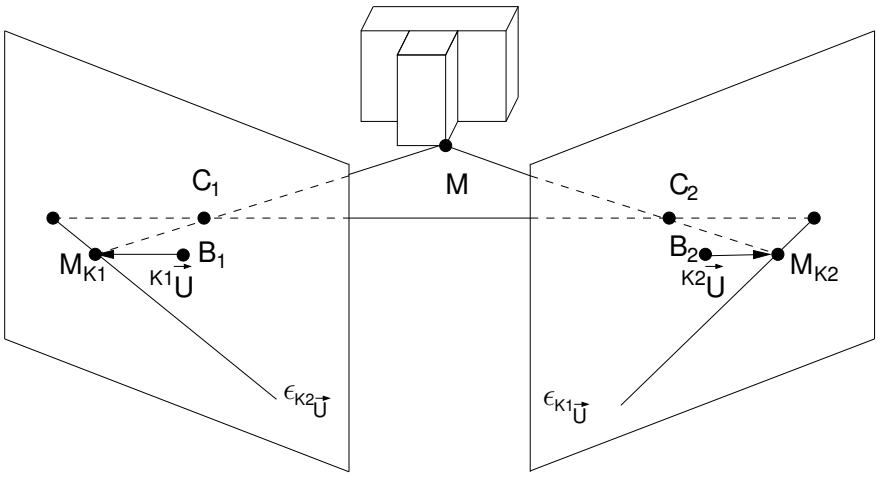


Abbildung 4.10: Epipolare Geometrie einer Szene. C_1, C_2 sind die Brennpunkte der Kameras und B_1, B_2 die entsprechenden Bildhauptpunkte.

Für zwei korrespondierende Bildpunkte sollte dann $d(\epsilon_{K_1\vec{U}},\ {}^{K_2}\vec{U}) = 0$ sein. In der Praxis ist jedoch der Abstand wegen Berechnungsungenauigkeiten und Rauschen klein, aber nicht gleich Null. Deswegen muss die Korrespondenzsuche in einer Region um die epipolare Linie stattfinden. Alle Eckpunkte, die in dieser Region liegen, sind Kandidaten für M_{K_2}. Um die richtige Korrespondenz zu ermitteln, werden zusätzlich die Intensitätswerte der Nachbarpixel berücksichtigt. Dabei wird die Intensität des Eckpunktes, sowie die minimale und die maximale Intensität der acht Nachbarpunkte mit den entsprechenden Werten von M_{K_1} verglichen. Zusätzlich darf die Verschiebung im Bild nicht größer als ein Schwellenwert sein. Der Eckpunkt mit der größten Ähnlichkeit zu M_{K_1} wird daraufhin als M_{K_2} ausgewählt.

Sobald die Eckpunkte in beiden Aufnahmen korrespondiert sind, kann man ihre 3D-Koordinaten berechnen. Sei die 3×4 Matrix $\mathbf{K}\,\mathbf{M}\,{}^{K_1}_{W}\mathbf{T}$ aus dem Gleichungssystem 4.12 mit vier 3×1 Vektoren ${}^{K1}\vec{t}_i$ zusammengefasst:

$$\mathbf{K}\,\mathbf{M}\,{}^{K_1}_{W}\mathbf{T} = [{}^{K1}\vec{t}_1\ {}^{K1}\vec{t}_2\ {}^{K1}\vec{t}_3\ {}^{K1}\vec{t}_4] \tag{4.16}$$

und ähnlich für $\mathbf{K}\mathbf{M}\,{}^{K_2}_{W}\mathbf{T}$. Dann kann Gleichungssystem 4.12 wie folgt transformiert werden [Aya91]:

$$\begin{aligned}
({}^{K1}\vec{t}_1 - u_{K_1}\,{}^{K1}\vec{t}_3)^T\,{}^{W}\vec{p} &= u_{K_1}\,{}^{K1}t_{34} - {}^{K1}t_{14}\\
({}^{K1}\vec{t}_2 - v_{K_1}\,{}^{K1}\vec{t}_3)^T\,{}^{W}\vec{p} &= v_{K_1}\,{}^{K1}t_{34} - {}^{K1}t_{24}\\
({}^{K2}\vec{t}_1 - u_{K_2}\,{}^{K1}\vec{t}_3)^T\,{}^{W}\vec{p} &= u_{K_2}\,{}^{K2}t_{34} - {}^{K2}t_{14}\\
({}^{K2}\vec{t}_2 - v_{K_2}\,{}^{K1}\vec{t}_3)^T\,{}^{W}\vec{p} &= v_{K_2}\,{}^{K2}t_{34} - {}^{K2}t_{24}
\end{aligned} \tag{4.17}$$

wobei ${}^{K1}t_{i4}$ und ${}^{K2}t_{i4}$ die Elemente von Vektoren ${}^{K1}\vec{t}_4$ und ${}^{K2}\vec{t}_4$ sind und ${}^{W}\vec{p}$ den Positionsvektor des Raumpunktes M im Weltkoordinatensystem darstellt. Hier handelt es sich um ein lineares homogenes Gleichungssystem mit vier Gleichungen und drei Unbekannten, nämlich die Elemente von ${}^{W}\vec{p}$. In Matrizenform kann Gleichungssystem 4.17 folgenderweise zusammengefasst werden:

$$\mathbf{A}\,{}^{W}\vec{p} = \vec{b} \tag{4.18}$$

und somit sind die Koordinaten des Raumpunktes M:

$${}^{W}\vec{p} = \mathbf{A}^{+}\vec{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b} \tag{4.19}$$

Wählt man dabei das Weltkoordinatensystem $\{W\}$ als das Kamerasystem bei der zweiten Aufnahme aus, dann ergibt Gleichung 4.19 bei bekannter Lageänderung der Kamera die Koordinaten der Raumpunkte relativ zur zweiten, aktuellen Kameraposition.

Aus den berechneten 3D-Koordinaten der Raumpunkte werden anschließend die drei von der Kameraposition aus sichtbaren Oberflächen der Hindernisse rekonstruiert und ihre Schwerpunkte sowie der Schwerpunkt des Hindernisses berechnet. Für die Lokalisierung der Auflageoberfläche sind drei Punkte notwendig, die zu dieser Oberfläche gehören. Da alle Hindernisse auf der Auflageoberfläche positioniert sind, kann man drei der gefunden Raumpunkte der Hindernisse mit der kleinsten z-Koordinate wählen; diese müssen dann auf der Auflageoberfläche liegen. Durch diese Punkte wird die Ebenengleichung der Auflageoberfläche aufgestellt.

Da die Lokalisierung der Hindernisse von Gleichungssystem 4.17 und daher von der Änderung der Kameralage abhängig ist, muss diese genau bekannt sein, um die Objekte robust im Raum zu lokalisieren. Insbesondere bei kleiner Lageänderung der Kamera haben Positionsungenauigkeiten einen großen Einfuß bei den Berechnungen. Experimente haben gezeigt, dass eine Verschiebung der Kamera um mindestens 6 cm stattfinden muss, um die Objekte robust zu lokalisieren. Die besten Ergebnisse sind mit einer Kameraverschiebung von 10 cm erzielt worden[14]; in diesem Fall lag der Lokalisierungsfehler im Bereich von 3 cm für die Greiferkamera und 2 cm für die Bordkamera[15]. Während dieser Fehler für die Zielführung sich als zu groß erwiesen hat, ist er bei den hindernisvermeidenden Verhalten, die keine große Genauigkeit erfordern, um eine Ausweichbewegung zu berechnen, akzeptabel.

4.2.2 Hindernisvermeidung des Greifers

Die Hindernisvermeidung des Greifers ist eines der beiden in dieser Arbeit entwickelten hindernisvermeidenden Verhalten, die den Manipulator vor Kollisionen schützen sollen. Das Verhalten verwendet die Bilder der Greiferkamera, um einen auf den Greifer bezogenen Ausweichpfad zu erstellen. Der Ablauf des Verhaltens ist in Abbildung 4.11 dargestellt. Die Merkmalsextraktion hat die Aufgabe, die Objekte im Raum nach Abschnitt 4.2.1 zu lokalisieren und anschließend die für die Steuerungskomponente erforderlichen Merkmale zu extrahieren. Die Steuerungskomponente, die aus einem lernfähigen Neurofuzzy-System besteht und in der virtuellen Umgebung erlernt wird, erzeugt für jedes Hindernis einen Ausweichvektor. Die sich ergebenden Vektoren werden anschließend interpoliert und in einem Pfad für den Manipulator umgewandelt.

Merkmalsextraktion

Die Merkmalsextraktion verläuft nach dem in Abschnitt 4.2.1 beschriebenen Ansatz. Nach dem dort vorgestellten Verfahren ist für die Objektlokalisierung die Kenntnis der Lageänderung der Kamera notwendig, um Gleichungssystem 4.17 lösen zu können[16]. Sei ${}^{MB}_{G_1}T$ die aus der Kinematik des Mani-

[14]Eine größere Kameraverschiebung hatte als Resultat, dass die Abbildung der Szene in den beiden Aufnahmen große Unterschiede aufweist. In diesem Fall ergeben sich bei der Korrespondenzsuche falsche Korrespondenzen zwischen den extrahierten Eckpunkten und dadurch Fehler bei der Lokalisierung der Hindernisse im Raum.

[15]Zur weiteren Reduzierung des Fehlers ist eine größere Genauigkeit bei der Positionierung der Kameras notwendig. Dies wird jedoch in dieser Arbeit durch die verwendete Hardware begrenzt. Alternativ könnten Verfahren aus der 3D Rekonstruktion zur Schätzung der Kamerapositionen aus den Aufnahmen bzw. der Einsatz einer festen Stereokamera den Fehler weiter reduzieren.

[16]Dabei werden die Aufnahmen vor und nach der letzten Greiferbewegung verwendet.

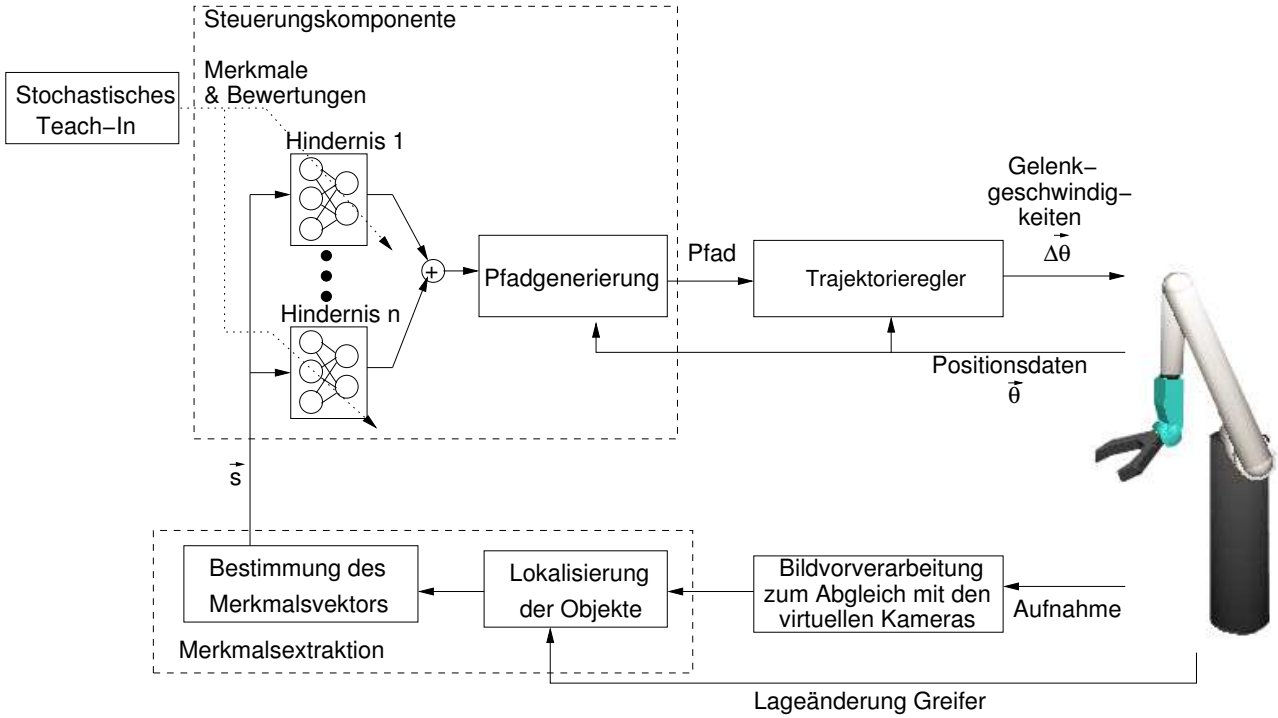


Abbildung 4.11: Ablauf der Hindernisvermeidung des Greifers.

pulators bekannte Lage des Greifers zur Manipulator-Basis bei der ersten Aufnahme und ${}^{MB}_{G_2}T$ bei der zweiten. Aus der Kalibrierung der Kameras (vergleiche Abschnitt 3.2.1) ist auch die Position der Kamera zum Greifer ${}^{G}_{K}T$ bekannt und bei beiden Aufnahmepositionen konstant, also ${}^{G}_{K}\mathbf{T} = {}^{G_1}_{K_1}\mathbf{T} = {}^{G_2}_{K_2}\mathbf{T}$. Dann ist die gesuchte Lageänderung des Kamera ${}^{K_1}_{K_2}T$:

$$ {}^{K_1}_{K_2}\mathbf{T} = {}^{G}_{K}\mathbf{T}^T \, {}^{MB}_{G_1}\mathbf{T}^T \, {}^{MB}_{G_2}\mathbf{T}^T \, {}^{G}_{K}\mathbf{T}^T \tag{4.20} $$

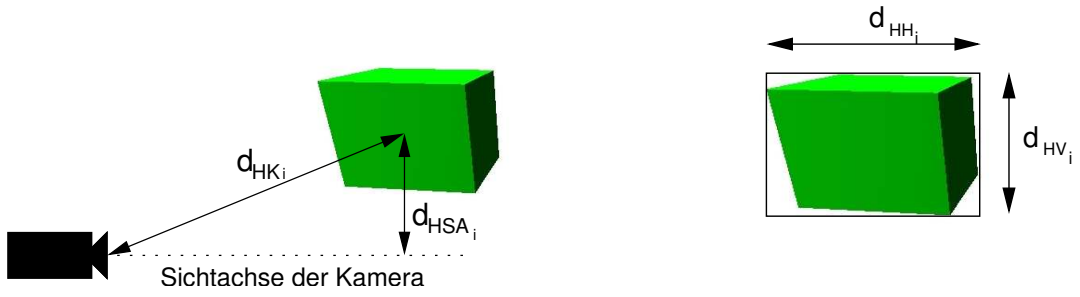
Somit kann man aus Gleichungssystem 4.17 die Objekte im Raum lokalisieren. Aus der berechneten Hindernislage sowie aus den im Bild segmentierten Objekten kann man anschließend die Eingangsmerkmale der Steuerungskomponente pro Hindernis berechnen (Abbildung 4.12):

- den Abstand d_{HK_i} des Objektschwerpunktes des Hindernisses von der Kamera,
- den Abstand d_{HSA_i} des Objektschwerpunktes von der Sichtachse der Kamera,
- die horizontale Größe d_{HH_i} des umgebenden Parallelogramms des Objektes im Bild, und
- die vertikale Größe d_{HV_i} des umgebenden Parallelogramms.

Diese werden im Merkmalsvektor $\vec{s}_{HG_i} = (d_{HK_i}, d_{HSA_i}, d_{HH_i}, d_{HV_i})^T$ zusammengefasst.

Steuerungskomponente der Hindernisvermeidung des Greifers

Die Steuerungskomponente des Verhaltens erzeugt die hindernisvermeidende Bewegung des Greifers. Dafür berechnet sie pro Hindernis einen Ausweichvektor und interpoliert anschließend alle Ausweichvektoren zu einem Bewegungsvektor. Der Ursprung des Bewegungsvektors fällt mit dem Ursprung des Koordinatensystems der Kamera zusammen und wird in diesem Referenzsystem beschrieben.



(a) (b)

Abbildung 4.12: Eingänge der Steuerungskomponente für die Hindernisvermeidung des Greifers.

Die übliche Verfahren zur Hindernisvermeidung gehen von einer Übersicht der Szene mit einer *eye-to-hand* Kamerakonfiguration aus; deshalb sind sie für eine Implementierung mit einer *eye-in-hand* Kamera ungeeignet. Da ein Mensch anhand der aktuellen Aufnahme der Greiferkamera eine robuste hindernisvermeidende Bewegungsstrategie mit einer Menge von qualitativen Regeln beschreiben kann, bietet sich hier der Einsatz von Fuzzy Logik [Zad73] an. Mit Fuzzy Logik kann man aus einem Eingangsvektor anhand qualitativer Regeln entsprechende quantitative Ausgangsgrößen ermitteln. Um das aufwendige Experimentieren zur Parameterbestimmung zu umgehen, wird hier ein lernfähiges Neurofuzzy-System angewandt[17].

Das eingesetzte Neurofuzzy-System besteht aus fünf Neuronenschichten (Abbildung 4.13). Die erste Neuronenschicht propagieren lediglich die Eingangswerte ins Netz. Für die Zugehörigkeitsfunktionen der Eingangsvariablen, die als Aktivierungsfunktionen der zweiten Schicht des Neurofuzzy-Systems implementiert sind, werden Gaußfunktionen verwendet. Die dritte und die vierte Zwischenschicht implementieren die Regelbasis und den Inferenzmechanismus des Fuzzy Logik Systems, während die Gewichtungen zwischen vierter und fünfter Schicht die Defuzzifizierung realisieren. In der Trainingsphase werden nur die Gewichtungen zwischen der vierten und der fünften Schicht und somit die Abszissen der Zugehörigkeitsfunktionen der linguistischen Ausgangsvariablen[18] angepasst.

Das Neurofuzzy-System verfügt über vier Eingangsneuronen, die als Eingabe die Elemente des Merkmalsvektors $\vec{s}_{HG_i}$ erhalten. Dabei werden für d_{HK_i}, d_{HSA_i} jeweils vier und für d_{HH_i}, d_{HV_i} drei linguistische Terme definiert (Anhang D). Für die Bestimmung des Ausweichvektors $\vec{\rho}_i$ sind dessen Betrag und dessen Richtung, dargestellt im Koordinatensystem der Kamera, notwendig. Verwendet man für die Darstellung von $\vec{\rho}_i$ Polarkoordinaten, dann sind folgende drei Größen nötig, um den Vektor zu beschreiben:

- das Azimut ϑ_i, d.h der Winkel zwischen der x-Achse des Kamerasystems und der auf der Bildebene projizierten Komponente des Ausweichvektors,
- die Elevation φ_i, d.h. der Winkel zwischen der Sichtachse bzw. z-Achse der Kamera und dem Ausweichvektor, und
- der Betrag $\mid \vec{\rho}_i \mid$ des Vektors, der als die Geschwindigkeit der Bewegung zu verstehen ist.

[17] Fuzzy Logik und Neurofuzzy werden im Anhang C präsentiert.

[18] In dieser Arbeit kommen Singletons für die Zugehörigkeitsfunktionen der linguistischen Ausgangsvariablen zum Einsatz.

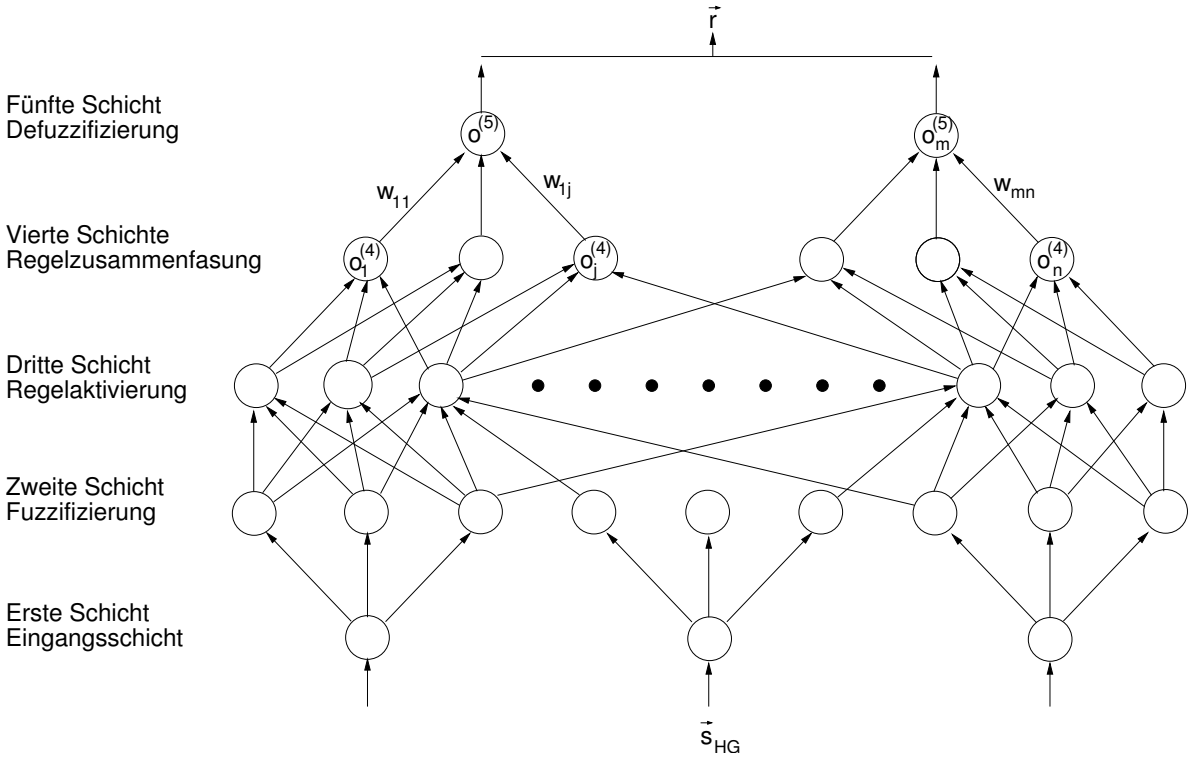


Abbildung 4.13: Fuzzy Logik System realsiert mit einem künstlichen neuronalen Netz.

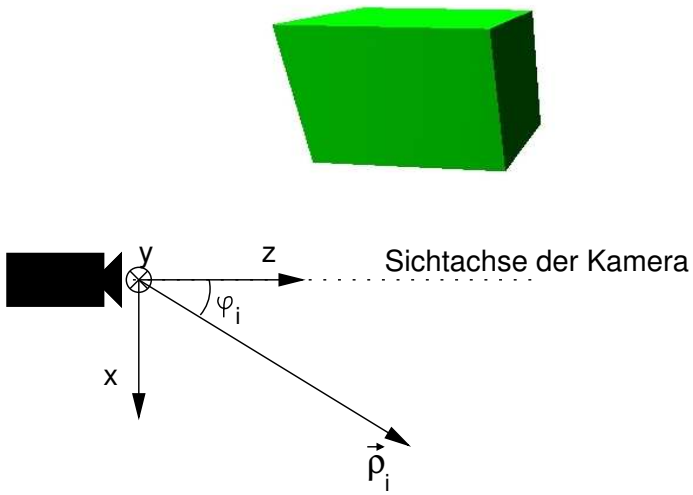


Abbildung 4.14: Ausgabewerte der Steuerungskomponente.

Das Neurofuzzy-System wird trainiert, als Ausgabe den Betrag und Winkel φ_i des Ausweichvektors zur Sichtachse der Kamera, dargestellt in Abbildung 4.14, zu liefern. Der Winkel ϑ_i wird aus der Position des Hindernisschwerpunktes im Bild relativ zum Bildhauptpunkt ermittelt (Abbildung 4.15). Sei (u_{H_i}, v_{H_i}) die Pixelposition des Schwerpunktes und (u_0, v_0) die Pixelkoordinaten des Hauptpunktes, dann ist:

$$\vartheta_i = atan2(u_0 - u_{H_i}, v_0 - v_{H_i}) \tag{4.21}$$

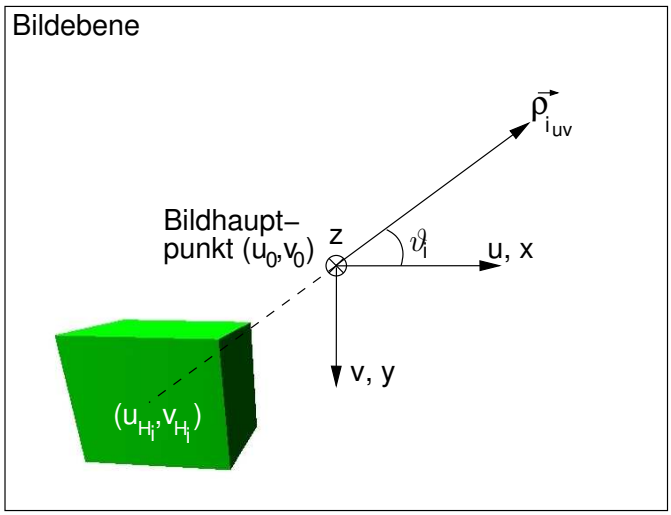


Abbildung 4.15: Berechnung des Winkels ϑ des Ausweichvektors mit der x-Achse der Kamera bzw. der u-Achse des Bildes.

Pfadgenerierung

Die Steuerungskomponente berechnet einen Ausweichvektor pro Hindernis. Der resultierende Bewegungsvektor des Verhaltens wird anschließend durch lineare Kombination der einzelnen Ausweichvektoren bestimmt (Abbildung 4.16). Dabei erhalten die Ausweichvektoren anhand des Abstandes des zugehörigen Hindernisses zur Kamera eine entsprechende Gewichtung. Seien n Hindernisse vorhanden und sei ${}^{K}\vec{\rho}_i$ der Ausweichvektor für das i-te Hindernis, d_i der Abstand der Kamera vom i-ten Hindernis und d_{min} der kleinste dieser Abstände. Dann lässt sich der Bewegungsvektor der Hindernisvermeidung des Greifers ${}^{K}\vec{\rho}_{HG}$ wie folgt berechnen:

$$ {}^{K}\vec{\rho}_{HG} = \sum_{i=1}^{n} \frac{d_{min}}{d_i} \, {}^{K}\vec{\rho}_i \tag{4.22} $$

Orientierung und Betrag des Vektors stellen dabei die von der Hindernisvermeidung vorgeschlagene Richtung und Geschwindigkeit der nächsten Bewegung des Greifers dar. Da der Bewegungsvektor relativ zur Kamera definiert ist, muss er in das Koordinatensystem der Manipulator-Basis transformieren werden. Sei ${}^{MB}\vec{\rho}_{HGkart}$ die Darstellung des Vektors in kartesischen Koordinaten im System der Kamera, dann ist:

$$ {}^{MB}\vec{\rho}_{HGkart} = {}^{MB}_{G}\mathbf{T} \, {}^{G}_{K}\mathbf{T} \, {}^{K}\vec{\rho}_{HGkart} \tag{4.23} $$

wobei ${}^{MB}_{G}\mathbf{T}$ aus der Kinematik des Roboterarmes und ${}^{G}_{K}\mathbf{T}$ aus der Kamerakalibrierung bekannt sind. Anschließend wird der Ausweichvektor mit einem geradlinigen Pfad dargestellt. Die Länge des Pfades ist proportional zum Betrag des Vektors, während die Orientierung des Pfades mit der Richtung des Vektors übereinstimmt.

Erlernen der Hindernisvermeidung des Greifers

Um die Parameter des verwendeten Neurofuzzy-Systems zu ermitteln, die ein robustes Ausweichen der Hindernisse ermöglichen, wurde die Steuerungskomponente in der virtuellen Umgebung mit dem

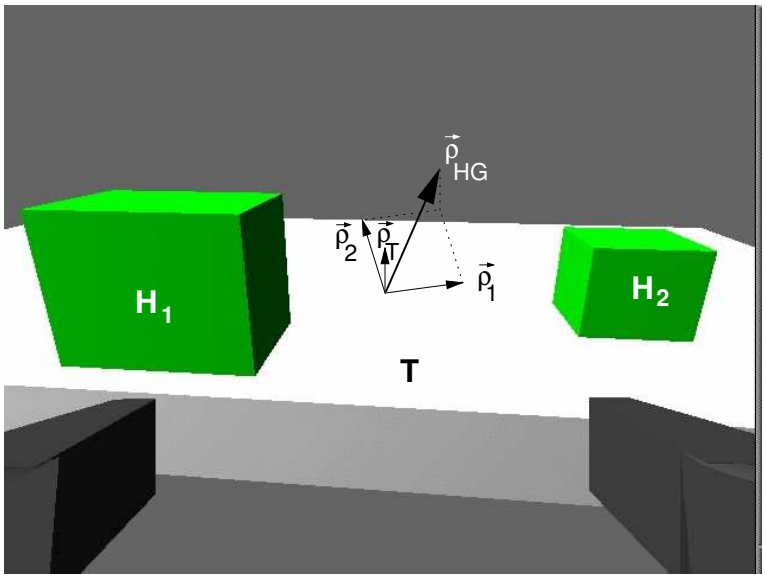


Abbildung 4.16: Fusion der Ausweichvektoren bei der Hindernisvermeidung des Greifers.

stochastischen Teach-In in 45 unterschiedlichen Lernszenarien trainiert. In jedem Lernszenario muss sich der Greifer von einer zufällig generierten Startposition kollisionsfrei nach vorne bewegen. Dabei soll eine Ausgangssituation mit einem Hindernis[19] bewältigt werden. Nach jeder Bewegung des Manipulators berechnet ein Bewerter aus den aktuellen Aufnahmen der Greiferkamera und der bekannten Position des Greifers zum Hindernis eine Bewertung der Konstellation. Anhand der vergebenen Bewertung wird dann die Steuerungskomponente angepasst. Unterschreitet der minimale Abstand des Objektes von der Kamera einen gesetzten Schwellenwert, wird dies als Kollision interpretiert und der Schritt wird abgebrochen. Das Szenario wird so oft wiederholt, bis der Manipulator am Hindernis kollisionsfrei vorbeikommt.

Der Bewerter implementiert die Belohnungsfunktion des *Reinforcement Learning* (Abbildung 4.17). Er ist hier mit einem Fuzzy System implementiert, das Bewertungen des aktuellen Umgebungszustands in Bezug auf eine Kollisionsgefahr erstellt, und wird nach der Trainingsphase vom System abgekoppelt. Als Eingabe erhält er den minimalen Abstand d_{HKmin} des Hindernisses von der Kamera und den minimalen Abstand des Hindernisses d_{HSAmin} von der Kamerasichtachse (Abbildung 4.18).

Jedem Eingang des Bewerters wird eine linguistische Variable mit drei Termen zugeordnet. Die Zugehörigkeitsfunktionen sowie die Regelbasis des Bewerters wurden nicht trainiert, sondern sind vorgegeben; sie sind in Anhang D dargestellt. Die Regelbasis ist auf Basis zweier empirischer Aussagen erstellt worden. Einerseits erhöht eine Annäherung des Greifers an das Hindernis die Kollisionsgefahr; in diesem Fall sollte das Verhalten eine negative Bewertung erhalten. Andererseits gibt es mehr Raum für eine Vorwärtsbewegung, je größer der Abstand des Hindernisses von der Sichtachse wird; deshalb sollte für solche Konstellationen eine positive Bewertung erteilt werden. Somit bestraft der Bewerter Bewegungen, die dem Manipulator in Kollisionsgefahr mit dem Hindernis bringen, belohnt jedoch Bewegungen, die den Greifer möglichst schnell voran führen.

Das eigentliche Training setzt den *Q-learning* Algorithmus [Wat89] ein, der von einem gegebenen Zustand Z_i nach der Aktion $\vec{\rho}$ sucht, die die akkumulierte zukünftige Belohnung und somit die Ak-

[19] Die Dimensionen des Hindernisses variieren pro Lernszenario.

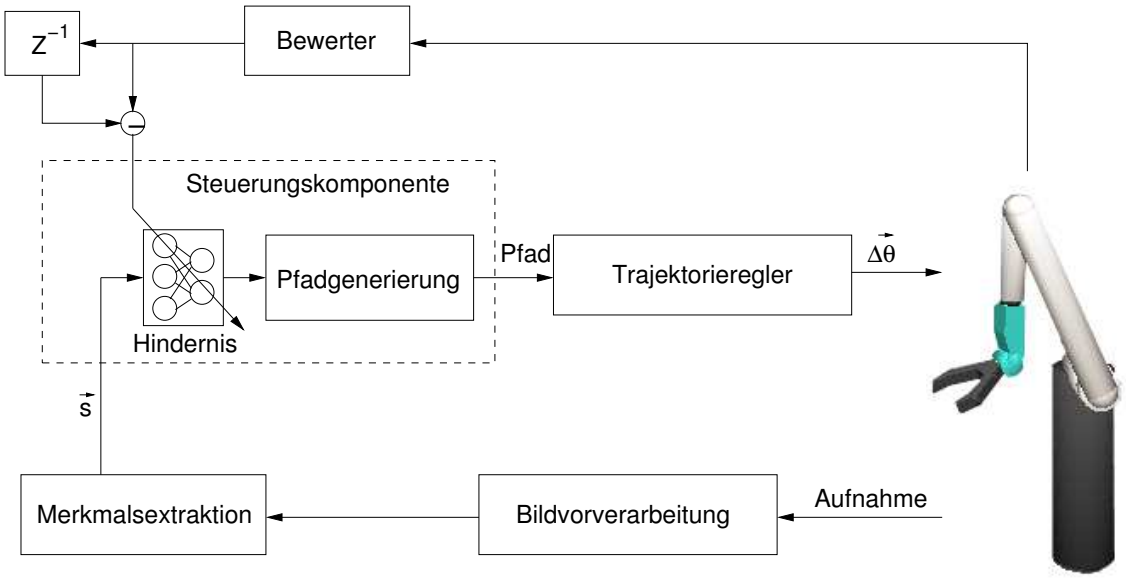


Abbildung 4.17: System während des Trainings der Hindernisvermeidung des Greifers.

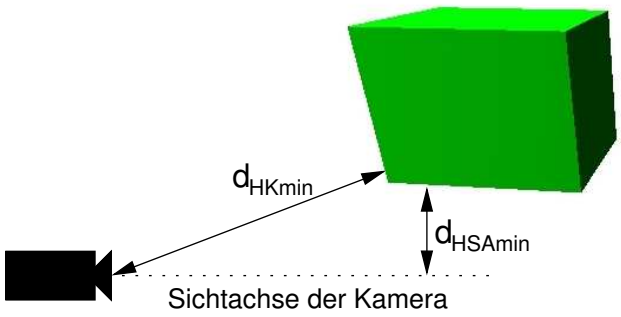


Abbildung 4.18: Eingänge des Fuzzy-Bewerters.

tionswertfunktion $Q(Z_i, \vec{\rho})$ maximieren kann[20]. Bei der Hindernisvermeidung des Greifers entspricht die Neurofuzzy-Regelbasis der Aktionswertfunktion[21] und wird dementsprechend während des Trainings verändert. Zu Beginn des Trainings sind die Singletons der Ausgabevariablen $|\vec{\rho}|$ und φ für alle Regeln so initialisiert, dass in jedem Fall der Roboterarm mit großer Geschwindigkeit geradeaus, in der z-Achse des Greifers, fährt.

Zur Anpassung der Gewichte wird nach dem Prinzip des *Q-Learning* die Differenz zwischen zwei nacheinanderfolgenden Ausgaben des Bewerters gebildet:

$$E(t) = r(t) - r(t-1) \tag{4.24}$$

[20] Der *Q-learning* Algorithmus und der verwandte TD(0) Algorithmus sind im Anhang C zusammengefasst.

[21] Bei der Hindernisvermeidung des Greifers entspricht die Aktion $\vec{\rho}$ dem Ausweichvektor des Roboterarmes. Die Neurofuzzy-Regelbasis muss dann denjenigen Ausweichvektor als besten bewerten und auswählen, der den Roboter in einer günstige Position auch für zukünftige Ausweichbewegungen bringt. Demnach entspricht die Regelbasis der Aktionwertfunktion.

Ist der Wert kleiner Null, dann ist der Roboter in eine Situation gekommen, die schlechter war als die frühere. Dadurch wurde eine schlechte Aktion ausgewählt und eine Anpassung der Gewichte des Netzes ist notwendig. Ist $E(t)$ größer Null, dann ist die gegenwärtige Situation besser und man muss das Netz nicht anpassen.

Die Anpassung der Gewichte zwischen der vierten und der fünften Schicht im Neurofuzzy System und demnach der Abszissen der Singletons beträgt (siehe auch Anhang C, Gleichungen C.72 und C.70):

$$\Delta w_{ij}(t) = \begin{cases} \eta_i \eta_{Ges} K_{sgn_i} \mid E(t) \mid (\frac{o_j^{(4)}}{\sum_{k=1}^{n} o_k^{(4)}}) & : \quad \text{falls} \quad E(t) < 0, \\ 0 & : \quad \text{falls} \quad E(t) \geq 0 \end{cases} \tag{4.25}$$

wobei n die Anzahl der Neuronen der vierten Schicht und somit die Anzahl der Regeln ist, $o_k^{(4)}$ die Ausgabe des k-ten Neurons der vierten Schicht angibt, η_i der Lerngeschwindigkeitskoeffizient für den i-ten Ausgang darstellt und η_{Ges} dem Lerngeschwindigkeitskoeffizient des gesamten Systems entspricht. Der Parameter K_{sgn_i} ist 1 im Fall des Winkels φ und -1 für $\mid \vec{\rho} \mid$. Dadurch nimmt die Geschwindigkeit des Greifers ab und gleichzeitig nimmt der Winkel der momentanen Bewegungsrichtung relativ zur Sichtachse der Kamera zu, falls eine Erhöhung der Kollisionsgefahr stattgefunden hat. Die aus dem Training resultierende Regelbasis ist in Anhang D präsentiert.

Resultate des Trainings

Für die Bewertung des Verhaltens wurden zufällig 10000 Szenarien erstellt und in der virtuellen Umgebung ausgewertet. Für jedes Szenario wurde ein virtuelles Hindernis erzeugt, dessen Position, Dimension und Orientierung unter den in Tabelle 4.3 aufgelisteten Einschränkungen zufällig gewählt wurden. In den Szenarios muss das Hindernisvermeidungsverhalten dem Hindernis erfolgreich ausweichen und darf keine Kollisionen verursachen.

Parameter	**von**	**bis**
Abstand Objektes zur Kamera d_K	12 cm	100 cm
Abstand Objektes zur Sichtachse d_{KSA}	0 cm	5 cm
Objektdimensionen	2 cm	25 cm
Objektorientierung	0^0	45^0 cm

Tabelle 4.3: Parametergrenzen bei der Evaluierung der Hindernisvermeidung des Greifers.

Die Resultate der Evaluierung sind in Tabelle 4.4 präsentiert. In $96,84\%$ der Fälle hat eine erfolgreiche Hindernisvermeidung stattgefunden. In den restlichen $3,16\%$ ist eine Kollision eingetreten; jedoch betrug bei 297 dieser 316 der Fälle der Abstand der Anfangsposition der Kamera zum Hindernis weniger als 18 cm. In diesem Fall ist der Greifer weniger als 8 cm vom Hindernis positioniert, was einen äußerst kleinen Manöverraum für den Manipulator erlaubt. Die restlichen $0,19\%$ sind auf Ungenauigkeiten bei der Bildverarbeitung und der Lokalisierung der Hindernisse zurückzuführen. In der Regel liegt bei realen Szenarien die Anfangsposition des Greifers weiter als 8 cm entfernt vom Hindernis, so dass die Ergebnisse des Verhaltens in der Evaluierung als sehr gut zu bewerten sind.

Fall		**Anzahl**	**%**
Erfolgreiche Ausführung des Szenarios		9684	$96,84\%$
Kollision mit Hindernis	Hindernisabstand zu Beginn ≤ 8 cm	297	$2,97\%$
	Hindernisabstand zu Beginn > 8 cm	19	$0,19\%$

Tabelle 4.4: Evaluierung der Hindernisvermeidung des Greifers in der virtuellen Umgebung.

4.2.3 Hindernisvermeidung der Manipulatorsegmente

Die Hindernisvermeidung des Greifers kann den Endeffektor, jedoch nicht die restlichen Segmente des Manipulators vor Kollisionen schützen. Deswegen ist ein zusätzliches Verhalten notwendig, das in der Nähe von Hindernissen eine Ausweichbewegung für die Manipulatorsegmente berechnet.

Der Verlauf der Hindernisvermeidung der Segmente ist in Abbildung 4.19 dargestellt. Das Verhalten verwendet Aufnahmen der Bordkamera des mobilen Manipulators, da diese einen größeren Überblick über die Szene und den Roboterarm liefern. Nach der Bildvorverarbeitung der Aufnahmen werden die Hindernisse und das Zielobjekt im Raum lokalisiert und im lokalen Konfigurationsraum um die aktuelle Manipulatorposition abgebildet. In dem berechneten Konfigurationsraum kann anhand der bekannten Geometrie und Kinematik des Manipulators eine schnelle Suche nach einem kollisionsfreien Pfad, der in Zielrichtung führt, stattfinden. Das Trainieren eines Verhaltens für den gesamten Manöverraum des Roboterarms ist dagegen sehr aufwendig und daher wurde auf ein Erlernen des Verhaltens verzichtet.

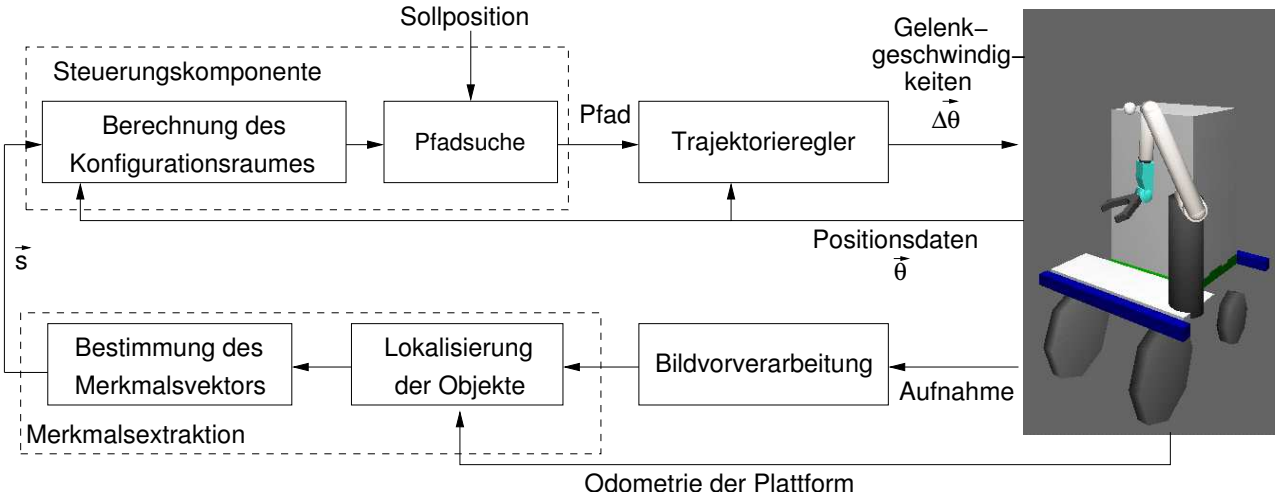


Abbildung 4.19: Ablauf der Hindernisvermeidung der Segmente.

Um die Objekte im Raum zu lokalisieren, akquiriert der mobile Manipulator mit der Bordkamera zwei Aufnahmen der Szene aus verschiedenen Positionen, indem sich die mobile Plattform zwischen den Aufnahmen nach vorne bewegt[22]. Da man im Testszenario von einer Situation ohne bewegte Hindernisse ausgeht und die observierte Szene sich nicht ändert, ist die Lokalisierung der Hindernisse nur einmal zu Beginn der Anwendung des Verhaltens notwendig.

[22]Da es sich um eine nicht holonome Plattform handelt, ist eine seitliche Bewegung des mobilen Manipulators nur umständlich durchzuführen.

Aus der Odometrie[23] der mobilen Plattform ist die Transformationsmatrix ${}^{P_2}_{P_1}\mathbf{T}$ der zweiten Aufnahmeposition der Plattform relativ zur ersten bekannt. Dadurch kann die Lageänderung der Kamera für die Bestimmung der Fundamentalmatrix $\mathbf{F}$ (Gleichung 4.14) und für die Auflösung des Gleichungssystem 4.12 berechnet werden:

$${}^{K_{P2}}_{K_{P1}}\mathbf{T} = {}^{K}_{P}\mathbf{T}\,{}^{P_2}_{P_1}\mathbf{T}\,{}^{K}_{P}\mathbf{T}^{T} \tag{4.26}$$

wobei ${}^{K}_{P}\mathbf{T}$ das Koordinatensystem der Bordkamera $\{K\}$ relativ zur Plattform $\{P\}$ angibt und aus der Konstruktion des mobilen Manipulators bekannt sein sollte.

Aus den Aufnahmen und der Matrix ${}^{K_{P2}}_{K_{P1}}\mathbf{T}$ können nach Abschnitt 4.2.1 die Hindernisse und die Auflagefläche relativ zur zweiten Kameraposition $\{K_{P2}\}$ lokalisiert und nachfolgend mit der Matrix ${}^{MB}_{K}\mathbf{T} = {}^{MB}_{P}\mathbf{T}\,{}^{P}_{K}\mathbf{T}$ relativ zur Manipulator-Basis $\{MB\}$ transformiert werden. Dabei werden die Objektdimension um einen Sicherheitsabstand d_{HS} vergrößert:

$$d_{HS} = d_{Segment} + d_{Lok} \tag{4.27}$$

wobei $d_{Segment}$ der Durchmesser eines Segmentes ist und d_{Lok} die Genauigkeit der Lokalisierung ausdrückt.

Anschließend folgt die Abbildung der Objekte in den Konfigurationsraum des Manipulators. Das Verhalten soll den Roboterarm vor Kollisionen in der lokalen Region um die aktuelle Gelenkstellung schützen; deswegen braucht man nur den lokalen Konfigurationsraum um die aktuelle Position des Manipulators zu berücksichtigen. Dadurch reduziert sich der Zeitaufwand bei der Bildung und der Suche im Konfigurationsraum erheblich. Da es sich beim eingesetzten Roboterarm um einen Manipulator mit sechs Gelenken handelt, verfügt der entsprechende Konfigurationsraum ebenfalls über sechs Dimensionen. Jedoch braucht man nur vier von sechs Gelenken zu berücksichtigen, ohne dass sich die Gefahr auf Kollisionen der Segmente mit Hindernissen erhöht; das vierte und das sechste Gelenk können vernachlässigen werden, da sie von der Hindernisvermeidung des Greifers geschützt werden. Die ersten drei Dimensionen des Raumes bilden sich aus den Werten der ersten drei Gelenke in einem Bereich von sechzig Grad um die aktuelle Konfiguration mit einer Auflösung von drei Grad. Die vierte Dimension besteht aus den Werten des fünften Gelenkes in der Anfangs- und Zielkonfiguration. Dadurch besteht der betrachtete lokale Konfigurationsraum aus einem Gitternetz der Dimension $40 \times 40 \times 40 \times 2$.

Jede Gelenkstellung, die im Gitter des lokalen Konfigurationsraumes dargestellt ist, wird auf Kollisionen der Manipulatorsegmente L_i[24] (Abbildung 4.20) mit einem der Hindernisse oder mit der Auflageoberfläche überprüft. Im Falle einer Kollision wird die entsprechende Konfiguration im Gitter als besetzt gekennzeichnet.

Für die Überprüfung auf Kollisionen werden die Segmentkoordinaten bei einer Gelenkstellung aus der Kinematik nach Gleichungen C.17 und C.18 berechnet. Da die Segmente des Manipulators als Zylinder darstellbar sind, muss man überprüfen, ob die Distanz eines Segmentes zu einem quaderförmigen Hindernis geringer als der Radius des entsprechenden Zylinders ist. Diese Überprüfung baut auf der Berechnung der Distanz eines Punktes M zu einem Quader auf. Sei ${}^{W}\vec{p}$ die Position des Punktes M im Raum und ${}^{W}_{Q}\mathbf{T}$ die Transformationsmatrix des Quaders relativ zur Welt. Dann ist die Position des

[23] Die Odometrie ist die Berechnung des zurückgelegten Weges der mobilen Plattform. Dazu verwendet sie hauptsächlich die Information aus den Drehgebern an den Rädern der Plattform.

[24] Da das Bewegungsvolumen des ersten Segmentes klein ist, braucht es bei der Kollisionsüberprüfung nicht berücksichtigt werden; nur L_2, L_3 oder L_6 werden auf Kollisionen getestet. Die Länge der Segmente ist aus der DH-Tabelle in Anhang C.1 zu entnehmen.

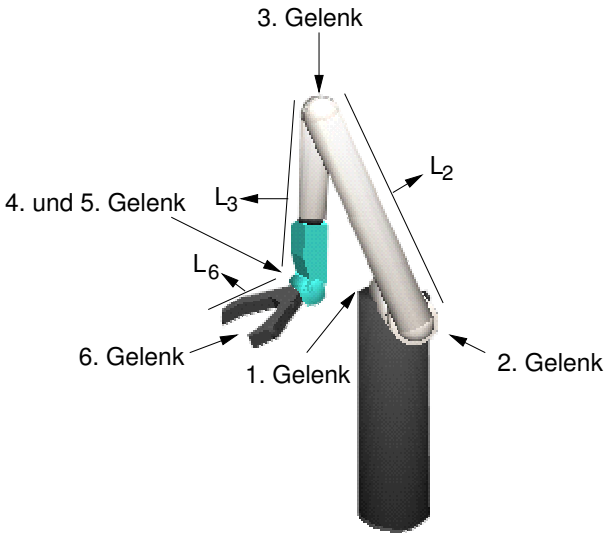


Abbildung 4.20: Segmente und Gelenke des eingesetzten Manipulators.

Punktes relativ zum Koordinatensystem des Quaders ${}^{Q}\vec{p} = {}^{W}_{Q}\mathbf{T}^{-1}\,{}^{W}\vec{p}$. Zur Distanzbestimmung wird der Punkt M' auf der Oberfläche des Quaders gesucht, der dem Punkt M am nächsten ist (Abbildung 4.21). Durch eine nachfolgende Berechnung des Abstandes dieser beiden Punkte ergibt sich der gesuchte Abstand vom Quader zum Punkt M:

$$\mathrm{dist}({}^{Q}\vec{p},\, {}^{Q}\vec{p'}) = \left| {}^{Q}\vec{p} - \begin{pmatrix} \min({}^{Q}p_x, \frac{d_x}{2}) \\ \min({}^{Q}p_y, \frac{d_y}{2}) \\ \min({}^{Q}p_z, \frac{d_z}{2}) \end{pmatrix} \right| \tag{4.28}$$

wobei $\vec{d} = (d_x, d_y, d_z)^T$ die Dimensionen des Quaders ist und ${}^{Q}\vec{p'}$ die Koordinaten von M' beinhält. Aus dieser Berechnung kann man den Abstand einer Strecke und somit eines Segmentes von einem Quader bestimmen. Die Strecke wird dabei durch mehrere äquidistante Punkte, für welche die beschriebene Abstandsberechnung einzeln durchgeführt wird, angenähert.

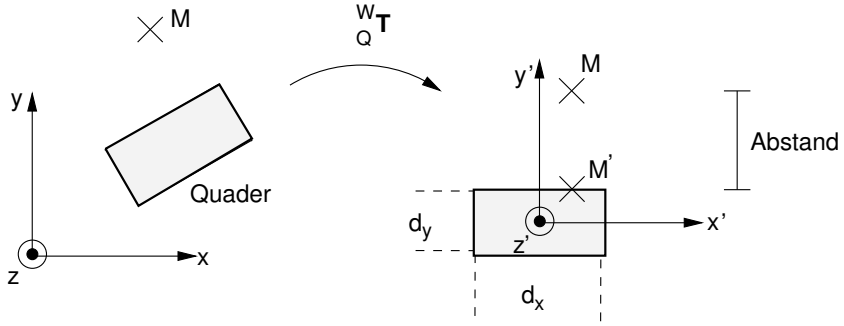


Abbildung 4.21: Berechnung des Abstandes eines Punktes von einem Quader. Um den zu M nächsten Punkt M' auf dem Quader zu finden, werden die Koordinaten von M im System des Quaders, also die Elemente von ${}^{Q}\vec{p}$, auf die Hälfte der jeweiligen Dimension des Quaders reduziert, ohne das Vorzeichen zu ändern. Ist die Koordinate bereits kleiner als die Ausdehnung in der betrachteten Dimension, bleibt sie unverändert.

Nach der Berechnung des lokalen Konfigurationraumes (Abbildung 4.22) findet die eigentliche Pfadsuche statt (Abbildung 4.23). Wenn das Zielobjekt außerhalb der betrachteten Konfigurationsregion liegt, wird als Ziel der Suche eine freie Konfiguration am Rande des lokalen Konfigurationsraumes in der Zielrichtung ausgewählt. Sonst ist das Ziel die Position des Zielobjektes selbst.

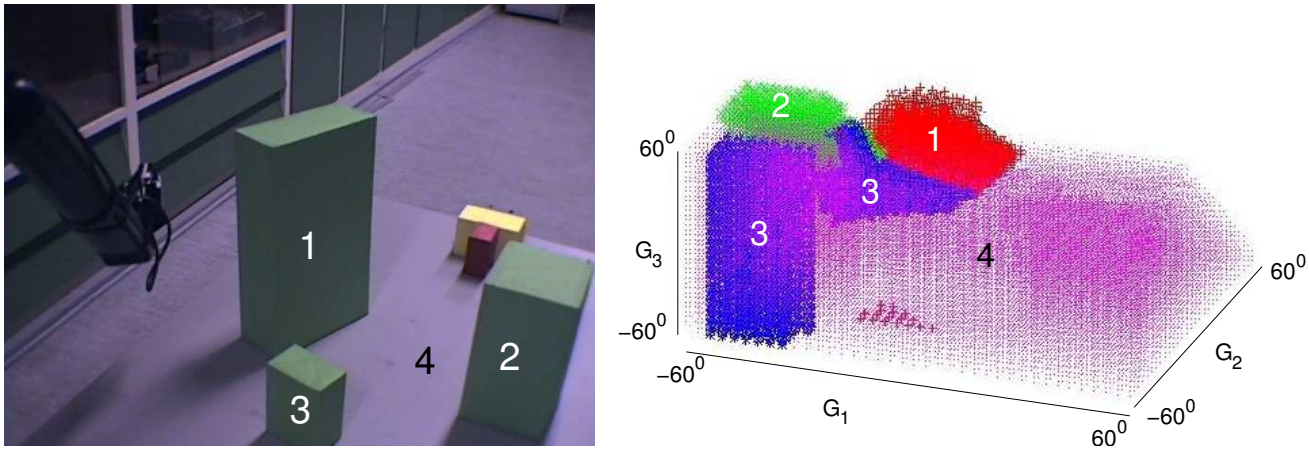


Abbildung 4.22: Szene und korrespondierender Konfigurationsraum um die aktuelle Manipulatorposition.

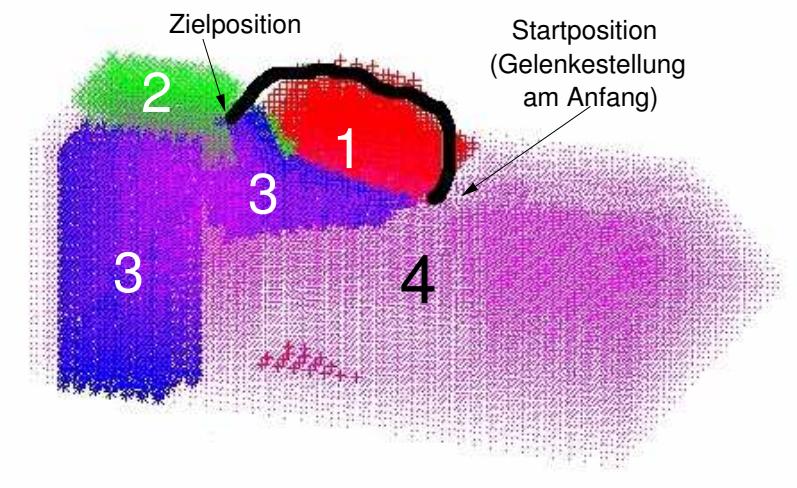


Abbildung 4.23: Ermittelter Pfad im Konfigurationsraum.

Für die Pfadberechnung wird ein Dekompositionsverfahren eingesetzt, nämlich die *Wavefront Expansion* [RH96] (Abbildung 4.24). Es handelt sich um ein schnelles Verfahren, das keine lokale Minima erzeugt. Das Verfahren liefert den kürzesten Pfad zur Zielposition unter der Voraussetzung, dass sie von der Anfangsposition erreichbar ist. Nach der Wegesuche übermittelt das Verhalten den berechneten Pfad dem Trajektorieregler (Abbildung 4.25) und wartet auf seine erneute Ausführung.

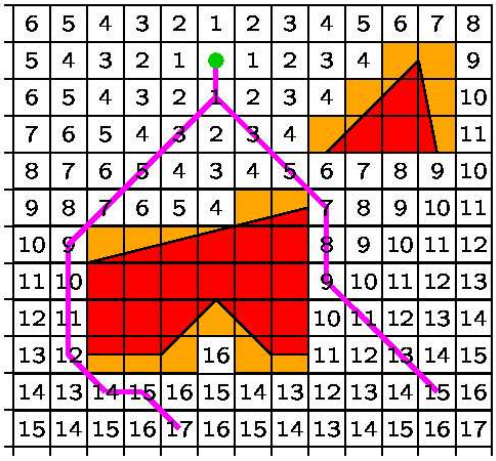


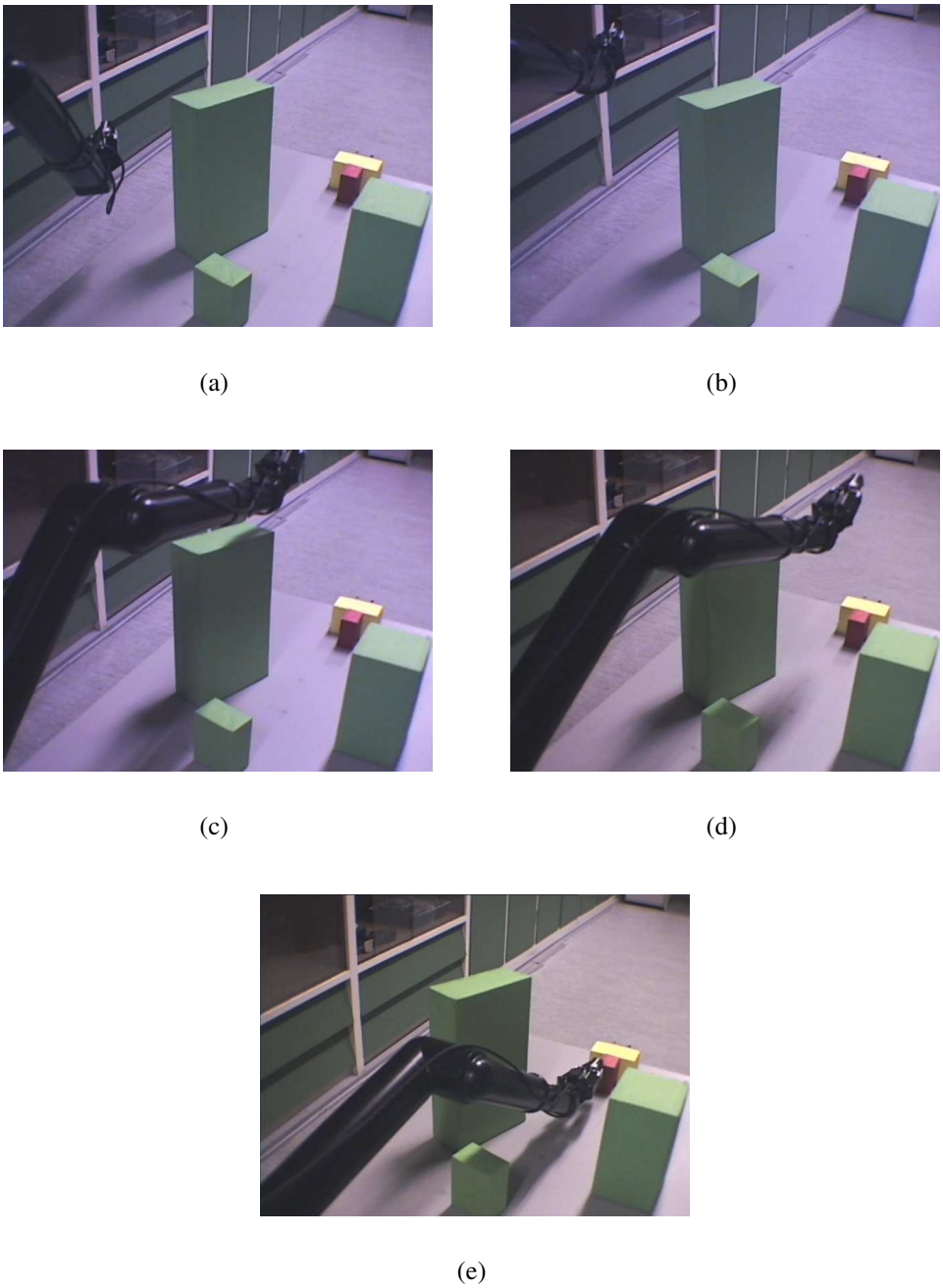
Abbildung 4.24: *Wavefront Expansion* im zweidimensionalen Raum. Von der Zielposition ausgehend wird jede Zelle bezüglich ihres Abstandes zum Ziel bewertet. Bei der Pfadsuche wird von der aktuellen Position diejenige Zelle als nächste anfahrbare Position ausgewählt, die über die kleinste Bewertung aller benachbarten Zellen verfügt. Das Verfahren funktioniert im n-dimensionalen Raum analog.

4.3 Pfadplanung im lokalen Manipulationsraum

Nach Brooks [Bro86] sind Verhalten Bedingungs-Aktions Paare, die kein Modell der Einsatzumgebung beinhalten sondern direkt auf den Sensordaten operieren und entsprechende Roboterbewegungen generieren. In diesem Sinne ist die Pfadplanung nicht als Verhalten zu realisieren, da sie ein Modell der Umgebung einsetzt. Andererseits ist die Information aus einem Pfadplaner nützlich, wenn das Zielobjekt wegen Verdeckungen nicht mehr von den Kameras erfasst wird oder wenn die reaktiven Verhalten zu lokalen Minima bzw. einer Deadlock-Situation führen. Deswegen wird hier ähnlich zur DAMN-Architektur [Ros95] ein lokaler Pfadplaner als ein weiteres Verhalten ausgewertet und trägt zur resultierenden Aktion bei. Um die Rechenzeit niedrig zu halten, berechnet das Verfahren einen Pfad für den Greifer im kartesischen Raum, ohne dessen Orientierung zu berücksichtigen; die Bildung des Konfigurationsraumes und die nachträgliche, ausführliche Pfadplanung für sechs Freiheitsgrade ist für eine Implementierung als Verhalten zu zeitaufwendig. Da der Ausgabepfad von diesem Verhalten nicht als verbindlich angesehen wird, liegt hier der Grundsatz *Plan-as-Resource*[25] vor [PRK90].

Das Verhalten verwendet ein grobes 3D-Modell der Szene, das aus den Daten der Bordkamera nach Abschnitt 4.2.1 bei der Aktivierung des Verhaltens rekonstruiert wird. Das Modell bestimmt auch den Suchraum bei der Pfadplanung. Anschließend finden die eigentliche Pfadsuche nach dem *Probabilistic Roadmap*-Prinzip [KL94a], [HST94], [ABD+98a] (siehe auch Abschnitt 2.4.2) und die Überprüfung der Gültigkeit des berechneten Pfades statt. Falls der Pfad Stützstellen enthält, die unerreichbar sind oder Kollisionen der Manipulatorsegmente mit Hindernissen verursachen, werden die Stützstellen mit zusätzlichen, so genannten virtuellen Hindernissen besetzt, so dass bei einer erneuten Pfadsuche diese Positionen nicht berücksichtigt werden. Dieser Ablauf ist auch im Flussdiagramm des Verhaltens in Abbildung 4.26 schematisch dargestellt.

[25] Der *Plan-as-Resource* Ansatz behandelt einen Plan als einen Vorschlag, der geändert werden darf. Demgegenüber steht der *Plan-as-Programm* Ansatz, bei dem der Plan als Befehl zu verstehen ist, der unabhängig von der aktuellen Situation auszuführen ist.



(a) (b)

(c) (d)

(e)

Abbildung 4.25: Bewegung des Manipulators bei der Hindernisvermeidung der Segmente. Dabei wurde das Verhalten zweimal ausgeführt. Die Bilder (a) bis (c) zeigen aus der Sicht der Bordkamera die Manipulatorbewegung nach der ersten Ausführung des Verhaltens und Aufnahmen (c) bis (e) nach der zweiten Ausführung.

Die Ausgabe der Pfadplanung dient als eine erste grobe Schätzung der gesamten Bewegung zu Beginn eines Greifvorgangs und wird anhand der Ausgaben der reaktiven Verhalten angepasst. Diese Anpassung ist notwendig, da die einmalige bildbasierte Lokalisierung der Hindernisse sowie des Zielobjektes nicht genügend genau ist, um einen kollisionsfreien, exakten Pfad zur Zielposition zu garantieren. Darüber hinaus kann die Pfadplanung keine Hindernisse berücksichtigen, die von der Bordkamera nicht registriert sind.

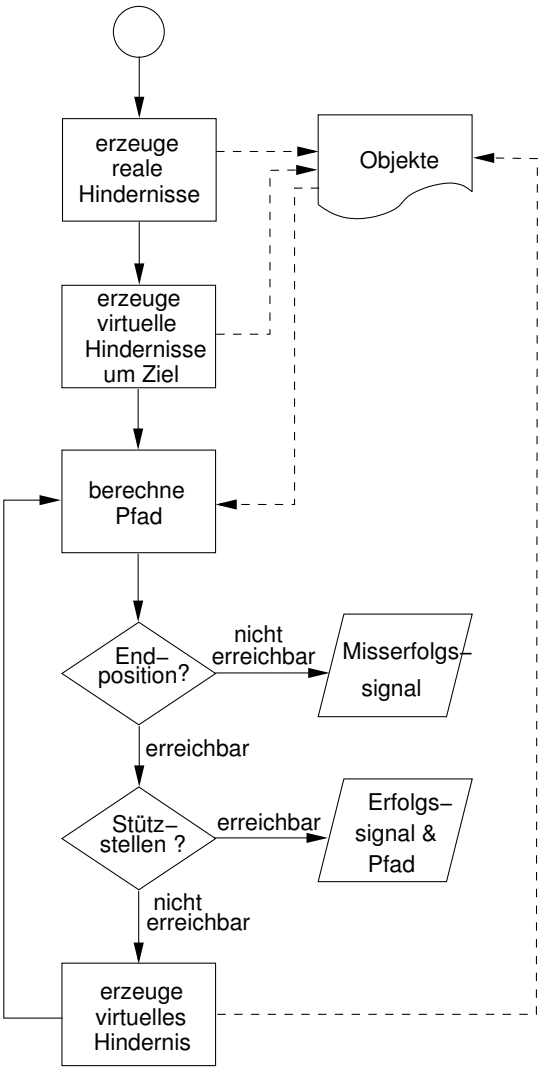


Abbildung 4.26: Flussdiagramm des Planungsverhaltens.

4.3.1 Bestimmung des Suchraumes

Der Suchraum wird durch eine Menge von Freiräumen und eine Menge von Hindernissen dargestellt. Jedoch entspricht nicht jeder Punkt, der im Freiraum liegt, auch einer kollisionsfreien Position des Greifers, denn der Greifer selbst belegt zusätzlichen Raum. Deshalb müssen die physikalische Abmessungen des Greifers bei der Pfadsuche mitberücksichtigt werden. Um dieses Problem zu umgehen, werden die reale Hindernisse um die Abmessungen des Greifers vergrößert und der Greifer selbst zu einem Punkt ohne Dimensionen reduziert. So kann man bei der Pfadsuche sicherstellen, dass der Abstand des Pfades zu den Hindernissen immer ausreichend ist. Da der in der Wegesuche errechnete Pfad sich auf den relativ weit vorne liegenden Punkt zwischen den beiden Fingern der Greifers[26] bezieht, ist die Kollisionsgefahr größer, wenn der Greifer sich zwischen Zielobjekt und Hindernis befindet. Die Hindernisse müssen daher auf der dem Zielobjekt zugewandten Seite stärker vergrößert werden als auf der gegenüberliegenden Seite (Abbildung 4.27). Wegen dieser unregelmäßigen Änderung der Dimension muss man die Position des Hindernisses entsprechend anpassen.

[26] Siehe auch die Definition der Manipulator-Koordinatensysteme und der DH-Parameter in Anhang C.1.

Sei also $\vec{d}_{O_i}$ die Dimension und ${}^{MB}\vec{p}_{O_i}$ die Position von Hindernis O_i relativ zur Manipulator-Basis, $\vec{d}_G = (d_{Gx}, d_{Gy}, d_{Gz})^T$ die Dimension des Greifers und ω die Orientierung des Zielobjektes um die z-Achse der Manipulator-Basis. Dann ist die neue, vergrößerte Dimension des Hindernisses:

$$\vec{d}'_{O_i} = \vec{d}_{O_i} + \frac{1}{2} \begin{pmatrix} d_{Gx} \,|\cos(\omega)| + d_{Gy} \\ d_{Gx} \,|\sin(\omega)| + d_{Gy} \\ d_{Gz} \end{pmatrix} \tag{4.29}$$

und die angepasste Position:

$${}^{MB}\vec{p}'_{O_i} = {}^{MB}\vec{p}_{O_i} + \begin{pmatrix} d_{Gx}\cos(\omega) \\ d_{Gx}\sin(\omega) \\ 0 \end{pmatrix} \tag{4.30}$$

Von besonderer Bedeutung ist auch die Form des Zielobjektes, die eine bestimmte Greifrichtung erzwingt. Um die Greifrichtung bei der Pfadplanung zu berücksichtigen, wird das Zielobjekt mit virtuellen Hindernissen umgeben (Abbildung 4.27), deren Lage relativ zum Koordinatensystem des Zielobjektes fest definiert ist.

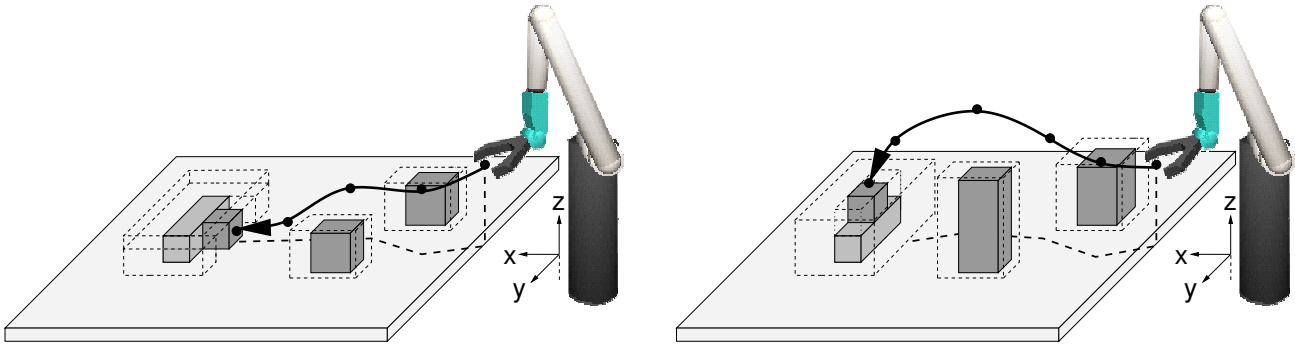


Abbildung 4.27: Reale und virtuelle Hindernisse (gestrichelte Linien) beim Planungsverhalten.

Zur schnelleren Berechnung und Ausführung des Verhaltens wird der Raum, in dem das Planungsverfahren nach einem Pfad sucht, eingeschränkt. Dies kann ohne Verlust an möglichen Lösungen stattfinden, da die Anordnung der Gelenke des Manipulators nur solche Gelenkstellungen erlaubt, bei denen sich der gesamte Arm in einer Ebene senkrecht zur Auflageebene der Objekte befindet. Das seitliche Ausweichen des Greifers vor einem Hindernis ist deshalb nur bedingt sinnvoll und erhöht in den meisten Fällen die Kollisionsgefahr für die übrigen Segmente (Abbildung 4.28). Deshalb kann man aufgrund der Kinematik des verwendeten Manipulators den Suchraum in einem schmalen Korridor limitieren. Die Position, Orientierung und Größe des Korridors werden aus den jeweiligen Positionen des Greifers und des Zielobjektes berechnet.

Sei ${}^{MB}\vec{p}_Z = (x_Z, y_Z, z_Z)^T$ die Position des Zielobjektes zur Manipulator-Basis und entsprechend ${}^{MB}\vec{p}_G = (x_G, y_G, z_G)^T$ die Position des Greifers. Dann ist die Position des Ursprungs des Koordinatensystems des Korridors:

$${}^{MB}\vec{p}_K = \frac{1}{2}({}^{MB}\vec{p}_Z + {}^{MB}\vec{p}_G) \tag{4.31}$$

Die Orientierung des Koordinatensystems des Korridors ist abhängig von der Position des Zielobjek-

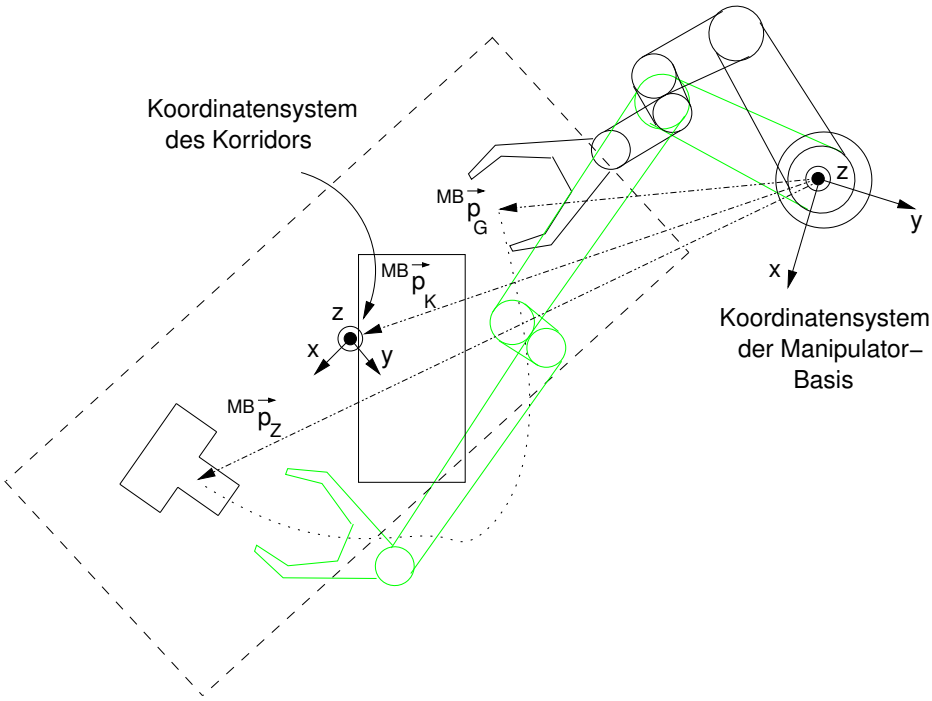


Abbildung 4.28: Kollisionsgefahr durch einen Pfad der seitlich an einem Hindernis verläuft. Der Suchkorridor ist mit gestrichelter Linie dargestellt.

tes und des Greifers auf der x-y Ebene des Koordinatensystems der Manipulator-Basis:

$$^{MB}\vec{o}_K = \begin{pmatrix} 0 \\ 0 \\ atan2(y_Z - y_G, x_Z - x_G) \end{pmatrix} \tag{4.32}$$

Der Dimensionsvektor, angegeben im Koordinatensystem des Korridors, ist sowohl von der relativen Lage des Greifers zum Zielobjekt als auch von der Reichweite des Roboterarmes abhängig:

$$^{K}\vec{d}_K = \begin{pmatrix} \frac{\sqrt{(x_Z - x_G)^2 + (y_Z - y_G)^2}}{2} + 100 \\ \frac{k_M}{2} \\ \frac{k_M}{2} \end{pmatrix} \tag{4.33}$$

wobei k_M die Reichweite der Roboterarmes ist.

4.3.2 Pfadsuche

Die Pfadsuche ist als Suche nach dem kürzesten Weg in einem Graphen mit gewichteten Kanten realisiert. Dazu muss die geometrische Darstellung der Umgebung in einem Graph abgebildet werden. Für die Realisierung dieser Abbildung wird eine vereinfachte Version des *Probabilistic Roadmap*-Verfahrens eingesetzt.

Zunächst wird eine große Anzahl von Punkten zufällig in den Freiräumen verteilt. Um aus diesen Punkten den Graphen zu bilden, wählt man zuerst einen beliebigen Punkt M_i aus. Alle Punkte, die in einem kleineren Abstand als ein Schwellenwert k_{D_1} zu M_i liegen, werden von der Punktmenge

entfernt. Alle restliche Punkte, die näher zu M_i liegen als k_{D_2}, verbindet man mit M_i über eine Kante. Danach wird einer der verbundenen Punkte ausgewählt und der Vorgang wird wiederholt, bis alle Punkte bearbeitet sind. Die Punkte, die nicht entfernt wurden, bilden dann die Knoten, ihre Verbindungen miteinander die Kanten des Graphen.

Anschließend findet die Wegesuche im Graphen statt. Der berechnete Weg muss möglichst kurz sein; gleichzeitig muss der Greifer genügend Abstand zu Hindernissen bewahren, damit die Ausweichmöglichkeit des Roboters in der Nähe von Hindernissen nicht eingeschränkt wird. Der Roboterarm sollte enge Passagen möglichst vermeiden solange zwar längere dafür aber sichere Wege vorhanden sind. Deswegen werden die Kanten des Graphen gewichtet. Die Gewichtung einer Kante g_{ij} hängt sowohl von der geometrischen Entfernung der zugehörigen Punkte M_i und M_j zueinander, als auch von ihrer Distanz zum nächsten Hindernis ab.

$$g_{ij} = \sqrt{((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2)\frac{w_i + w_j}{2}} = |{}^{MB}\vec{p}_i - {}^{MB}\vec{p}_j| \sqrt{(\frac{w_i + w_j}{2})} \quad (4.34)$$

wobei ${}^{W}\vec{p}_l = (x_l, y_l, z_l)^T$ die Position von M_l im Suchraum ist und w_l ein Maß der Hindernisentfernung:

$$w_l = max(1, k_{DF}(1 - d_{OM_l})) \quad (4.35)$$

Dabei ist k_{DF} eine Konstante und d_{OM_l} die Distanz von Punkt M_l von der nächsten Hindernisumrandung, die nach dem Verfahren in Abschnitt 4.2.3 berechnet wird. Durch Gleichung 4.34 werden Kanten in der Nähe von Hindernissen schlechter bewertet, um geometrisch längere Wege in größerer Entfernung von Hindernissen zu bevorzugen.

Der kürzeste Weg im Graph wird nach dem Algorithmus von Dijkstra [Dij59] ermittelt. Der Graph wird dazu in einen Baum umgewandelt, in dem jeder Knoten über den kürzesten Weg mit dem Ziel verbunden ist. Daher kann auf Neuberechnung des Graphen verzichtet werden, wenn ein Weg von einer neuen Anfangsposition zur selben Zielposition berechnet werden soll.

Die Qualität des ermittelten Pfades ist von der Genauigkeit der Hindernislokalisierung mit der Bordkamera abhängig. Ist der Lokalisierungsfehler gering, dann führt der ermittelte Pfad kollisionsfrei zur Zielposition. Je ungenauer jedoch die Lokalisierung, desto wahrscheinlicher werden Kollisionen mit Hindernissen. Da beim Pfadplanungsverhalten keine bildgestützte Regelung des Greifers beim Erreichen der Zielposition stattfindet, verursacht eine fehlerbehaftete Objektlokalisierung einen Misserfolg beim Greifen. Verlässt weiterhin der Manipulator den vom Planungsverhalten ermittelten Pfad, dann können nur die reaktiven Hindernisvermeidungsverhalten Kollisionen mit Objekten verhindern. Deshalb sind die reaktive Verhalten notwendig, um anhand der aktuellen Kameradaten den vom Pfadplanungsverhalten vorgeschlagenen Pfad anzupassen bzw. zu korrigieren.

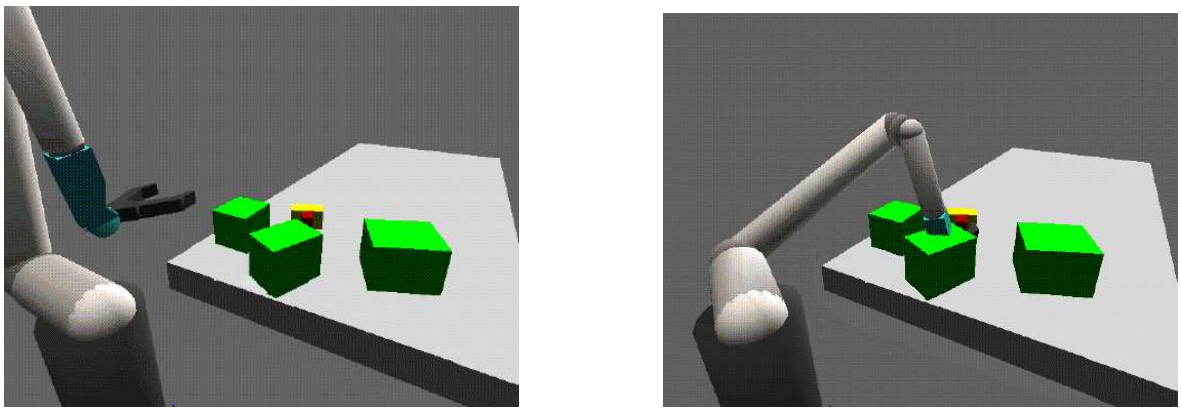
4.3.3 Überprüfung der Stützstellen des Pfades

Nachdem ein Pfad für den Greifer gefunden ist, muss man anhand der inversen Kinematik überprüfen, ob die kartesischen Stützstellen des Pfades vom Manipulator erreichbar sind, d.h ob die Gelenkstellungen es dem Greifer erlauben, die berechnete Position anzunehmen[27]. Die Stützstelle wird als unerreichbar deklariert, falls die inverse Kinematik keine Lösung liefert, oder die Lösung zu Selbstkollisionen der Manipulatorsegmente führt. Das Verhalten platziert ein virtuelles Hindernis an der

[27] Gleichungen C.20-C.25 im Anhang C.1.

entsprechenden kartesischen Position, so dass bei einer Neuplanung ein alternativer Lösungspfad berechnet wird.

Sind dagegen die Stützstellen erreichbar, so werden sie anschließend auf Kollisionen der Manipulatorsegmente mit Hindernissen nach dem Ansatz im Abschnitt 4.2.3 überprüft; eingefügte virtuelle Hindernisse werden dabei nicht berücksichtigt. Treten bei mehreren Stützstellen Kollisionen mit Hindernissen ein, dann werden an den entsprechenden Positionen virtuelle Hindernisse platziert und die Planung wird mit einer erhöhten Anzahl von im Freiraum zufällig verteilten Punkten wiederholt. Findet eine Kollision bei der letzten Stützstelle des Pfades statt, wie beispielsweise in Abbildungen 4.29a und 4.29b zu sehen ist, dann ist das Greifen unmöglich. Dieser Fall verursacht einen Abbruch des Verhalten mit einer Misserfolgsmeldung zur vermittelnden Ebene, ohne dass eine Greifbewegung stattfindet. Ansonsten wird der berechnete Pfad zur Ausführung weitergeleitet.



(a) (b)

Abbildung 4.29: Ausgangsstellung (a) und Endstellung (b) in einer Konfiguration, bei der kollisionsfreies Greifen unmöglich ist. Der Greifvorgang wird dann nicht initiiert.

Kapitel 5

Verhaltensauswahl und Verhaltenskoordination

Dieses Kapitel stellt die schwerpunktmäßig untersuchten Varianten der Verhaltensauswahl und Verhaltenskoordination vor, die in der mittleren, vermittelnden Ebene der drei-Ebenen Architektur realisiert sind. Die vermittelnde Ebene stellt die Verbindung zwischen den Verhalten der reaktiven Ebene und der Ausgabe der deliberativen Ebene[1] her. Die Aufgabe der vermittelnden Ebene besteht darin, anhand eines gegebenen Planungsschrittes ein bzw. mehrere geeignete Verhalten aus der Menge aller vorhandenen auszuwählen (Verhaltensselektion, *Action Selection Problem*). Falls mehrere Verhalten parallel aktiviert werden und zur Lösung beitragen, müssen zusätzlich die Ausgaben der Verhalten anhand des aktuellen Sensorkontext, der die gegenwärtige Situation der Umgebung wiederspiegelt, koordiniert werden (Verhaltenskoordination, *Behaviour Coordination Problem*). Während der Koordination kontrolliert die vermittelnde Ebene, ob ein Verhalten sein Ziel erreicht hat, nicht mehr zur Gesamtaufgabe beiträgt oder sich die Bedingungen für seine Aktivierung geändert haben. Dementsprechend lässt sich der Ablauf der vermittelnden Ebene in zwei Phasen gliedern, wie in Abbildung 5.1 schematisch dargestellt ist. Die erste Phase, auch als *Temporal Sequencing* bezeichnet [Bal97], übernimmt die Auswahl und Aktivierung der Verhalten, die zweite die Koordination der Verhaltensausgaben.

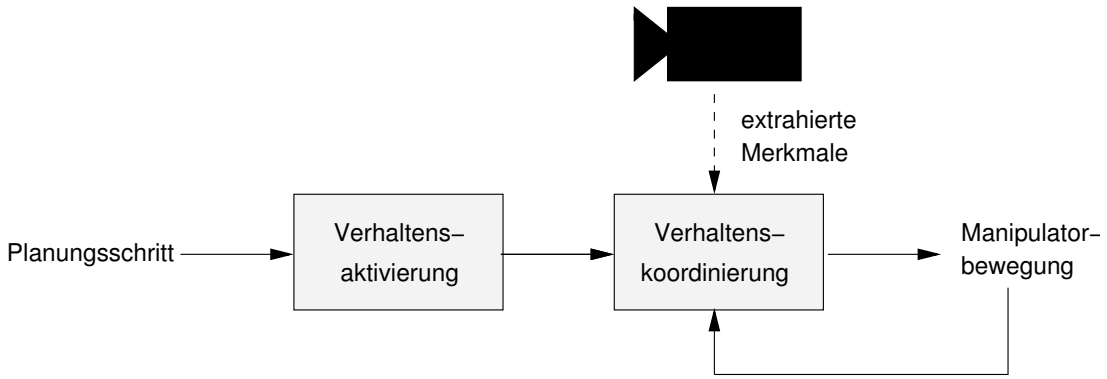


Abbildung 5.1: Struktur der vermittelnden Ebene.

[1]Die deliberative Ebene produziert einen Plan zur Lösung einer Aufgabe in Form einer Folge von auszuführenden Planungsschritten, die von der vermittelnden Ebene sequentiell abarbeitet werden.

Dieses Kapitel ist wie folgt organisiert: Zuerst wird der konzeptionelle Aufbau der vermittelnden Ebene präsentiert und auf die Verhaltensselektion eingegangen. Als nächstes folgt die Implementierung der Verhaltenskoordinierung und die Darstellung des Trainings in der virtuellen Umgebung. Zum Schluss werden die Ergebnisse des Trainings und der Einsatz der implementierten Struktur präsentiert und diskutiert.

5.1 Konzeptueller Aufbau der vermittelnden Ebene

Die realisierte vermittelnde Ebene besteht aus zwei unabhängigen Komponenten, eine für den Manipulator und eine für die mobile Plattform. Ihre Aufgabe ist, durch geeignete Kombination von Verhalten eine Vielzahl von möglichen Aktuator-spezifischen Planungsschritten erfolgreich zu lösen.

Der implementierte Ansatz stellt eine Kombination der *Arbitration*- und *Command Fusion*-Mechanismen dar (Abbildung 5.2). Durch die kombinierte Anwendung der beiden Mechanismen macht sich der hier realisierte Ansatz die Vorteile beider Verfahren zunutze (siehe auch Abschnitte 2.3.1 und 2.3.2). Als ein *Arbitration* Mechanismus zerlegt er eine von der deliberativen Ebene erhaltenen Planungsschritt in eine Reihenfolge von Gruppen von parallel auszuführenden Verhalten. Das Problem besteht dann darin, die Verhaltensgruppen in der richtigen Reihenfolge zu aktivieren. Zu diesem Zweck werden endliche Zustandsakzeptoren (*Finite State Acceptors*, FSA) eingesetzt. Die vermittelnde Ebene wählt entsprechend dem aktuellen Zustand im FSA eine Untermenge der Verhalten zur Aktivierung aus. Tritt ein vorher definiertes Signal ein, dann findet eine Transition in einen neuen Zustand statt.

Während der Ausführung einer Gruppe von Verhalten funktioniert die vermittelnde Ebene wie ein *Command Fusion*-Mechanismus. Hier findet die eigentliche Koordinierung der bei jedem Zustand Z_j aktivierten Verhalten mit Hilfe von *Bayesian Belief Networks* (BBN) statt. Anhand der aktuellen Sensorwerte und der Manipulatorposition werden die Ausgaben der aktiven Verhalten kombiniert. Dies umfasst die Berechnung der Gewichtungsmatrix $\mathbf{G}$ und der Koordinierungsfunktion C (siehe auch Gleichung 2.16), die die Ausgabe der Verhalten fusioniert und die nächste Bewegung des Roboterarmes bestimmt. Dieser Vorgang wiederholt sich, sobald neue Sensor- und Positionswerte vorliegen. Die Phase der Verhaltenskoordinierung wird abgeschlossen, wenn eines der Verhalten seine erfolgreiche Durchführung oder einen Misserfolg zurückmeldet.

5.2 Verhaltensauswahl

5.2.1 Endliche Zustandsautomaten und endliche Zustandsakzeptoren

Ein endlicher Automat (*Finite State Automaton* bzw. *Finite State Machine*, FSM) ist ein mathematisches Modell eines Systems mit diskreten Ein- und Ausgaben. Er besteht aus einer endlichen Menge von Zuständen und einer Menge von Übergängen (*Transitions*), die auf einem Eingang operieren und einen Zustand in einen anderen überführen. Dabei umfasst der Zustand des Systems die Information, die sich aus den bisherigen Eingaben ergeben hat und die benötigt wird, um die Reaktion des Systems auf noch folgende Eingaben zu bestimmen. Abhängig vom aktuellen Zustand verursacht eine Eingabe die Erzeugung einer bestimmten Ausgabe und einen Übergang des Automaten zu einem neuen Zustand bzw. zurück in den gleichen Zustand. Dementsprechend wird ein endlicher Automat formal

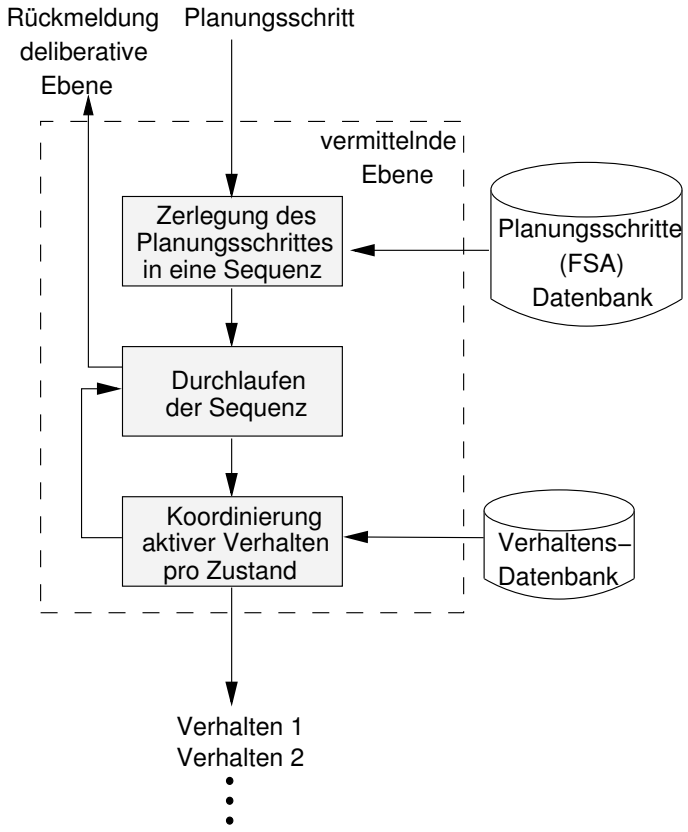


Abbildung 5.2: Ablauf der Verhaltensaktivierung und -koordination in der vermittelnden Ebene.

als ein Quintupel $(\mathcal{Z}_i, \mathcal{I}, \delta, Z_0, \mathcal{Z}_f)$ definiert, wobei $\mathcal{Z}_i$ eine endliche Menge von erlaubten, internen Zuständen ist, $\mathcal{I}$ eine endliche Menge von Eingabesignalen darstellt und $Z_0 \in \mathcal{Z}_i$ der Anfangszustand des Automaten ist. $\mathcal{Z}_f \subseteq \mathcal{Z}_i$ repräsentiert eine Menge von Endzuständen, auch als akzeptierende Zustände bekannt, und δ beschreibt die Übergangsfunktion, die die Abbildung $\mathcal{Z}_i \times \mathcal{I} \mapsto \mathcal{Z}_i$ realisiert.

Die Operationsweise der FSM wird über einen gerichteten Graphen, das so genannte Transitionsdiagramm (*State Transition Diagram*) dargestellt (vergleiche Abbildung 5.2.1). Dabei entsprechen die Knoten im Graph den Zuständen des FSM. Gibt die Übergangsfunktion δ an, dass beim Eingabesignal $a \in \mathcal{I}$ einen Übergang vom Zustand Z_0 in den Zustand Z_1 stattfindet, dann existiert ein mit a markierter Pfeil vom Zustand Z_0 in den Zustand Z_1 im Transitionsdiagramm. Der Übergang wird auch *Transition* genannt; das Signal, das eine Transition auslöst, wird als Ereignis oder *Event* bezeichnet.

Endliche Zustandsakzeptoren (*Finite State Acceptor*, FSA) sind Sonderfälle von endlichen Zustandsautomaten, die keine Ausgabe liefern und über zwei Endzustände verfügen. Ein FSA prüft, ob eine gegebene Sequenz von Eingabewerten zu einer gültigen Endkonfiguration führt oder nicht. Die Übergangsfunktion δ kann sowohl tabellarisch (siehe Tabelle 5.1) als auch graphisch wie bei den FSMs (siehe Abbildung 5.4) dargestellt werden.

Ein endlicher Zustandsakzeptor startet immer im Startzustand Z_0 und wechselt bei jeder Eingabe entsprechend der Übergangsfunktion seinen inneren Zustand. Sobald einer der beiden Endzustände erreicht ist, terminiert das FSA seinen Ablauf. In Abbildung 5.4 repräsentiert ein Kreis einen Zustand aus $\mathcal{Z}_i$, wobei ein Doppelkreis einen Endzustand aus $\mathcal{Z}_f$ angibt.

In der Robotik werden FSMs und FSAs zur Darstellung von Aktionssequenzen benutzt. Sie sind

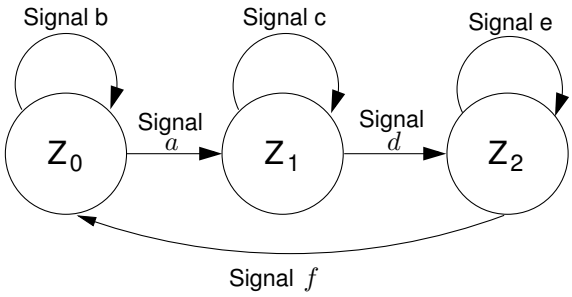


Abbildung 5.3: Beispiel eines Zustandsautomaten mit drei Zuständen und sechs Übergängen.

Zustand Z	Eingangssignal	$\delta(Z, a)$
Z_A	a	Z_B
Z_A	d	Z_D
Z_B	b	Z_C
Z_B	c	Z_D

Tabelle 5.1: Beispiel einer Tabelle für die Übergangsfunktion δ. Der entsprechende FSA-Graph ist in Abbildung 5.4 dargestellt.

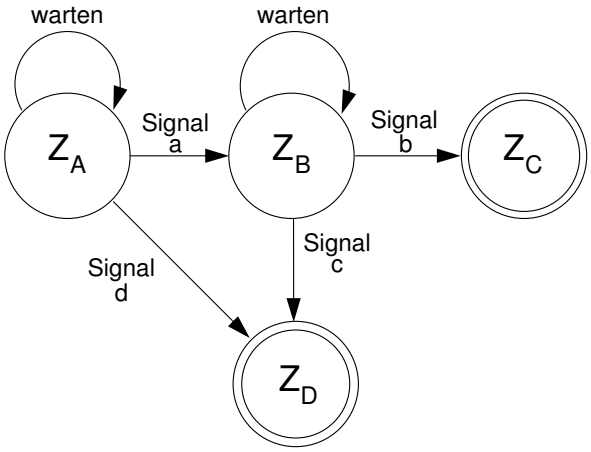


Abbildung 5.4: Beispiel eines endlichen Zustandsakzeptors, dessen Übergangsfunktion in Tabelle 5.1 dargestellt ist. Das FSA startet immer im Startzustand Z_A. Sobald einer der beiden Endzustände Z_C oder Z_D erreicht ist, terminiert das FSA seinen Ablauf.

daher zur Zerlegung einer Aufgabe in eine Sequenz von Lösungsschritte bzw. Aktionen geeignet, wobei jede Aktion einem Zustand des FSA-Diagrammes entspricht (vgl. Abschnitt 2.3.1). Nach Arkin [Ark98] kann bei jedem Zustand eine Teilmenge aller vorhandenen Verhalten aktiv sein. Sobald ein vorher fest definiertes Signal auftritt, das entweder einen Erfolg oder einen Misserfolg ausdrücken kann, findet eine Transition statt, die eine Änderung in der Menge der aktiven Verhalten bewirkt. Die Übergangsfunktion δ sowie die Gesamtmenge der möglichen Kombinationen von Verhalten sind in der Regel vorab bekannt und vom Entwickler fest programmiert.

5.2.2 Ablauf der Verhaltensauswahl

Die Verhaltensauswahl setzt FSAs ein, um die Gruppen der parallel auszuführenden Verhalten zu aktivieren. Jeder Knoten und demnach Zustand Z_i des FSA entspricht einer Gruppe von Verhalten, die parallel auszuführen sind und gemeinsam zur Bewegung des Aktuators beitragen. Dadurch ist die Reihenfolge der Aktivierung für jeden Planungsschritt im zugehörigen FSA kodiert. Eine Transition zum nächsten Knoten findet statt, sobald ein Verhalten seine erfolgreiche Durchführung oder einen Misserfolg meldet. Jedes FSA verfügt über zwei Endzustände. Der eine repräsentiert eine erfolgreiche Durchführung der Aufgabe, der andere den Fall des Misserfolges. Beim Erreichen eines der beiden Zustände erhält die deliberative Ebene die zugehörige Mitteilung und die vermittelnde Ebene initiiert entsprechend die Bearbeitung des nächsten Planungsschrittes.

Sobald ein neuer Planungsschritt die vermittelnde Ebene erreicht, lädt diese den entsprechenden FSA aus einer Aufgabendatenbank und aktiviert die Verhalten des ersten Knotens. Meldet eines der aktiven Verhalten seine erfolgreiche Durchführung bzw. einen Misserfolg beim Erreichen seines Zieles, dann findet eine Transition in einen entsprechenden Zustand des FSAs statt und die zugehörige Teilmenge von Verhalten wird aktiviert. Die Übergänge innerhalb einer Sequenz hängen somit von den Erfolgs- bzw. Misserfolgsmeldungen der Verhalten ab. Eine Erfolgsmeldung entspricht dem Erreichen des erwünschten Endzustands eines Verhaltens, wie beispielsweise die Positionierung des Greifers an der erwünschten Endposition am Objekt. Die Misserfolgsmeldung wird dagegen ausgelöst, sobald ein Verhalten sein fest einprogrammiertes Ziel nicht einhalten oder erreichen kann[2]. Eine Transition kann auch ausgelöst werden, sobald ein interner Timer einen vom jeweiligen Zustand abhängigen Schwellenwert überschreitet. Dieser Timer implementiert einen Deadlockerkennungs-Mechanismus[3] und, wenn aktiviert, erzeugt er eine Misserfolgsmeldung. Gibt ein Verhalten während der Ausführung eines FSAs eine Misserfolgsmeldung zurück, so geht der Akzeptor in den Misserfolgsendzustand über. Dabei wird der Manipulator zu einer sicheren Ausgangsposition zurückgeführt. Ist auch diese Aktion nicht mehr möglich, wird der Arm gestoppt.

Als Beispiel wird hier der Vorgang bei der Ausführung des Planungsschrittes `Greife_Objekt` schematisch dargestellt (Abbildung 5.5 und Tabelle 5.2). Im Anfangszustand des entsprechenden FSA aus Abbildung 5.5 ist zunächst nur das Verhalten der Ziellokalisierung aktiv, das die Bordkamera zur Lokalisierung des Objektes einsetzt. Sobald das Ziel lokalisiert ist, findet aufgrund des Erfolgssignals ein Übergang in den nächsten Zustand statt. Wenn kein Erfolg eintritt, wird ein Timeout ausgelöst und der Misserfolgspfad wird verfolgt. Nach der Detektion werden die Zielführung, das Zielfolgen, die Hindernisvermeidung des Greifers und der Manipulatorsegmente und das Planungsverhalten aktiviert. Das Planungsverhalten bewertet zunächst die Anfangsposition der Manipulatorbasis. Ist ein erfolgversprechender Greifvorgang nicht möglich, dann meldet das Planungsverhalten einen Misserfolg. Ansonsten werden die Ausgaben der Verhalten so koordiniert, dass das Zielobjekt sicher erreicht werden kann. Wenn das zielführende Verhalten mit Erfolg terminiert, hat der Greifer die Zielposition erreicht und kann seine Finger schließen. Nach dem Greifen des Objektes wird das zurückführende Verhalten aktiviert und der Manipulator wieder in seine Ausgangsposition bewegt; die gestellte Aufgabe wird als erfolgreich bewältigt bewertet. Ansonsten wird der deliberativen Ebene eine Misserfolgsmeldung mitgeteilt, die einen alternativen Lösungsweg finden muss.

[2] Dies ist z.B. der Fall, wenn die Zielführung feststellt, dass die Zielposition außerhalb des Greifradius des Manipulators liegt, oder wenn die Hindernisvermeidung eine Kollision nicht verhindern kann.

[3] In der betrachteten Anwendung können dann Deadlocks auftreten, wenn aufgrund mangelnder Sensorinformation keines der aktiven Verhalten einen Pfad berechnen kann, oder wenn mindestens zwei Verhalten sich gegenseitig blockieren.

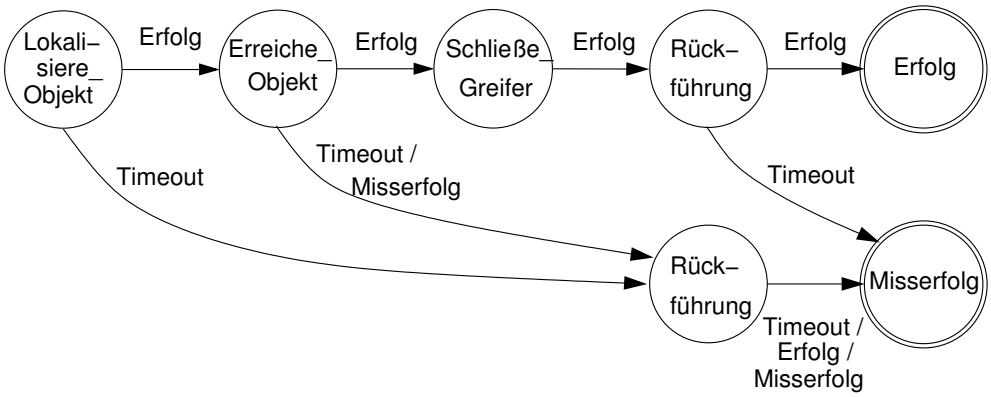


Abbildung 5.5: FSA für den Planungsschritt `Greife_Objekt`.

Zustand	Aktive Verhalten
`Lokalisiere_Objekt`	Ziellokalisierung
`Erreiche_Objekt`	Zielführung Kollisionsvermeidung Greifer Kollisionsvermeidung Segmente Planungsverhalten Zielfolgen
`Schließe_Greifer`	Greifer schließen
`Rückführung`	Zurückführendes Verhalten

Tabelle 5.2: Zustandsmenge $\mathcal{Z}_i$ und aktive Verhalten für den Planungsschritt `Greife_Objekt`.

Der zu jedem Planungsschritt zugehörige FSA ist in einer Aufgabendatenbank gespeichert. Die Einträge der Datenbank[4] spezifizieren die Durchführung des FSAs eindeutig und bestehen aus folgenden Elementen:

- Dem Namen des Planungsschrittes.
- Dem zugehörigen FSA. Alle erlaubten Übergänge sind dabei in der Übergangsfunktion δ enthalten.
- Den Eingabeparametern des Planungsschrittes, die sich in symbolische und numerische Parameter unterteilen lassen. Unter symbolischen Parametern ist beispielsweise die Form oder Farbe von Objekten zu verstehen. Ein numerischer Parameter ist z.B. eine kartesische Position im Raum.
- Den Ausgabeparametern, welche die vermittelnde Ebene an die deliberative Ebene zurückführt.

FSAs werden also zur Aktivierung einer Menge von parallel ausgeführten Verhalten in einer festen zeitlichen Sequenz eingesetzt. Dadurch können sie der deliberativen Ebene komplexe Planungschritte zur Verfügung stellen, die den Suchraum für die Planungsalgorithmen reduzieren und somit eine effiziente, schnelle Planung ermöglichen. FSAs sind jedoch nicht in der Lage, eine Gewichtung der

[4]Tabelle E.1 im Anhang E beinhaltet eine Übersicht über alle implementierten Planungsschritte in der vermittelnden Ebene des Manipulators.

Verhalten in Abhängigkeit des aktuellen Sensorkontexts zu errechnen oder ihre Struktur durch Lernalgorithmen an eine gestellte Aufgabe anzupassen. Im Gegensatz dazu bieten BBNs, die im nächsten Abschnitt beschrieben werden, die Möglichkeit, mit Hilfe von aktuellen Sensorwerten die Anwendbarkeit eines Verhaltens zu bewerten und durch Training zu erlernen.

5.3 Verhaltenskoordination

Nach Auswahl und Aktivierung der Verhalten sind deren Ausgaben nach dem *Command Fusion*-Prinzip zu koordinieren. Hierzu ist die Berechnung der Gewichtungsmatrix $\mathbf{G}$ (Gleichung 2.16) notwendig. $\mathbf{G}$ beschreibt, mit welchem Anteil eine Fertigkeit an der Bewegung des Roboters beteiligt ist. Dies entspricht der Bestimmung der so genannten Anwendbarkeit der Verhalten in einer Situation. Zu ihrer Berechnung werden in dieser Arbeit *Bayesian Belief Networks* (BBN) eingesetzt [Cha91]. BBNs sind zur Realisierung des Koordinationsmechanismus ausgewählt worden, da sie leicht um neue Knoten ergänzt werden können. Wird der Merkmalsvektor um ein zusätzliches Element, eine zusätzliche Beobachtung oder sogar einen neuen Sensor erweitert, kann einem bestehenden BBN ein entsprechender Knoten hinzugefügt werden. Die bisherige Struktur bleibt dabei unverändert bestehen. Desweiteren existieren für BBNs lernfähige Algorithmen. Sie erlauben es, die Anwendbarkeit eines Verhaltens in einer bestimmten Situation durch Erfahrung zu erlernen und nicht manuell zu bestimmen. möglich, insbesondere wenn mehrere erlernte Roboterfertigkeiten koordiniert werden müssen. BBNs erlauben unterschiedlich zu anderen Verfahren die robuste Ermittlung von Entscheidungen, auch wenn nicht alle Elemente des Merkmalsvektors vorhanden sind. In einem solchen Fall kann der Wert des betroffenen Knotens nicht angegeben werden. Dafür kommt an dessen Stelle im BBN bei der Inferenz eine a priori Wahrscheinlichkeit zum Einsatz.

In den nächsten Abschnitten wird zuerst kurz auf die theoretischen Grundlagen von BBNs und auf die Besonderheiten bei ihrem Entwurf für die Verhaltenskoordinierung eingegangen. Als Beispiel wird das BBN der Kollisionsvermeidung des Greifers vorgestellt und erläutert[5]. Anschließend wird der eigentliche Ablauf der Verhaltenskoordination präsentiert.

5.3.1 Bayesian Belief Networks

Ein *Bayesian Belief Network* (BBN) ist ein gerichteter azyklischer Graph (*Directed Acyclic Graph* - DAG), der die Verbundwahrscheinlichkeit eines Ereignisses, das von mehreren Zufallsvariablen abhängt, effizient repräsentiert [Cha91]. Bestandteile eines BBN sind Knoten, die Zufallsvariablen darstellen, und gerichtete Kanten, die Abhängigkeiten zwischen den Knoten modellieren. Die Zufallsvariablen können sowohl unendlich viele Werte mit einer kontinuierlichen Wahrscheinlichkeitsverteilung oder eine endliche Anzahl von Zuständen mit einer diskreten Wahrscheinlichkeitsverteilung annehmen. Häufig wird auch der Sonderfall eines boolschen Knotens mit zwei diskreten Zuständen verwendet.

Eine Zufallsvariable hängt von den Zuständen ihrer Vorgängerknoten im Graphen ab. Besitzt sie keinen Vorgänger, ist sie stochastisch unabhängig. Die zugehörige bedingte Wahrscheinlichkeitsverteilung wird im kontinuierlichen Fall als Funktion und im diskreten in der so genannten *Conditional Probability Table* (*cpt*) angegeben. Diese enthält für jede mögliche Kombination der Zustände der

[5] Die BBNs der restlichen Verhalten werden in Anhang E präsentiert.

Vorgängerknoten die entsprechenden Wahrscheinlichkeiten für die Zustände des betreffenden Knotens. Damit lässt sich nach Heckerman [Hec95] ein BBN für eine Menge von Variablen $\{Z_1, \ldots, Z_n\}$ definieren durch (Abbildung 5.6):

- eine Netzwerkstruktur, welche die Abhängigkeiten zwischen den Zufallsvariablen beschreibt, und
- eine Menge von bedingten Wahrscheinlichkeiten, welches einer cpt pro Knoten entspricht.

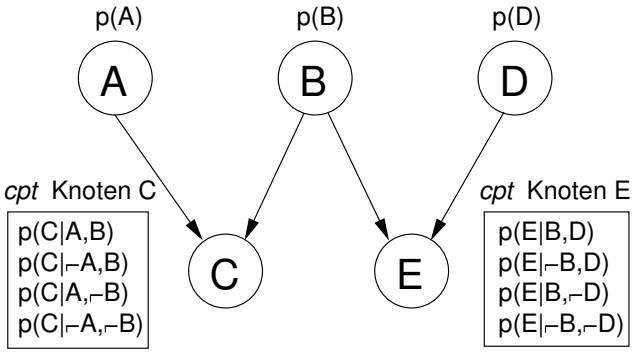


Abbildung 5.6: Beispiel eines Bayesian Belief Networks

Im Allgemeinen wird die Wahrscheinlichkeit für ein Verbundereignis mit der Kettenregel berechnet:

$$p(Z_1, Z_2, \ldots, Z_k) = p(Z_1) \prod_{i=2}^{k} p(Z_i | Z_{i-1}, \ldots, Z_1) \tag{5.1}$$

Unter der Berücksichtigung, dass die bedingte Wahrscheinlichkeit eines Knotens nur von den Wahrscheinlichkeiten seiner direkten Vorgänger $\mathcal{P}(Z_i)$ abhängt, wobei $\mathcal{P}(Z_i)$ die Menge der Vorgänger von Knoten i darstellt, ergibt sich eine vereinfachte Bestimmung der Wahrscheinlichkeit:

$$p(Z_1, Z_2, \ldots, Z_k) = \prod_{i=1}^{k} p(Z_i | \mathcal{P}(Z_i)) \tag{5.2}$$

Die Werte von $p(Z_i|\mathcal{P}(Z_i))$ sind in der cpt vom i-ten Knoten gegeben. Durch die vereinfachte Bestimmung der Wahrscheinlichkeit reduziert sich die Größe der zugehörigen cpt und damit ergibt sich eine erhebliche Reduktion in der notwendigen Berechnung. Hat ein Knoten keinen direkten Vorgänger, ist also die zugehörige Zufallsvariable stochastisch unabhängig, wird anstatt einer cpt nur eine a priori Wahrscheinlichkeit angegeben:

$$p(Z_i | \mathcal{P}(Z_i)) = p(Z_i)$$

Unter Inferenz in einem BBN wird das Schließen unter Zuhilfenahme von a priori Wissen über die Zustände einiger Knoten auf die Wahrscheinlichkeit eines Zustands eines anderen Knotens verstanden. Der gesuchte Knoten ist im Allgemeinen nicht direkt beobachtbar, über ihn liegt deshalb kein direktes Wissen vor. Es ist aber für eine Teilmenge der Knoten eines BBN der Wert der zugehörigen Zufallsvariable bekannt oder kann wenigstens mit einer Wahrscheinlichkeit (*Likelihood*) angegeben werden.

Nach Charniak [Cha91] sind die bekannten Inferenzalgorithmen in exakte (*exact*) und approximierte[6] (*approximate*) Inferenz zu unterteilen. Bei der exakten Inferenz wird nach der Richtung des Schließens zwischen *Top-down*-Inferenz und *Bottom-up*-Inferenz unterschieden. *Top-down*- oder kausale Inferenz liegt vor, wenn die Werte der Vorgänger eines Knotens bekannt sind und eine Wahrscheinlichkeit für den Knoten selbst gesucht ist. In diesem Sinne verursachen die Werte der Vorgängerknoten die derartige Aktivierung des Verhaltensknotens.

Die *Bottom-up*-Inferenz geht davon aus, dass der Zustand eines Knotens bekannt ist und eine Wahrscheinlichkeit für einen Vorgänger gesucht wird. Da hierbei die Auswirkung bekannt ist und der Grund hierfür gesucht wird, spricht man auch von diagnostischer Inferenz. Die gesuchte bedingte Wahrscheinlichkeit wird mit dem Bayes-Theorem umgeformt, weil bei der Angabe einer cpt die Kenntnis des Zustandes der Vorgängerknoten vorausgesetzt wird.

5.3.2 Aufbau und Topologie der BBNs der vermittelnden Ebene

Die mit der Greifer- und der Bordkamera extrahierten Merkmale[7] werden auf kontinuierliche Knoten in den BBNs abgebildet. Ein zusätzlicher Knoten pro BBN repräsentiert das jeweilige Verhalten und seine Beziehung zu den Merkmalen (Abbildung 5.7). Dieser Knoten beschreibt die Anwendbarkeit des Verhaltens in der von den Merkmalen beschriebenen Situation. Durch Inferenz wird von den beobachteten Merkmalsknoten auf den Zustand des unbekannten Verhaltensknoten geschlossen. Nach der Inferenz ergibt sich für den Verhaltensknoten eine Wahrscheinlichkeitsverteilung über den Wertebereich von Null bis Eins. Die Bestimmung dieses Erwartungswertes beschreibt, ob eine Anwendung des i-ten Verhaltens in der beobachteten Situation vorteilhaft ist, und wird als Gewichtung des Verhaltens in $\mathbf{G}$ verwendet.

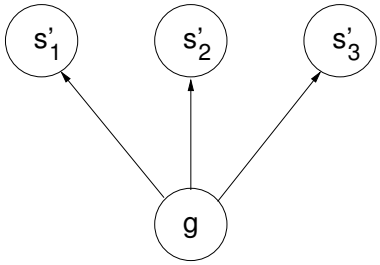


Abbildung 5.7: Beispiel eines BBNs mit drei Merkmalsknoten und einem Verhaltensknoten.

Zwei verschiedene Typen von Abhängigkeiten zwischen Merkmals- und Verhaltensknoten sind möglich:

- Die Abhängigkeit führt von den Merkmalsknoten zu den Verhaltensknoten, d.h. der Verhaltensknoten ist bedingt abhängig von den Merkmalen. Es wird also eine bestimmte Merkmalskombination festgestellt und aufgrund dieser wird die Anwendbarkeit des Verhaltens bestimmt (kausale Inferenz).

[6] Da die approximierte Inferenz in dieser Arbeit nicht eingesetzt wird, wird sie hier nicht weiter erläutert.

[7] Die in der vermittelnden Ebene verwendeten Merkmale sind in Anhang E aufgelistet.

- Die Abhängigkeit führt von den Verhaltensknoten zu den Merkmalsknoten. Hierbei wird davon ausgegangen, dass eine Anwendbarkeit des Verhaltens vorliegt und aus dieser Aktivierung eine bestimmte Merkmalskombination folgt. Da die Werte der Merkmalsknoten vorliegen, kann man über die so genannte diagnostische Inferenz auf die Anwendbarkeit des Verhaltensknotens schließen [Cha91].

Beide Darstellungen sind, was die Inferenz betrifft, ineinander überführbar. Aus mehreren Gründen wurde hier die zweite Möglichkeit realisiert:

- Die nötigen $cpts$ sind für die in dieser Arbeit implementierten BBNs im Fall der diagnostischen Inferenz wesentlich kleiner. Bei beispielsweise vier Merkmalen mit jeweils vier Zuständen und eine Anwendbarkeit mit zehn Zuständen ergibt sich bei der ersten Art eine cpt mit $4^4 \cdot 10 = 2560$ Einträgen für den Verhaltensknoten und vier $cpts$ mit vier Einträgen für die Merkmalsknoten. Bei der zweiten Art werden für den Verhaltensknoten zehn und für die vier Merkmalsknote jeweils $4 \cdot 10 = 40$ Einträge benötigt.
- Aufgrund der kleineren $cpts$ kann das Training des BBNs effizienter und mit einer schnelleren Konvergenz erfolgen.
- Das später vorgestellte, für beide Varianten durchgeführte Training zeigt einen geringeren Fehler bei Abhängigkeiten, die von den Verhaltensknoten zu den Merkmalsknoten führen. Dies resultiert zu einer besseren Berechnung der Gewichtung eines Verhaltens.

5.3.3 Bestimmung der Wertbereiche der Eingaben und Ausgaben der BBNs

Da die verwendeten Inferenz- und Lernalgorithmen nur für Netze mit rein diskreten Knoten einsetzbar sind, müssen die kontinuierlichen Einträge des Merkmalsvektors $\vec{s}_i'$, der als Eingabe für das BBN vom i-ten Verhalten dient, mit Hilfe von Schwellwerten k_j in diskrete Intervalle eingeteilt werden. Alle Werte innerhalb eines solchen Intervalls $(k_j,\ k_{j+1})$ werden als gleichwertig angesehen, da sie dem gleichen diskreten Zustand Z_j entsprechen. Um auch mit diskreten Merkmalswerten eine ausreichende Genauigkeit zu erzielen,wird die Anzahl der Intervalle hoch gewählt. Außerdem werden nicht äquidistante Abstände, also $k_j - k_{j-1} \neq k_{j+1} - k_j$, verwendet, so dass in kritischen Bereichen die Auflösung höher ist.

Neben den Eingabenknoten besitzt auch der die jeweilige Anwendbarkeit des Verhaltens repräsentierende Knoten einen kontinuierlichen Wertebereich. Dieser Bereich wird in zehn diskrete Intervalle bzw. Zustände der Länge 0.1 von 0 bis 1 unterteilt. Nach der Inferenz ergeben sich zu 1 summierende Wahrscheinlichkeitsangaben für alle Zustände des Verhaltensknotens. Die Gewichtung des i-ten Verhaltens errechnet sich dann aus der mit der entsprechenden Wahrscheinlichkeitsangabe gewichteten Summe der Zustandsmittelwerte. Diese Vorgehensweise entspricht der Maximum-Defuzzifizierungsmethode bei der Fuzzy Logik:

$$g_i = p(\beta_i(\vec{s}_i)|\vec{s}_i') = \sum_{j=1}^{10} p(Z = Z_j)\frac{k_j + k_{j+1}}{2} \tag{5.3}$$

Experimente haben gezeigt, dass dies zu einer ausreichenden Genauigkeit in der Berechnung der Anwendbarkeit des Verhaltens führt [Sch02].

5.3.4 Modellierung der Unsicherheit von Merkmalswerten

Die Elemente der Merkmalsvektoren können nicht immer exakt bestimmt werden, sondern unterliegen Rauschen bzw. Unsicherheiten wegen Überlappungen oder Schattenbildung in den Aufnahmen. Dadurch sind die genauen Werte der Merkmale nicht ermittelbar; sie können nur mit einem ungenauen Mittelwert und einer zugehörigen Standardabweichung angegeben werden. Es ist wünschenswert, diese Unsicherheit auch bei der Inferenz mitwirken zu lassen. So würde normalerweise bei der Eintragung eines Merkmalswertes derjenige Zustand des Merkmalsknotens als beobachtet ausgewählt werden, dessen zugehöriges Intervall den Merkmalswert enthält. Liegt jedoch der beobachtete Wert nahe an einer Intervallsgrenze, ist es sinnvoller, beide Intervalle mit einer entsprechenden Wahrscheinlichkeitsangabe als beobachtet auszuwählen. Dadurch wird die Ungenauigkeit der Merkmalsextraktion bei der Inferenz berücksichtigt und beide Möglichkeiten werden bei der Entscheidungsfindung einbezogen.

Aus diesem Grund wird eine gaußsche Wahrscheinlichkeitsverteilung für die Elemente des Merkmalsvektors mit dem als Messwert gegebenen Mittelwert μ und einem empirisch bestimmten Standardabweichungswert σ verwendet. Die Wahrscheinlichkeitsangaben für die einzelnen Intervalle werden aus der durch die jeweiligen Intervallsgrenzen eingeschlossenen Fläche unter der gaußschen Wahrscheinlichkeitsdichtefunktion errechnet. Mit der zugehörigen Verteilungsfunktion $\Phi(x, \mu, \sigma)$ und den Intervallgrenzen $(k_1, ..., k_{n+1})$ für n diskrete Intervalle ergibt sich der Wert der Wahrscheinlichkeitsangabe für einen Zustand Z_j des Knotens wie folgt:

$$p(Z = Z_j) = \Phi(k_{j+1}, \mu, \sigma) - \Phi(k_j, \mu, \sigma) \tag{5.4}$$

5.3.5 BBN der Hindernisvermeidung des Greifers

Situationen, in denen die Hindernisvermeidung angewandt wird, zeichnen sich durch die Abstände des Greifers zu den möglichen Hindernissen sowie durch den Winkel des Bewegungsvektors des Greifers zu den Richtungsvektoren der Hindernisse aus. Kollisionsgefahr besteht beispielsweise bei einem geringen Abstand d_{GH} vom Greifer zum nächsten Hindernis. Weiterhin weist ein in Richtung des Hindernisses deutender Bewegungsvektor des Greifers auf eine potentielle Kollision hin; je kleiner dann der relative Winkel ϕ_{GH} des Bewegungsvektors zum Positionsvektor des Hindernisses im System des Greifers ist, desto größer ist die Kollisionsgefahr. Eine ähnliche Bedeutung hat die Änderung des Abstandes zwischen Greifer und Hindernis $\dot{d}_{GZ}$. Ist sie negativ, bewegt sich der Manipulator auf ein Hindernis zu; ein positiver Wert kennzeichnet einen vom Hindernis abgewandten Pfad. Entsprechende Merkmale werden zur Bestimmung der Kollisionsgefahr mit der Auflageebene benutzt. Zusätzlich zu den beschriebenen Merkmalen ist hier auch der Abstand des Greifers zum Ziel entscheidend. Befindet sich der Greifer nah am Zielobjekt, dann darf trotz geringem Abstand zur Auflageebene keine Hindernisvermeidung angewandt werden.

Der Merkmalsvektor $\vec{s}'_{HG}$ am Eingang beinhaltet Einträge, die die obigen Beziehungen zum Ausdruck bringen:

$$\vec{s}'_{HG} = (d_{GZ}, d_{GH}, \dot{d}_{GH}, \phi_{GH}, d_{GT}, \phi_{GT})^T \tag{5.5}$$

Die Elemente von $\vec{s}'_{HG}$ sowie die zugehörigen Diskretisierungsintervalle sind in Tabelle 5.3 aufgeführt.

Durch eine nicht äquidistante Einteilung der Diskretisierungsintervalle und insbesondere durch die höhere Auflösung der kritischen Bereiche in der Nähe der Hindernisse wird eine ausreichende Ge-

Abkürzung	Beschreibung	Diskretisierungsintervalle
KV	Anwendbarkeit der Kollisionsvermeidung	{ 0, 0.1, 0.2,..., 1.0}
d_{GH}	Abstand des Greifers zum Hindernis	{ 0, 20, 50, 90, 1000} in mm
$\dot{d}_{GH}$	Änderung des Abstandes Greifer - Hindernis	{ -300, -10, 0, 10, 300} in mm
ϕ_{GH}	Winkel Greiferbewegung - Hindernis	{ 0, 10, 30, 60, 180} in Grad
d_{GT}	Abstand des Greifers zur Auflagefläche	{ 0, 20, 50, 90, 1000} in mm
d_{GZ}	Abstand des Greifers zum Ziel	{ 0, 20, 50, 90, 1000} in mm
ϕ_{GT}	Winkel Greiferbewegung - Auflagefläche	{ 0, 10, 30, 60, 180} in Grad

Tabelle 5.3: Beschreibung der Merkmale und der Diskretisierungsintervalle des BBNs für die Hindernisvermeidung des Greifers

nauigkeit in der Berechnung der Anwendbarkeit erreicht. Dies betrifft vor allem den Abstand zu den Hindernissen und der Auflagefläche, so dass eine kleine Distanz, gleichbedeutend mit einer hohen Kollisionsgefahr, besser aufgelöst ist.

Der Graph des BBNs, dargestellt in Abbildung 5.8, kodiert die Richtlinien des Einsatzes des Verhaltens entsprechend der Situation, die vom Merkmalsvektor $\vec{s}'_{HG}$ beschrieben wird. Die Kollisionsvermeidung kommt in zwei voneinander unabhängigen Situationen zum Einsatz, einerseits bei Kollisionsgefahr mit einem Hindernis (die drei linken Knoten des BBN in Abbildung 5.8) und anderseits bei Kollisionsgefahr mit der Auflageebene (die drei rechten Knoten).

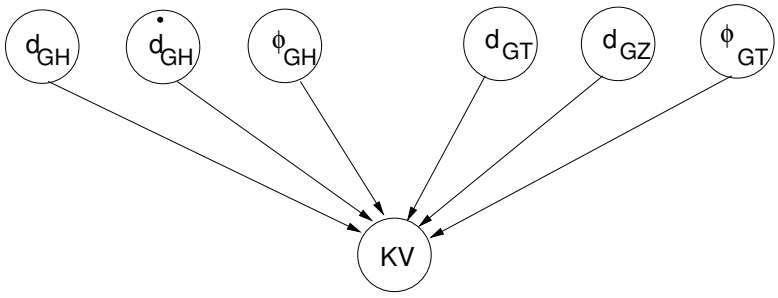


Abbildung 5.8: Der Graph des BBN für die Kollisionsvermeidung des Greifers.

5.3.6 Ablauf der Verhaltenskoordination

Der genaue Ablauf der Verhaltenskoordination ist in Abbildung 5.9 dargestellt. Jedes Verhalten erhält sein eigenes, von den anderen Verhalten unabhängiges BBN. Da jedem Verhalten ein BBN zugeordnet ist, wird bei der Implementierung einer neuen Fertigkeit lediglich ein neues BBN erzeugt. Die BBNs der vorhandenen Verhalten werden dabei ohne Änderungen übernommen. Auf diese Weise entfällt ein komplett neues Training bzw. ein Umstrukturieren der alten Komponenten der vermittelnden Ebene und die Anwendbarkeit jeder Fertigkeit ist getrennt erlernbar.

Die mit der Greifer- und der Bordkamera extrahierten Merkmale kommen als Eingang der BBNs zum Einsatz. Die BBNs berechnen dann einen Erwartungswert, die so genannte Anwendbarkeit, für die entsprechenden Verhalten. Diese Berechnung des Erwartungswertes für jedes Verhalten entspricht

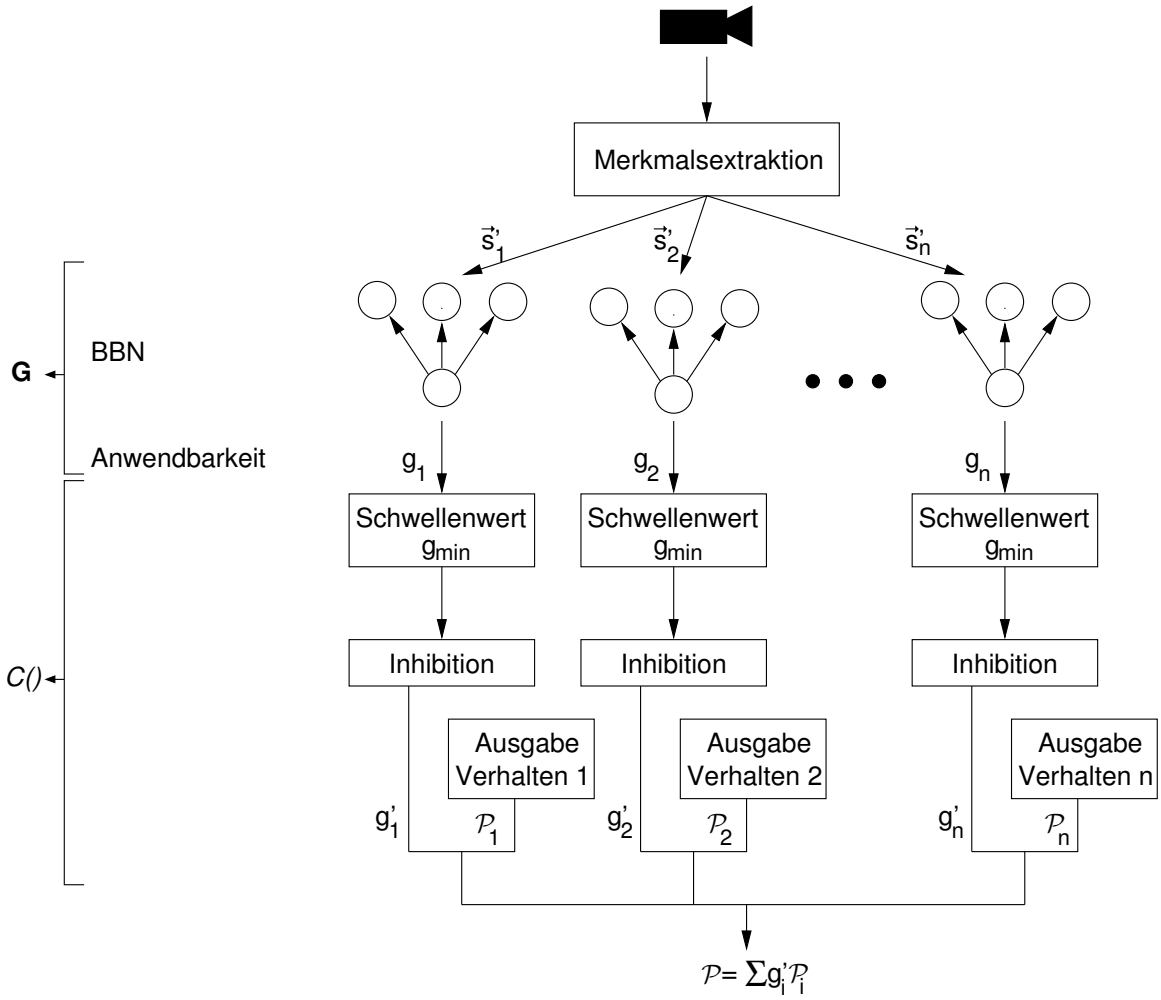


Abbildung 5.9: Ablauf der Verhaltenskoordination.

dem Aufstellen der Gewichtungsmatrix $\mathbf{G}$. Somit realisieren die BBNs eine Abbildung des Raumes der extrahierten Merkmale $\mathcal{S}'$ auf der Matrix $\mathbf{G}$: $\mathcal{S}' \mapsto \mathbf{G}$. Die Matrix $\mathbf{G}$ ist dann:

$$\mathbf{G} = \begin{pmatrix} p(\beta_1(\vec{s}_1)|\vec{s}'_1) & 0 & \dots & 0 \\ 0 & p(\beta_2(\vec{s}_2)|\vec{s}'_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p(\beta_n(\vec{s}_n)|\vec{s}'_n) \end{pmatrix} = \begin{pmatrix} g_1 & 0 & \dots & 0 \\ 0 & g_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & g_n \end{pmatrix} \tag{5.6}$$

wobei $\beta_i(\vec{s}_i)$ die Ausgabe von Verhalten i ist, $\vec{s}'_i$ der Merkmalsvektor und $p(\beta_i(\vec{s}_i)|\vec{s}'_i)$ die Ausgabe des BBNs von Verhalten i.

Aufgrund von Rauschen wird der Zustand der Verhaltensknoten nie genau Null. Um den Einfluss des Rauschens zu reduzieren, wird ein Schwellenwert g_{min} auf die Elemente von $\mathbf{G}$ angewandt. Zusätzlich löst ein Inhibitionsmechanismus Konflikte zwischen zwei sich gegenseitig ausschließenden Verhalten, indem er das Verhalten mit der niedrigeren Priorität deaktiviert. Diese Beziehungen zwischen den Verhalten lassen sich in einer quadratischen Matrix $\mathbf{H}$ zusammenfassen [SB00]. Sei h_{ij} das Element der i-ten Reihe und j-ten Spalte von $\mathbf{H}$, dann:

$$h_{ij} = \begin{cases} 1 & \text{Verhalten } j \text{ unterdrückt Verhalten } i \text{ mit höherer Priorität, falls } g_j \neq 0. \\ 0 & \text{Inhibition findet nicht statt.} \end{cases} \tag{5.7}$$

Ein aktives Verhalten wird also unterdrückt, falls

$$\sum_{j=1}^{n} h_{ij} \neq 0 \implies g_i = 0 \tag{5.8}$$

Dadurch entsteht die Matrix $\mathbf{G}'$ als:

$$\mathbf{G}' : g'_i = \begin{cases} 0, & g_i < g_{\min} \vee \sum_{j=1}^{n} h_{ij} \neq 0 \\ g_i, & g_i \geq g_{\min} \wedge \sum_{j=1}^{n} h_{ij} = 0 \end{cases} \tag{5.9}$$

Die auszuführende Gesamtaktion $\vec{\rho}$ ist die gewichtete Summe der Ausgaben aller Verhalten, deren Anwendbarkeit g'_i größer Null ist. Seien die Ausgabevektoren der Verhalten in einem Vektor $\vec{R}$ zusammengefasst. Aus Gleichungen 2.16 und 5.9 resultiert für $\vec{\rho}$:

$$\vec{\rho} = C(\mathbf{G} \cdot \vec{R}) = \sum_{i=1}^{n} (\frac{1}{det(\mathbf{G}')} \mathbf{G}' \cdot \vec{R})_{i(\cdot)} \tag{5.10}$$

wobei $()_{i(\cdot)}$ die i-Reihe der Matrix $\frac{1}{det(\mathbf{G}')} \mathbf{G}' \cdot \vec{R}$ repräsentiert und $det(\mathbf{G}')$ die Determinante von Matrix $\mathbf{G}'$.

In dieser Arbeit sind die Ausgaben der Verhalten Pfade, da sie eine asynchrone Ausführung der Fertigkeiten erlauben sowie keine hohe Ausführungsfrequenz der Verhalten und des Koordinationsmechanismus erfordern. Bei der Bestimmung des resultierenden Pfades wird die gewichtete Summe über alle vorgeschlagenen Pfade der aktiven Verhalten nach Gleichung 5.10 gebildet. Sei der Pfad vom i-ten Verhalten mit einer Sequenz von kartesischen Stützstellen $\vec{\mathcal{P}}_{il}$, $l = 1, \ldots, k$ repräsentiert. Dabei fasst der 6×1 Vektor $\vec{\mathcal{P}}_{il}$ sowohl die Position als auch die Orientierung des Greifers an der Stützstelle des Pfades zusammen. Seien zum Zeitpunkt t_l die Ausgaben aller Verhalten in $\vec{R}_l$ zusammengefasst (s.a. Gleichung 2.15):

$$\vec{R}_l = (\vec{\mathcal{P}}_{1l}, \vec{\mathcal{P}}_{2l}, \ldots, \vec{\mathcal{P}}_{nl})^T$$

Dann kann der resultierende Pfad zur Zielposition nach Gleichung 5.10 wie folgt geschrieben werden:

$$\mathcal{P} = \{\vec{\mathcal{P}}_1, \ldots, \vec{\mathcal{P}}_k\} : \vec{\mathcal{P}}_l = \sum_{i=1}^{n} (\frac{1}{det(\mathbf{G}')} \mathbf{G}' \cdot \vec{R}_l)_{i(\cdot)} \tag{5.11}$$

Falls manche Verhalten Pfade im kartesischen Raum und andere Verhalten Pfade im Gelenkraum[8] zurückliefern, dann muss die Bestimmung des Gesamtpfades in zwei Phasen verlaufen. Sei $\mathbf{G}'_K$ die Matrix, die die Gewichte für die kartesischen Pfade beinhaltet, und $\mathbf{G}'_G$ die entsprechende Gewichtungsmatrix für die mit Gelenkstellungen definierten Pfade, mit :

$$\mathbf{G}' = \mathbf{G}'_K + \mathbf{G}'_G \tag{5.12}$$

Zuerst werden alle im kartesischen Raum beschriebenen Pfade gewichtet interpoliert:

$$\mathcal{P}_K = \{\vec{\mathcal{P}}_{K_1}, \ldots, \vec{\mathcal{P}}_{K_k}\} : \vec{\mathcal{P}}_{K_l} = \sum_{i=1}^{n} (\frac{1}{det(\mathbf{G}'_K)} \mathbf{G}'_K \cdot \vec{R}_l)_{i(\cdot)}, \quad l = 1, ..., k \tag{5.13}$$

[8] Der Pfad ist dann mit einer Reihenfolge von Gelenkstellungen dargestellt.

Der resultierende kartesische Pfad wird mit Hilfe der inversen Kinematik in Gelenkstellungen transformiert. Hierbei wird die zugehörige Gelenkstellung für jede kartesische Stützstelle des Pfades berechnet und auf Erreichbarkeit durch den Manipulator überprüft (siehe Anhang C.1).

Entsprechend werden die im Gelenkraum definierten Pfade interpoliert:

$$\mathcal{P}_G = \{\vec{\mathcal{P}}_{G_1}, \ldots, \vec{\mathcal{P}}_{G_k}\} : \vec{\mathcal{P}}_{G_l} = \sum_{i=1}^{n} (\frac{1}{det(\mathbf{G}'_G)} \mathbf{G}'_G \cdot \vec{R}_l)_{i(\cdot)}, \quad l = 1, ..., k \tag{5.14}$$

und anschließend werden die Pfade aus Gleichungen 5.13 und 5.14 im Gelenkraum gewichtet addiert:

$$\mathcal{P} = \{\vec{\mathcal{P}}_1, \ldots, \vec{\mathcal{P}}_k\} : \vec{\mathcal{P}}_l = (\frac{1}{det(\mathbf{G}'_K)} \sum_{i=1,j=1}^{n,n} g'_{K_{ij}})\vec{\mathcal{P}}'_{K_l} + (\frac{1}{det(\mathbf{G}'_G)} \sum_{i=1,j=1}^{n,n} g'_{G_{ij}})\vec{\mathcal{P}}_{G_l} \tag{5.15}$$

wobei n die Dimension von Matrix $\mathbf{G}$ ist, $g'_{K_{ij}}$ das Element der i-ten Zeile und j-ten Spalte von $\mathbf{G}'_K$, $g'_{G_{ij}}$ das Element der i-ten Zeile und j-ten Spalte von $\mathbf{G}'_G$ und $\vec{\mathcal{P}}'_{K_l}$ die Gelenkstellungen der kartesischen Stützstellen $\vec{\mathcal{P}}_{K_l}$ aus Gleichung 5.13. Der resultierende Pfad $\mathcal{P}$ wird dann mit einem Trajektorieregler ausgeführt und die Verhaltenskoordinierung erneut mit den aktuellen Sensordaten aufgerufen, bis ein Verhalten seine erfolgreiche Durchführung meldet oder mit Misserfolg endet.

5.4 Erlernen der Verhaltenskoordination

Mit dem Training werden die Werte der Merkmale, die ein bestimmtes Verhalten befürworten, in der virtuellen Umgebung erlernt. Damit entspricht das Trainieren der BBNs dem Erlernen, wann und zu welchem Maße ein Verhalten ausgeführt werden soll. Im Gegensatz dazu wird beim Training eines Verhaltens erlernt, wie ein Verhalten auszuführen ist. Es gibt mehrere Gründe, die für das Lernen der Verhaltenskoordinierung sprechen:

- Im Allgemeinen ist nicht vorhersagbar, welche Ausgabe ein Verhalten in einer Situation erzeugt und welche Qualität diese Ausgabe hat, d.h. wie leistungsfähig das Verhalten in Bezug auf das angestrebte Ziel ist. Deshalb ist die Anwendbarkeit in einer bestimmten Situation nur erlernbar und nicht durch harte Grenzen im Voraus auszudrücken.
- Die Bewertung, ob ein Verhalten in einer bestimmten Situation sinnvoll ist oder nicht, wird auch von der aktuellen Bewegung des Manipulators beeinflusst. Diese Wirkung der aktuellen Bewegungsrichtung kann durch Erfahrung gelernt werden.
- Durch das Training ist die vermittelnde Ebene an wechselnde Einsatzumgebungen anpassbar. Die Verhaltenskoordinierung wird entsprechend der gestellten Aufgabe und den zur Verfügung stehenden Informationen über die Umwelt weitgehend optimiert. Manuelles Setzen der Parameter erfordert dagegen aufwendiges Experimentieren, damit der Roboter sinnvolle Aktionen für mehrere Situationen erzeugen kann.
- Das Training der Verhaltenskoordinierung erlaubt die nachträgliche Einbeziehung neuer Merkmale in die Bestimmung der Anwendbarkeit eines Verhaltens. Die Auswirkung der neuen Merkmale werden hinzugelernt, ohne dass man die bereits trainierten Abhängigkeiten der alten Merkmale ändern muss.
- Das System ist entsprechend leicht mit neuen Verhalten und neuen Aufgaben erweiterbar.

Im Allgemeinen ist die optimale Gewichtung der Verhalten in jeder Situation nicht vorab bekannt, so dass ein überwachtes Lernverfahren nicht einsetzbar ist. Man kann jedoch das Ergebnis der Anwendung eines Verhaltens in einer Situation, ob sich also der Zustand des Aktuators in Hinblick auf das Aufgabenziel verbessert oder verschlechtert hat, leicht bewerten. Deswegen gehört nach Mataric [Mat94] das Erlernen der Verhaltenskoordinierung zur Kategorie des Lernens mit Bewerter (*Reinforcement Learning*, RL) (siehe auch Abschnitt 3.3.2). Beim RL wird nach der Ausführung eines Verhaltens eine Belohnung für einen Erfolg vergeben und damit die Handlungsstrategie so geändert, dass in einer vergleichbaren Situation ähnlich verfahren wird. Erhält der Roboter jedoch im Fall eines Misserfolges eine Bestrafung, wird der Entscheidungsalgorithmus derart angepasst, dass das Verhalten in ähnlichen Situationen nicht mehr aktiviert wird. In diesem Rahmen beschreiben die BBNs die zu erlernende Handlungsstrategie, welche eine Abbildung aus dem Zustandsraum, also dem Raum der Merkmale $\mathcal{S}'$, in den Aktionsraum, d.h. die Aufstellung der Gewichtungsmatrix $\mathbf{G}$, darstellt. Dafür werden die $cpts$ der Knoten anhand der vergebenen Belohnung angepasst.

Die Belohnungsfunktion bewertet, ob das jeweilige Verhalten innerhalb des Betrachtungszeitraums erfolgreich war. Sie ist verhaltensspezifisch und kann entweder eine *Event-driven*[9] oder eine *Progress-estimator* Funktion[10] sein. Ihr Wert hängt von der Situation ab, die sich nach der Ausführung eines Verhaltens ergibt.

Das *Credit Assignment Problem* wird gelöst, indem bei jedem Lernvorgang immer nur die Anwendbarkeit eines einzelnen Verhaltens erlernt wird. Dies bedeutet, dass man bei jedem Lernschritt nur das BBN, das die zu erlernende Anwendbarkeit des betreffenden Verhaltens modelliert, angepasst. Andere Verhalten können gleichzeitig aktiv sein, sie dürfen jedoch nicht eine ähnliche Zielsetzung wie das zu erlernende Verhalten haben. Damit kann eine verhaltensspezifische Belohnung vergeben werden, die nur den Erfolg des Verhaltens, dessen Anwendbarkeit erlernt wird, bewertet.

Als Beispiel wird hier der Trainingsvorgang bei der Hindernisvermeidung des Greifers in der virtuellen Umgebung vorgestellt. Dabei sind alle zum Greifen notwendigen Verhalten, also das Planungsverhalten, die Zielführung, das Zielfolgen und die Hindernisvermeidung des Greifers aktiv. Die Trainingsdaten, also die Merkmals- und Positionsvektoren sowie die Bewertung des zu trainierenden Verhaltens, werden mit dem stochastischen Teach-In in der virtuellen Umgebung generiert und akquiriert (s.a. Abschnitt 3.3.4).

Die Bewertung für die Anwendbarkeit der Hindernisvermeidung zum Zeitpunkt t wird mit einer *Event-driven* Belohnungsfunktion gebildet:

$$R_{evKG,t} = \begin{cases} -r_{kol} & d_{GH,t+1} < k_{RLKG1} \vee d_{GT,t+1} < k_{RLKG1} \\ -r_{dist} & d_{GH,t+1} > k_{RLKG2} \wedge d_{GT,t+1} > k_{RLKG2} \\ 0 & \text{sonst.} \end{cases} \tag{5.16}$$

Fällt nach der Ausführung die Distanz zu einem Hindernis $d_{GH,t+1}$ oder zur Auflageebene $d_{GT,t+1}$ unter einen festen Mindestabstand k_{RLKG1}, dann wird dies als Kollisionsgefahr interpretiert und eine entsprechende negative Bewertung $-r_{kol}$ erzeugt. Ist die Distanz größer als ein fester, maximal nötiger Abstand k_{RLKG2}, dann produziert der Bewerter erneut ein negatives *Reinforcement Signal* $-r_{dist}$, da in diesem Fall das Verhalten die Gesamtbewegung des Manipulators zuviel beeinflusst und den Greifvorgang behindert. Dabei ist die Bewertung der Kollisionsgefahr betragsmäßig höher als die für einen zu großen Abstand. Auf diese Weise wird die Priorität, welche die Kollisionsfreiheit gegenüber einem kurzen Pfad zum Ziel hat, ausgedrückt.

[9] Beispielsweise bei der Hindernisvermeidung.

[10] Dies ist z.B. bei der Zielführung der Fall.

Nach dem erfolgreichen Abschluss eines Greifvorgangs bzw. nach einer Kollision startet der eigentliche Lernprozess. Für jeden Zeitpunkt t an dem ein Merkmalsvektor mit entsprechender Belohnung gespeichert wurde, findet eine Anpassung des BBNs statt. Betrachtet wird dabei ein Fenster von Belohnungen $R_{evKG,t+1}$, $R_{evKG,t+2}$, $R_{evKG,t+3}$ der Länge 3. Neben der unmittelbar folgenden Belohnung fließen also auch die sich in zukünftigen Situationen ergebenden Bewertungen $R_{evKG,t+2}$ und $R_{evKG,t+3}$ in die Gesamtbewertung ein. Dies berücksichtigt den Fall, dass bei unmittelbarer Nähe zu einem Hindernis die Hindernisvermeidung möglicherweise nicht mehr erfolgreich sein kann und stattdessen schon früher hätte aufgerufen werden müssen. Jedoch werden die Belohnungen, die weiter in der Zukunft liegen, mit einer abfallenden Gewichtung versehen, so dass die aktuellste Belohnung den größten Einfluss besitzt. Die neue Gewichtung des Verhaltens ergibt sich dann zu:

$$g_{KG,neu} = g_{KG,alt} + R_{KG,t} \tag{5.17}$$

mit $R_{KG,t}$:

$$R_{KG,t} = \begin{cases} -min(R_{evKG,t+1},\ \frac{1}{2}R_{evKG,t+2},\ \frac{1}{4}R_{evKG,t+3}) & \text{falls es einmal Kollisionsgefahr gab,} \\ min(R_{evKG,t+1},\ \frac{1}{2}R_{evKG,t+2},\ \frac{1}{4}R_{evKG,t+3}) & \text{falls zu große Abstände registriert sind,} \\ 0 & \text{sonst.} \end{cases} \tag{5.18}$$

Gab es mindestens eine Kollision innerhalb des Zeitfensters $(t+1)$ bis $(t+3)$, dann ergibt sich ein positives $R_{KG,t}$ und damit eine erhöhte Gewichtung der Kollisionsvermeidung. Je später diese Kollision auftrat, desto geringer wird die Erhöhung von $g_{KG,neu}$ ausfallen. War jedoch im gesamten Zeitfenster der Abstand zu den Hindernissen zu groß, dann ist $R_{KG,t}$ negativ und die Anwendbarkeit der Kollisionsvermeidung nimmt ab. Bei einem bereichsweise idealen Abstand bleibt die Gewichtung unverändert. Der Merkmalsvektor $\vec{s}'_{KG,t}$ und die neu berechnete Gewichtung des Verhaltens $g_{KG,neu}$ werden dann zur Anpassung des BBNs[11] eingesetzt.

Mit Hilfe dieser Anpassung verbessert die vermittelnde Ebene bei jedem Lerndurchgang die Auswahl des Verhaltens, indem sie die Reaktion der Umgebung, die als Belohnung vorliegt, in geeigneter Form verarbeitet. Durch die beschriebene Belohnungsfunktion lernt der Koordinationsmechanismus die Hindernisvermeidung so zu aktivieren, dass zumindest ein Sicherheitsabstand zu den Hindernissen gewahrt bleibt. Dieser Abstand kann durch die Konstanten k_{RLKG1} und k_{RLKG2} eingestellt werden.

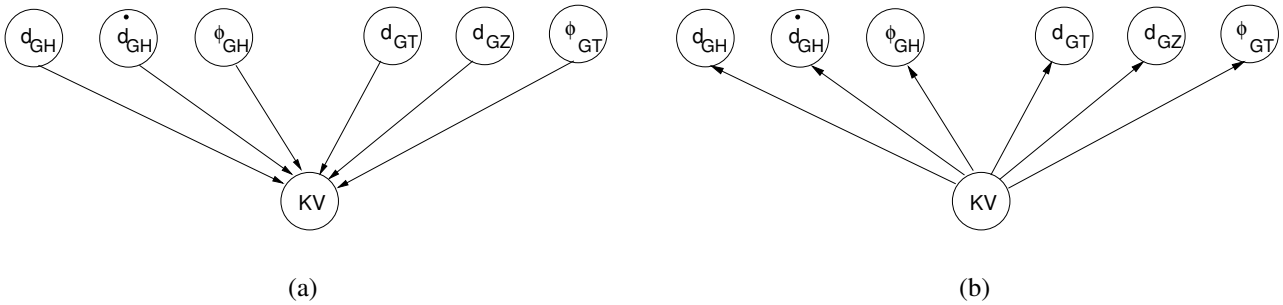
5.5 Resultate des Trainings

Das Training der Hindernisvermeidung des Greifers wurde für zwei verschiedene Topologien von BBNs durchgeführt und die Resultate sind in Abbildung 5.11 dargestellt. BBN2 bezeichnet hier die Struktur, die durch eine Abhängigkeit der Merkmalsknoten von dem Verhaltensknoten gekennzeichnet ist (Abbildung 5.10b). Demgegenüber ist im BBN1 der Verhaltensknoten von den Merkmalsknoten abhängig (Abbildung 5.10a). Zusätzlich ist die durchschnittliche Belohnung eines nicht trainierten BBN[12], bei dem die $cpts$ von Hand gesetzt wurden, dargestellt. Hierzu wurden nach einer längeren Experimentierphase sinnvolle Wahrscheinlichkeitsverteilungen angenommen.

Das Ziel des RL ist eine Maximierung der erzielten Belohnung über die Zeit. Im Fall der Hindernisvermeidung des Greifers bedeutet dies aufgrund der negativen Werte der Belohnungsfunktion eine

[11] Das verwendete Lernverfahren zum Anpassen der Werte eines BBNs ist im Anhang, Abschnitt C.7.2 schematisch aufgeführt.

[12] Das nicht trainierte BBN verfügt über dieselbe Topologie wie BBN2, das die besten Trainingsergebnisse erzielt hat.



(a) (b)

Abbildung 5.10: Mögliche Topologien des BBN für die Kollisionsvermeidung des Greifers: BBN1 (a) und BBN2 (b).

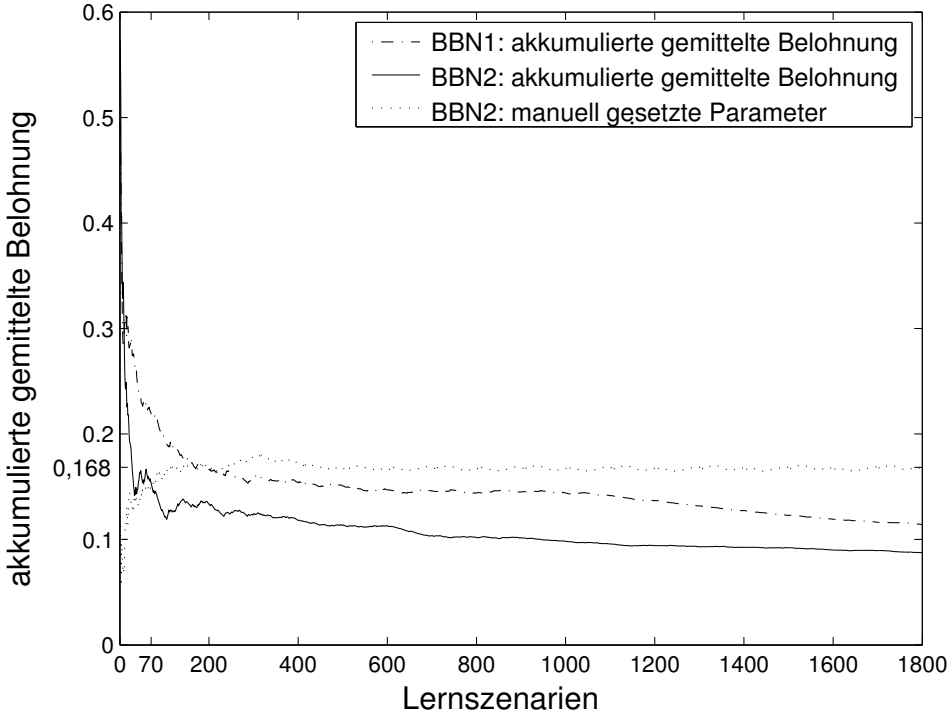


Abbildung 5.11: Lernkurve der akkumulierten gemittelten Belohnung für die Hindernisvermeidung des Greifers.

Minimierung der akkumulierten gemittelten absoluten Belohnung $R_{akk}(k) = \frac{1}{k}\sum_{t=1}^{k} \|R_{KG,t}\|$. Aus Abbildung 5.11 ist ersichtlich, dass BBN2 über den gesamten Lernvorgang bessere Resultate erzielt. Die erhaltene Belohnung konvergiert gegen eine durchschnittlich akkumulierte Belohnung von 0.09 gegenüber dem Wert von 0.11 für BBN1. Die beiden Kurven weisen ab ungefähr 1400 Lernszenarien keine deutliche Verbesserungen auf. Eine Konvergenz gegen Null ist nicht zu erwarten, da dies sowohl ein perfektes Verhalten als auch ein perfektes BBN bedingt. Die Reaktion des Verhaltens müsste dafür in jeder Situation exakt vorherbestimmbar und fehlerlos sein. Außerdem bedeutet eine Belohnung, die nicht gegen Null konvergiert, nicht, dass Kollisionen stattfinden, sondern dass der Greifer sich in der Gefahrenzone bewegt. Das nicht trainierte BBN hat einen ungefähr konstanten Verlauf bei 0.1675. Bereits nach 200 Schritten trifft es eine schlechtere Auswahl als die trainierten BBNs.

5.6 Ergebnisse der Verhaltenskoordination

Im Folgenden werden zwei Greifbewegungen, jeweils eine in der virtuellen Umgebung und eine mit dem realen Roboter, dokumentiert und die Bilder der Kameras zu mehreren Zeitpunkten dargestellt. Die Verhaltenskoordinierung ist mit einer Taktfrequenz von 1 Hz ausgeführt. Die auszuführende Aufgabe ist in beiden Fällen der Planungsschritt `Erreiche_Objekt`. Während des Greifvorgangs sind vier Verhalten aktiv: die Zielführung, das Planungsverhalten, die Hindernisvermeidung des Greifers und das Zielfolgen, das das Zielobjekt im Zentrum der Greiferkamera halten soll[13].

Abbildung 5.12 zeigt einen Ausschnitt der Sicht der Bordkamera zu Beginn der Ausführung des Planungsschrittes. Zusätzlich sind der vom Planungsverhalten berechnete Pfad und der im Laufe des Greifvorgangs tatsächlich durchlaufene Pfad abgebildet; letzterer liegt im Bereich der Hindernisse höher, da die Verhalten zur Hindernisvermeidung dort die Bewegung stärker beeinflussen. Die Gewichtung der einzelnen Verhalten über die Zeit ist in Abbildung 5.13 dargestellt. Dabei entspricht die Abszisse der Zeit vom Zeitpunkt t_1 zu Beginn des Greifvorgangs bis zum Zeitpunkt t_4 beim Erreichen der Zielposition am Objekt. Die Ordinate beschreibt die Gewichtung und daher die Anwendbarkeit der Verhalten in der jeweiligen Situation, die aus den zugehörigen BBNs berechnet wird. Abbildung 5.16 zeigt die ausgeführte Greifbewegung zu den Zeitpunkten t_1 bis t_4.

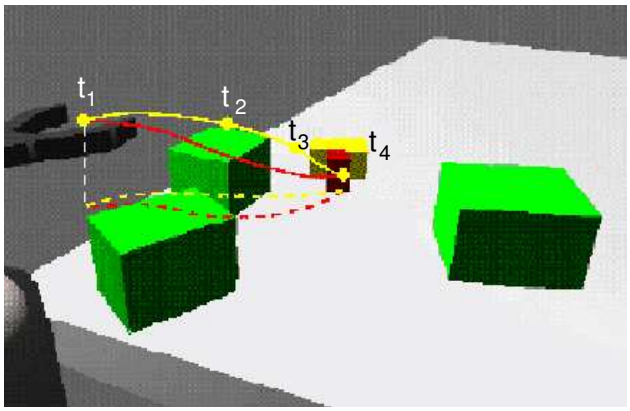


Abbildung 5.12: Berechneter (dunkle Linie) und tatsächlich ausgeführter Pfad (helle Linie) in der virtuellen Umgebung. Die gestrichelten Linien sind die Projektionen der Pfade auf der Auflagefläche.

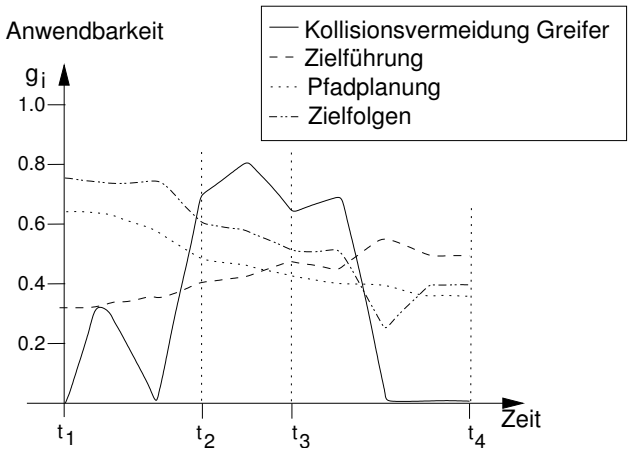


Abbildung 5.13: Die Gewichtung der Verhalten während der Ausführung der Greifbewegung.

Zu Beginn des Greifvorgangs erhält die Zielführung eine niedrige Gewichtung, da die Position der Greiferkamera ungünstig zur Berechnung eines Pfades ist. Aufgrund der besseren Ausrichtung der Greiferkamera und der Überwindung der Hindernisse in Greifrichtung erhöht sich die Anwendbarkeit der Zielführung zu den letzten Zeitpunkten. Die Nähe des Greifers zu Hindernissen im Zeitraum t_2 bis t_3 verursacht eine hohe Gewichtung der Hindernisvermeidung; am Ende der Greifbewegung ist jedoch die Kollisionsgefahr nicht groß und daher sinkt ihre Anwendbarkeit. Die Anwendbarkeit des Zielfolgens ist zu Beginn hoch, da der Greifer zunächst zum Ziel ausgerichtet werden muss. Je näher

[13] Im Experiment in der realen Umgebung ist anstatt des Zielfolgens die Hindernisvermeidung der Segmente aktiv.

das Objekt ins Zentrum des Bildes der Greiferkamera rückt, desto niedriger wird dann die Gewichtung des Zielfolgens.

Analog zu dem Experiment in der virtuellen Umgebung wird auch ein Greifversuch mit dem realen Manipulator dargestellt. Abbildung 5.14 vergleicht den vom Planungsverhalten vorab berechneten Pfad mit dem tatsächlich ausgeführten Pfad, während Abbildung 5.15 die Gewichtung der Verhalten zu verschiedenen Zeitpunkten darstellt. Abbildung 5.17 gibt die im Verlauf des Greifvorgangs aufgenommenen Bilder der Kameras zu den Zeitpunkten t_1 bis t_4 wieder.

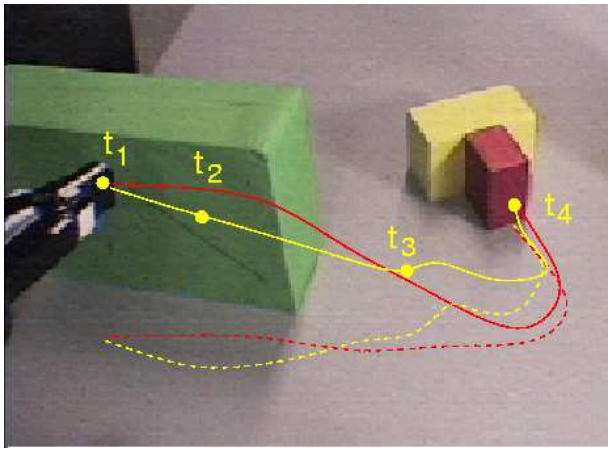


Abbildung 5.14: Berechneter (dunkle Linie) und tatsächlich ausgeführter Pfad (helle Linie) mit dem realen Manipulator. Die gestrichelten Linien sind die Projektionen der Pfade auf der Auflagefläche.

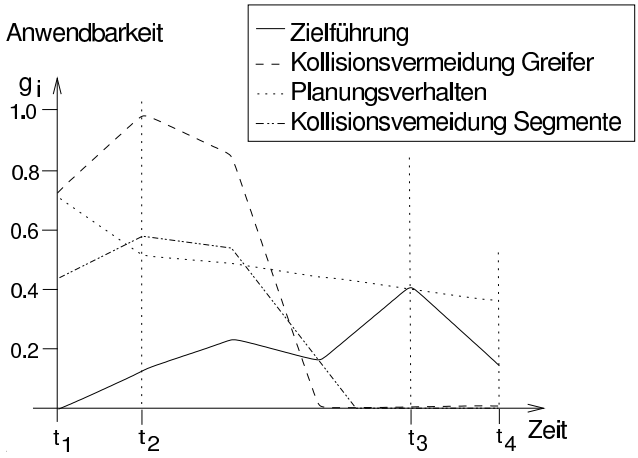


Abbildung 5.15: Die Gewichtung der Verhalten während der Ausführung der Greifbewegung.

Zu Beginn des Greifvorgangs erhält die Zielführung eine Gewichtung von Null, da das Zielobjekt vom Hindernis verdeckt wird. In diesem Fall kann das Verhalten keinen zuverlässigen Pfad berechnen. Im weiteren Verlauf steigt die Anwendbarkeit an, da sich die Position des Greifers relativ zum Ziel verbessert, d.h. das Zielobjekt erscheint vollständig im Kamerabild. Die Hindernisvermeidung des Greifer besitzt dagegen am Anfang einen hohen Wert, da der Greifer sich in sehr geringer Entfernung zu einem Hindernis bewegt und sich in Kollisionsgefahr befindet. Weiterhin verläuft seine Bewegungsrichtung parallel zur Oberfläche des Hindernisses, die Distanz nimmt also nicht zu. Danach entfernt sich der Greifer vom Hindernis, und deshalb wird die Kollisionsvermeidung des Greifers praktisch deaktiviert. Aus demselben Grund hat die Anwendbarkeit der Hindernisvermeidung der Segmente zu Beginn einen hohen Wert. Die Anwendbarkeit des Planungsverhaltens sinkt im Verlauf des Greifvorgangs, fällt jedoch nicht unter eine minimale Gewichtung.

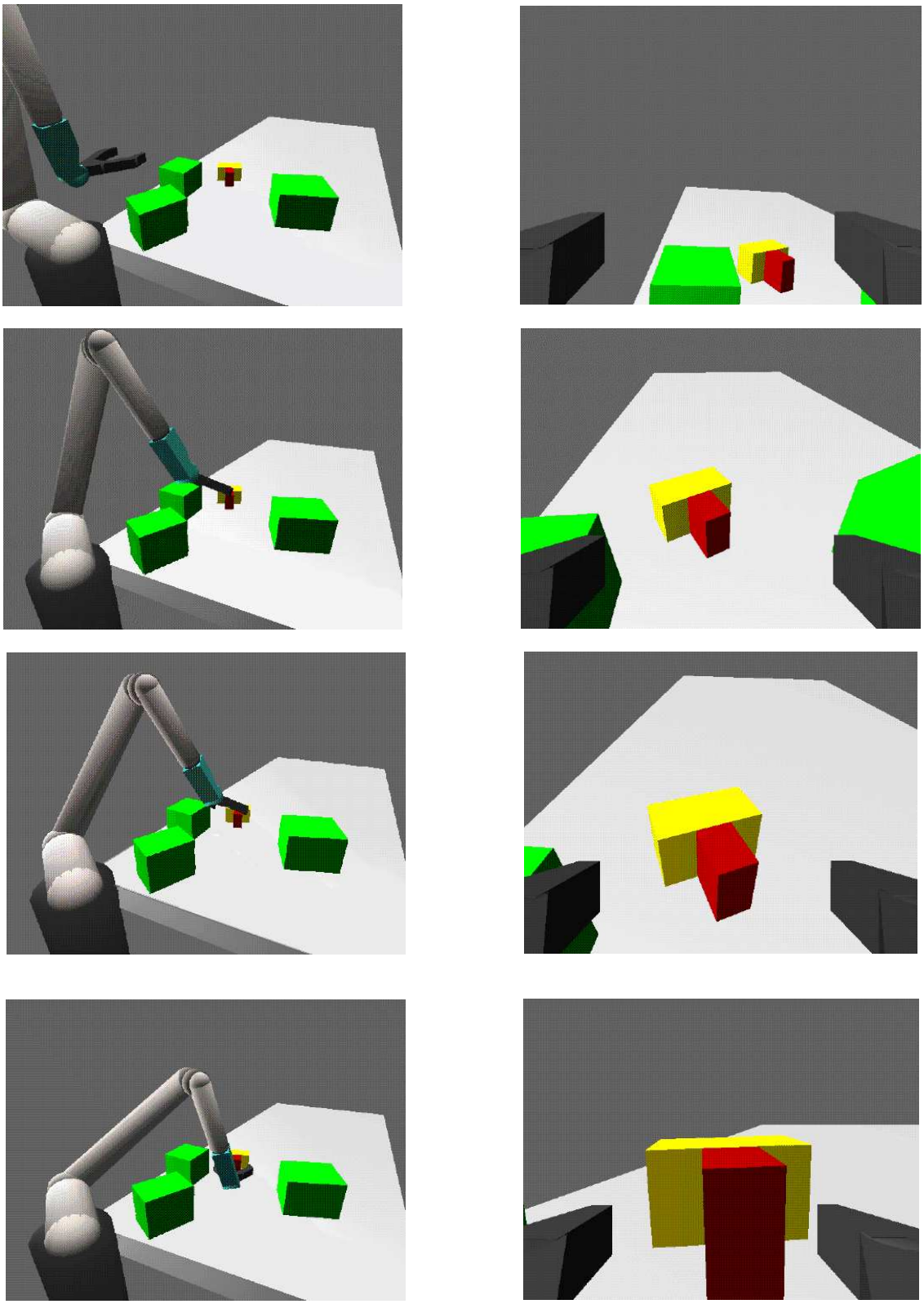
5.7 Bewertung und Einordnung des implementierten Koordinationsmechanismus

Der hier vorgestellte Ansatz kombiniert Elemente der *Arbitration*- und der *Command Fusion*-Mechanismen. In der Funktion als *Arbitration*-Mechanismus wird ähnlich zu MacKenzie [MA99] die Ver-

haltensauswahl mit endlichen Zustandsakzeptoren durchgeführt. Nach den *Command Fusion*-Mechanismen können jedoch in jedem Zustand des FSAs mehrere Verhalten gleichzeitig aktiv sein und zur Lösung beisteuern. So ist beispielsweise ein schnelles Erreichen des Zielobjektes mit gleichzeitiger Hindernisvermeidung möglich. Der Manipulator führt eine direkte und glatte Trajektorie aus, da nicht zwischen Verhalten gewechselt wird, sondern eine gewichtete Überlagerung der vorgeschlagenen Pfade stattfindet.

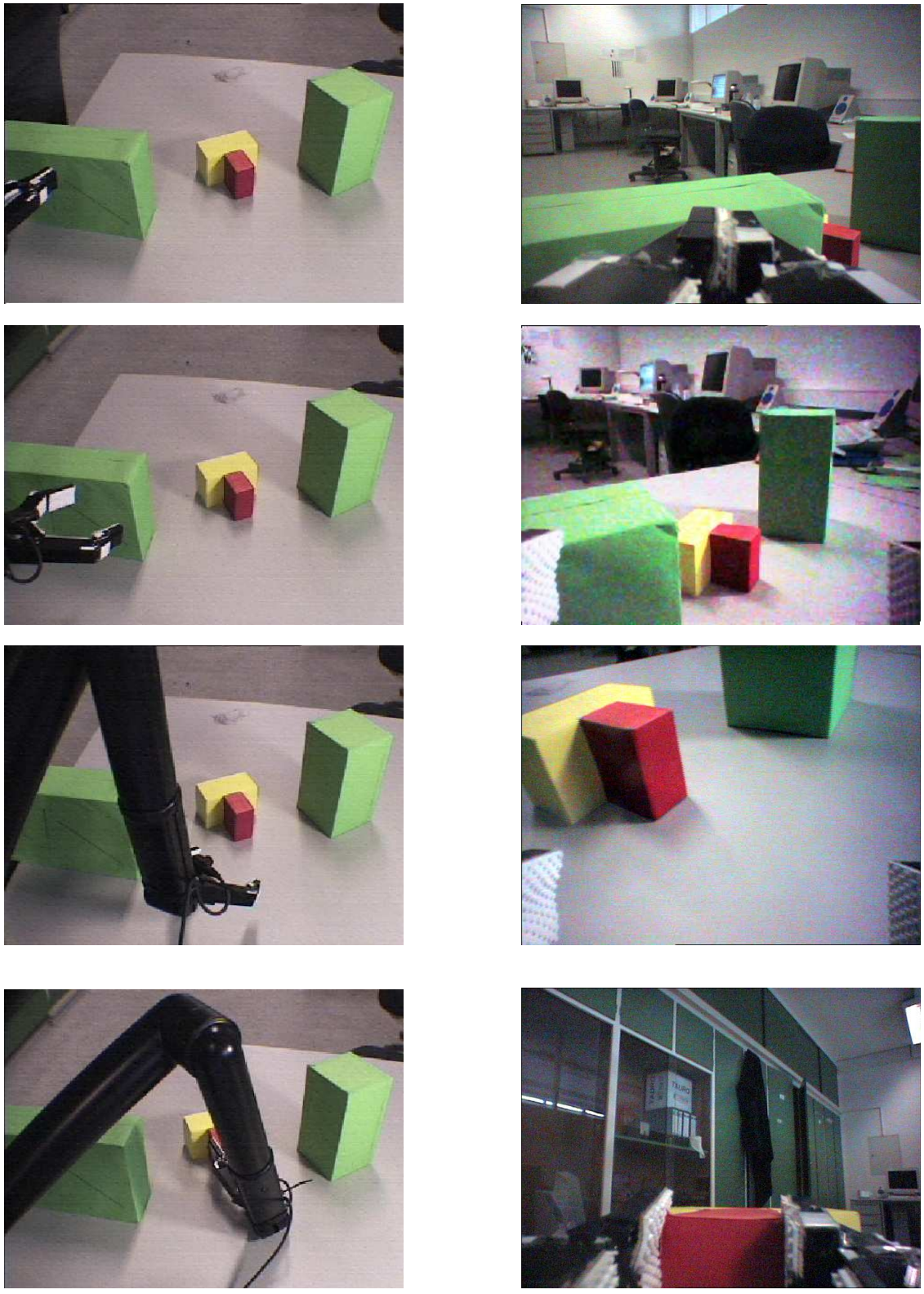
Die Haupterweiterung in der vermittelnden Ebene liegt jedoch in dem Einsatz der BBNs zur Verhaltenskoordinierung. BBNs sind bisher nur zur Aktionsauswahl zwischen verschiedenen Alternativen benutzt worden [AC96]. Im Gegensatz dazu wählt die entworfene vermittelnde Ebene nicht eine einzelne Aktion aus. Stattdessen dienen die BBNs der Berechnung einer Gewichtung für jedes aktive Verhalten und der Interpolation der gewichteten Verhaltensausgaben. Dabei bringen BBNs mehrere Vorteile mit sich. Zum einen können sie leicht um neue Knoten ergänzt werden. Wird der Merkmalsvektor um ein zusätzliches Element, eine zusätzliche Beobachtung oder sogar einen neuen Sensor erweitert, kann einem bestehenden BBN ein neuer Knoten mit einer entsprechenden cpt hinzugefügt werden. Die bisherige Struktur bleibt unverändert bestehen. Weiterhin erlauben BBNs die Modellierung der Unsicherheit der oft verrauschten Merkmalswerte und bilden dadurch ein gegenüber fehlerbehafteter Merkmalsextraktion robusten Entscheidungsmechanismus. Außerdem existieren für BBNs lernfähige Algorithmen. Sie erlauben es, die Anwendbarkeit eines Verhaltens in einer bestimmten Situation durch Erfahrung zu trainieren. Die $cpts$ müssen somit nicht manuell gesetzt werden. Oftmals ist dies auch nicht in einer ausreichenden Qualität möglich, weil das zugrunde liegende Roboterverhalten in einer Situation nicht vollständig vorhersagbar ist. Die vermittelnde Ebene kann durch das Lernen besser an die spezifische Umgebung und die verwendete Aktuatorik angepasst werden. Das Training der BBNs findet in der virtuellen Umgebung statt. Auf diese Weise lässt sich der Lernvorgang automatisieren, so dass wesentlich mehr Durchgänge als bei einem realen Training möglich sind. Da besonders zu Beginn des Trainings viele Kollisionen auftreten können, hat das virtuelle Training den Vorteil, gefahrlos abzulaufen. Außerdem besteht die Möglichkeit, die exakt gleiche Situation mit denselben Objekt- und Roboterpositionen zum mehrmaligen Training des Greifvorgangs zu nutzen.

Der Koordinationsmechanismus besitzt mehrere Gemeinsamkeiten mit der auf Fuzzy-Regeln beruhenden Lösung nach Saffioti [Saf97]. Für jedes Verhalten wird die Anwendbarkeit in Form einer Gewichtung ihrer Ausgabe abhängig von Merkmalsvektor berechnet. Anschließend wird eine Form von *Context-dependent Blending* durch die Bildung der gewichteten Summe aller Pfade durchgeführt. Durch den Einsatz von BBNs zur Bestimmung der Anwendbarkeit ist jedoch der Koordinationsmechanismus leichter um neue Merkmale und Verhalten erweiterbar als der Ansatz von Saffioti. Außerdem ist der Fuzzy Logik-Ansatz nicht auf mobile Manipulatoren zum Einsatz gekommen. Im Vergleich zu dem Ansatz von Wösch et al. [WN01] findet hier eine gewichtete Summierung anhand des gegenwärtigen Sensorkontexts statt, die besonders in kritischen Bereichen den Roboterarm effektiv vor Kollisionen schützt. Der Koordinationsmechanismus ist, unterschiedlich zu dem Verfahren in [HNI01], leicht um neue Verhalten erweiterbar; da jedes Verhalten ein eigenes BBN erhält, wird bei der Implementierung eines neuen Verhaltens lediglich ein neues BBN erzeugt und erlernt. Auf diese Weise entfällt ein komplettes neues Training bzw. ein Umstrukturieren der alten Komponenten. Der Koordinationsmechanismus erfordert auch kein aufwendiges Experimentieren zum Setzen der Parameter wie in [HNI01], da diese in einer virtuellen Umgebung erlernt werden.



(a) (b)

Abbildung 5.16: Aufnahmen der virtuellen Bord- (a) und Greiferkamera (b) zu den Zeitpunkten t_1 bis t_4.



(a) (b)

Abbildung 5.17: Aufnahmen der realen Bord- (a) und Greiferkamera (b) zu den Zeitpunkten t_1 bis t_4.

Kapitel 6

Aufgabenplanung

Die vermittelnde Ebene kann zwar die Fertigkeiten der mobilen Plattform ansteuern und einfache Planungsschritte ausführen, sie kann jedoch weder komplexe Aufgaben lösen, noch bei einem Misserfolg eines Planungsschrittes alternative Lösungswege finden und anwenden. Ihre Aufgabe ist hauptsächlich die Einhaltung der Stabilität und der Sicherheit des Roboters. Deshalb ist eine steuernde Instanz notwendig, die eine Aufgabenstellung analysiert, deren Durchführung in die Zukunft projiziert und entsprechende Lösungswege findet. Dies geschieht in der deliberativen Ebene.

Die deliberative Ebene hat die Aufgabe, einen Lösungsplan für eine Mission zu berechnen und ihn schnell an wechselnde Umgebungsbedingungen anzupassen. Misslingt ein Greifversuch mit dem Manipulator, dann berechnet die deliberative Ebene eine alternative Position des Roboters, aus der ein Greifvorgang erfolgversprechend ist, und führt den mobilen Manipulator dorthin.

6.1 Aufbau der deliberativen Ebene

Um die Vorteile sowohl von wissensbasierten als auch von reaktiven Planungssystemen zu kombinieren, setzt die hier implementierte deliberative Ebene in ähnlicher Weise wie die CLARAty Architektur [VNE+01] sowohl ein wissenbasiertes Planungsverfahren zur Erstellung eines Lösungplans für eine gestellte Aufgabe ein, als auch ein reaktives Planungssystem. Letzteres analysiert den erstellten Plan weiter und wählt die einzelnen auszuführenden Planungsschritte entsprechend des Erfolges bzw. Misserfolges der Roboteraktionen aus (Abbildung 6.1).

Der *High-Level Planner* (HLP) besteht aus einem wissensbasierten Planungsystem, das auf der höchsten Ebene der Abstraktion arbeitet. Er verfügt nur über symbolisches Wissen über die Welt und den mobilen Manipulator und eine topologische Darstellung der Umgebung, anhand dessen er Pläne zur erfolgreichen Durchführung einer Aufgabe erstellt. Der HLP kennt abstrakte Handlungsweisen, die der Roboter ausführen kann, jedoch nicht die Verhalten der reaktiven Ebene. Ein vom HLP erstellter Plan ist eine Liste von Teilaufgaben bzw. Teilplänen, die der Roboter ausführen muss, um die Mission zu erfüllen.

Beim *Low-Level Planner* (LLP) handelt es sich dagegen um ein reaktives Planungsystem. Seine Aufgabe ist es, eine vom HLP erstellte Liste von Teilplänen weiter zu verfeinern und deren Ausführung durch die vermittelnde Ebene zu überwachen. Im Fall eines Misserfolges sorgt der LLP für die Ausführung alternativer Planungsschritte, um ans Ziel des Teilplanes zu gelangen. Hat er dabei keinen

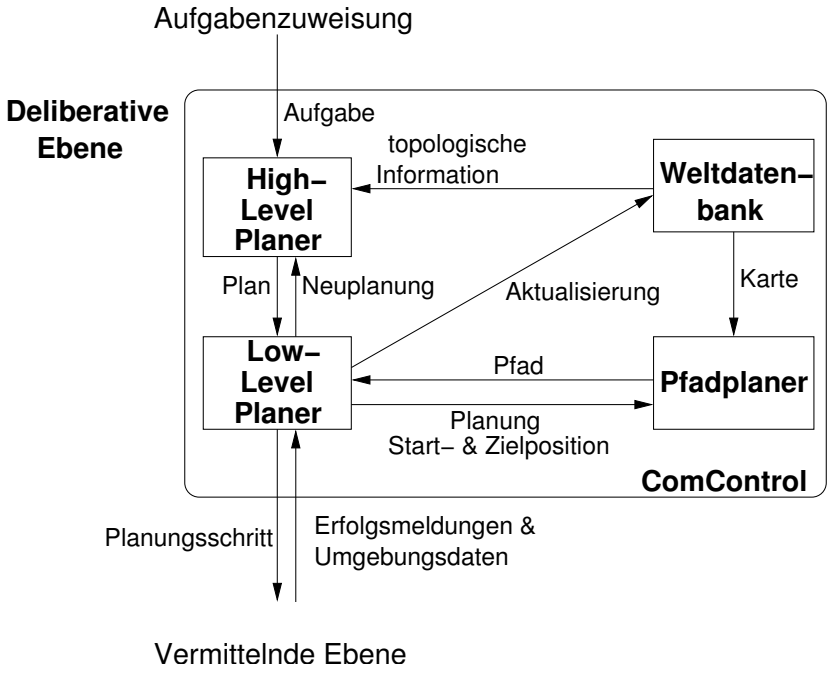


Abbildung 6.1: Architektur der deliberativen Ebene.

Erfolg, dann wird eine Neuplanung beim HLP angefordert. Dieser hat anhand der Weltdatenbank, die die Gesamtsituation wiederspiegelt, eine bessere Möglichkeit, eine alternative, erfolgversprechende Reihenfolge von Planungsschritten zu finden. Weiterhin aktualisiert der LLP die Daten in der Weltdatenbank anhand von Information aus der vermittelnden Ebene. Für Planungsschritte, die keine Greifbewegung, sondern eine Bewegung der Plattform zu einer Zielposition erfordern, lässt er von einem Pfadplaner einen kollisionsfreien Pfad berechnen, der dann von der vermittelnden Ebene der Plattform ausgeführt wird.

Die deliberative Ebene wird durch eine Kontrollkomponente ergänzt, die *Command and Communication Control*, kurz ComControl genannt. Die ComControl hat als Hauptaufgabe, die Schnittstellen zur Verfügung zu stellen, über die HLP, LLP, Weltmodell und Pfadplanung miteinander und mit der Außenwelt kommunizieren.

6.2 High-Level Planer

Die Aufgabe des *High-Level Planers* (HLP) ist, anhand einer Aufgabenstellung des Benutzers, einer Beschreibung der Umgebung und den Vorbedingungen und Auswirkungen von Teilplänen, die der Roboter ausführen kann, eine Reihenfolge von Teilplänen zu finden, die den mobilen Manipulator zum Ziel führen. Hier kommt die Sprache S-GOLOG[1] [Spr02], ein Dialekt von IndiGolog, zum Einsatz. Sie erlaubt eine einfache Implementierung eines *high-level* Programms auf einem hohen Abstraktionsniveau. Der HLP errechnet mit S-GOLOG und unter Verwendung des Situationskalküls

[1] S-GOLOG erweitert IndiGolog um die Fähigkeit, nichtdeterministische Aktionssequenzen anhand ihrer Erfolgswahrscheinlichkeit auszuwählen. Die Notation der Sprache ist an die Notation von Modula angelehnt, um die Erstellung von Programmen zu vereinfachen. S-GOLOG ermöglicht es, zwischen erfolgversprechenden und weniger erfolgversprechenden Teilplänen zu unterscheiden und aus den Erfolgswahrscheinlichkeiten und dem S-GOLOG Programmcode ein entsprechendes IndiGolog Programm zu generieren.

einen Plan als eine Folge von Teilplänen, die den Roboter in einen Zielzustand in der Welt überführen und somit die gestellte Aufgabe lösen. Hierzu verwendet die Planung die topologische Information aus der Weltdatenbank. Anschließend übergibt der HLP die Sequenz der Teilpläne dem LLP zur Ausführung. Tabelle 6.1 listet die Teilpläne auf, die der LLP ausführen kann.

Teilaufgabe	**Funktionalität**
`llp_plan_path(World,Pos)`	Plane Pfad mit dem Pfadplaner
`llp_get_coords_of_position(Pos)`	Hole die aktuelle Plattformposition
`llp_take_object_1(Obj)`	Greife Zielobjekt mit den Hindernisvermeidung des Greifers und der Segmente, der Zielführung und dem Planungsverhalten aktiviert
`llp_take_object_2(Obj)`	Greife Zielobjekt ohne die Hindernisvermeidung der Segmente zu aktivieren
`llp_take_object_3(Obj)`	Greife Zielobjekt mit aktiviertem Planungsverhalten und Zielführung
`llp_take_object_4(Obj)`	Greife Zielobjekt mit aktiviertem Planungsverhalten
`llp_take_object_5(Obj)`	Greife Zielobjekt mit der Zielführung
`llp_drop_object_1(Pos)`	Platziere Objekt an die Position `Pos`
`llp_drop_object_2(Pos)`	Bei aktivierter Hindernisvermeidung der Segmente platziere Objekt an Position `Pos`
`llp_global_localize(Obj)`	Lokalisiere Objekt mit der Bordkamera
`llp_camera_localize(Obj)`	Lokalisiere Objekt mit der Greiferkamera
`llp_db_localize(Obj)`	Lokalisiere Objekt anhand der Information in der Weltdatenbank
`llp_goto_position_1(Pos)`	Fahre zur Position `Pos` mit der Plattform
`llp_goto_position_2(Pos)`	Fahre zur Position `Pos` mit der Plattform ohne die Hindernisvermeidung zu aktivieren
`llp_platform_relocate(Pos)`	Relokalisiere Plattform mit dem Laserscanner
`llp_get_take_position(Obj,Nr)`	Bestimme eine Plattformposition anhand der Objektposition aus der Weltdatenbank
`llp_get_room(Pos)`	Finde aus der Weltdatenbank in welchen Raum sich der Roboter befindet
`llp_in_range(Obj)`	Überprüfe ob sich das Objekt in Reichweite des Manipulators befindet
`llp_object_found(Obj, Succ)`	Überprüfe ob das Objekt gefunden wurde
`llp_get_expected_position(Obj,Nr)`	Finde aus der Weltdatenbank welche die wahrscheinlichste Objektposition ist
`llp_replan(Objs, Pos)`	Bestimme eine neue Plattformposition aus der ein erfolgreiches Greifen möglich ist
`llp_pass_door(Src, Door, Dst)`	Fahre durch Tür `Door` von Raum `Src` zu Raum `Dst`

Tabelle 6.1: Teilaufgaben, die der LLP ausführen kann.

Im für das Testszenario implementierten *high-level* Programm findet keine ausführliche Suche im Zustandsraum statt (siehe auch Abschnitt 2.4.1). Grund dafür ist, dass für eine *Pick-and-Place* Aufgabe die Abfolge der Schritte, die der mobile Manipulator auszuführen hat, von Beginn an festgelegt ist: Der Roboter muss zuerst das Objekt finden und greifen, anschließend zum Ablageort fahren und das Objekt dort ablegen. Demnach wäre eine ausführliche Suche hier überflüssig und bei der Anzahl der Teilaufgaben, die der LLP ausführen kann, zeitlich überfordernd für den HLP-Planer. Deswegen wird hier die Möglichkeit von *high-level* Programmen ausgenutzt, eine bestimmte Reihenfolge der auszuführenden Planungsschritte vorzudefinieren. Der Ablauf, der durch das implementierte Programm ermöglicht wird, ist im Folgenden kurz beschrieben.

Sind mobiler Manipulator und Zielobjekt zu Beginn in unterschiedlichen Räumen, dann findet zuerst eine nicht deterministische topologische Suche in S-GOLOG statt. Diese ermittelt die Reihenfolge, mit der die einzelnen Räume zu betreten sind, um zu dem Raum zu gelangen, in dem sich das Objekt befindet. Anschließend ermittelt der Pfadplaner für jeden zu durchquerenden Raum einen Pfad zwischen Eingang und Ausgang.

Befinden sich der mobile Manipulator und das Zielobjekt im selben Raum, dann wird zuerst das Objekt in der Umgebung über die Sensorik lokalisiert. Bei einem Misserfolg findet eine Suche in der Weltdatenbank nach dem wahrscheinlichsten Aufenthaltsort des Objektes statt. Der Pfadplaner berechnet eine kollisionsfreie Route zu dieser Position und die mobile Plattform fährt dorthin, um erneut sensorbarsiert nach dem Objekt zu suchen. Dieser Vorgang wiederholt sich, bis der Roboter das Objekt gefunden oder alle möglichen Aufenthaltsorte in der Weltdatenbank abgearbeitet hat.

Für das Greifen überprüft der HLP zuerst, ob das Objekt in Reichweite liegt und versucht anschließend dieses zu greifen. Ist ein Greifen nicht möglich, dann erfolgt die Bestimmung einer besseren Roboterposition nach dem Ansatz, der in Abschnitt 6.5.2 beschrieben ist. Von der neuen Position wird erneut ein Greifversuch initiiert. Bei erfolgreichem Greifen ermittelt der Pfadplaner einen Weg zum Ablageort[2] und legt das Objekt mit einer der zwei möglichen Ablagemethoden ab.

6.3 Low-Level Planer

Der *Low-Level Planer* (LLP) ist für die Koordination der Ausführung der vom HLP erstellten Pläne verantwortlich. Er stellt eine zentrale Einheit dar, die sowohl mit dem HLP und der Weltdatenbank als auch mit der vermittelnden Ebene kommuniziert und damit eine verbindende Aufgabe ausführt. Nachdem der HLP einen Lösungsplan in Form einer Sequenz von Teilplänen ermittelt hat, muss der LLP jeden Teilplan in eine feste Reihenfolge von Planungsschritte zerlegen. Diese werden anschließend von der vermittelnden Ebene des entsprechenden Aktuators ausgeführt. Anhand der Ausgaben der vermittelnden Ebene aktualisiert der LLP anschließend die Daten über die Objekte, mit denen der mobile Manipulator interagiert, und speichert sie in der Weltdatenbank, um sie bei nachfolgenden Planungsprozessen zu berücksichtigen. Für Teilaufgaben, die eine Fahrt der Plattform von einem Start- zu einem Zielpunkt beinhalten, initiiert der LLP eine Pfadplanung beim Routenplaner. Der gefundene Pfad wird anschließend zur vermittelnden Ebene der Plattform weitergeleitet.

Die Basis des LLPs bilden Skripte, die die Teilpläne in feste Reihenfolgen von Planungsschritten zusammenfassen. Dabei können Skripte mehrere Tests und Entscheidungspunkte beinhalten, die anhand

[2] Befinden sich Ablageort und Roboter in unterschiedlichen Räumen, dann findet eine erneute topologische Wegsuche statt.

der Erfolgsmeldungen schon ausgeführter Aktionen und des aktuellen Umgebungszustands zur Ausführung alternativer Reihenfolgen von Planungsschritten führen. Planungsschritte, die von Plattform und Manipulator parallel auszuführen sind, müssen vom Programmierer entsprechend gekennzeichnet sein; so können Plattform und Manipulator parallel zueinander handeln. Da jedoch im Testszenario keine enge Kooperation zwischen Plattform und Manipulator notwendig ist und alle Teilaufgaben durch sequentiell ausgeführte Planungsschritte zu erreichen sind, wurde das Thema der Koordination zwischen Manipulator und Plattform in dieser Arbeit nicht weiter behandelt. Die Plattform bringt den Manipulator zum Einsatzort und bleibt dann während der Ausführung der Manipulationsaufgaben stehen; ebenso wird der Manipulator nicht verwendet, wenn die Plattform sich bewegt.

Die Funktionsweise des LLP ist vergleichbar mit der Verhaltensaktivierung der vermittelnden Ebene. Hier wird jedoch im Gegensatz zu den FSAs der vermittelnden Ebene, beim Misserfolg einer Aktion nicht direkt der Misserfolgszustand erreicht, sondern eine alternative Reihenfolge von Planungsschritten aktiviert, um den Misserfolgszustand aufzuheben. Da der LLP im Gegensatz zu der vermittelnde Ebene nicht für einen Aktuator spezialisiert ist, kann er sowohl Plattform als auch Manipulator koordiniert ansprechen, um eine größere Effizienz und Flexibilität bei der Lösung zu erlangen. So berechnet er beispielsweise bei der Neuplanung eines Greifversuches eine neue, erfolgversprechende Position für den Roboter, indem er sowohl die Hindernisse für die Plattform als auch für den Manipulator berücksichtigt (vergleiche auch Abschnitt 6.5.2). Falls jedoch diese alternativen Planungsschritte auch nicht erfolgreich sind, dann wird das Weltdatenbank aktualisiert und eine Neuplanung im HLP angestoßen, um einen umgestalteten Lösungsplan für die zugewiesene Aufgabe zu finden. Weiterhin ermöglicht der LLP eine effiziente und schnelle wissensbasierte Planung im HLP, indem er in den Teilplänen zusammengehörende Planungsschritte von verschiedenen Aktuatoren zusammenfasst. Dadurch ist im HLP keine zeitaufwendige, ausführliche Suche im Zustandsraum notwendig.

6.4 Weltdatenbank

Die Weltdatenbank dient zur Abspeicherung und Bereitstellung von Information aus der Umwelt des Roboters, um nach einer Anfrage den HLP und den Pfadplaner mit Wissen über die Einsatzumgebung zu beliefern. Zusätzlich unternimmt sie bei einem Teil der eingehenden Daten, insbesondere bei der Position der Objekte, die der mobile Manipulator transportieren soll, eine statistische Analyse, um die wahrscheinlichsten Aufenthaltsorte dieser Objekte dem HLP zu liefern.

Die Information in der Weltdatenbank beschreibt die topologischen und geometrischen Eigenschaften und Beziehungen der Objekte in der Einsatzumgebung zueinander. Die geometrische Information wird zum einen über die Position und das Ausmaß der Objekte und zum anderen in Form einer 2D-Karte der Einsatzumgebung in der Weltdatenbank abgespeichert. Die topologischen Beziehungen sind in der baumartigen Struktur ausgedrückt, mit der die Daten in der Weltdatenbank abgespeichert sind (Abbildung 6.2). Diese Struktur eignet sich auch zur einfachen Verwaltung der Daten.

Jedes Objekt, das in der Weltdatenbank eingetragen ist, besitzt neben seinem Namen auch einen Typ. Dadurch können Objekte sowohl über ihre Namen, z.B. `Raum1:TischA`, als auch über ihren Typ, beispielsweise `Raum1:tisch(1)` angesprochen werden. Die zweite Schreibweise erlaubt es, Schleifendurchgänge zu implementieren, um beispielsweise alle Tische in Raum 1 nach dem Zielobjekt abzusuchen. Über stationäre Objekte, die selten oder gar nicht bewegt werden, wie z.B. Türen, Wände oder Regale, werden die Position, ihr Ausmaß und topologische Information über deren Relation zu anderen Objekten abgespeichert. Für bewegliche Gegenstände, wie beispielsweise das

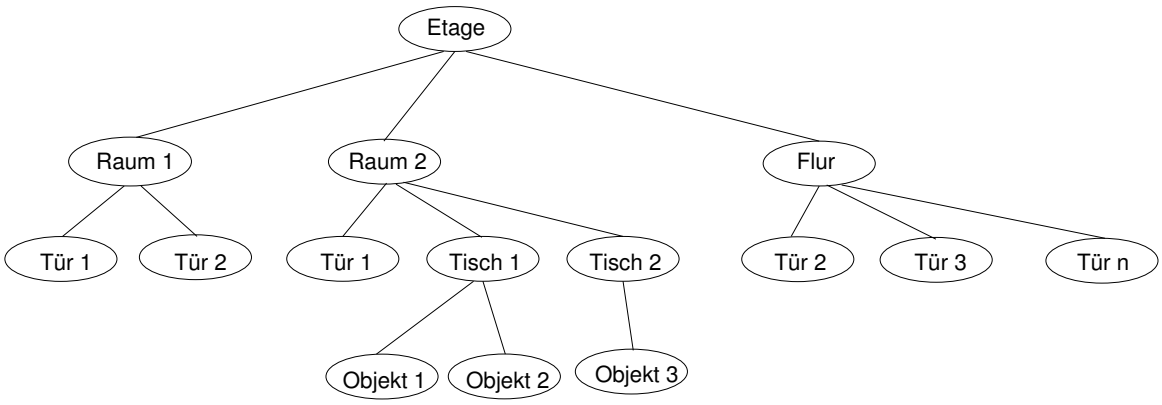


Abbildung 6.2: Graphische Darstellung der topologischen Beziehungen über eine baumartige Struktur.

Zielobjekt des verwendeten Testszenarios, wird zusätzlich statistische Information über vergangene, erfolgreiche Ortungen des Objektes an verschiedenen Positionen abgespeichert, die bei der nächsten Planung berücksichtigt werden. Als Beispiel folgt hier ein Ausschnitt der Weltdatenbank für das Testszenario (siehe auch Abbildung 6.3); die komplette Version ist im Anhang F zu finden.

```
Root = "LTISzenario"
{
 Obj = "Raum_1"
 {
  Subtype = "Room";
  Position = "3.0, 3.0, 0.0";
  Space = "(-3.5, -3.5), (-3.5, 3.0), (3.0, 3.0), (3.0, -3.5)";
  Obj = "Tisch_A"
  {
   Subtype = "Table";
   Position = "4.5, 2.0, 0.0";
   Space = "(-0.4, -0.9), (-0.4, 0.9), (0.4, 0.9), (0.4, -0.9)";
   Obj = "TischBereich_1"
   {
    Subtype = "TableArea";
    Position = "4.5, 2.45, 0.0";
    Space = "(-0.4, -0.45), (-0.4, 0.45), (0.4, 0.45),(0.4,-0.45)";
   }
   Obj = "TischBereich_2"
   {
    Subtype = "TableArea";
    Position = "4.5, 1.55, 0.0";
    Space = "(-0.4, -0.45), (-0.4, 0.45), (0.4, 0.45),(0.4,-0.45)";
   }
  }
 }
 Root = "Events"
 {
```

```
  Root = "ZielObjekt"
  {
   ss_type = "Obj#Raum_1:Obj#Tisch_A:Obj#TischBereich_1"
   {
    ss_succ = "0 0 0 0 0 1";
    ss_fail = "1 0 0 0 0 0";
   }
   ss_type = "Obj#Raum_1:Obj#Tisch_A:Obj#TischBereich_2"
   {
    ss_fail = "0 0 0 0 0 0";
    ss_succ = "0 1 0 0 0 0";
   }
  }
 }
}
```

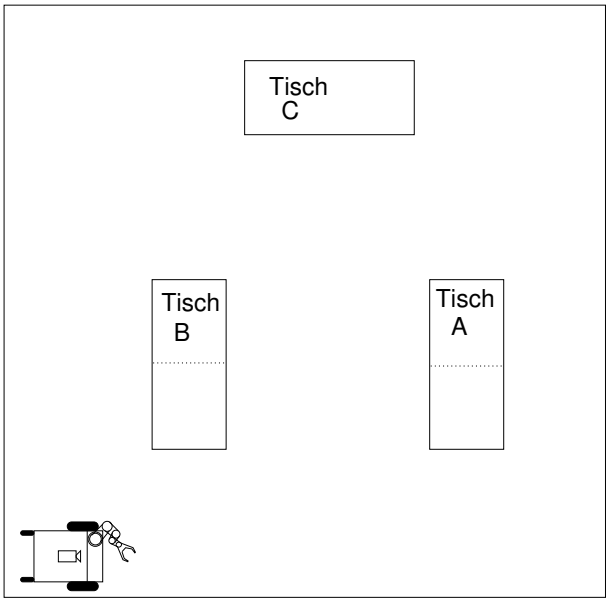


Abbildung 6.3: Geometrische Karte für das Testszenario.

6.5 Geometrische Planung

Damit der mobile Manipulator Objekte greifen kann, muss die mobile Plattform in der Nähe der Objekte geführt werden, so dass diese sich in Reichweite des Manipulators befinden. Misslingt anschließend ein Greifversuch, dann ist eine neue Greifposition für den mobilen Manipulator zu berechnen, aus der das Greifen des Zielobjektes möglich ist. Um diese Operationen auszuführen zu können, ist eine Komponente in der deliberativen Ebene erforderlich, die eine robuste geometrische Pfadplanung für die mobile Plattform unternimmt.

6.5.1 Pfadplanung für die mobile Plattform

Mit der Pfadplanung ist es möglich, Wege von einem Start- zu einem Zielpunkt für die mobile Plattform zu berechnen. Beim eingesetzten Pfadplaner handelt es sich um ein *Probabilistic Roadmap* Verfahren, das auf dem im Abschnitt 4.3.2 beschriebenen Pfadsucheansatz basiert. Allerdings findet hier die Pfadsuche im zweidimensionalen Raum statt, da nur die Bewegung der Plattform zu berücksichtigen ist. Die geometrische Darstellung der Einsatzumgebung (Abbildung 6.3) liegt als Karte[3] in der Weltdatenbank vor.

In der Karte wird die Umgebung durch eine Menge von Freiräumen, in denen sich der Roboter bewegen kann, und eine Menge von Hindernissen dargestellt. Es ist möglich, Wege innerhalb eines eingeschränkten Teilbereiches des Raumes zu berechnen, um so den Rechenaufwand zu reduzieren (Abbildung 6.4).

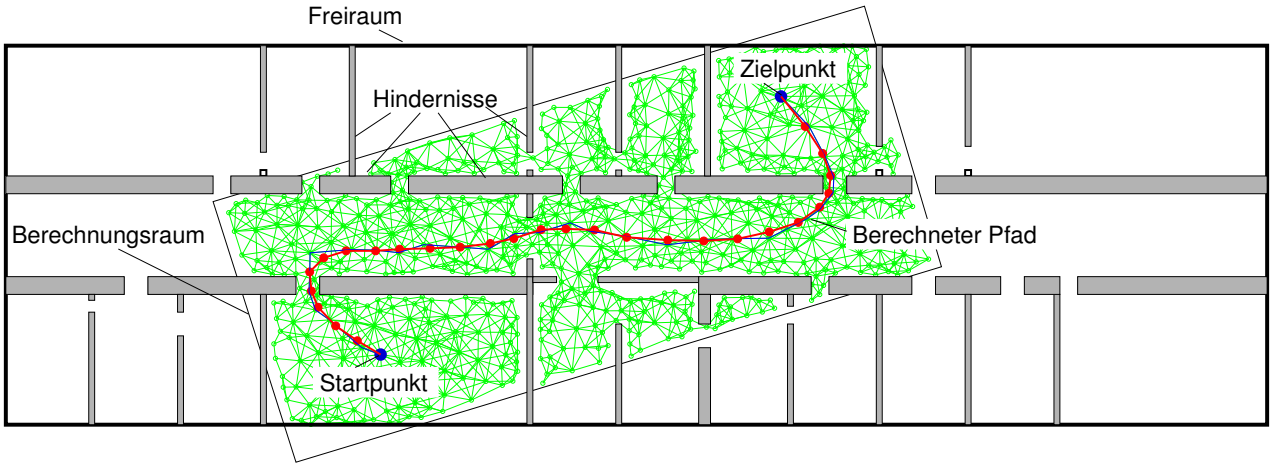
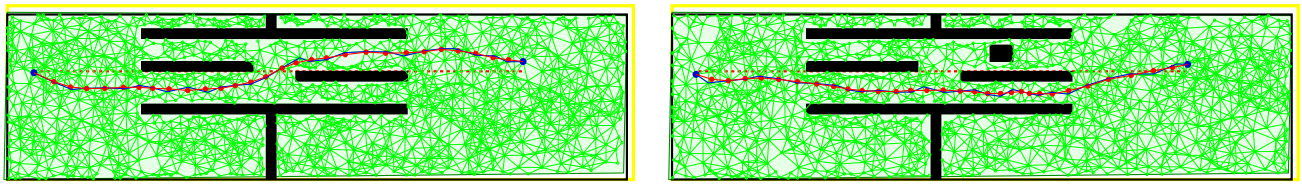


Abbildung 6.4: Beispiel der Pfadberechnung in einer zweidimensionalen Karte des Lehrstuhls für Technische Informatik.

Bei der Planung muss der gefundene Pfad zum Ziel im Freiraum liegen, wobei es günstig ist, einen Abstand zu Hindernissen zu wahren, da die Möglichkeit für kurzfristige Ausweichmanöver in der Nähe von Hindernissen stark eingeschränkt ist. Daher ist ein Kompromiss zwischen Hindernisabstand und optimaler Weglänge notwendig. Dies geschieht über die Gewichtungsfunktion aus Gleichung 4.34, die die Pfadabschnitte auch bezüglich ihrer Distanz zum nächsten Hindernis bewertet.

So ist beispielsweise in Abbildungen 6.5a und 6.5b ein Szenario mit zwei Gängen dargestellt. Beide Gänge sind in der Mitte verbunden; in der linken Hälfte ist der obere Gang enger als der untere Gang, während in der rechten Hälfte der untere Gang weniger Platz zum Manövrieren der Plattform zur Verfügung stellt als der obere. In Abbildung 6.5a berechnet das Planungsverfahren einen Weg, der in beiden Hälften durch den jeweils breiteren Gang führt. Dabei wechselt er in der Mitte den Gang, da in engen Fluren eher Probleme durch unbekannte Hindernisse zu erwarten sind als in breiteren Fluren. In Abbildung 6.5b ist der obere Gang in der rechten Hälfte teilweise versperrt. Das Verfahren plant deswegen den Weg durch den unteren Gang, der mehr Freiraum zum Manövrieren der Plattform anbietet.

[3] Auf eine Aktualisierung der Karte anhand der Sensordaten wurde verzichtet, da die Aktualisierung für das Testszenario unnötig ist.



(a) (b)

Abbildung 6.5: Pfadberechnung in einem Flur-Szenario.

6.5.2 Bestimmung einer möglichen Greifposition

Während der Ausführung des *Pick-and-Place* Szenarios kann es vorkommen, dass der mobile Manipulator nach der Detektion und Lokalisierung der Objekte relativ zur aktuellen Manipulatorposition feststellt, dass das Zielobjekt nicht erreichbar ist (Abbildung 6.6). Dann muss die deliberative Ebene eine neue Position für den mobilen Manipulator bestimmen, aus der der Greifvorgang möglich ist. Da der Roboter mit seiner Bordkamera die Objekte im Raum schon lokalisiert hat, kann diese Information zur Bestimmung einer neuen, erfolgversprechenden Position des mobilen Manipulators verwendet werden. Der genaue Ablauf dieses Verfahrens ist im Flussdiagramm in Abbildung 6.7 dargestellt.

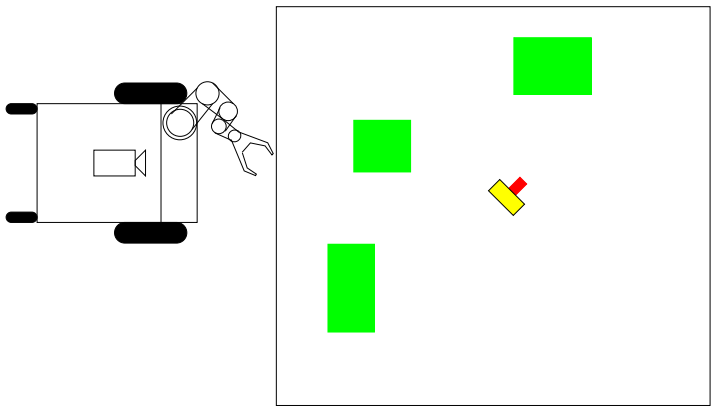


Abbildung 6.6: Plattformposition, aus der ein Greifvorgang nicht möglich ist.

Bei der Bestimmung der neuen Plattformposition wird zuerst eine Anfangsposition für den Greifer gesucht, aus der das Greifen möglich ist. Da das Objekt eine bestimmte Greifrichtung erfordert, brauchen nicht alle Anfangslagen des Greifers um das Objekt überprüft werden; die möglichen Lagen sind innerhalb der mit dem algorithmischen Teach-In definierten Region[4] beschränkt. Der hier implementierte Ansatz betrachtet zuerst Greiferlagen, die sich in der Mitte der trainierten Region befinden[5]; diese befinden sich auf einer geradlinigen Strecke, die einen Winkel von 0^0 mit der Ziellage des Greifers am Objekt bildet (Abbildung 6.8a, geradlinige Strecke 1). Wird keine geeignete Anfangslage für

[4]Es wird eine Region von 120^0 um das Objekt berücksichtigt.

[5]Diese Lagen erfordern die Bewegung entlang eines fast geradlinige Pfades, um die Ziellage am Objekt zu erreichen und können durch die Bewegung von nur drei Gelenken, nämlich des zweiten, dritten und fünften Gelenkes durchgeführt werden. Somit sind die Pfade robuster zu regeln.

0^0 gefunden, so überprüft der Ansatz Lagen, die sich auf einer geradlinigen Strecke mit Winkel

$$\phi_i = (-1)^i mod(\frac{i+1}{2})\phi_{NP} \tag{6.1}$$

befinden (Abbildung 6.8a, geradlinige Strecken 2 bis 9), wobei ϕ_{NP} ein fester Schrittwinkel[6] ist und die Funktion $mod()$ den Integer-Teil der Eingangsvariable zurückgibt. Es handelt sich also um einen iterativen Ansatz, wobei i sich um Eins erhöht bis eine Anfangslage gefunden wird, aus der ein Greifen erfolgversprechend ist, oder die Grenzen der Region erreicht sind.

Für ein bestimmtes ϕ_i liegen alle möglichen Anfangslagen auf einer geradlinigen Strecke mit Eckpunkten die Zielposition am Objekt und die Anfangposition des Greifers beim maximal möglichen Abstand[7] von der Zielposition. Für diese Strecke muss mindestens eine Position im Freiraum liegen. Dies ist beispielsweise für die geradlinige Strecke 1 in Abbildung 6.8a nicht der Fall.

Wurde für ein ϕ_i mindestens eine Lage im Freiraum gefunden[8], so muss für diese Anfangslage des Greifers die mobile Plattform eine entsprechende kollisionsfreie Position und Orientierung annehmen können. Angenommen, dass die Anfangslage des Greifers relativ zur Plattform fest definiert ist, dann ist anhand der Transformationsmatrix ${}^G_P\mathbf{T}$ des Koordinatensystems des Greifers zur mobilen Plattform die Plattformposition im Raum bekannt. Mit der Matrix ${}^W_T\mathbf{T}$ des Koordinatensystems des Tisches zur Welt richtet man den Roboter parallel zum Tisch aus und überprüft, ob in dieser Lage die Plattform im Freiraum liegt.

Auch aus der neuen Greiferanfangsposition können die Hindernisse den Greifvorgang stören, wie man beispielsweise in Abbildung 6.8a für die geradlinige Strecke 2 sehen kann. Deshalb ist eine Überprüfung des Greifvorgangs auf Kollisionen notwendig, bevor der mobile Manipulator die neue Position annimmt. Dafür werden die Lagen der Hindernisse und des Greifers relativ zur neuen Lage der Manipulator-Basis transformiert. Anhand der transformierten Hindernislagen kann man mit dem Planungsverhalten (siehe Unterkapitel 4.3) einen Pfad zur Endposition am Zielobjekt berechnen, wie das in Abbildung 6.8a für die geradlinige Strecke 3 der Fall ist. Ist dies nicht möglich, wie beispielsweise bei Strecken 1 und 2 in Abbildung 6.8a, dann wird die nächste Ausrichtung ϕ_i behandelt.

Anschließend sucht der Pfadplaner einen Pfad von der aktuellen Position der mobilen Plattform zur Zielposition (Abbildung 6.8b). Dabei wird ähnlich zum Planungsverhalten mit virtuellen Hindernissen sichergestellt, dass die mobile Plattform bei der Zielposition eine bestimmte Orientierung annimmt, so dass sie parallel zur nächsten Kante der Auflageebene ausgerichtet ist. Der ermittelte Pfad wird zum LLP weitergeleitet, der dessen Ausführung durch die mobile Plattform überwacht.

6.6 ComControl

Von den Komponenten der deliberativen Ebene tritt nur die ComControl nach außen in Erscheinung. Sie stellt die Schnittstelle zwischen der deliberativen Ebene und der Außenwelt, d.h. zwischen dem Operateur und der vermittelnden Ebene, dar. Sämtlicher Datenverkehr wird über die ComControl abgewickelt. Somit werden die Komponenten der deliberativen Ebene durch eine gemeinsame Schnittstelle nach außen abgekapselt, wodurch die Komponenten austauschbar bleiben, ohne dass sich an der

[6] Der Winkel hat hier einen Wert von 15^0.

[7] Dieser entspricht der maximalen Reichweite des Manipulators.

[8] Wurden mehrere Lagen ermittelt, dann kommt die am nächsten zum Tisch liegende zum Einsatz.

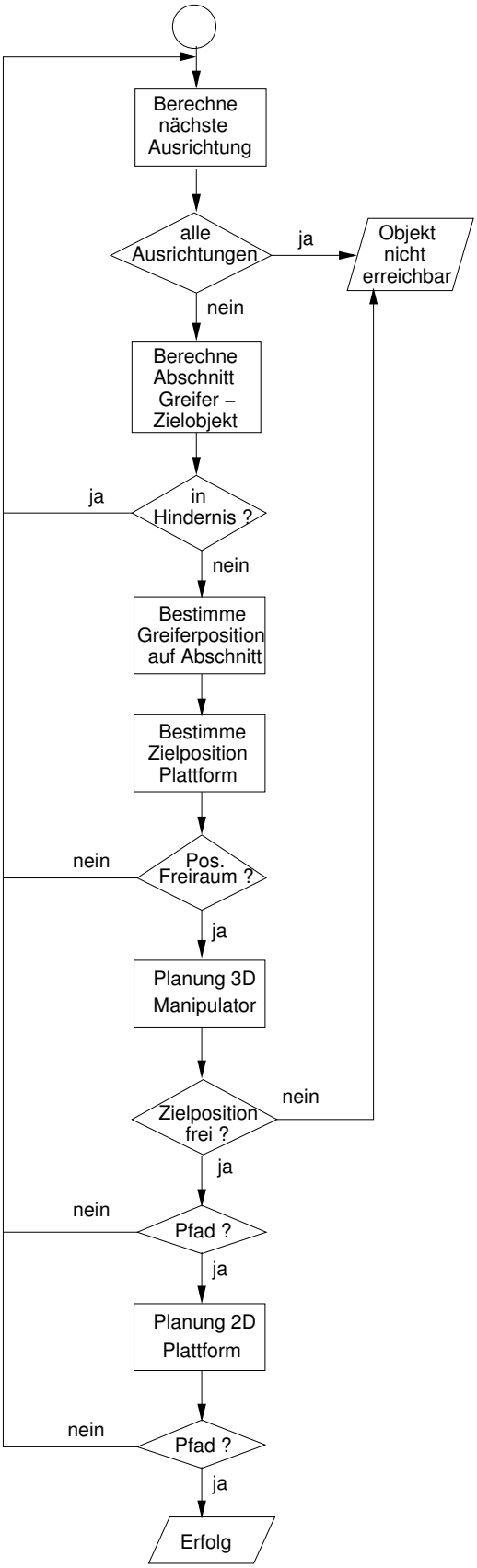
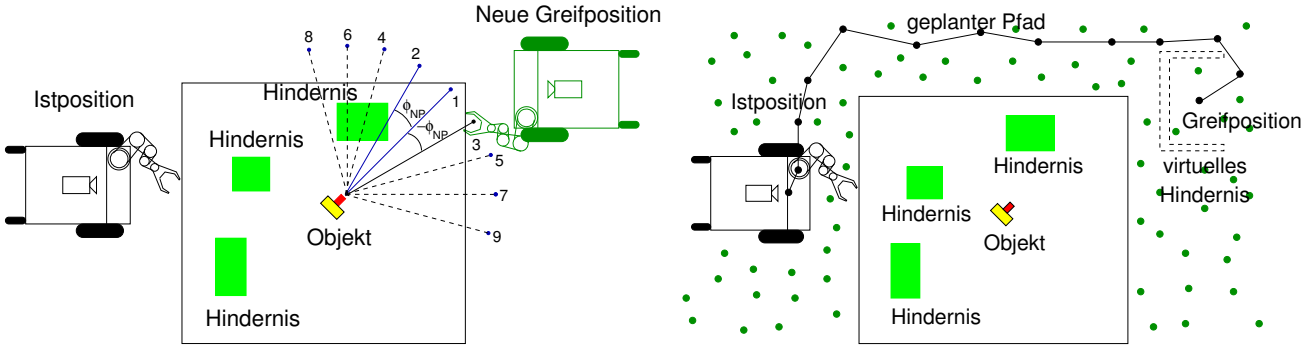


Abbildung 6.7: Flussdiagramm der Neuplanung für einen Greifvorgang.



(a) (b)

Abbildung 6.8: Ablauf der Neuplanung für einen Greifvorgang. Zuerst wird eine neuen Anfangsposition für den mobilen Manipulator bestimmt, aus der ein Greifen möglich ist (a). Anschließend sucht der Pfadplaner einen Pfad zur Zielposition (b). Dabei wird mit virtuellen Hindernissen (gestrichelte Linie) sichergestellt, dass die mobile Plattform bei der Zielposition eine bestimmte Orientierung annimmt, so dass sie parallel zur nächsten Kante der Auflageebene ausgerichtet ist.

Anbindung an das restliche System etwas ändert. Die ComControl unternimmt auch die Übersetzung von unterschiedlichen Formaten, da durch den heterogenen Aufbau des Systems kein einheitliches Format existiert.

Um die Austauschbarkeit der Komponenten der deliberativen Ebene zu gewährleisten, muss bei der Implementierung der Kommunikationswege auf sprachen- oder maschinenabhängige Datenstrukturen verzichtet werden. Daher erfolgt der Austausch von Daten über Strings. Der HLP gibt eine Folge von Teilplänen als eine Liste von Strings aus; auch der LLP schickt seine Anweisungen an die vermittelnde Ebene mittels Strings. Die Verwendung von Strings wird zusätzlich durch eine zweite Anforderung bedingt: den modularen Aufbau und die verteilte Ausführung der deliberativen Ebene auf unterschiedlichen Rechnern. Da Planer aufwendige Berechnungen durchführen und keinen kontinuierlichen Datenfluss zur vermittelnden Ebene haben, können die einzelnen Komponenten der deliberativen Ebene auf externen Rechnern betrieben werden und müssen nicht auf den Steuerungsrechnern der Plattform laufen. So haben die zeitkritischen reaktiven Verhalten und die Verhaltenskoordination mehr Rechenkapazität zur Verfügung. Zum Datenaustausch mit dem Roboter ist eine Netzwerkverbindung notwendig. Die einfachste Art der Kommunikation über eine solche Netzwerkverbindung findet mittels Strings statt, da sich kompliziertere Datenstrukturen nur schlecht übertragen lassen.

Wenn die Komponenten der deliberativen Ebene auf andere Rechner ausgelagert werden, muss die Kommunikation mit dem Roboter über ein Netzwerk erfolgen. Daher ist die Kommunikation zwischen den Komponenten über IP-Sockets (*Internet Protocol Sockets*) ausgeführt. Weiterhin muss die Kommunikation zwischen den Komponenten zuverlässig sein. Da die Kommunikationspartner feststehen und nur wenige gleichzeitige Verbindungen zwischen jeweils zwei Modulen aufgebaut werden, wird dem TCP (*Transmission Control Protocol*) der Vorzug vor dem paketorientierten UDP (*User Datagram Protocol*) gegeben. Um die Schnittstellen einheitlich zu halten, werden Sockets verwendet, auch wenn zwei Module auf dem gleichen Rechner laufen. Allerdings handelt es sich dann um die lokalen *Unix-Domain Sockets* (UDS).

Die ComControl übernimmt auch die koordinierte Ansprache von Plattform und Manipulator bei der Weitergabe der Planungsschritte an die vermittelnden Ebenen. Die Planungsschritte können bei

der Planerstellung im HLP und insbesondere im LLP mit zusätzlichen Attributen versehen werden, anhand derer die ComControl zuordnen kann, ob diese zeitlich nacheinander oder parallel erfolgen müssen. Parallel auszuführende Planungsschritte werden zugleich in den entsprechenden vermittelnden Ebenen aktiviert, wobei bei zeitlich sequentiell auszuführenden Planungsschritten die Aktivierung erst beim erfolgreichem Beenden der vorherigen Roboteraktion stattfinden kann. Dabei ist es die Aufgabe des Programmierers in den Teilplänen des LLP zu bestimmen, welche Planungsschritte parallel und welche sequentiell auszuführen sind.

6.7 Ausführung des Testszenarios

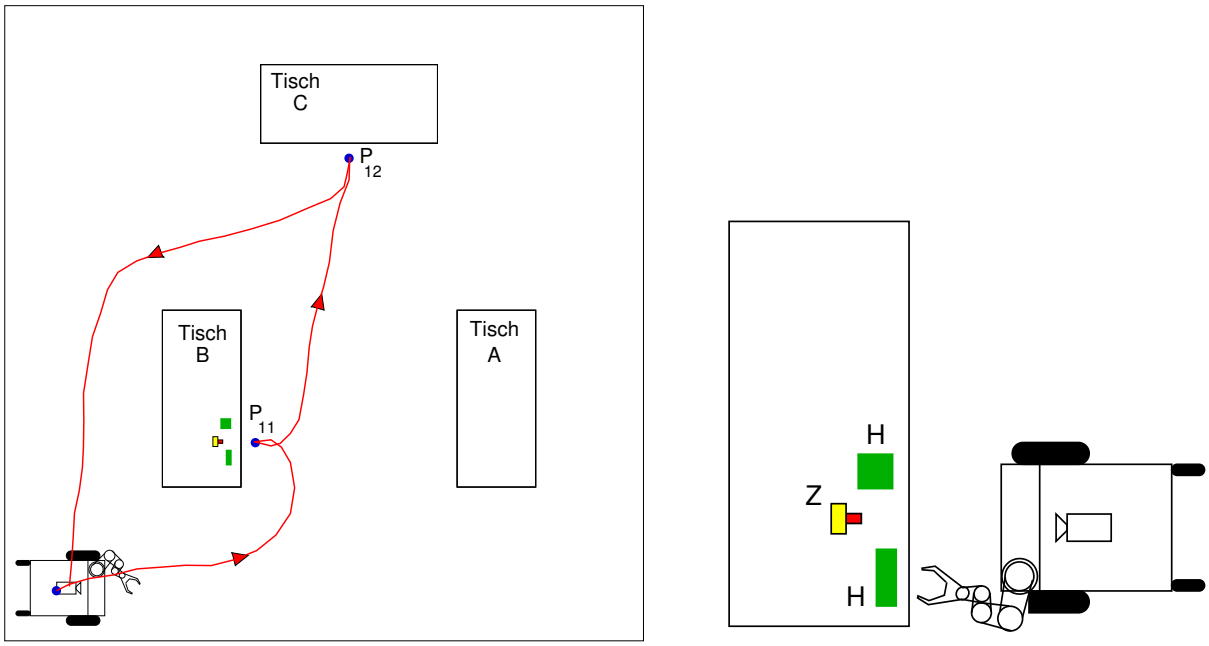
Das in dieser Arbeit entstandene System wurde mit vier unterschiedlichen Situationen beispielhaft getestet. Abbildungen 6.9 - 6.12 stellen diese Situationen sowie die Pfade des mobilen Manipulators dar, um die *Pick-and-Place* Aufgabe zu lösen. Der Roboterarm muss dabei das Zielobjekt, das auf einem der beiden Tischen A und B arbiträr positioniert ist, detektieren, greifen und zu Tisch C bringen. Sowohl Tisch A als auch Tisch B sind in zwei Bereiche eingeteilt (Abbildung 6.3). Der HLP bestimmt die Suchreihenfolge der Tischbereiche nach dem Zielobjekt anhand der Ortung der Objektes in vergangenen Missionen. Je öfter das Objekt in einem Bereich detektiert wurde, desto höher ist die Wahrscheinlichkeit, dass es sich bei der nächsten Mission im selben Bereich befindet. Zugleich werden Ortungen, die zeitlich erst vor kurzem stattgefunden haben, bei der Bestimmung der Suchreihenfolge höher bewertet als Objektdetektionen in älteren Missionen. Existiert keine Information über frühere Missionen, so fährt der mobile Manipulator die Positionen an den Tischen in einer festen Reihenfolge ab.

Im ersten Fall, der in Abbildung 6.9 dargestellt ist, führt die deliberative Ebene den mobilen Manipulator zuerst zu Position P_{11} an Tisch B, um nach dem Objekt zu suchen. Von dieser Position aus lokalisiert der mobile Manipulator das Zielobjekt zwischen zwei Hindernissen. Der Verhaltenskoordinationsmechanismus führt anschließend den Greifer kollisionsfrei zum Zielobjekt. Nach dem Greifvorgang fährt der Roboter zu Position P_{12} an Tisch C, legt dort das Objekt ab und kehrt zu seiner Anfangsposition zurück.

Die Situation in Abbildung 6.10 stellt höhere Ansprüche an den mobilen Manipulator. Der mobile Manipulator fährt zuerst zu Position P_{21}, da dort die letzte Ortung des Objektes stattgefunden hat. Da die Objektdetektion an dieser Position nicht erfolgreich ist, wird die nächste Position P_{22} abgearbeitet. Hier stellt das Planungsverhalten nach der Objektdetektion und der Lokalisierung fest, dass ein Greifen aus der gegenwärtigen Position des mobilen Manipulators nicht möglich ist. Deshalb stößt der LLP eine Planung nach Abschnitt 6.5.2 an, um eine günstige Anfangsposition für den Roboter zu bestimmen (Position P_{23}). Anschließend wird der Roboter dorthin geführt und initiiert einen neuen Greifversuch. Das Objekt wird anschließend zu Tisch C gebracht (Position P_{24}) und abgelegt.

Der Fall, dass ein Greifen von oben erforderlich ist, ist in Abbildung 6.11 dargestellt. Der mobile Manipulator sucht zuerst Tisch B ab[9]. Bei der Lokalisierung des Objektes wird festgestellt, dass der Griff des Objektes nach oben gerichtet ist. Dementsprechend werden ähnlich zum Planungsverhalten (Abschnitt 4.3) virtuelle Hindernisse um das Objekt platziert, um ein Greifen von oben zu erzwingen. Zugleich wird der Greifer um seine z-Achse um 90^0 rotiert, so dass die bildgestützte Zielführung ein

[9]Die Suchreihenfolge der Positionen P_{31} und P_{32} am Tisch wird dadurch festgelegt, dass das Objekt letztes Mal (Situation der Abbildung 6.10a in der Nähe von Position P_{31} (die der Position P_{22} in Abbildung 6.10a entspricht) an Tisch B geortet wurde.

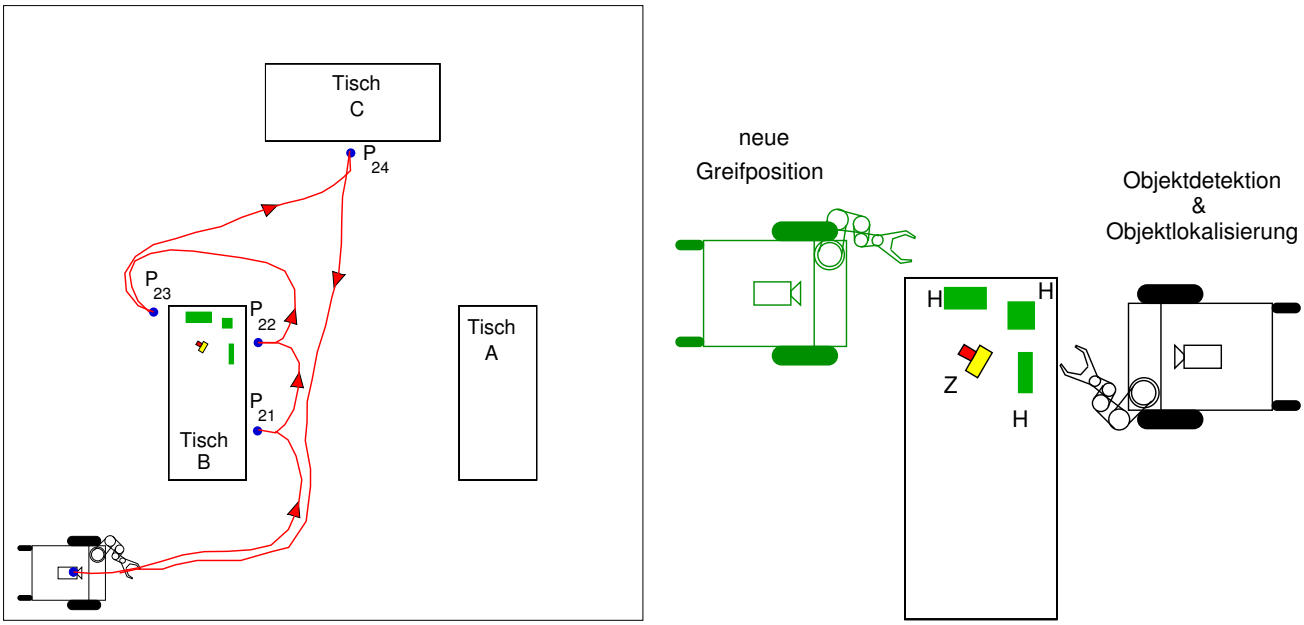


(a) (b)

Abbildung 6.9: Situation, die ein Greifen mit Hindernisvermeidung erfordert (a). Eingezeichnet ist auch der Pfad, den die Pfadplanung für die mobile Plattform ermittelt. In Abbildung b ist die Konstellation der Objekte auf dem Tisch genauer zu erkennen (H: Hindernisse, Z: Zielobjekt).

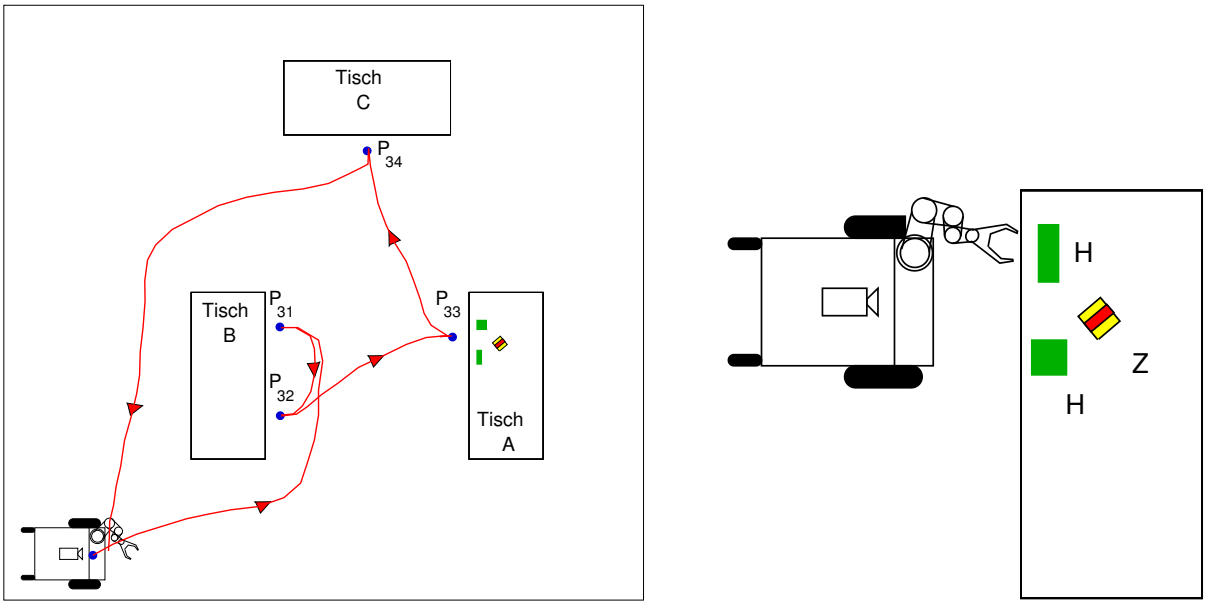
senkrechtes Greifen ausführen kann (Abbildung 3.15). Anschließend wird der Greifvorgang initiiert und das Objekt von oben gegriffen, um es zu Tisch C zu transportieren.

Abbildung 6.12 stellt einen Fall dar, wo das Greifen des Zielobjektes unmöglich ist. In diesem Fall meldet das Planungsverhalten nach der Lokalisierung des Objektes, dass der Greifer die Zielposition nicht erreichen kann. Auch der Planungsalgorithmus des LLPs, der eine alternative, geeignete Position für den mobilen Manipulator bestimmen soll, kann keine Greifposition bestimmen. Der mobile Manipulator wird deshalb direkt zur Anfangsposition zurückgeführt und beendet seine Mission mit einer Fehlermeldung.



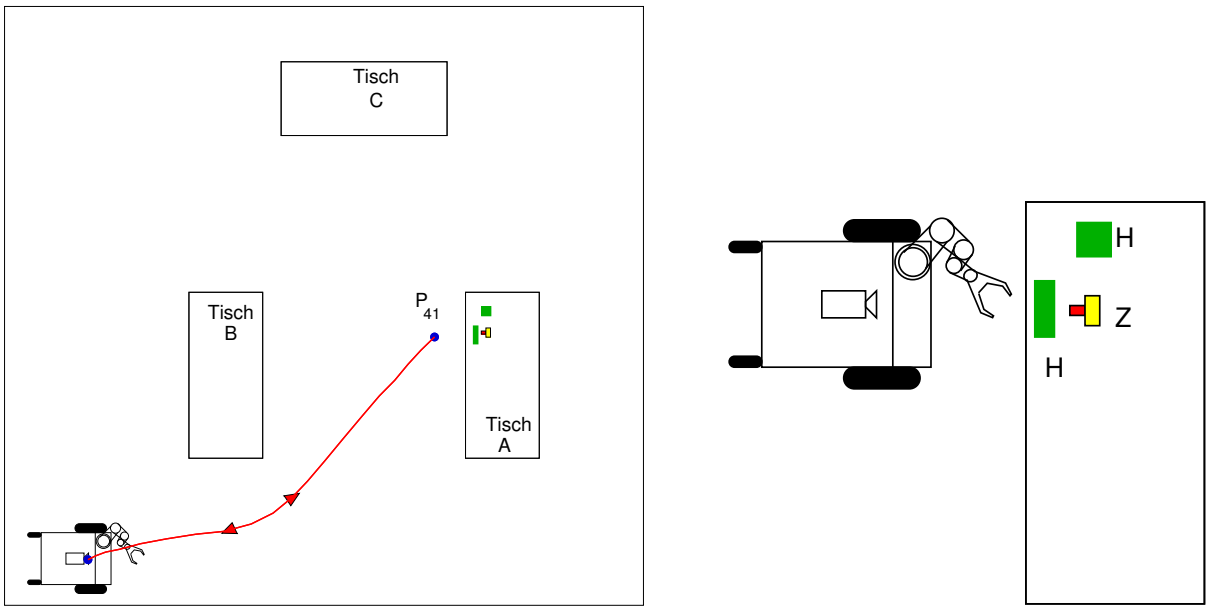
(a) (b)

Abbildung 6.10: Situation, bei der die Berechnung einer alternativen Anfangsposition für den mobilen Manipulator nötig ist (a). In b ist die Konstellation der Objekte auf dem Tisch genauer abgebildet



(a) (b)

Abbildung 6.11: Situation, die ein Greifen von oben erfordert (a) und Konstellation der Objekte auf dem Tisch (b).



(a) (b)

Abbildung 6.12: Situation, bei der ein Greifen des Zielobjektes nicht möglich ist (b). In diesem Fall wird der mobile Manipulator nach der Objektlokalisierung zur Anfangsposition zurückgeführt (a).

Kapitel 7

Zusammenfassung und Ausblick

Gegenstand dieser Arbeit ist das bildgestützte Greifen von Objekten in einer nur teilweise bekannten Umgebung unter Berücksichtigung von Hindernissen mit einem mobilen Manipulator. Weiterhin wird der Einsatz virtueller Umgebungen zum Teach-In der reaktiven, bildgestützten Verhalten sowie des Koordinationsmechanismus der Fertigkeiten untersucht. Die Schwierigkeit der Aufgabenstellung liegt dabei in der zunächst unbekannten Position von Zielobjekt und Hindernissen, die durch Bildverarbeitung detektiert und lokalisiert werden mussten. Die folgende, bildgestützte Ausführung des Greifvorgangs erfordert, dass sich der Manipulator dem Objekt aus einer optimalen Richtung annähert, wobei auf neue Bildinformation direkt zu reagieren ist, um Kollisionen zu vermeiden. Das System soll echtzeitfähig und in seiner Planung flexibel sein, um bei misslungenen Aktionen alternative Handlungsweisen zu wählen. Zur Evaluierung dient ein *Pick-and-Place*-Szenario. Der mobile Manipulator muss ein Objekt, das arbiträr auf einem Tisch positioniert ist, greifen und an einen anderen Ort transportieren. Dieses Objekt erfordert eine bestimmte Greifrichtung und ist von Hindernissen umgeben.

Das in dieser Arbeit vorgestellte Konzept zur mobilen Manipulation basiert auf zwei Ansätzen: einerseits wird eine erweiterte drei-Ebenen Architektur implementiert, um ein schnelles, situationsabhängiges Verhalten des mobilen Manipulators zu ermöglichen. Dabei existieren zwei reaktive und zwei vermittelnde Ebenen, je eine für Roboterarm und mobile Plattform. Andererseits werden die bildgestützten Komponenten der reaktiven und der vermittelnden Ebene in einer virtuellen Umgebung erlernt und anschließend auf den mobilen Manipulator übertragen.

Beim Roboterarm erlauben die bildgestützten Fertigkeiten der reaktiven Ebene eine schnelle Reaktion auf Änderungen der Umgebung und die Anpassung der Roboterbewegung an die neue Situation. Diese Verhalten setzen die Information der Roboterkameras ein, um den Manipulator bildgestützt zum Ziel zu führen, das Objekt im Sichtfeld der Greiferkamera zu halten oder den Greifer und die Manipulator-Segmente vor Kollisionen zu schützen. Sie sind entweder explizit programmiert oder mit neuronalen Netzen in der virtuellen Umgebung erlernt. Auf dieser Ebene wird auch ein lokaler Routenplaner als Verhalten eingesetzt. Dieser soll besonders zu Beginn einer Greifbewegung eine grobe Bewegungsrichtung vorgeben, während des Greifvorgangs den Manipulator vor Deadlocks bewahren und Konflikte zwischen Verhalten auflösen.

Die vermittelnde Ebene implementiert den Verhaltensaktivierungs- und Verhaltenskoordinationsmechanismus. Die Verhaltensaktivierung wählt nach dem *Arbitration*-Prinzip die Gruppen von parallel auszuführenden Fertigkeiten aus und aktiviert sie. Der Koordinationsmechanismus basiert auf dem *Superposition Command Fusion*-Prinzip und ist mit *Bayesian Belief Netzwerken* (BBN) realisiert.

Diese bestimmen die Anwendbarkeit der Verhalten in einer Situation und ermöglichen die gewichtete Interpolation der Ausgaben der aktiven Verhalten zum Roboter. Der Einsatz der BBN erlaubt das Lernen der Anwendbarkeit der Verhalten in verschiedenen Situationen, sowie die Modellierung der Fehlerwahrscheinlichkeit bei der Merkmalsextraktion aus den Bilddaten und erhöht dadurch die Robustheit des Koordinationsmechanismus.

Die mobile Plattform und der Manipulator werden über die deliberative Ebene koordiniert. Diese zerlegt eine Aufgabe in eine Reihenfolge von Planungsschritten für den Manipulator und die mobile Plattform. Hier arbeitet der *High-Level Planner* (HLP) strategische Pläne aus und setzt Zwischenziele für den mobilen Manipulator. Ein schnelles reaktives Planungssystem, der *Low-Level Planner* (LLP), zerlegt die Zwischenziele in vordefinierten Sequenzen von Planungsschritten. Der LLP umfasst auch einen probabilistischen Routenplaner, der mögliche Pfade für die mobile Plattform zum Lösen einer Aufgabe berechnet. Zu der deliberativen Ebene gehört zusätzlich ein Weltmodell mit einer geometrischen Karte für die Routenplanung und einer topologischen Darstellung der Umgebung für den HLP. Diese Aufteilung der deliberativen Ebene in den HLP und den LLP ermöglicht einerseits eine zielstrebige, globale Planung, die auch für komplexe Aufgaben Lösungen finden kann, und andererseits eine schnelle lokale Anpassung der existierenden Pläne auf der gegenwärtigen Umgebungssituation.

Ein weiterer Kernpunkt dieser Arbeit ist das Erlernen der gewünschten, bildgestützten Steuerungsalgorithmen des Roboterarmes anhand von Aufnahmen virtueller Kameras in einer virtuellen Umgebung mit einem Modell des realen Roboters. Nach der Trainings- und Evaluierungsphase findet ein Transfer der erlernten Algorithmen auf den realen Roboter statt und das System wird in der realen Umgebung eingesetzt. Dieser lernbasierte Ansatz bietet den Vorteil, dass er die Integration von Wissen, hier die bildgestützte Steuerung des Roboterarms, vereinfacht. Zugleich ermöglicht er die leichte Anpassung das Gesamtsystem an wechselnde Umweltbedingungen und Aufgaben ohne die Hardware zu gefährden. Zwei Verfahren wurden für die Automatisierung des Trainings in der virtuellen Umgebung vorgeschlagen. Für das Lernen von zielführenden Verhalten wurden objektspezifische Pfade anhand einer Vorgabe automatisch generiert und entlang der Pfade die Trainingsdaten gesammelt (algorithmisches Teach-In). Kollisionsvermeidende Verhalten sowie der Koordinationsmechanismus setzten zum Training *Reinforcement Learning*-Techniken ein (stochastisches Teach-In).

Mit dieser Arbeit ist ein System zur mobilen Manipulation in einer teilweise bekannten Umgebung unter Berücksichtigung von Hindernissen entstanden und innerhalb eines *Pick-and-Place* Szenarios erfolgreich getestet worden. Die realisierte drei-Ebenen Architektur ermöglicht den robusten parallelen Einsatz zielführender und kollisionsvermeidender Verhalten und die Kombination reaktiver und planender Fertigkeiten; bisher wurde bei existierenden mobilen Manipulatoren nur die sequentielle Ausführung der Komponenten der reaktiven Ebene implementiert. Dadurch macht sich das System die Vorteile sowohl der *Arbitration*- als auch der *Command Fusion*-Mechanismen zunutze. Weiterhin wurde der erfolgreiche Einsatz virtueller Umgebungen zum Teach-In der reaktiven, bildgestützten Verhalten sowie des Koordinationsmechanismus untersucht. Im Rahmen dieser Untersuchung wurden auch die erforderlichen Bildverarbeitungsschritte zum Abgleich der Daten der realen und der virtuellen Kameras festgelegt. Das Training in der virtuellen Umgebung mit dem algorithmischen und stochastischen Teach-In erlaubt dabei das effiziente Erlernen der Verhalten ohne die Hardware zu gefährden. Ein weiteres Novum dieser Arbeit ist das Training des Koordinationsmechanismus der Fertigkeiten, das mit dem stochastischen Teach-In in der virtuellen Umgebung stattfindet und das nicht in anderen Systemen vorgesehen ist.

In der reaktiven Ebene hat die bildgestützte Zielführung die Aufgabe, den Manipulator entlang eines objektspezifischen Pfades zur Greifposition am Objekt zu führen. Dadurch kann man einfach unter-

schiedliche Greifstrategien für unterschiedliche Objekte definieren und somit ein effizientes Greifen von Objekten ermöglichen. Ein weiterer Unterschied zu ähnlichen Arbeiten sind die implementierten Hindernisvermeidungsverhalten, die sowohl den Greifer als auch die Segmente des Manipulators berücksichtigen. Dabei wird für jedes Verhalten auch eine unterschiedliche Kamerakonfiguration verwendet: die *eye-in-hand* Kamera liefert die Daten für die Hindernisvermeidung des Greifers und die *eye-to-hand* Kamera der mobilen Plattform akquiriert die Daten für die Hindernisvermeidung der Manipulatorsegmente. Verhalten, die die Hindernisvermeidung für den Greifer über die *eye-in-hand* Kamera implementieren, sind bei anderen Systemen nicht vorhanden. Dabei kann die Greiferkamera eine größere Genauigkeit der Szene um den Greifer liefern , weist keine Verdeckungen durch Manipulatorsegmente auf und trägt somit bei der Robustheit der Hindernisvermeidung bei.

Die meisten hybriden Architekturen wenden keine Verhaltenskoordinierung bei einem Manipulator an; bestenfalls erlauben sie dem Programmierer den Koordinationsmechanismus für den Roboterarm selbst zu realisieren. In dieser Arbeit wird ein einheitliches Konzept zur Koordination der bildgestützten Verhalten des Roboterarmes präsentiert und mit einer *Pick-and-Place* Aufgabe getestet. Dabei handelt es sich um eine gewichtete Summierung der Verhaltensausgaben anhand des gegenwärtigen Sensorkontexts, die die situationsabhängige Fusion der Ausgaben reaktiver und planender Fertigkeiten ermöglicht und den Roboterarm in kritischen Bereichen effektiv vor Kollisionen schützt. Der Koordinationsmechanismus ist, unterschiedlich zu anderen Verfahren, leicht um neue Verhalten erweiterbar: Da jedes Verhalten ein eigenes BBN erhält, wird bei der Implementierung eines neuen Verhaltens lediglich ein neues BBN erzeugt und erlernt. Die BBNs der vorhandenen Verhalten werden dabei ohne Änderungen übernommen. Auf diese Weise entfällt ein komplettes neues Training bzw. ein Umstrukturieren der alten Komponenten. Der realisierte Ansatz erfordert auch kein aufwendiges Experimentieren zum Setzen der Parameter wie in anderen Arbeiten, da diese in einer virtuellen Umgebung erlernt werden. Im Koordinationsmechanismus ist auch die Unsicherheit der oft verrauschten Merkmale modelliert und im Gewichtungsprozess berücksichtigt.

Weiterführende Arbeiten könnten sich mit der robusteren Bildverarbeitung, Segmentierung und Erkennung der Objekte befassen. Dadurch kann der mobile Manipulator mit alltäglich benutzten Gegenständen ohne künstliche Markierungen in einer realen Umgebung eingesetzt werden. Es empfiehlt sich außerdem der Einsatz einer Stereokamera auf der Plattform, um mit der existierenden Hindernisvermeidung der Segmente auch bei bewegten Hindernissen ein Ausweichen des Manipulators zu ermöglichen. Weiterhin werden zur Zeit die Eingabemerkmale für die reaktive Verhalten noch manuell bestimmt. Es ist wünschenswert, die Merkmalsauswahl in der virtuellen Umgebung zu automatisieren und über statistische Kriterien, wie beispielsweise die *Mutual Information* [WT98], aus allen extrahierten Merkmalen die ausdruckreichsten für ein Verhalten auszuwählen und zum Training sowie zur Ansteuerung einzusetzen.

Die realisierten Fertigkeiten begrenzen den Einsatz des mobilen Manipulators nur für *Pick-And-Place* Aufgaben. Um weitere Aufgaben ausführen zu können, müssen neue Verhalten in die reaktive Ebene eingefügt werden. Solche Fertigkeiten könnten unter anderem das Verfolgen eines Objektes mit der Bordkamera, das Greifen einer Türklinke oder das Betätigen eines Schalters umfassen. In diesem Rahmen muss man auch die vermittelnde Ebene mit entsprechenden Planungsschritten erweitern. Der Verhaltenskoordinationsmechanismus kann auch erweitert werden, um die gleichzeitige, eng koordinierte Bewegung von Plattform und Manipulator bei Transportaufgaben zu ermöglichen. Weiterhin ist zu untersuchen, inwiefern das Training in der virtuellen Umgebung auch auf das Erlernen der BBN-Struktur zu erweitern ist. Dadurch kann ein höherer Automatisierungsgrad bei der Implementierung der vermittelnden Ebene erzielt werden.

Literaturverzeichnis

[ABD+98a] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.

[ABD+98b] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An Obstacle-Based PRM for 3D Workspaces. In P. K. Agarwal et al., editor, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundation of Robotics*, pages 155–168, Wellesley, MA, 1998. A. K. Peters.

[AC87] P. E. Agre and D. Chapman. PENGI : An implementation of a theory of activity. In *Proceedings AAAI-87*, San Mateo, CA, 1987.

[AC96] D. W. Aha and L. W. Chang. Cooperative Bayesian and Case-Based Reasoning for Solving Multiagent Planning Tasks. Technical Report AIC-96-005, Navy Center for Applied Research in Artificial Intelligence, Washington, DC, USA, January 1996.

[ACBGH97] R. Aylett, A. Coddington, D. Barnes, and R. Ghanea-Hercock. What Does a Planner Need to Know About Execution? In *Recent Advances in AI Planning, 4th European Conference on Planning (ECP'97)*, Lecture Notes in Computer Science, pages 26–38, Toulouse, France, September 1997.

[ACH97] O. Aycard, F. Charpillet, and J. P. Haton. A New Approach to Design Fuzzy Controllers for Mobile Robots Navigation. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics & Automation (CIRA97)*, Monterey, California, USA, July 10-11 1997.

[Alb75] J. S. Albus. A New Approach to Manipulator Control: The Cerebellas Model Articulations Controller (CMAC). *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 97:220–227, 1975.

[ALH+98] K. Arbter, J. Lagwald, G. Hirzinger, G. Wei, and P. Wunsch. Proven techniques for robust visual servo control. In *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop WS2 "Robust Vision for Vision-Based Control of Motion"*, pages 1–13, Leuven, Belgium, 1998.

[AML89] J. S. Albus, H. G. McCain, and R. Lumia. NASA/NBS standard reference model for telerobot control system architecture (NASREM). Technical report, NIST, Gaitherburg, 1989.

[ANH95] M. Asada, T. Nakamura, and K. Hosoda. Vision-based robot learning for behavior acquisition. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems 1995 (IROS '95), Workshop on Vision for Robots*, pages 110–115, Pittsburgh, PA, USA, 1995.

[Ark98] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, MA, USA, 1998.

[AUH99] M. Asada, E. Uchibe, and K. Hosoda. Cooperative Behavior Acquisition for Mobile Robots in Dynamically Changing Real Worlds via Vision-Based Reinforcement Learning and Development. *Artificial Intelligence*, 110:275–292, 1999.

[AW96] N. M. Amato and Y. Wu. A Randomized Roadmap Method for Path and Manipulation Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 113–120, Minneapolis, Minnesota, USA, 1996.

[Aya91] N. Ayache. *Artificial Vision for Mobile Robots*, chapter Calibration, pages 13–42. The MIT Press, 1991.

[AYH00] M. Asada, Y. Yoshikawa, and K. Hosoda. Learning by Observation without Three-Dimensional Reconstruction. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 555–560, Venice, Italy, 2000.

[BAB+00] M. Beetz, T. Arbuckle, T. Belker, M. Bennewitz, A. B. Cremers, D. Hähnel, and D. Schulz. Enabling Autonomous Robots to Perform Complex Tasks. *Künstliche Intelligenz (KI)*, 4:5–10, 2000.

[BAB+01] M. Beetz, T. Arbuckle, T. Belker, A. B. Cremers, D. Schulz, M. Bennewutz, W. Burgard, D. Hähnel, D. Fox, and H. Grosskreutz. Integrated, Plan-Based Control of Autonomous Robots in Human Environments. *IEEE Intelligent Systems*, pages 56–65, September/Oktomber 2001.

[Bag94] S. Bagchi. *Planning under Uncertainty by Spreading Activation Through an Adaptive Probabilistic Network*. PhD thesis, Faculty of Electrical Engineering of Vanderbilt University, 1994.

[Bag96] B. Baginski. The z^3-method for fast path planning in dynamic environments. In *Proceedings of the IASED Conference on Applications of Control and Robotics*, pages 47–52, Orlando, Florida, 1996.

[Bag98] B. Baginski. *Motion Planning for Manipulators with Many Degrees of Freedom - The BB-Method*. PhD thesis, Lehrstuhl für Robotik und Echtzeitsysteme, Fakultät für Informatik der Technischen Universität München, München, August 1998.

[BAH94] B. Brunner, K. Arbter, and G. Hirzinger. Task-directed Programming of Sensor-based Robots. In *IEEE/RSJ International Conference of Intelligent Robots and Systems*, pages 1080–1087, Munich, Germany, 12.-16. September 1994.

[BAH01] B. Brunner, K. Arbter, and G. Hirzinger. Aufgabenorientierte Fernprogrammierung von Robotern. *at - Automatisierungstechnik*, 49(07):312, 2001.

[BAHK95] B. Brunner, K. Arbter, G. Hirzinger, and R. Koeppe. Programming Robots via Learning by Showing in a Virtual Environment. In *Virtual Reality World 95*, pages 1080–1087, Stuttgart, Germany, February 1995.

[Bal97] T. Balch. Clay: Integrating Motor Schemas and Reinforcement Learning. Technical report, Georgia Institute of Technology, Atlanta, Georgia, 1997.

[BBD+00] A. Banerjee, P. Banerjee, T. DeFanti, A. Hudson, B. Dodds, and J.R. Curtis. A Behavioral Layer Architecture for Telecollaborative Virtual Manufacturing Operations. *IEEE Transactions on Robotics and Automation*, 16(3):218–227, June 2000.

[BCN+95] M. Bishay, M. E. Cambon, K. Negishi, R. A. Peters II, and K. Kawamura. Visual servoing in ISAC, a decentralized robot system for feeding the disabled. In *Proceedings of the 1995 International Symposium on Computer Vision*, pages 335–340, Coral Gables, Florida, USA, November 1995.

[BFG+97] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1):237–256, 1997.

[BG97] R. Bischoff and V. Graefe. Versuchsträger zur Entwicklung lernfähiger sehender Roboter. In *Workshop Selbstorganisation von adaptivem Verhalten (SOAVE'97)*, pages 55–64, Ilmenau, Germany, September 1997.

[BG98] R. Bischoff and V. Graefe. Machine vision for intelligent robots. In *Proceedings of the IAPR Workshop on Machine Vision Applications (MVA'98)*, pages 167–176, Makuhari/Tokyo, Japan, 1998.

[BG01] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[BH99] A. Billard and G. Hayes. DRAMA, a Connectionist Architecture for Control and Learning in Autonomous Robots. *Adaptive Behavior*, 7(1):35–63, 1999.

[BHK02] F. H. Broicher, C.-E. Hansjürgens, and K.-F. Kraiss. Verteilte Simulation von Gruppen kooperierender Roboter. In *Robotik 2002*, volume 1679 of *VDI-Berichte*, pages 289–294, Ludwigsburg, Germany, June 2002.

[BILM03] A. Bonarini, G. Invernizzi, H. Labella, and M. Matteucci. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134:101–115, 2003.

[BK95] R. P. Bonasso and D. Kortenkamp. Characterizing an Architecture for Intelligent, Reactive Agents. In *Working Notes of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, pages 29–34, Menlo Park, CA, USA, March 1995.

[BK98] O. Brock and O. Khatib. Executing motion plans for robots with many degrees of freedom in dynamic environments. In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, pages 1–6, Leuven, Belgium, May 1998.

[BK99] O. Brock and O. Khatib. Elastic strips: A framework for integrated planning and execution. In *6th International Symposium on Experimental Robotics*, pages 245–254, Sydney, Australia, 1999.

[BK00] O. Brock and L. E. Kavraki. Towards Real-time Motion Planning in High-dimensional Spaces. In *Proceedings of the International Symposium on Robotics and Automation (ISRA)*, pages 81–86, Monterrey, Mexico, November 2000.

[BKM+98] M. Becker, E. Kefalea, E. Maël, C. von der Malsburg, M. Pagel, J. Triesch, J. C. Vorbrüggen, and S. Zadel. GripSee: A Robot for Visually-Guided Grasping. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 1998)*, Skövde, Sweden, 1998.

[BKM+99] M. Becker, E. Kefalea, E. Maël, C. von der Malsburg, M. Pagel, J. Triesch, J. C. Vorbrüggen, R. P. Würtz, and S. Zadel. Gripsee: A Gesture-controlled Robot for Object Perception and Manipulation. *Autonomous Robots*, 6:203–221, 1999.

[BL99] A. L. Blum and J. C. Langford. Probabilistic planning in the graphplan framework. In *Proceedings of the Fifth European Conference on Planning (ECP99)*, pages 319–332, Durham, UK, 1999.

[BLL92] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):224–241, 1992.

[BM94] M. Beetz and D. McDermott. Improving Robot Plans During Their Execution. In *Proceedings of the 2nd International Conference on AI Planning Systems*, pages 7–12, Chicago, USA, 1994.

[BN94] M. R. Blackburn and H.G. Nguyen. Learning in Robotic Vision Directed Reaching: A Comparison of Methods. In *Proceedings of the 1994 Image Understanding Workshop*, pages 781–788, Monterey, CA, USA, 1994.

[BP98] W. Blase and J. Pauli. Vision-based Manipulator Navigation Using Mixtures of RBF Neural Networks. In *Proceedings of the International Conference on Neural Networks and Brain*, pages 531–534, Peking, China, 1998.

[Bro84] D. R. K. Brownrigg. The weighted median filter. *Communications of the ACM*, 27:807–818, 1984.

[Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 1:14–23, April 1986.

[Bro90] R. A. Brooks. Elephants Don't Play Chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[Bro91] R. A. Brooks. Intelligence without reason. In Morgan Kaufman, editor, *Proceedings of the 1991 International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991.

[BRS99] R. Basri, E. Rivlin, and I. Shimshoni. Image- based robot navigation under the perspective model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2578–2583, Detroit, Michigan, USA, 1999.

[BSP94] S. Brandt, C. Smith, and N. Papankolopoulos. The Minesota robotic visual tracker: A flexible testbed for vision-guided robotic research. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,"Humans, Information and Technology"*, pages 1363–1368, San Antonio, TX, USA, 1994.

[BU97] J. L. Buessler and J. P. Urban. Neural Networks for Visual Servoing in Robotics. Technical report, Control Neuromimètique et Parallèlisme, Groupe TROP, Facultè des Schiences et Techniques, Universitè de Haute-Alsace, 4 rue des Frères Lumière, 68093 Mulhouse Cedex, France, November 1997.

[CACE99] T. R. Collins, R. C. Arkin, M. J. Cramer, and Y. Endo. Field Results for Tactical Mobile Robot Missions. Technical report, Mobile Robot Laboratory, College of Computing, Georgia Institute of Technology, 1999.

[Can86] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[CB97] J. L. Crowley and F. Berard. Multi-modal tracking of faces for video communications. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 640–645, Puerto Rico, 1997.

[CD98] J. Carusone and G.M.T. D'Eleuterio. The "Feature CMAC": A Neural-Network-Based Vision System for Robotic Control. In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, pages 2959–2964, Leuven, Belgium, May 1998.

[CH98] P. C. Chen and Y. K. Hwang. SANDROS: A Dynamic Graph Search Algorithm for Motion Planning. *IEEE Transactions on Robotics and Automation*, 14(3):390–403, June 1998.

[Cha87] B. Chazelle. Approximation and Decomposition of Shapes. In T. Schwartz and C.-K. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1987.

[Cha91] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[Cha98] F. Chaumette. Potential problems of stability and convergence in image-based and position-based visual servoing. In D. Kreigman, G. Hager, and A. Morse, editors, *The Confluence of Vision and Control*, pages 66–78. Springer Verlag, 1998.

[Cha02] F. Chaumette. A first step towards visual servoing using image moments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2002*, pages 378–383, Lausanne, Switzerland, 2002.

[Cha04] F. Chaumette. Image Moments: A General and Useful Set of Features for Visual Servoing. *IEEE Transactions on Robotics*, 20(4):713–723, 2004.

[CHS95] T. Chadzelek, G. Hotz, and E. Schömer. Heuristic motion planning with many degrees of freedom. Technical report, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, August 1995.

[CK00] S. I. Choi and B. K. Kim. Obstacle avoidance control for redundant manipulators usind collidability measure. *Robotica*, 18:143–151, 2000.

[CKR97] H. Choset, I. Konukseven, and A. Rizzi. Sensor Based Planning: A Control Law for Generating the Generalized Voronoi Graph. In *Proceedings of the IEEE International Conference on Advanced Robotics*, pages 333–338, Monterey, CA, SUA, 1997.

[CL93] J. F. Canny and M. C. Lin. An opportunistic global path planner. *Algorithmica*, 10(2-4):102–120, 1993.

[Con89] J. H. Connell. A Behavior-Based Arm Controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, December 1989.

[Cor96] P. Corke. *Visual Control of Robots: High-Performance Visual Servoing*. Research Studies Press LTD, 1996.

[Cra89] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Series in Electrical and Computer Engineering: Control Engineering. Addison-Wesley Publishing Company, Inc., 2nd edition, 1989.

[Cro85] J. L. Crowley. Navigation for an Intelligent Mobile Robot. *IEEE Journal of Robotics and Automation*, 1(1):31–41, March 1985.

[CS00] G. Chronis and M. Skubic. Experiments in Programming by Demonstration: Training a Neural Network for Navigation Behaviors. In *Proceedings of the International Symposium on Robotics and Automation (ISRA)*, Monterrey, Mexico, November 2000.

[CV04] G. Chesi and A. Vicino. Visual Servoing for Large Camera Displacements. *IEEE Transactions on Robotics*, 20(4):724–735, 2004.

[dAT97] V. R. de Angulo and C. Torras. Self-calibration of a space robot. *IEEE Transactions on Neural Networks*, 8(4):951–963, 1997.

[DB94] R. Dearden and G. Boutilier. Integrating planning and execution in stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, pages 55–61, Stanford, CA, USA, 1994. AAAI-Press.

[DC99] T. Drummond and R. Cipolla. Visual tracking and control using Lie algebras. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition*, pages 652–657, Fort Collins, Colorado, USA, 1999.

[DC00] T. Drummond and R. Cipolla. Real-time tracking of mutliple articulated structures in multiple views. In *Proceedings of the 6th European Conference on Computer Vision*, pages 20–36, Trinity College, Dublin, Ireland, 2000.

[DF02] N. Dussault and E. Frigeri. iRobot introduces Roomba intelligent floorvac - the first automatic floor cleaner in the U.S. Press release, 2002. http://www.roombavac.com/news/pr091802-1.asp.

[DH55] J. Denavit and R. S. Hartenberg. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *ASME Journal of Applied Mechanics*, 22:215–221, 1955.

[Dig98] B. L. Digney. Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments. In *From Animals to Animats 5: The fifth Conference on the Simulation of Adaptive Behavior: SAB 98*, pages 321–330, University of Zurich, Zurich, Switzerland, 1998.

[Dij59] E. W. Dijkstra. A Note on two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Dil94] R. Dillmann. Programmieren durch Vormachen in der Robotik. In *18. Jahrestagung für Künstliche Intelligenz*, pages 396–401, Saarbrücken, Germany, 1994.

[Dil98a] R. Dillmann. *Robotik in Deutschland: Lehre, Forschung und Entwicklung*, chapter 1.3: Robotik, ein multidisziplinäres Forschungsgebiet. Shaker Verlag, 1998.

[Dil98b] R. Dillmann. *Robotik in Deutschland: Lehre, Forschung und Entwicklung*, chapter 1.4: Forschungsthemen. Shaker Verlag, 1998.

[DKKN93] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 574–579, Menlo Park, CA, USA, 1993. AAAI-Press.

[DR78] L. S. Davis and A. Rosenfeld. Noise cleaning by iterating local averaging. *IEEE Transactions on Systems, Man and Cybernetics*, 8(9):705–710, 1978.

[DRE+99] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zöllner, and M. Bordegoni. Learning Robot Behavior and Skills Based on Human Demonstration and Advice: The Machine Learning Paradigm. In *Proceedings of the 9th International Symposium of Robotics Research (ISRR'99)*, pages 229–238, Snowbird, Utah, USA, October 9-12 1999.

[Dyn] Exact Dynamics. Arm: Assistive Robotic Manipulator. Internet page. http://www.exactdynamics.nl/.

[EFHS98] E. Ettelt, R. Furtwängler, U. D. Hanebeck, and G. Schmidt. Design Issues of a Semi-Autonomous Robotic Assistant for the Health Care Environment. *Journal of Intelligent and Robotic Systems*, 22(3-4):191–209, July-August 1998.

[Ele01] Electrolux. Ultrasound helps world's frist automatic vacuum cleaner find its way. Press release, November 2001. http://trilobite.electrolux.se/presskit_en/index.asp.

[ERZD01] M. Ehrenmann, O. Rogalla, R. Zöllner, and R. Dillmann. Teaching Service Robots Complex Tasks: Programming by Demonstration for Workshop and Household Environments. In *Proceedings of the International Conference on Field and Service Robots (FSR)*, pages 397–402, Helsinki, Finnland, 2001.

[Fau93] O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. The MIT Press, 1993.

[FD94] J. Fröhlich and R. Dillmann. Sensorsimulation-based interactive telerobot control system. Technical report, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe (TH), 1994.

[FKPS95] R. J. Firby, R. E. Kahn, P. N. Prokopowicz, and M. J. Swain. An Architecture for Vision and Action. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 72–79, Montreal, Quebec, Canada, August 1995.

[FLM92] J. Feddema, C. Lee, and O. Mitchell. Model-based visual feedback control for a hand-eye coordinated robotic system. *Computer*, 25(8):21–31, 1992.

[FMB+97] W. Feiten, B. Magnussen, J. Bauer, G. D. Hager, and K. Toyama. Modeling and Control for Mobile Manipulation in Everyday Environments. In *Proceedings of the 1997 International Symposium on Robotic Research*, pages 123–128, Hayama, Japan, 1997.

[FPCR99] J. A. Fayman, P. Pirjanian, H. I. Christensen, and E. Rivlin. Exploiting Process Integration and Composition in the Context of Active Vision. *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 29(1):73–86, February 1999.

[FR00] E. Freund and J. Rossmann. Projective Virtual Reality: New Applications of VR-Technology in Space and Industry. In *Konferenzband der Robotik 2000 Fachtagung*, pages 359–364, Berlin, Germany, June 2000.

[FSC98] G. D. Finlayson, B. Schiele, and J. L. Crowley. Comprehensive Colour Image Normalization. In *Proceedings of the 5th European Conference on Computer Vision*, volume 1407 of *Lecture Notes in Computer Science*, pages 475–490, Freiburg, Germany, 1998.

[FSZ00] Y. Fu, R. Sharma, and M. Zeller. Vision-based Motion Planning for a Robot Arm Using Topology Representing Networks. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 1900–1905, San Francisco, CA, USA, April 2000.

[FW91] B. Fink and H.-D. Wend. Schnelle Bahnplanung für Industrieroboter mit veränderlichem Arbeitsraum. *at - Automatisierungstechink*, 39(6):197–294, 1991.

[FZ02] M. Frech and J. Zhang. Learning cooperative grasping with the graph representation of a state-action space. *Robotics and Autonomous Systems*, 38:183–195, 2002.

[Gat92] E. Gat. Integrating Planning and Reacting in an Heterogenous Asynchronous Architecture for Controlling Real-world Mobile Robots. In *Proceedings of the AAAI'92*, pages 809–815, San Jose, CA, USA, 1992.

[Gat98] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT/AAAI Press, 1998.

[GFZ00] C. Gaskett, L. Fletcher, and A. Zelinsky. Reinforcement Learning for a Vision Based Mobile Robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2000)*, Takamatsu, Japan, October 2000.

[GKG96] R. Glasius, A. Komofa, and S. Gielen. A biologically inspired neural net for trajectory formation and obstacle avoidance. *Biological Cybernetics*, 84:511–520, 1996.

[GL98] G. De Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. Technical report, Department of Computer Science, Univ. of Toronto, 1998.

[GL00] H. Grosskreutz and G. Lakemeyer. cc-golog: Towards More Realistic Logic-Based Robot Controllers. In *Proceedings of the 7th National Conference on Artificial Intelligence and the 12th Conference on Innovation Applications of Artificial Intelligence (AAAI/IAAI)*, pages 476–482, Austin, TX, USA, 2000.

[GL02] M. Garber and M. C. Lin. Constraint-based motion planning for virtual prototyping. In *Proceedings of the seventh ACM Symposium on Solid Model and Applications*, pages 257–264, Saarbrücken, Germany, June 2002.

[Gla91] B. Glavina. *Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichterter Suche und zufallsgesteuerter Zwischenzielerzeugung*. PhD thesis, Technische Universität München, 1991.

[GLL97] G. De Giacomo, Y. Lespérance, and H. J. Levesque. Reasoning about Concurrent Execution, Prioritized Interrupts, and Exogenous Actions in the Situation Calculus. In Martha Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1221–1226, San Francisco, CA, USA, 1997. Morgan Kaufmann.

[GMB97] R. Garcia-Martinez and D. Borrajo. Planning, Learning, and Executing in Autonomous Systems. In *Proceedings of the 4th European Conference on Planning (ECP 97)*, Recent Advances in AI Planning, pages 208–220, Toulouse, France, 1997.

[GMOS96] E. Grosso, G. Metta, A. Oddera, and G. Sandini. Robust visual servoing in 3d reaching tasks. *IEEE Transactions on Robotics and Automation*, 12(5):732–742, 1996.

[GSV01] N.R. Gracias and J. Santos-Victor. Trajectory reconstruction with uncertainty estimation using mosaic registration. *Robotics and Autonomous Systems*, 35:163–177, 2001.

[GVTM99] D. Gracanin, K.P. Valavanis, N.C. Tsourveloudis, and M. Matijasevic. Virtual-environment-based navigation and control of underwater vehicles. *IEEE Robotics & Automation Magazine*, 6(2):53–62, June 1999.

[HA94] K. Hosoda and M. Asada. Versatile visual servoing without knowledge of true jacobian. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 186–193, Munich, Germany, 1994.

[Hag95] G. D. Hager. Calibration-free visual control using projective invariance. In *Proceedings of the International Conference on Computer Vision*, pages 1009–1015, Cambridge, MA, USA, 1995.

[Hag97] G. D. Hager. A Modular System for Robust Positioning Using Feedback from Stereo Vision. *IEEE Transactions on Robotics and Automation*, 13(4):582–595, 1997.

[Has96] J.-M. Hasemann. *A control architecture for intelligent robots based on graph manipulation for planning, monitoring, execution, intention switching and fault recovery*. PhD thesis, Department of Electrical Engineering, University of Oulu, Oulu, Finland, 1996.

[Hay94] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan/IEEE Press, 1994.

[HB01] M. Hans and W. Baum. Concept of a Hybrid Architecture for Care-O-bot. In *Proceedings of the IEEE International Conference on Robot and Human Interaction, RO-MAN 2001*, pages 407–411, Bordeaux-Paris, France, September 2001.

[HDE98] R. Horaud, F. Doraika, and B. Espiau. Visually Guided Object Grasping. *IEEE Transactions on Robotics and Automation*, 14(4):525–532, 1998.

[Hec95] D. H. Heckerman. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, WA, March 1995.

[Hei00] J. Heikkilä. Geometric Camera Calibration Using Circular Control Points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1066–1077, October 2000.

[HH00] H. Hirukawa and I. Hara. Web-Top Robotics. *IEEE Robotics and Automation Magazine*, 7(2):40–45, June 2000.

[HHC96] S. Hutchinson, F. Hager, and P. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotcs and Automation*, 12(5):651–670, 1996.

[HHF+95] B. Hine, P. Hontalas, T. Fong, L. Piguet, E. Nygren, and A. Kline. Vevi: A Virtual Environment Teleoperations Interface for Planetary Exploration. In *SAE 25th International Conference on Environmental Systems*, San Diego, CA, USA, July 1995.

[HKLR00] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized Kinodynamik Motion Planning with Moving Obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2000.

[HLBS96] G. Hirzinger, K. Landzettel, B. Brunner, and B.M. Steinmetz. Steuerungskonzepte für internes und externes Roboter-Servicing in der Raumfahrt. In *Deutscher Luft- und Raumfahrtkongress*, Dresden, 24.-27. September 1996.

[HLS99] D. Hsu, J. C. Latombe, and S. Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Proceedings of the IEEE International Symposium in Assembly and Task Planning*, pages 280–285, Porto, Portugal, 1999.

[HNI01] K. Hashimoto, A. Namiki, and M. Ishikawa. A Visuomotor Control Architecture for High-Speed Grasping. In *Proceedings of the 40th IEEE Conference on Decision and Control*, pages 15–20, Orlando, Florida, USA, December 2001.

[HNS01] M. Hägele, J. Neugebauer, and R. D. Schraft. From Robots to Robot Assistants. In *Proceedings of the 32nd International Symposium on Robotics (ISR)*, pages 404–409, Seoul, Korea, 19-21 April 2001.

[HOB+04] U. Hillebrand, C. Ott, B. Brunner, C. Borst, and G. Hirzinger. Towards service robots for the human environment: the Robutler. In *Proceedings Mechatronics & Robotics (MechRob 2004)*, pages 1497–1502, Aachen, Germany, 2004.

[HSA95] K. Hosoda, K. Sakamoto, and M. Asada. Trajectory Generation for Obstacle Avoidance of Uncalibrated Stereo Visual Servoing without 3D Reconstruction. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems 1995*, pages 29–34, Pittsburgh, Pennsylvania, USA, August 1995.

[HST94] T. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom - random reflections at c-space obstacles. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 3318–3323, San Diego, California, USA, 1994.

[HSvdSS94] T. Hesselroth, K. Sarkar, P. van der Smagt, and K. Schulten. Neural networks control of a pneumatic robot arm. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(1):28–38, 1994.

[HT98] G. Hager and K. Toyama. The XVision system: A general-purpose substrate for portabe real-time vision applications. *Computer Vision and Image Understanding*, 69(1):23–37, 1998.

[HT01] A. Hourtash and M. Tarokh. Manipulator Path Planning by Decomposition: Algorithm and Analysis. *IEEE Transactions on Robotics and Automation*, 17(6):842–856, 2001.

[HV97] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In W. L. Lewis and B. Hayes-Roth, editors, *Proceedings of the 1st International Conference on Autonomous Agents*, pages 363–370, New York, USA, February 5-8 1997. ACM-Press.

[HV98] K. Z. Haigh and M. M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Journal of Autonomous Robots*, 5(1):79–95, March 1998.

[IBT+02] I. Iossifidis, C. Bruckhoff, C. Theis, C. Grote, C. Faubel, and G. Schöner. Cora:An Anthropomorphic Robot Assistant for Human Environment. In *Proceedings of the 2002 IEEE International Workshop on Robot and Human Interactive Communication*, pages 392–398, Berlin, Germany, September 25-27 2002.

[Inv] Invacare. Internet home page. http://www.invacare.com.

[IS01] I. Iossifidis and A. Steinhage. Controlling an 8 DOF Manipulator by Means of Neural Fields. In *Proceedings of the 3rd International Conference on Field and Service Robotics*, Helsinki, Finland, 2001.

[IS04] I. Iossifidis and G. Schöner. Autonomous reaching and obstacle avoidance with the anthropomorphic arm of a robotic assistant using the attractor dynamics approach. In *Proceedings of the IEEE 2004 International Conference on Robotics and Automation*, New Orleans, USA, April 26 - May 1 2004.

[JN96] M. Jägersand and R. Nelson. On-line Estimation of Visual-Motor Models using Active Vision. In *Proceedings of the ARPA Image Understanding Workshop 96*, Palm Springs, CA, USA, 1996.

[JU02] R. Jiang and R. Unbehauen. Robot Visual Servoing With Iterative Learning Control. *IEEE Transaction on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(2):281–287, March 2002.

[JV94] M. Jones and D. Vernon. Using neural netwoks to learn hand-eye co-ordination. *Neural Computing & Applications*, 2:2–12, 1994.

[Jäg95] M. Jägersand. Perception Level Control for Uncalibrated Hand-Eye Coordination and Motor Actions. Technical report, Dept. of Computer Science, University of Rochester, May 1995.

[KAHW00] K. Kawamura, A. Alford, K. Hambuchen, and M. Wilkes. Towards a Unified framework for Human-Humanoid Interaction. In *Proceedings of the first IEEE/RAS International Conference on Humanoid Robots*, Cambridge, MA, USA, 2000.

[KB94] J. Kosecka and R. Bajcsy. Diskrete Event Systems for Autonomous Mobile Agents. *Robotics and Autonomous Systems*, 12:187–198, 1994.

[KBC+02] O. Khatib, O. Brock, K.-S. Chang, F. Conti, D. Ruspini, and L. Sentis. Robotics and Interactive simulation. *Communications of the ACM*, 45(3):46–51, March 2002.

[KC99] D. Kragiè and H. Christensen. Integration of visual cues for active tracking on an end-effector. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 362–368, Kyongju, Korea, 1999.

[KC00] D. Kragić and H. I. Christensen. A Framework for Visual Servoing Tasks. In E. Pagello et al., editor, *Proceedings of the International Conference of Intelligent Autonomous Systems 6*, pages 835–842, Venice, Italy, 2000. IOS Press.

[KC02] D. Kragic and H. I. Christensen. Model-Based Techniques for Robotic Servoing and Grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[KD95] M. Kaiser and R. Dillmann. Hierarchical learning of efficient skill application for autonomous robots. In *International Symposium on Intelligent Robotic Systems (SIRS'95)*, Pisa, Italy, 1995.

[KD96] M. Kaiser and R. Dillmann. Building Elementary Robot Skills from Human Demonstration. In *IEEE Interantional Conference on Robotics and Automation*, pages 2700–2705, Minneapolis, Miennesota, USA, April 1996.

[KH95] R. Koeppe and G. Hirzinger. Learning complienat motions by task-demonstration in virtual environments. In *Fourth International Symposium on Expermental Robotics*, pages 299–307, Stanford, CA, USA, 1995.

[Kha98] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *International Journal of Robotics Research*, 5(1):90–98, 1998.

[KHW93] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. Technical Report 93-06-04, University of Washington, Seattle, USA, 1993.

[KL94a] L. Kavraki and J.-C. Latombe. Randomized Preprocessing of Configuration Space: Articulated Robots. In *Proceedings of the 1994 IEEE/RSI/GI International Conference on Intelligent Robots and Systems*, pages 1764–1771, München, Germany, September 1994.

[KL94b] L. Kavraki and J.-C. Latombe. Randomized Preprocessing of Configuration Space for Fast Path Planning. In *Proceedings of the 1994 International Conference on Robotics and Automation*, pages 2138–2145, San Diego, California, USA, May 1994.

[KM00] K. Konolige and K. Myers. The Saphira architecture for autonomous mobile robots. In *Artificial Intelligence and Mobile Robots - case studies of successful robot systems*, pages 211–242. AAAI-Press, 2000.

[KNHD97] J. Köhler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference on Planning*, pages 273–285, Toulouse, France, 1997.

[KPC02] D. Kragić, L. Petersson, and H. I. Christensen. Visually guided manipulation tasks. *Robotics and Autonomous Systems*, 40:193–203, 2002.

[Kra01] D. Kragić. *Visual Servoing for Manipulation: Robustness and Integration Issues*. PhD thesis, Royal Institute of Technology, Numerical Analysis and Computer Science, Computational Vision and Active Perception Laboratory, Stockholm, Sweden, 2001.

[Kra03a] K.-F. Kraiss. Mensch-Maschine Systeme I. Vorlesungsmanuskript, Lehrstuhl für Technische Informatik, RWTH Aachen, Germany, 2003.

[Kra03b] K.-F. Kraiss. Mensch-Maschine Systeme II. Vorlesungsmanuskript, Lehrstuhl für Technische Informatik, RWTH Aachen, Germany, 2003.

[KS98] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *Proceedings of the AIPS98 Workshop on Planning as Combinatorial Search*, pages 58–60, Pittsburgh, Pennsylvania, USA, 1998.

[KTP98] N. Katevas, S. G. Tzafestas, and C. G. Pnevmatikatos. The Aproximate Cell Decomposition with Local Node Refinement Global Path Planning Method: Path Nodes Refinement and Curve Parametric Interpolation. *Journal of Intelligent and Robotic Systems*, 22:289–314, 1998.

[Kug94] D. Kugelmann. Autonomous robotic handling applying sensor systems and 3D simulation. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 196–201, San Diego, California, USA, 1994.

[KvdKG90] B. J. A. Kröse, M. J. van der Korst, and F. C. A. Groen. Learning strategies for a vision based neural controller for a robot arm. In *Proceedings of the IEEE International Workshop on Intelligent Motor Control*, pages 199–203, Istanbul, Turkey, 1990.

[KVZ+04] S. Knoop, S. Vacek, R. Zöllner, C. Au, and R. Dillmann. A CORBA-Based Distributed Software Architecture for Control of Service Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sedai, Japan, 2004.

[Kär03] Kärcher. RC3000 Robo Cleaner: An electronic self-sufficient cleaning system that sweeps and cleans carpets and hard floors. Press release, September 2003. http://www.karcher.com.au/press-releases2003.htm#RC3000.

[Lat99] J.-C. Latombe. Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, November 1999.

[Lay00] K. Lay. Morpha: Intelligente anthropomorphe Assistenzsysteme - Die Interaktion zwischen Mensch und mobilen Assistenzsystemen als grundlegende Variante der Mensch-Technik-Interaktion. *it+ti - Informationstechnik und Technische Informatik*, 42(1):38–43, 2000.

[LP81] T. Lozano-Pérez. Automatic Planning of Manipulator Transfer Movements. *IEEE Transactions on Systems, Man and Cybernetics*, 11(10):681–698, 1981.

[LP83] T. Lozano-Pérez. Statial planning: A configuration space approach. *IEEE Transactions on Computer*, 32(2):108 – 120, 1983.

[LR99] H. Levesque and R. Reiter. High-level robotic control: beyond planning (Position paper). In *Papers from the AAAI Fall Symposium on Cognitive Robotics*, pages 106–108, 1999.

[MA99] D.C. MacKenzie and R.C. Arkin. Behavior-Based Mobile Manipulation for Drum Sampling. Technical report, College of Computing, Georgia Institute of Technology, 1999.

[Mae90] P. Maes, editor. *Designing Autonomous Agents*, pages 49–70. The MIT Press, Cambridge, Massachusets, USA, 1990.

[Mat94] M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1994.

[Mat01] M. J. Matarić. Learning in behavior-based multi-robot systems: policies, models, and other agents. *Journal of Cognitive Systems Research*, 2:81–93, 2001.

[MC96] A. McLean and S. Cameron. The virtual springs method: Path planning and collision avoidance for redundant manipulators. *International Journal of Robotics Research*, 15(4):300–319, 1996.

[MC00] É. Marchand and N. Courty. Image-Based Virtual Camera Motion Strategies. In *Proceedings of Graphics Interface, GI2000*, pages 69–76, Montreal, Quebec, May 2000.

[MC01] Y. Mezouar and F. Chaumette. Path Planning for Robust Image-based Visual Servoing. Technical report, Institut de Recherche en Informatique et Systémes Aléatoires, Projet Vista, 2001.

[MCB98] E. Malis, F. Chaumette, and S. Boudet. Positioning a coarse-calibrated camera with respect to an unknown object by 2D 1/2 visual servoing. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 1352–1359, Leuven, Belgium, May 1998.

[MCB00] E. Malis, F. Chaumette, and S. Boudet. Multi-Cameras Visual Servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3183–3188, San Fransisco, CA, USA, 2000.

[McC68] J. McCarthy. Situations, actions and causal laws. In M. Minsky, editor, *Semantic Information Processing*. MIT Press, Cambridge, MA, USA, 1968.

[McD81] M. J. McDonnell. Box-fitlering techniques. *Computer Graphics and Image Processing*, 17:65–70, 1981.

[MHGK90] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft. Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller. *IEEE Transactions on Robotics and Automation*, 6(1):1–9, 1990.

[MMK95] J. Matas, R. Marik, and J. Kittler. On representation and matching of multi-coloured objects. In *Proceedings of the fifth International Conference on Computer Vision*, pages 726–732, MIT, Cambridge, MA, USA, June 1995.

[MNPW98] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1/2):5–47, 1998.

[MZBK02] A. Matsikis, T. Zoumpoulidis, F. Broicher, and K.-F. Kraiss. Learning Object-specific Vision-based Manipulation in Virtual Environments. In *IEEE Proceedings of the 11th International Workshop on Robot and Human Interactive Communication ROMAN 2002*, pages 204–210, Berlin, Germany, September 2002.

[NC95] F. Noreils and R. Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions of Robotics and Automation*, 11(2):255–266, 1995.

[Nel01] B. J. Nelson. Assimilating disparate sensory feedback within virtual environments for telerobotic systems. *Robotics and Autonomous Systems*, 36:1–10, 2001.

[Nev00] S. Nevstruyev. VR-Modellierung des MANUS-Manipulators inklusive der Kinematik (offline/online). Technical report, Chair of Technical Computer Science, RWTH-Aachen, Aachen, Germany, April 2000.

[Nil69] N. J. Nilsson. A Mobile Automation: An Application of Artificial Intelligence Techniques. In *Proceedings of the first International Joint Conference on Artificial Intelligence*, pages 509–520, Washington D.C., USA, May 1969.

[Nil80] N. J. Nilsson. *Principles of Artificial Intelligence*. Troga Pub. Co., Palo Alto, CA, USA, 1980.

[NK95] B. Nelson and P. Koshla. An extendable framework for exceptation-based visual servoing using environmental models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 184–189, Nagoya, Aichi, Japan, 1995.

[NM02] M. N. Nicolescu and M. J. Matarič. A Hierarchical Architecture for Behavior-Based Robots. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS02)*, pages 227–233, Bologna, Italy, Juli 2002.

[NODP01] S. Nefti, M. Oussalah, K. Djouani, and J. Pontnau. Intelligent Adaptive Mobile Robot Navigation. *Journal of Intelligent and Robotic Systems*, 30(4):311–329, April 2001.

[NRT95] H. Noser, O. Renault, and D. Thalmann. Navigation for digital actors based on synthetic vision memory and learning. *Computer & Graphics*, 19(1):7–19, 1995.

[NY97] K. Nagatani and S. Yuta. Autonomous Mobile Robot Navigation Including Door Opening Behavior - System Integration of Mobile Manipulator to Adapt Real Environment. In *Proceedings of the International Conference on Field and Service Robotics*, pages 208–215, Camberra, Australia, 1997.

[OY82] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1982.

[PAKC00] L. Petersson, D. Austin, D. Kragić, and H. I. Christensen. Towards an Intelligent Service Robot System. In E. Pagello et al., editor, *Proceedings of the International Conference of Intelligent Autonomous Systems 6*, pages 835–842, Venice, Italy, 2000. IOS Press.

[Pau98] M. Pauly. *Ferninspektion mit mobiler Sensorik*. PhD thesis, Lehrstuhl für Technische Informatik, Fakultät für Elektrotechnik, RWTH-Aachen, 1998.

[Pau04] M. Paulin. The VirtualWorkcell: The Missing Link Between Robot Motion Planning and Visual Servoing. In *Proceedings Mechatronics & Robotics (MechRob 2004)*, Aachen, Germany, 2004.

[PC03] G. L. Peterson and D. J. Cook. Incorporating decision-theoretic planning in a robot architecture. *Robotics and Autonomous Systems*, 42:89–106, 2003.

[Pet02] L. Petersson. *A Framework for Integration of Processes in Autonomous Systems*. PhD thesis, Royal Institute of Technology, Numerical Analysis and Computer Science, Computational Vision and Active Perception Laboratory, Stockholm, Sweden, 2002.

[Pir99] P. Pirjanian. Behavior Coordination Mechanisms - State-of-the-art. Technical report, USC Robotics Research Laboratory, University of Southern California, Los Angeles, CA, USA, October 1999.

[PM99] P. Pirjanian and M. J. Mataric. A decision-theoretic approach to fuzzy behavior coordination. In *Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics & Automation (CIRA99)*, Monterey, CA, USA, November 1999.

[PML99] J. Peipmeier, G. McMurra, and H. Lipkin. A dynamic jacobian estimation method for uncalibrated visual servoing. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 944–949, Atlanta, USA, 1999.

[PPP02] E. Papadopoulos, I. Poulakakis, and I. Papadimitriou. On Path Planning and Obstacle Avoidance for Nonholonomic Platforms with Manipulators: A Polynomial Approach. *The International Journal of Robotics Research*, 21(4):367–383, 2002.

[PRG+03] A. Pape, O. Radchenko, A. Gräser, H. Jang, and Z. Bien. Obstacle Avoidance with Visual Servoing for Adjustable Zomm-Cameras. In *Proceedings of the 8th International Conference on Rehabilitation Robotics (ICORR 2003)*, Taejon, Korea, April 2003.

[PRK90] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1370–1382, December 1990.

[PW92] J. S. Penberthy and D. S. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning (KR-92)*, pages 103–114, Cambridge, MA, USA, 1992.

[RABP94] A. Ram, R. C. Arkin, G. Boone, and M. Pearce. Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation. *Journal of Adaptive Behaviors*, 2(3):277–305, 1994.

[RH96] E. Ralli and G. Hirzinger. A global and resolution complete path planner for up to 6 DOF robot manipulators. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 3295–3302, Minneapolis, Minnesota, USA, 1996.

[RH99] A. Ruf and R. Horaud. Rigid and articulated motion seen with an uncalibrated stereo rig. In *IEEE International Conference on Computer Vision*, pages 789–796, Kerkyra, Greece, 1999.

[RMS89] H. Ritter, T. Martinez, and K. Schulten. Topology-conserving maps for learning visuomotor-coordination. *Neural Networks*, 2(3):159–168, 1989.

[RMS92] H. Ritter, T. Martinez, and K. Schulten. *Neural Computation and Self-Organizing Maps: An Introduction*. MA: Addison-Wesley, 1992.

[Ros85] S. Rosenschein. Formal theories of knowledge in AI and robotics. *New Generation Computing*, pages 345–357, 1985.

[Ros95] J. K. Rosenblatt. DAMN: A Distributed Architecture for Mobile Navigation. *Proceedings of the AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford, CA. AAAI Press, Menlo Park, CA*, March 1995.

[RSB97] G. Richter, F. Śmieja, and U. Beyer. Integrative Architecture of the Autonomous Hand-Eye Robot JANUS. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics & Automation (CIRA97)*, Monterey, CA, USA, July 10-11 1997.

[RTHN97] A. Ruf, M. Tonko, R. Horaud, and H.-H. Nagel. Visual tracking of an end-effector by adaptive kinematic prediction. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 893–898, Grenoble, France, 1997.

[RV03] I. Refanidis and I. Vlahavas. Multiobjective heuristic state-space planning. *Artificial Intelligence*, 145:1–32, 2003.

[SA01] A. Stoytchev and R. C. Arkin. Combining Deliberation, Reactivity, and Motivation in the Context of a Behavior-Based Robot Architecture. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation (IEEE CIRA 2001)*, Baniff, Canada, July 29 - August 1 2001.

[Saf97] A. Saffiotti. The Uses of Fuzzy Logic in Autonomous Robot Navigation. *Soft Computing*, 1(4):180–197, 1997.

[San01] T. Sanchez. Entwicklung und Implementierung virtueller Sensoren für den Einsatz von Roboter-Avataren in simulierten Einsatzumgebungen. Master's thesis, Lehrstuhl für Technische Informatik, RWTH-Aachen, September 2001.

[Sar95] G. N. Saridis. Architectures for intelligent control. In M. M. Gupta and N. K. Sinhm, editors, *Intelligent Control Systems: Theory and Applications*. IEEE Press, 1995.

[SB97a] S. M. Smith and J. M. Brady. SUSAN - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997.

[SB97b] A. Steinhage and T. Bergener. Complex behavior by means of dynamical systems for an anthropomorphic robot. Technical report, Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany, 1997.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, Cambrige, MA, 1998.

[SB00] A. Steinhage and T. Bergener. Learning by Doing: a Dynamic Architecture for Generating Adaptive Behavioral Sequences. Technical report, Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany, 2000.

[SBP97] C. E. Smith, S. A. Brandt, and N. P. Papanikolopoulos. Eye-In-Hand Robotic Tasks In Uncalibrated Environments. *IEEE Transactions on Robotics and Automation*, 13(6):903–914, 1997.

[SC00] J. Stavnitzky and D. Capson. Multiple Camera Model-Based 3-D Visual Servo. *IEEE Transactions on Robotics and Automation*, 16(6):732–739, 2000.

[Sch87] M. J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. In John McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy, 1987. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Sch92] W. Schwinn. *Grundlagen der Roboterkinematik*. Verlag Liborius Schwinn, 1992.

[Sch01] M. Schmitt. *Konstruktion und Einsatz virtueller Umgebungen zur Navigationsunterstützung mobiler Roboter*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2001.

[Sch02] F. Schulte. Implementierung der Sequencingebene eines mobilen Manipulators. Master's thesis, Lehrstuhl für Technische Informatik, RWTH-Aachen, Germany, 2002.

[SD98] L. Sun and C. Doeschner. Visuo-motor coordination of a robot manipulator based on neural networks. In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, pages 1737–1742, Leuven, Belgium, May 1998.

[SDLC93] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell. Bayesian analysis in expert systems. *Statistical Sciece*, 8(3):219–283, 1993.

[Sen01] Sense8. Worldtoolkit. Internet home page, 2001. http://www.sense8.com/products/wtk.html.

[SHB99] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. PWS Publishing at Brooks/Cole Publishing Company, 2 edition, 1999.

[SIJ+00] C. Schlegel, J. Illmann, H. Jaberg, M. Shuster, and R. Wörz. Integrating Vision-Based Behaviors with an Autonomous Robot. *Videre: Journal of Computer Vision Research*, 1(4):32–60, 2000.

[SKC96] B. Sleman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.

[SKR95] A. Saffiotti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.

[SL98] C. Szepesvàri and A. Lörincz. An Integrated Architecture for Motion-Control and Path-Planning. *Journal of Robotic Systems*, 15(1):1–15, 1998.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, USA, 1992.

[SMCM91] P. Saint-Marc, J. S. Chen, and G. Medioni. Adaptive smoothing: A general tool for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):514–529, 1991.

[SP96] M. Sullivan and N. Papanikolopoulos. Using active deformable models to track deformable objects in robotic visual servoing experiments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2929–2934, Minneapolis, USA, 1996.

[Spr02] N. Springob. Planungsverfahren für mobile Manipulationsaufgaben. Master's thesis, Lehrstuhl für Technische Informatik, RWTH-Aachen, Germany, 2002.

[SS98] R. Sharma and N. Srinivasa. A framework for active vision-based robot control using neural networks. *Robotica*, 16:309–327, 1998.

[SS00a] L. Sciavicco and B. Sicialiano. *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing. Springer-Verlag, 2nd edition, 2000.

[SS00b] R. Sun and C. Sessions. Learning Plans without a priori Knowledge. *Adaptive Behavior*, 8(3/4):225–254, 2000.

[SS00c] R. Sun and C. Sessions. Self-Segmentation of Sequences: Automatic Formation of Hierarchies of Sequential Behaviors. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 30(3):403–418, Juni 2000.

[SSML90] S. Szabo, H. A. Scott, K. N. Murphy, and S. A. Legowik. Control System Architecture for a Remotly Operated Unmaned Land Vehicle. In *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, pages 876–883, Philadelphia, PA, USA, September 1990.

[SSV98] H. Sutanto, R. Sharma, and V. Varma. The role of exploratory movement in visual servoing without calibration. *Robotics and Autonomous Systems*, 23:153–169, 1998.

[STB99] F. Steele, G. Thomas, and T. Blackmon. An Operator Interface for a Robot-Mounted, 3D Camera System: Project Pioneer. In *Proceedings of the 1999 IEEE Virtual Reality Conference*, pages 126–132, Houston, TX, USA, March 1999.

[Ste94] M. R. Stein. *Behavior-Based Control for Time-Delayed Teleoperation*. PhD thesis, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA, USA, 1994.

[SvdLKG95] G. Schram, F. X. van der Linden, B. J. A. Kröse, and F. C. A. Groen. Predictive Robot Control with Neural Networks. In U. Rembold et al., editor, *Proceedings of the 1995 International Conference on Intelligent Autonomous Systems*, pages 117–122, Karlsruhe, Germany, 1995. IOS Press.

[SvS00] A. Steinhage and W. von Seelen. Dynamische Systeme zur Verhaltensgenerierung eines anthropomorphen Roboters. In *Autonome Mobile Systeme 2000: 16. Fachgespräch (AMS 2000)*, pages 260–269, Karlsruhe, Germany, 2000.

[SW95] B. Schiele and A. Waibel. Gaze tracking based on face-color. In *Proceedings of the International Worshop on Automatic Face- and Gesture-Recognition*, pages 344–349, Zurich, Switzerland, June 1995.

[Tan03] J. Tani. Learning to generate articulated behavior through the bottom-up and the bottom-down interaction processes. *Neural Networks*, 16:11–23, 2003.

[TK99] T. Tsuji and M. Kaneko. Noncontact Impedance Control for Redundant Manipulators. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 29(2):184–193, March 1999.

[TLJ97] E. Tunstel, T. Lippincott, and M. Jamshidi. Behavior Hierarchy for Autonomous Mobile Robots: Fuzzy-behavior modulation and evolution. *International Journal of Intelligent Automation and Soft Computing*, 3(1):37–50, 1997. Special Issue on Autonomous Control Engineering.

[TLK03] H. G. Tanner, S. G. Loizou, and K. J. Kyriakopoulos. Nonholonomic Navigation and Control of Cooperating Mobile Manipulators. *IEEE Transactions on Robotics and Automation*, 19(1):53–65, February 2003.

[Tor00] C. Torras. Neuroadaptive robots. In *Proceedings of the 15th International Conference on Pattern Recognition (ICPR-2000)*, pages 172 – 177, Barcelona, Spain, September 2000.

[Tsa87] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, August 1987.

[vA93] C. von Altrock. *Fuzzy Logic: Bd. 1. Technologie*, volume 1. Oldenbourg, 1993.

[vdS95] Patrick van der Smagt. *Visual robot arm guidance using neural networks*. PhD thesis, Universiteit van Amsterdam, 1995.

[VNE+01] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty Architecutre for Robotic Autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*, pages 121–132, Big Sky, Montana, USA, 2001.

[vWKN+02] G. von Wichert, C. Klimowicz, W. Neubauer, T. Wösch, G. Lawitzky, R. Caspari, H.-J. Heger, P. Witschel, U. Handemann, and M. Rinne. The Robotic Bar - An Integrated Demonstration of Man-Robot Interaction in a Service Scenario. In *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication, ROMAN 2002*, Berlin, Germany, September 2002.

[Wal96] J. Walter. *Rapid Learning in Robotics*. PhD thesis, Technische Fakultät, Universität Bielefeld, AG Neuroinformatik, 1996.

[War93] C. W. Warren. Fast path planning using modified A* method. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 662–667, San Diego, California, USA, 1993.

[WAS98] D. S. Weld, C. R. Anderson, and D. E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 897–904, Madison, Wisconsin, USA, 1998.

[Wat89] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[WB86] J. A. Worthey and M. H. Brill. Heuristic analysis of von Kries color constancy. *Journal of The Optical Society of America*, 3(10):1708–1712, 1986.

[Wel99] D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.

[WH97] P. Wunsch and G. Hirzinger. Real-time Visual Tracking of 3D Objects with Dynamic Handling of Occlusion. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 2868–2873, Albuqurque, New Mexico, USA, April 1997.

[WH99] G.-Q. Wei and G. Hirzinger. Multisensory Visual Servoing by a Neural Network. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 29(2):276–280, April 1999.

[WHW98] H. Wörn, D. Heinrich, and C. Wurll. Motion planning in dynamic environments - a parallel online approach. In *Proceedings of the 3rd International Symposium on Artificial Life and Robotics*, Oita, Japan, 1998.

[WN96] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the 15th Joint Conference on Artificial Intelligence*, pages 971–978, Nagoya, Japan, 1996.

[WN01] T. Wösch and W. Neubauer. Kollisionserkennung und -vermeidung für einen mobilen Manipulator. In *Autonome Mobile Systeme 2001: 17. Fachgespräch*, pages 104–112, Stuttgart, Germany, 2001. Springer.

[WNvWK02] T. Wösch, W. Neubauer, G. von Wichert, and Z. Kemeny. Robot Motion Control for Assistance Tasks. In *Proceedings of the International Workshop on Robot and Human Interactive Communication, ROMAN 2002*, pages 524–529, Berlin, Germany, 2002.

[WS93] J. A. Walter and K. J. Schulten. Implementation of Self-organizing Neural Networks for Visuo-motor Control of an Industrial Robot. *IEEE Transactions on Neural Networks*, 4(1):86–95, 1993.

[WS97] Q. M. J. Wu and K. Stanley. Modular Neural-Visual Servoing using a Neural-Fuzzy Decision Network. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 3238–3243, Albuquerque, New Mexico, USA, April 1997.

[WT98] G. Wells and C. Torras. Selection of image features for robot positioning using mutual information. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 2819–2826, Leuven, Belgium, May 1998.

[WVL81] D. C. C. Wang, A. H. Vagnucci, and C. C. Li. Gradient inverse weighted smoothing scheme and the evaluation of its performance. *Computer Graphics and Image Processing*, 15:167–181, 1981.

[WVT96] G. Wells, C. Venaille, and C. Torras. Vision-based robot positioning using neural networks. *Image and Vision Computing*, 14:715–732, 1996.

[WWH97] P. Wunsch, S. Winkler, and G. Hirzinger. Real-Time Pose Estimation of 3-D Objects from Camera Images Using Neural Networks. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 3232–3237, Albuqurque, New Mexico, USA, April 1997.

[YA94] B. Yoshimi and P. Allen. Visual control of grasping and manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 575–582, San Diego, CA, USA, 1994.

[Yam94] Y. Yamamoto. *Control and Coordination of Locomotion and Manipulation of a Wheeled Mobile Manipulator*. PhD thesis, University of Pennsylvania, 1994.

[YKD97] H. H. Yakali, B. J. A. Kröse, and L. Dorst. Vision-Based 6-DOF Robot End-effector Positioning Using Neural Networks. In *Proceedings 1997 Real World Computing Symposium (RWC'97)*, pages 191–198, Tokyo, Japan, 1997.

[YM01] S. X. Yang and M. Meng. Neural Network Approaches to Dynamic Collision-Free Trajectory Generation. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 31(3):302–317, Juni 2001.

[YYW03] Cang Ye, N. H. C. Yung, and Danwei Wang. A Fuzzy Controller With Supervised Learning Assisted Reinforcement Learning Algorithm for Obstacle Avoidance. *IEEE Transactions on Systems Man and Cybernetics - Part B: Cybernetics*, 33(1):17–27, February 2003.

[Zad73] L. A. Zadeh. Outline of a New Analysis of Complex Systems and Decision Processes. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(1):28–44, January 1973.

[ZGPP02] E. Zalama, J. Gómez, M. Paul, and J. R. Perán. Adaptive Behavior Navigation of a Mobile Robot. *IEEE Transaction on Systems, Man and Cybernetics-Part A: Systems and Humans*, 32(1):160–169, January 2002.

[ZK99] J. Zhang and A. Knoll. Integrating deliberative and reactive strategies via fuzzy modular control. In A. Saffioti and D. Driankov, editors, *Fuzzy logic techniques for autonomous vehilce navigation*, chapter 15, pages 367–387. Springer, 1999.

[ZKS00] J. Zhang, A. Knoll, and R. Schmidt. A neuro-fuzzy control model for fine-positioning of manipulators. *Journal of Robotics and Autonomous Systems*, 32:101–113, 2000.

[ZRDZ02] R. Zöllner, O. Rogalla, R. Dillmann, and M. Zöllner. Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, September 2002.

Anhang A

Symbolverzeichnis

Allgemeine Notation

Symbol	Bedeutung
a	Skalar
$\vec{a}$	Vektor
$\dot{\vec{a}}$	Erste Ableitung des Vektors $\vec{a}$
a_i	i-te Element des Vektors $\vec{a}$
$\mathbf{A}$	Matrix
a_{ij}	Element der i-ten Reihe und j-ten Spalte von Matrix $\mathbf{A}$
$\mathcal{A}$	Menge bzw. Raum
$f()$	Funktion
$p()$	Wahrscheinlichkeit
M	Raumpunkt
$\{A\}$, $\{B\}$	Koordinatensysteme
${}^{A}\vec{p}$	Position eines Raumpunktes $(x, y, z)^T$ relativ zu einem Koordinatensystem $\{A\}$
${}^{A}\vec{P}$	4×1 Vektor der homogenen Koordinaten eines Raumpunktes relativ zu $\{A\}$
${}^{A}\vec{o}$	Orientierungsvektor $(\theta, \phi, \omega)^T$ relativ zum Koordinatensystem $\{A\}$
${}^{A}_{B}\mathbf{R}$	Rotationsmatrix des Koordinatensystems $\{B\}$ zum Koordinatensystem $\{A\}$ (3×3). Der führende hochgestellte Buchstabe entspricht dem Bezugskoordinatensystem, hier $\{A\}$, und der führende tiefgestellte Buchstabe dem aktuelle Koordinatensystem, hier $\{B\}$
${}^{A}_{B}\mathbf{T}$	Homogene Transformationsmatrix des Systems $\{B\}$ zum System $\{A\}$ (4×4)
$\vec{\mathcal{P}}$	6×1 Vektor $(x, y, z, \theta, \phi, \omega)^T$, der Position und Orientierung zusammenfasst
$\mathcal{P}_i$	Pfad des i-ten Verhaltens
$\vec{\mathcal{P}}_{ji}$	j-te Stützstelle vom Pfad des i-ten Verhaltens
$\vec{u}$	Pixelposition $(u, v, 1)^T$ eines Punktes in einer Aufnahme
${}^{K_1}\vec{u}$	Pixelposition $(u, v, 1)^T$ eines Punktes in der Aufnahme aus Kameraposition $\{K_1\}$
$\vec{U}$	Vektor der homogene Pixelkoordinaten $(U, V, W)^T$ eines Punktes in einer Aufnahme. Es gilt: $(u, v, 1)^T = (\frac{U}{W}, \frac{V}{W}, 1)^T$
${}^{K_1}\vec{U}$	Vektor der homogene Pixelkoordinaten $(U, V, W)^T$ eines Punktes in der Aufnahme aus der Kameraposition $\{K_1\}$
$atan2(x, y)$	Die $\tan^{-1}(\frac{x}{y})$ Funktion, berücksichtigt jedoch die Vorzeichen von x, y, um den Winkel im Bereich $(-180, 180]$ zu berechnen

Verwendete Symbole

Symbol	Bedeutung
k_{xy}	Konstanten
d_{xy}	Distanz bzw. Dimension
$\vec{d}$	Dimensionsvektor
$\mathbf{I}$	Einheitsmatrix
$\vec{\rho}$	Aktion des Roboters
$\vec{s}$, $\vec{s}'$	Merkmalsvektoren
$\vec{q}$	Positionsvektor des Greifers
$\dot{\vec{q}}$	Geschwindigkeitsvektor des Greifers
θ, ϕ, ω	Winkel
$\vec{\theta}$	Vektor der Gelenkwinkel des Manipulators
a_i	DH-Parameter: Distanz der z_i-Achse des i-ten Gelenkes zur z_{i+1}-Achse des $i+1$-ten Gelenkes entlang der x_i-Achse
α_i	DH-Parameter: Winkel zwischen der z_i-Achse des i-ten Gelenkes und der z_{i+1}-Achse des $i+1$-ten Gelenkes um die x_i-Achse
θ_i	DH-Parameter: Winkel zwischen der x_i-Achse des i-ten Gelenkes und der x_{i-1}-Achse des $i-1$-ten Gelenkes um die z_i-Achse. Stimmt mit den i-ten Element des Gelenkwinkelvektors $\vec{\theta}$ überein
d_i	DH-Parameter: Distanz der x_i-Achse des i-ten Gelenkes zur x_{i-1}-Achse des $i-1$-ten Gelenkes entlang der z_i-Achse
$\mathbf{J}()$	Jacobi Matrix
$\mathbf{J}^+()$	Pseudoinverse der Jacobi Matrix
C_1, C_2	Brennpunkte der ersten bzw. zweiten Kamera
B_1, B_2	Bildhauptpunkte in den Aufnahmen der ersten bzw. zweiten Kamera
f	Brennweite
a_u	Skalierung in der u-Achse des Kamerabildes
a_v	Skalierung in der v-Achse des Kamerabildes
a_{schear}	Abweichung in Pixel der X-Achse des Kamerakoordinatensystems von der u-Achse des Kamerabildes
$\mathbf{K}$	Kalibrierungsmatrix
$\mathbf{F}$	Fundamentalmatrix
$\epsilon_{K_1 \vec{u}}$	Zum Pixel ${}^{K_1}\vec{u}$ in der ersten Aufnahme zugehörige epipolare Linie in der zweiten Aufnahme
(r, g, b)	RGB Intensitätswert eines Pixels
$I(\vec{u})$	Intensitätswert eines Pixels mit Koordinaten $\vec{u}$
$\mathbf{G}$	Gewichtungsmatrix der Anwendbarkeit der Verhalten
g_i	Gewichtung bzw. Anwendbarkeit des i-ten Verhaltens
$C()$	Koordinierungsfunktion
$\mathbf{H}$	Inhibitionsmatrix
$\beta_i(\vec{s}_i)$	Ausgabe vom i-ten Verhalten mit Merkmalsvektor $\vec{s}_i$ als Eingabe
$\mathcal{A}$	Aktionsraum
$\mathcal{Z}$	Menge von Zuständen bzw. Zustandsraum
$\mathcal{I}$	Menge der Eingabesignale
Z_j	Zustand der Umgebung zum Zeitschritt j
$\delta()$	Übergangsfunktion bei den FSAs und den FSMs

Symbol	**Bedeutung**
I_j	Interner Zustand des Roboters zum j-ten Zeitschritt
π	Handlungsstrategie
$R()$	Belohnungsfunktion
$V()$	Zustandswertfunktion
$Q()$	Aktionswertfunktion
r	Belohnung bzw. *Reinforcement Signal*
ϑ	Azimuth bei Polarkoordinaten
φ	Elevation bei Polarkoordinaten
$\vec{f_i}$	Kraftvektor auf das i-te Segment
$\vec{\tau}$	Gelenk-Moment Vektor der auf die Segmente des Manipulator wirkt
$E()$	Fehlerfunktion
$\mathcal{P}(Z_i)$	Menge der Vorgänger des i-ten Knotens in einem BBN
$\mu_A(x)$	Zugehörigkeitsfunktion für die linguistische Variable A
$w_{ki}^{(j)}$	k-te Gewichtung des i-ten Neurons der j-ten Schicht eines künstlichen neuronalen Netzes
$net_i^{(j)}$	Anregung des i-ten Neurons der j-ten Schicht eines künstlichen neuronalen Netzes
$o_i^{(j)}$	Ausgabe des i-ten Neurons der j-ten Schicht eines künstlichen neuronalen Netzes

Anhang B

Mobiler Service Roboter TAURO^3

In diesem Kapitel werden kurz die technischen Charakteristika des eingesetzten mobilen Manipulators und der verwendeten Kameras vorgestellt.

B.1 Die mobile Plattform

Die eingesetzte mobile Plattform (Abbildung B.1) basiert auf einer Rollstuhlplattform der Firma Invacare [Inv]. Die Plattform verfügt über zwei unabhängige Antriebe an den Vorderrädern, während die Hinterräder frei schwenkbar sind. Da die Plattform sich nicht in jeder Richtung direkt bewegen kann, handelt es sich um einen nicht-holonomen Roboter. Zur Navigation werden die Daten aus zwei Drehgeber an den Vorderrädern[1] und die Abstandsmessungen eines Laserscanners verwendet. Zusätzlich verfügt die mobile Plattform über vierzehn Ultraschallsensoren und ein Schwenk-Neige-Vorrichtung, auf dem die Bordkamera des mobilen Manipulators montiert ist. Da die Kameraposition von der Manipulatorbewegung nicht beeinflusst wird, handelt es sich um eine *eye-to-hand* Konfiguration.

Die Hardware umfasst a zwei Rechner zur Ansteuerung der Aktuatorik und zur Auswertung der Sensordaten. Ein Rechner übernimmt dabei die Regelung und Navigation der Plattform während auf einem zweiten PC die reaktive und die vermittelnde Ebene des Roboterarmes sowie die Bildverarbeitung der Kameras realisiert sind. Die deliberative Ebene des mobilen Manipulators wird auf einem externen Rechner ausgeführt; sie kommuniziert mit den vermittelnden Ebenen, die auf den Rechnern des mobilen Manipulator laufen, über eine drahtlose Ethernet Verbindung.

B.2 Der Manipulator

Der eingesetzte Manipulator ist der Manus Roboterarm der Firma Exact Dynamics[Dyn]. Es handelt sich um einen Manipulator mit sechs Gelenken und einem Greifer. Dadurch verfügt der Roboterarm über sieben Freiheitsgrade. Er wird über Gelenkgeschwindigkeiten angesteuert und liefert die aktuellen Werte der Gelenkwinkel zurück. Die Koordinatensysteme des Manipulators sind in Abbildung C.6 dargestellt, während die Denavit-Hartenberg Parameter in Tabelle C.1 zu finden sind. Der Roboterarm

[1] Diese Daten werden zur Bestimmung der Odometrie und somit zur Positionierung des Roboters eingesetzt.

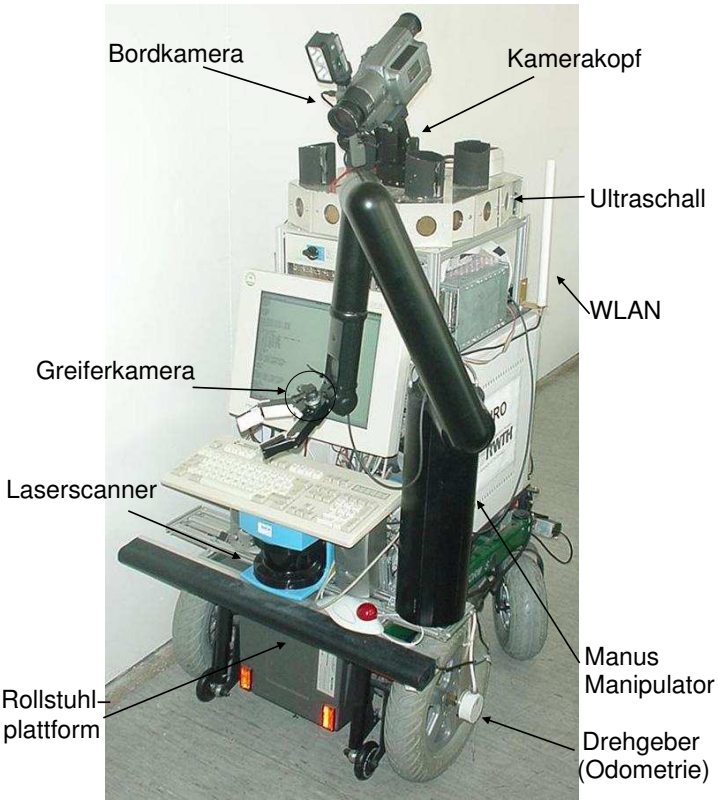


Abbildung B.1: Der mobile Manipulator TAURO[3].

verfügt zusätzlich über eine Kamera auf dem Greifer in einer so genannten *eye-in-hand* Konfiguration. Die homogene Transformationsmatrix des Koordinatensystems der Manipulator-Basis relativ zur mobilen Plattform ist (siehe auch Abbildung B.2):

$$ {}^{P}_{MB}\mathbf{T} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 300.02992 \\ 0.0 & 0.0 & -1.0 & -290.0 \\ -1.0 & 0.0 & 0.0 & -249.57572 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} $$

B.3 Kalibrierung der Roboterkameras

Die virtuellen Kameras wurden mit einem linearen Verfahren [Aya91] kalibriert. Die Matrix der intrinsischen Kalibrierungsparameter ist in beiden Fällen(virtuelle Greifer- und Bordkamera):

$$ \mathbf{K} = \begin{pmatrix} 379.3127 & 1.274 & 312.2 \\ 0 & 379.6930 & 239.9 \\ 0 & 0 & 1.0 \end{pmatrix} $$

Die Kalibrierung der realen Kameras fand nach dem Verfahren von Heikkilä [Hei00] mit einem planaren Schachbrettmuster statt. Die für die Greifer- und Bordkamera des mobilen Manipulators ermittelten Parameter sind in den Tabellen B.2 und B.1 angegeben.

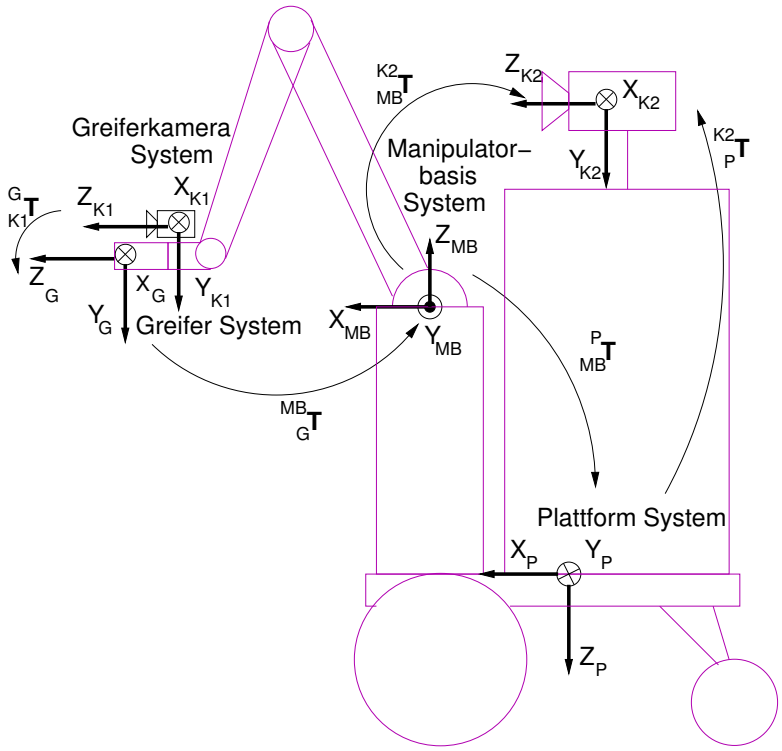


Abbildung B.2: Die Koordinatensystemen beim realen mobilen Manipulator TAURO[3].

Eigenschaft	**Wert**
Kameratyp	CS9000P
Konfiguration	*eye-in-hand*
Kamerasensor	Interline CCD
Typ	1/3
Aktive Pixel	752(H)x582(V)
Aktive Bildregion	4.89mm(H)x3.64mm(V)
Brennweite f (mm)	3.3
intrinsische Kalibrierungsmatrix $\mathbf{K}$	$\begin{pmatrix} 522.6212 & 0.7253 & 293.2 \\ 0 & 524.3822 & 241.8 \\ 0 & 0 & 1 \end{pmatrix}$
Homogene Transformationsmatrix relativ zum Greifer ${}^{G}_{K1}\mathbf{T}$	$\begin{pmatrix} -0.9996 & -0.0244 & 0.0127 & 5.0 \\ 0.0244 & -0.9997 & -0.0021 & 28.7 \\ 0.0128 & -0.0018 & 0.999 & -86.8 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Tabelle B.1: Kalibrierungsparameter der Greiferkamera des Manipulators.

Eigenschaft	**Wert**
Kameratyp	Sony DCR-VX700E
Konfiguration	*eye-to-hand*
Kamerasensor	CCD ICX059CK
Typ	1/3
Aktive Pixel	752(H)×582(V)
Pixelanzahl	795(H)×596(V)
Bildregion	6.00mm(H)×4.96mm(V)
Brennweite f (mm)	4.9
Verzerrungskoeffizient κ_1 $[1/mm^2]$	0.0078
intrinsische Kalibrierungsmatrix $\mathbf{K}$	$\begin{pmatrix} 665 & 0 & 388 \\ 0 & 602.5 & 296 \\ 0 & 0 & 1 \end{pmatrix}$
Homogene Transformationsmatrix relativ zur Manipulator Basis ${}^{MB}_{K2}\mathbf{T}$	$\begin{pmatrix} 0.0685 & -0.7963 & 0.6011 & -20 \\ -0.9969 & -0.0782 & -0.0101 & -280.9 \\ 0.0390 & -0.5999 & -0.7991 & 400.1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
Homogene Transformationsmatrix relativ zur mobilen Plattform ${}^{P}_{K2}\mathbf{T}$	$\begin{pmatrix} 0.0030 & -0.796 & 0.6053 & 0 \\ -0.999 & -0.0114 & -0.0101 & -5 \\ 0.0149 & -0.6053 & -0.7959 & 1050 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Tabelle B.2: Kalibrierungsparameter der Bordkamera von TAURO[3].

Anhang C

Theoretische Grundlagen

C.1 Theoretische Grundlagen der Manipulatorkinematik

Ein Manipulator besteht aus einer Folge von Segmenten (*Links*), die durch eine Kette von Gelenken (*Joints*) miteinander verbunden sind (siehe Abbildung C.1). Da diese Folge von Segmenten meistens mit einem Greifer bzw. Werkzeug endet und nicht zum ersten Segment zurückgeführt wird, wird sie als offene kinematische Kette bezeichnet. Die Gelenke des Manipulators sind verantwortlich für dessen Bewegung, da sie rotatorische oder translatorische Bewegungen ausführen können. Als Gelenkstellung bezeichnet man die Stellung, die die Segmente des Manipulators im Raum bei bestimmten Werten der Gelenke annehmen.

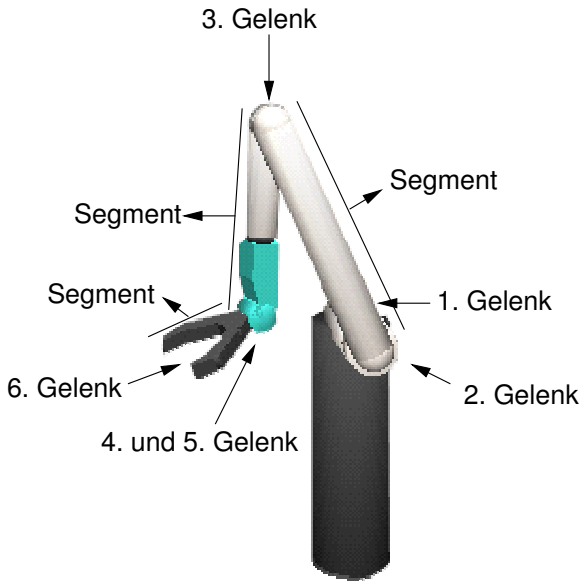


Abbildung C.1: Segmente bei einem Roboterarm mit sechs rotatorischen Gelenken.

Die meisten Manipulatoren, wie auch der hier eingesetzte Roboterarm, verfügen nur über rotatorische Gelenke[1]. Drehungen der Gelenke führen zu Bewegungen der Segmente. Um diese Bewegung

[1] Im weiteren Verlauf des Kapitels werden nur Manipulatoren mit rotatorischen Gelenken behandelt.

zu beschreiben, muss man die Beziehungen der einzelnen Segmenten und Gelenken zueinander und zu einem absoluten Koordinatensystem bestimmen können. Deswegen werden hier zuerst die Grundlagen zur Darstellung von Position und Orientierung bezüglich eines Koordinantensystems sowie die Transformation zwischen Koordinatensystemen vorgestellt. Eine ausführliche Herleitung dieser Formeln ist in [Cra89] zu finden.

C.1.1 Koordinatentransformationen

Beschreibung von Position und Orientierung eines Körpers

Ein Körper im Raum wird durch dessen kartesische Position und Orientierung eindeutig beschrieben. Abbildung C.2 verdeutlicht, dass für dieselbe Position im Raum mehrere Ausrichtungen des Greifers möglich sind. Die genaue Lage des Greifers ist erst dann eindeutig definiert, wenn dessen Orientierung bekannt ist. Um die Orientierung eines Körpers zu beschreiben, wird dem Körper ein Koordinatensystem zugeordnet. Die Lage, d.h. Position und Orientierung, dieses körperspezifischen Koordinatensystems zu einem Bezugskoordinatensystem entspricht dann der Position und Orientierung des Körpers zum Bezugskoordinatensystem.

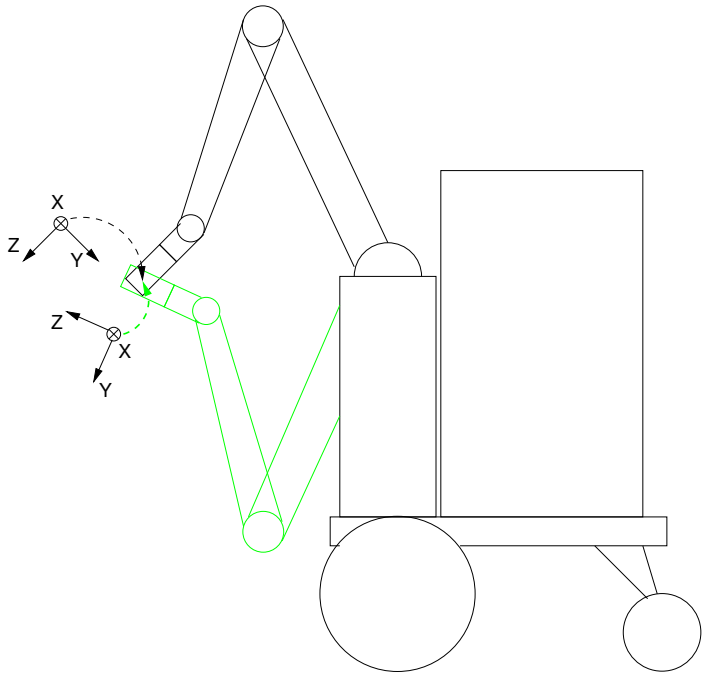


Abbildung C.2: Die gleiche Greiferposition mit unterschiedlicher Orientierung des Greifers. Das Koordinatensystem ist zwischen den Fingern des Greifers positioniert und definiert die Lage des Greifers eindeutig.

Eine Position im Raum wird bezüglich eines Bezugskoordinatensystems $\{A\}$ durch einen 3×1 Vektor ${}^{A}\vec{p}$ definiert[2]. Dieselbe Position bezüglich eines zweiten Koordinatensystems $\{B\}$ mit derselben

[2]In dieser Arbeit wird die Schreibweise nach Craig [Cra89] verwendet. Demnach entspricht ${}^{A}\vec{p} = (x, y, z)^T$ einem 3×1 Positionsvektor, der im Bezugskoordinatensystem $\{A\}$ definiert ist.

Orientierung, also ${}^{B}\vec{p}$, kann durch Vektoraddition beschrieben werden (Abbildung C.3):

$$ {}^{B}\vec{p} = {}^{A}\vec{p} + {}^{B}\vec{p}_{AOrig} \tag{C.1} $$

wobei ${}^{B}\vec{p}_{AOrig}$ der 3×1 Vektor ist, der den Ursprung des Koordinatensystems $\{A\}$ zu $\{B\}$ angibt.

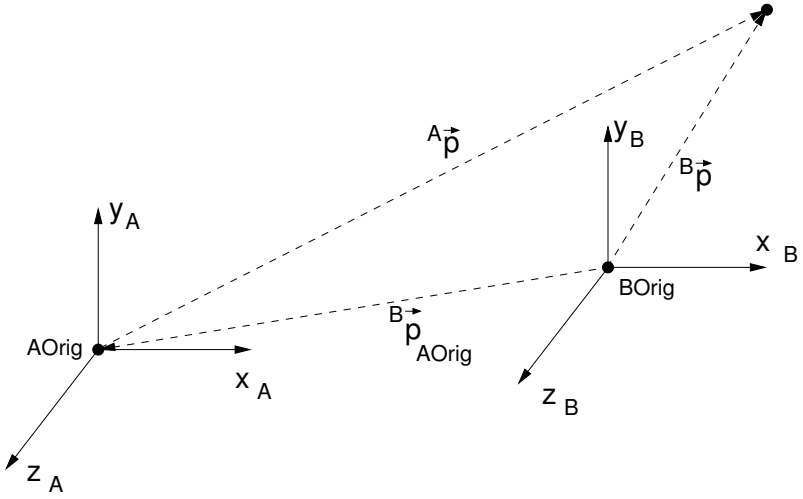


Abbildung C.3: Beschreibung einer Position relativ zu einem System $\{B\}$ und zu einem Bezugssystem $\{A\}$.

Die Orientierung eines Koordinatensystems $\{B\}$ zu einem Bezugskoordinatensystem $\{A\}$ kann durch die Einheitsvektoren der Achsen von $\{B\}$ bezüglich $\{A\}$ definiert werden. Da es sich um 3×1 Spaltenvektoren handelt, können sie in den Spalten einer 3×3 Matrix ${}^{A}_{B}\mathbf{R}$ zusammengefasst werden, die auch als Rotationsmatrix des Systems $\{B\}$ zum Bezugssystem $\{A\}$ bekannt ist [Cra89][3].

Sei ${}^{A}_{B}\mathbf{R}$ die Rotationsmatrix von System $\{B\}$ zu Bezugssystem $\{A\}$ und ${}^{B}_{A}\mathbf{R}$ die Rotationsmatrix von System $\{A\}$ zu Bezugssystem $\{B\}$. Da sich die Rotationsmatrix aus orthonormalen Vektoren zusammensetzt, hat sie folgende Eigenschaft:

$$ {}^{B}_{A}\mathbf{R}^{-1} = {}^{B}_{A}\mathbf{R}^{T} = {}^{A}_{B}\mathbf{R} \tag{C.2} $$

d.h. die Rotationsmatrix von $\{A\}$ bezüglich $\{B\}$ ist die Transponierte der Rotationsmatrix von $\{B\}$ bezüglich $\{A\}$.

Für den einfachen Fall, dass das Koordinatensystem $\{B\}$ durch die Rotation von $\{A\}$ um die x-Achse um einen Winkel θ entstanden ist, sind die Einheitsvektoren der Achsen von $\{B\}$ bezüglich $\{A\}$ wie folgt:

$$ {}^{A}\vec{x}_B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad {}^{A}\vec{y}_B = \begin{pmatrix} 0 \\ cos(\theta) \\ sin(\theta) \end{pmatrix}, \qquad {}^{A}\vec{z}_B = \begin{pmatrix} 0 \\ -sin(\theta) \\ cos(\theta) \end{pmatrix} \tag{C.3} $$

und demnach wird die Rotationsmatrix:

$$ {}^{A}_{B}\mathbf{R}_x(\theta) = \begin{bmatrix} {}^{A}\vec{x}_B & {}^{A}\vec{y}_B & {}^{A}\vec{z}_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{C.4} $$

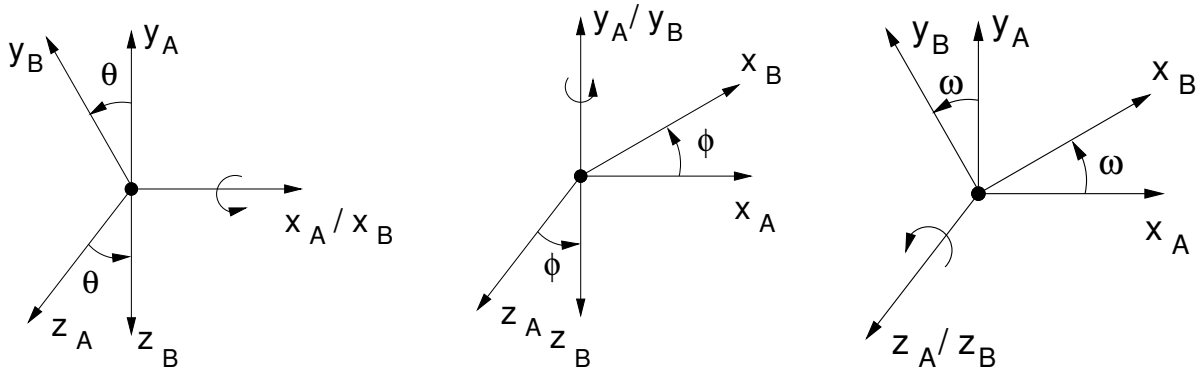
[3]Nach der Schreibweise von Craig [Cra89] entspricht ${}^{A}_{B}\mathbf{R}$ einer Matrix, die die Orientierung von Koordinatensystem $\{B\}$ relativ zum Bezugssystem $\{A\}$ beschreibt. Der führende hochgestellte Buchstabe entspricht dem Bezugskoordinatensystem, hier $\{A\}$, und der führende tiefgestellte Buchstabe dem aktuelle Koordinatensystem, hier $\{B\}$.

Entsprechend resultiert für eine Rotation ϕ nur um die y-Achse:

$$ {}^{A}_{B}\mathbf{R}_{y}(\phi) = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix} \tag{C.5} $$

und für eine Rotation ω nur um die z-Achse:

$$ {}^{A}_{B}\mathbf{R}_{z}(\omega) = \begin{bmatrix} cos(\omega) & -sin(\omega) & 0 \\ sin(\omega) & cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{C.6} $$



(a) (b) (c)

Abbildung C.4: Einzelne Drehungen um die x-Achse (a), um die y-Achse (b) und um die z-Achse (c).

Sei jetzt ${}^{B}\vec{p}$ die Position eines Punktes im Raum relativ zu einem Koordinatensystem $\{B\}$ und sei $\{A\}$ ein Bezugskoordinatensystem mit demselben Ursprung wie $\{B\}$, jedoch rotiert zu $\{B\}$ um ${}^{A}_{B}\mathbf{R}$. Dann ist die Position des Punktes relativ zu $\{A\}$ durch folgende Gleichung angegeben:

$$ {}^{A}\vec{p} = {}^{A}_{B}\mathbf{R}\,{}^{B}\vec{p} \tag{C.7} $$

Sei zusätzlich ein Koordinatensystem $\{C\}$ mit Rotationmatrix ${}^{B}_{C}\mathbf{R}$ bezüglich $\{B\}$, und sei der Punkt bezüglich dem Koordinatensystem $\{C\}$ angegeben, also ${}^{C}\vec{p}$, dann gilt nach Gleichung C.7:

$$ {}^{A}\vec{p} = {}^{A}_{B}\mathbf{R}\,{}^{B}\vec{p} = {}^{A}_{B}\mathbf{R}\,({}^{B}_{C}\mathbf{R}\,{}^{C}\vec{p}) = ({}^{A}_{B}\mathbf{R}\,{}^{B}_{C}\mathbf{R})\,{}^{C}\vec{p} \tag{C.8} $$

Demnach ist für die Rotationsmatrix von $\{C\}$ bezüglich $\{A\}$:

$$ {}^{A}_{C}\mathbf{R} = {}^{A}_{B}\mathbf{R}\,{}^{B}_{C}\mathbf{R} \tag{C.9} $$

und daher lassen sich Rotationstransformationen über Multiplikation der Rotationsmatrizen kombinieren.

Eine alternative Beschreibung der Drehung von $\{B\}$ relativ zu einem Bezugssystem $\{A\}$ kann man mit einer Folge von drei Rotationen um die Achsen von $\{A\}$ herleiten. Am Anfang hat $\{B\}$ dieselbe

Orientierung wie das Bezugssystem $\{A\}$. Dann kann $\{B\}$ jede beliebige Orientierung im Raum annehmen indem es zuerst um die x-Achse von $\{A\}$ um einen entsprechenden Winkel ω rotiert wird, dann um Winkel ϕ um die y-Achse von $\{A\}$ und anschließend um θ um die z-Achse von $\{A\}$. Aus Gleichungen C.4, C.5, C.6 und C.9 folgt dann für die Rotationsmatrix von $\{B\}$ relativ zu $\{A\}$:

$$ {}^A_B\mathbf{R}(\omega,\phi,\theta) = {}^A_B\mathbf{R}_z(\theta)\,{}^A_B\mathbf{R}_y(\phi)\,{}^A_B\mathbf{R}_x(\omega) = \begin{bmatrix} c\theta\,c\phi & c\theta\,s\phi\,s\omega - s\theta\,c\omega & c\theta\,s\phi\,c\omega + s\theta\,s\omega \\ s\theta\,c\phi & s\theta\,s\phi\,s\omega + c\theta\,c\omega & s\theta\,s\phi\,c\omega - c\theta\,s\omega \\ -s\phi & c\phi\,s\omega & c\phi\,c\omega \end{bmatrix} \quad \text{(C.10)} $$

wobei einfachheitshalber $c\theta$ anstatt $\cos(\theta)$ und $s\theta$ anstatt $\sin(\theta)$ verwendet wird. Diese Darstellung heißt auch RPY-Winkel-Darstellung aus den entsprechenden englischen Bezeichnungen für die einzelnen Drehungen mit *roll*, *pitch* und *yaw*. Aus Gleichung C.10 ist die Ermittlung der entsprechenden Drehungen für eine beliebige Rotationsmatrix einfach. Sei r_{ij} das Element der i-ten Reihe und j-ten Spalte der Rotationsmatrix. Dann ist:

$$ \begin{aligned} \phi &= atan2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}) \\ \omega &= atan2(\frac{r_{21}}{c\phi}, \frac{r_{11}}{c\phi}) \\ \theta &= atan2(\frac{r_{32}}{c\phi}, \frac{r_{33}}{c\phi}) \end{aligned} \quad \text{(C.11)} $$

Die Funktion $atan2(\alpha,\beta)$ berechnet den Winkel $\gamma = \tan^{-1}(\frac{\beta}{\alpha})$, sie benutzt jedoch auch das Vorzeichen von α und β, um γ im Bereich $(-180^0, 180^0]$ zu bestimmen.

Außer dieser Beschreibungsform gibt es auch weitere Möglichkeiten, Rotationen zu beschreiben, wie die ZYX-Euler Winkel, die ZYZ-Euler Winkel oder die Euler Parameter, die auch als Quartenionen bekannt sind.

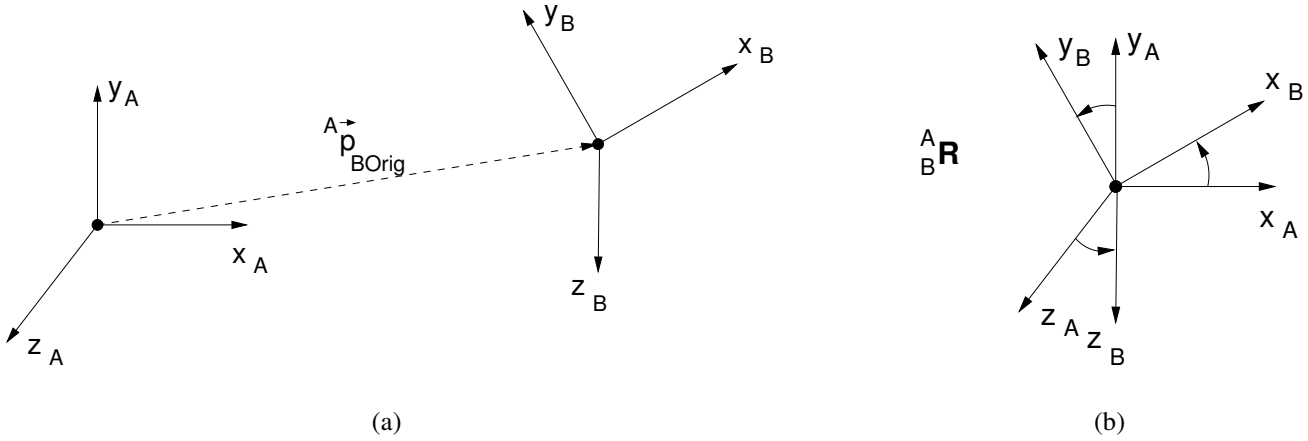
Homogene Transformationsmatrix

Aus Gleichungen C.1 und C.7 kann man die allgemeine Gleichung für die Transformation der Beschreibung eines Punktes ${}^B\vec{p}$ von einem Koordiantensystem $\{B\}$ zu einem Bezugssystem $\{A\}$ herleiten. $\{A\}$ und $\{B\}$ sind zueinander sowohl verschoben als auch rotiert:

$$ {}^A\vec{p} = {}^A_B\mathbf{R}\,{}^B\vec{p} + {}^A\vec{p}_{BOrig} \quad \text{(C.12)} $$

Aus Gleichung C.12 folgt, dass die Lage eines kartesischen Koordinatensystems relativ zu einem Referenzsystem durch eine Translation des Nullpunktes des Referenzsystems zum Ursprung des neuen Koordinatensystems und eine nachfolgende Rotation um die drei Hauptachsen zu beschreiben ist. Gleichung C.12 kann weiterhin vereinfacht werden. Dabei werden die 3×1 Vektoren ${}^A\vec{p}$ und ${}^B\vec{p}$ um ein Element mit dem Wert 1 erweitert; die so entstehenden 4×1 Vektoren ${}^A\vec{P}$ und ${}^B\vec{P}$ sind als homogene Positionsvektoren bekannt. Die 3×3 Rotationsmatrix ${}^A_B\mathbf{R}$ und der 3×1 Vektor ${}^A\vec{p}_{BOrig}$ werden in einer 4×4 Matrix ${}^A_B\mathbf{T}$ zusammengefasst, wobei die unterste Zeile der Matrix mit $(0\,0\,0\,1)$ belegt wird:

$$ {}^A\vec{P} = \begin{bmatrix} {}^A\vec{p} \\ \hdashline 1 \end{bmatrix} = \left[\begin{array}{ccc:c} & {}^A_B\mathbf{R} & & {}^A\vec{p}_{BOrig} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^B\vec{p} \\ \hdashline 1 \end{bmatrix} = {}^A_B\mathbf{T}\,{}^B\vec{P} \quad \text{(C.13)} $$



(a) (b)

Abbildung C.5: Lage eines kartesischen Koordinatensystems $\{B\}$ relativ zu einem Referenzsystem $\{A\}$. Translation (a) und nachfolgende Rotation (b).

Die 4×4 Matrix ${}^A_B\mathbf{T}$ ist als die homogene Transformationsmatrix des Koordinatensystems $\{B\}$ bezüglich $\{A\}$ bekannt. Ähnlich zu Gleichung C.2 gilt hier auch:

$$ {}^B_A\mathbf{T} = {}^A_B\mathbf{T}^{-1} = \left[\begin{array}{c|c} {}^A_B\mathbf{R}^T & -{}^A_B\mathbf{R}^T\,{}^A\vec{p}_{BOrig} \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right] \tag{C.14} $$

Die wichtigste Eigenschaft der homogenen Transformationsmatrix ist, dass sie Transformationen zwischen Koordinatensysteme durch Matrizenmultiplikationen darstellen lässt. Sei ${}^A_B\mathbf{T}$ die homogene Transformationsmatrix von $\{B\}$ zu $\{A\}$ und ${}^B_C\mathbf{T}$ die Transformationsmatrix von $\{C\}$ zu $\{B\}$. Dann gilt für $\{C\}$ bezüglich $\{A\}$:

$$ {}^A_C\mathbf{T} = {}^A_B\mathbf{T}\,{}^B_C\mathbf{T} = \left[\begin{array}{c|c} {}^A_B\mathbf{R}\,{}^B_C\mathbf{R} & {}^A_B\mathbf{R}\,{}^B\vec{p}_{COrig} + {}^A\vec{p}_{BOrig} \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right] \tag{C.15} $$

C.1.2 Kinematische Beschreibung von Manipulatoren

Unter Freiheitsgrad eines Körpers versteht man die Anzahl der möglichen unabhängigen Bewegungen, d.h. Verschiebungen oder Drehungen, eines starren Körpers gegenüber einem Bezugskoordinatensystem. Demnach hat ein im Raum frei beweglicher, starrer Körper maximal sechs Freiheitsgrade, die sich aus drei translatorischen und drei rotatorischen Bewegungsmöglichkeiten zusammensetzen. Aus mechanischen Konstruktionsgründen haben in den meisten Fällen die Gelenke von Manipulatoren einen einzelnen Freiheitsgrad. Falls ein Gelenk mit n Freiheitsgrade vorhanden ist, kann es mit n Gelenken von einem Freiheitsgrad modelliert werden.

Bei Manipulatoren mit nur rotatorischen Gelenken sind über Drehgeber die Gelenkdrehungen zu jedem Zeitpunkt bekannt. Aus diesen Drehungen muss die entsprechende kartesische Lage der Segmente des Manipulators und insbesondere die Lage des Greifers berechnet werden. Diese Berechnung wird als Vorwärtskinematik (*Forward Kinematics*) oder auch einfach Kinematik (*Kinematics*) des Roboterarmes bezeichnet.

Für die Beschreibung der Kinematik eines Manipulators wird jedem Gelenk ein Koordinatensystem zugeordnet und seine Lageänderung bezüglich des vorherigen Gelenkes beschrieben. Als Manipulator-Basis wird ein Segment 0 vereinbart, das selbst keine Bewegungsmöglichkeit besitzt und dessen Position von den Bewegungen der Gelenke nicht beeinflußt wird. Segment 0 dient zur Plazierung des Roboterarmes in dem umgebendem System. Dadurch kann die Position des letzten, n-ten Gelenkes (Greifer) über die nacheinanderfolgende Berechnung der Position der i vorherigen Gelenke,bezüglich der Manipulator Basis berechnet werden. Durch die Beschreibung der Robotergelenke im Koordinatensystem des direkten Vorgängergelenkes erhält man eine Darstellung für die Transformationmatrix ${}^0_nT$ des letzten Gelenkes zur Manipulator Basis. ${}^0_nT$ ist nur noch von den Gelenkwinkeln abhängig und ist durch eine einfache Matrixmultiplikation zu berechnen:

$$ {}^0_n\mathbf{T} = {}^0_1\mathbf{T}\,{}^1_2\mathbf{T}\ldots{}^{n-1}_n\mathbf{T} \tag{C.16} $$

wobei ${}^{i-1}_i\mathbf{T}$ die homogene Transformationsmatrix des Koordinatensystems des Gelenkes i zu Gelenk $i-1$ ist. Entsprechend ist die Lage des i-ten Gelenkes zur Manipulator-Basis, die mit 0 gekennzeichnet ist:

$$ {}^0_i\mathbf{T} = {}^0_1\mathbf{T}\,{}^1_2\mathbf{T}\ldots{}^{i-1}_i\mathbf{T} \tag{C.17} $$

Um die geometrische Beziehungen zwischen den Koordinatensystemen des Roboterarmes zu beschreiben, hat sich im Laufe der Zeit ein Verfahren durchgesetzt, das ursprünglich von Denavit und Hartenberg [DH55] zur Beschreibung von starren mechanischen Körpern eingeführt wurde. Abhängig davon, ob man die Koordinatensysteme relativ zum vorherigen Gelenk definiert bzw. auch das nächste Gelenk bei der Definition verwendet, unterscheidet man zwischen dem DH- und dem modifizierten DH-Verfahren.

Dabei werden die Koordinatensysteme nach einer bestimmten Vorschrift pro Gelenk definiert. Die z_i-Achse des Koordinantensystems des i-ten Gelenkes stimmt mit der Rotationsachse des Gelenkes i (DH) bzw. $i+1$ (modifiziertes DH) überein. Die x_i-Achse ist dann senkrecht zur Ebene von z_i und z_{i+1} (DH) bzw. z_i und z_{i-1} (modifiziertes DH) mit Richtung von i zu $i+1$ definiert. Die y_i-Achse wird nach einem rechtshändigen Koordinatensystem definiert. Abbildung C.6 zeigt die Koordinatensysteme der Gelenke des bei in dieser Arbeit eingesetzten Manipulators, während Abbildung C.7 die Koordinatensysteme zweier nacheinanderfolgender Gelenke im allgemeinen Fall präsentiert.

Die Parameter des Verfahrens drücken die geometrischen Beziehungen zwischen den Koordinatensystemen bzw. ihre relative Lage aus. Pro Gelenk und Koordinatensystem sind vier Parameter definiert (Abbildung C.7):

a_i ist die Distanz von z_i zu z_{i+1} entlang der Achse x_i.

α_i ist der Winkel zwischen z_i und z_{i+1} um x_i.

θ_i ist der Winkel zwischen x_i und x_{i-1} um z_i[4].

d_i ist die Distanz von x_i zu x_{i-1} entlang z_i.

Dann ergibt sich die Lage des Koordinatensystems des Gelenkes i zu Gelenk $i-1$ nach:

$$ {}^{i-1}_i\mathbf{T} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\,\cos(\alpha_{i-1}) & \cos(\theta_i)\,\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})\,d_i \\ \sin(\theta_i)\,\sin(\alpha_{i-1}) & \cos(\theta_i)\,\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})\,d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.18} $$

[4]Bei Manipulatoren mit ausschließlich rotatorischen Gelenken entspricht θ_i dem Winkelwert von Gelenk i.

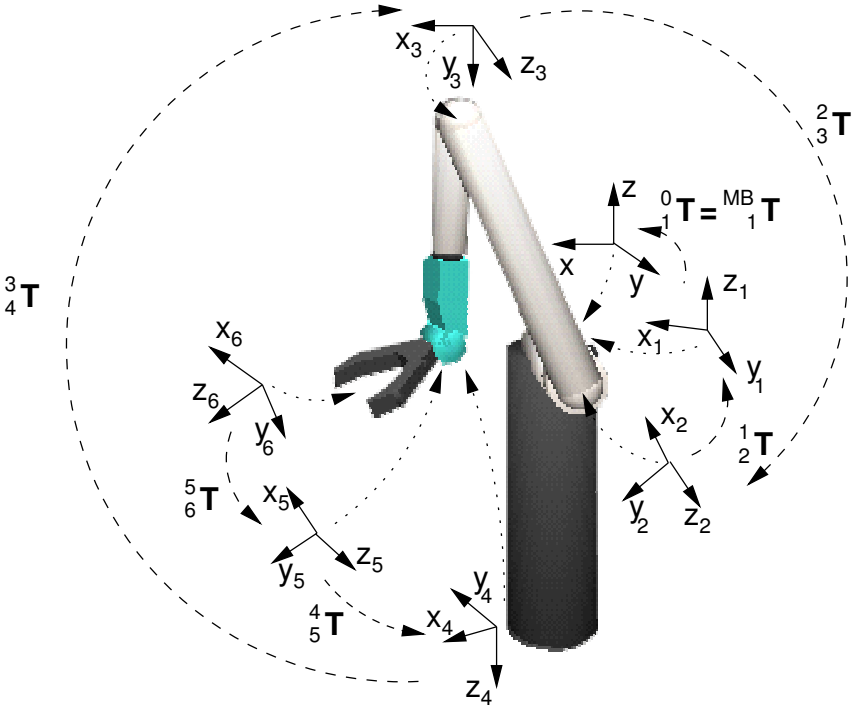


Abbildung C.6: Koordinatensysteme beim eingesetzten Manus Manipulator.

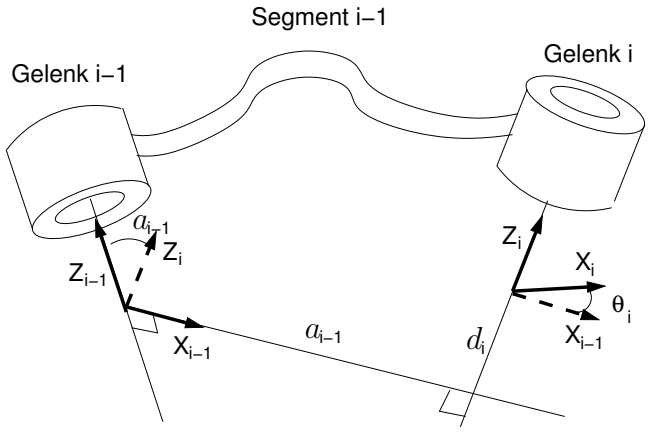


Abbildung C.7: Koordinatensysteme und DH-Parameter.

Der in dieser Arbeit eingesetzte Manipulator ist der Manus Roboterarm der Firma Exact Dynamics [Dyn]. Es handelt sich um einen Manipulator mit sechs rotatorischen Gelenken und einem Greifer, also insgesamt sieben Freiheitsgrade. Die kinematische Beschreibung entspricht der eines PUMA 560. Die Denavit-Hartenberg (DH) Parameter von Manus sind in Tabelle C.1 beschrieben:

Gelenk	a	α	d	θ **in Grad**
1	0.0	-90.0^0	0.0	θ_1
2	a_2	0.0^0	d_2	θ_2
3	0.0	90.0^0	0.0	θ_3
4	0.0	-90.0^0	d_4	θ_4
5	0.0	90.0^0	0.0	θ_5
6	0.0	0.0^0	d_6	θ_6

Tabelle C.1: DH-Parameter des Manus Manipulators.

mit

$$a_2 = 400mm$$
$$d_2 = 105mm$$
$$d_4 = 320mm$$
$$d_6 = 160mm$$

Inverse kinematische Transformation

Unter der inversen kinematischen Transformation, oft auch als Rückwärtsrechnung oder inverse Kinematik bezeichnet, versteht man die Berechnung der Winkelstellungen der Gelenke bei einer vorgegebenen Lage des Koordinatensystems des letzten Gelenkes bzw. des Greifers. Wird mit n die Anzahl der Gelenke des Roboters bezeichnet, so bedeutet dies, dass aus der Matrix ${}^0_n\mathbf{T}$ die Gelenkwinkel θ_1,..., θ_n berechnet werden. Dabei existieren im Allgemeinen für eine bestimmte kartesische Lage des Greifers mehr als eine mögliche Gelenkstellungen (Abbildung C.8). Bei einem Manipulator mit sechs Freiheitsgraden kann es, wie später gezeigt, bis zu acht mögliche Gelenkstellungen geben [Cra89]. Weiterhin ist zu beachten, dass aufgrund der physikalischen Eigenschaften und Abmessungen eines Manipulators nicht alle Positionen im Raum bzw. im Aktionsraum des Roboterarmes einzunehmen sind.

Zur Berechnung der inversen Kinematik muss ein nichtlineares Gleichungssystem mit Hilfe von geschlossenen oder numerischen Algorithmen gelöst werden. Die normalerweise weniger rechenaufwändigen geschlossenen Algorithmen lassen sich weiter in algebraische und geometrische Methoden unterteilen [Sch92], [SS00a], [Cra89]. Die Grenzen zwischen den beiden Verfahren sind jedoch unscharf, da jeder geometrische Ansatz algebraische Ausdrücke einsetzt, so dass sich beide Methoden ähnlich sind. Numerische Lösungen sind iterativ, rechenintensiv, können nicht immer alle mögliche Gelenkstellungen für eine kartesische Position berechnen und sind nicht robust in der Nähe von Singularitäten[5]. Numerische Lösungen haben jedoch den Vorteil, dass sie universell einsetzbar sind. Geschlossene Algorithmen dagegen sind sehr robust, können aber nur in speziellen Fällen eingesetzt werden. Deswegen ist es wichtig, einen Manipulator so zu entwerfen, dass eine geschlossene Lösung existiert.

In dieser Arbeit ist auf eine algebraische Lösung der inversen Kinematik zurückgegriffen worden. Hier wird kurz der Lösungweg präsentiert, für die Herleitung der Formeln wird auf [Cra89] verwiesen.

[5]Singularitäten entstehen sobald zwei oder mehr Gelenke in Reihe gestellt sind und deshalb dieselbe Bewegung der nachfolgenden Segmente verursachen.

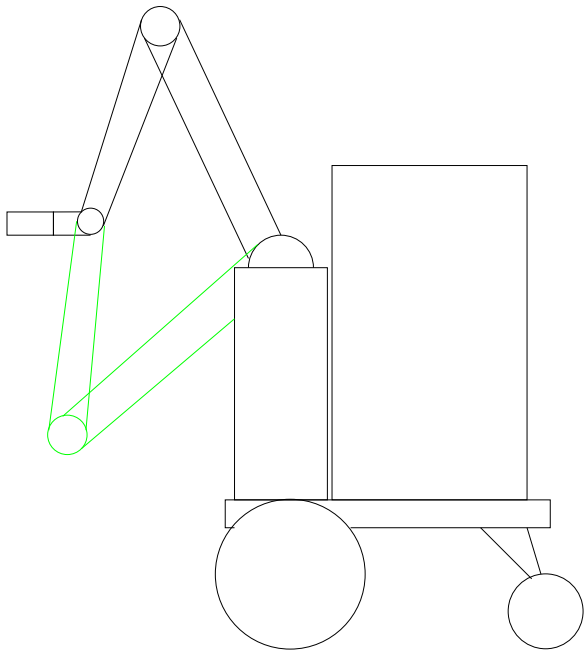


Abbildung C.8: Unterschiedliche Gelenkstellungen mit derselben kartesischen Lage des Greifers.

Sei die gewünschte kartesische Position und Orientierung des Greifers mit der homogenen Matrix ${}^0_6\mathbf{T}$ ausgedrückt:

$$ {}^0_6\mathbf{T} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.19} $$

Seien zusätzlich die Denavit-Hartenberg (DH) Parameter in der Tabelle C.1 angegeben und $\cos(\theta_i) = c_i$ und $\sin(\theta_i) = s_i$. Dann ist die algebraische Lösung für das Problem der inversen Kinematik:

$$ \theta_1 = atan2(p_y, p_x) - atan2(d_3, \pm\sqrt{p_x^2 + p_y^2 - d_3^2}) \tag{C.20} $$

$$ \theta_2 = \theta_{23} - \theta_3 \tag{C.21} $$

$$ \theta_3 = atan2(a_3, d_4) - atan2(K, \pm\sqrt{a_3^2 + d_4^2 - K^2}) \tag{C.22} $$

mit

$$ K = \frac{p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - d_3^2 - d_4^2}{2a_2} $$

$$ \theta_{23} = atan2((-a_3 - a_2c_3)p_z - (c_1p_x + s_1p_y)(d_4 - a_2s_3), (a_2s_3 - d_4)p_z - (a_3 + a_2c_3)(c_1p_x + s_1p_y)) $$

Falls $s_5 \neq 0$ ist:

$$ \theta_4 = atan2(-r_{13}s_1 + r_{23}c_1, -r_{13}c_1c_{23} - r_{23}s_1c_{23} + r_{33}s_{23}) \tag{C.23} $$

Mit $s_5 = 0$ befindet sich der Manipulator in einer singulären Gelenkstellung, in der die Achsen 4 und 6 in einer Reihe aufgestellt sind und dieselbe Bewegung des letzten Segmentes des Roboterarmes verursachen. Diese Situation kann detektiert werden, falls beide Argumente von Gleichung C.23 gegen Null gehen. Dann kann θ_4 frei gewählt werden und θ_6 wird entsprechend bestimmt. Meistens behält θ_4 seinen gegenwärtigen Wert.

$$\theta_5 = atan2(s_5, c_5) \tag{C.24}$$

$$\theta_6 = atan2(s_6, c_6) \tag{C.25}$$

mit

$$\begin{aligned}
s_5 &= -r_{13}(c_1c_{23}c_4 + s_1s_4) - r_{23}(s_1c_{23}c_4 - c_1s_4) + r_{33}(s_{23}c_4) \\
c_5 &= r_{13}(-c_1s_{23}) + r_{23}(-s_1s_{23}) + r_{33}(-c_{23}) \\
s_6 &= -r_{11}(c_1c_{23}s_4 + s_1c_4) - r_{21}(s_1c_{23}s_4 - c_1c_4) + r_{31}(s_{23}s_4) \\
c_6 &= r_{11}((c_1c_{23}c_4 + s_1s_4)c_5 - c_1s_{23}s_5) + r_{21}((s_1c_{23}c_4 - c_1s_4)c_5 - s_1s_{23}s_5) - r_{31}(c_{23}c_4c_5 + c_{23}s_5)
\end{aligned}$$

Wegen der Plus-Minus Vorzeichen in Gleichungen C.20 und C.22 entstehen vier mögliche Lösungen. Zusätzlich existieren vier weitere Lösungen, da man das Handgelenk des Manipulators, das aus den Segmenten 4, 5 und 6 besteht, um 180^0 drehen kann. Für jede der vier Lösungen aus Gleichungen C.20 bis C.25 entsteht eine weitere alternative Lösung durch:

$$\begin{aligned}
\theta'_4 &= \theta_4 + 180^0 \\
\theta'_5 &= -\theta_5 \\
\theta'_6 &= \theta_6 + 180^0
\end{aligned} \tag{C.26}$$

Daraus ergeben sich höchstens acht Lösungen, also acht mögliche Gelenkstellungen. Diese werden überprüft, ob sie eine Kollision der Segmente miteinander produzieren und ob sie eventuell die Grenzwerte für bestimmte Gelenke überschreiten. Aus den gültigen Gelenkstellungen $\vec{\theta}_j$ wird dann die günstigste zur gegenwärtigen Gelenkposition des Roboterarmes $\vec{\theta}'$ ausgewählt. Bei der Auswahl werden sowohl die Distanz der Lösungen zur aktuellen Gelenkstellung als auch die Gefahr vor Selbstkollisionen der Manipulatorsegmente berücksichtigt. Weiterhin dürfen die Lösungen die Gelenke nicht ausserhalb ihrer Drehungsgrenzen führen. Lösungen, die eine große Bewegung der ersten drei Gelenke verursachen, erhalten eine schlechtere Bewertung, da diese Gelenke eine größere Massenverschiebung des Manipulators produzieren und dadurch die Kollisionswahrscheinlichkeit mit Hindernissen erhöhen.

Diese Kriterien werden in einer Kostenfunktion zusammengefasst. Sei $\vec{\theta}_j = (\theta_{1j}, ..., \theta_{6j})^T$, $j = 1...8$, eine der acht möglichen Lösungen, $\vec{\theta}' = (\theta'_1, ..., \theta'_6)^T$ die aktuelle Gelenkstellung des Manipulators und θ_{Ui} die Obergrenze und θ_{Di} die Untergrenze des Drehungswinkels von Gelenk i. Dann ist die Kostenfunktion:

$$f_{GK}(\vec{\theta}_j, \vec{\theta}') = \sum_{i=1}^{6} g_{GKij} \mid \theta_{ij} - \theta'_i \mid \tag{C.27}$$

mit g_{GKij} Gewichtungen, die die obigen Kriterien zusammenfassen:

$$g_{GKij} = \begin{cases} k_{GK4} & \text{falls ein Übergang von der letzten Konfiguration zur nächsten Selbstkollisionen verursacht,} \\ k_{GK3} & \text{falls } \theta_{ij} < \theta_{Di} + 2^0 \text{ oder } \theta_{ij} > \theta_{Ui} - 2^0, \\ k_{GK2} & \text{falls } \mid \theta_{ij} - \theta'_i \mid \geq 60^0 \text{ und } i = 1, 2, 3, \\ k_{GK1} & \text{falls } \mid \theta_{ij} - \theta'_i \mid \geq 80^0 \text{ und } i = 3, 4, 5, \text{ und} \\ 1 & \text{sonst.} \end{cases} \tag{C.28}$$

mit $1 < k_{GK1} < k_{GK2} < k_{GK3} \ll k_{GK4}$. Die Auswahl der günstigsten Gelenkstellung entspricht dann der Minimierung der Kostenfunktion.

Geschwindigkeiten und Kräfte

Ein Regler, der den Manipulatorgreifer entlang einer Trajektorie im kartesischen Raum führt, berechnet das Fehlersignal als Positions- und Orientierungsänderung des Greifers. Betrachtet man einen kurzen zeitlichen Abschnitt, dann entspricht die Positions- und Orientierungsänderung der Geschwindigkeit des Greifers. Jedoch wird ein Manipulator meistens über Gelenkdrehungen angesteuert. Daher benötigt man die Kenntnis der Beziehung zwischen der Geschwindigkeit des Greifers und den entsprechenden Gelenkdrehungsgeschwindigkeiten. Hier wird nur kurz auf das Prinzip der Berechnung dieser Beziehung eingegangen.

Sei $\vec{\mathcal{P}}$ der Vektor, der die Position und Orientierung des Greifers beinhaltet und $\vec{\theta}$ der Vektor der Gelenkwinkel des Roboterarmes. Sei zusätzlich die Beziehung von $\vec{\mathcal{P}}$ zu $\vec{\theta}$ mit n Funktionen f_i angegeben:

$$\begin{aligned} p_1 &= f_1(\theta_1, ..., \theta_m) \\ &\vdots \\ p_n &= f_n(\theta_1, ..., \theta_m) \end{aligned} \tag{C.29}$$

wobei n und m die Anzahl der Elemente von $\vec{\chi}$ und $\vec{\theta}$ sind. Die gesuchte Beziehung zwischen der Geschwindigkeit des Greifers $\dot{\vec{\chi}}$ und den entsprechenden Gelenkdrehungsgeschwindigkeiten $\dot{\vec{\theta}}$ ergibt sich aus C.29 durch Differenzierung über die Zeit:

$$\begin{aligned} \frac{\partial p_1}{\partial t} &= \frac{\partial f_1(\vec{\theta})}{\partial \theta_1}\frac{\partial \theta_1}{\partial t} + ... + \frac{\partial f_m(\vec{\theta})}{\partial \theta_m}\partial t \\ &\vdots \\ \frac{\partial p_n}{\partial t} &= \frac{\partial f_n(\vec{\theta})}{\partial \theta_1}\frac{\partial \theta_1}{\partial t} + ... + \frac{\partial f_m(\vec{\theta})}{\partial \theta_m}\partial t \end{aligned} \tag{C.30}$$

Die Gleichungen in C.30 können mit Hilfe der Jacobi Matrix, die die partiellen Ableitungen der Funktionen f_i beinhaltet, in Matrizenform zusammengefasst werden:

$$\dot{\vec{\mathcal{P}}} = \left[\tfrac{\partial p_1}{\partial t} \dots \tfrac{\partial p_n}{\partial t}\right] = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \cdots & \frac{\partial f_1}{\partial \theta_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial \theta_1} & \cdots & \frac{\partial f_n}{\partial \theta_m} \end{bmatrix} \left[\tfrac{\partial \theta_1}{\partial t} \dots \tfrac{\partial \theta_m}{\partial t}\right] = \mathbf{J}(\vec{\theta})\dot{\vec{\theta}} \tag{C.31}$$

Dadurch ist die gesuchte Roboterbewegung:

$$\dot{\vec{\theta}} = \mathbf{J}^{+}(\vec{\theta})\dot{\vec{\mathcal{P}}} \tag{C.32}$$

Dabei ist $\mathbf{J}$ die Roboter Jacobi Matrix und $\mathbf{J}^{+}$ ihre Pseudoinverse. Mit der Roboter Jacobi Matrix wird der Raum der kartesischen Greifergeschwindigkeiten auf den Raum der Gelenkdrehungsgeschwindigkeiten abgebildet. Die genaue Form und Herleitung der Roboter Jacobi Matrix für einen Manipulator mit sechs Gelenken ist in [SS00a] zu finden.

Wenn Kräfte und Momente auf einem Manipulator wirken, verursachen sie Momente auf die Gelenke und dadurch eine Bewegung der Segmente. Durch die Bewegung verrichten die Kräfte Arbeit. Diese muss gleich der Arbeit der verursachten Gelenkmomente sein. Sei $\vec{f}$ ein kartesischer Kraft- und Momentvektor der Dimension 6×1, der am Greifer des Manipulators wirkt, und $\vec{\tau}$ ein Momentvektor der Dimension 6×1 an den sechs Gelenken des Manipulators. Sei zusätzlich $\delta\vec{\mathcal{P}}$ die kartesische Lageänderung des Greifers und $\delta\vec{\theta}$ die entsprechende Drehung der Gelenke in einem sehr kleinen zeitlichen Abschnitt δt. Dann ist die Arbeit durch das innere Produkt von $\vec{f}$ mit $\delta\vec{\mathcal{P}}$ bzw. von $\vec{\tau}$ mit $\delta\vec{\theta}$ angegeben:

$$\vec{f} \cdot \delta\vec{\mathcal{P}} = \vec{\tau} \cdot \delta\vec{\theta} \tag{C.33}$$

Dividiert man durch die Zeit δt, in der die Lageänderung des Manipulators stattfindet, dann erhält man:

$$\vec{f} \cdot \frac{\delta\vec{\mathcal{P}}}{\delta t} = \vec{\tau} \cdot \frac{\delta\vec{\theta}}{\delta t} \tag{C.34}$$

Da der zeitliche Abschnitt sehr klein ist, kann man Gleichungen C.31 verwenden:

$$\vec{f}^{T}\mathbf{J}(\vec{\theta})\dot{\vec{\theta}} = \vec{\tau}^{T}\dot{\vec{\theta}} \tag{C.35}$$

und daher folgt:

$$\mathbf{J}^{T}(\vec{\theta})\,\vec{f} = \vec{\tau} \tag{C.36}$$

Gleichung C.36 gibt dann die Beziehung zwischen den Kräften, die auf den Manipulatorgreifer wirken, und den Momenten an den Gelenken, die sie verursachen, wieder. Sie erlaubt es, dreidimensionale Kraftvektoren auf den hoch-dimensionalen Gelenkraum des Manipulators relativ einfach abzubilden. Dieses Prinzip machen sich auch die Vektorfeldverfahren zunutze, um eine reaktive Hindernisvermeidung für Manipulatoren zu realisieren (siehe auch Unterkapitel 2.2.2).

C.1.3 Pfadausführung

Ein Pfad (*Path*) besteht aus einer Sequenz von kartesischen Greiferpositionen bzw. von Gelenkstellungen, auch als Stützstellen bekannt, die der Roboter nacheinander abfahren muss, um zum Ziel zu kommen. Der Pfad ist somit eine reine geometrische Beschreibung einer Bewegung. Eine Trajektorie (*Trajectory*) beinhaltet dagegen zusätzliche Information bezüglich der zeitlichen Ausführung der Bewegung; hier ist die erwünschte Position und Geschwindigkeit des Roboters zu jedem Zeitpunkt eindeutig definiert.

Damit der Roboterarm einen Pfad, den ein Verhalten ermittelt hat, abfahren kann, muss er ihn in eine Trajektorie transformieren, die er dann mit Hilfe eines Trajektoriereglers ausführt. Da aber die Bestimmung der Verhaltensausgabe eine gewisse Zeit benötigt und der Roboter in diesem Zeitabschnitt sich weiterbewegt, ist der Pfad zur tatsächlichen Ausführungszeit nicht mehr aktuell und muss vor der Ausführung entsprechend angepasst werden.

Pfadkorrektur

Die Pfadkorrektur bezieht sich auf die Anpassung der Stützstellen des Pfades, so dass die Roboterbewegung während der Berechnung eines Verhaltens kompensiert wird. Dabei nimmt der neue Anfangspunkt des Pfades die aktuelle Position des Roboters an und die beiden letzten Stützstellen bleiben erhalten, um das ursprüngliche Ziel und die Bewegungsrichtung an der Zielposition nicht zu verfälschen. Alle anderen Stützstellen müssen so geändert werden, dass sich der neue Pfad dem ursprünglichen Pfad anpasst, je mehr er sich dem Ziel nähert.

Sei $\mathcal{P}_{sol} = \{\vec{\mathcal{P}}_{1,sol}, \ldots, \vec{\mathcal{P}}_{k,sol}\}$ der gegebene Pfad, $\vec{\mathcal{P}}_{1,tat}$ die aktuelle Anfangsposition des Manipulators und k die Anzahl der Stützstellen des Pfades. Zur Korrektur des Pfades wird der Fehler in der Anfangsposition mit der Anzahl der noch bevorstehenden Stützstellen $k - l - 2$ im Verhältnis zu $k - 2$, also zu der Anzahl der Stützstellen bis der vorletzten Stützstelle, gewichtet und zur anzupassende Stützstelle hinzuaddiert. Demnach wird die angepasste Stütztelle $\vec{\mathcal{P}}_{l,kor}$ wie folgt bestimmt:

$$\vec{\mathcal{P}}_{l,kor} = \vec{\mathcal{P}}_{l,sol} + (\vec{\mathcal{P}}_{1,tat} - \vec{\mathcal{P}}_{1,sol})\frac{k-l-2}{k-2} \tag{C.37}$$

Dadurch wird die Abweichung vom gewünschten Pfad geringer, je mehr Stützstellen der Manipulator durchläuft. Gleichung C.37 ergibt einen glatten Pfad, der die ursprüngliche Form der Bewegung erhält.

Trajektorieregler

Aus dem Ausgabepfad eines Verhaltens muss eine entsprechende Trajektorie bestimmt werden, die mit einem Trajektorieregler abgefahren wird. Da der Manipulator mit Gelenkgeschwindigkeiten angesteuert wird, müssen zuerst die Stützstellen, falls sie im kartesischen Raum definiert sind, anhand der inversen Kinematik in Gelenkstellungen umgewandelt werden (siehe Abschnitt C.1.2)[6]. Danach wird für jedes Gelenk eine kontinuierliche Funktion der Winkelstellung in Abhängigkeit von der Zeit definiert, die das Gelenk von der aktuellen Stellung in die Winkelstellung der nächsten Stützstelle überführt. Dabei müssen alle Gelenke die Winkelstellungen der Stützstellen zu derselben Zeit erreichen, um die angegebene Position im Raum anzunehmen. Der gesamte Programmablauf des Trajektoriereglers ist in Abbildung C.9 dargestellt.

Als Funktionen für die Winkelstellungen werden in dieser Arbeit kubische Polynome verwendet (Gleichung C.38), da sie stetig differenzierbar sind und eine kontinuierliche erste und zweite Ableitungen haben. Dies erzeugt eine glatte Bewegung der Gelenke und hält dadurch die mechanische Anspannung der Gelenkmotoren gering [Cra89]. Sei $\theta_i(t)$ der Winkelwert von Gelenk i zu Zeitpunkt t, dann ist die Funktion für die Winkelstellungen:

$$\theta_i(t) = a_{0i} + a_{1i}t + a_{2i}t^2 + a_{3i}t^3 \tag{C.38}$$

Aus der Gleichung C.38 ergibt sich ein parabolisches Geschwindigkeitsprofil:

$$\dot{\theta}_i(t) = a_{1i} + 2a_{2i}t + 3a_{3i}t^2 \tag{C.39}$$

und ein lineare Beschleunigung:

$$\ddot{\theta}_i(t) = 2a_{2i} + 6a_{3i}t \tag{C.40}$$

[6]Die Ausführung der Trajektorie wird im Gelenkraum geregelt, um somit Singularitäten der Roboter Jacobi Matrix (Gleichung C.32, die bei einer Regelung im kartesischen Raum vorkommen können, zu umgehen.

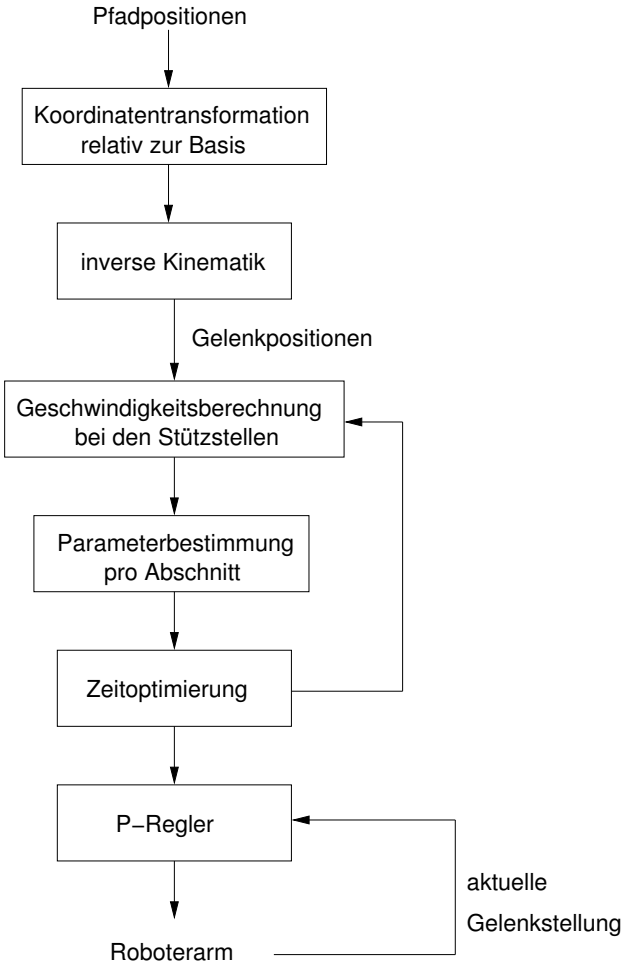


Abbildung C.9: Programmablauf des Trajektoriereglers.

Der Manipulator erreicht dann die maximale Geschwindigkeit zum Zeitpunkt:

$$t = \frac{-2a_{2i}}{6a_{3i}} \tag{C.41}$$

Sei ein Pfad als eine Folge von Gelenkstellungen $\{\vec{\theta}_0, ..., \vec{\theta}_{t_k}\}$ definiert[7], wobei $\vec{\theta}_0$ die aktuelle Gelenkstellung angibt. Zuerst wird ein einzelner Abschnitt j zwischen zwei nacheinanderfolgenden Stützstellen $\vec{\theta}_j$ und $\vec{\theta}_{j+1}$ betrachtet und danach werden die Resultate auf dem gesamten Pfad generalisiert.

Für einen Abschnitt j sind die vier Parameter $a_{0ji}, a_{1ji}, a_{2ji}, a_{3ji}$ des kubischen Polynomes zunächst unbekannt. Deswegen werden 4 Randbedingungen (*Constraints*) für einen einzelnen Abschnitt gebraucht, um die Parameterwerte zu bestimmen. Dazu werden die Randbedingungen zu den Zeitpunkten $t = t_j$ und $t = t_{j+1}$, also am Anfang und beim Erreichen der $j+1$ Stützstelle, ermittelt. Zwei Randbedingungen ergeben sich aus der Anfangsposition $\theta_i(t_j)$ und der erwünschten Zielposition $\theta_i(t_{j+1})$ für Gelenk i. Zwei weitere kann man aus den gewünschten Anfangs- und Endgeschwindigkeiten, $\dot{\theta}_i(t_j)$ und $\dot{\theta}_i(t_{j+1})$, herleiten.

[7]Ist der Pfad im kartesischen Raum definiert, dann werden zuerst mit der inversen Kinematik die kartesischen Stützstellen in Gelenkstellungen transformiert.

Solange man einen einzelnen Abschnitt j betrachtet kann man $t_j = 0$ und $t_{j+1} = t_f$ setzen; dann folgt aus Gleichungen C.38, C.39, C.40 und den vier Randbedingungen [Cra89], [SS00a]:

$$\begin{aligned} a_{0ji} &= \theta_i(t_j) \\ a_{1ji} &= \dot{\theta}_i(t_j) \\ a_{3ji}t_f^3 + a_{2ji}t_f^2 + a_{1ji}t_f + a_{0ji} &= \theta_i(t_{j+1}) \\ 3a_{3ji}t_f^2 + 2a_{2ji}t_f + a_{1ji} &= \dot{\theta}_i(t_{j+1}) \end{aligned} \tag{C.42}$$

Löst man Gleichungsystem C.42 nach a_{ji} auf, dann ergibt sich:

$$\begin{aligned} a_{0ji} &= \theta_i(0) \\ a_{1ji} &= \dot{\theta}_i(0) \\ a_{2ji} &= \frac{3}{t_f^2}(\theta_i(t_{j+1}) - \theta_i(t_j)) - \frac{2}{t_f}\dot{\theta}_i(t_j) - \frac{1}{t_f}\dot{\theta}_i(t_{j+1}) \\ a_{3ji} &= -\frac{2}{t_f^3}(\theta_i(t_{j+1}) - \theta_i(t_j)) - \frac{1}{t_f^2}(\dot{\theta}_i(t_j) + \dot{\theta}_i(t_{j+1})) \end{aligned} \tag{C.43}$$

Diese Parameterberechnung muss für jeden Abschnitt j zwischen zwei nacheinanderfolgenden Stützstellen stattfinden. Bei der allerersten Stützstelle kann man die aktuelle Geschwindigkeit als Randbedingung verwenden, bei der allerletzten kann die Geschwindigkeit auf Null gesetzt werden. Im Allgemeinen sind jedoch die Geschwindigkeiten an den restlichen Stützstellen nicht vorab bekannt und sind deswegen nicht als Randbedingungen verwendbar. Sind aber die erwünschten Zeitpunkte $t_1, ..., t_k$, zu denen die entsprechenden Stützstellen erreicht werden müssen, vorhanden, dann kann man diese als zusätzliche Randbedingungen, eine pro Abschnitt, einsetzen. Eine zusätzliche Randbedingung ergibt sich, wenn man von einer kontinuierlichen Beschleunigung an den Stützstellen ausgeht. Seien $\theta_i(t_l)$, $\theta_i(t_{l+1})$, $\theta_i(t_{l+2})$ drei aufeinanderfolgende Stützstellen, dann resultiert aus C.40, C.43 und den zusätzlichen Randbedingungen für beide Abschnitte:

$$\frac{2\dot{\theta}_i(t_l)}{t_{l+1}-t_l} + (\frac{4}{t_{l+1}-t_l} + \frac{4}{t_{l+2}-t_{l+1}})\dot{\theta}_i(t_{l+1}) + \frac{2\dot{\theta}_i(t_{l+2})}{t_{l+2}-t_{l+1}} = \frac{6(\theta_i(t_{l+2})-\theta_i(t_{l+1}))}{(t_{l+2}-t_{l+1})^2} + \frac{6(\theta_i(t_{l+1})-\theta_i(t_l))}{(t_{l+1}-t_l)^2} \tag{C.44}$$

So kann man $k-1$ Gleichungen mit $\dot{\theta}_i(l)$, $l = 1, \ldots, k-1$, als Unbekannte aufstellen. Diese $k-1$ Gleichungen können in Matritzenform zusammengefasst werden:

$$\mathbf{A}\vec{\dot{\theta}}_i = \vec{b} \tag{C.45}$$

mit:

$$\mathbf{A} = \begin{pmatrix} \beta_0 & \gamma_0 & 0 & 0 & \ldots & 0 & 0 \\ \alpha_1 & \beta_1 & \gamma_1 & 0 & \ldots & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & \ldots & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & \ldots & & 0 & \alpha_{k-3} & \beta_{k-3} & \gamma_{k-3} \\ 0 & \ldots & & 0 & 0 & \alpha_{k-2} & \beta_{k-2} \end{pmatrix} \tag{C.46}$$

$$\vec{\dot{\theta}}_i = (\dot{\theta}_i(t_1), \dot{\theta}_i(t_2), \ldots, \dot{\theta}_i(t_{k-1}))^T \tag{C.47}$$

$$\vec{b} = (\delta_0, \delta_1, \ldots, \delta_{k-2})^T \tag{C.48}$$

und

$$\begin{aligned}\alpha_l &= \frac{2}{t_{l+1}-t_l}\\ \beta_l &= (\frac{4}{t_{l+1}-t_l}+\frac{4}{t_{l+2}-t_{l+1}})\\ \gamma_l &= \frac{2}{t_{l+2}-t_{l+1}}\\ \delta_l &= \frac{6(\theta_i(t_{l+2})-\theta_i(t_{l+1}))}{(t_{l+2}-t_{l+1})^2}+\frac{6(\theta_i(t_{l+1})-\theta_i(t_l))}{(t_{l+1}-t_l)^2}\end{aligned} \tag{C.49}$$

Sonderfälle sind δ_0 und δ_{k-2} mit

$$\begin{aligned}\delta_0 &= \frac{6(\theta_i(t_2)-\theta_i(t_1))}{(t_2-t_1)^2}+\frac{6(\theta_i(t_1)-\theta_i(t_0))}{(t_1-t_0)^2}-\frac{2}{t_1-t_0}\dot{\theta}_i(t_0)\\ \delta_{k-2} &= \frac{6(\theta_i(t_k)-\theta_i(t_{k-1}))}{(t_k-t_{k-2})^2}+\frac{6(\theta_i(t_{k-1})-\theta_i(t_{k-2}))}{(t_{k-1}-t_{k-2})^2}-\frac{2}{t_k-t_{k-1}}\dot{\theta}_i(t_{k-2})\end{aligned} \tag{C.50}$$

Dann sind die Gelenkgeschwindigkeiten an den Stützstellen:

$$\vec{\dot{\theta}}_i = \mathbf{A}^{-1}\vec{b} \tag{C.51}$$

Die Parameter der Trajektorie können anschließend aus Gleichungssystem C.43 berechnet werden.

Diese Berechnung setzt voraus, dass die Zeit bekannt ist, um von einer Stützstelle zur nächsten zu gelangen. Meistens steht diese Zeit nicht vorab fest, es ist jedoch erwünscht, die Trajektorie so schnell wie möglich auszuführen. Die Ausführungszeit der Trajektorie wird aber von der maximal möglichen Manipulatorgeschwindigkeit begrenzt. Deswegen wird in dieser Arbeit zu Beginn die Trajektorie mit einem geringen Zeitwert von 0,5 sec pro Abschnitt berechnet. Ist in einem Trajektorieabschnitt die errechnete maximale Geschwindigkeit (siehe auch Gleichungen C.39 und C.41) größer als die maximal mögliche Geschwindigkeit des Manipulators, dann wird die Zeit für die Ausführung des entsprechenden Zeitabschnittes um einen konstanten Zeitschritt erhöht. Nach Überprüfung aller Abschnitte wird die Trajektorieberechnung nach Gleichungen C.51 und C.43 und die Überprüfung der maximalen Geschwindigkeit wiederholt, bis kein Abschnitt eine höhere Geschwindigkeit als die maximal mögliche aufweist.

Die errechnete Trajektorie wird anschließend mit einem P-Regler ausgeführt. Sei $\theta_i(t)$ die berechnete Position, $\dot{\theta}_i(t)$ die berechnete Geschwindigkeit und $\theta'_i(t)$ die tatsächliche Position von Gelenk i zu Zeitpunkt t. Dann versucht der P-Regler den Positionsfehler zu minimieren:

$$\dot{\theta}'_i(t) = \dot{\theta}_i(t) + k_p(\theta_i(t)-\theta'_i(t)) \tag{C.52}$$

wobei k_p der Verstärkungsfaktor des P-Reglers ist, der experimentell ermittelt wurde.

Die berechnete Geschwindigkeit $\dot{\theta}'_i(t)$ wird anschließend zur Motorregelung des i-ten Gelenks übertragen.

C.2 Epipolare Geometrie

Aus einer bekannten geometrischen Anordnung zweier Kameras lässt sich eine wirkungsvolle Einschränkung in Bezug auf die Lage der Abbildungen eines Raumpunktes in den Bildebenen ableiten[8].

[8] Vergleiche hierzu auch Abschnitt 4.2.1.

Die epipolare Geometrie (*Epipolar Geometry*) ermöglicht es, zur Abbildung eines Raumpunktes im ersten Bild eine Gerade in der zweiten Aufnahme zu berechnen, auf der die Abbildung Raumpunktes im zweiten Bild liegen muss.

Betrachtet man die in Abbildung C.10 vorgestellte Anordnung zweier Kameras, so wird ein Raumpunkt M auf die Bildpunkte mit homogenen Pixelkoordinaten ${}^{K1}\vec{U}$ und ${}^{K2}\vec{U}$ abgebildet. Diese befinden sich auf der so genannten epipolaren Ebene (*Epipolar Plane*), die von den Brennpunkten der Kameras und dem Raumpunkt M definiert wird. Ihre Schnitte mit den beiden Bildebenen sind die zwei epipolaren Linien (*Epipolar Line*). Ein zu einem Pixelpunkt ${}^{K_1}\vec{U}$ der ersten Aufnahme korrespondierender Bildpunkt ${}^{K_2}\vec{U}$ kann sich nur auf der zugehörigen epipolaren Linie in der zweiten Aufnahme befinden [Fau93].

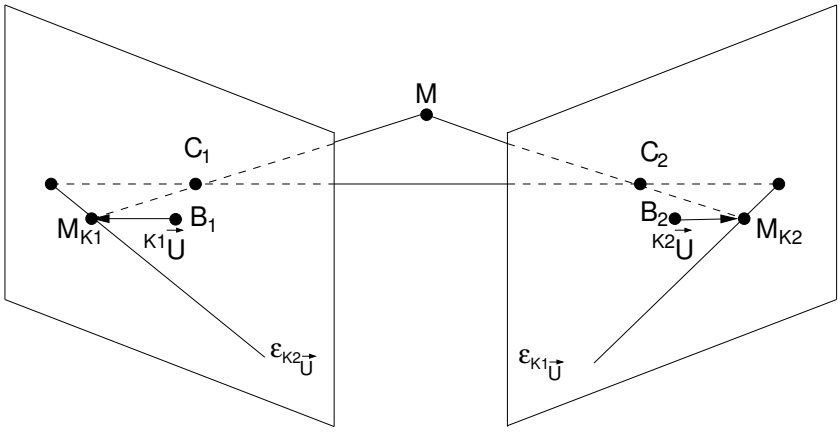


Abbildung C.10: Epipolare Geometrie einer Szene. C_1, C_2 sind die Brennpunkte der Kameras und B_1, B_2 die entsprechenden Bildhauptpunkte.

Diese Eigenschaft kann die Suche nach Punktkorrespondenzen von der zweidimensionalen Bildebene auf die korrespondierende eindimensionale epipolare Linie, beschränken. In der Praxis jedoch muss ein Bereich um die epipolare Linie berücksichtigt werden, da aufgrund von Rauschen die extrahierten Eckpunkte nicht zwingend auf der entsprechenden epipolaren Linie liegen.

Nach der Abbildungvorschrift der Kameras (Gleichung 3.4) gilt für die homogenen Pixelkoordinaten ${}^{K_1}\vec{U}$ und ${}^{K_2}\vec{U}$ der Projektionen des Raumpunktes M auf beiden Aufnahmen:

$$\begin{aligned} {}^{K_1}\vec{U} &= \mathbf{K}\,\mathbf{M}\,{}^{K_1}_{W}\mathbf{T}\,{}^{W}\vec{P} \\ {}^{K_2}\vec{U} &= \mathbf{K}\,\mathbf{M}\,{}^{K_2}_{W}\mathbf{T}\,{}^{W}\vec{P} \end{aligned} \tag{C.53}$$

wobei ${}^{W}\vec{P}$ der Vektor der homogenen kartesischen Koordinaten von M im Weltkoordinatensystem ist.

Wird als Ursprung der Welt das Koordinatensystem der ersten Kamera angenommen, dann ist ${}^{K_1}_{W}\mathbf{T} = \mathbf{I}$, mit $\mathbf{I}$ die Einheitsmatrix, und ${}^{K_2}_{W}\mathbf{T} = {}^{K_2}_{K_1}\mathbf{T}$, mit ${}^{K_2}_{K_1}\mathbf{T}$ die Transformationsmatrix der ersten Kamera zur zweiten. Wird nur eine Kamera verwendet (*Stereo-by-Motion* Ansatz), dann entspricht ${}^{K_2}_{K_1}\mathbf{T}$ der Lageänderung der Kamera zwischen den beiden Aufnahmen.

Sei ${}^{K_2}\vec{P}_{K_1 Orig} = (x_{K_1 Orig}, y_{K_1 Orig}, z_{K_1 Orig}, 1)^T$ der homogene Positionsvektor des Bildhauptpunktes der ersten Kamera relativ zum Hauptpunkt der zweiten Kamera. Faugeras [Fau93] beweist, dass für

zwei korrespondierende Bildpunkte mit Pixelkoordinaten ${}^{K_1}\vec{U}$ und ${}^{K_2}\vec{U}$ folgende Beziehung gilt:

$$ {}^{K_1}\vec{U}\ (\mathbf{K}^{-1})^T\ \mathbf{S}(\ {}^{K_2}\vec{P}_{K_1Orig})\ {}^{K_2}_{K_1}\mathbf{R}^{-1}\mathbf{K}^{-1}\ {}^{K_2}\vec{U} = 0 \tag{C.54}$$

mit ${}^{K_2}_{K_1}\mathbf{R}^{-1}$ die Rotationsmatrix des Kamerakoordinatensystems in der ersten Aufnahme bezüglich der zweiten Aufnahme und

$$\mathbf{S}({}^{K_2}\vec{P}_{K_1Orig}) = \begin{pmatrix} 0 & -z_{K_1Orig} & y_{K_1Orig} \\ z_{K_1Orig} & 0 & -x_{K_1Orig} \\ -y_{K_1Orig} & x_{K_1Orig} & 0 \end{pmatrix}$$

Der mittlere Teil von Gleichung C.54 ist ausschließlich von den intrinsischen und extrinsischen Kameraparametern abhängig und kann in einer Matrix $\mathbf{F}$, der so genannten Fundamentalmatrix (*Fundamental Matrix*), zusammengefasst werden:

$$\mathbf{F} = (\mathbf{K}^{-1})^T\mathbf{S}({}^{K_2}\vec{P}_{K_1Orig}){}^{K_2}_{K_1}\mathbf{R}^{-1}\mathbf{K}^{-1} \tag{C.55}$$

Die epipolare Linie in der zweiten Kameraaufnahme, die dem Pixelpunkt mit Bildkoordinaten ${}^{K_1}\vec{u}$ in der ersten Aufnahme entspricht, ist dann aus Gleichung C.54:

$$\epsilon_{K_1\vec{u}}(u,v) : au + bv + c = 0 \tag{C.56}$$

mit

$$(a,b,c)^T = {}^{K_1}\vec{u}^T\mathbf{F}$$

Der Abstand eines Bildpunktes aus der zweiten Aufnahme zur epipolaren Linie beträgt in diesem Fall:

$$d(\epsilon_{K_1\vec{u}}, {}^{K_2}\vec{u}_i) = |\frac{a\ {}^{K_2}u_i + b\ {}^{K_2}v_i + c}{\sqrt{a^2+b^2}}| \tag{C.57}$$

Für zwei korrespondierende Bildpunkte gilt:

$$d(\epsilon_{K_1\vec{u}},\ {}^{K_2}\vec{u}) = 0 \tag{C.58}$$

C.3 Radial Basis Function Netze

Ein wichtiger Anwendungsbereich von künstlichen neuronalen Netzen ist das Schätzen des Verlaufs einer Funktion $y(\vec{x}_i)$ aus einer Menge von Daten. Die Berechnung der Näherung stützt sich auf eine aus m Wertepaaren $(\vec{x}_i, y_i)$ bestehenden Trainingsmenge. Da die Ausgabe y_i der Funktion für jeden Vektor $\vec{x}_i$ benötigt wird, lässt sich die Schätzung als ein überwachtes Lernverfahren implementieren.

Eine Möglichkeit, um die Schätzung zu realisieren, ist die Darstellung der Funktion als eine Kombination von einfachen Modellen, so genannten Basisfunktionen, wobei die Anzahl von Basisfunktionen unabhängig von der Größe der Trainingsmenge ist. So kann beispielsweise die Funktion $f(\vec{x})$ mit einer linearen Kombination von b Basisfunktionen $\phi_i(\vec{x})$ angenähert werden:

$$\widehat{y}(\vec{x}) = \sum_{i=1}^{b} w_i\phi_i(\vec{x}) \tag{C.59}$$

Die Bestimmung der Parameter w_i findet durch die Lösung eines linearen Gleichungssystems statt, das mit Hilfe der m Trainingsdaten $(\vec{x_i}, y_i)$ aufgestellt wird.

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} \phi_1(\vec{x_1}) & \phi_2(\vec{x_1}) & \cdots & \phi_b(\vec{x_1}) \\ \phi_1(\vec{x_2}) & \phi_2(\vec{x_2}) & \cdots & \phi_b(\vec{x_2}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\vec{x_m}) & \phi_2(\vec{x_m}) & \cdots & \phi_b(\vec{x_m}) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} \tag{C.60}$$

oder in Matrixform zusammengefasst:

$$\vec{y} = \mathbf{\Phi}\vec{w} \tag{C.61}$$

Um die Gewichte zu berechnen setzt man die Pseudoinverse $\mathbf{\Phi}^+$ der Matrix $\mathbf{\Phi}$ ein:

$$\vec{w} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\vec{y} = \mathbf{\Phi}^+\vec{y} \tag{C.62}$$

Diese Beziehungen lassen sich mit einem neuronalen Netz realisieren. RBF Netze stellen einen spezialisierten Fall solcher neuronalen Netzen dar, die für $\phi_i()$ radiale Basisfunktionen einsetzen [Kra03b]. Als Basisfunktion kommt oft die nicht-normierte Gaußfunktion zur Anwendung:

$$\phi(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{C.63}$$

Jede Basisfunktion erhält am Eingang den Abstand des Eingabevektors $\vec{x}$ von einem der Funktion zugehörigen Stützstellenvektor $\vec{x_i}$, der während der Trainingsphase festgelegt wird. Somit erhält man für die Approximation der Funktion:

$$\widehat{y}(\vec{x}) = \sum_{i=1}^{b} w_i\phi_i(\|\vec{x} - \vec{x_i}\|) \tag{C.64}$$

Diese Beziehung lässt sich als ein dreischichtiges neuronales Netz interpretieren (siehe auch Abbildung C.11). Die erste Schicht funktioniert als Eingang des Netzes und die Anzahl der Neuronen entspricht der Anzahl der Elemente des Merkmalsvektors, der klassifiziert werden soll. Die Ausgaben der verdeckten Schicht entsprechen den ausgewerteten radialen Basisfunktionen $\phi_i(\|\vec{x} - \vec{x_i}\|)$. Die letzte Schicht gewichtet die Ausgaben mit w_i, die aus Gleichung C.62 berechnet werden, und summiert sie.

Die versteckte Neuronenebene ist verantwortlich für das Approximationsverhalten des Netzes, da sie Gleichung C.64 umsetzt. Drei Elemente bestimmen das Verhalten eines Neurons i dieser Ebene:

- die Position der Stützstelle im Eingangsraum, d.h. der Wert von $\vec{x_i}$,
- die Methode zur Bestimmung des Abstandes des Eingangsvektor von der Stützstelle, also $\|\vec{x} - \vec{x_i}\|$, und
- die Aktivierungsfunktion des Neurons, also $\phi_i()$.

Die Position der Stützstellen wird während der Trainingsphase festgelegt. Nach Haykin [Hay94] gibt es drei Möglichkeiten:

- sie können willkürlich gesetzt werden,

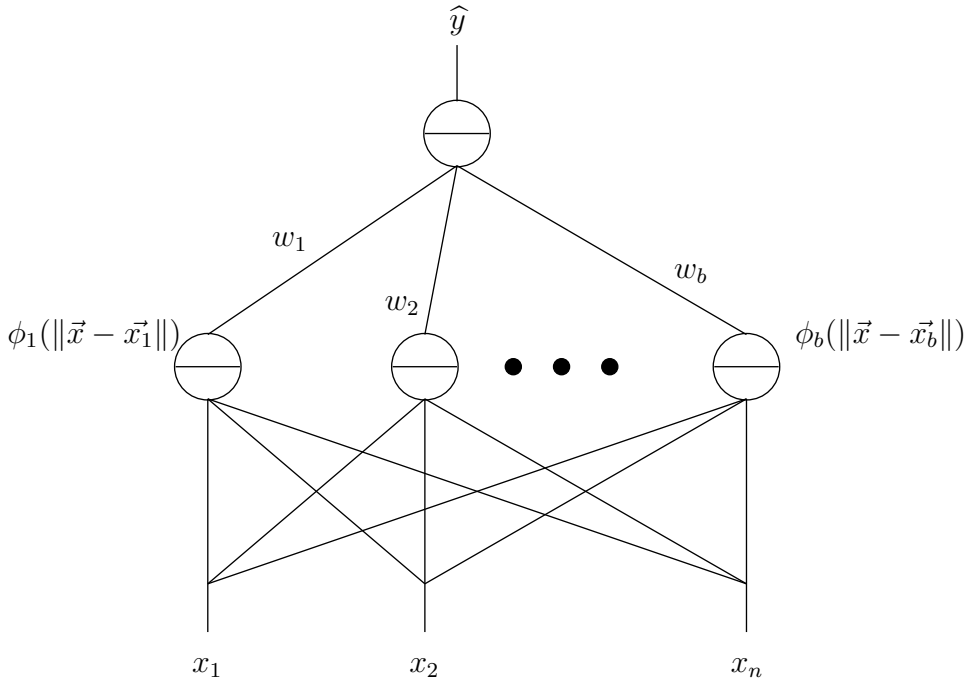


Abbildung C.11: *Radial Basis Function* Netz mit einem Ausgang.

- sie werden über einen überwachten Ansatz mit einem Gradientenabstiegsverfahren ausgewählt, oder
- sie werden über einen selbstorganisierenden Ansatz berechnet[9].

Um den Abstand $\|\vec{x} - \vec{x_i}\|$ zu messen wird oft die euklidische Distanz eingesetzt. Alternativ kann man auch die Mahalanobis Distanz verwenden. Die Aktivierungsfunktion eines Neurons der verdeckten Schicht ist dann:

$$\phi(\|\vec{x} - \vec{x_i}\|) = e^{-\frac{(\|\vec{x} - \vec{x_i}\|)^2}{2\sigma^2}} \tag{C.65}$$

Dabei bestimmt die Standardabweichung σ den Einflussbereich des Neurons, also die Region der $\vec{x}$-Werte um die Stützstelle $\vec{x_i}$, in der das Neuron mit hohen Werten aktiv wird. Dadurch werden RBF-Netze trainiert nur auf lokale Stimuli zu reagieren; es werden nur die Neuronen aktiviert, deren Stützstelle $\vec{x_i}$ sich in der Nähe des Eingangsvektors befinden. Dadurch gehören RBF Netze zur Gruppe der lokalen Approximatoren[10], da nur die aktiven Neuronen zur Lösung beitragen[11] .

Die Ausgangsebene eines RBF Netzwerkes besteht aus Neuronen mit linearen Aktivierungsfunktionen. Alternativ kann man auch die sigmoidal Aktivierungsfunktion einsetzen:

$$f(x) = \frac{1}{1 + e^{-\frac{x}{T}}} \tag{C.66}$$

wobei T eine Konstante ist.

[9] In dieser Arbeit kommt der selbstorganisierende Ansatz zum Einsatz.

[10] Zu den lokalen Approximatoren gehören beispielsweise auch die SOM Netze.

[11] Bei *Back-Propagation* Netzen tragen dagegen alle Neuronen zur Lösung bei. BP Netze produzieren auch immer eines Ausgabe und klassifizieren den Eingangsvektor zu einer trainierten Gruppe, was bei lokalen Approximatoren nicht der Fall ist.

Außer bei Funktionsapproximationen werden RBF Netze auch für Klassifikationsprobleme eingesetzt. Die zu approximierende Funktion besagt in diesem Fall, zu welcher Klasse ein bestimmtes Muster gehört. Dabei wird pro Klasse ein Ausgangsneuron definiert, das als Ausgabe Werte von 0 bis 1 hat, wobei 1 die sichere Zugehörigkeit des Eingangsvektors zur Klasse bedeutet. Haykin [Hay94] zeigt, dass die Ausgabe der Ausgangsschicht einen Wahrscheinlichkeitsvektor der Klassenzugehörigkeit des Eingangs entspricht.

C.4 Fuzzy Logik

Im Fall eines komplexen, nichtlinearen Systems ist es in der Regel schwierig ein exaktes mathematisches Modell aufzustellen, um einen entsprechenden Regler für das System zu konzipieren. Alternativ kann ein Mensch relativ anschaulich komplexe Regelvorgänge mit einer Menge von Regeln beschreiben. In den Regeln werden im Allgemeinen keine exakte Werte angegeben, sondern es handelt sich um eine qualitative, unscharfe (*fuzzy*) Beschreibung der Größen.

Die Theorie der unscharfen Mengen (*Fuzzy Sets*) erlaubt es, unscharfes Wissen zu beschreiben und daraus Schlüsse zu ziehen. Im Gegensatz zur konventionellen Mengentheorie, die eindeutig festlegt, ob ein Element zu einer Menge gehört oder nicht, kann in der Theorie der unscharfen Mengen durch die Definition von Zugehörigkeitsfunktionen (*Membership Functions*) ein Element zu einem bestimmten Grad zu einer oder mehreren Mengen gehören. So kann z.B. der Temperaturwert 22^0C mit einem Zugehörigkeitsgrad von 0.75 zur unscharfen Menge `angenehm` und mit 0.25 zu `kalt` gehören. Diese Beschreibungsform macht sich Fuzzy Logik [Zad73] zunutze, um eine gemessene numerische Eingangsgröße qualitativ zu beschreiben und so den Ausdruck von Regeln zu ermöglichen:

```
WENN Temperatur IST angenehm UND Luftfeuchtigkeit IST
niedrig, DANN Klimaanlage IST abgestellt.
WENN Temperatur IST kalt UND Luftfeuchtigkeit IST niedrig,
DANN Klimaanlage IST niedrig heizen.
```

Dabei werden `Temperatur`, `Luftfeuchtigkeit` und `Heizung` als linguistische Variablen bezeichnet, da sie eine qualitative Beschreibung der gemessenen Werte der Eingangs- und Ausgangsgrößen vornehmen.

Bei einem Fuzzy Logik System kann man mehrere Verarbeitungsschritte unterscheiden, wie in Abbildung C.12 zu sehen ist [Kra03b]. Zunächst findet in der Fuzzifizierung-Phase die qualitative Beschreibung der gemessenen numerischen Eingangsgrößen durch linguistische Variablen statt. Dabei wird jede linguistische Variable mit mehreren unscharfen Mengen, so genannten linguistischen Termen, abgedeckt. Über Zugehörigkeitsfunktionen, eine pro unscharfe Menge, wird für jeden numerischen Wert ein Zugehörigkeitsgrad zu den linguistischen Termen zugeordnet. Als Zugehörigkeitsfunktionen kommen beispielsweise Dreiecksfunktionen und normalverteilte Glockenkurven in Betracht (Abbildung C.13).

Die Wissensbasis kodiert das Expertenwissen über das System mit unscharfen Regeln, die folgendermaßen aussehen:

```
WENN x ist X UND y ist Y DANN z ist Z.
```

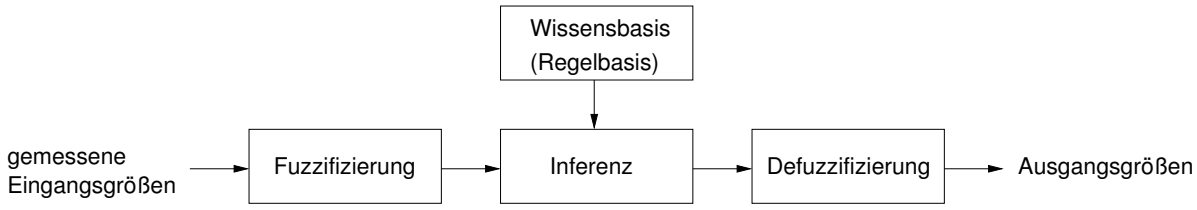
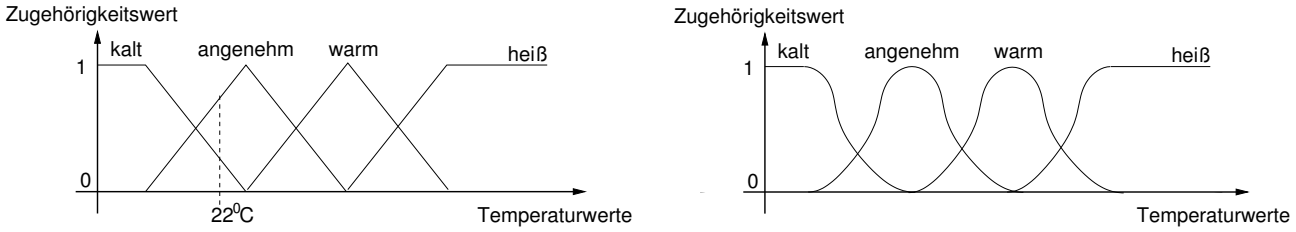


Abbildung C.12: Verarbeitungsschritte eines Fuzzy Logik Systems.



(a) (b)

Abbildung C.13: Mögliche Zugehörigkeitsfunktionen für die linguistische Variable `Temperatur`.

Hierbei stellen `x` und `y` die Eingangsgrößen und `z` die Ausgangsgröße dar. `X`, `Y` und `Z` repräsentieren einen Term der entsprechenden linguistischen Variable.

Zur Verknüpfung mehrerer Bedingungen in einer Regel stehen verschiedene Operatoren[12] zur Verfügung, wie:

- der MAX-Operator (logisches ODER) : $\max(\mu_A(x); \mu_B(x))$,
- der MIN-Operator (logisches UND) : $\min(\mu_A(x); \mu_B(x))$,
- der T-Conormen ASUM (logisches ODER) : $1 - (1 - \mu_A(x))(1 - \mu_B(x))$ und
- der T-Normen APROD (logisches UND) : $\mu_A \cdot \mu_B$.

Dabei bezeichnen $\mu_A(x)$ und $\mu_B(x)$ die Zugehörigkeitsfunktionen.

Anschließend wird mit Hilfe der Wissensbasis durch Inferenz ein qualitativer Ausgangswert in Form einer weiteren linguistischen Variable berechnet. Dieser Vorgang wird als unscharfes Schließen bzw. Inferenz bezeichnet. Dabei geht man davon aus, dass die Schlussfolgerung einer Regel immer zum gleichen Teil erfüllt ist wie ihre Vorbedingung. Hier sind zwei Inferenzmethoden gebräuchlich. Bei der Produktinferenz werden die unscharfen Mengen des DANN-Teils mit dem Zugehörigkeitsgrad des WENN-Teils der Regel multipliziert. Bei der MIN-Inferenz werden dagegen die unscharfen Mengen des DANN-Teils auf den Zugehörigkeitsgrad des WENN-Teils begrenzt (siehe auch Abbildung C.14). Betrachtet man Regeln, deren DANN-Teile sich auf denselben linguistischen Term der Ausgangsgröße beziehen, dann können sie mit einer ODER-Verknüpfung zusammengefasst werden. Der Zugehörigkeitsgrad der Ausgangs wird dann als das Maximum der Zugehörigkeitsgrade der WENN-Teile der mit ODER verknüpften Regeln berechnet [Kra03b].

In der Defuzzifizierungsphase werden aus den linguistischen Variablen der Ausgangsgrößen exakte numerische Werte hergeleitet. Dabei kann das Ergebnis der Inferenz aus Teilbeiträgen mehrerer

[12] Die verschiedenen Operatoren werden anwendungsspezifisch eingesetzt [Kra03b].

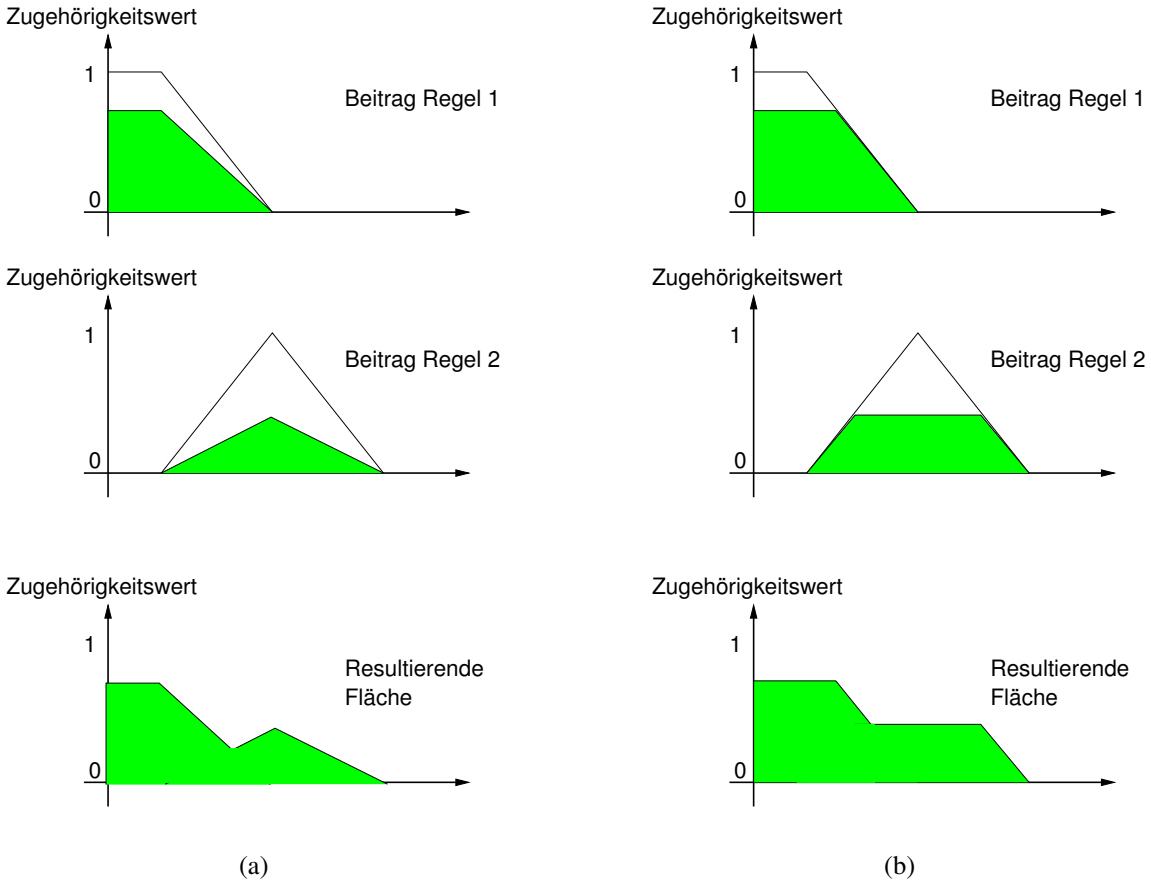


Abbildung C.14: Inferenzmethoden: (a) Produkt-Methode, (b) Min-Methode.

Regeln zusammengesetzt sein, die eine Vereinigungsfläche bilden. Das numerische Ergebnis ergibt sich mit Hilfe der Schwerpunkt-Methode oder der Maximum-Methode. In der Schwerpunktmethode wird der Ausgangswert als Abszissenwert x_s des Schwerpunktes der Vereinigungsfläche ermittelt (Abbildung C.15):

$$x_s = \frac{\int_{x_a}^{x_e} x f(x) dx}{\int_{x_a}^{x_e} f(x) dx} \tag{C.67}$$

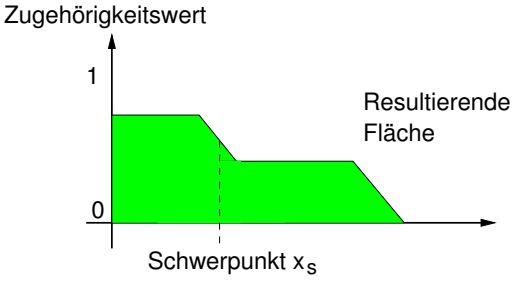


Abbildung C.15: Schwerpunktmethode.

Die Maximum-Methode ist eine stark vereinfachte aber vom Rechenaufwand ökonomische Lösung. Zunächst werden die unscharfen Ausgangsmengen durch Singletons angenähert. Diese werden im

Maximum einer Zugehörigkeitsfunktion gesetzt und deren Amplitude entspricht dem Zugehörigkeitsgrad an dieser Stelle (Abbildung C.16). Dann ist der numerische Wert der Ausgangsgröße:

$$x_s = \frac{\sum_{i=1}^{n} x_i f(x_i)}{\sum_{i=1}^{n} f(x_i)} \tag{C.68}$$

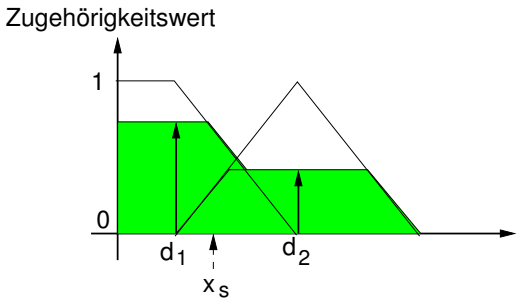


Abbildung C.16: Maximum-Methode.

C.5 Neurofuzzy

Ein grundlegendes Problem beim Entwurf eines Fuzzy Systems ist die Bestimmung der geeigneten Zugehörigkeitsfunktionen und der Regelbasis. Wenn kein Expertenwissen vorliegt, ist die Implementierung des Fuzzy Systems eine langwierige Prozedur. Deswegen ist in solchen Fällen das Erlernen der Parameter des Fuzzy Systems willkommen.

Neurofuzzy Systeme kombinieren die Lernfähigkeit neuronaler Netze mit der Transparenz von Fuzzy Systemen. Dabei werden die verschiedenen Phasen eines Fuzzy Logik Systems durch ein mehrstufiges neuronales Netze realisiert. Durch Rückführung des Fehlers zwischen Ist- und Sollausgabe während der Trainingsphase wird die Anpassung der Neuronengewichte und dadurch die Anpassung der Parameter des Fuzzy Logik Systems ermöglicht. Voraussetzung dafür ist die Differenzierbarkeit der Aktivierungsfunktion der Neuronen und daher müssen einige Änderungen beim Fuzzy System vorgenommen werden [Kra03b]. Insbesondere werden die Zugehörigkeitsfunktionen bei der Fuzzifizierung mit Gaußfunktionen implementiert. Bei der unscharfen Inferenz werden statt den MIN- und MAX Operatoren die differenzierbaren ASUM- und APROD- Operatoren verwendet. Zur Defuzzifizierung eignet sich bei Neurofuzzy Systemen die differenzierbare Maximum-Methode (Gleichung C.68). Somit werden die linguistischen Ausgangsvariablen mit Singletons angenähert.

Die Verwendung von Neuronen mit differenzierbarer Aktivierungsfunktion ermöglicht die Anwendung von Lernverfahren zum Anpassen der Position und Varianz der gaußschen Zugehörigkeitsfunktionen in der Fuzzifizierungsphase und der Position der Singletons der Ausgangsgrößen[13]. Das Training des Netzes kann mit der generalisierten Delta Regel, wie bei den normalen BP-Netzen stattfinden [Kra03b].

In Abbildung C.17 wird eine neuronales Netz zur Implementierung eines Fuzzy Logik Systems präsentiert. Das Netz besteht aus fünf Schichten. Dabei ist die Anzahl der Neuronen pro Schicht (und demnach die Anzahl der linguistischen Termen und der Regeln) ein Parameter, der unter Berücksichtigung der Anwendung gewählt werden muss. Die Neuronen der ersten Schicht repräsentieren

[13] In vielen Anwendungen reicht sogar das Anpassen der Singletonposition der linguistischen Ausgabevariablen aus.

die Eingangsvariablen und propagieren die Elemente des Merkmalsvektors zur nächsten Schicht. In der zweiten Schicht findet die Fuzzifizierung statt. Jedes Neuron dieser Schicht entspricht einem unscharfen linguistischen Term und die Aktivierungsfunktion des Neurons entspricht der Zugehörigkeitsfunktion. Demnach ist die Anzahl der Neuronen dieser Schicht gleich der Anzahl aller Terme von allen linguistischen Variablen. Verwendet man gaußsche Zugehörigkeitsfuktionen, dann ist die Aktivierungsfunktion:

$$f(x_i) = e^{-\frac{(x_i - m_{ij})^2}{2\sigma_{ij}^2}} \tag{C.69}$$

mit m_{ij} den Mittelwert und σ_{ij} die Standardabweichung der Gaußfunktion des j-ten Termes der i-ten linguistischen Variable.

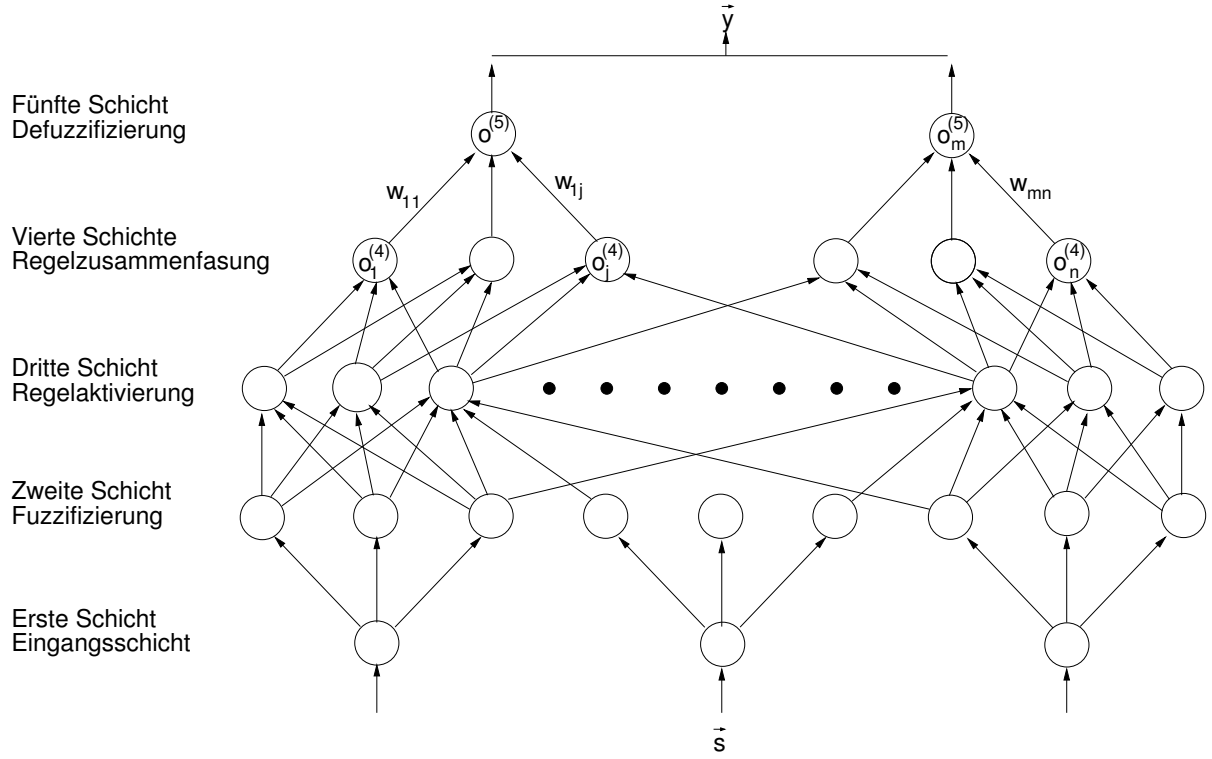


Abbildung C.17: Fuzzy Logik System realisiert mit einem künstlichen neuronalen Netz.

Die dritte Schicht entspricht der Regelbasis. Jedem Neuron ist hier eine Regel zugeordnet. Die Verbindungen zwischen zweiter und dritter Schicht repräsentieren die Voraussetzungen (WENN-Teil), die Kanten zwischen dritten und vierten Schicht die Schlussfolgerungen (DANN-Teil) der Regeln. Die Neuronen der dritten Schicht führen dann die Operation ASUM aus, um die Voraussetzungen zu kombinieren[14]. Entsprechend führen die Neuronen der vierten Schicht die Operation APROD bzw. MAX, um diejenigen Regeln zusammenzufassen, die dieselbe Schlussfolgerung haben.

Die letzte Schicht ist die Defuzzifizierungsschicht, die als Aktivierungsfunktion die differenzierbare Maximum-Methode anwendet. Demnach ist für die i-te linguistische Ausgangsvariable :

$$y_i = \frac{\sum_{j=1}^{n} m_{ij} x_j}{\sum_{j=1}^{n} x_j} \tag{C.70}$$

[14]Wenn man jedoch nur die Singletons der Defuzzifizierungsschicht anpassen möchte, kann man hier den MIN-Operator als Aktivierungsfunktion benutzen.

Die Gewichte w_{ij} dieser Schicht entsprechen den Abszissen der Singletons, also $w_{ij} = m_{ij}$, und x_j den Ausgaben der vierten Schicht, also $x_j = o_j^{(4)}$ (siehe auch Abbildung C.17.)

C.6 Temporal Differencing Verfahren und Q-learning Algorithmus

Beim Lernen mit Bewerter steht in vielen Anwendungen die Information, ob die ausgeführte Aktion sinnvoll gewesen ist, erst nach mehreren Schritten zur Verfügung[15]. Jedoch muss man sofort nach der Ausführung die Aktion bewerten und die Handlungsstrategie anpassen. Dies ist als *Temporal Credit-Assignement* Problem bekannt. In diesem Fall muss die Bewertung jeder Aktion nicht nur die gegenwärtige Situation berücksichtigen, sondern versuchen, die Auswirkung der Aktion zu vorhersagen. Ein verbreiteter Ansatz, um dieses Problem zu lösen, ist das *Temporal Differencing* (TD) Verfahren [SB98]. Beim TD-Ansatz wird die Erfahrung aus früheren Bewertungen benutzt, um das zukünftige Verhalten vorherzusagen; dadurch ist kein Modell der Umgebung nötig. In der einfachsten Version, dem TD(0) Algorithmus, wird die aktuell erhaltene Belohnung und der Fehler zwischen zwei zeitlich nacheinander folgenden Schätzungen der Wertfunktion verwendet, um die Wertfunktion entsprechend anzupassen bzw. die Schätzung zu verbessern:

$$V(Z_i) = V(Z_i) + a(r_{i+1} + \gamma V(Z_{i+1}) - V(Z_i)) \tag{C.71}$$

wobei $V(Z_i)$ den Wert der Wertfunktion für Zustand Z_i und r_{i+1} die Belohnung für die ausgeführte Aktion ist, die das System in den Zustand Z_{i+1} gebracht hat. Sutton [SB98] hat bewiesen, dass das Verfahren gegen die reale Wertfunktion konvergiert, was die Bestimmung der optimalen Wertfunktion erlaubt. Indem zwei nacheinanderfolgende Schritte sofort nach der Ausführung der Aktion betrachtet werden, kann der TD-Algorithmus so implementiert werden, dass das Lernen während des normalen Betriebs des Agenten stattfindet.

Falls die Wertfunktion mit einem neuronalen Netz realisiert ist, dann werden die Gewichte folgendermassen angepasst [Hay94]:

$$\vec{w}_{i+1} = \vec{w}_i + \eta(V_{i+1} - V_i) \sum_{k=1}^{i} \lambda^{i-k} \nabla_{\vec{w}} V_k \tag{C.72}$$

wobei V_i den Ausgang des Netzes zum Zeitpunkt i beschreibt, η die Lernrate und $\nabla_{\vec{w}} V_k$ der Gradientenvektor von V_k bezüglich des Gewichtvektors $\vec{w}$ zum Zeitschritt k. Der Parameter λ ist der so genannte *Forgetting Factor*, der Werte zwischen 0 und 1 annimmt. Gleichung C.72 stellt den allgemeinen Fall des TD-Algorithmus vor und für $\lambda = 0$ entspricht sie der Implementierung von TD(0).

Das *Q-learning* [Wat89] stellt eine Erweiterung des TD(0) Algorithmus dar. Es wird hier nicht nach dem nächsten Zustand Z_{i+1} gesucht, der die akkumulierte zukünftige Belohnung maximieren kann, sondern direkt nach der Aktion $\vec{\rho}$, die zu diesem Zustand führt. Diese Aktion maximiert auch den Wert der Aktionswertfunktion $Q(Z_i, \vec{\rho})$. Der *Q-learning* Algorithmus verläuft dann entsprechend des TD(0), nur wird hier anstatt der Zustandswertfunktion (*State Value Function*) $V(Z_i)$ die Aktionswertfunktion (*Action Value Function*) $Q(Z_n, \rho)$ verwendet.

[15] Dies ist beispielsweise der Fall bei der Kollisionsvermeidung und der Verhaltenskoordination.

C.7 Bayesian Belief Networks

Ein *Bayesian Belief Network* (BBN) ist ein gerichteter azyklischer Graph (*Directed Acyclic Graph* - DAG), der die Verbundwahrscheinlichkeit eines Ereignisses, das von mehreren Zufallsvariablen abhängt, effizient repräsentiert [Cha91]. Bestandteile eines BBN sind Knoten, die Zufallsvariablen darstellen, und gerichtete Kanten, die Abhängigkeiten zwischen den Knoten modellieren. Die Zufallsvariablen können sowohl unendlich viele Werte mit einer kontinuierlichen Wahrscheinlichkeitsverteilung oder eine endliche Anzahl von Zuständen mit einer diskreten Wahrscheinlichkeitsverteilung annehmen. Häufig wird auch der Sonderfall eines boolschen Knotens mit zwei diskreten Zuständen verwendet.

Eine Zufallsvariable hängt von den Zuständen seiner Vorgängerknoten im Graphen ab. Besitzt sie keinen Vorgänger, ist sie stochastisch unabhängig. Die zugehörige bedingte Wahrscheinlichkeitsverteilung wird im kontinuierlichen Fall als Funktion und im diskreten in der so genannten *Conditional Probability Table* (cpt) angegeben. Diese enthält für jede mögliche Kombination der Zustände der Vorgängerknoten die entsprechenden Wahrscheinlichkeiten für die Zustände des betreffenden Knotens. Damit lässt sich nach Heckerman [Hec95] ein BBN für eine Menge von Variablen $\{Z_1, \ldots, Z_n\}$ definieren durch (Abbildung C.18):

- eine Netzwerkstruktur, welche die Abhängigkeiten zwischen den Zufallsvariablen beschreibt, und
- eine Menge von bedingten Wahrscheinlichkeiten, welches einer cpt pro Knoten entspricht.

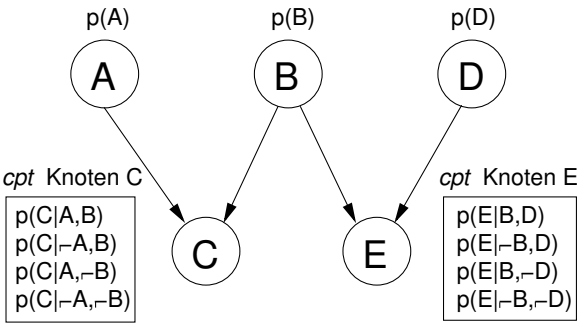


Abbildung C.18: Beispiel eines Bayesian Belief Networks

Im Allgemeinen wird die Wahrscheinlichkeit für ein Verbundereignis mit der Kettenregel berechnet:

$$p(Z_1, Z_2, \ldots, Z_k) = p(Z_1) \prod_{i=2}^{k} p(Z_i | Z_{i-1}, \ldots, Z_1) \tag{C.73}$$

Unter der Berücksichtigung, dass die bedingte Wahrscheinlichkeit eines Knotens nur von den Wahrscheinlichkeiten seiner direkten Vorgänger $\mathcal{P}(Z_i)$ abhängt, wobei $\mathcal{P}(Z_i)$ die Menge der Vorgänger von Knoten i darstellt, ergibt sich eine vereinfachte Bestimmung der Wahrscheinlichkeit zu:

$$p(Z_1, Z_2, \ldots, Z_k) = \prod_{i=1}^{k} p(Z_i | \mathcal{P}(Z_i)) \tag{C.74}$$

Die Werte von $p(Z_i|\mathcal{P}(Z_i))$ sind in der cpt vom i-ten Knoten gegeben. Durch die vereinfachte Bestimmung der Wahrscheinlichkeit reduziert sich die Größe der zugehörigen cpt und damit ergibt sich eine erhebliche Reduktion in der notwendigen Berechnung. Hat ein Knoten keinen direkten Vorgänger, ist also die zugehörige Zufallsvariable stochastisch unabhängig, wird anstatt einer cpt nur eine a priori Wahrscheinlichkeit angegeben:

$$p(Z_i|\mathcal{P}(Z_i)) = p(Z_i)$$

C.7.1 Inferenz in einem BBN

Unter Inferenz in einem BBN wird das Schließen unter Zuhilfenahme von dem gegebenen Wissen über die Zustände einiger Knoten auf die Wahrscheinlichkeit eines Zustands eines anderen Knotens verstanden. Der gesuchte Knoten ist normalerweise nicht direkt beobachtbar, über ihn liegt deshalb kein direktes Wissen vor. Im Normalfall aber ist für eine Teilmenge der Knoten eines BBN der Wert der zugehörigen Zufallsvariable bekannt oder kann wenigstens mit einer Wahrscheinlichkeit (*Likelihood*) angegeben werden.

Nach Charniak [Cha91] sind die bekannten Inferenzalgorithmen in exakte (*exact*) und approximierte (*approximate*) Inferenz zu unterteilen. Bei der exakten Inferenz wird nach der Richtung des Schließens zwischen *Top-down*-Inferenz und *Bottom-up*-Inferenz unterschieden. *Top-down*- oder kausale Inferenz liegt vor, wenn die Werte der Vorgänger eines Knotens bekannt sind und eine Wahrscheinlichkeit für den Knoten selbst gesucht ist. In dem Sinne verursachen die Werte der Vorgängerknoten den Wert des Ausgangsknotens[16]. Ein solcher Fall ist beispielweise bei der gesuchten Wahrscheinlichkeit $p(C|A)$ in dem BBN aus Abbildung C.18 zu sehen. Wenn man die Knoten der Einfachheit halber als boolsche Variablen auffasst, ergibt sich durch Aufsummieren über die möglichen Zustände aller übrigen Vorgänger:

$$p(C|A) = p(C, B|A) + p(C, \neg B|A)$$

Aus Gleichung C.73 folgt:

$$p(C|A) = \underbrace{p(C|B, A)p(B|A)}_{p(C,B|A)} + \underbrace{p(C|\neg B, A)p(\neg B|A)}_{p(C,\neg B|A)}$$

Mit Hilfe von Gleichung C.74 und der Tatsache, dass A und B keine Vorgänger haben, ergibt sich:

$$p(C|A) = p(C|B, A)p(B) + p(C|\neg B, A)p(\neg B)$$

Alle Wahrscheinlichkeiten auf der rechten Seite sind in den entsprechenden $cpts$ oder als a priori Wahrscheinlichkeit gegeben. Durch Einsetzen in die entsprechenden Terme kann die gesuchte Wahrscheinlichkeit $p(C|A)$ berechnet werden.

Die *Bottom-up*-Inferenz geht davon aus, dass der Zustand eines Knotens bekannt ist und eine Wahrscheinlichkeit für einen Vorgänger gesucht wird. Da hierbei die Auswirkung bekannt ist und der Grund hierfür gesucht wird, spricht man auch von diagnostischer Inferenz. Die gesuchte bedingte Wahrscheinlichkeit muss mit dem Bayes-Theorem umgeformt werden, weil bei der Angabe einer cpt die Kenntnis des Zustandes der Vorgängerknoten vorausgesetzt wird. Das Bayes-Theorem lautet in allgemeiner Form:

$$p(a_i|b) = \frac{p(b|a_i)p(a_i)}{\sum_j p(b|a_j)p(a_j)} \tag{C.75}$$

[16] In dieser Arbeit den Wert der Anwendbarkeit eines Verhaltens.

In dem Beispiel aus Abbildung C.18 ist die Berechnung von $p(A|C)$ ein Fall von diagnostischer Inferenz. Mit dem Bayes-Theorem aus Gleichung C.75 folgt:

$$p(A|C) = \frac{p(C|A)p(A)}{p(C|A)p(A) + p(C|\neg A)p(\neg A)}$$

Die Wahrscheinlichkeiten $p(C|A)$ und $p(C|\neg A)$ können durch eine kausale Inferenz bestimmt werden. Die Wahrscheinlichkeiten $p(A)$ und $p(\neg A) = 1 - p(A)$ sind als a priori-Wahrscheinlichkeiten bekannt. Damit ist die diagnostische Inferenz auf die kausale Inferenz zurückzuführen.

Diese Vorgehensweise lässt sich nur bei Netzen mit einer geringen Knoten- und Zustandsanzahl durchführen. Bei komplexen Graphen würde sie zu einem zu großen, in der Praxis nicht vertretbaren, Rechenaufwand führen. Aus diesem Grund wurde eine Reihe von Inferenzalgorithmen entwickelt, die zuerst das Netz transformieren, um dann eine weniger aufwendige Berechnung durchzuführen. Zu ihnen gehört unter anderem der *Junction-Tree* Algorithmus nach Spiegelhalter [SDLC93]. Bei diesem Algorithmus erfolgt zunächst eine Transformation des gerichteten BBN Graphes in einen ungerichteten. Anschließend wird der Graph in Gruppen, so genannte *Belief Universes*, unterteilt, in Form eines Baumes geordnet und initialisiert. Nach dieser einmaligen Transformation können verschiedene Beobachtungen eingetragen und schnell durch den Baum propagiert werden, um eine Zustandswahrscheinlichkeit für einen gewünschten Knoten auszurechnen. Da ein großer Teil der nötigen Gesamtberechnung nur einmal durchgeführt wird und zwar bei der Initialisierung, sind erhebliche Geschwindigkeitsvorteile erzielbar. Dies setzt jedoch voraus, dass der Graph und die $cpts$ nach der Transformation unverändert bleiben.

Im Gegensatz zu der beschriebenen exakten Inferenz berechnet die approximierte Inferenz Ergebnisse, welche die tatsächliche Wahrscheinlichkeit nur näherungsweise angeben. Da sie in dieser Arbeit nicht eingesetzt wird, wird sie hier nicht weiter behandelt. Eine Übersicht über vorhandene Algorithmen kann bei Charniak [Cha91] gefunden werden.

C.7.2 Lernen eines BBN

Der erste Schritt beim Entwurf eines BBN ist die Festlegung der Struktur des Graphen anhand der beobachteten Abhängigkeiten zwischen den Zufallsvariablen. Anschließend werden die Wahrscheinlichkeiten in den $cpts$ durch Befragen eines Experten besetzt. Darüber hinaus ist es jedoch wünschenswert, das BBN mit Hilfe von Lernalgorithmen anzupassen, d.h. zu trainieren. Dies hat den Vorteil, dass das Expertenwissen nicht vorliegen muss, damit das BBN die gewünschten Zusammenhänge modelliert.

Generell werden Lernverfahren nach zwei Kriterien unterteilt [Hec95]:

Strukturlernen umfasst das Erlernen der Struktur des Graphen, also der Abhängigkeiten zwischen den Zufallsvariablen.

Parameterlernen geht von einer bekannten Graphenstruktur aus und erlernt die Wahrscheinlichkeiten der $cpts$. Beim Parameterlernen wird weiter nach der Beobachtbarkeit der Knoten unterschieden, d.h. ob die Zustände aller Zufallsvariablen bekannt sind oder einige nicht beobachtet werden können.

Im Folgenden wird das Parameterlernen nach Heckerman[17] [Hec95] unter voller Beobachtbarkeit bei bekannter Struktur näher erläutert. Zunächst wird nur ein einzelner Knoten mit r möglichen Zuständen Z_k, $k = 1, ..., r$, betrachtet. Für jeden dieser Zustände existieren unbekannte a priori Wahrscheinlichkeiten $\Theta_Z = \{\theta_{Z=Z_1}, \ldots, \theta_{Z=Z_r}\}$, die angeben, mit welcher Wahrscheinlichkeit der jeweilige Zustand Z angenommen wird. Wenn Θ_Z bekannt wäre, könnte die Wahrscheinlichkeit dafür, dass ein bestimmter Zustand $Z = Z_k$ eintritt, exakt bestimmt werden und ein Lernen wäre unnötig:

$$p(Z = Z_k|\Theta_Z) = \theta_{Z=Z_k} \tag{C.76}$$

Da die Verteilungen Θ_Z aber unbekannt sind, kann nur eine Wahrscheinlichkeitsverteilung $p(\Theta_Z)$ angegeben werden. Somit beträgt die Wahrscheinlichkeit, dass sich der Knoten im Zustand $Z = Z_k$ befindet:

$$p(Z = Z_k) = \int p(Z = Z_k|\Theta_Z)\, p(\Theta_Z)\, d\Theta_Z = \int \theta_{Z=Z_k}\, p(\Theta_Z)\, d\Theta_Z \equiv E(\theta_{Z=Z_k}) \tag{C.77}$$

Die Wahrscheinlichkeit, dass der Zustand Z_k beobachtet wird, entspricht also dem Erwartungswert $E(\theta_{Z=Z_k})$, der in der *cpt* eingetragen ist und beim Lernen angepasst wird. Sollen die Wahrscheinlichkeitsverteilungen Θ_Z nach Beobachtung eines Falles $Z = Z_k$ angepasst werden, ist es notwendig, die a posteriori Wahrscheinlichkeit $p(\Theta_Z|Z = Z_k)$ mit Hilfe des Bayes-Theorem (Gleichung C.75) und der Gleichung C.76 zu berechnen:

$$p(\Theta_Z|Z = Z_k) = c\, p(Z = Z_k|\Theta_Z)\, p(\Theta_Z) = c\, \theta_{Z=Z_k}\, p(\Theta_x) \tag{C.78}$$

Der Parameter c ist hierbei eine Normalisierungskonstante.

Sei $\mathcal{D} = \{Z_1, \ldots, Z_m\}$ eine Datenmenge von m beobachteten Fällen, wobei jedes Z_i einen der r möglichen Zustände darstellt. Falls die Datenmenge $\mathcal{D}$ beobachtet wird, ergibt sich analog zu Gleichung C.78:

$$p(\Theta_Z|D) = c \prod_{k=1}^{r} \theta_{Z=Z_k}^{N_k} p(\Theta_Z) \tag{C.79}$$

Dabei ist N_k die Anzahl von Fällen, bei denen der Zustand Z_k aufgetreten ist. Hiermit ist allgemein eine Formel zur Anpassung der a priori Wahrscheinlichkeit für einen Knoten in einem BBN nach Beobachtung von m Fällen hergeleitet. Für Θ_x kann eine beliebige Wahrscheinlichkeitsverteilung angenommen werden, in der Praxis wird jedoch häufig die Dirichletverteilung gewählt:

$$p(\Theta_Z) = \frac{\Gamma(\sum_{k=1}^{r} N_k')}{\sum_{k=1}^{r} \Gamma(N_k')} \prod_{k=1}^{r} \theta_{Z=Z_k}^{N_k'-1} \tag{C.80}$$

Dabei stellt Γ die Gammafunktion dar und die N_k' beschreiben die Exponenten der Dirichletfunktion.

Nach dem Erlernen der Datenmenge $\mathcal{D}$ ergibt sich für die a posteriori Wahrscheinlichkeit nach den Gleichungen C.79 und C.80 mit

$$p(\Theta_Z|\mathcal{D}) = c' \prod_{k=1}^{r} \theta_{Z=Z_k}^{N_k'+N_k-1} \tag{C.81}$$

[17] Das Parameterlernen nach Heckerman [Hec95] wird auch in dieser Arbeit zur Anpassung der BBNS der Vehaltenskoordination verwendet.

wiederum eine Dirichletverteilung mit angepassten Exponenten $N_k^{'} + N_k$ und einer neuen Normalisierungskonstante c'. Damit beschreiben diese Exponenten den Wissensstand eines Knotens, der angibt, wie oft welcher Fall gelernt wurde. Die Wahrscheinlichkeit dafür, dass im nächsten Fall der Zustand $Z = Z_k$ beobachtet wird, ergibt sich durch Berechnung des Erwartungswertes der Dirichletverteilung zu:

$$p(Z = Z_k) = E(\theta_{Z=Z_k}) = \frac{N_k^{'}}{\sum_{k=1}^{r} N_k^{'}} \tag{C.82}$$

Das entspricht dem aktualisierten Eintrag in der cpt.

Weiterhin ist die Varianz der Dirichletverteilung gegeben durch:

$$Var(\theta_{Z=Z_k}) = \frac{p(Z = Z_k)\,(1 - p(Z = Z_k))}{\sum_{k=1}^{r} N_k^{'} + 1} \tag{C.83}$$

Ein Maß für das Vertrauen in Θ_x ist die Gesamtanzahl der gelernten Fälle $\sum_{k=1}^{r} N_k^{'}$. Je mehr Fälle gelernt werden, desto geringer wird die Varianz.

Diese Art des Lernens eines einzelnen Knotens lässt sich auf ein gesamtes BBN übertragen. Hierzu ist jedoch die Annahme der so genannten Parameterunabhängigkeit (*Parameter Independence*) notwendig [Hec95]. Für den Fall, dass zwei Knoten X und Y existieren und Y von X abhängt, erhält man drei Wahrscheinlichkeitsverteilungen, θ_X, $\theta_{Y|X}$ und $\theta_{Y|\neg X}$. Allgemein müssten sich auch die Verteilungen $\theta_{Y|X}$ und $\theta_{Y|\neg X}$ ändern, wenn die Verteilung θ_X angepasst wird. Unter Berücksichtigung der Annahme der Parameterunabhängigkeit entfällt die beschriebene Abhängigkeit und die $cpts$ der einzelnen Knoten können separat voneinander, nacheinander angepasst werden. Die Unabhängigkeitsannahme betrifft nicht die Abhängigkeit der Zufallsvariable Y von X.

Jeder Knoten des Netzwerks besitzt für jede mögliche Konfiguration j der Zustände der Elternknoten eine entsprechende a priori Wahrscheinlichkeitsverteilung Θ_{ij}. Dann ist nach Erlernen der Datenmenge $\mathcal{D}$ unter der Voraussetzung einer Dirichletverteilung:

$$p(\Theta_{ij}|D) = c' \prod_k \theta_{ijk}^{N_{ijk}{}' + N_{ijk} - 1} \tag{C.84}$$

N_{ijk} ist die Anzahl aller Fälle in $\mathcal{D}$ mit $Z_i = Z_k$ und der Konfiguration j der Zustände der Elternknoten. Pro beobachteten Fall wird immer nur die entsprechende Wahrscheinlichkeitsverteilung aktualisiert, während alle anderen unverändert bleiben.

Anhang D

Implementierungsdaten der Hindernisvermeidung des Greifers

In diesem Kapitel werden sowohl die Zugehörigkeitsfunktionen als auch die Regelbasis des Bewerters und der Steuerungskomponente der Hindernisvermeidung des Greifers vorgestellt.

D.1 Bewerter der Hindernisvermeidung des Greifers

Beim Bewerter werden die Zugehörigkeitsfunktionen der linguistischen Eingangsvariablen mit Gaußfunktionen realisiert:

$$\mu(x) = e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{D.1}$$

wobei m den Mittelwert und σ die Varianz der Gaußfunktionen sind.

Für den minimalen Abstand des Objektes von der Kamera d_{HKmin} (Abbildung 4.18) sind drei linguistische Terme definiert; die entsprechenden Zugehörigkeitsfunktionen sind in Abbildung D.1 dargestellt.

sehr nah mit $m = 32$ cm und $\sigma = 8$.
nah mit $m = 52$ cm und $\sigma = 8$.
weit mit $m = 72$ cm und $\sigma = 8$.

Für den minimalen Abstand des Objektes von der Sichtachse der Kamera d_{HSAmin} sind drei linguistische Terme definiert (siehe auch Abbildung D.2):

sehr nah mit $m = 0$ cm und $\sigma = 8$.
nah mit $m = 12.5$ cm und $\sigma = 8$.
weit mit $m = 25$ cm und $\sigma = 8$.

Die Ausgabe, also die Bewertung der neuen Situation, ist mit drei linguistischen Termen definiert, die als Singletons realisiert sind:

schlecht mit $m = -1.0$.

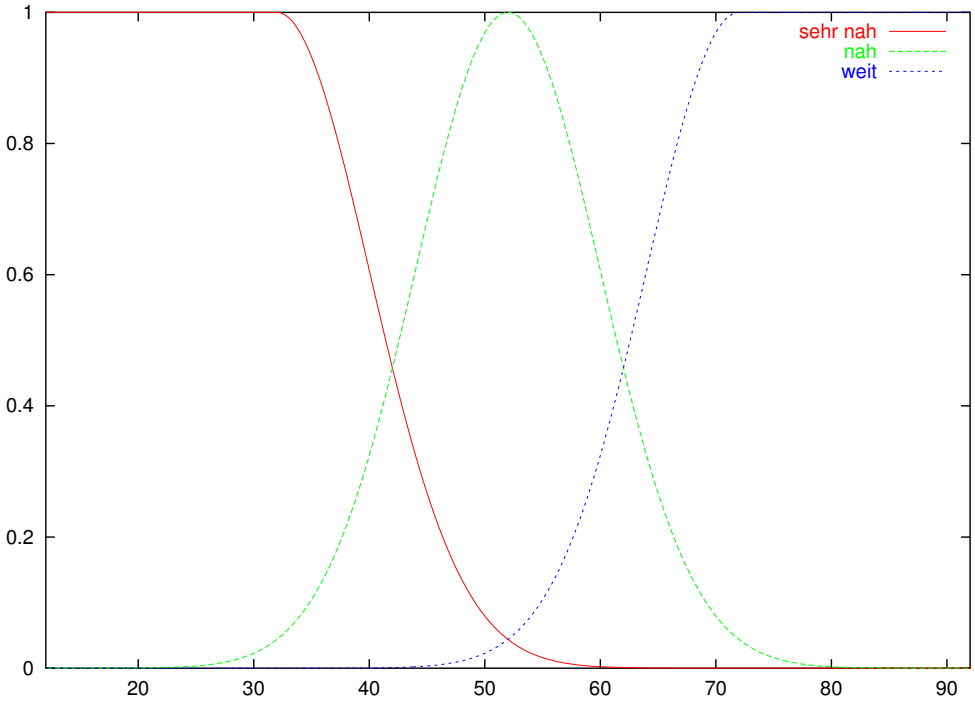


Abbildung D.1: Zugehörigkeitsfunktionen für d_{HKmin}.

mittel mit $m = -0.5$.
gut mit $m = 0$.

Die Regelbasis des Bewerters ist in Tabelle D.1 dargestellt.

Regel	**Abstand Hindernis-Kamera** $\mathbf{d_{HKmin}}$	**Abstand Hindernis-Sichtachse** $\mathbf{d_{HSAmin}}$	**Bewertung** $\mathbf{r_{HGr}}$
1	sehr nah	sehr nah	schlecht
2	sehr nah	nah	schlecht
3	sehr nah	weit	mittel
4	nah	sehr nah	schlecht
5	nah	nah	mittel
6	nah	weit	gut
7	weit	sehr nah	mittel
8	weit	nah	gut
9	weit	weit	gut

Tabelle D.1: Regelbasis des Bewerters.

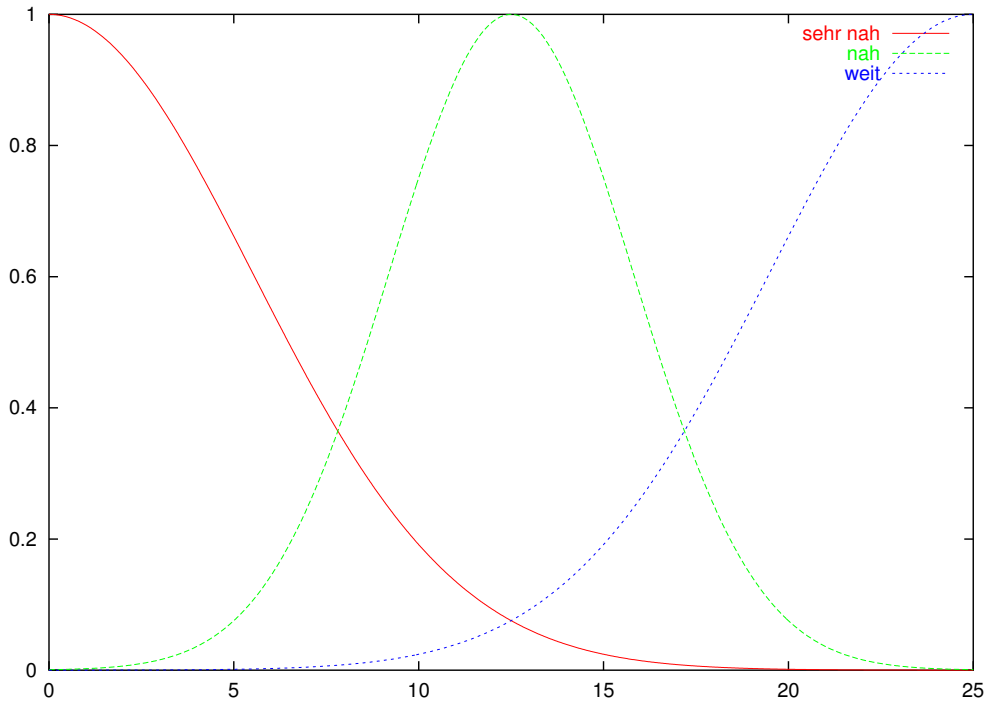


Abbildung D.2: Zugehörigkeitsfunktionen für d_{HSAmin}.

D.2 Steuerungskomponente der Hindernisvermeidung des Greifers

Die Zugehörigkeitsfunktionen der linguistischen Eingangsvariablen bei der Steuerungskomponente werden mit Gaußfunktionen realisiert:

$$\mu(x) = e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{D.2}$$

wobei m der Mittelwert und σ die Varianz der Gaußfunktionen sind.

Für den Abstand des Objektschwerpunktes von der Kamera d_{HK} (Abbildung 4.12a) sind vier linguistische Terme definiert; die entsprechenden Zugehörigkeitsfunktionen sind in Abbildung D.3 dargestellt.

sehr nah mit $m = 12.0$ cm und $\sigma = 13.3$.
nah mit $m = 36.6$ cm und $\sigma = 13.3$.
nicht sehr weit mit $m = 65.3$ cm und $\sigma = 13.3$.
weit mit $m = 92.0$ cm und $\sigma = 13.3$.

Für den Abstand des Objektschwerpunktes von der Sichtachse d_{HSA} (Abbildung 4.12a) sind auch vier linguistische Terme definiert; die entsprechenden Zugehörigkeitsfunktionen sind in Abbildung D.4 dargestellt.

sehr nah mit $m = 0.0$ cm und $\sigma = 3.3$.

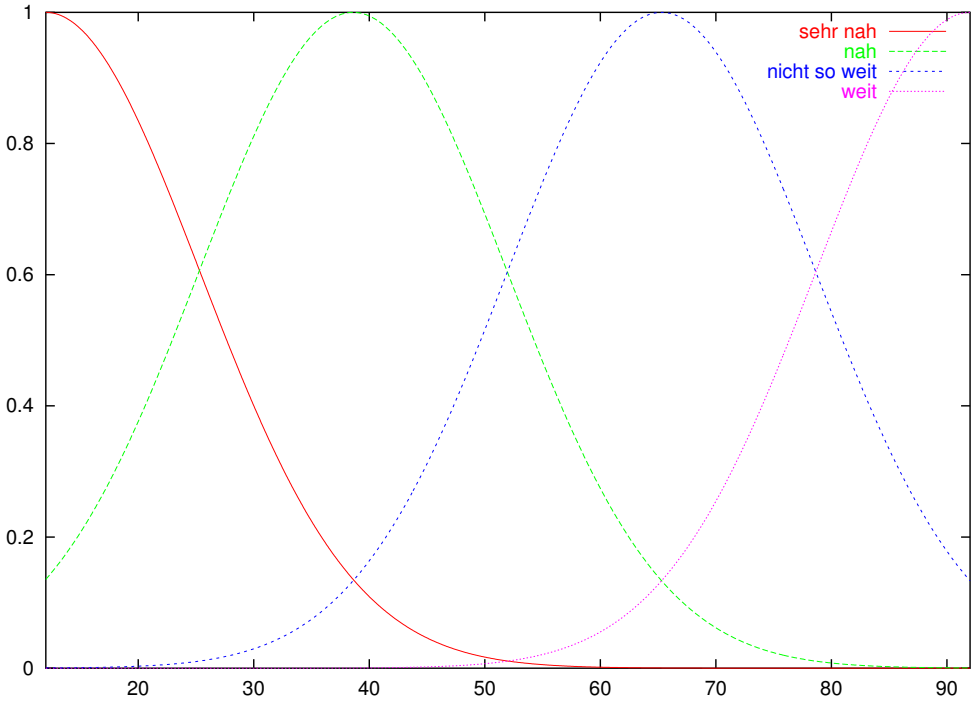


Abbildung D.3: Zugehörigkeitsfunktionen für d_{HK}.

nah mit $m = 8.3$ cm und $\sigma = 3.3$.
nicht sehr weit mit $m = 16.6$ cm und $\sigma = 3.3$.
weit mit $m = 25.0$ cm und $\sigma = 3.3$.

Die horizontale Größe d_{HH_i} des umgebenden Parallelogramms des Objektes im Bild (Abbildung 4.12b) wird mit drei linguistischen Termen angenähert (Abbildung D.5):

klein mit $m = 0.0$ Pixel und $\sigma = 60$.
mittel mit $m = 150$ Pixel und $\sigma = 60$.
groß mit $m = 300$ Pixel und $\sigma = 60$.

Dasselbe gilt für die vertikale Größe d_{HV_i} des umgebenden Parallelogramms (Abbildung 4.12b), dessen Zugehörigkeitsfunktionen in Abbildung D.6 dargestellt sind.

klein mit $m = 0.0$ Pixel und $\sigma = 60$.
mittel mit $m = 150$ Pixel und $\sigma = 60$.
groß mit $m = 300$ Pixel und $\sigma = 60$.

Die Anzahl der unscharfen Terme der linguistische Ausgangsvariablen[1] ist gleich der Anzahl der Regeln der Regelbasis. Die Werte der linguistischen Terme werden während der Trainingsphase im Bereich $[2, 10]$ für den Betrag bzw. $[0^0, 135^0]$ für die Elevation angepasst. Bei der Initialisierung erhalten sie den optimalen Wert, wenn keinen Hindernisse vorhanden sind: den größten möglichen Betrag für den Bewegungsvektor, also 10, und einen sehr kleinen Winkel für die Elevation, also 0.1^0. Die nach dem Training resultierende Regelbasis ist in Tabelle D.2 dargestellt.

[1]Die Ausgangsvariablen des Neurofuzzy Systems sind hier der Betrag $|\vec{\rho}|$ und die Elevation φ des Ausweichvektors.

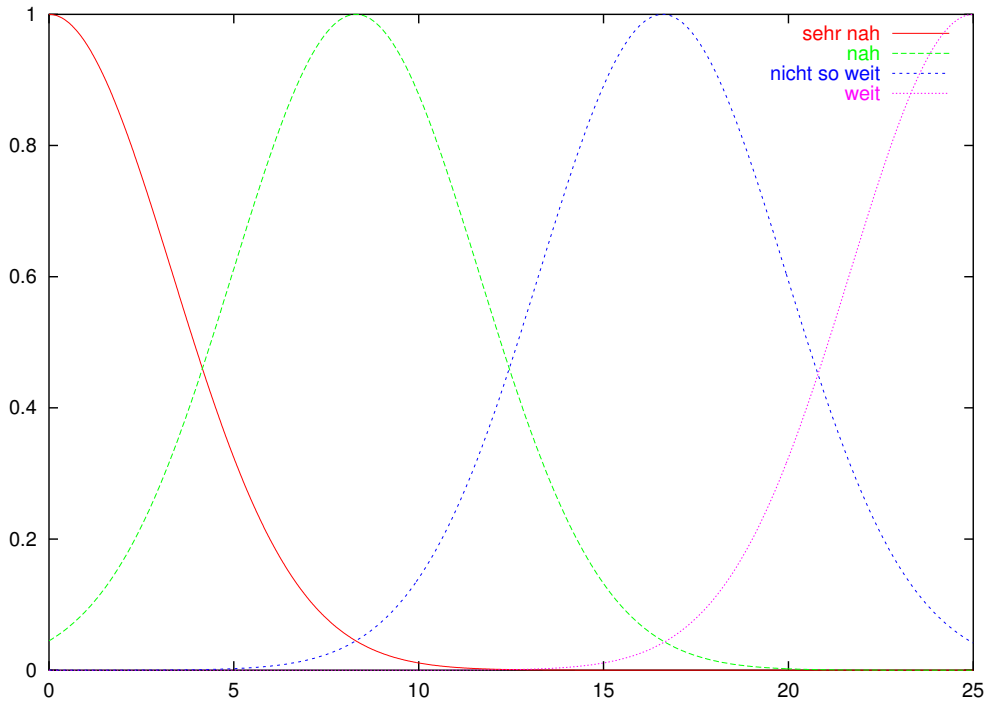


Abbildung D.4: Zugehörigkeitsfunktionen für d_{HSA}.

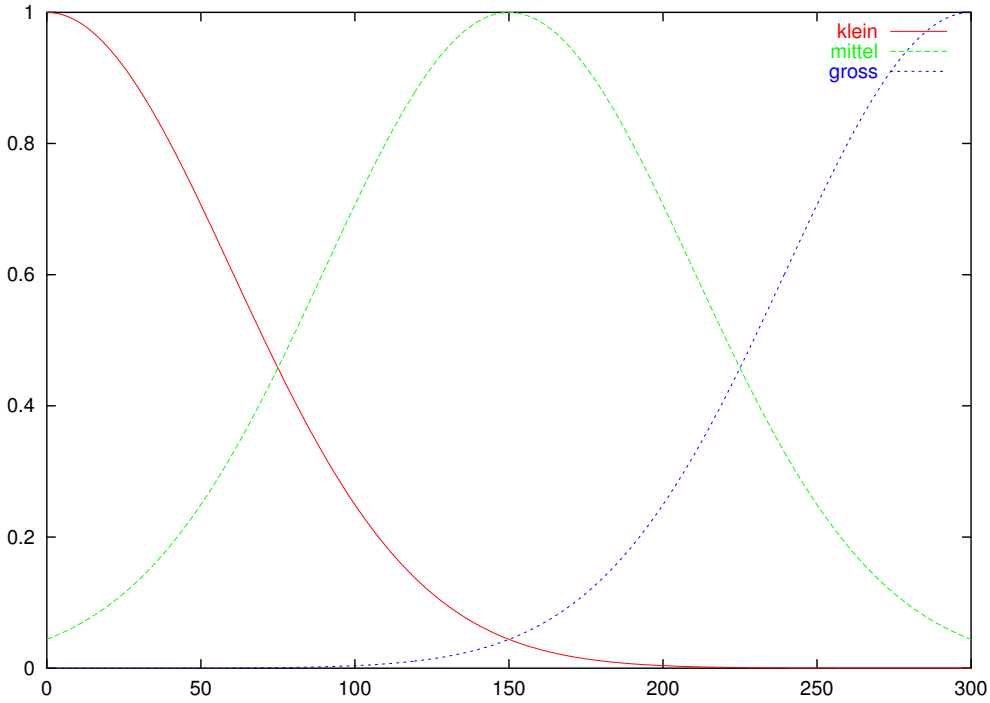


Abbildung D.5: Zugehörigkeitsfunktionen für d_{HH}.

Regel	$\mathbf{d_H K}$	$\mathbf{d_{HSA}}$	$\mathbf{d_H H}$	$\mathbf{d_H V}$	$\mid \vec{\rho} \mid$ **(cm)**	φ **(Grad)**
0	SN	SN	K	K	1.99976	27.2031
1	SN	SN	K	M	1.99999	45.8364
2	SN	SN	K	G	1.99998	34.4723
3	SN	SN	M	K	1.99999	52.4494
4	SN	SN	M	M	1.99988	89.8328
5	SN	SN	M	G	1.99987	96.6567
6	SN	SN	G	K	4.07929	23.7997
7	SN	SN	G	M	1.99442	135.045
8	SN	SN	G	G	1.99001	135.104
9	SN	N	K	K	6.87944	12.5759
10	SN	N	K	M	5.61309	17.6313
11	SN	N	K	G	6.13912	15.6113
12	SN	N	M	K	5.71593	17.2637
13	SN	N	M	M	1.9998	59.3647
14	SN	N	M	G	1.99941	69.3042
15	SN	N	G	K	6.76194	13.0839
16	SN	N	G	M	1.99978	68.4776
17	SN	N	G	G	1.99967	102.49
18	SN	NSW	K	K	9.15693	3.47256

fortgesetzt in der nächsten Seite

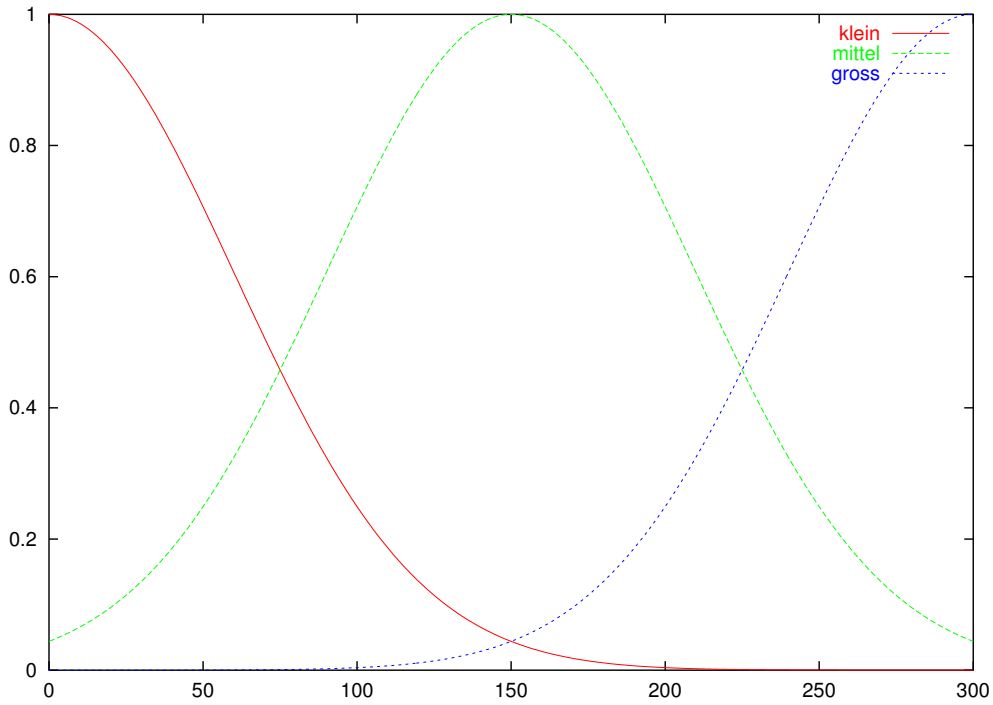


Abbildung D.6: Zugehörigkeitsfunktionen für d_{HV}.

Regel	$\mathbf{d_H K}$	$\mathbf{d_{HSA}}$	$\mathbf{d_H H}$	$\mathbf{d_H V}$	$\mid \vec{\rho} \mid$ **(cm)**	φ **(Grad)**
19	SN	NSW	K	M	9.81248	0.757248
20	SN	NSW	K	G	9.95968	0.263711
21	SN	NSW	M	K	9.8487	0.618188
22	SN	NSW	M	M	9.03502	3.88222
23	SN	NSW	M	G	5.88855	16.5486
24	SN	NSW	G	K	9.21072	3.2591
25	SN	NSW	G	M	8.33263	6.77202
26	SN	NSW	G	G	7.34781	10.7136
27	SN	W	K	K	9.16388	3.44351
28	SN	W	K	M	9.85483	0.585786
29	SN	W	K	G	9.99891	0.104246
30	SN	W	M	K	9.86055	0.568556
31	SN	W	M	M	9.73633	1.08067
32	SN	W	M	G	9.816	0.836074
33	SN	W	G	K	9.99245	0.130434
34	SN	W	G	M	9.74104	1.13646
35	SN	W	G	G	9.73341	1.16654
36	N	SN	K	K	2.22771	31.1723
37	N	SN	K	M	1.99999	33.7307
38	N	SN	K	G	8.90007	4.64303

fortgesetzt in der nächsten Seite

Regel	$\mathbf{d_H K}$	$\mathbf{d_{HSA}}$	$\mathbf{d_H H}$	$\mathbf{d_H V}$	$\mid \vec{\rho} \mid$ **(cm)**	φ **(Grad)**
39	N	SN	M	K	3.51832	25.941
40	N	SN	M	M	1.93642	66.3905
41	N	SN	M	G	3.26307	27.0247
42	N	SN	G	K	8.10571	7.85409
43	N	SN	G	M	1.99814	130.226
44	N	SN	G	G	1.98149	135.049
45	N	N	K	K	6.08214	15.7836
46	N	N	K	M	3.65935	25.5225
47	N	N	K	G	8.165	7.58141
48	N	N	M	K	5.94018	16.4089
49	N	N	M	M	1.98067	83.3376
50	N	N	M	G	1.99994	41.6235
51	N	N	G	K	8.29716	7.08831
52	N	N	G	M	1.99242	81.311
53	N	N	G	G	1.99768	91.093
54	N	NSW	K	K	9.10262	3.68997
55	N	NSW	K	M	9.75404	0.991291
56	N	NSW	K	G	9.95055	0.300201
57	N	NSW	M	K	9.73835	1.059
58	N	NSW	M	M	6.36823	14.6263
59	N	NSW	M	G	1.99762	35.9198
60	N	NSW	G	K	9.82932	0.784934
61	N	NSW	G	M	1.92697	65.948
62	N	NSW	G	G	2.20472	31.2824
63	N	W	K	K	9.16328	3.44591
64	N	W	K	M	9.85419	0.588375
65	N	W	K	G	9.99827	0.106804
66	N	W	M	K	9.85979	0.571659
67	N	W	M	M	9.39096	2.45676
68	N	W	M	G	9.52311	2.00759
69	N	W	G	K	9.99169	0.133528
70	N	W	G	M	2.71877	29.225
71	N	W	G	G	8.07583	7.79681
72	NSW	SN	K	K	8.05814	7.86934
73	NSW	SN	K	M	3.37335	26.6135
74	NSW	SN	K	G	9.74564	1.11778
75	NSW	SN	M	K	8.66245	5.36001
76	NSW	SN	M	M	1.99746	64.431
77	NSW	SN	M	G	9.68845	1.34742
78	NSW	SN	G	K	9.89256	0.531078
79	NSW	SN	G	M	9.55274	1.89158
80	NSW	SN	G	G	9.92811	0.389414

fortgesetzt in der nächsten Seite

Regel	$\mathbf{d_H K}$	$\mathbf{d_{HSA}}$	$\mathbf{d_H H}$	$\mathbf{d_H V}$	$\mid \vec{\rho} \mid$ **(cm)**	φ **(Grad)**
81	NSW	N	K	K	8.03206	7.97375
82	NSW	N	K	M	6.83157	12.7667
83	NSW	N	K	G	9.6797	1.3816
84	NSW	N	M	K	8.57103	5.72579
85	NSW	N	M	M	1.83295	106.166
86	NSW	N	M	G	9.57308	1.8099
87	NSW	N	G	K	9.82226	0.812503
88	NSW	N	G	M	1.85999	44.1664
89	NSW	N	G	G	9.84662	0.716438
90	NSW	NSW	K	K	9.09597	3.71545
91	NSW	NSW	K	M	9.78302	0.874001
92	NSW	NSW	K	G	9.98884	0.144969
93	NSW	NSW	M	K	9.72476	1.11145
94	NSW	NSW	M	M	2.98696	28.1527
95	NSW	NSW	M	G	9.65895	1.4656
96	NSW	NSW	G	K	9.92312	0.40804
97	NSW	NSW	G	M	6.52006	14.0203
98	NSW	NSW	G	G	9.48023	2.18052
99	NSW	W	K	K	9.16328	3.44591
100	NSW	W	K	M	9.85512	0.584908
101	NSW	W	K	G	9.99921	0.103337
102	NSW	W	M	K	9.85981	0.571491
103	NSW	W	M	M	9.80264	0.81593
104	NSW	W	M	G	9.89564	0.518328
105	NSW	W	G	K	9.99896	0.104368
106	NSW	W	G	M	9.60468	1.6821
107	NSW	W	G	G	9.70356	1.28664
108	W	SN	K	K	9.08103	3.77516
109	W	SN	K	M	9.77275	0.915059
110	W	SN	K	G	9.99683	0.112822
111	W	SN	M	K	9.73086	1.08671
112	W	SN	M	M	9.70696	1.19797
113	W	SN	M	G	9.99506	0.119968
114	W	SN	G	K	9.95063	0.297541
115	W	SN	G	M	9.94493	0.320342
116	W	SN	G	G	9.99557	0.117832
117	W	N	K	K	9.08474	3.76022
118	W	N	K	M	9.77679	0.898788
119	W	N	K	G	9.99562	0.11767
120	W	N	M	K	9.72892	1.09433
121	W	N	M	M	9.77063	0.943078
122	W	N	M	G	9.99264	0.129671

fortgesetzt in der nächsten Seite

Regel	$\mathbf{d_H K}$	$\mathbf{d_{HSA}}$	$\mathbf{d_H H}$	$\mathbf{d_H V}$	$\mid \vec{\rho} \mid$ **(cm)**	φ **(Grad)**
123	W	N	G	K	9.94353	0.325909
124	W	N	G	M	9.939	0.344002
125	W	N	G	G	9.99319	0.127268
126	W	NSW	K	K	10	0.1
127	W	NSW	K	M	10	0.1
128	W	NSW	K	G	9.99777	0.109007
129	W	NSW	M	K	10	0.1
130	W	NSW	M	M	9.89085	0.461194
131	W	NSW	M	G	9.99501	0.120071
132	W	NSW	G	K	9.98859	0.14557
133	W	NSW	G	M	9.98856	0.145697
134	W	NSW	G	G	9.9952	0.119181
135	W	W	K	K	10	0.1
136	W	W	K	M	10	0.1
137	W	W	K	G	9.99965	0.10151
138	W	W	M	K	10	0.1
139	W	W	M	M	9.90559	0.402384
140	W	W	M	G	9.99949	0.102161
141	W	W	G	K	9.99963	0.10148
142	W	W	G	M	9.9996	0.101599
143	W	W	G	G	9.9996	0.101599

Tabelle D.2: Erlernte Regelbasis der Steuerungskomponente der Hindernisvermeidung des Greifers (SN = sehr nah, N = nah, NSW = nicht sehr weit, W = weit, G = groß, M = mittel, K = klein).

Anhang E

FSAs und BBNs der vermittelnden Ebene

E.1 Die endliche Zustandsakzeptoren der Planungsschritte

In Tabelle E.1 sind alle implementierten Planungsschritte aufgelistet, die die vermittelnde Ebene ausführen kann. Die entsprechenden endliche Zustandsakzeptoren (FSA) sind in Abbildungen E.1 bis E.5 enthalten, während Tabelle E.2 die in den jeweiligen Zuständen der FSAs aktiven Verhalten enthält.

Planungsschritt	**Beschreibung**
`Detektiere_Objekt`	Suche mit der Bordkamera nach dem Objekt in einer Region von $[-60^0, 60^0]$ um den Roboter.
`Lokalisiere_Objekt`	Bestimme mit der Bordkamera die kartesische Lage des Objektes im Raum relativ zur Manipulator-Basis.
`Greife_Objekt_1`	Greife Zielobjekt mit den Hindernisvermeidungsverhalten des Greifers und der Segmente, der Zielführung und dem Planungsverhalten aktiviert.
`Greife_Objekt_2`	Greife Zielobjekt ohne die Hindernisvermeidung der Segmente zu aktivieren.
`Greife_Objekt_3`	Greife Objekt mit aktiviertem Planungsverhalten und Zielführung.
`Greife_Objekt_4`	Greife Zielobjekt nur mit aktiviertem Planungsverhalten.
`Greife_Objekt_5`	Greife Zielobjekt nur mit der Zielführung.
`Erreiche_Objekt_1`	Führe den Greifer von der aktuellen Position zur Greifposition am Objekt mit den Hindernisvermeidungsverhalten des Greifers und der Segmente, der Zielführung und dem Planungsverhalten aktiviert.
`Erreiche_Objekt_2`	Führe den Greifer von der aktuellen Position zum Ziel am Objekt ohne das Hindernisvermeidungsverhalten für die Segmente zu aktivieren.
`Erreiche_Objekt_3`	Führe den Greifer von der aktuellen Position zum Ziel am Objekt mit der Zielführung und dem Planungsverhalten aktiviert.
`Erreiche_Objekt_4`	Führe den Greifer nur mit dem Planungsverhalten zum Ziel.
`Erreiche_Objekt_5`	Führe den Greifer nur mit der Zielführung zum Ziel.
fortgesetzt in der nächsten Seite	

Planungsschritt	**Beschreibung**
`Platziere_Objekt_1`	Platziere Objekt an einer erwünschten Position.
`Platziere_Objekt_2`	Bei aktivierter Hindernisvermeidung der Segmente platziere Objekt an einer erwünschten Position.
`Schließe_Greifer`	Schließe den Greifer.
`Öffne_Greifer`	Öffne den Greifer.
`Bewege_Roboterarm`	Bewege den Roboter zu einer Position `Pos`.
`Rückführung`	Bewege den Roboterarm zurück zu einer sicheren Stellung entlang des schon ausgeführten Pfades.

Tabelle E.1: Implementierte Planungsschritte, die die vermittelnde Ebene ausführen kann.

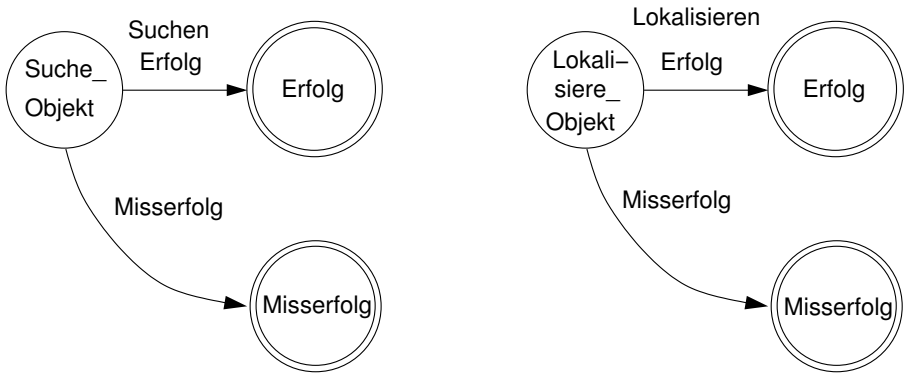


Abbildung E.1: Entsprechende FSAs für die Planungsschritte `Detektiere_Objekt` (a) und `Lokalisiere_Objekt` (b).

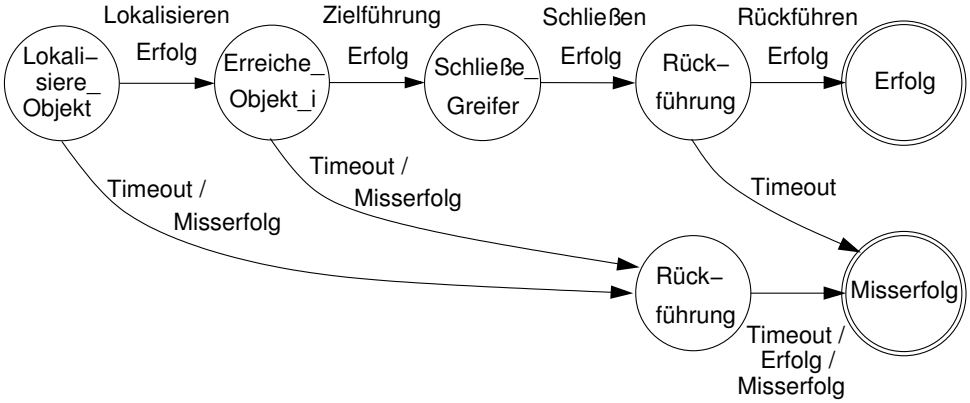


Abbildung E.2: FSA für die fünf Varianten des Planungsschrittes `Greife_Objekt_i` mit `i`=1,..,5.

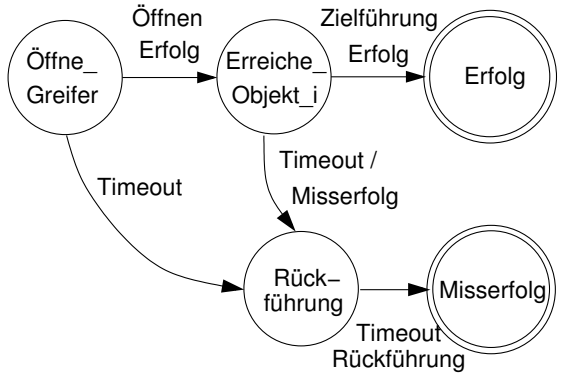
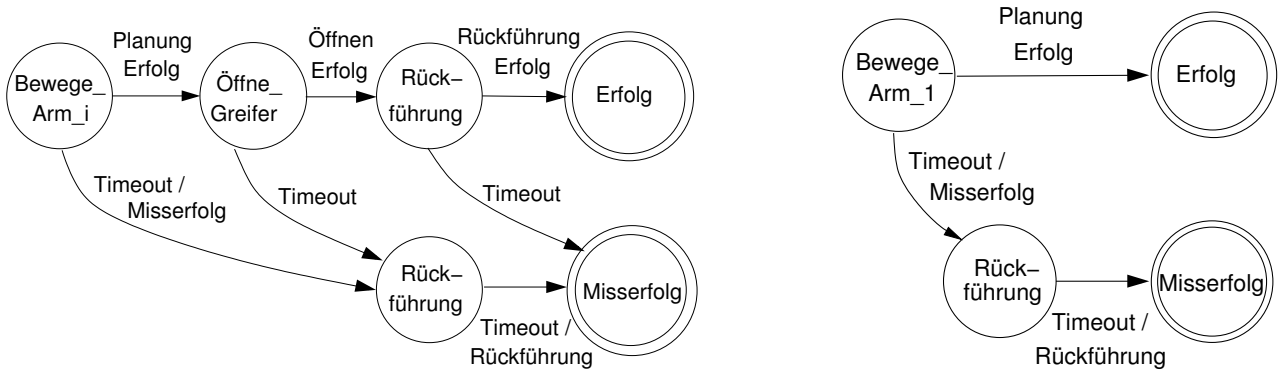


Abbildung E.3: FSA für die fünf Varianten des Planungsschrittes `Erreiche_Objekt_i` mit `i`=1,..,5.



(a) (b)

Abbildung E.4: FSAs für die zwei Varianten des Planungsschrittes `Platziere_Objekt_i` mit `i`=1,2 (a) und für Planungsschritt `Bewege_Roboterarm` (b).

E.2 Realisierte Bayesian Belief Networks

E.2.1 Verwendete Merkmale

Die Merkmale, die als Eingabe für die *Bayesian Belief Networks* (BBNs) verwendet werden, stammen aus den Aufnahmen der Bord- und Greiferkameras (s.a. Kapitel 4). Folgende Merkmale kommen in der vermittelnden Ebene zum Einsatz:

d_{GZ}: Die Distanz zwischen der aktuellen Greiferposition und dem Schwerpunkt des Zielobjektes.

d_{GH}: Die Distanz zwischen Greifer und dem nächsten Punkt auf der Oberfläche des nächsten Hindernisses. Hierzu wird eine Strecke zwischen den beiden Fingern des Greifers bestimmt und anschließend die minimale Distanz dieser Strecke zu allen Hindernissen berechnet (vergleiche Gleichung 4.21).

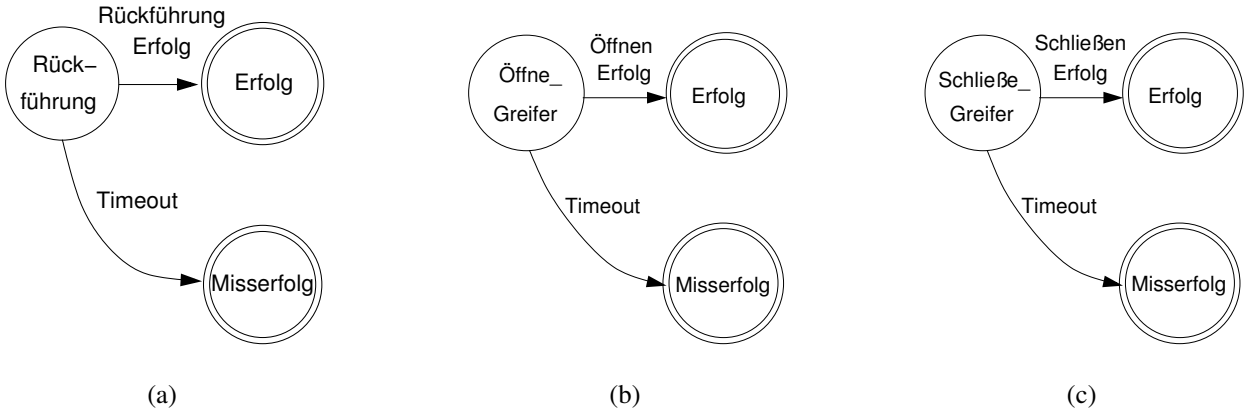


Abbildung E.5: FSAs für die Planungsschritte `Rückführung` (a), `Öffne_Greifer` (b) und `Schließe_Greifer` (c).

$d_{S_{23}H}$: Die Distanz zwischen Oberarm[1] und dem nächsten Punkt auf der Oberfläche des nächsten Hindernisses. Die Distanz wird auch hier nach Gleichung 4.21 ermittelt.

$d_{S_{35}H}$: Die Distanz zwischen Unterarm[2] und dem nächsten Punkt auf der Oberfläche des nächsten Hindernisses.

d_{GT}: Die Distanz zwischen Greifer und Auflageebene bzw. Tisch.

$d_{S_{23}T}$: Die Distanz zwischen Oberarm und Tisch.

$d_{S_{35}T}$: Die Distanz zwischen Unterarm und Tisch.

$\dot{d}_{GH}$: Die Änderung der Distanz zwischen Greifer und dem nächsten Hindernis.

ϕ_{GH}: Der Winkel zwischen aktueller Bewegungsrichtung des Greifers und der Richtung, in der sich das nächste Hindernis befindet. Es wird aus der Definition des Skalarproduktes zwischen dem aktuellen Bewegungsvektor und dem Positionsvektor des Hindernisses relativ zum Greifer berechnet.

ϕ_{S_3H}: Der Winkel zwischen aktueller Bewegungsrichtung von Gelenk 3 und der Richtung, in der sich das nächste Hindernis befindet.

ϕ_{S_5H}: Der Winkel zwischen aktueller Bewegungsrichtung von Gelenk 5 und der Richtung, in der sich das nächste Hindernis befindet.

ϕ_{GT}: Der Winkel zwischen aktueller Bewegungsrichtung des Greifers und der Richtung, in der sich die Auflageebene befindet.

ϕ_{S_3T}: Der Winkel zwischen aktueller Bewegungsrichtung von Gelenk 3 und der Richtung, in der sich der Tisch befindet.

ϕ_{S_5T}: Der Winkel zwischen aktueller Bewegungsrichtung von Gelenk 5 und der Richtung, in der sich der Tisch befindet.

ϕ_{ZK}: Die Abweichung des Zielobjektes von der Kameraachse. Dieser Winkel ist ein Maß dafür, wie zentral sich das Ziel im Sichtfeld der Kamera befindet.

ϕ_{ZP}: Der Winkel zwischen Bewegungsvektor des zielführenden Verhaltens und des Planungsverhaltens.

$sicht_Z$: Die Sichtbarkeit des Zielobjektes. Auch wenn sich das Ziel im Zentrum des Kamerabildes befindet, kann es durch ein Hindernis teilweise verdeckt sein. Aus diesem Grund wird der

[1] Roboterarm-Segment zwischen Gelenke 2 und 3.

[2] Der Unterarm besteht aus den Segmenten zwischen Gelenke 3, 4 und 5.

Zustand	Aktive Verhalten
`Suche_Objekt`	Objektsuche mit der Bordkamera
`Lokalisiere_Objekt`	3D-Objektlokalisierung mit der Bordkamera
`Erreiche_Objekt_1`	Zielführung Kollisionsvermeidung Greifer Kollisionsvermeidung Segmente Planungsverhalten Zielfolgen
`Erreiche_Objekt_2`	Zielführung Kollisionsvermeidung Greifer Planungsverhalten Zielfolgen
`Erreiche_Objekt_3`	Zielführung Planungsverhalten Zielfolgen
`Erreiche_Objekt_4`	Planungsverhalten
`Erreiche_Objekt_5`	Zielführung
`Bewege_Arm_1`	Planungsverhalten
`Bewege_Arm_2`	Planungsverhalten Hindernisvermeidung der Segmente
`Rückführung`	Zurückführendes Verhalten
`Schließe_Greifer`	Greifer schließen
`Öffne_Greifer`	Greifer öffnen

Tabelle E.2: Aktive Verhalten für die Zustände der FSAs der implementierten Planungsschritte.

kleinste Winkel zwischen dem Vektor, der vom Greifer aus in Richtung des Zieles zeigt, und dem Vektor, der vom Greifer aus in Richtung eines Hindernisses zeigt, bestimmt. Je größer der Winkel, desto höher die Wahrscheinlichkeit, das Objekt vollständig zu sehen.

t: Ein Zähler, der die Zeit ab Beginn des Greifvorgangs misst.

Jedes BBN erhält eine entsprechende Untermenge dieser Merkmale, die in einem Merkmalsvektor $\vec{s}_i'$ zusammengefasst ist. Im Allgemeinen verwenden Verhalten und entsprechende BBNs nicht dieselben Merkmale, also $\vec{s}_i \neq \vec{s}_i'$, da sie unterschiedliche Aufgaben zu bewältigen haben: ein Verhalten muss den Aktuator regeln bzw. führen, während das BBN entscheiden muss, zu welchem Anteil ein Verhaltens bei der Gesamtbewegung beitragen soll.

E.2.2 Topologie der BBNs der weiteren Verhalten

In den folgenden Abbildungen (E.6 bis E.9) werden die implementierten BBNs der gleichzeitig angewendeten Verhalten dargestellt (siehe auch Tabelle E.2). Tabellen E.3 bis E.6 beinhalten die Beschreibung der Knoten und der Diskretisierungsintervalle der entsprechenden BBNs.

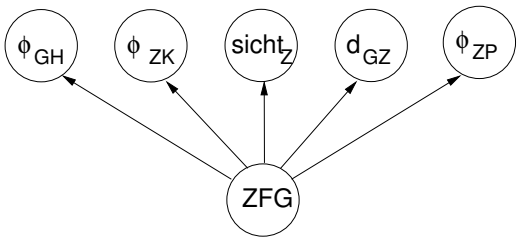


Abbildung E.6: BBN für das zielführende Verhalten.

Abkürzung	Beschreibung	Diskretisierungsintervalle
ZFG	Anwendbarkeit der Zielführung	$\{0; 0.1; 0.2; \ldots; 0.9; 1.0\}$
d_{GZ}	Abstand des Greifers zum Zielobjekt	$\{0; 10; 30; 50; 1000\}$ in mm
ϕ_{ZK}	Abweichung des Zielobjektes von der Kameraachse	$\{0; 5; 15; 30; 180\}$ in Grad
ϕ_{ZP}	Winkel zwischen Planung und Zielführung	$\{0; 10; 30; 60; 180\}$ in Grad
ϕ_{GH}	Winkel zwischen Greifer und nächstem Hindernis	$\{0; 30; 60; 90; 180\}$ in Grad
$sicht_Z$	Sichtbarkeit des Zielobjektes im Bild	$\{0; 10; 30; 120; 180\}$ in Grad

Tabelle E.3: Knoten und Diskretisierungsintervalle für die Zielführung.

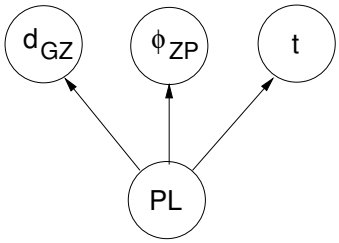


Abbildung E.7: BBN für das Planungsverhalten.

Abkürzung	Beschreibung	Diskretisierungsintervalle
PL	Anwendbarkeit der Planung	$\{0; 0.1; 0.2; \ldots; 0.9; 1.0\}$
d_{GZ}	Abstand des Greifers zum Zielobjekt	$\{0; 10; 30; 50; 1000\}$ in mm
t	Timer	$\{0; 0.1; 0.2; \ldots; 0.9; 1.0\}$
ϕ_{ZP}	Winkel zwischen Planung und Zielführung	$\{0; 10; 30; 60; 180\}$ in Grad

Tabelle E.4: Knoten und Diskretisierungsintervalle für das Planungsverhalten.

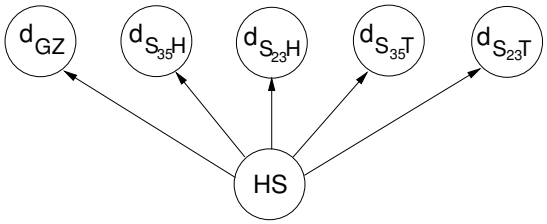


Abbildung E.8: BBN für die Hindernisvermeidung der Segmente.

Abkürzung	**Beschreibung**	**Diskretisierungsintervalle**
HS	Anwendbarkeit der Hindernisvermeidung der Segmente	$\{0; 0.1; 0.2; \ldots; 0.9; 1.0\}$
$d_{S_{23}H}$	Kleinster Abstand des nächsten Hindernisses von dem Oberarm des Manipulators	$\{0; 40; 70; 120; 1000\}$ in mm
$d_{S_{35}H}$	Kleinster Abstand des nächsten Hindernisses von dem Unterarm des Manipulators	$\{0; 40; 70; 120; 1000\}$ in mm
$d_{S_{23}T}$	Kleinster Abstand der Auflagefläche von dem Oberarm des Manipulators	$\{0; 30; 60; 100; 1000\}$ in mm
$d_{S_{35}T}$	Kleinster Abstand der Auflagefläche von dem dem Unterarm des Manipulators	$\{0; 30; 60; 100; 1000\}$ in mm
d_{GZ}	Abstand des Greifers zum Zielobjekt	$\{0; 30; 120; 180; 1000\}$ in mm

Tabelle E.5: Knoten und Diskretisierungsintervalle für das zielfolgende Verhalten.

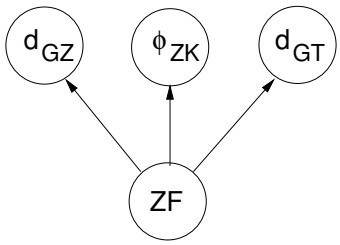


Abbildung E.9: BBN für das zielfolgende Verhalten.

Abkürzung	**Beschreibung**	**Diskretisierungsintervalle**
ZF	Anwendbarkeit des zielfolgenden Verhaltens	$\{0; 0.1; 0.2; \ldots; 0.9; 1.0\}$
d_{GZ}	Abstand des Greifers zum Zielobjekt	$\{0; 50; 100; 880; 1000\}$ in mm
ϕ_{ZK}	Abweichung des Objektes von der Kameraachse	$\{0; 10; 40; 60; 180\}$ in Grad
d_{GT}	Distanz zwischen Greifer und Auflageebene	$\{0; 20; 50; 200; 1000\}$ in mm

Tabelle E.6: Knoten und Diskretisierungsintervalle für das zielfolgende Verhalten.

Anhang F

Deliberative Ebene

F.1 S-GOLOG Programm für das Testszenario

In diesem Unterkapitel werden Teile des S-GOLOG Programms[1], das für das Testszenario zum Einsatz kommt, vorgestellt und analysiert[2].

Das Programm der *Pick-and-Place* Aufgabe in S-GOLOG ist wie folgt definiert:

```
  action move_object_to_position(Obj,Pos,World)
    get_object(Obj,World);
    gotoPlaned(World, Pos);
    llp_platform_relocate(pos);
    put_object(Obj, Pos);
  end;
end;
```

Der Roboter versucht zuerst das Objekt zu finden und zu greifen, plant anschließend mit dem Pfadplaner einen Weg zum Ablageort und legt das Objekt dort ab. Wie man sehen kann, findet keine komplette Suche im Planungsraum statt, da für eine *Pick-and-Place* Aufgabe die Abfolge der Schritte, die der mobile Manipulator auszuführen hat, klar definiert sind. Demnach wäre eine Suche in dieser Phase überflüssig und bei der Anzahl der Teilaufgaben, die der LLP ausführen kann, zeitlich überfordernd für den HLP.

Die Prozedur `get_object(Obj,World)` wird weiter verfeinert[3]:

```
  action get_object(Obj, World)
    try
      llp_global_localize(Obj);
```

[1] Zur Ausführung eines S-GOLOG bzw. IndiGolog Programms ist der Prolog-Interpreter Eclipse benötigt - hier liegt IndiGolog als zusätzliche Bibliothek für die Programmiersprache Prolog vor.

[2] Das vollständige Programm im HLP beinhaltet zusätzlich die Möglichkeit über mehrere Räume zu operieren und durch Türen zu fahren. Allerdings ist kein Planungsschritt zum Türöffnen implementiert worden; der Roboter kann nur durch geöffnete Türen fahren.

[3] Siehe hierzu auch Tabelle 6.1 mit den Teilplänen des LLPs.

```
    or
      set_counter(get_object_cnt,0);
       while counter[get_object_cnt]<10 do
        add_counter(get_object_cnt,1);
        pick x, pos do
          test x is counter[get_object_cnt]-1;
          llp_get_expected_position(Obj,x);
          test pos=last_position[];
          gotoPlaned(World, pos);
          llp_platform_relocate(pos);
          try
            llp_global_localize(Obj);
            set_counter(get_object_cnt,10);
          or
            test 1=0;
          end;
        end;
      end;
    end;
    try
      take_object(Obj);
    or
      set_counter(get_object_cnt,0);
      while counter[get_object_cnt]<10 do
        add_counter(get_object_cnt,1);
        pick x, pos do
          test x is counter[get_object_cnt]-1;
          llp_get_take_position(Obj,x);
          test pos=last_position[];
          llp_plan_path(World,pos);
          llp_goto_position(pos);
          llp_platform_relocate(pos);
          take_object(Obj);
          set_counter(get_object_cnt,10);
        end;
      end;
    end;
  end;
```

Hier wird versucht, zuerst aus der Anfangsposition das Objekt in der Umgebung mit der Sensorik zu lokalisieren. Falls das nicht klappt, dann soll im Weltmodell nach der wahrscheinlichsten Position gesucht werden. Ein Pfad zu dieser Position wird anschließend berechnet und der Roboter dorthin geführt, um erneut nach dem Objekt zu suchen. Zehn Positionen sollen insgesamt überprüft werden, angefangen von der wahrscheinlichsten Position. Falls der mobile Manipulator das Objekt gefunden hat und dies in Reichweite ist, dann hat er einen Greifversuch vorzunehmen. Ansonsten wird eine alternative Greiflage für die Plattform ermittelt und der Roboter dorthin geführt.

Für das eigentliche Greifen, muss man zuerst überprüfen, ob das Objekt in Reichweite liegt. Da der LLP fünf mögliche Greifmethoden zur Verfügung stellt, sollte die beste Methode[4] als erstes ausgewählt und ausprobiert werden. Das Konstrukt `alternative ... do` von S-GOLOG bietet die Möglichkeit, anhand einer Präferenzdatei, die aus der Analyse der Daten vergangener Missionen ermittelt wird, die Reihenfolge der Aktionen so umzustellen, dass der Roboter die erfolgversprechendste Methode als erste ausprobiert.

```
action take_object(Obj)
  llp_in_range(Obj);
  alternative whichtake do
    llp_take_object_1(Obj);
  or
    llp_take_object_2(Obj);
  or
    llp_take_object_3(Obj);
  or
    llp_take_object_4(Obj);
  or
    llp_take_object_5(Obj);
  end;
end;
```

Entsprechend gibt es auch für das Ablegen eines Objektes zwei mögliche Methoden:

```
action put_object(Obj,Pos)
  alternative whichput do
    llp_drop_object_1(Pos);
  or
    llp_drop_object_2(Pos);
  end;
end;
```

Die Prozedur `gotoPlaned` versucht einen Pfad zu finden, der die mobile Plattform von der aktuellen Position zur Zielposition am Objekt führt, falls mobiler Manipulator und Objekt sich im selben Raum befinden. Falls dies nicht der Fall ist, dann findet zuerst eine nicht deterministische topologische Suche mit `go_to_room(room)` statt, um einen Weg zu dem Raum zu finden, in dem sich das Objekt befindet. Anschließend wird für jeden einzelnen zu durchquerenden Raum ein geometrischer Pfad zwischen Eingang und Ausgang mit dem Routenplaner ermittelt. Da jedoch das Testszenario nicht mehrere Räume berücksichtigt, wird diese Möglichkeit hier nicht weiter analysiert.

```
action gotoPlaned(World, Pos)
  llp_get_room(Pos);
  pick room do
    test room = last_querry[];
    go_to_room(room);
```

[4]Die beste Methode wird anhand einer aus früheren Missionen erstellten Erfolgsstatistik ermittelt.

```
    end;
    pick myPos do
      llp_get_coords_of_position(Pos);
      test myPos = last_position[];
      llp_plan_path(World, myPos);
      llp_goto_position(Pos);
    end;
  end;
```

F.2 Weltdatenbank für das Testszenario

```
Root = "LTISzenario"
{
  Obj = "Raum_1"
  {
    Subtype = "Room";
    Position = "3.0, 3.0, 0.0";
    Space = "(-3.5,-3.5),(-3.5,3.0),(3.0,3.0),(3.0,-3.5)";
    Obj = "Tisch_A"
    {
      Subtype = "Table";
      Position = "4.5, 2.0, 0.0";
      Space = "(-0.4,-0.9),(-0.4 ,0.9),(0.4,0.9),(0.4,-0.9)";
      Obj = "TischBereich_1"
      {
        Subtype = "TableArea";
        Position = "4.5, 2.45, 0.0";
        Space = "(-0.4,-0.45),(-0.4,0.45),(0.4,0.45),(0.4,-0.45)";
      }
      Obj = "TischBereich_2"
      {
        Subtype = "TableArea";
        Position = "4.5, 1.55, 0.0";
        Space = "(-0.4,-0.45),(-0.4,0.45),(0.4,0.45),(0.4,-0.45)";
      }
    }
    Obj = "Tisch_B"
    {
      Subtype = "Table";
      Position = "1.5, 2.0, 0.0";
      Space = "(-0.4,-0.9),(-0.4,0.9),(0.4,0.9),(0.4,-0.9)";
      Obj = "TischBereich_1"
      {
        Subtype = "TableArea";
        Position = "1.5, 2.45, 0.0";
```

```
        Space = "(-0.4,-0.45),(-0.4,0.45),(0.4,0.45),(0.4,-0.45)";
      }
      Obj = "TischBereich_2"
      {
        Subtype = "TableArea";
        Position = "4.5, 1.55, 0.0";
        Space = "(-0.4,-0.45),(-0.4,0.45),(0.4,0.45),(0.4,-0.45)";
      }
    }
    Obj = "Tisch_C"
    {
      Subtype = "Table";
      Position = "3.0, 5.0, 0.0";
      Space = "(-0.9,-0.4),(-0.9,0.4),(0.9,0.4),(0.9,-0.4)";
    }
  }
  Root = "Events"
  {
    Root = "ZielObjekt"
    {
      ss_type = "Obj#Raum_1:Obj#Tisch_A:Obj#TischBereich_1"
      {
        ss_succ = "0 0 0 0 0 1";
        ss_fail = "1 0 0 0 0 0";
      }
      ss_type = "Obj#Raum_1:Obj#Tisch_A:Obj#TischBereich_2"
      {
        ss_fail = "0 0 0 0 0 0";
        ss_succ = "0 1 0 0 0 0";
      }
      ss_type = "Obj#Raum_1:Obj#Tisch_B:Obj#TischBereich_1"
      {
        ss_fail = "0 0 1 1 0 0 ";
        ss_succ = "0 0 0 0 1 0 ";
      }
      ss_type = "Obj#Raum_1:Obj#Tisch_B:Obj#TischBereich_2"
      {
        ss_fail = "0 0 0 0 0 0 ";
        ss_succ = "0 0 0 0 0 0 ";
      }
    }
  }
}
```

Lebenslauf

Persönliche Daten

Name:	Alexandros Matsikis
Geburtsdatum/-ort:	17.07.1974, Thessaloniki, Griechenland
Staatsangehörigkeit:	griechisch

Schulausbildung

1980-1983	Private Volksschule der Republik Griechenland in München
1983-1986	Griechische Grundschule in Thessaloniki
1986-1992	Deutsche Schule Thessaloniki (Gymnasium und Lykeion)

Studium

1992-1998	Aristotle University of Thessaloniki, Fakultät für Elektrotechnik, Studiengang Elektronik und Computer
1998-2000	DAAD Stipendiat am Lehrstuhl für Technische Informatik, Fakultät für Elektrotechnik und Informationstechnik, RWTH Aachen

Studienbegleitende Tätigkeiten

1995 (1 Monat)	OTE (Griechische Telekommunikationsgesellschaft), Thessaloniki, Griechenland
1996 (4 Monate)	Gruppe für Angewandte Informatik (GfAI), Herrenschwanden, Schweiz
1997 (1 Monat)	Department of Computer Systems, State Academy of Aerospace Instrumentation, St. Petersburg, Russland

Berufstätigkeit

2000-2003	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Technische Informatik, Fakultät für Elektrotechnik und Informationstechnik, RWTH Aachen

Aachen, den 31.01.2005